PID Control for Robotics

Aranya Saha
MSc (Ongoing) & BSc in EEE, BUET
ML Engineer, ACI Limited

Institute of Robotics and Automation
Bangladesh University of Engineering and Technology

June 25, 2025

What We'll Learn Today

What is Control? I

Think About Daily Life

- When you drive a car, you control the steering wheel
- When you adjust room temperature, you control the AC
- When you ride a bicycle, you control your balance

Control in Simple Terms

Control = Making something behave the way you want it to
You have a goal (where you want to go) and you take actions (steering, accelerating) to reach that goal.

What is Control? II

Human Control:

- You see with your eyes
- Your brain decides what to do
- Your hands/feet take action
- You check if it worked

Robot Control:

- Sensors "see" the environment
- Computer brain decides
- Motors take action
- Sensors check if it worked

Key Point

Both humans and robots use feedback - they look at the result and adjust their actions accordingly.

Why Control Matters in Robotics?

- **Robots are not perfect:**
  - Motors don't turn exactly as commanded
  - Wind pushes drones off course
  - Wheels slip on the ground
- **Environment changes:**
  - Different object weights
  - Uphill vs downhill motion
  - Temperature affects motors
- **We need precision:**
  - Surgery robots must be accurate
  - Factory robots repeat tasks perfectly
  - Self-driving cars stay in lanes

Without Control

A robot would run **open-loop** – like driving with eyes closed!

The Concept of Error

What is Error?

Error = Where you want to be − Where you actually are

Everyday Example: Room Temperature

- You want the room at 22°C (this is your setpoint)
- Room is currently 25°C (this is the current value)
- Error = 22°C − 25°C = **-3°C**
- Negative error means it's too hot!

Interpreting Positive vs Negative Error

Positive Error:

- Want to go faster
- Want to go higher
- Want to turn more

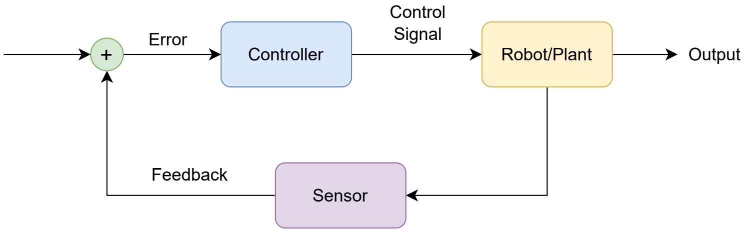
Negative Error:

- Want to go slower
- Want to go lower
- Want to turn less

Tip

Understanding the **sign** of the error is crucial for deciding the control action.

Feedback Loop – The Heart of Control I



Feedback Loop – The Heart of Control II

1. **Setpoint:** Where you want the robot to be
2. **Sensor:** Measures where the robot actually is
3. **Error:** Calculate the difference
4. **Controller:** Decides what action to take
5. **Robot:** Performs the action
6. **Repeat:** Check again and adjust

Remember

This loop runs continuously – many times per second!

Bang-Bang Control – Simplicity in Action

What is Bang-Bang Control?

A basic control strategy where the system switches **fully ON** or **fully OFF**—no in-between. It's called "bang-bang" because it abruptly jumps between extremes like a light switch.

How it works

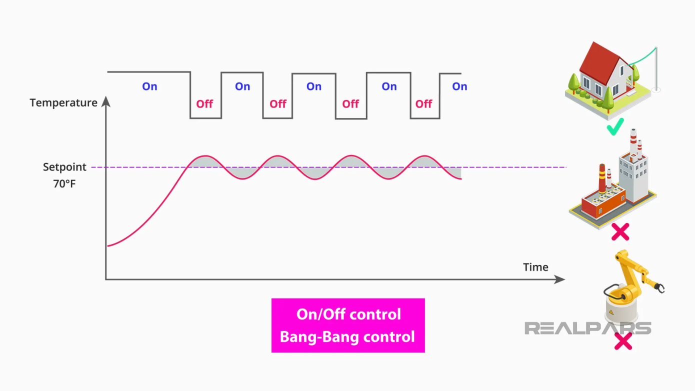
- If error $> 0 \rightarrow$ turn actuator ON
- If error $< 0 \rightarrow$ turn actuator OFF
- No proportional response
- Binary decision making

