Data-Driven PID controller tuning using bayesian neural networks: A study on automatic voltage regulator control

Tu M. Pham, Son T. Nguyen*, Anh Hoang

Hanoi University of Science and Technology, Vietnam
**Corresponding author E-mail: son.mguyenthanh@hust.edu.vn*
DOI: https://doi.org/10.64032/mca.v30i4.443

Abstract

In this paper, a data-driven method to tune proportional-integral-derivative (PID) controllers for automatic voltage regulators (AVRs) using Bayesian neural networks (BNNs) is presented. The AVR is simulated as a dynamic control system consisting of an amplifier, exciter, generator and sensor. The data set is generated by systematically changing the PID gains and recording the corresponding closed loop performance indices, settling time and maximum overshoot. Then, a BNN is trained to learn the inverse mapping from the desired transient-performance specifications to the PID gains. The proposed method is validated using real-time simulation platform based on Arduino Uno. The results indicate that the BNN-based PID controller achieves shorter settling time and lower overshoot than the initial PID controller under the tested nominal AVR conditions. The method proposed offers a practical data-driven approach to PID tuning from desired transient-performance specifications. In this study, parameter uncertainty, external disturbance and measurement noise are not taken into consideration.

Keywords: *Bayesian neural networks, PID controller, Automatic voltage regulator, Arduino, Data-driven control.*

1. Introduction

The proportional-integral-derivative (PID) controller is the most widely used control algorithm in industrial automation, robotics, electrical drives, process control and power systems. PID controllers are widely used owing to their simple structure, easy implementation and effective control performance for a variety of dynamic systems. The controller continuously measures the error between the setpoint and the actual output of the system and generates a control signal based on proportional, integral and derivative action. The proportional term improves the response speed, the integral term eliminates the steady state error, and the derivative term improves the system stability and reduces the overshoot.

Many methods for PID tuning have been developed over the years. The conventional tuning methods are the Ziegler-Nichols method [1], Cohen-Coon method [2] and trial-and-error methods [3]. These techniques are simple and popular, but result in large overshoots, long settling time, and poor robustness for nonlinear or uncertain systems. Optimization based methods such as genetic algorithm (GA) [4, 5], particle swarm optimization (PSO) [6, 7], ant colony optimization (ACO) [8] and simulated annealing (SA) [9] have been proposed to improve the controller performance. Although these intelligent optimization techniques can achieve near-optimal PID gains by minimizing predefined objective functions, they usually require high computational effort, long optimization time, and careful parameter selection. Moreover, performance of these algorithms can also be highly sensitive to the chosen objective function and optimization parameters.

Recently, approaches based on artificial intelligence (AI) and machine learning (ML) are gaining popularity for PID tuning. Methods such as artificial neural networks (ANNs) [10], fuzzy logic systems (FLS) [11, 12] and adaptive neuro-fuzzy inference systems (ANFIS) [13, 14] offer adaptive and data driven tuning capability. However, these approaches usually require large training datasets, heavy training processes, expert knowledge to design the models, and may suffer from overfitting or degraded generalization performance under different operating conditions. Overall, the three main classes of PID tuning techniques are conventional tuning techniques, optimization-based tuning techniques, and artificial intelligence (AI)-based tuning techniques, as shown in Figure 1.

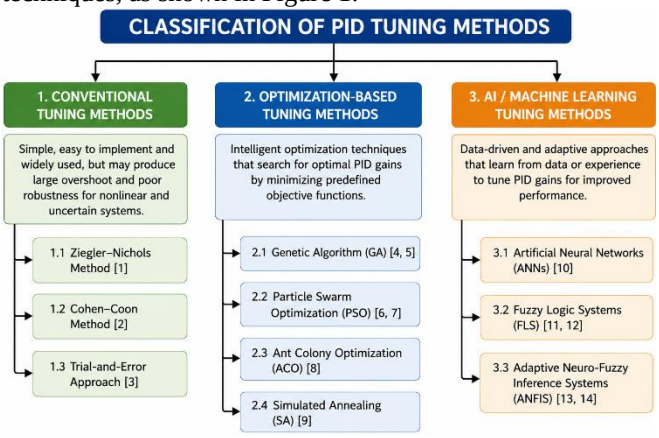


Figure 1: Classification of PID controller tuning methods.

An automatic voltage regulator (AVR) is a device which adjusts the excitation of a generator depending on the real-time conditions of the system to maintain a stable output voltage [15]. This is usually done through a closed-loop control scheme where a feedback loop continuously measures the output voltage and adjusts the excitation current to compensate for disturbances and load variations. PID controllers are the most widely used among the different control strategies due to their simplicity, robustness and effectiveness to minimize the voltage deviations and oscillations, which ensures an accurate and stable voltage regulation [16-18]. To enhance the performance of the AVR system in complex and fast changing operating environments,

Received: 27 May 2026; Revised: 29 September 2026; Accepted: 8 October 2026.

the recent AVR systems have started employing advanced control techniques such as fuzzy logic (FL) [19], sliding mode control (SMC) [20], ANNs [21] and model predictive control (MPC) [22].

BNNs are a type of ANNs which use Bayesian probability theory during the learning process [23, 24]. In conventional ANNs, weights are fixed values, whereas BNNs model these parameters as probability distributions. This probabilistic representation allows the network to model uncertainty and enhance robustness and generalization performance.

In this work, a BNN-based approach was proposed to find the optimal PID controller gains for an AVR system. The dynamic behavior of digital control implementation was shown by modelling the AVR system with PID controller in discrete-time domain. The complete training data set was achieved by simulating the closed loop AVR system under various operating conditions and PID controller settings. The dataset includes the main transient-response performance indices, such as settling time and maximum overshoot, used as input features to the BNN model. The outputs of BNN are the optimal PID controller gains that give the desired dynamic performance of the AVR system.

