INDOOR POSITIONING SYSTEM USING DEEP LEARNING

Clyde Noronha (161105036)
Mustafa Zaki (161105062)
Shane Gracias (161105017)
Ruman Mulla (161105029)

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

BACHELOR OF ENGINEERING

Branch: COMPUTER ENGINEERING

OF GOA UNIVERSITY

July 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	(161105036)
Mustafa Zaki	(161105062)
Shane Gracias	(161105017)
Ruman Mulla	(161105029)

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 **Dr. J.A. Laxminarayana**

Faculty Guide **Head of Computer Engg Dept**

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

…………………….. …………………………..

(Internal Examiner) (External Examiner)

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.

Contents

List of Figures

List of Tables

List of Abbreviations

1. GPS: Global Positioning System
2. IPS: Indoor Positioning System
3. AU: Activity Unit
4. RSSI: Received Signal Strength Indicator
5. MEMS: Micro-Electro-Mechanical System
6. LSTM: Long Short Term Memory
7. SS: Short Step
8. NS: Normal Step
9. LS: Long Step
10. AP: Access Point
11. RSS: Received Signal Strength
12. AMID: Accurate Magnetic Indoor Localization using Deep Learning
13. CNN: Convolutional Neural Network
14. RNN: Recurrent Neural Network
15. GUI: Graphical User Interface
16. CLI: Command Line Interface
17. BFS: Breadth First Search

Abstract

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.

Chapter 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]

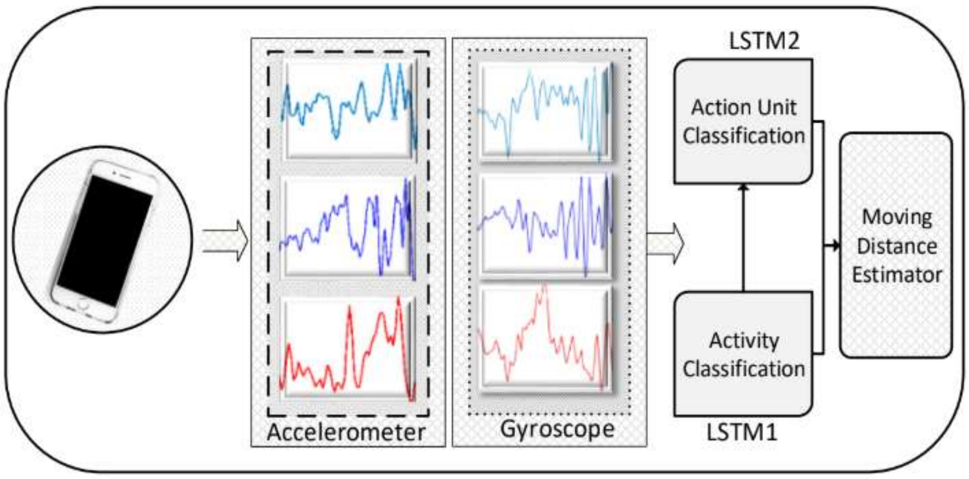


Fig 1.1.1 Block Diagram of Stages.

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.

1. 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: $P\kappa = P\kappa{-}1 + AU\kappa$, --- (1.1.1)

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

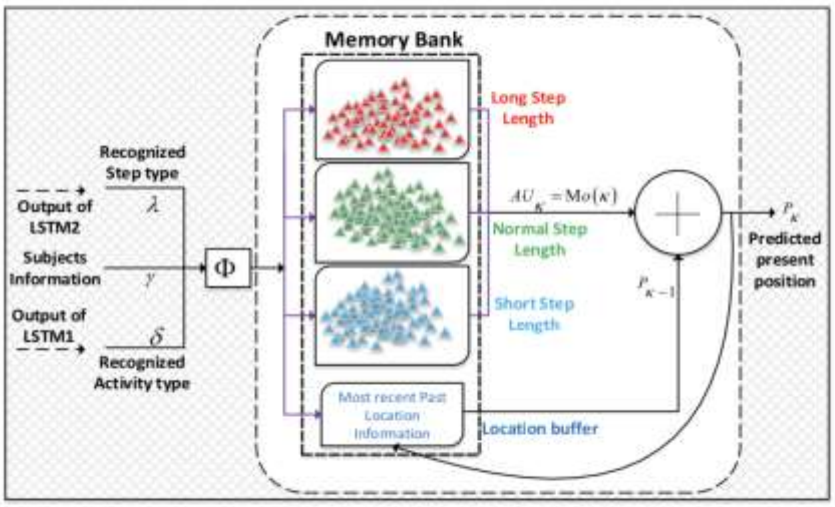


Fig 1.1.2 Moving Distance Estimator Block Diagram.

Chapter 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.

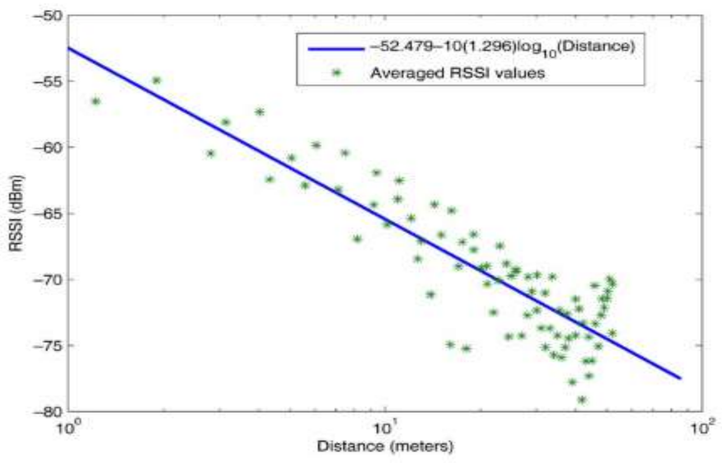


Fig 2.1.1 Relationship between distance and RSSI values in a corridor.

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:

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

3. Trilateration

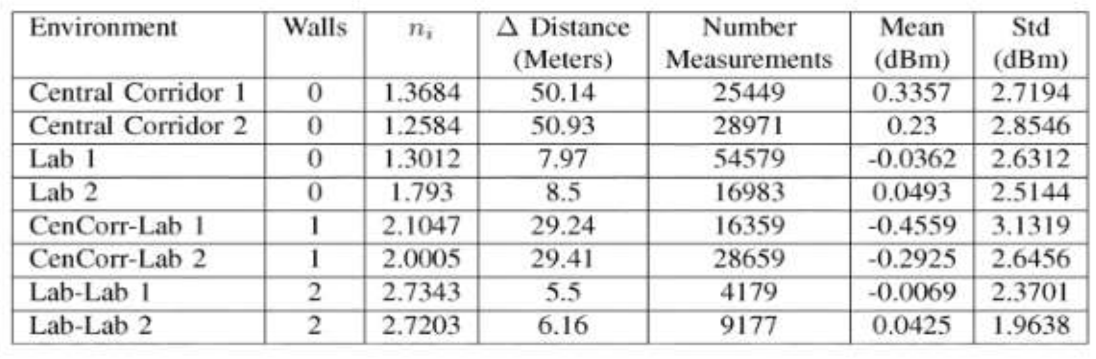
Trilateration is the determination of the position of a point by the measurement of distances of that point from other points. Once the distance between the device and the APs are found in the second step, we can perform trilateration to find the real-time position of the device.

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	Δ 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.

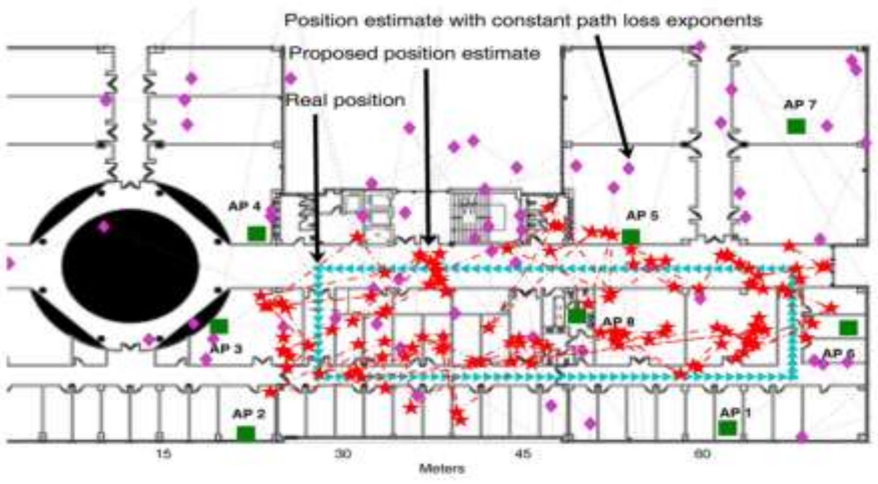

Position estimate with constant path loss exponents
Proposed position estimate
Real position
AP 1
AP 2
AP 3
AP 4
AP 5
AP 6
AP 7
AP 8
15
30
45
60
Meters

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.

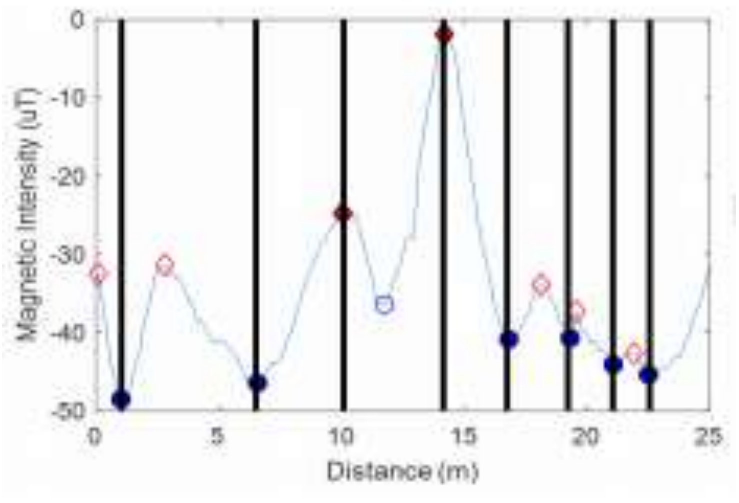


Fig 2.2.1 Magnetic Landmark Peaks

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.

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:

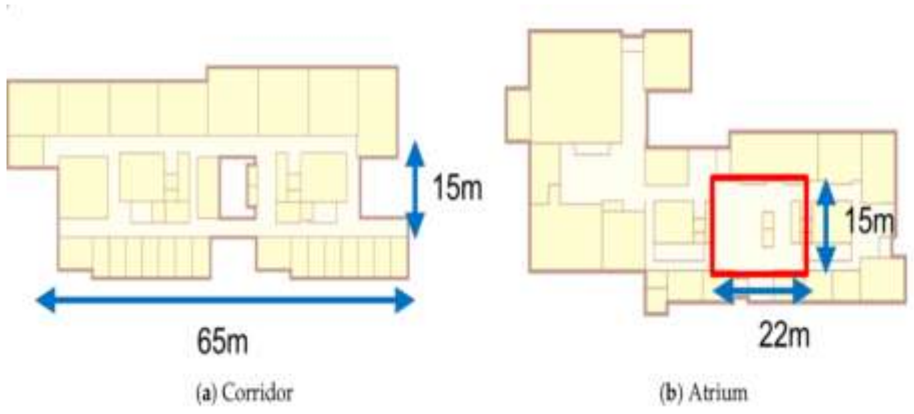
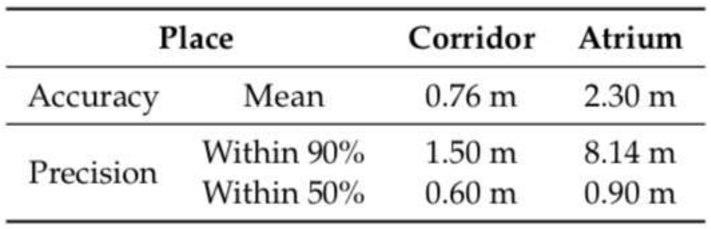


Fig 2.2.2 Actual and predicted trajectories of participants

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:

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

-- (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.

$$X_{\mathrm{Tx}} = \{x_{1\mathrm{Tx}}, x_{2\mathrm{Tx}}, \ldots, x_{M\mathrm{Tx}}\}.$$

-- (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.

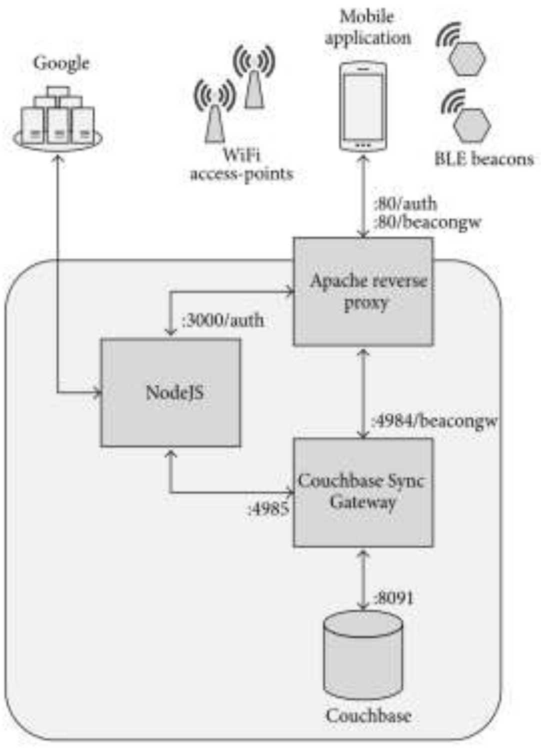


Fig 2.3.1 System Architecture

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.

