Temperature Control of a Simulated Nonlinear Greenhouse System with Reinforcement Learning

Aaron Rosenberg
Old Dominion University
5115 Hampton Boulevard, Norfolk, VA 23529
arose019@odu.edu

Abstract

This paper applies Reinforcement Learning (RL) using the Deep Deterministic Policy Gradient (DDPG) method to control the inside temperature of a nonlinear, MIMO greenhouse system. DDPG is considered model-free RL and is appropriate for systems with continuous states and actions. The DDPG algorithm trains an Actor and Critic network to solve the control problem of maintaining the inside temperature at a specific value. The trained agent achieves good setpoint control and has comparable performance to a PID controller designed to control the linearized version of the system. The trained agent exhibits some undesirable traits, and this paper presents some ideas for actions to correct the behavior.

I. Introduction

Greenhouses perform a critical role in protected crop cultivation by providing a specific microclimate year-round. This allows for numerous benefits such as the production of crops outside of their natural growing season or region as well as protecting crops from stressful changes in temperature [1].

Control of the greenhouse microclimate is of utmost importance to maximize the growth of the crops. The greenhouse system presented in [1] is modeled as a nonlinear, coupled, multiple-input multiple-output (MIMO) system of equations relating the temperature and humidity inside the greenhouse.

Traditionally, systems with these characteristics can be considered difficult to predict and control; however, advances in sensor technology and computer processing provide the means to implement various control strategies. Cevallos et al. demonstrate how to control the system by linearizing the nonlinear differential equations about an equilibrium point, decoupling the two equations, and tuning Proportional-Integral-Derivative (PID) controllers to control both the temperature and the humidity of the greenhouse system [1].

In recent years, with advancements in computer processing and machine learning, the use of Reinforcement Learning (RL) algorithms to control systems has grown in popularity. Reinforcement learning is a branch of machine learning in which a system, or agent, learns to interact with a complex environment with the goal of maximizing expected rewards over time [2]. Unlike other branches of machine learning, in RL the algorithm is not given examples of optimal outputs, but instead must discover the optimal outputs through trial and error [3]. RL allows for the simplification of learning complex behaviors because only a reward must be defined; the algorithm learns reward-maximizing behaviors itself [2].

RL presents a unique approach to solving control problems, because a model of the system is not necessary; the agent learns to control the system solely through interaction with the environment. However, this feature also makes RL difficult to implement on real systems; often, the RL agent cannot afford to train on systems that are expensive or safety critical. One workaround for this issue is to train the RL agent in a simulation of the system and to deploy the pretrained agent on the physical system [4].

RL has been applied to a variety of control problems using numerous methods in the literature. For example, Bates applied a Policy-Gradient, Actor-Critic, and Proximal Policy Optimization to an inverted pendulum system with the goal of keeping the pendulum upright for 500 time steps. The algorithms were trained in a simulation and then deployed to balance a real-life pendulum [4].

Another common control application in the literature involves using RL to control liquid levels in mixing tanks. The papers by Rastogi et al. and Noel both utilize Artificial Neural networks (ANN) to implement controllers to maintain a desired liquid level h_1 in one tank of the two-tank system [5], [6]. Jones and Kanagalakshmi unsuccessfully attempted to design a controller using a Deep Deterministic Policy Gradient (DDPG) algorithm to

control the height of two interacting fluid tanks. This implementation was unsuccessful due to the computational complexity of the proposed system [7]. Yifei and Lakshminarayanan use a Multi-Agent RL (MARL) implementation to control temperature and volume in a chemical tank [8]. In this case, each agent was constructed with the DDPG algorithm, and each agent controlled a separate state. The MARL implementation shown in [8] was able to achieve good tracking performance and good disturbance rejection.

This work aims to implement a DDPG RL Actor-Critic to control temperature in the greenhouse environment. The performance of the RL implementation is compared to a standard control theory PID implementation, obtained in the same manner as described in [1]. This paper is organized as follows: Section II mathematically derives the greenhouse system, Section III explains the RL theory and DDPG algorithm, and Section IV discusses the implementation of the RL environment, and actor-critic network. Section V presents the results and Section VI presents conclusions and further work.

