THESIS

# SMART INDOOR LOCALIZATION USING MACHINE LEARNING TECHNIQUES

Submitted by

Viney Anand Ugave

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2014

Master's Committee:

Advisor: Sudeep Pasricha

Charles Anderson
Sourajeet Roy

# ABSTRACT

## SMART INDOOR LOCALIZATION USING MACHINE LEARNING TECHNIQUES

The advancement of smartphone devices has led to a generation of new applications and solutions. These devices give away a great deal of information about the user (*location, posture, communication patterns, etc*.), which helps in capturing the user's context. Such information can be utilized to create smarter apps from which the user can benefit. A challenging new area that is receiving a lot of attention is Indoor Localization whereas interest in location-based services is also rising. While numerous smartphone based indoor localization techniques have been proposed, these techniques have many shortcomings related to accuracy and consistency. More importantly, these techniques entail high-energy consumption that can quickly drain a smartphone battery. In this thesis, we propose innovative techniques based on machine learning algorithms and smart sensor management for effective Indoor Localization using smartphones.

We evaluated our techniques on several indoor environments with diverse characteristics and show improvements over several state-of-the-art techniques from prior work. The extensive use of sensors and Wi-Fi scans can deplete the smartphone battery and so we quantitatively accounted for all the modules that consume the battery power. We also performed energy and accuracy tradeoff analysis to provide a broader understanding of how to smartly use these techniques. Furthermore, we investigated, implemented and tested both *sensor* and *machine learning* based techniques. Our best technique achieved an average accuracy between 1-3 meters across most of our evaluated indoor paths.

# ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support has made the completion of this thesis possible.

First, I would like to express my deep gratitude to Prof. Sudeep Pasricha, whose guidance support and motivation made this project possible. His counsel, not only in the matters of my research project but on various matters throughout my Master's program has turned out to be very fruitful. I first had the opportunity to meet him even before classes started for my Master's program. Ever since my first meeting, he has always inspired me to conduct research studies. Due to his inspiration and guidance I decided to conduct a Thesis project. He was kind enough take me in his research group and encouraged me to work on a new research topic. Without his support I could not have overcome the numerous challenges that I faced during the course of my Masters program. Thank you, Prof. Pasricha for providing such a memorable journey in the past two years.

Secondly I would like to thank my committee members Prof. Charles Anderson and Prof. Sourajeet Roy for their valuable time and input. I was fortunate to study under Prof. Anderson for the Machine Learning course. Prof. Anderson's teaching and advice allowed me to understand and utilize the machine learning algorithms in this work. My sincere gratitude goes to Prof. Sourajeet Roy for agreeing to be on my thesis committee.I would like to thank Kristi Buffington for providing me with the Indoor Maps of various buildings in CSU. She was very cooperative in providing me with the resources as and when needed which led to a precise accuracy study in my project. I would also like to thank my colleagues in Multicore Embedded

Systems (MECS) lab for their advice and unwavering support. Their willingness to listen to my presentations, provide peer reviews, and offer helpful suggestions which significantly improved my work.

Special thanks to my spiritual master Baba Hardev Singh Ji for his divine blessings and love. Last, but not least, I would like to thank my family and friends. I would like to thank my family in India the Ugaves, especially my parents, my niece, my brother and sister in law for the their support and faith in me. My grandparents, the Kadams who have always stood behind me. I want to thank all my friends and colleagues Saket Doshi, Swaroop Sahoo, Sahil Mehta, Harshad Kulkarni, Anmol Shahani, Nikhil Patil and special thanks to Ritika Thohan for her continued support and love.

DEDICATION

*To my parents, Anand and Madhuri*

*Without their support, understanding, encouragement, and love this work would not have been possible.*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1

# Introduction

With the proliferation of smartphone devices, mobile computing is at its peak. It has led to a new generation of services and applications. Context aware applications simplify the lives of people and involve social engagement. Service providers have set up infrastructure *(4G LTE for fast internet and GPS satellites for navigation)* to support these vivid applications and location based services *(LBS)*. Smartphone manufacturers have also integrated a myriad of sensors and circuitry that allow for development of such applications. Mobile location platforms have enabled location-based services for public safety, national security and commercial services. After years of research on precise mapping and outdoor navigation, people are now interested in Indoor Maps. Regulatory bodies like the Federal Communications Commission (FCC) in their Enhanced 911 system mandates for location of mobile emergency calls. Conceivably they are even considering updated mandates to introduce accuracy requirements for emergency calls placed indoors [1]. Indoor location technology provides a valuable benefit to the consumer whether for convenience, entertainment or utility. A few examples of common uses for Indoor Location are locating people, and places, coordinating joint activities, augmented reality gaming, monitoring and tracking pertinent information.

## 1.1 Motivation

Since the inception of smartphones GPS has been one of the key sensors that the smartphones were shipped with. This suggest the interest of companies and consumers at large in providing and knowing the location of the user. Governments and corporations have made huge investments in creating infrastructure for map based location services [2]. Global Navigation Satellite System (GNSS) Asia reports that over the coming decade the installed base of GNSS

devices will increase almost four-fold. It is expected that the number of GNSS devices will increase in Europe and North America from 1 to 3 per inhabitant over the coming decade [3]. Smartphones are going to dominate the global GNSS revenues and are also expanding into other market segments. As seen in Figure 1.1 almost half of the revenue from GNSS services for 2012-2022 is projected to come from from location-based services (LBS) on smartphone devices.

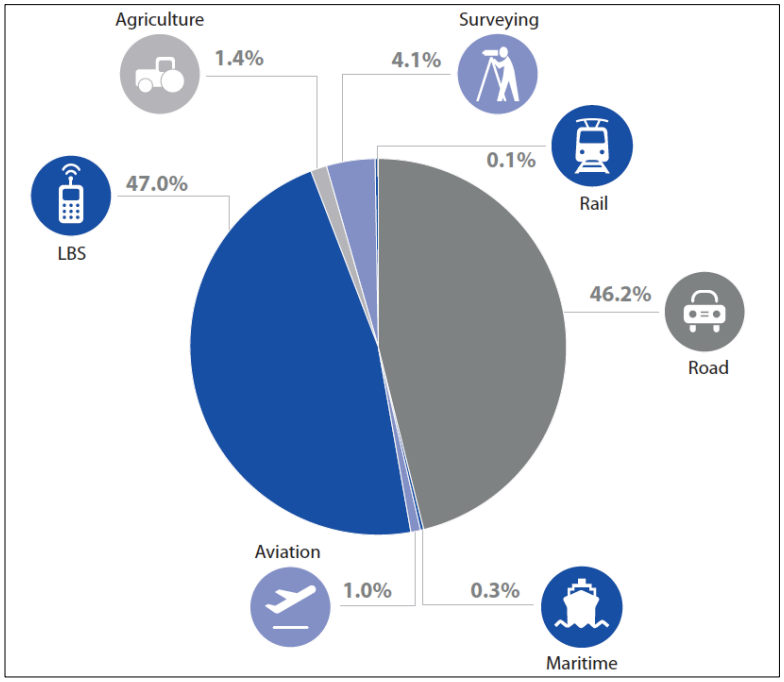


**Figure 1.1 Cumulative Core GNSS Revenue for 2012-2022 [3]**

Furthermore, Strategy Analytics [4] reports that people spent 80-90% of their time indoors; 70% of the cellular calls originate from indoors and 80% of data connections are made inside. Research conducted by yellow pages graphically shows, the closer the users are to a business, the more likely they are to click on a mobile banner ad for that business [5] as shown in Figure 1.2. Unfortunately, the current infrastructure for navigation services like GPS and GLONASS cannot be used indoors as their weak signals are critically compromised by obscuration and environmental degradation (e.g., signals cannot pass through concrete and other solid obstacles

within the building structures) [6]. The majority of the current Indoor Localization techniques primarily depend on new and non-existent infrastructure [6]. Wi-Fi, which was originally not intended for the purpose of navigation, has been successfully used in some techniques as apositioning technology indoors. The success rate for Wi-Fi is high primarily because of abundance of Wi-Fi access points at major residential and commercial places [6]. It is easy for Wi-Fi signals to penetrate through walls and major building structures. Therefore, the Wi-Fi data in conjunction with smartphone sensors can be used for the benefit of Indoor Localization.

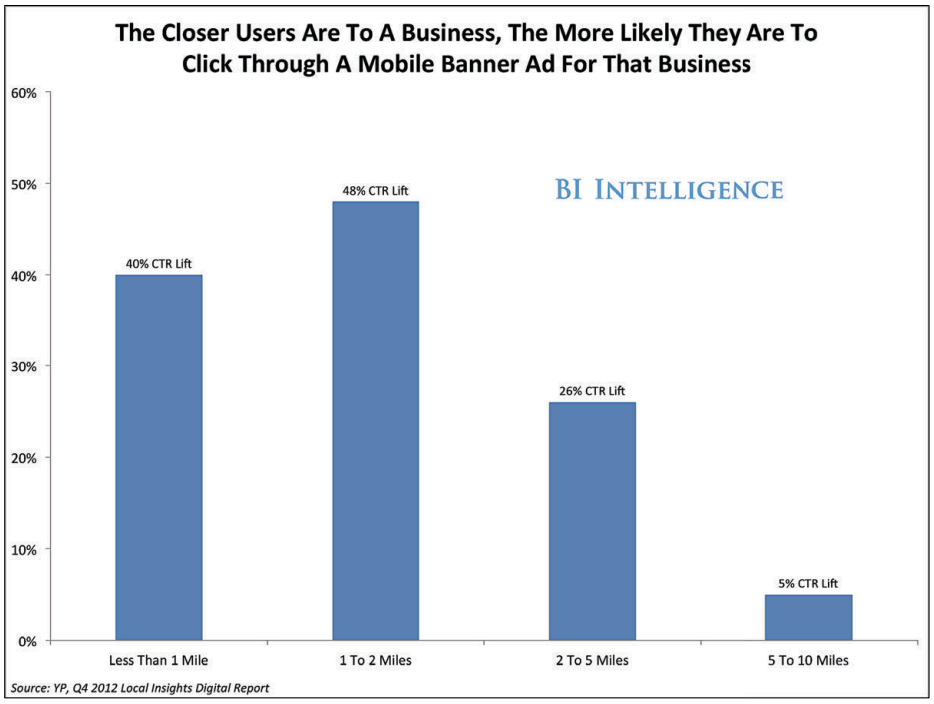


**Figure 1.2 Click through rate (CRT) proximity to a business destination [5]**

One major limitation of techniques, that make use of Wi-Fi data with smartphone sensors for indoor localization is that they are energy inefficient, i.e., they can drain the phone battery quickly. This is because the algorithms used in these techniques are computationally intensive and use the smartphone resources *(sensors and Wi-Fi modem),* which require high power. It is necessary to use "smart" strategies to optimize and reduce energy consumption. Hence having an

accurate and optimized Indoor Localization technique that consumes minimal energy is the need of the hour.

### 1.2 Contributions

This thesis presents five different techniques for Indoor Localization with an accuracy and energy study.

The first two techniques are sensor-based. The first technique is also known the ***classic technique***. Classic technique utilizes only the accelerometer and magnetometer sensor. These sensors are used for tracing the paths of the user, which are also known as dead reckoning. The second technique called ***sensor fusion*** is a patented technique that uses the accelerometer, magnetometer and gyroscope sensors for dead reckoning. Sensor fusion uses Kalman filters that combine the value from the three sensors to give the final accurate values.

The other three techniques are machine learning based techniques that use the three position sensors *(accelerometer, magnetometer, and gyroscope)* and the Wi-Fi access points. The three machine learning techniques are combined in a single platform called ***LearnLoc***. In *LearnLoc* data from the position sensors are fused using the sensor fusion algorithm and a Wi-Fi fingerprint of the area is collected. The Wi-Fi signal strength fingerprints are used by the machine-learning algorithms to accurately predict the user's location. We use three different machine-learning techniques – ***Linear Regression, Neural Networks*** and ***K Nearest Neighbors***.

We quantify and present the energy consumption for each technique. An accuracy study is also presented for all our techniques. An energy and accuracy trade-off study has been done to find out what is the optimum Wi-Fi scan rate for accurate Indoor Localization with the least energy consumption.

### 1.3 Outline

The rest of the thesis is organized as follows:

- Chapter 2 provides an overview of contemporary smartphone platforms and location based sensors on the smartphones.
- Chapter 3 describes the problem statement for the thesis.
- Chapter 4 reviews the current work related to Indoor Localization on smartphones.
- Chapter 5 provides a detailed overview of the machine learning techniques used in the thesis.
- Chapter 6 presents the Indoor Localization strategy using sensors and machine learning techniques.
- Chapter 7 presents the experiments and results.
- Chapter 8 concludes the thesis with a summary and future work.
- The appendix offers the source code of the strategy presented.

Chapter 2

# An Overview of Contemporary Smartphone Platforms, Positioning System and Sensors

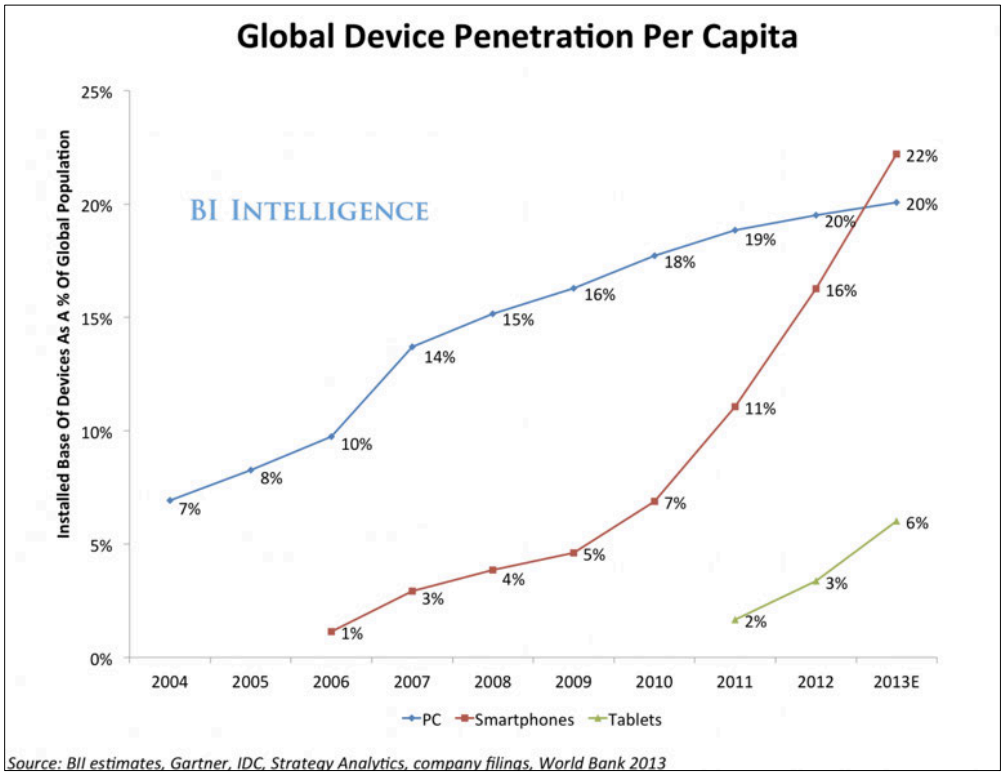


**Figure 2.1: Global Device Penetration Per Capita as of Oct. 2013 [7]**

*Smartphones* are mobile embedded devices with advanced capabilities beyond ordinary mobile phones. Smartphones have gone from becoming a luxury to becoming a necessity these days. As seen in Figure 2.1 Business Insider reports that 22% of the global population owns a smartphone as compared to 1% eight years ago [7]. The number of smartphone users has exceeded the number of desktop computer users and this numbers continues to grow everyday. Google's Android has the highest share (52.2%) in the smartphone market followed by Apple's IOS (41.4%), Blackberry (2.7%) and Microsoft (3.3%) [8]. The computational capabilities of mobile devices are comparable to desktop PC's. The only limitations are the memory and battery power.

Smartphones are mobile, wireless and they provide the consumer with functionality to do almost everything a desktop PC does. Therefore, wired devices, which require a persistent electricity connection, are no longer a need for today's consumer. Additionally, the availability of high-speed Internet connections and cloud services have minimized the memory constraints for smartphones however the energy consumption still remains an issue.

Smartphones have seen a growing trend in processing capabilities but a decreasing trend in term of the battery life. The battery life of the old mobile phones was as high as 2 to 3 days, but the smartphone batteries today hardly last a day. All smartphones have some variant of the Li-ion battery [9], which contains a sealed bag of anode and cathode sheets with separators between them. A liquid electrolyte permeates all these layers. The smartphone circuitry is connected to the positive and negative terminals of the battery that powers up the device [9]. A smartphone battery may contain single or more Li-ion cells inside it. The electrolyte inside the cells can react with the residual atmosphere to form corrosive compounds. This reaction increases with high temperatures therefore it is necessary for the smartphone batteries to cool. There have been no breakthroughs in battery technology in recent years and the smartphone providers have tried to increase the cell count inside the batteries to provide higher battery power. But there have been limitations on the number of cells a smartphone can have due to the limited size and space inside the smartphones. Recent smartphones have a battery with capacity ranging between 1500-2500 mAh and the battery life lasts less than a day.

The dawn of 4G LTE cellular networks is upon us. Internet speeds are at an all-time high and all the latest smartphones come equipped with 4G LTE capabilities. 4G LTE speeds are ten times higher than 3G [10]. They can handle download speeds between 5 and 12 Mbps and upload speeds between 2 and 5 Mbps. The 4G LTE coverage is no longer limited to big cities and has

been expanded to small towns and remote places. Mobile applications today require higher bandwidths to meet consumer needs for high definition video streaming and real time data sync.

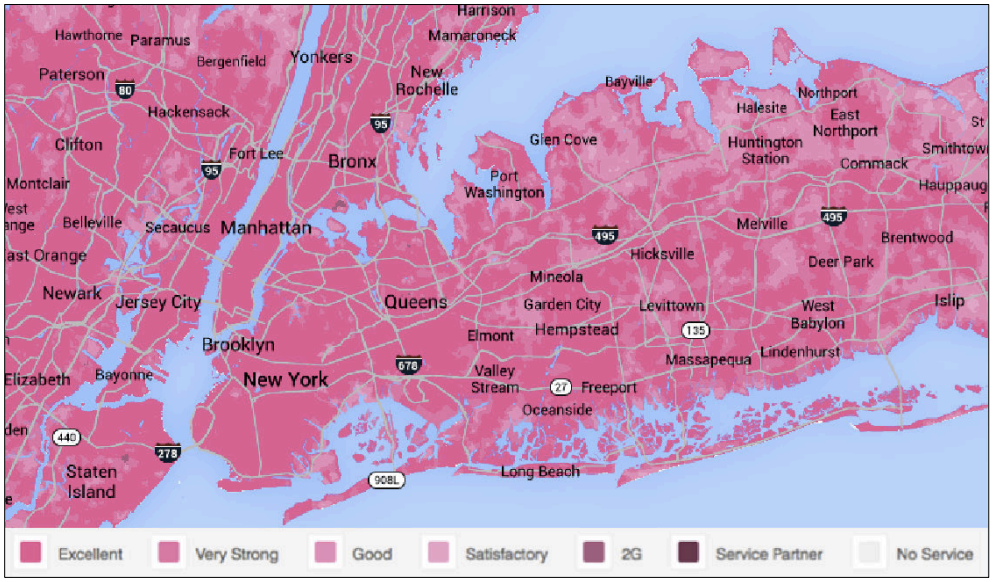


**Figure 2.2: Coverage Map for T-Mobile's 4G-LTE Service in NY [11]**

Apart from cellular network technologies, smartphones contain other data and location interfaces such as Wi-Fi and GPS. Wi-Fi is one of the most popular data interface due to high availability and bandwidth and therefore is also very popular among consumers. Figure 2.2 shows the 4G LTE coverage map for T-Mobile cellular service provider [11]. When Figure 2.2, coverage map for T-mobile's 4G-LTE service in New York City is compared with Figure 2.3 [12], Hotspot location map for Boingo Wireless provider's Wi-Fi hotspots in the New York City it is evident that the Wi-Fi coverage is as good as the 4G LTE coverage.

Smartphone CPU architecture is in its matured stages and is continuously advancing. Like the desktops, smartphones are moving from 32 to 64 bit architectures [13]. ARM is the leader in creating high-speed smartphone CPU's [14]. The majority of the Smartphone SOC's like Qualcomm, Samsung, Texas Instrument, etc. use Arm CPU's. The Samsung Galaxy S5 which is

the current state of the art Android device houses a quad core Qualcomm Snapdragon chip that runs at a frequency of 2.5 GHz [15]. The latest flagship device from Apple, the iPhone 5s also has the fast running A7 processor with a 64-bit processor [16].

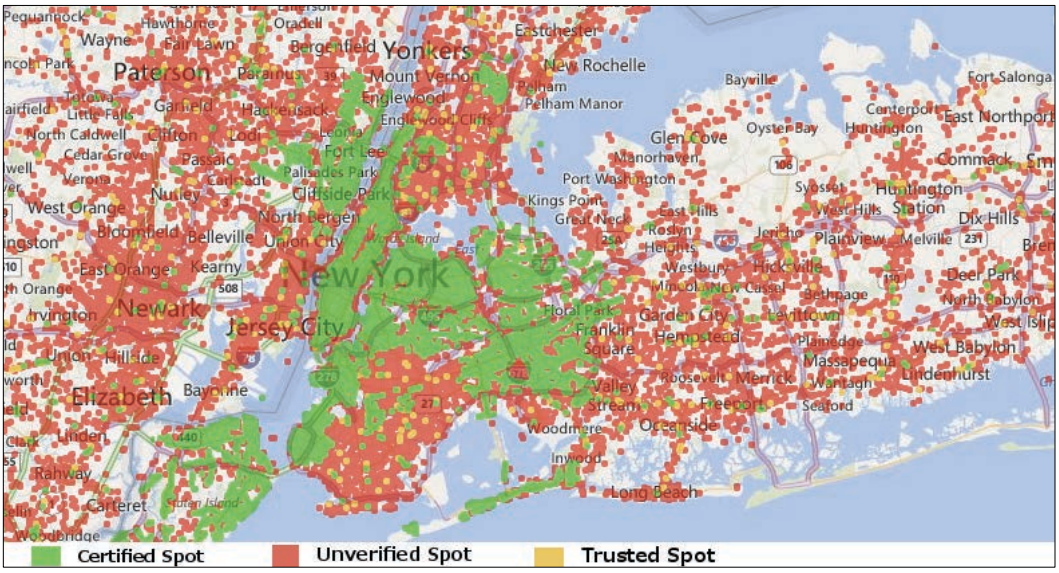


**Figure 2.3: Hotspot Location Map for Boingo Wireless's Wi-Fi Service in NY [12]**

## 2.1 Google Nexus 4 Positioning Sensors and System

To comprehend and understand positioning on smartphone we give an example of the Nexus 4 smartphone. Nexus 4 comes equipped with a Qualcomm quad-core Krait processor that runs at a frequency of 1.5 GHz and a 2 GB memory RAM. It is provisioned with Qualcomm's GNSS module that supports global positioning using GPS and GLONASS satlellites. These modules are used for global positioning and are useful for navigation outdoors, hence we do not talk about these modules in this thesis. In this section we talk in detail about the positioning sensors available on the smartphones for indoor navigation. We also investigate the Wi-Fi sensor as a potential positioning sensor indoors. It is first important to understand how positioning systems works, therefore positioning systems are explained in Section 2.1.1.

### 2.1.1 Positioning Systems

A positioning system tries to determine the location of an object in space. A positioning system can locate an object with varying accuracies. One such positioning system is the coordinate system with 2 or 3 dimensions. There are different types of the coordinate system like the Cartesian coordinate system, Polar coordinate system and Speherical coordinate system. In the three dimensional ***Cartesian coordinate system*** [17] the axes are determined as *X, Y* and *Z* coordinate axis. A point *P* in this system has *X, Y* and *Z* coordinates represented as P=(X, Y, Z). The point *P* can also be represented as a vector from the origin (0,0,0) to the point *P*. The coordinate system can be relative to a "*frame of reference*". A frame of reference is a small region of space where the coordinate system is expected to function. The coordinates can be relative to a world, local or body frame. We explain all the types of frames we use in our research below.

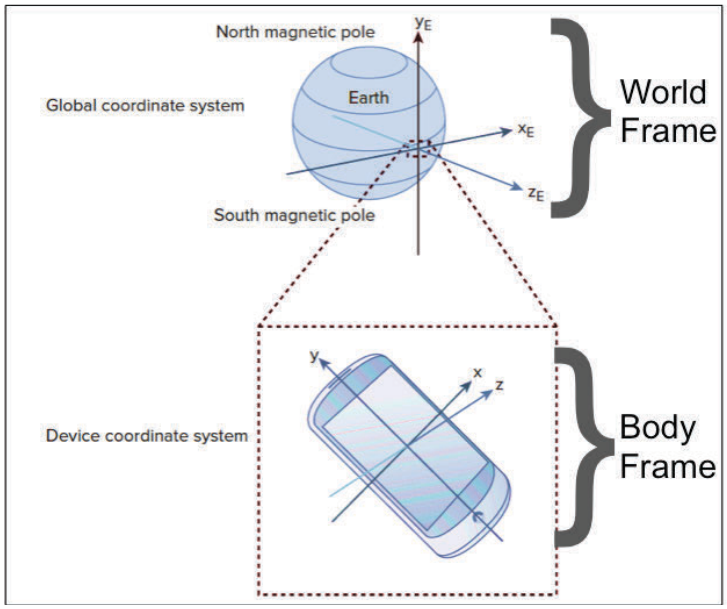


**Figure 2.4: Android Coordinate System**

### a) World Frame and Global Coordinate System

In a world frame all the sensors and methods refer to an absolute orientation with respect to earth using the global coordinate system. In Figure 2.4 the globe signifies the world frame. The coordinate system with reference to the world frame is known as global coordinate system. In the global coordinate system the axes are directed as follows:

- $Y_E$ points towards magnetic north, which is the true north.
- $X_E$ is 90 degrees from $Y_E$ and is parallel to Earth's surface pointing east.
- $Z_E$ points away from the center of the earth.

### b) Body frame and Device Coordinate System

The Android positioning sensors (*i.e., the accelerometer, magnetometer and gyroscope)* report values corresponding to the device or the smartphone body. Hence the frame is known as the body frame and the coordinate system with reference to the body frame as the device coordinate system. A smartphone device can have two different orientations, the default being the portrait orientation and the other being the landscape orientation.

The smartphone in Figure 2.4 signifies the device coordinate system. For a default landscape orientation the axes for the device coordinate system are directed as follows:

- The *X-axis* horizontal with positive values in the right direction.
- The *Y-axis* is vertical with positive values in the upward.
- The *Z-axis* is positive values in front of the screen.

In device coordinate system the coordinates are fixed to the device. The device coordinate system does not change when the orientation of the device changes. The origin of the device

coordinate system is at the center of the screen. Angular quantities around the axes are given by a 3-vector rotation matrix or a quaternion that maps the device coordinate system to the global coordinate system as shown in Figure 2.4 [18].

