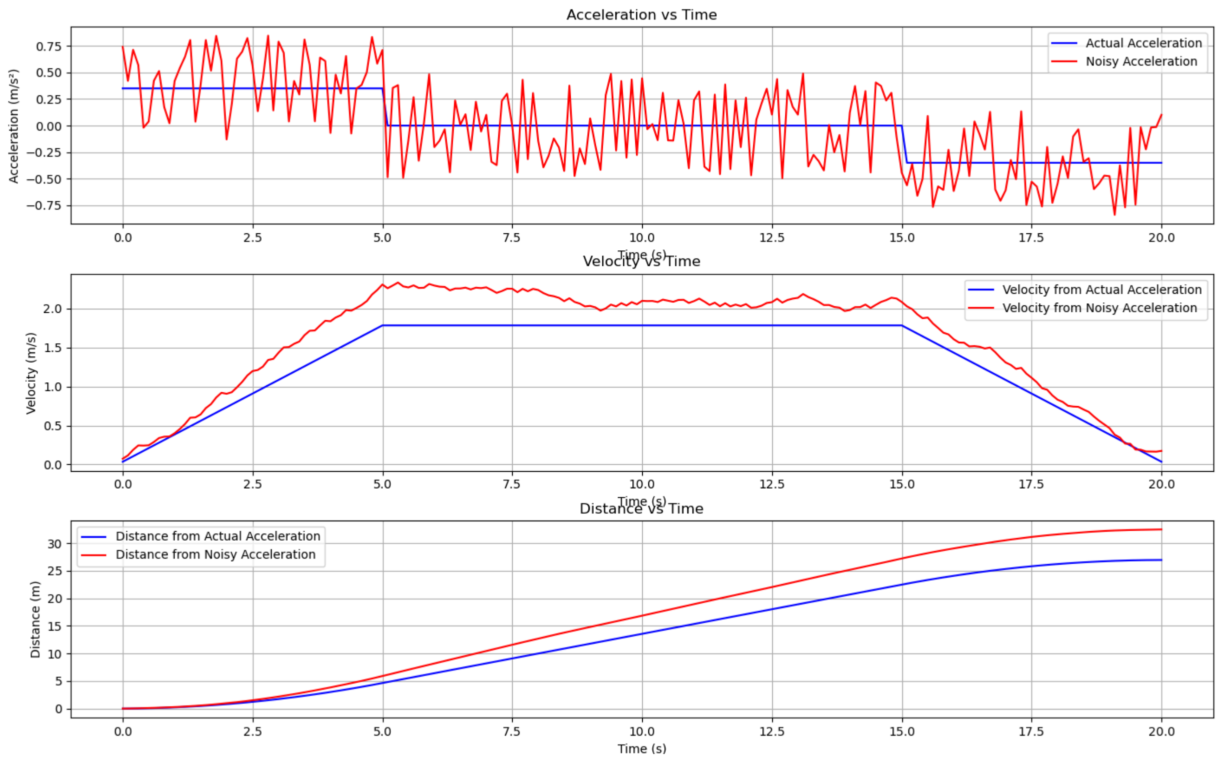
Ahmad Imam, aaimam@wisc.edu, ahmad-i
Xerxis Palsetia, xpalsetia@wisc.edu, x-pal
Shivam Ratnani, ratnani@wisc.edu, shivamratnani