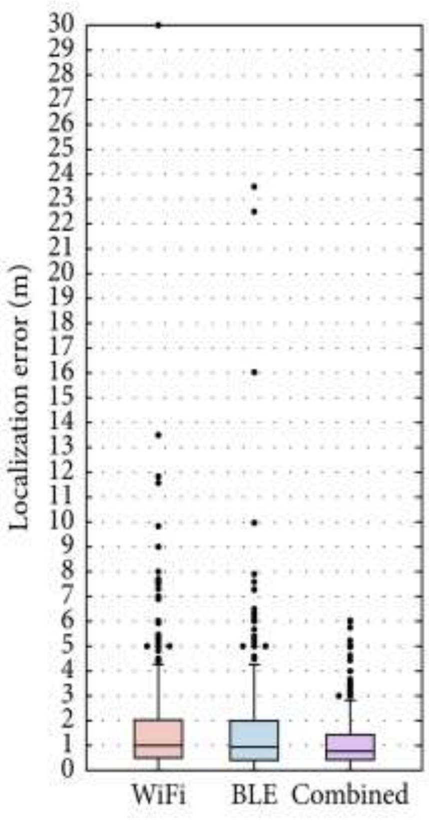


Fig 2.3.2 Results of WiFi, BLE and Combined

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.

2.4 Neural Networks

2.4.1. Convolutional Neural Networks [5]

In neural networks, Convolutional Neural Networks (ConvNets or CNNs) are one of the main categories to do images recognition, images classifications. Object 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).

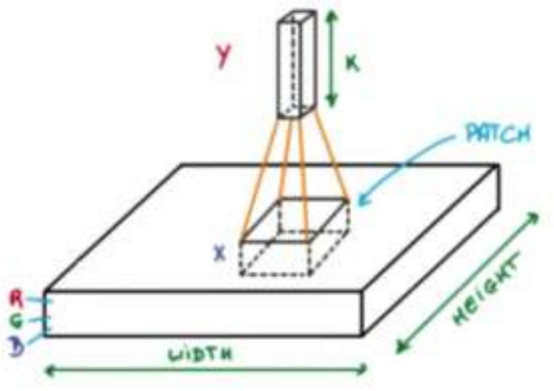


Fig 2.4.1 CNN

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. his 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 of image of dimension 32 x 32 x 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 x 32 x 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 x 32 x 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 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

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.

2.4.2. Recurrent Neural Networks [6]

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 ht becomes ht-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.

2.4.3. Long Short Term Memory [7]

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.

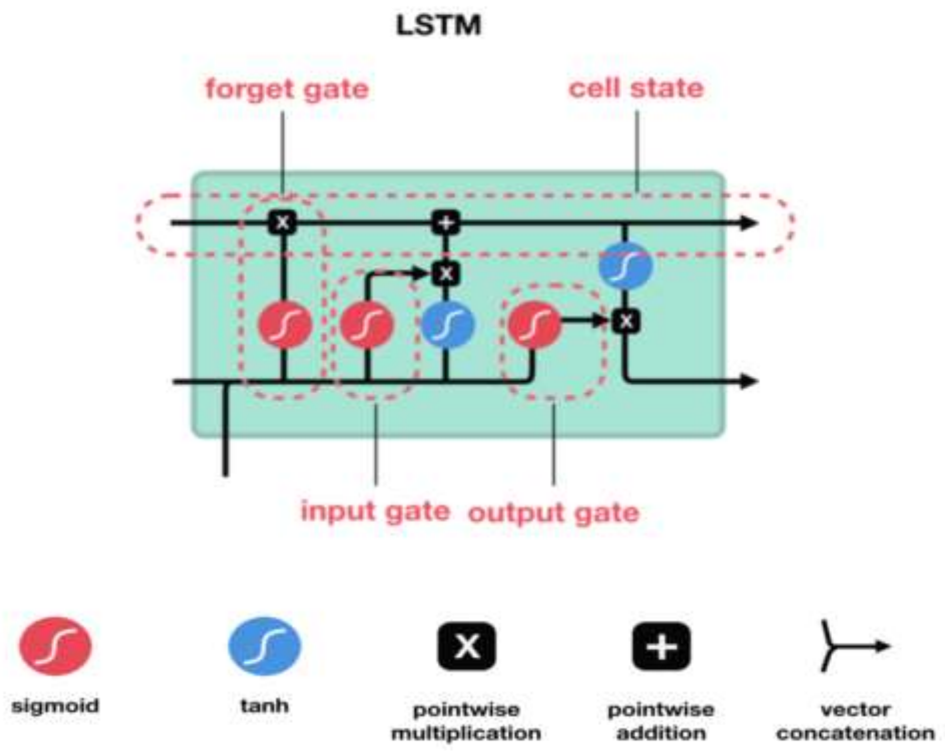


Fig 2.4.2 LSTM Cell and its operations

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 disappears 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

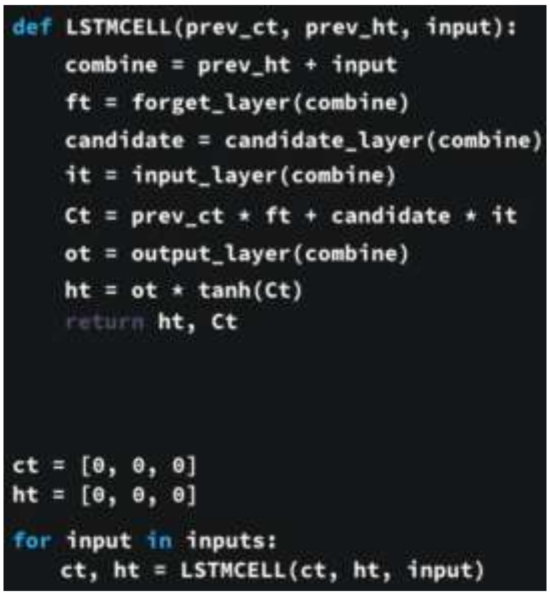
To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state 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

2.5. Why Python? [8]

- 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.

2.6. Machine Learning Libraries [8]

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.

2.7. Anaconda Environment [9]

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).

2.7.1. 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

2.8. Jupyter Notebook [10]

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.

2.9. Android Studio [11]

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++.

Chapter 3

Requirements

3.1 Software Requirements

1. Model

- Anaconda
- Jupyter Notebook
- Google Colab
- Tensorflow
- Python
- Libraries
    - Pandas
    - NumPy
    - SciPy
    - Matplotlib

2. App Front-end

- Android Studio
- Libraries
    - Android.graphics
    - Android.widget

3.2 Hardware Requirements

1. Data Collection and Testing

- Accelerometer Sensor (In-built)
- Gyroscope Sensor (In-built)

2. Model Training

- CPU: Intel Core i5 6th Gen or higher
- RAM: 8GB minimum. 16GB or higher is recommended
- GPU: NVIDIA GeForce GTX 960 or higher
- OS: Ubuntu, Windows 10 or MacOS High Sierra

Chapter 4

Design

4.1 Conceptual Design

4.1.1 Data Collection

We propose 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 Walking, Running 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 'Walking' 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.

4.1.2 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.

4.1.3 Activity Unit Classification

Activity Units are divided into Short Step (SS), Normal Step (NS) and Long Step (LS) in order to improve the accuracy of the final application. These activity units 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.

4.1.4 Moving Distance Estimator

The Moving Distance Estimator works on the principle that “a person will walk along a path with a consistent step type, unless an obstacle occurs.”

- Output of Activity Unit Classification LSTM (SS, NS, LS) will be fed into the Moving Distance Estimator.
- Estimator has a buffer of a particular size, initially empty. It stores the step type values.
- If the buffer is full, the oldest value is discarded and the new value enters.
- The mode is calculated continuously, and the output is considered as the step type of that particular moment in time.
- The step type is converted to an actual step length value according to the user’s height.
- The step length is outputted from the Estimator and given as an input for the movement of the Blue Dot in the final Android app.

4.1.5. Android Application

An android application using Android Studio that consists of a basic floor plan of a room whose dimensions are mapped to the real world dimensions of the room.

The user has to select his/her initial position on the floor plan which is indicated by a Blue Dot, and the user’s position and direction is traced as the user starts moving based on the outputs of the first and second LSTMs and the Moving Distance Estimator.

Back-end

- The two LSTMs, after being created, should be exported as protocol buffer (.pb) files and used as assets for the android application.
- The application reads the probability outputs of these LSTMs and classifies the activity and activity unit accordingly.
- This is then sent to another program in the application known as the Moving Distance Estimator, which takes as its input the activity and activity unit that has been classified earlier.
- It outputs a specific step length X (in px) and step length Y (in px), and this is used in order to update the Blue Dot on the user’s screen.

Positioning

The user’s heading direction is to be computed using the magnetic sensor of the smartphone. By computing a device's orientation, you can monitor the position of the device relative to the earth's frame of reference (specifically, the magnetic north pole). We can calculate the orientation angles by using a device's geomagnetic field sensor in combination with the device's accelerometer.

Front-end

- The frontend of the application consists of a basic floor plan image to which the room’s dimensions will be mapped.
- It also consists of a Blue Dot, which can be freely placed anywhere on the map in order to designate the starting position.
- The Blue Dot is unable to cross the boundaries of the map.
- The Blue Dot traces the user’s path as he moves inside the room.

- For Navigation, the user gets to search for their destination, and a line is drawn from the user's position to the destination.
- This is done using a node map, where the map is divided into several nodes.

Navigation

- The map consists of nodes and edges where the nodes refer to any location on the map and the edges refer to the path between them.
- Starting from the source, Breadth First Search is used to traverse the nodes until the destination is reached.

Shortest Path Calculation

- The idea is to use a modified version of Breadth First Search in which we keep storing the predecessor of a given node while doing the search till we reach the destination.
- And the shortest path is the shortest no. of hops required to reach the destination.

Graphical User Interface (GUI)

The GUI consists of the following components:

1. Search Bar – Contains a drop-down list of all the locations on the map.
2. "Directions" button – Upon tapping, draws a line from the user's position to the destination searched for.
3. Blue Dot – Denotes the user's position. User can tap to set their starting position.
4. Map – The map of the GEC Main Building's ground floor.

4.2 Detailed Design

4.2.1 Design Flow Diagrams

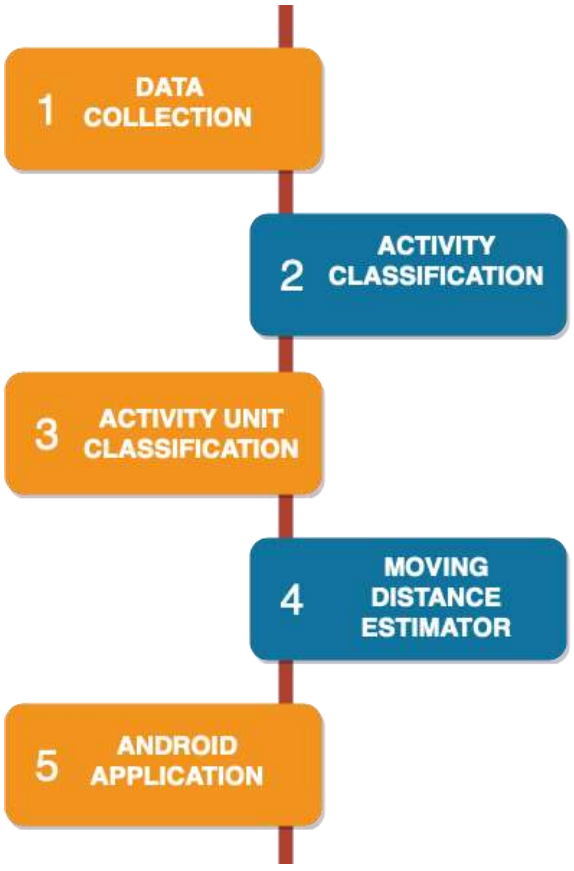

1 DATA COLLECTION
2 ACTIVITY CLASSIFICATION
3 ACTIVITY UNIT CLASSIFICATION
4 MOVING DISTANCE ESTIMATOR
5 ANDROID APPLICATION

Fig 4.2.1 Level 0: Overall Design Flow Diagram

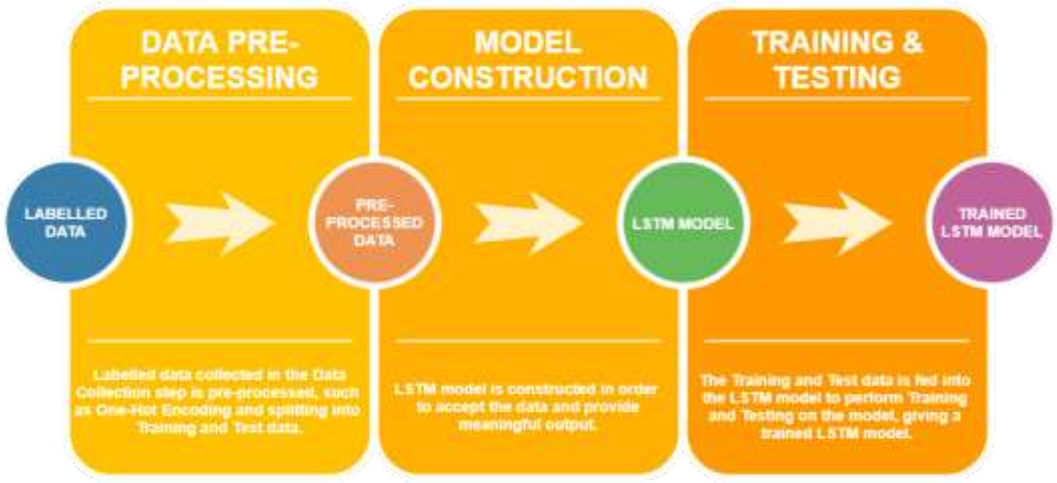

DATA PRE-PROCESSING
MODEL CONSTRUCTION
TRAINING & TESTING
LABELLED DATA
PRE-PROCESSED DATA
LSTM MODEL
TRAINED LSTM MODEL
Labelled data collected in the Data Collection step is pre-processed, such as One-Hot Encoding and splitting into Training and Test data.
LSTM model is constructed in order to accept the data and provide meaningful output.
The Training and Test data is fed into the LSTM model to perform Training and Testing on the model, giving a trained LSTM model.

