

# **INDOOR POSITIONING SYSTEM USING DEEP LEARNING**

**Clyde Noronha** (190508)

**Mustafa Zaki** (190532)

**Shane Gracias** (190485)

**Ruman Mulla** (190498)

Project report submitted in partial fulfillment of the requirements for the degree of

**BACHELOR OF ENGINEERING**

**Branch: COMPUTER ENGINEERING**

OF GOA UNIVERSITY



**January 2020**

**COMPUTER ENGINEERING DEPARTMENT**

**GOA COLLEGE OF ENGINEERING**

(GOVERNMENT OF GOA)

**FARMAGUDI, PONDA, GOA - 403401**

# GOA COLLEGE OF ENGINEERING

(GOVERNMENT OF GOA)

FARMAGUDI, PONDA, GOA - 403401

## INDOOR POSITIONING SYSTEM USING DEEP LEARNING

Bona fide record of work done by

**Clyde Noronha** (190508)

**Mustafa Zaki** (190532)

**Shane Gracias** (190485)

**Ruman Mulla** (190498)

Project Report submitted in partial fulfillment of the requirements for the degree of

### BACHELOR OF ENGINEERING

Branch: **COMPUTER ENGINEERING**

of GOA UNIVERSITY

**January 2020**

.....  
**Prof. Maruska Mascarenhas**  
Faculty Guide

.....  
**Dr. J.A. Laxminarayana**  
Head of Computer Engg Dept

---

Certified that the candidate was examined in the viva-voce examination held on .....

.....  
(Internal Examiner)

.....  
(External Examiner)

# CONTENTS

CHAPTER	Page No.
ACKNOWLEDGEMENT	i
SYNOPSIS	ii
1. INTRODUCTION	1
1.1. Objective . . . . .	1
1.2. Overview . . . . .	1
1.3. Motivation . . . . .	2
1.4. Working . . . . .	3
2. LITERATURE SURVEY	6
2.1. Robust Indoor Positioning Provided by Real-Time RSSI Values in Unmodified WLAN Networks . . . . .	6
2.1.1. Task . . . . .	6
2.1.2. Methodology . . . . .	7
2.1.3. Results . . . . .	8
2.1.4. Limitations . . . . .	9
2.1.5. Conclusions . . . . .	10
2.2. AMID: Accurate Magnetic Indoor Localization Using Deep Learning . . . . .	10
2.2.1. Task . . . . .	10
2.2.2. Methodology . . . . .	10
2.2.3. Results . . . . .	12
2.2.4. Limitations . . . . .	13
2.2.5. Conclusions . . . . .	14
2.3. Improving Indoor Localization Using Bluetooth Low Energy Beacons . . . . .	14
2.3.1. Task . . . . .	14
2.3.2. Methodology . . . . .	15
2.3.3. Results . . . . .	17
2.3.4. Limitations . . . . .	18
2.3.5. Conclusions . . . . .	18
2.4. Neural Networks . . . . .	18
2.4.1. Convolutional Neural Network . . . . .	18
2.4.2. Recurrent Neural Network . . . . .	20
2.4.3. Long Short Term Memory . . . . .	21
2.5. Why Python? . . . . .	24

2.6.	Machine Learning Libraries . . . . .	24
2.7.	Anaconda Environment . . . . .	25
2.7.1.	Why Anaconda? . . . . .	25
2.8.	Jupyter Notebook . . . . .	25
2.9.	Android Studio . . . . .	26
<b>3.</b>	<b>PROPOSED WORK</b>	<b>26</b>
3.1.	Data Collection . . . . .	26
3.2.	Activity Classification . . . . .	26
3.3.	Activity Unit Classification . . . . .	27
3.4.	Moving Distance Estimation . . . . .	27
3.5.	Android Application . . . . .	28
<b>4.</b>	<b>REQUIREMENTS</b>	<b>28</b>
4.1.	Software Requirements . . . . .	28
4.2.	Hardware Requirements . . . . .	29
<b>5.</b>	<b>DESIGN</b>	<b>29</b>
5.1.	Design Flow Diagrams . . . . .	29
5.2.	GUI Mockups . . . . .	30
<b>6.</b>	<b>IMPLEMENTATION</b>	<b>32</b>
6.1.	Data Collection . . . . .	32
6.2.	Activity Classification . . . . .	34
6.2.1.	Algorithm . . . . .	39
	<b>CONCLUSIONS</b>	<b>40</b>
	<b>BIBLIOGRAPHY</b>	<b>41</b>

# ACKNOWLEDGEMENT

We would like to extend our heartfelt gratitude towards all those who have helped us learn, grow and accomplish this endeavor. Without their able guidance, help and encouragement, this project would not have been possible.

We express our sincere appreciation to our Principal, Dr. Krupashankar M.S. for providing us with all facilities. We are indebted to our professor, Dr. J.A. Laxminarayana, Head of Computer Engineering Department, for bolstering our confidence and providing full-fledged support. We are profoundly thankful to our Professor and Project Guide Prof. Maruska Mascarenhas, who has been a buttress throughout our journey and whose overwhelming encouragement and guidance helped us throughout this project. We also thank our teachers and other staff members of the Computer Engineering Department for their help and assistance. We deeply thank our families for their moral and economic support. Last but not the least; we thank all our friends and acquaintances for making this voyage possible.

# SYNOPSIS

In the current age of technology, being able to easily and accurately find out your location at any point in the world is trivial with the help of a Global Positioning Systems (GPS). However, such systems are limited by their accuracy of up to 5 meters. This means that GPS would not be of much use in the interior of buildings.

It is very common to be lost in large indoor spaces such as malls, hospitals, or office campuses. Indoor Positioning Systems (IPS) are a solution to the problem proposed and the equivalent of GPS for smaller distances. Several approaches to such systems are already present in the market, such as WLAN, WiFi, Bluetooth, and Magnetic Fingerprints. However, these approaches have limited accuracy and have a dependency on hardware or the environment.

Our approach to Indoor Positioning which incorporates machine learning, aims to provide a purely software-based approach that can be easily downloaded on the user's smartphone and be used. All the user will need to do is mark his starting point on the map, and the system will trace his path as he walks through the indoor area. Additionally, the user can also mark his destination on the map, upon which a path will be shown, and the system will update the user's position in real-time to aid in his progress towards the destination. It is a cost-effective solution which makes full use of the developments in the field of machine learning in order to solve a well known problem.

# 1. INTRODUCTION

## 1.1. Objective

The objective of this project is to provide the accurate, real-time indoor location of a user using deep learning, with the help of the accelerometer and gyroscope sensors available in smartphones.

## 1.2. Overview

The number of studies on the development of indoor positioning systems has increased recently due to the growing demands of the various location-based services. Inertial sensors available in smartphones play an important role in indoor localization owing to their highly accurate localization performance.

A global positioning system (GPS) is commonly used for navigation, but fails to reach certain locations such as tunnels or the inside of buildings. Hence, different alternative methods have been invented known as Indoor Positioning Systems. Such systems can be used at hospitals, malls, museums, etc. which will help people to locate their desired destinations faster.

Our approach for indoor positioning is based on Long Short-Term Memory (LSTM). The LSTM accurately recognizes physical activities and related action units (AUs) by automatically extracting the efficient features from the distinct patterns of the input data. Experiment results show that LSTM provides a significant improvement in the indoor positioning performance through the recognition task.

Various approaches can be used to achieve indoor positioning, some of the common methods are

- RSSI/WLAN
- Bluetooth
- Geomagnetic Fingerprinting

## 1.3. Motivation

- Current approaches based on the Received Signal Strength (RSS) using WiFi and Bluetooth have limitations, such as unreliability, high complexity, low precision and expensive hardware. Moreover, wireless signal fades rapidly in indoor environment due to fading caused by reflection, diffraction and absorption. Furthermore, the trajectory paths obtained using RSS values have high navigation error.
- The inertial sensors are a preferred option to determine the exact position in an indoor area. Compared to fingerprinting-based methods, the inertial sensor-based positioning system does not require expensive infrastructure and avoids regular time-consuming updates to the data.
- Indoor location-based services include significantly large applications such as network management and security, personal navigation, healthcare monitoring, and context awareness.
- Different research groups have developed indoor positioning systems based on different techniques such as ultra-wideband, Bluetooth, sound, visible light, geomagnetic field, and inertial systems. Among these, inertial sensor-based systems are gaining wide popularity due to the widespread deployment of micro-electro-mechanical system (MEMS) sensors in smartphones.
- As an emerging technology, the smartphone has been explored in the context of indoor positioning. The number of smartphone users in 2019 was approximately 2.08 billion, which is expected to increase to 2.66 billion in 2020. This trend motivated us to employ the motion sensors used in a smartphone.



