Lab 2: A Simple Step Counter

Total possible points: 920 Points
Bonus points: 300 points
Minimum points required: 180 points

In this project, you will build a simple step counter application using the accelerometer data captured directly from the mobile device. By completing this project, you will gain hands-on experience working with mobile sensors, understanding the basic techniques to denoise the sensor data, and applying algorithms to extract useful information from the sensor data.

The project has three parts corresponding to a common workflow of how a mobile sensing researcher will tackle the problem. You could attempt Part 1 and Part 2 in any order, but the completion of Part 3 will depend on the previous two parts. Upon successfully completing this project, the mobile app should display step counts as the user walks around with the phone. The bonus part gives you the opportunities to practice more mobile programming concepts in the context of a step counting app.

For all parts, it is your responsibilities to demonstrate to the teaching staff the successful completion of the requirements, e.g., by including the requested deliverables, clearly demonstrating and explaining in the screen recording, voice over, and README the completion, to earn points for each rubric item.

Part 1: Step Counting Algorithm (180 Points)

In this part, you will write a simple step-counting algorithm that takes raw accelerometer data and estimates the number of steps. At the high level, we provide five accelerometer traces and step-by-step instructions to guide you through the process of turning the raw data into step counts. Please refer to the iPython notebook, as part of the starter code, for detailed instructions.

Specifically, you will follow the instructions in the iPython notebook to complete each task. Some tasks will ask you to generate plots or answer questions. Write your responses using any text editor of your choice. Please clearly label each answer using task and question IDs.

Note, if you are using VSCode, you will need to trust the iPython notebook to see the full content.

Deliverables

For this part, you should submit the following files and place them in the Part I folder.

1. The Python source code you write to answer questions in the iPython notebook. You can either directly modify the `.ipynb` notebook or write standalone Python scripts. Hint: The standalone Python script might save you some time for Part 3.
2. A writeup report with the file name clearly labeled as `part1` consists of your answers to the questions from the iPython notebook. Both PDF and markdown files are fine.

Gradings

Requirement	**Brief Description**	**Points total**
Task 1 – 1. Plotting code	Correctly generating the plots	15
Task 1 – 2. Plots	The correct plots are included	15
Task 1 – 3. Questions	Correctly answered	15
Task 2 – 1. Plots	The correct plots are included	15
Task 2 – 2. Questions	Correctly answered	15
Task 3 – 1. Bandpass filter code	Correctly generating the plots	15
Task 3 – 2. Plots	The correct plots are included	15
Task 3 – 3. Exponential moving average code	Correctly generating the plots	15
Task 3 – 4. Plots	The correct plots are included	15
Task 4 – 1. Steps counts	Steps are reported	15
Task 4 – 2. Plots	The correct plots are included	15
Task 4 – 3. Step count tables	Each passed test will get 2 points	15

Part 2: A Simple Sensor Recording Android App (140 Points)

In this part, you will implement a simple Android app that can interact with the Sensor Framework to save accelerometer data to the Android file system. This simple sensor data recording app gives you an example of how to work with Android sensor data. Upon completing this app, you will have the ability to record more accelerometer data to test your algorithm from Part 1.

You will be given a starter code that can save one type of sensor data to the Android file system. Set up the starter code in Android Studio and run it on an Android device. Make sure you understand the behaviors of this version of the app. Specifically, you will observe two key features. First, users can press the start recording button, and start walking around while holding the phone, to record the accelerometer data. Second, after the user presses the stop recording button, the resulting CSV file will be available in the `Download` directory. *If you don't have access to a physical Android phone, you can observe similar things using the AVD.*

Your task is to extend the starter code so that for each recording, it will save **both** the linear accelerometer and the accelerometer data to the file system. For simplicity, these two types of data should be saved in two different CSV files. Once complete, use the app to record some sensor data and save **two CSV files** from the same recording session for submission.

Hint: now that you can capture accelerometer data, you can go back to double-check your answer for Part 1, Task 1, Question 3!

Deliverables

For this part, you should submit the following deliverables in the Part 2 folder:

- **The debug apk with the name saveAccelerometerData.apk**. You can do so in Android Studio and the debug apk can be located in the app/build/outputs/apk/debug directory.
- **The source code of your app**. Please do not include build-related files in the submission, which can significantly increase the submission size. Similarly, you can do so in Android Studio by following the screenshot below with the *Clean Project* option.
- **A screen recording of the implemented features.** This recording should include your voice-over briefly explaining to the teaching staff how the implementation satisfies the feature requirements. Physical Android phones have a built-in screen recorder that can record both the screen and microphone. If you are using Android virtual device (AVD)[1], you could use and screen recording apps from your laptop. Please keep the recording as concise as possible.
- **Two sample CSV files that were generated by your app**. Those two CSV files should contain the linear and accelerometer data from the same recording.
- **A README file.** You need to include your team information (name and email address) and brief explanations of how you implement the required app features. Only plaintext writeups are accepted, and markdown files are also okay.

Gradings

Requirement	**Brief Description**	**Points total**
apk	Included?	10
source code	Professionalism?	10
Screen recording	Clear?	20

[1] As of 2025D, Android Studio seems to remove the support of enabling audio-recording when using AVD.

