

Discrete-Time PID Control

1.0 Learning Outcomes.....	1
2.0 Introduction.....	1
3.0 Discrete-Time PID Difference Equations.....	2
4.0 Practical Considerations.....	7
5.0 Summary.....	11

1.0 Learning Outcomes

After studying this learning guide, you should be able to demonstrate the Learning Outcomes listed in Table 1. On completion of the learning guide, revisit Table 1 and confirm whether each Learning Outcome has been achieved.

Table 1: *Learning Outcomes from the Discrete-Time PID Control learning guide*

ID	Learning Outcome	Achieved
1.	Explain how a discrete-time PID controller can be expressed using difference equations for implementation on a microcontroller.	☐
2.	Explain the main practical considerations associated with implementing PID control, including sampling, measurement filtering, actuator saturation, anti-windup, derivative kick, numerical resolution and controller start-up.	☐

2.0 Introduction

A discrete-time PID (proportional, integral and derivative) controller is widely used in embedded and digital control systems to regulate a measured output in real time. The controller operates at a fixed sampling interval, T_s , and calculates a new control signal at each sample using the current and previous error values.

In practical implementations, the three-term PID (i.e., proportional, integral and derivative) actions are expressed using difference equations so they can be executed directly on a microcontroller, as illustrated in Figure 1. The integral term can be implemented using the Trapezoidal Rule, while the derivative term can be approximated using Backward Euler. These methods are well suited to embedded control because they require only a small number of stored values and relatively little computation, with their practical implementation explored in the following section.

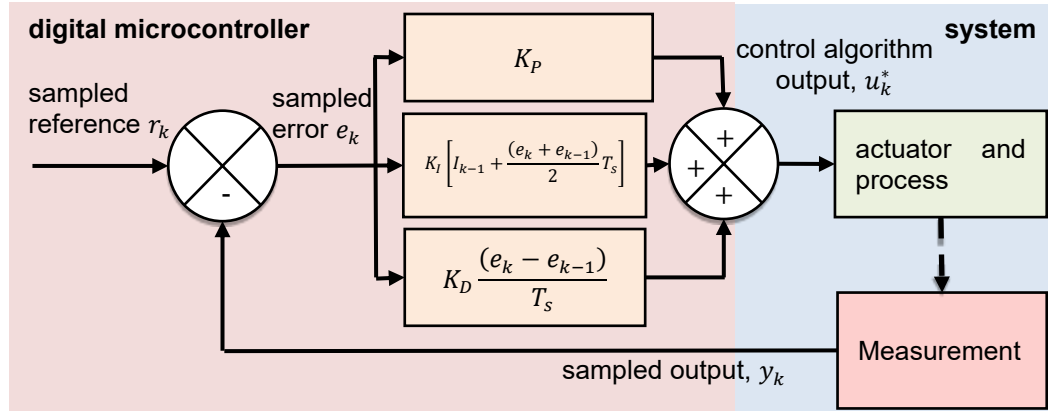


Figure 1: Discrete-Time *Proportional, Integral and Derivative (PID) Control*

A key practical consideration is selecting an appropriate sampling interval, T_s , and tuning the PID gains to achieve the required system response. Other important considerations include measurement filtering, particularly where derivative action may amplify high-frequency noise; actuator saturation and anti-windup, to prevent excessive integral accumulation; signal scaling and limits, such as voltage, PWM or current constraints; and the numerical implementation of the controller on the chosen microcontroller. Analytical or simulation-based methods may be used as part of the tuning process, but the final controller should be evaluated and refined using the measured response of the physical system. Together, these considerations help ensure that the discrete-time PID controller performs reliably when applied to the physical system.

3.0 Discrete-Time PID Difference Equations

For implementation on a microcontroller, the PID controller can be expressed in a discrete-time difference equation form. This allows the control signal to be calculated at each sampling instant using the current and previously stored error values.

