CS407 Fall 2024

Pair Programming: **ALLOWED**
Due: **12:00PM CT on December 2nd**

Lab08 Sensor Processing

Overview

In this lab,

- In **Milestone 1**, you will (1) read gravity data in real-time, (2) write an app to simulate a ball rolling on the phone's screen due to gravity.
- In **Milestone 2**, you will (1) calculate the displacement based on the given acceleration data, ACCELERATION.csv, and (2) implement an algorithm to detect and count the steps with the given dataset, WALKING.csv, (3) implement another algorithm to detect the directions with the given dataset, TURNING.csv, and finally, (4) you will combine the two algorithms to reproduce the trajectory of a walking user with the given dataset, WALK_AND_TURN.csv.

Learning Objectives

By completing this lab, a cs407 student will be able to:

1. **Get the sensor readings** from "SensorManager" class when developing a sensor data-based application.
2. **Understand and handle noisy sensor data**.
3. **Apply data smoothing algorithms** to the real-world sensor dataset for further data analysis tasks.
4. **Implement algorithms** to detect the steps and direction of a person when walking, given the sensor data.
5. **Approximate the trajectory** of a person given the sensor data.

Directions

Milestone 1 – Tilt 'n Roll!

In this milestone, you will implement a sensor-based interactive application where a ball moves according to the device's orientation. You will use the gravity sensor to detect device rotation and translate this into ball movement on the screen.

Implementation Overview

The application processes the device's gravity sensor data to control a ball's movement on the screen. Starting with the device's Y-axis vertical to the ground (positive direction pointing away from the ground, shown in Fig 2 (a)), the ball will respond to gravity. As you tilt the device, the gravity projection on the device screen changes, which affects the ball's movement, causing it to roll accordingly.

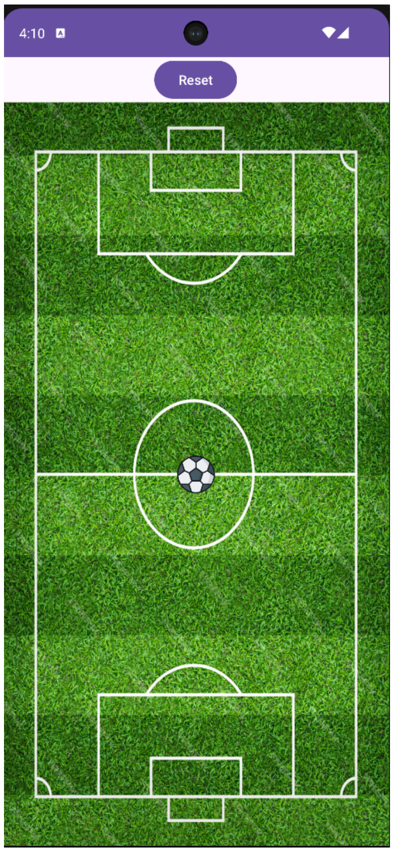
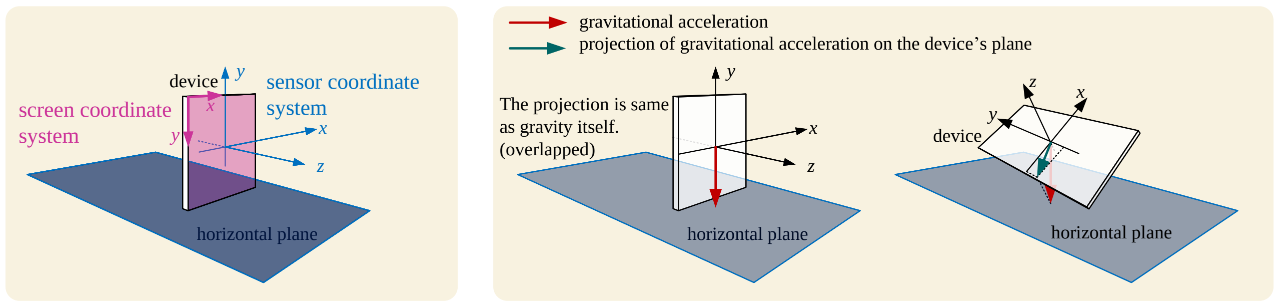