### c) Local frame or Relative coordinate system

For Indoor Navigation systems the world frame is very large and so we use something that is local to a small area. Such a coordinate system with respect to the local frame is called the relative coordinate system where the origin is fixed at the start and all other positions are relative to this origin. Usually the *X* and *Y-axis* are used to define the position on the map and the *Z-axis* is used to define the altitude above the ground. This system is mainly used to define positions inside a building or a map. In such systems the map is first rotated to match the true North in the World frame and then axis of the building is aligned to it.


**Figure 2.5: Local Frame**

### 2.1.2 Position Sensors

Starting with Android 1.5 as a standard set of sensors and associated *Sensors API* [19] has been made available. The standard sensors now include the accelerometer, gyroscope, magnetometer, light sensor, proximity sensor, humidity and pressure sensor. These raw sensors are Micro-electromechanical Sensors (MEMS), which are made on a tiny scale on silicon chips. They have the ability to detect, capture and analyze motion and normally contain a part that physically moves or vibrates. The main position sensors are the accelerometer, gyroscope and magnetometer. Figure 2.6 shows a MEMS Gyroscope and Accelerometer sensor on the Nexus 4 motherboard [20]. These MEMS sensors are factory tested and trimmed so that no additional sensor calibration is required.

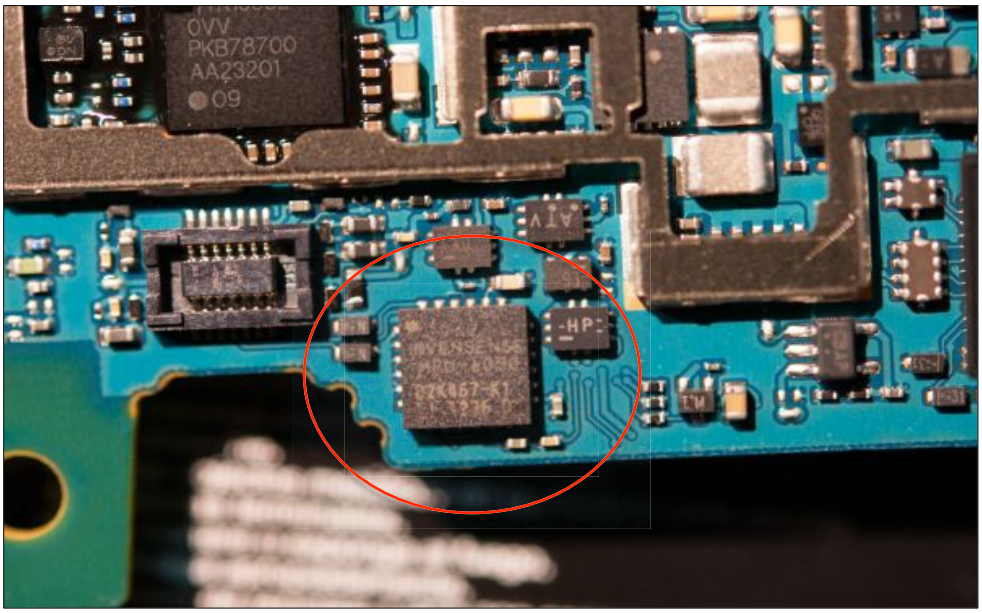


**Figure 2.6: Nexus 4 MEMS Gyroscope and Accelerometer**

#### a) Accelerometer

MEMS accelerometers are tiny masses of springs that can measure the Earth's gravity, which is 1g downwards (*g* is a unit of acceleration and is equal to 9.8 $m/s^2$), sense speeding up or slowing

down in a straight line. Acceleration is measured by attaching a mass to spring and observing how far the mass moves from the equilibrium position. Figure 2.7 A corresponds to the device sitting on table. Figure 2.7 B corresponds to the device moving towards the right and Figure 2.7 C corresponds to the device being dropped and is in a free fall motion [18]. To find the actual acceleration of the device the force due to gravity needs to be factored out. The velocity of the device can be found out by integrating the accelerometer value. The position of the device can also be found by double integrating the acceleration value. Normally in smartphones the accelerometer and gyroscope are embedded in a single MEMS die.

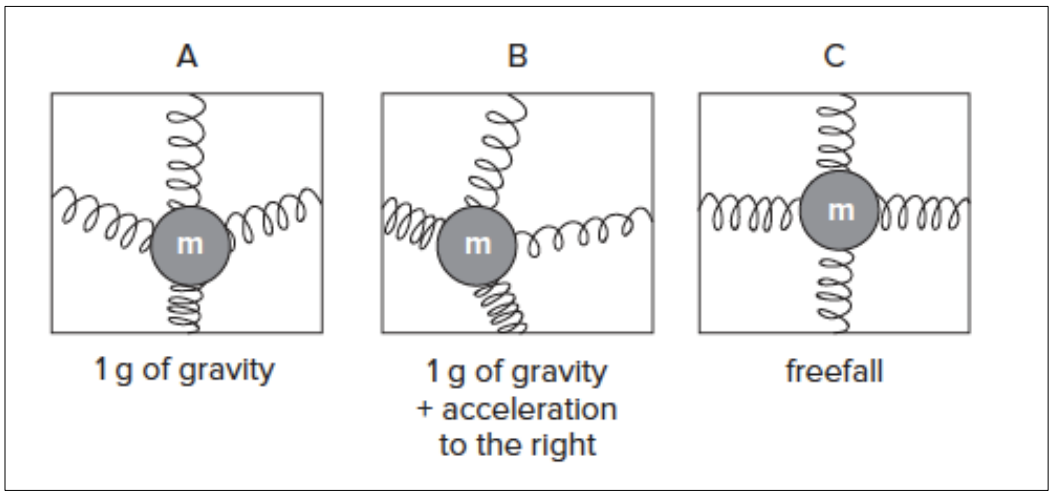


**Figure 2.7: Accelerometer Principle**

### b) Magnetometer

Magnetometers measure the strength and the direction of a magnetic field. They operate on different principles based on the manufacturer and architecture. *Hall effect* magnetometers are the most common [22]. In the Hall effect a magnetic field component that is perpendicular to the wire causes electrons to have a higher density on one side of the wire compared to the other, which results in the voltage across the width of the wire that is proportional to the magnetic field.

For location purposes, the accelerometer and magnetometer data is combined to determine the angle to which the user is pointing with respect to the north, also known as the heading angle. The accelerometer and magnetometer values are combined to get a rotation vector that determines the rotations as shown in Figure 2.8. *Azimut* is the rotation along the *Z-axis*, *pitch* is the long *X-axis* and *roll* around is the *Y-axis*.

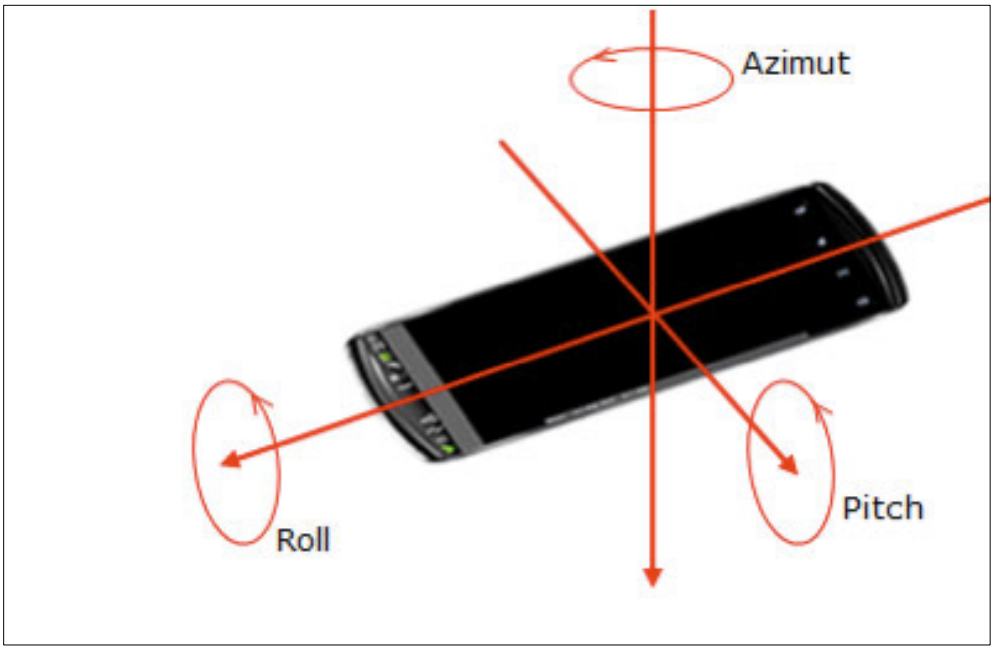


**Figure 2.8: Android Device Orientation**

### c) Gyroscope

The gyroscope provides the angular velocity (*i.e., how fast something is spinning)* about all three axes. Unlike the accelerometer it is not affected by gravity and is less susceptible to magnetic influences compared to the magnetometer and accelerometer. Therefore it's more accurate than the accelerometer and magnetometer and also has a very short response time. To get the device orientation, the angular speed from the gyroscope is multiplied with the time interval between the current and last sensor output. In MEMS technology a vibrating structure gyroscope is

usually used which works on the principal that vibrating objects continue to vibrate in the same plane as its support rotates. A MEMS structure die of digital gyroscope is shown in Figure 2.9.

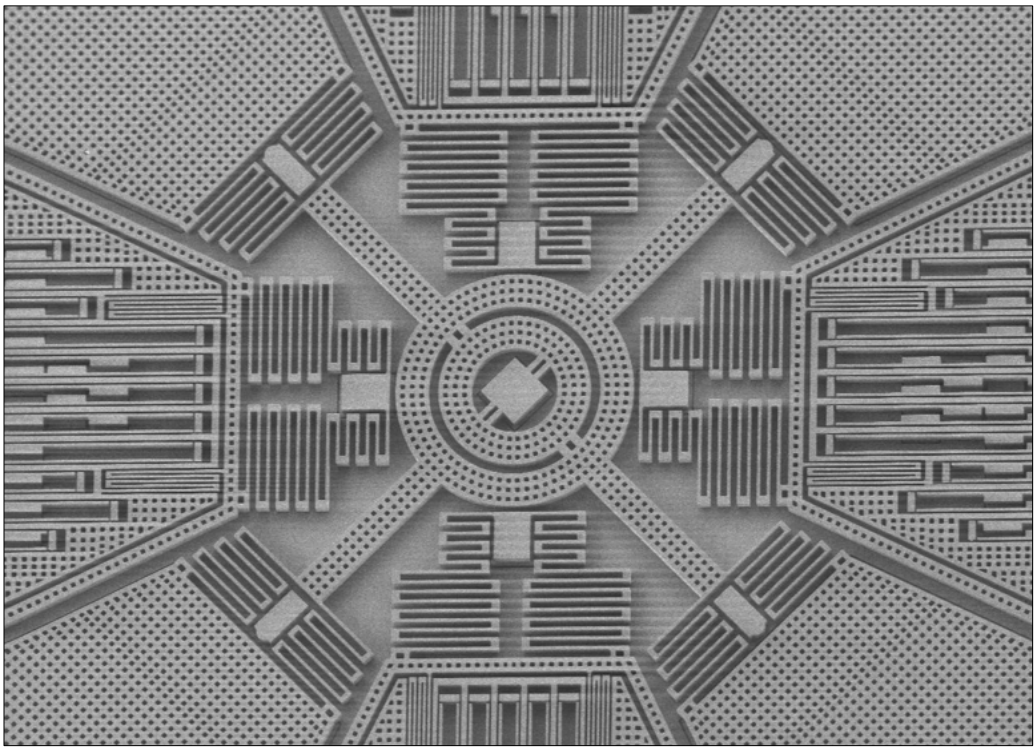

**Figure 2.9: MEMS Structure die of digital gyroscope**

**d) Wi-Fi**

Wi-Fi, which is an alternate data module for smartphones, supports high-speed data transfers. As mentioned in Section 2, Wi-Fi systems are ubiquitous. Hence Indoor Positioning techniques based on patterns of observations associated with multiple Wi-Fi hotspots are now a possibility. There are two ways in which this can be done, fingerprinting or trilateration. *Fingerprinting* is where observations are compared to previously mapped locations and *trilateration* is where received power is used as an indicator of distance from the transmitter and a geometric calculation against known transmitter location is used to locate the device. Nexus 4 uses Murata SS2908001 802.11 Wi-Fi module as shown in Figure 2.10 [22].

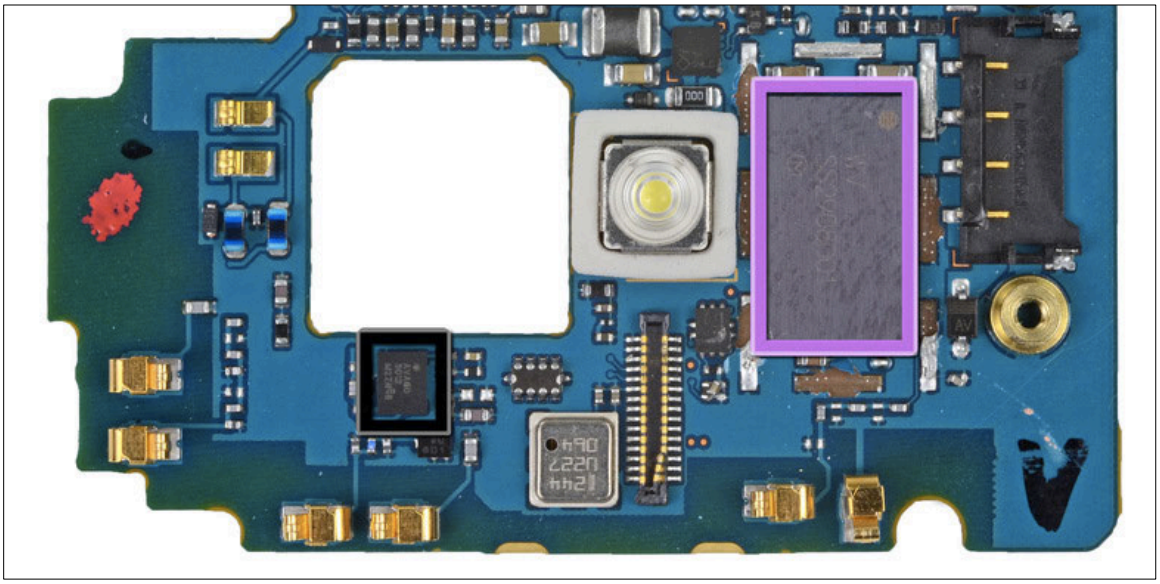

**Figure 2.10: Nexus 4 Wi-Fi module [22]**

# Chapter 3

## Problem Statement

The previous chapters put forth some key issues in smartphone and positioning technology. The most important issue impacting the smartphone technology is the need to reduce the energy consumption. Smartphones are shipped with a variety of sensors, wireless modules and high speed CPU's, but their utility has and will remain severely limited due to the battery life. Therefore it is critical to manage the resources in mobile systems. Different sensors have different behaviors and they operate differently under different situations. Some sensors may require frequent and detailed attention from the processor – such as the gyroscope sensor. The gyroscope sensor is repeatedly read to get the orientation data even though there is no movement detected. In such scenarios the processor is burdened with extra overhead which prevents it from entering the sleep mode. This results in an increase in system energy consumption. All of these concerns suggest the need for "smart" sensor management strategy, which minimizes the sensor calls without disrupting the *quality of service* (QoS).

Indoor Positioning technology is not yet effectively established, due to several concerns. At the crest of these concerns are the inaccuracies in the past localization techniques and their dependence on added or unavailable infrastructure. Outdoor navigation is now thoroughly established due to the infrastructure that was build over the years through government agencies and large corporations that have resulted in various GNSS based applications [2]. Having a similar infrastructure for indoor buildings is highly impossible. This would need cooperation of large communities and individuals to establish such framework indoors. But this would also mean a breach of privacy for different individuals. The need to come up with a solution using the current available infrastructure is essential. As discussed in the previous chapter, Wi-Fi signals

are accessible and pervasive. Almost every household today has a Wi-Fi modem to facilitate Internet connectivity in their residence. Commercial places and public buildings also host Wi-Fi services for the visitors and/or customers. Wi-Fi services are readily available and therefore patterns of data associated with multiple Wi-Fi hotspots can be used for Indoor Positioning.

There are a variety of algorithms and techniques available that can navigate a person indoors. But these techniques are inaccurate and drain the smartphone battery. Accurate techniques involve the use of compute intensive machine learning algorithms. The compute intensive tasks are responsible for the large power dissipation. The use of the Wi-Fi module like the positioning sensors consumes a lot of energy. Hence, a smart strategy is advised for optimizing Indoor Navigation techniques. The strategy should involve minimizing the frequency of calling the sensors and using the algorithms in such a way that brings down the computational costs.

The goal of this thesis is ***to create an accurate Indoor Navigation technique*** and ***to optimize mobile device energy consumption via smart sensor management strategies***. This thesis discusses multiple strategies. Two of the strategies are sensor only strategies that use the positioning sensors along with algorithms that combine the data from these sensors to navigate a person indoors. We also discuss our Indoor Localization platform *LearnLoc* involving three strategies that use the positioning sensors, Wi-Fi module and learning algorithms for accurate Indoor Navigation. We do a comparative energy study using all these techniques. This study shows how to manage the sensors and how to provide an accurate Indoor Navigation technique with minimal energy consumption.

# Chapter 4

# Related Work

A large amount of work has been done in the area of Indoor Localization for mobile devices in recent years. This work can be categorized into the following: (1) work on Indoor Navigation techniques, (2) work on usage of machine learning techniques on smartphones, (3) work on energy optimization on smartphones for location and data interfaces.

## 4.1 Work on Indoor Navigation Techniques

There has been substantial work in the area of Indoor Localization techniques [23]-[33]. *Dead Reckoning* based techniques as discussed in Chapter 2 are among the most widely used for indoor location sensing. The classic dead reckoning technique combines magnetometer and accelerometer readings while better techniques like sensor fusion combine magnetometer, accelerometer and gyroscope readings. Sensor fusion [25] based techniques have been shown to be less susceptible to magnetic drifts and give more stable readings. Both these techniques inevitably suffer from error accumulation over time. Some commercial providers such as IndoorAtlas [31] propose using magnetic fingerprints for creating indoor maps. But such systems do not work well in buildings that have metallic structures, which create magnetic interference. Some researchers have suggested using the fingerprint from received signal strength (RSS) for Indoor Localization [25], [26], [27]. Others have suggested techniques based on custom infrastructure support, such as ultra-wideband [28], ultrasound [29], and RFID [23]. But setting up this custom infrastructure is often impractical, expensive, and not easily scalable. To keep costs manageable, ambient fingerprinting using available infrastructure (Wi-Fi signal strength, light, sound, etc.) is a more viable option. *SurroundSense* [30] proposes using ambient fingerprinting with Wi-Fi triangulation and a Support Vector Machine (SVM) based

classification-learning approach for Indoor Localization. However, the approach is extremely time consuming and resource hungry, and is impractical for real time implementation on smartphones. Indeed, the authors in [30] leave real-time implementation of their localization methods as future work. Moreover, no energy analysis of the localization techniques is performed.

### 4.2 Work on Machine Learning Techniques on Smartphones

There has been extensive research conducted in the area of machine learning techniques. A lot of these techniques have been utilized in various smartphone applications. They have been unpopular among smartphones due to the long training phase and need of high processing power. But in spite of these issues, it has become inevitable that they will be used in smartphone applications. Researchers have investigated several classifying and regression based techniques like *Decision Trees*, *Naïve Bayes Classifiers*, *Neural Networks*, *Linear Regression*, etc. [34]. Cheung et al. [35] use Markov Decision process to prolong battery life by using a user defined reward function. In [36] the authors propose a model that predicts spatial context through supervised learning, and the authors in [37] take advantage of signal strength and signal quality history data and model user locations using an extreme machine learning algorithm.

### 4.3 Work on Energy Optimization for Data and Location Interfaces

Various papers have also talked of energy-based studies and energy models for quantifying and estimating energy consumption of apps through services and system calls on cell phones [38], [39]. Though these models maybe generalized, every application is unique and the same model might not be good for a particular application and so we created our own energy model and applied it in this paper. Some techniques used a power monitor to quantify the actual power consumption on the device [40]. We used a similar approach. There has been extensive research

in optimizing energy usage of wireless interfaces like 3G, Wi-Fi, Bluetooth, etc. Since Wi-Fi is primarily used for our navigation technique, we concentrate on robust energy modeling of the Wi-Fi interface. Other work [41]-[56] focuses on energy-efficient location-sensing schemes aiming to reduce high battery drain caused by location interfaces (e.g., WiFi, GPS). Lee et al. [47] propose a Variable Rate Logging (VRL) mechanism that disables location logging or reduces the GPS logging rate by detecting if the user is standing still or indoors. Most of this work on energy optimization is for outdoor navigation and none for indoor navigation.

# Chapter 5

# Machine Learning Techniques

Machine learning algorithms search for patterns and regularities in any given data and have found wide usage across various application domains. They automatically learn from data by generalizing from examples. As more data becomes available more ambitious problems can be tackled. These algorithms are typically implemented in two phases. In the first phase, called ***training phase***, data is gathered and provided to the algorithm, so it can learn patterns and create a model to classify data or predict data properties. In the second phase, called ***testing phase***, new data is tested against the model that was built during the training phase, and the effectiveness of the model is revealed. Such two-phase learning algorithms are called *supervised* learning algorithms [57]. There are also algorithms in which the testing phase is not used, and such algorithms are called *unsupervised* learning algorithms. These algorithms use unlabeled data to cluster the data in different classes. Machine learning algorithms can be used for classification or regression. In *classification,* the machine learning algorithm learns to classify the data in different classes while in *regression* it predicts a continuous variable by learning from the train data. To improve Indoor Localization prediction capabilities over prior work, we propose to integrate machine-learning techniques that intelligently make use of different modules in our Indoor Navigation techniques.

## 5.1 K-Nearest Neighbor

The K-Nearest Neighbor (KNN) algorithm is a simplistic non-parametric algorithm. Non-parametric methods assume that similar inputs have similar outputs. KNN can be used for both regression and classification. It is a supervised algorithm but has a minimal training phase. Most of the effort in this algorithm is expended in the testing phase. As all of the work for obtaining

the best prediction for the location of a user is done at every instance of the testing phase, KNN is an instance based learning algorithm. Our KNN regression implementation works as an extension of the KNN classification algorithm. In KNN classification, new samples are classified by assigning the class that is the most common among the *k* closest sample in the training set. In our KNN regression implementation, the output value is calculated as the average of the value of its *k* nearest neighbors. To determine the closest sample, some form of a distance function is required. We make use of the Euclidean distance (*D*) measure between any two points *a* and *b*, each containing *i* attributes. This measure is defined by the following equation:

$$\mathrm{D}\,(\mathrm{a},\mathrm{b}) \;=\; \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2} \qquad (5.1)$$

This can be illustrated by the following classification example: Figure 5.1 shows a data set, characterized as blue squares and red triangles. The green circle is a new sample that needs to be classified as either a blue square or as a red triangle. If $k = 3$ *(represented by the smaller inner circle with radius 3),* the new sample is classified as a red triangle because there are more red triangles within the considered area. Similarly, if $k = 5$ the new sample is classified as a blue square.

In the *training phase* of our KNN algorithm, only the training data is collected and rearranged according to the training data for the testing phase. This greatly reduces the time needed for the training phase but the time during the testing phase increases as a majority of the mathematical calculations in the algorithm are performed during the testing phase. These calculations typically require high memory and processing overhead. However, mobile devices such as smartphones have stringent energy and memory capacity constraints. Thus it was essential for us to optimize the KNN implementation for it to work on smartphones. We implemented approximation

techniques that constrain the search space to a small subset of nearest neighbors, when estimating user location. We explain this in Section 6.2.1.

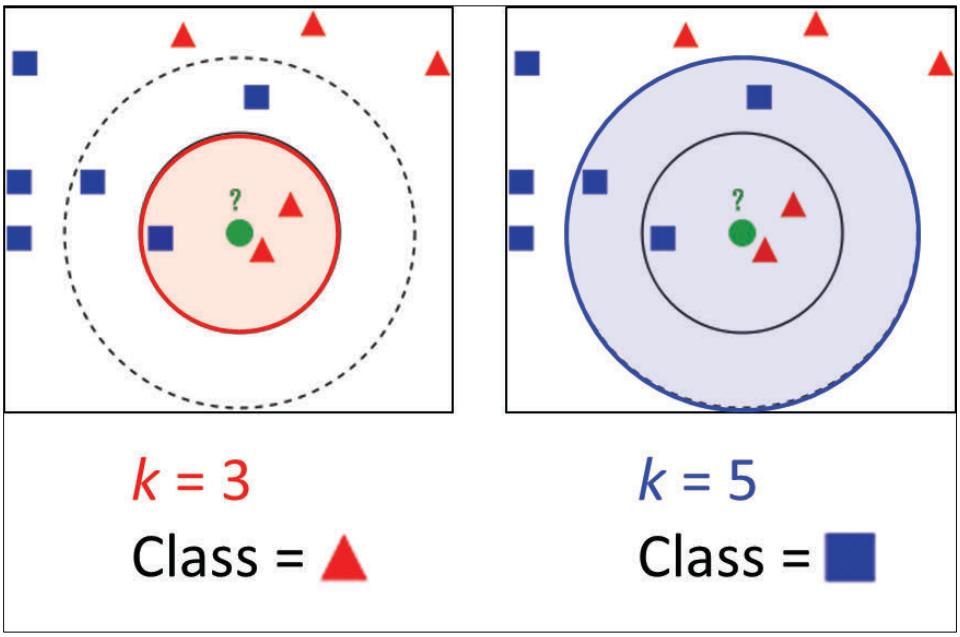


**Figure 5.1: K-nearest neighbor example**

## 5.2 Linear Regression

Linear regression models capture relationships between output dependent variables and input variables. These linear models are built during the training phase and are used to make predictions during the *testing phase*. The key assumption of this approach is that the output variable is a linear combination of certain weights and input variables. For non-linear relationships, these models would provide inaccurate predictions. However, efficient non-linear models are also harder to derive. We found that for our purposes of Indoor Localization estimation, linear models provided fairly good accuracy.