Proportional (P) Control

The proportional (P) controller calculates the control output from the current error between the desired reference value and the measured system output. At sampling instant, k , the error is defined as:

$$e_k = r_k - y_k \quad (1)$$

where r_k denotes the desired reference value and y_k denotes the measured system output. The proportional contribution to the controller output, $u_{P,k}$, is then given by:

$$u_{P,k} = K_P e_k \quad (2)$$

where K_P is the proportional gain. The proportional action therefore responds directly to the magnitude of the current error. A larger error produces a larger

control action, providing an immediate response that drives the measured output towards the desired reference, as illustrated in Figure 2.

The value of K_p influences how strongly the controller responds to the error and, consequently, the resulting closed-loop system response. A value that is too low may result in a slow response and persistent steady-state error, whereas a value that is too high may produce excessive overshoot, oscillation or, in some systems, instability.

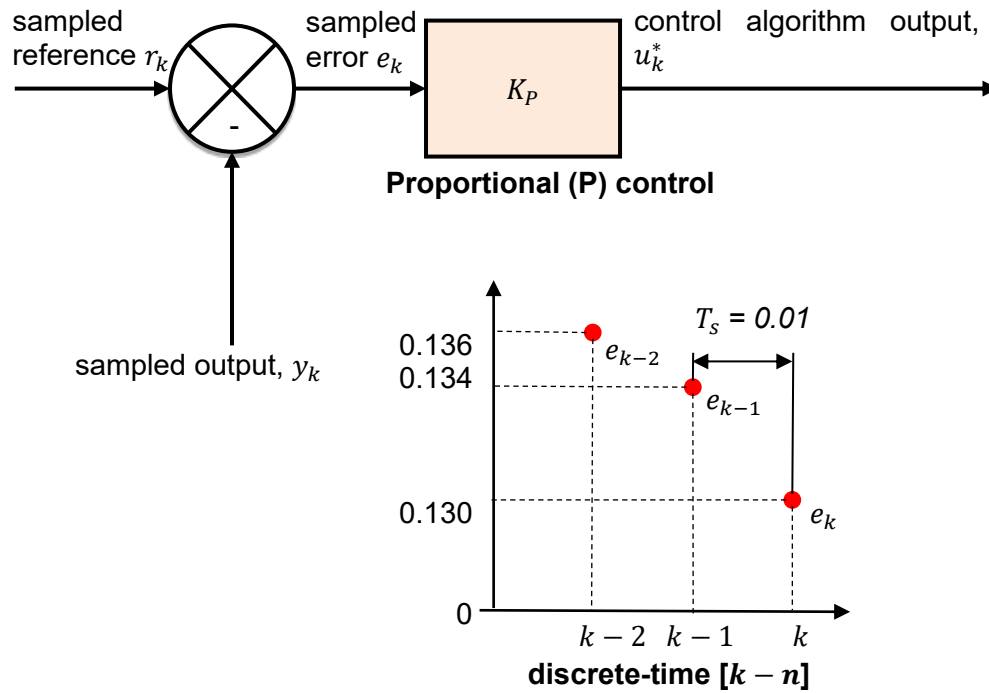


Figure 2: Proportional (P) Control

Integral (I) Control

The integral action accounts for the accumulation of error over time. In a discrete-time implementation, this accumulated error can be approximated using the Trapezoidal Rule, which calculates the area between consecutive error samples, as illustrated in Figure 3.

For the current sampling interval, the **incremental** area is given by:

$$\Delta I_k = \frac{(e_k + e_{k-1})}{2} T_s \quad (3)$$

where e_k and e_{k-1} denote the error values at the current and previous sampling instants, respectively, and T_s denotes the sampling period. Equation (3) determines the mean of the two error values and multiplies it by the width of the sampling interval, T_s .

At each sampling instant, this new incremental area is added to the **accumulated** integral value from the previous sampling instant:

$$I_k = I_{k-1} + \frac{(e_k + e_{k-1})}{2} T_s \quad (4)$$

where I_k represents the accumulated integral of the error and I_{k-1} is the previously accumulated integral of the error. The addition in Equation (4) performs the summation required to accumulate the error over time. In a practical implementation, I_{k-1} is stored in memory and fed back into the calculation at the next sampling instant.

