algorithms

MDPI

Article

Investigations into the Design and Implementation of Reinforcement Learning Using Deep Learning Neural Networks

Roxana-Elena Tudoroiu [1], Mohammed Zaheeruddin [2], Daniel-Ioan Curiac [3], Mihai Sorin Radu [1] and Nicolae Tudoroiu [4,*]

[1] Sciences and Electrical Engineering, University from Petrosani, 332006 Petrosani, Romania; tudelena@mail.com (R.-E.T.); sorinradu@upet.ro (M.S.R.)
[2] Centre for Building Studies, Concordia University from Montreal, Montreal, QC H3G 1M8, Canada; zaheer@encs.concordia.ca
[3] Department of Automation and Applied Informatics, Polytechnic University of Timisoara, 300006 Timisoara, Romania; daniel.curiac@aut.upt.ro
[4] Engineering Technologies Department, John Abbott Collège in Sainte-Anne-de-Bellevue, Montréal, QC H9X 3L9, Canada
* Correspondence: nicolae.tudoroiu@johnabbott.qc.ca; Tel.: +1-514-966-5627

Abstract: This paper investigates the design and MATLAB/Simulink implementation of two intelligent neural reinforcement learning control algorithms based on deep learning neural network structures (RL DLNNs), for a complex Heating Ventilation Air Conditioning (HVAC) centrifugal chiller system (CCS). Our motivation to design such control strategies lies in this system's significant control-related challenges, namely its high dimensionality and strongly nonlinear multi-input multi-output (MIMO) structure, coupled with strong constraints and a substantial impact of measured disturbance on tracking performance. As a beneficial vehicle for "proof of concept", two simplified CCS MIMO models were derived, and an extensive number of simulations were run to demonstrate the effectiveness of both RL DLNN control algorithm implementations compared with two conventional control algorithms. The experiments involving the two investigated data-driven advanced neural control algorithms prove their high potential to adapt to various types of nonlinearities, singularities, dimensions, disruptions, constraints, and uncertainties that inherently characterize real-world processes.

Keywords: centrifugal chiller system; reinforcement learning; deep learning neural network

Academic Editors: Chih-Lung Lin, Bor-Jiunn Hwang, Shaou-Gang Miaou and Chi-Hung Chuang

Received: 17 January 2025
Revised: 26 February 2025
Accepted: 13 March 2025
Published: 16 March 2025

Citation: Tudoroiu, R.-E.; Zaheeruddin, M.; Curiac, D.-I.; Radu, M.S.; Tudoroiu, N. Investigations into the Design and Implementation of Reinforcement Learning Using Deep Learning Neural Networks. *Algorithms* **2025**, *18*, 170. https://doi.org/10.3390/a18030170

1. Introduction

Control algorithms are essential parts of the successful operation of processes. In recent years, data-driven intelligent control algorithms mainly relying on artificial intelligence and soft computing concepts have emerged as a valuable alternative to conventional model-based control methods, especially in cases involving complex systems. A special category of such new algorithms that employ the reinforcement learning principle to tune deep neural networks for optimal control of processes is still in the infancy stage but has shown promising results when applied to the control of highly nonlinear, high-dimensional, time-delayed, time-varying, partially known, or hard-to-mathematically-formalize systems. From this perspective, centrifugal chiller systems (CCSs) from complex Heating Ventilation Air Conditioning (HVAC) are no exception, needing specifically tailored control strategies. In this context, the current paper aims to explore the development and MATLAB/Simulink implementation of two neural reinforcement learning (RL) control algorithms utilizing deep learning neural network (DLNN) frameworks in

relation to a specific intricate HVAC centrifugal chiller system. We address the key limitations of existing approaches by making the following contributions:

- We developed an accurate simplified CCS model in a state-space representation. Based on the input–output measurement dataset of the open loop multi-input multi-output (MIMO) CCS extended nonlinear model, having 39 states, three inputs and two outputs, a MIMO autoregressive moving average with an exogenous input (ARMAX) delayed fourth-order polynomial model of z-representation in the complex domain is obtained. This ARMAX model was converted into a linearized MIMO CCS model with four states, three inputs, and two outputs;
- We designed and tuned a standard PID controller and also a standard model predictive control (MPC) for comparison purposes;
- We built two RL DLNN controllers connected in series in a forward path with the MIMO CCS simplified model represented in state space, for temperature control inside the evaporator subsystem and liquid refrigerant level control within the condenser subsystem;
- We rigorously evaluated the tracking performance for both RL DLNNs and compared it with the results obtained using classic PID or MPC controllers.

The remainder of this research paper is structured in the following six sections. Section 2 is devoted to a brief literature review and some preliminaries regarding HVAC control systems. Section 3 outlines four conventional closed-loop control methods, designed and implemented in Matlab/Simulink, with three being based on a standard PID controller, and the fourth being an MPC. Section 4 focuses on the design of two RL DLNN controllers. The extensive simulation results are reported in Section 5. Section 6 is dedicated to discussions, and Section 7 concludes the paper, briefly highlighting the main yields and the future work directions.

2. Related Work and Preliminaries on HVAC Systems

2.1. Literature Review of HVAC Control Systems

It is well known that the electricity market is undergoing substantial transformations in grid modernization, large-scale energy storage, and efficient energy transfer management. Many elements must work together to effectively provide domestic and commercial consumers with the electricity services necessary for sustainable development. Both supply and demand will need to adapt to a new and diverse energy mix, including expanding demand-side management, building new energy storage capacities, and investing in modern and efficient power grids. In this context, a considerable amount of energy consumption in any commercial or residential building is due to heating, ventilation and air conditioning (HVAC) systems [1–4]. Therefore, improving their efficiency becomes critical for energy and environmental sustainability [5–8]. In general, centrifugal chillers are the widely preferred cooling units for a wide range of HVAC control system applications due to their high efficiency, reliability, and low maintenance costs. More precisely, they are suitable for providing cold water for the cooling needs of all the air-handling units in a building [1]. Since a centrifugal chiller is the most energy-consuming HVAC device, its efficiency can be improved by using advanced model-based as well as data-driven controller design strategies [9–11]. A brief review of the literature on HVAC centrifugal chillers reveals that a significant amount of work has been carried out on steady-state and transient modeling. Several dynamic models for the vapor compression cycle have been extensively studied [8]. Also, as previously discussed [1], mechanistic models of single-stage and two-stage centrifugal chillers have been developed in which the centrifugal compressor is modeled based on the Euler turbomachinery, the balance energy, and the impeller velocity equations. The energy coefficient of performance (COP) of the chiller is simulated by considering the compressor polytropic efficiency and hydrodynamic, mechanical, and electrical losses. Meanwhile, the condenser and evaporator are modeled

based on the lumped parameter approach, and the heat transfer is calculated based on the effectiveness model. Taking into account the fact that their dynamics are of high nonlinearity, complexity, and dimensionality, from an appropriate control perspective, the developed models must be simplified [8]. An interesting dynamic model of a semi-hermetic reciprocating compressor was developed based on the first law of thermodynamics applied to a lumped control volume, the expansion valve modeled based on a simple orifice flow model [1]. Over the past two decades, our research team has worked intensively with HVAC field control systems, focusing on developing high-fidelity dynamic modeling of centrifugal chillers and building the most suitable control strategies based on them, as a priority task of our research [8]. Interested readers and specialists working in the field can find in [8] a valuable support for the development of simplified, linearized, low-dimensionality, accurate, robust, and stable models in different state-space representations or in the complex domain. Moreover, these models have been validated and adapted to the harshest realistic working environment including a variety of uncertainties, nonlinearities, high dimensionality, and disturbances. Therefore, this paper presents some models of multi-input multi-output (MIMO) centrifugal cooling systems (CCS) with enough details for good readability and understanding. Based on these, advanced closed-loop intelligent neural control strategies of real practical interest were developed. Specifically, the models chosen for our case study served as a valuable vehicle for "proof of concept" and simulation purposes. These models were inspired by the literature in the field, with one of the works being particularly fundamental [12], and others [13–16] providing some valuable and useful references for the design of models based on discrete time data, such as AutoRegressive with eXogenous input (ARX) and Autoregressive Moving Average with eXogenous input (ARMAX), which are polynomial models built in state space. Compared with standard control strategies such as proportional integral derivative (PID) control [17] and also Model Predictive Control (MPC), the advanced neural intelligent reinforcement learning deep learning neural network (RL DLNN) control systems [17–19] demonstrated better performance in various abilities such as possessing human-like expertise in a particular domain, self-adjusting, adaptively learning environmental changes, and taking the best decisions or most appropriate actions [20–24].

2.2. Preliminaries—Adopted CCS Model and Its Implementation

This research used a valuable vehicle, namely three MIMO Centrifugal Chiller System (CCS) models, adopted as a case study, for "proof of concept" and MATLAB R2024a simulation purposes.

2.2.1. MIMO Centrifugal Chiller Modeling Assumptions

The proposed MIMO (three inputs, two outputs) Centrifugal Chiller model used in the case study for this research was previously developed with sufficient detail [8] and also validated [7]. In the current research, this model has been simplified to be suitable for control purposes. In the case study, the proposed dynamic model of the Centrifugal Chiller System was constructed by interconnecting the following subsystems: a water-cooled centrifugal chiller, a centrifugal compressor, shell-and-tube heat exchangers, a thermal expansion valve, and the controller. The water-cooled centrifugal chiller was modeled based on the mechanics of fluid theory, and the centrifugal compressor was modeled based on turbo-machinery theory, similar to those developed previously [8]. The chiller's capacity control was achieved by the combination of variable inlet guide vane and variable speed drive and the entire system consisted of four major components [1,8], as shown in Figure 1, namely, a centrifugal compressor, a condenser, an expansion valve, and an evaporator. Typically, the chilled-water system consists of one circulating refrigerant loop and two water loops. The first water loop circulates between the condenser and the cooling tower, and the second water loop circulates between the evaporator and the air handling units (AHUs) that produce chilled water for the cooling coil.

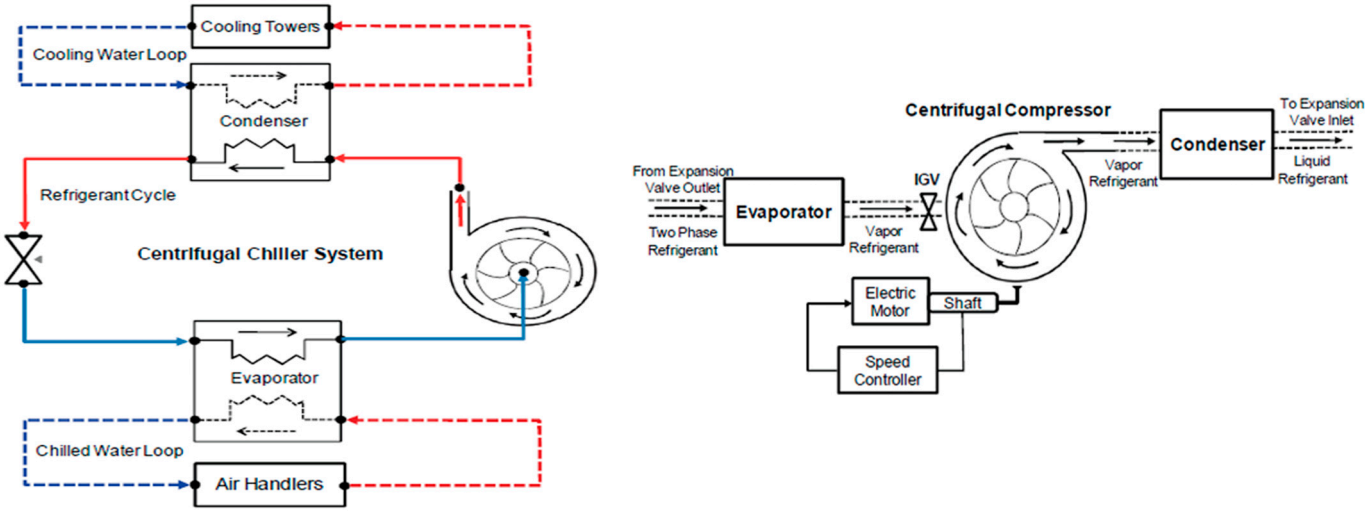


Figure 1. Following the direction of the arrows, chilled water flows through centrifugal chiller evaporator and condenser subsystems, with all four main components [1,8].

A thermal expansion valve is used to regulate the pressure levels at the condenser and evaporator sides. In this research, the overall dynamics of the MIMO chiller control system are calculated in a state-space representation via a set of 39 nonlinear differential equations, with 39 states, two inputs (compressor driver relative speed, expansion valve opening), and two outputs (water temperature, liquid level in the evaporator). The impact of load temperature as a main disturbance was also investigated. Even if a centrifugal chiller is a highly dimensional system, it can easily be integrated into various control HVAC control applications in commercial buildings, with the chilled-water system supplying chilled water for the cooling needs of all the building's air-handling units. In order to accomplish these tasks HVAC, control systems usually include a water pump to circulate the chilled water through the evaporator and throughout the building for cooling. Also, another water loop from the chiller moves the heated condenser water, and another pump circulates the heated water to the cooling tower and cools it back [1,8].

2.2.2. Case Study—MIMO Centrifugal Chiller System Assumptions and Decomposition

The dynamic model of the overall centrifugal chiller control system is of high complexity in terms of state dimensionality and nonlinearity and is described in state space by a set of 39 first-order differential equations (ODE), as a natural mathematical form to represent a physical system. It is beyond the current scope to write out all these equations, since the MIMO Centrifugal Chiller model used in the case study has already been developed and validated in sufficient details in other work [14,15], but we briefly present some significant aspects of the modeling methodology in this section. The first assumption under consideration in the case study related to the decomposition of the overall centrifugal chilller control system into two embedded open-loop subsystems, the first one an open-loop temperature control of the chilled water Tchw_sp inside the evaporator subsystem, and the second an open-loop level control of the liquid refrigerant level L inside the refrigerator subsystem. The second assumption related to the interference and independence of both loops. Based on two of these assumptions, the three-input two-outputs MIMO model with temperature load disturbance and two independent loops was attached to a data-based CCS model to be investigated. More precisely, the input triplet (compressor speed Ucom, expansion valve opening U_EXV, temperature load disturbance Tchw_rr (chilled water return temperature)) and output doublet (temperature Tchw_sp, level L) were considered. It is worth noting that this modeling strategy, using a deterministic input disturbance Tchw_rr, was the most reflective of real life, which is of particular interest for investigation. As an oriented object, the MIMO Centrifugal Chiller

plant is considered in MATLAB as an *iddata* object, generated based on the dataset of input–output measurements loaded in the MATLAB workspace, using the following MATLAB code lines:

load InputOutputChiller_Data.mat *Tchw_sp Level Ucom U_EXV Tchw_rr;*
CentrChiller = iddata (y,u,1)
where the input–output measurement dataset is collected in an open loop using a specific data acquisition equipment and stored in the input vector u = [u1 u2 u3], which is a three-columns vector assigned as follows:
u1= Ucom; u2 = U_EXV; u3 = Tchw_rr
and in the output vector y = [y1 y2] that is a two-column vector assigned as follows:
y1 = Tchw_sp of the chilled water temperature inside the evaporator subsystem
y2 = Level of the liquid refrigerant level of the refrigerator subsystem

2.2.3. Open-Loop MIMO Centrifugal Chiller System (MIMO) MATLAB Simulink Extended Model Diagram

Representation of the MIMO Centrifugal Chiller System dynamics in state space is achieved with a MIMO Simulink model with three inputs (u1 = Ucom, u2 = U_EXV, u3 = Tchw_rr), two outputs (y1 = Tchw_sp, y2 = Level), and 39 states described by 39 nonlinear first-order differential equations encapsulated in a MATLAB function block, as shown in Figure 2. Also, in Figure 3 are presented the MATLAB Simulink open-loop simulation results.

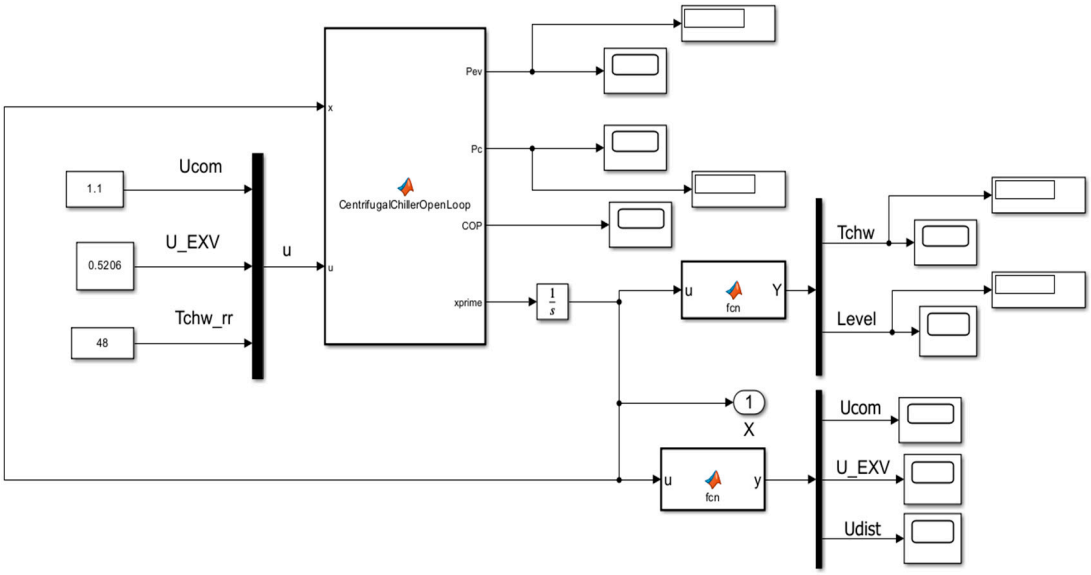


Figure 2. Simulink model of MIMO centrifugal chiller system in state-space representation.

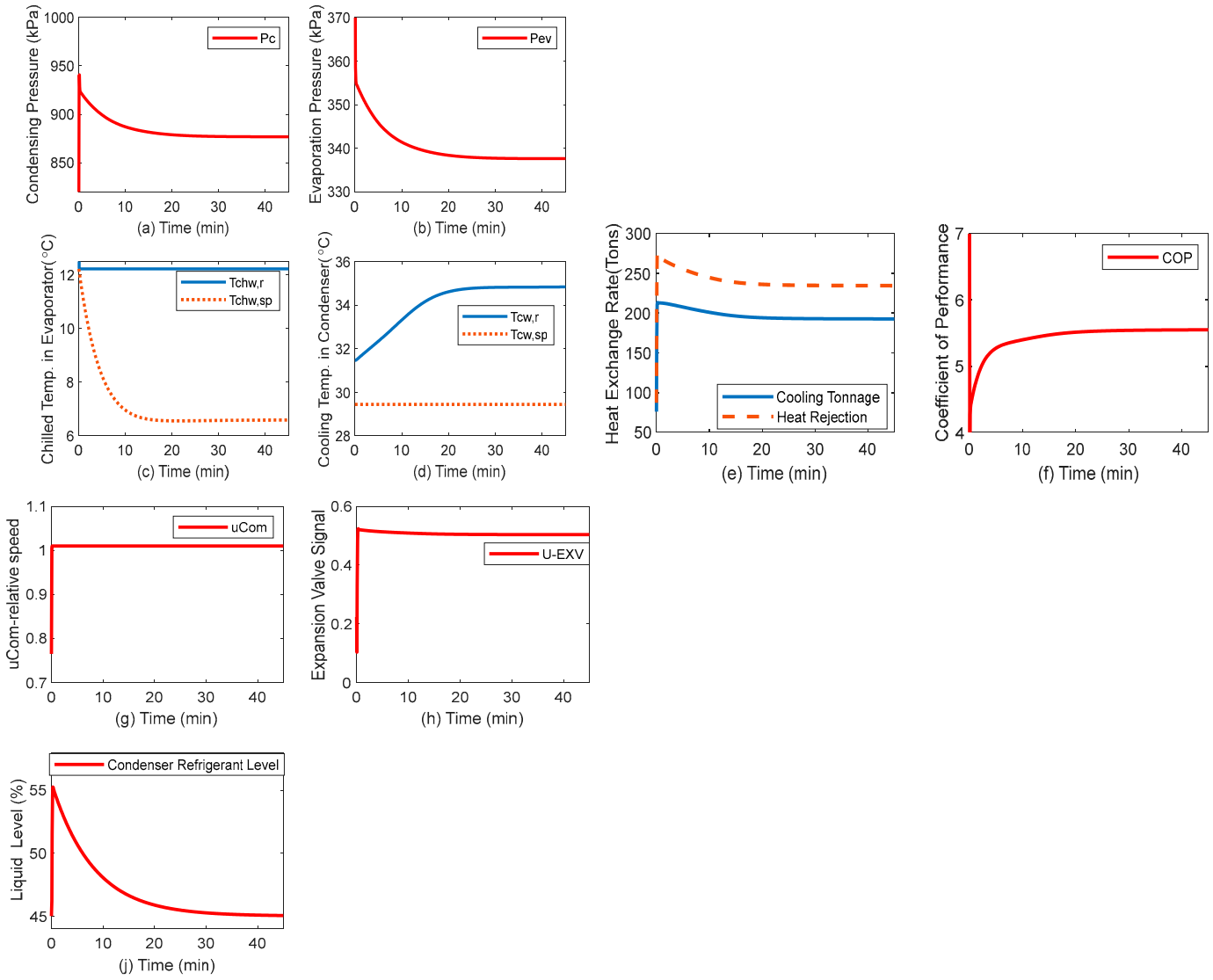


Figure 3. MATLAB Simulink open-loop simulation results: (**a**) condensing pressure (Pc); (**b**) evaporator pressure (Pev); (**c**) chilled water temperature (Tchwr) in evaporator; (**d**) cooling water temperature (Tchwsp) in condenser; (**e**) heat exchange rate; (**f**) coefficient of performance (COP); (**g**) uCom input—relative compressor speed (RPM/RPMnominal); (**h**) u_EXV input–expension valve opening in (%); (**i**) liquid refrigerant level in the condenser (%); (**j**) the MATLAB function selected for the visualization block.

A detailed Simulink model of an extended MIMO Centrifugal Chiller closed-loop control system (three inputs, two outputs, and 39 states) using a proportional integral derivative (PID) controller described in Section 3, and the MATLAB simulations result is depicted in Section 5.

2.2.4. Case Study—The Data-Driven ARMAX Model for MIMO Centrifugal Chiller Subsystems in Discrete-Time State-Space Representation.

The MATLAB System Identification Toolbox provided valuable tools for developing a simplified second-order polynomial ARMAX model of high accuracy, which was assigned to the full model of the MIMO Centrifugal Chiller. This simplified model proved very useful for building two traditional closed-loop control strategies, namely a PID and MPC, as well as two alternative advanced intelligent reinforcement learning control strategies based on the reinforcement learning deep learning neural networks (RL DLNNs) as a viable alternative to traditional approaches. Finally, a rigorous performance analysis

was carried out for comparison purposes, in order to decide which model performed better in terms of meeting the control design requirements and process control constraints, rejecting the effect of possible disturbances acting on the controlled process, accuracy and tracking error. The original MIMO ARMAX model was developed in state-space representation, using a nonlinear set of ordinary differential equations (ODE) as a natural representation of a physical system, easily solved using a suitable MATLAB solver. Due to its nonlinearity and high dimensionality (39 states), a new simplified MIMO Centrifugal Chiller model is developed in a state space representation.

The following two MATLAB code lines generated the MIMO ARMAX (MIMO) discrete state-space model in the z-complex domain [8]:

MIMO = armax(CentrifugalChiller_DATA, [[2 0; 0 2],[2 2 2; 3 2 3], [1; 1], [1 1 1; 1 1 1]], options)

where the *options* are selected by using the MATLAB code line:
options = armaxOptions('InitialCondition','estimate','Focus','prediction')

In the z-complex domain, the MIMO ARMAX centrifugal chiller model generated in MATLAB is three-input, two-output, four-state linear model described in discrete-time state space by a triplet of matrices (A, B, C) given as follows [8]:

$$A(z^{-1}) = I_{2\times 2} + \begin{bmatrix} -1.757 & 0 \\ 0 & -1.581 \end{bmatrix} z^{-1} + \begin{bmatrix} 0.7601 & 0 \\ 0 & 0.6884 \end{bmatrix} z^{-2}, n_a = 2 \quad (1)$$

$$B(z^{-1}) = \begin{bmatrix} -0.0282 & -0.008134 & 0.001173 \\ 1.469 & -3.07 & 0.03456 \end{bmatrix} z^{-1} + \begin{bmatrix} 0 & 0.001721 & 0 \\ 0 & 2.079 & 0 \end{bmatrix} z^{-2} \quad (2)$$

$$C(z^{-1}) = I_{2\times 2} + \begin{bmatrix} 0.3064 & 0 \\ 0 & 0.7141 \end{bmatrix} z^{-1} \quad (3)$$

or condensed in discrete-time z-transform transfer matrix representation:

$$H_y^u(z) \stackrel{\text{def}}{=} \begin{bmatrix} H_{y_1}^{u_1}(z) & H_{y_1}^{u_2}(z) & H_{y_1}^{u_3}(z) \\ H_{y_2}^{u_1}(z) & H_{y_2}^{u_2}(z) & H_{y_2}^{u_3}(z) \end{bmatrix} \quad (4)$$

such that:

$$Y(z) = H_y^u(z) \times U(z), Y(z) = \begin{bmatrix} Y_1(z) \\ Y_1(z) \end{bmatrix}, \text{and } U(z) = \begin{bmatrix} U_1(z) \\ U_2(z) \\ U_3(z) \end{bmatrix} \quad (5)$$

where the input and output channels correspond to the following z-transfer functions.

From the channel input u1 to output y1, the discrete-time transfer function is given as follows:

$$H_{y_1}^{u_1}(z) \stackrel{\text{def}}{=} \frac{-0.0282z^{-1}}{1 - 1.756z^{-1} + 0.7601z^{-2}}$$

From the channel input u1 to output y2, the discrete-time transfer function is denoted thus:

$$H_{y_2}^{u_1}(z) \stackrel{\text{def}}{=} \frac{1.469z^{-1}}{1 - 1.628z^{-1} + 0.6884z^{-2}}$$

From the channel input u2 to output y1, the discrete-time transfer function is represented by:

$$H_{y_1}^{u_2}(z) \stackrel{\text{def}}{=} \frac{-0.008134z^{-1} + 0.001721z^{-2}}{1 - 1.756z^{-1} + 0.7601z^{-2}}$$

From the channel input u2 to output y2, the discrete-time transfer function is designated as:

$$H_{y_2}^{u_2}(z) \stackrel{\text{def}}{=} \frac{-3.07z^{-1} + 2.079z^{-2}}{1 - 1.628z^{-1} + 0.6884z^{-2}}$$

From the channel input u3 to output y1, the discrete-time transfer function is given as follows:

$$H_{y_1}^{u_3}(z) \triangleq \frac{0.001173z^{-1}}{1 - 1.756z^{-1} + 0.7601z^{-2}}$$

From the channel input u3 to output y2, the discrete-time transfer function is defined as:

$$H_{y_2}^{u_3}(z) \triangleq \frac{0.03456z^{-1}}{1 - 1.628z^{-1} + 0.6884z^{-2}}$$

The Simulink diagram of MIMO centrifugal chiller model in the z-domain including all these z-transfer functions is represented in Figure A1 in Appendix A.

A compact matrix description of the MIMO ARMAX centrifugal chiller system in discrete-time state space (three inputs, four states, two outputs, and sampling time Ts = 1 s) is given by the following state-space equation:

$$x(t+1) = Ax(t) + Bu(t), y(t) = Cx(t) + Du(t) \quad (6)$$

where the values of the matrix's quadruplet (A, B, C and D) coefficients are given in Equation (7):

$$A = \begin{bmatrix} 0 & -0.7601 & 0 & 0 \\ 1 & 1.756 & 0 & 0 \\ 0 & 0 & 0 & -0.6884 \\ 0 & 0 & 1 & 1.628 \end{bmatrix}, B = \begin{bmatrix} 0 & 0.006884 & 0 \\ -0.1128 & -0.03254 & 0.004691 \\ 0 & 1.04 & 0 \\ 0.7347 & -1.535 & 0.01728 \end{bmatrix}, C = \begin{bmatrix} 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}, D = 0_{2\times 3} \quad (7)$$

The MIMO output responses with the disturbance temperature set to Tchw-rr = 48 [°F] and the actuator inputs (Speedcomp, ExpValve) are shown in Figure 4.

