Original Research Article

Optimizing Automatic Voltage Regulation: A Deep Reinforcement Learning Approach

Ahmed Sweilam O. Mohsen [1], M.A. Moustafa Hassan[2]

1 Electrical Power Engineering Department, Faculty of Engineering, Cairo University, Giza, Egypt,

e-mail: ahmed.sweilam95@gmail.com

2 Electrical Power Engineering Department, Faculty of Engineering, Cairo University, Giza, Egypt

e-mail: mmustafa@eng.cu.edu.eg

***Corresponding author**： *ahmed.sweilam95@gmail.com*

Abstract: This paper presents a novel approach to enhance the performance of Automatic Voltage Regulator (AVR) systems in power systems using Deep Reinforcement Learning (DRL). The AVR plays a critical role in maintaining voltage stability and ensuring reliable power delivery. However, conventional control strategies, such as PID controllers, have limitations in handling complex and nonlinear power system dynamics. In this study, the application of DRL techniques is explored, particularly the Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm, to AVR control. This algorithm offers the advantage of handling continuous action spaces and enable the controller to learn optimal control policies directly from the system's state information. The results show that the DRL approach outperforms the traditional PID and Neural Network-based control approaches, with the shortest time response and the best voltage regulation performance. The use of DRL in AVR system control shows promising potential for improving the efficiency and accuracy of power system control. This research provides insights into the advantages of DRL for process control and highlights its potential for future applications in power system control.

Keywords: Deep reinforcement learning, Process control, AVR system, TD3 agent, PID controller, Neural network controller, Optimization, Artificial intelligence.

1. Introduction

Ensuring a stable terminal voltage and frequency is crucial for maintaining a reliable power system that can meet the needs of all its users. Voltage stability is a critical issue that greatly impacts the reliability of power systems, and has been the subject of extensive research over the years. By adjusting the excitation current, it is possible to regulate both the generator terminal voltage and the amount of reactive power supplied to the grid. The stability of the power system is critically dependent on the synchronous generator's excitation mechanism. AVR is a closed-loop control system made to regulate the synchronous generator's terminal voltage and maintain it within specified limit ranges[1].

While new algorithms have been proposed for regulating synchronous generator’s terminal voltage, many industrial systems still rely on the conventional Proportional-Integral-Derivative (PID) based AVR design due to its simplicity and ease of implementation[2]. Conventional PID controllers have certain drawbacks. Firstly, they require full knowledge of the system, which is a difficult task to achieve. Additionally, they are linear controllers which makes them unsuitable for nonlinear systems that are prevalent in real-world applications.

Nonlinear control techniques have gained popularity in recent years and are continuously evolving. These techniques have been demonstrated to be more efficient in ensuring stability in power systems. Artificial Neural

Received: 2 June 2023 Accepted: 1 August 2023 Available online: 13 Novermber 2023

Networks (ANNs) are one such nonlinear approach that can be particularly useful for modeling complex, dynamic power systems. ANNs with their nonlinear structure are well-suited for modeling systems that are challenging to represent mathematically. Moreover, ANNs' capability as universal function approximators enables them to accurately approximate any continuous nonlinear function with arbitrary precision[3].

In this paper, conventional PID based AVR controller will be developed where PID parameters will be optimized using Particle Swarm Optimization (PSO) till satisfactory response is achieved then several input and output patterns will be compiled and stored as data which will be used to train the proposed AVR controllers, ANN and Adaptive Neuro-Fuzzy Inference System (ANFIS), in MATLAB/Simulink environment. Model free DLR will be introduced then finally results will be compared.

2. Related Work

While PID-based controllers are commonly used for AVR, their tuning process is manual and can result in suboptimal performance. Various optimization techniques, such as Ziegler Nichols (ZN), Genetic Algorithms (GA) and PSO, have been proposed to enhance PID tuning, but they still have limitations in dealing with nonlinearities and uncertainties.

ZN method has been widely employed for tuning controllers in the past, as evidenced by[4]. However, this method has limitations in terms of ensuring optimal performance, as its efficacy is primarily dependent on the complexity and order of the plant. Both GA and PSO are advanced techniques that are often utilized to improve PID tuning. The study in[5] employed both ZN and GA for PID tuning, with GA exhibiting better performance over ZN. The study conducted in[6], introduced the use of PSO to efficiently search for the optimal PID controller parameters of an AVR system.

New control system design strategies, such as ANNs and ANFIS, have been investigated and developed in recent decades to deal with the significant nonlinearities seen in most real control systems. In[7], it was demonstrated that ANNs AVR exhibited superior performance when compared to conventional PID controllers. In[8], Probabilistic Neural Network (PNN) based AVR controllers were implemented in MATLAB/Simulink and showed improved transient stability and reduced overshoot and settling time compared to conventional PID-based controllers at various loading conditions.

The usage of ANFIS based controller combines the advantages of both fuzzy logic and NN to achieve a more accurate control of the AVR system. In[9], a design procedure for ANFIS-based AVR is presented, which results in an improved system dynamic response when compared to conventional AVRs.

The application of Deep Reinforcement Learning (DRL) in controlling AVR systems is a recent research trend, where the controller learns from interacting with the environment to achieve optimal performance. While still in the early stages, DRL-based AVR controllers have presented promising results and potential for further improvement. The use of DRL is particularly suitable for controlling AVR systems in dynamic and uncertain environments, where traditional control methods may struggle to adapt. Therefore, DRL has the potential to revolutionize the design and operation of AVR systems in the future.

3. System Model and Proposed Techniques

It is widely recognized in the power systems community that variations in real power demand primarily impact system frequency, while changes in reactive power demand mainly influence the voltage magnitude. To regulate the reactive power output of a generator, the most commonly employed method is to manipulate the generator excitation control using AVR system.

An increase in the generator's reactive power load leads to a reduction in the magnitude of the terminal voltage. A potential transformer can be used for measuring the voltage magnitude on a single phase, which is then rectified and compared to a DC set point signal. The generated error signal is then employed to regulate the exciter field and raise the exciter terminal voltage. As a result, there is an increase in the generator field current, leading to a rise in the generated emf. Therefore, the production of reactive power is adjusted to reach a new

equilibrium, resulting in an increase in the terminal voltage to the desired level[10].

3.1 Model of an AVR System

To simplify the comparison between different algorithms, a linearized model of the AVR system has been utilized for performance evaluation. AVR system is composed of four primary components: the amplifier, exciter, generator, and sensor. In order to establish the mathematical model and transfer function of these components, linearization involves considering the primary time constants and neglecting nonlinearities such as saturation [11]. The schematic diagram of a simplified AVR is illustrated in Figure 1.

As presented in[12], the adopted values of an AVR system are given by: KA = 10; KE =1; KG =1; KS =1; TA=0.1s; TE =0.4s; TG =1s; TS =0.01s.

3.2 PSO Based PID Controller

PSO is an evolutionary computation technique inspired by the collective behavior of social animals such as birds and fish. The algorithm works by having individual particles where each moves individually and accelerates towards the personal best location while evaluating the fitness value of its current position. The fitness value for a particular position is determined through the evaluation of a fitness function at that specific location.

