IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

Demystifying PID Controllers: Implementation and Tuning in Embedded System Design

Mr. Yogesh R Chauhan

Senior Engineer, PES, E-infochips (An Arrow Company), Ahmedabad, Gujarat

Abstract: Proportional-Integral-Derivative (PID) controllers are needed for ensuring the accuracy and stability of embedded systems. In this research, we explore the role of PID controllers in robust feedback control for a range of applications. We explore the theoretical underpinnings of the proportional, integral, and derivative terms, evaluating their respective contributions and their combined effects on system response. This paper also includes practical implementation approaches for embedded design. It discusses significant challenges faced in embedded control systems, such as integral wind-up, the impacts of sampling intervals, output saturation, and noise reduction for the derivative term. Finally, two well-known tuning algorithms, Manual Tuning and the Ziegler-Nichols, are set out as systematic approaches to maximum controller performance. This research presents an applied manual for engineers designing embedded systems for implementing and tuning PID controllers effectively to deliver stable, accurate, and responsive system performance.

Keywords: PID Control, BLDC Motor, PID Design, Embedded Systems, Motor Control Tuning

I. INTRODUCTION

The closed-loop system is an important concept in embedded design. It ensures that the system maintains the desired behaviour, e.g., maintaining a specified temperature or controlling a motor's speed. For example, a motor's position or speed can be regulated using PWM (Pulse Width Modulation) pulses that drive it. However, in the absence of feedback, there is no way to verify motor speed or position. The same PWM signal can cause a motor to overshoot under no-load and undershoot in loaded conditions. To give reliable accuracy every time, we need to integrate feedback into the embedded system.

Let's first understand the need for a PID controller. In the given example, we need to move a BLDC (Brushless Direct Current) motor from its current position to a given target position. To reach the target position, we can apply PWM signal to drive the motor until it reaches the target position. However, this won't work. If we apply a constant power signal, the motor may overshoot the target position because of its momentum. If we apply too little power to reduce overshoot, the motor may never reach the target position or may take a long time. If the load changes (something starts resisting the motor's movement), the motor may stop moving and fail to reach the desired position.

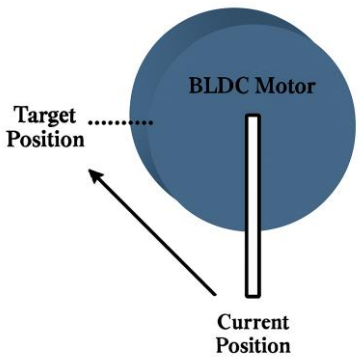

Figure 1: BLDC Motor position control using PID

This is why there is the need for Closed loop control system. In which instead of sending constant blind command, we constantly monitor the actual position of the motor and compare it with the target. We can use this error to adjust the effort we apply to the motor.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

II. LITERATURE SURVEY

PID controllers continue to be one of the most popular control algorithms for motor drive applications for a few reasons including their simple structure, tunablility, and ease of use in real-time embedded systems. In BLDC motor applications, PID tends to contend with nonlinear effects such as back-EMF, cogging torque, friction and is required to stay within the timing and resource constraints of microcontroller-based systems that can only provide Field Oriented Control (FOC) of the motor.

There have been several studies in recent years that utilized intelligent and metaheuristic tuning strategies to enhance the PID performance of BLDC systems. Guntay and Saritas [1] identified that dynamic adjustments of the PID coefficients via fuzzy logic created a better transient response overall and diminished steady-state error compared to fixed-gain PID. Hu et al. [2] implemented a genetic algorithm-optimized fuzzy PID that diminished overshoot and decrease settling time and Sarma and Bardalai [3] applied driving-training-based optimization for PID tuning, allowing for enhanced load disturbance handling. Nguyen et al. [4] applied the Nelder-Mead direct search optimization strategy to refine the PID gains of commercial BLDC motors, achieving decreased overshoot, without modeling the plant. Leeart et al. [5] implemented a Lévy-flight intensified search algorithm for the PID tuning optimization problem, while Vu et al. [6] implemented robust PID with sliding mode control for immunity to noise and anti-windup ability.