The creation of linear regression models requires fitting the input data, using one of several functions. We make use of the *least squares* approach, which involves a mathematical procedure

for finding the best fitting curve to a given set of points (training data) by minimizing the sum of the squares of the offsets or the residuals of the points from the curve [58], as expressed below:

$$y(x; w) = w_0 + \sum_{i=1}^{N} w_i x_i \tag{5.2}$$

$$w_{best} = \underbrace{argmin}_{w} \sum_{n=1}^{N} (t_n - y(x_n; w))^2 \tag{5.3}$$

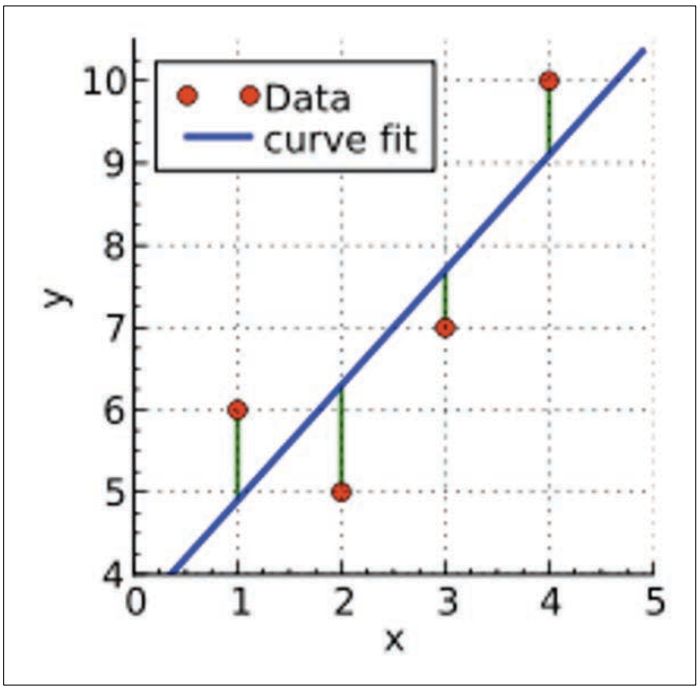


**Figure 5.2: Fitting curve using least squares approach**

Equation 5.2 shows the output values (*y*) as a function of the inputs (*x*) using the optimal weights (*w*) in our linear regression model. We determine the weight vectors by minimizing the error between the target values (*t*) and the output of the function *y* as shown in equation 5.3. Figure 5.2 shows fitting data using the linear least squares approach. The training phase for linear regression is time consuming and extremely compute intensive. Therefore, in *LearnLoc* this

*training phase* is performed on a server and not on the smartphone. However, we did utilize the smartphone in the *testing phase* to make predictions.

### 5.3 Neural Networks

Neural network models, also known as *Artificial Neural Networks* (*ANN*), are inspired by the way the central nervous systems in animals is believed to function. The brain which is the principle part of the central nervous system is capable of learning as well as recognizing patterns. The brain is capable of processing information from the sensory inputs, learning and storing in memory. Even though the brain is different from the digital computers today, it is believed that the basic concepts still apply. There is a computational unit, known as a *neuron*, and connections to memory stored in *synapses*. The biggest difference is that the brain is significantly bigger than any computer, with complex network of billions of neurons.

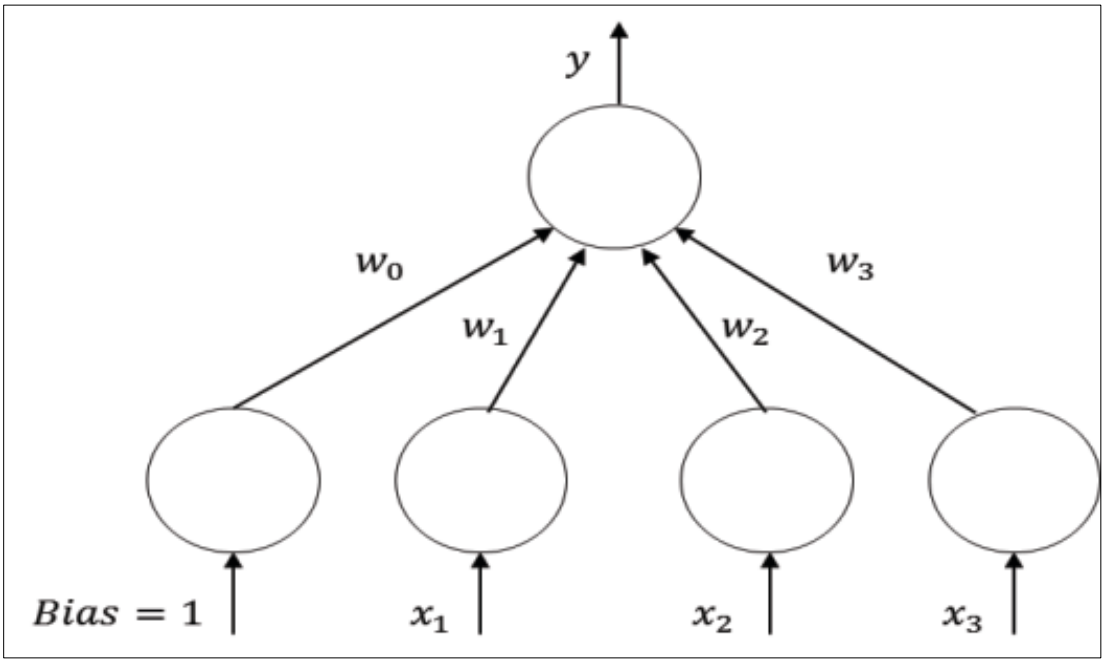


**Figure 5.3: Neural network perceptron model**

In machine learning ANN's try to model this relationship between neurons in the brain. Figure 5.3 shows an ANN perceptron model. The relation between the output *y* and inputs *x* is shown in equation :

$$y = \sum_{i=1}^{n} w_i x_i + w_0 \qquad (5.4)$$

The weight parameter *w* is determined in the training phase of ANN. ANN algorithm with back propagation can be divided in to the following four steps:

a) **Feed Forward computation:**

   This is a two-step process. In the first part the values of the hidden layer nodes are obtained. These values are then used in the second part to compute the values of the output layer.

b) **Back propagation from output layer:**

   In this part the errors are calculated at the output layer and then propagated back to the hidden layer.

c) **Back propagation from hidden layer:**

   In this part errors are propagated from the hidden layer to the input layer.

d) **Weight updates:**

   After a single iteration of forward pass and back propagation is done then the weights are updated.

The algorithm is stopped when the value of the error function has become sufficiently small. The last step of weight updates happens throughout the algorithm. In order to introduce non-linearity in the hidden layers of the Neural Network, activation functions are needed. The sigmoid function is one such activation function as shown in equation:

$$y_i' = sigmoid(y_i) = \frac{1}{1 + \exp(-\boldsymbol{w^T x})} \quad (5.5)$$

One of the biggest criticisms about the use of neural networks is the time required for training. Neural networks can be used for classification and are also known as non -linear logistic regression. We account for the training time in our approach. A neural network can have multiple hidden units, inputs and outputs.

# Chapter 6

## LearnLoc: Mobile Learning for Smart Indoor Localization

This chapter presents the LearnLoc framework that combines machine-learning techniques with smart fingerprinting using Wi-Fi and supplemental sensors to improve the accuracy and energy-efficiency of Indoor Localization. Our work makes the following key contributions:

- We propose the integration of inertial and Wi-Fi fingerprinting with three regression-based machine learning techniques that we have adapted and enhanced for Indoor Localization sensing;
- We implement these techniques on an actual smartphone and quantify their performance in real-time for Indoor Localization;
- We perform extensive benchmarking and testing of these techniques across multiple paths in diverse indoor building environments with different structural compositions;
- We compare the accuracy of these methods with state-of-the-art techniques from prior work for Indoor Navigation;
- Unlike any prior work, we also compare and contrast the energy consumption of our proposed Indoor Localization techniques with techniques from prior work, and explore trade-offs between energy and accuracy for Indoor Localization.

Our proposed *LearnLoc* framework shuns resource-hungry classification learning in favor of low overhead regression-based learning. This allows *LearnLoc* to be implemented and used in real-time on smartphones. Our framework also emphasizes energy-efficiency, unlike any of the prior works in the area. We show experimental results for energy costs of Indoor Localization

techniques as well as an analysis of trade-offs between energy-efficiency and accuracy of the techniques.

Section 6.1 describes the *LearnLoc* framework whereas Section 6.2 presents the enhancements using Machine Learning Algorithms and its implementations.

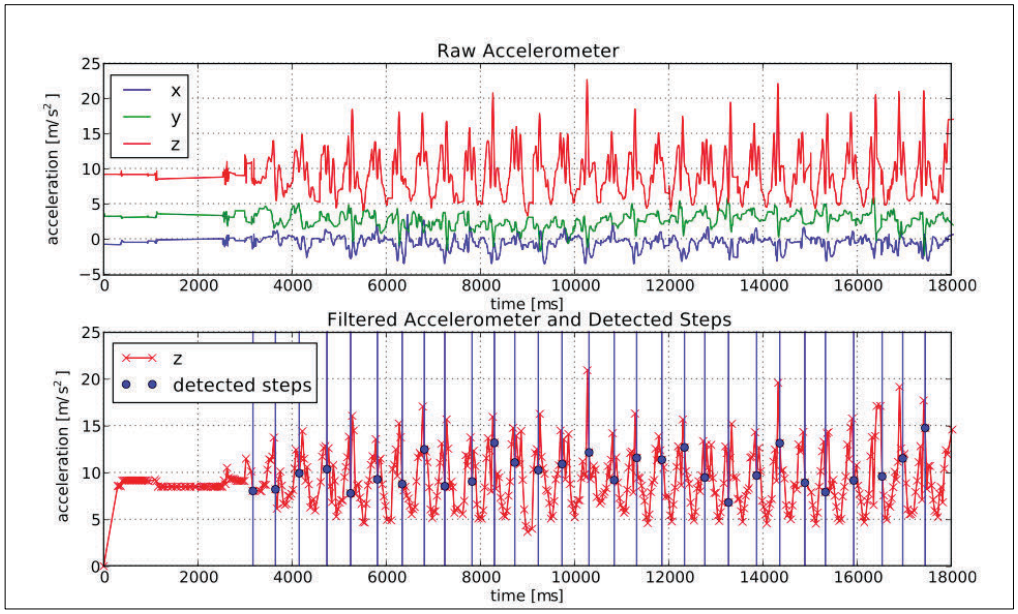


**Figure 6.1: Graph of Raw and Filtered Accelerometer values by Footpath Step Detector Algorithm [32]. The Filtered values are for the Z-axis accelerometer. The blue dots represent the points where a step was detected.**

## 6.1 LearnLoc Indoor Localization Framework

In this section we first discuss the three fundamental components of *LearnLoc* required for Indoor Localization. Subsequently, we present details of machine learning techniques that improve the performance of Indoor Localization in *LearnLoc*. LearnLoc is a platform that hosts multiple Indoor Navigation techniques with smart sensor management to reduce energy consumption. The principal components of *LearnLoc* involve Step Detection and Inertial Navigation for Pedometric dead reckoning. And lastly Wi-Fi fingerprinting that is done in the initial phase for the machine learning based techniques.

### 6.1.1 Step Detection

Step detection is a vital component of our Indoor Localization system. We use the same technique as Bitsch et al. [32] for step detection, where only the accelerometer *Z-axis* value is needed to detect a step. The accelerometer values display a regular pattern and a step is detected when there is a sharp drop in acceleration due to the jiggling of a phone in the hand or pocket. Figure 6.1 above shows the raw accelerometer data and then shows the filtered data of the *Z-axis* accelerometer and the detected steps.

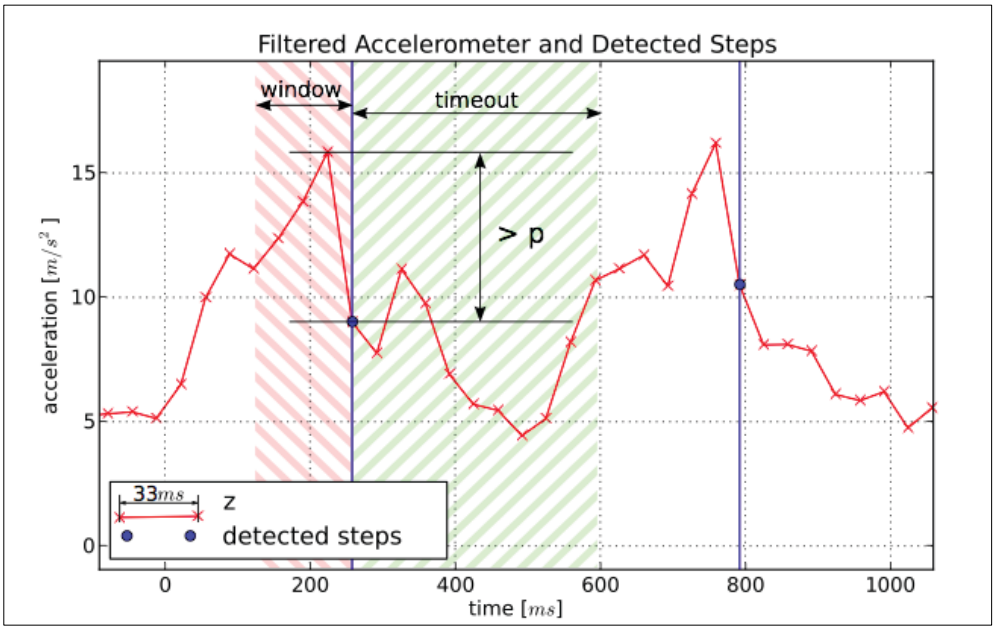


**Figure 6.2: Step Detection Algorithm [32]**

In our implementation of the step detection algorithm, the sensor data is sampled at the fastest possible rate (1 ms). The step detection algorithm is shown in Figure 6.2. A low pass filter is applied to this data to include major movements and improve detection. A step is detected when the difference between consecutive z-axis acceleration values ($\Delta p$) changes by 2 m/s$^2$. The difference is checked for a window of five consecutive readings (~166 ms) and after each detected step a timeout of 333 ms is used to avoid false detection. These values are adapted from

*Footpath* [32]. As mentioned above, steps are detected by the characteristic change of acceleration when the user moves forward due to up and down motion of the device in the hand. Experiments have suggested that the amounts of movements depend on the user and how they concentrate on holding the device still. So we make provisions for *sensor calibration* in which the user can set the step detection algorithm parameters that best suit them. The sensor calibration activity is shown in Figure 6.3.


**Figure 6.3: Step Calibration Activity**

The calibration process starts with the user walking around and tapping the graph area each time a step is done. The database logs the timestamps of the steps and all sensor values received for those time stamps. The calibration activity then varies the parameter values. First, it tries to vary the difference threshold p between the consecutive Z-axis accelerometer values $\{p \ \varepsilon \ R \mid 0.2<p<3; \ p_{n+1}=p_n+0.1\}$. Second, it varies the smoothening factor l for the low pass filter $\{l \ \varepsilon \ R \mid 0.05<l<0.8; \ l_n+1=l_n+0.05\}$. The step calibrator then tries to find steps by varying the parameters

and obtains their timestamps $t_d$. These timestamps are then matched against time stamps $t_t$, which the user provided earlier by tapping on the graph area. The user taps the screen when they think they have taken a step, which is a short time. But our algorithm searches for negative peaks in acceleration, which involves up and down motion that takes some time. Both time stamps $t_t$ and $t_d$ cannot be the same, therefore in our calibration activity we can set a tolerance window $w_d$ between which the times stamps will be compared. We match the timestamps for a tolerance window $w_d$ as in equation 6.1. The tolerance window can be changed by the user using the calibration activity.

$$|t_r - t_d| < \frac{w_d}{2} \tag{6.1}$$

So for every parameter *p* and *l*, a score is calculated to compare the performance of the values. We use a reward and punishment method to measure performance:

- One reward is added, if the timestamps $t_t$ and $t_d$ match, which also means a step is detected right by the step detection algorithm.
- If a user reported step is not detected by the step detection algorithm, then we punish by subtracting a reward.
- If the step detection algorithm detects a step but the user does not report one then we punish twice by subtracting two rewards.

For every parameter combination we calculate a percentage score *P*(*step*) which describes the success of these parameters as shown in equation 6.2. *c* is the number of steps detected right, *n* is the number of steps not detected by the step detection algorithm and f is the number of false steps detected by the step detection algorithm. The denominator *s* is the total step count.

$$\mathrm{P(step)} = \frac{c - n - 2f}{s} \tag{6.2}$$

The user can also set ones step size by calculating the distance between the two feet as shown in figure. We use a step size of 0.6 m

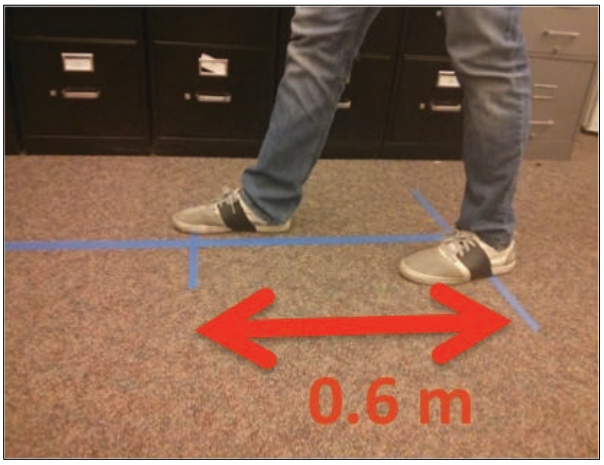


**Figure 6.4: Step Size**

### 6.1.2 Inertial Navigation

Inertial Navigation or *dead reckoning* is a popular technique for Indoor Localization which can be accomplished by combining readings from two or more inertial sensors. These systems are based on Newton's Laws of Physics and determine the position of an object with the knowledge of the original position and forces applied to the object [33]. If the initial position is not known only the relative position to the origin can be determined. Due to this limitation inertial navigation systems solely cannot be used for localization and tracking systems. They can track device movement over time, but the initial position needs to be manually set or provided by another system. As long as the laws of physics apply, inertial navigation systems can almost be used anywhere and there is no need of an external source of information. Hence, they can be used both indoors and outdoors.

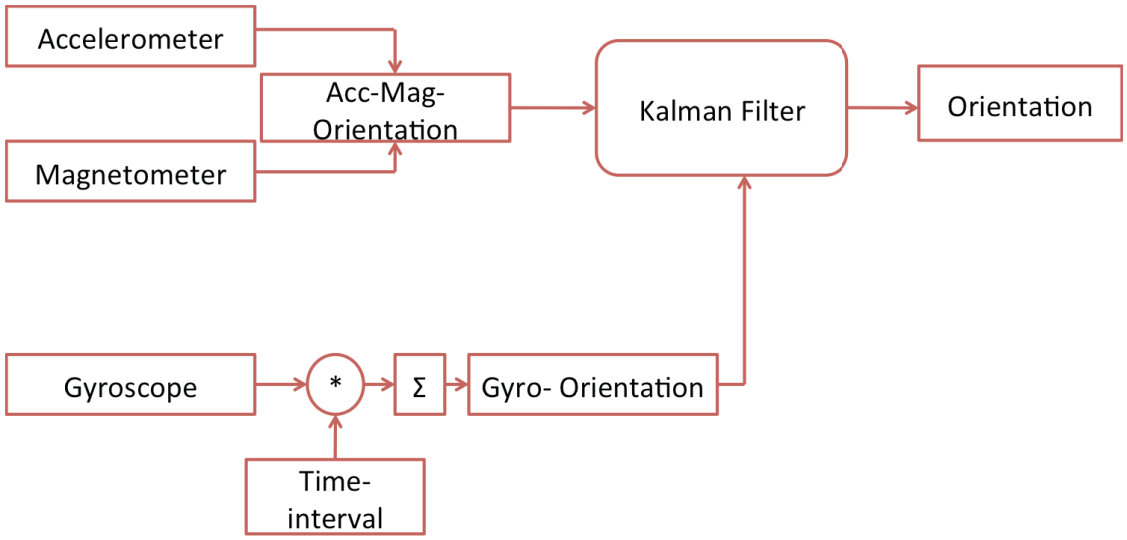


**Figure 6.5 Sensor Fusion Algorithm using Kalman Filtering**

Inertial Navigation is fundamental to *LearnLoc*, where we use it to determine the heading angle of a person and for step detection. The heading angle is the angle in which the user is facing with respect to the true North. We obtain the angle by combining the accelerometer, gyroscope, and magnetometer readings. These values are combined with the help of a technique called *sensor fusion*, as discussed in [25]. The accelerometer provides the gravity vector and the magnetometer works as a compass. The data from these sensors is combined to obtain a mobile device's orientation. But both of the sensor values often include inaccuracies due to noise. The gyroscope, which provides angular rotation speed, is used to determine the device orientation as it is far more accurate with a short response time. The angular rotation speed is first integrated over a time interval to determine the orientation of the mobile device. Then the sensor data from all three sensors is combined using Kalman filtering to give the precise orientation that avoids both gyro drift and noisy orientation as shown in Figure 6.5. The gyroscope output is applied orientation changes in short time intervals, while the magnetometer and accelerometer data is used as support information over long intervals of time. We use this resulting data to determine

the change in position from the current position.

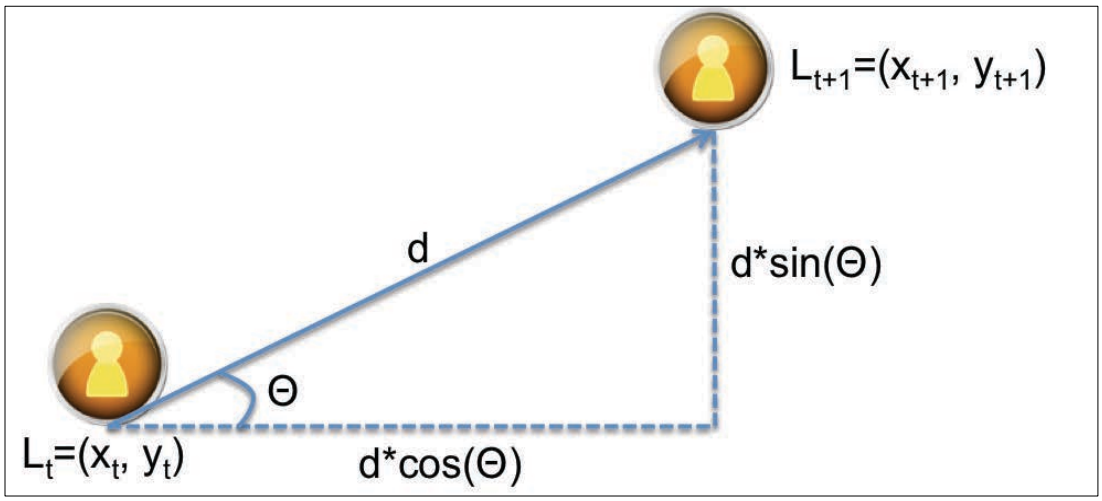


**Figure 6.6: Change in position calculation for inertial navigation**

Figure 6.6 shows how the new position is calculated. The angle ($\theta$) is the heading angle that is multiplied by a factor *d*, which is the step distance of the user that is set at the beginning for a user (with a default value of 0.5 meters). The step distance is determined by averaging the distance between the two feet for five consecutive steps. The equations to calculate the new position of $L_{t+1}$ ($x_{t+1}$,$y_{t+1}$) are:

$$x_{t+1} = x_t + dcos(\Theta) \quad (6.3)$$

$$y_{t+1} = y_t + dsin(\Theta) \quad (6.4)$$

### 6.1.3 Wi-Fi Fingerprinting

We use IEEE 802.11 wireless signal strength as an ambient location fingerprint for indoor environments. The MAC (Media Access Control) addresses of visible access points (AP's), the RSSI (Received Signal Strength Indication) value, and the location coordinates are stored in a tuple. This is a form of a passive, listen only wardriving technique where we do not communicate over the network (i.e., without any crowdsourcing). Fingerprints are collected

along the indoor path on the mobile device, by the user. This manual step is a prerequisite for all techniques that use Wi-Fi for indoor tracking.

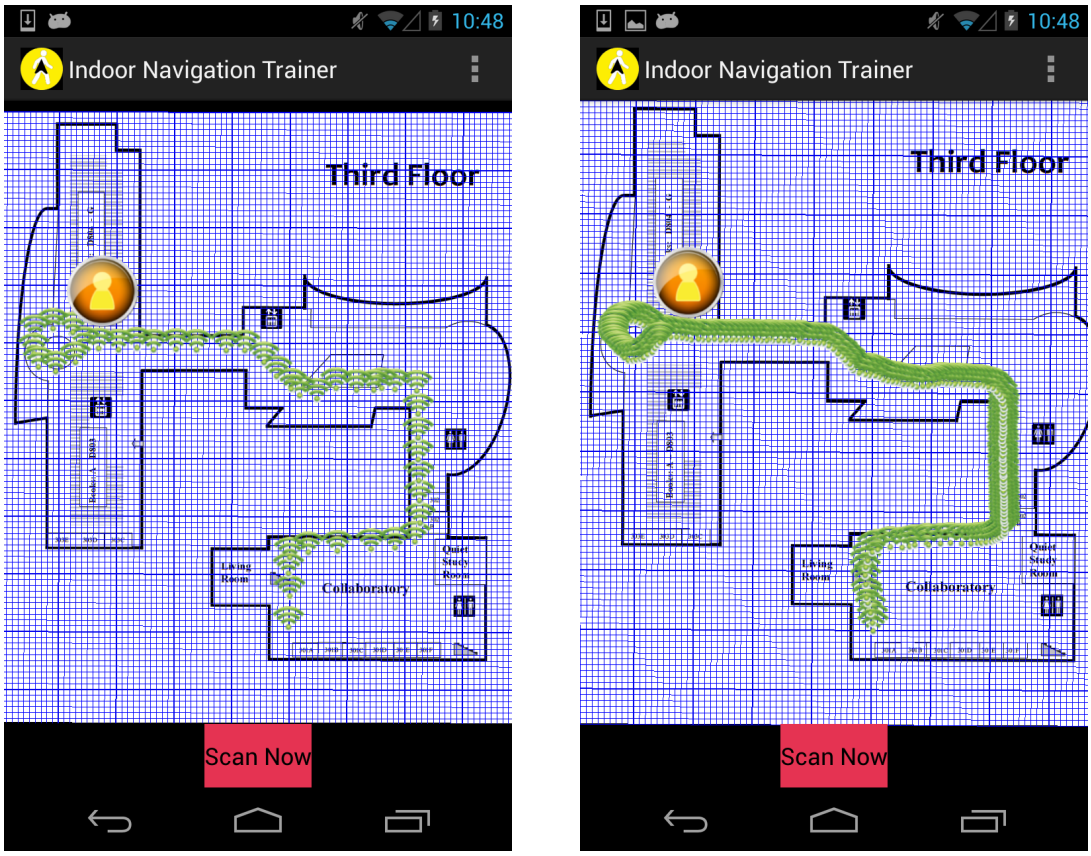


**Figure 6.7: Map showing Fingerprint data. Greend dots represent an instance of Wi-Fi scan along the path. Figure (a) on the left shows a Wi-Fi scan done every 3-4 meters. Figure (b) on the right shows oversampled data where a Wi-Fi scan is done every 0.5-1 meter**

The fingerprint details are logged into an *SQLite* database that is accessed by our machine learning algorithms (discussed in the next section). We found that a fingerprint gathered after every 3-4 meters on the path works well for the algorithms. Every point on a path typically has a large number of visible MAC addresses, which requires filtering out the addresses that are significant for tracking purposes. We filter out and select only those MAC addresses that are at least present at *j* unique locations on the fingerprint map.