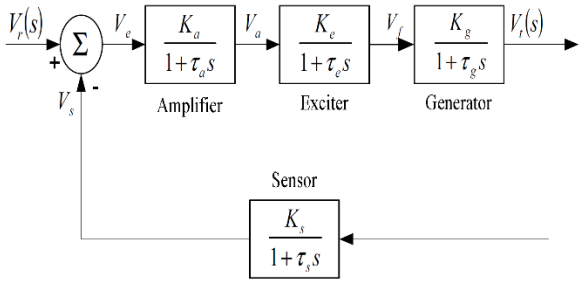

Figure 1 Synchronous generator with AVR only.

The fitness function is the objective function that is used to determine the optimality of a solution. The PSO algorithm uses the fitness function to determine the quality of the current solution and guide the particles towards the optimal solution. So, in the event that a particle's current position has a fitness value superior to its personal best, the personal best is then updated to the current position. Every particle within the swarm is aware of the global best, which is the location with the best fitness value for the entire swarm. As the particles move along their trajectory, they compare the fitness value of their Personal Best (Pbest) with that of the Global Best (gbest) at each point. If a particle's Pbest has a higher fitness value than the current gbest, then the gbest is updated to the Pbest of that particle. This allows the swarm to collectively converge towards the global optimum. Finally, once all particles approach the position with the best fitness value of the swarm, their movement is stopped[13,14]. Each particle within the swarm has its own position and velocity, (X_i, V_i) which are updated as following:

$$v_i^{k+1} = wv_i^k + c_1 r_1 (Pbest_i^k - x_i^k) + c_2 r_2 (Gbest^k - x_i^k) \quad \text{(Eq.1)}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad \text{(Eq.2)}$$

Where:

v_i^k : Velocity of the i^{th} particle at the k^{th} iteration.

x_i^k : Current position (or solution) of the i^{th} particle at the k^{th} iteration.

c_1, c_2 : Acceleration coefficients, usually are Eq. 2.

r_1, r_2 : Two random variables with uniform distribution between 0 and 1.

w : Inertia weight which shows the effect of the previous velocity vector on the new vector.

$Pbest_i^k$: Personal best position of the i^{th} particle at the k^{th} iteration.

$Gbest^k$: Global best position within the swarm at the k^{th} iteration.

As presented in[14], Figure 2 describes pseudocode of PSO algorithm:

Step 1. Initialization

For each particle $i = 1$: N, do

a) Initialize the particle's position uniformly distribution as $X_i\ (0) \sim$ U (LB, UB), where LB and UB represent the lower and upper bounds of the search space
b) Initialize $pbest$ to its initial position: $Pbest_i(0) = X_i(0)$.
c) Initialize $gbest$ to the minimal value of the swarm:$Gbest_i(0) = argmin[X_i(0)]$.
d) Initialize velocity: $V_i \sim (-|\text{UB} - \text{LB}|, |\text{UB} - \text{LB}|)$.

Step 2. Repeat until a termination criterion is met

For each particle $i = 1$: N, do

a) Pick random numbers: $r_1, r_2 \sim (0,1)$.
b) Update particle's velocity using Equation (4.19).
c) Update particle's position using Equation (4.20).
d) ***If*** $fitness[X_i(Iter)] < fitness[Pbest_i(Iter)]$ do,
   i. Update the best-known position of particle i: $Pbest_i(Iter) = X_i(Iter)$
   ii. ***If*** $fitness[X_i(Iter)] < fitness[Gbest_i(Iter)]$,
   Update the swarm's best-known position: $Gbest_i(Iter) = X_i(Iter)$
e) $Iter \leftarrow (Iter + 1)$;

Step 3. Output $Gbest_i(Iter)$ that holds the best-found solution.

Figure 2 Peseudo code of PSO Algorithm.

In the context of tuning a PID controller using PSO, the fitness function can be designed to evaluate the performance of the PID controller. So, a fitness function based on dynamic performance indices, as expressed in Eq. 3, is utilized as follows:

$$J = w_1 * E_{ss} + w_2 * \%OS + w_3 * t_s + w_4 * t_r \quad \text{(Eq.3)}$$

Where:

$e(t)$	: Error signal in time domain
$\%OS$	: Overshoot Percentage
t_r	: Rise Time
t_s	: Settling Time
w_1, w_2, w_3	: Weighting factors used to determine which performance criteria is more important.

3.3 Artificial Neural Networks

The human brain is a sophisticated and intricate system that operates in a nonlinear and parallel manner. This enables it to perform complex tasks such as pattern recognition, perception, and cognitive control at remarkable speeds, surpassing any current computer technology available.

ANNs are a type of machine learning that imitates the structure of the human brain. The objective of ANNs is to replicate the biological mechanisms that form the basis of information processing in the human brain, such as pattern recognition, decision-making, and perception. ANNs consist of a network of artificial neurons that communicate with each other through connections, or synapses. These artificial neurons are modeled using a nonlinear differential function, such as a sigmoidal function. ANNs can be composed of multiple layers, including input, hidden, and output layers, to enable complex computations[15].

ANNs are commonly used in supervised learning tasks, which involves training a model to make predictions or classifications based on input data that is labeled with the correct output, due to their ability to approximate and learn complex non-linear relationships between input and output variables. The back-propagation training algorithm is the most commonly used training algorithm for supervised learning using ANN. In this algorithm, the input data is fed into the network, which processes it through a series of layers that apply a non-linear transformation to the input. The output is then compared to the correct output label, also known as the target value, and the error is backpropagated through the network to adjust the weights of the connections between the

neurons[16,17].

Algorithm 1 demonstrates the pseudo-code for the back-propagation training algorithm[23].

Algorithm 1: The back-propagation algorithm for learning in multilayer networks.[23]

function BACK-PROP-LEARNING ($examples, network$) **returns** a neural $network$
  inputs: examples, a set of examples, each with input vector **x** and output vector **y**
    network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
  for each weight $w_{i,j}$ in $network$ **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example (**x**, **y j**) in examples **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
    for each node j in layer ℓ **do**
      $in_j \leftarrow \sum_i w_{i,j} a_i$
      $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ to 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in $network$ **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return $network$

3.4 Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) is a subfield of machine learning that combines the principles of Reinforcement Learning (RL) with Deep Neural Networks (DNN). Figure 3 illustrates the fundamental elements of the RL framework[18]. The agent is the responsible for learning and decision-making. It interacts with the environment by selecting actions A_t, which lead the environment to a new state S_{t+1}. The environment provides feedback on performance through rewards or penalties, represented as R_{t+1}. As the agent interacts with the environment, It seeks to maximize the rewards which encourages good actions allowing it to learn the best policy.

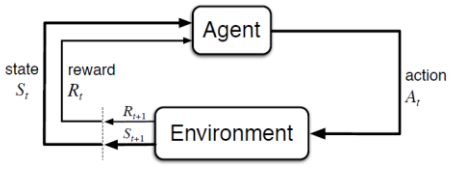

Figure 3 Building blocks of standard RL problem.

In the field of control systems, it is common to refer to the controller being designed as the agent and to the system outside the controller, including the industrial process, reference signal, and other sensors, as the environment. The desired optimal-control behavior that the designer seeks is referred to as the policy. RL makes it possible to learn the desired behavior without requiring excessively detailed modeling of the system.