Real-life Example

Thermostat controlling a heater:

- Too cold $\rightarrow$ heater ON
- Warm enough $\rightarrow$ heater OFF
- Results in temperature oscillation
- Simple but not smooth

Bang-Bang Control – Example and Effects



Pros: Simple, fast reaction

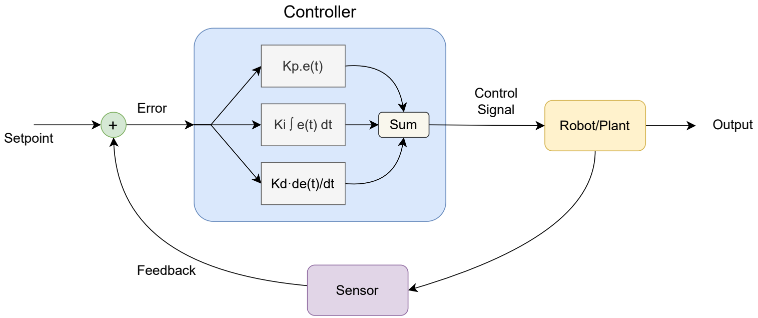
Cons: Causes oscillation

Meet PID – Your Robot's Brain

What is PID?

Stands for Proportional + Integral + Derivative

It's like having three different "personalities" working together to control your robot.



Breaking Down PID Components

P – Present "How big is the error **right now**?"

I – Past "How **long** have we been wrong?"

D – Future "How **fast** is the error changing?"

P Control

I Control

D Control

Now, let's visualize PID control of a line following car

Now, let's visualize PID control of a self-balancing car

PID – Driving Analogy

Analogy Think of PID like a skilled driver:

- **P:** Steers based on how far off center they are
- **I:** Corrects consistent drift (like wind)
- **D:** Slows down steering near the target

Tip

A good PID controller balances all three actions to stay smooth and accurate.

The PID Equation (Don't Panic!)

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

Let's break this down in simple terms:

Control Output = P term + I term + D term

$$u(t) = K_p \cdot e(t) + K_i \sum e + K_d \cdot \Delta e$$

Where:

- $u(t)$ = What we tell the robot to do (e.g., motor speed, steering angle)
- $e(t)$ = Current error (setpoint – actual)
- K_p, K_i, K_d = Tuning knobs that adjust behavior

Proportional Control – The Immediate Responder

P Control in Simple Terms

Proportional means "in proportion to the error"

Big error $\Rightarrow$ Big response   Small error $\Rightarrow$ Small response

Driving Example

- If you're way off the road center $\rightarrow$ Turn the wheel a lot
- If you're slightly off center $\rightarrow$ Turn the wheel a little
- If you're perfectly centered $\rightarrow$ Don't turn at all

Proportional Control – The Immediate Responder

Mathematical Form: $u_p(t) = K_p \times e(t)$

- K_p is the proportional gain – it's like the "sensitivity" knob
- Higher K_p = More aggressive response
- Lower K_p = Gentler response

Remember

P control reacts instantly to error – but doesn't care about the past or future!

P Control Behavior (1/2)

What P Control Does Well:

- ✓ Fast response to large errors
- ✓ Simple to understand
- ✓ Stable for most systems
- ✓ Good starting point

Problems with Only P:

- ✗ Never reaches exact target
- ✗ Always has some steady-state error
- ✗ Can oscillate if gain too high
- ✗ Affected by disturbances

P Control Behavior (1/2)

Robot Arm Example You want the arm at position 90°, but it stops at 87°.

