LAB 9 Report

Darshil Kedia, dkedia2@wisc.edu , dkedia2

Anay Baheti, abaheti@wisc.edu , abaheti

Part 1: Error Growth in Acceleration Data

Figure 1 (Acceleration vs. time) compares the “true” acceleration values to the noisy acceleration measurements from ACCELERATION.csv. The true signal is piecewise constant, while the noisy signal wiggles rapidly around it with high-frequency variations. When we later integrate this data, those small fluctuations accumulate, which is why noisy sensor measurements can cause large errors in downstream estimates.

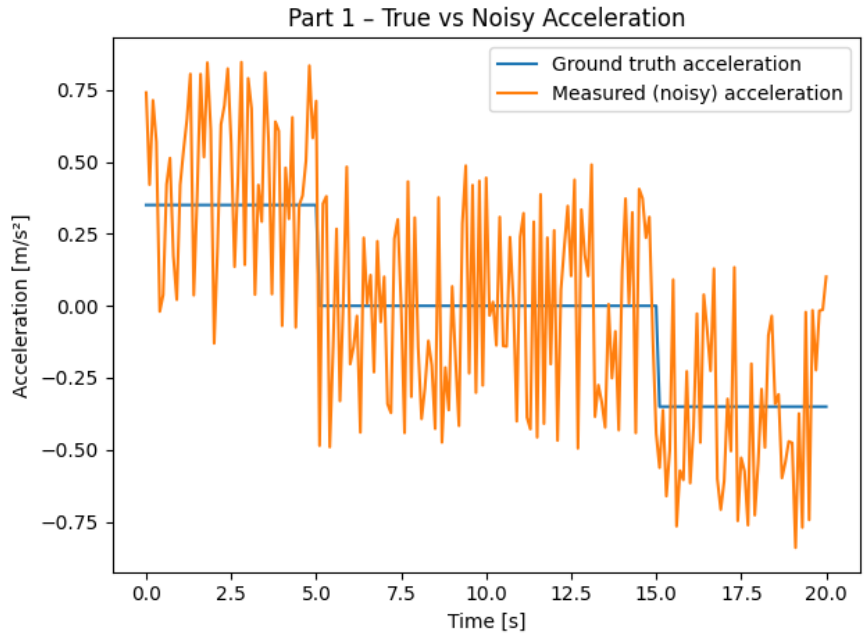


Figure 2 (Speed vs. time) shows the velocities obtained by integrating both acceleration traces once. The velocity computed from the true acceleration forms a clean triangle: speed increases linearly, stays roughly constant, and then symmetrically decreases back toward zero. In contrast, the velocity derived from the noisy acceleration slowly drifts away from the ideal profile and has visible bumps and irregularities. This illustrates how integration amplifies noise and introduces drift over time.