The remainder of this paper is organized as follows. Section 2 presents the discrete-time sub-models of the AVR system and the PID controller. The formulation of the discrete-time PID controller is mentioned in Section 3. BNNs for regression are introduced in Section 4. Next, Section 5 describes a detailed procedure for deploying a simple real-time simulation platform using an Arduino microcontroller to validate the proposed method. Results and discussions are also presented in Section 5. Finally, Section 6 concludes the paper.

2. The Automatic Voltage Regulator

An AVR is an electronic control device which is used to automatically maintain a constant level of output voltage in electrical systems. It is widely used in generators, alternators, power plants, industrial systems and electrical equipment to maintain the constant voltage of the supply when load conditions or input voltage changes. The AVR is very important to protect electrical devices from voltage fluctuations that may cause overheating, malfunction or damage.

2.1 Continuous-Time Sub-Models

The AVR operates on the principle of feedback control as shown in Figure 2. The regulator constantly monitors the output voltage and compares it to the desired reference voltage. If the output voltage decreases as load demand increases, the AVR increases the excitation current to the generator field winding. If the output voltage is too high, on the other hand, the AVR reduces the excitation current. In synchronous generators, the voltage generated is proportional to the field excitation current:

$$V \propto I_f \tag{1}$$

where:

- V : represents the generated voltage,
- I_f : presents the field excitation current.

The AVR system can be seen as a basic closed-loop structure with the following components:

- **The amplifier:** amplifies the power of the control signal as the controller output is usually too weak to drive the excitation system directly. The amplified signal is then fed to the exciter.
- **The exciter:** supplies the field excitation current to the generator rotor winding. By modifying the excitation current, the magnetic flux inside the synchronous generator changes directly affecting the generated terminal voltage.
- **The generator:** makes electrical output AC voltage.
- **The sensor:** constantly monitors and measures the terminal voltage and feeds it back to the summing junction.

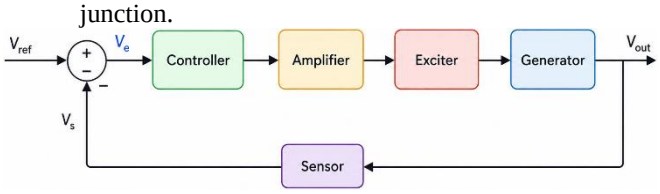


Figure 2: Basic closed-loop structure of the AVR.

The Laplace-domain transfer functions of the sub-models of the AVR system are presented in Table 1. Each sub-model is represented as a first-order transfer function with a gain parameter and a time constant. These first-order models represent the dynamic behavior of the main AVR components: amplifier, exciter, generator and sensor. The gain is the size of the system response, and the time constant is the speed of the response to changes in the input. This type of modeling approach provides a simplification of the analysis and design of the AVR system while retaining the essential dynamic characteristics.

Table 1: Transfer functions of sub-models in an AVR system.

Component	Transfer Function	Parameter Range
Amplifier	$G_a(s)=\frac{k_a}{\tau_a s+1}$	$10<k_a<400$ $0.02<\tau_a<0.1(s)$
Excitier	$G_e(s)=\frac{k_e}{\tau_e s+1}$	$1<k_e<400$ $0.4<\tau_e<1(s)$
Generator	$G_g(s)=\frac{k_g}{\tau_g s+1}$	$0.7<k_g<1$ $1<\tau_g<2(s)$
Sensor	$G_s(s)=\frac{k_s}{\tau_s s+1}$	$0.001<\tau_s<0.06(s)$

2.2 Discrete-Time Sub-Models

The discrete-time form of the amplifier is given by:

$$V_a[k+1]=\frac{T_a}{T_a+T}V_a[k]+\frac{K_a T}{T_a+T}u[k] \tag{2}$$

where:

- $V_a[k]$: the output voltage of the amplifier at the k -th sampling instant,
- $V_a[k+1]$: the output voltage of the amplifier at the $k+1$-th sampling instant,

- $u[k]$: the input voltage of the amplifier at the k-th sampling instant,
- T_a: the time constant of the amplifier,
- K_a: the gain of the amplifier,
- T: the sampling interval.

The discrete-time form of the exciter is given by:

$$V_e[k+1] = \frac{T_e}{T_e+T} V_e[k] + \frac{K_e T}{T_e+T} V_a[k] \quad (3)$$

where:

- $V_e[k]$: the output voltage of the exciter at the k-th sampling instant,
- $V_e[k+1]$: the output voltage of the exciter at the $k+1$-th sampling instant,
- T_e: the time constant of the exciter,
- K_e: the gain of the exciter.

The discrete-time form of the generator is given by:

$$V_g[k+1] = \frac{T_g}{T_g+T} V_g[k] + \frac{K_g T}{T_g+T} V_e[k] \quad (4)$$

where:

- $V_g[k]$: the output voltage of the generator at the k-th sampling instant,
- $V_g[k+1]$: the output voltage of the generator at the $k+1$-th sampling instant,
- T_g: the time constant of the generator,
- K_g: the gain of the generator.

The discrete-time form of the sensor is given by:

$$V_s[k+1] = \frac{T_s}{T_s+T} V_s[k] + \frac{K_s T}{T_s+T} V_g[k] \quad (5)$$

where:

- $V_s[k]$: the output voltage of the sensor at the k-th sampling instant,
- $V_s[k+1]$: the output voltage of the sensor at the $k+1$-th sampling instant,
- T_s: the time constant of the sensor,
- K_s: the gain of the sensor.

3. The Discrete-Time PID Controller

The time-domain model of the PID controller is given by:

$$u(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt} \quad (6)$$

where:

- $u(t)$: the output voltage of the controller,
- $e(t) = r(t) - y(t)$: the tracking error, in which $r(t)$ is the reference and $y(t)$ is the plant output,
- K_P: the proportional gain,
- K_I: the integral gain,
- K_D: the derivative gain.