## 1.4. Working [1]

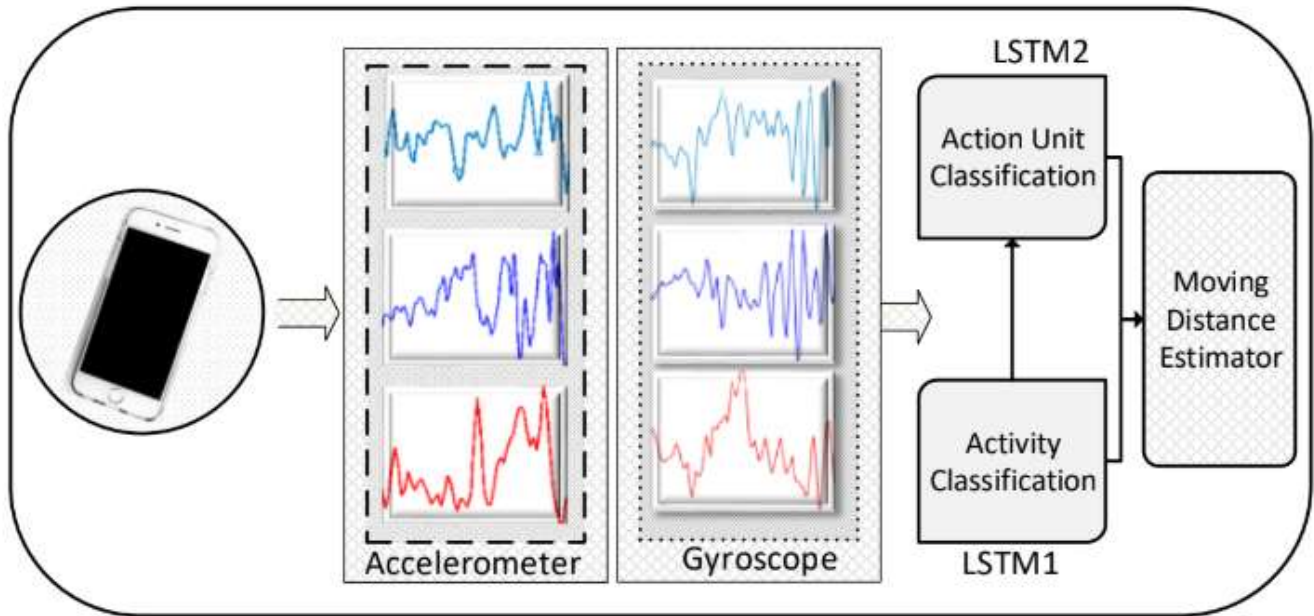


Fig 1.1.1 Block Diagram of Stages.

In our project, an LSTM network is used, which is a type of artificial neural network, to develop an indoor positioning system. The proposed system based on LSTM processes the sequential data containing the trajectory information of the pedestrians. It performs the task as follows:

### 1. Data Collection

The inertial sensors present in smartphones collect the bodily acceleration and angular velocity of the subjects as sequential data. This data consists of dynamic segments carrying six signals, (three accelerometer and three gyroscope values), of the inertial sensors. These dynamic segments carry the trajectory information of the subjects in an indoor environment.

### 2. Data Pre-processing

The pre-processing of this data is automatically done by the LSTM model, which automatically extracts the most efficient features from the data.

### 3. Activity Classification

The first stage of classification done by the LSTM model, here the recorded signal is classified into different motion states including walking, running, and stopping (i.e., standing/sitting). These motion states are called 'activities'. This is done with the help of the first LSTM.

### 4. Activity Unit (AU) Classification

During the second stage, these motion states are classified further into action units such as turning left, turning right, stepping normally, taking short steps or taking long steps. The short, normal, and long steps are denoted by SS, NS, and LS, respectively. This is done with the help of the second LSTM.

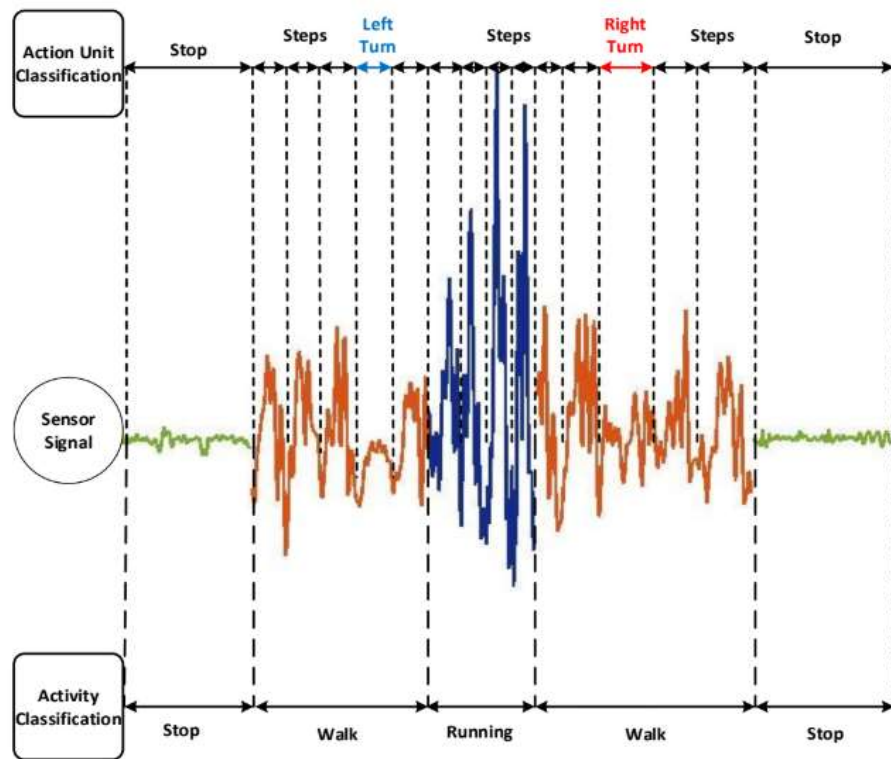


Fig 1.1.2 Accelerometer Signal and Activities.

## 5. Moving Distance Estimator

Finally, the proposed system accurately estimates the trajectory of the target in an indoor environment by selecting a suitable length according to the recognized step type of an action unit. This is done by using a Moving Distance Estimator. The moving distance estimator, shown in the figure given below, sums the different AUs and forms the accurate trajectory of the subjects in an indoor area. The moving distance estimator updates the previous indoor positioning of the participant by adding the recent recognized AUs to the participant's previous position. The moving distance estimator is given by:

$$P_K = P_{K-1} + AU_K, \quad \text{--- (1.1.1)}$$

[where  $\kappa$  shows the current time index of the position ( $P$ ) and the recognized action unit ( $AU$ ).]

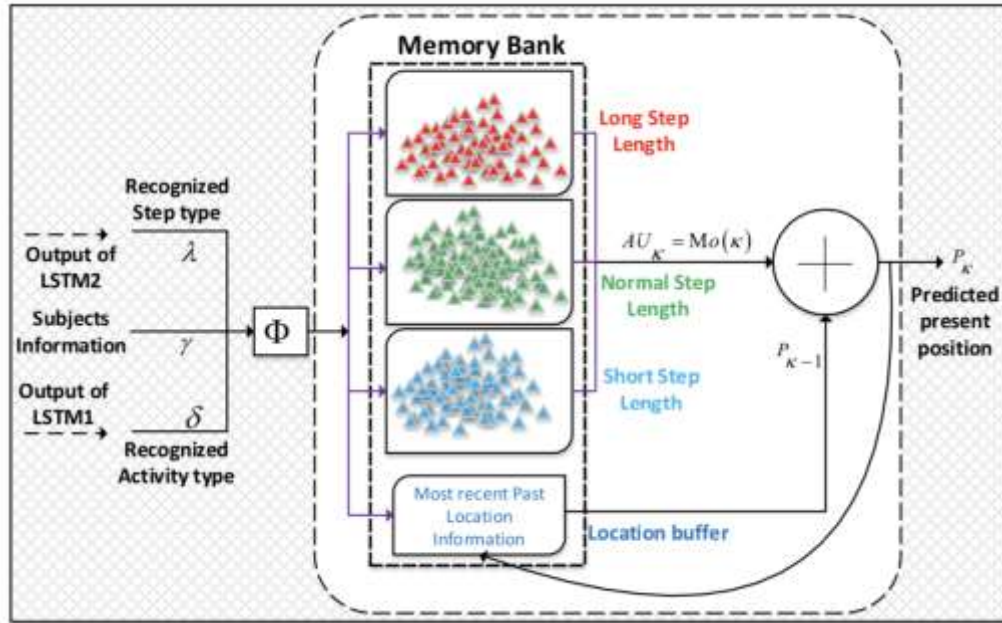


Fig 1.1.3 Moving Distance Estimator Block Diagram.

The automated feature extraction and classification at the AU levels, which are activity sub-classes, provide a higher accuracy.

## 2. LITERATURE SURVEY

### 2.1. Robust Indoor Positioning Provided by Real-Time RSSI Values in Unmodified WLAN Networks. [2]

-S. Mazuelas et al. IEEE Journal of Signal Processing, November 2009

#### 2.1.1. Task

The method in this paper is based on received signal strength (RSS) measurements, which refers to the strength of the signal as it traverses between the device and the Access Point (AP). It dynamically determines which model is the most accurate representation of the area with the help of RSS values and predicts the indoor position of a device with high accuracy.

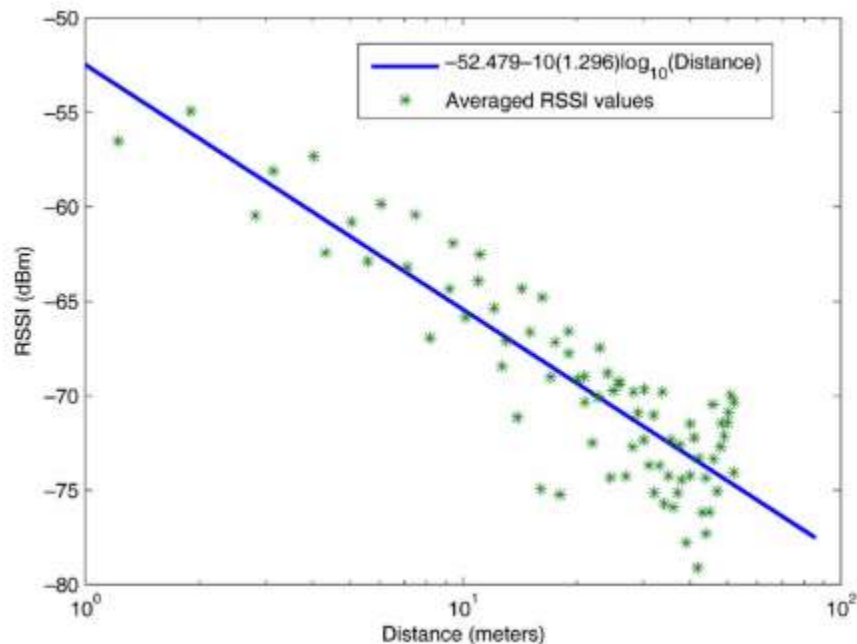


Fig 2.1.1 Relationship between distance and RSSI values in a corridor.

