HYBRID NEURAL NETWORK AND REINFORCEMENT LEARNING APPROACH FOR PID CONTROLLER TUNING

Anonymous Authors[1]

ABSTRACT

In this project, we explore the enhancement of Proportional-Integral-Derivative (PID) controller tuning through an innovative hybrid approach that leverages both Neural Networks (NN) and Reinforcement Learning (RL). Traditional PID tuning techniques often do not account for the complexities of dynamically changing operational conditions, which can lead to suboptimal performance. Our approach seeks to address this limitation by employing NNs to predict optimal PID settings and utilizing RL to adaptively fine-tune the control parameters in response to the system's behavior. The system has been developed in a MATLAB/Simulink environment, where the integration of AI-driven methods with traditional control mechanisms is demonstrated. While the project is still in progress, with the system not fully operational, preliminary learning outcomes from both the NN and RL components indicate promising potential. Initial evaluations suggest improved responsiveness and adaptability of control systems, highlighting the benefits of combining AI-based techniques with classical control designs. The anticipated model aim to not only validate the efficiency of our model but also pave the way for more intelligent and adaptive control systems.

1 INTRODUCTION

Proportional-Integral-Derivative (PID) controllers stand as a cornerstone in the realm of control systems, vital for maintaining the desired performance in a multitude of applications, from industrial automation to vehicle dynamics. Despite their ubiquity, the tuning of PID controllers remains a complex task, particularly in systems where operational conditions are dynamic and unpredictable. Such systems, often characterized as 'black-box', present a significant challenge as traditional model-based tuning methods grapple with real-time adaptability and optimization.

The quest for a model-free approach that can surpass the constraints of classical tuning techniques has steered the field towards leveraging the computational prowess of artificial intelligence (AI). Our project introduces a novel synergy of a Neural Network (NN) and Reinforcement Learning (RL), converging towards a self-optimizing PID controller. This hybrid methodology is designed to anticipate and adapt to the ever-changing dynamics of black-box systems, offering a nuanced balance between precision and adaptability.

While our project is still in the developmental phase, with some aspects not fully operational, we are dedicated to transparency in our process. The preliminary outcomes of our learning algorithms demonstrate promise, with initial evaluations suggesting improvements in system responsiveness and adaptability. In the following sections, we will delve into the background and motivations behind our approach, elaborate on our implementation methods, and discuss our validation techniques, all while remaining cognizant of our current limitations and the iterative nature of our research.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the Machine Learning and Systems (MLSys) Conference. Do not distribute.

2 BACKGROUND, MOTIVATION, AND OVERVIEW

The pursuit of precision in control systems has long been underpinned by the use of PID controllers, whose inception dates back to the early 20th century. Governed by a seemingly simple control equation, these controllers have become the linchpin in an array of applications, notably in industrial automation and process control (Bennett, 2001). The control signal $u(t)$ in a PID system is a linear combination of the error $e(t)$ and its temporal integral and derivative, encapsulated by the equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \quad (1)$$

with the proportional (K_p), integral (K_i), and derivative (K_d) gains modulating the system's response to the error between the desired and actual outputs.

Despite the PID controller's widespread adoption, its traditional tuning methods, such as the Ziegler-Nichols rules,

are often ill-suited to the complexities of modern systems that exhibit non-linear or time-variant characteristics. These traditional methods do not adapt well to changing conditions, leading to control inefficiencies and, in some cases, instability.

The challenges are further compounded when dealing with black-box systems where the internal dynamics are either unknown or too complex to model accurately. This calls for a model-free approach that can dynamically adjust control parameters in real time, ensuring robustness and high performance under varying operational scenarios.

Here, machine learning techniques such as NN and RL, offers a compelling solution. NNs serve as universal function approximators, capable of capturing complex, nonlinear relationships within data (Wray & Green, 1995) . When trained on simulation data representing different operational scenarios, they can generalize and dynamically adjust PID parameters to optimize system performance. Reinforcement learning, on the other hand, introduces an adaptive element where an agent learns optimal actions through trial and error interactions with the environment, maximizing a cumulative reward signal.

Our project is an embodiment of such integration, where a hybrid of neural network predictions and reinforcement learning adaptability is employed to fine-tune PID controllers in real time. The engine idle speed control system, crucial for efficient engine performance and emission reduction, is selected as the test bed for this approach. By operating under a black-box assumption, we aim to showcase the applicability of our AI-augmented PID controller over conventional methods in terms of adaptability, efficiency, and overall performance.