The discrete-time form of the controller is given by:

$$u[k] = K_P e[k] + K_I \sum_{k=1}^{N} e[k]T + K_D \frac{e[k]-e[k-1]}{T} \quad (7)$$

where:

- $u[k]$: the output voltage of the controller at the k-th sampling instant,
- $e[k]$: the tracking error at the k-th sampling instant,
- $e[k-1]$: the tracking error at the $k-1$-th sampling instant.

4. Bayesian Neural Networks for Regression

This section describes the formulation of BNNs for regression problems. In contrast to conventional neural networks, where the model parameters are represented by fixed deterministic values, BNNs treat the network weights and biases as probabilistic variables described by probability distributions. This probabilistic framework enables the model to capture uncertainty in the learning process and improve generalization performance, especially when the training data are limited, noisy, or affected by uncertainties.

4.1 Feedforward Neural Networks for Regression

In FFNNs for regression, information flows in one direction-from the input layer through a single hidden layer and finally to the output layer-without any feedback loops. The network output of FFNNs is given as follows:

$$z_k = \sum_{j=1}^{M} w_{kj}^{(2)} \left(\tanh\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + b_j^{(1)} \right) \right) + b_k^{(2)} \quad (8)$$

where:

- x_i: the i-th input value to the network,
- d: the number of network inputs,
- $w_{ji}^{(1)}$: the weight on the connection from the i-th input to the j-th hidden node,
- $b_j^{(1)}$: the bias of the j-th hidden node,
- $w_{kj}^{(2)}$: the weight on the connection from the j-th hidden node to the k-th output node,
- $b_k^{(2)}$: the bias of the k-th output node,
- z_k: the k-th output node value,
- M: the number of hidden nodes.

4.2 Bayes' Theorem

The use of Bayes' theorem can result in the posterior distribution of weights and biases in a neural network given a dataset as follows:

$$p(w|D) = \frac{p(D|w)\,p(w)}{p(D)} \quad (9)$$

where w is the vector of weights and biases. $p(w|D)$ is the posterior distribution of weights and biases given a dataset

D. $p(w)$ is the prior distribution of weights and biases. $p(D|w)$ is the dataset likelihood. Finally, $p(D)$ is a normalization factor to ensure the posterior distribution integrates to 1. $p(D)$ is well-known as the evidence, which is given by an integral over the network space as follows:

$$p(D)=\int p(D|w)p(w)dw \tag{10}$$

4.3 The Evidence Procedure

The evidence procedure is an iterative algorithm for determining the optimal vector of weights and biases, and hyperparameters. In the evidence procedure, the following cost function needs to be minimized:

$$S(w)=\beta E_D(w)+\alpha E_W(w) \tag{11}$$

where:

- $S(w)$: the cost function,
- $E_D(w)$: the data error function depending on the weights and biases,
- $E_W(w)$: the weight function depending on the weights and biases,
- α and β: the hyperparameters.

In (11), the data error function $E_D(w)$ is given by:

$$E_D(w)=\frac{1}{2}\sum_{n=1}^{N}\left\{z(x^n;w)-t^n\right\}^2 \tag{12}$$

where:

- $z(x^n;w)$: the network output corresponding to the n-th training patterns,
- t^n: the desired network output (target) corresponding to the n-th training patterns,
- N: the number of training patterns.

The weight function in (11) has the following form:

$$E_W(w)=\frac{1}{2}\|w\|^2=\frac{1}{2}\sum_{i=1}^{W}w_i^2 \tag{13}$$

where:

- w: the vector of weights and biases,
- w_i: the i-th weight or bias.
- W: the number of weights or biases.

The value of α can be estimated by using the following equation [25, 26]:

$$\alpha=\frac{\gamma}{2E_W(w_{MP})} \tag{14}$$

where:

- w_{MP}: the most probable vector of weights and biases,
- γ: the number of well-determined parameters.

The value of γ is determined as follows:

$$\gamma=\sum_{i=1}^{W}\frac{\lambda_i}{\lambda_i+\alpha} \tag{15}$$

where $\lambda_i\,(i=1,...,W)$ are the eigenvalues of the Hessian matrix of the data error, $E_D(w)$.

The values of β can be estimated by using the following equation:

$$\beta=\frac{N-\gamma}{2E_D(w_{MP})} \tag{16}$$

In the evidence procedure, there is also a need of maximizing $p(D|\alpha,\beta)$, which is known as the evidence. This term can be found by integrating data likelihood over all possible weights and biases.

$$p(D|\alpha,\beta)=\int p(D|w,\alpha,\beta)p(w|\alpha,\beta)dw \tag{17}$$

Instead of directly evaluating the evidence, it is more convenient to compute the log evidence by using the equation below [25, 26]:

$$\begin{aligned}\ln p(D|\alpha,\beta)&=-\alpha E_W(w_{MP})-\beta E_D(w_{MP})\\&-\frac{1}{2}\ln(\det A)+\frac{W}{2}\ln\alpha+\frac{N}{2}\ln\beta-\frac{N}{2}\ln(2\pi)\end{aligned} \tag{18}$$

where A is the Hessian matrix of the cost function evaluated at w_{MP} as follows:

$$A=\nabla\nabla S(w_{MP})=\beta\nabla\nabla E_D(w_{MP})+\alpha I \tag{19}$$

where:

- $\nabla\nabla E_D(w_{MP})$: the Hessian matrix of $E_D(w)$ at w_{MP},
- I: the identity matrix.

The log evidence $\ln p(D|\alpha,\beta)$, as determined in (19), can be used to identify the optimal number of hidden nodes in the BNN. In particular, the optimal number of hidden nodes corresponds to the configuration that yields the highest log evidence.

4.4 Update of the Weights and Biases

The vector of weights and biases can be updated via an iterative process as follows:

$$w[k+1]=w[k]+\eta[k]d[k] \tag{20}$$

where:

- $w[k]$: the vector of weights and biases at the k-th iteration,
- $w[k+1]$: the vector of weights and biases at the $k+1$-th iteration,
- $d[k]$: the search direction at the $k+1$-th iteration,
- $\eta[k]$: the learning rate at the $k+1$-th iteration.