Fig 4.2.2 Level 1: LSTM Design Flow Diagram

4.2.2 Algorithms

Activity Classification

1. LSTM Model

```
Input features: 3 (accelerometer x, y, z)
Hidden units: 64
Output classes: 3 (Walking, Running, 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: 8948
        test_count: 2238
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
```

Activity Unit Classification

1. LSTM Model

```
Input features: 3 (accelerometer x, y, z)
Hidden units: 64
Output classes: 3 (SS, NS, LS)
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: 140
train_count: 7748
test_count: 1937
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
```

Moving Distance Estimator

```
Input: Activity Unit Classification (SS, NS, LS)
Output: Step Length (in cm)
def estimator(input):
    initialize buffer size to n
    buffer[n] = { }
    add value (SS, NS, LS) to buffer up to n
    if size(buffer) = n
        discard oldest value and insert newest value
    step_type = mode(buffer)
    calculate step_type continuously
```

```
switch(step_type):
    case SS:
        step_length = 42
        break
    case NS:
        step_length = 75
        break
    case LS:
        step_length = 92
        break
return step_length
```

Positioning

Let (x , y) be the current position of the device.

Θ (Azimuth) is the angle between the device's current compass direction and magnetic north.

Input: Step Length (in cm), Θ (Azimuth) angle between device’s current compass direction and magnetic north

Output: (x, y) position of Blue Dot

```
def blueDot(step_length):
    x  = x + step_length * cos(θ);
y  = y + step_length * sin(θ);
return (x, y)
```

Navigation

1. Breadth First Search (BFS)

Input: Source, Destination, Predecessors, Distances

Output: True, False

```
def BFS(inputs):
    Initialize queue
    Set all nodes to unvisited
    Initialize all distances to a max value and initially there are no predecessors
    Set the source as visited, update the distance for the source to 0 and add it to the
    queue
    While queue is not empty repeat:
        Remove the node from the queue
        For each neighbour:
            Add it to the queue if it is not visited
            Add 1 to the distance for that node
            Make that node a predecessor
            If any of the neighbours is the destination
                return True
            Else return false
    return false
```

2. Shortest Path

```
Input: Source, Destination
Output: Shortest Path
def getShortestPath(inputs)
    Initialize predecessors and distances for each node
    If BFS(source, destination, predecessors, distances) = false
        return null
    Initialize empty path
    Starting from destination, add all the predecessors of the nodes till we reach the
    source to the path
    return path
```

Chapter 5

Implementation

5.1 Jupyter Notebook

5.1.1 Setting up Jupyter Notebook & Tensorflow [13]

The following steps are involved in setting up Jupyter Notebook and the Tensorflow environment:

1. Download and install Anaconda.
2. Open Anaconda Navigator and Anaconda CLI.
3. Create the Tensorflow environment by typing the following command into the CLI.

```
conda create --name tensorflow python=3.7
```

4. Install all the libraries and packages required using the following commands:

```
   conda install -y scipy
pip install --exists-action i --upgrade sklearn
pip install --exists-action i --upgrade pandas
pip install --exists-action i --upgrade matplotlib
```

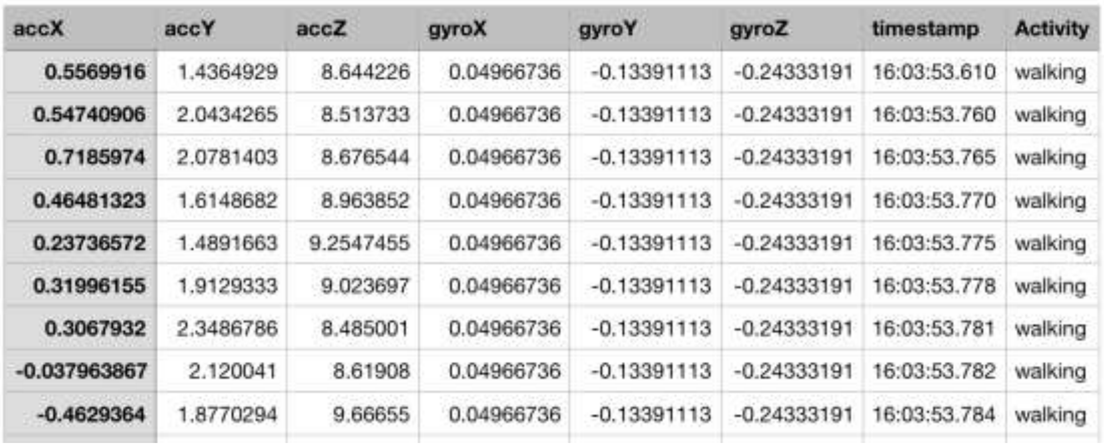
5. In Anaconda Navigator, select the Tensorflow environment and install Jupyter Notebook.

5.1.2 Collecting Data

The data being used is the data that is collected with the help of the Data Collection application, which will be elaborated on later.

The dataset contains 223,816 rows and 6 columns, namely: Accelerometer X, Y and Z and Gyroscope X, Y and Z. There are no missing values. The data is labelled with 3 activities, namely: Walking, Standing and Running.

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



accX	accY	accZ	gyroX	gyroY	gyroZ	timestamp	Activity
0.5569916	1.4364929	8.644226	0.04966736	-0.13391113	-0.24333191	16:03:53.610	walking
0.54740906	2.0434265	8.513733	0.04966736	-0.13391113	-0.24333191	16:03:53.760	walking
0.7185974	2.0781403	8.676544	0.04966736	-0.13391113	-0.24333191	16:03:53.765	walking
0.46481323	1.6148682	8.963852	0.04966736	-0.13391113	-0.24333191	16:03:53.770	walking
0.23736572	1.4891663	9.2547455	0.04966736	-0.13391113	-0.24333191	16:03:53.775	walking
0.31996155	1.9129333	9.023697	0.04966736	-0.13391113	-0.24333191	16:03:53.778	walking
0.3067932	2.3486786	8.485001	0.04966736	-0.13391113	-0.24333191	16:03:53.781	walking
-0.037963867	2.120041	8.61908	0.04966736	-0.13391113	-0.24333191	16:03:53.782	walking
-0.4629364	1.8770294	9.66655	0.04966736	-0.13391113	-0.24333191	16:03:53.784	walking

Table 5.1.1 Data Rows for Walking, Running, Standing

```
walking      90584
standing     75329
running      57903
```

Data Distribution

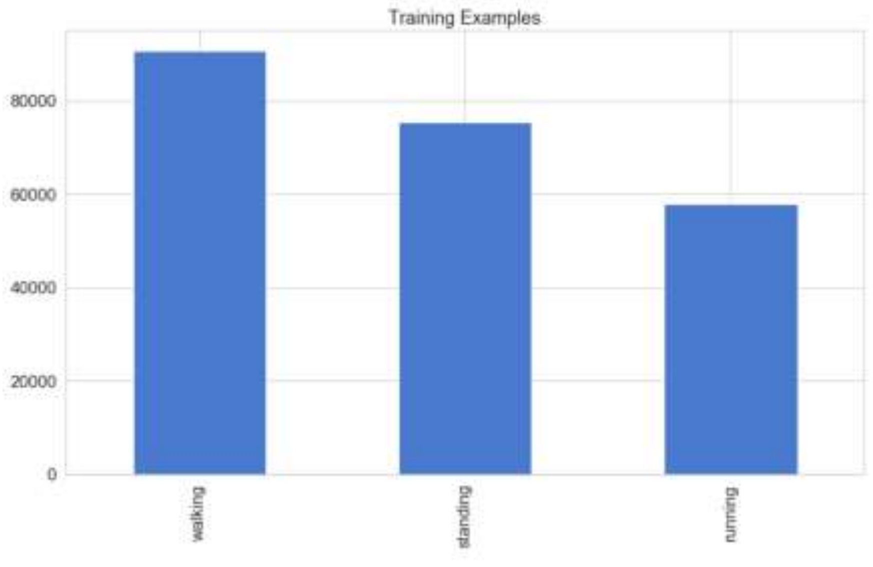


Fig 5.1.1 Training Examples by Activity Type

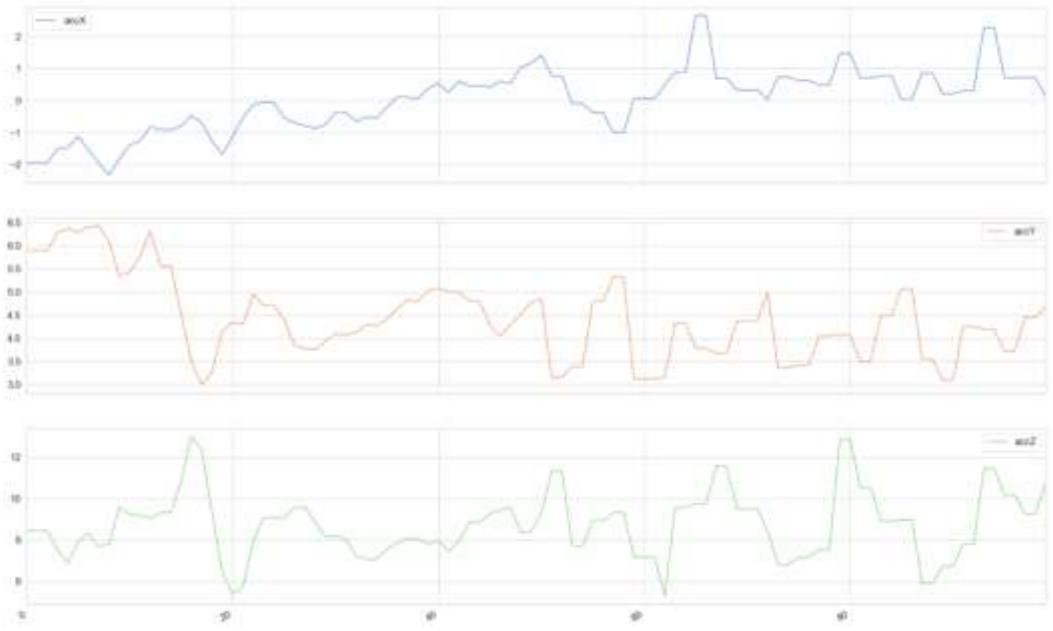


Fig 5.1.2(a) Accelerometer Data for Walking

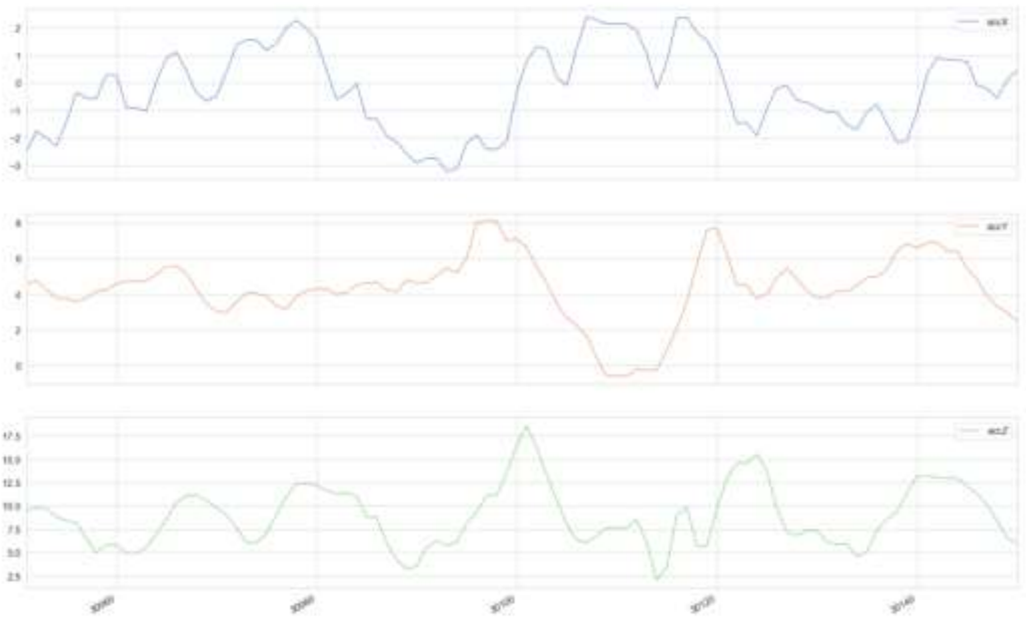
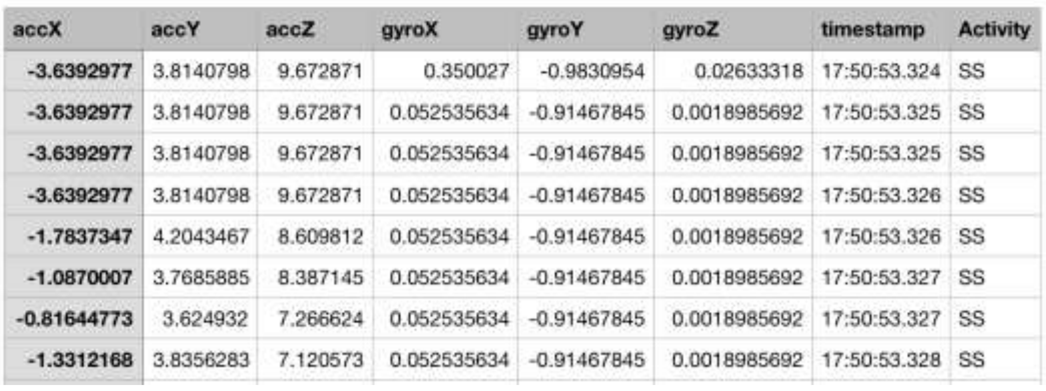