- Error $= 90^\circ - 87^\circ = 3^\circ$
- P control gives small signal (because error is small)
- Small signal might not be enough to overcome friction
- Arm stays at 87° forever!

Key Insight

P control alone is like a person who gets lazier as they get closer to their goal!

Tuning the P Gain I

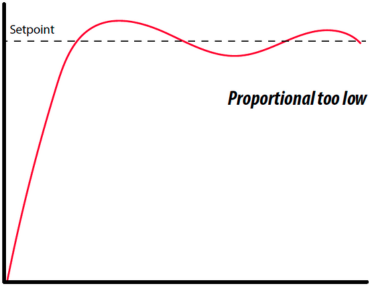
Table 1: Effects of Different K_p Values

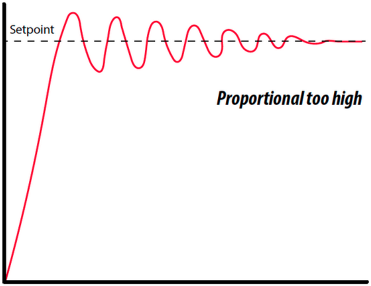
K_p **Value**	**Response Speed**	**Overshoot**	**Stability**
Too Low	Very Slow	None	Stable
Just Right	Fast	Minimal	Stable
Too High	Very Fast	Large	Oscillates
Way Too High	Unstable	Extreme	Unstable

Goal

We want a K_p that gives fast response, little overshoot, and good stability.

Tuning the P Gain II





Tip

Tune K_p by observing response curves. Aim for minimal overshoot and fast settling.

Integral Control - The Persistent Helper I

I Control in Simple Terms

Integral means "sum up all the errors over time"
It keeps track of how long you've been wrong and tries to fix it.

Shower Temperature Analogy

You set the shower to "warm" but it stays lukewarm:

- P control: "It's a little cold, adjust a little"
- I control: "It's been cold for 30 seconds! Time for bigger adjustment!"
- I control accumulates the "coldness" over time

Integral Control - The Persistent Helper II

Mathematical form: $u_i(t) = K_i \times \int_0^t e(\tau)d\tau$

In digital systems: $u_i[n] = K_i \times \sum_{k=0}^{n} e[k]$

- K_i is the integral gain
- Higher K_i = Faster elimination of steady-state error
- But too high can cause instability!

Why We Need I Control – I

Problems I Control Solves:

- ✓ Eliminates steady-state error
- ✓ Handles constant disturbances
- ✓ Adapts to system changes
- ✓ Improves accuracy

Potential Issues:

- ✗ Can cause overshoot
- ✗ Slower to respond initially
- ✗ Can make system oscillate
- ✗ Sensitive to noise

Why We Need I Control – II

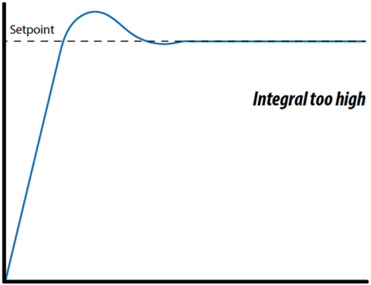
Robot on a Hill A mobile robot trying to maintain 1 m/s speed:

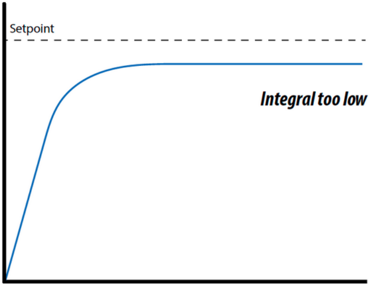
- On flat ground: P control works fine
- Going uphill: Gravity slows it down consistently
- P control gives same small signal for small error
- I control notices the persistent slowness
- I control adds extra power to overcome gravity!

Key Point

I control has "memory" - it remembers past errors and builds up its response.

I Control Behavior Examples - I





Integral Windup

If error stays large for too long, I term can become huge! This is called "integral windup".

