Group-7

DSP Project Report

April 16, 2022

-Laxmi Sreenivas, IMT2020510
-Sai Teja, IMT2020538

Question

Use GPS data to estimate the jogging path traced and distance over 10 min.

a. Identify an application that enables recording the raw data from the sensors (e.g., GPS logger). Remember to note the sampling frequency of the data.

b. Collect data from your phone while jogging. Repeat data acquisition for three different jogging trials.

c. Understand the sensor signal and process it to identify the path traced from some initial point. If the data is large, try down-sampling.

d. Estimate the jogging speed and distance from the data. Try to smoothen the estimated trajectory.

e. Find the average jogging speed in each of the trials.

f. Demonstrate what more you can do with this dataset.

g. Write a detailed report on the entire activity.

Summarizing The Problem Statement

Step-1: Find an application that logs the GPS data.

Step-2: Select a sampling frequency that provides a good trade-off between accuracy and processing time.

Step-3: Data Acquisition.

Step-4: Trace the path of all the tracks.

Step-5: Estimate various parameters from the datasets obtained.

Step-6: Future aspects.

The Application used and Sampling frequency

Application used

We used an application called **GPS logger** which is available on the playstore.

What is a GPS logger?

It is an application that helps us in recording the raw data from the sensors by logging the position of the device at regular intervals of time.

What does it record?

The GPS logger records many parameters such as latitude, longitude, speed, distance, and direction at specific intervals of time. But only latitude and longitude data from the datasets were used.

How does a GPS logger work?

GPS logger connects to a chain of satellites and follows a process called *trilateration*[1] to compute and determine the aforementioned parameters. It is a process that uses the position of multiple satellites from the *Global Navigation Satellite System Network* and their distance from them to determine the latitude, longitude, and elevation.

Sampling frequency

The sampling frequency of the data is set to 1Hz.

Why this frequency?

We have two requirements: The first is measuring speed with decent accuracy & the second is minimizing samples for faster processing.

- On average, we cover 0.89 m to 1.34 m in a second. For instantaneous speed to be accurate up to at least 1 decimal in m/s, the sampling rate is chosen to be 1 Hz. (not oversampling)

- Adding to the above argument, the speed variation within a second of jogging is not significant. We are not losing much data. (not undersampling)

Data Acquisition

- The data was collected over three different jogging tracks, each for a duration of 10 minutes with the help of GPS logger.
- This data was stored and exported in a text file to process it further.
- Appropriate measures were taken to minimize the effects of waggling.

Path Tracing

- The processing of the data acquired is done using a simple python script.
- And the tracing of this processed data is done with the help of the turtle module.

Code

Prior To Processing

```
15  #Collecting Only Position Data
16  for line in listOfLines:
17      data = line.split(",")
18      coordinates.append([float(data[2]), float(data[3])])
```

The raw data in the text file is first read and stored in a list of lists. Each sublist has two entries in it, the first being latitude and the second being longitude.

Processing The Data

```
for i in range(1,len(coordinates),stepSize):
    #Co ordintates of Tail & Head
    x1,y1 = coordinates[i-1]
    x2,y2 = coordinates[i-0]

    #Calculating Individual Components
    distanceX = (x2-x1)*111045
    distanceY = (y2-y1)*87870.18

    #Final Vector Length & Angle
    totalDistance = math.sqrt( distanceX**2 + distanceY**2)
    angleCovered  = math.atan2(distanceY,distanceX)*(180/math.pi)
```

This list of coordinates is then iterated. In each iteration, the linear and angular displacements between two adjacent points are calculated.

- *distanceX* and *distanceY* in the snippet refer to displacements in latitude and longitude.
- “111045” & “87870.18” are conversion constants from their respective degrees to meters.
- *totalDistance* is the linear displacement calculated using Pythagoras Theorem.
- *angleConvered* is the angular displacement (in degrees), using trigonometry.