Fig 5.1.2(b) Accelerometer Data for Standing

The dataset contains 193,796 rows and 6 columns, namely: Accelerometer X, Y and Z and Gyroscope X, Y and Z. There are no missing values. The data is labelled with 3 activity units, namely: Short Step (SS), Normal Step (NS) and Long Step (LS).
The following is an example of some of the data rows and their visualization:



accX	accY	accZ	gyroX	gyroY	gyroZ	timestamp	Activity
-3.6392977	3.8140798	9.672871	0.350027	-0.9830954	0.02633318	17:50:53.324	SS
-3.6392977	3.8140798	9.672871	0.052535634	-0.91467845	0.0018985692	17:50:53.325	SS
-3.6392977	3.8140798	9.672871	0.052535634	-0.91467845	0.0018985692	17:50:53.325	SS
-3.6392977	3.8140798	9.672871	0.052535634	-0.91467845	0.0018985692	17:50:53.326	SS
-1.7837347	4.2043467	8.609812	0.052535634	-0.91467845	0.0018985692	17:50:53.326	SS
-1.0870007	3.7685885	8.387145	0.052535634	-0.91467845	0.0018985692	17:50:53.327	SS
-0.81644773	3.624932	7.266624	0.052535634	-0.91467845	0.0018985692	17:50:53.327	SS
-1.3312168	3.8356283	7.120573	0.052535634	-0.91467845	0.0018985692	17:50:53.328	SS

Table 5.1.2 Data Rows for SS, NS, LS

```
LS    64879
SS    64711
NS    64206
```

Data Distribution

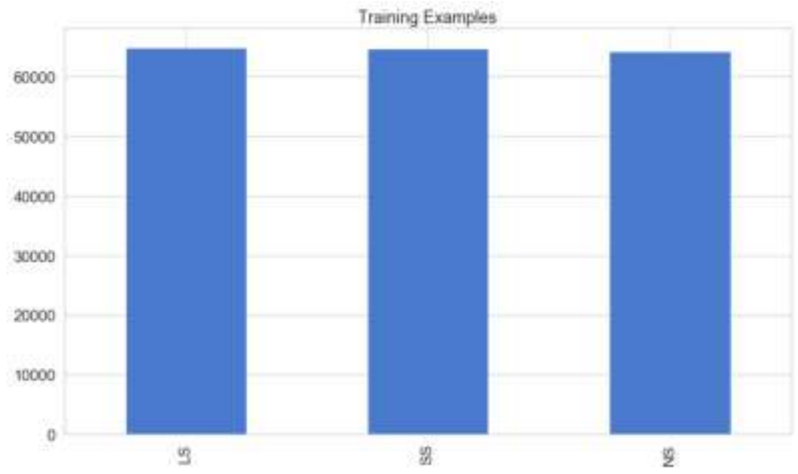


Fig 5.1.3 Training Examples by Activity Unit Type

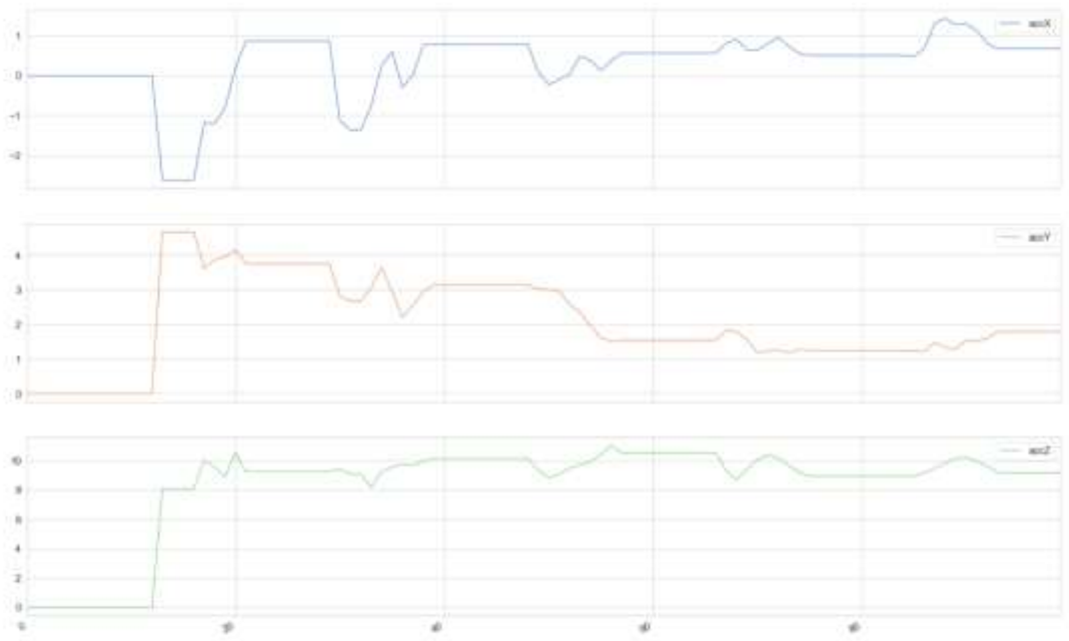


Fig 5.1.4(a) Accelerometer Data for Short Step

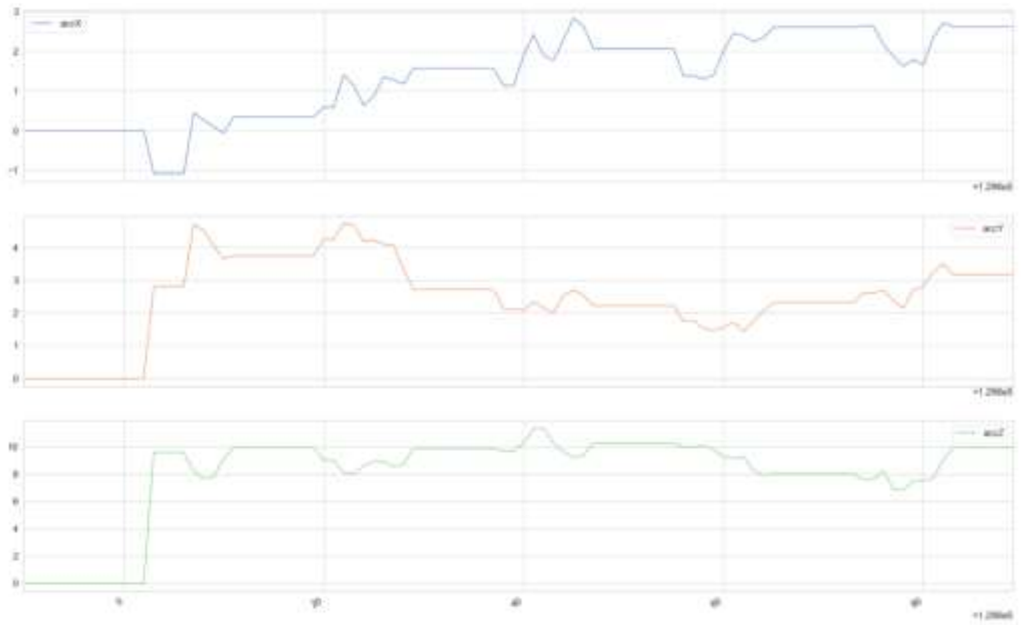


Fig 5.1.4(b) Accelerometer Data for Normal Step

5.1.3 Creating LSTMs [14]

We have implemented the LSTMs 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.

```
import tensorflow as tf
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split
import seaborn as sns
```

1. **Data Pre-Processing**

The above data has to be pre-processed for training since the LSTM model expects fixed-length sequences as training data. The data has been transformed to include only the accelerometer x-axis, y-axis and z-axis data i.e. 3 features. We have decided not to include gyroscope data as this interferes with the accuracy of the LSTM. Accelerometer data is sufficient.

```
N_FEATURES = 3
step = 20
segments = []
labels = []
for i in range(0, len(df) - N_TIME_STEPS, step):
    xs = df['accX'].values[i: i + N_TIME_STEPS]
    ys = df['accY'].values[i: i + N_TIME_STEPS]
    zs = df['accZ'].values[i: i + N_TIME_STEPS]
    label = stats.mode(df['Activity'][i: i + N_TIME_STEPS])
    label = label[0][0]
    segments.append([xs,ys,zs])
    labels.append(label)
```

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

```
reshaped_segments = np.asarray(segments, dtype=np.float32).reshape
(-1, N_TIME_STEPS, N_FEATURES)
labels = np.asarray(pd.get_dummies(labels), dtype = np.float32)
```

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

```
X_train, X_test, y_train, y_test = train_test_split
(reshaped_segments, labels, test_size = 0.2,
                                        random_state = RANDOM_SEED)
```

2. Model Construction

The LSTM model contains 2 LSTM layers stacked on top of each other with 64 hidden units each. It consists of 3 class labels (Walking, Running, Standing / SS, NS, LS), 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.

```
LEARNING_RATE = 0.0025

optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=LEARNING_RATE)
.minimize(loss)

correct_pred = tf.equal(tf.argmax(pred_softmax, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, dtype=tf.float32))
```

```
N_CLASSES = 3
N_HIDDEN_UNITS = 64
def create_LSTM_model(inputs):
    W = {
        'hidden': tf.Variable(tf.random.normal([N_FEATURES, N_HIDDEN_UNITS])),
        'output': tf.Variable(tf.random.normal([N_HIDDEN_UNITS, N_CLASSES]))
    }
    biases = {
        'hidden': tf.Variable(tf.random.normal([N_HIDDEN_UNITS], mean=1.0)),
        'output': tf.Variable(tf.random.normal([N_CLASSES]))
    }
    X = tf.transpose(inputs, [1, 0, 2])
    X = tf.reshape(X, [-1, N_FEATURES])
    hidden = tf.nn.relu(tf.matmul(X, W['hidden']) + biases['hidden'])
    hidden = tf.split(hidden, N_TIME_STEPS, 0)
    #stack 2 LSTM layers
    lstm_layers = [tf.compat.v1.nn.rnn_cell.LSTMCell(N_HIDDEN_UNITS,
                                  forget_bias=1.0) for _ in range(2)]
    lstm_layers = tf.compat.v1.nn.rnn_cell.MultiRNNCell(lstm_layers)

    outputs, _ = tf.compat.v1.nn.static_rnn(lstm_layers, hidden,
                                                        dtype=tf.float32)

    lstm_last_output = outputs[-1]
      return tf.matmul(lstm_last_output, W['output']) + biases['output']
```

3. 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.

```
saver = tf.compat.v1.train.Saver()
history = dict(train_loss=[], train_acc=[], test_loss=[], test_acc=[])
sess = tf.compat.v1.InteractiveSession()
sess.run(tf.compat.v1.global_variables_initializer())
train_count = len(X_train)

for i in range(1, N_EPOCHS + 1):
    for start, end in zip(range(0, train_count, BATCH_SIZE),
                          range(BATCH_SIZE, train_count + 1, BATCH_SIZE)):
        sess.run(optimizer, feed_dict={X: X_train[start:end],
                                       Y: y_train[start:end]})
    _, acc_train, loss_train = sess.run([pred_softmax, accuracy, loss],
                                        feed_dict={X: X_train, Y: y_train})
    _, acc_test, loss_test = sess.run([pred_softmax, accuracy, loss],
                                        feed_dict={X: X_test, Y: y_test})
    history['train_loss'].append(loss_train)
    history['train_acc'].append(acc_train)
    history['test_loss'].append(loss_test)
    history['test_acc'].append(acc_test)

    print(f'epoch: {i} test accuracy: {acc_test} loss: {loss_test}')
predictions, acc_final, loss_final = sess.run([pred_softmax, accuracy,
loss], feed_dict={X: X_test, Y: y_test})

print()
print(f'Final Results: Accuracy: {acc_final} Loss: {loss_final}')
```

5.1.3.1 Activity LSTM

Here, we make use of the following hyperparameters for the code given above:

No. of Features	No. of Classes	Shape of Input	Shape of Output
3	3	{1, 200, 3}	{3}
No. of Samples	Learning Rate	No. of Epochs	Batch Size
200	0.0025	40	1024

Table 5.1.3 Activity LSTM Features

The 3 features are Accelerometer x, y and z. The 3 classes are Walking, Running and Standing.

5.1.3.2 Activity Unit LSTM

In addition to the three steps mentioned above, we also performed another step after Model Construction where we test for the best hyperparameters to be used to obtain the best possible accuracy for the model.

1. **Hyperparameter Testing [15]**

In this step, we first mixed and matched various hyperparameters to calculate the ones that give us the highest accuracy. We performed this step here since we do not know the optimal hyperparameters.

The hyperparameters tested are the time step, train/test split, number of hidden units, learning rate, number of epochs and batch size.

The first algorithm we use is a simple brute force. This gets us closer to the optimal parameters.

Once the key hyperparameters are known - Batch Size, Learning Rate and No. of Samples, we can perform more advanced tests to fine tune the accuracy. In our case, we found Bayesian Optimization to work the best.

```
from bayes_opt import BayesianOptimization

pbounds = {'lr': (--,--), 'bs':(--,--), 'ts':(--,--)}

optimizer = BayesianOptimization(
    f=fit_with,
    pbounds=pbounds,
    verbose=2,
    random_state=1,
)

optimizer.maximize(init_points=0, n_iter=60,)
```

The details of the final results are mentioned under Chapter 6.

In summary, we make use of the following hyperparameters in the main code:

No. of Features	No. of Classes	Shape of Input	Shape of Output
3	3	{1, 200, 3}	{3}
No. of Samples	Learning Rate	No. of Epochs	Batch Size
200	0.004	40	140

Table 5.1.4 Activity Unit LSTM Features

The 3 features are Accelerometer x, y and z. The 3 classes are SS, NS and LS.