Figure 1: Screenshot when run the

Setup

1. As you did in previous labs, click **this link** and accept the invitation for the **lab8** repository.
2. Set the Git VCS correctly as what you did in the previous labs. The repository contains the starter code, which should run without any modifications. Upon running, you should see a field with a soccer ball positioned at the center, as illustrated in Figure 1.


device
y
sensor coordinate system
screen coordinate system
x
y
x
z
horizontal plane
gravitational acceleration
projection of gravitational acceleration on the device's plane
The projection is same as gravity itself. (overlapped)
y
x
z
horizontal plane
z
x
y
device
horizontal plane

(a) Coordinate system, screen *vs* sensor (b) "Rotation" and projection of the gravity

Figure 2: Coordinate System and Gravity Projection

Implementation Guide

Project Overview. The starter code includes all necessary files—`Ball.kt`, and `MainActivity.kt`. No additional files are required. The `Ball` class handles the ball's position and motion state. Gravity vector data is read in `MainActivity` to update the ball's movement trajectory. **To ensure the ball's movement remains consistent and unaffected by screen rotation**, we've already locked the screen orientation to portrait mode by setting

```
android:screenOrientation="portrait"
```

in the `AndroidManifest.xml`. We will now walk through each file—`Ball.kt`, `Rotation.kt`, and `MainActivity.kt`. Note that in this experiment, we frequently work with two different coordinate systems: the screen coordinate system and the sensor coordinate system, which are distinct. See Figure 2 (a) for reference. The former is a two-dimensional plane coordinate system where the positive x-axis extends to the right side of the screen and the positive y-axis extends downward. The latter, on the other hand, is a three-dimensional coordinate system where the y-axis points in the opposite direction compared to the former.

Ball The `Ball` class represents a soccer ball that can move across the field. It has several member variables; please refer to the code comments to understand their meaning.

You will need to analyze how an object moves based on the acceleration data when implementing the `updatePositionAndVelocity` function. Suppose an object is moving in a one-dimensional direction. We examine its motion at two moments: at time t_0, where the velocity and acceleration are v_0 and a_0, and at time t_1, where they are v_1 and a_1. Given that $t_1 - t_0$ is very short, we **assume that the acceleration changes linearly within this interval**. Therefore, acceleration and velocity can be modeled as functions of time t ($t_0 \le t \le t_1$):

$$a(t) = a_0 + \frac{a_1 - a_0}{t_1 - t_0} \times (t - t_0)$$

$$v(t) = v_0 + \int_{t_0}^{t} a(\tau)\, d\tau = v_0 + a_0 \cdot (t - t_0) + \frac{a_1 - a_0}{2(t_1 - t_0)} \cdot (t - t_0)^2$$

When $t = t_1$, the velocity becomes:

$$v_1 = v(t_1) = v_0 + \frac{1}{2}(a_1 + a_0)(t_1 - t_0) \tag{1}$$

The distance traveled by the ball from t_0 to t_1 is:

$$l = \int_{t_0}^{t_1} v(\tau)\, d\tau = v_0 \cdot (t_1 - t_0) + \frac{1}{6} \cdot (t_1 - t_0)^2 \cdot (3a_0 + a_1) \tag{2}$$

Utilize Equation 1 and 2 to estimate the velocity and traveled distance of the ball in both x and y axis in `updatePositionAndVelocity`. You also need to implement 2 more functions, `checkBoundaries` and `reset` following the comments.

MainActivity. In this class, you will read the data from the gravity sensor. Please refer to the Gravity Sensor, Sensor, You can get the gravitational acceleration component on each axis (sensor coordinate system) by reading the value of the event

```
(event.values[0], event.values[1], event.values[2])
```

