Optimal tuning of a fractional-order PID controller using the nelder-mead method: A case study on brushed DC motor control

Son T. Nguyen[1]*, Nhung P.T. Vu[1], Tu M. Pham[1], Anh Hoang[1], Trung T. Cao[1], Long H. Nguyen[2]

[1]Hanoi University of Science and Technology
[2]Viettel Networks
[*]Corresponding author E-mail: son.nguyenthanh@hust.edu.vn
DOI: https://doi.org/10.64032/mca.v29i4.281

Abstract

The classical proportional-integral-derivative (PID) controller has long been used in control systems due to its simplicity and ease of implementation. However, the fractional-order PID (FOPID) controller, an extension of the traditional PID, offers significant advantages, particularly for complex or nonlinear systems where higher accuracy, robustness, and stability are required. By introducing fractional orders of integration and differentiation, the FOPID controller provides enhanced flexibility and a greater degree of freedom in control design. This flexibility allows for superior performance in the presence of system uncertainties, external disturbances, and dynamic variations. This paper presents a comprehensive methodology for implementing FOPID control in single-input single-output (SISO) systems. Both classical PID and FOPID controllers are tuned using the derivative-free Nelder-Mead optimization method to enhance system performance. A real-time simulation of a brushed DC motor is conducted using MATLAB Simulink, the FOMCON Toolbox, and an Arduino Nano. The results demonstrate the superior performance of the FOPID controller compared to the classical PID controller, confirming its effectiveness in real-world control applications.

***Keywords**: PID controller; fractional-order PID controller; the Nelder-Mead method.*

Abbreviations

ACO	Ant colony optimization
AEO	Artificial ecosystem-based optimization
AVR	Automatic voltage regulator
BLDC	Brushless DC
CASO	Chaotic atom search optimization
DC	Direct current
FODE	Fractional order differential equation
FOTF	Fractional order transfer function
FOPID	Fractional order proportional-integral-differential
GA	Genetic algorithm
IAE	Integral of absolute error
ITAE	Integral of time-weighted absolute error
MBITF	Modified Bode's ideal transfer function
PID	Proportional-integral-differential
PWM	Pulse width modulation
PSO	Particle swarm optimization
PMSM	Permanent magnet synchronous motor
SISO	Single-input and single-output

1. Introduction

As the FOPID controller was first introduced in 1999 [1, 2], there is hope that such a control is superior to that of traditional PID in SISO control systems. The conceptual structure of the FOPID controller is formed by fractional order integrator and fractional order differentiator. As opposed to the classical PID controller, the FOPID controller has two extra parameters: the integer order of the fractional integrator and the integer order of the fractional derivative. An ideal FOPID controller is chosen for its ability to improve the robustness and flexibility of control systems just as the standard PID controller.

Various optimization algorithms have been proposed for optimal tuning of the FOPID controller including chaotic atom search optimization (CASO) [3], genetic algorithm (GA) [4], particle swarm optimization (PSO) [5, 6], ant colony optimization (ACO) [7], modified Bode's ideal transfer function (MBITF) [8]. Even though the FOPID controller has several superiorities over the PID controller, the FOPID controller utilization in the industry is still limited, since the FOPID controller realizes it is more complex in comparison with the PID controller. This cause may affect the reliability of the FOPID controller.

The Nelder-Mead method outperforms CASO [3], PSO [6], and MBITF [8] in FOPID controller design. It is simple, derivative-free, and computationally efficient, ideal for real-time or hardware-limited applications. Unlike stochastic PSO and CASO, which require careful parameter tuning and multiple runs for stable results, the Nelder-Mead method can be used with minimal settings, offering robustness and repeatability. It excels at fine-tuning controllers once near an optimal solution. It integrates smoothly with simulation tools and supports standard continuous-time criteria like IAE or ITAE. Unlike the frequency-dependent and inflexible MBITF, the Nelder-Mead method is frequency-independent and effectively shapes time-domain responses, making it a practical, easy choice for FOPID controller design.

The combination of artificial ecosystem-based optimization (AEO) with the classical Nelder-Mead method for PID controller design fuses global and local search processes, achieving better results compared to using only the Nelder-Mead method [9]. Although the Nelder-Mead method is suitable for employing its local optimization property, it is sensitive to initial conditions and can easily fall into local minima. AEO further improves this by searching the solution

Received: 12 February 2025; Revised: 12 September 2025; Accepted: 18 September 2025.

space globally, allowing the algorithm to escape prematurely from a suboptimal solution because the range of exploration becomes larger. With the hybrid AEO and Nelder-Mead method, PID tuning will always offer increased accuracy and robustness, especially for systems with complexity or nonlinearity, but at a higher computational cost.

Until now, FOPID controllers have been used in various applications such as reduction of vibration and noise for induction motors [10], the speed control of permanent magnet synchronous motors (PMSM) [11], automatic voltage regulator (AVR) systems [12], the speed control of brushless DC (BLDC) motors [13], liquid level control [14] and temperature control [15].

This study focuses on exploring the Nelder-Mead method to optimally tune the parameters of the classical PID and FOPID controllers. The contribution of this study is that it provides a complete construction approach to FOPID controller-based systems compared to classical PID controller-based systems, with greater robustness, precision, and stability. Both controller designs are optimized by means of the Nelder-Mead optimization method, which is based on a derivative-free approach. The proposed FOPID controller is tested in real-time on the Simulink model representing a brushed DC motor that is implemented in the FOMCON Toolbox and uses the Arduino Nano.