5.1.4 Freezing LSTM Models

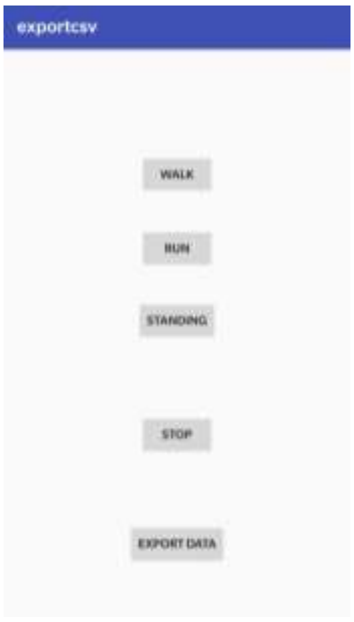
The final step, in order to make the LSTMs that were created into usable assets, is to freeze the models and obtain protocol buffer (.pb) files. This is done with the help of the following code:

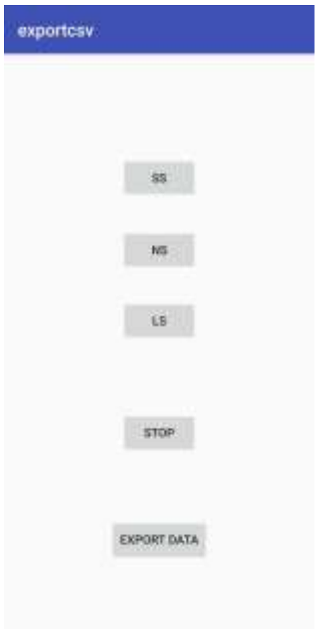
```
from tensorflow.python.tools import freeze_graph
from tensorflow.python.framework import graph_util
from tensorflow.python.tools import optimize_for_inference_lib

graph = tf.compat.v1.get_default_graph()
```

```
input_graph_def = graph.as_graph_def()
output_node_names = ['y_']

output_graph_def =
tf.compat.v1.graph_util.convert_variables_to_constants(sess, input_graph_def,
output_node_names)

with tf.compat.v1.gfile.GFile('./AU batchsize512/ActivityLSTM.pb', 'wb') as f:
    f.write(output_graph_def.SerializeToString())
```

5.2 Android Studio [16]

5.2.1 Creating Data Collection Application





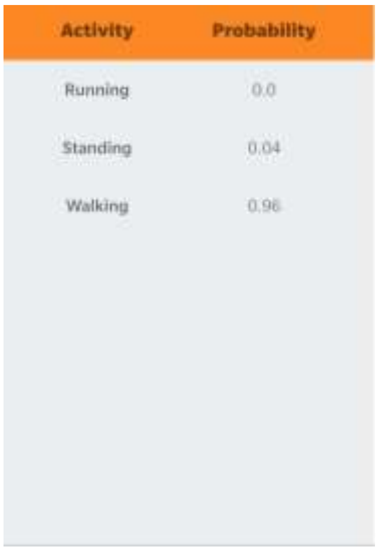
Activity Data Collection App

Activity Unit Data Collection App

```
public void onSensorChanged(SensorEvent event) {

        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            System.arraycopy(event.values, 0, accelerometerReading,
                        0, accelerometerReading.length);
        }

        if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
            System.arraycopy(event.values, 0, gyroscopeReading,
                0, gyroscopeReading.length);
        }
        if(walking==1)
        data.append("\n"+String.valueOf(accelerometerReading[0])+"
              "+String.valueOf(accelerometerReading[1])+"
              "+String.valueOf(accelerometerReading[2])+"
              "+String.valueOf(gyroscopeReading[0])+"
              "+String.valueOf(gyroscopeReading[1])+"
              "+String.valueOf(gyroscopeReading[2])+"
              "+java.time.LocalTime.now()+" SS");
        else if(running==1)
         data.append("\n"+String.valueOf(accelerometerReading[0])+"
              "+String.valueOf(accelerometerReading[1])+"
              "+String.valueOf(accelerometerReading[2])+"
              "+String.valueOf(gyroscopeReading[0])+"
              "+String.valueOf(gyroscopeReading[1])+"
              "+String.valueOf(gyroscopeReading[2])+"
              "+java.time.LocalTime.now()+" NS");
        else if(standing==1)
         data.append("\n"+String.valueOf(accelerometerReading[0])+"
              "+String.valueOf(accelerometerReading[1])+"
              "+String.valueOf(accelerometerReading[2])+"
              "+String.valueOf(gyroscopeReading[0])+"
              "+String.valueOf(gyroscopeReading[1])+"
              "+String.valueOf(gyroscopeReading[2])+"
              "+java.time.LocalTime.now()+" LS");
}
```

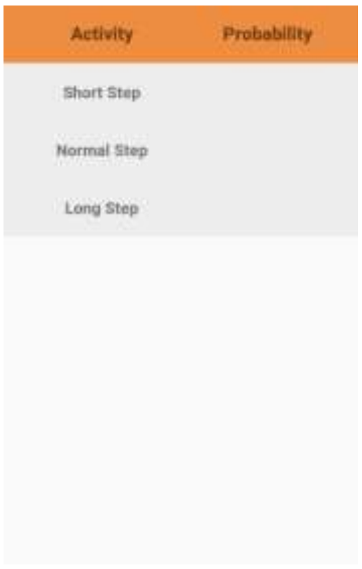
5.2.2 Creating Test Application

We created an application that interfaces with the LSTM models 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 change and output which activity is currently occurring.



Activity	Probability
Running	0.0
Standing	0.04
Walking	0.96

Activity LSTM Testing



Activity	Probability
Short Step	
Normal Step	
Long Step	

Activity Unit LSTM Testing

```
private void activityPrediction() {
    if (x.size() == N_SAMPLES && y.size() == N_SAMPLES &&
       z.size() == N_SAMPLES ) {
        List<Float> data = new ArrayList<>();
        data.addAll(x);
        data.addAll(y);
        data.addAll(z);

        results = classifier.predictProbabilities(toFloatArray(data));
        results2 = HAUclassifier.predictProbabilities(toFloatArray(data));

        runningTextView.setText(Float.toString(round(results[0], 2)));
        standingTextView.setText(Float.toString(round(results[1], 2)));
        walkingTextView.setText(Float.toString(round(results[2], 2)));

        lsTextView.setText(Float.toString(round(results2[0], 2)));
        nsTextView.setText(Float.toString(round(results2[1], 2)));
        ssTextView.setText(Float.toString(round(results2[2], 2)));
       }
}
```

5.2.3 Importing LSTM Models

In order to use the LSTM models that have been created, it is first necessary to import them into the application. This is done as follows:

```
public class HARClassifier {
    static {
        System.loadLibrary("tensorflow_inference");
    }
    private TensorFlowInferenceInterface inferenceInterface;
    private static final String MODEL_FILE = "file:///android_asset/
       ActivityLSTM.pb";
    private static final String INPUT_NODE = "input";
    private static final String[] OUTPUT_NODES = {"y_"};
    private static final String OUTPUT_NODE = "y_";
    private static final long[] INPUT_SIZE = {1, 200, 3};
    private static final int OUTPUT_SIZE = 3;

    public HARClassifier(final Context context) {
        inferenceInterface = new TensorFlowInferenceInterface
       (context.getAssets(), MODEL_FILE);
    }

    public float[] predictProbabilities(float[] data) {
        float[] result = new float[OUTPUT_SIZE];
        inferenceInterface.feed(INPUT_NODE, data, INPUT_SIZE);
        inferenceInterface.run(OUTPUT_NODES);
        inferenceInterface.fetch(OUTPUT_NODE, result);
        return result;
    }

}
```

In case of the 2nd LSTM, the asset imported would be AULSTM.pb.

5.2.4 Reading Accelerometer Data

The input to the models is the accelerometer data, which is continuously read by the application in real-time at 100 Hz. It is implemented using a method in the MainActivity class.

```
protected void onResume() {
    super.onResume();
    getSensorManager().registerListener(this,
getSensorManager().getDefaultSensor(Sensor.TYPE_ACCELEROMETER),10000,10000);
    getSensorManager().registerListener(this,
getSensorManager().getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD), 10000,10000);
}
```

5.2.5 Moving Distance Estimator

The Moving Distance Estimator is the integral component of the application, and is implemented using buffers and a mode function.

```
public class DistanceEstimator {

    private static List<Integer> bufferX = new ArrayList<>();
    private static List<Integer> bufferY = new ArrayList<>();
    private Integer unitValue;
    private int bufferSize = 3;

    public float getStepLengthX(int activity, int unit)
    {
        if (activity == 0) //run
        {
            bufferX.add(unit);
            unitValue = mode(bufferX);
            System.out.println(bufferX);
            System.out.println(unitValue);

            if(bufferX.size() == bufferSize)
                bufferX.remove(0);

            return 75;
        }
        else if (activity == 2) //walk
        {
            bufferX.add(unit);
            unitValue = mode(bufferX);
            System.out.println(bufferX);
            System.out.println(unitValue);


            if(bufferX.size() == bufferSize)
                bufferX.remove(0);

            if(unitValue == 0) //ls
            {
                if(bufferX.contains(3))
                    return 59.24f; //ls pixel value
                else
                    return 39.5f;
            }
            else if(unitValue == 1) //ns
            {
                if(bufferX.contains(3))
                    return 52.66f; //ns pixel value
                else
                    return 35.11f;
            }
            else //ss
            {
                if(bufferX.contains(3))
                    return 47.4f; //ss pixel value
                else
                    return 31.6f;
            }
        }
        return 0;
    }
}
```

getStepLengthY is defined in a similar way.

5.2.6 Positioning

The first major feature of this application, positioning of the Blue Dot as the user moves, is implemented by first checking if the Running or Walking activity is being performed, and then varying the animation speeds based on the output of the Moving Distance Estimator module.

```
private void activityPrediction()
{

        if (x.size() == N_SAMPLES && y.size() == N_SAMPLES &&
                                          z.size() == N_SAMPLES ) {
            List<Float> data = new ArrayList<>();
            data.addAll(x);
            data.addAll(y);
            data.addAll(z);

        activity = activityClassifier.getActivity(results);

            if(activity ==0 || activity ==2) //RUN || WALK
            {
                stepLengthX = distanceEstimator.getStepLengthX(activity);
                stepLengthY = distanceEstimator.getStepLengthY(activity);
                Xcpos = Xnpos;
                Xnpos = Xnpos +  (nSteps*stepLengthX*Math.cos
                                                   (Math.toRadians(azimuth)));

                Ycpos = Ynpos;
                Ynpos = Ynpos +  (nSteps*stepLengthY*Math.sin
                                                   (Math.toRadians(azimuth)));

                objectAnimatorX = ObjectAnimator.ofFloat(blueDot,"x",density
                                  * (float)Xcpos, density* (float)Xnpos);
                objectAnimatorX.setDuration(2400);
                objectAnimatorY = ObjectAnimator.ofFloat(blueDot,"y",density
                                  * (float)Ycpos,density* (float)Ynpos);
                objectAnimatorY.setDuration(2400);

                AnimatorSet animSetXY = new AnimatorSet();
                animSetXY.playTogether(objectAnimatorX, objectAnimatorY);
                animSetXY.start();

            }
            x.clear();
            y.clear();
            z.clear();

        }
 }
```

5.2.7 Orientation

The orientation of the Blue Dot is crucial for the user to know which direction he is facing with respect to the location on the map. It is implemented using the geomagnetic sensor of the smartphone.

```
public void onSensorChanged(SensorEvent event) {
        final float alpha = 0.97f;

        activityPrediction();
        synchronized (this){


        if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD ){
            mGeomagnetic[0] = alpha*mGeomagnetic[0]+(1-alpha)*event.values[0];
            mGeomagnetic[1] = alpha*mGeomagnetic[1]+(1-alpha)*event.values[1];
            mGeomagnetic[2] = alpha*mGeomagnetic[2]+(1-alpha)*event.values[2];

            }
        }

        float R[] = new float[9];
        float I[] = new float[9];
        boolean success = SensorManager.getRotationMatrix
                                                    (R,I,mGravity,mGeomagnetic);

        if(success)
        {
            float orientation[] = new float[3];
            SensorManager.getOrientation(R,orientation);
            azimuth = (float) Math.toDegrees(orientation[0]);
            azimuth = (azimuth+360) % 360;


            animRot = ObjectAnimator.ofFloat(blueDot,"rotation",
                                                        cAzimuth,azimuth);
            animRot.setDuration(1);
            animRot.start();

            cAzimuth=azimuth;

        }


    }
```

5.2.8 Navigation

The second major feature of the application, navigation, is implemented using Breadth First Search and the Shortest Path algorithm.

```
public boolean BFS(int src, int dest, int pred[], int dist[]) {

        Queue<Integer> queue = new LinkedList<>();
        int INT_MAX = 1000;

        boolean visited[] = new boolean[n];

        // initially all vertices are unvisited
        // so v[i] for all i is false
        // and as no path is yet constructed
        // dist[i] for all i set to infinity
        for (int i = 0; i < n; i++) {
            visited[i] = false;
            dist[i] = INT_MAX;
            pred[i] = -1;
        }

        // now source is first to be visited and
        // distance from source to itself should be 0
        visited[src] = true;
        dist[src] = 0;
        queue.add(src);

        int u;
        // standard BFS algorithm
        while (  queue.size() != 0) {

            u = queue.remove();

            for (int i = 0; i < nodes[u].neighbours.length; i++) {
                if (visited[nodes[u].neighbours[i]] == false) {
                    visited[nodes[u].neighbours[i]] = true;
                    dist[nodes[u].neighbours[i]] = dist[u] + 1;
                    pred[nodes[u].neighbours[i]] = u;
                    queue.add(nodes[u].neighbours[i]);

                    // We stop BFS when we find
                    // destination.
                    if (nodes[u].neighbours[i] == dest)
                        return true;
                }
            }
        }

        return false;
    }


public int[] getShortestPath(int s,int dest) {
        // predecessor[i] array stores predecessor of
        // i and distance array stores distance of i
        // from s
        int pred[] = new int[n];
        int dist[] = new int[n];

        if (BFS(s,dest,pred,dist) == false) {
            System.out.println("Given source and destination are not connected");
            return null;
        }

        // vector path stores the shortest path
        List<Integer> path = new ArrayList<Integer>();
        int crawl = dest;
        path.add(crawl);
        while (pred[crawl] != -1) {
            path.add(pred[crawl]);
            crawl = pred[crawl];
        }

        // distance from source is in distance array
        System.out.println("Shortest path length is : "+dist[dest]);

        int[] path1 = new int[path.size()];
        // printing path from source to destination
        System.out.println("\nPath is: \n");
        for (int i = path.size() - 1,j=0; i >= 0; i--,j++)
        {
            path1[j] = path.get(j);
            System.out.println(path.get(i) + " ");
        }

        return path1;
    }
```

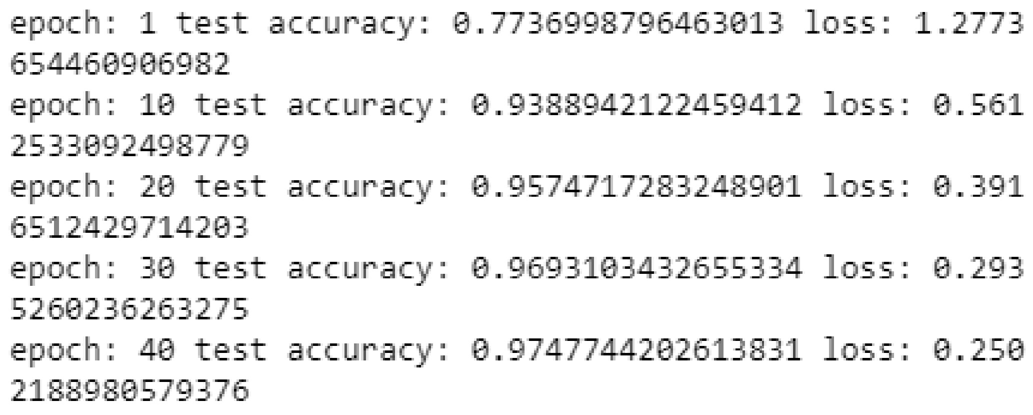
Chapter 6

Testing and Results

6.1 LSTM Results

6.1.1 Activity LSTM Results

We feed arrays of data 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 use during implementation.



```
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
```

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.