The fundamental elements of RL are briefly described as follows:

Policy: is a mechanism that defines an agent's behavior at any given time. It maps the states of the environment to the corresponding actions that the agent should take in those states. It is a critical component of an RL agent since it determines the agent's behavior. The policy can be deterministic, represented by (s), or stochastic, represented by $\pi(a|s)$. Where $\pi(a|s)$ represents the probability of taking action $a_t = a$ when the state $s_t = s$.

Reward: A reward function, denoted as r_t, serves as a numerical signal provided to an RL agent, reflecting its performance in relation to its objectives. This function supplies immediate feedback from the environment based on the agent's actions. As the agent interacts with the environment, it receives rewards for favorable actions and penalties for unfavorable ones. In this way, the reward function guides the agent's decision-making policy, motivating it to pursue actions that lead to higher cumulative rewards over time.

Value function: is a function that estimates the desirability of taking a specific action in a given state. While a reward signal gives feedback on the immediate goodness of the current action, value function provides an evaluation of long-term goodness based on the expected cumulative sum of discounted rewards that the agent is likely to receive starting from a given time step t onwards. Where the return, denoted by R_t, is expressed as follows:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad , \quad 0 \leq \gamma \leq 1. \tag{Eq.4}$$

The value function, denoted by $V_\pi(s)$, provides an estimation of the expected return for a given state s when the agent follows a specific policy π,

$$V_\pi(s) = \mathbb{E}[R_t | s_t = s] \tag{Eq.5}$$

The example, illustrated in Figure 4, demonstrates the difference between Reward and Value functions.

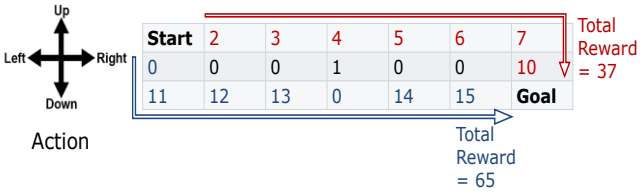

Figure 4 Examples for The Difference Between Reward and Value Functions.

Action-Value function: also referred to as the Q-function, denoted by $Q_\pi(s, a)$, represents the expected cumulative sum of rewards the agent can expect to receive starting from state s, taking a specific action a, and thereafter following a specific policy π. It provides an estimate of how good it is to perform a given action in a given state.

It can be expresses as follows:

$$Q_\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a] \tag{Eq.6}$$

Model-based and model-free RL methods: Model-based methods rely on having an accurate model of the environment, which provides access to a table of probabilities for being in a state given an action, as well as associated rewards. This allows for planning the next action and reward. In contrast, model-free methods for RL do not require an explicit environment model and rely solely on trial-and-error learning using sensory input[18].

The agent learns either a value function or a policy that enables it to make decisions without the need for environment simulation. However, this approach may require more data and experience to converge to an optimal solution. Although both methods have been extensively studied in the literature, model-free techniques have gained greater prevalence[19]. Therefore, this research paper will focus on model-free RL in the following sections.

Q-learning: is a Temporal-Difference (TD) control algorithm that enables the iterative learning of Q-values for each state-action pair. The algorithm tracks the value of $Q_\pi(s, a)$ for every state-action pair. Upon performing an action, a in a state s, the algorithm updates $Q_\pi(s, a)$ using two feedback elements from the environment: the reward R and the subsequent state S_{k+1}, as demonstrated in Eq.7, where alpha (α) represents the learning rate. The off-policy nature of Q-learning allows the algorithm to learn the optimal policy, even when following a different exploration policy.

$$Q^{new}(s_k, a_k) = Q^{old}(s_k, a_k) + \alpha \left(r_k + \gamma \max_a Q(s_{k+1}, a) - Q^{old}(s_k, a_k) \right) \tag{Eq.7}$$

Actor–Critic methods: The actor-critic algorithm is a popular real-time RL method that combines aspects of value-based and policy-based methods. It consists of two main components: the actor and the critic. The actor learns a policy for selecting actions, while the critic learns the value function for each state. Under a given policy, the actor applies an action to the environment and receives feedback, which is evaluated by the critic. The learning process involves two steps: first, the critic performs policy evaluation, and then the actor performs policy improvement. This algorithm enables continuous state and action spaces and supports online learning. Its success has been demonstrated in various applications such as robotics, and control systems.

Within the field of DRL, parameterized policies are commonly employed. A parameterized policy is a policy that outputs a value based on a set of adjustable parameters, represented as θ. These parameters can be modified using an optimization algorithm, resulting in changes to the policy's output. The policy that is controlled by θ is referred to as π_θ. Parameterized policies are frequently implemented using NN, where θ corresponds to the weights and biases of the network. It is important to note that in this context, the optimization algorithm aims to find the optimal set of parameters θ that maximizes the expected return.

Commonly used algorithms in control systems: Applying RL in continuous control systems poses significant challenges, such as dealing with high-dimensional state and action spaces and ensuring stable learning. Recently, two algorithms have exhibited promising results in addressing these challenges: Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3).

DDPG: is an off-policy actor-critic RL algorithm that extends the deterministic policy gradient algorithm to work with continuous action spaces. It utilizes an actor network to estimate the optimal policy and a critic network to approximate the Q-value function. DDPG is known for its stability and scalability in dealing with high-dimensional state and action spaces.

Algorithm 2 demonstrates pseudo-code for DDPG algorithm as explained in[20].

Algorithm 2: DDPG algorithm

Randomly initialize critic network $Q(s, a \mid \theta^Q)$ and actor $\mu(s \mid \theta^\mu)$ 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 $\mathcal{R}$
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 $a_t = \mu(s_t \mid \theta^\mu) + \mathcal{N}_t$ 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 $\mathcal{R}$
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
    Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right)$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i \mid \theta^Q)\right)^2$
    Update the actor policy using the sampled policy gradient:

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

    Update the target networks:

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

  end for
end for

TD3: is an extension of the DDPG algorithm that uses two critics to prevent overestimation of the Q-value function. It also employs a target policy smoothing technique to regularize the learned policy and improve its stability. TD3 has been demonstrated to outperform DDPG on various continuous control tasks and is considered one of the state-of-the-art algorithms in this field.

Algorithm 3 demonstrates pseudo-code for DDPG algorithm [21].

Algorithm 3: TD3 algorithm

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{R}$
for $t = 1$ to T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{R}$
  Sample mini-batch of K transitions (s, a, r, s') from $\mathcal{R}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \ \epsilon \sim \text{clip}\ (\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \underset{\theta_i}{arg\,min}\, K^{-1} \sum \left(y - Q_{\theta_i}(s, a)\right)^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:

$$\nabla_\phi J(\phi) = K^{-1} \sum \nabla_a Q_{\theta_1}(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$

    Update target networks:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$$
$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

  end if
end for

4. Simulation Results and Discussion

In this section, the simulation results and discussion will be presented for the proposed controller design for AVR systems using DRL. At first, the model of the AVR system is introduced, and the time response of the system without any controller is presented. This serves as a baseline for comparing the performance of the controllers were implemented in our study. The results of the PSO-based PID controller are then presented, including details about the experimental setup, such as the number of populations in the swarm and the objective function weights used for optimization. Insights into the tuning of the PSO-based PID controller are also provided, and its limitations in terms of performance are discussed. Subsequently, the NN-based controller is presented, which is implemented as a supervised learning algorithm using training data obtained from the previous technique. The design of the NN, including the number of layers, the number of neurons, and the chosen activation function, is explained. Finally, the DRL-based AVR controller is presented. Additionally, the time response of the AVR system under the DRL-based controller is presented and compared to the other controllers. In summary, a comprehensive analysis of the performance of different controllers for AVR systems is provided, with the strengths and limitations of each approach being highlighted.