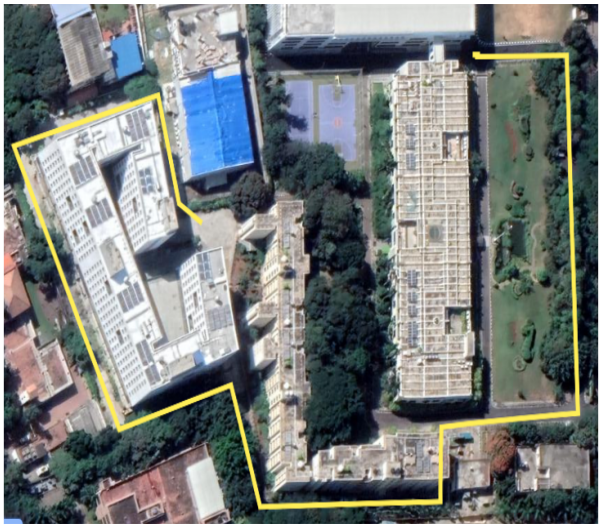
Tracing The Path

```
#Adjusting The Angle
if angleCovered >= 0:
    path.setheading(angleCovered)
else:
    path.setheading(360+angleCovered)

    #Drawing The Line
screeHeight = turtle.screensize()[1]
path.forward((totalDistance*screeHeight/(1.78*60*10))*3)
```

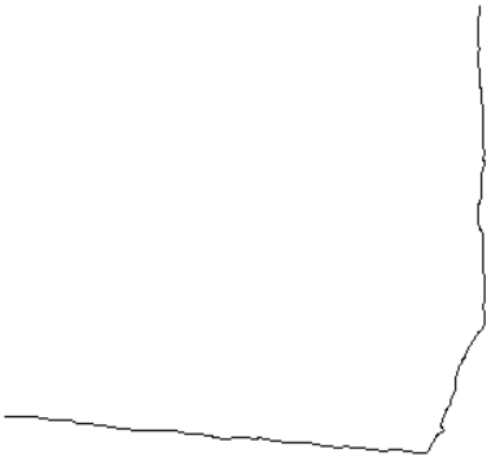
The path is then drawn using the turtle module. The plot is drawn with the starting point of the track as its origin. The *if-else* branch adjusts the angle of the turtle. The *forward* function moves the turtle from the previous point to the current point in the specified direction. (All the constants are scaling factors)

Traces

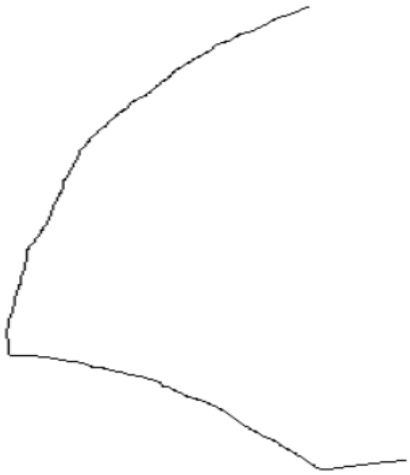
The images on the left side are the actual jogging tracks (using Google Maps) and the images on the right side are the traces of the jogging tracks obtained by processing the datasets.



Jogging Track 1



Jogging Track 2



Jogging Track 3

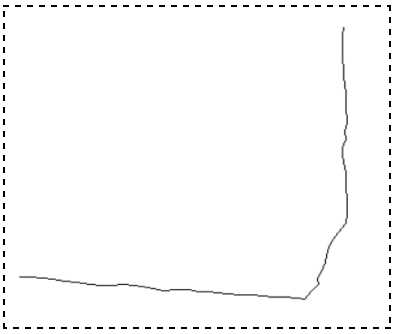
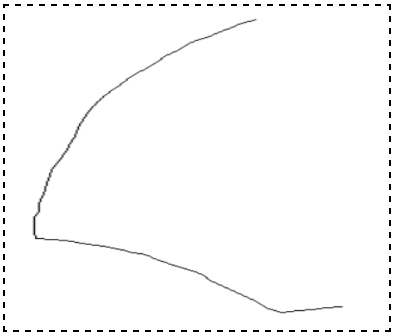
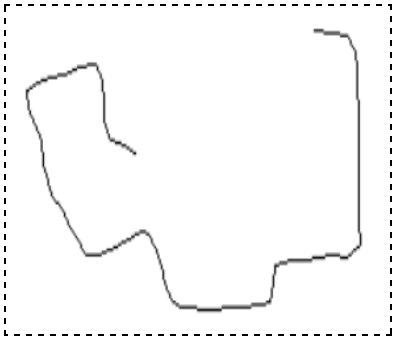
Option To DownSample

```
8   #Choosing The Decimation Level
9   stepSize = int(input("Decimate The Input By Factor of : "))
```

A provision to decimate the input data has been made. It only considers data points whose indices are multiples of the *stepSize*.