The vector of weights and biases is updated to minimize the difference between the predicted output and the real output. Training algorithms, usually backpropagation combined with an optimization algorithm like the scaled conjugate gradient (SCG) method or quasi-Newton method [25, 26]. The BNN training procedure includes the following steps:

- **Step 1:** Initialize the values for α and β. The network weights and biases are initialized from the prior distribution defined for α.

- **Step 2:** Update the vector of weights and biases to minimize the cost function, $S(w)$, using SCG method.
- **Step 3:** When the cost function has reached a local minimum, the values of α and β can be re-estimated as follows:

$$\alpha_{new} = \frac{\gamma_{old}}{2E_W} \tag{21}$$

$$\beta_{new} = \frac{N - \gamma_{old}}{2E_W} \tag{22}$$

- **Step 4:** Repeat Steps 2 and 3 until a convergence has reached.

The principle of computing PID controller gains using a trained BNN is shown in Figure 3. The inputs to the network are the normalized minimum values of the settling time vector and the overshoot vector and these represent the desired dynamic performance of the AVR system. The trained BNN predicts the optimal PID controller gains based on these performance indices. This approach gives an efficient nonlinear mapping between the system performance specifications and the corresponding controller parameters.

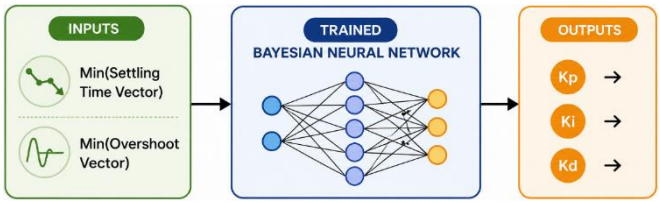


Figure 3: Principle of determining the PID controller gains using a trained BNN.

5. Real-Time Simulation

To verify the effectiveness of the proposed method, a simple real-time simulation system based on Arduino Uno was developed as shown in Figure 4. Arduino Uno is an open-source microcontroller development board based on ATmega328P microcontroller. It is widely used in embedded control, data acquisition and real time monitoring applications because of its low cost, flexibility and ease of programming. In this work, the real-time simulation of the AVR system and PID controller algorithm was performed using the Arduino Uno.

A data acquisition (DAQ) software was developed using the Python programming language for the real-time monitoring and visualization of the terminal voltage response of the AVR system. The graphical user interface (GUI) shown in Figure 5 can provide continuous communication between the computer and the Arduino Uno through serial communication, with the measured signals displayed and recorded during operation. In addition, the software has the feature to store the transient speed response for off-line analysis and performance evaluation using MATLAB. The experimental test bed provides a simple and effective platform to validate the proposed BNN-based PID tuning method under real-time operating conditions.

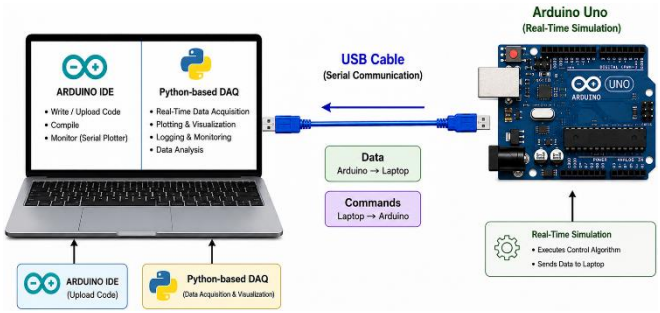


Figure 4: Simple real-time simulation system using Arduino Uno.

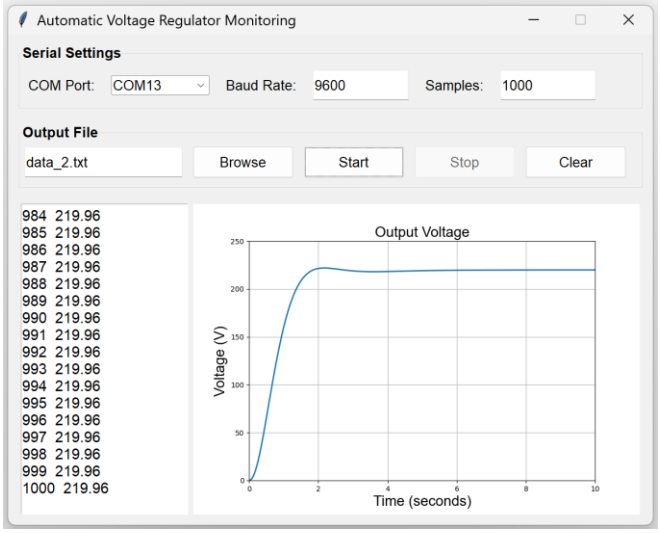


Figure 5: GUI of the DAQ software.

The training process of the BNN requires the formation of a representative dataset containing the relationships between the PID controller gains and the corresponding dynamic performance indices of the AVR system. Table 2 presents the transfer functions of the sub-models in the AVR system used for real-time simulation, including the amplifier, exciter, generator, and sensor models.

Table 2: Transfer functions of the sub-models in the AVR system for real-time simulation.

Component	Transfer Function	Parameter Range
Amplifier	$G_a(s) = \frac{k_a}{\tau_a s + 1}$	$k_a = 10$ $\tau_a = 0.1(s)$
Excitier	$G_e(s) = \frac{k_e}{\tau_e s + 1}$	$k_e = 1$ $\tau_e = 0.4(s)$
Generator	$G_g(s) = \frac{k_g}{\tau_g s + 1}$	$k_g = 1$ $\tau_g = 2(s)$
Sensor	$G_s(s) = \frac{k_s}{\tau_s s + 1}$	$k_s = 1$ $\tau_s = 0.01(s)$

Before generating the training dataset, the initial values of the PID controller gains must be specified in advance following the flowchart in Figure 6. In this step, the initial PID parameters (K_{P0} , K_{I0} and K_{D0}) are determined using the PID auto-tuning function in MATLAB through the command **pidtune(G,'pid')**, where G represents the overall transfer function of the AVR system. The obtained PID gains serve as

the reference values for generating a range of controller parameters used in the dataset creation process.