While these approaches display performance improvement, they also increase computational complexity, paramterization burden, and development time. For cost-sensitive or resource-constrained embedded platfroms, especially in high-rate haptic control loops, these approaches are not always feasible. This work will use a classical PID structure, discretized with a trapezoidal integral and filtered derivate term. The PID controller was selected for its computational efficiency, predictable behavior, and ease of tuning in the field.

III. PID CONTROL

This section explains Proportional, Integral and derivative controller implementation in the microcontroller. The controller parts can be used individually or combined depending on the use case.

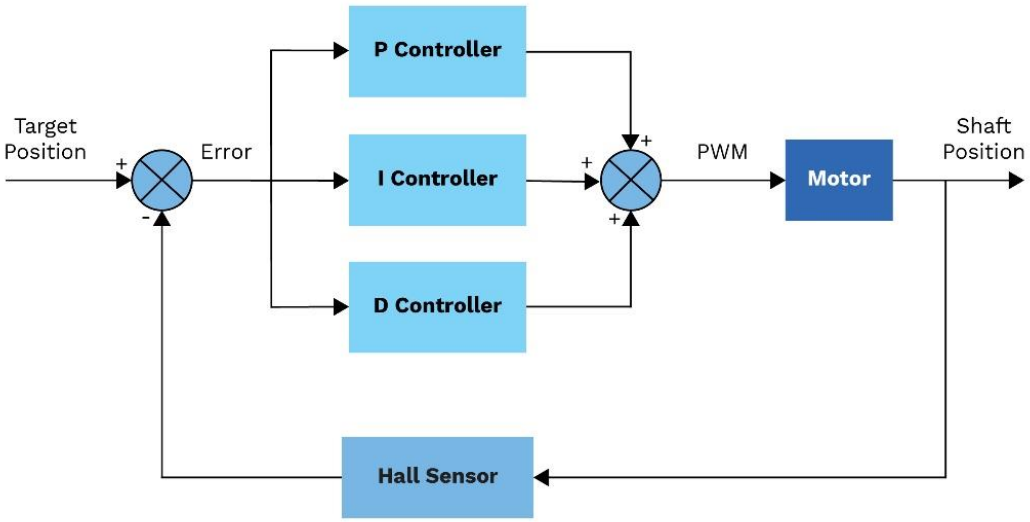

Figure 2; The Block diagram of PID

A PID controller consists of three terms:

- Proportional (P) – Responds to the current error - How far we are from the target
- Integral (I) – Responds to the accumulation of past errors - How long we've been away from the target (accumulated error)
- Derivative (D) – Predicts future error based on its rate of change How fast we are approaching the target (rate of error change)

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

A. Mathematical Representation

The continuous-time PID formula is:

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

In discrete time for embedded implementation:

$$u[n] = K_p e[n] + K_i \sum_{i=0}^{n} e[i] \Delta t + K_d \frac{e[n] - e[n-1]}{\Delta t}$$

B. Proportional Controller

In our motor example, if the motor shaft is far from the desired position, we want to apply more voltage or power to reach the desired position. To make the motor stop at the desired position, reduce the voltage or power as it nears the destination. If the motor position overshoots, apply negative voltage or power to force the motor to return to its proper position. In brief, we must apply voltage or power in proportion to the distance between the desired position and the current position.

Error = Desired Position − Current Position

Output Voltage = K_p × Error

C. Integral Controller

In the P controller, as the current position approaches the near-target position, the error reduces, and the computed output power is small to overcome friction or gravity. This error is called steady-state error.

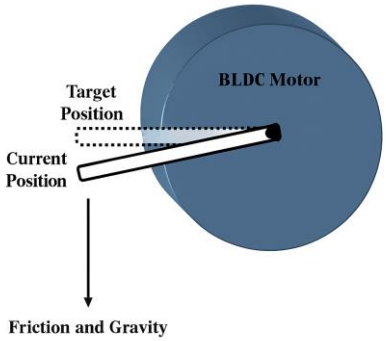

Figure 3: Steady state error

To eliminate steady-state error, we can use the integral controller in addition to the P controller. In the Integral controller, the error will accumulate from the beginning.

$$\text{Integral} = \sum_{k=0}^{n} e(k)$$

The integrated error is multiplied by the constant Ki. The error will continue to accumulate, and at one point the output will be large enough to overcome friction or gravity.