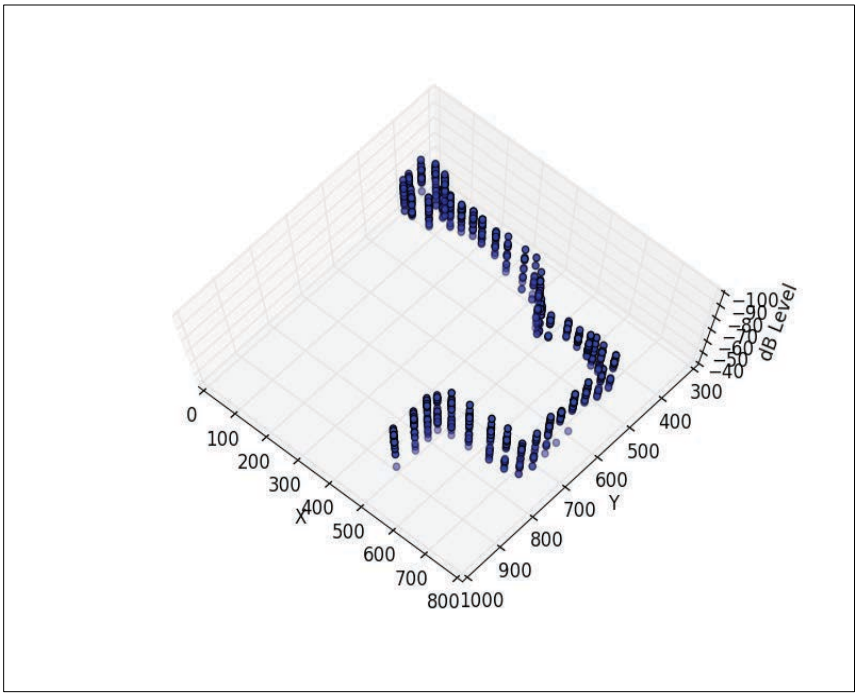


**Figure 6.8 WiFi level distributions over the map area shown in figure 6.7(a). The blue dots represent Wi-Fi signal strength level for all the MAC id's present on the path**

We experimented by trying to increase the density of fingerprints i.e. gathering fingerprints every 0.5 to 1 meter. This resulted in oversampling the data. The Figure 6.7 shows the map for oversampled data. The Wi-Fi signals vary less in neighboring areas in small locations compared to large areas. Hence it does not make sense to collect fingerprints at close locations. Figure 6.8 shows the Wi-Fi strength distribution over the area. This was done by collecting a fingerprint every 3-4 meters, resulting in a well spread out distribution of Wi-Fi signal strength for all the MAC id's over the area.

## 6.2 Enhancements with Machine Learning

To improve indoor location prediction capabilities over prior work, we propose to integrate machine-learning techniques that intelligently make use of our step detection, inertial navigation, and Wi-Fi fingerprinting phases.

In our *LearnLoc* framework, we adapt three supervised learning algorithms to assist with Indoor Localization. We use the K-Nearest Neighbor, Artificial Neural Networks and Linear Regression algorithms. All the algorithms are used for regression and not for classification. This is because a classification technique requires dividing the complete map area into a fine-grained grid for accurate localization, which creates a prohibitively large input space that limits prediction effectiveness and is impractical for resource-constrained smartphones [30]. Regression on the other hand can allow us to quickly predict a continuous dependent value, with significantly lower resource demands, which is what is needed for Indoor Localization with mobile devices. This is also the reason why we do not explore unsupervised learning algorithms, as there is no straightforward unsupervised machine learning algorithm that can be used for regression.

All the algorithms allow us to predict the location of a user over a majority of the map area with the few fingerprints that are collected in the training phase. We implement the Linear Regression training phase on a server and then *offload* the predictions (testing phase) on the mobile device, while both the training and testing phases for the KNN and ANN algorithm are implemented on the mobile device.

### 6.2.1 KNN

We do a complete implementation of the KNN algorithm in Java. In the training phase we aggregate all the Wi-Fi fingerprint data and sort out only the significant Wi-Fi fingerprints. The data is sorted and stored in a 2-dimensional array. During the test phase the new acquired data is filtered and sorted. Using the *Euclidean distance* the nearest neighbors are determined. Finally the data is averaged to get the regressional prediction from the KNN algorithm. Since most of the calculations are done in the test phase only the time for KNN training is very small as seen in Table 6.1.

**Table 6.1: Train times for ANN and KNN algorithm**

| Technique | Train time (ms) |
|---|---|
| KNN Train | 24000 |
| ANN Train | 253200 |

The KNN regression algorithm can predict only one value at a time. So in order to get the prediction for another value the KNN algorithm needs to be re-run for the second value. But this adds to the computational costs. In LeanLoc we need to predict the *X*, *Y* locations of the user. In order to avoid the computational costs we just run the KNN algorithm once and find the nearest neighbor based on the Wi-Fi Fingerprint data. We then obtain the *X* and *Y* locations for the nearest neighbors and average them separately to give the predicted *X* and *Y* values. This saves the device of additional computational cost of finding the nearest neighbor twice.

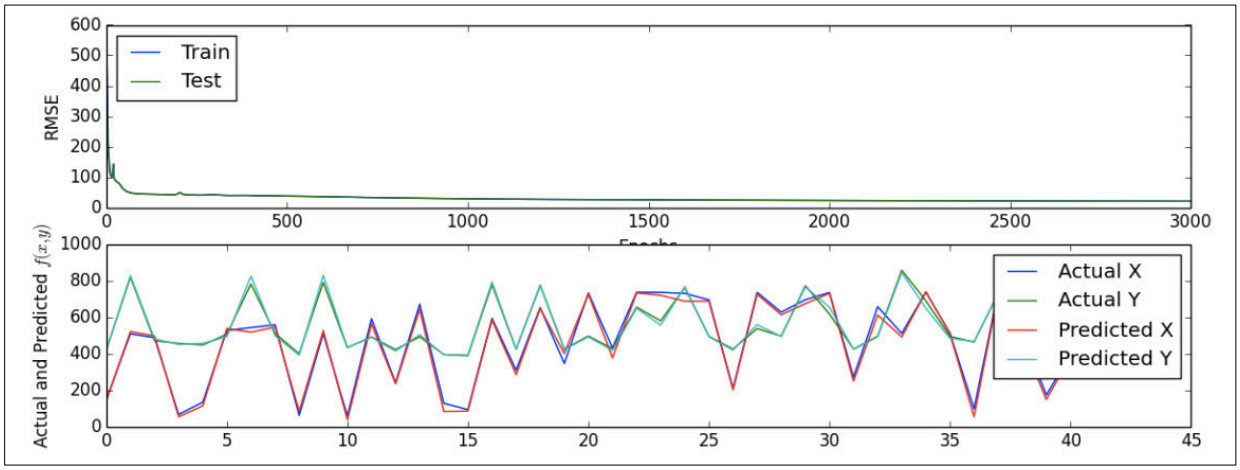


**Figure 6.9: Actual vs. Predicted values by increasing the number of Epochs**

### 6.2.2 Artificial Neural Networks

The ANN algorithm involves a long training phase, as mentioned in Section 5.3 earlier. To set the weights in the ANN algorithm, errors are back propagated and minimized to update the weights. This process usually takes long. Increasing the number of hidden units too adds more complexity to the ANN algorithm and the errors take longer to converge. There are a number of parameters that can be set for the ANN algorithm.

The user can set a number of parameters like the number of hidden units, the epochs or number of repetitions performed to converge to the minimum error and the learning rates. We experiment with a few of these parameters. We found that keeping the number of hidden units as 14 leads to precise training, while decreasing the learning rate leads to more accurate training. We also see in the figure that increasing the number of epochs after 1500 doesn't result in any better training, hence we use the number of epochs as 1500. The whole implementation for Neural Nets is done in Java.

### 6.2.3 Linear Regression

Linear Regression as mentioned earlier is done offline on the computer. Hence we create the implementation using python code. We see that the Wi-Fi signal strengths have a non-linear relationship with the locations. We did a java implementation first but the training times we so large that we had to go with the *offline* version of the linear regression algorithm.

Chapter 7

Experiment and Results

In this section we discuss various experiments performed using *LearnLoc*. The experimental setup and methodology are discussed first followed by experimental results and discussion.

## 7.1 Experimental Setup

### 7.1.1 Device Power Modeling

We use two mobile devices, HTC Sensation [59] and Nexus 4 [60] both running on the Android 4.2 Operating System. We discuss the energy modeling on these devices in this section.

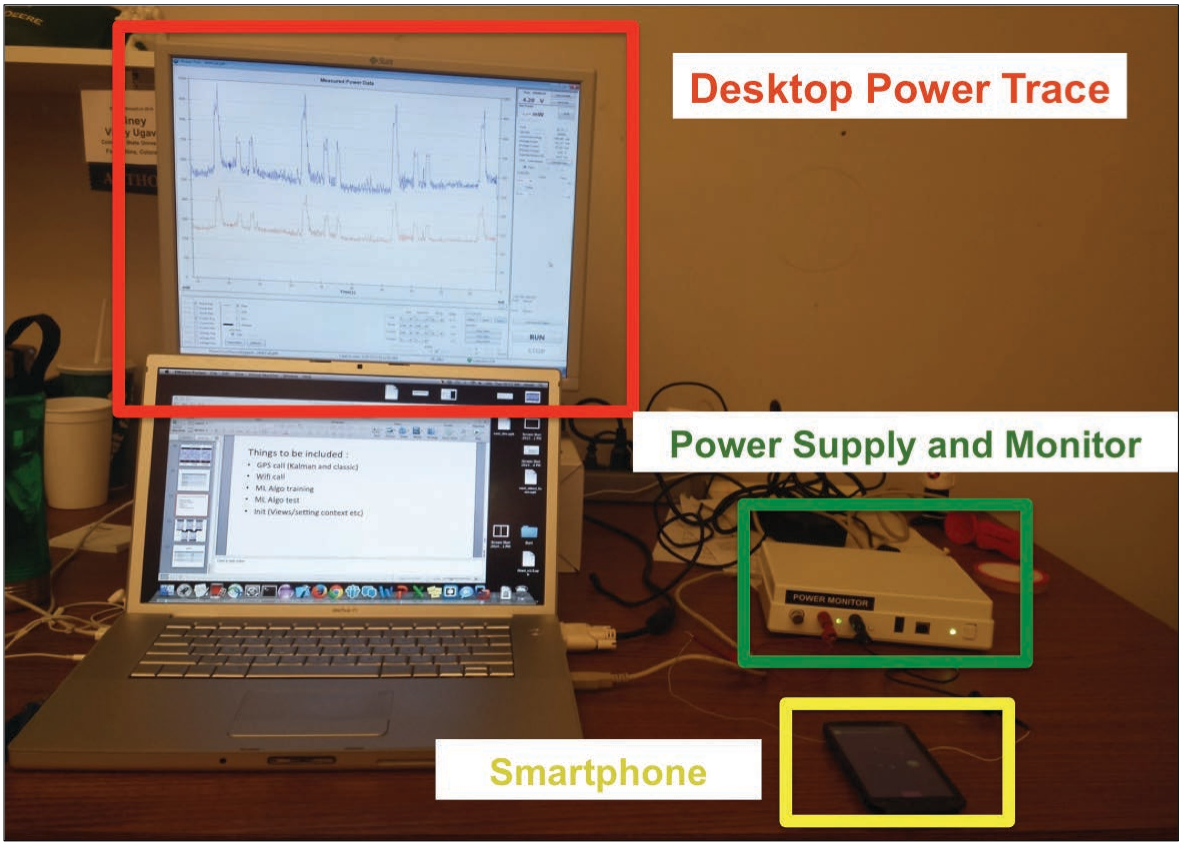


**Figure 7.1: Power Monitor Setup**

### a) Overview

We thought of a number of different ways in which we could model the energy effectiveness of the Indoor Localization technique. We eventually narrowed down to create an energy model specific to Indoor Navigation techniques. To estimate energy for the Indoor Localization techniques, we used a Monsoon Solutions power meter [61] that connects to a smartphone and provides a profile of power dissipated over time when the smartphone is in use. We used this meter to determine energy consumption for the various steps involved as part of the Indoor Localization techniques. The energy meter can offload the power trace data to the desktop computer.

Using the power meter, general energy models were created for both the mobile devices. All energy values were averaged over five readings. “Init” is the energy overhead of invoking the user interface (UI) and context for a single instance of localization prediction. The energy values for a single instance of the step detection task using the two inertial sensing techniques (“Classical” and “Sensor Fusion”) are shown next. The energy spent to perform a single instance of Wi-Fi fingerprinting (“Wi-Fi scan”) in *LearnLoc* can be seen to be quite large. This step takes about 500 ms. Given the high overhead of a Wi-Fi scan we explored various Wi-Fi scan intervals to balance location prediction accuracy with energy costs and implementation efficiency (see Section 7.3.1). Table (Table 7.1) for our energy model is shown below. The details for each module are explained in the subsequent sections.

**Table 7.1: Energy Model**

| Action/Module | Energy (mJ) |
|---|---|
| Init | 831.21 |
| Classical (Acc + Mag) | 1335.93 |
| Sensor Fusion (Acc + Mag + Gyro) | 1715.88 |
| Wi-Fi Scan | 2380.25 |
| KNN Train | 949.60 |
| KNN Test | 4621.70 |
| Linear Regression Train | Not Applicable |
| Linear Regression Test | 290.31 |
| NN Train | 31592.27 |
| NN Test | 580.63 |

### b) Wi- Fi Power Model

We created a sample app that can do a simple Wi-Fi scan by the click of a button. The mobile device was then connected to the power meter and the power was recorded. A single Wi-Fi instance took about 500 ms to complete. The power consumed in this process is modeled using the energy meter as shown in Figure 7.2 where the power trace is shown in orange while the current trace is shown in blue. This graph also shows the power trace for three instances of Wi-Fi scan. The average energy consumed for a single Wi-Fi scan is 2380.25 mJ.

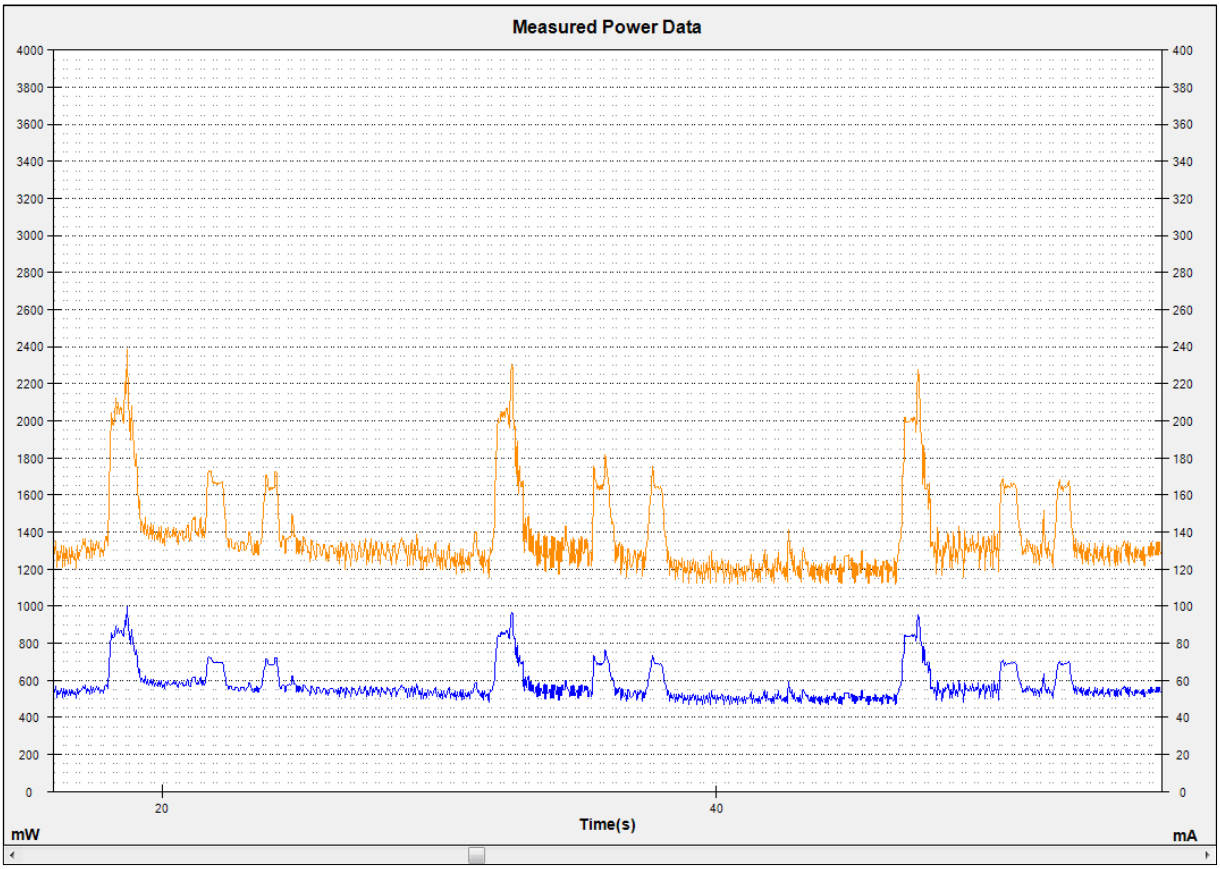


**Figure 7.2: Wi-Fi scan Power Trace**

### c) Step Detection

The step detection algorithm uses the three positioning sensors primarily to tell us if a step is detected and to give us the heading angle to which the user is pointing. It involves compute intensive techniques like Kalman Filtering that demands high processing power. The Figure 7.3 below shows the graph for 5 consecutive steps detected using Kalman Filtering in the Sensor fusion algorithm. The average energy consumption for step detection using the Sensor Fusion Algorithm is 1715.88 mJ. We record the power dissipated with the use of Classic inertial navigation technique for comparison purposes later.

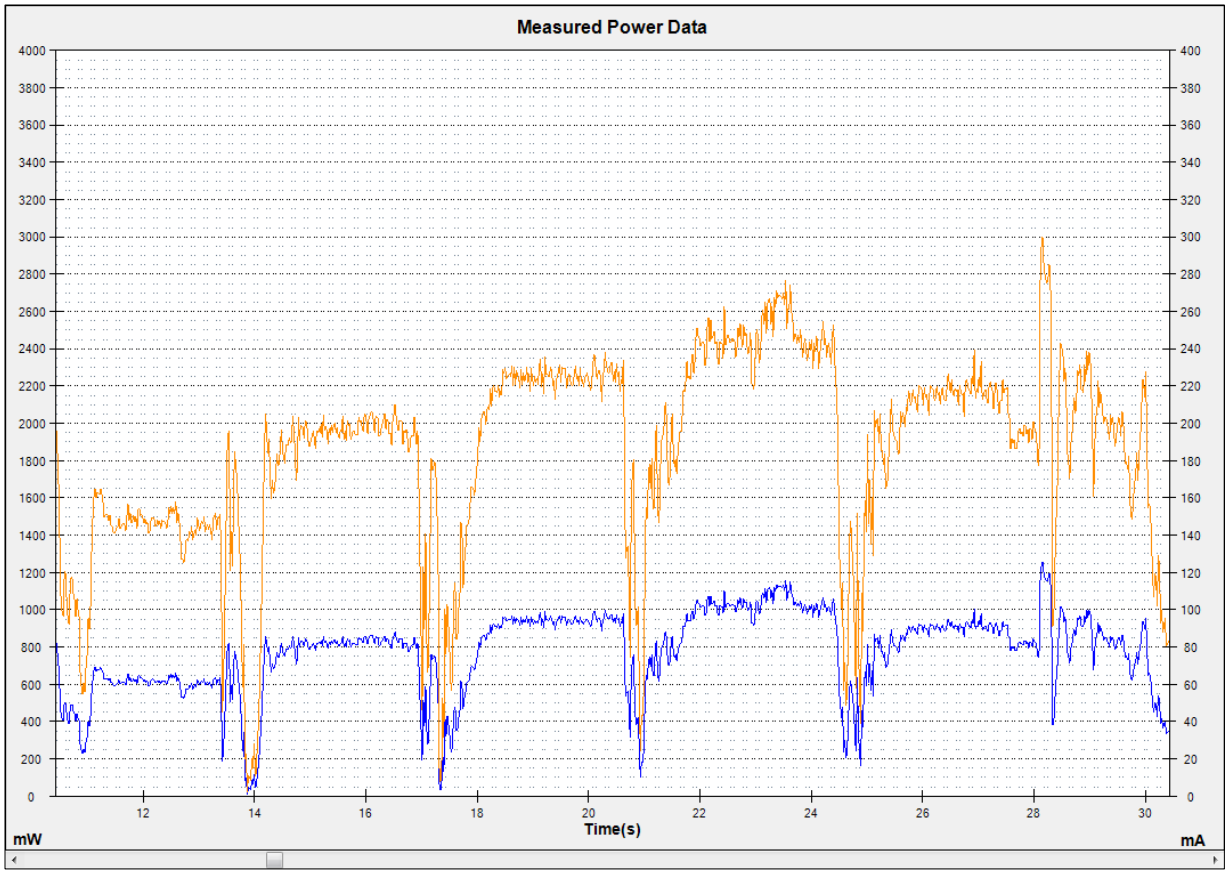


**Figure 7.3: Power Trace for Step Detection using Sensor Fusion technique**

### d) Energy modeling of the machine-learning algorithms

To model the train and test times for the machine-learning algorithms we use the same power monitor. We run the training module in *LearnLoc* for all the learning algorithms and record the power trace. The Figure 7.4 shows the energy consumed for a single instance of the ANN train. This energy is modeled for a train set having 250-300 samples, which is the case in most of our experimental paths. The test module for the learning algorithm is run every time after the Wi-Fi scan. We run the test modules for all the learning algorithms we use in *LearnLoc* and record their power traces.

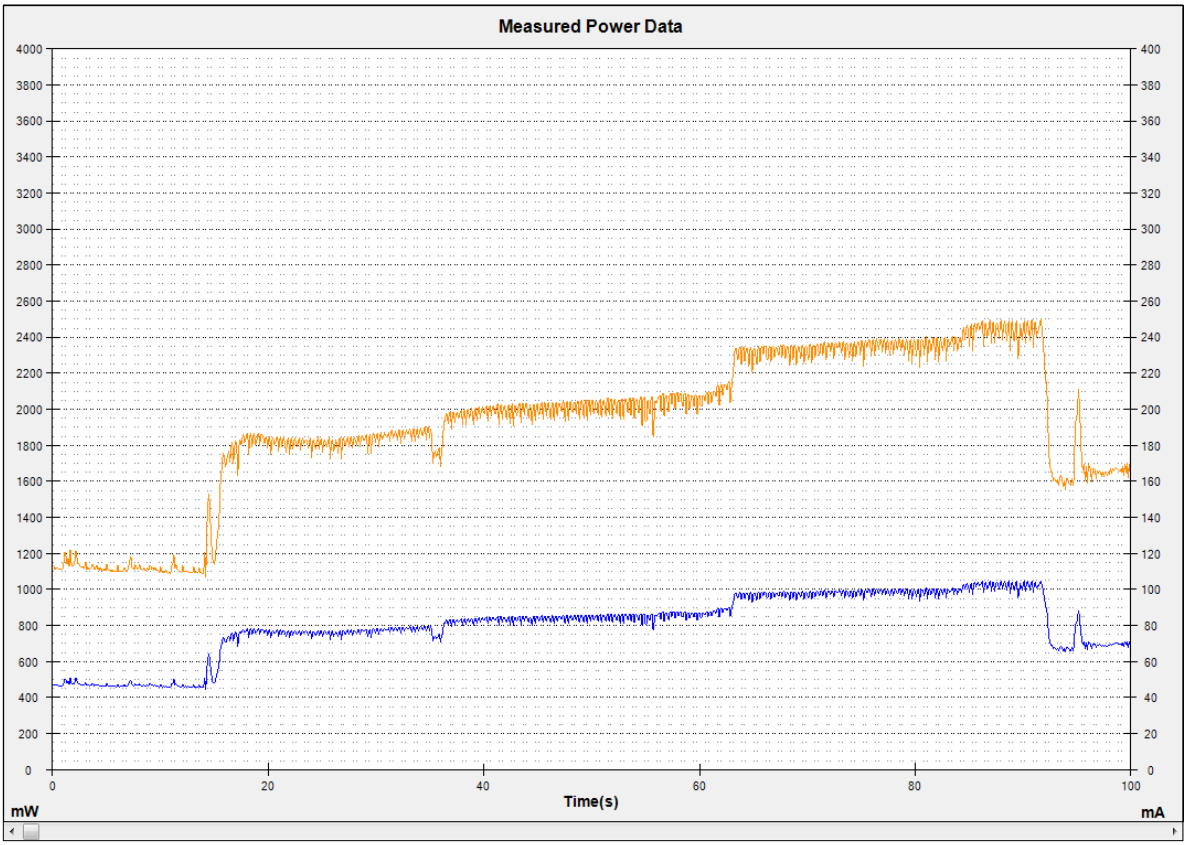


**Figure 7.4: Power trace for the training module using Artifitial Neural Network Algorithm**

### 7.1.2 LearnLoc Mobile App and Implementation

We designed a *LearnLoc* mobile app for the Android ecosystem. The app significantly extends the scope of an open source project called Wi-Fi Compass [33] by integrating our machine learning algorithms and optimizations for low overhead implementations on resource-constrained smartphones. The app allows fingerprinting paths in a given map as part of a training phase. In this phase, the user must initially select a map, set the scale for the map, and specify the starting position on the map. The app then performs Wi-Fi scans at regular intervals while the user moves along the indoor environment. This phase continues till the user explicitly indicates an end to the training phase in the app. The captured data is used to train our machine learning algorithms. The testing phase uses the trained learning algorithms to make predictions. In this

phase, the app provides indoor location estimates that are highlighted on the map. Our mobile app implemented all variants of the *LearnLoc* framework: using KNN, Linear Regression and Neural Networks. The app provides the flexibility to set parameters for the step detection algorithm, such as step distance and window size for step detection. It is also possible to set the Wi-Fi scan frequency and the maximum distance thresholds up to which the predictions from the learning algorithm are valid.

A sample use case of LearnLoc would involve the following steps:

I. Sensor Calibration
II. Create a new Training Project
    a) Select Map Background
    b) Set Map Scale
III. Gather Wi-Fi Fingerprints
IV. Save Project
V. Create a new Testing Project
    a) Select the training project for the training data
    b) Set the maximum threshold for the LearnLoc algorithm
    c) Set the Wi-Fi scan frequency
VI. Press start button and start walking
VII. Press stop when done

Screenshots for the train and test project use cases in *LearnLoc* are shown in Figure 7.5. All the steps above need not be followed in the same order. The user can always go back and change the parameters and other custom settings by opening the project and selecting the settings menu.