The way in which the .pb file is imported into the android application is tricky. It requires that the shape of the features and classes are the same as those in the frozen graph LSTM model. Once imported into the android application, it outputs probabilities corresponding to each class in real time.

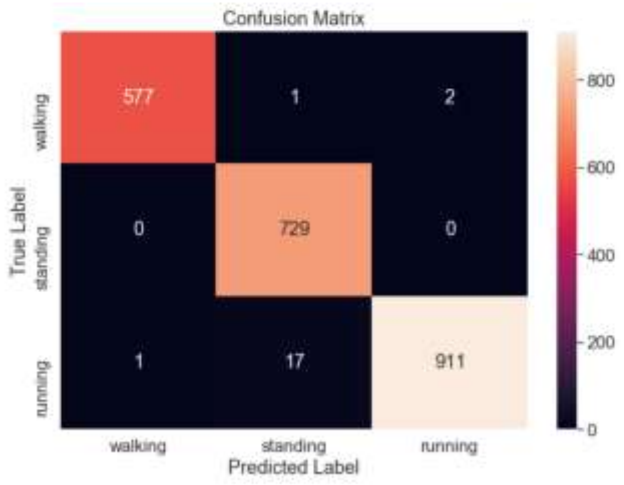


Fig 6.1.1 Confusion Matrix for 1st LSTM

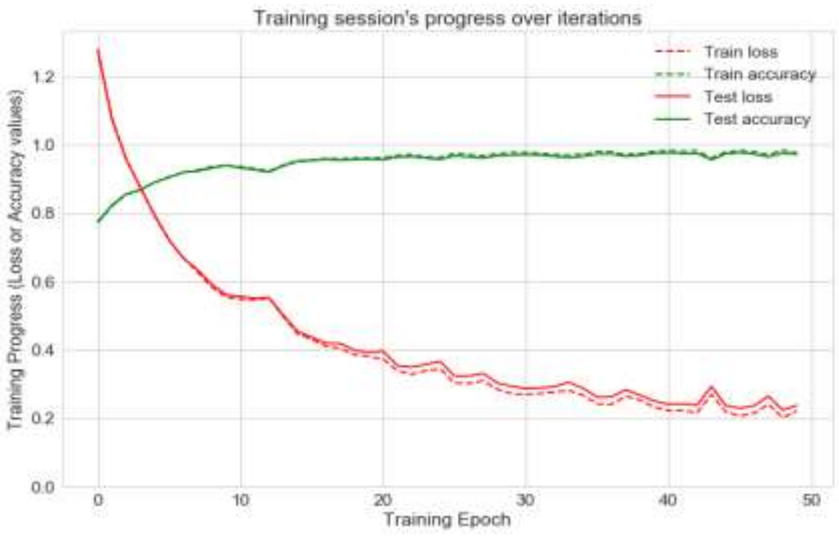
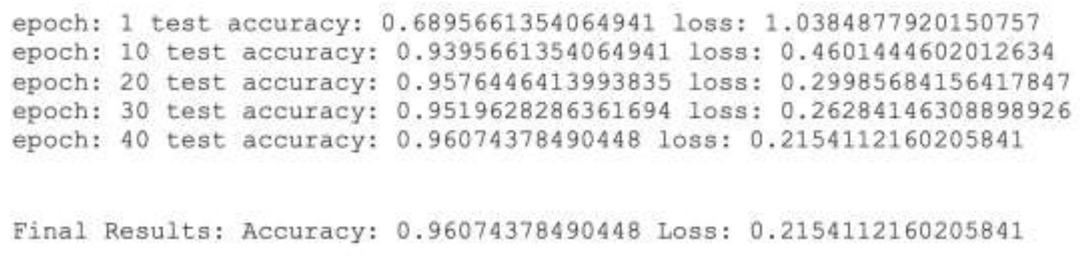


Fig 6.1.2 Training Session Progress over Iterations for 1st LSTM

6.1.2 Activity Unit LSTM Results

We feed arrays of data 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 use during implementation.

Output:



```
epoch: 1 test accuracy: 0.6895661354064941 loss: 1.0384877920150757
epoch: 10 test accuracy: 0.9395661354064941 loss: 0.4601444602012634
epoch: 20 test accuracy: 0.9576446413993835 loss: 0.29985684156417847
epoch: 30 test accuracy: 0.9519628286361694 loss: 0.26284146308898926
epoch: 40 test accuracy: 0.96074378490448 loss: 0.2154112160205841


Final Results: Accuracy: 0.96074378490448 Loss: 0.2154112160205841
```

It took us approximately 24 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.

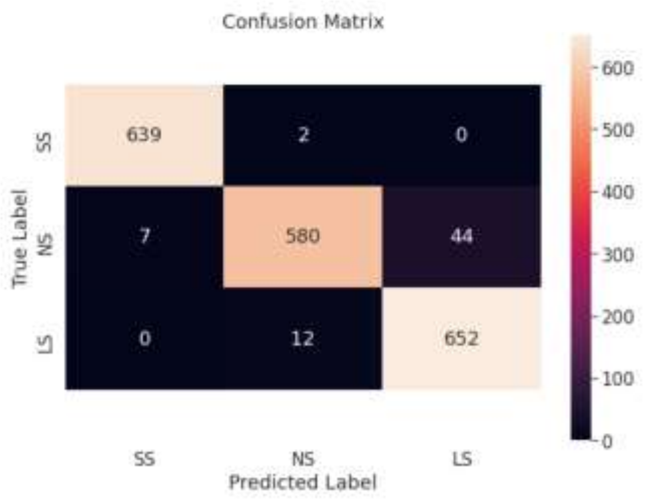


Fig 6.1.3 Confusion Matrix for 2nd LSTM

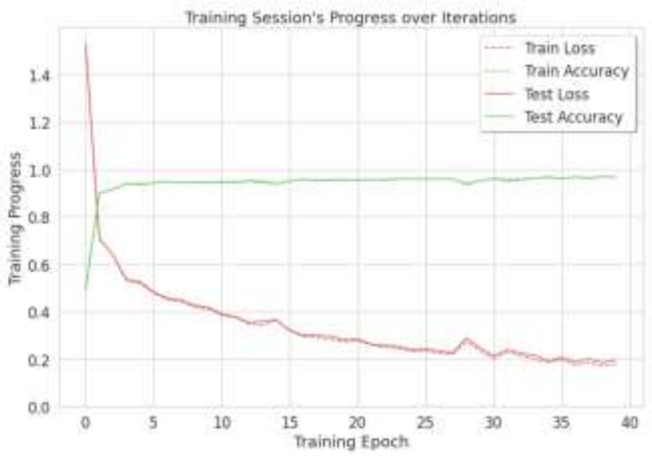


Fig 6.1.4 Training Session Progress over Iterations for 2nd LSTM

6.2 Mapping Results

We have chosen to make use of the floor map of the ground floor of the GEC Main Building the map on which our app will operate.

We have also divided it into several nodes and created a node map or node graph. This is what is used for navigation.

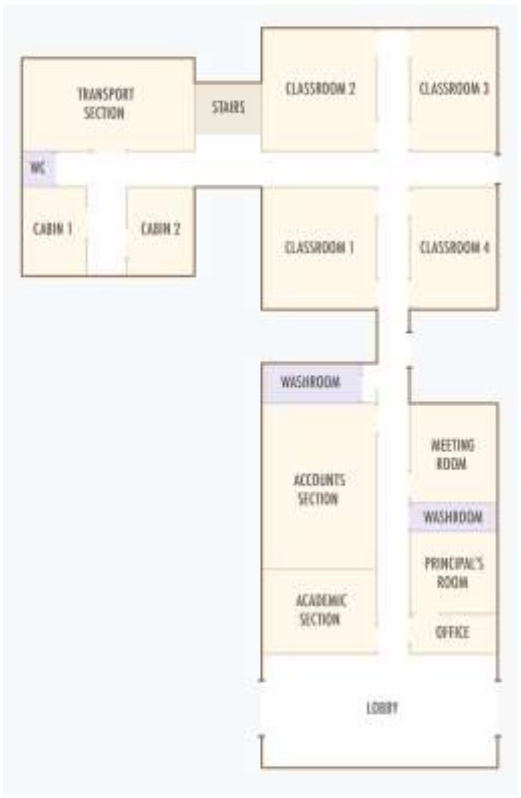


Fig 6.2.1 Map

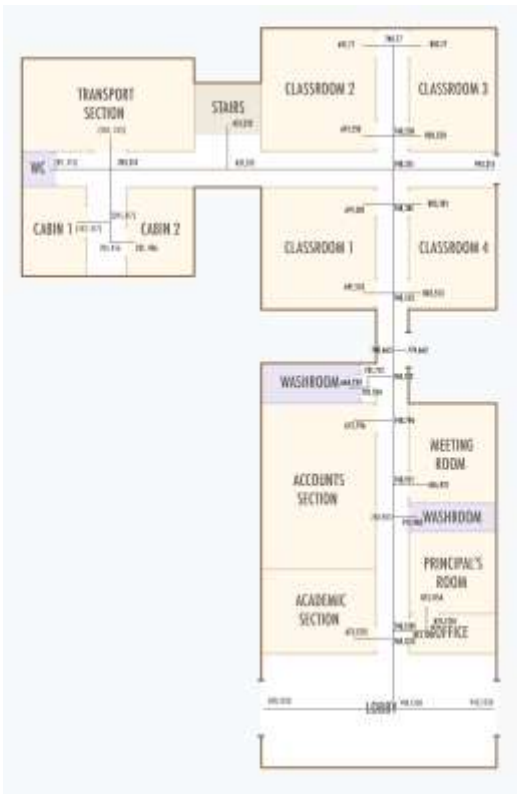


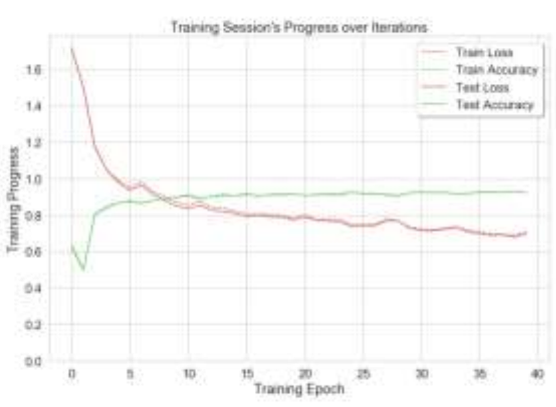
Fig 6.2.2 Node Graph

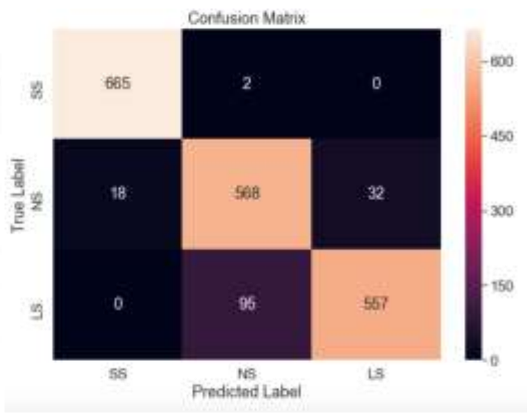
6.3 Brute Force Hyperparameter Identification

This is carried out in order to identify the key hyperparameters for the 2nd LSTM. The default hyperparameters are as follows, with which the experiments are to be compared with:

- Time Step = 100
- Train/Test Split = 80/20 %
- Hidden Units = 64
- Learning Rate = 0.0025
- Epochs = 40
- Batch Size = 1024

The default accuracy is 92.4% and loss is 69.8%.



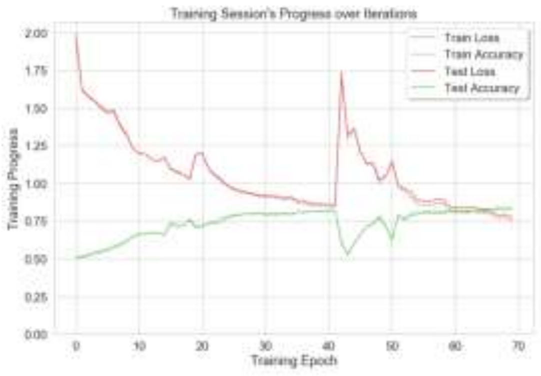


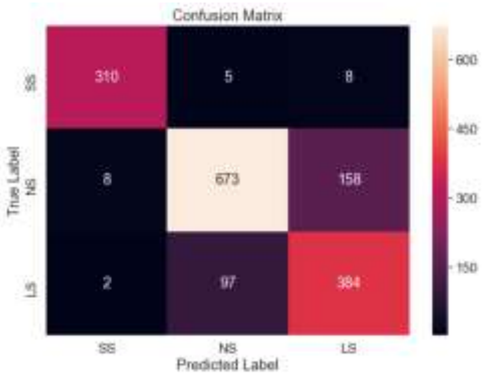
Test #1

Time Step = 150 Epochs = 70

Accuracy = 83.1% Loss = 77.3%

Increasing the time step and epochs reduces accuracy.