II. Derivation of Mathematical Model

Cevallos et al. [1] defined the greenhouse system as a MIMO system with three inputs, three disturbances, and two outputs. Fig. 1 illustrates the relationship between all the variables in the greenhouse system.

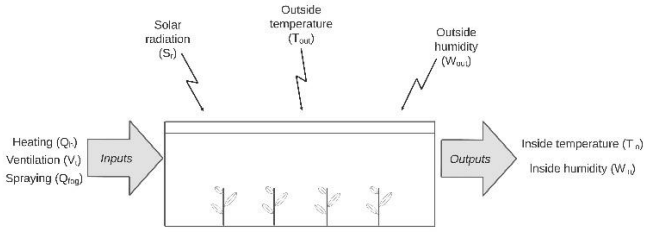

Fig. 1. Greenhouse System Variables [1]

The two outputs, or controlled variables, are the temperature and the humidity inside of the greenhouse. These variables are directly controlled with the three inputs: heating, fogging, and ventilation. The three disturbance variables include outside temperature, outside humidity, and heating due to solar radiation. These disturbance variables directly affect the output variables but cannot be directly controlled. The goal of a good control system is to achieve good tracking performance of the set point while simultaneously minimizing the impacts of the disturbances [9].

The governing differential equations of the greenhouse system model are derived from the energy and mass balances of the system [1]. Eq. (1) illustrates the change in temperature over time, derived from the energy balance of the system. Eq. (2) illustrates the change in the absolute humidity over time, derived from the mass balance of the system.

$$\frac{dT_{in}}{dt} = \frac{1}{\rho V C_p}\left(Q_h + S_r A - \gamma Q_{fog}\right) - (T_{in} - T_{out})\left[\frac{V_t}{V} + \frac{UA}{\rho V C_p}\right] \tag{1}$$

$$\frac{dW_{in}}{dt} = \frac{1}{\rho V}\left[Q_{fog} + E - V_t(W_{in} - W_{out})\right] \tag{2}$$

$$E = \alpha\frac{S_r A}{\gamma} - \beta_T W_{in} \tag{3}$$

Eq. (1) consists of five terms, from left to right: the heater term, solar term, fogger term, ventilation term, and conduction term. The heater and solar terms add energy to the system, increasing the temperature. The fogger term removes heat from the system through the addition of water vapor, decreasing the temperature. The ventilation and conduction terms can add or remove heat from the system, depending on the difference in temperature of the inside air and the outside air.

T_{in} represents the inside air temperature output variable (°C), ρ is the density of air (kg/m^3), V is the volume of air in the greenhouse (m^3), and C_p is the specific heat of air (J/kgK). Q_h is the heat added to the system from the heater input (W), S_r is the heat added to the system from the solar radiation disturbance (W/m^2), A is the surface area of the greenhouse (m^2), γ represents the latent heat of vaporization of water (J/g), and Q_{fog} represents the water vapor added to the system from the fogger input (gH_2O/s). T_{out} represents the outside air temperature disturbance (°C), V_t is the ventilation rate input variable (m^3/s), and U is the total heat transfer coefficient of the greenhouse glass (W/m^{2}K) [1].

Eq. (2) consists of three terms, from left to right: fogger term, evapotranspiration term, ventilation term. The fogger and evapotranspiration terms add water vapor to the system, while the ventilation term can add or remove water vapor depending on the difference in humidity of the inside and outside air. Evapotranspiration refers to the process that transfers water to the atmosphere via plants [10].

W_{in} represents the absolute humidity inside the greenhouse (gH_2O/m^3), W_{out} represents the absolute humidity outside the greenhouse (gH_2O/m^3), and E represents the evapotranspiration rate (gH_2O/s) [1].

Eq. (3) shows the governing equation for the evapotranspiration rate. The evapotranspiration rate is directly proportional to solar radiation and decreases as the humidity increases. α and β_T are dimensionless coefficients

that represent the leaf area index and thermodynamic constants affecting evapotranspiration, respectively [1].

