Week 10 Assignments: PID Control and Algorithm Analysis

Assignment 1: PID Control for Spring-Damping System

Objective:

Develop a graphical application using MATLAB App Designer to simulate a spring-damping system controlled by a PID algorithm. The application should allow the user to set parameters for mass, damping coefficient, spring constant, and PID gains.

Requirements:

1. **Graphical Interface**:
   - Input fields for:
     - Mass m (kg)
     - Damping coefficient c ($\mathrm{N} \cdot \mathrm{s/m}$)
     - Spring constant k (N/m)
     - PID gains: kP, kI, kD
     - Desired setpoint (position)
   - Button to **run the simulation**.
   - Plot showing the position of the system over time.
   - Ability to **show the last two runs**:
     - The most recent run should be plotted with solid lines.
     - Example: The previous run can be plotted with dashed lines however, you may choose a different method if you feel it offers a better user experience.
2. **Functionality**:
   - Implement the PID control using the provided `simulatePIDControl` function.
   - Ensure the app clears previous plots and shows only the last two runs.
   - The app should handle different sets of inputs and visualize how changes affect the system response.
3. **Submission**:
   - Submit your MATLAB `.mlapp` file along with a short write-up (1-2 paragraphs) explaining
     - How your app works
     - What did you find to be the best numbers
     - Were you able to determine a pattern in the three constants regarding how they control the results

Tips:

- Experiment with various values for kP, kI, and kD to understand their effects.
- Consider edge cases, such as very high or very low damping coefficients, and explain what happens.

Example

As always, this is merely an example, you have free rein to design however you feel so long as it meets the requirements.