Part 1



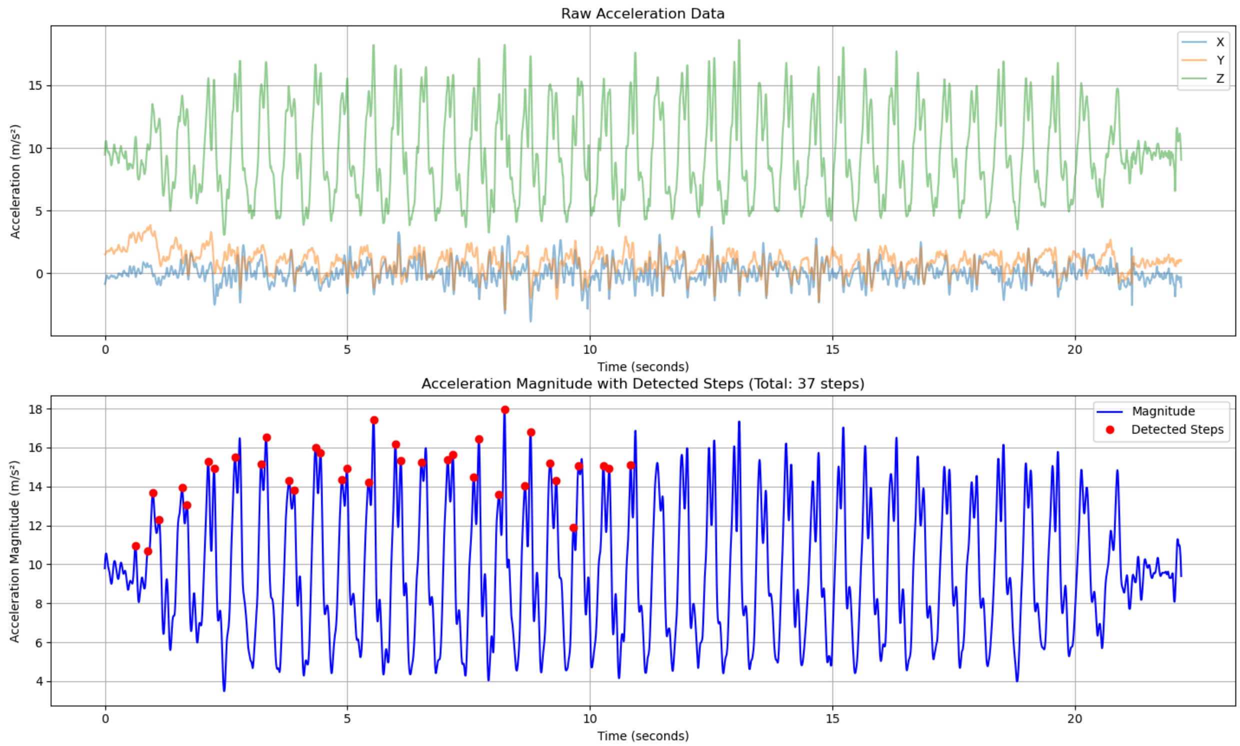
Final distances

Actual: 3.5 m

Noisy: ~4.12 m

The difference is ~0.62 meters, which shows 17.7% error.

Part 2



The data smoothing process uses the Savitzky-Golay filter

- Window length = 21 samples (must be odd)
- Polynomial order = 3

We chose these parameters because:

- Window length 21 provides enough points to smooth out noise while preserving the step pattern
- Polynomial order 3 allows the filter to capture the natural curved shape of acceleration during steps
- This combination effectively removes high-frequency noise while maintaining the important peaks that indicate steps

```
(base) ai@SouljaBook-Air PART2 % python3 step-detection.py
Number of steps detected: 37
```

1. Data Loading and Preparation:
    - The function load_and_prepare_data reads a CSV file containing accelerometer data (accel_x, accel_y, accel_z) and timestamps.