The remaining sections of this paper are organized as follows. Section 2 provides the concept of the FOPID controller. In Section 3, the Nelder-Mead optimization method is described in detail. Section 4 gives key steps for optimizing classical PID and FOPID controllers for SISO control systems using the Nelder-Mead method. Section 5 presents a real-time simulation of a brushed DC motor control system using Simulink, the FOMCON Toolbox for MATLAB, and Arduino Nano. Finally, Section 6 provides a conclusion for this study.

2. The FOPID controller

The FOPID controller can be seen as a generalization of the classical PID controller with an integrator of real order λ and a differentiator of real order μ. The transfer function of the FOPID controller is given by:

$$C(s) = K_P + \frac{K_I}{s^{\lambda}} + K_D s^{\mu} \qquad (\lambda, \mu > 0) \quad (1)$$

Where:

- K_P : Proportional gain
- K_I : Integral gain
- K_D : Derivative gain
- λ : Order of integration
- μ : Order of differentiation

If $\lambda = 1$ and $\mu = 1$, the controller yields a classical PID controller. If $\lambda = 0$, the controller is a PD^{μ} controller. If $\mu = 0$, the controller is a PI^{λ} controller. In other words, these controllers are specific cases of the general $PI^{\lambda}D^{\mu}$ controller, which is a flexible function usually requiring a fine adjustment for five parameters including K_P, K_I, λ, K_D and μ.

3. The nelder-mead optimization method

For many years, various gradient descent techniques have been used to find the minimum or maximum of objective functions in a multidimensional space. However, gradient descent techniques always require the evaluation of partial derivatives of the objective function. Due to the challenging nature of obtaining partial derivatives of objective functions with gradient descent optimization methods, the Nelder-Mead optimization method can be regarded as an optimal alternative. Nevertheless, the Nelder-Mead approach is a heuristic search algorithm that may converge to non-stationary locations for problems that can be addressed using alternative optimization strategies. Therefore, selecting suitable initial values for variables is essential to attain the necessary convergence. In each iteration, the Nelder-Mead method operates by generating a simplex of the following points:

- Reflex
- Expand
- Contract outside
- Contract inside
- Shrink

The Nelder-Mead approach can be used by following steps below:

Step 1: A simplex consists of points $x(i)$, $i = 1,...,n+1$.

Step 2: The points in the simplex are arranged in ascending order based on their function values, from $f(x(1))$ (the lowest value) to $f(x(n+1))$ (the highest value). At each iteration, the current worst point $x(n+1)$ is removed and replaced with another point in the simplex.

Step 3: Calculate the point of reflection:

$$r = 2m - x(n+1) \quad (2)$$

where $m = \sum_{i=1}^{n} \frac{x_i}{n}$ and calculate $f(r)$.

Step 4: If $f(x(1)) \le f(r) < f(x(n))$, the r is accepted and this iteration is ended (**Reflect**).

Step 5: If $f(r) < f(x(1))$, then generate the expand point s as follows:

$$s = m + 2(m - x(n+1)) \quad (3)$$

and calculate $f(s)$.

If $f(s) < f(r)$, then s is accepted and the iteration is ended (**Expand**).

Otherwise, r is accepted and the iteration is stopped (**Reflect**).

Step 6: If $f(r) \ge f(x(n))$, then a contraction is performed between m and the better of $x(n+1)$ and r.

If $f(r) < f(x(n+1))$ (r is better than $x(n+1)$), then c is calculated as follows:

$$c = m + (r - m)/2 \tag{4}$$

and calculate $f(c)$.

a) If $f(c) < f(r)$, the c is accepted and the iteration is ended (**Contract outside**).
Otherwise, Step 7 is continued (**Shrink**).

b) If $f(r) \geq f(x(n+1))$, then generate cc as follows:

$$cc = m + (x(n+1) - m)/2 \tag{5}$$

and calculate $f(cc)$.

c) If $f(cc) < f(x(n+1))$, then cc is accepted and the iteration is stopped (**Contract inside**).
Otherwise, Step 7 is continued (**Shrink**).

Step 7: The point n are computed as follows:

$$v(i) = x(1) + (x(i) - x(1))/2 \tag{6}$$

Compute $f(v(i))$, $i = 1,...,n+1$. At the next iteration, the simplex consists of $x(1), v(2),..., v(n+1)$ (**Shrink**).

Figure 1 displays the points generated by the Nelder-Mead algorithm within a simplex, illustrating how the algorithm explores the search space by updating and transforming the simplex during the optimization process.

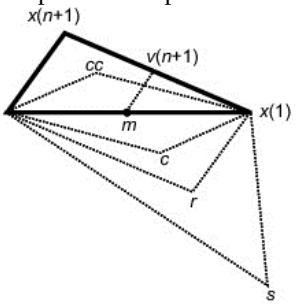

Figure 1: The principle of generating points in a simplex.