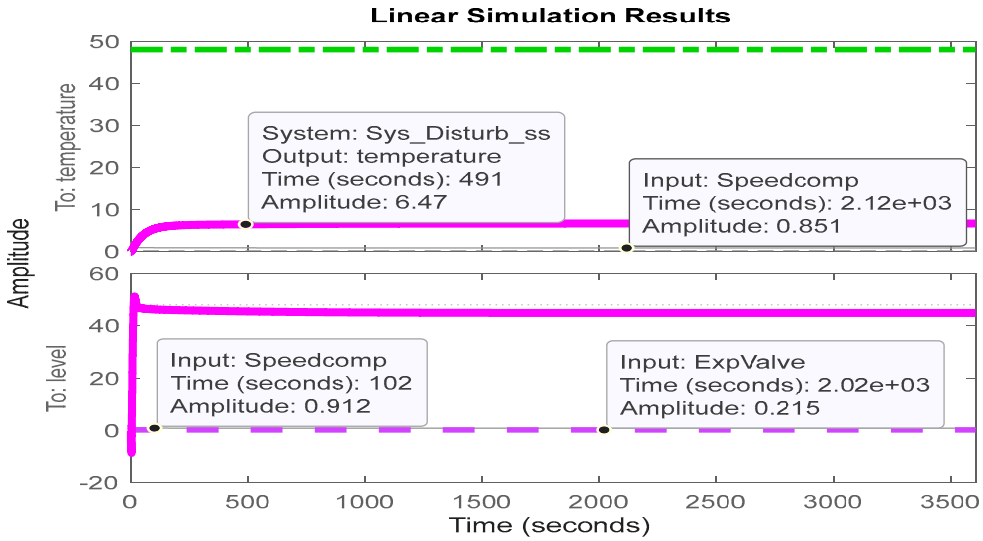


Figure 4. Open-loop linear simulation results for the MIMO ARMAX CCS state-space model.

3. Traditional Controllers: PID and MPC Closed-Loop Control Strategies

In the time domain, the general form of the PID control law in a closed-loop control strategy is given by the following input–output equation:

$$u_{PID}(t) = k_P\varepsilon(t) + k_I \int \varepsilon(t)\,dt + k_D \frac{d\varepsilon(t)}{dt} \quad (8)$$

where $\varepsilon(t)$ represents the error between the actual measurement of system output $y(t)$ and its desired value $ysp(t)$, also called the setpoint, reference, or tracking value, as follows:

$$\varepsilon(t) = y(t) - ysp(t) \tag{9}$$

where $u_{PID}(t)$ refers to the controller output.
$k_P, k_I = \frac{1}{T_I}, k_D = T_D$ represent the PID controller parameters, k_P is the proportional gain, k_I refers to the integral gain, k_D denotes the derivative gain, and T_I and T_D are the time constants for the integral and derivative blocks, respectively. For a particular parameter setting, different controller architectures (e.g., P, I, PI, PD, and PID) may be obtained.

In the complex domain, the transfer function of an ideal PID controller is given bellow:

$$H(s) = \frac{U_{PID}(s)}{\varepsilon(s)} = k_P + \frac{k_I}{s} + k_D s \tag{10}$$

Furthermore, adding a compensator formula to Equation (10) using a low pass filter (LPF) of coefficient N can serve as a practical PID controller via the following transfer function:

$$H(s) = \frac{U_{PID}(s)}{\varepsilon(s)} = k_P + \frac{k_I}{s} + \frac{k_D N}{1 + \frac{N}{s}} \tag{11}$$

3.1. DTI MIMO Centrifugal Chiller Closed-Loop System Control in State-Space Representation

The discrete-time integrator (DTI) closed-loop control strategy architecture of the proposed MIMO CCS model, a valuable "vehicle" for MATLAB Simulink simulations, is depicted in Figure A2, Appendix A. Its discrete-time transfer function in the z-complex domain is given as follows:

$$H_1(z) = \frac{k_1 T_s}{1 - z^{-1}}, k_1 = [-1\ 1], H_1(z) = \frac{k_2 T_s}{1 - z^{-1}}, k_2 = [0.001\ -0.01], T_s = 1 \tag{12}$$

The parameters of both DTI controller blocks $k_1,\ k_2$ are easy to adjust. However, the parameter adjustment process encounters some difficulties due to the constraints on the model inputs $0 \le u_1 = \text{uCom} \le 1.1, 0 \le u_2 = \text{u\_EXV} \le 1$. Also, the "trial and error" procedure to track evaporator temperature and condenser level performance was inaccurate. Figure A2a presents the compact DTI Simulink diagram, with valuable details of the two integrators shown in Figure A2b,c.

3.2. PID MIMO Centrifugal Chiller Closed Loop System Control in Extended State Space Representation

The PID closed-loop control strategy for the MIMO CCS extended model with 39 internal states is depicted in Figure A3a–c. The PID transfer function in the s-complex domain is given by Equation (11). Both controllers' parameter datasets for evaporator temperature (indexed by *T*) and condenser liquid refrigerant level (indexed by *L*) were set to the following values:

$$k_{PT} = 0.1, k_{IT} = 0.0001, k_{DT} = 0.1, N_T = 100, k_{PL} = 0.1, k_{IL} = 0.0001, k_{DL} = 0.1,\ N_L = 100,$$

The temperature setpoint within the evaporator was Tchw-sp = 6.67 [degC], and the level of liquid refrigerant inside the condenser was *L* = 45 (%), with a much longer simulation time of Tf = 13,600 s (3 h 30 min).

Therefore, alternative simplified ARMAX, ANFIS and neural network models of the MIMO Centrifugal Chillaer capable of capturing its entire dynamic evolution and suitable for control purposes were investigated in this research.

Figure A3a shows the Simulink diagram of the PID closed-loop control strategy, and Figure A3b presents two subsystem components of the Simulink diagram (model + visualization block). The code lines of the Simulink MATLAB function from the visualization block are given in Figure A3c. The Simulink simulation results are discussed in Section 5, and a rigorous performance analysis is conducted in Section 6.

3.3. Digital PID Control of MIMO Centrifugal Chiller Closed-Loop System in Extended State-Space Representation (39 States)

A large number of simulations performed on a MATLAB Simulink R2024a platform showed a slow response of about 3 h and 30 min to reach a steady state and achieve accurate tracking performance for control of the evaporator subsystem temperature and the levels of liquid refrigerant in the condenser subsystem. The development of a fast-tuning digital PID controller, similar to an interesting approach proposed previously [15], provided a substantial improvement. Furthermore, an alternative to the extended MIMO centrifugal chiller model, a simplified Adaptive Neural Fuzzy Inference System (ANFIS) is under investigation for integration with the digital PID in the same closed-loop control strategy whose Simulink block diagram is shown in Figure 5. The MATLAB Simulink simulations are presented and discussed in Section 5 and a rigorous performance analysis is carried out in Section 6, revealing a fast response (settling time) and high tracking accuracy. Furthermore, the model's performance was compared with an advanced deep learning neural network (RL DLNN) in which the reward function was generated based on the step response block specifications of digital PID control of the MIMO CCS ANFIS model developed in the next section.

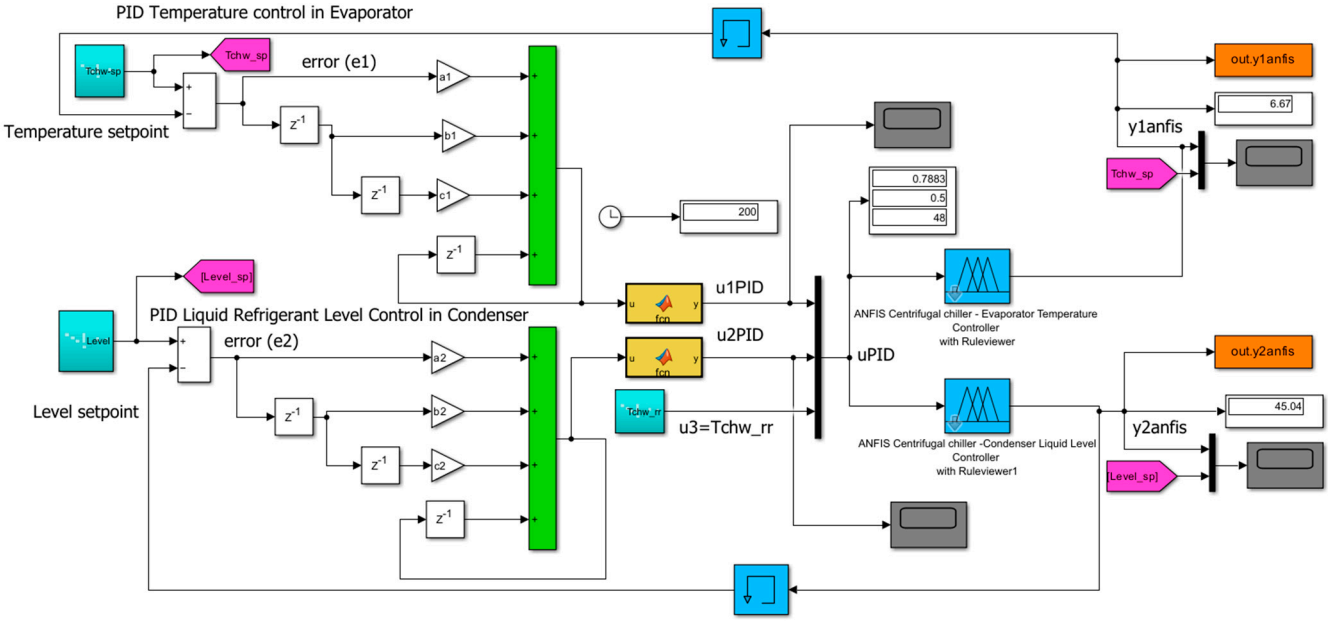


Figure 5. Digital PID ANFIS discrete-time control temperature inside evaporator and liquid refrigerant level within condenser—overall Simulink diagram.

In the Simulink diagram shown in Figure 5 the digital PID controller appears on the left side, implemented based on Equation (8). The simplest way to implement a digital PID is to discretize Equation (8) and then to transform it into a difference equation of the form suggested in [22]. Specifically, this is a recursive method that calculates the PID controller output at $t = kT_s, k \in \mathbb{Z}$, T_s − sampling time, based on the previous value of the controller output and its growth [22], as follows:

$$u_{PID}(kT_s) = u_{PID}[(k-1)T_s] + \Delta u_{PID}(kT_s) \tag{13}$$

For simplicity, assuming $T_s = 1[s]$, Equation (13) becomes:

$$u_{PID}(k) = u_{PID}(k-1) + k_P\big(e(k) - e(k-1)\big) + k_I e(k) + k_D(e(k) - 2e(k-1) + e(k-2)) \tag{14}$$

or, compact:

$$u_{PID}(k) = u_{PID}(k-1) + (k_P + k_I + k_D)e(k) - (k_P + 2k_D)e(k-1) + k_D e(k-2) \tag{15}$$

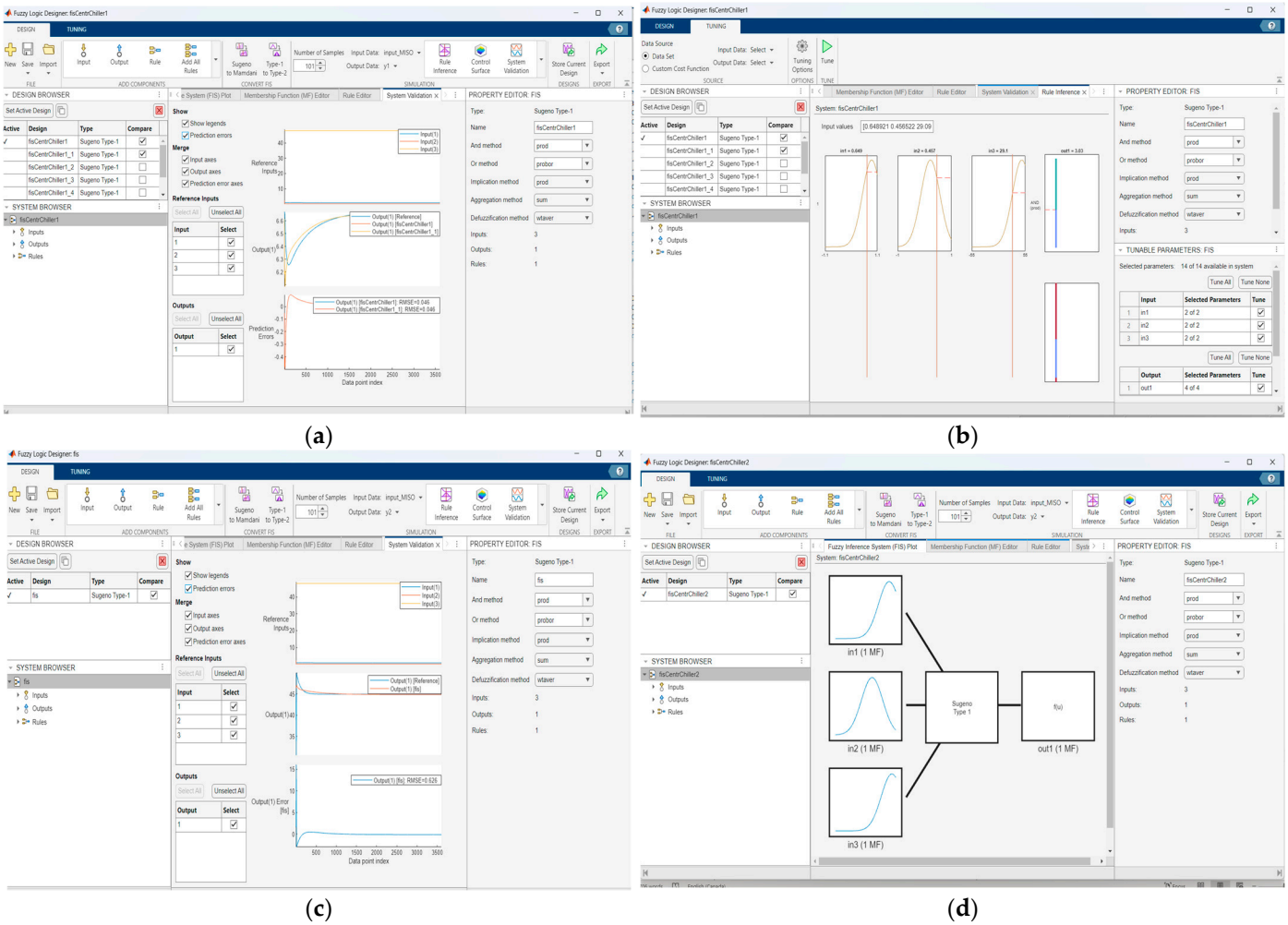
$$u_{PID}(k) = u_{PID}(k-1) + ae(k) + be(k-1) + ce(k-2) \quad (16)$$

with the coefficients a, b and c given by:

$$a = k_P + k_I + k_D, b = -k_P - 2k_D, c = k_D \quad (17)$$

In this research, the controller coefficients were set to (0.095, −0.225, 0.1) for the first digital PID controller integrated into the closed-loop feedback control of the chilled water temperature inside the ANFIS model evaporator subsystem.

The set of values (0.025, −0.105, 0.05) was selected for the second digital PID controller that controlled the liquid refrigerant level in the ANFIS model condenser subsystem. As can be seen, the MIMO Centrifugal model was split into two accurate ANFIS models of the MIMO centrifugal chiller; the first a MISO ANFIS CentrChillerfis1 attached to the evaporator subsystem, and the second a MISO ANFIS CentrChillerfis2 assigned to the condenser, as shown in Figure 6a–h. Both MISO ANFIS models were generated using the Fuzzy Logic Designer app based on the measurement input–output dataset of the open-loop MIMO Centrifugal Chiller plant. The first ANFIS MISO evaporator centrifugal chiller model (fisCentrChiller1) is shown along with the membership functions in Figures 6a,b, and its estimation performance is reported in Figure 6e. Data for the second ANFIS MISO condenser centrifugal chiller, including the membership functions, are shown in Figure 6c,d and its estimation performance in Figure 6f. Figure 6g,h show a Rule Viewer for each MISO ANFIS model.



(**a**) (**b**)

(**c**) (**d**)

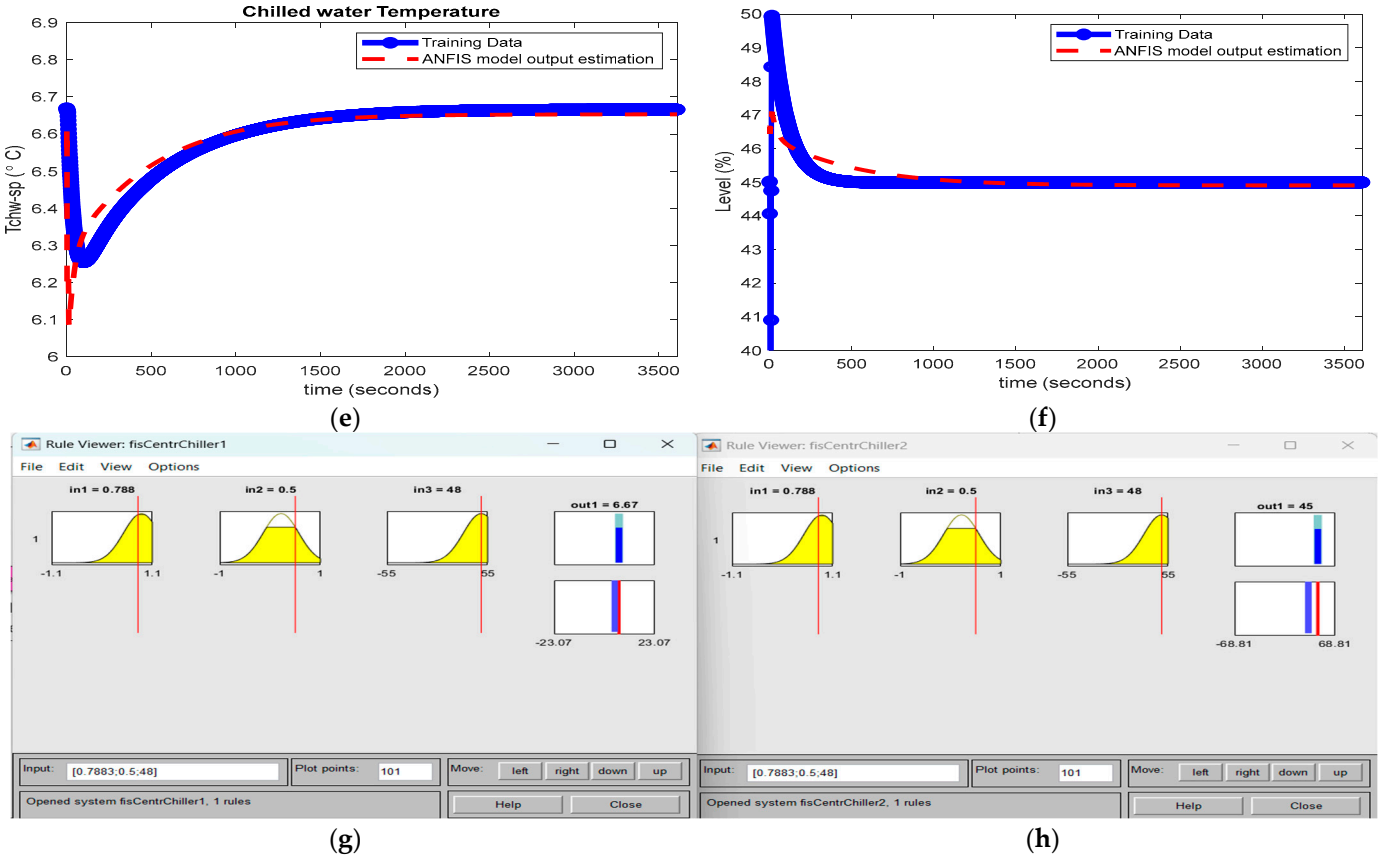


Figure 6. Fuzzy logic design application, MATLAB Simulink simulation results: (**a**) Centrifugal Chiller FIS temperature within evaporator; (**b**) Centrifugal Chiller FIS membership functions; (**c**) Centrifugal Chiller FIS temperature within condenser; (**d**) Centrifugal Chiller FIS membership functions; (**e**) Chilled water temperature, evaporator training data and ANFIS model output estimation; (**f**) liquid refrigerant level inside condenser training data and ANFIS model output estimation; (**g**) Rule Viewer: fis MISO chilled water temperature inside the evaporator subsystem; (**h**) Rule Viewer: fis MISO liquid refrigerant level within the condenser subsystem.

The MATLAB Simulink simulation results are discussed in Section 5, and a rigorous comparative performance analysis is made in Section 6.

3.4. Model Predictive Control Based on Centrifugal Chiller MIMO State-Space Representation

The MPC closed-loop control strategy architecture for MIMO CCS is shown in Figure 7. The model predictive control (MPC) object was generated in MATLAB, as explained is detail in Section 4.2, and the MPC Simulink diagram shown in Figure 7 was obtained using the MPC MATLAB Simulink Toolbox. The MPC object has two manipulated variables (MVs): uCom and u_EXV; one measured disturbance (MD) (temperature Trr set to 48 degrees Fahrenheit); and two measured outputs variables (OV), namely, evaporator temperature Tchw_sp and liquid refrigerant level inside condenser. The MIMO model of the Centrifugal Chiller plant is represented in a discrete-time state space via Equation (6), with the values of the matrices' coefficients given in Equation (7). The MATLAB Simulink simulation results are shown in Section 5.2 and the performance analysis is reported in Section 6.

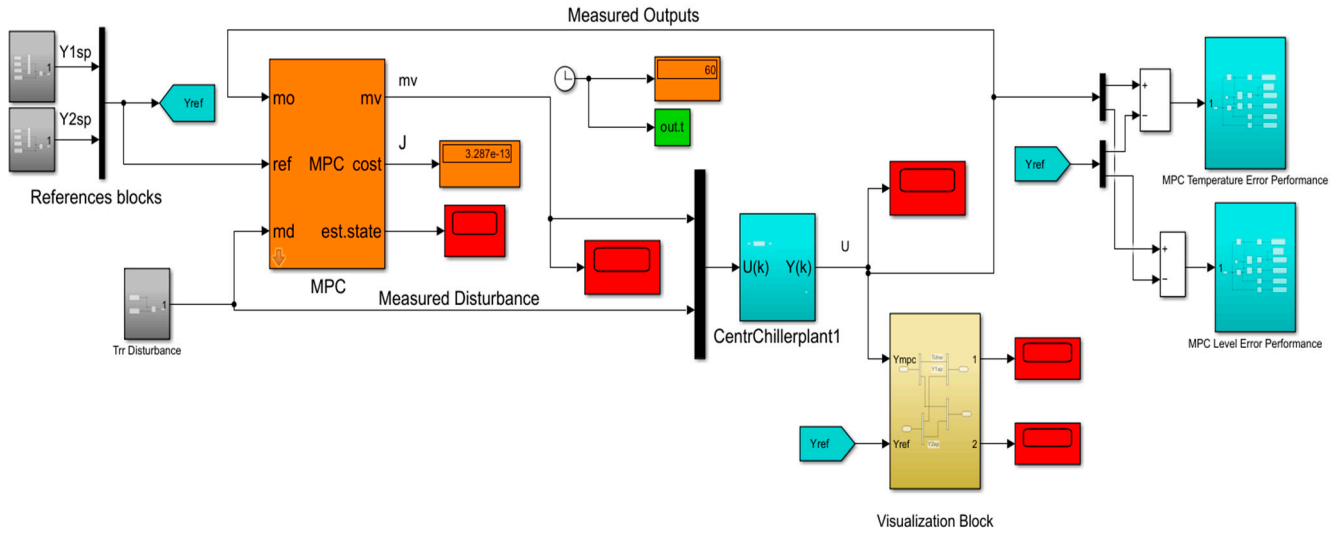


Figure 7. Simulink models of MPC using Simulink MPC Toolbox and MPC Design application.

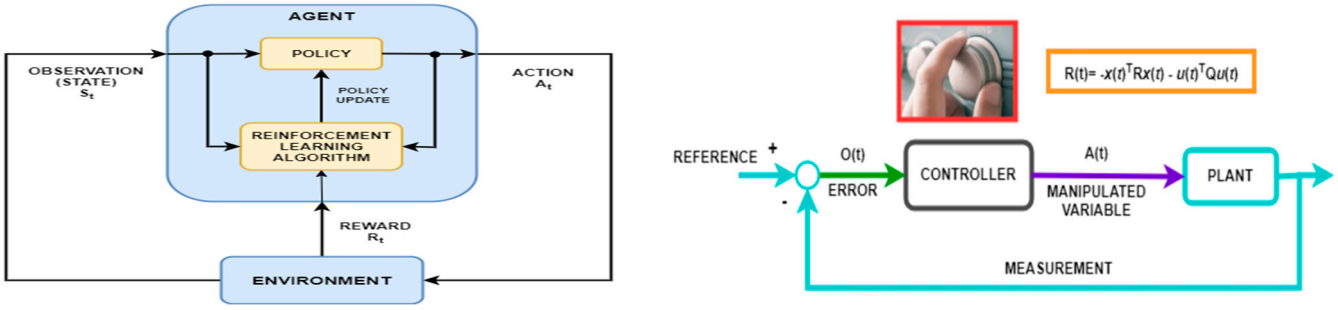
4. Design and Implementation of Reinforcement Learning Using Deep Learning Neural Networks-MIMO CCS Closed-Loop Control Strategies

This section focuses on the design and implementation in MATLAB Simulink of two RL DLNN advanced intelligent control strategies for a MIMO CCS simplified state-space model. The first uses the MPC specifications to generate the reward function, and the second uses step response specifications imposed on a MIMO CCS model represented in the simplified state space. The tracking accuracy performance was compared with the MPC integrated in the first controller structure and an improved digital PID based on the MISO CCS ANFIS models.

4.1. Reinforcement Learning Closed-Loop Control Strategy Using Deep Learning Neural Network MIMO Centrifugal Chiller Plant Model Represented in State Space

4.1.1. Reinforcement Learning Process—Description

Reinforcement learning (RL) involving multi-layered deep learning neural networks is a well-used and well-suited method for research and development in modern artificial intelligence, as is shown in [18]. More precisely, RL is a modeling process in which an agent (controller) learns to make decisions by interacting with an unknown environment through trial and error, as shown in Figure 8a [18,19]. Mathematically speaking, the RL modeling process (algorithm) depicted in Figure 8a is typically based on a Markov decision process (MDP). In this type of process, the RL agent block receives in its current state the observation St as a result of the interaction with the unknown Environment, performing an action that it then sends to the same Environment. The latter reacts by sending a scalar reward Rt to the same RL block for a new transition to the next state St+1, according to the conditional probability of the Environment dynamics; p(St+1 | St, At). The RL agent attempts to learn a policy π(At | St) or map from observations to actions, in order to maximize its returns (expected sum of rewards). In RL (as opposed to optimal control), the algorithm has access to Environment dynamics p(St+1 | St, At) only during the sampling time. During the training process, the RL blocks learn the unknown environment and "make a series of decisions to maximize the cumulative reward for the task without human intervention and without being explicitly programmed to achieve the task" [18].



(**a**) (**b**)

Figure 8. Reinforcement learning architecture [18,19]: (**a**) RL process; (**b**)standard feedback closed-loop control schematic. In this figure, the blue color is used to identify the Agent and Environment blocks, the color green to show the direction of the information flow in the closed-loop of traditional system, and the red color is to highlight that the parameters of the controller are adjustable and a particular Reward function, respectively.

From the perspective of a system-oriented approach, the diagram shown in Figure 8a shows a general representation of a reinforcement learning (RL) algorithm. Figure 8b reveals its equivalence to a traditional closed-loop control strategy used in system applications [19]. In other words, the key role of the RL policy in Figure 8a is to observe the unknown environment (state St) and generate actions (At) to "complete a task optimally", similar to a traditional controller operating in a control system application. Of course, the RL algorithm comes with a valuable improvement to the RL policy, known as policy updating. More specifically, RL can be seen as a mapping of a conventional feedback control system depicted in Figure 8b, and the correspondence between both diagrams is well presented in [19]. The concept is based on "*rewarding or punishing an agent's performance in a specific environment*" [24]. According to this definition, a state is a description of the environment that provides the agent with helpful information for taking a decision at each time step. The agent receives observations and a reward from the Environment and the actions generated are sent into the same Environment. Reward is a measure of the achievement of an action on the completion of the task goal; i.e., the reward signal evaluates the outcome of past actions [18]. RL policy is based on cost function to map each state to the optimal action in order to maximize its reward function during the episode [17].