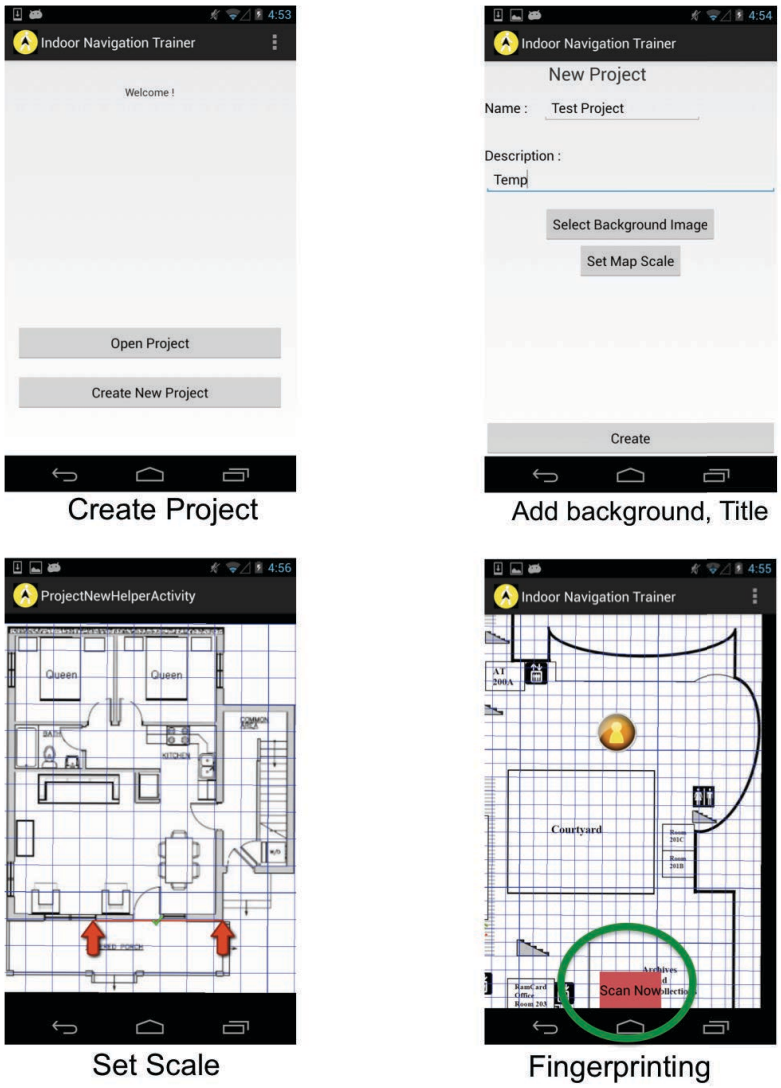


**Figure 7.5: LearnLoc Train Project Use Case**

On using our *LearnLoc* app, we found that for several indoor environments there are a large number of visible Wi-Fi MAC *id*'s. To keep run-time overheads low, which is critical for resource constrained mobile devices, and prevent overfitting of learning data, we filtered and considered only the most significant MAC *id*'s, defined as those MAC *id's* that are at least present at 12 different points in the training data. For a location where a MAC *id* was not present (after the filtering step), we set the signal strength to zero.

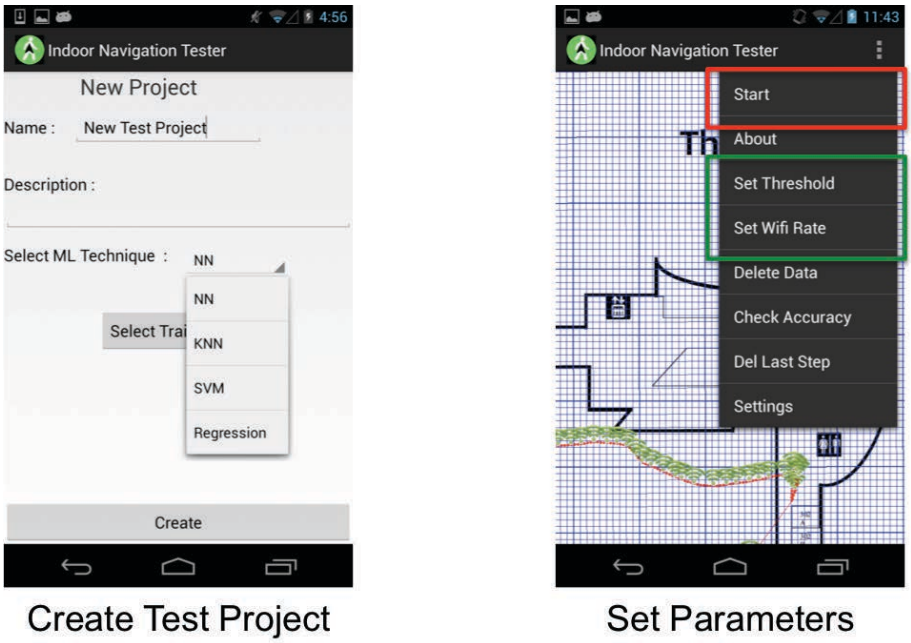


**Figure 7.6: LearnLoc Test Project Use Case**

We also encountered a convergence problem when running *LearnLoc* and this problem is illustrated in Figure 7.7. It was observed that when a user is in a particular region (e.g., the circle in the figure), the learning algorithms predicted the same location repeatedly until the user had moved completely out of the region. To address this issue, we store the previous prediction from the learning algorithm. We then calculate the distance between the new and the previous predictions. If we observe that the distance is below a particular threshold, we discard the prediction and use the inertial trace to obtain the location prediction. We set the threshold in this case to be less than the distance for one step (~120 cm).

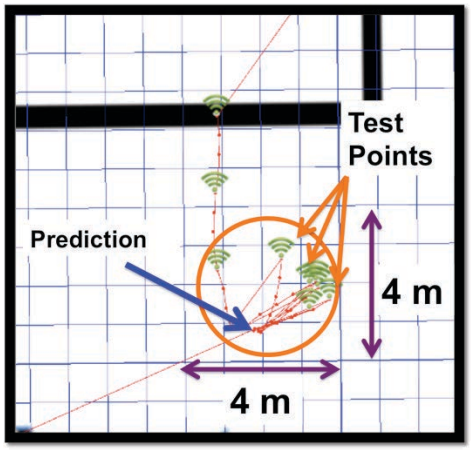


**Figure 7.7: Convergence Problem**

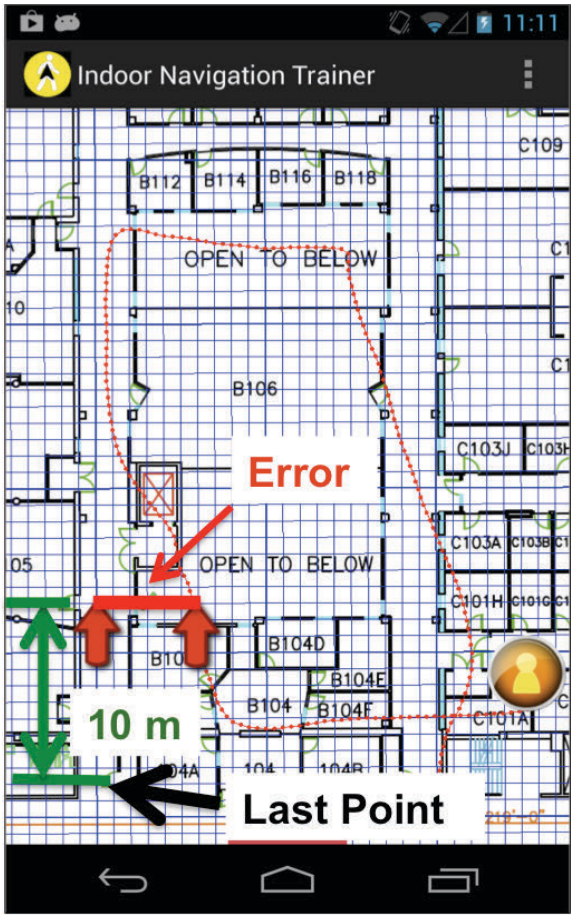


**Figure 7.8: Accuracy Estimation: to measure the error the two red arrows must be stretched between the actual point on the path and the corresponding point on the traced path**

### 7.1.3 Accuracy Estimation

We estimate the accuracy of the Indoor Localization techniques by checking for the deviation of prediction error along the path after every 10 meters. We implemented *LearnLoc* and techniques from prior work as apps. A widget in the app as shown in Figure 7.8 calculates the distance between the traced path and the actual path (specified by the user). The widget then uses the scale factor for the map *(as discussed in Section 7.1.2)* to obtain the actual distances and then averages the piecewise errors to give an overall average error for each technique.

### 7.1.4 Indoor Paths for Benchmarking

To show the effectiveness of our *LearnLoc* framework as well as the prior works that we compare against, we selected four indoor paths in three buildings that are part of the Colorado State University, Fort Collins campus. These paths were used as benchmarks for our accuracy and energy comparisons between the Indoor Localization techniques (discussed in Section 7.3.2). Figure 7.9 shows the paths highlighted in red against the backdrop of the indoor floorplan. The starting and end points are marked as “S” and “E” respectively. The path lengths range from 80 meters to 140 meters.

Each building was chosen because of its unique characteristics that differed from other buildings. The ***Clark building*** is one of the oldest buildings on campus, and primarily made of wood and concrete. We choose two paths *(Clark L2 South and North)* in this building. The ***Library*** is a relatively new building that has a mix of metal and wooden structures with open spaces. We chose three paths *(Library L3, Library L2, and Library Basement)* in this building. Library Basement is the shortest of our paths. The ***Engineering building*** is neither new nor old and has a significant amount of metal in its structure as well as in the equipment in the labs. The presence

of a large quantity of metal creates magnetic disturbances, which can complicate Indoor Localization. We chose one path *(Engineering B)* in this building.

For a precise accuracy analysis we obtained the building floor plans with the actual map dimensions. We used these dimensions to set the scale on the map during our project setup in *LearnLoc*. With the help of Image tools we trim the map area containing our paths and create the background map images for the *LearnLoc* app.

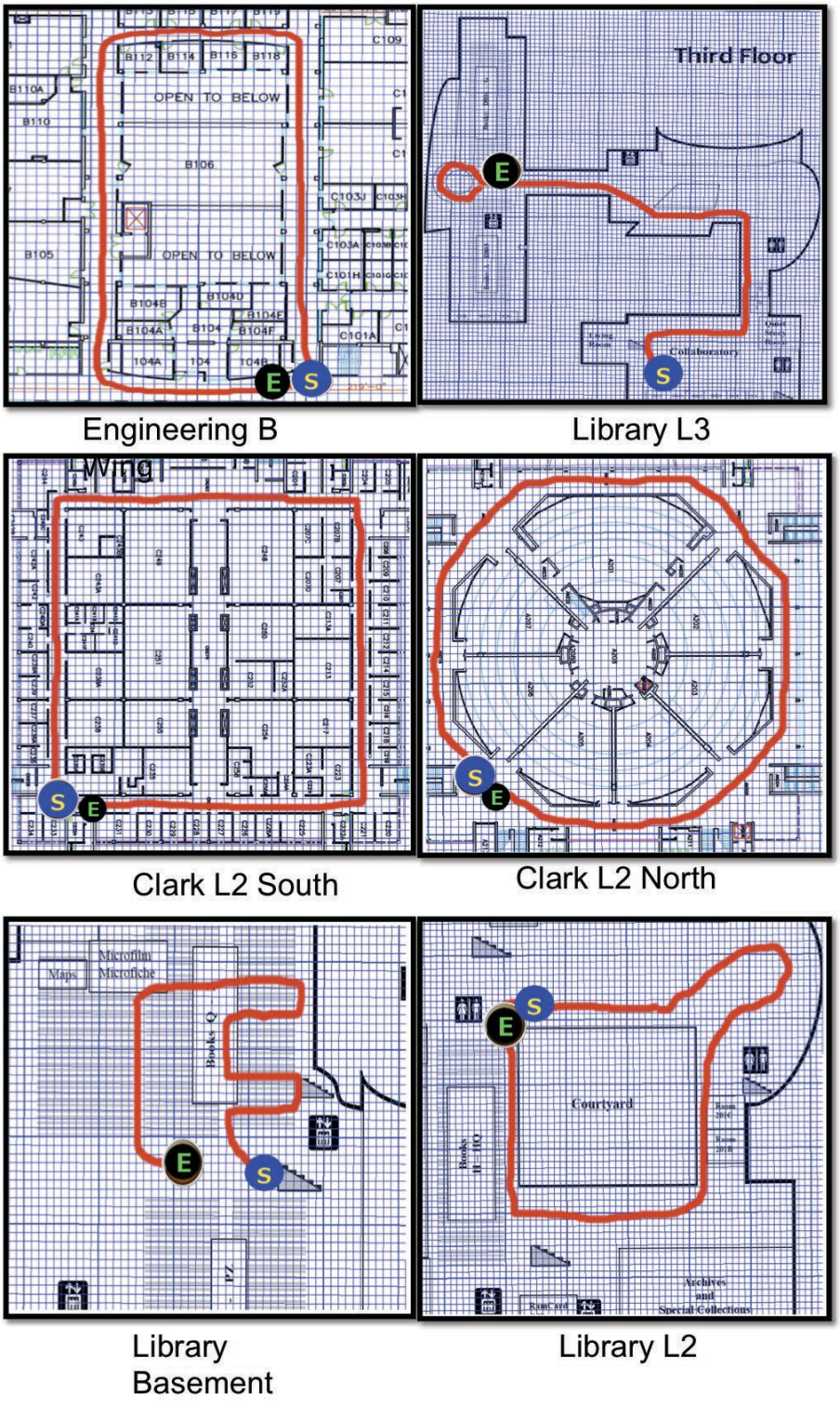


**Figure 7.9: Indoor Paths for Benchmarking**

### 7.3 Experimental Results

In this section, we first present results that explore the impact of changing the Wi-Fi scan interval within *LearnLoc*. Subsequently, we present results comparing *LearnLoc* with prior work in the area.

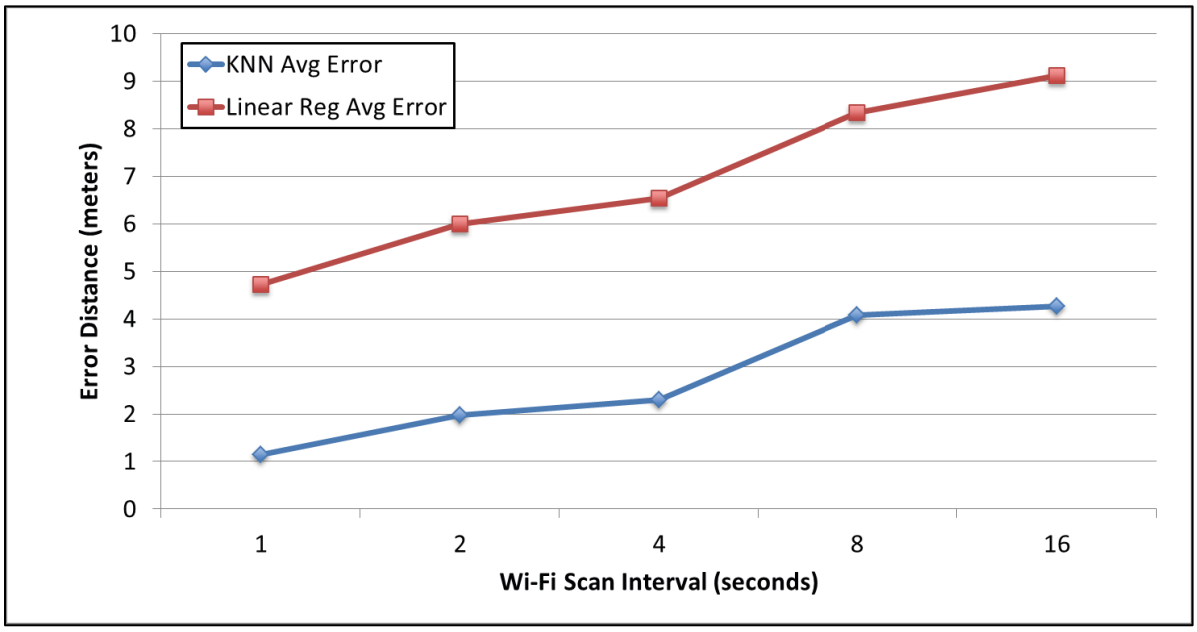


**Figure 7.10: Error distances with changing Wi-Fi scan interval**

#### 7.3.1 Wi-Fi Scan interval Sensitivity Analysis

Given the high energy and time overhead of a Wi-Fi scan as discussed in the previous section, we were interested in determining the most suitable value of a Wi-Fi scan interval for our *LearnLoc* framework. We therefore conducted a sensitivity analysis and recorded the indoor location estimation accuracy and energy costs for the KNN and Linear Regression variants of *LearnLoc*. Figure 7.10 shows the average location estimation error and Figure 7.11 shows the energy consumed for the Indoor Localization on the HTC Sensation smartphone with Wi-Fi scan intervals varying from 1-16 seconds, for the KNN and Linear Regression variants. The results show an average across all four paths. Figure 7.12 shows a detailed look of the predicted paths

for different Wi-Fi scan intervals when using the KNN variant of *LearnLoc* on the Clark L2 South path.

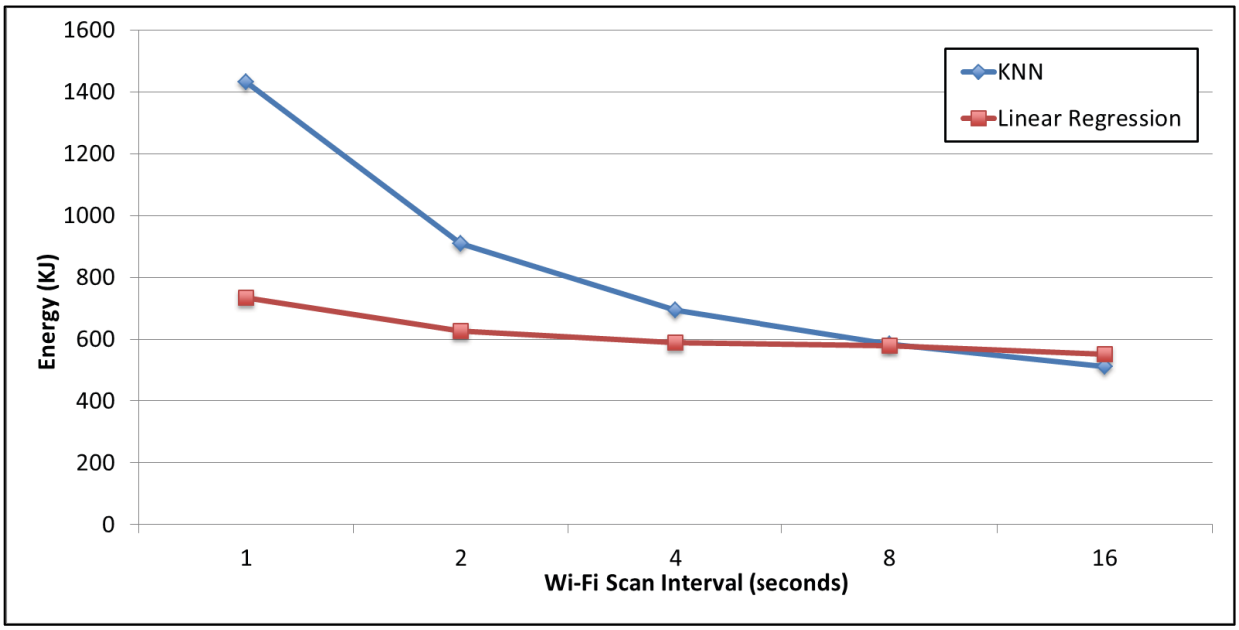


**Figure 7.11: Energy consumption with changing Wi-Fi scan interval**

From the Figure 7.10, it can be observed that the lowest Wi-Fi scan interval (1 second) results in the highest accuracy, but also incurs a high energy consumption overhead because scanning is performed very frequently (as can be seen by the high density of green dots that represent Wi-Fi scan instances in Figure 6 for the 1 second interval case). As the Wi-Fi scan interval increases, the paths traced start deviating notably from the actual path, and the estimation errors increase. In general, KNN provides better accuracy than Linear Regression. For a 1 second interval, the average error distance for KNN is only 1.138 meters. However, KNN also consumes more energy to make predictions compared to Linear Regression, especially for low scan intervals. The energy cost of KNN drops dramatically when going from a scan interval of 1 to an interval of 2 seconds.

The reduction in the calls to the KNN testing phase algorithm accounts for this notable drop, with diminishing returns for higher interval values.


**Figure 7.12: Paths traced for various Wi-Fi scan intervals for the LearnLoc framework using KNN along the Clark L2 South path; green dots represent an instance of a Wi-Fi scan along the path**

Based on the results from this study, we concluded that a 1 second scan interval is too expensive to be viable. A very high interval value (e.g., 8-16 seconds) is also not viable due to the associated high inaccuracy of location estimation. We ultimately decided to consider a scan interval of 2 seconds, which provides sufficiently high prediction accuracy while consuming a reasonable amount of energy. We use this scan interval value for *LearnLoc* in our experiments in the next section where we compare *LearnLoc* to prior work.

### 7.3.2 Location Algorithm Comparison

To show the effectiveness of our proposed *LearnLoc* framework for localization, we compared it to two prior works that use classical inertial navigation [33] and sensor fusion based inertial navigation [25] for Indoor Localization. Figure 7.13 shows the localization accuracy results for the two variants of *LearnLoc* and the approaches from [33] and [25], while Figure 7.14 shows the energy consumed to obtain the Indoor Localization predictions across the techniques. Results are shown for the four indoor path benchmarks discussed earlier. Figures 7.15-7.20 provide a detailed look at the predicted indoor paths when using the five techniques for all the benchmarked indoor paths.

It can be observed from Figure 7.13 that the KNN variant of our proposed *LearnLoc* framework achieves the best accuracy across all paths, out of all the techniques considered. The accuracy for Neural Network technique is between the accuracy for KNN and Linear Regresson variants. The accuracy of the Linear Regression variant and sensor fusion techniques is quite similar, while the classical inertial navigation technique performs the worst. It is interesting to observe that KNN performs the best in the Engineering building while the Classical technique has one of its worst performances in that building. This can be attributed to the high amount of magnetic interference due to the abundance of lab equipment and metal walls and surfaces present in that building. It is

also interesting to note that all the techniques work well in the Clark building. This is because Clark is a relatively old building with wooden and brick walls. There are very few metallic structures present in the building and hence there is a very little magnetic interference that can impact magnetometer or Wi-Fi readings. The lackluster performance of the Linear Regression variant was found to be the result of non-linearities in the relationship between the Wi-Fi fingerprints and location data that a linear model is unable to capture. Neural Networks does a better job at handling non-linearities and so the higher accuracy.

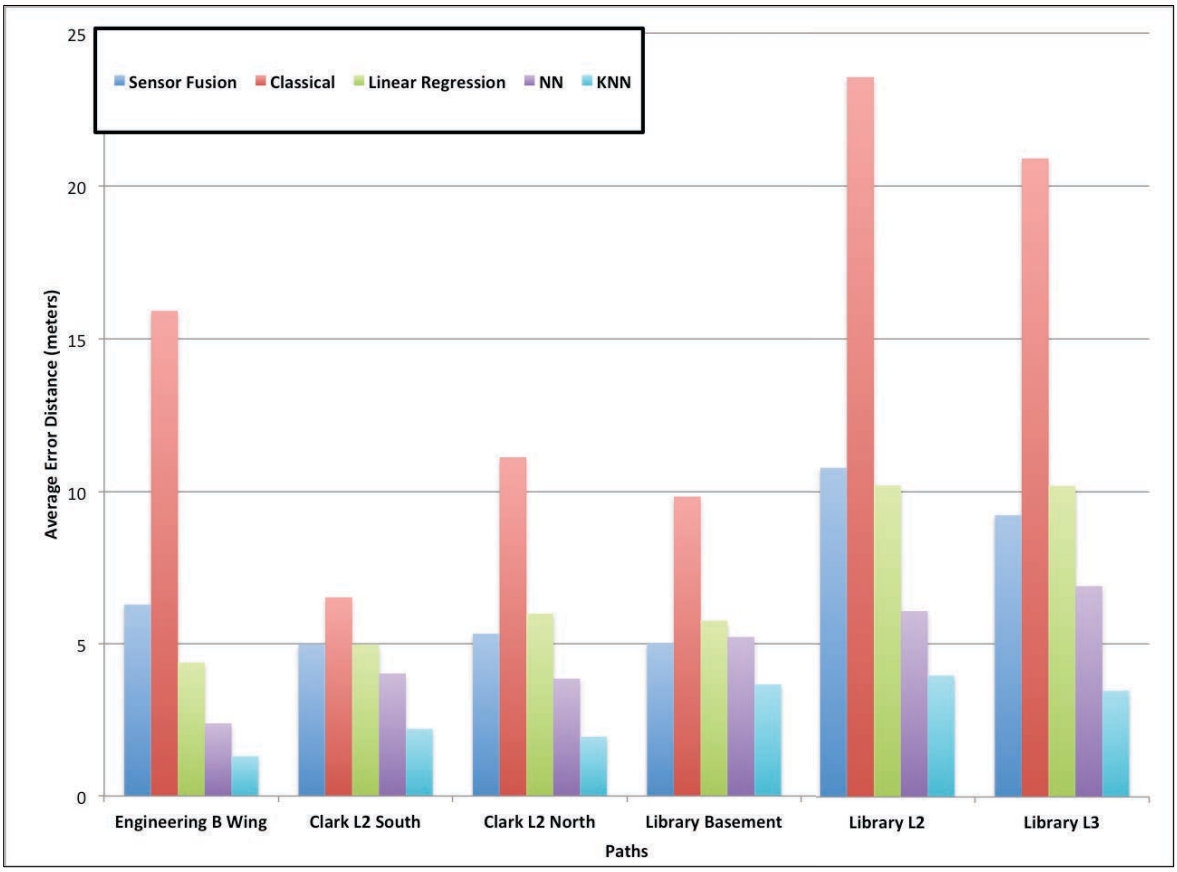


**Figure 7.13: Avg. error distance for Indoor Localization techniques**

From the illustration of the paths traced by the Indoor Localization techniques for the Clark L2 South benchmark path in Figure 7.15, several observations can be made. The path traced by the classical technique greatly deviates from the actual path. This is due to the error accumulation

over time. The sensor fusion technique performs better but still suffers from error accumulation. It is clear that both of these techniques require some form of recalibration periodically, if their errors are to be bounded.

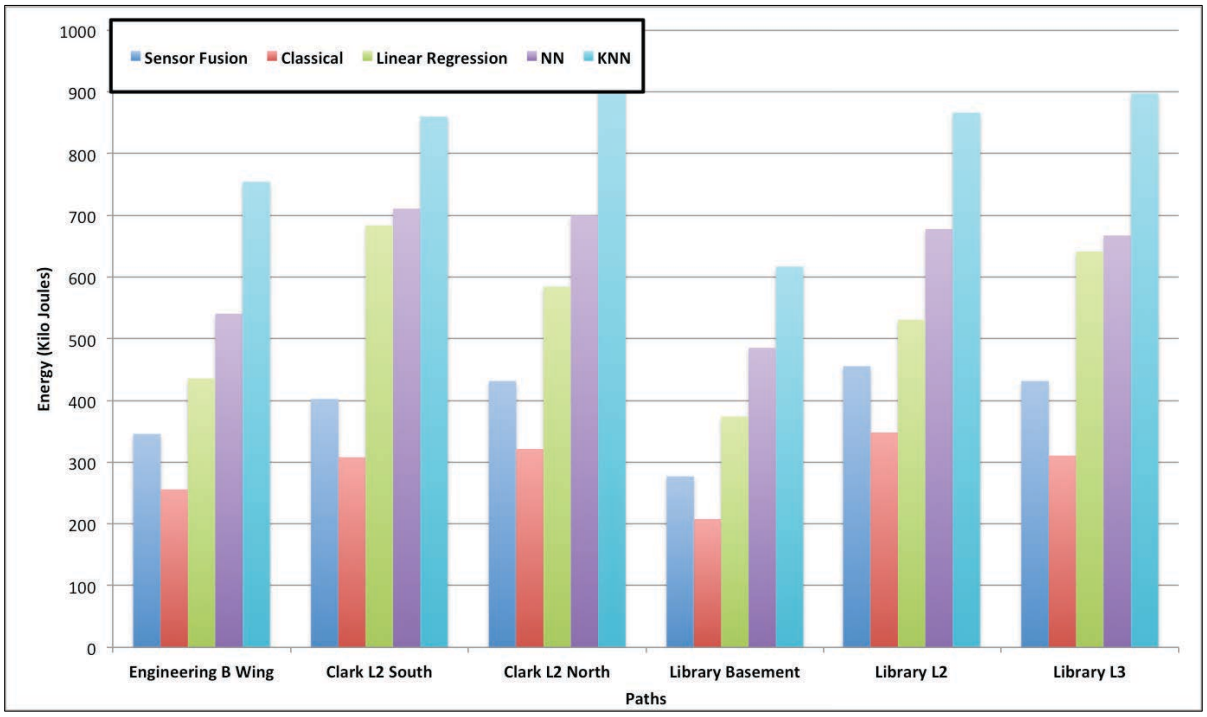


**Figure 7.14: Energy consumption for Indoor Localization techniques**

The use of Wi-Fi together with our learning techniques allows us the ability to generate more robust predictions over time with *LearnLoc*. The green points in the Linear Regression, Neural Networks and KNN variants of *LearnLoc* show the instances where a Wi-Fi scan was performed. For the Linear Regression variant, the generated linear model is not very tolerant to noisy readings and as a result its predictions are not consistently accurate along the path. The KNN variant of *LearnLoc* performs the best out of all the localization techniques, with an average error of 2.228 meters.

