**🧪 Lab Activity 1: Tuning a PID Controller for Straight-Line Motion**

**🧭 Objective**

In this lab, you will tune the **Proportional-Integral-Derivative (PID)** parameters to help the robot travel in a **straight line** for a specific distance.

**Ideal behavior**:  
If the PID parameters are perfectly tuned, the robot will drive straight toward the target distance **without veering left or right**, and stop **precisely at the desired location**.

**🔧 Setup Instructions**

1. Upload the code *5\_encoder\_PID\_Tune.py* to the robot
2. Place the robot on a flat, open surface with enough room to drive at least 0.5 meters forward.
3. Make sure the variable target\_value = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is set in the code before uploading.

**🔢 Tuning Parameters**

Locate these values in the code and adjust them during the lab:

Kp = \_\_\_ # Start with 30

Ki = \_\_\_ # Start with 1

Kd = \_\_\_ # Start with 0

You will experiment with each term **one at a time**, observing how the robot’s behavior changes.

**🔬 Part 1: Proportional Control Only (P)**

**1. Set Ki = 0 and Kd = 0. Vary Kp.**

|  |  |  |
| --- | --- | --- |
| **Kp Value** | **Behavior Observed** | **Does it go straight? Is it wobbly? Does it correct too much?** |
| 10 |  |  |
| 20 |  |  |
| 30 |  |  |
| 40 |  |  |

💬 Which value gave the straightest motion with minimal correction?

**🔬 Part 2: Add Integral Control (PI)**

**2. Set Kp = your best value, try small Ki values.**

|  |  |  |
| --- | --- | --- |
| **Ki Value** | **Behavior Observed** | **Does it fix long-term drift? Any overshooting or overcorrection?** |
| 0.5 |  |  |
| 1.0 |  |  |
| 2.0 |  |  |

💬 Does integral control help reduce drifting over time or increase it?

**🔬 Part 3: Add Derivative Control (PID)**

**3. Use your best Kp and Ki, and try adding Kd.**

|  |  |  |
| --- | --- | --- |
| **Kd Value** | **Behavior Observed** | **Does it dampen the motion? Does it reduce wiggling?** |
| 5 |  |  |
| 10 |  |  |
| 15 |  |  |

💬 Did adding Kd make the robot smoother or more sluggish?

**✅ Final Testing**

Using your best tuned values:

**🏁 Bonus Challenge (Optional)**

Try increasing the target\_value to 12,000 or 20,000.

Can your robot **maintain straightness over a longer run** with your tuned values?

Kp = \_\_\_

Ki = \_\_\_

Kd = \_\_\_

Place the robot at the start and run the code.  
**Record the final behavior:**

**🧪 Lab Activity 2: Warehouse Navigation Challenge**

In this challenge the goal is to use a pre-tuned PID system in order to navigate precisely between a few waypoints.

1) Start by uploading code ***\_6\_PID\_controller\_*** to the robot.

While we CAN tune ***base\_speed***, ***kp***, and ***kd*** in this code, we will attempt to leave it as it is (at least at first)

2) Upload Code ***6\_PID\_main***

Inside this code there is a function that says ***def move\_sequence*** this is where you will program your route. There are three commands to be aware of:

1) ***controller.drive\_straight(<ticks>)*** This controls how far the robot should move forward

This can only be a positive number

2) ***controller.turn\_degrees(<degrees>)*** This controls how many degrees to turn.

Positive value is **TURN LEFT**, negative value is **TURN RIGHT**

3) ***process\_movement()*** This has to follow any straight or turn movement (It would have made the code more complicated by not having it in there)

3) Find the correct sequence of moves that can precisely hit the waypoints of the ‘warehouse’

Tip #1: Make sure to align the robot as straight as possible when starting the drive routine

Tip #2: If you work out how long a distance of say 1000 ticks is in centimeters or inches then it could reduce some of the guesswork when making the routine.