This RL policy can be a deterministic policy, defined as $A_t = \pi(S_t)$ for $\pi: \mathcal{S} \to \mathcal{A}$, or stochastic policy, described as $A_t \sim \pi(\cdot|S_t)$ for $\pi: \mathcal{S} \to \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is a set of probability distributions over actions. For an MDP, these probability distributions have an interesting feature; they factorize over trajectories [17,19]:

$$p(S_1, A_1, \ldots, S_n, A_n) = p(S_1)\pi(A_1, S_1)\,\mathcal{P}(S_2|S_1, A_1)\pi(A_2, S_2)\ \ldots\ldots\mathcal{P}(S_n|S_{n-1}, A_{n-1})\pi(A_n, S_n) \quad (18)$$

Thus, this feature is useful in the field of control applications, since RL policies need only to consider the current state, which is a strong consequence of the Markov assumption and full observability [23]. Otherwise, if the RL environment is partially observable, then the RL policy must depend on the history of observations. It is important to note that the RL deep learning method differs from supervised learning because it does not require correct sets of actions and labelled input/output pairs [24–27]. The general reinforcement learning structure shown in Figure 8a is described by the following Equations [17]:

$$A_t \sim \pi(A_t|S_t) \quad (19)$$

$$S_{t+1} \sim f_{state}(S_{t+1}|S_t, A_t) \quad (20)$$

$$R_{t+1} = f_{reward}(S_t, A_t, S_{t+1}) \quad (21)$$

$$R = \sum_{t=0}^{t=\infty} \gamma^t R_{t+1} \tag{22}$$

$$Q^{new}(S_t, A_t) = Q^{old}(S_t, A_t) + \alpha\left(R_t + \max_{a' \in \mathcal{A}} Q^{next}(S_{t+1}, a') - Q(S_t, A_t)\right) = Q^{old}(S_t, A_t) + \alpha(Target - Prediction) \tag{23}$$

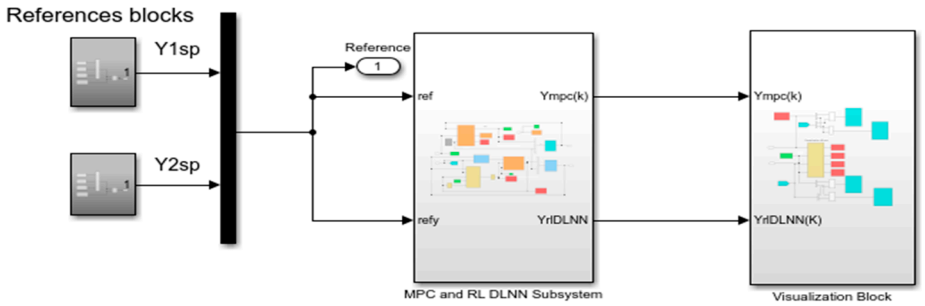
where $\pi(A_t|S_t)$ is the policy at time t, A_t denotes the action of time t, S_t is the state (observation) at time t, f_{state} and $f_{reward}$are the transition functions from time t to t+1, and Q is the state-action value function. The so-called Q function, $0 \leq \gamma < 1$ is an important hyperparameter called the discount factor, which determines how much we care about rewards now versus rewards later; more precisely, this is an exponential decay factor, which means longer term planning is harder.

Also, $0 < \alpha < 1$ represents the learning rate—a value that controls how quickly an agent updates its Q-values based on new information, essentially determining how much weight is given to newly acquired experiences compared with previously learned information; a higher value for α means faster learning and more significant updates to Q-values with each new interaction. When adopting a Monte Carlo technique to estimate the Q-values, every iteration of the Q-value requires updating Q for every state. Monte Carlo estimation of the expectation $\mu = \mathbb{E}[X]$ repeatedly samples X, and the convex relationship [23] is updated as follows:

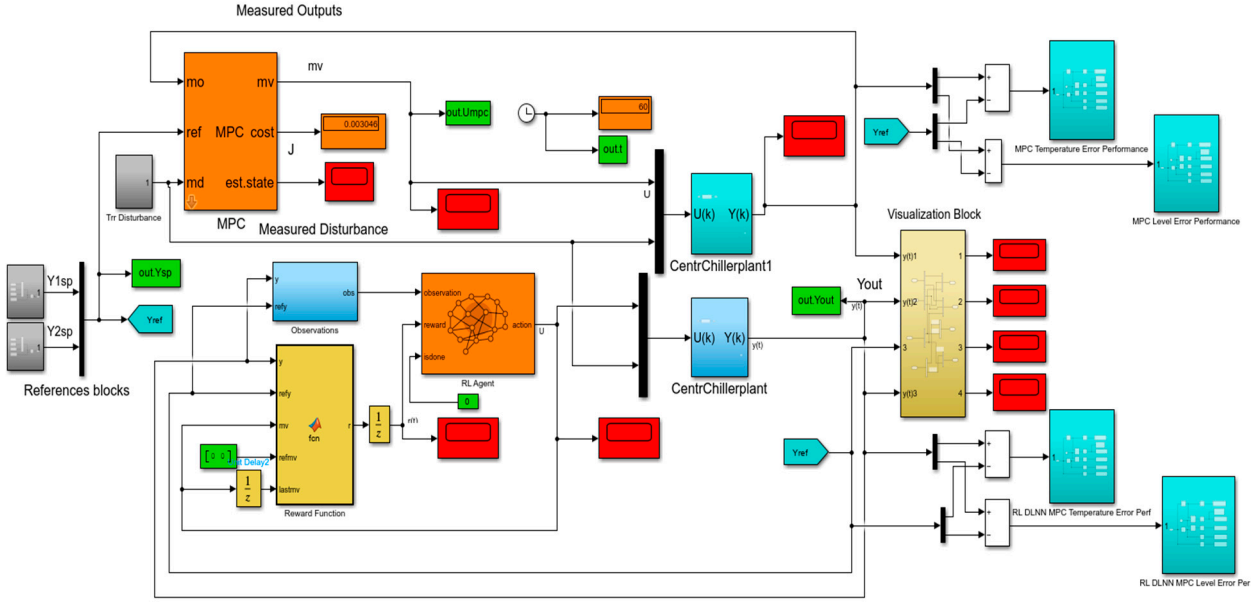
$$\mu \leftarrow \mu + \alpha(X - \mu) \tag{24}$$

In this relationship, the X dynamics is required to evaluate the expectation μ.

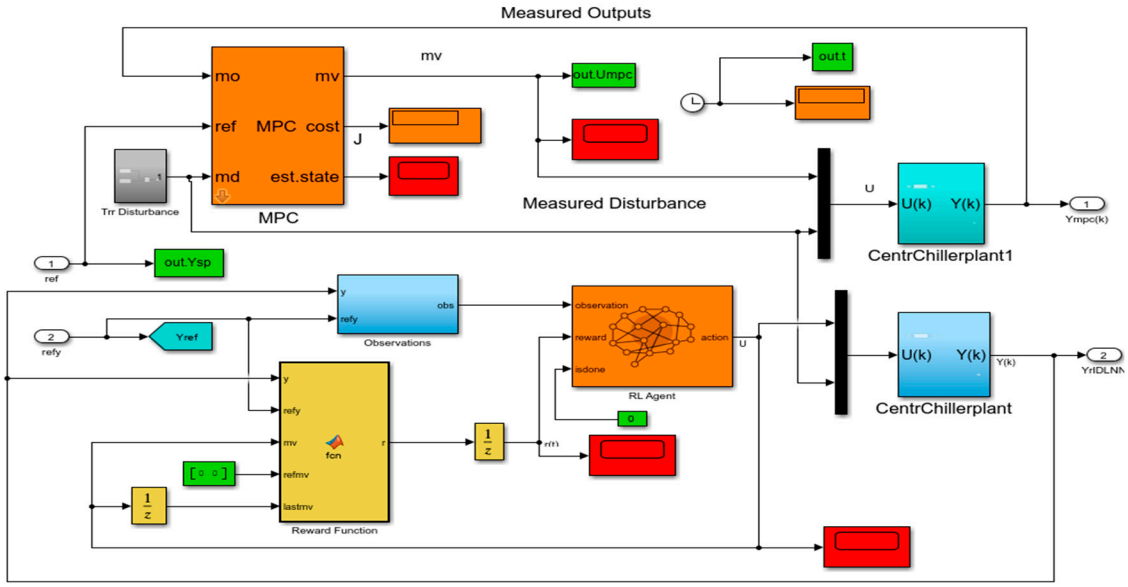
The ε-greedy policy that appears in Figure 9f for a block of parameters of the TD3 RL agent is defined as a policy that chooses $argmax_{a \in \mathcal{A}} \mathrm{Q(S, A)}$ with probability 1-ε and a random action with probability ε. The typical value for the parameter ε is 0.005 [23].



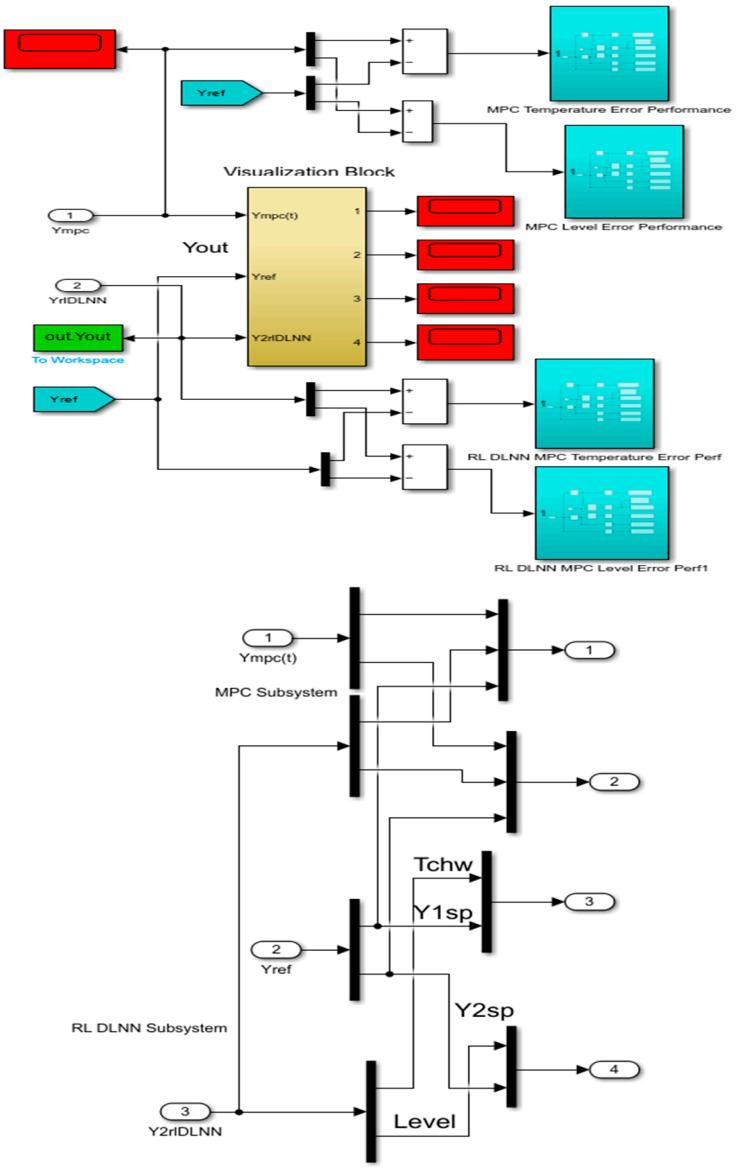
(**a**)


Measured Outputs
mv
mo
mv
ref
MPC cost
md
est.state
MPC
J
0.003046
Trr Disturbance
Measured Disturbance
out.Umpc
out.t
60
U
U(k) Y(k)
CentrChillerplant1
Yref
MPC Temperature Error Performance
MPC Level Error Performance
Visualization Block
Y1sp
Y2sp
References blocks
out.Ysp
Yref
Observations
RL Agent
CentrChillerplant
out.Yout
Yout
Reward Function
RL DLNN MPC Temperature Error Perf
RL DLNN MPC Level Error Per

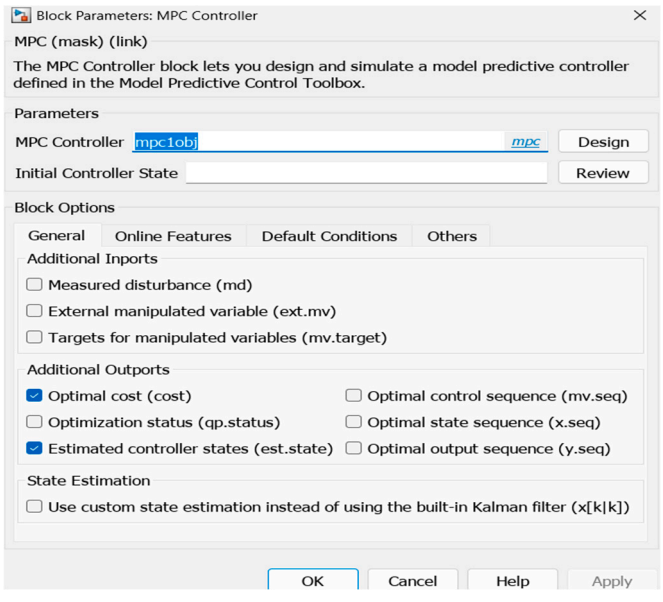
(**b**)


Measured Outputs
mv
mo
mv
ref
MPC cost
md
est.state
MPC
J
Trr Disturbance
out.Umpc
out.t
Measured Disturbance
U
U(k) Y(k)
CentrChillerplant1
Ympc(k)
ref
out.Ysp
refy
Yref
Observations
RL Agent
CentrChillerplant
YrfDLNN
Reward Function

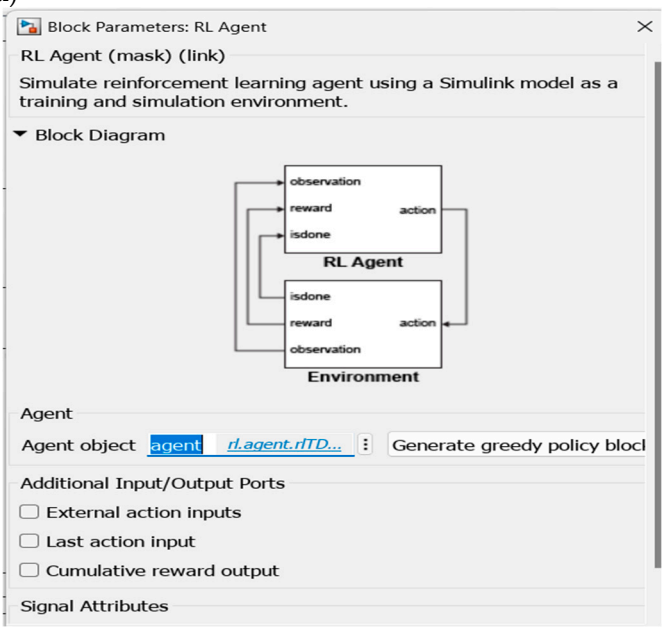
(**c**)


Yref
MPC Temperature Error Performance
MPC Level Error Performance
Visualization Block
Ympc
Yout
YrlDLNN
out.Yout
To Workspace
Ympc(t)
Yref
Y2rlDLNN
RL DLNN MPC Temperature Error Perf
RL DLNN MPC Level Error Perf1
Ympc(t)
MPC Subsystem
Tchw
Y1sp
Yref
Y2sp
RL DLNN Subsystem
Y2rlDLNN
Level

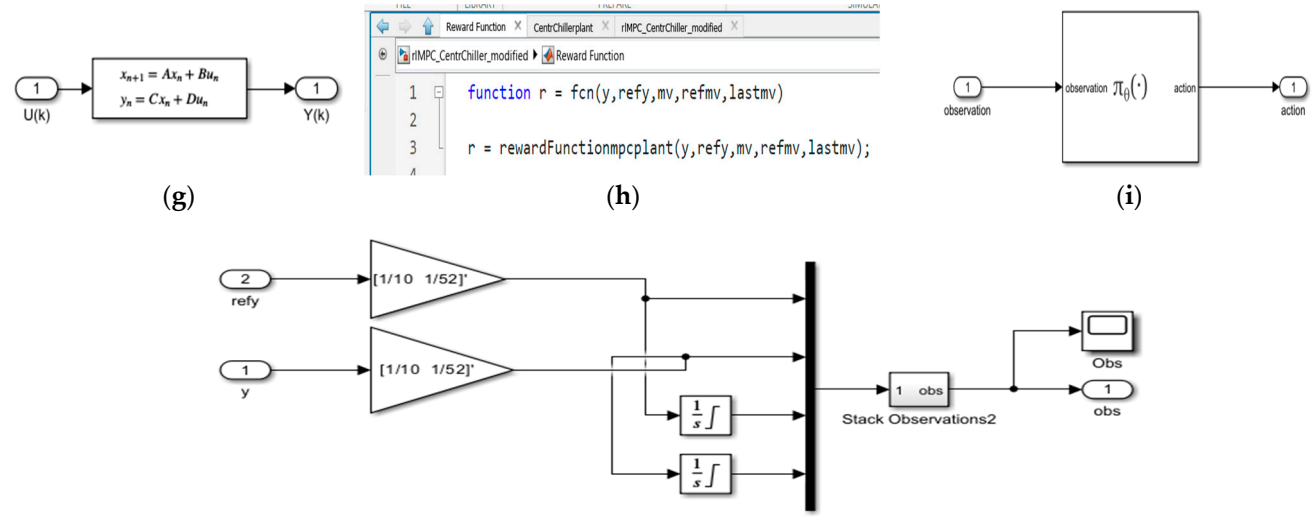
(**d**)


Block Parameters: MPC Controller
MPC (mask) (link)
The MPC Controller block lets you design and simulate a model predictive controller defined in the Model Predictive Control Toolbox.
Parameters
MPC Controller
mpc1obj
mpc
Design
Initial Controller State
Review
Block Options
General
Online Features
Default Conditions
Others
Additional Inports
Measured disturbance (md)
External manipulated variable (ext.mv)
Targets for manipulated variables (mv.target)
Additional Outports
Optimal cost (cost)
Optimal control sequence (mv.seq)
Optimization status (qp.status)
Optimal state sequence (x.seq)
Estimated controller states (est.state)
Optimal output sequence (y.seq)
State Estimation
Use custom state estimation instead of using the built-in Kalman filter (x[k|k])
OK
Cancel
Help
Apply

(**e**)


Block Parameters: RL Agent
RL Agent (mask) (link)
Simulate reinforcement learning agent using a Simulink model as a training and simulation environment.
Block Diagram
observation
reward
isdone
action
RL Agent
Environment
Agent
Agent object
agent
rl.agent.rlTD...
Generate greedy policy block
Additional Input/Output Ports
External action inputs
Last action input
Cumulative reward output
Signal Attributes

(**f**)



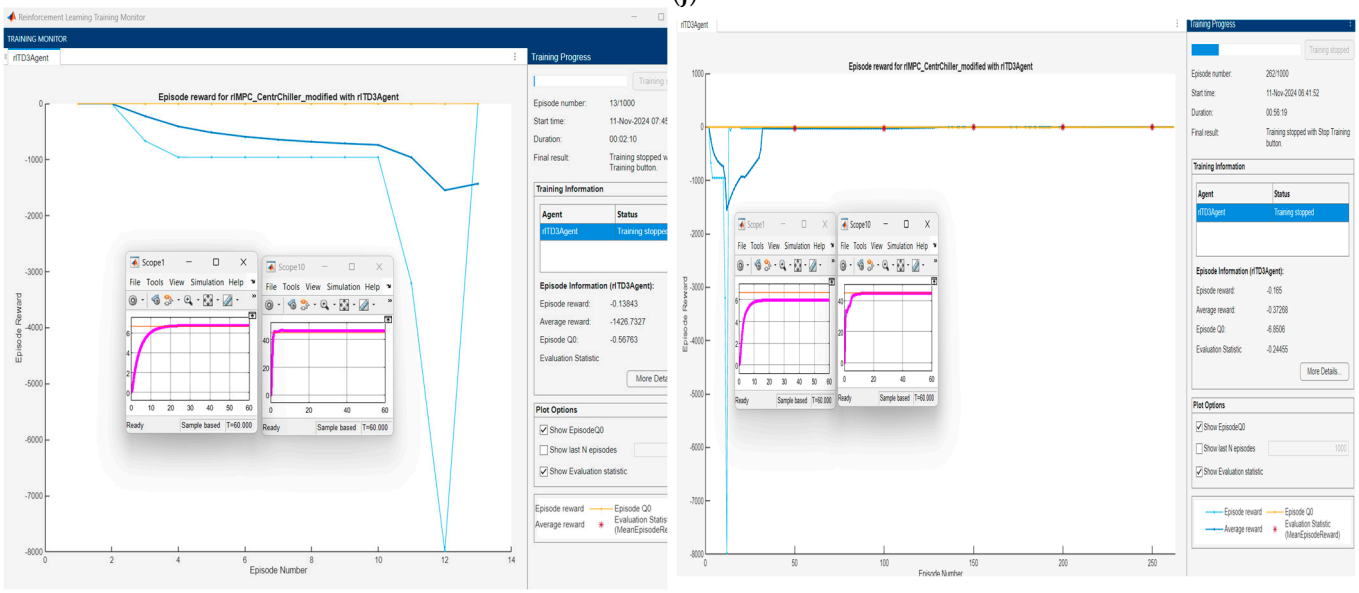


Figure 9. Reinforcement learning control strategy based on deep neural network for a MIMO Centrifugal Chiller simplified model in state-space representation: (**a**) compact Simulink diagram of MIMO Centrifugal Chiller plant and RL DLNN; (**b**) detailed Simulink diagram; (**c**) MPC and RLDNN subsystem; (**d**) visualization block; (**e**) MPC parameters block; (**f**) RL Agent block (controller); (**g**) state-space representation block; (**h**) reward function; (**i**) RL Agent greedy policy block; (**j**) observation block-detail; (**k**) agent training process-print screen snapshot after 13 epochs; (**l**) agent training process-print screen snapshot after 262 epochs.

Combining all the information on Monte Carlo estimation of the expectation μ, a helpful pseudocode algorithm for implementers called Q-learning with ε-greedy policy is given in [23].

The ε-greedy policy is a valuable mechanism for managing the exploration–exploitation balance, as has been stated [23]. It operates according to the following equation:

$$\pi_\varepsilon(S;Q) = \begin{cases} argmax_{a\in\mathcal{A}} \mathrm{Q}(S_t, \mathrm{A}) \text{ with probability } 1 - \varepsilon \\ Uniformly\ random\ action\ in\ \mathcal{A}\ with\ probability\ \varepsilon \end{cases} \quad (25)$$

Due to its simplicity, the ε-greedy policy has become the most widely used method to balance exploitation and exploration. It ensures that most of the time (probability 1-ε), the RL agent exploits its incomplete knowledge of its environment by choosing the best

action, and occasionally (probability ε) explores other actions. There may be some good actions that the agent will never find without exploration [23]. This is why the exploration–exploitation trade-off is an important research topic. Also, for our readers and implementers interested in being well-documented, the pseudocode of the RL Deep Learning NN algorithm described in [17] is reproduced below for its great informal value:

Algorithm 1: RL Deep Learning NN [17]
RL Agent states: Observations $\mathcal{S}$
RL Agent actions; $\mathcal{A}$
Define the optimal policy given by the Bellman equation [23]:

$$Q^{\pi^*}(s,a) = r(s,a) + \gamma(\sum_{s'} p(s'|s,a) \max_{a'} Q^{\pi^*}(s',a') \quad (26)$$

and
$Q^* = Q^{\pi^*}$ is the optimal state-action value function;
Initialize the hyperparameter learning rate α,discount factor γ, and exploration parameter ε.
for t in range (epoch) do
Calculate the action A_t according to the $optimal\ policy$:

$$A_t \leftarrow \begin{cases} argmax_{a\in\mathcal{A}} \mathrm{Q}(S_t, \mathrm{A}) \text{ with probability } 1 - \varepsilon \\ Uniformly\ random\ action\ in\ \mathcal{A}\ with\ probability\ \varepsilon \end{cases} \quad (27)$$

Send the action A_t into the agent environment;
Evaluate the next value of the observation S_{t+1} starting from the previous state S_t according Equation (20) and [23]:

$$S_{t+1} \sim f_{state}(S_{t+1}|S_t, A_t) = \mathcal{P}((\cdot\,|S_t, A_t)) \quad (28)$$

The agent is sensed by the next observation S_{t+1} and the reward function R_{t+1} given by Equations (21) and (22) (could be $r(S_t, A_t)$, or could be stochastic);
Update the new action-value function $Q^{new}(S_t, A_t)$ at state-action (S_t, A_t), according the Equation (23);
Evaluate the estimation using the following loss function in terms of RMSE, MSE, and MAE as defined in [17].
End

The MATLAB software package Simulink Reinforcement Learning Toolbox provides all the valuable tools to create and train reinforcement learning agents (controllers), as previously demonstrated [20,21]. In many practical decision-making problems, the MDP states are high-dimensional and cannot be solved by traditional RL algorithms. Deep reinforcement learning algorithms incorporate deep learning (DL) to solve such MDPs, often representing the $\pi(A_t \mid S_t)$ policy or other learned functions as a neural network (NN) and developing specialized algorithms that perform well in this context. Therefore, in this study, the agent policy (control law) was implemented using deep neural networks (DNNs) created using the most appropriate tools provided in the MATLAB Simulink Deep Learning Toolbox software package [24–26]. Some problems related to the graphics issues encountered during the MATLAB simulations were solved based on the advice provided in [27].

4.1.2. Reinforcement Learning Workflow

The following steps are suggested in [19] to illustrate the RL workflow:

Step 1. Formulate problem: define the task for the RL agent to learn, including its interaction with the environment and primary and secondary achievement goals;

Step 2. Create environment: employ MATLAB or Simulink to define the environment within which the RL agent will operate, including the interface between the agent and environment and the environment dynamic model;

Step 3. Create RL Agent: create the RL agent containing *policy* and a *learning algorithm* [18,19];

A learning algorithm continuously updates the policy parameters based on the actions, observations, and rewards;

Step 4. Create Deep Neural Network Policies and Value Functions: as discussed in Section 4.1.1.

The agents define the DLNN inputs and outputs based on specified actions and observations from the Environment. Also, an actor net and an actor critic representations for the RL agents can be created using Deep Learning Toolbox functionality. Pretrained deep neural networks or deep neural network layer architectures can be imported using the MATLAB Simulink Deep Learning Toolbox network import functionality.

Step 5. Train RL Agent: the RL Agent policy representation is trained using the defined environment, reward, and agent learning algorithm. Training an agent using reinforcement learning is an iterative process.

Step 5. Validate RL Agent: Evaluate the performance of the trained agent by simulating the agent and environment together.

Step 6. Deploy policy: Deploy the trained policy representation using a generation code.

4.2. Deep Learning Neural Network

In this section, the deep learning neural networks (DLNN) is discussed. A deep learning neural network is structured as a series of interconnected layers with multiple nodes called "neurons". Deep learning is used for training multi-layered, allowing them to learn hierarchical representations of datasets in order to make predictions and to generate outputs.

4.2.1. DLNN Architectures, Components, Algorithms and Applications

The deep learning (DL) architecture originated from artificial neural networks (ANNs) that were inspired by the structure of the human brain. Neural network (NN) architectures consist of several components working together to process and learn from massive datasets, extracting patterns and abstract representations of data as information flows through each layer. The components include the input layer, hidden layers, neurons (nodes), weights and biases, activation functions, the output layer, and the loss function; these are well explained in a comprehensive review presented in [24]. In that review, deep learning is described as a multi-layer machine learning method for performing complex tasks. This layered structure is why the deep learning technique differs from conventional machine learning methods; within this structure, the data flows from an input layer through several hidden layers to an output layer. These added hidden layers increase the accuracy performance of the neural network design.