The energy consumed by all Indoor Localization techniques is shown in Figure 7.14. The energy consumed on the Library Basement path was the lowest out of all other paths as it was the

shortest path in the study. It can be observed from the figure that KNN is one of the most expensive techniques when it comes to energy consumption. This is attributed to the high computation overhead to generate predictions during the Test phase of the KNN algorithm. However, this energy value is still low enough to enable viable implementation on a smartphone. If lower energy is desired, it is possible to increase the Wi-Fi scan interval to trade-off energy with estimation accuracy (as shown in Section 7.3.1). The Neural Network technique consumes less energy than the KNN variant with accuracy slighty less than KNN. But Neural Network variants require the lengthy training phase. Empowering smartphones with higher processing power can bring down these training times enabling the Neural Network variant to be usable. The sensor fusion technique can be seen to consume less energy than the Linear Regression variant even though the energy for an individual prediction for sensor fusion from Table 6.1 is higher than that for Linear Regression. The high energy for Linear Regression is primarily due to the need for energy consuming Wi-Fi scans which the sensor fusion technique avoids.

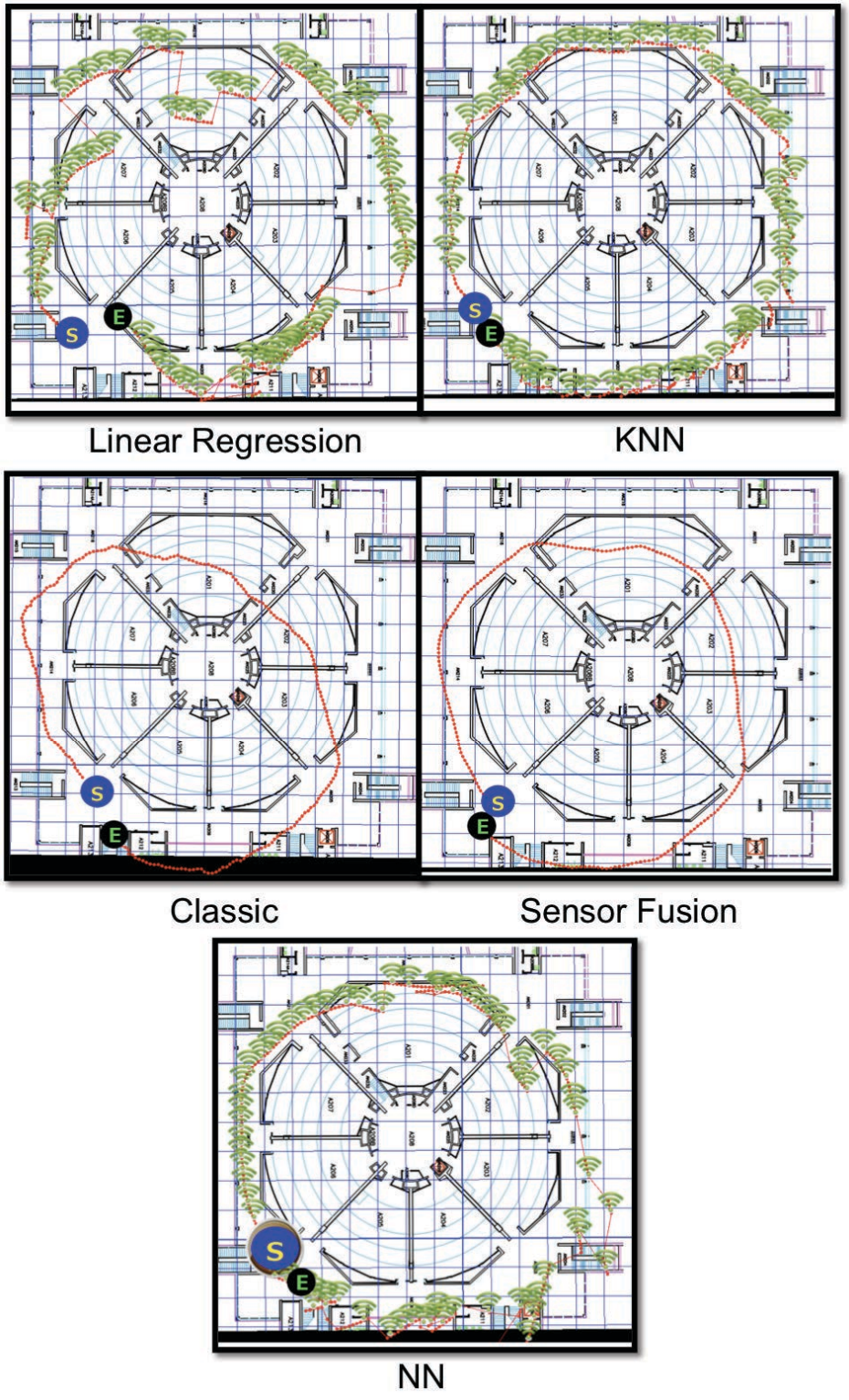


**Figure 7.15: Paths traced by Indoor Localization techniques along the Clark L2 South benchmark path**

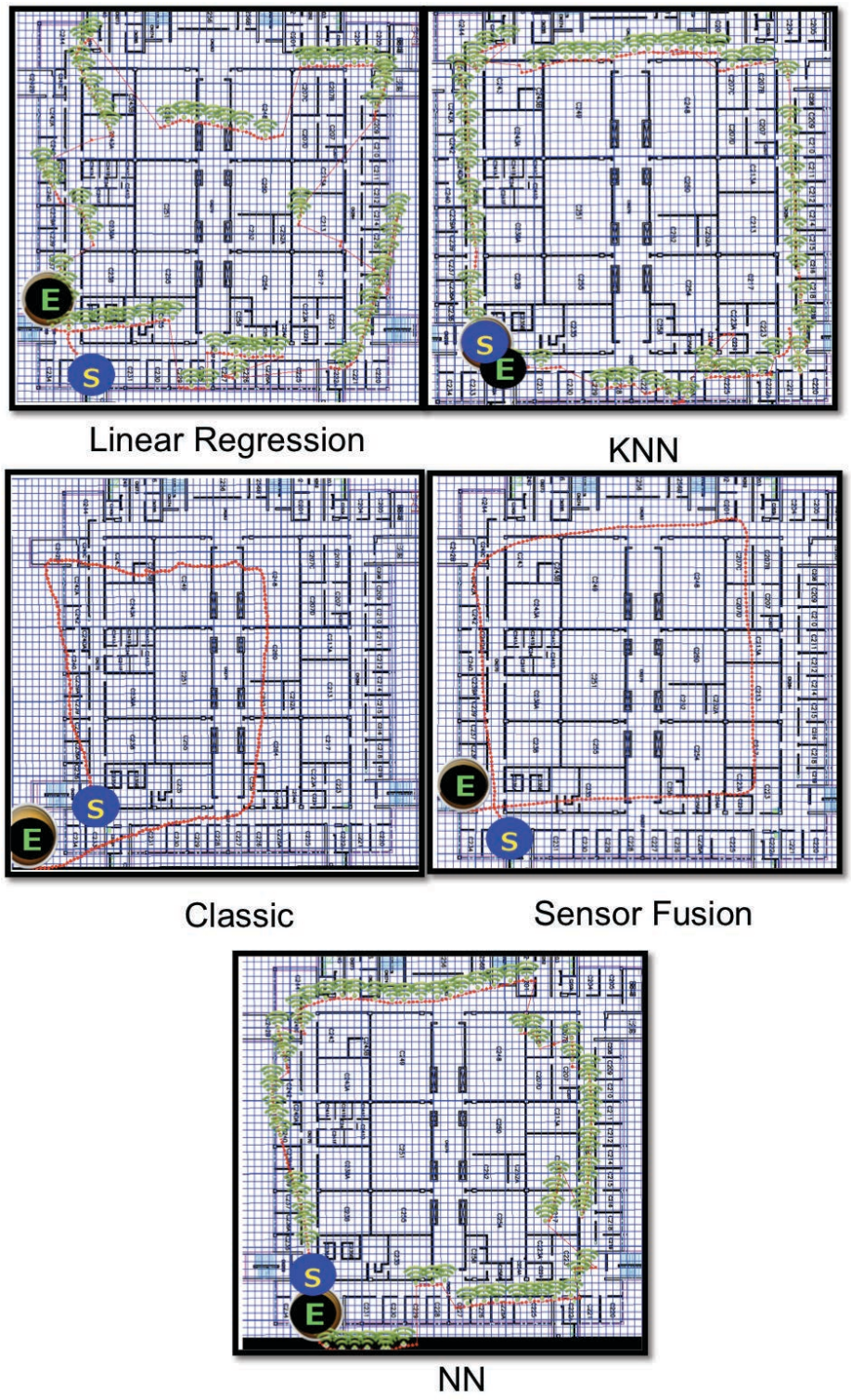


**Figure 7.16: Paths traced by Indoor Localization techniques along the Clark L2 North benchmark path**

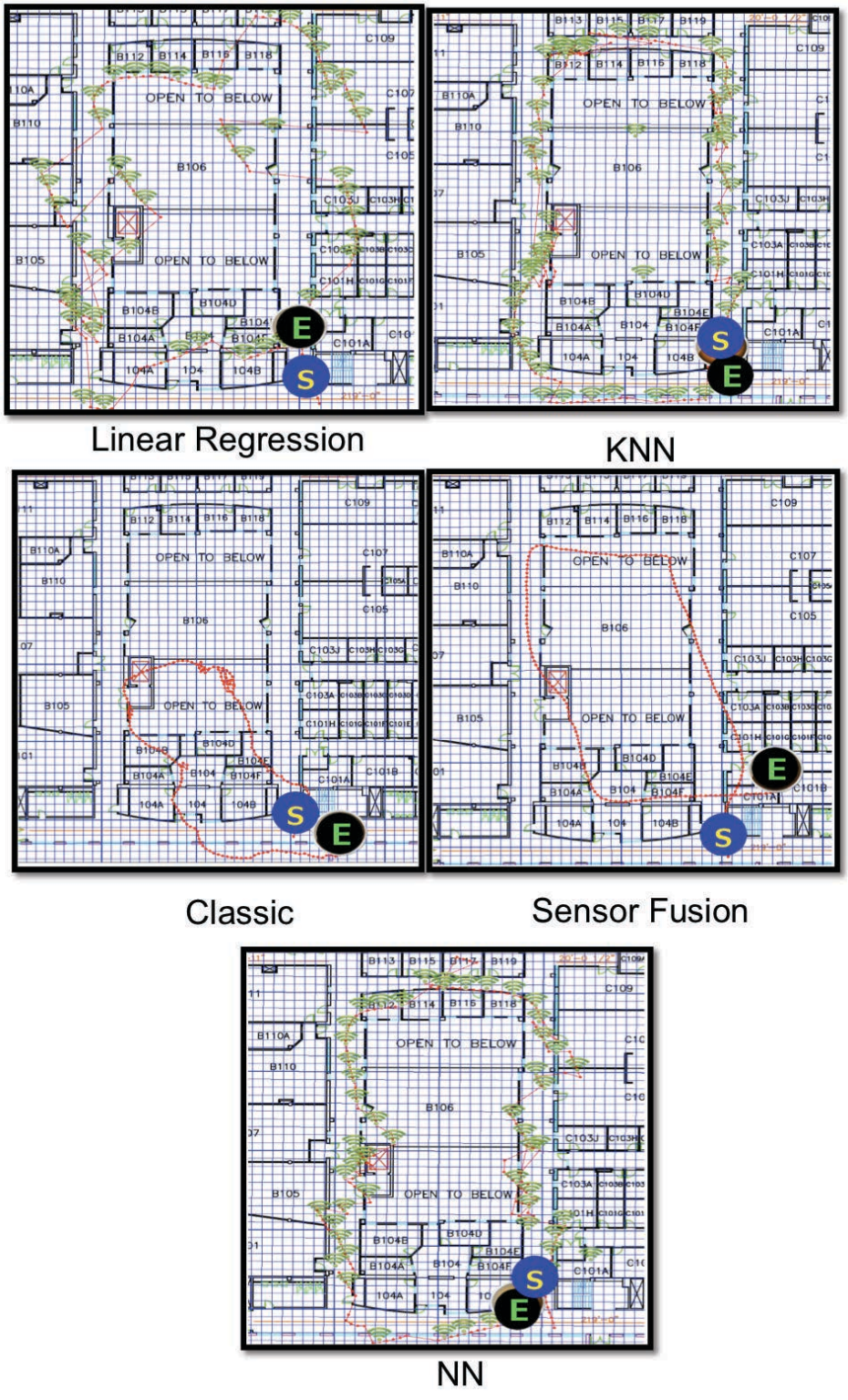


**Figure 7.17: Paths traced by Indoor Localization techniques along the Engineering B benchmark path**

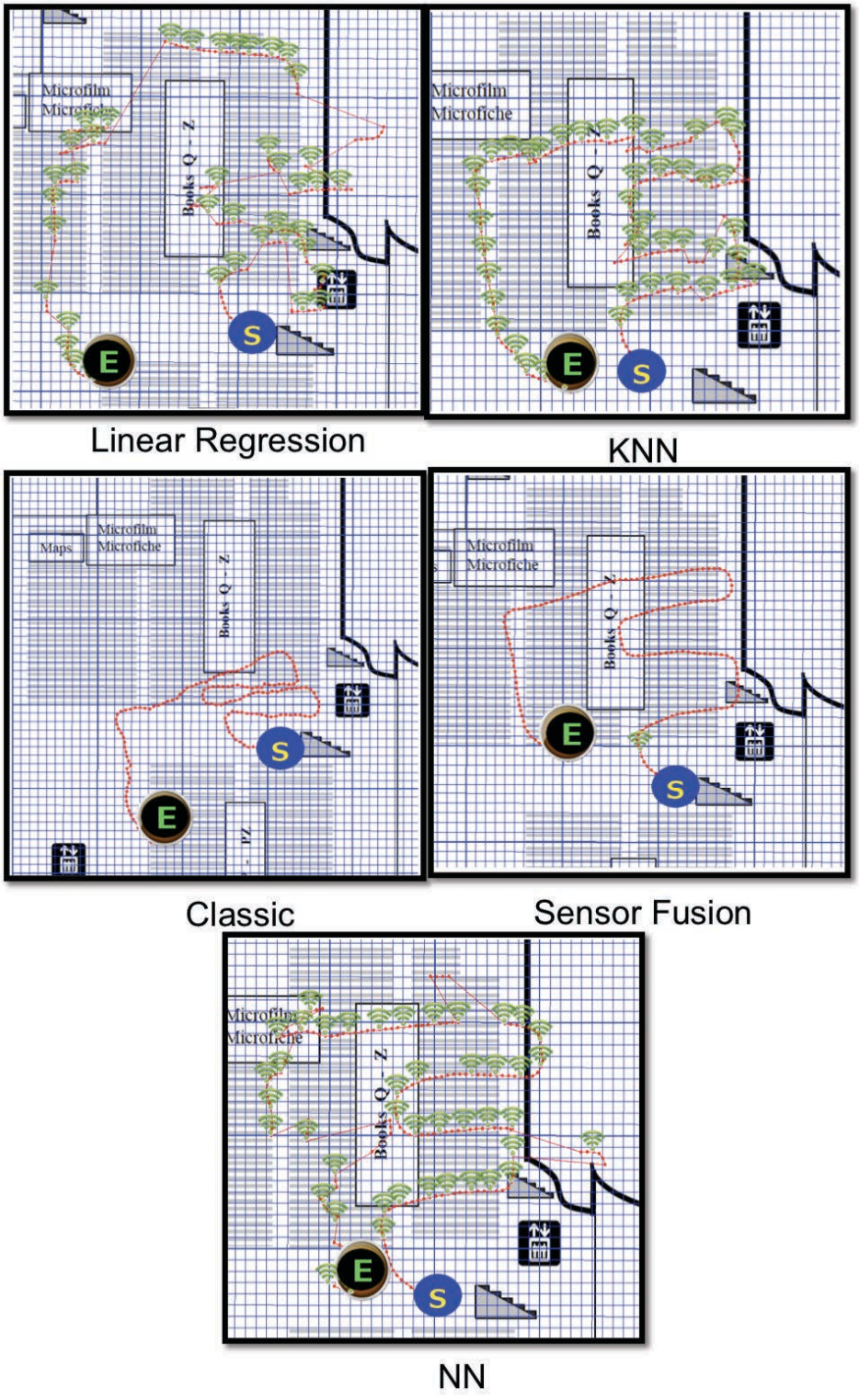


**Figure 7.18: Paths traced by Indoor Localization techniques along the Library Basement benchmark path**

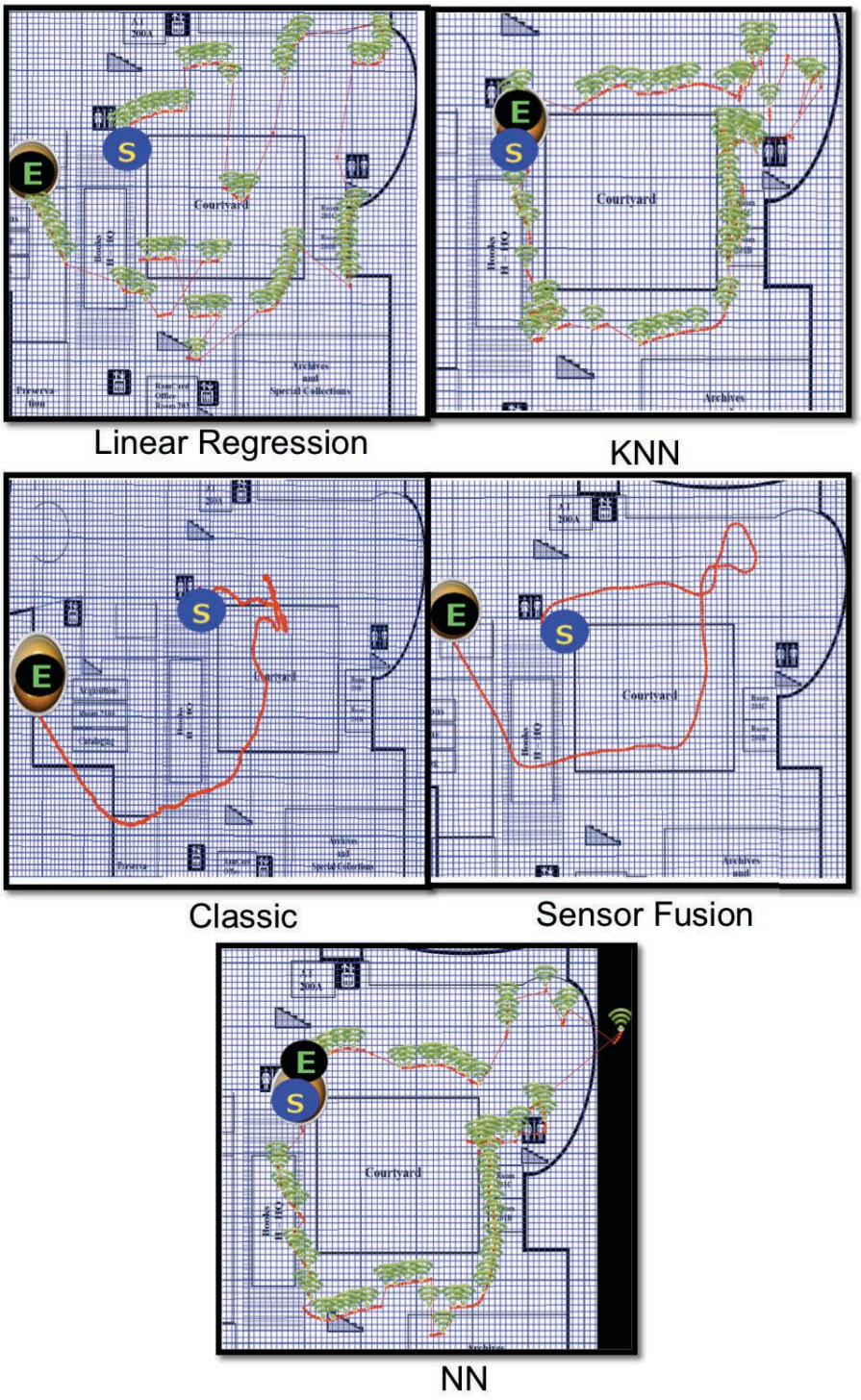


**Figure 7.19: Paths traced by Indoor Localization techniques along the Library L2 benchmark path**

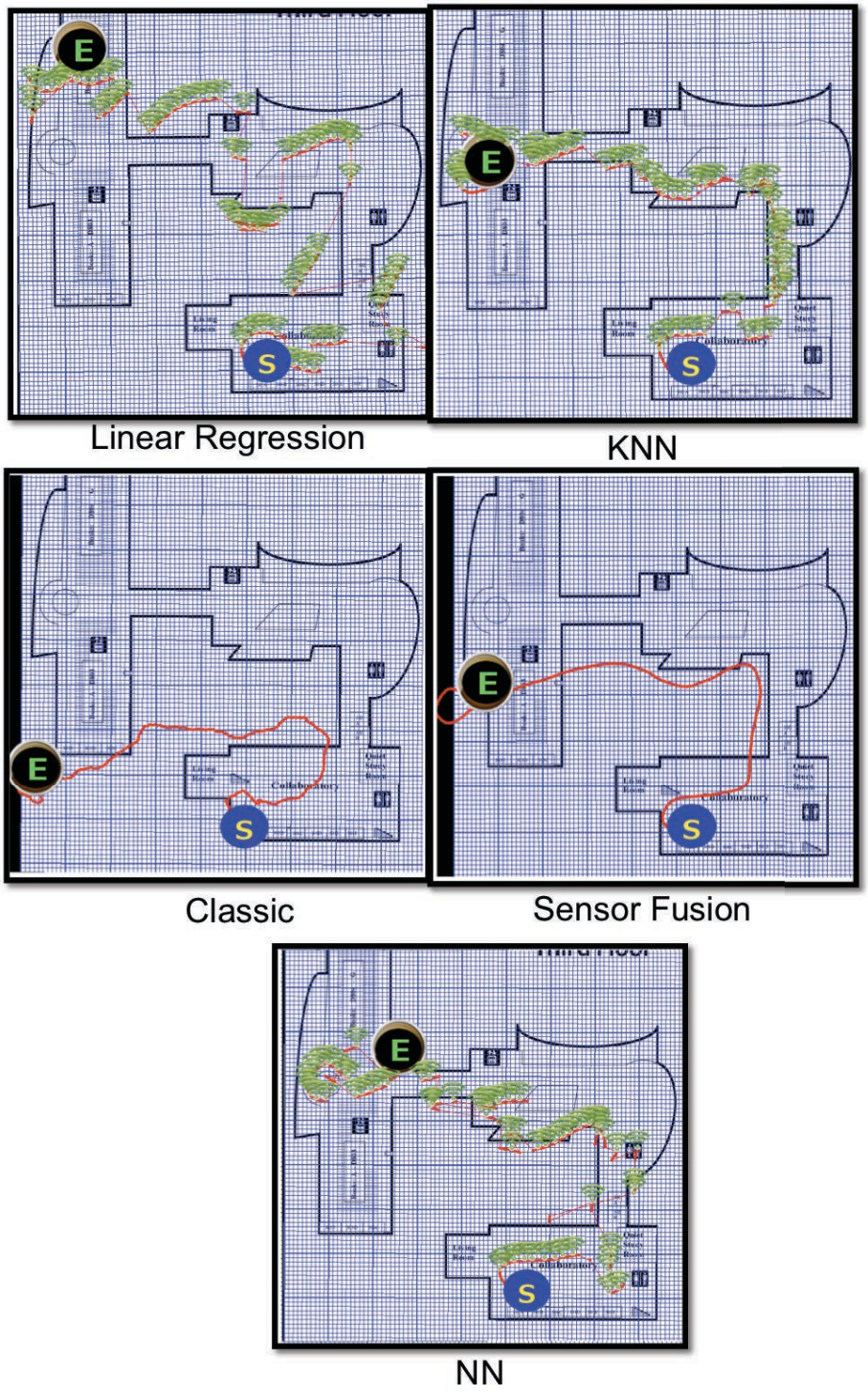


**Figure 7.20: Paths traced by Indoor Localization techniques along the Library L3 benchmark path**

# Chapter 8

## Summary and Future Work

Context aware and location-based services will inevitably continue to grow in the global market. These services have entered the market space already as commercial positioning and navigation systems manifested as GPS devices that were accessible to only a handful of people. But with the surge of the mobile device market these services are available to almost everyone who owns a smartphone today. The advancement in research and technology has led to precise navigation outdoors but absence of satellite signals in the indoor space poses challenges for Indoor Navigation. In spite of the increasing customer expectation, the adoption of Indoor Navigation services is relatively slow due to lack of research resources. The availability of varied sensors on smartphone platforms has led to sensor based Indoor Positioning solutions but the in accuracies of these techniques and their power demand are deceptive in their claim as an outright solution for Indoor Positioning. Wi-Fi signal strength based solutions work in the favor of the notion to control infrastructural costs. Hence, the increasing demand of accurate, energy efficient Indoor Positioning system within the available infrastructure is shaping the future of Indoor Localization solutions and platforms.

### 8.1 Summary

In this thesis we presented the *LearnLoc* Indoor Localization framework. We presented three variants of our framework: one that uses K Nearest Neighbor (KNN) regression based learning and the other that use Linear Regression and Artifitial Neural Networks (ANN). Our experimental studies show that the KNN machine learning based *LearnLoc* approach is robust to noise and magnetic interference and significantly outperforms other approaches from prior work. Our KNN-based approach provides highly accurate Indoor Localization with approximately 1 to

3 meters accuracy. This accuracy however comes at a cost: high-energy consumption. If energy is a critical concern for a given mobile device, one possibility is to trade-off prediction accuracy with energy costs by adapting the KNN algorithm. We showed that it is possible to perform such a trade-off by varying the Wi-Fi scan interval. Our ANN approach achieves accuracy comparable to the KNN-based approach. It even has the advantage of consuming less energy than the KNN approach but it involves a lengthy training phase that cannot be improved for real time usage of the ANN algorithm. The promising results from this study are very encouraging.