Voice over	Clear?	10
Feature: save two types of sensor data	Complete?	50
Sample CSV files	same recording session?	10
README	Clear?	30

Part 3: A Simple Step Counting App (300 Points)

Now that you have implemented a step-counting algorithm in Part 1, and a basic Android sensor app in Part 2, you are ready to put things together for a *shiny* [2]step-counting app. *Note that we do not care too much about the accuracy of the step-counting algorithm* in this part for two reasons. First, the algorithm accuracy was tested in Part 1 with the provided traces. Second, it is actually very difficult to nail down the accuracy to the point that Android has a built-in step count sensor. So, in this part, we will focus on the more general mechanisms for designing and implementing a mobile sensing app and give you some more opportunities to work with mobile programming.

Specifically, you will design and implement the step-counting app with the following key features.

1. When the user launches the app, the UI will display the step counts and the buttons to start, stop, and reset the step counts. Please refer to the screen recording of a reference app for a reasonable UI design.
2. The app will invoke the step-counting algorithm to update the step counts while the app is in the foreground.
3. If the user rotates the screen, the step counts should not be reset to the default value.

In terms of design, you can do a mobile-only architecture or a client-server architecture. It is up to you to decide which architecture is more appealing and explain your design in the README. Below, we discuss these two options at a high level to help you get started.

For the mobile-only architecture, you will need to reimplement the step-counting algorithm in Java or Kotlin. The challenge is to find a third-party library that provides the functions for filtering the signal and finding the peaks as we used in the `scipy.signal` and figure out how to use it. The one library we found is easy to use is https://jdsp.dev.

For the client-server architecture, you can implement a simple Python server, e.g., Flask, which will then reuse the algorithm you already implemented in Part 1. Additionally, since you will need to send the data from the mobile app to the server, you will also need to serialize the data. You can do so with an HTTP library, such as OkHttp or Retrofit, for handling the network connections between the Android app and the server, and use JSON to encode and decode the data.

[2] Like a shiny Pokémon, a shiny app is a special app with enhanced features. It will look different and do more stuff.

In both cases, you will need to design a mechanism to accumulate the accelerometer data and then pass the data to the algorithm, which will then return the detected step counts. Each time new step counts are detected, the UI should be updated to reflect the total step counts.

We have provided a starter Android project (inside the Part 3 folder) that consists of basic UI, the recommended practice of ViewModel and LiveData to manage the step counts, as well as the library dependencies inside the `build.gradle.kts` file. The app will compile and run without any problem, but it does not detect any meaningful steps.

Additionally, we have also included an example python-based server that expects data in JSON format and converts to pandas DataFrame. If you want to use and extend the server code to implement the client-server architecture, you will need to work with the server specification. Please refer to the README.txt inside the Part 3/server directory for more detail.

*You are **not required** to use provided starter codes as long as your app satisfies the three features described above. But if you are looking for an easier way to get started, the starter Android project should be most useful for implement the mobile-only architecture. See the starter code for more details.*

Deliverables

For this part, you should submit the following deliverables:

- **The debug apk with the name stepCounter.apk**. You can do so in Android Studio and the debug apk can be located in the app/build/outputs/apk/debug directory.
- **The source code of your app**. Please do not include build-related files in the submission, which can significantly increase the submission size. Similarly, you can do so in Android Studio by following the screenshot below with the *Clean Project* option.
- **A screen recording of the implemented features.** This recording should include your voice-over briefly explaining to the teaching staff how the implementation satisfies the feature requirements. Please keep the recording as concise as possible.
- **A README file.** You need to include your team information (name and email address) and brief explanations of how you implement the required app features. Only plaintext writeups are accepted, and markdown files are also okay.

Gradings

Requirement	**Brief Description**	**Points total**
apk	Included?	10
source code	Professionalism?	10
Screen recording	Clear?	20
Voice over	Clear?	10

UI design	Reasonable improvement upon starter code?	100
Step counting algorithm integration	Complete?	100
Step counts: screen rotation	Persist?	20
README	Clear?	30

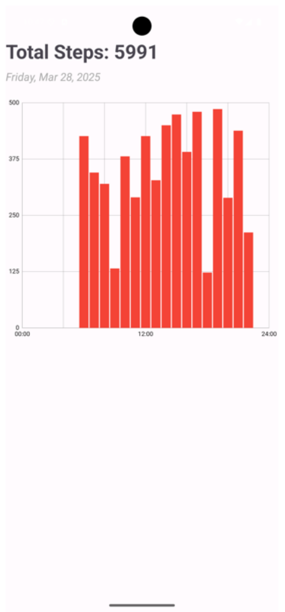
Bonus Part: Charting The Steps (300 Points)

Now that you have a functional step counter app, you could make it even more useful so that the user can save the recorded steps and visualize the step data in an intuitive way. The main learning objectives for this part is data storage, visualization, and again lifecycle-aware UI data.

The required features for this part are as following. More visual details can be found in the screen recording of the reference implementation we supply.