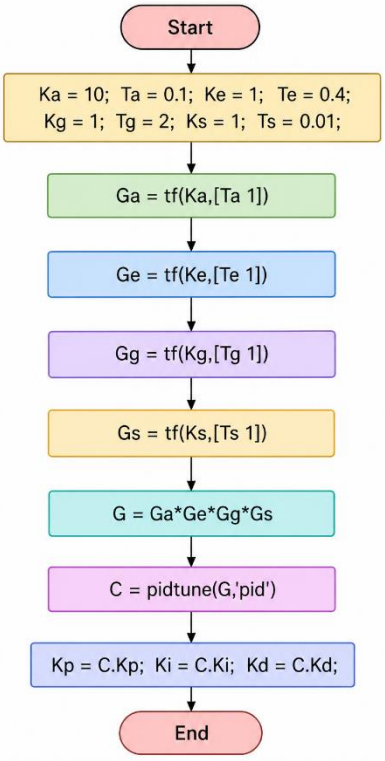


Figure 6: Flowchart for computing initial values of the PID controller gains.

The procedure for generating the training dataset is illustrated in the flowchart shown in Figure 7. Based on the initial PID gains (K_{P0} , K_{I0} and K_{D0}), multiple combinations of K_P , K_I and K_D are generated systematically. For each combination, the AVR system response is evaluated to obtain important dynamic performance indices such as rise time, settling time, and overshoot. These data are then stored and used to train the proposed BNN model for determining the optimal PID controller gains. The initial PID gains can be obtained by using the MATLAB pidtune(G,'pid') function. Then, the parameter ranges for generating the training dataset were defined as:

$$K_P = 0.5K_{P0} - 1.5K_{P0}$$

$$K_I = 0.5K_{I0} - 1.5K_{I0}$$

$$K_D = 0 - 0.05$$

Each parameter range was uniformly divided into 10 grid points. Since each vector of K_P , K_I , and K_D contains 10 grid resolutions, the total number of parameter combinations is 10×10×10=1000. Therefore, a total of 1000 operating conditions of the AVR system are generated for the dataset. Each operating condition corresponds to a unique combination of PID controller gains and produces a different dynamic response of the AVR system. For every combination, important performance indices such as rise time, settling time, and overshoot are evaluated and recorded. This large dataset provides sufficient input-output mapping information for training the BNN to accurately estimate the optimal PID controller gains.

The flowchart in Figure 8 illustrates the whole procedure for training a BNN for the determination of the best PID controller gains for the AVR system. The process begins with loading the training dataset from "training_data.mat" that contains the performance indices and PID controller gains from different operating conditions of the AVR system. Also, the dataset is loaded, and the input variables are normalized to achieve better training performance of the BNN. The settling time vector and the overshoot vector are normalized by dividing each element with their maximum value. These normalized quantities are the matrix input. Similarly, the PID controller gain vectors are normalized to construct the target matrix. Then, the BNN structure and parameters are defined.

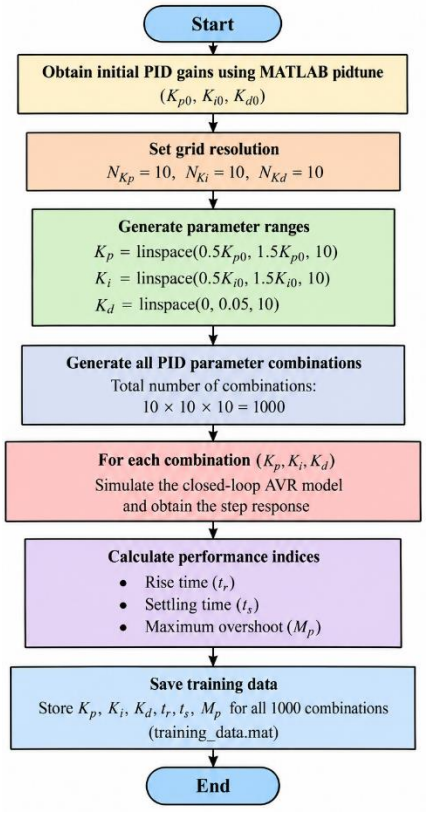


Figure 7: Flowchart for generating the training dataset for BNN.

The network consists of two input neurons corresponding to the normalized settling time and normalized overshoot, a hidden layer with a variable number of neurons ranging from 1 to 8, and three output neurons corresponding to the PID controller gains (K_P , K_I and K_D). Also, initial values for the hyperparameters (α and β) are defined to enable Bayesian learning. Next, the BNN model is designed using FFNN architecture. Initialization of training options: outer loops, inner loops, training cycles. The network is trained with the scaled conjugate gradient (SCG) optimization algorithm. At each iteration, the evidence framework is used to update the hyperparameters and improve the generalization ability of the network.

The variation of the log evidence as a function of the number of hidden nodes in the BNN is shown in Figure 9. The maximum value of the log evidence is obtained for the network with two hidden nodes. This indicates that the BNN with two hidden nodes gives the best trade-off between the model complexity and the model generalization ability. A

higher number of hidden nodes may cause high risk of over-fitting. Therefore, the two-hidden-node BNN is chosen as the optimal network structure as it yields the highest evidence and best generalization performance for the prediction of the optimal PID controller gains of the AVR system.