Constant	Value	Units
V	4000	m^3
U	4	$W/(m^2K)$
A	1000	m^2
ρ	1.2	kg/m^3
C_p	1006	J/(kgK)
γ	2257	J/g
α	0.125	-
β_T	0	-

Table 1. Constants used in Greenhouse System Model

Table 1 lists the values for each constant used in the RL simulation for this paper. The values used for physical constants, such as density of air, specific heat of air, and latent heat of vaporization of water are the standard values at sea level for each of these constants. The remaining variables were chosen arbitrarily. For simplicity and ease of comparison, most of the values used by Cevallos et al. [1] were used in this simulation as well.

III. Reinforcement Learning

A. Reinforcement Learning in Control Systems

Aggarwal defines RL as a "reward-driven trial-and-error process" where a system, or agent, learns to interact with an environment to achieve rewarding outcomes [2]. In RL, the algorithm explores and exploits the environment to learn the optimal sequence of actions to maximize a given reward.

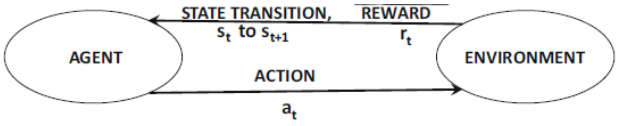

Fig. 2. RL Framework [2]

Fig. 2 illustrates a simplified framework for RL. At a given time step, t, the environment has a given state, s_t. The agent observes this state and provides an action, a_t, which changes the state of the environment to s_{t+1}. The agent also obtains a reward, r_t, from the environment at each time step based on the applied action. The agent updates its policy based on the reward achieved through interacting with the environment. One strength of RL is that it can utilize either a deterministic policy or a stochastic policy. Typically, in the context of control systems, a deterministic policy will be used [7].

This framework is analogous to a traditional closed-loop control system in that the controller (agent) provides an action based on the control law (policy) that affects the plant. The measured output and the difference between the output signal and the reference signal are passed to the controller as the observation, and then the controller outputs a new action based on the updated observation. In this example, the reward would be maximized when the difference between the measured output and the reference signal is zero [11].

Reinforcement learning can be further subdivided into "model-free" and "model-based" RL [4]. Model-free RL algorithms choose actions based only on observations of the state and rewards. In model-based RL, algorithms use a model of the environment to generate predictions of the next state, to calculate the optimal action to take. Despite using a model of the greenhouse to simulate the environment, the RL simulation performed in this paper is considered model-free, because the agent does not have access to the model to choose actions. The agent only knows the action provided to the environment and the subsequent observation resulting from the action.

Both model-free and model-based RL have a plethora of algorithms, as depicted in Fig. 3 [4].

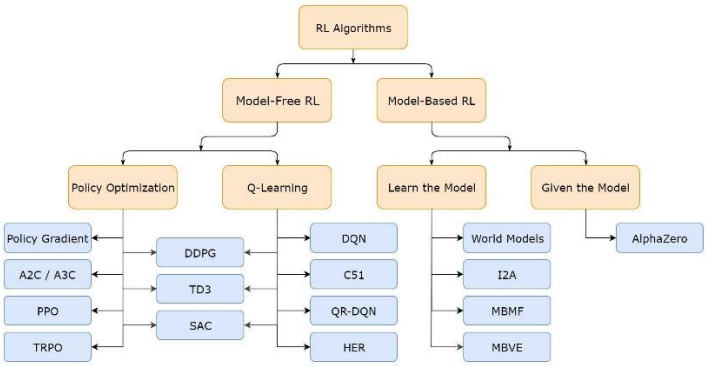

Fig. 3. Breakdown of RL Algorithms [4]

For the greenhouse model described in Section II, a DDPG Actor/Critic algorithm was chosen to control the system. This algorithm was chosen because both the action space and the observation space are continuous.

B. Deep Deterministic Policy Gradient

In RL, an agent's policy, π, maps states in the environment to a probability distribution of the actions [12]. This map defines the behavior of the agent. The agent's behavior can be modeled as a Markov Decision Process (MDP) with the following parameters [12]:

1. State Space, S.
2. Action Space, $A \in \mathbb{R}^N$, where N is the number of inputs.
3. Initial State distribution, $p(s_1)$.
4. Transition Dynamics, $p(s_{t+1}|s_t, a_t)$.
5. Reward Function, $r(s_t, a_t)$.

