**INSTRUCTIONS FOR INSTALLING NODE ON WINDOWS:**

Download node from : <https://nodejs.org/en/download/>

Install the appropriate setup file for windows

Restart your computer.

After restart, open CMD and type:

**node -v**

the output should provide you with the current version of node installed on your PC.   
Something like: **V6.9.1**

This will confirm that node is successfully installed on your system.

**INSTRUCTIONS FOR EXECUTING THE CODE:**

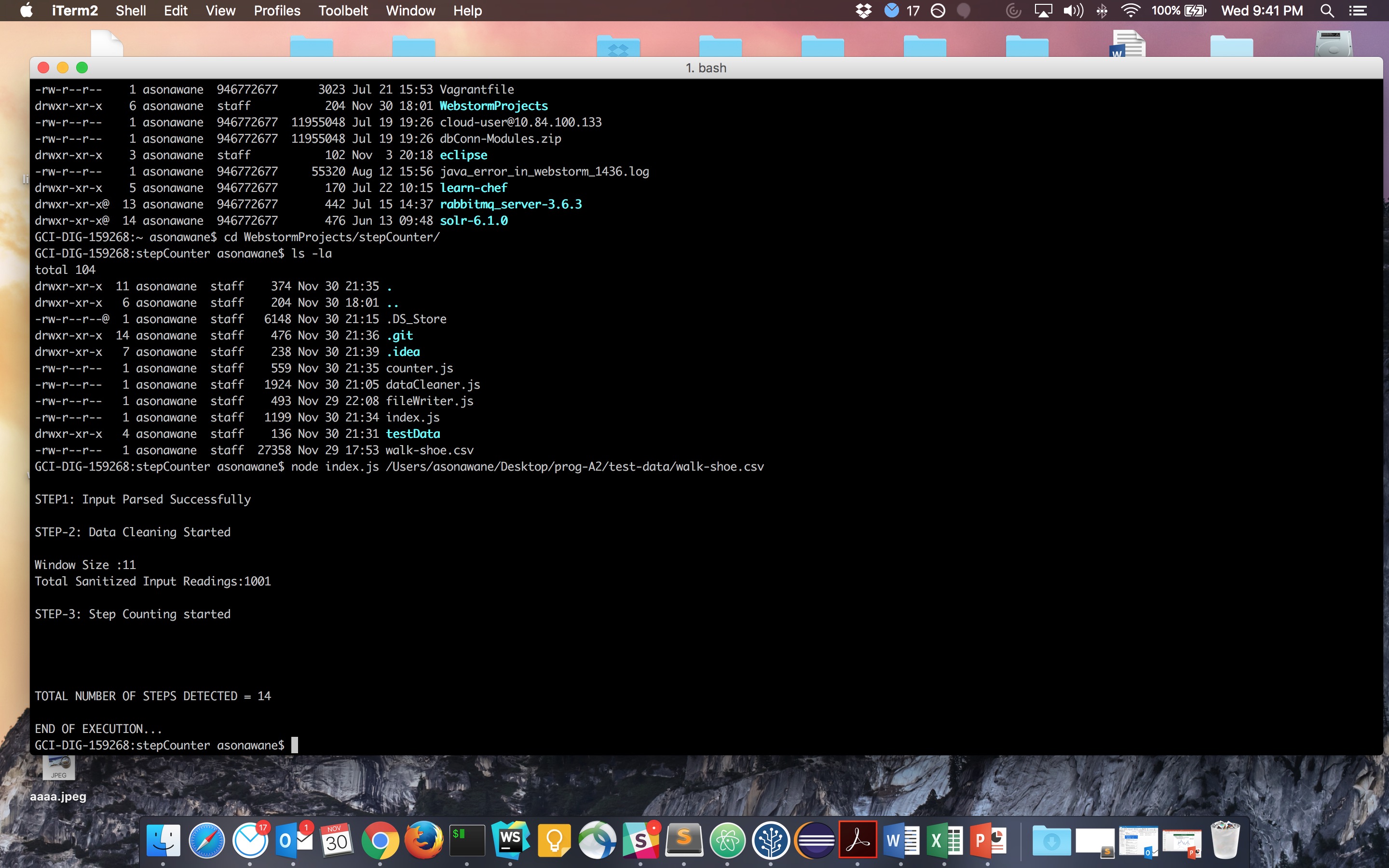
1. Navigate to this folder using terminal
2. Type the following command and hit enter to start the execution:

“node index.js ‘absolute-path-to-the-input-file’ “ ( a sample input file can be found in the ./input-file folder).

Example:

**node index.js /Users/asonawane/Desktop/prog-A2/test-data/walk-shoe.csv**

this will generate an output like this:



**Description of the approach/algorithm that is implemented for step**

**counting.**

I have used **Peak Detection** algorithm for counting the steps in the given .csv files

The central idea here is that the accelerometer reading will definitely contain noise which needs to be taken care of before we can start detecting the peaks.