## 2.1.2. Methodology

The methodology proposes a pure software solution that only makes use of the RSSI values obtained by the device from the Access Points (APs) in its vicinity. It dynamically determines which model of the area is the most optimal for the propagation environment present between the device and the APs using only the RSS values obtained at that point in time.

In brief, this method is based on trilateration of the device based on the RSSI values obtained in real time. It involves the following steps:

### 1. Path Loss Exponent Estimation

The propagation environment refers to the specific environment between the device and each AP. Distance between the device and the AP causes attenuation in RSS values. This attenuation is known as path loss, and it is inversely proportional to the distance between the device and the AP raised to a certain exponent. This exponent is known as the path loss exponent, and it represents the propagation environment. Hence, we can estimate this path loss exponent from RSSI values by using particular mathematical functions.

### 2. Distance Estimation

Once the path loss exponent is estimated, the distance between the device and each AP can be obtained with the help of the following formula:

$$\hat{d} = 10^{(\alpha - \overline{P_{R_i}})/10n_i} \quad \text{-- (2.1.1)}$$

### 3. Trilateration

Trilateration is the determination of the position of a point by the measurement of distances of that point from other points. Once the distance between the device and the APs are found in the second step, we can perform trilateration to find the real-time position of the device.

### 2.1.3. Results

The proposed experiment was carried out on the second floor of the Higher Technical School of Telecommunications, University of Valladolid (Spain).

For APs, eight identical wireless broadband routers with two antennas each were used, in diversity mode, which is typically found on most IEEE 802.11 WLAN routers. APs were configured to send a beacon frame every 10 ms to constant power. APs have omnidirectional rubber duck antennas mounted.

The experiment was carried out to compare the device position values when using estimated distances by the proposed method and when using estimated distances from constant path loss exponents. The distance equation mentioned above was used to estimate distances in various environments, as shown in the table.

Environment	Walls	$n_i$	$\Delta$ Distance (Meters)	Number Measurements	Mean (dBm)	Std (dBm)
Central Corridor 1	0	1.3684	50.14	25449	0.3357	2.7194
Central Corridor 2	0	1.2584	50.93	28971	0.23	2.8546
Lab 1	0	1.3012	7.97	54579	-0.0362	2.6312
Lab 2	0	1.793	8.5	16983	0.0493	2.5144
CenCorr-Lab 1	1	2.1047	29.24	16359	-0.4559	3.1319
CenCorr-Lab 2	1	2.0005	29.41	28659	-0.2925	2.6456
Lab-Lab 1	2	2.7343	5.5	4179	-0.0069	2.3701
Lab-Lab 2	2	2.7203	6.16	9177	0.0425	1.9638

Table 2.1.1 Results of Distance Estimation

The subjects walked with the laptop along the route shown in blue in the figure below taking measurements in 121 points. In red, we can see the trilateration positions obtained by using the method proposed and, in purple, some positions obtained by using constant path loss exponents, where 50.4% of the estimated positions obtained by this method are not shown, since they lie outside this map.

Therefore, as we can see in the figure given below, the method proposed achieves comparatively a high level of accuracy. The mean of errors with the method proposed was 3.97 m with a standard deviation of 1.18 m.



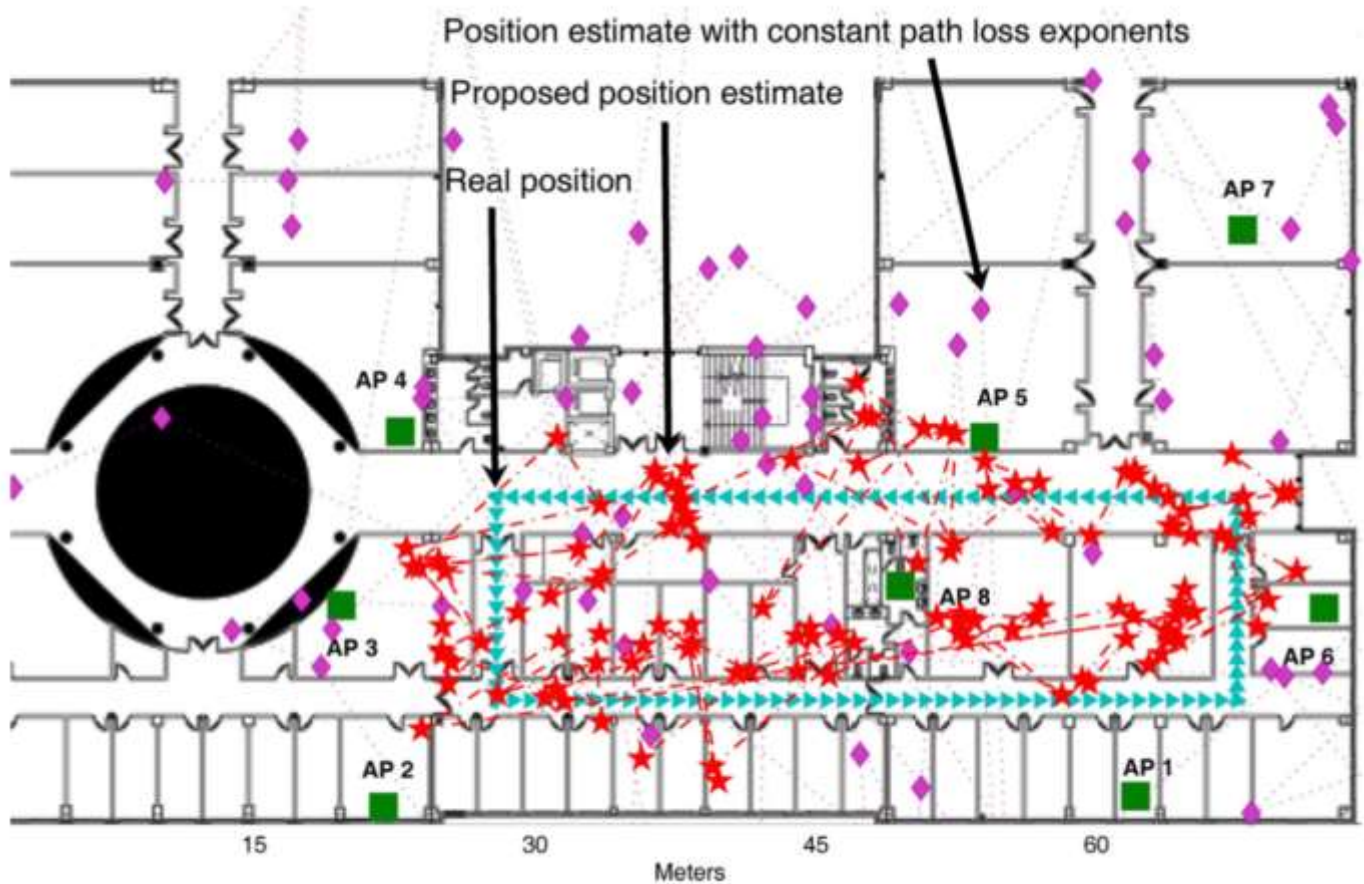


Fig 2.1.2 Actual and predicted trajectories of participants

## 2.1.4. Limitations

### 1. Hardware Requirements

WLAN routers are required for the feasibility of the proposed model, which may not be easy to procure and maintain.

### 2. RSS fluctuation

Since this method depends heavily on the RSS values to find the distances between the device and the APs, hence it is more vulnerable to changes in the RSS values which occur at large distances from the APs.

### **3. Relatively Inaccurate**

This method gives a mean error of 4m, which is relatively higher than most other indoor positioning methods. The inaccuracy can range from 4 - 15 m.

## **2.1.5. Conclusion**

This method is based on trilateration through RSSI values obtained in real time. It does not require additional calibration or fingerprinting information. Results using both simulations and measurements are performed in order to prove the reliability and suitability of the method proposed. A mean error slightly lower than 4m is obtained in a WLAN network with a device and multiple access points, without using any other tracking technique.

## **2.2. AMID: Accurate Magnetic Indoor Localization Using Deep Learning. [3]**

-N. Lee, S. Ahn, D. Han. sensors, May 2018

### **2.2.1. Task**

This paper proposes accurate magnetic indoor localization using deep learning (AMID), an indoor positioning system that recognizes magnetic sequence patterns using a deep neural network. Features are extracted from magnetic sequences, and then the deep neural network is used for classifying the sequences by patterns that are generated by nearby magnetic landmarks. Locations are estimated by detecting the landmarks.

### **2.2.2. Methodology**

1. Magnetic data collection
2. Magnetic landmark localization
3. Magnetic landmark classification
4. Localization prediction

These are the main steps of the AMID System Design.



1. The first step in the AMID system design is the collection of magnetic sensor data. This is achieved by mounting a smartphone on a robot that is maneuvered around the locations of interest. The location data is also collected, which is used to construct magnetic fingerprints by matching the previously collected sensor data with the robot's locations. In addition, inertial sensor data is collected to estimate user walking distances. All this data is pre-processed to remove the noise from the raw data and improve positioning accuracy.
2. The next step identifies the magnetic landmarks. Here, the magnetic map is created to find the magnetic landmarks. This map is constructed by interpolating the magnetic fingerprints collected in the first step. The locations of the landmarks are identified by finding the peaks in the map, shown in the figure below. These locations are stored and used for data labelling in the classification step.