3 IMPLEMENTATION

3.1 System Modeling and Design

Our study utilizes a black-box model of an engine idle speed control system implemented through a transfer function sourced from an open-source GitHub repository[1]. This transfer function simulates the engine's throttle response without revealing the internal dynamics, fitting our model-free approach for PID controller tuning.

3.1.1 Transfer Function Implementation

The transfer function, integral to our Simulink setup, abstracts the engine's dynamics by focusing solely on observable inputs (throttle position) and outputs (engine RPM). This encapsulation mimics real-world scenarios where the exact internal mechanisms of a system are not fully disclosed or understood.

[1] https://github.com/mrmaleki1376/PID_EngineIdle_Speed.git

3.1.2 Reference PID Implementation

We employed MATLAB's PID Autotuner as a benchmark for our AI-enhanced controllers. This tool adjusts the PID parameters automatically based on the system's observed responses, aiming to set a performance standard for our experimental comparisons.

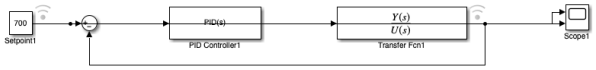

Figure 1. Schematic of the PID control system with MATLAB's PID Autotuner, illustrating the process of automatic tuning of PID parameters based on observed system outputs.

3.1.3 Neural Network and Reinforcement Learning Integration

The implementation of our control system involves:

- **Neural Network (NN)**: Configured to predict PID settings from observed data, the NN is trained on various input-output sequences representative of different system states.

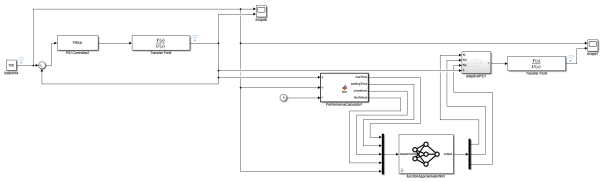

Figure 2. .

- **Reinforcement Learning (RL)**: Complements the NN by adaptively tuning the PID parameters in real-time based on performance feedback, enhancing the controller's responsiveness to unanticipated dynamics.

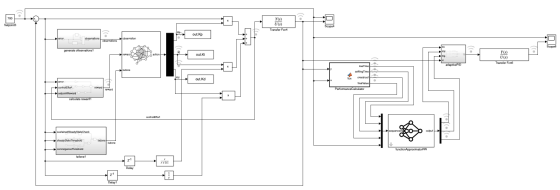

Figure 3. .

3.2 Neural Network-Based PID Tuning

We adopt a model-free approach to develop a neural network capable of tuning PID controllers for an engine idle speed system. The underlying model of the system is treated as a black box; we infer optimal PID settings through observed system responses to various controller configurations.

Data Preparation We commence by preparing the dataset for training the neural network. The data, encapsulating system performance under varying PID settings, is sourced from simulations. The key steps involve:

- Loading the dataset and handling missing values, if any.
- Normalizing input features, namely setpoint, rise time, settling time, overshoot, and final value, using z-score normalization.
- Defining the output targets, which are the PID gains K_p, K_i, and K_d.

Neural Network Architecture The neural network's architecture comprises two hidden layers, with the ReLU function serving as the activation function:

$$f(x) = \max(0, x) \tag{2}$$

where x is the input to the ReLU function. This architecture is designed to capture the nonlinear relationships inherent in the PID tuning process.

Training and Validation The prepared dataset is partitioned into training and validation sets. The network is trained using the Adam optimization algorithm with the following loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3}$$

where N is the number of samples, y_i is the true output, and $\hat{y}_i$ is the predicted output.

Training Process During training, weights are updated iteratively to minimize the loss function. The training process is conducted over changing epochs, and progress is iteratively monitored through the validation set performance.

Validation and Performance Upon completion of the training, the network's prediction power is observed using the validation set created. We calculate the mean squared error (MSE) and root mean squared error (RMSE) as metrics to evaluate performance:

$$\text{MSE} = \frac{1}{N_v}\sum_{i=1}^{N_v}(y_i - \hat{y}_i)^2 \tag{4}$$

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{5}$$

where N_v is the number of validation samples.

Dataset Creation without System Model The dataset for NN training is generated by simulating the engine idle speed control system across a spectrum of PID parameters and setpoints. This process is meticulously conducted without access to the system's internal model, reinforcing our model-free strategy. The dataset creation algorithm is as shown in Algorithm 1.