### 8.2 Future Work

All the variants of our LearnLoc show promising results for accurate Indoor Positioning by controlling the energy cost. However much work can be done to improve the strategies. Our ongoing work is attempting to explore other learning algorithms and perform more aggressive trade-offs between accuracy and energy.

Ongoing work is focusing on collecting more ambient fingerprints other than Wi-Fi and use classification based learning algorithms to achieve higher accuracy. We are exploring ways of determining a fine-grained energy model that accounts for such compute and memory intensive algorithms. We are also working on the use of more complicated learning algorithms like Support Vector Machine (SVM) on the device.

In the linear regression based approach we offload and upload the data from the phone manually. One solution to improving this would be to make this more dynamic by enabling LearnLoc to automatically offload the data with the help of a wireless data connection like Wi-Fi.

There are other plans for future work in the area of simulatenous location and mapping (SLAM). This would enable *LearnLoc* to create maps automatically by crowdsourcing paths traced by

users in the same building. Another plan is to crowdsource the fingerprinting data that can account for the variations of the Wi-Fi signals over time and also for any changes to the Wi-Fi router positions that has resulted in change of Wi-Fi signal strength over an area.

In *LearnLoc* we did not analyze the memory usage by the different variants in the platform. Most of the learning algorithms have matrix operations and computations that require sizable memory. We plan to account for the memory usage of these operations and do a comparative study of the memory requirements for all the variants in *LearnLoc*.

Although there are always ways for software optimization and unexplored techniques for energy efficient Indoor Positioning, the work presented in this thesis brings us one-step closer to the rising consumer demands in Indoor Positioning based location services using smartphones.

References


[1] Andre M., "LBS Platforms and Technologies", Berg Insight AB, 2013, http://www.berginsight.com/ReportPDF/ProductSheet/bi-lpt4-ps.pdf

[2] Monty G., "GPS use in U.S. Critical Infrastructure and Emergency Communications", http://www.gps.gov/multimedia/presentations/2012/10/USTTI/graham.pdf

[3] Carlo D, "GNSS Market Report", Issue 3, October 2013, http://www.gnss.asia/sites/gnss.asia/files/GNSS_Market%20Report_2013.pdf

[4] Fabio B., "Bringing Navigation Indoors", Nokia Research Centre, 2013, http://geta.aalto.fi/en/courses/bringing_navigation_indoors.pdf

[5] Yellow Pages, "Local Insights Digital Report", 2012, http://corporate.yp.com/news/press-and-media/2012/year-in-review-yp-local-insights-digital-report-reveals-consumer-and-advertiser-local-search-trends/

[6] Christian L., Harald S., "Overview of Indoor Navigation Techniques and Implementation Studies", Morocco, FIG Working week, 18-22 May 2011

[7] John H., "Smartphone and Tablet Penetration", Business Insider, October 2013,http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

[8] ComScore, "Smartphone Subscriber Market Share" , March 2014, https://www.comscore.com/Insights/Market_Rankings/comScore_Reports_March_2014_US_Smartphone_Subscriber_Market_Share

[9] Noam K., "Six things to know about Smartphone Batteries",http://www.cnet.com/news/six-things-to-know-about-smartphone-batteries/

[10] Verizon Wireless , “4G LTE Speeds vs. Your Home Network”, http://www.verizonwireless.com/insiders-guide/network-and-plans/4g-lte-speeds-compared-to-home-network/

[11] T-Mobile Coverage Map, http://www.t-mobile.com/coverage.html

[12] Boingo Wireless Wi-Fi Internet Hot Spots, http://wifi.boingo.com/wireless-internet-hotspots/

[13] Agam S., "64 Bit smartphones are coming, but Apple's iPhone 5s still stands alone", Feb 2014, http://www.pcworld.com/article/2102620/64bit-smartphones-are-coming-but-apples-iphone-5s-still-stands-alone.html

[14] Kristen B., "ARM Snags 95 percent of Smartphone Market, Eyes New Areas of Growth", July 2012, http://www.crn.com/news/components-peripherals/240003811/arm-snags-95-percent-of-smartphone-market-eyes-new-areas-for-growth.htm

[15] Samsung Galaxy S5 Specs, http://www.samsung.com/global/microsite/galaxys5/specs.html

[16] Apple iPhone 5S Specs, https://www.apple.com/iphone-5s/specs/

[17] Cartesian Coordinates, http://mathworld.wolfram.com/CartesianCoordinates.html

[18] Greg M., Adam S., "Professional Android Sensor Programming", June 2012, John Wiley and Sons, Indianapolis, Indiana.

[19] Android Sensors API, http://developer.android.com/reference/android/hardware/Sensor.html

[20] AnandTech Nexus 4 Teardown, http://www.anandtech.com/Gallery/Album/2435#6

[21] Yongyao C., YangZ., Xianfeng D.,James F.,"Magnetometer basics for mobile phone Applications", Feb 2012, http://www.memsic.com/userfiles/files/publications/Articles/Electronic_Products_Feb_%202012_Magnetometer.pdf

[22] Murata Wi-Fi Modules, http://www.murata-ws.com/products/wi-fi_modules.php

[23] Addlesee M., Curwen R., Hodges S., Newman J., Steggles P., Ward A., Hopper A, "Implementing a sentient computing system", IEEE Computer Society Press, Aug 2001.

[24] Bahl, P.; Padmanabhan, V.N., "RADAR: an in-building RF-based user location and tracking system," *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* , vol.2, no., pp.775,784 vol.2, 2000

[25] David P., Michael S., Sharon D., Mark S, 2012, Sensor Fusion Algorithm, Patent US8548608. USA

[26] Yang Liu; Dashti, M.; Jie Zhang, "Indoor localization on mobile phone platforms using embedded inertial sensors," *Positioning Navigation and Communication (WPNC), 2013 10th Workshop on* , vol., no., pp.1,5, 20-21 March 2013

[27] Incheol K., Eunmi C., Huikyung O., "Observation and Motion Models for Indoor Pedestrian Tracking. International", IEEE DICTAP, May 2012.

[28] Ubisense Research Network (2008), http://www.ubisense.net/

[29] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. 2000. The Cricket location-support system. In *Proceedings of the 6th annual international conference on Mobile computing and networking* (MobiCom '00). ACM, New York, NY, USA, 32-43.

[30] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. 2009. SurroundSense: mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th annual international conference on Mobile computing and networking* (MobiCom '09).

[31] Thompson S., IndoorAtlas, https://www.indooratlas.com/

[32] Link, J.A.B.; Smith, P.; Viol, N.; Wehrle, K., "FootPath: Accurate map-based indoor navigation using smartphones," *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on* , vol., no., pp.1,8, 21-23 Sept. 2011

[33] Konrad T., Wolfel P, "WiFi Compass WiFi Access Point Localization with Android Devices", 2012, https://code.google.com/p/wificompass/

[34] Asaf S., Yuval F., Yuval E, "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning", 2010

[35] L. Batyuk, et al., “Context-aware device self-configuration using self-organizing maps” in OC ’11, pp. 13-22, June 2011.

[36] T. Anagnostopoulos, C. Anagnostopoulos, S. Hadjiefthymiades, M. Kyriakakos, A. Kalousis, “Predicting the location of mobile users: a machine learning approach,” in ICPS ’09, pp. 65-72, July 2009

[37] T. Mantoro, et al., “Mobile user location determination using extreme learning machine,” in ICT4M, pp. D25-D30, 2011.

[38] Abhinav, P., Y. Charlie and Ming Z. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof. Eurosys.

[39] Abhinav, P., Y. Charlie and Ming Z . 2011. Fine-Grained Power Modeling for Smartphones Using System Call Tracing. Eurosys.

[40] Donohoo, B.K.; Ohlsen, C.; Pasricha, S., "AURA: An application and user interaction aware middleware framework for energy optimization in mobile devices," *Computer Design (ICCD), 2011 IEEE 29th International Conference on* , vol., no., pp.168,174, 9-12 Oct. 2011

[41] Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, L. Cox, “EnLoc: Energy-Efficient Localization for Mobile Phones,” in INFOCOM ’09, pp. 19-25, Jun. 2009.

[42] K. Lin, A. Kansal, D. Lymberopoulos, F. Zhao, “Energy-accuracy trade-off for continuous mobile device location,” in MOBISYS, pp. 285-298. Jun 2010.

[43] F. B. Abdesslem, A. Phillips, T. Henderson, “Less is more: energy-efficient mobile sensing with SenseLess,” in MOBIHELD ’09, pp. 61-62, Aug. 2009.

[44] J. Paek, J. Kim, R. Govindan, “Energy-efficient rate-adaptive GPS-based positioning for smartphones,” in MOBISYS ’10, pp. 299-314, Jun. 2010.

[45] I. Shafer, M. L. Chang, “Movement detection for power-efficient smartphone WLAN localization,” in MSWIM ’10, pp. 81-90, Oct. 2010.

[46] M. Youssef, M. A. Yosef, M. El-Derini, “GAC: energy-efficient hybrid GPS-accelerometer-compass GSM localization,” in GLOBECOM ’10, pp. 1-5, Dec. 2010.

[47] C. Lee, M. Lee, D. Han, “Energy efficient location logging for mobile device,” in SAINT ’11, pp. 84, Oct. 2010.

[48] K. Nishihara, K. Ishizaka, J. Sakai, “Power saving in mobile devices using context-aware resource control,” in ICNC ’10, pp. 220-226, 2010.

[49] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, N. Sadeh, “A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition,” in MOBISYS ’09, pp. 179-192, Jun. 2009.

[50] Z. Zhuang, K. Kim, J. P. Singh, “Improving Energy Efficiency of Location Sensing on Smartphones,” in MOBISYS, pp. 315-330,. 2010.

[51] T. L. Cheung, K. Okomoto, F. Maker, X. Liu, V. Akella, “Markov Decision Process (MDP) Framework for Optimizing Software on Mobile Phones,” in EMSOFT ’09, pp. 11-20, Oct. 2009.

[52] W. Liang, P. Lai, “Design and Implementation of a Critical Speed-based DFS Mechanism for the Android Operating System,” in EMS ’10, pp. 1-6, Sept. 2010.

[53] R. Jurdak, P. Corke, D. Dharman, G. Salagnac, “Adaptive GPS Duty Cycling and Radio Ranging for Energy-Efficient Localization,” in SENSYS ’10, pp. 57-70, Nov. 2010.

[54] H. Lu, J. Yang, Z Liu, N. D. Lane, T. Choudhury, A. T. Campbell, “The Jigsaw Continuous Sensing Engine for Mobile Phone Applications,” in SENSYS ’10, pp. 71-84, Nov. 2010.

[55] M. B. Kjaergaard, S. Bhattacharya, H. Blunck, P. Nurmi, “Energy-efficient trajectory tracking for mobile devices,” in MOBISYS ’11, pp. 307-320, Jun. 2011.

[56] Y. Liu, S. Lu, Y. Liu, “COAL: Context Aware Localization for High Energy Efficiency in Wireless Networks,” in WCNC ’11, pp. 2030-2035, May 2011.

[57] Mehryar, M., Afshin, R., Ameet, T. Foundations of Machine Learning. The MIT Press, 2012.

[58] Least Squares Fitting. [Online]. Available: http://mathworld.wolfram.com/LeastSquaresFitting.html

[59] HTC Sensation, https://www.htc.com/us/smartphones/htc-sensation/

[60] Nexus 4, http://www.google.com/intl/all/nexus/4/

[61] Monsoon Power Monitor. [Online]. Available: http://www.msoon.com/LabEquipment/PowerMonitor/

Appendix A

# Source Code

This section presents the majority of the source code for the implementation of the three strategies in LearnLoc. Sections A.1 provides the source code for inertial navigation techniques i.e the Classic and Sensor Fusion techniques. Section A.2 and A.3 provides the source code for the step detection algorithm. Section A.4 provides the source code for the Wi-Fi scan activity to collect fingerprints.

Section A.5 shows the source code for the main Activity in the *LearnLoc* Android application, and Section A.6 to A.9 provide the source code for the three strategies that use the KNN, ANN and Linear Regression algorithms.

## A.1 CompassSensorWatcher.Java

```
/**
 * Class to get Orientation (Azimuth) using Sensor Fusion and Classic Approach
 * @author Viney Ugave (vinzzz@rams.colostate.edu)
 * Improved and Customized from  https://code.google.com/p/wificompass/
 */
package com.colostate.mecs.vinzzz.IL.location;

import com.colostate.mecs.vinzzz.IL.helper.ToolBox;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorListener;
import android.hardware.SensorManager;
import android.util.Log;


public class CompassSensorWatcher implements SensorEventListener {

       private static final String TAG = "CompassSensorWatcher";
```

```
	protected SensorManager sensorManager;

	protected Sensor compass;

	protected Sensor accelerometer;

	protected Sensor rotationVector;

	protected Context context;

	float[] inR = new float[16];

	float[] I = new float[16];

	float[] gravity = new float[3];

	float[] geomag = new float[3];

	float[] rotVec = new float[3];


	float[] orientVals = new float[3];

	float azimuth = 0;

	float angle = 0;

//	String azimuthText = "";

	int minX = 0, minY = 0, maxX = 0, maxY = 0, centerX = 0, centerY = 0, width =
0, height = 0;

	float l = 0.3f;

	protected CompassListener listener;

	protected float lastAzimuth = 0f;



	public	CompassSensorWatcher(Context	context,CompassListener	cl,float
lowpassFilter) {
		Log.d(TAG, "Instantiated CompassSensorWatcher");
		this.context = context;
		this.listener=cl;
		this.l=lowpassFilter;

		sensorManager	=	(SensorManager)
context.getSystemService(Context.SENSOR_SERVICE);
		compass = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
		accelerometer	=
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
		rotationVector	=
sensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
//
	Log.d(TAG,sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER).toString() );
		try {
			sensorManager.registerListener(this,	compass,
SensorManager.SENSOR_DELAY_UI);
			sensorManager.registerListener(this,	accelerometer,
SensorManager.SENSOR_DELAY_UI);
```

```
                  sensorManager.registerListener(this,                rotationVector,
SensorManager.SENSOR_DELAY_UI);

                  Log.d(TAG, "Registered Sensor Listeners");
            } catch (Exception e) {
                  Log.e("could not register listener", e.toString());
            }
      }


      /*
       * (non-Javadoc)
       *
       *                                                                                @see
android.hardware.SensorEventListener#onAccuracyChanged(android.hardware.Sensor, int)
       */
      @Override
      public void onAccuracyChanged(Sensor sensor, int accuracy) {
//            Log.d(TAG, "Magnetometer Accuracy: "+accuracy);
      }

      /*
       * (non-Javadoc)
       *
       *                                                                                @see
android.hardware.SensorEventListener#onSensorChanged(android.hardware.SensorEvent)
       */
      @SuppressWarnings("deprecation")
      @Override
      public void onSensorChanged(SensorEvent event) {

            // Logger.d("sensor changed "+event);
            // we use TYPE_MAGNETIC_FIELD to get changes in the direction, but use
SensorManager to get directions
            if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE)
                  return;

            // Gets the value of the sensor that has been changed
            switch (event.sensor.getType()) {
            case Sensor.TYPE_ACCELEROMETER:
                  gravity = event.values.clone();
//                Log.d(TAG, "Accelerometer onSensorChanged() ");
//                Log.d(TAG,String.valueOf(gravity[0]) );
                  break;
            case Sensor.TYPE_MAGNETIC_FIELD:
                  geomag = event.values.clone();
//                Log.d(TAG, "Magnetic Field onSensorChanged() ");
//                Log.d(TAG,"Magnetic : "+String.valueOf(geomag[0]));
                  break;
            case Sensor.TYPE_ROTATION_VECTOR:
                  rotVec=event.values.clone();
                  break;
            }
//
            //Classic = Acc+Mag
            // If gravity and geomag have values then find rotation matrix
            if (gravity != null && geomag != null) {

                  // checks that the rotation matrix is found
                  boolean success = SensorManager.getRotationMatrix(inR, I, gravity,
geomag);
                  if (success) {
                        SensorManager.getOrientation(inR, orientVals);
```

```
                        angle = (float) ToolBox.normalizeAngle(orientVals[0]);
                        azimuth = (float) Math.toDegrees(angle);
//                      Log.d(TAG,String.valueOf(azimuth) );
                        lowPassFilter();

                        angle=(float) Math.toRadians(azimuth);

//                      azimuthText   =   getAzimuthLetter(azimuth)   +   "   "   +
Integer.toString((int) azimuth) + "¬∞";

                        if(listener!=null){

      listener.onCompassChanged(azimuth,angle,getAzimuthLetter(azimuth));
                        }
                  }
            }//Acc+Mag


            //Sensor Fusion
            if (rotVec != null) {
                  SensorManager.getRotationMatrixFromVector(inR,rotVec);
                  SensorManager.getOrientation(inR, orientVals);

                  angle = (float) ToolBox.normalizeAngle(orientVals[0]);
                  azimuth = (float) Math.toDegrees(angle);
                  angle=(float) Math.toRadians(azimuth);

                  if(listener!=null){

      listener.onCompassChanged(azimuth,angle,getAzimuthLetter(azimuth));
                  }


            }//Sensor Fusion


      }


      public void stop(){
            try {
                  sensorManager.unregisterListener(this);
            } catch (Exception e) {
                  Log.w("could not unregister listener", e);
            }
      }

      public String getAzimuthLetter(float azimuth) {
            String letter = "";
            int a = (int) azimuth;

            if (a < 23 || a >= 315) {
                  letter = "N";
            } else if (a < 45 + 23) {
                  letter = "NO";
            } else if (a < 90 + 23) {
                  letter = "O";
            } else if (a < 135 + 23) {
                  letter = "SO";
            } else if (a < (180 + 23)) {
                  letter = "S";
            } else if (a < (225 + 23)) {
```

```
                        letter = "SW";
                } else if (a < (270 + 23)) {
                        letter = "W";
                } else {
                        letter = "NW";
                }

                return letter;
        }

        protected void lowPassFilter() {
                // lowpass filter
                float dazimuth = azimuth -lastAzimuth;

//              // if the angle changes more than 180¬∞, we want to change direction and
follow the shorter angle
                if (dazimuth > 180) {
                        // change to range -180 to 0
                        dazimuth = (float) (dazimuth - 360f);
                } else if (dazimuth < -180) {
                        // change to range 0 to 180
                        dazimuth = (float) (360f + dazimuth);
                }
                // lowpass filter
                azimuth = lastAzimuth+ dazimuth*l; //maybe use one in the book

                azimuth%=360;

                if(azimuth<0){
                        azimuth+=360;
                }

                lastAzimuth=azimuth;

//              lastAzimuth=azimuth=ToolBox.lowpassFilter(lastAzimuth, azimuth, l);

//              oldValue + filter * (newValue - oldValue);

        }



}
```

## A.2 StepDetection.Java

```
package com.colostate.mecs.vinzzz.IL.location;

import java.util.Timer;
import java.util.TimerTask;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;


/**
 *
```

```
 * This class is fed with data from the Accelerometer and Compass sensors. If a step is detected on the acc data it calls the trigger function on its interface StepTrigger, with the given direction.
 * Usage: Create an object: stepDetection = new StepDetection(this, this, a, peak, step_timeout_ms);
 * Adopted from WiFi Compass (https://code.google.com/p/wificompass/ )
 */
public class StepDetection implements CompassListener {
	public static final long INTERVAL_MS = 1000 / 30;

	// Hold an interface to notify the outside world of detected steps
	/**
	 * @uml.property name="st"
	 * @uml.associationEnd
	 */
	protected StepTrigger st;

	// Context needed to get access to sensor service
	protected Context context;

	protected static SensorManager sm; // Holds references to the SensorManager

	// List<Sensor> lSensor; // List of all sensors

	protected float lastComp;

	protected Timer timer;

	protected StepDetector detector;

	protected Sensor accelerometer;

	/**
	 * Handles sensor events. Updates the sensor
	 */
	public SensorEventListener mySensorEventListener = new SensorEventListener() {
		@Override
		public void onAccuracyChanged(Sensor sensor, int accuracy) {
			// Auto-generated method stub
		}

		@Override
		public void onSensorChanged(SensorEvent event) {
			switch (event.sensor.getType()) {
			case Sensor.TYPE_ACCELEROMETER:
				st.onAccelerometerDataReceived(System.currentTimeMillis(), event.values[0], event.values[1], event.values[2]);
				// just update the oldest z value
				detector.addSensorValues(System.currentTimeMillis(), event.values);
				break;

			default:
			}// switch (event.sensor.getType())
		}
	};

	public StepDetection(Context context, StepTrigger st, double a, double peak, int step_timeout_ms) {
		this.context = context;
		this.st = st;

		this.detector = new StepDetector(a, peak, step_timeout_ms);
```

```
	}

	public void load() {
		load(SensorManager.SENSOR_DELAY_FASTEST);
		CompassMonitor.registerListener(context, this);
	}

	/**
	 * Enable step detection
	 */
	public void load(int sensorDelay) {

		if (timer == null) {
			// Sensors
			sm                          =                          (SensorManager)
context.getSystemService(Context.SENSOR_SERVICE);

			accelerometer = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

			sm.registerListener(mySensorEventListener,           accelerometer,
sensorDelay);

			// Register timer
			timer = new Timer("UpdateData", false);
			TimerTask task = new TimerTask() {

				@Override
				public void run() {
					updateData();
				}
			};
			timer.schedule(task, 0, INTERVAL_MS);
		}
	}

	/**
	 * Disable step detection
	 */
	public void unload() {
		if (timer != null) {
			timer.cancel();
			timer.purge();
			timer = null;
			sm.unregisterListener(mySensorEventListener);
		}
	}

	/**
	 * This is called every INTERVAL_MS ms from the TimerTask.
	 */
	protected synchronized void updateData() {
		// Get current time for time stamps
		long now_ms = System.currentTimeMillis();

		st.onTimerElapsed(now_ms,     detector.getLastAcc(),     new     double[]
{lastComp});

		// Check if a step is detected upon data
		if (detector.checkForStep()) {
			// Call algorithm for navigation/updating position
			st.onStepDetected(now_ms, lastComp);

		}
```

```
	}

	/**
	 * @return
	 * @uml.property name="a"
	 */
	public double getA() {
		return detector.getA();
	}

	/**
	 * @return
	 * @uml.property name="peak"
	 */
	public double getPeak() {
		return detector.getPeak();
	}

	/**
	 * @return
	 * @uml.property name="step_timeout_ms"
	 */
	public int getStep_timeout_ms() {
		return detector.getStepTimeoutMS();
	}

	/**
	 * @param a
	 * @uml.property name="a"
	 */
	public void setA(double a) {
		detector.setA(a);
	}

	/**
	 * @param peak
	 * @uml.property name="peak"
	 */
	public void setPeak(double peak) {
		detector.setPeak(peak);
	}

	/**
	 * @param stepTimeoutMs
	 * @uml.property name="step_timeout_ms"
	 */
	public void setStep_timeout_ms(int stepTimeoutMs) {
		detector.setStepTimeoutMS(stepTimeoutMs);
	}

	/* (non-Javadoc)
	 *	@see	at.fhstp.wificompass.CompassListener#onCompassChanged(float,
java.lang.String)
	 */
	@Override
	public void onCompassChanged(float azimuth, float angle, String direction) {
		st.onCompassDataReceived(System.currentTimeMillis(), azimuth, 0, 0);
		this.lastComp=azimuth;
	}
}
```

### A.3 StepDetector.Java

```
package com.colostate.mecs.vinzzz.IL.location;

import android.util.Log;

/**
 * Class used by StedDetection for detecting Steps
 * @author Viney Ugave(vinzzz@rams.colostate.edu)
 *
 */
public class StepDetector {
	protected static final int vhSize = 6;

	protected double[] values_history = new double[vhSize];

	protected int vhPointer = 0;

	public static final int WINDOW = 5;

	private static final String TAG = "StepDetector";


	protected double a;

	protected double peak;

	protected int stepTimeoutMS;

	protected long lastStepTs = 0;

	// last acc is low pass filtered
	protected double[] lastAcc = new double[] {0.0, 0.0, 0.0};


	protected int round = 0;


	protected boolean logSteps=true;


	protected long lastUpdateTimestamp=0;


	protected long lastSecond=0;
	protected int valuesPerSecond=0;

	public StepDetector( double a, double peak, int step_timeout_ms) {
		this.a = a;
		this.peak = peak;
		this.stepTimeoutMS = step_timeout_ms;
	}

	public synchronized void addSensorValues(long timestamp,float values[]) {
		// simple lowpass filter
		lastAcc[0]+=a*(values[0]-lastAcc[0]);
		lastAcc[1]+=a*(values[1]-lastAcc[1]);
		lastAcc[2]+=a*(values[2]-lastAcc[2]);
		lastUpdateTimestamp=timestamp;
		if(timestamp<lastSecond+1000){
			valuesPerSecond++;
		}else {
```

```
                    if(true/*logSteps&&Logger.isTraceEnabled()*/)
//                          Log.v(TAG,valuesPerSecond+" sensor values received in the
last second");
                    lastSecond=timestamp;
                    valuesPerSecond=0;
             }
      }

      protected double lowpassFilter(double oldValue, double newValue) {
             return oldValue + a * (newValue - oldValue);
      }

      /**
       * This is called every INTERVAL_MS ms from the TimerTask.
       */
      public synchronized boolean checkForStep() {
             boolean ret = false;

             // Get current time for time stamps


             addData(lastAcc[2]);//adding z values

             // Check if a step is detected upon data
             if ((lastUpdateTimestamp - lastStepTs) > stepTimeoutMS) {

                    for (int t = 1; t <= WINDOW; t++) {
                          if ((values_history[(vhPointer - 1 - t + vhSize + vhSize) %
vhSize] - values_history[(vhPointer - 1 + vhSize) % vhSize] > peak)) {

                                 if(logSteps)
//                                       Log.v(TAG,"Detected step with t = " + t + ",
diff = " + peak + " < "
//                                                 + (values_history[(vhPointer - 1 - t +
vhSize + vhSize) % vhSize] - values_history[(vhPointer - 1 + vhSize) % vhSize]));
                                 // Set latest detected step to "now"
                                 lastStepTs = lastUpdateTimestamp;
                                 // Call algorithm for navigation/updating position
                                 // st.trigger(now_ms, lCompass);
//                                 Logger.i( "Detected step  in  round = " + round + " @
" + now_ms);
                                 ret = true;
                                 break;
                          }
                    }

             }
             round++;
             return ret;
      }

      protected void addData(double value) {
             values_history[vhPointer % vhSize] = value;
             vhPointer++;
             vhPointer = vhPointer % vhSize;
      }

      /**
       * @return the a
       */
      public double getA() {
             return a;
      }
```

```
/**
 * @param a the a to set
 */
public void setA(double a) {
	this.a = a;
}

/**
 * @return the peak
 */
public double getPeak() {
	return peak;
}

/**
 * @param peak the peak to set
 */
public void setPeak(double peak) {
	this.peak = peak;
}

/**
 * @return the stepTimeoutMS
 */
public int getStepTimeoutMS() {
	return stepTimeoutMS;
}

/**
 * @param stepTimeoutMS the stepTimeoutMS to set
 */
public void setStepTimeoutMS(int stepTimeoutMS) {
	this.stepTimeoutMS = stepTimeoutMS;
}

/**
 * @return the lastStepTs
 */
public long getLastStepTs() {
	return lastStepTs;
}

/**
 * @return the lastAcc
 */
public double[] getLastAcc() {
	return lastAcc;
}

/**
 * @return the round
 */
public int getRound() {
	return round;
}

/**
 * @return the logSteps
 */
public boolean isLogSteps() {
	return logSteps;
}
```

```
	/**
	 * @param logSteps the logSteps to set
	 */
	public void setLogSteps(boolean logSteps) {
		this.logSteps = logSteps;
	}

	/**
	 * @return the valuesPerSecond
	 */
	public int getValuesPerSecond() {
		return valuesPerSecond;
	}

}
```