In a continuous-time PID controller, the integral term is represented as:

$$u_I(t) = K_i \int_0^t e(\tau)\, d\tau$$

where:

- u(t) is the integral output at time t,
- Ki is the integral gain,
- e(τ) is the error signal.

In digital controllers, where time is sampled at intervals of Ts, the integral must be approximated in discrete time as below:

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

$$u_I(k) \approx K_i T_s \sum_{i=0}^{k} e\,(i)$$

To calculate this efficiently we can write a recursive form:

$$u_I(k) = u_I(k-1) + \text{Area between } k-1 \text{ and } k$$

The simplest method, and the one that assumes constant error from one step to the next:

$$u_I(k) = u_I(k-1) + K_i \cdot e(k) \cdot T_s$$

This is simple to implement but can be inaccurate, especially if the error varies rapidly.
A more accurate way to integrate is to take the average of the error and the previous error:

$$u_I(k) = u_I(k-1) + K_i \cdot \frac{e(k) + e(k-1)}{2} \cdot T_s$$

This approximates the area as a trapezoid and will give better estimates of the true integral.

D. Derivative Controller

Let's say you are driving a car and need to travel an exact distance of 500 m. Now, to reach quickly, you would accelerate up to the speed limit, and when you are about to reach, you would slowly apply the brake to make it stop. So here, you are accelerating and decelerating based on the current position. This is what exactly the derivative controller does.

The derivative controller measures how fast the error is changing over time. The controller will slow down the motor if it moves too quickly while reaching the desired position.

$$\text{Derivative} = \frac{e(k) - e(k-1)}{T_s}$$

Consider the scenario where motor need to move from 0◦ to 90◦. Suppose in the first control loop, the error is 90◦, and in the second control loop, the error is 45◦. Here, based on the current error and last error, it can be observed that the motor is moving very quickly. For smoother operation, the derivative component is −45◦, which then will be multiplied with the derivative coefficient, and hence it will make the motor slow down. And if the motor is moving at a desired rate, the derivative component will be small, and hence it will allow motor to move.

In digital systems, the derivative can be calculated using the difference between the current and previous error values, divided by the sampling time Ts:

$$u_D(k) = D \cdot \frac{e(k) - e(k-1)}{T_s}$$

Here:

- $u_D(k)$ is the derivative output at step k,
- D is the discrete derivative gain (often K_d),
- $e(k)$ and $e(k-1)$ are the current and previous error values respectively,
- Ts is the sampling time.

IV. PID GAIN AND ITS IMPACT

There are many important factors to consider when selecting the proportional gain (Kp) of your system, as Kp has a significant impact on system performance. For example, a small Kp will produce a relatively slow response.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