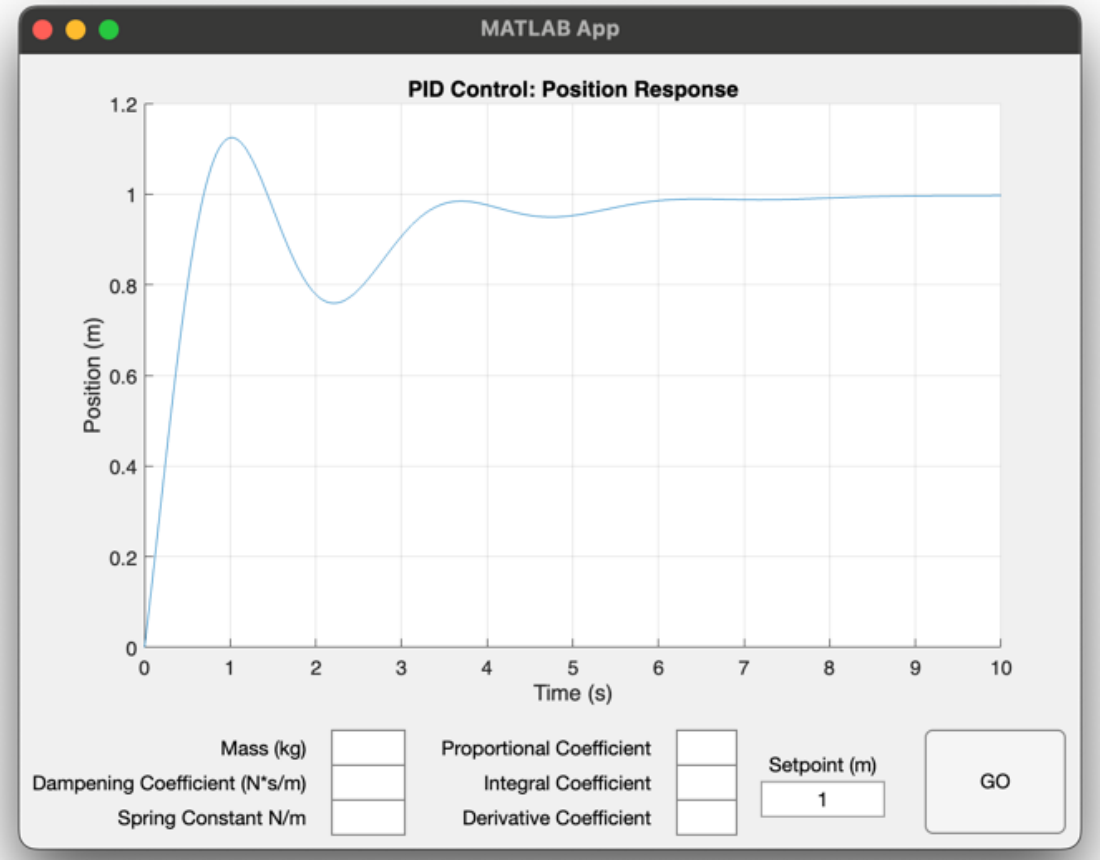

Figure 1: spring dampening comparison

Assignment 2: PID Line-Following Robot

Objective:

Develop a graphical application using MATLAB App Designer to simulate a line-following robot controlled by a PID algorithm. The application should allow the user to set PID gains and observe how the robot adjusts its path.

Requirements:

1. **Graphical Interface**:
   - Input fields for:
     - PID gains: kP, kI, kD
     - Setpoint for the Pololu sensor (e.g., 3500)
     - Random seed to allow repeatable simulation runs
       - Changing the seed will generate a new set of random numbers
       - Be sure to record the seed, along with the other numbers that you provide in the report.
   - Button to **run the simulation**.

- Plot showing sensor readings and motor speeds over time.
- Ability to **show the last two runs**:
  - The most recent run should be plotted with solid lines.
  - Example: The previous run can be plotted with dashed lines however, you may choose a different method if you feel it offers a better user experience.

2. **Functionality**:
   - Implement the PID control using the provided `simulateLineFollowerPololu` function.
   - Ensure the app clears previous plots and shows only the last two runs.
   - Allow the user to adjust the PID parameters and run the simulation multiple times, observing the differences.
3. **Submission**:
   - Submit your MATLAB `.mlapp` file along with a short write-up (1-2 paragraphs) explaining
     - How your app works
     - What did you find to be the best numbers
     - Were you able to determine a pattern in the three constants regarding how they control the results

Tips:

- Adjust the input random seed to see consistent deviations, making it easier to observe how the PID settings respond.
- Describe how changes in PID parameters affected the robot's performance and any observations you found interesting.
- Be sure to include the two datasets for Deviation, it may appear as though this dataset doesn't change, but it will if you change the seed which will be important to see for testing values across different datasets (seeds)

Example

As always, this is merely an example, you have free rein to design however you feel so long as it meets the requirements.