4.1 System Without Controller

As a first step, AVR system response without controller is discussed using linearized model of system presented in Section 0.

4.1.1 Simulink Model

4.1.2 Terminal Voltage Response Without Controller

Simulation result displayed in Figure , demonstrates the importance of using a controller. where the system response exhibits oscillatory response with a large SSE, which is not acceptable for stable and reliable power systems.

4.2 PSO Based AVR Controller

As discussed earlier, a dynamic performance indices-based multi-objective function will be used. Where its weights are chosen as follows:

$$w_1 = 0.5\ ,\ w_2 = 0.5\ ,\ w_3 = 0.5\ ,\ w_4 = 0.5$$

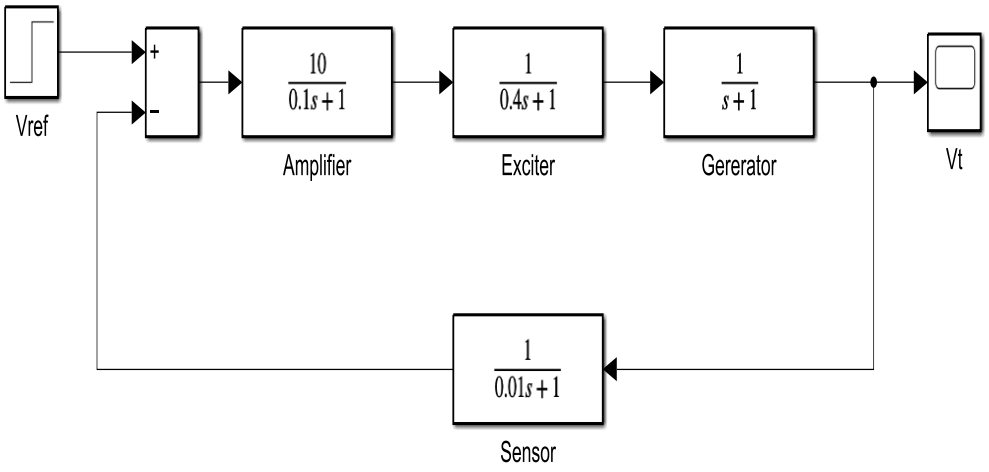

Figure 5 Block diagram of AVR system without controller.

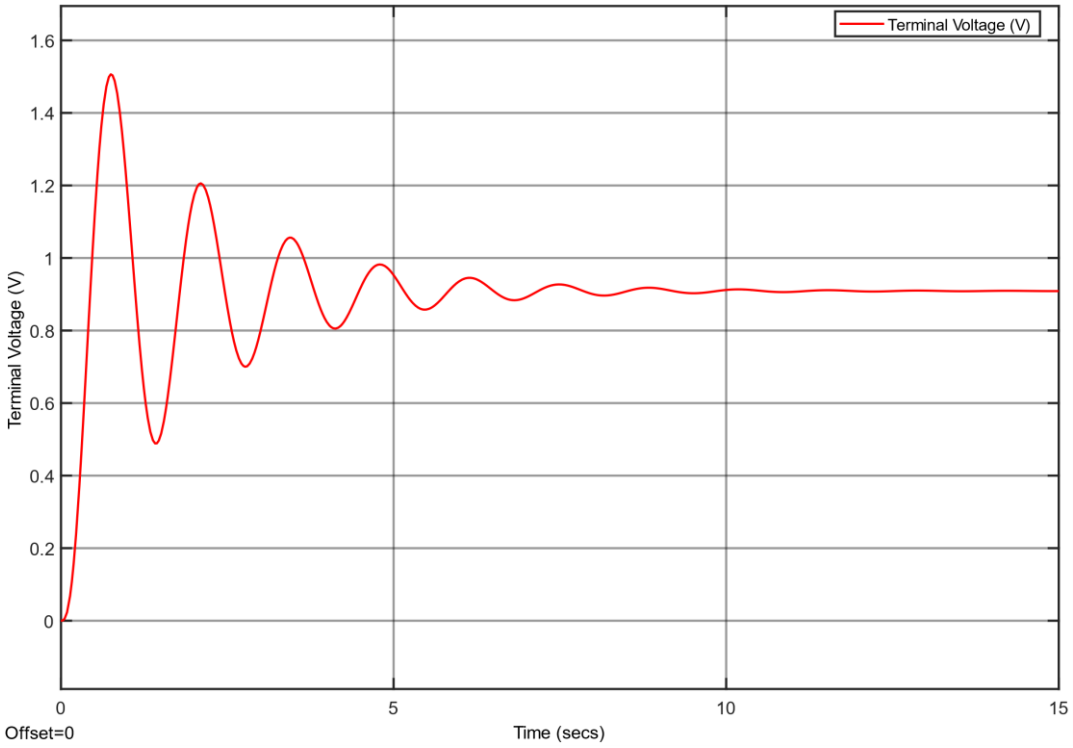

Figure 6 Response of AVR system without a controller.

PSO parameters are chosen as follows:

$$c_1 = 2, c_2 = 2, r_1, r_2 = rand(0,1), w = 0.7$$

$$No.Iteration = 100, No.Particles = 20$$

The searching space of each parameter of PID controller is chosen as follows:

$$K_P \in [0.1\ 10\ ], K_I \in [0.1\ 10], K_D \in [0.1\ 10]$$

After running PSO, obtained PID parameters is as follows:

$K_p = 2.4182$ $K_I = 0.2546$ $K_p = 0.5533$

Where best cost value obtained is displayed in Figure 7:

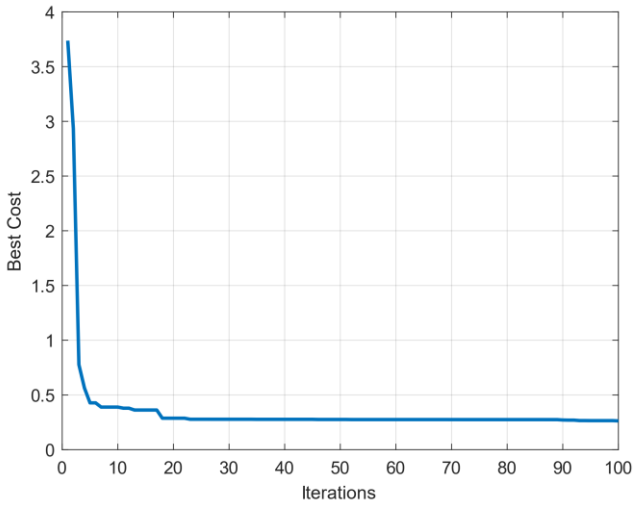

Figure 7 PSO Best Cost Versus iterations.