The Nelder-Mead method is a widely applied optimization technique for solving unconstrained optimization problems. Compared with other optimization methods, such as GA, PSO and ACO, it is more efficient in some cases. Here are some of its strengths:

- Simple and easy implementation: The Nelder-Mead method is easy to implement and requires tuning fewer parameters than GA, PSO, ACO, and other methods, which involve complicated population dynamics.
- Low dimensional efficiency: The Nelder-Mead method is known to be very efficient in low dimensions (usually less than 10 dimensions). It can therefore be applied more rapidly and effectively than GA, PSO, and ACO in those cases where GA, PSO, and ACO require many iterations and incur higher computational costs.
- Derivative-free optimization: The Nelder-Mead method does not use gradients and is therefore useful for optimizing non-differentiable functions. On the other hand, gradient descent-based or gradient-based methods could have difficulty with these functions.
- Local search power: The Nelder-Mead method is a local search algorithm, which can refine solutions effectively about a given point. This feature makes it well-suited for problems where a reasonably good starting approximation is known and convergence to a local minimum is desired.
- Lower computational overhead: The Nelder-Mead method is a simplex (a geometric object) and tends to have lower computational overhead than other deterministic population-based methods like GA, PSO, and ACO, where multiple candidate solutions need to be evaluated in each iteration.
- Noise robustness: There may be times when the Nelder-Mead method is more robust to noise added to the objective function than stochastic methods (such as GA and PSO, which can be affected by random blips in the evaluation of fitness).

Example:

This section gives a procedure to optimize Rosenbrock's function using the Nelder-Mead method [16]. Rosenbrock's function is a non-convex function proposed by Howard H. Rosenbrock in 1960. This function is usually used as a performance test problem for various optimization algorithms. The function has the following form:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2 \tag{7}$$

The function has a global minimum at $(x, y) = (a, a^2)$, where $f(x, y) = 0$. Usually, $a = 1$ and $b = 100$. If $a = 0$, the function is symmetrical and the minimum at the origin. A MATLAB script can be written to find the minimum of Rosenbrock's function. Firstly, a MATLAB function is created for an objective function as follows:

```
function f = rosen_function(xy)
x = xy(1);
y = xy(2);
a = 1;
b = 100;
f = (a - x)^2  + b*(y - x^2)^2;
```

The initial values of x and y are set to 20. The MATLAB script to find the minimum of the function has the following form:

```
x1 = 20;
y1 = 20;
optimInput = [x1 y1];
options = optimset('Display','iter',…
                   PlotFcns'@optimplotfval,'MaxIter',…
                    200);
[a, fval] = fminsearch(@rosen_function,optimInput,…
                       options);
x2 = a(1)
y2 = a(2);
x2_y2 = [x2 y2]
```

The minimum of the function is:

$$f\left(x=1,\mathrm{y}=1\right)=5.21856\times10^{-10}\approx0$$

4. Optimization of controllers

Both classical PID and FOPID controllers can be optimized using the Nelder-Mead method. Figure 2 is the block diagram of a typical closed-loop control system. $G(s)$ is the transfer function of the plant, $H(s)$ is the transfer function of the sensor and $C(s)$ is the transfer function of the controller.

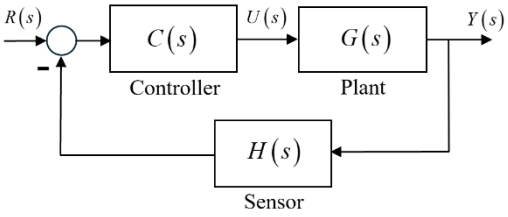

Figure 2: Diagram of a closed-loop control system.

The transfer function of the system is given by:

$$\frac{Y(s)}{R(s)}=\frac{C(s)G(s)}{1+C(s)G(s)H(s)} \tag{8}$$

Minimizing the integral of time-weighted absolute error (ITAE) is commonly used as a good performance index for optimizing PID controllers. Mathematically, the ITAE performance index is given by:

$$ITAE=\int_0^\infty t\left|e(t)\right|dt \tag{9}$$

Where t is the time and $e(t)$ is the difference between the reference (setpoint) and the controlled variable.

The computational complexity of the Nelder-Mead method in designing optimal FOPID controllers depends on the following aspects:

- The number of the search space: $n=5$ (the number of parameters in the FOPID controller is 5).
- T_f : time to evaluate the cost function such as $ITAE$.
- I : the number of iterations
- Worst case: up to $2n=10$ function evaluations per iteration.
- Average: roughly $n+1=6$.

The computational cost per iteration is:

$$T_{iter}=O\left(n.T_f\right)=O\left(5.T_f\right) \tag{10}$$

The total computational cost for the entire iterative process is:

$$T_{iter}=I.T_{iter}=O\left(I.n.T_f\right)=O\left(I.5.T_f\right) \tag{11}$$

5. Real-time simulation-based validation

To visualize the effectiveness of the Nelder-Mead method in optimally tuning the FOPID controller, the brushed DC motor is chosen as a typical SISO system.

5.1 Mathematical model of the brushed DC motor

The mathematical model of the brushed DC motor consists of two equations as follows:

$$v_a(t)=R_a i_a(t)+L_a\frac{di_a(t)}{dt}+K\omega(t) \tag{12}$$

$$Ki_a(t)-D\omega(t)-T_L=J\frac{d\omega(t)}{dt} \tag{13}$$

In which $v_a(t)$ is the voltage applied to the motor's armature, $i_a(t)$ is the armature current and $\omega(t)$ is the angular velocity of the rotor. R_a and L_a are the resistance and inductance of the armature circuit, respectively. K is the motor constant and D is the damping constant of the rotor. Finally, J and T_L are the inertia moment and load torque, respectively.