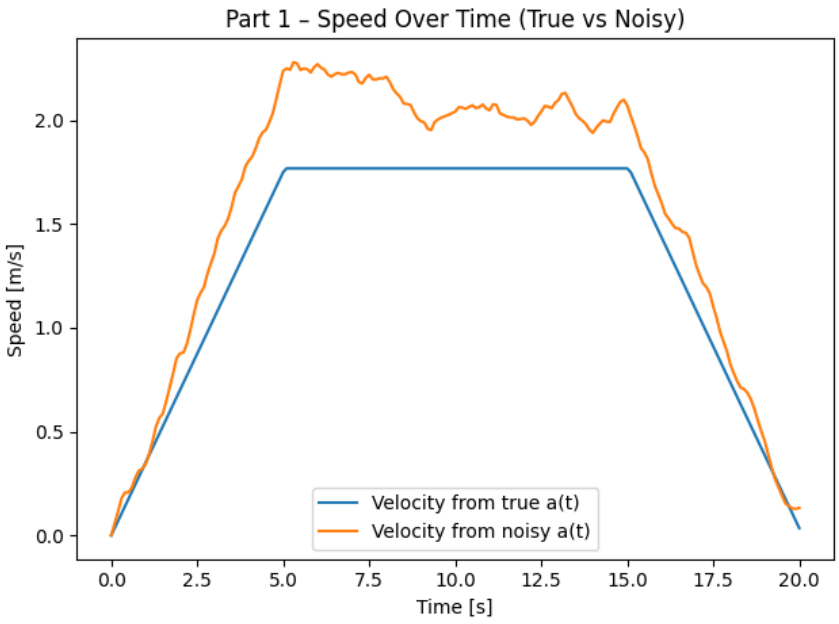
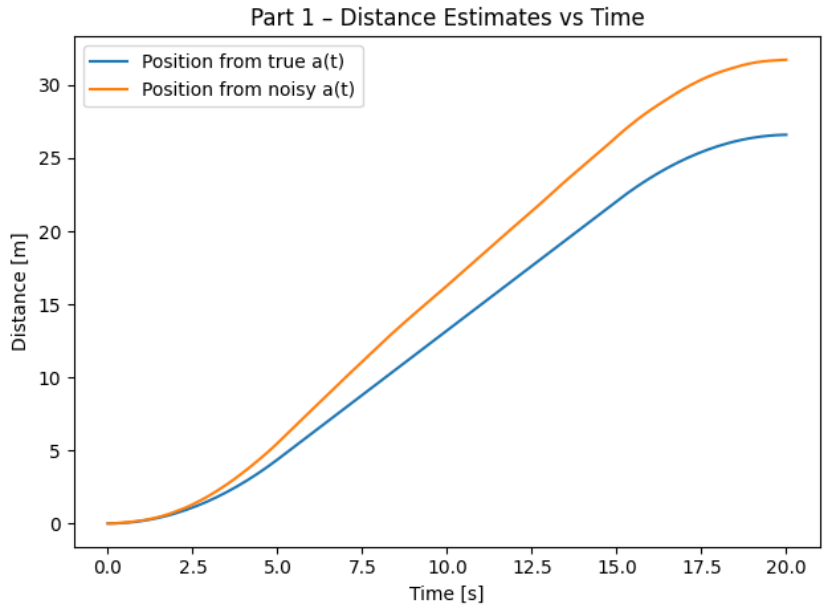


Figure 3 (Distance vs. time) shows the distances after integrating the velocities a second time. The distance from the true acceleration increases smoothly and matches the expected motion of the experiment. The distance from the noisy acceleration grows more quickly and ends noticeably higher because the integration is accumulating the noise as a bias.

From the script output:

- Final distance (true acceleration) ≈ **26.6 m**
- Final distance (noisy acceleration) ≈ **31.7 m**
- Difference (noisy − true) ≈ **5.1 m**

Even though the noise at the acceleration level is relatively small, integrating twice (acceleration → velocity → position) magnifies this error into several meters of position offset by the end of the run. This demonstrates why raw accelerometer readings must be filtered or fused with other sensors before using them for precise distance estimates.



Part 2: Step Detection from Accelerometer Magnitude

To detect steps in WALKING.csv, I first treated the three acceleration axes as a vector and computed the magnitude at each timestamp:

$$\text{mag} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

The raw magnitude signal is quite noisy and contains small spikes that are not actual steps. To make the step pattern clearer, I applied exponential smoothing with parameter $\alpha = 0.1$:

- $s[0] = \text{mag}[0]$
- $s[i] = \alpha \cdot \text{mag}[i] + (1 - \alpha) \cdot s[i-1]$