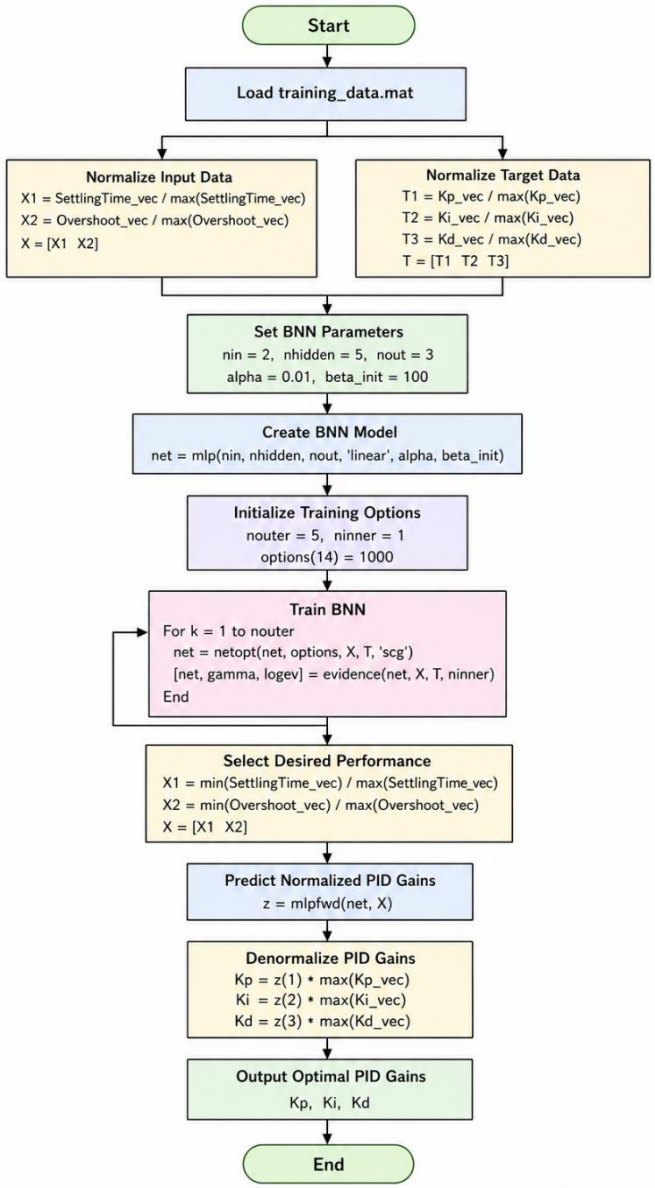


Figure 8: Procedure for training the BNN to find the optimal values of the PID controller gains.

After the training process, the desired system performance is specified using the minimum normalized settling time and minimum normalized overshoot obtained from the generated dataset. These two values are provided as the input to the trained BNN. The BNN then predicts the corresponding normalized PID gains, which are subsequently denormalized to obtain the final PID controller parameters.

The trained BNN predicts the normalized PID controller gains via forward propagation. Finally, the predicted normalized gains are denormalized to the original scales. The resulting values of K_P, K_I and K_D represent the optimal PID controller gains obtained using the trained BNN. It should be noted that the inverse mapping from the two performance indices (t_s and M_p) to the three PID controller gains (K_P, K_I and K_D) is generally non-unique. Therefore, the trained BNN does not provide a mathematically unique solution. Instead, it learns a representative mapping from the performance specifications to a feasible PID-gain vector based on the distribution of the training data. The Bayesian formulation accounts for uncertainty in the learned model through probabilistic network parameters; however, the present implementation uses the BNN prediction to obtain one representative PID-gain set rather than explicitly estimating all possible PID solutions corresponding to the same performance indices.

The role of the BNN is different from that of direct optimization or exhaustive search. The generated dataset contains 1000 predefined combinations of (K_P, K_I and K_D), for which the corresponding transient-response indices are explicitly calculated. Direct selection from these samples can therefore identify the best controller within the predefined candidate set. In contrast, the BNN is trained to learn an inverse mapping from the desired transient-performance specifications to the PID parameters. Consequently, the BNN is intended to provide a rapid prediction of PID gains for a specified performance requirement without repeating the complete evaluation of all candidate PID combinations. The BNN prediction is subsequently validated by applying the predicted gains to the AVR model and evaluating the resulting transient response.

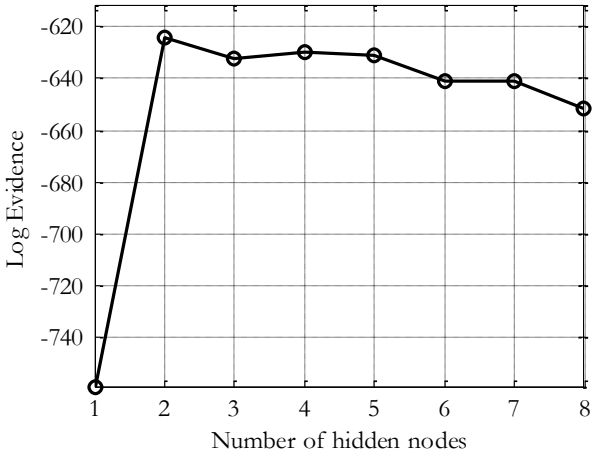


Figure 9: The log evidence versus the number of hidden nodes.

Table 3 shows how the hyperparameters vary throughout the BNN training process over five re-estimation periods. The value of the hyperparameter α is relatively small during the re-estimation periods, which indicates that the weight decay term has little effect on the training process. i.e. the network weights are free to adapt to the training data more flexibly. On the other hand, the hyperparameter β is rather large for all the re-estimation periods, suggesting a high confidence in the training data and a small, estimated noise variance. The large value of β implies that the BNN focuses more on the reduction of data error, thus improves the regression accuracy and model fitting performance.