Taking the Laplace transform for equation (12) and (13) gives:

$$V_a(s)=R_a I_a(s)+L_a sI_a(s)+K\omega(s) \tag{14}$$

$$KI_a(s)-D\omega(s)-T_L=Js\omega(s) \tag{15}$$

Re-arranging (14) gives:

$$I_a(s)=\frac{V_a(s)-K\omega(s)}{L_a s+R_a} \tag{16}$$

Re-arranging (15) results in:

$$\omega(s)=\frac{KI_a(s)-T_L}{Js+D} \tag{17}$$

Equations (16) and (17) are used to model the brushed DC motor in Simulink.

5.2 Brushed DC motor driven by a DC chopper

The brushed DC motor can be driven by a DC chopper shown in Figure 3. The conduction states of semiconductor switch S results in two levels (low and high levels) of the voltage applied to the motor.

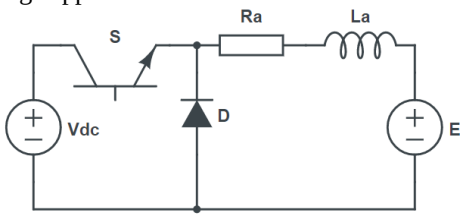

Figure 3: Principle of the brushed DC motor driven by a DC chopper.

The average value of the voltage applied to the motor is calculated as follows:

$$V_m = \frac{T_{on}}{T_{on}+T_{off}} V_{dc} = \frac{T_{on}}{T} V_{dc} = dV_{dc} \tag{18}$$

Where V_{dc} is the constant DC voltage, T_{on} and T_{off} are durations switch S switches on and off in a period. $T = T_{on} + T_{off}$ is the period and d is called the conduction duty cycle of switch S. This control technique is well-known as the pulse width modulation (PWM). The DC motor has the following parameters:

- $R_a = 17.5887(\Omega)$
- $L_a = 1.7047(H)$
- $K = 1.8095(V/rad/s)$
- $J = 0.0579(kg.m^2)$
- $D = 24\times10^{-4}(N.m.s)$
- $T_L = 2(N.m)$

5.3 Optimization of controllers using the nelder-mead method

When the load torque is equal to zero $(T_L = 0)$, by using equations (16) and (17), the transfer function of the brushed DC motor has the following form:

$$G(s) = \frac{\omega(s)}{V_a(s)} = \frac{a}{s^2 + bs + c} \tag{19}$$

Where

$$a = \frac{K}{L_a J};\ b = \frac{R_a J + L_a D}{L_a J};\ c = \frac{R_a D + K^2}{L_a J} \tag{20}$$

Transfer function (19) is used to optimally design the PID and FOPID controllers.

The MATLAB function of the objective function for optimizing the classical PID controller has the following form:

```
function ITAE = objectiveFcn_1(x)
Ra = 17.5887;
La = 1.7047;
K = 1.8095;
J = 0.0579;
D = 24e-4;
a = K/La/J;
b = (Ra*J + La*D)/La/J;
c = (Ra*D + K^2)/La/J;
num = a;
den = [1 b c];
G = 30/pi*tf(num, den);
H = tf(1,[0.1 1]);
Kp = x(1);
Ki = x(2);
Kd = x(3);
C = pid(Kp,Ki,Kd);
T = 0.01;
t = 0:T:10;
e = 1 - step(feedback(C*G,H),t);
ITAE = sum(t'.*abs(e));
```

The MATLAB script for optimizing the classical PID controller has the following form:

```
Ra = 17.5887;
La = 1.7047;
K = 1.8095;
J = 0.0579;
D = 24e-4;
TL = 2;
Ts = 0.01;
a = K/La/J;
b = (Ra*J + La*D)/La/J;
c = (Ra*D + K^2)/La/J;
num = a;
den = [1 b c];
G = 30/pi*tf(num, den);
H = tf(1,[0.1 1]);
C = pidtune(G,'pid');
Kp1 = C.Kp;
Ki1 = C.Ki;
Kd1 = C.Kd;
optimInput = [Kp1 Ki1 Kd1];
options = optimset('Display','iter','PlotFcns',…
                   @optimplotfval,'MaxIter',100);
[a, fval] = fminsearch(@objectiveFcn_1,…
                       optimInput,options);
Kp2 = a(1);
Ki2 = a(2);
Kd2 = a(3);
```

The MATLAB function of the objective function for optimizing the FOPID controller has the following form:

```
function ITAE = objectiveFcn_2(x)
s = fotf('s');
Ra = 17.5887;
La = 1.7047;
K = 1.8095;
J = 0.0579;
D = 24e-4;
a = K/La/J;
b = (Ra*J + La*D)/La/J;
c = (Ra*D + K^2)/La/J;
G = 30/pi*a/(s^2 + b*s + c);
H = 1/(0.1*s + 1);
Kp = x(1);
Ki = x(2);
lamda = x(3);
Kd = x(4);
mu = x(5);
C = fracpid(Kp,Ki,lamda,Kd,mu);
T = 0.01;
t = 0:T:10;
e = 1 - step(feedback(C*G,H),t);
ITAE = sum(t'.*abs(e));
```