Given these parameters, an action-value function can be defined as the expected return after taking an action based on the current state and following the given policy. For a deterministic policy, this value function can be described with the Bellman equation where the deterministic policy, μ, maps states directly to actions [12].

The Bellman equation shows that the expected value of Q^{μ} only depends on parameters from the environment; the reward function depends on the state and the action and Q^{μ} depends on the next state and the action associated with the next state. This feature allows Q^{μ} to be learned "off-policy" [12]. Aggarwal defines an off-policy algorithm as an algorithm where target values used to update the neural network may differ from observed future actions [2]. The target prediction values are the best possible actions calculated using the Bellman equation. This target value may differ from the actual observed action, as the observed actions are selected based on the largest predicted reward [2].

Q-learning is a common off-policy algorithm in reinforcement learning; however, it is generally difficult or impossible to use Q-learning for continuous action spaces due to the size of the action space [12]. The DDPG algorithm gets around this limitation by using actor-critic methods and four function approximator networks.

C. DDPG Algorithm

The DDPG algorithm uses four networks: two for the actor and two for the critic. The networks are defined as follows [13], [7]:

1. Deterministic Actor, $\mu(s|\theta_{\mu})$
2. Critic, $Q(s,A|\phi)$
3. Target Actor, $\mu_t(s|\theta_t)$
4. Target Critic, $Q_t(s,A|\phi_t)$

The actor selects a deterministic action given observation, s. The actor's action is selected such that the long-term reward is maximized. The critic network calculates the expected long-term reward given the observation, s, and the action, A [13]. Essentially, the actor network maps the states to actions, and the critic network evaluates this mapping.

The target networks have the same structure as the regular networks and have parameters that update slower than the regular networks. This "lagging" helps to prevent divergence and improve training stability.

The DDPG algorithm also takes advantage of a replay buffer. This buffer stores past experiences and allows the agent to update the weights of the actor and critic via a randomly sampled mini-batch of past experiences, rather than only using the most recent experience to train the network [7], [13].

Due to the deterministic nature of the actor network, an important problem to consider is the problem of exploration versus exploitation [7]. The goal of the RL agent is to maximize a reward; however, if the agent only selects actions that immediately reward the agent, then the agent may not discover advantageous states or actions that maximize the long-term reward. One solution to this problem is to add noise to the output of the existing network, to force the agent to sometimes take "non-optimal" actions and explore the action space [7].

Lillicrap et al. created the standard algorithm for training the DDPG agents in [12]. The DDPG training algorithm starts by initializing the both the actor and critic network with random parameters, θ_{μ} and ϕ, respectively. The target actor and critic network parameters are set equal to the parameters of the regular actor and critic, i.e. $\theta_t = \theta_{\mu}$ and $\phi_t = \phi$. The algorithm also requires that the environment be initialized to a random state for each training iteration, to allow the agent to explore more of the state space.

After initialization, the algorithm iterates through simulation time steps. For each time step, the agent selects an action based on the current state and policy with added noise, applies the selected action to the environment, and observes both the reward and the next state. The "experience" consists of the current state, action, reward, and new state, and is stored in the replay buffer every training step. The algorithm randomly samples experiences from the buffer and updates the "target value function", y_t.

The critic parameters are updated by minimizing the loss, L, which is the mean-squared difference between the original Q-value and the new Q-value. The actor parameters are updated using policy gradient methods, which involve calculating the gradient of the expected return function. Finally, the target networks are updated using smoothing functions that change parameters slower than the gradients for the regular networks. Fig. 4 shows the complete pseudocode for the DDPG algorithm [12].

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^{\mu})$ with weights θ^Q and θ^{μ}.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

  end for
end for

Fig. 4. DDPG Algorithm Pseudocode [12]

IV. Application of RL to Greenhouse System

The implementation of RL to the Greenhouse System described in Section II consisted of three major steps: creating the simulation environment, creating the agent, and training and validating the agent in the simulated environment. In this work, a custom MATLAB environment template was used to simulate the greenhouse environment. The template consists of five key sections.

The first template section is the properties section, where the constants described in Table 1 are defined. Each of these constants were chosen to approximate realistic values for a greenhouse in Norfolk, VA. In addition to the constants defined above, the properties section also holds the state variable, time steps, and set point variables.