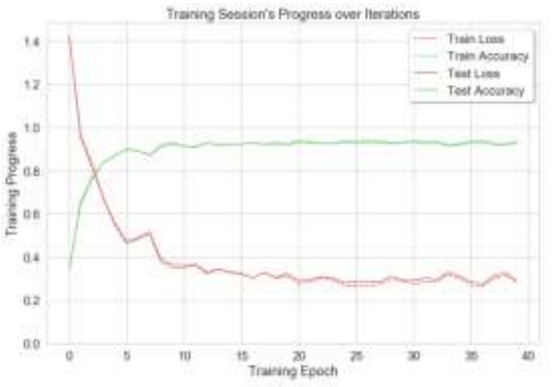


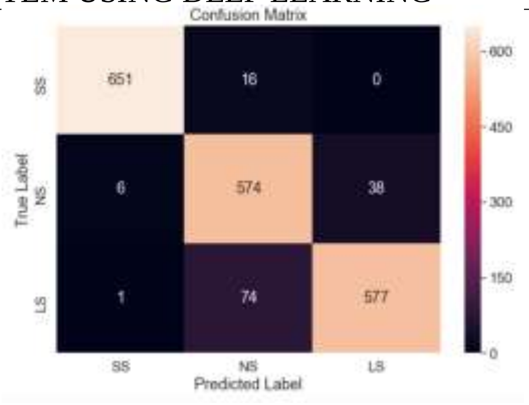
Test #2

Learning Rate = 0.01 Batch Size = 90

Accuracy = 93% Loss = 28.9%

Increasing the learning rate and decreasing the batch size drastically improves the LSTM.



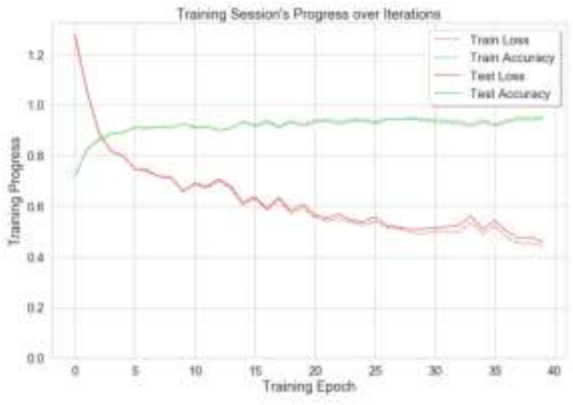


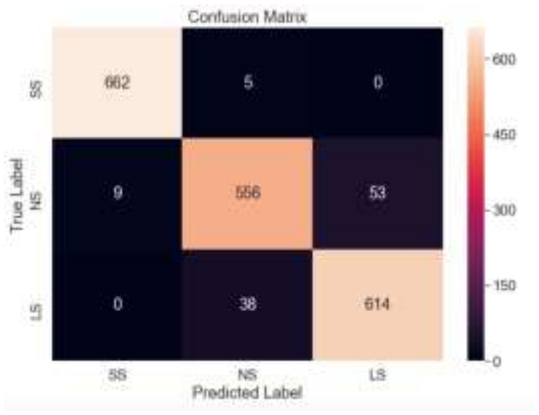
Test #3

Batch Size = 512

Accuracy = 94.57% *Loss = 45.8%*

Higher loss but the accuracy is much better.



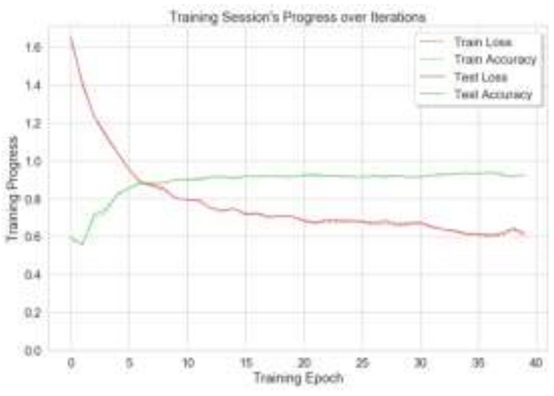


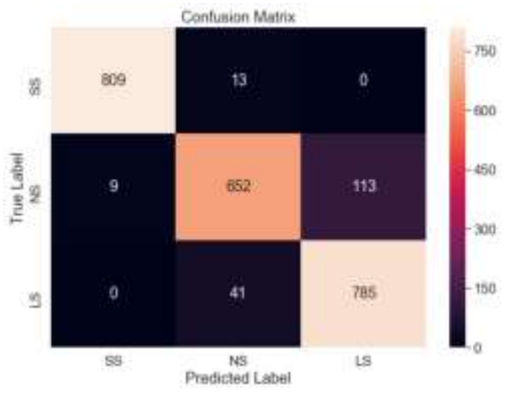
Test #4

Train/Test Split = 75/25 %

Accuracy = 92.7% Loss = 61.7%

Negligible change from the default parameters.



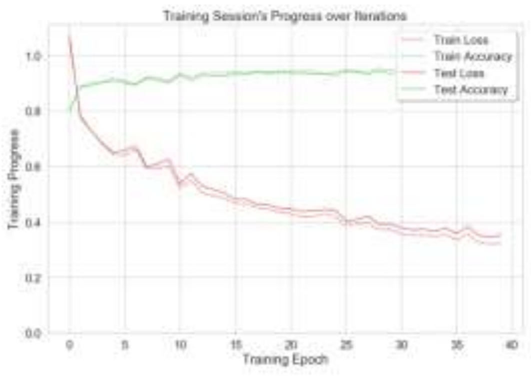


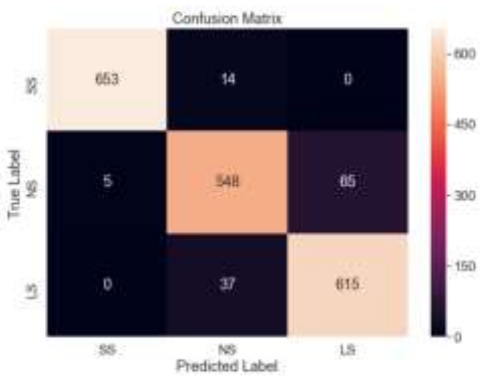
Test #5

Batch Size = 256

Accuracy = 93.75%   Loss = 35.5%

In between results of Test #2 and #3. Batch Size seems to be the key hyperparameter.



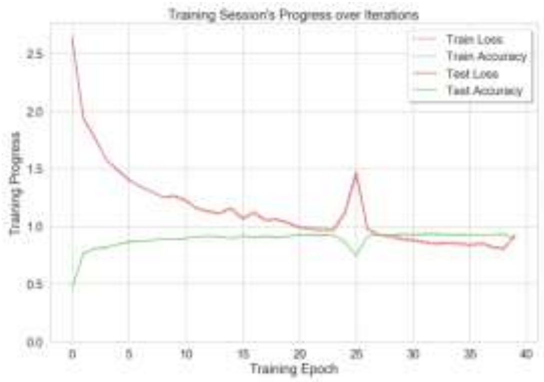


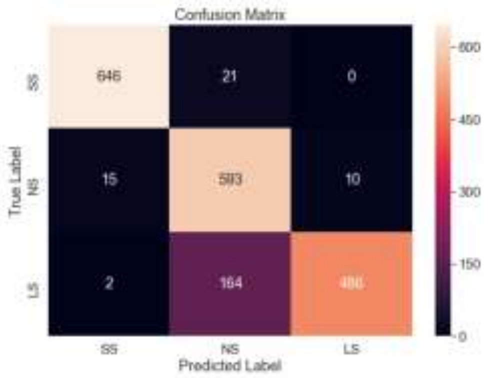
Test #6

Hidden Units = 128

Accuracy = 89%   Loss = 92.17%

Terrible accuracy and loss.



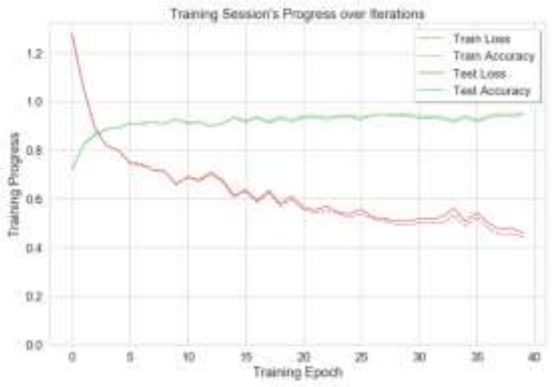


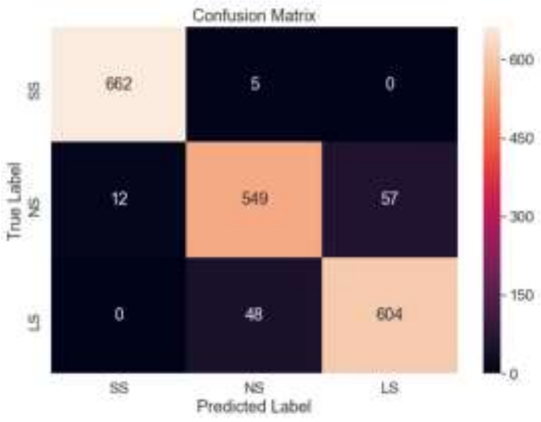
Test #7

Batch Size = 512   Learning Rate = 0.001

Accuracy = 93.7%   Loss = 60.3%

Decreasing the learning rate seems to have a negative effect.





Hence, we can conclude from the above experiments that increasing the time steps, epochs and hidden units, and decreasing the learning rate has a negative effect, changing the train/test split has negligible effects, and batch size seems to be the key hyperparameter. Thus we use Test #3 as our 2nd LSTM model.

6.4 Bayesian Optimization

Bayesian Optimization was chosen as the optimization algorithm in order to test for the optimal hyperparameters for the 2nd LSTM.

Here are a few tables in order to illustrate the testing process undertaken.

Test 1 – Test to find the best Batch Size & Learning Rate for No. of Samples/Time Step = 100

Index	Accuracy	Loss	Batch Size	Learning Rate	Time Step
1	0.9396		32	0.003093	100
2	0.9458	0.4579	512	0.0025	100
3	0.9448	0.5445	92	0.000387632	100
4	0.9437	0.2712	143	0.003672	100
5	0.9432	0.2699	139	0.003752	100
6	0.9458	0.2789	130	0.003903	100
7	0.9463	0.2734	143	0.003152	100
8	0.9479	0.2855	146	0.003	100
9	0.9484	0.2686	143	0.003468	200

Table 6.4.1 Results of Bayesian Optimization Test 1

We notice here that the upper limit is 94.84% accuracy. This is attainable with Batch Size = 143 and Learning Rate = 0.003468. This gives us a range of Batch Size = (130, 150) and Learning Rate = (0.003, 0.004).

Test 2 – Test to find the best Batch Size (130, 150) & Learning Rate (0.003, 0.004) for No. of Samples/Time Step = 200

Index	Accuracy	Loss	Batch Size	Learning Rate	Time Step
1	0.9711		150	0.003493	200
2	0.9737	0.1998	142	0.003652	200
3	0.9783	0.1708	140	0.004	200

Table 6.4.2 Results of Bayesian Optimization Test 2

Here, the upper limit is much higher – 97.8%. We notice that the Batch Size is 140 and the Learning Rate is 0.004.

Test 3 – Test to find the best Time Step (100, 300) for Batch Size = 143 & Learning Rate = 0.003652

Index	Accuracy	Loss	Batch Size	Learning Rate	Time Step
1	0.9551	0.251	143	0.003652	128
2	0.9582	0.247	143	0.003652	131
3	0.9592	0.2255	143	0.003652	140
4	0.9711	0.2388	143	0.003652	190
5	0.9861	0.1393	143	0.003652	277
6	0.984	0.1488	143	0.003652	245
7	0.984	0.1675	143	0.003652	273
8	0.9835	0.169	143	0.003652	276

Table 6.4.3 Results of Bayesian Optimization Test 3

Much better upper limit – 98.35%. However, this cannot be used due to inconsistencies that may arise.

Test 4 – Test to find the best Time Step (100, 200), Batch Size (130, 150) & Learning Rate (0.003, 0.004)

Index	Accuracy	Loss	Batch Size	Learning Rate	Time Step
1	0.953	0.2456	130	0.004	133
2	0.9582	0.247	143	0.003652	131

Table 6.4.4 Results of Bayesian Optimization Test 4

This test does not reveal any new information.

In the end, we chose the result of Test #2 – Batch Size = 140, Learning Rate = 0.004 and Time Step = 200 with Accuracy = 96.1% and Loss = 21.5%.

6.5 Optimal Frequency

- We performed multiple tests in order to find out the optimal frequencies and sample size.
- There are two main frequencies:
    - The first frequency is the rate at which the program reads the accelerometer data.
    - The second frequency is the rate at which we collect the data.
- The sample size is the number of timesteps of accelerometer data that are considered at a given time as input to the LSTMs.

The tests we conducted are as follows:

Test #1

Here, we still considered gyroscope data along with accelerometer data. However, we noticed that there was a higher delay (23 seconds) and the accuracy was lower (95%). Hence we decided to remove gyroscope data and that improved both the delay (20 seconds) and the accuracy (98%).

Test #2

We found that the frequency and sample size were key parameters for the delay.

The frequency at which we collected the data was 10 Hz. This means that 10 accelerometer readings were taken every second.

