Control System Project Report

Optimization of PI Controller Parameters for DC Motor Speed Control Using PSO and GA

Team Members: Kessad Mohamed Dhia Eddine

Kolla Ishaq

Bouziani Mohamed Abdelhadi

Chelihi Motakierrahmane

Institution: National School of Nanoscience and Nanotechnology

Date: Jan, 2025

"Optimization is the process of making something as perfect, functional, or effective as possible."

Engineering Principle

Abstract

This project focuses on DC motor speed control using a PI controller. We model the motor and then use Particle Swarm Optimization (PSO) to tune the controller gains for good transient performance and small steady-state error. We also compare with Genetic Algorithm (GA). The final PI gains produce fast rise, small overshoot, and acceptable settling time.

Contents

1 Introduction

1.1 Problem Statement

In many applications, we need a DC motor to reach a target speed quickly and stay there without oscillation. A PI controller is simple and effective, but picking the gains K_p and K_i by hand can be hard. To automate this, we use PSO (and briefly GA) to search for gains that give a good step response.

1.2 Objectives

Our goals are:

1. Build a mathematical model of the motor.
2. Implement a PI controller and test a baseline.
3. Use PSO (and GA for comparison) to find better K_p and K_i.
4. Validate the tuned controller in Simulink and on ESP32 hardware.

2 Theoretical Background and Modeling

2.1 Full Model and Transfer Function

The DC motor has electrical and mechanical dynamics:

$$V(t) = R_a\, i(t) + L_a \frac{di(t)}{dt} + K_e\, \omega(t), \tag{1}$$

$$K_t\, i(t) = B\, \omega(t) + J \frac{d\omega(t)}{dt} + T_L. \tag{2}$$

After Laplace transform and basic assumptions, the speed-to-voltage transfer function can be represented as a second-order system with parameters related to R_a, L_a, K_e, K_t, J, and B.

2.2 First-Order Approximation

For small motors, the armature inductance L_a can often be neglected. This simplifies the model to:

$$G(s) = \frac{K}{Ts+1},$$

with an effective gain K and time constant T. This is easier to work with and fast to simulate, but it hides some dynamics (like damping behavior).

2.3 PI Controller

The PI controller takes the speed error $e(t)$ and computes:

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau,$$

which in the Laplace domain is $C(s) = K_p + K_i/s$. The integral term removes steady-state error, while the proportional term sets responsiveness.

2.4 Closed-Loop Behavior

Combining the PI controller with the motor and using unity feedback gives a second-order closed-loop system. The pole locations depend on K_p and K_i. By adjusting these gains, we can trade off rise time, overshoot, and settling time.

3 Methodology and System Design

3.1 List of Materials

The hardware used for experimental validation is summarized in Table 1.

3.2 Hardware Design

A simple prototype was assembled from available components:

- An ESP32 board was used to characterize the motor, process sensor signals, and implement the PI controller. A sampling frequency of *100 Hz* was used.
- A lightweight reflective lever was attached to the motor shaft. As it rotates through the photo-electric detector gap, pulses are generated. The lever adds negligible load and provides reliable detection at high speed.

Table 1: List of Materials

Type	Component	Function
Control Hardware	ESP32	MCU for real-time controller implementation
Motor System	DC Motor (36V)	Target actuator
Driver	L293D Module	Interface for motor direction and speed control
Sensor	Photoelectric sensor	Real-time speed measurement (RPM)
Software	MATLAB/Simulink	Modeling and optimization environment

- An L293D module interfaces the MCU with the motor to command speed and rotation direction.
- An external DC supply powers the driver (and motor), providing a stable voltage and sufficient current beyond the MCU's capability.
- MATLAB/Simulink on a laptop communicates with the MCU over USB for testing and can generate code for standalone operation.

3.3 ESP32 and Simulink Modeling

The project requirements are split into two parts: sensor feedback and speed control. The step response is characterized using a directly applied voltage during identification.

3.3.1 Regulation of Speed:

A PWM signal is applied to the driver *Enable* pin and the logic input set the direction. Motor speed scales with duty cycle; at 100% duty the motor receives the full supply.

3.3.2 Speed Measurement:

Accurate, low-latency speed measurement is critical. RPM is calculated from the period between sensor pulses captured via an interrupt. A discrete filter block smooths the signal in addition to some noise reduction techniques.