The time steps were chosen based on the control theory analysis performed in [1]. Through linearizing the differential equations, decoupling the inputs, and designing PID controllers for each input, Cevallos et al. were able to achieve good tracking performance and disturbance rejection within 15 to 30 minutes of a change in reference or disturbance. Given these results, in this simulation, time steps of one second were chosen, with a total simulation time of one hour (3600 seconds) for each training iteration.

The set point variables are the desired values of the inside temperature and humidity and are set to 25°C and 16 g/m^3, respectively. The state variable consists of the observations from the environment and is the only information that is passed to the actor. The state variable is a vector containing the two output variables, inside temperature and inside humidity, as well as the three disturbance variables, outside temperature, outside humidity, and solar radiation.

The second section of the environment template is the constructor function. This function sets the limits of the observation (the state variables) and the actions (the input variables). Both the observation and the action vectors are continuous, with each observation variable ranging from negative infinity to positive infinity. The action variables are restricted to operate within realistic limits; heater values range from [0,100] kW, the fogger values range from [0,100] gH_2O/s, and the ventilation values range from [0,100] m^3/s. For simplicity in the simulation, these action values have been normalized on the range [0,1] and scaled to the appropriate values in the environment.

The reset and step functions form the third and fourth sections of the environment template, respectively. To capture a realistic range of temperatures and humidities that the greenhouse system would be expected to operate in, climate normal data from the World Meteorological Organization (WMO) was obtained. WMO collects the climate standard normal data over a 30-year period, from 1991-2020 and presents the averages by location, by month [14]. This data was used to generate a normal distribution with the mean and standard deviations of the temperature and humidity data as parameters. A similar process was used for the solar radiation disturbance, with data obtained from [15]. The reset function uses these calculated distributions to randomly set the state at the start of each training iteration and provide the state to the agent.

The step function provides the mechanism for the agent to interact with the environment. This function takes the current state of the environment and the chosen action and calculates the next state using the differential equations presented in Eq. (1) and Eq. (2).

The reward function is the final key section of the environment template and significantly affects the behavior of the agent. The reward function in this simulation experienced numerous iterations; the first function used was simply the "error" signal from control theory, i.e. the difference between the current value and the desired value. This reward was not effective for two reasons: the magnitude of the temperature error differed from the magnitude of the humidity error, so the agent would prioritize controlling one while neglecting the other, and the agent would take the entire simulation time to reach the desired value. The final reward function used in this simulation attempted to diminish these behaviors by introducing rewards for driving the temperature within certain bounds under a given time, and penalizing the agent if the temperature was outside of these bounds.

The agent contains the actor and the critic networks. The actor network is comprised of a feature input layer that takes the environment state as an input and three 100-node fully connected layers with Rectified Linear Unit (ReLU) activation functions in between. The final ReLU layer connects to a fully connected layer with three nodes, one node for each action. Each action node is passed to a sigmoid activation layer to keep the value of each action between zero and one.

The critic network has two branches because it takes both the observation and the action as inputs. The observation branch connects the observation to two 100-node fully connected layers with a ReLU layer in between. The action branch takes the action as input and connects the action to one 100-node fully connected layer. The addition layer combines both branches and passes the output to a ReLU layer, a fully connected layer, and another ReLU layer. The final ReLU layer connects to a one-node fully connected layer to output the critic's Q-value. Fig. 5 illustrates the actor and critic networks.

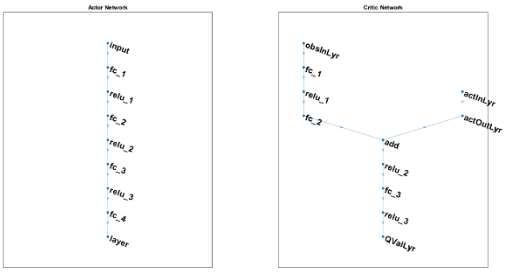

Fig. 5. Actor Network (left), Critic Network (right)

Training was performed over 2440 iterations with the following parameters: the agent used an experience buffer with a length of 10^6 and mini-batches of 64 experiences, the actor had a learning rate of 10^{-4} while the critic's learning rate was 10^{-3}, and the noise added to the actor for choosing actions was Ornstein-Uhlenbeck with a variance of 0.3 for each action.