4.2.1 Simulink Model

Simulink model for implementing the obtained controller's parameters is illustrated in Figure .

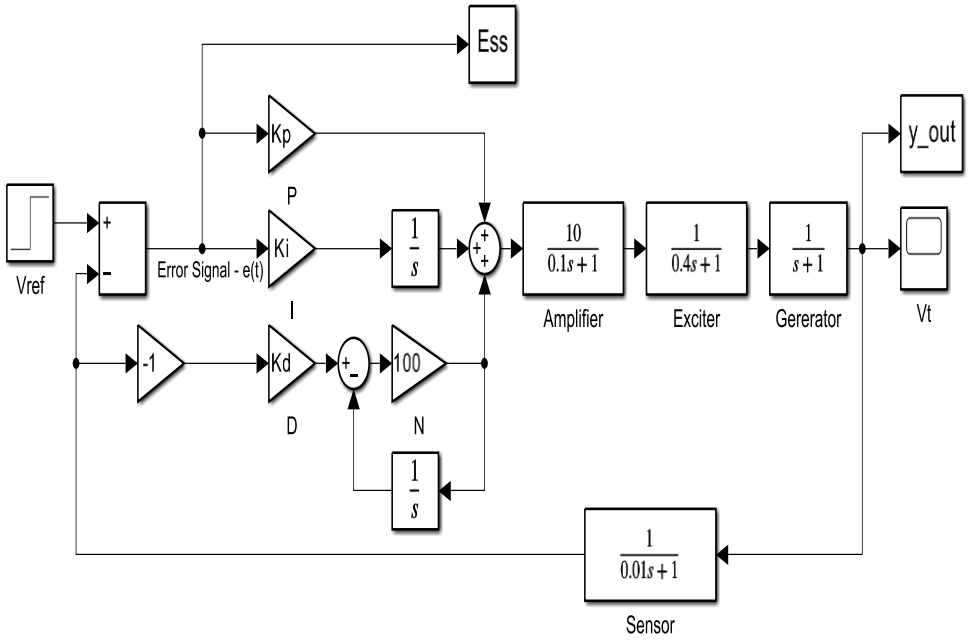

Figure 8 Simulation block diagram of AVR system with PID controller.

4.2.2 Terminal Voltage Response with PSO-based PID Controller

AVR time response using PSO-based PID is displayed in Figure as follows:

The results illustrated, depicts an improvement in system time response.

Before going throw implementation of supervised learning via NN, training data sets will be generated utilizing the previous controller designed with PSO.

4.3 NN based AVR Controller

4.3.1 Training Input/Output Data

As it is important to capture all information about the system, input data captured is error signal, integration of error and derivative of error. Where the output signal is the control action. Figure describes the model used to generate training data.

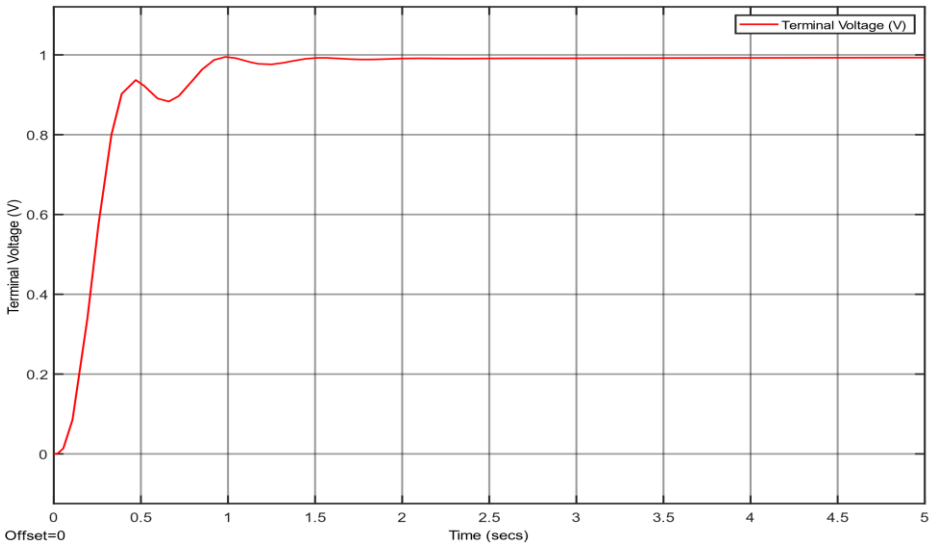

Figure 9 Response of AVR system with PSO-based PID controller.

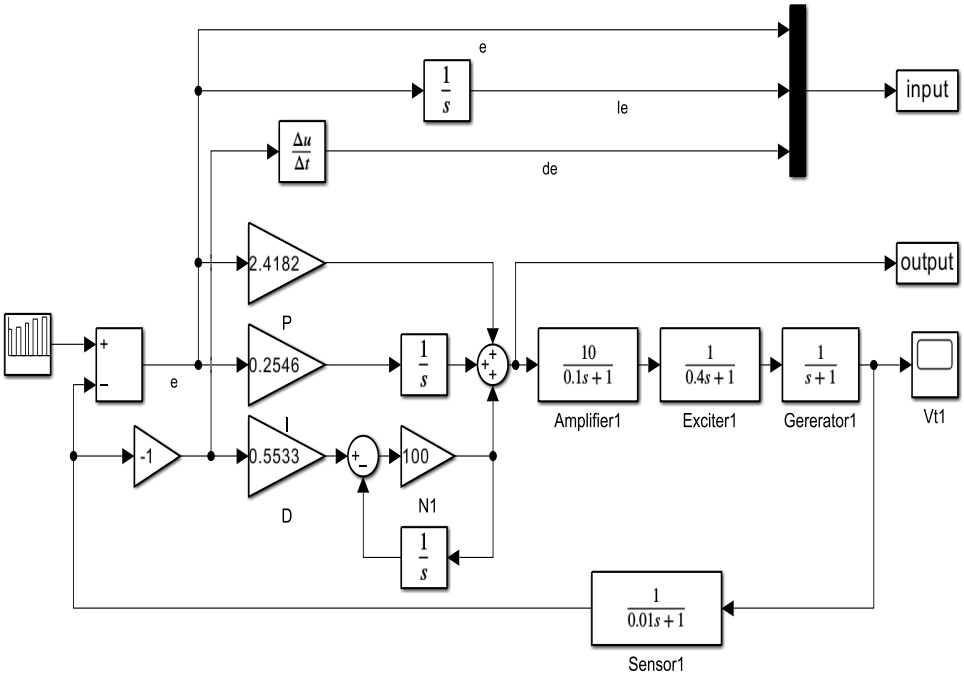

Figure 10 Generating Training Data Set.