1. The main screen should display the step data for the current date.
2. The step data should be saved to the database. At the minimal, your database's table should consist of the timestamp and the associated step data, to support the features described below. We recommend you using the Room persistence library to work with the on-device SQLite database. You can get started from this official documentation: https://developer.android.com/training/data-storage/room
3. While in the main screen of the app, the user should be able to view the hourly aggregate of the step counts. Figure 1 shows a reasonable design and implementation. Your app's UI does not need to match exactly as what were shown in Figure 1. But it should look visually pleasing to get the full points for the UI. You can design the UI and use any charting library as you see fit. A good charting library is: https://github.com/PhilJay/MPAndroidChart
4. The user should be able to interact with the chart to view the hourly aggregates of different days. For example, if the default is to display the view for current day, swiping left will bring up the previous day's view and swiping right will still stay in the current view. Hower, if the user is in the current view, then swipe left first, the user should be able to swipe right to return the view. To demonstrate this feature, you should have step count data for multiple days---you can programmatically create dummy data and insert them to the database. One way to implement the swipe is to use a GestureDetector: https://developer.android.com/reference/android/view/GestureDetector
5. When the user swipes left to view the previous day's step counts and then rotates the screen from the portrait to landscape, the same step counts should be displayed. That is, the app should not reset the step counts to be the default option.
6. Finally, when the app is in the landscape mode, you should make sure all the UI information is properly displayed. In Figure 2, we show a working example of the landscape UI. If the BarChart or the TextViews were partially hidden, then it is considered not properly displayed.

Although we recommend you integrate the features for this part on top of Part 3 so you get to practice working with larger codebase, but doing so is not required to get all the points for this part. This flexibility is built in so to allow more students to attempt this part, even without having completed previous parts.



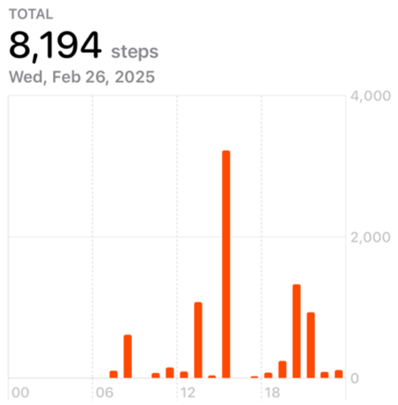


Figure 1: (Left) A screenshot of the reference app displaying the hourly step counts. (Right) A screenshot of the iOS Health app, which is what the reference app's UI design based.

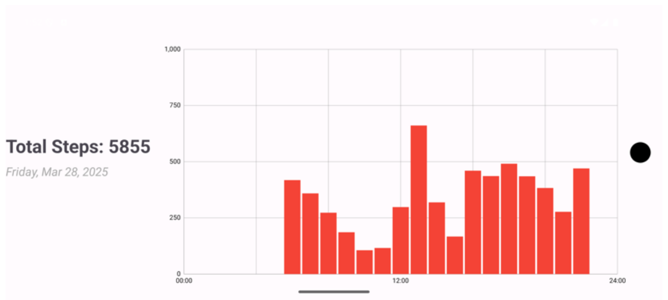


Figure 2: A screenshot of the reference app displaying the hourly step counts in the landscape mode. Note the three widgets, i.e., the two TextView and the BarChart, are correctly displayed.

Deliverables

For this part, you should submit the following deliverables:

- **The debug apk with the name stepCounterChart.apk**. You can do so in Android Studio by following the screenshot below. Afterwards, the debug apk can be located in the app/build/outputs/apk/debug directory.

- **The source code of your app**. Please do not include build-related files in the submission, which can significantly increase the submission size. Similarly, you can do so in Android Studio by following the screenshot below with the *Clean Project* option.
- **A screen recording of the implemented features.** This recording should include your voice-over briefly explaining to the teaching staff how the implementation satisfies the feature requirements. Please keep the recording as concise as possible.
- **A README file.** You need to include your team information (name and email address) and brief explanations of how you implement the required app features. Only plaintext writeups are accepted, and markdown files are also okay.

Gradings

Requirement	**Brief Description**	**Points total**
apk	Included?	10
source code	Professionalism?	10
Screen recording	Clear?	20
Voice over	Clear?	10
Feature 1: persistence with database	Complete?	60
Feature 2: hourly step counts visualization for current date	Visually pleasing?	60
Feature 3: left swiping support	Complete?	20
Feature 4: right swiping support	Correct?	30
Feature 5: screen rotation	Correct step count?	30
Feature 6: landscape UI	Properly displayed?	20
README	Clear?	30

Checkpoint Contributions

Students must submit work that demonstrates substantial progress towards completing the project on the checkpoint date. Substantial progress is judged at the discretion of the grader to allow students flexibility in prioritizing their efforts. For this assignment, completion of either Part 1 or Part 2 will be considered as making substantial progress. Projects that successfully submit a checkpoint demonstrating significant progress will receive 10 points.

Errata
Report errors by emailing tian@wpi.edu with the subject line *[CS4518] Lab [Number]*. Thank you for taking the time to help us improve our courses.