This recursive form means that the complete history of the error does not need to be stored or recalculated. Only the previous accumulated integral value, the current error and the previous error are required, making the method well suited to real-time implementation on a microcontroller.

The integral contribution to the controller output is then calculated as:

$$u_{I,k} = K_I I_k \quad (5)$$

where K_I is the integral gain. As error persists over successive sampling intervals, the accumulated integral value increases, allowing the integral action to progressively increase its contribution to the control signal. This is particularly useful for reducing or eliminating persistent steady-state error, as illustrated in Figure 3.

After each controller update, the newly calculated value, I_k is stored and used as I_{k-1} , during the next sampling interval.

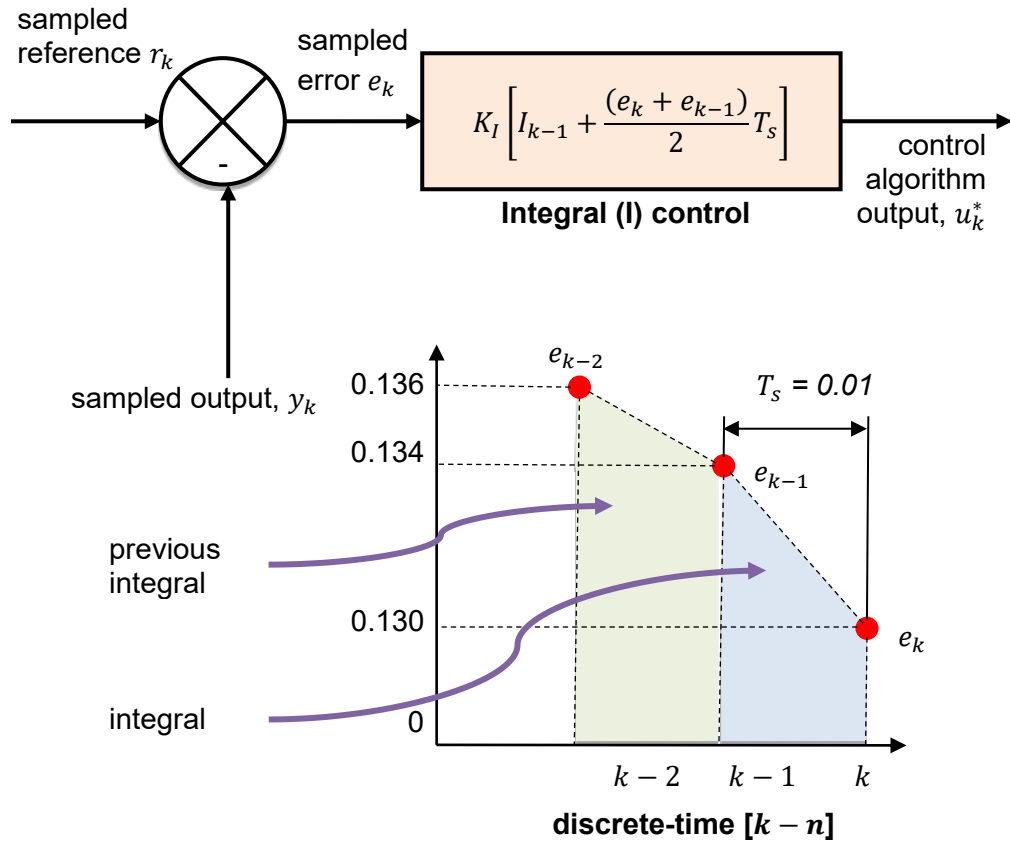


Figure 3: Integral (I) Control

Derivative (D) Control

The derivative action responds to the rate at which the error is changing. In a discrete-time implementation, this rate of change can be approximated using the backward-difference approximation (also referred to as the Backward Euler approximation), obtained using the current and previous error samples over the sampling period:

$$\frac{de}{dt} \approx \frac{(e_k - e_{k-1})}{T_s} \quad (6)$$

Using this approximation, the derivative contribution to the controller output is given by:

$$u_{D,k} = K_D \frac{(e_k - e_{k-1})}{T_s} \quad (7)$$

where K_D is the derivative gain, see Figure 4.