Traces after DownSampling

The following are the plots obtained after decimating the data by 10.

Observations

By comparing the downsampled figures with the original figures, we notice that the plot smoothens.

Sources of Error

The following are the reasons for the plots to be irregular compared to those on the google maps:

- Inaccuracy in the readings of the sensor.
- The lines on the map are plotted using straight lines but the path traveled may not always be a straight line.
- Waggling the phone while jogging.

Estimating Distance, Speed, and smoothening it

Prior To Processing

```python
def readContents(fileName):
    #Opening File & Reading Contents
    rawData = open(fileName, "r")
    listOfLines = rawData.readlines()[1:]

#Collecting Only Position Data
    coordinates = []
    for line in listOfLines:
        data = line.split(",")
        coordinates.append([float(data[2]), float(data[3])])

    return coordinates
```

Contents of the file are read and a list of coordinate data is made.

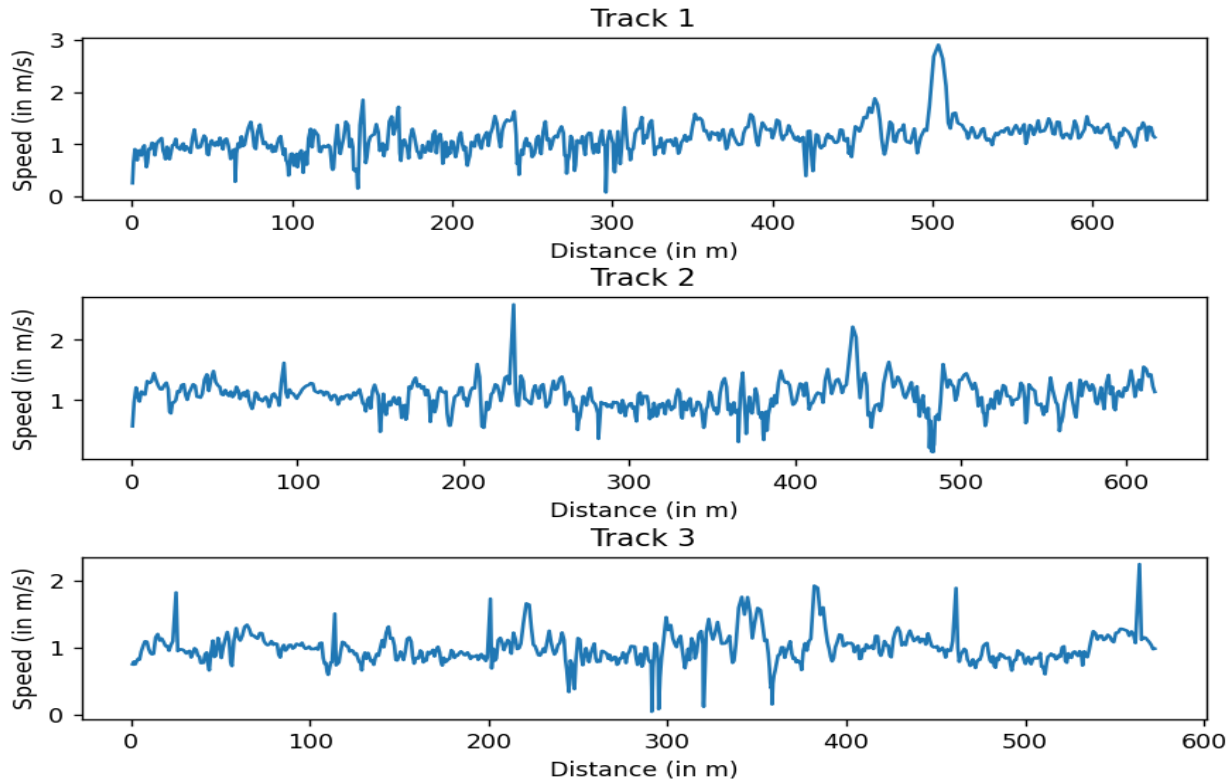
Processing The Data

In each iteration, the linear displacement between adjacent points is calculated (Similar to the processing done in path tracing). Using this, instantaneous speed *speedT* in that interval is determined. A variable *totalDistance* keeps track of the total distance covered up till this point. With *speedT* & *totalDistance*, we fully get the information of the motion at that moment.

```python
def processData(coordinates):