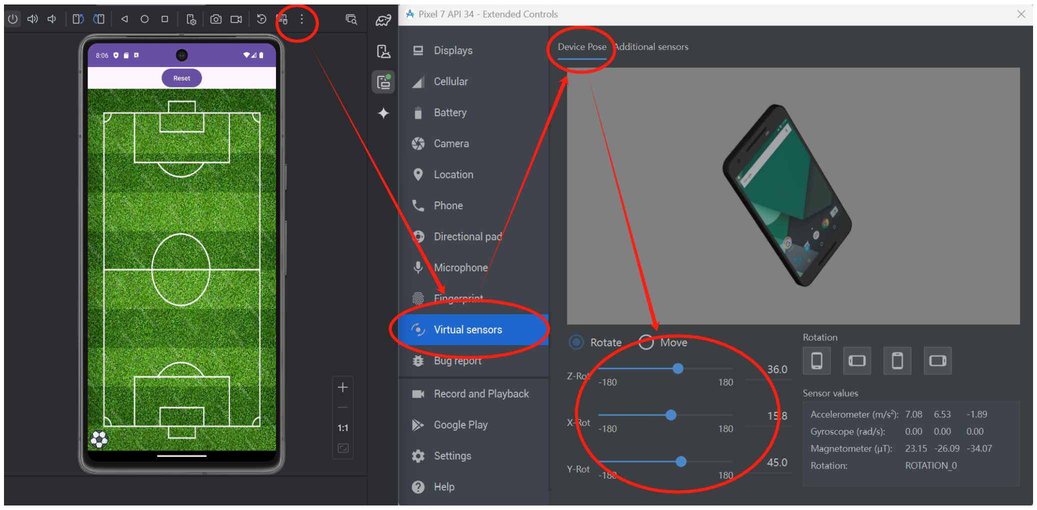
Note that the raw data from this reading is oriented opposite to the gravitational force in the sensor coordinate system. You need to determine the ball's acceleration and update its position by calling the relevant function.

Testing and Verification

When testing your implementation in the Android Emulator, use the sensor controls (Figure 3) to simulate device rotation. Test the Reset functionality frequently to manage accumulated errors. Verify that the ball moves smoothly across different rotation angles and handles screen boundaries appropriately.

Milestone 1 Deliverables

Good job, you have finished **milestone 1**. Be sure to push the changes to your application to **CS407 Lab08 Milestone 1** *GitHub* classroom, and submit it to the *GradeScope*! Now that you've learned how to read data from the gravity sensor, try experimenting with the gyroscope, accelerometer, and other sensors. You can play with the data in similar ways on your own!



Z-Rot: 36.0 (-180 to 180)
X-Rot: 15.8 (-180 to 180)
Y-Rot: 45.0 (-180 to 180)

Sensor values

Accelerometer (m/s²):	7.08	6.53	-1.89
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	23.15	-26.09	-34.07
Rotation:	ROTATION_0		

Figure 3: Testing

Milestone 2: Sensor Data Processing

In this Milestone, you will implement basic versions of two algorithms used in Pedestrian Dead-Reckoning (PDR). PDR is a technique to estimate someone's displacement from a certain location based on how far they walk, and what direction they walk. They can be used indoors, where GPS does not work, or even outdoors when the high power-usage of GPS is undesirable. Advanced PDR algorithms have many parts, but most PDR algorithms have 2 basic parts: step detection, and direction detection.

Dataset Description

Please download the dataset `lab8-dataset.zip`. Four files are provided with this homework, ACCELERATION.csv, WALKING.csv, TURNING.csv, and WALKING_AND_TURNING.csv. The contents of each file are summarized in Table 1.

The files are all formatted in the csv (comma-separated values) format. The first row has the title for each row, and every subsequent row is a new sample.

For ACCELERATION.csv, the sampling rate is 0.1 s, and the unit for column `acceleration` and `noisyacceleration` is m/s^2. For the latter 3 files, the sampling rate is 5ms nominally, but the exact sampling rate varies a little bit. Timestamps (which are expressed in nanoseconds) are provided in case you want to do any integration or anything like that. The files contain data from the Accelerometer, Gyroscope, and Magnetometer. You might not need the data from all sensors, but some techniques require them, so I've included them. Accelerometer values are provided in m/s^2, Gyroscope values in rad/s, and Magnetometer values in uT (micro-Teslas).

Table 1: Dataset Description

Filename	Description
ACCELERATION.csv	This file contains synthetic data of a person, moving in a single dimension. There are 3 columns, timestamp, acceleration, and noisyacceleration.
WALKING.csv	This file contains data for someone walking in a straight line. The phone was held face-up in an extended hand. The top of the phone pointed toward the walking direction.
TURNING.csv	This file contains a set of 90 degree turns. The phone was placed face-up on a table and rotated four turns clockwise (1 full circle) and then four turns counter-clockwise (another full circle).
WALKING_AND_TURNING.csv	This file also has walking data, but the user walked in a pre-specified path. (So there's walking and turns in there).

