PID Controller

PID stands for **Proportional-Integral-Derivative**, and it’s a type of control algorithm used to help systems reach a desired goal or setpoint (like keeping the temperature of a room constant).

Here’s a simple breakdown of the three components of a PID controller:

1. **Proportional (P):**

   This part looks at the **current error**, which is how far the system is from the target value. The proportional part tries to correct the error by applying a correction that’s proportional to the size of the error.

   - **Example:** If your room is 5°C cooler than desired, the system will apply a correction that's proportional to that error (say, 5 times some constant).

2. **Integral (I):**

   This part looks at the **accumulated error over time**. If there’s been a small error for a long time, the integral part will work to correct it by adding up all the past errors. This helps eliminate any steady-state errors that the proportional part might miss.

   - **Example:** If the room temperature has been slightly too low for a long time, the integral part will push harder to correct this over time.

3. **Derivative (D):**

   This part looks at the **rate of change of the error**. It anticipates how fast the error is changing and tries to adjust the correction to prevent overshooting or instability.

   - **Example:** If the temperature is rapidly increasing towards the target, the derivative part will slow down the heating to avoid going past the target temperature.

Putting it all together:

A PID controller adjusts the system's output by combining the three components:

- The **Proportional** part responds to how big the error is right now.

- The **Integral** part compensates for past errors.
- The **Derivative** part anticipates future changes.

By balancing these three components, the PID controller tries to minimize the error and make the system as stable as possible.

When tuning a PID controller, it’s helpful to understand how each of the three constants (Kp, Ki, and Kd) affects the system behavior. Here's how you can figure out which value to adjust:

1. Proportional (Kp) – Adjusting for the Size of the Error

The **proportional** term determines how much correction is applied based on the current error (the difference between your setpoint and the current value).

- **If Kp is too high:**
  - The system will respond quickly, but it might overshoot the setpoint, causing the system to oscillate back and forth around the target.
  - It might cause the system to be "too aggressive," trying to correct too much.
- **If Kp is too low:**
  - The system will respond too slowly or may never reach the setpoint. It won't be able to overcome the error, and the response will be sluggish.

To adjust Kp:

- Start with a small value and increase it gradually.
- The goal is to make the system respond quickly to error, but without causing too much overshoot or oscillation.

---

2. Integral (Ki) – Adjusting for Accumulated Error

The **integral** term accounts for the **sum of past errors**. This helps the system eliminate **steady-state errors** (errors that don’t go away with just the proportional term, like if the system stays a little off target over time).

- **If Ki is too high:**

    - The system might "overcompensate" by continually adding up past errors and trying to correct them too much. This can lead to **oscillation** or **instability**.
- **If Ki is too low:**
    - The system might not be able to eliminate small errors over time, and it might leave a **persistent error** even after the system seems stable.

To adjust Ki:

- Increase Ki slightly if the system is unable to reach the setpoint or if there's a persistent small error over time.
- Reduce Ki if the system becomes unstable or oscillates after reaching the setpoint.

---

3. Derivative (Kd) – Adjusting for Rate of Change

The **derivative** term looks at how fast the error is changing. It tries to anticipate how the error will evolve and dampens any sudden changes to prevent overshooting.

- **If Kd is too high:**
    - The system might become **overly sensitive** to small fluctuations in error, making it too slow or too "nervous," correcting even small changes too much.
- **If Kd is too low:**
    - The system might overshoot and oscillate because it's not responding to rapid changes in the error.

To adjust Kd:

- Increase Kd if the system is overshooting (going past the setpoint) or if it's oscillating.
- If the system is responding too slowly, try lowering Kd to reduce unnecessary dampening.

A Practical Process for Tuning:

1. **Start with Kp first:**
    - Set Ki and Kd to zero initially.
    - Gradually increase Kp until the system starts to respond to the error. Look for a value where the system reacts quickly without oscillating too much.
2. **Add Ki:**
    - Once Kp is set, start increasing Ki gradually.
    - If there's a persistent error (like the system never quite reaches the setpoint), increase Ki to eliminate it.
    - Be careful not to set Ki too high, as it can cause instability.
3. **Add Kd:**
    - Finally, adjust Kd to reduce oscillations and overshooting.
    - If the system is overshooting or oscillating, increase Kd to help "dampen" the response.

A Tuning Strategy to Try:

There's a common strategy called the **Ziegler-Nichols Method** that can help guide you through the process of tuning PID constants:

1. **Set Ki and Kd to 0.**
2. **Increase Kp** until the system starts oscillating.
3. **Record the value of Kp** at which the system starts oscillating. This is called the **ultimate gain** (Ku).
4. **Measure the period** of oscillation, which is the time it takes for one full cycle of oscillation (the **ultimate period**, Pu).
5. Use the following rules to calculate your PID values:
    - **Proportional (Kp):**

      Kp=0.6×KuKp = 0.6 \times Ku
    - **Integral (Ki):**

      Ki=2×Kp/PuKi = 2 \times Kp / Pu

- **Derivative (Kd):**

  Kd=Kp×Pu/8Kd = Kp \times Pu / 8

After applying these values, fine-tune them manually for the best performance.

Other Tips:

- **Fine-tuning** is key. You'll likely need to tweak the values slightly after using the Ziegler-Nichols method or your initial guesswork.
- **Monitor system behavior** carefully (e.g., through serial prints or graphs) to make sure you're not causing instability.
- For **very slow systems** (like temperature control), a high Ki value is often useful, while for **fast-reacting systems** (like motors or robots), you might rely more on the proportional and derivative components.

Example Process:

Let's say you're controlling a heater's temperature:

1. Set **Ki** and **Kd** to 0.
2. Slowly increase **Kp** until you notice the temperature starts to increase faster but doesn't oscillate too wildly.
3. If the temperature overshoots and oscillates, add a small **Kd** to dampen the response.
4. If there's still a persistent low error, slowly increase **Ki** to help the system get to the target temperature more precisely.

By adjusting these three constants gradually, you can find a balance where the system is stable, responsive, and accurate.