Each neuron performs calculations on the received input and passes the result to the next layer, allowing the network to learn complex patterns from the data [24]. Also, a perceptron simulates a neuron with a set of inputs, each of these having a particular weight, such that the inputs with higher weights have a significant impact on the neural network. The neuron computes and produces an output based on the weighted inputs. Each neuron receives n inputs (features), sums them, applies a transformation (activation) and generates the output. A second parameter called bias is used to adjust the output based on the input weights; thus the model fits the data in the best possible way. An activation function converts the inputs into the output produced using a threshold. Briefly, the neural network architecture serves as the support for the understanding and processing of diverse data types, and generative models unlock the ability to create new data samples that resemble the training data. In the literature [24], it is mentioned that deep learning is "among the fastest-growing areas in computational science, employing complex multi-layered networks to model high level patterns in data". The state-of-the-art of deep learning techniques has been enhanced in several fields, including "visual object recognition, speech recognition, genomics, and discovery of drugs, along with plenty of others". Among the well-known deep learning algorithms discussed [24] are backpropagation models, autoencoders, variational autoencoders, restricted Boltzmann machines, deep belief networks, convolutional natural networks, forward neural networks, recurrent neural

networks, LSTM, generative adversarial networks, transformer deep learning structure, embedding from language models, and bidirectional encoder representations from transformers. A brief overview of the steps involved in using deep learning is expressed in the following sequence:

$$1.\,Data\ Preparation \rightarrow 2.\,Building\ NN \rightarrow 3.\,Training\ NN \rightarrow 4\frac{Evaluation}{Tuning} \rightarrow 5\frac{Prediction}{Generation} \rightarrow 6.\,Iterative\ Improvement$$

These steps can be seen in the MATLAB code lines of both the RL DLNN algorithms developed in the present research, presented in the Annex B.1 Algorithm B.1.1 and B.1.2.

In the first step, the data are collected and preprocessed (cleaning by removing errors or inconsistences, normalization, and splitting, such as for example, 70% into training data and 30% testing data). The data format must be compatible with the NN. The architecture of the NN is defined in the second step. In the third step, the training process is launched for fine-tuning of the network's weights and biases in order to minimize the difference between the predicted output and the true output. This is followed by step 4, evaluation and validation, required for a rigorous evaluation of performance accuracy, followed by a test validation to gather valuable information about how well the NN generalizes to unseen data. In step 5, the trained DLNN evaluated and validated in the previous step acquires good ability to predict or generate results for unseen data. If the proposed DLNN performance does not meet the control objective in the last step, an iterative improvement process is initiated to adjust the DLNN parameters, including modifications to the DLNN architecture using the "trial and error" procedure by changing the number of layers and hidden neurons, or exploring new optimization control algorithms to achieve the proposed control objectives.

4.2.2. NN Table Models

Neural network tables can take various forms depending on the specific application and the type of data being processed. The most usual NN tables built to analyze NN performance using evaluation metrics include the confusion matrix, the weight matrix, activation tables, and loss tables. In the following, we examine these particular tables.

1. **Confusion Matrix**: This type of table is used to evaluate the performance of a classification model. It shows the numbers of true positives, true negatives, false positives, and false negatives. A generic example is presented as follows:

Actual/Predicted	Positive	Negative
Actual	60	20
Predicted	10	25

2. **Weight Matrix**: In each NN, the weights are critical parameters that transform input data within the network. A weight matrix represents the weights between layers of neurons, for which a simple matrix might look like this:

Neuron k	Neuron k + 1	Neuron k + 2
0.2	0.3	−0.1
−0.3	0.5	0.2

3. **Activation Table**: This type of table illustrates the activation values of the neurons in a NN for a given input, as shown below:

Input k	Input k + 1	Neuron k activation	Neuron k + 1 activation
0.4	0.2	0.5	0.5
0.3	0.7	0.2	0.8

4. **Loss Table**: This table tracks the loss values during NN training, a valuable information for understanding how well the model is learning over time; a figurative example is presented below:

Epoch	Training Loss	Validation Loss
1	0.32	0.35
2	0.15	0.2
3	0.25	0.25

These tables are essential for analyzing and understanding the performance and behavior of neural networks in various applications. Appendix B2 details two DLNN tables, namely the Table A1 and Table A2 with parameter simulation and monitoring training for the two proposed RL DLNN control algorithms developed in Section 4.3. Since both RL DLNN algorithms solve a tracking reference control regression problem, the most suitable evaluation metrics are RMSE and MSE [17]. The strong dependence of reward function of RMSE and MSE gives us the possibility of using an evaluation metric based on reward function, since the problem of minimization of RMSE or MSE to the smallest possible value closest to zero becomes a maximization problem of reward function, for which the most appropriate metrics are the cumulative reward value (return value) or average reward value. The evaluation metric used for DLNN performance in both NN tables is the average reward value for each training episode, since this value appears automatically in the monitoring of the RL agents' training progress, as shown in Figures 9k,l and 11m and Tables A1 and Table A2 from Appendix B2. The RL agents are trained using the adopted RL DLNNs control algorithms whose reward functions are generated from MPC and the step response specifications of the MIMO Centrifugal Chiller System discussed below (paragraphs 4.3.1 and 4.3.2).

4.3. Reinforcement Learning Deep Learning Neural Networks Closed-Loop Control Strategies Applied to a Centrifugal Chiller System

In this section, we develop two advanced intelligent reinforcement learning deep learning neural network (RLDLNN) control algorithms to control the evaporator temperature and the level of refrigerant in the condenser. The first algorithm generates a reward function for an RL Agent based on model predictive control (MPC) specifications and the second algorithm generates two reward functions for two RL agents (controllers) based on the step response block specifications.

4.3.1. Generate Reward Function from the Cost and Constraint Specifications Defined in an MPC Object

This intelligent control strategy and the training phase results were implemented in MATLAB Simulink in Figure 9.

Its compact and detailed architecture is presented in Figure 9a–d and is of practical interest. It automatically generates the reward function from the cost and constraint specifications defined in a Simulink model predictive controller (MPC) object, as shown in Figure 9e. It is connected in a closed loop with the MIMO centrifugal cooling plant model given by Equation (6) and Equation (7) in Section 2.2 to control the chilled water temperature Tchw-sp in the evaporator subsystem and the level of liquid refrigerant in the second condenser subsystem, depicted in the state-space representation shown in Figure 9g. Implementing this control strategy follows the same procedure that is well described in the updated MathWorks documentation for MIMO Centrifugal Chiller System (CCS) model [18,19]. The MPC object appearing at the top of Figure 9b,c with its parameters block depicted in Figure 9e was created based on the CCS MIMO model, using an inline MATLAB code procedure. For the CCS MIMO plant model under investigation, the sampling time value is set to Ts = 0.1 s, the prediction horizon was assumed to be p = 10 steps, and the control horizon m = 2 steps.

The following specifications of the MPC object are helpful for RL DLNN MPC strategy design:

- Standard linear bounds for output variables (OVs) and manipulated variables (MVs):

y_{min} = [0 0], y_{max} = [10 52], mv_{min} = 0, mv_{max} = 1.1, $mv_{ratemin}$ = −1000, and $mv_{ratemax}$ = Inf

- Scale factors as specified for OVs and MVs:

$$S_y = [10\ 52],\ S_{mv} = [1.1\ 1]$$

- Standard cost weights:

$$Q_y = [0.1\ 0.1],\ Q_{mv} = [0\ 0], \text{ and } Q_{mvrate} = [0.1\ 0.1]$$

The reward function generated based on the MPC object specifications was the starting point for the reward design. The reward function is used to train the RL Agent, as shown in Figure 9f. The implementer can modify the reward function with different penalty function options and adjust the weights:

The reward function requires the following two components to be computed:

1. The cost component, calculated according to the following equations:

$$dy = \big(refy(:) - y(:)\big)./{S_y}^T \tag{18}$$

$$dmv = \big(refmv(:) - mv(:)\big)./{S_{mv}}^T \tag{19}$$

$$dmvrate = \big(mv(:) - lastmv(:)\big)./{S_{mv}}^T \tag{20}$$

$$J_y = dy^T diag(Q_y.\hat{}2)dy \tag{21}$$

$$J_{mv} = dmv^T diag(Q_{mv}.\hat{}2)dmv \tag{22}$$

$$J_{mvrate} = dmvrate^T diag(Q_{mvrate}.\hat{}2)dmvrate \tag{23}$$

$$Cost = J_y + J_{mv} + J_{mvrate} \tag{24}$$

2. The penalty component for violation of linear bound constraints, with the following components:
   - Penalty function weight (specify nonnegative):

$$W_y = [1\ 1], W_{mv} = [10\ 10],\ W_{mvrate} = [10\ 10] \tag{25}$$

     - Choose the **step** or **quadratic** penalty method to calculate the **exteriorPenalty**;
     - Set the Pmv value to 0 if the RL Agent action specification has appropriate "*LowerLimit*" and "*UpperLimit*" values.

   - Penalty functions:

$$P_y = W_y(exteriorPenalty(y, y_{min}, y_{max}, '\boldsymbol{step}')) \tag{26}$$

$$P_{mv} = W_{mv}(exteriorPenalty(m_v, m_{vmin}, m_{vmax}, '\boldsymbol{step}')) \tag{27}$$

$$P_{mvrate} = W_{mvrate}(exteriorPenalty(m_v - lastmv, m_{vratemin}, m_{vratemax}, '\boldsymbol{step}')) \tag{28}$$

and finally:

$$Penalty = P_y + P_{mv} + P_{mvrate} \quad (29)$$

To calculate the reward value as a result of the MATLAB function block illustrated in Figure 9h, the following relationship between all three components—reward, cost, and penalty—is used:

$$reward = -(Cost + Penalty) \quad (30)$$

For the actual control application under research, the specifications of the RL DLNN environment were the following:

- The observations reference signals (Y_{1sp} and Y_{2sp}), output variables (T_{chw} and *Level*), and their integrals, were as shown in Figure 9j;
- The T_{chw} and *Level* signals were normalized by multiplying with the gain [1/10 1/52];
- The action of U_{com} and u_{EXV} was limited to between [0 1.1] for $U_{com}$and [0 1] for u_{EXV};
- The sample time and total simulation time were $T_s = 0.1\ s$. In order to capture the full evolution of the dynamics for both plant outputs, the simulation time was set to Tsim = 60 s.

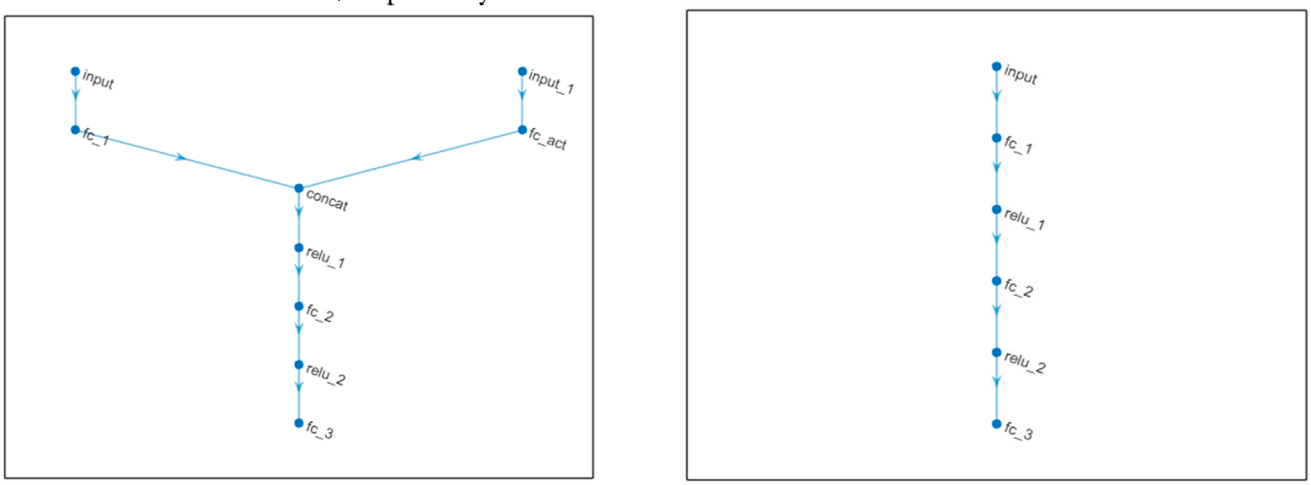
The MATLAB Simulink simulations result is presented in Figure 9.

The block parameters and the block diagram for simulation of reinforcement learning of the RL Agent using a Simulink model as a training and simulation environment are depicted in Figure 9f. The same block also generates the RL Agent greedy policy $\pi_0(.)$, as shown in Figure 9i. Figure 9h shows a screen capture of the MATLAB reward function, and Figure 9j presents a detail of the observation block that appears in Figure 9b,c.

The RL agent chosen in the case study was a twin-delayed deep deterministic policy gradient (TD3) agent [20,21] that used two parametrized Q-value function approximators to estimate the value (that is, the expected cumulative long-term reward) of the policy, as shown in Figure 9f,i. To model the parametrized Q-value function, a neural network that was used had two inputs (the observation and action) and one output corresponding to the value of the policy $\pi_0(\,)$ when taking a given action from the state corresponding to a given observation, as shown in Figure 9i. Before training the RL Agent, each network path of the DLNN was defined as an array of layer objects, and names assigned to the input and output layers of each path in order to connect the paths. Thus, a layer graph object was created and layers aded to generate the criticNet, as is shown in Figure 10a. The critic function objects were created using a dedicated MATLAB *rlQValueFunction* command. To make sure the critics had different initial weights, each network was explicitly initialized before being used create critic 1 and critic 2 [20,21]. The TD3 agent learns a parametrized deterministic policy over continuous action spaces. The policy is learned by a continuous deterministic actor that takes the current observation as input and returns as output an action that is a deterministic function of the observation. The neural network that is used to model the parametrized policy within the actor has one input layer (which receives the content of the environment observation channel) and one output layer (which returns the action to the environment action channel) [20,21]. An actor network named actorNet is created; its layer graph is shown in Figure 10b. The deterministic actor function is generated for the purposes of modeling the policy of the RL Agent. A set of agent options is specified to train the agent from an experience buffer of maximum capacity 1×10^6 by randomly selecting mini-batches of size 256. It has been reported that a discount factor of 0.995 favors long-term rewards [20,21]. The optimizer options are specified for the actor and critic functions: a learning rate of 1×10^{-3} and a gradient threshold of 1, as set out in [20,21]. During training, the agent explores the action space using a Gaussian noise model of action. The standard deviation and decay rate of the noise are set by using an ExplorationModel property, as is shown in [20,21]. The RL Agent was trained using *train* function, as depicted in Figure 9k; after 13 epochs, it is produced the best tracking performance for chilled water temperature inside the evaporator and for liquid refrigerant level within the

condenser. Figure 9l also illustrates the results of the training process after 262 epochs, when the tracking performance of the liquid refrigerant level in condenser was at its best, while that for the chilled water temperature inside the evaporator was slightly attenuated compared with the first result obtained, as shown in Figure 9k. A snapshot of the main steps of the MPC RL DLNN Algorithm 1 is presented in Appendix B1.

The layer graphs of criticNet and actorNet are displayed in Figure 10a and Figure 10b, respectively.



(**a**) (**b**)

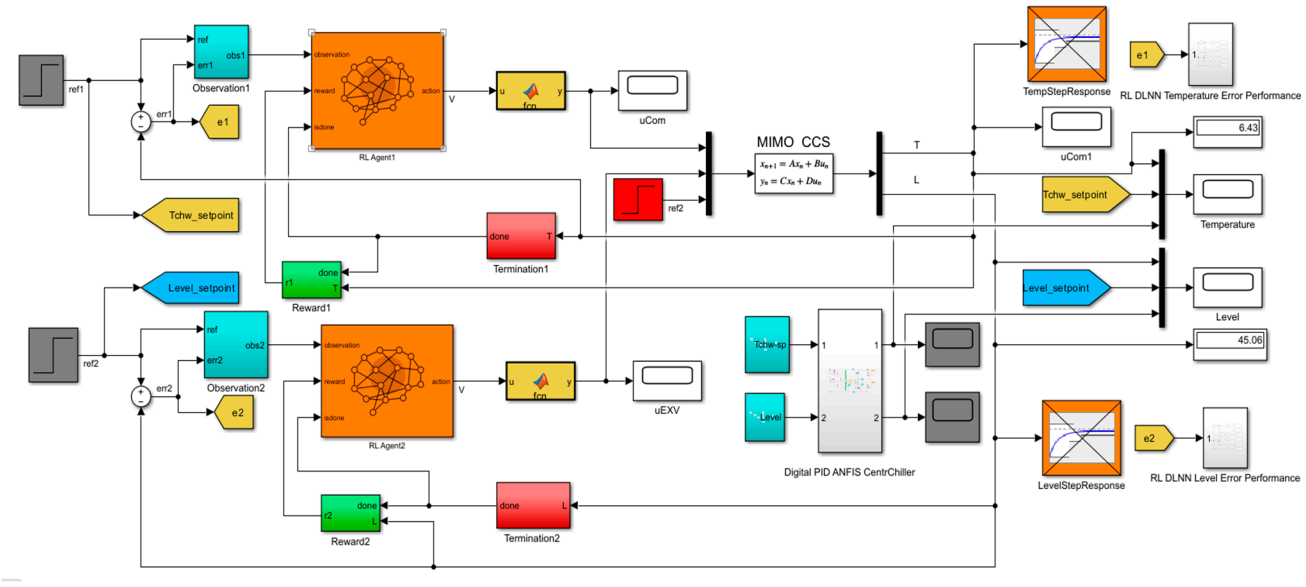
Figure 10. Layer graphs for (**a**) criticNet; (**b**) actorNet.

4.3.2. Reinforcement Learning Deep Learning Neural Network Control Strategy—Generate Reward Function from a Step Response Specifications of MIMO Centrifugal Chiller Simplified Model in State-Space Representation

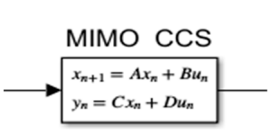
Both The digital PID and RL DLNN closed-loop control strategies are depicted in Figure 11a in a compact Simulink diagram architecture. A description in state space of MIMO CCS model is provided in Figure 11b. Figure 11c shows the block parameters of RL Agent1, and Figure 11d details the observations block as a component of the same compact Simulink diagram. The compact Simulink diagram also shows how to automatically generate two reward functions—rewardfunctionVfb1 and rewardfunctionVfb2, shown in Figure 11e,g from the performance requirements defined in the Simulink Design Optimization model verification blocks provided in MATLAB R2024a Simulink Toolbox for evaporator temperature control (chosen as the first agent, RL Agent1) and condenser liquid refrigerant level control (the second agent, RL Agent2), respectively. Also in the compact Simulink diagram appear two Boolean logic termination blocks, which stop the training of the RL Agent1 and RL Agent2 when the evaporator temperature and the condenser level reach good accuracy performance, as shown in Figure 11f,h. The rewardfunctionVfb1 and rewardfunctionVfb2 are used to train both reinforcement learning RL Agent1 and RL Agent2, following the stated steps and the MATLAB Reinforcement Learning subroutine procedure similar to those developed in [20,21] applied for MIMO plants. The training of both agents, RL Agent1 and RL Agent2, is performed similarly to the MPC RL DLNN procedure described above in Subsection 4.3.1. Compared with MPC RL DLNN, both RL agents are decentralized with two distinct paths. The untrained simulation results for evaporator temperature control and condenser liquid refrigerant level control are presented in Figure 11i,j, respectively. Figure 11k,l presents the training simulation results for the evaporator temperature and liquid refrigerant level inside of the condenser after 200/200 epochs. As shown in Figure 11m, the training process of both RL Agent1 and RL Agent2 finishes after the agents reach the criteria to stop training set out in the training options MATLAB subroutine presented in the print screen snapshot of

Algorithm 2 from Annex B1. The penalty function weight is specified nonnegative and set to the value 2. To compute the penalty for violation of linear bound constraints, the same MATLAB functions were used as described in previous section for MPC strategy. The user can specify the penalty method as step or quadratic; the quadratic is typically preferred.

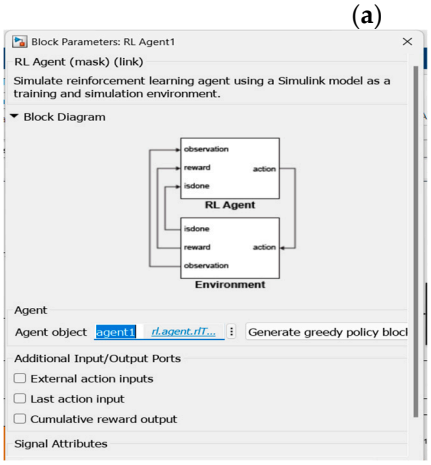
The corresponding MATLAB code lines inside the MATLAB blocks built based on the step response specifications for each reward function are presented in the same Algorithm 2 from Annex B1. The MATLAB Simulink simulations results are shown in Section 6.2, below.



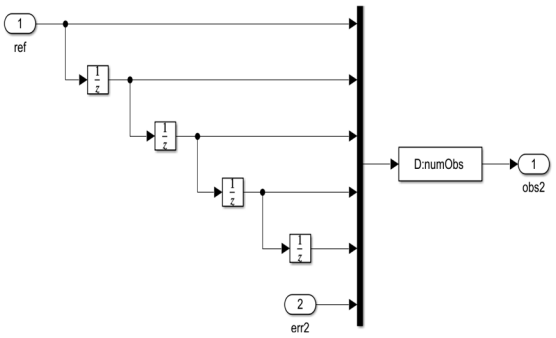
(**a**)



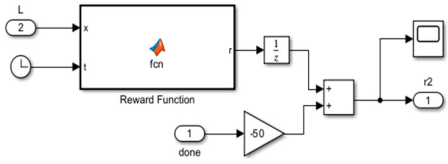
(**b**)



(**c**)



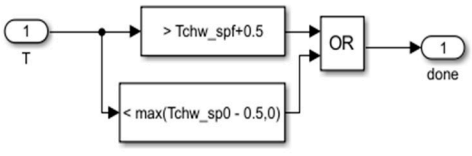
(**d**)



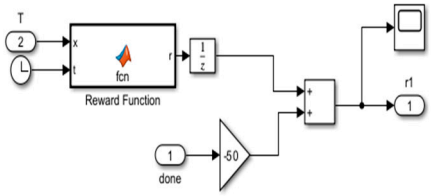

```
function r = fcn(x,t)

r=rewardFunctionVfb1(x,t);
end
```

(**e**)



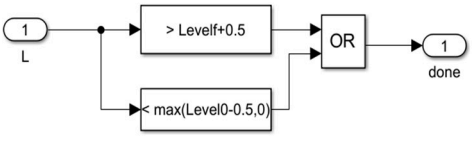
(**f**)



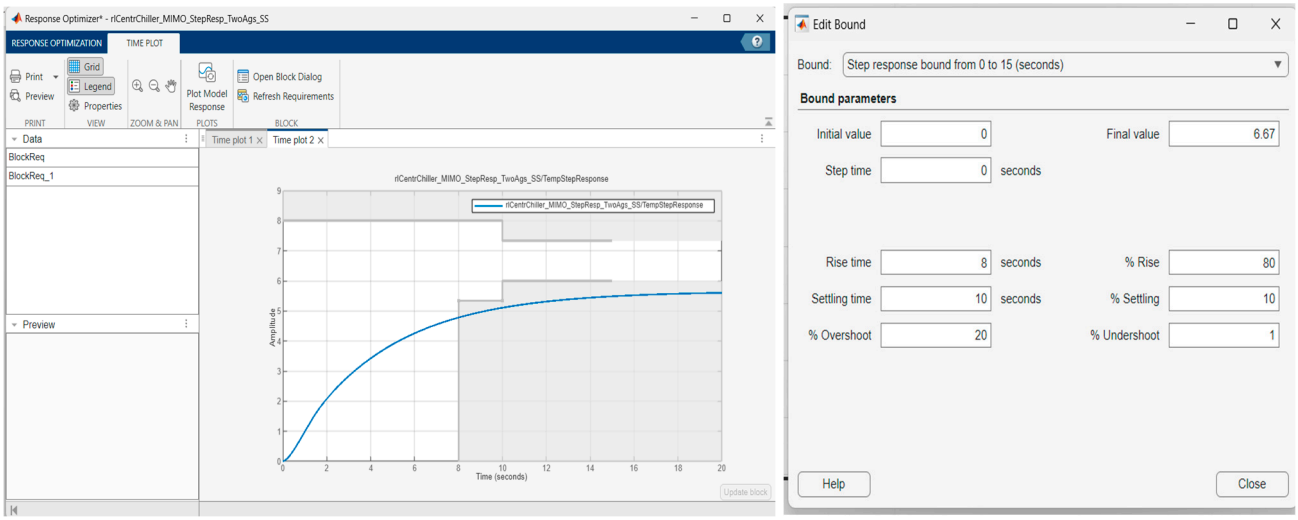

```
function r = fcn(x,t)

r=rewardFunctionVfb2(x,t);
end
```

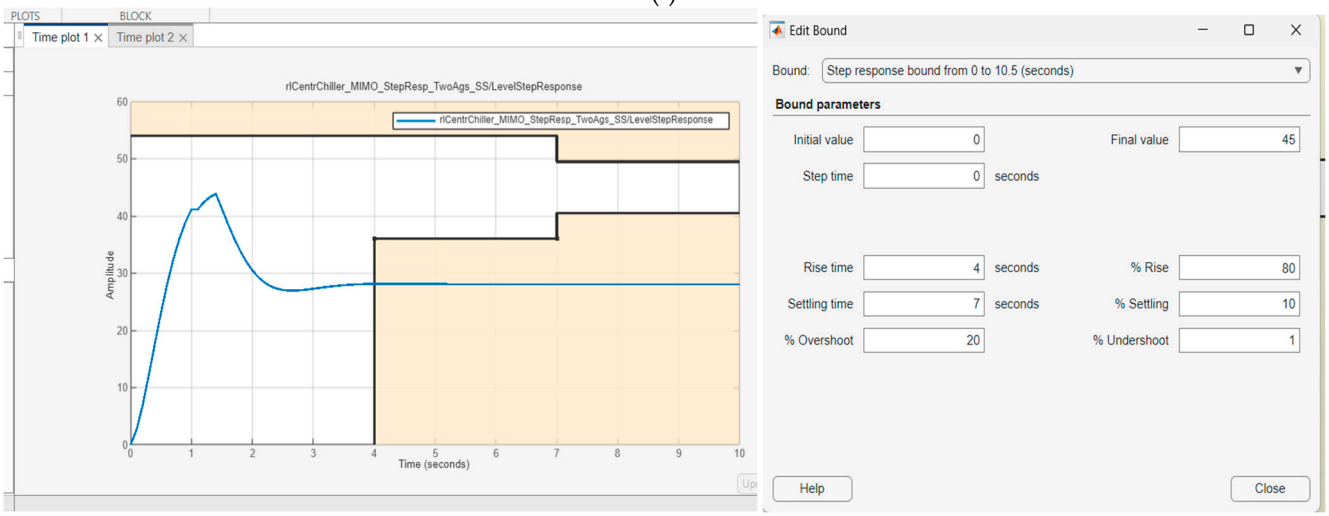
(**g**)



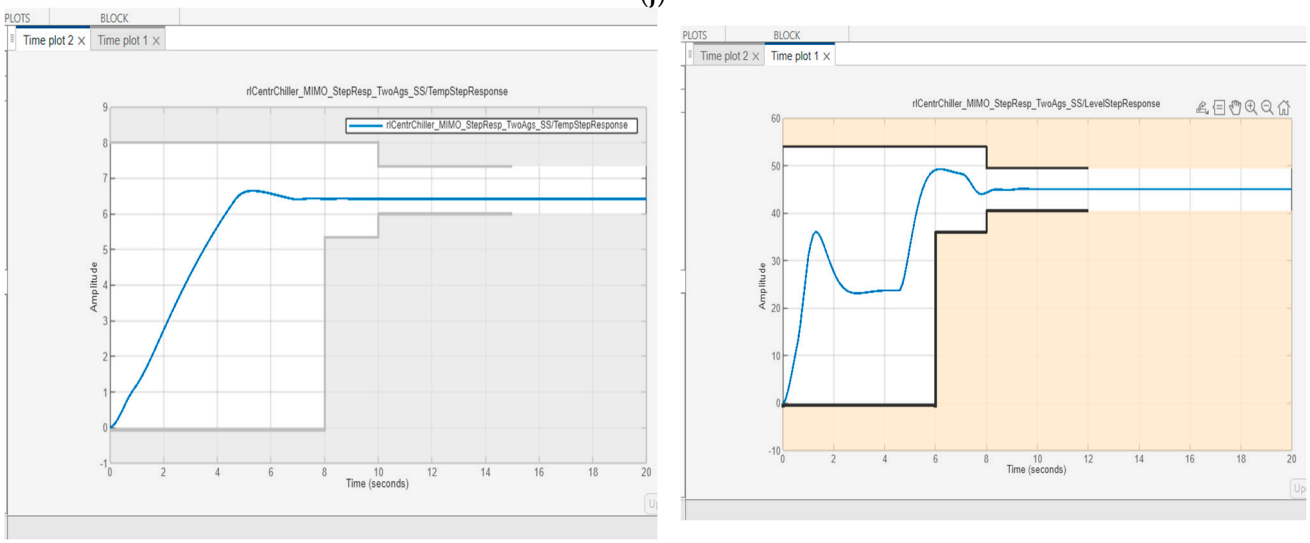
(**h**)


Response Optimizer* - rlCentrChiller_MIMO_StepResp_TwoAgs_SS
RESPONSE OPTIMIZATION
TIME PLOT
Print
Preview
Grid
Legend
Properties
Plot Model Response
Open Block Dialog
Refresh Requirements
Data
BlockReq
BlockReq_1
Preview
Time plot 1
Time plot 2
rlCentrChiller_MIMO_StepResp_TwoAgs_SS/TempStepResponse
Amplitude
Time (seconds)
Edit Bound
Bound: Step response bound from 0 to 15 (seconds)
Bound parameters
Initial value 0
Final value 6.67
Step time 0 seconds
Rise time 8 seconds
% Rise 80
Settling time 10 seconds
% Settling 10
% Overshoot 20
% Undershoot 1
Help
Close

(**i**)


PLOTS
BLOCK
Time plot 1
Time plot 2
rlCentrChiller_MIMO_StepResp_TwoAgs_SS/LevelStepResponse
Amplitude
Time (seconds)
Edit Bound
Bound: Step response bound from 0 to 10.5 (seconds)
Bound parameters
Initial value 0
Final value 45
Step time 0 seconds
Rise time 4 seconds
% Rise 80
Settling time 7 seconds
% Settling 10
% Overshoot 20
% Undershoot 1
Help
Close

(**j**)


PLOTS
BLOCK
Time plot 2
Time plot 1
rlCentrChiller_MIMO_StepResp_TwoAgs_SS/TempStepResponse
Amplitude
Time (seconds)
rlCentrChiller_MIMO_StepResp_TwoAgs_SS/LevelStepResponse

(**k**) (**l**)

(**m**)

Figure 11. Reinforcement learning deep learning neural network Simulink diagram: (**a**) overall Simulink diagram of RL DLNN control architecture and MIMO CCS; (**b**) MIMO CCS state-space model; (**c**) RL Agent1 block; (**d**) observation block; (**e**) Reward Function Vfb1 block; (**f**) termination block for first RL Agent1; (**g**) Reward Function Vfb2 block; (**h**) termination block for first RL Agent2; (**i**) chilled water temperature within evaporator and step response block specifications-untrained RL1 Agent; (**j**) liquid refrigerant level inside condenser and step response block specifications-untrained RL Agent2; (**k**) chilled water temperature within evaporator-trained RL1 Agent after 33/200 epochs; (**l**) liquid refrigerant level inside condenser-trained RL2 Agent after 33/200 epochs; (**m**) training process ofRL1 and RL2 Agents.