The derivative action therefore responds to how quickly the error is changing. A rapid change in error produces a larger derivative contribution, whereas little or no change in error produces a small or zero derivative contribution.

As differentiation is particularly sensitive to high-frequency measurement noise, small fluctuations in the measured signal can produce relatively large changes

in the derivative output. In practical implementations, filtering is therefore commonly applied to reduce the influence of high-frequency noise. The derivative gain, K_D , should also be tuned carefully, as an excessively large value can amplify noise and produce excessive or rapidly changing control effort, whereas a value that is too small may provide little useful derivative action.

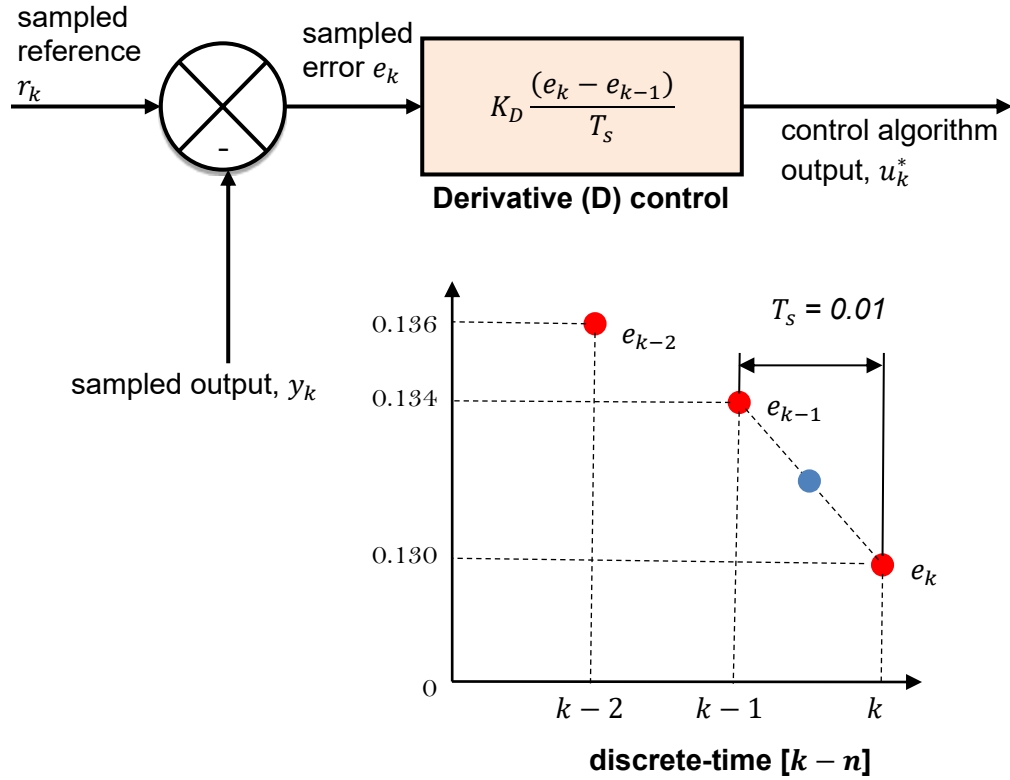


Figure 4: Derivative (D) Control

Proportional, Integral and Derivative (PID) Control

The three-term discrete-time PID controller combines the proportional, integral and derivative contributions to form the overall control signal, as illustrated in Figure 5, with this represented by:

$$u_k^* = u_{P,k} + u_{I,k} + u_{D,k} \quad (8)$$

Substituting the individual proportional, integral and derivative contributions gives:

$$u_k^* = K_P e_k + K_I \left[I_{k-1} + \frac{(e_k + e_{k-1})}{2} T_s \right] + K_D \frac{(e_k - e_{k-1})}{T_s} \quad (9)$$

In practice, these gains are tuned to achieve the required system response while also considering the limitations of the physical implementation. These may include actuator saturation, PWM resolution, sensor resolution and noise, the selected sampling period, and the numerical precision and computational

capability of the microcontroller. Appropriate filtering and anti-windup may also be required to ensure reliable controller operation.