The endeavor to tune PID parameters via a neural network is, at its core, an exploratory exercise aimed at enhancing controller performance. While promising advancements have been achieved, further refinement is essential to reach optimal control behavior.

3.3 Reinforcement Learning for Real-Time Adaptation

The RL framework implemented for real-time adaptation of the PID controller parameters is centered around a DDPG architecture. This method is adept at handling continuous action spaces, making it an ideal choice for the fine-tuning required in PID control.

Simulink Model Interface The Simulink model serves as the environment in which the RL agent operates. Observations are generated based on the error between the desired setpoint and the system's current state. The observations include the integrated error (e_I), the error (e), and the derivative of the error (e_D), mathematically expressed as:

$$e = \text{Setpoint} - y(t), \quad e_I = \int e\,dt, \quad e_D = \frac{de}{dt} \tag{6}$$

where $y(t)$ is the system's output at time t.

Reward Function The reward function $R(t)$ for time t is designed to optimize control actions by balancing error minimization and energy efficiency. It is defined as:

$$R(t) = -k_e e(t)^2 - k_c u(t)^2 + k_s 1_{\{|u(t)|<\text{threshold}\}} \tag{7}$$

with $e(t)$ representing the error, $u(t)$ the control effort, and k_e, k_c, k_s acting as weights for the error penalty, control effort penalty, and efficiency bonus, respectively. Here, 1 is an indicator rewarding efficient control within a set threshold. This formulation incentivizes the RL agent to learn a

Algorithm 1 Dataset Creation for Neural Network Training

$Kp\_range \leftarrow$ logspace$(-4, -1, 10)$ {Proportional gain from 0.0001 to 0.1}
$Ki\_range \leftarrow$ logspace$(-3, 0, 10)$ {Integral gain from 0.001 to 1}
$Kd\_range \leftarrow$ logspace$(-6, -3, 10)$ {Derivative gain from 0.000001 to 0.001}
$setpoints \leftarrow [500, 600, 700, 800]$ {Example setpoints}
$data \leftarrow []$ {Initialize dataset array}
Load the Simulink model $PID\_IdleSpeed$
for each Kp in Kp_range **do**
  for each Ki in Ki_range **do**
    for each Kd in Kd_range **do**
      for each Ne in $setpoints$ **do**
        Set PID parameters in Simulink model
        Set the setpoint in the model
        $simOut \leftarrow$ Run Simulink model
        $setpointData \leftarrow simOut.logsout.get('SetpointSignal').Values.Data$
        $outputData \leftarrow simOut.logsout.get('TransferFcnSignal').Values.D$
        $S \leftarrow$ Compute metrics from $outputData$
        Append $[Kp, Ki, Kd, Ne, S.RiseTime, S.SettlingTime, S.Overshoot, outputData(end)]$ to $data$
      end for
    end for
  end for
end for
$data\_table \leftarrow$ Convert $data$ to table format
Save $data\_table$ to 'pid_tuning_data.mat'
Unload Simulink model
=0

policy that minimizes error and conserves energy, enhancing the PID control system's performance.

In the context of the RL agent, this reward signal is pivotal in learning an optimal policy that achieves the control objectives with minimal energy expenditure, leading to an efficient and effective PID control strategy.

Actor-Critic Architecture

Actor-Critic Architecture The actor network's policy function, $\mu(s)$, maps the observed state s to actions a. The critic network approximates the action-value function $Q(s, a)$, where the state s is given by the observations and a is the vector of PID parameters (K_p, K_i, K_d). The critic network is structured to combine both state and action values for its assessment, a structure that is illustrated in Figure 4.

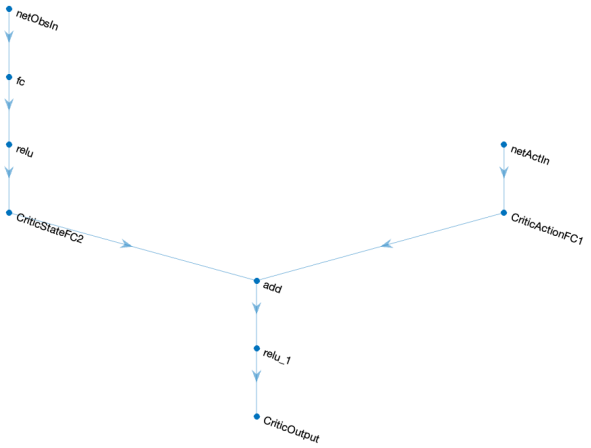

Figure 4. The critic network architecture illustrating the flow from state and action inputs to the Q-value output.