5. Traditional and Advanced Intelligent Closed-Loop Control Strategies—MATLAB Simulink Simulation Results

This section presents the MATLAB Simulink simulation results of traditional closed-loop control strategies. Subsection 5.1.1 discusses DTI control of the MIMO CCS, and in Subsection 5.1.2 is depicted the simulations result for a PID control of MIMO CCS extended nonlinear model with 39 states, inputs subjected to constraints, and under measured temperature disturbance. In Section 5.1.3 are shown the simulation results of an improved version of PID control, more precisely, a digital PID control of a MIMO CCS ANFIS model. Model predictive control of the MIMO CCS model represented in state space with four states is discussed in Subsection 5.1.4. Also, the reinforcement learning MPC deep learning neural network (RL DLNN) control MIMO CCS model is compared versus MPC control in Subsection 5.2.1, and the RL DLNN control of MIMO CCS model in state-

space representation versus the improved digital PID control of MIMO CCS ANFIS model is compared in Subsection 5.2.2.

5.1. Traditional Closed-Loop Control Strategies

5.1.1. DTI Closed-Loop Control

The simulation results are presented in Figure A2 in Appendix A. Figure A2a shows the Simulink diagram of the DTI controller. Figure A2b depicts the DTI control of chilled water temperature control within the evaporator subsystem, and Figure A2c the liquid refrigerant level control inside the condenser subsystem. Figure A2d presents the compressor and expansion valve opening actuator control efforts.

5.1.2. PID Closed-Loop Control—Centrifugal Chiller Extended Model (39 States)

The Simulink simulation result is depicted in Figure 12. Figure 12a presents the PID MIMO Centrifugal Chiller closed-loop temperature control inside the evaporator subsystem; in Figure 12b, the results for PID control of the liquid refrigerant level in the condenser subsystem are revealed. Figure 12c depicts the compressor actuator control effort and Figure 12d discloses the expansion valve opening actuator control effort.

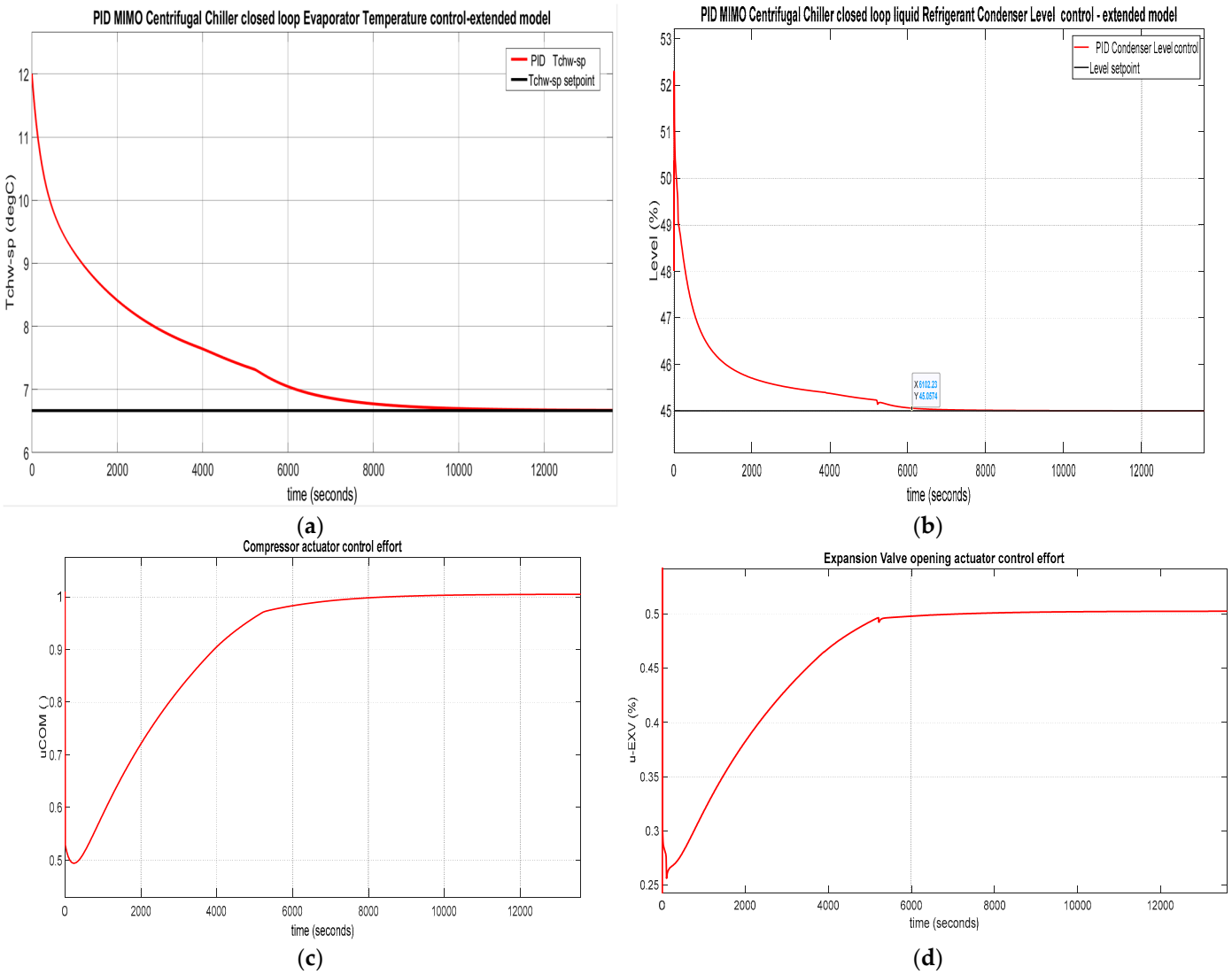
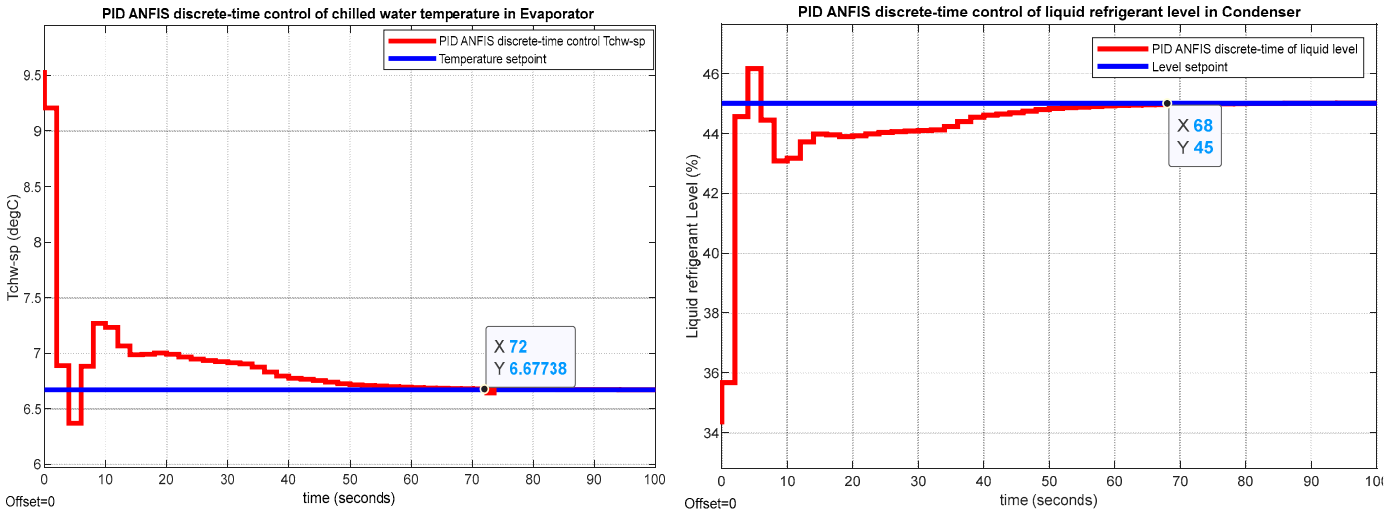


Figure 12. MATLAB Simulink simulation results: (**a**) PID evaporator subsystem temperature control; (**b**) PID liquid refrigerant level control in condenser subsystem; (**c**) compressor relative speed actuator control effort; (**d**) expansion valve opening actuator control effort.

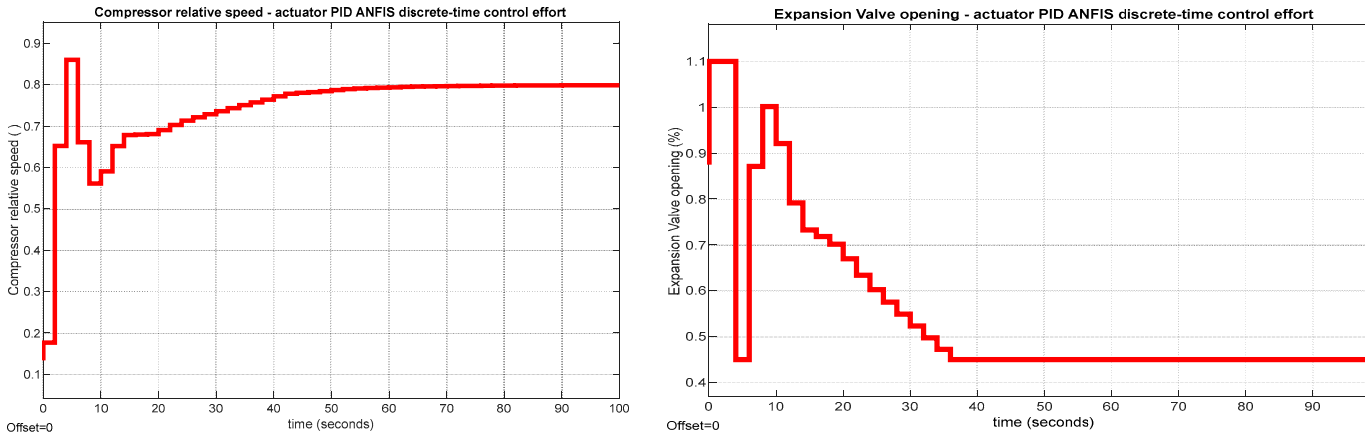
5.1.3. Digital PID Control of MIMO Centrifugal Chiller ANFIS Model

The MATLAB Simulink simulations results are revealed in Figures 13 and 14, and also in Figures A4–A7 in Appendix A. The MATLAB Simulink simulation results for the digital PID control CCS ANFIS model without changes in temperature and level setpoints are depicted in Figure 13a and Figure 13b, respectively. The actuators' control efforts for the compressor and expansion valve opening are shown in Figure 14a,b. On simple visual inspection of both figures, it seems that the fast digital PID control had a fast step response, reaching zero steady-state for a settling time of 50 s with a 9% overshoot for chilled water inside the evaporator and 2% for liquid refrigerant level within the condenser.



(**a**) (**b**)

Figure 13. MATLAB Simulink simulation results without changes in temperature and level setpoints: (**a**) PID ANFIS discrete-time control of Tchw-sp in evaporator; (**b**) PID ANFIS discrete-time control of liquid refrigerant level in condenser.



(**a**) (**b**)

Figure 14. PID ANFIS discrete-time control—MATLAB Simulink simulation results of actuator control efforts: (**a**) compressor relative speed; (**b**) expansion valve opening.

The ability of the digital PID controller to overcome the effects of temperature disturbance is revealed in Figure A4a for temperature and in Figure A4b for liquid refrigerant level, respectively. Figure A5a shows the impact of temperature disturbance on the compressor actuator control effort, and Figure A5b on expansion valve opening actuator effort, respectively. Figure A6a shows the evolution of the chilled water temperature inside the evaporator subsystem, and Figure A6b shows the evolution of the liquid refrigerant level in the condenser subsystem with changes in setpoint values for temperature and

level, respectively. The impact of these setpoint changes on the compressor and expansion valve opening actuator control efforts are shown in Figure A7a and Figure A7b, respectively.

5.1.4. Model Predictive Control of MIMO Centrifugal Chiller Nonlinear Extended Model in State-Space Representation (39 States) with Input Constraints

The MPC simulation results are described in Figure 15a–d. The control effects of both actuators, compressor relative speed and expansion valve opening, are represented in Figure 15a. Figure 15b reveals the chilled water temperature disturbance, Trr. Figure 15c shows the MPC of chilled water temperature (OV1) inside the evaporator in degrees Celsius, and Figure 15d displays the liquid refrigerant level (OV2) within the condenser with a change in temperature disturbance Trr from 48 [degF] to 54 [degF] applied at time instant t = 40 [s].

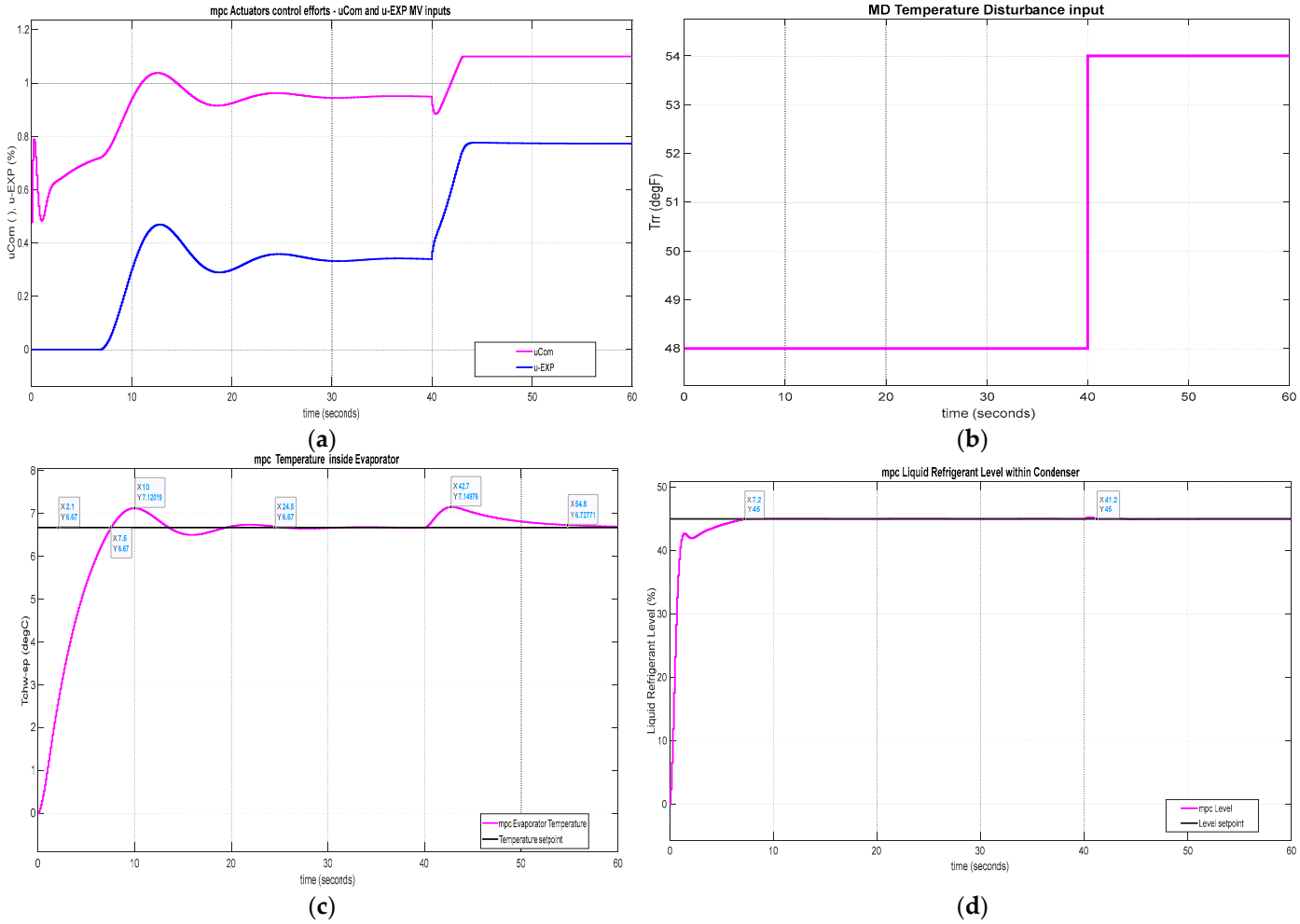


Figure 15. MPC MATLAB Simulink simulations resulted in a closed loop for input (u1=Ucom, u2=uEXV, u3=Trr) and output (y1=Tch-sp, y2=Level) responses against internal plant using mpcDesigner application based on the input–output measurement dataset for the MIMO CCS extended state model: (**a**) actuator control effort; (**b**) temperature disturbance; (**c**) chilled water temperature in evaporator; (**d**) liquid refrigerant level in condenser;

5.2. Advanced Reinforcement Learning Using Deep Learning Neural Network Control Strategies

This subsection presents the simulation results of the advanced reinforcement learning deep learning neural networks (RL DLNNs) developed in Section 4, conducted on the MATLAB Simulink R2024 software programming platform.

The Section .2.1 refers to the Simulink simulation results for the RL DLNN that generated the reward function from an MPC dataset extracted from the MIMO CCS simplified model represented in state-space, given in Equation (7). Section 5.2.2 presents the simulations result for the RL DLNN that generated the reward function from two step responses' block specifications for the same MIMO CCS simplified model in state-space representation.

5.2.1. Reinforcement Learning Deep Learning Neural Network Control Strategies–Generate Reward Function from MPC of MIMO Centrifugal Chiller Simplified Model in State-Space Representation

The Simulink simulations results are shown in Figure 16a–c. Results for the RL DLNN control of evaporator chilled water temperature are displayed in Figure 16a, and Figure 16b represents the liquid refrigerant level in the condenser, both versus their corresponding MPC step responses on the same graphs. The Figure 16c illustrates the separate RL DLL MPC control efforts.

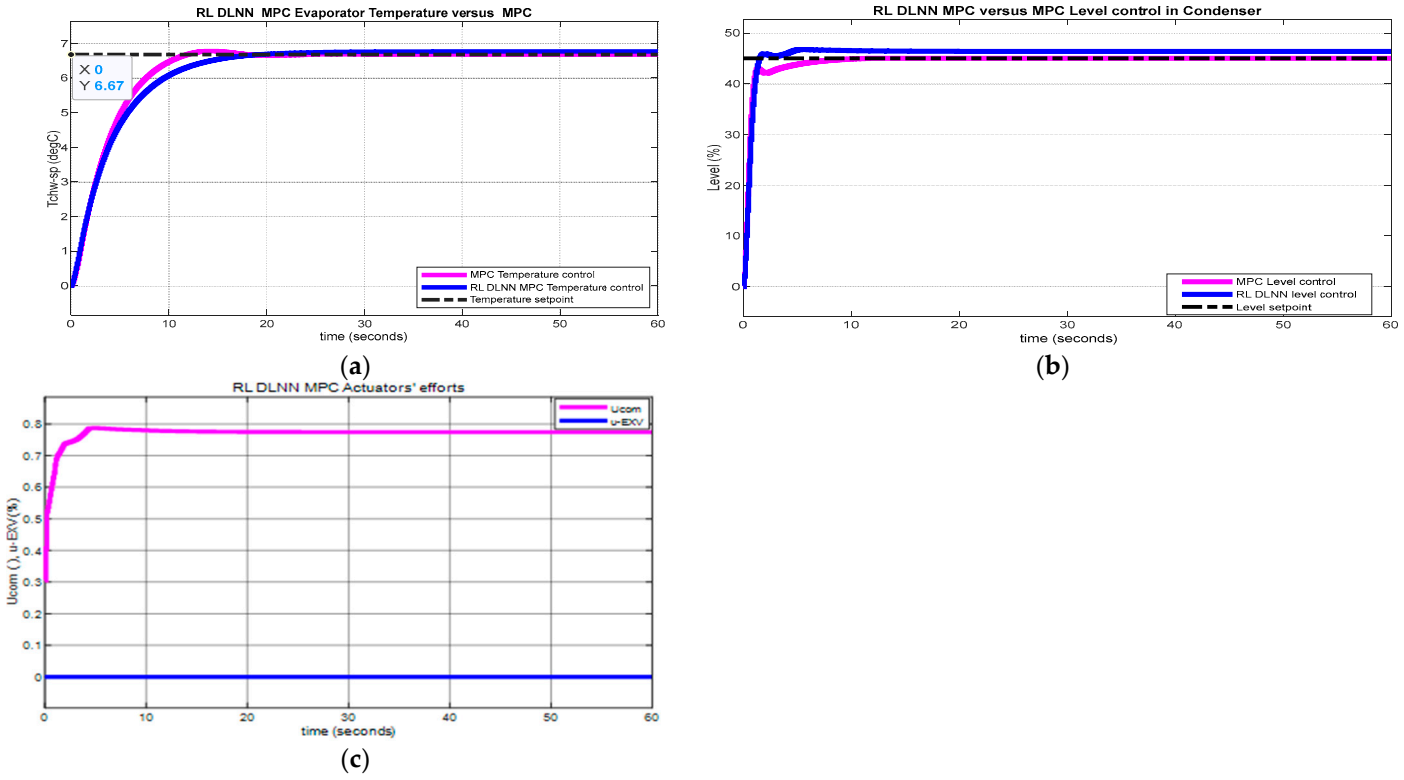
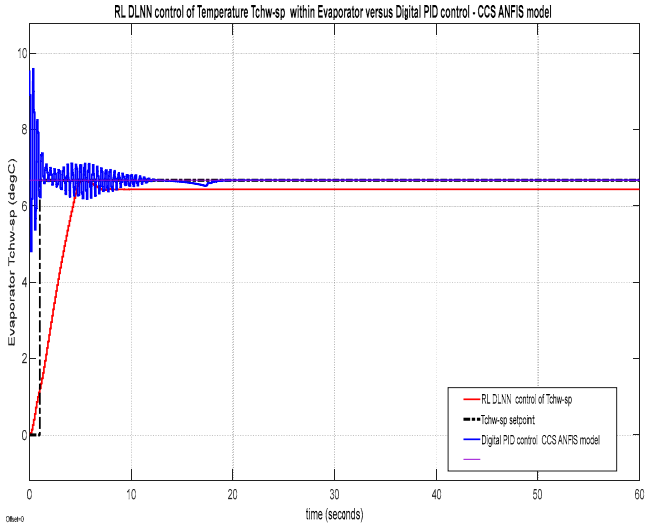


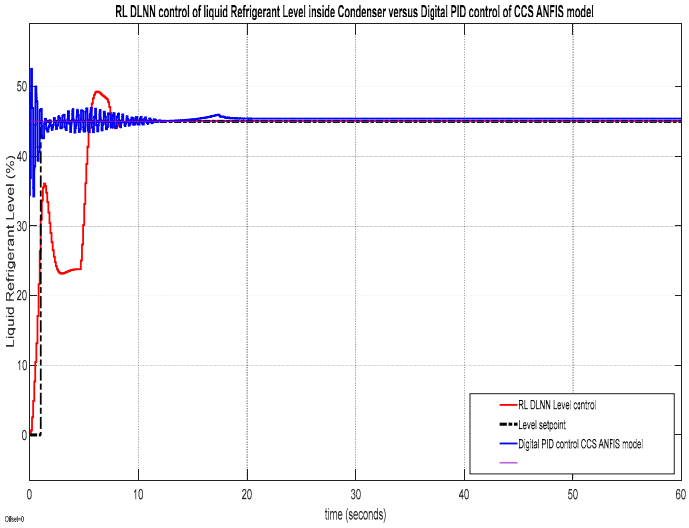
Figure 16. RL DLNN—Generate reward function based on MPC specifications: (**a**) RL DLNN MPC evaporator temperature control versus MPC; (**b**) RL DLNN MPC condenser level control versus MPC; (**c**) RL DLNN MPC actuator efforts.

5.2.2. Reinforcement Learning Deep Learning Neural Network Control Strategies—Generate Reward Function from Step Response Specifications of a MIMO Centrifugal Chiller Simplified Model in State-Space Representation

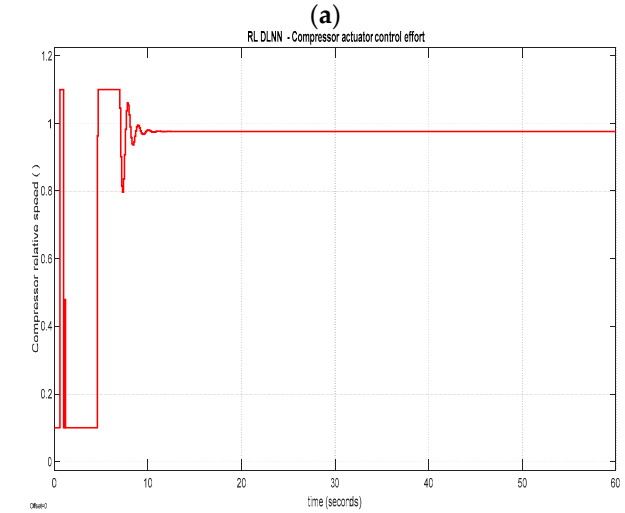
The simulation results for RL DLNN control of evaporator chilled water temperature are depicted in Figure 17a, and Figure 17b illustrated the liquid refrigerant levels versus digital PID control of MIMO ANFIS model responses.