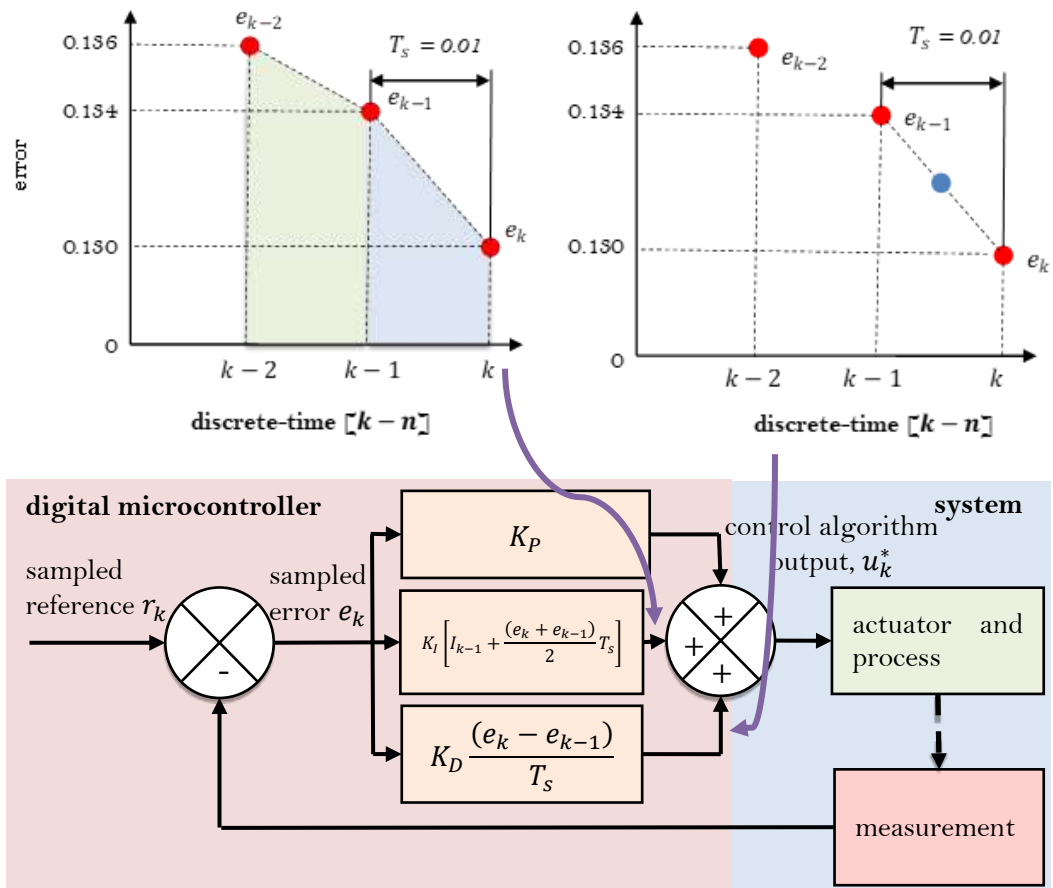


Figure 5: Proportional, Integral and Derivative (PID) Control

4.0 Practical Considerations

Implementing a discrete-time PID controller on a physical system requires more than selecting suitable controller gains. Factors such as the sampling period, measurement quality, actuator limitations, numerical implementation and controller protection mechanisms can all influence the resulting closed-loop performance.