    #Sensor Specific
    totalDistance = 0
    timeInterval = 1

    speeds,distances = [],[]
    for i in range(1,len(coordinates)):
        #Co ordintates of Tail & Head
        x1,y1 = coordinates[i-1]
        x2,y2 = coordinates[i-0]

        #Calculating Individual Components
        distanceX = (x2-x1)*111045
        distanceY = (y2-y1)*87870.18

        #Distance Covered in The Interval (1 second in our Case)
        distanceT = math.sqrt( distanceX**2 + distanceY**2)
        totalDistance += distanceT
        speedT = distanceT/timeInterval

        #Collecting Instaneous Data
        distances.append(totalDistance)
        speeds.append(speedT)

    return [distances,speeds]
```

Estimating the jogging speed and distance

DataSet Plots

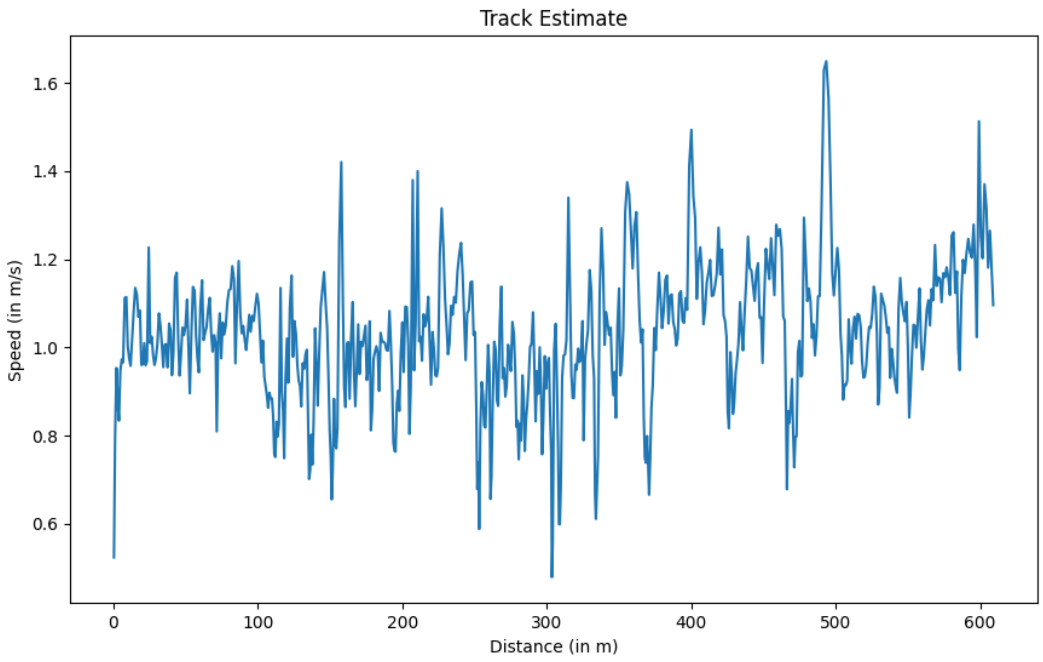


Code to estimate

```
#Estimated Distance
estimatedDistance = []
minDistValues  = min([len(x[0]) for x in dataSets])
for i in range(minDistValues):
    estimatedDistance.append((dataSets[0][0][i]+dataSets[1][0][i]+dataSets[2][0][i])/3)

#Estimated Speed
estimatedSpeed = []
minSpeedValues = min([len(x[1]) for x in dataSets])
for i in range(minSpeedValues):
    estimatedSpeed.append((dataSets[0][1][i]+dataSets[1][1][i]+dataSets[2][1][i])/3)
```

The estimated *speed vs distance* curve is the point-point average of *speed vs distance* plots of all the data sets. This is done under the assumption that the variation in distance covered for the same time interval is approximately the same for all the data sets.