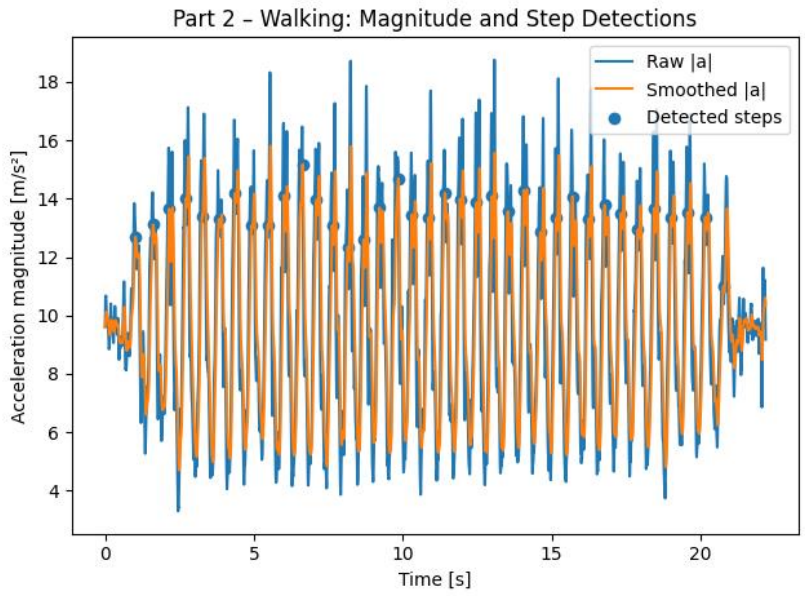
This filter reduces high-frequency noise but preserves the larger peaks associated with footsteps.

Figure 4 shows both the raw magnitude and the smoothed magnitude over time, along with markers for the detected steps. Steps are identified by scanning the smoothed signal for local maxima that satisfy three conditions:

1. The sample is a local peak:
   $s[i] > s[i-1]$ and $s[i] > s[i+1]$.

2. The peak is “large enough”:
   $s[i] > \text{mean}(s) + 1.0$.
   This threshold rejects small fluctuations while keeping the true stride peaks.
3. At least **0.4 seconds** have passed since the previous detected step.
   This enforces a realistic minimum time between steps and prevents double-counting a single step with multiple nearby peaks.

Using this algorithm on WALKING.csv, the script detects **37 steps**, which matches the expected number for the dataset. This indicates that the combination of smoothing, thresholding, and temporal spacing works well for separating actual steps from sensor noise.



Part 3: Detecting Turn Angles from Gyroscope Data

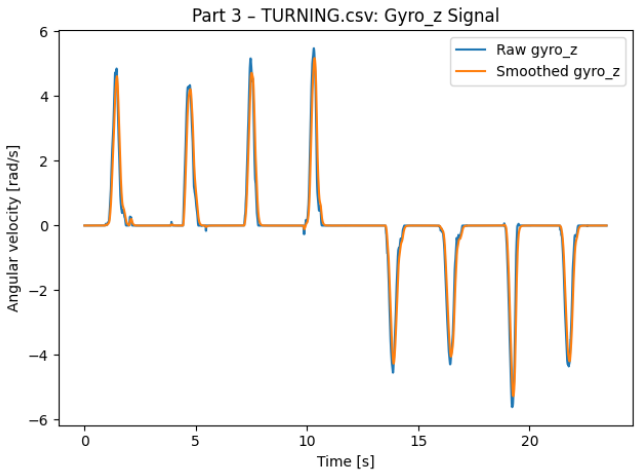
Data processing

For TURNING.csv, I focus on the gyro_z signal, which measures angular velocity around the device’s vertical axis. Positive values correspond to rotation in one direction (counter-clockwise), and negative values correspond to rotation in the opposite direction (clockwise).

As in Part 2, I smooth the raw gyro signal using exponential smoothing with $\alpha = 0.1$:

- $g_s[0] = \text{gyro\_z}[0]$
- $g_s[i] = \alpha \cdot \text{gyro\_z}[i] + (1 - \alpha) \cdot g_s[i-1]$

Figure 5 plots both the raw and smoothed gyro_z. The smoothing removes small jitter while keeping the sharp peaks that indicate actual turns.