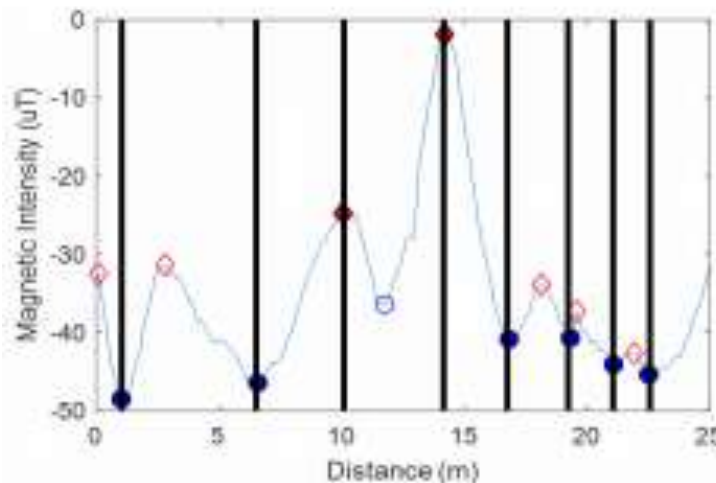


Fig 2.2.1 Magnetic Landmark Peaks

3. In the third step, we train the model with the data collected from the first step and perform classification. The points in the magnetic data (magnetic sequences) are labelled with the magnetic landmark data determined from the second step. The magnetic sequences are used as the input for the classification model.
4. In the final step, we estimate the location using the magnetic landmarks that were detected from the classification model constructed in the previous step.

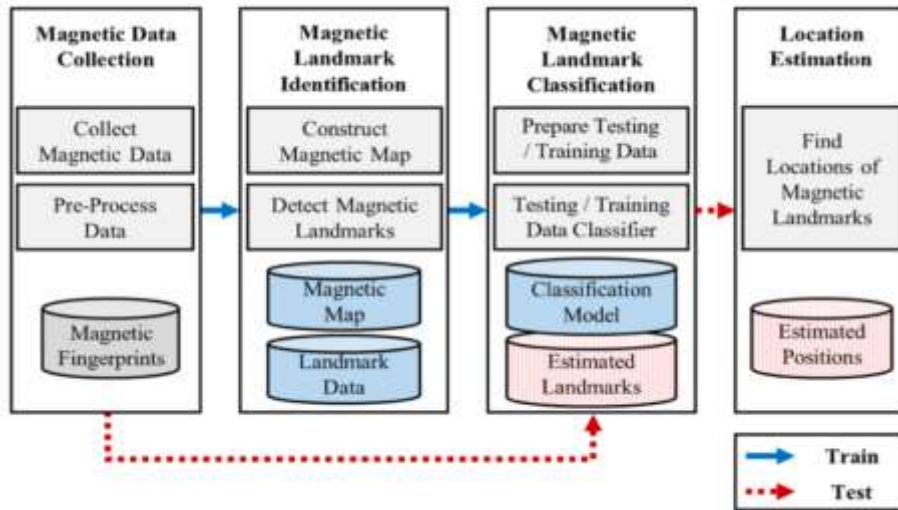


Fig 2.2.2 Steps of AMID system

### 2.2.3. Results

The experiment for the proposed system was conducted in a one-dimensional (corridor) and two-dimensional (atrium) environment. The corridor and atrium have the following sizes:

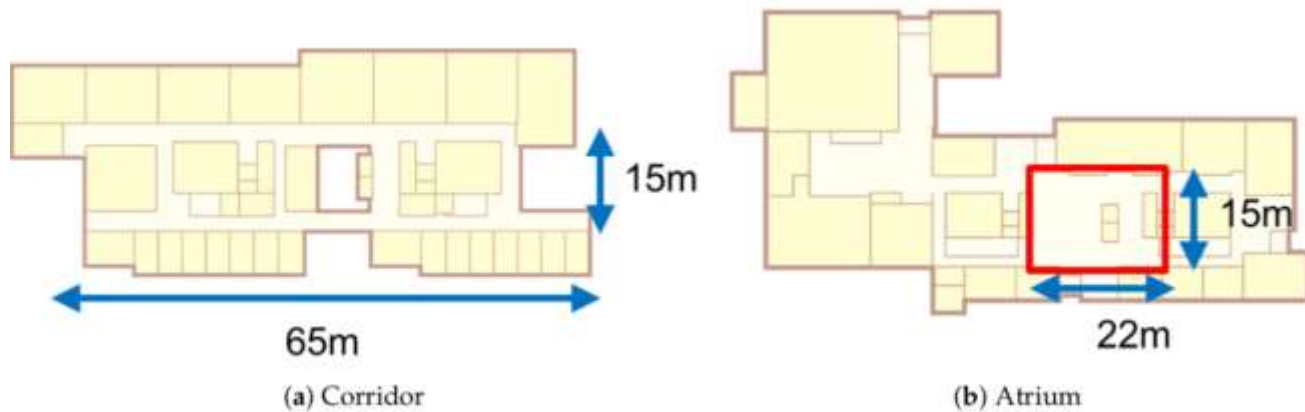


Fig 2.2.3 Actual and predicted trajectories of participants

In order to collect the training and testing dataset, the same smartphone was used to maintain consistency. The magnetic sensor of the smartphone was calibrated before data

collection by rotating it on the 3 axes. Data collection was performed over three days without the changing of the locations of ferromagnetic materials in the room.

Magnetic data used for map construction was used to train the model. For testing, 4 test paths for corridor and 3 test paths for atrium were selected. Test paths did not match training paths. The classification accuracy for the corridor was found to be 100% in the final phase, while the accuracy for the atrium was found to be 80.2%.

Based on these results, the best classification model was chosen. In order to find the error of positioning, the Euclidean distance between the user's actual location and the predicted location was found. The overall positioning accuracy is given in the table below:

Place		Corridor	Atrium
Accuracy	Mean	0.76 m	2.30 m
Precision	Within 90%	1.50 m	8.14 m
	Within 50%	0.60 m	0.90 m

Table 2.2.1 Magnetic indoor positioning results.

The average positioning error of the corridor (0.76 m) was superior to that of the atrium (2.30 m) because of the significant classification accuracy difference.

## 2.2.4. Limitations

### 1. Built-in sensor variation

Different smartphones consist of different models of built-in magnetic field sensors that vary in their sensitivities. This can affect the positioning accuracy.

### 2. Long-term variation

Long-term variations in the magnetic field magnitudes could be a cause for changes in the magnetic fingerprints and hence the magnetic peaks and landmarks, which would require additional maintenance costs to collect and re-train the classification model.

### **3. Presence of personal metallic objects**

Presence of small metallic objects (key chains, coins) in the user's pocket along with the smartphone could affect the magnetic fingerprints, once again affecting the localization accuracy.

### **4. Presence of furniture**

The other factor that would affect the magnetic signatures is the presence of ferromagnetic furniture along hallways. Unless the furniture consists of heavy metallic objects that are frequently replaced, there will not be any issue with the system.

## **2.2.5. Conclusion**

This paper proposed AMID, an indoor positioning system that estimates locations by recognizing landmark patterns using a Deep Neural Network (DNN). AMID can be used not only for one-dimensional but also for two-dimensional positioning. With the help of a well-trained classifier and a DNN, AMID achieved an accuracy of 0.8 m in a corridor and 2.3 m in an atrium.

## **2.3. Improving Indoor Localization Using Bluetooth Low Energy Beacons. [4]**

-P. Kriz, F. Maly, K. Tomas. Mobile Information Systems, April 2016

### **2.3.1. Task**

The proposed system provides basic principles of a radio-based indoor localization and improves the results of it using new Bluetooth Low Energy Technology. A distributed system is used to collect the radio fingerprints by mobile devices. Bluetooth and Wi-Fi transmitters are installed in different parts of the location to estimate the current position of the user.

## 2.3.2. Methodology

A combination of Wi-Fi access points and BLE beacons are used for localization of a stationary device. The goal is to improve the accuracy of the results using BLE beacons along with the Wi-Fi access points. This can be done using the following:

### 1. Learning Data Acquisition

The data required for learning is collected in this step. Data is collected from accelerometer, compass and gyroscope using the sensors of a smartphone for future processing. The smartphone scans signals of all available networks and beacons around the room, using which the user can create a fingerprint of the given place using an application. The application records the strengths of individual signals in a given place for 10 seconds. This recorded fingerprint is stored in the fingerprint database.

### 2. Positioning Data

In order to localize an object, we measure the fingerprint of a place where the object is using an application. The measured fingerprint is then compared with all the other fingerprints inside the fingerprint database and one or more with the highest similarity are searched. The accuracy of the localization depends on 2 factors, the first one is the quality of fingerprints saved in the database, and the second one is the algorithm used in order to calculate the similarity between the tagged fingerprint in the database with the measured untagged fingerprint.

The KNN algorithm was used to calculate the first K fingerprints using the Euclidean Distance.

Once the fingerprints are sorted based on the distances, first K fingerprints are chosen. From this we calculate the estimated position using the following formula:

$$P = \frac{\sum_{i=1}^k P_i Q_i}{\sum_{i=1}^k Q_i}, \text{ where } Q_i = \frac{1}{D_i}. \quad \text{--(2.3.1)}$$

During the measurement, the measuring device can receive the signal of the same WiFi or BLE beacon several times with different signal strength. From this set of

signals only one value is chosen for further processing- the median value.

$$X_{Tx} = \{x_{1Tx}, x_{2Tx}, \dots, x_{MTx}\} . \quad \text{--(2.3.2)}$$

### 3. System Architecture

The system uses a CouchDB to obtain data during the measurements on a mobile device. It supports searching primarily according to the keys. The Couchbase Sync Gateway allows the database to be replicated among the server and mobile devices, this feature made it easier to replicate the data from mobile devices to the server where the data is further processed.