The frequency at which the data was being read was also 10 Hz, which means that during the running of the app, it will take 10 accelerometer readings every second in real time.

The sample size at the time was taken to be 200, which means that the program waits for 200 accelerometer readings to be read before performing the prediction of the class label.

This resulted in a delay of 20 seconds, since it has to wait 20 seconds for 200 readings to be read. (200 readings/10Hz = 20 seconds)

Test #3

Knowing the above, we decided to first decrease the sample size to 100 readings. This served a double purpose. Firstly, it made it so that the program would wait for less time since it required fewer readings to make a prediction. Secondly, a smaller sample made it so that the LSTMs focus on the main part of the signal and are less susceptible to noise.

This resulted in a delay of 10 seconds, keeping the frequencies the same (10 Hz). (100 readings/10Hz = 10 seconds)

Test #4

We then decided to take a step back and increase both the frequencies to 50 Hz. This would make it so that more readings are taken and matched every second.
We chose to keep the sample size as 200 readings here, just to check the effect of sample size on accuracy.
This resulted in a delay of 4 seconds, and the accuracy didn't change much in comparison with Test #3. (200 readings/50 Hz = 4 seconds)

Test #5

Finally, we chose to use frequencies of 50 Hz and a sample size of 100.
This resulted in the most optimal delay of 2 seconds, without affecting the accuracy too much.
If we increased the sample size over 200 it would decrease the accuracy (more noise), and if we decreased it below 100 it would also decrease the accuracy (less data available).
If we made the frequencies 100 Hz it would give a nearly flat graph, since the first 100 readings would be taken instantaneously and not be very informative.
Hence we initially decided to go with this. (100 readings/50 Hz = 2 seconds)

Test #6

The above values did not yield good results for the second LSTM, which requires faster input. Hence, we tried using 100 Hz, and 200 samples to make sure it doesn't give a flat graph. This gave the same optimal delay of 2 seconds, and did not affect accuracy. Since the second LSTM worked well with it, to avoid inconsistencies, we chose to go with the same for the first LSTM. (200 readings/100 Hz = 2 seconds)

6.6 Optimal Step Length Pixel Values

In order to obtain good accuracy for the movement of the Blue Dot across the screen, we tested various pixel values that could be used in the Moving Distance Estimator to correspond to Short Step, Normal Step and Long Step.

The tests conducted are as follows:

Tests for finding Optimal Pixel Values for Step Length x

Test Room Length (along x) = 5.84m

Map Room Length (along x) = 6.58m

Normal Step Length = 584/9 = 64.8cm

Long Step Length = 584/8 = 73cm

Short Step Length = 584/10 = 58.4cm

Map Room pixels = (1015,y) - (837,y) = 178px

For NS,

658cm = 178px

64.8cm = 18px

For LS,

658cm = 178px

73cm = 20px

For SS,

658cm = 178px

58.4cm = 16px

Test 1: walk for 584cm with x=(18)

It results in the Blue Dot moving much more than 584cm.

Assumption:

It takes 2 seconds to take 3 normal steps, so initially there is a 3 step offset before the Blue Dot begins moving.

Once it starts, it checks every step (since it does it continuously at 100Hz).

Hence, as it moves, it should add 18px every step, but initially it should have a 18*3 = 54px offset.

Test 2: walk for 584cm with x=(54+18)

The expected result is that it should take 9 normal steps (64.8cm)

The result is that it moves approximately 89px, which means that it only considers 3 steps (54px) + 2 steps (18px+18px) or 329cm (658*89/178).

Assumption:

It is missing 4 more steps.

It checks every 2 steps, so the increment should actually be 18*2 = 36px.

Test 3: walk for 584cm with x=(54+36)

The result is that it moved approximately 162px (or 599cm), which is much more than 584cm.

Assumption:

The problem occurs due to approximating 17.5px to 18px, which adds up over time.

To get a better accuracy, we should consider NS as 17.5px.

Test 4: walk for 584cm with x=(52.66+35.11)

The result is that it moved approximately 155px (or 573cm).

This gives an accuracy of ±11cm, which is quite good.

Hence, we take the NS step length pixels as 52.66 for the initial step, and 35.11 as the increment steps. The same is done for LS and SS with their respective values.

Tests for finding Optimal Pixel Values for Step Length y

Test Room Length (along y) = 9.26m

Map Room Length (along y) = 9.26m

Normal Step Length = 584/9 = 64.8cm

Long Step Length = 584/8 = 73cm

Short Step Length = 584/10 = 58.4cm

Map Room pixels = (x,50) - (x,300) = 250px

For NS,

926cm = 250px

64.8cm = 17.49px

For LS,

926cm = 250px

73cm = 19.71px

For SS,

926cm = 250px

58.4cm = 15.77px

Test 1: walk for 926cm with y=(same as x)

It results in the Blue Dot moving much more than 926cm, overshooting by almost 40px.

Assumption:

The same value might work by rounding to the highest, but there is an issue when you consider pixel values as floats. There might be stretching due to the map dimensions as well.

Test 2: walk for 926cm with y=(52.48+34.98)

It results in an offshoot of about ±15cm, which is fine.

Hence, we take the NS step length pixels as 52.48 for the initial step, and 34.98 as the increment steps. The same is done for LS and SS with their respective values.

6.7 Consistency Test

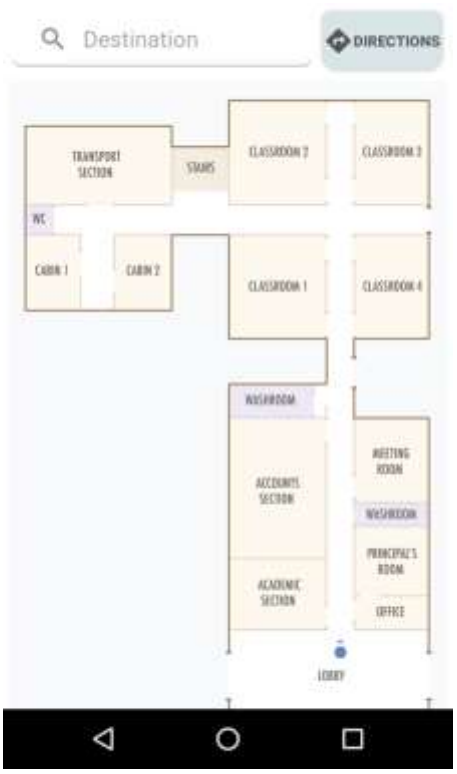
To test the application, we had to see if it gave consistent results along the same path. Consistency testing was done by walking along the same path, to and fro, and checking if we arrived back at the starting point. The test was considered consistent if there was an error of less than 20cm, and inconsistent if there was an error of more than 20cm. This was done 10 times for 10 different paths. The results are as follows:

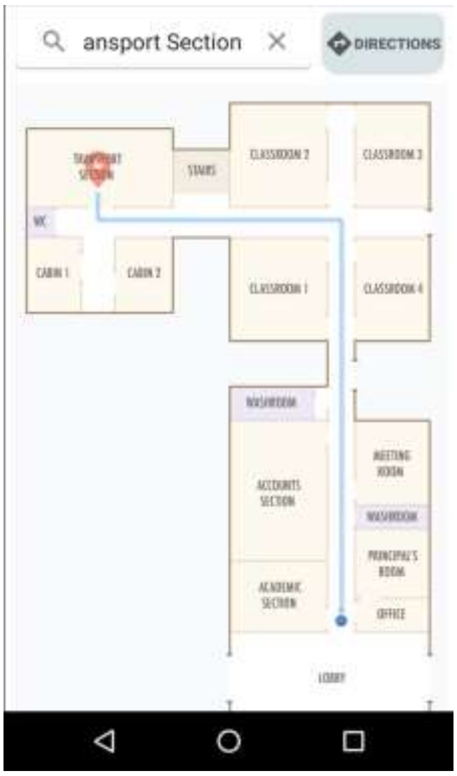
Path	No. of Consistent	No. of Inconsistent	Consistency (%)	Average Error (Consistent)	Average Error (Inconsistent)
1	7	3	70%	13.365	23.67
2	6	4	60%	14.11	21.635
3	8	2	80%	13.96	21.64
4	6	4	60%	16.33	22.13
5	5	5	50%	18.26	24.82
6	7	3	70%	12.22	21.71
7	7	3	70%	13.14	20.94
8	6	4	60%	14.82	23.654
9	9	1	90%	8.45	21
10	7	3	70%	12.435	21.39
AVG	6.8	3.2	68%	13.71	22.26

Table 6.7.1 Consistency Test Results

Average Consistency was found to be 68%, with the Average Consistent Error as 13.71 and Average Inconsistent Error as 22.26.

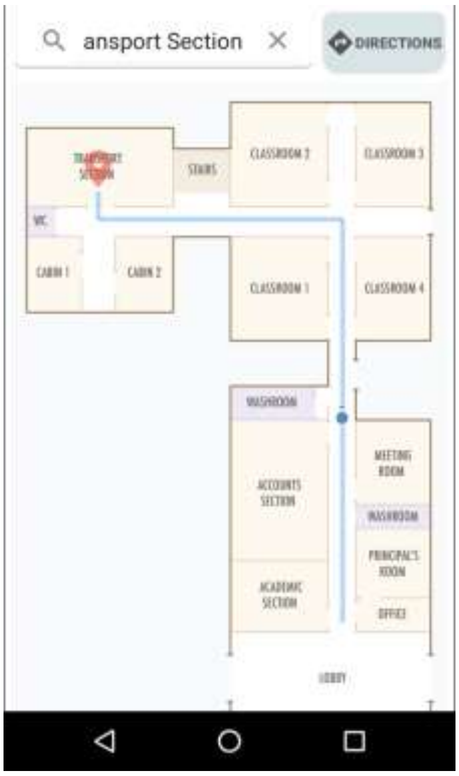
6.8 GUI Snapshots

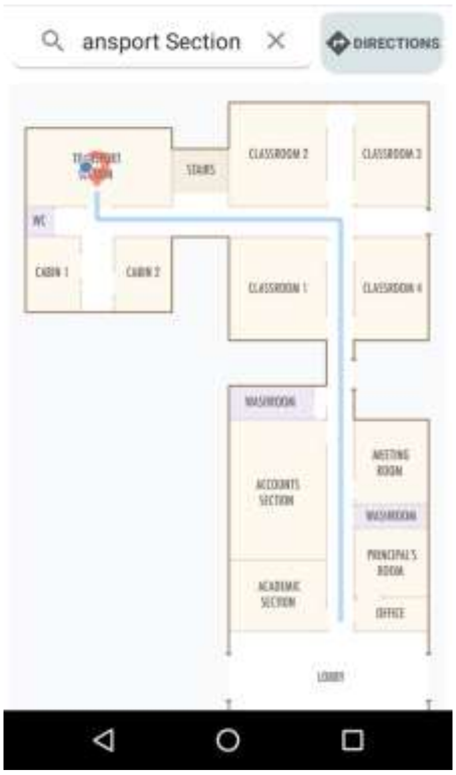




User selects starting position User searches for destination

Fig 6.8.1 GUI Snapshots





User walks towards destination User reaches destination

Chapter 7

Conclusion

7.1 General Conclusion

The project presents an Indoor Positioning System that uses Deep Learning in order to provide accurate real-time positioning and tracing of a user. It uses two LSTMs in order to provide better accuracies than most other IPS solutions. We have also implemented Indoor Navigation on top of positioning, which will enable the user to find the shortest path to their destination. We have used the map of the GEC Main Building for our application, however any map could be used – as long as the node graph is available. This is due to the fact that the positioning is relative, unlike most other solutions that employ absolute positioning, and so it is tailored to the user rather than the environment. Hence, it is low cost and requires low maintenance.

In conclusion, all five modules were completed, and an additional Navigation application was implemented over the final product. The final accuracy was found to be 15(±3)cm if the smartphone is held the way the data was collected, and 30(±10)cm if not.

7.2 Future Work

A few improvements can be made to our application in order to provide a better user experience such as detection of walls and smoother movement of the Blue Dot. In terms of the backend, there is still room for improvement of the 2nd LSTM by collecting better data. It is also possible to optimize the code by storing the nodes in a database or cloud rather than as a list in Java. This would also enable storing node graphs and maps of multiple places, hence scalable. Another feature could be having multiple floors, and switching to the map of the other floor upon reaching the stairs. We can further improve the system by training the LSTM to detect abnormal activities like picking up the phone, keeping the phone in the pockets, or any sudden movements.

For applications, it could be used as a fitness tracker, for security in an office campus, or as a way to keep track of the elderly. For commercial purposes, it could be used in large avenues such as malls or festivals to help people navigate, or in hospitals to navigate easily.

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.
5. Akhand Pratap Mishra, “Introduction to Convolution Neural Networks”, *geeksforgeeks* [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/
6. aishwarya.27, “Introduction to Recurrent Neural Networks”, *geeksforgeeks* [Online]. Available: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
7. Michael Phi, “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation”, *Towards Data Science* [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
8. Python, “Python Documentation”, python [Online]. Available: https://docs.python.org/3/
9. Anaconda, “Anaconda Website”, *Anaconda* [Online]. Available: https://www.anaconda.com/
10. Jupyter, “Jupyter Website”, *Jupyter* [Online]. Available: https://jupyter.org/
11. Android, “Android Studio Website”, *Android Studio* [Online]. Available: https://developer.android.com/studio
12. Diagrams, “Diagramming Software”, *diagrams* [Online]. Available: https://app.diagrams.net/
13. Jeff Heaton, “Installing Python and TensorFlow”, *github* [Online]. Available: https://github.com/jeffheaton/t81_558_deep_learning/blob/master/install/tensorflow-install-jul-2020.ipynb

14. TensorFlow Documentation, “API Documentation”, tensorflow [Online]. Available: https://www.tensorflow.org/api_docs

15. fmfn, “Bayesian Optimization”, github [Online]. Available: https://github.com/fmfn/BayesianOptimization

16. Android Studio Documentation, “Documentation”, Android Studio [Online]. Available: https://developer.android.com/docs