The PID controller gains before and after the optimal tuning based on the trained BNN are shown in Table 4. Figures 10 and 11 show the terminal voltage responses of the AVR system for the reference voltages of 220V and 110V

respectively. The curves show the performance of the PID controller tuned to optimum at working points. From the responses, it is observed that the BNN-based PID (BNN-PID) controller can give better performance indices such as less settling time and smaller overshoot in comparison with the initial settings of the controller.

Table 3: Changes of the hyperparameters according to three re-estimation periods.

Re-estimation Period	α	β
1	0.25222	40.85538
2	0.40534	40.83342
3	0.43186	40.82383
4	0.43515	40.82237
5	0.43554	40.82220

Table 4: The PID controller gains before and after optimal tuning using the trained BNN.

PID Controller	K_P	K_I	K_D
Before optimal tuning	0.177	0.108	0.000
After optimal tuning	0.115	0.072	0.026

The quantitative dynamic performances of the AVR system for reference voltages of 220 V and 110 V are given in Tables 5 and 6, respectively. The results show the effectiveness of the proposed BNN-PID controller in two operating conditions. The BNN-PID controller has a shorter settling time and a great reduction of overshoot compared with the initial PID controller settings. The improvements show that the proposed tuning approach improves the transient response and stability of the AVR system while keeping the accurate voltage regulation performance.

Table 5: Dynamic performance of the AVR corresponding to a 220 V reference voltage.

	Settling Time (s)	Overshoot (%)
Before optimal tuning	3.5966	14.3728
After optimal tuning	2.4340	0

Table 6: Dynamic performance of the AVR corresponding to a 110 V reference voltage.

	Settling Time (s)	Overshoot (%)
Before optimal tuning	3.5964	14.3728
After optimal tuning	3.0218	0

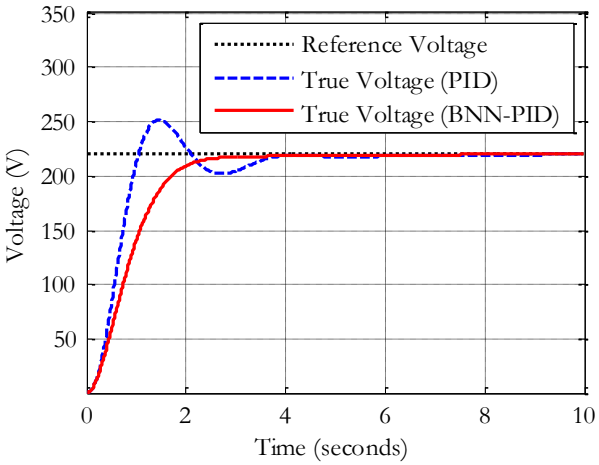


Figure 10: Terminal voltage responses for a 220 V reference voltage.

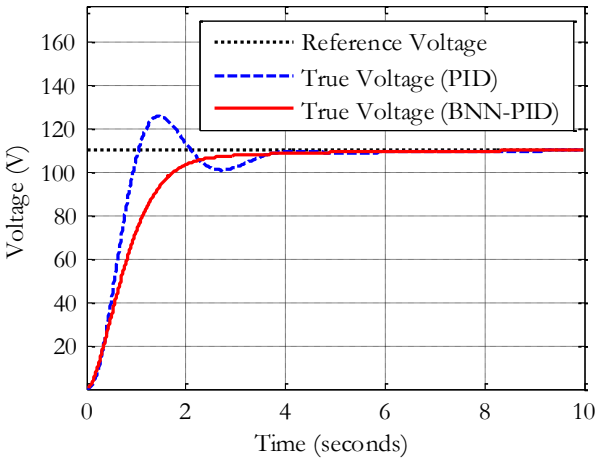


Figure 11: Terminal voltage responses for a 110 V reference voltage.

6. Conclusion

The paper proposes a data-driven approach for tuning PID controllers for AVR systems based on BNNs. The AVR was represented as a dynamic system made up of an amplifier, exciter, generator, and sensor. We created a dataset by systematically changing the PID gains and evaluating the corresponding settling time and maximum overshoot. Then the BNN was trained to learn the inverse mapping from the desired transient-performance specifications to the PID gains. The proposed method was tested using an Arduino Uno based real-time simulation. The results show that the BNN-based PID controller achieves less overshoot and a faster settling time than the original PID controller for the nominal AVR conditions that were tested. But the parameter uncertainty, external disturbance and measurement noise were not explicitly considered. Hence, robustness and generalization in these conditions cannot be inferred from the present results. In future work the proposed method will be studied under parameter variations, disturbances and measurement noise.

Acknowledgement

This research is funded by Hanoi University of Science and Technology (HUST) under project number T2025-PC-084.

References

[1] A. S. Bazanella, L. F. A. Pereira and A. Parraga, “A New Method for PID Tuning Including Plants Without Ultimate Frequency”, in *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 637-644, March 2017, doi: 10.1109/TCST.2016.2557723.

[2] J. Hao, G. Zhang, D. Zhu and H. Ye, “High Efficiency Data Driven Control Based on Dynamic Linearization and PIDNN With Cohen-Coon for Discrete Nonlinear Fast Time-Varying Systems”, in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 2, pp. 782-786, Feb. 2024, doi: 10.1109/TCSII.2023.3311805.

[3] H. H. A., M. Y. Mashor and M. C. Mahdi, “Performance of Manual and Auto-Tuning PID Controller for Unstable Plant - Nano Satellite

Attitude Control System", *2018 6th International Conference on Cyber and IT Service Management (CITSM)*, Parapat, Indonesia, 2018, pp. 1-5, doi: 10.1109/CITSM.2018.8674375.

[4] J. Li and W. Li, "On-Line PID Parameters Optimization Control for Wind Power Generation System Based on Genetic Algorithm," in *IEEE Access*, vol. 8, pp. 137094-137100, 2020, doi: 10.1109/ACCESS.2020.3009240.