Turn-detection logic

The turn-detection algorithm proceeds in two main steps:

1. **Identify turning segments**
    - Define a velocity threshold of **0.3 rad/s**.
    - Whenever $| g_s[i] | > 0.3$, the user is turning.
    - Consecutive samples above this threshold form a single turn segment.
2. **Integrate angular velocity to get total angle**
    - Within each segment, I integrate angular velocity over time: $\theta \approx \sum g_s[i] \cdot \Delta t_i$, where $\Delta t_i$is the time step in seconds.
    - The resulting angle (in radians) is converted to degrees.
    - A positive angle is labeled **CCW**, and a negative angle is labeled **CW**.

Using this method, the script finds **8 distinct turns**:

- Four turns of roughly **+85° to +90°** (counter-clockwise)
- Four turns of roughly **−85° to −90°** (clockwise)

These angles line up with the experiment design: several 90° rotations in one direction, followed by the same number of 90° turns back the other way.

Part 4: Reconstructing the Walking Trajectory

In the final part, I combine step events and turn events from WALKING_AND_TURNING.csv to reconstruct an approximate 2D path.

Step estimation

I reuse the same approach as in Part 2:

1. Map the accelerometer columns into a_x, a_y, a_z for this file.
2. Compute acceleration magnitude at each time step and smooth it with $\alpha = 0.1$.
3. Use the same local-maximum detector with:
    - Threshold $=$ mean(smoothed) $+ 1.0$
    - Minimum spacing of **0.4 s** between detected steps.

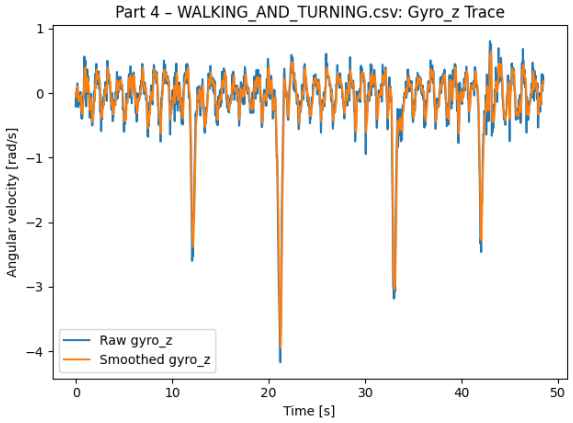
Each detected step is assumed to represent **1 meter** of forward motion.

Turn estimation

I then process gyro_z from the same file:

1. Smooth gyro_z with exponential smoothing ($\alpha = 0.1$), like Part 3.
2. Use a threshold of **0.3 rad/s** to find intervals where the user is rotating.
3. Integrate the smoothed angular velocity over each interval to estimate the turn angle in degrees.
4. Keep only "significant" turns with $|\theta| > 30°$ to focus on real direction changes rather than tiny wobbles.

Figure 6 shows the raw and smoothed gyro_z used for this turning analysis.



Building the 2D path

To form the trajectory, I merge the step and turn events into a single time-ordered list:

- The user starts at position $(0,0)$facing **north** (along the positive y-axis).
- For each event:
    - If it is a **turn**, I update the current heading by adding the signed turn angle.
    - If it is a **step**, I move the position forward 1 meter in the current heading:
        - $x_{\text{new}} = x_{\text{old}} + \sin(\theta_{\text{heading}})$
        - $y_{\text{new}} = y_{\text{old}} + \cos(\theta_{\text{heading}})$

Repeating this for all events yields a sequence of (x, y)coordinates.

Figure 7 shows the final trajectory plot using these reconstructed positions. The path clearly shows:

- Several straight walking segments where consecutive steps line up,
- Distinct corners where the heading changes due to detected turns,
- A closed-ish loop shape that matches the walking pattern encoded in WALKING_AND_TURNING.csv, though not perfectly due to sensor noise and the simplifying assumption of 1 meter per step.

Overall, the trajectory demonstrates how combining step counting from accelerometer data with heading changes from gyroscope data can approximate a user's path, even though each individual sensor by itself is noisy and imperfect.