The MATLAB script for optimizing the FOPID controller has the following form:

```
Ra = 17.5887;
La = 1.7047;
K = 1.8095;
J = 0.0579;
D = 24e-4;
TL = 2;
```

```
Ts = 0.01;
a = K/La/J;
b = (Ra*J + La*D)/La/J;
c = (Ra*D + K^2)/La/J;
num = a;
den = [1 b c];
G = 30/pi*tf(num, den);
H = tf(1,[0.1 1]);
C = pidtune(G,'pid');
Kp1 = C.Kp;
Ki1 = C.Ki;
Kd1 = C.Kd;
lamda1 = 0.5;
mu1 = 0.5;
optimInput = [Kp1 Ki1 lamda1 Kd1 mu1];
options = optimset('Display','iter','PlotFcns',…
                   @optimplotfval,'MaxIter',100);
[a, fval] = fminsearch(@objectiveFcn_2,…
                       optimInput,options);
Kp3 = a(1);
Ki3 = a(2);
lamda3 = a(3);
Kd3 = a(4);
mu3 = a(5);
```

Tables 1 and 2 show the values of the parameters of the classical PID and FOPID controllers before and after the optimization. These parameters are used in the real-time simulation routines and allow for comparison of control performance for a set of tuning.

Table 1: Parameters of the classical PID controller before and after optimization

	K_p	K_i	K_d
Before optimization	0.4051	1.6637	0.0152
After optimization	0.3600	1.0507	0.0428

Table 2: Parameters of the FOPID controller before and after optimization

	K_p	K_i	λ	K_d	μ
Before optimization	0.4051	1.6637	0.5	0.0152	0.5
After optimization	0.1588	0.5926	0.9996	0.0163	0.6901

Based on the parameters of the FOPID controller after optimization in Table 2, the stability of the entire control system can be evaluated by using the following MATLAB script:

```
s = fotf('s');
Ra = 17.5887;
La = 1.7047;
K = 1.8095;
J = 0.0579;
D = 24e-4;
a = K/La/J;
b = (Ra*J + La*D)/La/J;
c = (Ra*D + K^2)/La/J;
G = 30/pi*a/(s^2 + b*s + c);
H = 1/(0.1*s + 1);
Kp = 0.1588;
Ki = 0.5926;
lamda = 0.9996;
Kd = 0.0163;
mu = 0.6901;
C = fracpid(Kp,Ki,lamda,Kd,mu);
sys = feedback(C*G,H)
K = isstable(sys)
```

Running the MATLAB script above results in 1 for K; therefore, the system is stable.

5.4 Real-time simulation using the simulink and arduino nano

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 microcontroller. The Arduino Nano can be accessed in the Simulink environment using the Arduino IO Package. Figure 4 depicts inexpensive hardware with the Arduino Nano. Figure 5 provides an arrangement of real-time simulation hardware with Simulink and the Arduino Nano microcontroller. For the feedback system, an RC circuit that looks like Figure 6 is attached to pin 5 (PWM) of the Arduino Nano. The voltage on the capacitor is connected to pin A0 (ADC) of the Arduino Nano. The RC circuit takes the following parameters: $R=10(k\Omega)$ and $C=10(\mu F)$. The transfer function of the RC circuit is given by:

$$H(s)=\frac{1}{RCs+1}=\frac{1}{0.1s+1} \tag{21}$$

Figure 7 presents the graphical user interface (GUI) of the data acquisition (DAQ) software developed for real-time monitoring of the motor speed. This interface allows users to observe the motor's behavior as it operates, providing immediate feedback and facilitating better analysis and control.

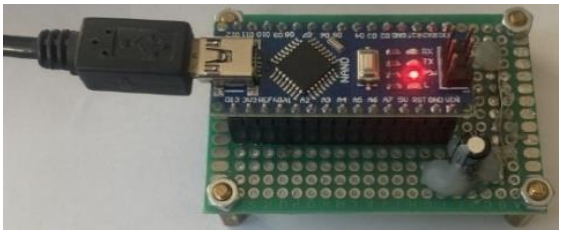

Figure 4: Real-time simulation hardware using Arduino Nano.

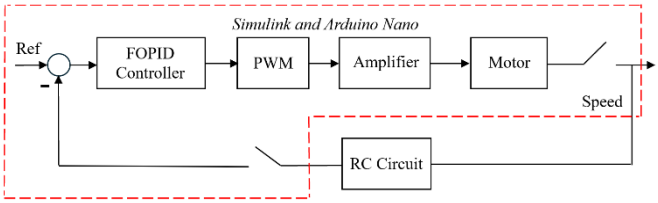

Figure 5: Structure of the real-time simulation hardware using Simulink and the Arduino Nano microcontroller.

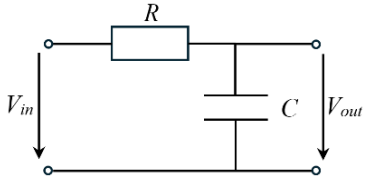

Figure 6: RC circuit.

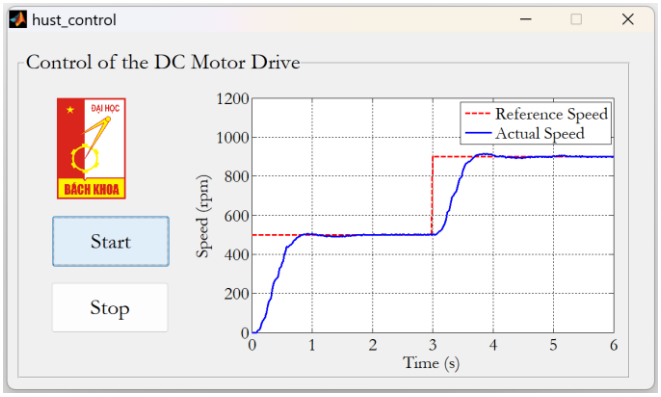

Figure 7: GUI of the DAQ software.