Derivative Control - The Predictor - I

D Control in Simple Terms

Derivative means "rate of change" or "how fast is the error changing"
It looks at the trend and tries to predict the future.

Car Braking Analogy

You're approaching a red light:

- P control: "I'm far from the stop line, keep going fast"
- D control: "Wait! I'm approaching fast, better start braking now!"
- D control prevents overshooting the stop line

Derivative Control - The Predictor - II

Mathematical form: $u_d(t) = K_d \times \frac{de(t)}{dt}$

In digital systems: $u_d[n] = K_d \times (e[n] - e[n-1])$

- K_d is the derivative gain
- D control acts on the rate of change of error
- It provides "damping" to prevent overshoot

Understanding D Control Better

When D Control Helps:

- ✓ Reduces overshoot
- ✓ Improves stability
- ✓ Faster settling time
- ✓ Smoother response

D Control Challenges:

- ✗ Very sensitive to noise
- ✗ Can amplify high-frequency signals
- ✗ Harder to tune
- ✗ Sometimes not needed

Robot Arm Positioning

- **Without D:** Arm swings past target, then back, then past again...
- **With D:** As arm approaches target, D control says "Slow down!"
- **Result:** Smooth arrival with no overshoot

Important

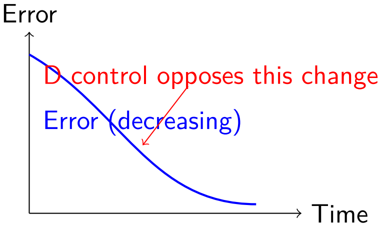
D control is like having anticipation – it acts on where the system is heading, not just where it is.

D Control in Action

How D Control Changes Based on Error Trends

Table 2: Robot Position Control, $K_d = 0.5$

Time	Error	Error Change	D Output	Meaning
1s	$+10^\circ$	–	0	Starting
2s	$+7^\circ$	-3°	-1.5	Error decreasing, ease up
3s	$+3^\circ$	-4°	-2.0	Getting closer faster, slow down
4s	$+1^\circ$	-2°	-1.0	Almost there, gentle approach
5s	0°	-1°	-0.5	Reached target, prevent overshoot



PID: The Dream Team

The Complete PID Controller:

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

P Component

The Responder

- Acts on current error
- Provides main driving force
- Fast response

I Component

The Perfectionist

- Eliminates steady error
- Has memory
- Ensures accuracy

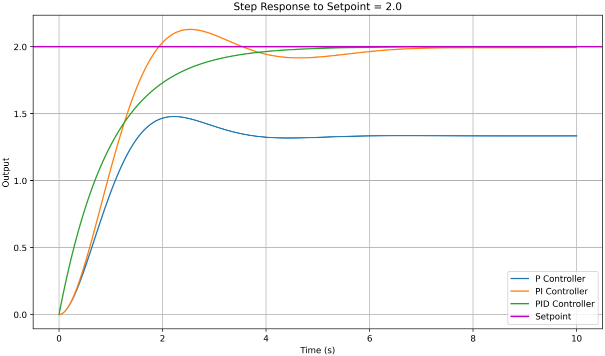
D Component

The Predictor

- Prevents overshoot
- Provides damping
- Smooths response

Each component has a job, and together they create a robust, accurate, and stable control system!

PID Controller Response Comparison



Best of All Worlds

PID combines the speed of P, the accuracy of I, and the stability of D control!

The Art of PID Tuning

What is PID Tuning?

Finding the right values for K_p, K_i, and K_d to make your robot behave the way you want.

Tuning Goals:

- ✓ Fast response
- ✓ No overshoot
- ✓ No steady-state error
- ✓ Stable operation
- ✓ Good disturbance rejection

Reality Check:

- You can't have everything perfect
- Trade-offs are necessary
- Different applications need different priorities
- Tuning takes practice!

Important: There's no "magic formula" for tuning. Every robot and application is different. But there are systematic approaches to help you!