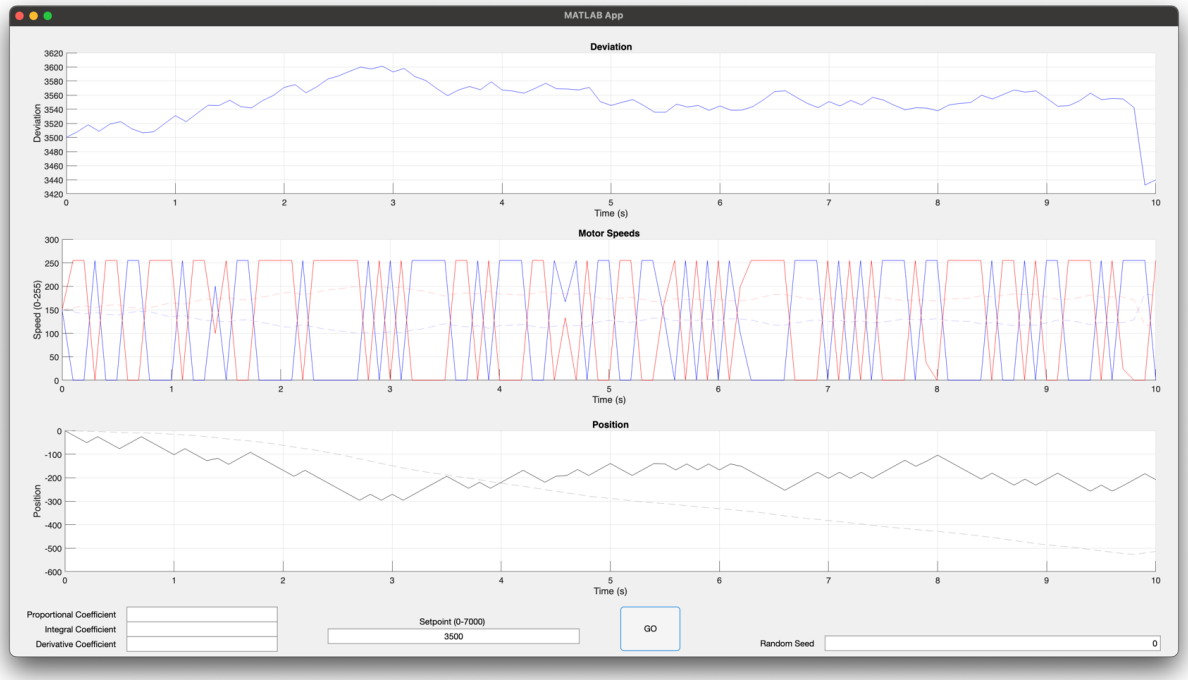

Figure 2: line follow comparison

Assignment 3: Custom Algorithm Comparison

Objective

Build a small App Designer tool that compares algorithm behavior. Choose ONE of the comparison modes below and present results clearly.

- Mode A — Same algorithm, different inputs/parameters: Run one algorithm against the same dataset while varying inputs, hyperparameters, or noise. Examples: random walk with different step distributions, k-means with different `k` or initializations.
- Mode B — Different algorithms, same input: Run two algorithms side-by-side on the same dataset or task. Examples: merge sort vs. bubble sort, MATLAB `sort` vs. your bubble sort, looped implementation vs. vectorized implementation.

Implementation expectations: Implement at least one algorithm yourself in MATLAB. If one of your comparisons uses a built-in (e.g., MATLAB `sort`), you may implement only the other algorithm (e.g., bubble sort) and compare them fairly on the same input.

Requirements

1. **Graphical Interface**
   - Input fields for parameters and/or dataset selection (as appropriate for your choice).
   - Button to run the comparison.
   - Plots and/or metrics clearly showing results.
   - Show the last two runs for quick visual comparison (e.g., solid vs dashed, or different colors).
2. **Functionality**
   - Implement at least one algorithm yourself unless both are MATLAB built-ins and you are comparing inputs/parameters (Mode A).

- Use a consistent dataset across comparisons (Mode B) or a consistent algorithm with varying inputs (Mode A).
- Report relevant metrics. Suggested options (pick what fits your task):
  - Runtime (`tic/toc`) or operation counts
  - Accuracy, error, or convergence (e.g., residuals, number of iterations)
  - Stability/overshoot/settling time (for control problems like PID)
- Make it easy to re-run with the same settings (e.g., an optional random seed field).

3. **Submission**
   - Submit your `.mlapp` and a short write-up (1–2 paragraphs) including:
     - Which mode you chose (A or B) and why.
     - What you implemented vs. what (if anything) you used from MATLAB built-ins.
     - A summary of your findings with at least one figure or table (screenshots are fine).

Examples (choose or adapt)

- Sorting: Bubble sort (implemented) vs MATLAB `sort` on random arrays; vary `N`, measure runtime.
- Control: PID on the same plant with different gains; measure overshoot, rise time, steady-state error.
- Clustering: k-means with different `k` or seeds; compare inertia or cluster consistency.
- Simulation: Random walk with different step distributions; compare variance growth.
- Vectorization: For-loop implementation vs vectorized equivalent; measure runtime and discuss readability/maintainability trade-offs.

General Submission Guidelines:

1. Each `.mlapp` file should be well-organized and user-friendly.
2. Include your write-up in any format (word, text, md, pdf, etc), clearly explaining your approach and observations.

Definition of Done

Your `Week10` folder shall contain **at minimum** the following files:

- Week10/
  - `SpringDampingPID.mlapp`
  - `LineFollowingPID.mlapp`
  - `CustomAlgorithmComparison.mlapp`