Figure 8 presents the detailed Simulink model of the brushed DC motor used for dynamic performance analysis. Figure 9 shows the real-time simulation setup with an optimized classical PID controller regulating the motor speed. In comparison, Figure 10 illustrates the same setup using an optimized FOPID controller, highlighting the differences in control strategy and performance. Collectively, these figures depict the system's design, implementation, and monitoring.

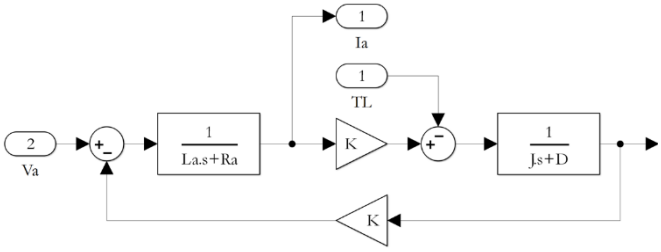

Figure 8: Model of the brushed DC motor in Simulink.

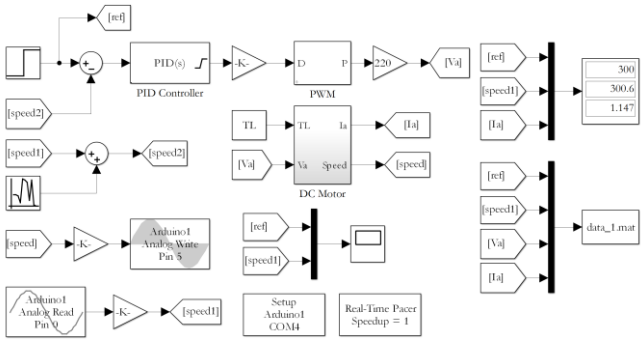

Figure 9: Real-time simulation diagram of the motor control system using the classical PID controller in Simulink.

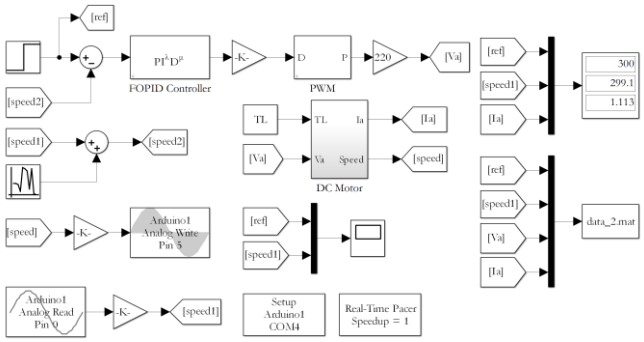

Figure 10: Real-time simulation diagram of the motor control system using the FOPID controller in Simulink.

Figures 11, 12, and 13 clearly illustrate the enhanced performance of the optimized FOPID controller in comparison to the classical PID controller under three different reference speeds: 300 rpm, 500 rpm, and 700 rpm, respectively. The FOPID controller demonstrates significantly faster setpoint tracking, along with noticeable reductions in both settling time and overshoot. These improvements lead to superior transient behavior and more accurate tracking performance, which are critical for dynamic motor control applications. Furthermore, the quantitative data presented in Tables 3, 4, and 5 reinforce these findings by consistently showing reduced settling times and overshoot values across various operating scenarios. These performance gains translate into a more stable and responsive system, capable of maintaining reliability and control accuracy even under changing load and environmental conditions. Overall, the FOPID controller proves to be a more robust and effective solution for high-performance motor control.

Figures 14, 15, and 16 illustrate the motor's armature current waveforms when operating under three different reference speeds: 300 rpm, 500 rpm, and 700 rpm, respectively. As observed from these results, the FOPID controller significantly reduces the magnitude of the initial current surge during the startup phase compared to the standard PID controller.

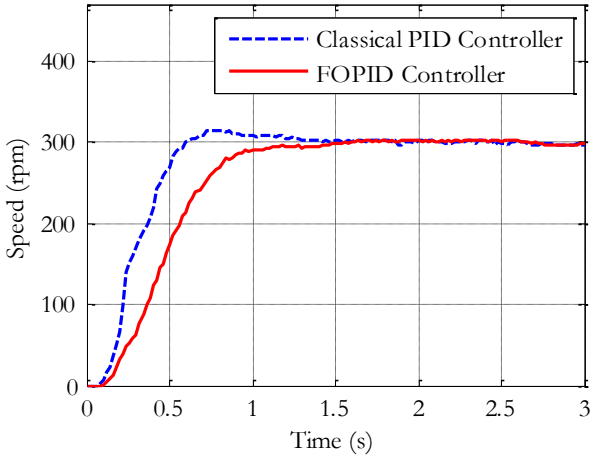

Figure 11: Speed responses at a reference of 300 rpm.