Since a direct access to Couch Sync Gateway is not recommended due to security reasons, Apache reverse proxy is put in front of the Sync Gateway. The JavaScript application deployed to the NodeJS server provides external authentication of users for Couchbase Sync Gateway using Google accounts

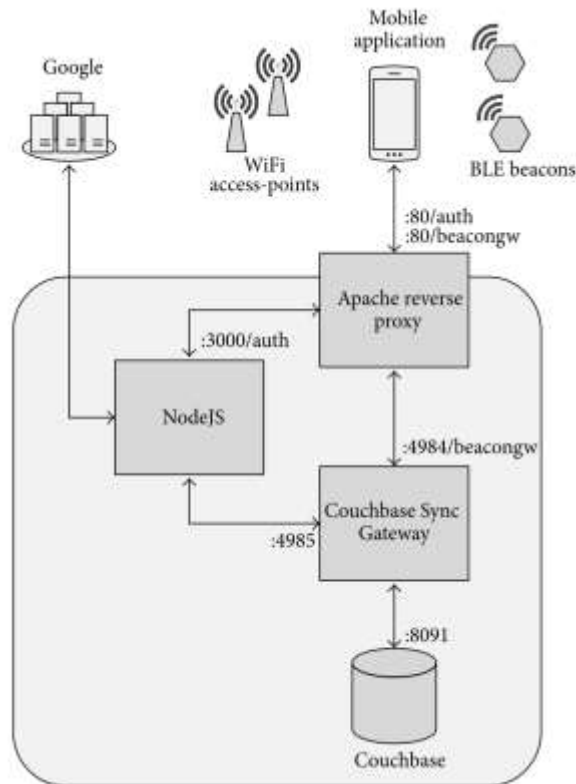


Fig 2.3.1 System Architecture

### 2.3.3. Results

A weighted K-Nearest Neighbour in special space algorithm is used to estimate the position. The results for  $k = \{2, 3, 4\}$  were similar but the highest accuracy was achieved at  $k = 2$ . As can be seen from the figure given below, accuracy of the position when using only WiFi was lower than BLE transmitters, however when WiFi and BLE transmitters were combined to localize an object, the accuracy was highest as compared to WiFi and BLE. The median accuracy improved from 1 m when using WiFi to 0.78 m when combining both the technologies.

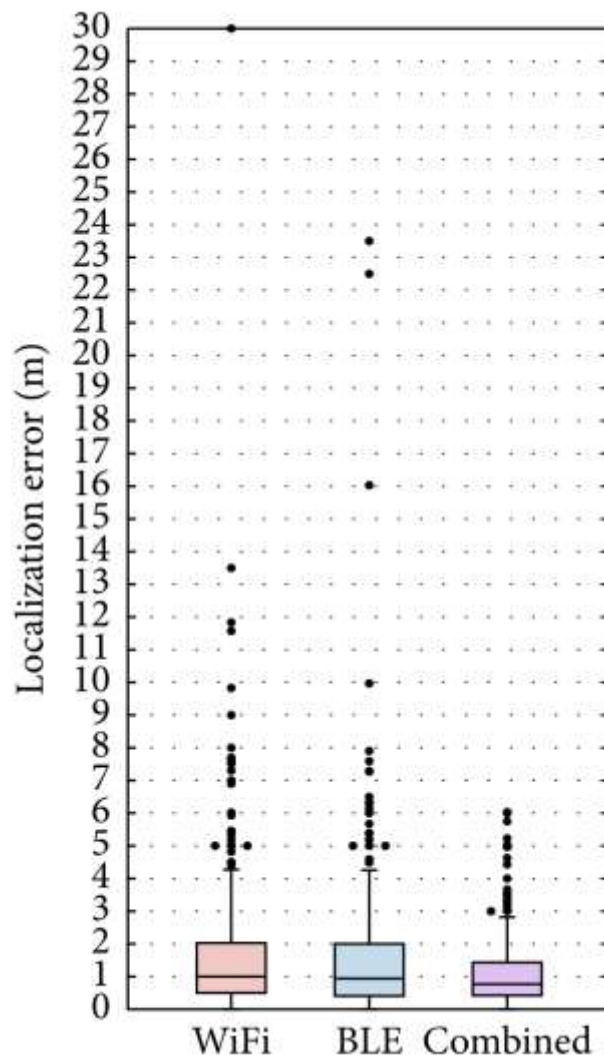


Fig 2.3.2 Results of WiFi, BLE and Combined

## 2.3.4. Limitations

1. Additional Hardware cost of using Bluetooth Low Energy Beacons and WiFi Transmitters.
2. An Application is required for Client Based Solutions.
3. The range up to which Bluetooth can effectively operate is approximately 30 m.
4. Inefficiency in relatively larger indoor spaces increases due to the positioning of the routers.

## 2.3.5. Conclusion

A new way to improve the accuracy of the results using WiFi and Bluetooth Low Energy transmitters combined was successfully implemented for the indoor localization of an object.

## 2.3. Neural Networks

### 2.3.1. Convolutional Neural Networks

In neural networks, Convolutional Neural Networks (ConvNets or CNNs) are one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used. CNNs are neural networks that share their parameters. In an image it can be represented as a cuboid having its length, width (dimension of the image) and height (as images generally have red, green, and blue channels).

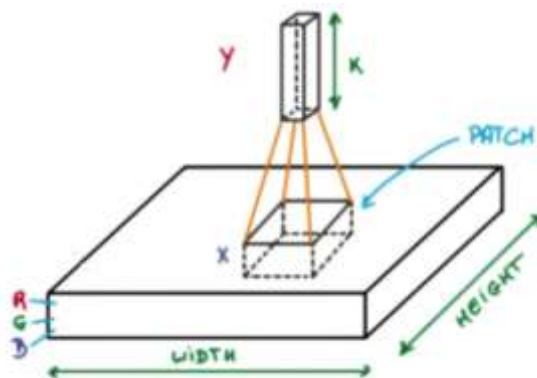


Fig 2.3.1 CNN



We take a small patch of this image and run a small neural network on it, with say,  $k$  outputs and represent them vertically. We then slide this neural network across the whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G and B channels now we have more channels but lesser width and height. This operation is called Convolution. If patch size is same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

## Layers

Let's take an example by running a CNNs on an image of dimension  $32 \times 32 \times 3$ .

- 1. Input Layer:** This layer holds the raw input of image with width 32, height 32 and depth 3.
- 2. Convolution Layer:** This layer computes the output volume by computing dot product between all filters and image patch. Suppose we use total 12 filters for this layer we'll get output volume of dimension  $32 \times 32 \times 12$ .
- 3. Activation Function Layer:** This layer will apply element wise activation function to the output of convolution layer. Some common activation functions are RELU:  $\max(0, x)$ , Sigmoid:  $1/(1+e^{-x})$ , Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimension  $32 \times 32 \times 12$ .
- 4. Pool Layer:** This layer is periodically inserted in the CNNs and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with  $2 \times 2$  filters and stride 2, the resultant volume will be of dimension  $16 \times 16 \times 12$ .
- 5. Fully-Connected Layer:** This layer is regular neural network layer which takes input from the previous layer and computes the class scores and outputs the 1-D array of equal to the number of classes.

## Limitations

CNN do not encode the position and orientation of the object into their predictions. They completely lose all their internal data about the pose and the orientation of the object and they route all the information to the same neurons that may not be able to deal with this kind of information.

### 2.3.2. Recurrent Neural Networks

Recurrent Neural Networks are a type of neural network where the outputs from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.

RNN have a memory which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

Training through RNN is done as follows:

1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current  $h_t$  becomes  $h_{t-1}$  for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e. the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

## Limitations

1. Vanishing Gradients: This occurs when the gradients become very small and tend towards zero.
2. Exploding Gradients: This occurs when the gradients become too large due to back-propagation.

### 2.3.3. Long Short Term Memory

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells. These operations are used to allow the LSTM to keep or forget information.