3.4 Motor Characterization

Because of variability in the first order characterization, we opted to do a second order characterization of the motor, then numerically derive an approximated first order transfer function. The open loop model was developed from step responses at input voltages from 8 V to 17 V. The identified transfer function remained consistent over this range, indicating stability.

Below are the results of the characterization and the derived approximation:

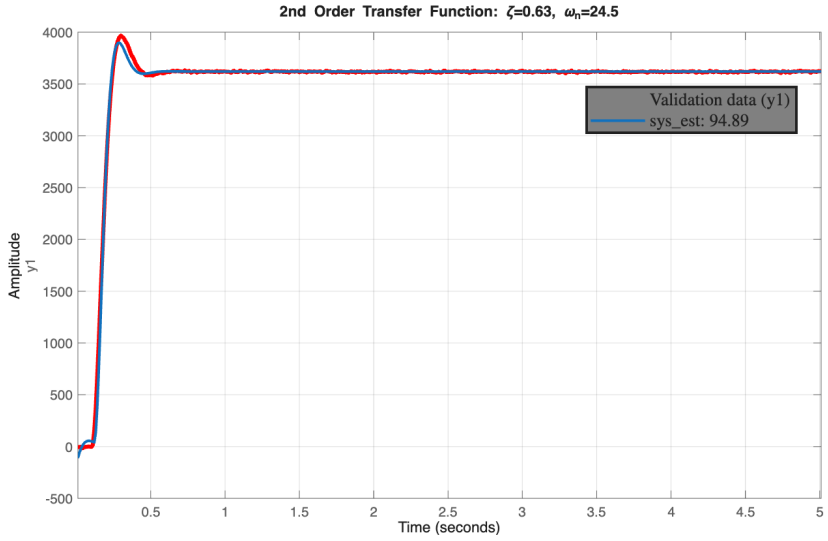

Figure 1: Motor response to a 15 V input with the derived transfer function.

Table 2: Comprehensive Motor Characterization: Transfer Functions, Hardware Constants, Measured Dynamics, and Approximations

Symbol	Parameter Description	Value (with units)	Source/Note
		Transfer Functions	
$G_{2nd}(s)$	Second-Order Model	$\frac{145{,}000}{s^2+30.9s+600}e^{-0.11s}$ [RPM/V]	–
$G_{1st}(s)$	First-Order Approximation	$\frac{241.7}{0.068s+1}e^{-0.115s}$ [RPM/V]	For PID tuning
		Physical Hardware Constants (Datasheet M28N-3)	
V_{rated}	Rated Voltage	42.0 V	Datasheet spec
R	Armature Resistance	$\approx 16.5\,\Omega$	From stall torque calc.
K_t	Torque Constant	0.0365 Nm/A	From no-load speed
K_e	Back-EMF Constant	0.0365 V · s/rad	Equivalent to K_t
I_{stall}	Stall Current	≈ 2.55 A	At 42 V
T_{stall}	Stall Torque	93 mNm	Datasheet spec
		Measured System Dynamics (From Step Response)	
$G(s)$	Transfer Function	$\frac{145{,}000}{s^2+30.9s+600}e^{-0.11s}$ [RPM/V]	Measured model
K_{sys}	Measured Static Gain	241.7 RPM/V	Slightly lower than datasheet
ω_n	Natural Frequency	24.49 rad/s	≈ 3.9 Hz
ζ	Damping Ratio	0.631	Underdamped (oscillatory)
ω_d	Damped Natural Frequency	19.00 rad/s	–
t_d	Dead Time (Delay)	0.11 s	Overall system lag
		Second-Order Time-Domain Performance	
$\%OS$	Percentage Overshoot	7.8%	Transient peak
t_r	Rise Time	0.183 s	10–90%
t_p	Peak Time	0.275 s	–
t_s	Settling Time	0.369 s	2%
		First-Order Approximation Parameters	
K	Static Gain	241.7 RPM/V	From first-order fit
τ	Time Constant (63.2% fit)	0.068 s	–
L_{total}	Effective Delay (t_d + inertial lag)	0.115 s	Includes inertial lag