V. Results

Training the agent over the 2440 iterations required approximately 16 hours with results shown in Fig. 6. Initially, training involved controlling both temperature and humidity, but the results were inconsistent, and the computation was too time intensive. For simplicity, the requirement to control the humidity was removed and the focus was shifted to simply control the temperature using the three actions. Overall, the agent did not seem to adequately maximize the reward function; however, a few episodes had large spikes in the resultant reward.

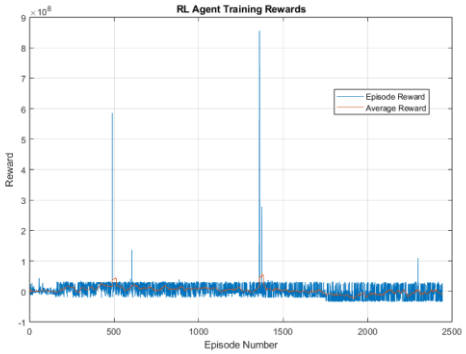

Fig. 6 Training Rewards

Despite the relatively low training rewards, the agent was able to achieve decent control results. Fig. 7 displays the results of the agent controlling the inside temperature for four separate initial states.

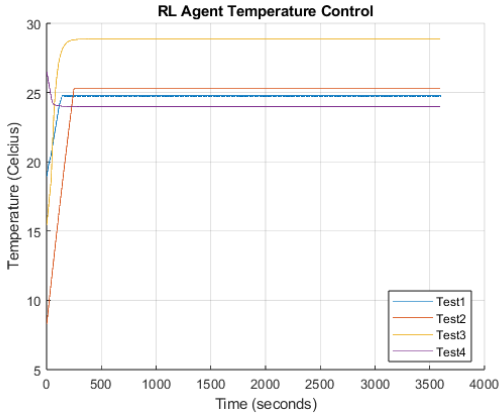

Fig. 7 Agent Test Results

With control systems, typically the goal is for the controller (agent) to drive the system exactly to the desired state. The current iteration of the agent is not able to achieve this requirement; however, it can drive the system close to the required setpoint, which is acceptable for this application. Additionally, the agent successfully learned to control the system to its steady-state value within 250 seconds, which is desirable.

Fig. 8 compares the RL controller results for Test 1 to a PID controller derived through the methods discussed in [1]. The RL controller is significantly faster than the PID controller, reaching its steady state value in approximately 240 seconds, while the PID reaches its settling time in approximately 900 seconds. However, the PID does not have a steady state error while the RL controller does.

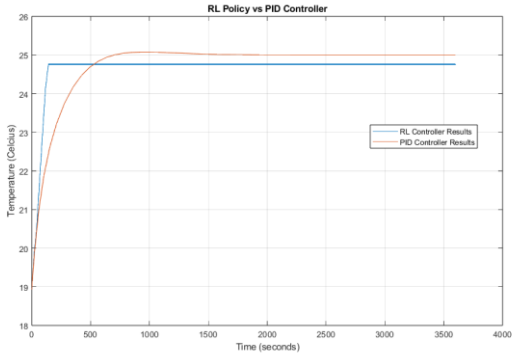

Fig. 8 RL Controller vs PID for Test 1

Table 2 summarizes the results and compares the integral square error (ISE) of the RL agent controller to the PID controllers for each initial state. The error is defined as the difference between the setpoint temperature and the inside temperature at each timestep, and the ISE is the sum of the squared error over the entire simulation time.

Test	Initial State $[T_{in}$; T_{out} ; $S_r]$	RL ISE (10^3)	PID ISE (10^3)
1	[18.93 ; 19.27 ; 143.62]	1.27	8.12
2	[8.28 ; 20.57 ; 215.91]	1.05	21.25
3	[15.43 ; 26.02 ; 253.55]	-13.00	12.42
4	[26.52 ; 3.76 ; 93.24]	3.54	0.820

Table 2. Comparison of RL Agent Controller to Linearized PID

The RL agent achieved good performance for the tests presented in Table 2, with the RL controller outperforming the PID in Test 1 and Test 2 and the PID outperforming the RL controller in Test 3 and Test 4.