Estimated Plot



Smoothening the estimated trajectory

Available Filters

- Moving Average Filter
- Savitzky-Golay Filter
- Peak Envelope Filter

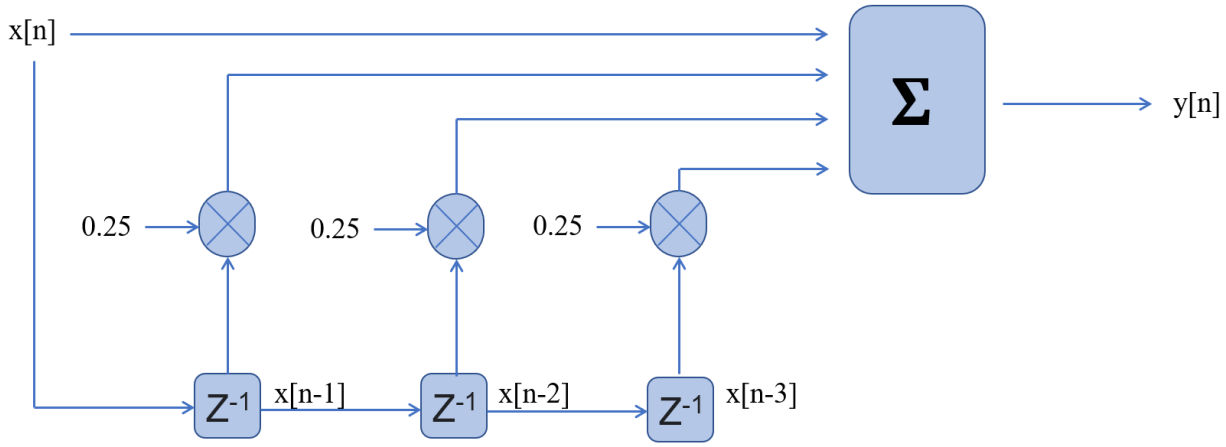
Filter used

To smoothen the estimated trajectory, we used a **Moving Average Filter (MAF)**. It is a simple low pass filter with a finite impulse response.

How does it work?

- It takes *l* samples of input and then the average of all those points is taken to produce a single point.
- So, the output of the filter can be represented as $y[n] = \frac{1}{l}\sum_{k=0}^{l-1} x[n-k]$
- The smoothness of the graph is proportional to the number of samples taken as the input i.e., *l*.
- For instance, consider a 4-point MAF. It works as follows:

$$y[n] = (0.25)\,(x[n] + x[n-1] + x[n-2] + x[n-3])$$



Advantages

- It is optimal for reducing random noise while retaining a sharp step response.

- Provides a better SNR.

Limitations

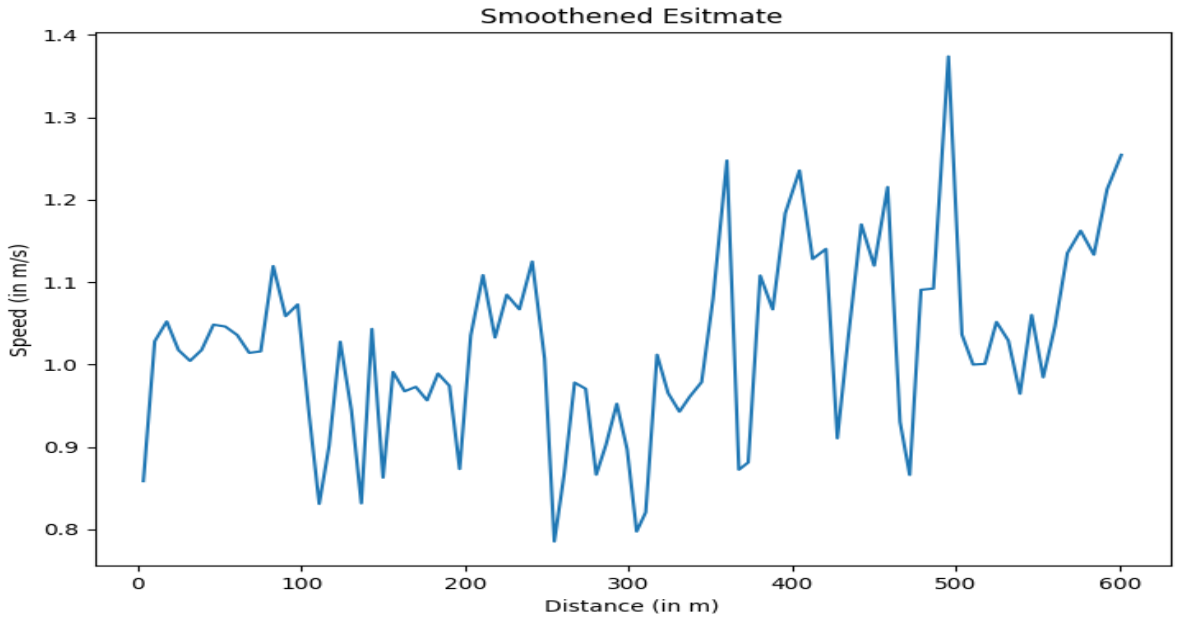
- The size of the MAF is proportional to the delay of the output. So, if the parameter *l* is significantly large, then the computation slows down due to the increased number of arithmetic operations and as a result, the output might be delayed compared to the input.