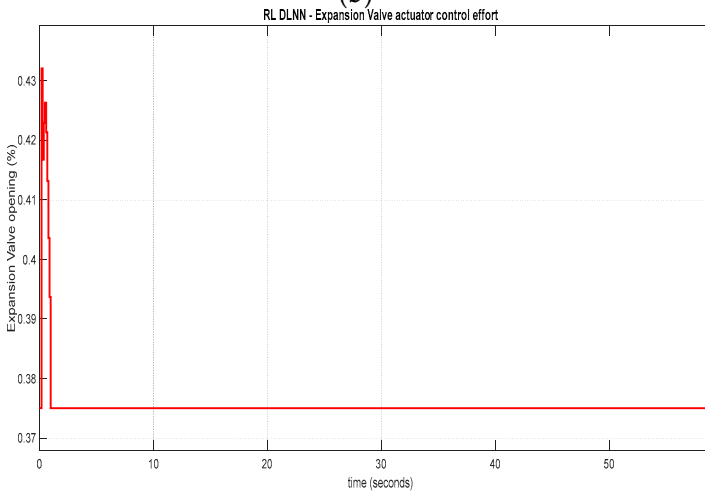
(**a**)



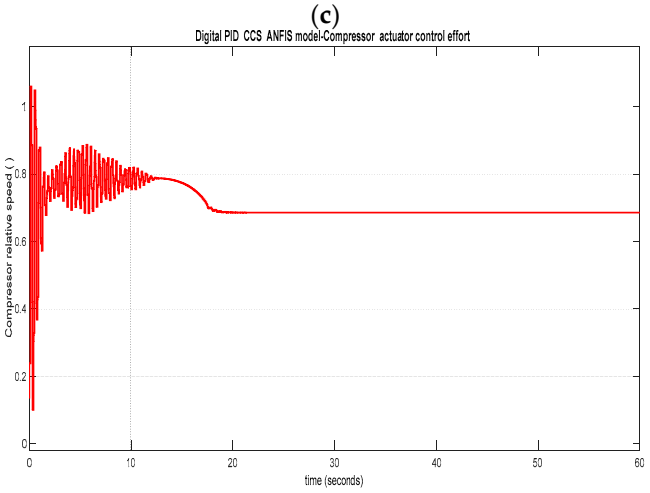
(**b**)



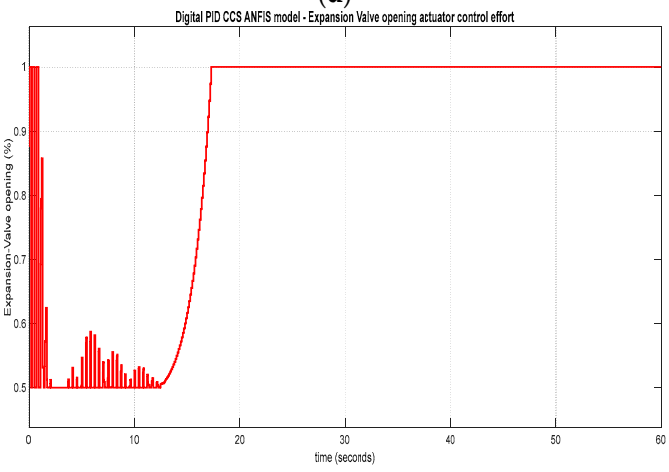
(**c**)



(**d**)



(**e**)



(**f**)

Figure 17. MATLAB Simulink simulation results of RL DLNN versus digital PID MIMO ANFIS model generate reward function from the step response specifications of the MIMO Centrifugal Chiller simplified model in state-space representation. (**a**) RL DLNN of chilled water temperature inside the evaporator versus digital PID controller using ANFIS CCS models; (**b**) RL DLNN control of liquid refrigerant level within the condenser versus digital PID controller using ANFIS CCS models; (**c**) compressor actuator control effort, RL DLNN; (**d**) expansion valve opening actuator control effort, RL DLNN; (**e**) compressor actuator control effort, digital PID CCS ANFIS model; (**f**) expansion valve opening actuator control effort, digital PID CCS ANFIS model.

The RL DLNN actuators control efforts for the compressor and for the expansion valve opening is illustrated in Figure 17c and Figure 17d, respectively, while the control efforts of digital PID control CCS ANFIS models are shown in Figure 17e for the compressor actuator and Figure 17f for expansion valve opening actuator.

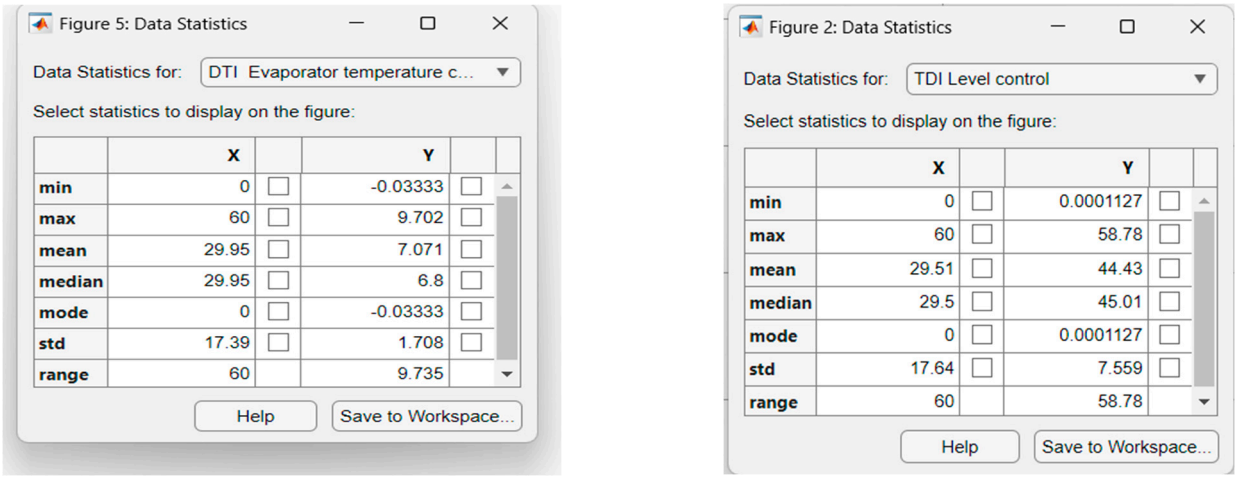
6. Discussion

6.1. Conventional Control Strategies

In this section, a rigorous performance analysis is presented based on the statistical data and stability performance error indicators extracted from step responses and the standard structures used to calculate these indicators, such as ISE, ITSE, IE, IAE, and ITAE.

6.1.1. DTI Controllers

For the discrete-time integrator (DTI) controller developed in Section 3.1, through a simple inspection of both the step responses represented in Figure A2a,b and the statistics extracted for the same step responses for evaporator temperature, depicted in Figure 18a, and liquid refrigerant level inside the condenser, reported in Figure 18b, the following performances during transient and steady state regimes were calculated: acceptable time responses (settling time, Ts) and rise time (Tr) for temperature Ts = 40, Tr = 5.5 s and for liquid refrigerant level Ts = 20, Tr = 1.7 s, an overshoot of σ_max = 45.27% for temperature, and a larger number of maximum amplitude oscillations, of about 10%, both reaching zero steady-state errors, therefore showing high tracking accuracy.



(**a**) (**b**)

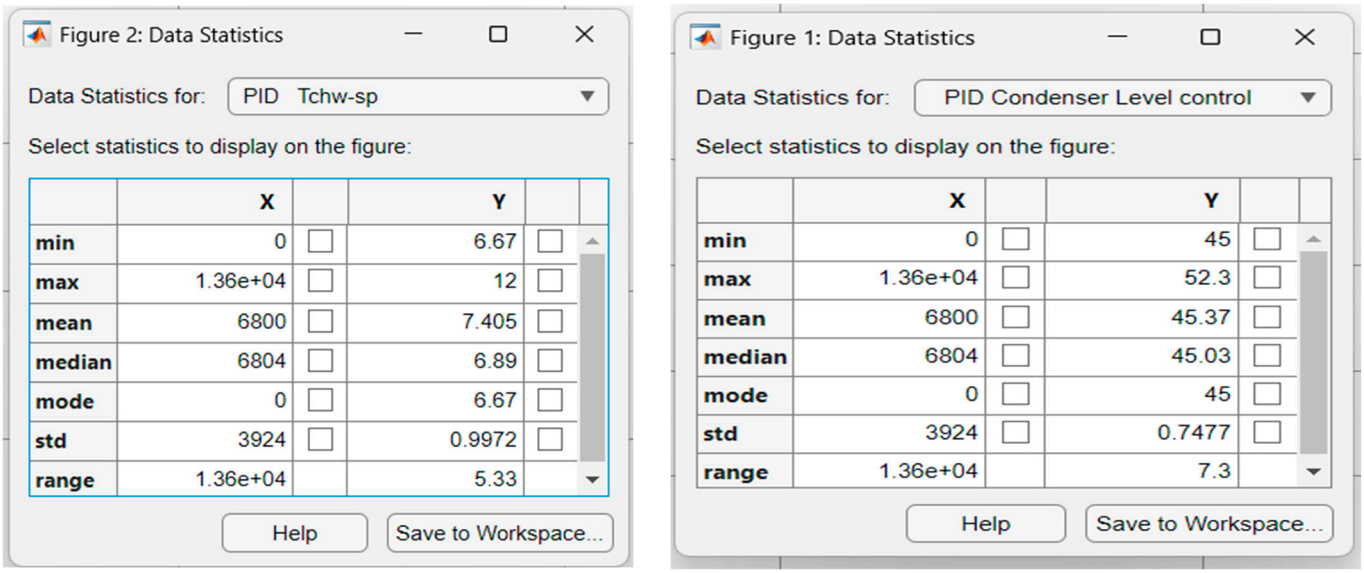
Figure 18. Statistical data extracted from the time responses: (**a**) TDI evaporator temperature control; (**b**) TDI condenser liquid refrigerant level control.

Mean = 7.071 and standard deviation (std) = 1.708 for DTI temperature control can be extracted from Figure 18a, while mean = 44.43 and std = 7.759 for DTI liquid refrigerant level control is given in Figure 18b. It is also worth noting the strong control effort of the oscillating compressor and the almost smooth opening of the expansion valve.

6.1.2. PID Control MIMO Centrifugal Chiller System Extended Model with 39 States

The step responses shown in Figure 12a,b reveal a very long Ts = 10,000 s, Tr = 10,000 s, no overshoot, zero steady-state error, smooth compressor control effort for PID temperature control, and a very long Ts = 6000, Tr = 6000 s, no overshoot, zero steady-state error, and smooth expansion valve opening control effort for PID level control. The main issues encountered for this closed-loop control strategy were a very slow time response and the need for accurate tuning of the parameter values. Also, mean = 7.405 and std = 0.9972 for

PID temperature control are revealed in Figure 19a, and mean = 45.03 and std = 0.7477 for PID level control are shown in Figure 19b.



(**a**) (**b**)

Figure 19. Statistical data extracted from the time responses: (**a**) PID evaporator temperature control; (**b**) PID condenser liquid refrigerant level control.

6.1.3. Model Predictive Control of MIMO Centrifugal Chiller Simplified Model in State-Space Representation

The rigorous performance analysis of step responses depicted in Figure 15c for the evaporator temperature subsystem and in Figure 15d for the liquid refrigerant level in the condenser subsystem revealed the following features:

- settling time Ts = 24.6 s, rising time Tr = 7.6 s, an overshoot of σ_max = 6.75%, zero steady-state error, and an excellent disturbance rejection for evaporator temperature control;
- settling time Ts = 7.2 s, rising time Tr = 7.2 s, no overshoot, a high tracking performance accuracy, and significant disturbance rejection for the liquid refrigerant level inside the condenser subsystem.

These results recommend the MPC closed-loop strategy as among the most suitable conventional control strategies. It performs very well, and neither the compressor relative speed and nor the expansion valve opening input violated the linear bound constraints. The MPC adopted in the first RL DLNN closed-loop control strategy was developed in Section 4.3.1 to generate the reward function from the MPC specifications of the MIMO Centrifugal Chiller System model proposed in the case study.

6.2. Advanced Intelligent Closed-Loop Neural Control Strategies

Digital PID Control MIMO CCS MISO ANFIS Models

The MATLAB Simulink simulation results of the digital PID control CCS ANFIS model without changes in temperature and level setpoints, depicted in Figure 13a and Figure 13b, respectively, show a fast time response Ts = 50 s from the evaporator temperature control subsystem, with zero steady-state error and a 9% temperature of chilled water overshoot, compared with the liquid refrigerant level control inside the condenser subsystem which had Ts = 64 s, steady-state error, and 2% overshoot of the liquid refrigerant level in the condenser. Moreover, the actuator control efforts for the compressor and expansion valve opening, correspondingly shown in Figure 14a and Figure 14b, are sharp at

the beginning and smooth later. The Simulink simulation results depicted in Figure A4a and Figure A4b reveals that the effects of changes in disturbance temperature load Trr from 48 to 54 degrees Fahrenheit applied at time t = 70 s were completely overcome via the chilled temperature control inside the evaporator, as were those in the level of liquid refrigerant within the condenser, almost within 40 s. Also, the figures indicate that the steady state for each controlled output was very close to zero, and the settling times of the step responses were fast enough for both digital PID controllers, which can be interpreted as excellent performance using the improved digital PID control strategy, outperforming the previous control PID control strategy based on the extended model developed in Section 3.2. For this reason, the digital PID with the proposed tuning procedure for the parameters' values was used for performance comparison with the second RL DLNN control structure.

6.3. Advanced Reinforcement Learning Deep Learning Neural Networks Control Strategies

For a rigorous analysis of tracking performance, it is essential to minimize errors in any closed-loop feedback control system. In order to keep track of errors at all time, from zero to infinity, and to minimize them continuously, performance measures were calculated including integral time absolute error (ITAE), integral square error (ISE), integral time square error (ITSE), and integral absolute error (IAE), as previously defined [15] and given by Equations (18)–(22).

An effective tracking performance measure uses data statistics such as min, max, mean, median, mode, standard deviation (std), and range, which are included in Figure A8a–d and Figure A10a–d. As discussed elsewhere [15], the integral time indicators defined by Equations (18)–(21) are so-called fitness functions. Their values are calculated by the standard hardware structures shown in Figures A9a–d and A11a–d, defined as follows:

$$ISE = \int e(t)^2\, dt \tag{18}$$

$$ITSE = \int t[e(t)^2]\, dt \tag{19}$$

$$IAE = \int |e(t)| dt \tag{20}$$

$$ITAE = \int t|e(t)| dt \tag{21}$$

$$IE = \int e(t) dt \tag{22}$$

Specifically, all of these time integral criteria are generic and comprehensive tools to evaluate the performance of a control system. Hence, a system may have a good rise time but a poor settling time, or vice versa, while other basic criteria for evaluating the step response tracking performance of a closed-loop feedback system include the overshoot and the steady-state error extracted from the statistical data, although all these statistics describe only one characteristic. The time integral criteria are generic and comprehensive; they allow comparison between different controller designs or even differently structured controllers. The fitness functions are not actually limited to the above equations. Engineers can provide custom fitness functions depending on the target design and control system. The overall performance (convergence speed and optimization accuracy) of an interesting evolutionary algorithm previously developed [15] for optimal tuning of the PID parameter values depends on the fitness functions. Note that integral time absolute error (ITAE) is widely used in control processing since it is simple to implement and to define the energy of signals which demonstrate symmetry and differentiability.

6.3.1. Reinforcement Learning Deep Learning Neural Network Control Strategies—Generating a Reward Function from the MPC of MIMO Centrifugal Chiller Simplified Model in State-Space Representation

The statistical data tables for the RL DLNN temperature and level control algorithms versus MPC of evaporator temperature and condenser refrigerant level are shown in the Figure A8a–d. Figure A8a presents the statistical data for the MPC for evaporator temperature control, Figure A8b for the RL DLNN temperature control, Figure A8c visualizes the data for the condenser refrigerant level, and Figure A8d includes the RL DLNN refrigerant level control data. The performance error indicator values extracted from the hardware structures are given in Figure A9a–d; precisely, for MPC evaporator temperature in Figure A9a, RL DLNN MPC evaporator temperature in Figure A9b, MPC condenser refrigerant level in Figure A9c, and finally, RL DLNN MPC condenser refrigerant in Figure A9d.

Attentive performance analysis based on the data statistics for the MPC RL DLNN evaporator temperature control subsystem revealed a slight superiority compared with MPC based on the corresponding data statistics and vice versa for the condenser liquid level control subsystem. Regarding the final fitness function values after 60 iterations, the Simulink simulations results indicated a slight superiority on the part of the MPC RL DLNN evaporator temperature control subsystem compared with the MPC, and vice versa for the condenser liquid level control subsystem. The MPC RL DLNN performed better for temperature control and similarly for liquid refrigerant level control. After many other repetitions of the parameter values tuning procedure, which was a so-called "trial and error" procedure, the performance of MPC RL DLNN simulation results was observed to change significantly, such that it was finally able to outperform the conventional MPC for control of both temperature and liquid refrigerant level. Our investigations will continue to improve the performance io the MPC RL DLNN controller in future work.

6.3.2. Reinforcement Learning Deep Learning Neural Network Control Strategies—Generating Reward Function from a Step Response Specifications of MIMO Centrifugal Chiller Simplified Model in State Space Representation

Similarly, the statistical data tables for the RL DLNN temperature and level control algorithms versus the digital PID control–CCS ANFIS model of evaporator temperature and condenser refrigerant level are shown in Figure A10a–d. The Figure A10a presents the statistical data for RL DLNN evaporator temperature control, Figure A10b for The digital PID temperature control CCS ANFIS model, Figure A10c visualizes the statistics for RL DLNN condenser refrigerant level, and Figure A10d illustrates the digital PID refrigerant level control CCS ANFIS model results. The performance indicator values were extracted from the hardware structures illustrated in Figure A11a–d: for RL DLNN evaporator temperature in Figure A11a, the digital PID evaporator temperature control CSS ANFIS model in Figure A11b, RL DLNN condenser refrigerant level in Figure A11c, and finally the digital PID condenser refrigerant level control CCS ANFIS model in Figure A11d. Similar to MPC RL DLNN, rigorous performance analysis conducted for the RL DLNN based on the statistical data and fitness functions' final values after 60 iterations indicated a slight superiority of the improved digital PID controller connected in the same forward-path closed loop architecture with the MISO ANFIS CCS models, compared with the RL DLNN's performance, with both these controllers outperformed the standard PID controller. Furthermore, by increasing the number of "trial-and-error" procedures for tuning parameter values, it seems certain that the performance of RL DLNN results can improve significantly, until it can finally outperform the improved digital PID control structure for control of both temperature and liquid refrigerant level. Our investigations in future work will continue in this direction.

6.4. Considerations Regarding the RL DLNN Control Strategies' Applicability

The proposed RL DLNN control strategies are well suited for handling the complexities of real-world centrifugal chiller systems, due to the abilities of the RL agents to cope with dynamic environments (e.g., varying cooling loads, weather conditions, or equipment performance) and to detect and compensate for system faults or degradation. Moreover, with the rapid advancements in edge computing and hardware acceleration, RL controllers can perform real-time optimization, making them practical for real-world applications. Moreover, RL DLNN controllers can be effectively scaled for centrifugal chiller systems due to a combination of attributes, including their modularity, ease of transfer learning (i.e., pre-trained RL models can be fine-tuned for different chiller systems or operating conditions, reducing the need for extensive retraining), and distributed control capabilities (i.e., RL agents can be deployed in a distributed manner, controlling multiple chillers in a plant or across different locations, and coordinated to achieve global objectives).

From another point of view, the applicability of the proposed RL DLNN control strategies in real-life scenarios can be affected by three important factors: (a) the customized reward function design based on step response specifications may not cover all operational constraints, such as energy efficiency, or the long-term stability of the MIMO Centrifugal Chiller control system, requiring fine-tuning for each specific HVAC application; (b) safety and stability concerns resulting from possibly damaging actions the RL agents explore during training; and, (c) high computational cost, since the RL DLNN models can be computationally expensive to train and deploy, demanding significant hardware resources. These limitations may be mitigated by employing carefully pre-trained RL agents deployed on real-time and low-cost embedded systems.

7. Conclusions

This study explored the development and implementation of two intelligent neural reinforcement learning control algorithms utilizing deep learning neural network frameworks in the specific case of a complex HVAC centrifugal chiller system characterized by high dimensionality and nonlinearity, strict constraints, and significant impact of measured disturbances. For this, two simplified MIMO models of the CCS were generated, and a comprehensive series of simulations were conducted to showcase the efficacy of both RL DLNN control algorithm implementations when compared to two traditional control methods. The two data-driven advanced neural control algorithms this paper proposes have demonstrated their viability and adaptability to various kinds of nonlinearities, singularities, and uncertainties. In future work, we intend to take a further step by implementing our control strategies in real-life scenarios.

Author Contributions: conceptualization, N.T.; methodology, D.-I.C. and N.T.; software, D.-I.C. and R.-E.T.; validation, M.Z. and N.T.; formal analysis, R.-E.T.; investigation, N.T. and D.-I.C.; resources, M.S.R.; data curation, M.Z.; writing—original draft preparation, R.-E.T.; writing—review and editing, N.T.; visualization, R.-E.T. and S-M Radu; supervision, N.T. and M.Z.; project administration, M.S.R.; funding acquisition, M.S.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data Availability Statements are available in section "MDPI Research Data Policies" at https://www.mdpi.com/ethics.

Acknowledgments: N. Tudoroiu expresses his gratitude to Roberta Silerova, John Abbott College Program Dean Science, Chair Mark Evanchyna and Co-Chair Hana Chammas of the Intech Department, as well as his departmental colleagues Demartonne Ramos França, Shiwei Huang, and Evgeni

Kiriy for their support, guidance, and valuable advice during the course of this scientific research project.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PID	Proportional integral derivative
DTI	Discrete time integrator
HVAC	Heating Ventilation Air Conditioning
CCS	Centrifugal Chiller System
MIMO	Multi-input multi-output
SISO	Single-input single-output
MISO	Multi-input single-output
ARMAX	Autoregressive moving average with exogenous input
ANFIS	Adaptive Neural Fuzzy Inference System
IAE	Integral absolute error
ITAE	Integral time absolute error
ISE	Integral square error
ITSE	Integral time-weighted square error

Appendix A

Appendix A.1. Figures

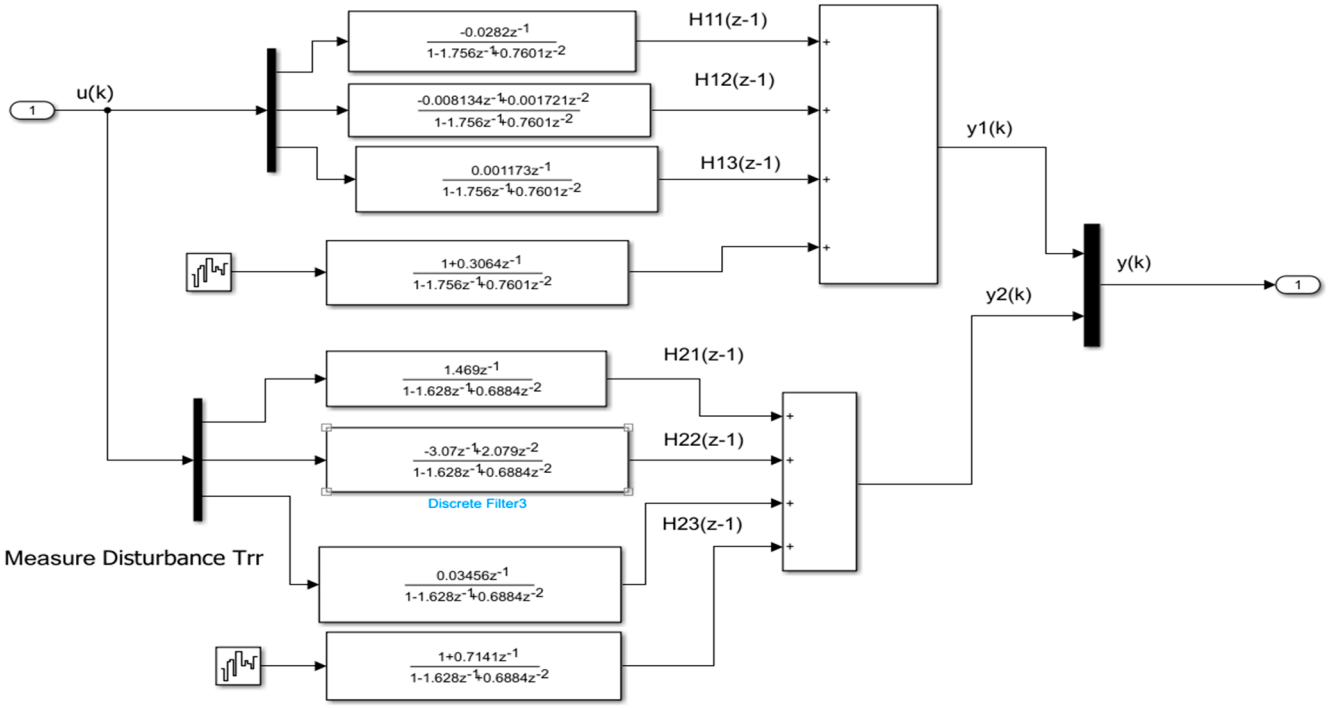
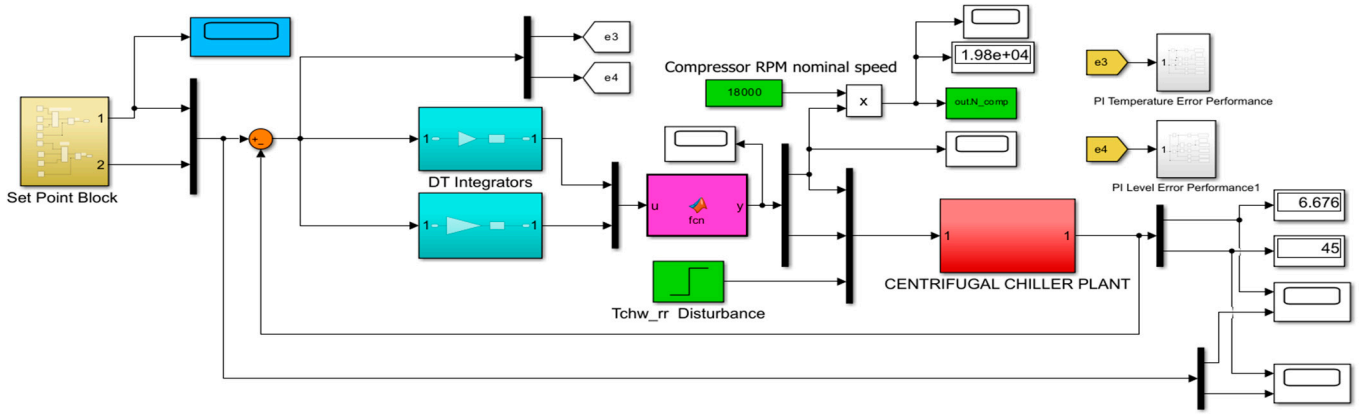
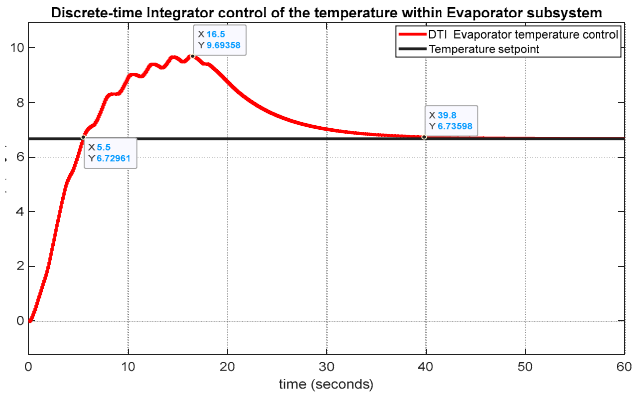


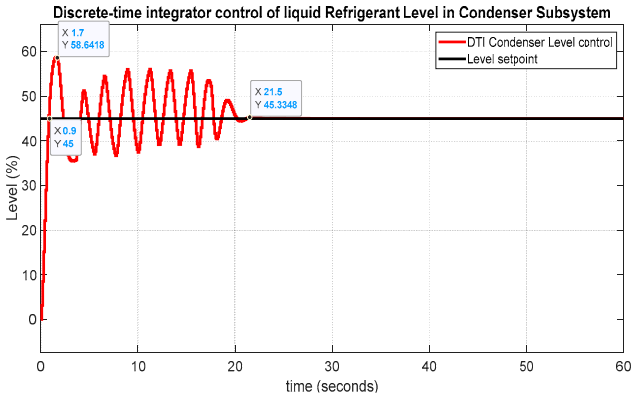
Figure A1. Simulink diagram of discrete-time transfer function of the MIMO Centrifugal Chiller plant.