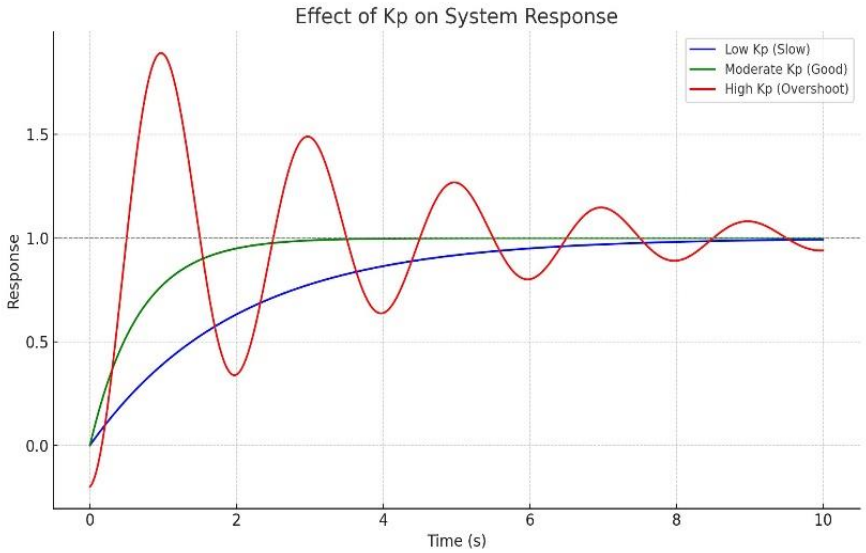

Figure 4: Effect of Kp values on system response

This means it will take an extended period for the system to get to its desired target position. Additionally, a small Kp may not result in eliminating steady-state error, if it occurs. Conversely, a moderate Kp usually has a good balanced response and therefore response will be fast enough to minimize unwanted overshoot and oscillation, while maintaining stability. Caution is needed if you take a high Kp since Kp can provide a very fast response, however, the cost of stability is very high. Sometimes a very high Kp can induce severe oscillations, which is detrimental to the performance of your system.

It is recommended to start at a low Kp, then slowly increase. When increasing Kp you need to continually observe how your system responds otherwise you will lose control of the oscillations and have a large overshoot, which compromises system integrity and the quality of your system performance.

A Small Ki will increase the response time of the system, and a large Ki will result in quicker response but can introduce the oscillations.

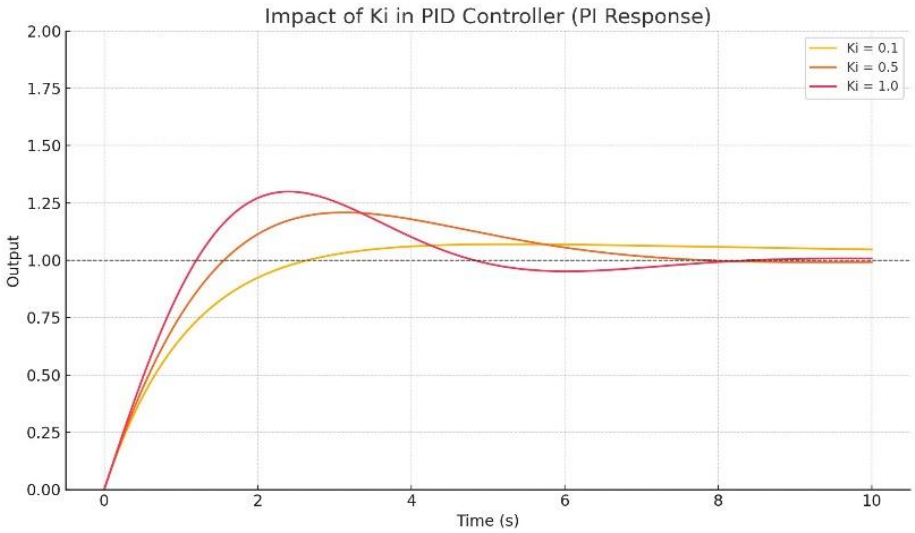

Figure 5: Effect of Ki values on system response

The derivative controller smoothens motion. The derivative controller will allow higher Ki and Kp value as it gives damping effect.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

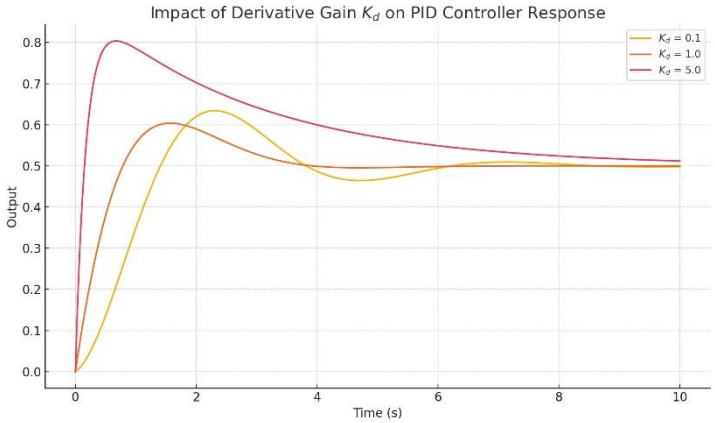

Figure 6: Effect of Kd values on system response

A small Kd will have a small damping effect, which can lead to overshoot and oscillations, while a large Kd will provide a stronger damping effect and reduce overshoot, but it may slow the system's responsiveness and can amplify noise.