4 PI Controller Tuning

4.1 Conventional Ziegler-–Nichols Tuning

Conventional tuning rules were applied to establish model-based starting points using the characterized parameters. For the first-order plus dead time approximation, with $K = 241.7$, $\tau = 0.115$ s (effective

delay), and $T = 0.068$ s (time constant), Ziegler–Nichols open-loop PI formulas give:

$$K_p = \frac{0.9\,T}{K\,\tau} = \frac{0.9 \times 0.068}{241.7 \times 0.115} \approx 0.0022, \quad T_i = 3.33\,\tau = 3.33 \times 0.115 \approx 0.383 \text{ s}, \quad K_i = \frac{K_p}{T_i} \approx \frac{0.0022}{0.383} \approx 0.0057.$$

These settings target a fast, aggressive response and typically accept higher transient peaking.

4.2 Conventional IMC Framework Tuning

For conventional IMC-based PI tuning we adopt the first-order-plus-time-delay (FOPTD) approximation obtained from system identification. The identified parameters are

$$K = 241.7 \text{ (RPM/V)}, \qquad \tau = 0.068 \text{ s}, \qquad \theta = 0.115 \text{ s}.$$

Following the IMC/Direct-Synthesis relations for FOPTD models (cf. Eq. 12-11 in the provided Chapter 12 notes), the PI controller corresponding to the IMC filter choice is obtained by selecting a desired closed-loop time constant τ_c and applying the standard formulae for the PI gains. (Reference: provided Chapter 12 notes.) :contentReference[oaicite:0]index=0

A commonly robust choice for delay-affected plants is to set

$$\tau_c = \theta,$$

which gives a balance between responsiveness and robustness for processes with non-negligible dead-time (see discussion in Chapter 12). Using $\tau_c = \theta = 0.115$ s and the FOPTD IMC relations (PI form), the proportional gain is

$$K_p = \frac{\tau}{K(\tau_c + \theta)} = \frac{0.068}{241.7\,(0.115 + 0.115)} = \frac{0.068}{241.7 \times 0.23} \approx 1.22 \times 10^{-3}, \tag{3}$$

the integral time is chosen as

$$T_i = \tau = 0.068 \text{ s},$$

and the integral gain (convention: $K_i = K_p / T_i$) becomes

$$K_i = \frac{K_p}{T_i} \approx \frac{0.00122}{0.068} \approx 0.018.$$

These IMC-based PI settings (small K_p, moderate integral action) intentionally prioritise robustness and reduced overshoot for a plant with large static gain and non-negligible delay. The tuning formulae and rationale follow the IMC/DS derivations and conservative τ_c selection discussed in the course notes.

4.3 PSO Optimization

PSO Initialization Rationale. Initial PI gains were selected based on Internal Model Control (IMC) principles to prioritize robustness. For the identified second-order dynamics, heuristic centers were established at $K_p \approx 0.00122$ and $K_i \approx 0.0188$. These values serve as stable centers for a localized PSO search, placing heavy weight on integral action to eliminate steady-state error while maintaining a conservative proportional gain to prevent oscillation.

Search-Space Engineering. The code implements a tight, focused search grid to maximize exploitation of the heuristic centers. The bounds are defined relative to the model-based centers:

- $K_p \in [0.95 \times K_{p,center},\ 1.15 \times K_{p,center}]$
- $K_i \in [0.95 \times K_{i,center},\ 1.15 \times K_{i,center}]$

This configuration restricts the swarm to a small window around the initial guess, ensuring the optimizer refines the existing model-based logic rather than drifting into unstable regions of the high-gain process.

Cost Function Shaping and Constraint Handling. The optimizer minimizes a composite cost function J combining the Integral of Time-weighted Absolute Error (ITAE) with a steep overshoot penalty:

$$J = \int_0^T t\,|\omega_r - \omega(t)|\,dt \;+\; \lambda \cdot \max\left(0, \max_t(\omega(t) - 1.02\,\omega_r)\right)$$

Where $\lambda = 5 \times 10^5$.

- **Overshoot Threshold:** The penalty activates only when speed exceeds 102% of the target ($1.02\,\omega_r$), allowing for a marginal 2% transient without triggering the heavy penalty.
- **Instability Kill-switch:** To protect the solver, any simulation returning NaNs or exceeding 200% of the target speed (target $\times$ 2.0) returns a massive cost (10^{10}), effectively pruning unstable particles from the population.