The training involves updating the policy to maximize the expected reward, leading to the following updates for the actor and critic:

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta} \left[\nabla_a Q(s, a \mid \theta^Q) \right. \\ \left. |_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s \mid \theta^\mu) |_{s=s_t} \right] \tag{8}$$

$$\nabla_{\theta^Q} L(\theta^Q) \approx E_{s_t \sim \rho^\beta, a_t \sim \beta} \left[(r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}) \mid \theta^Q) \right. \\ \left. - Q(s_t, a_t \mid \theta^Q))^2 \right] \tag{9}$$

Environment Reset Function The reset function configures the initial conditions, essential for the stability of the learning process:

$$\text{Initialize: } e(0) = \text{Setpoint} - y(0), \quad e_I(0) = 0, \quad e_D(0) = 0 \tag{10}$$

Algorithm Description The DDPG algorithm is a model-free, off-policy actor-critic algorithm for learning continuous actions. Details are shown in Algorithm 2.

Algorithm 2 Deep Deterministic Policy Gradient (DDPG) for PID Control

Require: PID model environment M, initial actor network $\mu(s \mid \theta^{\mu})$, initial critic network $Q(s, a \mid \theta^{Q})$, replay buffer $\mathcal{B}$ discount factor γ, soft update parameter τ, sample time Ts, final time Tf

0: Initialize target networks μ' and Q' with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$

0: Initialize the replay buffer $\mathcal{B}$

0: **for** each episode **do**

0: Initialize a random process $\mathcal{N}$ for exploration

0: Receive initial observation state s_1

0: **for** each step t until Tf/Ts **do**

0: Select action $a_t = \mu(s_t \mid \theta^{\mu}) + \mathcal{N}_t$ using current policy and exploration noise

0: Execute action a_t and observe reward r_t and new state s_{t+1}

0: Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$

0: Sample random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{B}$

0: Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$

0: Update critic by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^{Q}))^2$

0: Update actor policy using sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a) \nabla_{\theta^{\mu}} \mu(s) \mid_{s=s_i, a=\mu(s_i)} \tag{11}$$

0: Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{12}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{13}$$

0: **end for**

0: **end for**

0: **end for**

0: Validate the policy by simulating the PID control system with the trained DDPG agent

0: Save the trained DDPG agent and training statistics Trained DDPG agent =0

With these RL elements in place, our system aims to achieve a level of adaptability and precision in PID control that adapts in real-time to the dynamics of the environment, paving the way for more intelligent control systems.

3.4 Simulation and Data Acquisition

To evaluate our hybrid approach, we created a simulation environment using MATLAB/Simulink. This simulation is providing a platform for training the NN and the RL agent. Data acquisition involved running multiple setpoints with varied PID parameters to record system responses such as settling time, rise time, and overshoot, which served as input features for the neural network.

3.5 Challenges and Considerations

During the implementation, we faced several challenges, including the selection of appropriate neural network architectures and hyperparameters for the reinforcement learning algorithm. The model-free nature of our approach necessitated extensive simulation-based training, which is computationally expensive. Additionally, ensuring real-time performance of the RL agent posed significant challenges due to the requirement for quick adaptation to changing system dynamics.

4 Evaluation

4.1 Performance Metrics

To gauge the efficacy of the PID controller tuning, we considered various performance metrics such as response time, rise time, overshoot, settling time, and steady-state error. These metrics are pivotal for assessing the quality of control system tuning, as they collectively represent the system's response to changes and disturbances. We chose these specific metrics to align with our objectives of minimizing the error and achieving a quick and stable response from the control system.

4.2 Baseline Comparison

Our hybrid tuning approach was planned to be benchmarked against the conventional tuning method provided by MATLAB's PID Autotuner. The baseline comparison was planned to be structured around the same performance metrics for both systems.

4.3 Data Collection and Analysis

The data acquisition process was automated through the Simulink model's logging features. Following the completion of each simulation run, model extracts the relevant metrics from the system logs for analysis. These metrics included the error signal, control effort, observations made by the RL agent, instantaneous rewards, and the PID parameters adjusted by the RL agent and suggested by the NN. The collected data is then can be used to compute mean squared error (MSE), integral absolute error (IAE), integral squared error (ISE), and integral time-weighted ab-

solute error (ITAE), providing a multi-faceted view of the system's performance.