V. PRACTICAL CONSIDERATIONS

A. Noise and Derivative Filtering

The derivative term is very sensitive to noise as it depends on the change in the error. So, A small spike can be amplified by derivative term which can destabilize the system. The derivative term amplifies high frequency noise. This can be resolved by introducing low pass filter on derivative terms in our embedded design [6].

$$D_f(k) = \alpha \cdot D_f(k-1) + (1-\alpha) \cdot D(k)$$

Where:

- $D_f(k)$ is the filtered derivative at time step k,
- $D(k)$ is the raw derivative at time step k,
- $\alpha \in [0.8,0.95]$ is the smoothing factor that controls the trade-off between responsiveness and noise suppression.

B. Impact of Sampling time

The sampling time is how frequently the PID controller checks the system and adjusts. If the sampling time is too long, the controller is slow to determine a change, and it may slow or change how accurately the system operates. If the sampling time is very short, the controller may react too many times, meaning there may be noise input, instability or high processing power overhead.

For motor control, the system changes very fast, so a very short sampling period (milliseconds) helps make a fast response and a smoother system. For temperature control the change in the system is very slow, so a long sampling time is fine, and does not require as frequent checks. Using an informed sampling time helps the PID controller work properly for each kind of system.

C. Integral Wind-up

The integral term of a PID controller accumulates error over time to remove steady-state error. However, when the controller output is saturated (or limited to maximum or minimum output), the integral term keeps accumulating even though the system is operating at 100%. When the system now reaches the setpoint, the controller will overreact due to all the previously added integral errors. This causes integral windup where the controller overreacts, once back in the controllable range and creates instability [6].

Consider a thermostat that is attempting to heat a room from 10◦C to 50◦C. Since the room's temperature is below setpoint, the heater is operating at 100%. All the while, the integral term has been accumulating large temperature error, thus causing the integral term to build up to far too large. Once the room is close to the desired temperature, the integral term results in the heater remaining on longer than necessary and instead raises the temperature to a much higher value than the setpoint.

In actual PID control processes it is essential to avoid integral wind-up, as integral wind-up will amplify system overshoot and/or delay system response. The most common strategies to prevent integral wind-up include:

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

- Conditional integration: this method only integrates when the controller output is not at its maximum or minimum limits.
- Clamping integral action by limiting how large or how small the integral term may get.
- Back-calculation in which the integral value is adjusted when the controller output reaches saturated limits and then corrected based on closed-loop feedback.
- Integrator freeze when the actuator reaches a limit or the system error is low and steady the integral action effectively is frozen.

Any of these strategies are designed to help keep the system stable and responsive, especially during periods of rapid change and when the actuator is at limits.

D. Limiting Output

If there is an abrupt change to the control signal with dynamic systems, it can generate mechanical stress, electric spikes, or totally destabilize the system. Setting limits on how much an output can change creates predictability and safe behaviors for the system.

Consider a car that must drive 100 m ahead. Jumping abruptly to 100 km/h would not only be undeliverable but would also cause instability for the system. Instead, the acceleration is gradual and controlled. Similarly, adjusting the rate of change of the PID output in control systems produces smooth transitions.

VI. PID TUNING

Implementing PID Controller is easy, the challenging part is the Tuning. Tuning requires the right combination of (Kp), integral (Ki), and derivative (Kd) gains to ensure system responds quickly, accurately and stably. Several PID tuning methods are available. Some of the most common approaches are discussed below:

E. Manual Tuning

This is the easiest and most direct way to do it [4].

- Make both the integral (Ki) and derivative (Kd) equal to zero.
- Then raise the proportional gain (Kp) until the system oscillates.
- Once you've got that point, lower Kp a little to avoid continuous oscillation.
- Then add Ki, so that the steady-state error can be eliminated.
- Finally, you can add Kd, to dampen any oscillation that may persist.