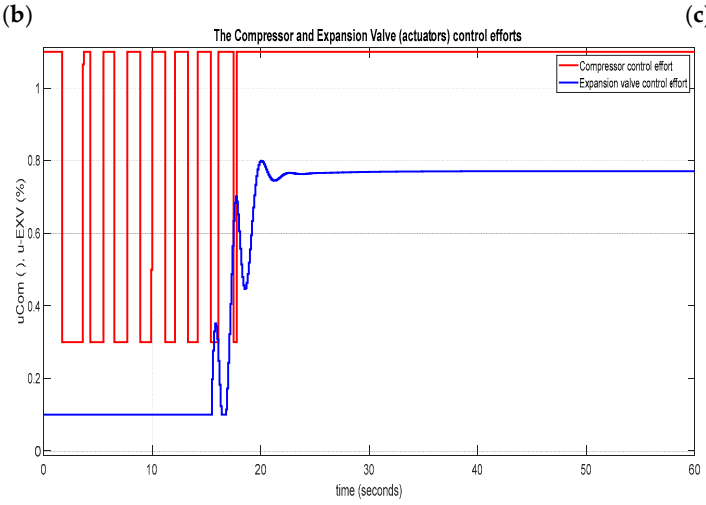
(**a**)



(**b**)

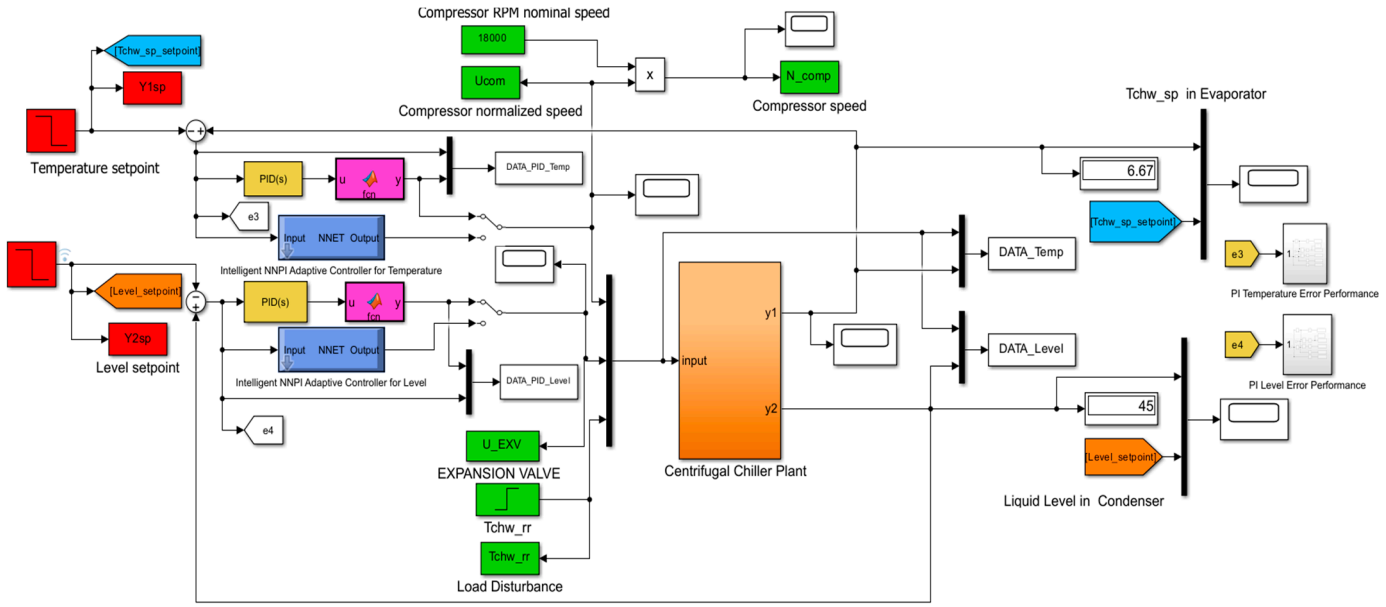


(**c**)

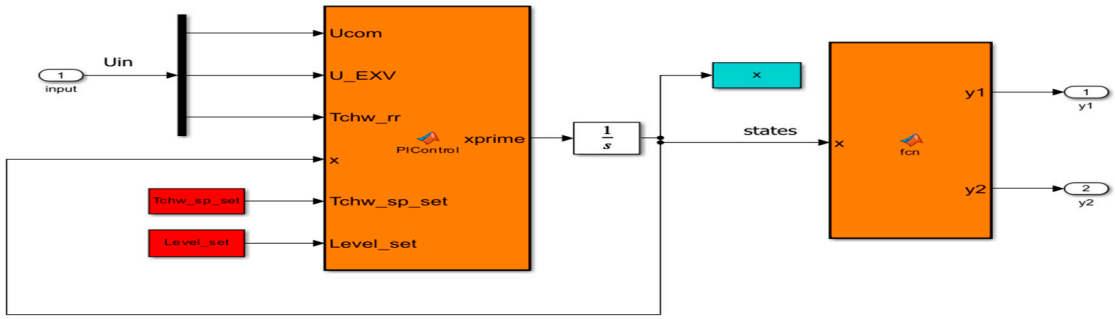


(**d**)

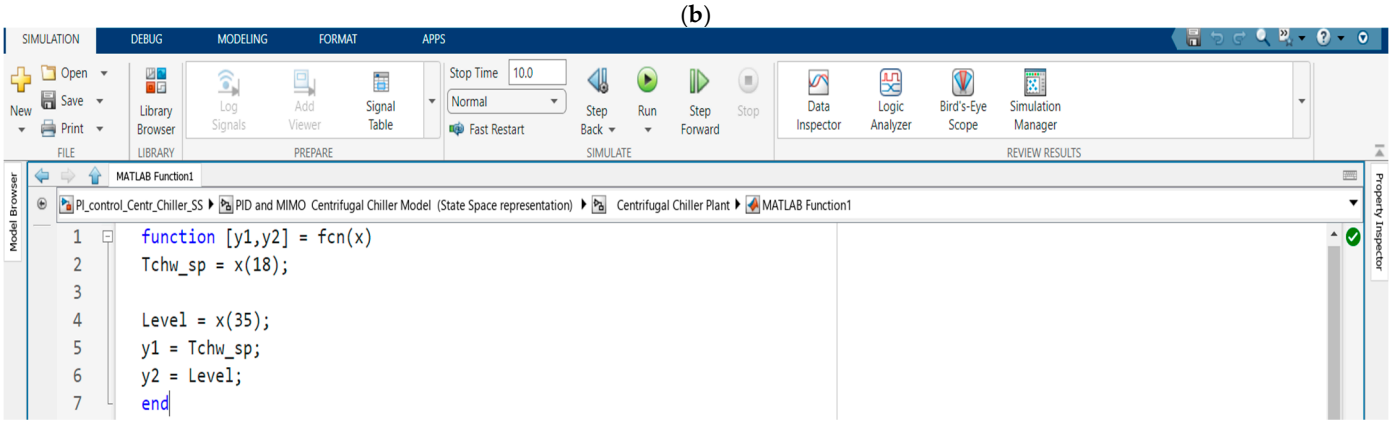
Figure A2. DTI control—MIMO Centrifugal Chiller system extended model and Simulink simulations: (**a**) DTI control Simulink diagram; (**b**) chilled water temperature in evaporator; (**c**) liquid refrigerant level in condenser; (**d**) compressor and expansion valve-opening actuator control efforts.



(**a**)

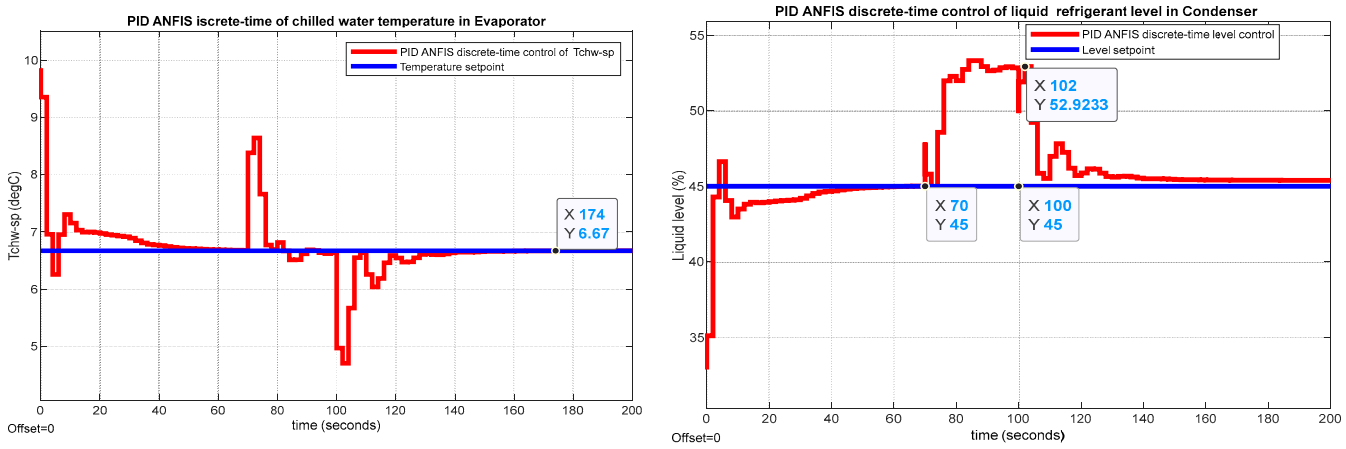


(**b**)



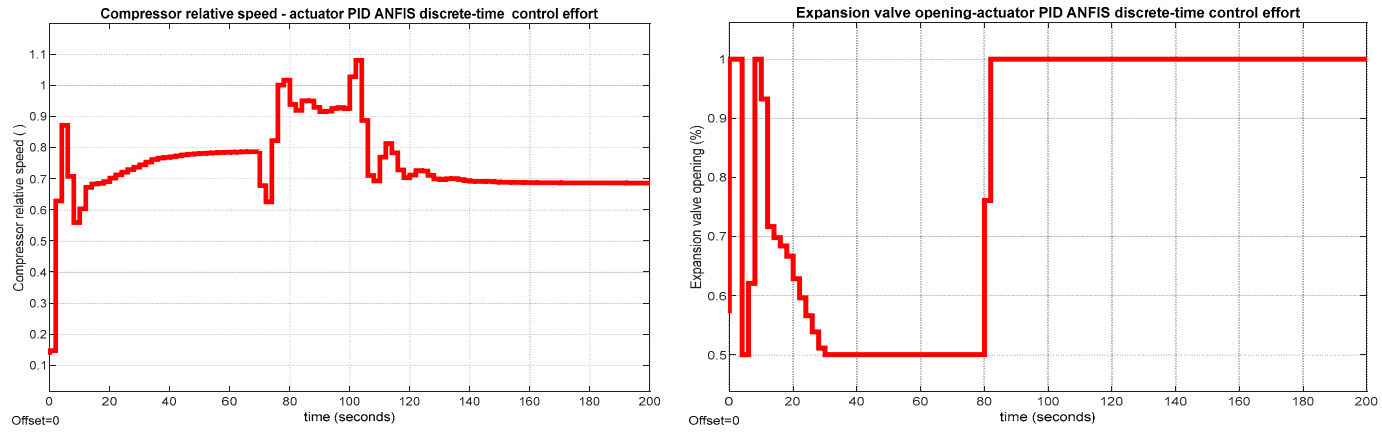
(**c**)

Figure A3. Simulink model of PID closed-loop control strategies for the MIMO Centrifugal Chiller nonlinear extended model (39 states): (**a**) overall diagram; (**b**) components of the Simulink block diagram (MATLAB function of the extended model and MATLAB function visualization block); (**c**) MATLAB function visualization block.



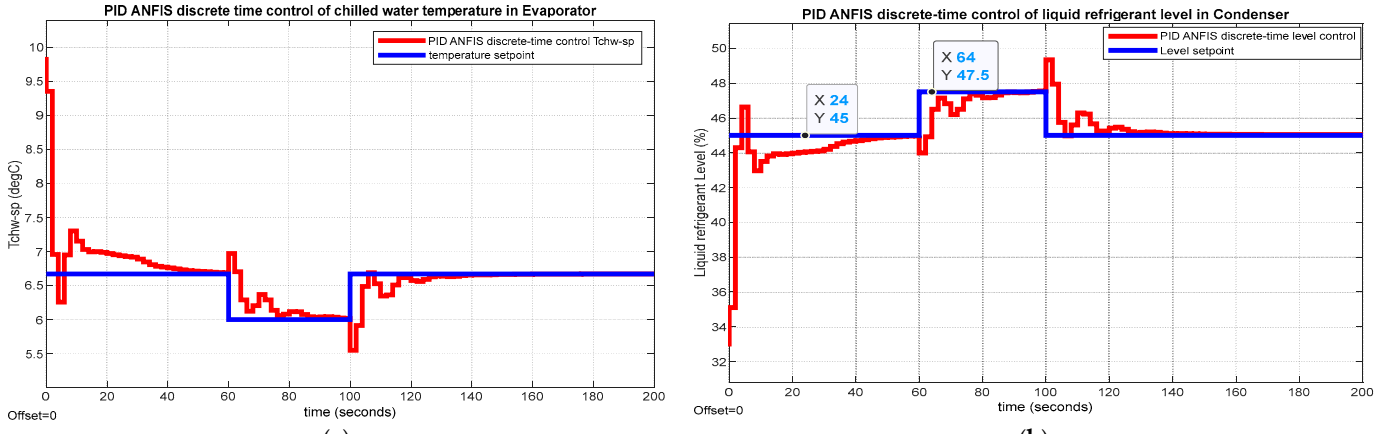
(**a**) (**b**)

Figure A4. PID ANFIS discrete-time simulation results, rejection temperature disturbance Tchw_rr: (**a**) chilled water temperature inside evaporator; (**b**) liquid refrigerant level within condenser.



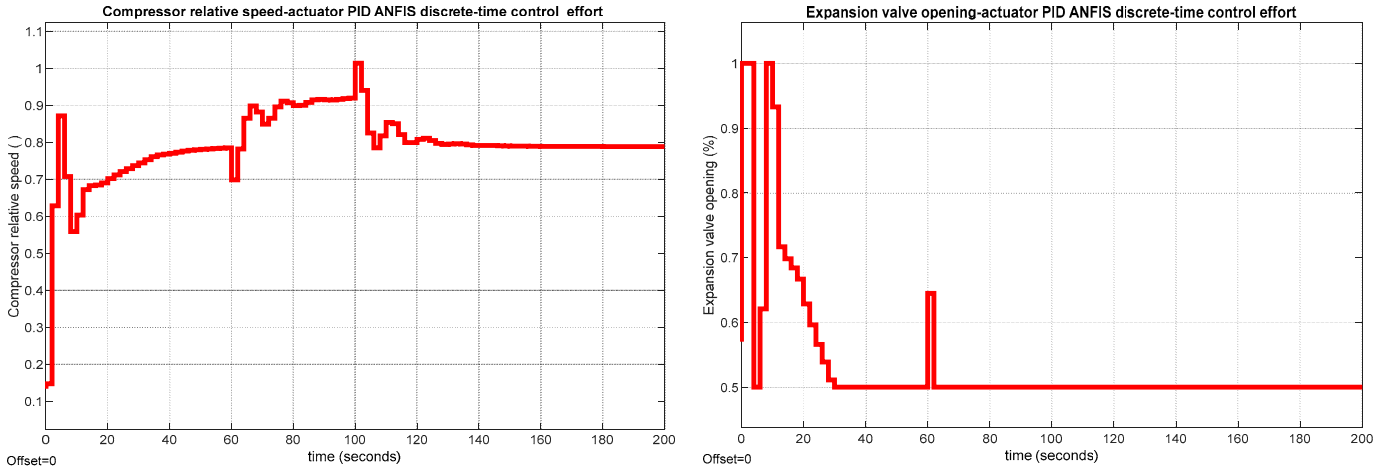
(**a**) (**b**)

Figure A5. PID ANFIS discrete-time control MATLAB Simulink simulation results of actuator control effort: (**a**) compressor relative speed; (**b**) expansion valve opening.



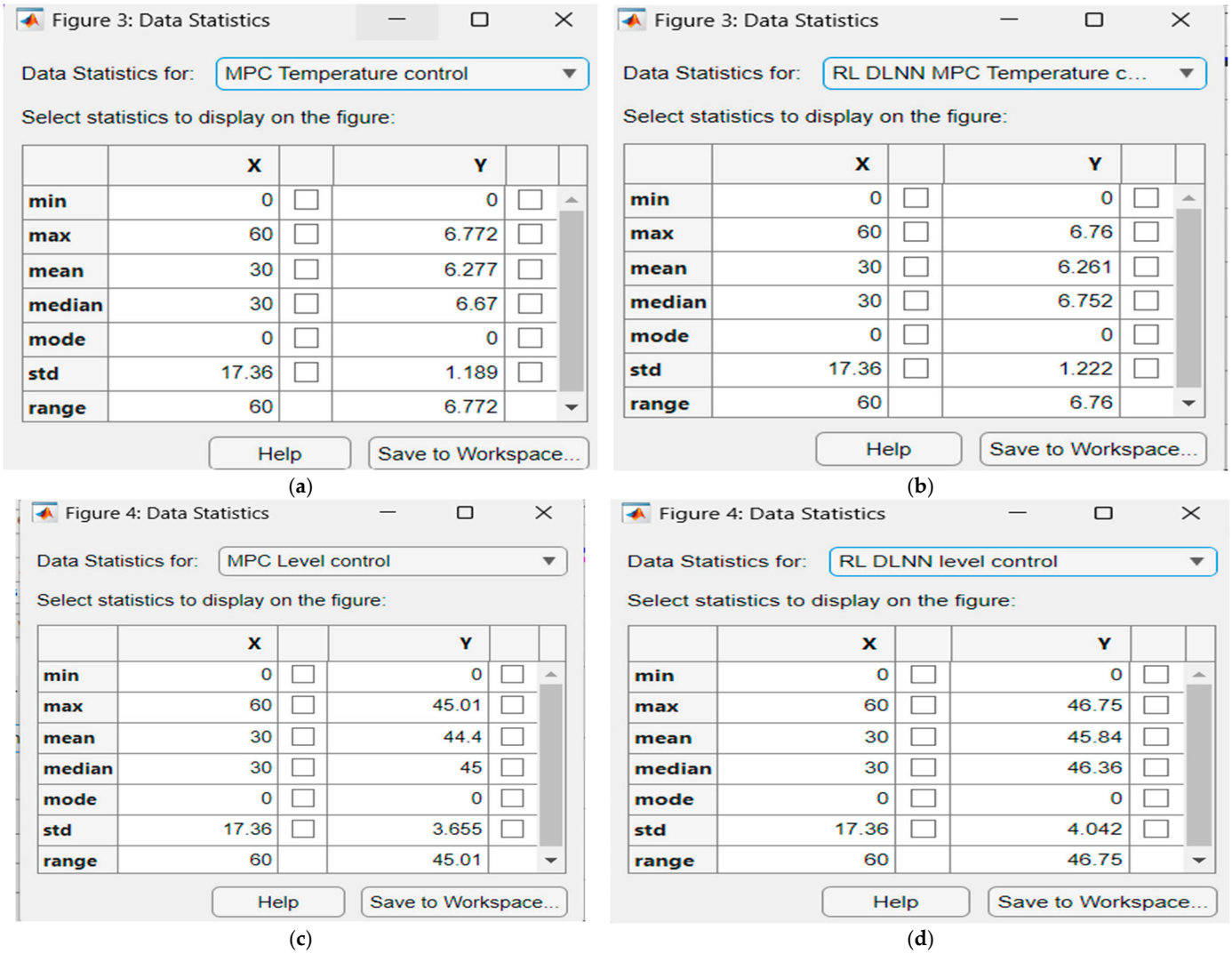
(**a**) (**b**)

Figure A6. PID ANFIS discrete-time control MATLAB Simulink simulation results: (**a**) chilled water temperature within evaporator; (**b**) liquid refrigerant level inside condenser.



(**a**) (**b**)

Figure A7. PID ANFIS discrete-time control—MATLAB Simulink simulation results of actuator control effort: (**a**) compressor relative speed; (**b**) expansion valve opening.



(**a**) (**b**)

(**c**) (**d**)

Figure A8. Statistical data tables: (**a**) MPC evaporator temperature control; (**b**) RL DLNN evaporator temperature control; (**c**) MPC condenser refrigerant level control; (**d**) RL DLNN MPC condenser refrigerant level control.

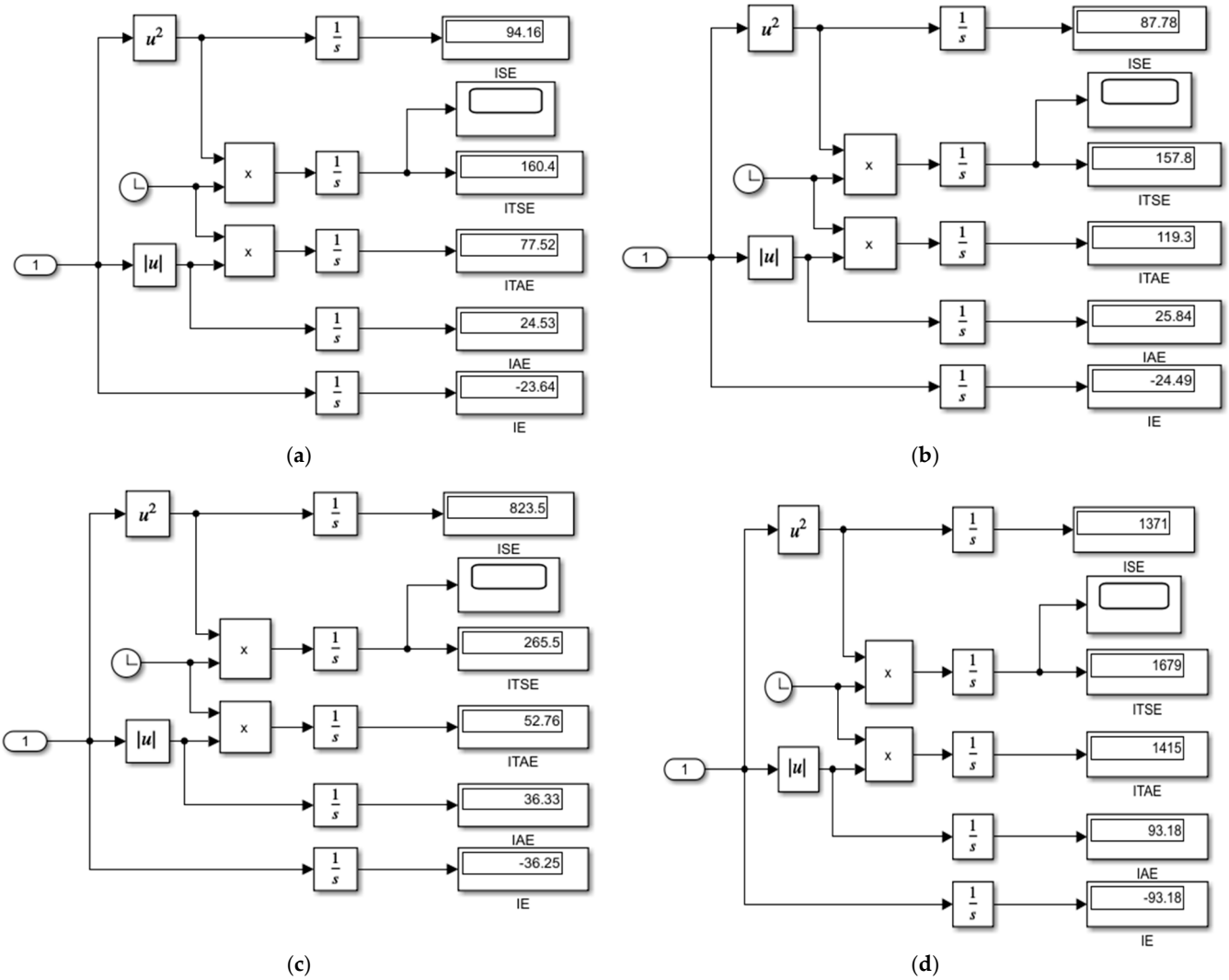
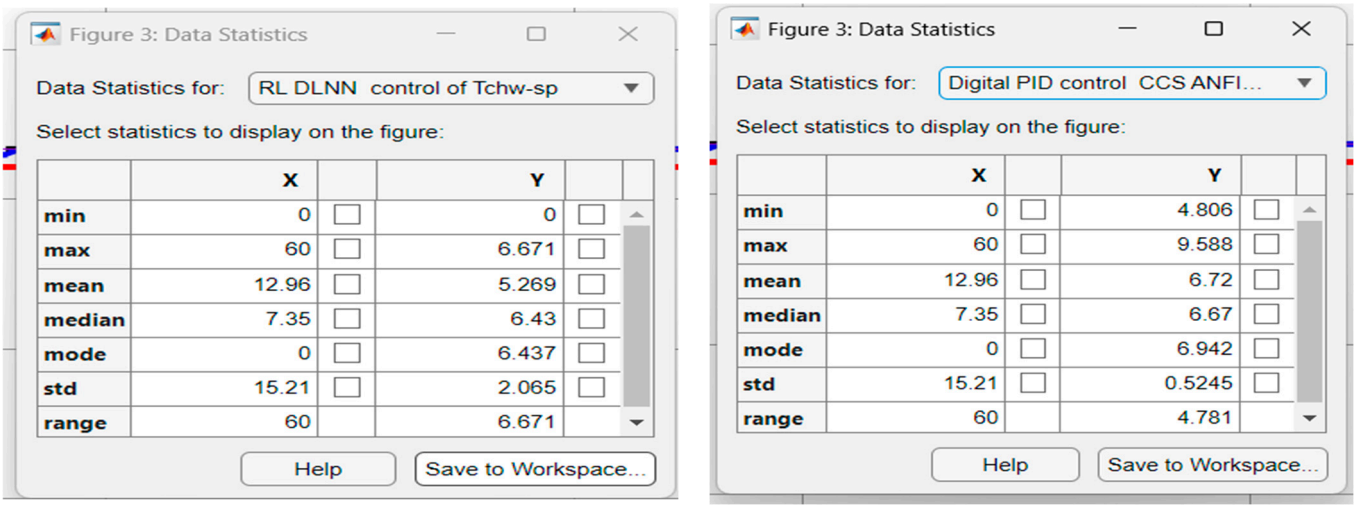
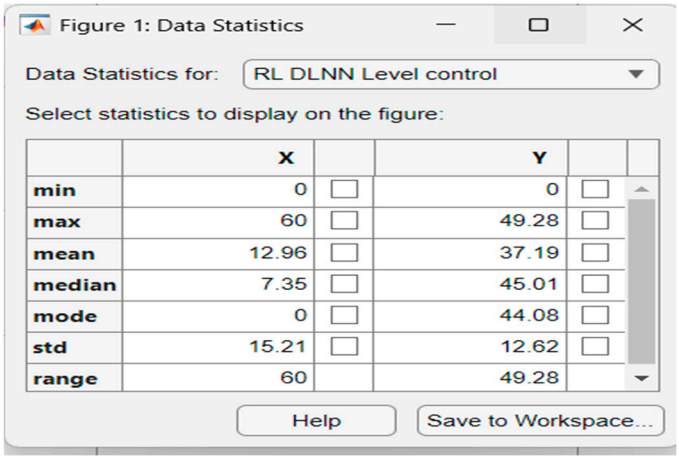
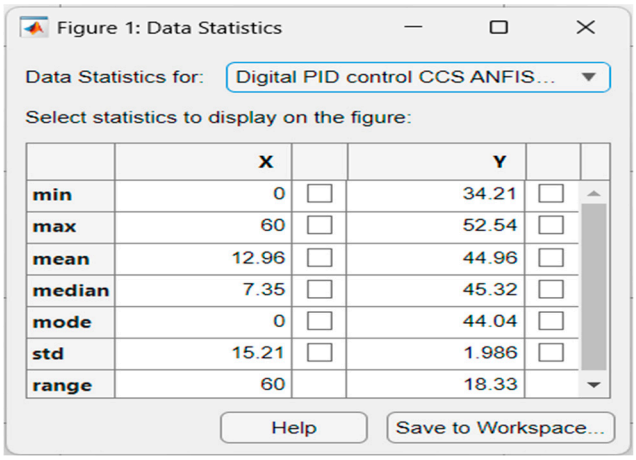


Figure A9. Performance error indicators: (**a**) MPC evaporator temperature control; (**b**) RL DLNN evaporator temperature control; (**c**) MPC condenser refrigerant level control; (**d**) RL DLNN MPC condenser refrigerant level control.