4.3.2 NN Structure

After obtaining Input/Output data sets needed to train NN, MATLAB NN Toolbox is used to create, configure and train the NN. Where four hidden layers were used, each with five neurons and tanh activation function. Network structure is illustrated in

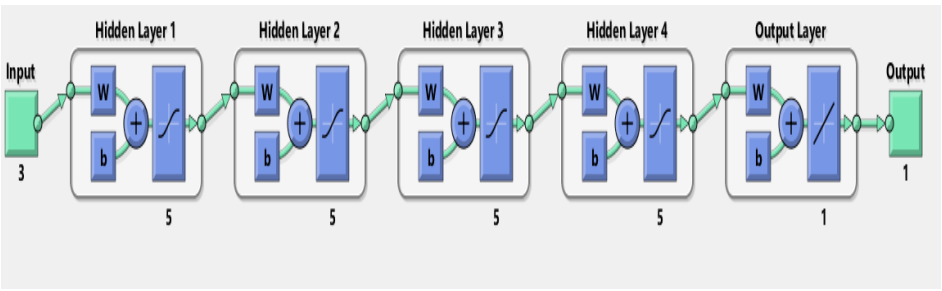

Figure 11 Neural Network Structure used for Supervised Learning Based AVR controller.

4.3.3 Simulink Model

After training is done, NN based controller can by implemented as presented in Figure :

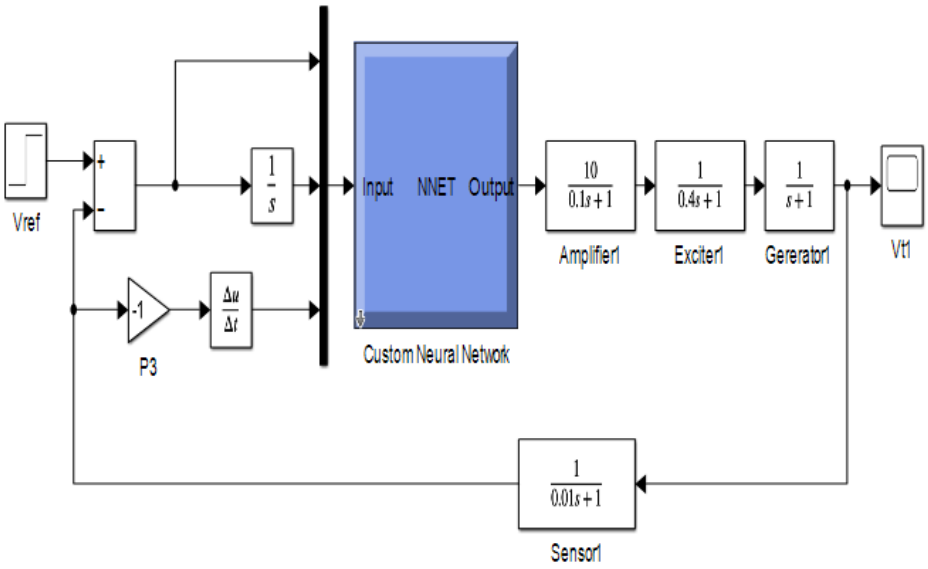

Figure 12 AVR System with Neural Network Based Controller.

4.3.4 Terminal Voltage Response With Controller

As represented in Figure 13, simulation result demonstrates how much NN is good in learning system dynamics. Where it has a response similar to PSO-based PID with a slightly higher overshoot.

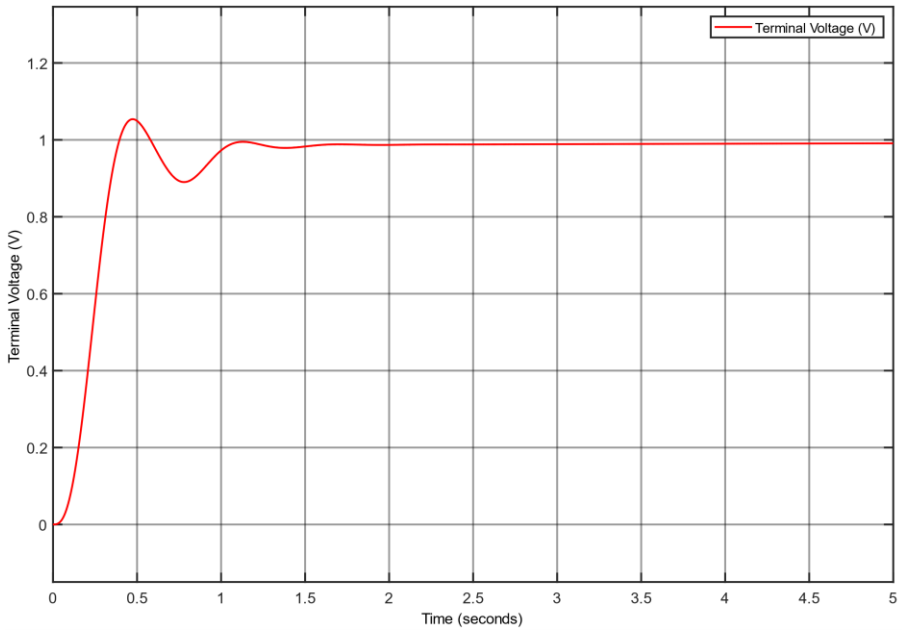

Figure 13 Response of AVR system with NN supervised learning.

4.4 DRL Based AVR Controller

4.4.1 Simulink Model

Since DRL incorporates different terms in the control system, such as: (reward, observation, agent, etc.), system model will a slightly different. Where Simulink diagram for DRL agent is represented in Figure as follows:

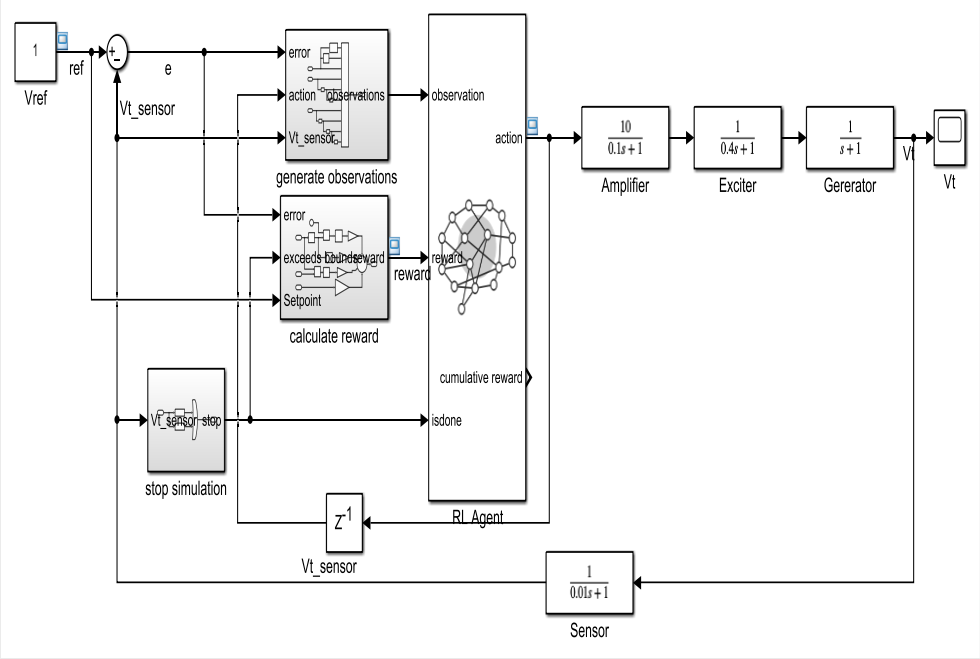

Figure 14 Closed-loop control structure of AVR System incorporating DRL agent.