F. Ziegler–Nichols Tuning Method

The Ziegler–Nichols method is the most common and popular PID tuning method [3].

- Begin by turning off the integral and derivative components, i.e., set Ki = 0 and Kd = 0.
- Increase the proportional gain Kp step by step until the system output shows persistent and stable oscillations. The gain at which this occurs is referred to as the ultimate gain (Ku), and the duration of the oscillation is the ultimate period (Pu).
- Once Ku and Pu are known, use the following empirical formulae to compute the PID parameters:

Table 1: Ziegler–Nichols PID Tuning Table

Controller Type	Kp	Ki	Kd
P-only	0.5Ku	–	–
PI	0.45Ku	$\frac{2K_p}{P_u}$	–
PID	0.6Ku	$\frac{2K_p}{P_u}$	$\frac{K_p P_u}{8}$

This technique provides a good initial tuning technique and works effectively for many industrial processes. It may produce aggressive responses in some systems and may, therefore, necessitate fine-tuning afterward. When performing Ziegler–Nichols tuning, be sure to continuously observe the system closely; large oscillations can be dangerous in sensitive or safety-critical systems.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

VII. PID CONTROL IN BLDC GIMBAL MOTOR

This experiment describes the application of a PID control in BLDC Gimbal motor. The goal was to replicate virtual detents—the familiar mechanical knob "clicks" tactile sensation—entirely within software. Instead of sturdy mechanical detents, this allows for detent strength, the number of detent positions to be changed dynamically, and tactile feedback to be updated dynamically. This is done using precise motor motion control with a PID controller in a Field Oriented Control (FOC) scheme [4],[6].

G. Hardware Setup

The experimental setup was formulated on these basic elements:

BLDC Gimbal Motor – A direct drive motor was used for its smooth buttery motion without cogging and high torque density for quick haptic responses.
High-Resolution Encoder – A magnetic offers precise, real-time angular feedback, allowing the controller to react quickly to any position change.
Microcontroller – A performance-focused MCU (e.g., an STM32G4) ran both the PID control loop and the FOC algorithm at some kilohertz for ultra-low-latency torque control.
Motor Driver – A three-phase FOC-enabled driver board drove the motor with accurately controlled currents, mapping the PID torque commands to smooth and accurate motion.

H. The Virtual Detent Algorithm

Mechanical detents work because they create low energy "valleys" for the knob to come back to naturally. Between detents, a restoring force brings the knob back to the next valley. In this experiment, we modelled that restoring force completely in software. Firmware stores detent positions in an array like {0°, 90°, 180°, 270°} for four evenly spaced detents.The encoder consistently identifies the angular position of the motor, and the firmware determines which detent position is nearest. This specific position is used as the dynamic setpoint for the PID controller.

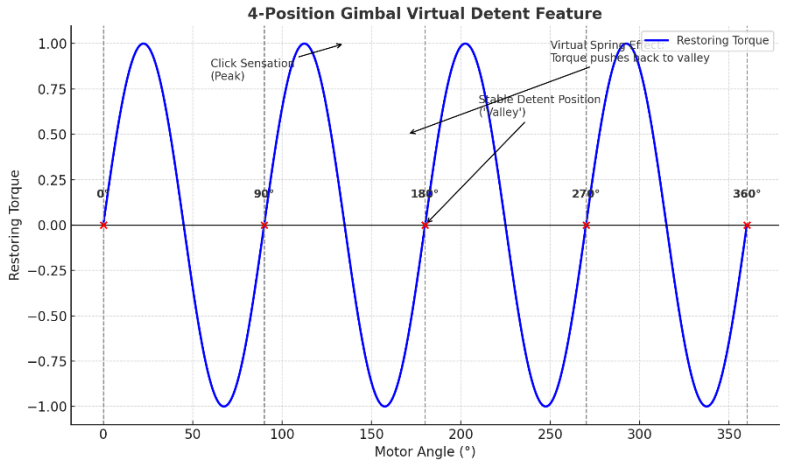

Figure 7: Virtual Detent Implementation

- **Motor Angle (X-Axis):** This represents the physical rotation of the motor, from 0 to 360 degrees.
- **Restoring Torque (Y-Axis):** This shows the direction and strength of the force the motor applies to either push or pull the shaft into a stable position.
- **Stable Detent Positions:** These are the four points (at 90°, 180°, 270°, and 360°/0°) where the blue curve crosses the zero-torque line. At these exact angles, the system is perfectly balanced, and no torque is applied. These are the "valleys" where the virtual knob rests.