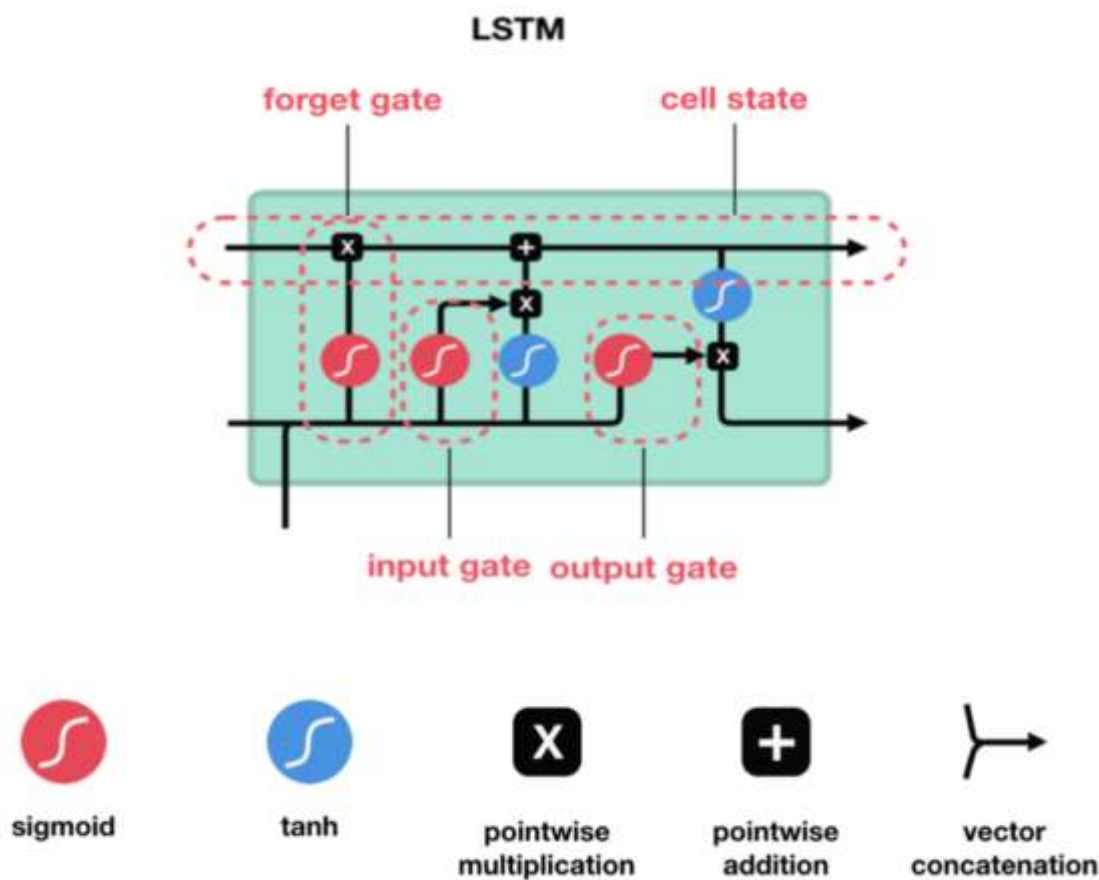



Fig 2.3.1 LSTM Cell and its operations

## Sigmoid Activation

Gates contains sigmoid activations. It squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappear or be “forgotten.” Any number multiplied by 1 is the same value therefore that value is “kept.” The network can learn which data is not important therefore can be forgotten or which data is important to keep.

We have three different gates that regulate information flow in an LSTM cell.  Forget gate, input gate, and output gate.

## Tanh Activation

The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.

Multiple transformations in LSTM may cause some vectors to explode making other values insignificant, Tanh functions ensures the values stay between -1 and 1.

## Forget Gate

This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

## Input Gate

To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

## Cell State

Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

## Output Gate

Last we have the output gate. The output gate decides what the next hidden state should be. The hidden state contains information on previous inputs; it is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

```
def LSTMCELL(prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)
    Ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(Ct)
    return ht, Ct

ct = [0, 0, 0]
ht = [0, 0, 0]
for input in inputs:
    ct, ht = LSTMCELL(ct, ht, input)
```

Pseudocode for Basic LSTM Cell

## 2.4. Why Python?

- It offers a simple and concise code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python's simplicity allows developers to write reliable systems.
- To reduce development time, programmers turn to a number of Python frameworks and libraries. A software library is pre-written code that developers use to solve common programming tasks.
- Python is supported by many platforms including Linux, Windows, and macOS. Python code can be used to create standalone executable programs for most common operating systems, which means that Python software can be easily distributed and used on those operating systems without a Python interpreter.

## 2.5. Machine Learning Libraries

We will make use of the following Python machine learning libraries in our project:

### 1. Tensorflow

It is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

### 2. Pandas

It is a Python library with many helpful utilities for loading and working with structured data. We will use pandas to load the dataset into a dataframe.

### 3. NumPy

It is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

### 4. SciPy

It is an open-source Python library which is used to solve scientific and mathematical problems. It is built on the NumPy extension and allows the user to manipulate and visualize data with a wide range of high-level commands.

## 5. Matplotlib

It is a data visualization and plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications.

## 2.5. Anaconda Environment

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

### 2.5.1. Why Anaconda?

The open-source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. Anaconda provides the tools needed to easily:

- Collect data from files and databases
- Manage environments with Conda
- Share, collaborate on, and reproduce projects

## 2.6. Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization and machine learning. It provides ease of performing and training neural network models.

## 2.7. Android Studio

Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. Languages supported include Java, XML, Kotlin, C++.

## 3. PROPOSED WORK

The work has been divided into 5 separate modules (detailed in the Level 0 DFD in Chapter 5, Section 5.1) namely Data Collection, the Activity Classification LSTM, the Activity Unit Classification LSTM, Moving Distance Estimator and the final Android Application.

### 3.1. Data Collection\*

We plan to collect accelerometer and gyroscope data in an uncontrolled environment using a smartphone sensor. We will stroll on foot to record the data for Walking, Running and Standing activities. Similarly, we will collect data for Left Turn, Right Turn, Short Step, Normal Step and Long Step. The data will be recorded with sessions each lasting 10 seconds.

We are planning on building a basic app for Android to collect the data and temporarily store it (in .csv format) before uploading it to the environment where pre-processing will be performed.

A list of activities will be provided in the app, and depending on which is selected, the data will be labelled as the same.

\* We have implemented this module and detailed it in Chapter 6, Section 6.1.

### 3.2. Activity Classification\*

Activities are divided into Walking, Running and Standing. These activities will be classified in real-time with the help of a trained LSTM model.

The creation of the LSTM will involve three phases, namely:



### **1. Data Pre-processing**

The labelled data collected in the previous module will first be pre-processed using Python libraries in order to make the data viable for training and testing

### **2. Model Construction**

The LSTM model will be built as per our requirements with the help of Tensorflow functions.

### **3. Training and Testing**

The pre-processed data from the first step will be input into the LSTM model from the second step in order to train the model. Testing is also performed to check the accuracy of its predictions.

\* We have implemented this module and detailed it in Chapter 6, Section 6.2.

## **3.3. Activity Unit Classification**

We plan on collecting data for Left Turn, Right Turn, Short Step, Normal Step and Long Step using the Data Collection application detailed in Section 3.1. Once the data is collected, it will be fed to the Activity Unit Classification LSTM in a way similar to Section 3.2 in order to classify the data and provide the classified data output as input to the Moving Distance Estimator in the following section.

## **3.4. Moving Distance Estimator**

Once the classified data from Activity and Activity Unit Classification are obtained, they are fed as input to the Moving Distance Estimator, which will have three memory banks for storing – Long Step Length, Normal Step Length and Short Step Length. It will also consist of a Location Buffer, which will store the most recent past location information.

If the output obtained from AU classification is a Left or Right Turn, it adds 90 degrees to the trajectory of the user. If the recognized AU is a Short Step, Normal Step or Long Step, it will compute the length of each step as follows:

- a. Some 'ground truth' values i.e. step lengths, will be assigned for each type of step. These will be stored initially as distributions in the memory bank.

- b. Next, the length distributions will be used by the moving distance estimator, which picks the most frequent value (i.e. mode) from the particular distribution as the step length according to the step type sent by the second LSTM.

To compute the updated current position of the target, the estimated step length will then added to the previous position of the trajectory. The estimator will maintain a record of the previous location information of the trajectory in the location buffer.

### 3.5. Android Application

For the final module, we will create an android application for the user to visualize his position inside a room, as well as his trajectory whilst moving around it. The app will display the floor layout with the position of the user shown as a blue dot. A mock-up of the final application is shown in the GUI mockups section in Chapter 5, Section 5.2.

## 4. REQUIREMENTS

### 4.1. Software Requirements

#### 1. Model

- Anaconda
- Jupyter Notebook
- Tensorflow
- Python
- Libraries
  - Pandas
  - NumPy
  - SciPy
  - Mathplotlib

#### 2. App Front-end

- Android Studio
- Java
- XML

## 4.2. Hardware Requirements

### 1. Smartphone (Data Collection and Testing)

- Accelerometer Sensor
- Gyroscope Sensor

### 2. Macbook Pro (Model Training)

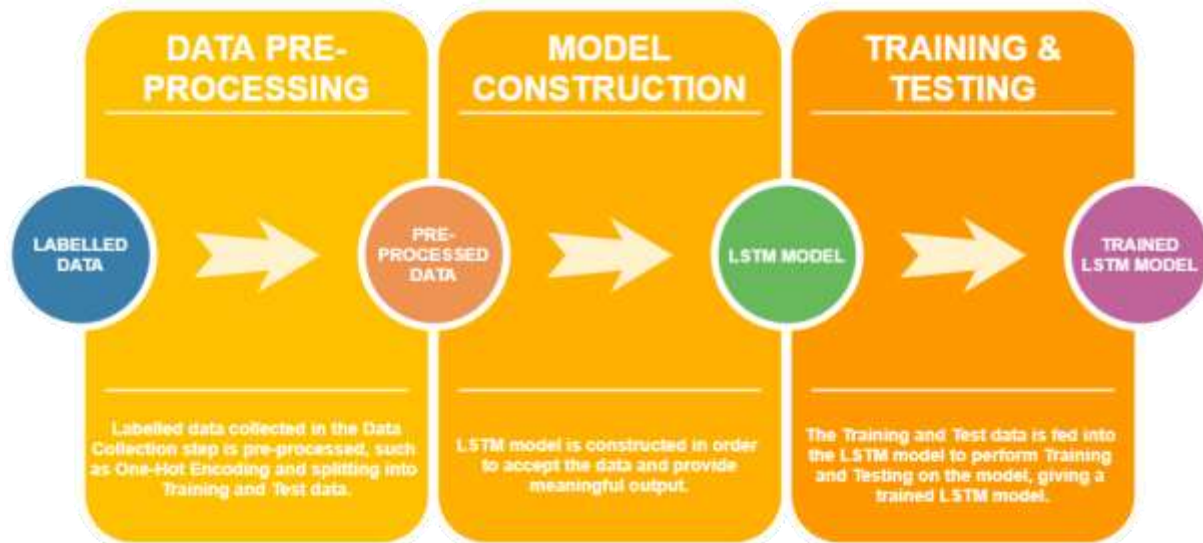
- CPU: 2.5 GHz Quad-Core Intel Core i7
- GPU: Intel Iris Pro

## 5. DESIGN

### 5.1. Design Flow Diagrams



Level 0: Overall Design Flow Diagram



Level 1: LSTM Design Flow Diagram

## 5.2. GUI Mockups



(a) User selects starting position



(b) User walks and system updates position in real time



(c) User being positioned



(d) User selecting a point on the map



(e) User travelling towards the point on the map



(f) User arriving at point on the map

## 6. IMPLEMENTATION

### 6.1. Data Collection

We have created a basic Android app to easily collect and export sensor data in the .csv format. The working is as follows:

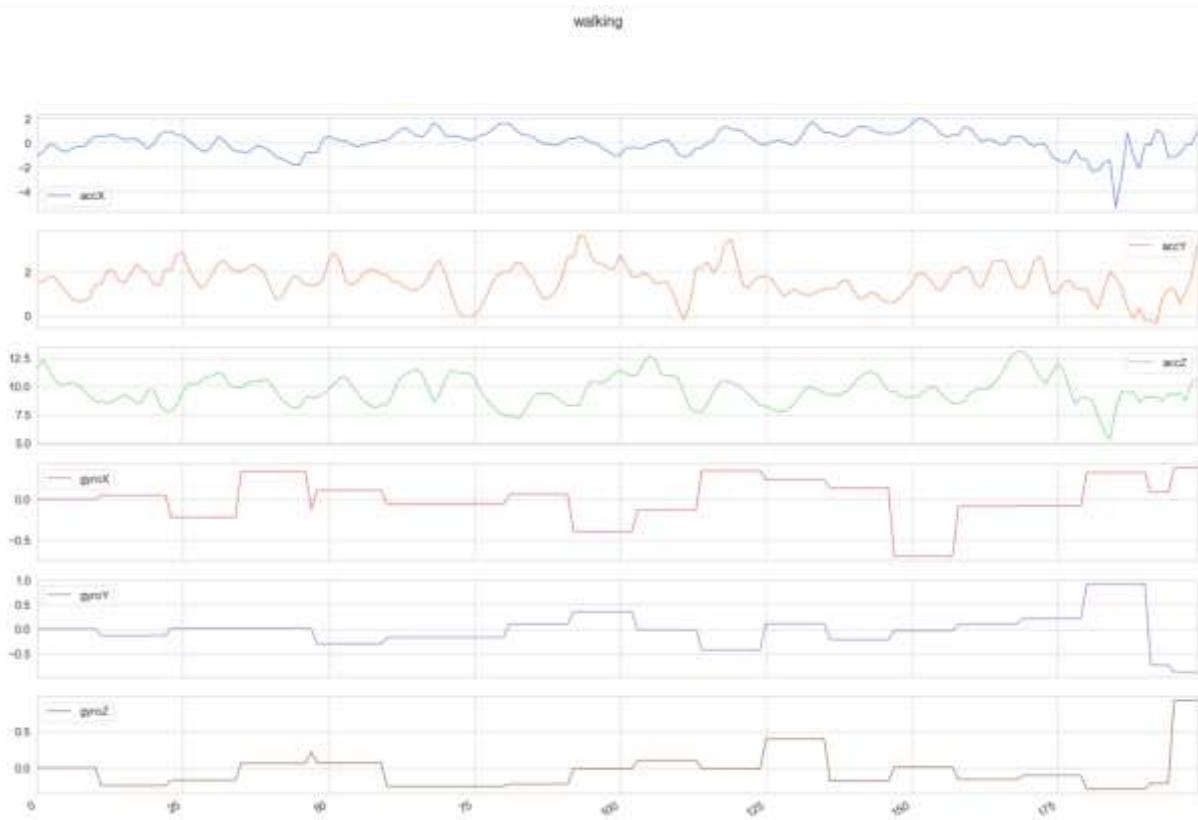
- Buttons with labels are provided for activities such as Walk, Run and Standing.
- Upon tapping a button, the app begins to record accelerometer and gyroscope sensor data, and appends to that data the corresponding label of the button tapped. Eg. If 'Walk' is tapped and we begin walking, each record in the table gets the data 'walking' under its Activity column.
- The 'Stop' button needs to be tapped for the app to stop recording the data.
- If another activity needs to be recorded, we tap the corresponding button and the data continues to get recorded in the same .csv file as the previous data.
- If data collection is over, we tap the 'Export Data' button to export the recorded data as a .csv file to read and perform further tasks on it.



2:29 walking.csv 20261 Rows

No.	accX	accY	accZ	gyroX	gyroY	gyroZ	timestamp	Activity
150	-1.4589233	2.832306	9.6306305	-0.0042266846	0.19013977	-0.11529541	15:17:18.640	walking
151	-0.9178467	2.6264038	11.028839	-0.0042266846	0.19013977	-0.11529541	15:17:18.642	walking
152	-0.7586212	2.598877	11.182068	-0.0042266846	0.19013977	-0.11529541	15:17:18.644	walking
153	-1.0016327	3.1064453	9.149399	-0.0042266846	0.19013977	-0.11529541	15:17:18.645	walking
154	-1.4421692	3.3757935	7.8637085	-0.0042266846	0.19013977	-0.11529541	15:17:18.647	walking
155	-1.4325867	3.0944824	9.175735	-0.0042266846	0.19013977	-0.11529541	15:17:18.648	walking

★ Copy Selected Cell      ★ Line Number      ★ Scroll To Top



We managed to procure 20,261 records for Walking in 6 minutes.

## 6.2. Activity Classification

We have implemented the Activity Classification LSTM with the help of Python via Jupyter Notebook and Tensorflow. We have imported packages such as pandas, numpy, scipy, sklearn, etc in order to make pre-processing and visualization easier.

The data being used is from Wireless Sensor Data Mining (WISDM) Lab. The dataset contains 1,098,207 rows and 6 columns. There are no missing values. The data is labelled with 6 activities, namely: Walking, Jogging, Upstairs, Downstairs, Sitting, Standing; and consists of the accelerometer data collected of 36 participants doing the same. (We have chosen not to include data collected by ourselves for the first prototype, but we plan on doing so in the future as it will provide better accuracy for our particular use cases.)

The following is an example of some of the data rows and their visualization:

	<b>user</b>	<b>activity</b>	<b>timestamp</b>	<b>x-axis</b>	<b>y-axis</b>	<b>z-axis</b>
<b>0</b>	33	Jogging	49105962326000	-0.694638	12.680544	0.503953
<b>1</b>	33	Jogging	49106062271000	5.012288	11.264028	0.953424
<b>2</b>	33	Jogging	49106112167000	4.903325	10.882658	-0.081722
<b>3</b>	33	Jogging	49106222305000	-0.612916	18.496431	3.023717
<b>4</b>	33	Jogging	49106332290000	-1.184970	12.108489	7.205164