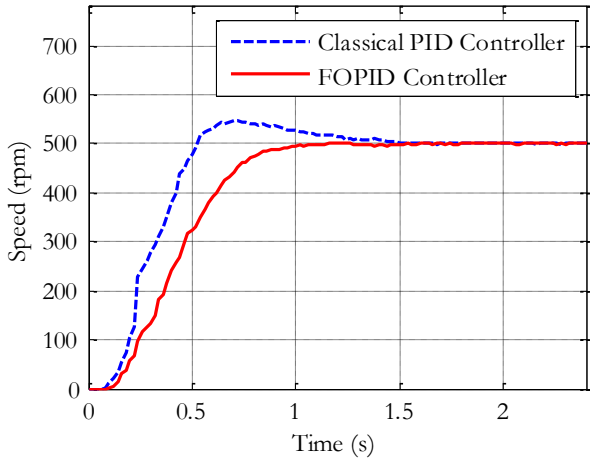

Figure 12: Speed responses at a reference of 500 rpm.

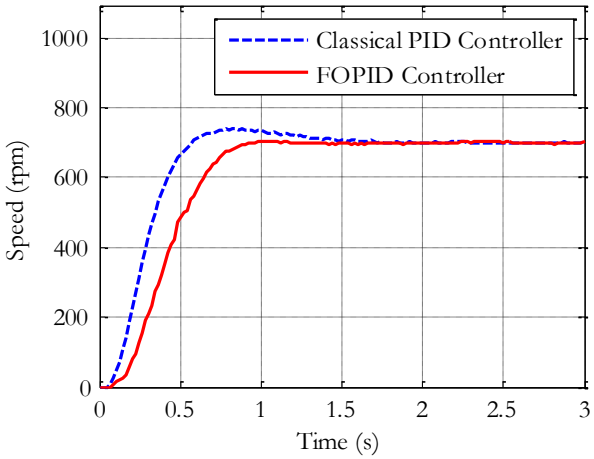

Figure 13: Speed responses at a reference of 700 rpm.

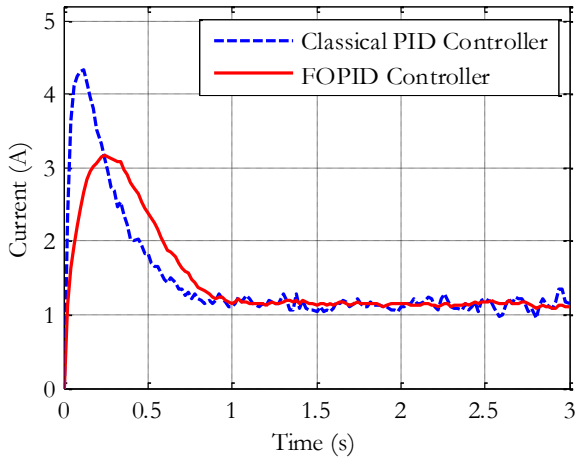

Figure 14: Armature currents at a reference of 300 rpm.

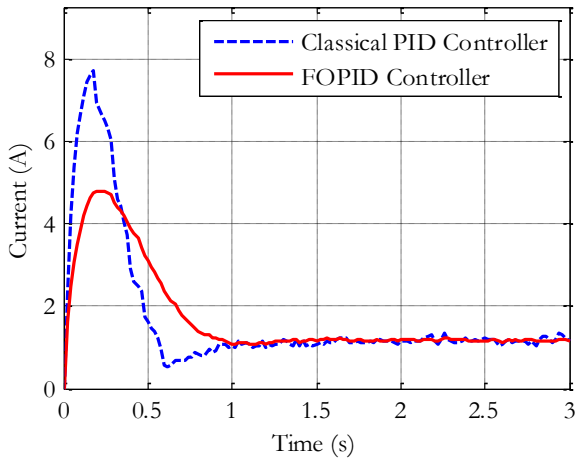

Figure 15: Armature currents at a reference of 500 rpm.

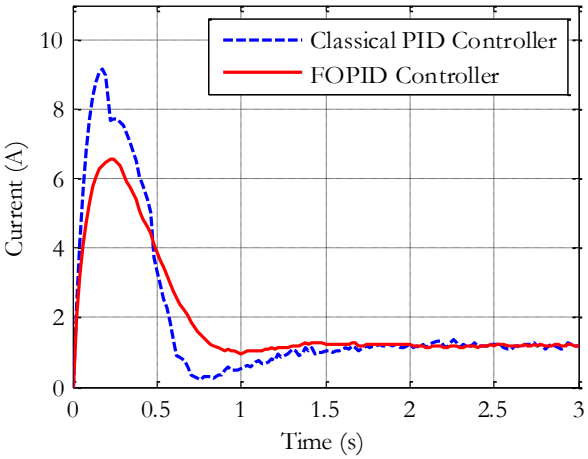

Figure 16: Armature currents at a reference of 700 rpm.

Table 3: Comparison between the classical PID and FOPID controllers at a reference speed of 300 (rpm)

	Rise Time (s)	Settling Time (s)	Overshoot (%)
Classical PID Controller	0.3418	1.2894	5.9113
FOPID Controller	0.6083	1.0784	1.4706

Table 4: Comparison between the classical PID and FOPID controllers at a reference speed of 500 (rpm)

	Rise Time (s)	Settling Time (s)	Overshoot (%)
Classical PID Controller	0.3008	1.4213	9.7059
FOPID Controller	0.5213	0.904	0.8824

Table 5: Comparison between the classical PID and FOPID controllers corresponding to a reference of 700 (rpm)

	Rise Time (s)	Settling Time (s)	Overshoot (%)
Classical PID Controller	0.3338	1.2828	5.428
FOPID Controller	0.4992	0.8571	0.4175