5 Related Work

Advancements in PID controller tuning have explored the integration of neural networks due to their capability as function approximators, particularly in scenarios where system dynamics are non-linear or not well understood. Many work has demonstrated the effectiveness of neural networks in PID tuning, showcasing rapid learning capabilities when compared to traditional methods such as Ziegler-Nichols (Marino & Neri, 2019; Xu, 2022; Yongquan et al., 2003; Chen & Huang, 2004).

The adaptability of reinforcement learning in PID control has been highlighted in various studies. For instance, its application in real-time adaptive control has shown significant promise in the field of mobile robotics, indicating the versatility of reinforcement learning in control system applications (Qin et al., 2018; Sedighizadeh & Rezazadeh, 2008).

The intersection of neural networks and reinforcement learning presents a novel approach in the PID tuning domain. This hybrid methodology is expected to leverage the strengths of both techniques, aiming to improve controller responsiveness and system performance under dynamic conditions.

6 Conclusion and Future Work

6.1 Conclusion

This study embarked on the ambitious task of enhancing PID controller tuning with a hybrid approach that synergizes NN and RL. We developed a framework that aims to adaptively adjust PID parameters in response to dynamic system behaviors within a MATLAB/Simulink environment. Despite facing challenges and encountering bugs that have prevented the presentation of conclusive results, the preliminary indicators are encouraging and suggest that the proposed methodology has the potential to outperform traditional tuning methods.

6.2 Limitations

The primary limitation of this study is the incomplete integration of the system, which has delayed the experimental validation and, consequently, the ability to present empirical results. The development process has uncovered several bugs that need to be addressed to ensure the system's operational efficacy.

6.3 Future Work

The project's completion within the original timeline was hindered by unforeseen health issues and time management challenges. Moving forward, the immediate priority is to debug and refine the system, ensuring its progression to full functionality. Future work will entail a comprehensive evaluation of the system's performance across a range of operational conditions, including various setpoints and external disturbances. The future plan could be to explore the following areas:

- **Robustness Testing:** Assessing the system's robustness to noise and other real-world complexities to ensure reliable performance across different applications.
- **Algorithm Optimization:** Further optimization of both neural network architectures and reinforcement learning algorithms to enhance learning efficiency and control precision.
- **MIMO Systems:** Extending the approach to Multi-Input Multi-Output (MIMO) systems, which present a more complex challenge for PID tuning.

The overarching goal of this ongoing research is to develop an intelligent PID tuning system that is both robust and adaptable, capable of addressing the complexities of modern control applications. The promising direction of this research offers a glimpse into the potential advancements in automated control systems enabled by machine learning. Our repository for this project can be found online[2].

References

Bennett, S. The past of pid controllers. *Annual reviews in control*, 25:43–53, 2001.

Chen, J. and Huang, T.-C. Applying neural networks to on-line updated pid controllers for nonlinear process control. *Journal of Process Control*, 14(2):211–230, 2004. ISSN 0959-1524. doi: 10.1016/S0959-1524(03)00039-8. URL https://www.sciencedirect.com/science/article/pii/S0959152403000398.

Marino, A. and Neri, F. Pid tuning with neural networks. In *Advances in Intelligent Systems and Computing*, pp. 476–487, 2019. doi: 10.1007/978-3-030-14799-0_41.

Qin, Y., Zhang, W., Shi, J., and Liu, J. Improve pid controller through reinforcement learning. In *2018 IEEE CSAA Guidance, Navigation and Control Conference*

[2] https://github.com/rabiakonuk/RLandNN_PID.git

(CGNCC), pp. 1–6, 2018. doi: 10.1109/GNCC42960.2018.9019095.

Sedighizadeh, M. and Rezazadeh, A. Adaptive pid controller based on reinforcement learning for wind turbine control. In *Proceedings of world academy of science, engineering and technology*, volume 27, pp. 257–262. Citeseer, 2008.

Wray, J. and Green, G. G. Neural networks, approximation theory, and finite precision computation. *Neural Networks*, 8(1):31–37, 1995. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(94)00056-R. URL https://www.sciencedirect.com/science/article/pii/089360809400056R.

Xu, P. Neural network based self-tuning pid controller. In *2022 2nd International Conference on Algorithms, High Performance Computing and Artificial Intelligence (AHPCAI)*, pp. 655–661, 2022. doi: 10.1109/AHPCAI57455.2022.10087411.

Yongquan, Y., Ying, H., and Bi, Z. A pid neural network controller. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pp. 1933–1938 vol.3, 2003. doi: 10.1109/IJCNN.2003.1223703.