Part 1: Understanding Sensor Data Errors

Consider a person walking with a mobile device that reports acceleration. In this part you will calculate how much a noisy accelerometer can introduce errors in estimating distance traveled.

The file called "ACCELERATION.csv" contains a comma-separated set of values with three columns. The first column is a timestamp, the second column is the actual acceleration, and the third column is the acceleration reported by a noisy accelerometer sensor. The first row of the line indicates these field names.

Imagine this acceleration to be in a single dimension.

Using some software tools (such as Python, Excel, Matlab, etc.) write code that will calculate the real distance traveled, and the distance traveled if the measurements from the noisy accelerator is used.

Here are the things to submit for this part.

- A single plot showing the acceleration and the noisy acceleration.
- A single plot showing the speeds that would be obtained using the actual acceleration and the noisy acceleration values respectively.
- A single plot showing the distance traveled that would be obtained using the actual acceleration and the noisy acceleration values respectively.
- Report the final distances calculated using the two different accelerations.
- What is the difference between the two estimates?

Part 2: Step Detection

Step Detection does what its name implies: it detects steps. Step detection is used in a wide variety of applications. Phones and wearable devices commonly have step-detection algorithms in them to help them track fitness goals, runs, and other activities. Companies base whole devices on step detection algorithms (ex: Fitbit). For PDR, a good step detector is important because it gives PDR a sense of how far someone has walked. If a user puts in their height, the PDR algorithm can count the number of steps detected and multiply it by the users estimated step length to get an estimate of how far they've walked. Step detection might give you a good estimate of the distance you've traveled, but it doesn't give you any indication of what direction you've traveled.

In part 2, you will smooth the data with a technique you learned in the lecture and write an algorithm to count the number of steps in WALKING.csv.

Hint: you should be able to detect 37 steps in WALKING.csv.

Part 3: Direction Detection

Direction detection can be implemented using a variety of techniques. Some integrate gyroscope input to see direction differences, some use the magnetometer like a compass to track direction with respect to the Earth's magnetic field, and the most advanced algorithms combine gyroscope and magnetometer data for a more accurate direction estimation.

In part 3, you will smooth the data if necessary and write an algorithm to detect the 90-degree turns in TURNING.csv.

Part 4: Trajectory Plotting

Combining the steps and turning directions, you are able to reproduce the walking trajectory of a person. Each step is approximately 1 m in length, and every turn is a multiple of 45 degrees in either direction, e.g., 45, -90, 135.

In part 4, you will utilize the algorithms implemented in the previous tasks and make an analysis on WALKING_AND_TURNING.csv to plot the path the person took.

Some Tips

If you've never worked with sensor data before, this might not seem like a very trivial thing. Here are some tips to help you:

- Before you do anything, plot the various sensor axis in a tool like MATLAB or Excel. CSV files can be imported into these programs, and both provide pretty easy methods of plotting data. This will help you design your algorithms.

- When you make your algorithms, plot the sample different events were detected on, so you can see where your algorithm works and where it does not. This makes it easy to see what's wrong and tweak things to make it work better. You might even want to format your data in csv format so you can add it to your plots more easily.
- You might want to smooth your data first, to remove bumps or jitters that could mess up your algorithm.
- There's a lot of information on step-detection and heading/direction estimation online. Feel free to use it (but write your own code)

Milestone 2 Deliverables:

- You will submit a **report.pdf** summary of what you did to Gradescope. The template of the report is provided on the next page. Please refer to *Assignment deliverables for Lab 8 Milestone 2 - Sensor Processing* for more details about the report.
- Submit all your code (source files used to process the data and generate the plots) to the GitHub repository in GitHub Classroom and Gradescope. Please accept the **invitation to Milestone2 GitHub Submission**.

Conclusion

Great job on getting through this lab. Sensor processing plays an integral role in our daily lives, from fall detection to health monitoring. Exploring various algorithms and techniques is not only fascinating but also offers the opportunity to apply your knowledge to real-world scenarios.