4.4.2 Reward Function

It's important to note that reward function could be considered the main driver that pushes the system for a particular response, so it's vital to properly design it. The following figure represents the reward function in terms of Simulink blocks:

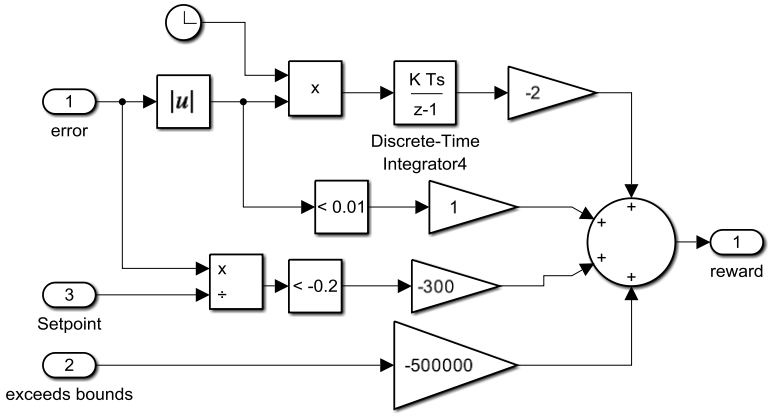

Figure 15 DRL Reward Function Formulation.

Reward function can be mathematically expressed as illustrated in Eq.8:

$$
\begin{aligned}
&r_t = R_1 + P_1 + P_2 + P_3 \\
&R_1 = 1, \qquad\qquad |e| < 0.01 \\
&P_1 = -300, \qquad\quad e < -0.2 \\
&P_2 = -2 \times \int t \times |e| \\
&P_3 = -500000, \quad V_{ref} \geq 5 \;||\; V_{ref} \leq -5
\end{aligned}
\qquad \text{(Eq.8)}
$$

4.4.3 Terminal Voltage Response with Controller

As shown in Figure , time response for AVR system incorporating DRL agent as a controller depicts a huge improvement in voltage regulation which seems to be the best among previously presented controllers.

The following section will delve into the comparative evaluation of the DRL-based controller in relation to the controllers introduced within this paper as well as those featured in the existing literature.

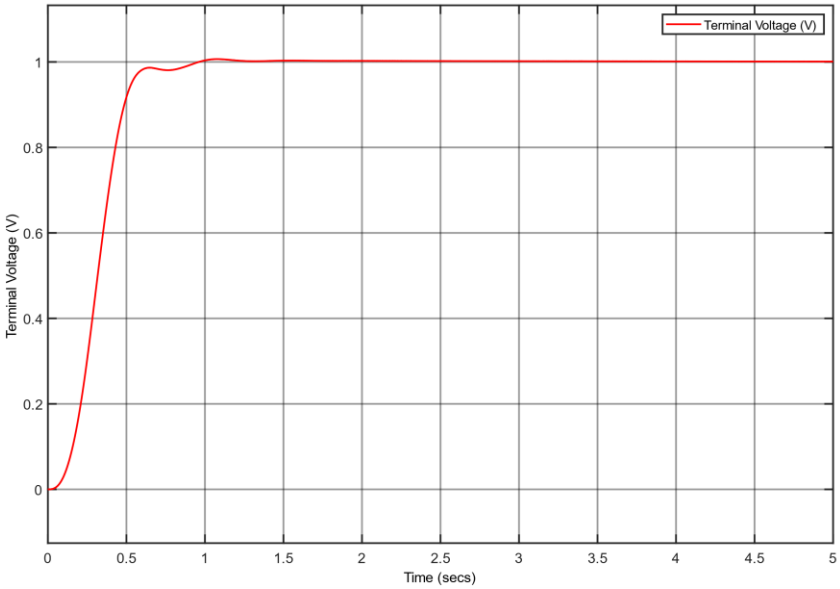

Figure 16 Response of AVR system with a DRL-based controller, TD3.

4.5 Discussion

The following figure present performance comparison between different control techniques presented in this paper:

The remarkable outcomes attained by the application of DRL in the regulation of the AVR system are presented in Figure . The system's response, as depicted in Figure , highlights the remarkable improvements realized by the DRL controller concerning time response characteristics, including settling time, overshoot, and steady-state error.

The resulted time response notably exceeds those obtained from alternative controllers, such as PSO-based PID and supervised learning using NN, thus establishing DRL as the preferred choice for achieving optimal and precise control of the AVR system.

The table presented below provides a summary of the performance metrics obtained from the simulation results of the AVR system. The table includes the values of steady-state error, settling time, and percentage of overshoot for each controller employed in this paper for the control of AVR system.

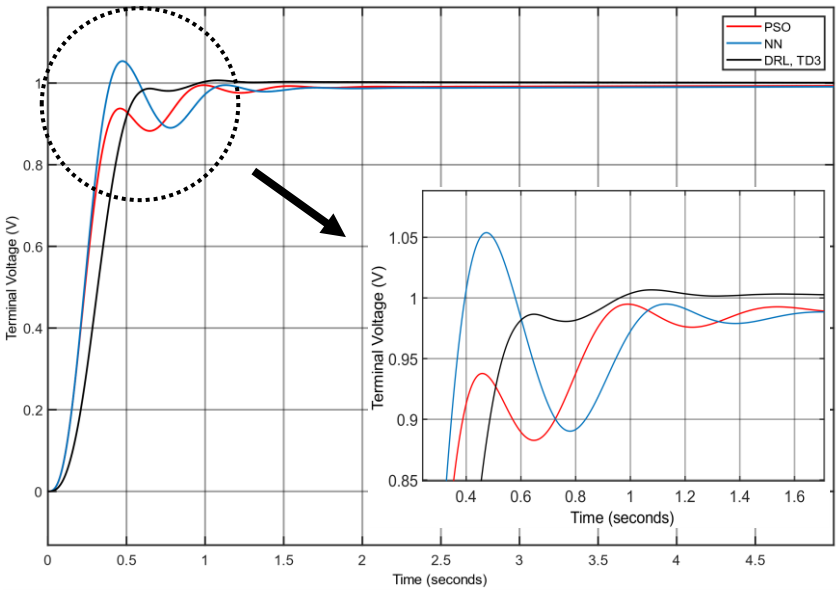

Figure 17 Response of AVR system with PSO-based PID, NN supervised learning and TD3 agent.

Table 1 Time Response Comparison for Different Proposed Controllers.

Controller	SSE	Ts(sec)	Tr(sec)	% O.S
PSO-based PID	0.004	1.3162	0.2735	0
NN supervised learning	0.0049	1.4381	0.2317	5.3791
DRL, TD3 agent	0.0002	0.5929	0.3285	0.6661

The results presented in this table are based on previously published papers that have investigated the performance of various control strategies in similar systems as presented in[22].