[5] H. Wang, S. Xu and H. Hu, "PID Controller for PMSM Speed Control Based on Improved Quantum Genetic Algorithm Optimization," in *IEEE Access*, vol. 11, pp. 61091-61102, 2023, doi: 10.1109/ACCESS.2023.3284971.

[6] A. A. Al Barazi and E. Aykut, "Automated Tuning of PID Controllers Using a Generalized Discrete PID Structure With Particle Swarm Optimization," in *IEEE Access*, vol. 13, pp. 185994-186013, 2025, doi: 10.1109/ACCESS.2025.3625055.

[7] W. Assawinchaichote, C. Angeli and J. Pongfai, "Proportional-Integral-Derivative Parametric Autotuning by Novel Stable Particle Swarm Optimization (NSPSO)", in *IEEE Access*, vol. 10, pp. 40818-40828, 2022, doi: 10.1109/ACCESS.2022.3167026.

[8] Z. Liang, L. Fu, X. Li, Z. Feng, J. W. Sleigh and H. K. Lam, "Ant Colony Optimization PID Control of Hypnosis With Propofol Using Renyi Permutation Entropy as Controlled Variable", in *IEEE Access*, vol. 7, pp. 97689-97703, 2019, doi: 10.1109/ACCESS.2019.2927321.

[9] M. -H. Hung, L. -S. Shu, S. -J. Ho, S. -F. Hwang and S. -Y. Ho, "A Novel Intelligent Multiobjective Simulated Annealing Algorithm for Designing Robust PID Controllers", in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 38, no. 2, pp. 319-330, March 2008, doi: 10.1109/TSMCA.2007.914793.

[10] L. R. R. Dos Santos, F. R. Durand, A. Goedtel and T. Abrao, "Auto-tuning PID distributed power control for next-generation passive optical networks", in *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D110-D125, Oct. 2018, doi: 10.1364/JOCN.10.00D110.

[11] Han-Xiong Li, Lei Zhang, Kai-Yuan Cai and Guanrong Chen, "An improved robust fuzzy-PID controller with optimal fuzzy reasoning", in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 6, pp. 1283-1294, Dec. 2005, doi: 10.1109/TSMCB.2005.851538.

[12] E. Harinath and G. K. I. Mann, "Design and Tuning of Standard Additive Model Based Fuzzy PID Controllers for Multivariable Process Systems", in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 3, pp. 667-674, June 2008, doi: 10.1109/TSMCB.2008.919232.

[13] K. -H. Choi and H. -J. Chang, "Improvement of Drone Gimbal System Performance Using a Parallel Structure of Explicit Model Predictive Control and Adaptive Neuro-Fuzzy Inference System", in *IEEE Access*, vol. 14, pp. 27108-27124, 2026, doi: 10.1109/ACCESS.2026.3665975.

[14] F. Liu, H. Wang, Q. Shi, H. Wang, M. Zhang and H. Zhao, "Comparison of an ANFIS and Fuzzy PID Control Model for Performance in a Two-Axis Inertial Stabilized Platform", in *IEEE Access*, vol. 5, pp. 12951-12962, 2017, doi: 10.1109/ACCESS.2017.2723541.

[15] Giuseppe Fusco, Mario Russo, "Adaptive Voltage Regulator Design for Synchronous Generator", *IEEE Transactions on Energy Conversion*, vol. 23, no. 3, pp. 946-956, Sep, 2008.

[16] Hany M. Hasanien, "Design Optimization of PID Controller in Automatic Voltage Regulator System Using Taguchi Combined Genetic Algorithm Method", *IEEE Systems Journal*, vol. 7, no. 4, pp. 825-831, Dec, 2013.

[17] Pravin Sonawane, V.B. Savakhande, M.A. Chewale, R.A. Wanjari, "Optimization of PID controller for Automatic voltage regulator system using Taguchi method", *2018 International Conference on Computer Communication and Informatics (ICCCI)*, 2018.

[18] Veena Sharma, R. Naresh, Vineet Kumar, "Automatic Voltage Regulator System with State-Feedback and PID based Sliding Mode Control Design", *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, Bhilai, India, 2021.

[19] A.R. Hasan, T.S. Martis, A.H.M.S. Ula, "Design and implementation of a fuzzy controller based automatic voltage regulator for a synchronous generator", IEEE Transactions on Energy Conversion, vol. 9, no. 3, pp. 550-557, Sep, 1994.

[20] Murat Furat, Gökçen Gıdemen Cücü, "Design, Implementation, and Optimization of Sliding Mode Controller for Automatic Voltage Regulator System", *IEEE Access*, vol. 10, pp. 55650-55674, May, 2022.

[21] Gungor Bal, Orhan Kaplan, Suleyman Samet Yalcin, "Artificial Neural Network Based Automatic Voltage Regulator for a Stand-Alone Synchronous Generator", *2019 8th International Conference on Renewable Energy Research and Applications (ICRERA)*, Brasov, Romania, 2019, pp. 1032-1037.

[22] Vineet Kumar, Veena Sharma, "Automatic Voltage Regulator with Particle Swarm Optimized Model Predictive Control Strategy", *2020 First IEEE International Conference on Measurement, Instrumentation, Control and Automation (ICMICA)*, Kurukshetra, India, 2020.

[23] T. Auld, A. W. Moore and S. F. Gull, "Bayesian Neural Networks for Internet Traffic Classification", in *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223-239, Jan. 2007, doi: 10.1109/TNN.2006.883010.

[24] S. T. Nguyen, H. T. Nguyen, P. B. Taylor and J. Middleton, "Improved Head Direction Command Classification using an Optimised Bayesian Neural Network," *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, New York, NY, USA, 2006, pp. 5679-5682, doi: 10.1109/IEMBS.2006.260430.

[25] Bishop, C. M. *Neural Networks for Pattern Recognition*. New York: Oxford University Press, 1995.

[26] Nabney, I. T. NETLAB: *Algorithms for Pattern Recognition*. London: Springer, 2002.