Simple Tuning Method: Start with P

Step-by-Step Beginner Approach

Start Simple and Build Up

1. **Set** $K_i = 0$ **and** $K_d = 0$ (P controller only)
2. **Increase** K_p **gradually:**
   - Start with small value (like 0.1)
   - Increase until system responds quickly
   - Stop when it starts to oscillate
   - Back off a bit for safety
3. **Add I if needed:**
   - If there's steady-state error, add small K_i
   - Start with $K_i = K_p/10$
   - Increase slowly until error disappears
4. **Add D if needed:**
   - If there's overshoot, add small K_d
   - Start with $K_d = K_p/4$
   - Adjust until overshoot is acceptable

Ziegler-Nichols Tuning Method (Self Study)

A More Systematic Approach

Developed by engineers Ziegler and Nichols in 1942, still used today!

Steps:

1. Set $K_i = 0$ and $K_d = 0$
2. Increase K_p until system just starts to oscillate continuously
3. Note this critical gain K_c and oscillation period T_c
4. Use the Ziegler-Nichols table to calculate final gains

Ziegler-Nichols Tuning Method (Self Study)

Table 3: Ziegler-Nichols Tuning Rules

Controller	K_p	K_i	K_d
P	$0.5K_c$	0	0
PI	$0.45K_c$	$1.2K_p/T_c$	0
PID	$0.6K_c$	$2K_p/T_c$	$K_pT_c/8$

Note: This gives you a good starting point, but you'll likely need to fine-tune from there!

Parameter Effects on System Behavior

Parameter	Rise Time	Overshoot	Settling Time	Steady Error	Stability
Increase K_p	Faster	Increases	Small Change	Decreases	Degrades
Increase K_i	Faster	Increases	Increases	Eliminates	Degrades
Increase K_d	Small Change	Decreases	Decreases	No Change	Improves

Practical Tips

- **Too much P:** System oscillates around target
- **Too much I:** System overshoots and takes long to settle
- **Too much D:** System becomes very sensitive to noise

Remember

Good tuning is often about finding the right balance between competing requirements.

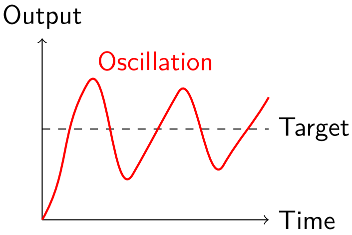
Problem 1: Oscillation

Symptoms

- Robot "hunts" around the target
- Continuous back-and-forth motion
- Never settles to a steady value
- May get worse over time

Likely Causes

- K_p too high
- K_i too high
- K_d too low (not enough damping)
- Delays in the system



Problem 1: Oscillation

Solutions

- **Reduce K_p:** Decrease proportional gain by 20-50%
- **Reduce K_i:** Lower integral gain or set to zero temporarily
- **Increase K_d:** Add derivative action for damping
- **Check for delays:** Ensure sensors and actuators respond quickly

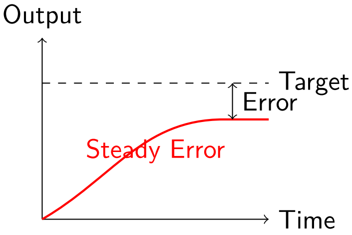
Problem 2: Steady-State Error

Symptoms

- System reaches a steady value
- But it's not the target value
- Error remains constant
- System seems "stuck" near target

Likely Causes

- No integral action ($K_i = 0$)
- K_i too small
- Friction or other constant disturbances
- Actuator saturation



Problem 2: Steady-State Error

Solutions

- **Add integral action:** Set K_i to a small positive value.
- **Increase K_i:** Gradually increase until the error disappears in a reasonable time.
- **Check actuator limits:** Ensure the motor/servo can provide enough force to overcome disturbances.
- **Consider feedforward:** If the disturbance is predictable, a feedforward term can cancel it out.

