Control Assessment

Implementing a Longitudinal PID Controller

Overview

In this task, you will implement a **Longitudinal PID controller** to control the speed of a simulated vehicle.
The objective is to maintain the desired speed by adjusting the throttle based on the current speed error.

This exercise aims to test your programming and problem-solving skills, as well as your understanding of basic control systems.

Task Overview

You are provided with a partial code base for a PID controller. Your goal is to complete the implementation, ensuring the vehicle's speed closely follows the desired setpoint.

The provided files include necessary classes and utility functions, allowing you to focus on the core task of implementing the PID control logic.

Driving Scenario

- The vehicle starts its journey from rest, gradually accelerating for 30 seconds.
- It maintains a constant speed for a short period before slightly decelerating and continuing at a new constant speed.
- At around 80 seconds, the vehicle decelerates sharply, followed by cruising at a lower speed.
- Another sharp deceleration occurs, bringing the speed close to zero, followed by a slight acceleration to a steady pace for the remainder of the journey.

Key Components

PID Control Theory

- **Error**:
  e(t) = desired_speed - current_speed
- **Proportional (P)**:

  P = Kp * e(t)

  Corrects the present error proportionally.
- **Integral (I)**:

  I = Ki * (integration of e(t) from 0 to t)

  Corrects cumulative error over time.
- **Derivative (D)**:

  D = K_d * d(e(t))/dt

  Predicts future error based on the rate of change.
- **Control Output**:
  u(t) = P + I + D

Files Provided

1. **controller.py**:
   - `PIDController` : Implements PID logic to compute control signals.
2. **grade.py**:
   - `read_waypoints` : Reads speed/time waypoints from `waypoints.csv` .
   - `simulate` : Runs the simulation using the PID controller.
   - `grade_performance` : Calculates MSE.
   - `main` : Plots results using `matplotlib` .
3. **PID_Controller.ipynb** *(Optional)*:
   - Visualizes the PID’s performance.

    - Allows users to tweak PID gains and initial conditions.
    - Displays results and calculates MSE.

4. **waypoints.csv**: CSV format of speed/time waypoints.

Instructions

1. Setup

Python:

- Install required packages:

  ```
  pip install numpy pandas matplotlib
  ```

---

2. Running the Code

Python:

```
 # Navigate to directory
python grade.py
```

OR

Use the jupyter library `PID_Controller.ipynb`

---

3. Editing the Files

- **Python**: Edit `controller.py`, `grade.py`, or `PID_Controller.ipynb`.

---

4. Implementation

- **Objective**: Complete the PID control logic in the respective files.

---

5. Deliverables

- Submit the completed code.
- A short (1-2 mins) **screen**-recording or a video (presentation slides or whatever you prefer) explaining your process.
- A 1-2 page report documenting your research and process.

Submission (2 Options):

1. Send it back

1. Zip your workspace folder.
2. Rename it to "FIRSTNAME_LASTNAME_GRADUATIONYEAR.zip"
3. Submit the zipped folder.

2. Github Repository

1. Create a **public** repository from the assessment workspace.
2. Submit the link to the **public** repository.

Evaluation Criteria

1. **Correctness**: Controller should match the desired speed accurately.
2. **Code Quality**: Clean, well-documented, and follows best practices.
3. **Performance**: Smooth and minimal speed error.

Resources

- Introduction to PID Controllers – Wikipedia
- Understanding PID Control – YouTube Playlist
- How to Tune a PID Controller – YouTube

> **Note to Candidate**:
> This task is meant to encourage self-research. Your ability to effectively find and apply information from online resources is key to success.

Good luck, and we look forward to your submission!