Submission

Congratulations on finishing this CS407 Lab! There are 2 steps to submit your assignment,

1. Submit all your Milestone1 Deliverables to both GitHub classroom and Gradescope.
2. Submit all your Milestone2 Deliverables to both GitHub classroom and Gradescope.

Your score for this assignment will be based on your **"active"** submissions to Milestone1 and Milestone2 Deliverables made prior to the deadline of Due: **12:00PM CT on December 2nd**.

Please note that, as there will be no demo for this lab during office hours, you must submit the assignment strictly before the deadline. Penalties to your lab's grade are applied if your lab is not submitted before Due: **12:00PM CT on December 2nd**.

Assignment Deliverables for Lab 8 Milestone 2 - Sensor Processing

You should follow the following template to create a report for Milestone 2.

name1, `name1@wisc.edu`, **githublogin1**
name2, `name2@wisc.edu`, **githublogin2**
name3, `name3@wisc.edu`, **githublogin3** [OPTIONAL]

1 Milestone 2 Part 1: Understanding Sensor Data Errors

In this section, you will show

- A single plot showing the acceleration and the noisy acceleration.
- A single plot showing the speeds that would be obtained using the actual acceleration and the noisy acceleration values respectively.
- A single plot showing the distance traveled that would be obtained using the actual acceleration and the noisy acceleration values respectively.
- Report the final distances calculated using the two different accelerations.
- What is the difference between the two estimates?

2 Milestone 2 Part 2: Step Detection

2.1 Data Preparation

In this initial phase, your objective is to identify the most appropriate time-series dataset for step detection. Specifically, determine which column(s) from WALKING.csv will be utilized. For each column selected, show a graph depicting the raw data and the data after applying a smoothing technique. The graphs must have the timestamp on the x-axis and the respective values on the y-axis. This will visually articulate the effectiveness of your smoothing algorithms.

Detail the smoothing process by providing the relevant code snippets and an accompanying explanation. Ensure that the code is identical to the version submitted on GitHub. If an interactive tool was employed to process the data, include screenshots to facilitate reproducibility.

In summary, you will show

- A single figure for each timeseries you choose, showing the raw data and the smoothed data.
- An explanation of how you smooth the data, including necessary parameters.

2.2 Step Detection Algorithm

Here, you will delineate the methodology for detecting steps. Present critical information about your algorithm, which could be in the form of annotated code snippets, clear screenshots, or structured pseudocode—these should align with your GitHub submission.

Merely presenting code will not suffice. You must elucidate the logic behind the code segments you provide. Should you define any variables as thresholds within your step detection algorithm, offer a rationale for the selected values and justify why they are deemed optimal.

Your description of the implementation must provide sufficient detail to ensure the reproducibility of your GitHub submission.

Finally, report the step count results prominently in this segment.

In summary, you will show

- An explanation of how you detect steps.
- The result of step counting.

3 Milestone 2 Part 3: Direction Detection

3.1 Data Preparation

This task mirrors the prerequisites of Section 2.1, but focuses on the TURNING.csv dataset. Display the methods used to smooth the dataset and present the corresponding visuals.

In this subsection, you will show

- A single figure for each timeseries you choose, showing the raw data and the smoothed data.
- An explanation of how you smooth the data, including necessary parameters.

3.2 Direction Detection Algorithm

This subsection is dedicated to illustrating your approach to determining the direction of turning. As with step detection, detail the employed code and supplement it with explanations. Make sure to convey the logical framework of the code in a manner that is comprehensible and well-defined.

In this subsection, you will show,

- An explanation of how you detect the direction of turning.
- The result of direction turning detection, such as the number of turns detected and the angle of each turn.

4 Milestone 2 Part 4: Trajectory Plotting

The culmination of this project involves plotting the trajectory of the walking path. Provide a graphical representation that outlines the trajectory, assuming an initial northward orientation. Ensure that the diagram encapsulates (1) The path traversed by the individual; (2) Accurate depictions of the distances covered and turning angles; (3) You can annotate these directly on the figure or describe them in the accompanying text.

Include the pertinent code and a thorough explanation of how you accomplished the trajectory plotting task. Ensure the final plot can be obtained by following the steps you took to finish this task.

In summary, in this subsection you will show:

- A diagram showing the trajectory of the path.
- A comprehensive explanation of the methods used to achieve the final plot.