We achieve this by sliding a variable sized window over the input reading and updating the value at the center of the window as an average of all the values lying inside the window and then storing it in a separate array.

This cleaned data is then processed and steps are counted.

The implementation goes on as follows:

There are THREE distinct files interlinked and each of them is explained below:

**Index.js:**

This is where the execution start from. This file takes as input, an absolute path to the .csv file via terminal/ cmd.

It initializes 4 arrays for storing the accelerometer readings for each axis along with the time.

We are sanitizing the input file here by,

1. Removing the first row which acts as an heading for each column.
2. Removing the last blank row which is appended at the end of each csv file.

After sanitizing the input, we sort and store the input in the respective arrays.

Finally, I am passing one of the X, Y or Z axis array to the dataCleaner.js for applying the data smoothening to smoothen the data.

The selection of which axis value is passed to dataCleaner is made by manually plotting all the three accelerometer axis readings on a graph and observing the peaks.

**dataCleaner.js**

Here we declare a window size (an odd number) for computing the average and assigning it to the value at the center of the window.

We store the updated values in a new array so that the updated value does not affect the cleaning process.

Since the values at the starting and ending of the signal are not updated during the cleaning and as they can also contain some peaks, we restore these values from the original input to the cleaned input This cleaned input is then passed to the counter.js

**counter.js**

Counter function works on a simple idea: each ascending pattern in the readings is followed by a descending set of readings.

Here, I have defined:

1. threshold value: Any value which is >= threshold is ‘considered’ for being counted as a step.
2. Flag (true/false): for each value which is considered for being counted as a step, the flag value is checked. If it is true, then we increment the step count and set the flag to False.
3. resetValue: The resetValue acts as a threshold for resetting the flag to True so that the next peak can be detected.

This is based on the logic that after we get a value which crosses our ‘threshold’ value and gets counted as a step, there MUST be a point where the values in our input array get reduced to an extent where they are less than or equal to the ‘resetValue’ thereby signaling the logic that it is safe to count in the next peak as a step.

Heel strike peak is easier to detect

Use a threshold to determine heel strike peaks

Count a step from heel strike to heel strike

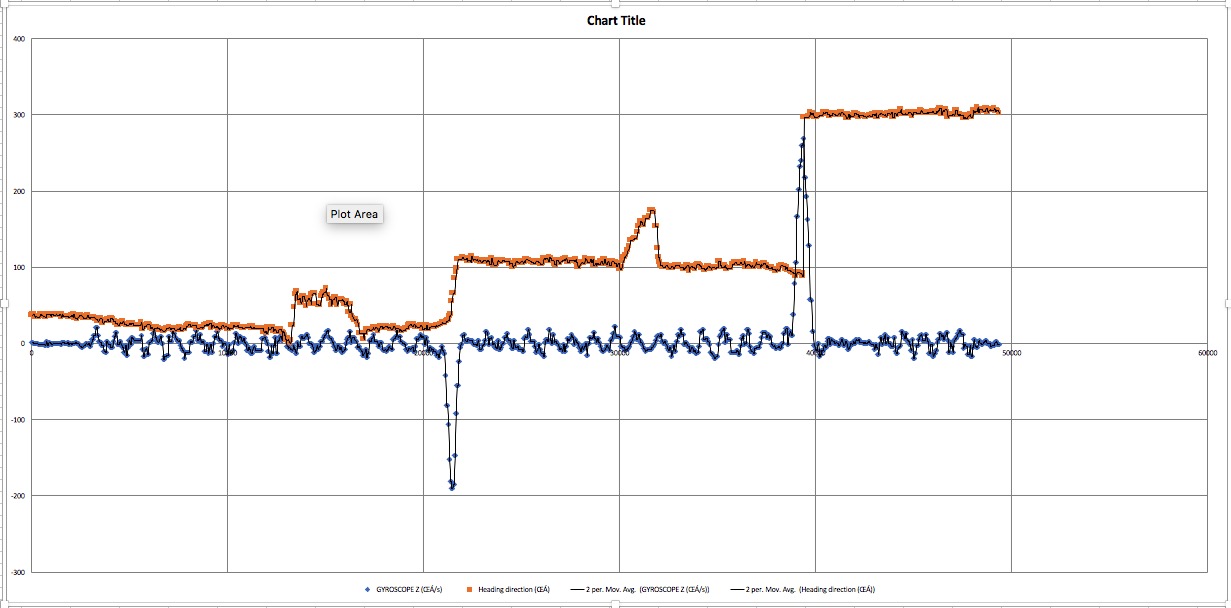


figure shows plotting of the Gyroscope Z axis (in blue) in comparison with the Heading Detection (in orange)