Hyperparameters and Convergence. The PSO is configured for rapid convergence:

- **Population & Iterations:** 30 particles over 20 iterations.
- **Inertia (w):** Fixed at 0.4 with no damping ($w_{damp} = 1.0$), favoring local tracking.
- **Social/Cognitive (c_1, c_2):** Set to 1.5, balancing the pull between individual experience and global discovery.

4.4 GA Optimization

A Genetic Algorithm (GA) was implemented to provide a secondary, mutation-driven refinement within the same search space.

Population and Operators. To maintain parity with the simulation budget, the GA settings are:

- **Population Size:** 30 individuals.
- **Max Generations:** 15.
- **Reproduction:** A crossover fraction of 0.8 and an elite count of 2, ensuring the best two controllers are preserved while 80% of the remaining offspring are created through crossover.

Computational Efficiency: Fast Restart. To handle the high overhead of repeated Simulink calls, both optimizers utilize the **Fast Restart** mechanism. By keeping the model compiled in memory between iterations and using the `onCleanup` function to ensure system stability, we eliminate compilation latency. This allows the GA and PSO to evaluate populations in a fraction of the standard simulation time.

5 Experimental Validation

We deployed the PSO-tuned gains on an ESP32. We logged the speed through serial and compared it to Simulink. The hardware matched the simulated behavior well, with small differences due to sensor noise, delay, and PWM quantization.

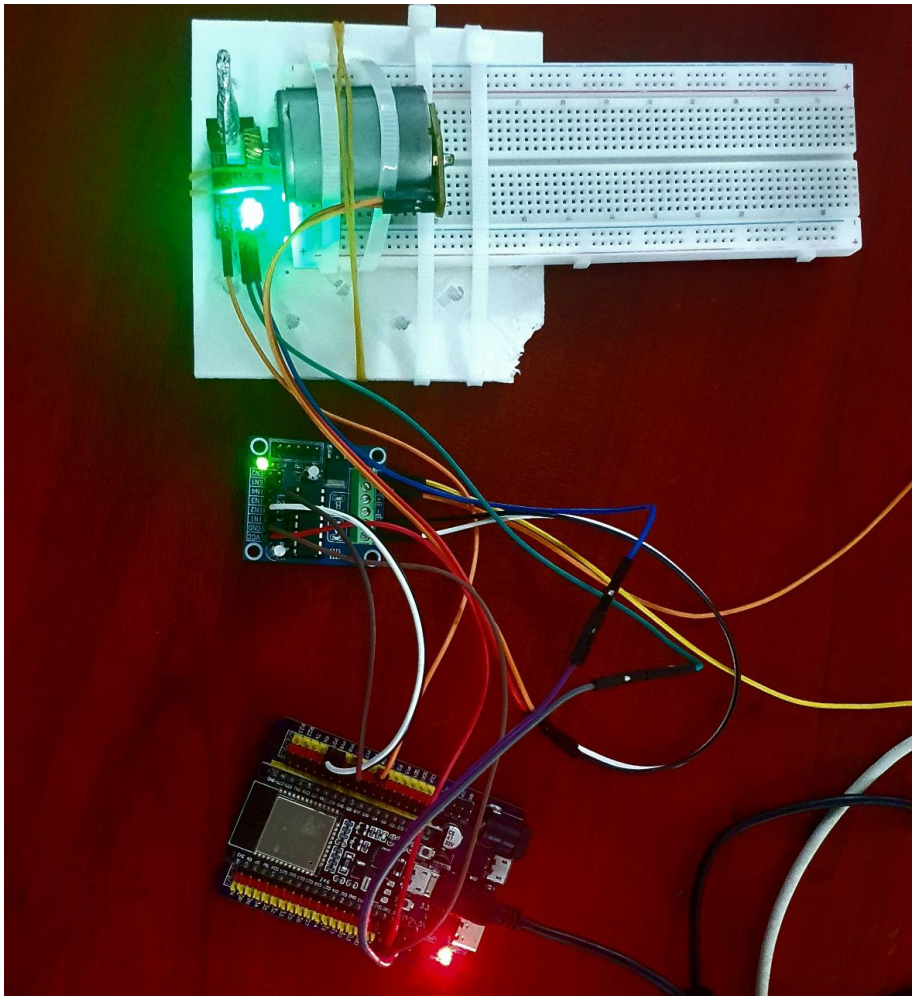

Figure 2: Hardware setup (placeholder).