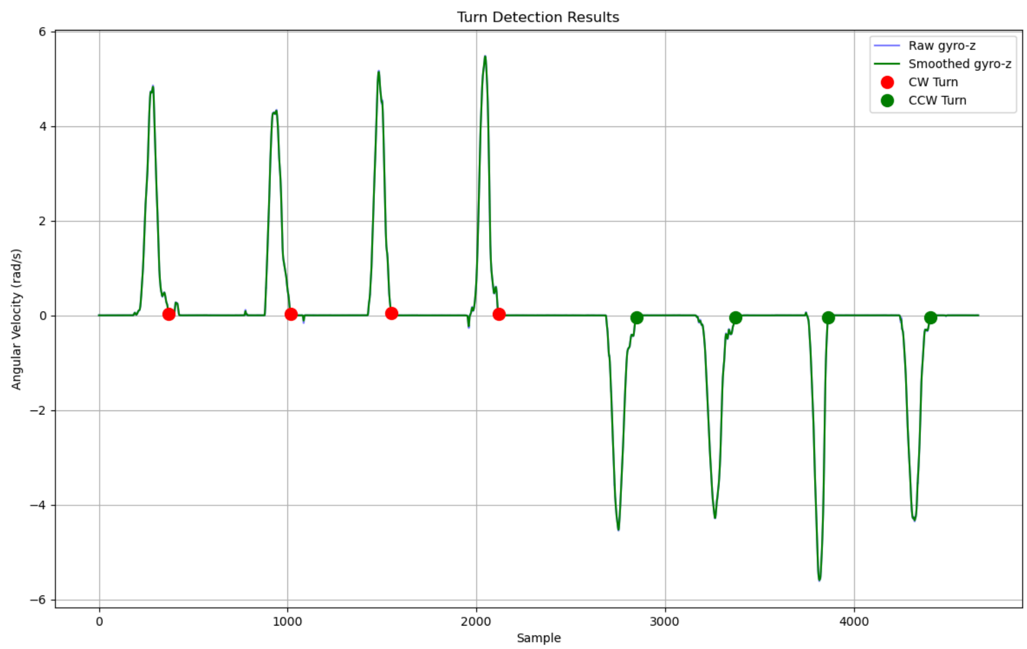
    - It calculates the elapsed time in seconds from the initial timestamp for better plotting and analysis.
2. Smoothing Signals:
    - The smooth_signals function applies a Savitzky-Golay filter to smooth out noise in the accelerometer data for all three axes (accel_x, accel_y, accel_z).
    - Smoothing is crucial to reduce high-frequency noise that might interfere with peak detection.
3. Step Detection:
    - The script calculates the magnitude of acceleration using the formula:

$$\text{magnitude} = \sqrt{(\text{accel\_x\_smooth}^2 + \text{accel\_y\_smooth}^2 + \text{accel\_z\_smooth}^2)}$$

    - Peaks in the magnitude signal are identified as steps if:
        - Their value is above a predefined threshold (min_peak_height = 10.5).
        - There are sufficient samples between peaks to avoid detecting the same step multiple times (min_samples_between_peaks = 20).
4. Step Detection Logic:
    - The script scans the magnitude signal, identifying points that are higher than their neighbors and exceed the threshold.
    - The detected peaks are considered steps. A simplistic method divides the detected peaks by 2, assuming peaks represent up-and-down motion.
5. Results and Visualization:
    - The plot_results function generates plots of the raw acceleration data and the magnitude signal with detected steps marked.
    - It prints the total number of detected steps.

Results of Step Counting:
The script, when executed, it reads a file named WALKING.csv, processes it, and outputs:
Total Detected Steps & a visualization

Part 3



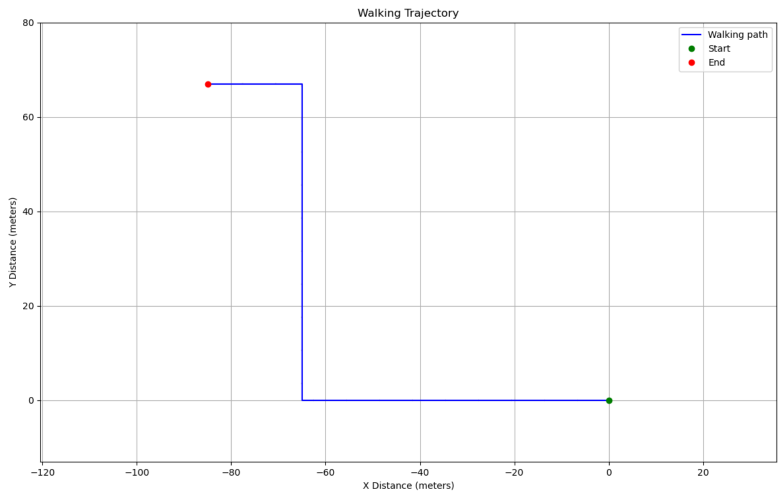
1. Data Preparation:
   - Load and visualize the raw gyroscope and magnetometer data to identify relevant patterns for turns.
   - Apply smoothing techniques (e.g., moving average or low-pass filters) to reduce noise.
2. Direction Detection Algorithm:
   - Analyze changes in angular velocity from the gyroscope data to detect 90-degree turns.
   - Integrate or compute cumulative angular displacement to determine the angles of rotation.
   - Cross-reference magnetometer data (if needed) for better accuracy in detecting direction relative to Earth's magnetic field.
3. Code Implementation:
   - Modify or use the turn-detection.py script to:
     - Read and preprocess the TURNING.csv dataset.
     - Detect turning events and classify them as clockwise or counterclockwise.
     - Output the number and angles of the turns detected.