6. Conclusions

This study shows that the Nelder-Mead method can be applied to optimize the parameters of the classical PID controller and the FOPID controller. Although the PID controller is a popular and simple answer applicable to many control problems, including complex, nonlinear, and time-varying systems. However, the FOPID controller, as an extension of the conventional PID controller, provides greater flexibility and control accuracy based on the fact that more degrees of freedom can be obtained by introducing fractional calculus, resulting in the integration and differentiation of non-integer orders. The Nelder-Mead method is used to optimize the five parameters of the FOPID controller, including proportional gain, integral gain, derivative gain, integral order,

and derivative order, to maximize the system performance. In order to validate the usefulness of the proposed approach, a real-time simulation environment has been established, which is based on MATLAB/Simulink with the incorporated FOMCON toolbox and is supplemented by an Arduino Nano microcontroller connected to a DC motor with commutation. The experimental results show that the proposed optimized FOPID controller performs better than a traditional PID controller in terms of the fundamental performances, such as settling time and overshooting. This enhancement not only showcases the better control potential of the FOPID controller but also verifies that the FOPID controller can be used for advanced control tasks where classical PID controllers have limitations. Finally, the study highlights the benefits of coupling a fractional-order control approach with a metaheuristic optimization methodology for improving the efficiency and robustness of control systems in practical environments.

Acknowledgement

This research is funded by Hanoi University of Science and Technology (HUST) under project number T2024-PC-059.

References

[1] Podlubny, "Fractional-order systems and PI/sup /spl lambda//D/sup /spl mu//-controllers," *IEEE Transactions on Automatic Control*, vol. 44, no. 1, pp. 208-214, Jan, 1999.

[2] N. Lachhab, F. Svaricek, F. Wobbe, H. Rabba, "Fractional Order PID Controller (FOPID)-Toolbox," *2013 European Control Conference (ECC)*, Zürich, Switzerland, 2013, pp. 3694-3699.

[3] B. Hekimoğlu, "Optimal Tuning of Fractional Order PID Controller for DC Motor Speed Control via Chaotic Atom Search Optimization Algorithm," *IEEE Access*, vol. 7, pp. 38100-38114, Mar, 2019.

[4] J. Wan, B. He, D.Wang, T. Yan, Y. Shen, "Fractional-Order PID Motion Control for AUV Using Cloud-Model-Based Quantum Genetic Algorithm," *IEEE Access*, vol. 7, Aug, 2019, pp. 124828-124843.

[5] H. O. Erkol, "Optimal PIλDμ Controller Design for Two Wheeled Inverted Pendulum, *IEEE Access*," vol. 6, Nov, 2018, pp. 75709-75717.

[6] F. Meng, S. Liu, A. Pang, K. Liu, "Fractional Order PID Parameter Tuning for Solar Collector System Based on Frequency Domain Analysis," *IEEE Access*, vol. 8, Aug, 2020, pp. 148980-148988.

[7] A. Mughees, S.A. Mohsin, "Design and Control of Magnetic Levitation System by Optimizing Fractional Order PID Controller Using Ant Colony Optimization Algorithm," *IEEE Access*, vol. 8, Jun, 2020, pp. 116704-116723.

[8] Nie Zhuo-Yun, Zheng Yi-Min, Wang Qing-Guo, Liu Rui-Juan, Xiang Lei-Jun, "Fractional-Order PID Controller Design for Time-Delay Systems Based on Modified Bode's Ideal Transfer Function," *IEEE Access*, vol. 8, May 2020, pp. 103500-103510.

[9] Davut Izci, Baran Hekimoglu, Serdar Ekinci, "A new artificial ecosystem-based optimization integrated with Nelder-Mead method for PID controller design of buck converter," *Alexandria Engineering Journal*, vol. 61, Mar 2022, pp. 2030-2044.

[10] Chang-Hung Hsu, "Fractional Order PID Control for Reduction of Vibration and Noise on Induction Motor," *IEEE Transactions on Magnetics*, vol. 55, no. 11, Nov 2019, pp. 1-7.

[11] Peng Gao, Guangming Zhang, Huimin Ouyang, Lei Mei, "An Adaptive Super Twisting Nonlinear Fractional Order PID Sliding Mode Control of Permanent Magnet Synchronous Motor Speed Regulation System Based on Extended State Observer," *IEEE Access*, vol. 8, Mar 2020, pp. 53498 - 53510.

[12] Bekkouche Hanane, Abdelfatah Charef, "IMC based fractional order control design for automatic voltage regulator system," *2015 7th International Conference on Modelling, Identification and Control (ICMIC)*, Sousse, Tunisia, 2015.

[13] Fan Zhang, Zhuangju Li, "Design of fractional PID control system for BLDC motor based on FPGA," *2018 Chinese Control And Decision Conference (CCDC)*, Shenyang, China, 2018, pp. 2293-2296.

[14] K. Sundaravadivu, B. Arun, K. Saravanan, "Design of Fractional Order PID controller for liquid level control of spherical tank," *2011 IEEE International Conference on Control System, Computing and Engineering*, Penang, Malaysia, 2011, pp. 291-295.

[15] Rongxuan Li, Feng Wu, Pingzhi Hou, Hongbo Zou, "Performance Assessment of FO-PID Temperature Control System Using a Fractional Order LQG Benchmark," *IEEE Access*, vol. 8, Jun 2020, pp. 116653-116662.

[16] Rosenbrock, H.H., "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, no. 3, 1960, pp. 175-184.