The agent frequently displayed undesirable behavior if the initial T_{in} state was higher than the set point. After analyzing the actions taken by the agent over a variety of initial states where T_{in} was greater than 25°C, it appears that the agent does not use the fogger input, and only cools the system using ventilation. The ventilation can rapidly cool the system but becomes less effective if the difference between the inside and outside temperature is small, whereas the fogger input should provide a constant cooling effect. It is likely that the agent did not train on a sufficient number of situations where the starting initial temperature was higher than the setpoint due to the temperature data distribution used in the reset function.

VI. Conclusion

This paper presents an implementation of reinforcement learning using the Deep Deterministic Policy Gradient algorithm to train an agent to control a nonlinear, MIMO system. The results of the trained agent were compared to a linearized PID controller, and the RL agent achieved comparable results. The agent works well when the initial temperature is lower than the setpoint but exhibits undesirable behavior when the initial temperature is higher than the setpoint.

Future work for this project would involve correcting the undesired behavior and improving overall performance. There are numerous possible options to achieve these goals, including: increasing the number of training iterations, training separate agents for specific temperature ranges, adjusting the reward function to reward a tighter range to the setpoint, and attempting a MARL approach and have separate agents for each action.

References

[1] G. Cevallos, J. Pinzon, and O. Camacho, "A microclimate greenhouse multivariable control: A guide to use hardware in the Loop Simulation," 2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA), 2022.

[2] C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook. New York: Springer, 2018.

[3] C. M. Bishop, Pattern Recognition and Machine Learning. New York: Springer, 2006.

[4] D. Bates, "Virtual Reinforcement Learning for Balancing an Inverted Pendulum in Real Time.," Dissertation, North Carolina State University, 2021.

[5] D. Rastogi, M. Jain, M. M. Rayguru, and S. K. Valluru, "Design & Validation of ANN based Reinforcement Learning Control Algorithm for Coupled Tank System," Apr.2023,doi: https://doi.org/10.1109/i2ct57861.2023.10126494.

[6] M. M. Noel and B. J. Pandian, "Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach," Applied Soft Computing, vol. 23, pp. 444–451, Jun. 2014. doi:10.1016/j.asoc.2014.06.037

[7] D. M. Jones and S. Kanagalakshmi, "Data driven control of interacting two tank hybrid system using deep reinforcement learning," 2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA), 2021. doi:10.1109/iccca52192.2021.9666405

[8] Y. Yifei and S. Lakshminarayanan, "Multi-agent reinforcement learning system for multiloop control of Chemical Processes," 2022 IEEE International Symposium on Advanced Control of Industrial Processes(AdCONIP),2022. doi:10.1109/adconip55568.2022.9894204

[9] S. Skogestad and I. Postlethwaite, Multivariable Fedback Control: Analysis and Design. Chichester: John Wiley & Sons, Ltd., 2005.

[10] S. Trimble, "Transpiration in plants: Its importance and applications," CID Bio-Science, https://cid-inc.com/blog/transpiration-in-plants-its-importance-and-applications/ (accessed Nov. 15, 2023).

[11] "Reinforcement Learning for Control Systems Applications," Reinforcement learning for control systems applicationsMATLAB&Simulink, https://www.mathworks.com/help/reinforcement-learning/ug/reinforcement-learning-for-control-systems-applications.html (accessed Sep. 15, 2023).

[12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, Continuous control with deep reinforcement learning, CoRR abs/1509.02971 (2015).

[13] Deep Deterministic Policy Gradient (DDPG) Agents, https://www.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html (accessed Nov. 15, 2023).

[14] WMO Member Nations (2023). WMO Climatological Standard Normals for 1991-2020 (NCEI Accession 0253808). Temperature Normals. NOAA National Centers for Environmental Information. Dataset. https://www.ncei.noaa.gov/archive/accession/0253808. Accessed October 30, 2023

[15] Solar calculator | irradiance and angle calculators, http://www.solarelectricityhandbook.com/solar-calculator.html (accessed Oct. 30, 2023).