Table 2 summarises the principal practical considerations and explains why each is important during implementation. These topics are briefly introduced in this learning guide, with more detailed coverage provided in later learning guides.

Table 2: Practical Considerations and Why They Matter

Consideration	Why it matters
Sampling period, T_s	Determines how frequently measurements and control actions are updated.
Gain tuning	Sets the balance between response speed, error removal, damping and control effort.
Measurement filtering	Reduces the influence of sensor noise, especially on derivative action.
Actuator saturation	Represents physical limits such as voltage, current and PWM range.
Anti-windup	Prevents the integral term from accumulating excessively during saturation.
Derivative kick	Can produce a large derivative action following a sudden change in the reference.
Initial conditions / bumpless transfer	Reduces abrupt control changes when starting or switching control modes.
Quantisation and numerical precision	Accounts for finite sensor, PWM and processor resolution.
Execution time	Ensures all calculations complete within each sample period.
Verification	Confirms the measured response and control signal satisfy the requirements.

Sampling Period

The PID controller is evaluated at discrete sampling instants separated by the sampling period, T_s . The choice of T_s determines how frequently the controller reads the measured output, calculates the error and updates the control signal. A sampling period that is too large may fail to capture important system dynamics and can reduce control performance or lead to instability. An unnecessarily small sampling period increases computational demand and, particularly when derivative action is used, can increase sensitivity to measurement noise and quantisation. The sampling period should therefore be sufficiently small to represent the important dynamics of the physical system while allowing the complete control algorithm to execute reliably before the next sample.

Controller Gain Tuning

The PID gains, K_P , K_I and K_D , influence the behaviour of the controller and must be selected for the physical system being controlled. Initial values may be obtained through manual tuning, mathematical modelling, simulation or optimisation, but the final gains should be evaluated using measured results from the physical system.

Changing one gain can influence several features of the response. Controller tuning should therefore consider the complete closed-loop response, including rise time, overshoot, steady-state error, disturbance rejection and the required actuator effort.

Measurement Noise and Filtering

Physical measurements often contain noise. This is especially important when derivative action is used because differentiation responds strongly to rapid changes in the measured signal. Small high-frequency variations can therefore produce relatively large derivative contributions and a rapidly changing actuator command.

A low-pass filter may be applied to the measured signal, or filtering can be incorporated directly into the derivative action. Filtering should be selected carefully because excessive filtering introduces additional delay and can slow the apparent measured response.

Actuator Saturation

Physical actuators cannot produce unlimited control effort. For example, a motor may be constrained by the available supply voltage, current capability or PWM range. The PID controller may calculate an unconstrained control signal, u_k^* , that exceeds these physical limits. The command applied to the actuator, u_k , must therefore be limited to the range that can be produced by the hardware. This can be represented as:

$$u_k = \text{sat}(u_k^*) \quad (10)$$

where u_k^* is the control signal calculated by the PID algorithm and u_k is the control signal applied to the actuator after the saturation limits have been imposed.

The upper and lower saturation limits should therefore be selected to represent the physical operating range of the actuator. When the calculated control signal remains within these limits, $u_k = u_k^*$. When the calculated control signal exceeds either limit, the applied actuator command is restricted to the corresponding maximum or minimum value.

When integral action is present, prolonged actuator saturation can also lead to integral windup, as discussed in the following section.

Integral Windup and Anti-Windup

Actuator saturation creates an important problem for controllers containing integral action. If the actuator is saturated while a persistent error remains, the integral state can continue to accumulate even though the actuator cannot provide any additional control effort.

This behaviour is known as integral windup. When the system later approaches the reference value, the excessively accumulated integral action must first unwind before the controller returns to its normal operating region. The result can be increased overshoot and a long recovery time.

Anti-windup methods modify the behaviour of the integrator when saturation occurs so that the internal controller state remains consistent with the control effort that can be applied.

Derivative Kick

If derivative action is calculated directly from the error, a sudden change in the reference can create a large change in error between two samples. The derivative term can consequently produce a large, short-duration control action, commonly referred to as derivative kick.