Take the stable position at 180°. If you manually turn the motor slightly to the left (e.g., to 170°), the graph shows a **positive restoring torque**. The arrows point right, indicating the motor is actively pushing itself back towards 180°. If you turn it slightly to the right (e.g., to 190°), the torque becomes **negative**, and the arrows point left, again pulling the motor back to 180°.

DOI: 10.17148/IJIREEICE.2025.13912

The "click" or "snap" that a user feels happens when they push the motor over a "hill" (the peak of the curve, e.g., at 135°). Once they push past that peak, the direction of the restoring torque flips, and the motor actively pulls itself into the next valley (at 90°).

I. Proportional Gain

The proportional term responds directly to position error, with a restoring torque that increases with detent travel. Large Kp makes the detent feel more positive and stiffer, so the shaft returns rapidly when pushed out of alignment. Pushed too far, however, large Kp will cause the system to overshoot the detent and oscillate [5], particularly in a low-friction, direct-drive motor such as the one employed here.

J. Integral Gain

The integral term compensates small, continuous errors—like sensor offset-induced drift or stray interference—by gradually injecting torque over time. This ensures the shaft will settle precisely at the detent center. The compromise is that large Ki introduces integral wind-up, where the accumulated correction overshoots the objective before reversing, decelerating the return to stability. In a gimbal motor with virtually no natural damping, Ki must be decreased and combined with an anti-windup technique.

K. Derivative Gain

The derivative component reacts to the rate at which the error is changing. It serves to decelerate the shaft as it reaches a detent, thereby limiting overshoot and enhancing the smoothness of movement. Consequently, this adjustment makes the "snap" into position more precise. if Kd is too large, the system will become sluggish and over-damped, resulting in reduced sharpness.

Balancing the Gains Good detent feel is the reward of careful tuning. Kp sets how strongly the system reacts to errors, Ki keeps the motor from slowly drifting away from the target, and Kd helps it settle smoothly without bouncing back and forth.Because the BLDC gimbal motor has good response to small changes in torque, small gains can make a big difference. Tuning on the actual loop frequency (on the order of kilohertz) and inspecting both the measured response and the touch feel yields the best results.

VIII. CONCLUSION

PID controllers are extremely effective in embedded design; if properly tuned and well-implemented, they provide precise and stable control in real-world situations.

REFERENCES

[1]. S. Guntay and İ. Saritas, “BLDC Motor speed control with dynamic adjustment of PID coefficients: Comparison of fuzzy and classic PID,” *J. Appl. Methods Electron. Comput.*, vol. 12, no. 1, pp. 22–32, Mar. 2024, doi: 10.58190/ijamec.2023.80.

[2]. S. H. Hu, T. Wang, and C. Wang, “Speed control of brushless direct current motor using a genetic algorithm–optimized fuzzy proportional integral differential controller,” *Adv. Mech. Eng.*, vol. 12, no. 10, 2019, doi: 10.1177/1687814019890199.

[3]. H. Sarma and A. Bardalai, “An intelligent PID controller tuning for speed control of BLDC motor using driving training-based optimization,” *Int. J. Power Electron. Drive Syst.*, vol. 14, no. 4, pp. 2474–2486, 2023, doi: 10.11591/ijpeds.v14.i4.pp2474-2486.

[4]. S. T. Nguyen, V. N. Pham, N. D. Nguyen, T. V. Nguyen, and T. Q. Tran, “Optimal tuning of digital PID controllers for commercial BLDC motors using the Nelder–Mead method,” *Power Electron. Drives*, vol. 10, no. 1, pp. 210–226, Jan. 2025, doi: 10.2478/pead-2025-0015.