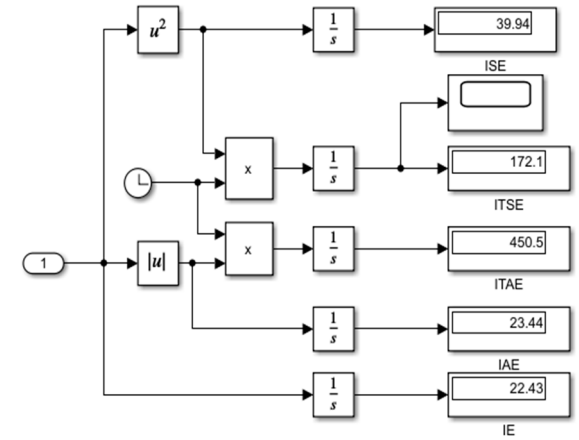


(**c**)

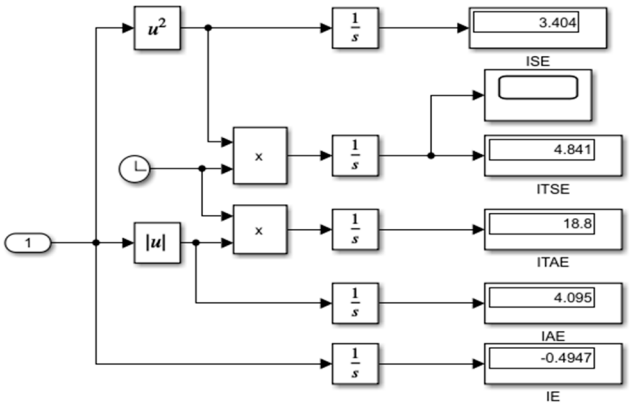


(**d**)

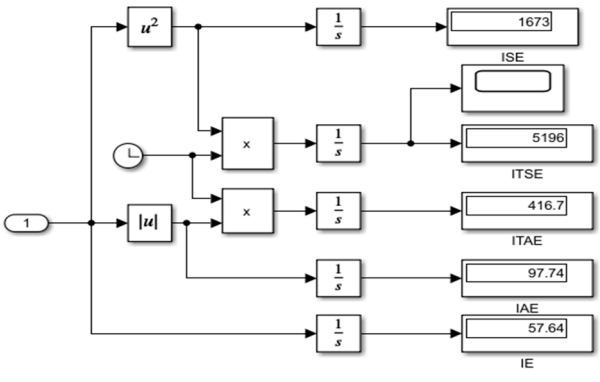
Figure A10. Statistical data tables: (**a**) RL DLNN evaporator temperature control; (**b**) digital PID evaporator temperature control—CCS ANFIS model; (**c**) RL DLNN MPC condenser refrigerant level control; (**d**) digital PID condenser refrigerant level control—CCS ANFIS model.



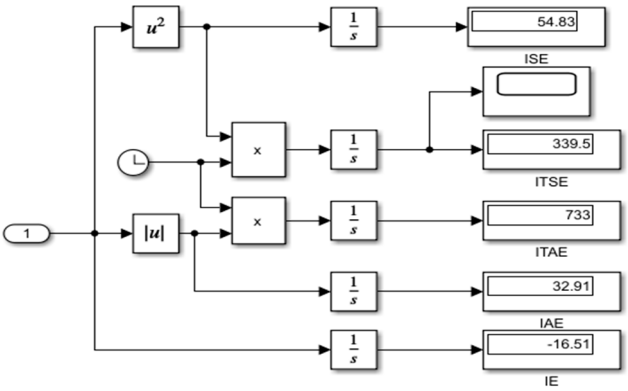
(**a**)



(**b**)



(**c**)



(**d**)

Figure A11. Performance control error indicators: (**a**) RL DLNN evaporator temperature control; (**b**) digital PID evaporator temperature control—CCS ANFIS model; (**c**) RL DLNN condenser refrigerant level control; (**d**) digital PID condenser refrigerant level control—CCS ANFIS model.

Appendix B

Appendix B.1. Algorithms

Algorithm A1. Generate Reward Function Based on MPC Specifications for a MIMO Centrifugal Chiller System-MPC RL DLNN

Step 1. Plant Dynamics (State Space):

```
clear
clc
close all
A=[0 -0.7601 0 0; 1 1.756 0 0; 0 0 0 -0.6884;0 0 1 1.628];
B= [0 0.006884 0; -0.1128 -0.03254 0.004691; 0  1.04  0; 0.7347 -1.535 0.01728];
C=[0 0.25 0 0; 0 0 0 2];
D=[0 0 0;0 0 0];
Ts =0.1;
plant = ss(A,B,C,D,Ts);
```

Step 2: Create an MPC object:

```
plant = ss(A,B,C,D);
plant.InputName = {'Ucom','uEXV','Trr'};
plant.InputGroup.MV = 1:2;
plant.InputGroup.MD = 3;
plant.OutputName = {'Tchw','Level'};
```

Step 2.1. Specify the sampling time (Ts), prediction horizon (p) and control horizon (m):

```
Ts = 0.1;
p = 10;
m = 3;
```

Step 2.2. Create the MPC object

```
mpcobj = mpc(plant,Ts,p,m);
```

Step 2.3. Open the Simulink model of the plant and MPC controller:

```
mdl = "rlMPC_CentrChiller_modified";
open_system(mdl);
```

Step 2.4. Specify the minimum and maximum values for manipulated variables (MVs):

```
mpcobj.MV(1).Min = 0;
mpcobj.MV(1).Max = 1.1;
mpcobj.MV(2).Min = 0;
mpcobj.MV(2).Max = 1;
```

Step 2.5. Nominate precise values for outputs:

```
mpcobj.Model.Nominal.Y = [6.67 45];
```

Step 2.6. Nominate precise values for all inputs:

```
mpcobj.Model.Nominal.U = [1.1 1 48];
```

Step 2.6. Specify output tuning weights:

```
mpcobj.Weights.OV = [10 10];
```

Step 2.7. Define the MPC object using structure format for output variables, tuning weights, and manipulated output variables:

```
mpcobj = mpc(plant, 0.1, 10, 2, mpcobj.Weights,mpcobj.MV,mpcobj.OV);
```

Step 3. Generate the reward function:

```
generateRewardFunction(mpc1obj)
type rewardFunctionmpcplant
```

Step 3.1. Open the reward function in the Simulink model and append a new MATLAB function r = rewardFunctionMPC (y,refy,mv,refmv,lastmv):

```
open_system("rlMPC_CentrChiller_modified/Reward Function")
```

Step 4: Create an RL environment (Ts, Tsim, observations and actions specifications):

```
Tf = 60;
Ts = 0.1;
```

```
numObs = 24;
numAct = 2;
oinfo = rlNumericSpec([numObs 1]);
ainfo = rlNumericSpec([numAct 1], ...
    LowerLimit=[0 0]', ...
    UpperLimit=[1.1 1]');
```

```
blk = "rlMPC_CentrChiller_modified/RL Agent";
env = rlSimulinkEnv(mdl, blk, oinfo, ainfo);
```

Step 5: Create a RL agent (random seed for reproducibility, layer graph object—connect and define network path, plot criticNet):

```
rng(0)
```

```
mainPath = [
    featureInputLayer(numObs)
    fullyConnectedLayer(128)
    concatenationLayer(1, 2, Name="concat")
    reluLayer
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(1)];
actionPath = [
    featureInputLayer(numAct)
    fullyConnectedLayer(8, Name="fc_act")];

criticNet = layerGraph(mainPath);
criticNet = addLayers(criticNet, actionPath);
criticNet = connectLayers(criticNet, "fc_act", "concat/in2");
```

```
plot(criticNet);
```

Step 6. Define actorNet, plot it and create a deterministic actor function:

```
actorNet = [
    featureInputLayer(numObs)
    fullyConnectedLayer(128)
    reluLayer
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(numAct)];
```

```
plot(layerGraph(actorNet));
```

```
actordlNet = dlnetwork(actorNet);
actor = rlContinuousDeterministicActor(actordlNet, oinfo, ainfo);
```

Step 7. Specify the agent, critic optimizer, and actor optimizer options:

```
agentOpts = rlTD3AgentOptions(SampleTime=Ts, ...
    DiscountFactor=0.995, ...
    ExperienceBufferLength=1e6, ...
    MiniBatchSize=256);
for idx = 1:2
    agentOpts.CriticOptimizerOptions(idx).LearnRate = 1e-3;
    agentOpts.CriticOptimizerOptions(idx).GradientThreshold = 1;
end

agentOpts.ActorOptimizerOptions.LearnRate = 1e-3;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;
```

```
agentOpts.ExplorationModel.StandardDeviationMin = 1e-3;
agentOpts.ExplorationModel.StandardDeviation = 100;
agentOpts.ExplorationModel.StandardDeviationDecayRate = 1e-5;
```

Step 8. Create the TD3 agent:

```
agent = rlTD3Agent(actor, [critic1,critic2], agentOpts);
validateEnvironment(env)
```

Step 9. Train the TD3 agent (options, evaluation, random seed, training function):

```
trainOpts = rlTrainingOptions(...
    MaxEpisodes=1000, ...
    MaxStepsPerEpisode=ceil(Tf/Ts), ...
    StopTrainingCriteria="EvaluationStatistic", ...
    StopTrainingValue=-0.2, ...
    ScoreAveragingWindowLength=20);
evl = rlEvaluator( ...
    NumEpisodes=1, ...
    EvaluationFrequency=50);
```

```
rng(0,"twister");
```

```
%doTraining = false;
doTraining=true;
if doTraining
    trainResult = train(agent, env, trainOpts, Evaluator=evl);
else
    load('rlDCServomotorTD3Agent.mat')
end
```

Step 10. Validate the closed-loop controller:

```
sim(mdl);
```

Algorithm A2. Generate Reward Function from a Step Response Block for a MIMO Centrifugal Chiller System—RL DLNN

Step 1. MIMO ARMAX Centrifugal Chiller model dynamics (state space):

```
clc
clear
close all
A=[0 -0.7601 0 0; 1 1.756 0 0; 0 0 0 -0.6884;0 0 1 1.628];
B= [0 0.006884 0; -0.1128 -0.03254 0.004691; 0  1.04  0; 0.7347 -1.535 0.01728];
C=[0 0.25 0 0; 0 0 0 2];
D=[0 0 0;0 0 0];
```

Step 1.1 Initialize the model:

```
Level0=0;
Levelf=60;
Tchw_sp0=0;
Tchw_spf=12;
Tf =20;
Ts =0.1;
```

Step 1.2 Open the Simulink model:

```
mdl = "rlCentrChiller_MIMO_StepResp_TwoAgs_SS";
open_system(mdl)
```

Step2. Open BLk1 and BLk2 step response blocks:

```
verifBlk1 = mdl + "/TempStepResponse";
open_system(verifBlk1)
verifBlk2 = mdl + "/LevelStepResponse";
open_system(verifBlk2)
```

Step 3. Generate the reward functions Vfb1 and Vfb2:

```
generateRewardFunction(verifBlk1)
type rewardFunctionVfb1.m
generateRewardFunction(verifBlk2)
type rewardFunctionVfb2.m
```

Step 4. Combine observations and actions specifications:

```
oinfo = {oinfo1,oinfo2};
ainfo = {ainfo1,ainfo2};
```

Step 5. Create an RL environment:

```
env = rlSimulinkEnv(mdl, agentBlk, oinfo, ainfo);
```

Step 6. Create a Simulink environment interface:

```
agentBlk = mdl + ["/RL Agent1", "/RL Agent2"];
```

Step 7. Create a learning environment:

```
env = rlSimulinkEnv(mdl, agentBlk, oinfo, ainfo);
```

Step 8. Create the reinforcement learning environment for RL Agent1 and RL Agent2 (TDR3s):

Step 8.1. Set the random reproducibility seed:

```
rng(0,"twister")
```

Step 8.2. Define network path:

```
mainPath = [
    featureInputLayer(numObs)
    fullyConnectedLayer(128)
    concatenationLayer(1, 2, Name="concat")
    reluLayer()
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(1)];
actionPath = [
    featureInputLayer(numAct)
    fullyConnectedLayer(8, Name="fc_act")];
```

Step 8.3. Create a layer graph object for criticNet:

```
criticNet = layerGraph(mainPath);
criticNet = addLayers(criticNet, actionPath);
```

Step 8.4. Connect all network layers:

```
criticNet = connectLayers(criticNet, "fc_act", "concat/in2");
```

Step 9. Plot the critic network structure:

Step 9.1. Convert network to dlnetwork:

```
criticdlNet = dlnetwork(criticNet, Initialize=false);
```

Step 9.2. Convert the critic functions for TD3 agents:

```
critic1 = rlQValueFunction(initialize(criticdlNet), oinfo1, ainfo1);
critic2 = rlQValueFunction(initialize(criticdlNet), oinfo2, ainfo2);
```

Step 9.3. Define actorNet:

```
actorNet = [
    featureInputLayer(numObs)
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(128)
    reluLayer()
    fullyConnectedLayer(numAct)];
```

Step 9.4. Plot the actorNet:

```
plot(layerGraph(actorNet));
```

Step 9.5. Create a deterministic actor function:

```
actordlNet = dlnetwork(actorNet);
actor1 = rlContinuousDeterministicActor(actordlNet, oinfo1, ainfo1);
actor2 = rlContinuousDeterministicActor(actordlNet, oinfo2, ainfo2);
```

Step 9.6. Specify the agent options:

```
agentOpts = rlTD3AgentOptions( ...
    SampleTime=Ts, ...
    DiscountFactor=0.99, ...
    ExperienceBufferLength=1e6, ...
    NumWarmStartSteps=1024, ...
    NumEpoch=20, ...
    MiniBatchSize=256);
```

```
agentOpts.ExplorationModel.StandardDeviation = 0.5;
agentOpts.ExplorationModel.StandardDeviationDecayRate = 1e-5;
agentOpts.ExplorationModel.StandardDeviationMin = 0;
% agentOpts.ExplorationModel.StandardDeviation = 0.5;
agentOpts.ActorOptimizerOptions.LearnRate = 1e-3;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;
agentOpts.CriticOptimizerOptions(1).LearnRate = 1e-3;
agentOpts.CriticOptimizerOptions(1).GradientThreshold = 1;
agentOpts.CriticOptimizerOptions(2).LearnRate = 1e-3;
agentOpts.CriticOptimizerOptions(2).GradientThreshold = 1;
```

Step 10. Create the TD3 agents:

```
agent1 = rlTD3Agent(actor1, [critic1, critic2], agentOpts);
agent2 = rlTD3Agent(actor2, [critic1, critic2], agentOpts);
agents={agent1 agent2};
```

Step 11. Train the TD3 agents

```
trainOpts = rlMultiAgentTrainingOptions(...
    AgentGroups={1,2},...
    LearningStrategy=("decentralized"),...
    MaxEpisodes=200,...
   MaxStepsPerEpisode=ceil(Tf/Ts),...
   StopTrainingCriteria="AverageReward",...
   StopTrainingValue=[1,1],...
   ScoreAveragingWindowLength=20);
```

```
% doTraining = false;
doTraining =true;
if doTraining
trainingStats = train([agent1,agent2], env, trainOpts);
else
    load("rlTD3Agent.mat")
end
```

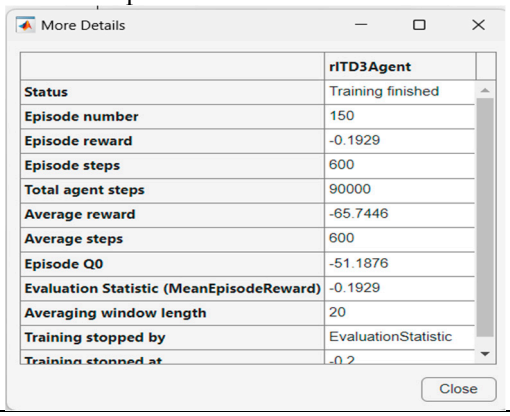
Step 12. Closed-loop simulation:

```
maxsteps=ceil(Tf/Ts);
simOpts = rlSimulationOptions(MaxSteps=maxsteps);
experience = sim(env,[agent1,agent2],simOpts);
set(0, 'GraphicsTimeout', Inf);
sim(mdl);
```

Appendix B.2

Appendix B.2.1. DLNN Table for Reinforcement Learning Based on MPC Specifications

Table A1. Simulation Parameters: Reinforcement Learning Actor Critic and actorNet.

Parameter Name	Value/Description
RL Algorithm	TD3 RL agent **Description:** TD3 agents parametrize Q-value function **NN architecture:** NN with two inputs (one for observation and the second one for action-see the layer graphs for both actors) to model the parametrized Q-value function within both critics **Metrics:** • cumulative long-term reward (expected) • Average reward: −65.7446 at epoch 150 when the training process is finished after the agent reaches the stop training criteria (when evaluation statistic is −0.2); • Episode reward: −0.1929; • Episode Q0: −51.1876; • Evaluation statistic: −0.1929. * See the episode information from RL Training Monitor: 
DNN Architecture	1. **DLNN for actorCritic:**

	One Input layer of size 24 (number of observations), one input layer of size 2 (number of actions), three fully connected layers on the main path of output sizes 128, 64, and 1; One concatenation layer (1,2) and two RELU layers; One fully connected layer of size 8 on the action path. * See the layer graphs for Actor Critic and Actornet 2. **DLNN for actorNet:** One input layer of size 24 (number of observations or features); Three fully connected layers of sizes 128, 64, and 2 (number of actions); Two RELU layers
Reward Function	Derived from MPC model verification block
Training episodes	1000
Test Episodes	1
Maximum Steps per Episode	600 (Tsim/Ts = 60/0.1)
Stop training criteria	Evaluation statistic: −0.2
Score Averaging Window length	20
Number Episodes for Evaluator	1
Evaluation frequency	50 episodes
Discount factor (γ)	0.995
Exploration model property	Noise: std = 100, exponential decay rate = 1×10^{-5}, minimum value reached of 1×10^{-3}.
Learning rate	0.001
Batch size	256
Replay Buffer Size	1×10^{6}

Appendix B.2.2. DLNN Table for Reinforcement Learning Based on Step Responses Specifications

Table A2. Simulation Parameters: Reinforcement Learning Actor Critic and actorNet.

Parameter Name	**Value/Description**
RL Algorithm **rlTD3Agent** / **rlTD3Agent_1** **Status**: Training finished / Training finished **Episode number**: 200 / 200 **Episode reward**: 0 / -50 **Episode steps**: 1 / 1 **Total agent steps**: 31045 / 31045 **Average reward**: 0 / -50 **Average steps**: 1 / 1 **Episode Q0**: -1.493065 / -457.5257 **Averaging window length**: 20 / 20 **Training stopped by**: MaxEpisodes / MaxEpisodes **Training stopped at**: Episode 200 / Episode 200	Two TD3 agents for RL Agent1 (evaporator temperature) and RL Agent2 (condenser level). **Description:** two parametrized Q-value function approximators to estimate the value of the policy (expected cumulative long-term reward as metric). **NN architecture:** two NNs with two inputs for each agent (observation and action-see the layer graphs for Actor Critic and actorNet) that model the parametrized Q-value function within both critics. A critic function object is created to encapsulate the critic by wrapping around the critic deep neural network. To make sure the critics have different initial weights, each network is initialized before using them to create the critics. **Metrics:** • Cumulative long-term reward (expected metric); • Average reward: [0–50] at epoch 200 when the training process ends after the maximum number of episodes, even if both RL agents do not reach the training stop criteria (when average reward is set for both RL agents [1 1]); • Episode reward: [0–50]; • Episode Q0: [-1.493065–457.5267]. * See the episode information from RL Training Monitor.

DNN Architecture	**DLNN for each Actor Critic:** One input layer of size 6 (number of observations), one input layer of size 1 (number of actions), three fully connected layers on the main path with output sizes 128, 64, and 1; One concatenation layer (1,2) and two RELU layers; One fully connected layer of size 8 on the action path. * See the layer graphs for Actor Critic and actorNet. **DLNN for actorNet:** One input layer of size 6 (number of observations or features); Three fully connected layers of sizes 128, 64, and 1 (number of actions); Two RELU layers.
Reward Function	Two reward functions that derive from step responses of two model verification blocks, one for evaporator temperature, other one for condenser level blocks.
Training episodes	200
Test Episodes	1
Maximum Steps per Episode	200 (Tsim/Ts = 20/0.1)
Stop training criteria	Average Reward: [1 1]
Score Averaging Window length	20
Number Episodes for Evaluator	1
Evaluation frequency	10 episodes
Discount factor (γ)	0.995
Exploration model property	Noise: std = 100, exponential decay rate = 1×10^{-5}, minimum decay rate value reached = 1×10^{-3}.
Learning rate	0.001
Batch size	256
Replay Buffer Size	1×10^{6}

References

1. Li, P.; Li, Y.; Seem, J.E. Modelica Based Dynamic Modeling of Water-Cooled Centrifugal Chillers. In Proceedings of the International Refrigeration and Air Conditioning Conference, West Lafayette, IN, USA, 12–15 July 2010; Purdue University: West Lafayette, IN, USA, 2010; pp. 1–8. https://docs.lib.purdue.edu/iracc/1091.
2. Popovic, P.; Shapiro, H. N. Modeling Study of a Centrifugal Compressor. *ASHRAE Trans*. **1998**, *104 Pt 2*, 121.
3. Ning, M. Neural Network Based Optimal Control of HVAC&R Systems. Ph.D. Thesis, Department of Building, Civil and Environmental Engineering, Concordia University, Montreal, QC, USA, 2008.
4. Wong S.P.W.; Wang, S.K. System Simulation of the performance of a Centrifugal chiller Using a Shell-and-Tube-Type Water-Cooled Condenser and R-11 as Refrigerant. *ASHRAE Trans.* **1989**, *95*, 445–454.
5. Wang, S.; Wang, J.; Burnet, J. Mechanistic Model of a Centrifugal Chiller to Study HVAC Dynamics. *Build. Serv. Eng. Res. Technol.* **2000**, *21*, 73–83. https://doi.org/10.1177/014362440002100201.
6. Browne, M.W.; Bansal, P.K. Steady-State Model of Centrifugal Liquid Chillers. *Int. J. Refrig.* **1998**, *21*, 343–358.
7. Li, S.; Zaheeruddin, M. A Model and Multi-Mode Control of a Centrifugal Chiller System: A Computer Simulation Study. *Int. J. Air-Cond. Refrig.* **2019**, 27, 1950031.
8. Tudoroiu, R.E.; Zaheeruddin, M.; Radu, S.M.; Budescu, D.D.; Tudoroiu, N. The Implementation and the Design of a Hybrid digital PI Control Strategy Based on MISO Adaptive Neural Network Fuzzy Inference System Models–A MIMO Centrifugal Chiller Case Study. In *Machine Learning Paradigms. Learning and Analytics in Intelligent Systems*; Tsihrintzis, G., Jain, L., Eds.; Springer: Cham, Switzerland, 2020; Volume 18. https://doi.org/10.1007/978-3-030-49724-8_9.
9. Zadeh, L.A. Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems-Selected Papers by Lotfi Zadeh. In *Advances in Fuzzy Systems—Applications and Theory*; Klir, G.J., Yuan, Bo., Eds.; State University of New York at Binghamton, USA: Binghamton, NY, USA, 1996; Volume 6, 840p. https://doi.org/10.1142/2895.

10. Haykin, S. *Neural Networks and Learning Machines*, 3rd ed.; Horton, M.J., Mars, D., Opaluch, W., Disanno, S., Eds.; Pearson Education, Inc.: Upper Saddle River, NJ, USA, 2009.
11. Zurada, J.M. *Introduction to Artificial Neural Systems*, 1st ed.; Davis, G., Quadrata, P., L., Eds.; West Publishing Company: St. Paul, MN, USA, 1992.
12. Ljung, L. System Identification Toolbox Getting Started Guide. The MathWorks, Inc.,1 Apple Hill Drive Natick, MA 01760-2098 © COPYRIGHT 1988–2023 by The MathWorks, Inc., MATLAB R2023a. Available online: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://control.dii.unisi.it/sysid/book/ident_gs.pdf(accessed on 12 June 2024).
13. Walia, N.; Singh, H.; Sharma, A. ANFIS: Adaptive Neuro-Fuzzy Inference System-A Survey. *Int. J. Comput. Appl.* **2015**, *123*, 32–38.
14. Jang, R. ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Trans. Syst. Man Cybern.* **1993**, *23*, 665–685. https://doi.org/10.1109/21.256541.
15. Almabrok, A.; Psarakis, M.; Dounis, A. Fast Tuning of the PID Controller in An HVAC System Using the Big Bang–Big Crunch Algorithm and FPGA Technology. *Algorithms* **2018**, *11*, 146. https://doi.org/10.3390/a11100146.
16. MathWorks Support Documentation-Help Center. Estimate ARMAX or ARMA Model. Available online: https://www.mathworks.com/help/ident/ref/armax.html (accessed on 12 June 2024).
17. Namdari, A.; Samani, M.A.; Durrani, T.S. Lithium-Ion Battery Prognostics through Reinforcement Learning Based on Entropy Measures. *Algorithms* **2022**, *15*, 393. https:// doi.org/10.3390/a15110393.
18. MathWorks Support Documentation-Help Center. Reinforcement Learning Using Deep Neural Networks. Available online: https://www.mathworks.com/help/deeplearning/ug/reinforcement-learning-using-deep-neural-networks.html (accessed on 10 September 2024).
19. MathWorks Support Documentation-Help Center. Reinforcement Learning for Control Systems Applications. Available online: https://www.mathworks.com/help/reinforcement-learning/ug/reinforcement-learning-for-control-systems-applications.html (accessed on 10 September 2024).
20. MathWorks Support Documentation-Help Center. Generate Reward Function from a Model Predictive Controller for a Servomotor. Available online: https://www.mathworks.com/help/reinforcement-learning/ug/generate-reward-fcn-from-mpc-for-servomotor.html (accessed on 10 September 2024).
21. MathWorks Support Documentation-Help Center. Generate Reward Function from a Model Verification Block for a Water Tank System. Available online: https://www.mathworks.com/help/reinforcement-learning/ug/generate-reward-fcn-from-verification-block-for-watertank.html (accessed on 20 September 2024).
22. Uzunovic, T.; Zunic, E.; Badnjevic, A.; Miokovic, I.; Konjicija, S. Implementation of digital PID controller. In Proceedings of the IEEE 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 24–28 May 2010; Volume 33, pp. 1357–1361.
23. Grosse, G.; Maddison, C.; Bae, J.; Pitis, S. Introduction to Machine Learning. Lecture 11-Reinforcement Learning, University of Toronto, Fall. 2020. Available online: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec11.pdf (accessed on 23 February 2025).
24. Abdel-Jaber, H.; Devassy, D.; Al Salam, A.; Hidaytallah, L.; El-Amir, M. A Review of Deep Learning Algorithms and Their Applications in Healthcare. *Algorithms* **2022**, *15*, 71. https://doi.org/10.3390/a15020071.
25. Kranthi Kumar, P.; Detroja, K.P. Design of Reinforcement Learning based PI controller for nonlinear Multivariable System. In Proceedings of the 2023 European Control Conference (ECC), Bucharest, Romania, 13–16 June 2023; pp. 1–6. https://doi.org/10.23919/ECC57647.2023.10178182.
26. You, W.; Yang, G.; Chu, J.; Ju, C. Deep reinforcement learning-based proportional–integral control for dual-active-bridge converter. *Neural Comput. Appl.* **2023**, 35, 1–14. https://doi.org/10.1007/s00521-023-08667-x.
27. MathWorks Support Documentation-Help Center. Resolving Low-Level Graphics Issues. Available online: https://www.mathworks.com/help/matlab/creating-plots/resolving-low-level-graphics-issues.html (accessed on 20 September 2024).