One practical solution is to apply derivative action to the measured output rather than directly to the error. A step change in the reference then does not directly enter the derivative calculation.

Initial Conditions and Controller Start-Up

At controller start-up, stored quantities such as I_{k-1} and e_{k-1} must be initialised. For many simple applications these values can initially be set to zero. However, if the controller is enabled while the system is already operating, inappropriate initial values can produce an abrupt change in the control signal.

The initial integral state should therefore be considered whenever the controller is started, reset or re-enabled.

Bumpless Transfer and Tracking

Some control systems allow operation to switch between manual and automatic PID control, or between different controllers. When switching control modes, the newly enabled controller may initially calculate a control signal that differs from the signal currently being applied to the actuator. This can cause a sudden change, or bump, in the actuator command.

Bumpless transfer is used to minimise this sudden change by ensuring that the PID controller output closely matches the existing actuator command at the instant of switching. This is typically achieved using tracking, where the controller's internal state, particularly the integral term, is adjusted to follow the applied control signal before automatic control is enabled.

PWM and Actuator Resolution

A digital controller ultimately produces a command through hardware with finite resolution. For example, an 8-bit PWM output provides:

$$2^8 = 256 \text{ possible command values}$$

The calculated PID output must therefore be mapped onto one of the available PWM levels. If the required change in control effort is smaller than the PWM resolution, the physical actuator command may remain unchanged even though the calculated control output has changed. PWM scaling and saturation should be designed together so that the numerical controller output corresponds correctly to the physical actuator range.

Sensor Resolution and Quantisation

The measured output is also represented with finite resolution. Encoders, analogue-to-digital converters (ADCs) and other digital sensors cannot represent infinitely small changes in the physical variable. Small physical changes may therefore produce no change in the measured digital value, while measurements near a quantisation boundary may alternate between adjacent levels.

Quantisation can be particularly noticeable in derivative calculations because the derivative is based on the change between successive samples. Sensor resolution, filtering and sampling should therefore be considered together.

Computational Execution Time

All sensing, filtering, control calculations, communications and actuator updates required by the controller must complete within the selected sampling period. A useful practical requirement is:

$$T_{Execution} < T_s \quad (11)$$

If the execution time approaches or exceeds the sampling period, controller updates may be delayed or missed. This can change the effective timing of the control algorithm. Execution time should therefore be checked on the target hardware rather than assumed from desktop simulation performance. Ideally, the controller should also execute at consistent intervals, since variation in the actual sampling interval can introduce sampling jitter and affect controller behaviour.

Practical Verification

After implementation, the controller should be evaluated using measured data from the physical system. Verification should consider both the system response and the control signal. A satisfactory output response can sometimes hide undesirable controller behaviour such as repeated actuator saturation, excessive integral accumulation or rapidly changing actuator commands. The following checks should therefore be carried out during practical verification:

- Check the measured output against the engineering requirements.
- Inspect the controller output and confirm that actuator limits are respected.
- Check whether the controller spends significant time in saturation.
- Observe the integral state and confirm that windup is controlled.
- Inspect measurement noise and the derivative contribution.
- Confirm the chosen sampling period is maintained on the target hardware.
- Refine the controller gains, filtering and protection mechanisms where required.

5.0 Summary

This learning guide introduced proportional, integral and derivative control in discrete-time form for real-time execution on a microcontroller. The proportional action was calculated directly from the current error, while the integral and

derivative actions were implemented using the Trapezoidal Rule and backward-difference approximation, respectively.

Practical PID implementation requires consideration of factors beyond the controller gains. The sampling period, measurement noise, actuator saturation, integral windup, derivative kick, controller initialisation, numerical resolution and execution time can all influence the behaviour of the physical closed-loop system. The final controller should therefore be verified using measured data from the physical system.

Return to the Learning Outcomes at the beginning of this learning guide and consider whether you can now confidently demonstrate each one.