[5]. X. Leeart, W. Romsai, and A. Nawikavatan, “PID Controller Design for BLDC Motor Speed Control System by Lévy-Flight Intensified Current Search,” in *Intelligent Computing and Optimization*, P. Vasant, I. Litvinchev, M. C. Vasquez, and J. W. Maringa, Eds., *Advances in Intelligent Systems and Computing*, vol. 1324, Springer, Cham, 2021, pp. 1176–1185, doi: 10.1007/978-3-030-68154-8_99.

[6]. N. S. Vu, V. C. Pham, P. A. Nguyen, M. L. Dao Thi, and T. H. Vu, “Robust PID sliding mode control for DC servo motor speed control,” *arXiv preprint*, arXiv:2508.06567, Aug. 2025.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

BIOGRAPHY

Yogesh Chauhan is an Embedded Systems Engineer with 5 years of hands-on experience in real-time software development, currently working at eInfochips. He holds a master's degree in electrical engineering from LD College of Engineering, with a strong academic foundation in embedded control systems and digital electronics. His core technical competencies include FreeRTOS, Zephyr RTOS, bare-metal firmware development, and low-level driver integration. He has authored a research paper titled "Multi-Stepper Motor Control using CAN Bus Communication Protocol," addressing efficient multi-node motor control strategies in distributed systems**. He is also an inventor on a filed U.S. patent for a novel Gimbal Encoder.** His work focuses on developing robust, scalable, and maintainable embedded software architectures with a strong emphasis on modular design, unit testing, and real-time performance optimization.

APPENDIX

Proportional Controller Pseudo Code

```
while (true)
{
// Step 1: Read the current motor position current_position = get_current_position();
// Step 2: Calculate the error error = target_position - current_position;
// Step 3: Calculate control output using proportional gain
output = Kp * error;
// Step 4: Limit control output to max/min values if (output > max_val) output = max_val;
else if (output < min_val) output = min_val;
// Step 5: Control motor based on output if (output > 0)
run_motor_clockwise(output);
else if (output < 0)
run_motor_counterclockwise(-output);
else stop_motor();
}
```

PID Controller implementation in C

```
float pid_compute(PIDController *pid, float error) {
    uint32_t timestamp_now = micros();
    float Ts = (timestamp_now - pid->timestamp_prev) * 1e-6f;
    // Handle overflow or invalid sampling time
    if (Ts <= 0.0f || Ts > 0.5f) Ts = 1e-3f;
    // Proportional term
    float proportional = pid->P * error;
    // Trapezoidal integration for integral term
    float integral = pid->integral_prev + pid->I * Ts * 0.5f * (error + pid->error_prev);
    integral = constrain(integral, -pid->limit, pid->limit);

    // Derivative term (raw)
    float raw_derivative = (error - pid->error_prev) / Ts;

    // Filtered derivative using low-pass filter
    float alpha = pid->D_filter_alpha;  // Typically 0.8 to 0.95
    float filtered_derivative = alpha * pid->derivative_prev + (1.0f - alpha) * raw_derivative;

    // Derivative contribution
    float derivative = pid->D * filtered_derivative;

    // Combine all terms and apply output limit
```

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 9, September 2025

DOI: 10.17148/IJIREEICE.2025.13912

```
    float output = proportional + integral + derivative;
    output = constrain(output, -pid->limit, pid->limit);

    // Output ramp limiting
    float output_rate = (output - pid->output_prev) / Ts;
    if (output_rate > pid->output_ramp)
        output = pid->output_prev + pid->output_ramp * Ts;
    else if (output_rate < -pid->output_ramp)
        output = pid->output_prev - pid->output_ramp * Ts;

    // Save state for next iteration
    pid->integral_prev = integral;
    pid->output_prev = output;
    pid->error_prev = error;
    pid->derivative_prev = filtered_derivative;
    pid->timestamp_prev = timestamp_now;

    return output;
}
```