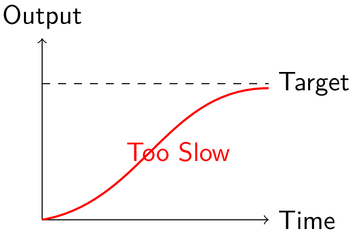
Problem 3: Slow Response

Symptoms

- System eventually reaches target
- But takes too long to get there
- "Sluggish" or "lazy" behavior
- Works fine but not fast enough

Likely Causes

- K_p too low
- All gains too conservative
- Actuator too weak
- Heavy load or high friction



Problem 3: Slow Response

Solutions

- **Increase K_p:** Higher proportional gain for faster response
- **Check actuator:** Ensure motor/servo has enough power
- **Reduce load:** If possible, reduce friction or weight
- **Verify setpoint:** Make sure target is achievable

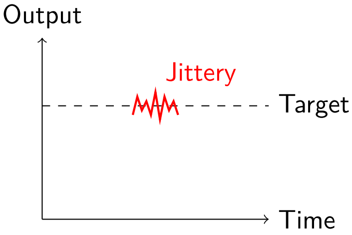
Problem 4: Noisy/Jittery Behavior

Symptoms

- System reaches target approximately
- But output is constantly jittering
- Small rapid movements around target
- Motor/servo makes noise

Likely Causes

- K_d too high
- Noisy sensor readings
- High-frequency disturbances
- Poor sensor resolution



Problem 4: Noisy/Jittery Behavior

Solutions

- **Reduce K_d:** Lower or eliminate derivative gain
- **Filter sensors:** Add low-pass filter to sensor readings
- **Increase deadband:** Don't react to very small errors
- **Check sensor quality:** Use higher resolution or better sensors

Sensor Filtering and Noise Handling

Why Filtering Matters?

Real sensors are noisy! A position sensor might read: 45.1°, 44.9°, 45.2°, 44.8°, 45.0°...
The derivative of noisy signals is VERY noisy, making D control problematic.

Filter Trade-offs

- More filtering = Less noise but slower response
- Less filtering = Faster response but more noise
- Choose based on your application needs

When PID Might Not Be Enough

Highly Nonlinear Systems:

- Walking robots (complex dynamics)
- Flying robots in turbulent conditions
- Systems with significant dead zones

Systems with Constraints:

- Maximum motor torques
- Joint angle limits
- Obstacle avoidance requirements

Multi-Variable Systems:

- Quadcopter (4 motors, 6 degrees of freedom)
- Robotic hands (many fingers, complex coordination)
- Mobile manipulators (driving + arm control)

Time-Varying Systems:

- Robots with changing payloads
- Systems with wear and aging
- Environmental changes (temperature, humidity)

But Remember Even in these cases, PID concepts are still fundamental! Advanced controllers often build upon PID principles.

Advanced PID Techniques – Self Study

PID is Just the Beginning!

While PID is powerful, real-world robotics often needs more advanced techniques.

Adaptive PID

- Gains change based on conditions
- Robot learns optimal parameters
- Handles varying loads/environments
- Example: Arm adjusts to different payloads

Feedforward Control

- Predicts what control is needed
- Combined with PID feedback
- Faster response to known disturbances
- Example: Compensating for gravity

Advanced PID Techniques – Self Study

Cascade Control

- Multiple PID loops nested together
- Inner loop: Motor current control
- Outer loop: Position control
- Better disturbance rejection

Model Predictive Control

- Uses robot model to predict future
- Optimizes control over time horizon
- Handles constraints explicitly
- More computation but better performance

Learning Path

Master PID first! It's the foundation for understanding all other control methods.

What We've Learned Today

Key Concepts Covered

Fundamental Ideas:

- ✓ What control is
- ✓ Why it matters
- ✓ Error and feedback concepts
- ✓ The PID control algorithm

Mathematical Understanding:

- ✓ PID equation breakdown
- ✓ Effects of each gain
- ✓ Response characteristics
- ✓ Stability basics

You now understand one of the most powerful tools in robotics!

Thank You!

Questions?