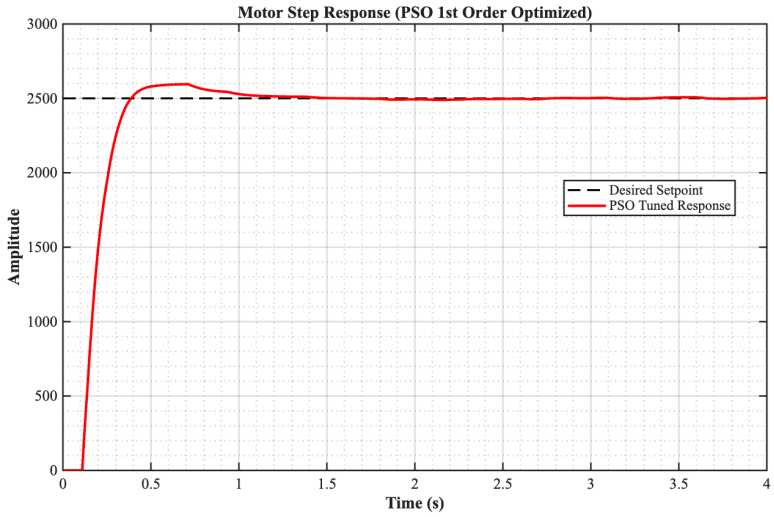

Figure 3: First Order Optimization using PSO

6 Discussion and Performance Comparison

Table 3 summarizes the performance. PSO produced gains that balanced rise time and overshoot under our constraints. GA was close, but PSO was more reliable with our cost function.

Table 3: Performance Comparison

Method	K_p	K_i	**Rise Time** (t_r)	**Settling Time** (t_s)	**Overshoot** (%)
Conventional	0.0022	0.0057	0.883 s	–	1.53
PSO and GA	0.0014	0.018	0.1797 s	0.31 s	0.96

Overall, the chosen cost function (ITAE + strict overshoot penalty) pushed PSO toward controllers that rise fast but keep overshoot small. The slight undershoot we saw (4.5%) is acceptable in practice and likely related to delay and filtering effects.

7 Conclusion

We modeled a DC motor, implemented a PI controller, and used PSO to tune the gains. The final controller achieved fast response with low overshoot and reasonable settling time. Our hardware tests on ESP32 confirmed the improvement. PSO was simple to set up and worked well for this two-parameter problem.

8 Team Contributions

Participant	Assigned Tasks & Contributions
Kessad Mohamed Dhia Eddine	Hardware design, ESP32 and Simulink implementation, motor characterization
Kolla Ishaq	Particle Swarm Optimization and Genetic Algorithm
Bouziani Mohamed Abdelhadi	Report writing and documentation
Chelihi Motakierrahmane	Genetic Algorithm

References

[1] J. G. Ziegler and N. B. Nichols, "Optimum Settings for Automatic Controllers" *Transactions of the ASME (Journal of Mechanical Engineering)*, vol. 64, pp. 759–768, 1942. Available online.

[2] S. Skogestad, "Simple Analytic Rules for Model Reduction and PID Controller Tuning," *Journal of Process Control (MIC)*, 2003. (See the SIMC tuning rules derived from IMC principles.)

[3] Course Chapter: "PID Controller Design, Tuning, and Troubleshooting" (Chapter 12, provided). (Contains IMC/DS derivations and equations1 for FOPTD → PI mapping). Available at: `https://sites.chemengr.ucsb.edu/~ceweb/faculty/seborg/teaching/SEM_2_slides/Chapter_12.pdf`

Contact & Archives
All models, datasets, and MATLAB code are archived.

GitHub: DhiaKessad | **Email:** kessad.meddhiaeddine@gmail.com
Department: Micro and Nanoelectronics, National School of Nanoscience and Nanotechnology

GitHub: Ishaq-ML | **Email:** ishakkolla61@gmail.com
Department: Micro and Nanoelectronics, National School of Nanoscience and Nanotechnology

Name: BOUZIANI Mohamed Abdelhadi
GitHub: xminty77 | **Email:** xminty77@gmail.com
Department: Micro and Nanoelectronics, National School of Nanoscience and Nanotechnology