Implementation

```
45    def movingAverageFilter(dataSet,n):
46        distance,speed = [],[]
47
48        for i in range(n, len(dataSet[0]), n):
49            speed.append(sum(dataSet[1][i-n:i])/n)
50            distance.append(sum(dataSet[0][i-n:i])/n)
```

dataset contains a list of distances and a list of speeds. In each iteration, we take the average of n consecutive speeds and append it to the list *speed*. We repeat the same process for each distance in the list *distance*.

Smoothened Estimate



Calculating the Average Speed

The average speed in each of the trials is calculated using a python script as follows:

```
63  for i in range(1, len(coordinates)):
64      x1,y1 = coordinates[i-1]
65      x2,y2 = coordinates[i-0]
66
67      distanceX = (x2-x1)*111045
68      distanceY = (y2-y1)*87870.18
69
70      distanceT = math.sqrt(distanceX**2 + distanceY**2)
71      totalDistance += distanceT
72
73  averageSpeed = totalDistance/len(coordinates)*timeInterval
74
75  print("Average Speed(in m/s) = ", averageSpeed)
```

While traversing through the list of coordinates, the distance between every two successive points is calculated and is added to the *totalDistance*. This *totalDistance* is then divided by the total time of the journey to give us the value of the *average speed.*

What more can we do?

Using the data acquired, we can also compute the following parameters apart from those mentioned above:

Calculate the number of calories burned[2]

```
def burnedCalories(bodyWeight=62,time=10,MET=7):
    return ((bodyWeight*time*MET)/200)
```

Number of Calories Burned = (Time × MET × Body Weight) ÷ 200

- The default time was set to be 10 min, but it can be overwritten.
- The average body weight was taken to be 62kg but the user can overwrite this value.
- MET - Metabolic Equivalent of a Task - It has a value of 7 milliliters[3] of oxygen consumed per kilogram (kg) of body weight per minute while jogging.

Calculate the number of footsteps[4]

```
def calculateSteps(distance, stepSize=0.762):
    return (distance//stepSize)
```

1. Step length is defined as the distance covered with each step forward.
2. The number of footsteps is calculated as (Total distance covered) ÷ (Step Length)
3. The step length of a typical human being is considered 2.5 feet ≡ 0.762 meters.
4. So, the equation can be rewritten as Number of footsteps = (Total distance covered) ÷ (0.762)

Here is the link to all our codes

Bibliography

[1]

Murphy, W. and Hereman, W., 1995. Determination of a position in three dimensions using trilateration and approximate distances. *Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, Colorado, MCS-95*, *7*, p.19.

[2]

https://www.medicinenet.com/how_to_calculate_calories_burned_during_exercise/article.htm

[3]

https://www.omicsonline.org/articles-images/2157-7595-6-220-t003.html

[4]
https://www.uwyo.edu/wintherockies_edur/win%20steps/coordinator%20info/step%20conversions.pdf