Table 2 Performance Evaluation Compared To Other Controllers In The Literature.

S. No.	Author/Year	Algorithm	TS(sec)	% O.S
1.	Proposed	DRL, TD3 agent	0.5929	0.6661
2.	Hassan, M. A. M., et al. (2013).	GA Tuned PID	1.3	0.3
3.	Hassan, M. A. M., et al. (2014).	Modified GA	1.22	7.82
4.	Hassan, M. A. M., et al. (2014).	PSO	0.926	5.95
5.	Gupta, T., et al. (2017)	FLC	4	0
6.	Yegireddy/2015	NSGA-II	4.402	23.6
7.	Aberbour/2015	PSO	4.966	20.9
8.	Kumar/2015	GSA	6.1	14.9
9.	Sahu/2012	PSA	5.697	13.9
10.	Panda/2012	MOL	5.328	15.8
11.	Rahimian/2011	PSO	5	16.8

The results of this study suggest that the DRL approach utilizing the TD3 agent outperformed the other controllers tested, including the PSO-based PID AVR controller and the neural network-based AVR controller. The DRL approach exhibited the best time response, indicating that it was the most effective at regulating the AVR system.

This finding is consistent with previous research in the field of process control, which has demonstrated the effectiveness of DRL approaches in a variety of applications. The ability of DRL to learn and adapt in real-time through trial and error makes it particularly well-suited for complex control problems, such as those encountered in power systems.

The superior performance of the DRL approach may be attributed to several factors. Firstly, the TD3 agent was able to learn optimal control policies through interactions with the AVR system, allowing it to adapt to changing conditions and disturbances. Secondly, the use of a neural network-based function approximator enabled the agent to generalize its learned policies to unseen states, improving its robustness and reducing the risk of overfitting. Finally, the incorporation of a replay buffer and target networks further enhanced the stability and convergence of the TD3 algorithm.

Overall, the results of this study demonstrate the potential of DRL approaches in process control applications, particularly in the area of power systems. Future research could investigate the application of DRL in other control problems, as well as the development of hybrid control strategies that combine the strengths of multiple control algorithms.

5. Conclusion

In conclusion, our study highlights the potential of DRL for process control applications, specifically for the AVR system. The results demonstrate that DRL is a powerful approach for optimizing control actions and achieving better system performance. With further research and development, DRL could become a widely adopted tool for process control in various industries. It could be suggested to investigate following future work as follows:

Using DRL to optimize parameters of Fractional PID; Using exponential reward function that has terms decay with time; Going in depth for the optimization of DRL hyper parameters

Conflict of Interest

The authors declare no conflict of interest.

References

1. Chatterjee, V. Mukherjee, and S. P. Ghoshal, Velocity relaxed and craziness-based swarm optimized intelligent PID and PSS controlled AVR system, International Journal of Electrical Power & Energy Systems, Vol. 31, No. 7–8, pp. 323–333, 2009.
2. M. R. ESTAKHROUIEH and A. L. I. A. GHARAVEISI, Optimal iterative learning control design for generator voltage regulation system, Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 21, No. 7, pp. 1909–1919, 2013.
3. G. Bal, O. Kaplan, and S. S. Yalcin, Artificial neural network based automatic voltage regulator for a stand-alone synchronous generator, in 2019 8th International Conference on Renewable Energy Research and Applications (ICRERA), IEEE, pp. 1032–1037, 2019.
4. Tripathi, R. L. Verma, and M. S. Alam, Design of Ziegler Nichols tuning controller for AVR System, International Journal of Research in Electronics & Communication Technology, Vol. 1, No. 2, pp. 154–158, 2013.
5. S. F. M. Khedr, M. E. Ammar, and M. A. M. Hassan, Multi objective genetic algorithm controller's tuning for non-linear automatic voltage regulator, in 2013 International Conference on Control, Decision and Information Technologies (CoDIT), IEEE, pp. 857–863, 2013.
6. Z.-L. Gaing, A particle swarm optimization approach for optimum design of PID controller in AVR system, IEEE Transactions on Energy Conversion, Vol. 19, No. 2, pp. 384–391, 2004.
7. P. Memon, A. S. Memon, A. A. Akhund, and R. H. Memon, Multilayer perceptrons neural network automatic voltage regulator with applicability and improvement in power system transient stability, International Journal of Emerging Trends in Electrical and Electronics (IJETEE ISSN: 2320-9569), IRET publication, Vol. 9, No. 1, pp. 30–38, 2013.
8. A. Bhutto, F. A. Chachar, M. Hussain, D. K. Bhutto, and S. E. Bakhsh, Implementation of probabilistic neural network (PNN) based automatic voltage regulator (AVR) for excitation control system in Matlab, in 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), IEEE, pp. 1–5, 2019.
9. R. Meléndez-Pérez, F. Ortiz-Rodríguez, and D. Ruiz-Vega, Design of an ANFIS Automatic Voltage Regulator of a Synchronous Generator, in 2020 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), IEEE, pp. 1–6, 2020.
10. M. L. Amer, H. H. Hassan, and H. M. Youssef, Modified evolutionary particle swarm optimization for AVR-PID tuning, in Communications and Information Technology, Systems and Signals 2008, pp. 164–173, 2008.
11. K. Yavarian, F. Hashemi, and A. Mohammadian, Design of intelligent PID controller for AVR system using an adaptive neuro fuzzy inference system, International Journal of Electrical and Computer Engineering (IJECE), Vol. 4, No. 5, pp. 703–718, 2014.
12. T. Gupta and D. K. Sambariya, Optimal design of fuzzy logic controller for automatic voltage regulator, in 2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC), IEEE, pp. 1–6, 2017.
13. M. I. Solihin, L. F. Tack, and M. L. Kean, Tuning of PID controller using particle swarm optimization (PSO), in Proceeding of the International Conference on Advanced Science, Engineering and Information Technology, pp. 458–461, 2011.
14. N. K. Bahgaat and M. A. Moustafa Hassan, Swarm intelligence PID controller tuning for AVR system, Advances in Chaos Theory and Intelligent Control, pp. 791–804, 2016.
15. P. Engelbrecht, Computational Intelligence: An Introduction. John Wiley & Sons, 2007.

16. T. J. Stray, Application of deep reinforcement learning for control problems, NTNU, 2019.
17. L. N. Magangane and K. A. Folly, Neural networks for designing an automatic voltage regulator of a synchronous generator, in 2013 Africon, IEEE, pp. 1–5, 2013.
18. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. MIT Press, 2018.
19. Welcome to Spinning Up in Deep RL!-Spinning Up documentation. https://spinningup.openai.com/en/latest/ (accessed Jan. 26, 2023).
20. T. P. Lillicrap et al., Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971, 2015.
21. S. Fujimoto, H. Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, in International Conference on Machine Learning, PMLR, pp. 1587–1596, 2018.
22. S. Priyambada, B. K. Sahu, and P. K. Mohanty, Fuzzy-PID controller optimized TLBO approach on automatic voltage regulator, in 2015 International Conference on Energy, Power and Environment: Towards Sustainable Growth (ICEPE), IEEE, pp. 1–6, 2015.