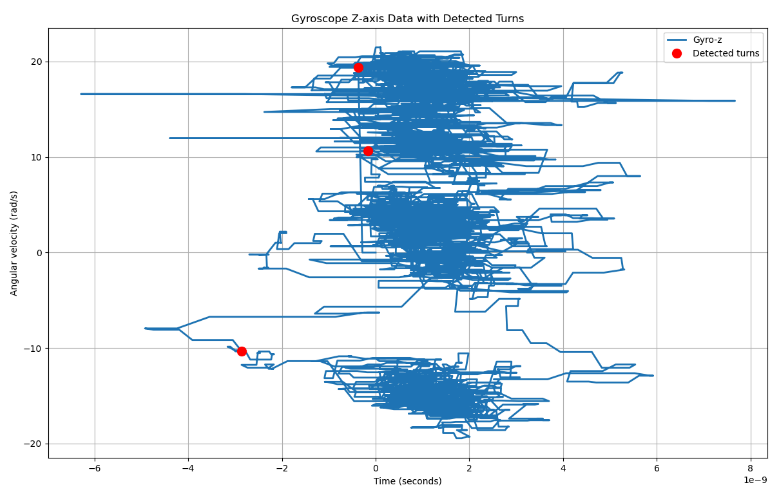
1. CSV Parsing:

    - Reads the TURNING.csv file.
    - Extracts timestamp and Z-axis gyroscopic data, ignoring rows with invalid entries.
2. Smoothing:
    - Uses the scipy.signal.savgol_filter, which can be applied to smooth noisy gyroscopic data.
3. Visualization:
    - The inclusion of matplotlib.pyplot suggests that plotting might be used for debugging or analysis.

Key Components:
1. Time Difference Calculation:
    - Computes time intervals (dt) between consecutive timestamps and normalizes them into seconds.
2. Data Smoothing:
    - Applies a simple moving average with a window size of 5 to smooth the gyroscopic data (gyro_z).
3. Turn Detection Logic:
    - Compare smoothed gyroscope values (smoothed_gyro) against a threshold (z_threshold).
    - Tracks accumulated angular displacement and identifies turns when specific conditions are met.

Part 4





1. Data Integration:
    - The WALKING AND TURNING.csv dataset was loaded, which contains timestamped sensor data (acceleration, gyroscope, magnetometer) for both walking and turning.
    - The raw data was pre-processed to account for:
        - Using smoothing techniques such as moving averages or low-pass filters.
        - Ensuring all data are in consistent units

2. Step Detection:
    - The step detection algorithm from Part 2 was utilized to identify steps in the walking sequence.

- Each step was assumed to be 1 meter.
- The timestamps for detected steps were aligned with the trajectory calculations.

3. Direction Detection:
   - Data from the gyroscope and/or magnetometer was analyzed to determine the turning angles.
   - The turning detection algorithm from Part 3 identified discrete turns and the associated angles
   - These angles were applied sequentially to update the trajectory direction.

4. Trajectory Computation:
   - The trajectory was computed iteratively starting from an initial position and orientation
   - At each step:
     1. The new position was calculated based on the current orientation and step length:
     2. If a turn was detected, the orientation angle was updated accordingly.

5. Visualization:
   - A plot was created to represent the calculated trajectory:
     - The x-axis and y-axis show spatial displacement.
     - Each point represents a step, and lines connecting the points represent the movement.
     - Turns were annotated with arrows or labels to highlight changes in direction.
   - A grid or compass rose was overlaid to provide a reference for orientation (north, east, south, west).

6. Implementation:
   - The implementation involved scripting in Python (or a similar language):
     - Libraries like matplotlib or seaborn were used for plotting.
     - Data manipulation was performed using numpy or pandas.
     - Custom functions were developed to handle step detection, angle updates, and trajectory plotting.