Table 6.2.1 Data Rows



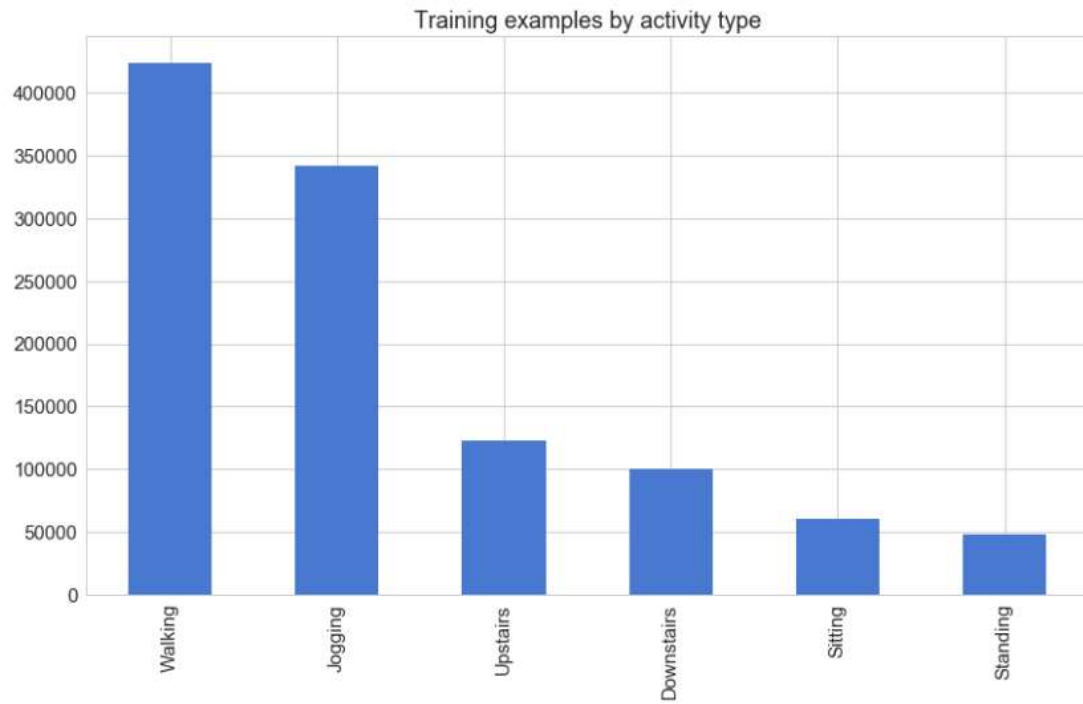


Fig 6.2.2 Training Examples by Activity Type



Fig 6.2.3(a) Accelerometer Data for Walking

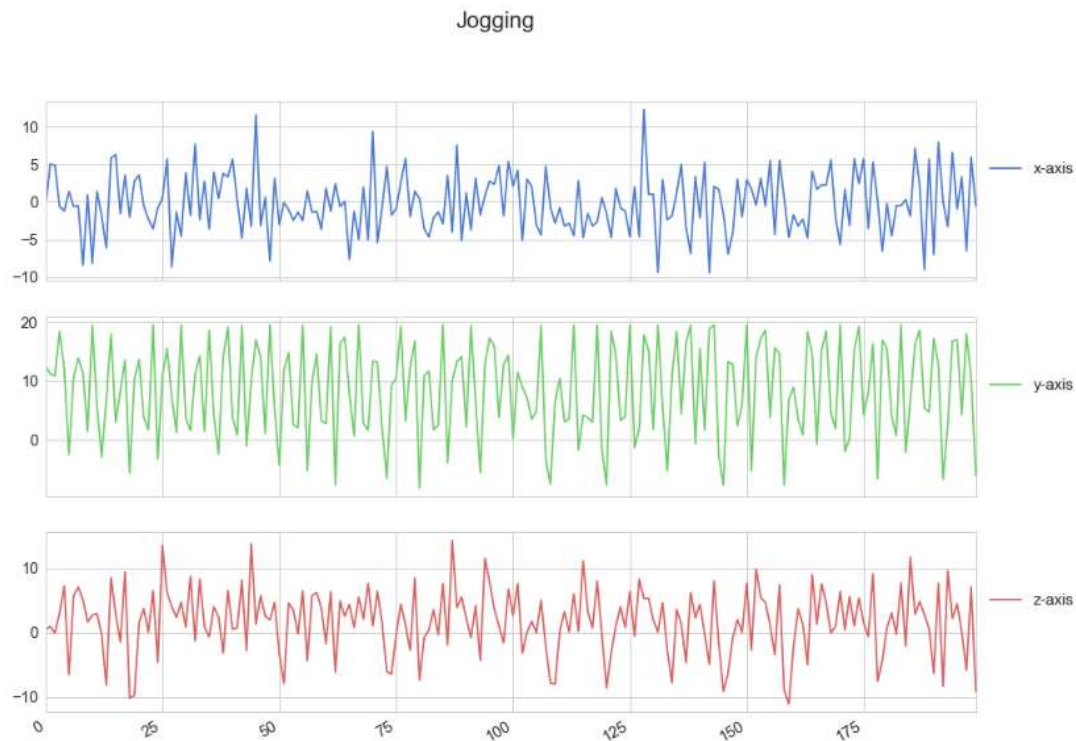


Fig 6.2.3(b) Accelerometer Data for Jogging

## 2. Data Pre-processing

The above data has to be pre-processed for training since the LSTM model expects fixed-length sequences as training data. Each sequence of data contains 200 training examples. The data has been transformed to include only the accelerometer x-axis, y-axis and z-axis data i.e. 3 features.

One-Hot Encoding i.e. converting the activities which are categorical data into binary vector arrays, is also done.

The data is then split into training (80%) and test (20%) data.

## 3. Model Construction

The LSTM model contains 2 LSTM layers stacked on top of each other with 64 hidden units each. It consists of 6 class labels (Walking, Jogging, Sitting, Standing, Upstairs, Downstairs), with the inputs being the accelerometer x-axis, y-axis and z-axis. We use the LSTM model to predict the output, and use the softmax function over it. We then find the loss and backpropagate it in order to optimize the model.

## 4. Model Training & Testing

For training and testing, we take 40 epochs with a batch size of 1024. We feed arrays of data obtained in previous steps into the training and testing inputs for the model. The output is given as the train and test loss and accuracy. This gives us a working model that we can use for further steps.

### Output:

```
epoch: 1 test accuracy: 0.7736998796463013 loss: 1.2773
654460906982
epoch: 10 test accuracy: 0.9388942122459412 loss: 0.561
2533092498779
epoch: 20 test accuracy: 0.9574717283248901 loss: 0.391
6512429714203
epoch: 30 test accuracy: 0.9693103432655334 loss: 0.293
5260236263275
epoch: 40 test accuracy: 0.9747744202613831 loss: 0.250
2188980579376
```

Using a Macbook Pro, it took us approximately 30 minutes per 10 epochs for training.

The model is evaluated by plotting training and test accuracy and loss over the epochs for comparison. The model is then exported and saved as a pb file for further use in a basic Android app to test the model.

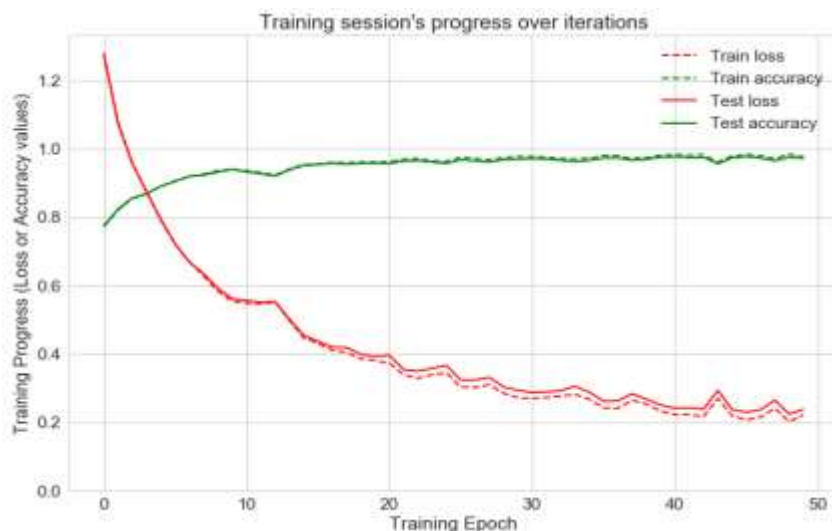
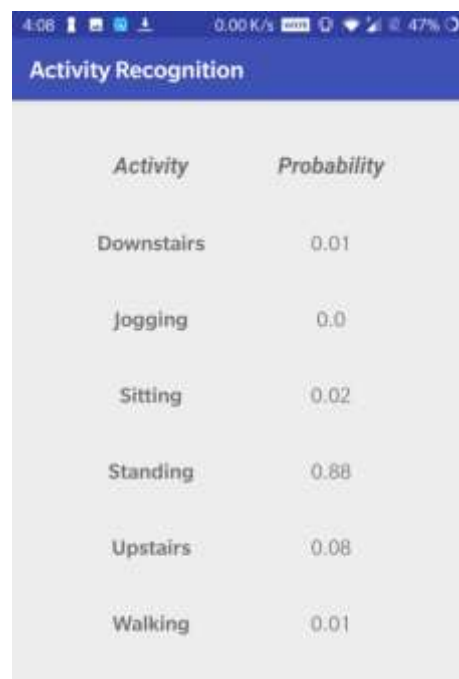


Fig 6.2.4 Training Session Progress over Iterations

## 5. Android App

We also created a basic android app that interfaces with the LSTM model to check if the predictions of the model are accurate. The app consists of a table with the class labels with the corresponding probability that it is occurring. As the user travels across the room, the probabilities will and output which activity is currently occurring.



The screenshot shows an Android application interface with a blue header bar labeled "Activity Recognition". Below the header is a table with two columns: "Activity" and "Probability". The table lists six activities with their respective probabilities: Downstairs (0.01), Jogging (0.0), Sitting (0.02), Standing (0.88), Upstairs (0.08), and Walking (0.01). The status bar at the top indicates the time is 4:08, the data speed is 0.00 K/s, and the battery level is 47%.

<i>Activity</i>	<i>Probability</i>
Downstairs	0.01
Jogging	0.0
Sitting	0.02
Standing	0.88
Upstairs	0.08
Walking	0.01

## 3.2.1 Algorithm

### 1. LSTM Model

*Input features: 3 (accelerometer x, y, z)*

*Hidden units: 64*

*Output classes: 6 (Walking, Jogging, Upstairs, Downstairs, Sitting, Standing)*

*def create\_LSTM\_model(inputs):*

*initialize weights (from input to hidden and hidden to output)*

*initialize biases (of hidden and output)*

*reshape the input to a 3d matrix as LSTM requires 3d input*

*updates hidden unit's weights and biases using 'relu' activation function that uses the combination of sigmoid and tanh functions to accept only the relevant values*

*split the data into 200 time steps*

*stack 2 LSTM cells together with forget bias = 1 and get the combined output*

*get the output for the last time step*

*multiply output for the last time step with the final weight and add the final bias, and return the result*

### 2. Training & Testing

*epochs: 40*

*batch\_size: 1024*

*train\_count: 43920*

*test\_count: 10981*

*def train\_test:*

*start session*

*for i in range(1, epochs + 1)*

*for start, end in range(0, train\_count, BATCH\_SIZE),*

*range(BATCH\_SIZE, train\_count + 1, BATCH\_SIZE)):*

*feed input training data and get training output*

*estimate the accuracy and loss of training and test outputs*

*print the accuracy and loss of training and test outputs for debugging*

*end session*

## CONCLUSIONS

Through this report, we have demonstrated the work that we have done for this project for the duration of this semester. We began by introducing indoor positioning and why it is important, and also detailed our particular approach. We then surveyed and summarized three different approaches to indoor positioning and elicited their drawbacks. Next, we described the components that we will be using, along with the work to be done in the form of modules. We have included high-level Design Flow Diagrams and examples of how the final system will work. Finally, we detailed the work completed in the data collection and activity classification modules, and added appropriate images for the same. Based on our objectives and the proposed work detailed, we will continue with the rest of the implementation in the next semester.

# BIBLIOGRAPHY

1. G. Hussain, M.S. Jabbar, J.D. Cho, and S. Bae, 2019, "Indoor Positioning System: A New Approach Based on LSTM and Two Stage Activity Classification", electronics, March 2019.
2. S. Mazuelas, A. Bahillo, R. Lorenzo, et al, 2009, "Robust Indoor Positioning Provided by Real-Time RSSI Values in Unmodified WLAN Networks", IEEE Journal of Selected Topics in Signal Processing, November 2009.
3. N. Lee, S. Ahn, and D. Han, 2018, "AMID: Accurate Magnetic Indoor Localization Using Deep Learning", sensors, May 2018.
4. P. Kriz, F. Maly, and K. Tomáš, 2016, "Improving Indoor Localization Using Bluetooth Low Energy Beacons", Mobile Information Systems, April 2016.