## A.4 WifiScanner.Java

```
package com.colostate.mecs.vinzzz.IL.wifi;

import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import com.colostate.mecs.vinzzz.IL.exceptions.WifiException;
import com.colostate.mecs.vinzzz.IL.location.LocationServiceFactory;
import com.colostate.mecs.vinzzz.IL.model.BssidResult;
import com.colostate.mecs.vinzzz.IL.model.Location;
import com.colostate.mecs.vinzzz.IL.model.WifiScanResult;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
import android.util.Log;
/**
 * Class for WiFi Scan
 * @author viney
 *
 */
public class WifiScanner {

	private static final String TAG = "WifiScanner";
	protected static Vector<BroadcastReceiver> receivers = null;

	public static BroadcastReceiver startScan(Context ctx,
			WifiResultCallback callback) throws WifiException {
		if (receivers == null) {
			receivers = new Vector<BroadcastReceiver>();
		}

		BroadcastReceiver wifiScanReceiver = null;
		final Context context = ctx;
		final WifiResultCallback resultCallback = callback;

		WifiManager wm = (WifiManager) context
```

```
                        .getSystemService(Context.WIFI_SERVICE);

                // Logger.d( "trying to start a wifi scan");

                if (!wm.isWifiEnabled()) {

                        Log.d(TAG, "WiFi is disabled, trying to enable it");
                        wm.setWifiEnabled(true);
                        try {
                                Thread.sleep(2500);
                        } catch (InterruptedException e) {

                        }

                        if (wm.isWifiEnabled()) {

                                Log.d(TAG, "WiFi could not be enabled");
                        } else {
//                              Log.d(TAG, "WiFI enabled successfully");

                        }
                }

                if (!wm.isWifiEnabled()) {

                        throw new WifiException(
                                        "WiFi could not be enabled, please enable it!");
                }

//              Log.d(TAG, "WiFi is enabled");

                IntentFilter i = new IntentFilter();
                i.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);

                wifiScanReceiver = new BroadcastReceiver() {
                        public void onReceive(Context c, Intent i) {

//                              Log.d(TAG, "received ScanResult");
                                // Code to execute when SCAN_RESULTS_AVAILABLE_ACTION event
                                // occurs
                                WifiManager w = (WifiManager) c
                                                .getSystemService(Context.WIFI_SERVICE);
                                List<ScanResult>  l  =  w.getScanResults();  //  Returns  a
<list> of

                // scanResults
                                context.unregisterReceiver(this);

                                if (receivers.contains(this))
                                        receivers.remove(this);

                                Location curLocation = LocationServiceFactory
                                                .getLocationService().getLocation();


//                              for (Iterator<ScanResult> it = l.iterator(); it.hasNext();)
{
//                                      ScanResult sr = it.next();
//                                      System.out.println(sr.BSSID + " " + sr.SSID + " "
//                                                      +  sr.level  +  "dBm  "  +  sr.frequency  +
"MHz "
//                                                      + sr.capabilities + "\n");
//                              }
```

```
                        WifiScanResult wifiScanResult = new WifiScanResult(
                                    new Date().getTime(), curLocation, null);

                        for (ScanResult sr : l) {

                              BssidResult    bssid    =    new    BssidResult(sr,
wifiScanResult,curLocation);
                              wifiScanResult.addTempBssid(bssid);

                        }

                        if (resultCallback != null)
                              resultCallback.onScanFinished(wifiScanResult);

                  }
            };

            context.registerReceiver(wifiScanReceiver, i);

            receivers.add(wifiScanReceiver);

//          Log.d(TAG, "starting Wifi Scan");
            // Now you can call this and it should execute the broadcastReceiver's
            // onReceive()
            wm.startScan();

            return wifiScanReceiver;

      }

      public static void stopScanning(Context ctx) {
            // we don't stop scanning, we just unregister all Broadcast Intent
            // Receivers

            if (receivers != null)
                  for (BroadcastReceiver rcvr : receivers) {
                        try {
                              ctx.unregisterReceiver(rcvr);
                        } catch (Exception e) {
                        }
                  }
            receivers = new Vector<BroadcastReceiver>();

      }

      public static void stopScanner(Context ctx, BroadcastReceiver receiver) {
            try {

                  ctx.unregisterReceiver(receiver);
            } catch (Exception ex) {
                  // Logger.e("could not unregister receiver",ex);
            }
            if (receivers.contains(receiver)) {
                  receivers.remove(receiver);
            }

      }

}
```

## A.5 ProjectActivity.Java

```
package com.colostate.mecs.vinzzz.IL.IndoorNavTest;

import java.text.ParseException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.Vector;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.database.SQLException;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.PointF;
import android.graphics.drawable.AnimationDrawable;
import android.hardware.SensorManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.text.InputType;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.Toast;

import com.colostate.ML.KNN.KNN;
import com.colostate.ML.LinearRegression.Regression;
import com.colostate.ML.LinearRegression.Regression2;
import com.colostate.ML.NeuralNetwork.NeuralNetwork;
import com.colostate.ML.SVM.SupportVectorMachine;
import com.colostate.mecs.vinzzz.IL.IndoorNavTest.R.menu;
import com.colostate.mecs.vinzzz.IL.database.BssidResultDataSource;
import com.colostate.mecs.vinzzz.IL.database.BssidResultHelper;
import com.colostate.mecs.vinzzz.IL.database.LocationCopyDataSource;
import com.colostate.mecs.vinzzz.IL.database.ProjectListDataSource;
import com.colostate.mecs.vinzzz.IL.exceptions.WifiException;
import com.colostate.mecs.vinzzz.IL.location.LocationChangeListener;
import com.colostate.mecs.vinzzz.IL.location.LocationServiceFactory;
```

```
import com.colostate.mecs.vinzzz.IL.location.SensorFusion;
import com.colostate.mecs.vinzzz.IL.location.StepDetectionProvider;
import com.colostate.mecs.vinzzz.IL.model.BssidResult;
import com.colostate.mecs.vinzzz.IL.model.Location;
import com.colostate.mecs.vinzzz.IL.model.LocationCopy;
import com.colostate.mecs.vinzzz.IL.model.ProjectList;
import com.colostate.mecs.vinzzz.IL.model.TrainingData;
import com.colostate.mecs.vinzzz.IL.model.WifiScanResult;
import com.colostate.mecs.vinzzz.IL.view.MeasuringPointDrawable;
import com.colostate.mecs.vinzzz.IL.view.MultiTouchDrawable;
import com.colostate.mecs.vinzzz.IL.view.MultiTouchView;
import com.colostate.mecs.vinzzz.IL.view.OkCallback;
import com.colostate.mecs.vinzzz.IL.view.RefreshableView;
import com.colostate.mecs.vinzzz.IL.view.ScaleLineDrawable;
import com.colostate.mecs.vinzzz.IL.view.SiteMapDrawable;
import com.colostate.mecs.vinzzz.IL.view.UserDrawable;
import com.colostate.mecs.vinzzz.IL.wifi.WifiResultCallback;
import com.colostate.mecs.vinzzz.IL.wifi.WifiScanner;
/**
 * Main Test Project Activity
 * @author viney
 *
 */
public class ProjectActivity extends Activity implements RefreshableView,
             LocationChangeListener, WifiResultCallback, OnClickListener {

       private static final String TAG = "ProjectActivity";

       protected static String pID = null;
       protected static String mapScaleX = null;
       protected static String mapScaleY = null;
       protected static String backgroundImagePath = null;
       protected static String mlTechnique = null;
       protected static String tProjId = null;

       SensorManager sensorManager;
       SensorFusion sensorFusion;

       protected final Context context = this;
       protected MultiTouchView multiTouchView;
       protected ProjectList site;
       protected SiteMapDrawable map;
       protected UserDrawable user;
       protected Vector<Location> stepsLoc;

       private LocationCopy locationCopy;
       private LocationCopyDataSource datasourceLocation;

       // ML

       // Training
       private LocationCopyDataSource datasourceTrainingLocation;
       private List<String> uniqueBssids;
       private int xyTrainingPoints;

       // Neural Network
       private NeuralNetwork nn;
       // SVM
       private SupportVectorMachine svm;
       // KNN
       private KNN knn;
       // Linear regression
       private Regression reg;
```

```
private Regression2 reg2;

// ML thresholds
private float mlMaxThreshold;
private float mlMinThreshold;
// for step pred algo
int counterFirstStep = 0;
double lastPredX;
double lastPredY;

private BssidResult BssidResult;
private BssidResultDataSource datasourceBssidResult;
private BssidResultDataSource datasourceTrainBssidResult;
public Cursor bssidsCursor;
private List<BssidResult> trainingData;
private List<TrainingData> tData;

protected StepDetectionProvider stepDetectionProvider = null;
protected boolean walkingAndScanning = false;

// Wifi
public static final String SCAN_INTERVAL = "scan_interval";
protected int schedulerTime = 2;
protected final ScheduledExecutorService scheduler = Executors
            .newScheduledThreadPool(1);
protected BroadcastReceiver wifiBroadcastReceiver;
protected boolean ignoreWifiResults = false;
protected Runnable wifiRunnable;
protected ScheduledFuture<?> scheduledTask = null;
protected ArrayList<WifiScanResult> unsavedScanResults;

protected Handler messageHandler;
protected static final int MESSAGE_REFRESH = 1, MESSAGE_START_WIFISCAN = 2;
private static final int DIALOG_SET_THRESHOLDS = 0;
private static final int DIALOG_SET_WIFI_SCAN_RATE = 1;

private BssidResultHelper database;

// Scaler to measure accuracy
protected ScaleLineDrawable scaler = null;
protected float scalerDistance;

ProgressDialog progressBar;

@Override
protected void onCreate(Bundle savedInstanceState) {

       multiTouchView = new MultiTouchView(this);

       this.setContentView(R.layout.activity_project);
       super.onCreate(savedInstanceState);
       Bundle extras = getIntent().getExtras();

       // Show progress dialog

       // getting all pre set values from Project database
       site = new ProjectList();
       site.setId(Long.parseLong(extras.getString("siteID")));
       site.setGridSpacingX(Float.parseFloat(extras.getString("mapScaleX")));
       site.setGridSpacingY(Float.parseFloat(extras.getString("mapScaleY")));
       site.setNorth(Float.parseFloat(extras.getString("mapNorth")));
       backgroundImagePath = extras.getString("backgroundImagePath");
       mlTechnique = extras.getString("mlTechnique");
```

```
            tProjId = extras.getString("tProjId");

            stepsLoc = new Vector<Location>();

            MultiTouchDrawable.setGridSpacing(site.getGridSpacingX(),
                        site.getGridSpacingY());
            map = new SiteMapDrawable(this, this);
            map.setAngleAdjustment(site.getNorth());

            user = new UserDrawable(this, map);
            user.setNorth(site.getNorth());

            if (!(backgroundImagePath.equals(null) | backgroundImagePath
                        .equals("null"))) {
                  setBackgroundImage(backgroundImagePath);

            } else {
                  Log.d(TAG, "No Background");
                  // For bigger blank screen
                  //     Log.d(TAG,     "Width     :     "+map.getWidth()+"     Height
:"+map.getHeight()
                  // );
                  site.setSize(map.getWidth() * 6, map.getHeight() * 4);
                  map.setSize(map.getWidth() * 6, map.getHeight() * 4);
                  user.setRelativePosition(map.getWidth() / 2, map.getHeight() / 2);

            }

            // for (WifiScanResult wsr : site.getScanResults()) {
            // new MeasuringPointDrawable(this, map, wsr);
            // }

            LocationServiceFactory.getLocationService().setRelativeNorth(
                        site.getNorth());
            LocationServiceFactory.getLocationService().setGridSpacing(
                        site.getGridSpacingX(), site.getGridSpacingY());
            stepDetectionProvider = new StepDetectionProvider(this);
            stepDetectionProvider.setLocationChangeListener(this);

            // Message Handler implementation
            messageHandler = new Handler() {
                  @Override
                  public void handleMessage(Message msg) {
                        switch (msg.what) {
                        case MESSAGE_REFRESH:
                              /* Refresh UI */
                              if (multiTouchView != null)
                                    multiTouchView.invalidate();
                              break;

                        case MESSAGE_START_WIFISCAN:
                              // start a wifiscan
                              startWifiBackgroundScan();
                              break;

                        }
                  }
            };

            wifiRunnable = new Runnable() {

                  @Override
                  public void run() {
```

```
                        // try {
                        // Thread.sleep(10000);
                        // } catch (InterruptedException e) {
                        // Log.d(TAG, "Error in Wifi Thread");
                        // e.printStackTrace();
                        // }
                        messageHandler.sendEmptyMessage(MESSAGE_START_WIFISCAN);
                }

        };

        unsavedScanResults = new ArrayList<WifiScanResult>();

        // schedulerTime = this.getPreferences(Activity.MODE_PRIVATE).getInt(
        // SCAN_INTERVAL, schedulerTime);

        Log.d(TAG, "On Creat'd");

        init();
        Toast.makeText(context, "Please be patient training the ML Algo....",
                        Toast.LENGTH_LONG).show();

        Thread T = new Thread(new Runnable() {
                public void run() {
                        prepareforML();
                }
        });
        T.start();
        try {
                T.join();
                Toast.makeText(context,
                                "Training Complete.. You may now use the app !! :) ",
                                Toast.LENGTH_LONG).show();
        } catch (InterruptedException e) {
                e.printStackTrace();
        }

}

/**
 * Func that will prepare training data and call select ML Technique
 */
private void prepareforML() {
        getTrainingData();
        prepareTrainingData();

}

/**
 * Func that will prepare the training data(Identify Unique Bssids and list
 * them)
 */
@SuppressWarnings("unchecked")
private void prepareTrainingData() {
        if (trainingData != null) {
                // Get all BSSIDs
                List<String> unsortedBssids = new ArrayList<String>();
                for (int i = 0; i < trainingData.size(); i++) {
                        unsortedBssids.add(trainingData.get(i).getBssid());
                }

                // Get Unique BSSIDS
                Set sortedBssids = new HashSet<String>();
```

```
                        sortedBssids.addAll(unsortedBssids);

                        uniqueBssids = new ArrayList<String>();
                        Iterator iterator = sortedBssids.iterator();
                        while (iterator.hasNext()) {
                               String tempBssid = iterator.next().toString();
                               int j = 0;
                               for (int i = 0; i < trainingData.size(); i++) {
                                      if (tempBssid.equals(trainingData.get(i).getBssid()
                                                    .toString())) {
                                             j++;
                                      }
                               }
                               /* Set bssid threshold as necessary */if (j > 0) {
                                      uniqueBssids.add(tempBssid);
                               }
                        }
                        Log.d(TAG,
                                      "Got all unique BSSIDS. Size = "
                                                    + String.valueOf(uniqueBssids.size()));
                        sortedBssids = null;
                        unsortedBssids = null;

                        // Get quadruple(x,y,level,bssid) of all Unique BSSIDs

                        tData = null;
                        tData = new ArrayList<TrainingData>();
                        Iterator iterator1 = uniqueBssids.iterator();
                        while (iterator1.hasNext()) {
                               String tempBssid1 = iterator1.next().toString();
                               for (int i = 0; i < trainingData.size(); i++) {
                                      if (tempBssid1.equals(trainingData.get(i).getBssid()
                                                    .toString())) {
                                             tData.add(new
TrainingData(trainingData.get(i).getX(),
                                                           trainingData.get(i).getY(),
trainingData.get(i)
                                                                        .getLevel(),
trainingData.get(i)
                                                                        .getBssid()));
                                      }
                               }
                        }

                        Log.d(TAG, "No. of Training Samples = " + tData.size());
                        trainingData = null;

                        // Get unique X,Y

                        Set xy = new HashSet<TrainingData>();
                        for (int i = 0; i < tData.size(); i++) {
                               xy.add(new TrainingData(tData.get(i).getX(), tData.get(i)
                                             .getY(), tData.get(i).getLevel(), tData.get(i)
                                             .getBssid()));
                        }

                        Log.d(TAG, "No. of training wifi scans(xy) = " + xy.size());
                        setXYTrainingPoints(xy.size());

                        // Create a training 2D array for sorted final training array
                        double[][] X = new double[getXYTrainingPoints()][];
                        int n = 0;
```

```
                    // Group BSSID based on XY
                    Iterator it = xy.iterator();
                    while (it.hasNext()) {

                          TrainingData temp = (TrainingData) it.next();
                          List<TrainingData> setXY = new ArrayList<TrainingData>();//
Set

                                      // of

                                      // single

                                      // XY

                          for (int i = 0; i < tData.size(); i++) {
                                if (temp.getX() == tData.get(i).getX()
                                            &&  temp.getY()  ==  tData.get(i).getY())
{
                                      setXY.add(tData.get(i));
                                }
                          }

                          // See what dont match and give def value
                          Iterator it1 = uniqueBssids.iterator();
                          while (it1.hasNext()) {
                                String tempUBssid = it1.next().toString();
                                int T = 0;
                                int F = 0;
                                for (int j = 0; j < setXY.size(); j++) {
                                      if
(setXY.get(j).getBssid().equals(tempUBssid)) {
                                            T++;
                                      } else {
                                            F++;
                                      }
                                }
                                if (T == 1) {
                                      // This means BSSID present

                                } else {
                                      if (T == 0 && F > 0) {
                                            setXY.add(new
TrainingData(setXY.get(0).getX(),
                                                        setXY.get(0).getY(),     0,
tempUBssid));
                                            //   Log.d(TAG,   "Adding   for   "   +
tempUBssid);
                                      }

                                }

                          }
                          // Log.d(TAG, "X,Y= (" + setXY.get(0).getX() + ","
                          // + setXY.get(0).getY() + ") Size= " + setXY.size());

                          //  then  sort  according  to  uniqueBSSIDS  and  put  in  an
Training 2D
                          // Array
                          X[n] = sortTrainingBSSIDs(setXY);

                          n++;
                    }
```

```
                n = 0;

                Log.d(TAG, "Got all training data in X[][]");
                trainML(X);

            } else {
                Log.d(TAG, "Please load training data!");
            }

        }

        /**
         * Func to train the ML Algo from prepared data
         *
         * @param x
         */
        private void trainML(double[][] x) {
            // init data
            setMlMaxThreshold(120);
            setMlMinThreshold(40);

            if (mlTechnique.equals("NN")) {
                Log.d(TAG, "Training ML technique using Neural Networks");

                int inNodes = uniqueBssids.size();// Unique BSSIDS contains all
the
                                                                                //
bssids
                int hidNodes = 14;
                int outNodes = 2;

                nn = new NeuralNetwork(inNodes, hidNodes, outNodes);

                int maxEpochs = 4000;

                // nn.train(x, maxEpochs);
                // Log.d(TAG, "NN Training complete !!");
                //
                // Log.d(TAG, "Testing for Train set ");
                // // Test train set
                // double[][] xy = nn.test(x);
                // printMatrix(xy, "XYPredicted");
                // double error = nn.Accuracy(nn.getT(), xy);
                // System.out.println("Accuracy = " + error);

            } else if (mlTechnique.equals("SVM")) {
                Log.d(TAG, "Training ML technique using SVM Regression");

                int inNodes = uniqueBssids.size();// Unique BSSIDS contains all
the
                // bssids
                int outNodes = 2;

                svm = new SupportVectorMachine(3, 3, inNodes, outNodes, 3,
                            0.0000001f, false);

                svm.train(x);
                Log.d(TAG, "SVM Training complete !!");
                Log.d(TAG, "Testing for Train set ");
                float[] predict = svm.test(x, 1);
                for (int i = 0; i < x.length; i++) {
                      Log.d(TAG, "x = " + x[i][inNodes + 1]);
                }
```

```
                for (int i = 0; i < predict.length; i++) {
                        Log.d(TAG, "predict X= " + predict[i]);
                }
        } else if (mlTechnique.equals("KNN")) {
                Log.d(TAG, "Training ML technique using KNN Regression");

                int inNodes = uniqueBssids.size();// Unique BSSIDS contains all
the
                // bssids
                int outNodes = 2;
                int kNN = 2;

                knn = new KNN(inNodes, outNodes, kNN);

                knn.train(x);
                Log.d(TAG, "KNN Training complete !!");

                // Log.d(TAG, "Testing for Train set ");
                // double[][] xy =knn.test(x);
                // // Test train set
                // printMatrix(xy, "XYPredicted");

        } else if (mlTechnique.equals("Regression")) {
                int inNodes = uniqueBssids.size();// Unique BSSIDS contains all
the
                // bssids
                int outNodes = 1;

                reg = new Regression(inNodes, outNodes);
                // reg2= new Regression2(inNodes, outNodes);

                // reg.train(x);
                // reg2.train(x);
                Log.d(TAG, "Linear Regression Training complete !!");

                // Log.d(TAG, "Testing for Train set ");
                // double[][] xy = reg.test(x);
                // printMatrix(xy, "XPredicted");

        }
    }

    /**
     * Func to test the ML Algo from test data
     *
     * @param t
     */

    private double[][] testML(double[][] t) {

        if (mlTechnique.equals("NN")) {
                Log.d(TAG, "Testing ML technique for Neural Networks");
                return nn.test(t);

        } else if (mlTechnique.equals("SVM")) {
                Log.d(TAG, "Testing ML technique for SVM");
                // return svm.test(t);
                return null;
        } else if (mlTechnique.equals("KNN")) {
                Log.d(TAG, "Testing ML technique for KNN");
                return knn.test(t);
        } else {
                return null;
```

```
		}
	}

	private void setXYTrainingPoints(int size) {
		this.xyTrainingPoints = size;
	}

	private int getXYTrainingPoints() {
		return this.xyTrainingPoints;
	}

	/**
	 * Sort the BSSIDS according to Unique BSSIDS
	 */
	private double[] sortTrainingBSSIDs(List<TrainingData> data) {
		//
		// Iterator I = data.iterator();
		// while (I.hasNext()) {
		// TrainingData TD = (TrainingData) I.next();
		// Log.d(TAG,
		// "X= " + TD.getX() + " Y= " + TD.getY() + " BSSID= "
		// + TD.getBssid() + " Level= " + TD.getLevel());
		// }

		Iterator uniqueIt = uniqueBssids.iterator();
		/**
		 * tempRow contains sorted BSSID in the order: [db,db,db ......x,y]
		 */
		double[] tempRow = new double[uniqueBssids.size() + 2];
		int m = 0;
		while (uniqueIt.hasNext()) {

			String bssidTemp = uniqueIt.next().toString();
			for (int i = 0; i < data.size(); i++) {
				if (data.get(i).getBssid().equals(bssidTemp)) {
					tempRow[m] = data.get(i).getLevel();
				}
			}
			m++;
		}
		tempRow[m] = data.get(0).getX();
		tempRow[m + 1] = data.get(0).getY();

		return tempRow;
	}

	/**
	 * Get training data(bssids) from Content Provider
	 */
	@SuppressWarnings("deprecation")
	private void getTrainingData() {
		try {
			trainingData = null;
			String[] projection = { database.COLUMN_ID, database.COLUMN_X,
					database.COLUMN_Y, database.COLUMN_BSSID,
					database.COLUMN_SSID, database.COLUMN_CAPABILITIES,
					database.COLUMN_FREQUENCY, database.COLUMN_LEVEL };
			String uri = null;
			uri                                                                =
"content://com.colostate.mecs.vinzzz.IL.contentproviderbssid/bssids/"
					+ tProjId;
			bssidsCursor = this.managedQuery(Uri.parse(uri), projection, null,
					null, null);
```

```
                datasourceTrainBssidResult = new BssidResultDataSource(this);
                trainingData = datasourceTrainBssidResult
                              .getAllBssidResult(bssidsCursor);
                bssidsCursor.close();

                Log.d(TAG, "Got Training Data");

          } catch (ParseException e) {
                Log.e(TAG, "Error Getting Training Data");
                e.printStackTrace();
          }
    }

    /**
     * InitUI Method
     *
     * @return void
     */
    private void init() {
          // UI Stuff
          ((Button) findViewById(R.id.start_wifiscan_button))
                        .setOnClickListener(this);

          // Open DB - Steps/Location
          datasourceLocation = new LocationCopyDataSource(this, "location_"
                        + site.getId());
          try {
                datasourceLocation.open();
                Vector<PointF> oldSteps = new Vector<PointF>();
                List<LocationCopy> mLocations;
                mLocations = datasourceLocation.getAllLocations();
                boolean stepFlag = false;

                for (LocationCopy value : mLocations) {
                      if (value != null) {
                            oldSteps.add(new PointF(value.getX(), value.getY()));
                            stepFlag = true;
                      } else {
                            stepFlag = false;
                      }
                }
                if (stepFlag) {
                      Log.d(TAG, "No. of Steps =" + oldSteps.size());
                      map.setSteps(oldSteps);
                      LocationCopy temp = mLocations.get(mLocations.size() - 1);
                      user.setRelativePosition(temp.getX(),  temp.getY());//  Last
known

                      // location
                      Log.d(TAG, "Old Steps loaded successfully ");
                      stepFlag = false;
                }

          } catch (SQLException e) {
                Log.d(TAG, "Could not find location database table");
                e.printStackTrace();
          } catch (ParseException e) {
                Log.d(TAG, "Error retrieving old locations");
                e.printStackTrace();
          }

          // Open DB - Wifi
          datasourceBssidResult = new BssidResultDataSource(this, "wifi_"
```

```
                        + site.getId());
            try {
                  datasourceBssidResult.open();
                  List<BssidResult> mBssidResult;
                  mBssidResult = datasourceBssidResult.getAllBssidResult();

                  // Get unique X,Y
                  Set xy = new HashSet<TrainingData>();
                  for (int i = 0; i < mBssidResult.size(); i++) {
                        xy.add(new TrainingData(mBssidResult.get(i).getX(),
                                    mBssidResult.get(i).getY(),
mBssidResult.get(i)
                                                .getLevel(),
mBssidResult.get(i).getBssid()));
                  }

                  Log.d(TAG, "No. of TESTING wifi scans(xy) = " + xy.size());
                  // Get unique X,Y

                  boolean stepFlag = false;

                  float compX = 0;
                  float compY = 0;

                  // Dont show wifi results
                  for (BssidResult value : mBssidResult) {
                        // Add wifi drawables
                        float tempX = value.getX();
                        float tempY = value.getY();

                        if (tempX != compX || tempY != compY) {
                              WifiScanResult    wsr    =    new    WifiScanResult(new
Location(tempX,
                                          tempY));
                              new MeasuringPointDrawable(this, map, wsr);
                        }

                        compX = tempX;
                        compY = tempY;

                        // Log.d(TAG, "Old Wifi results loaded successfully ");

                  }// Dont show wifi results

            } catch (SQLException e) {
                  Log.d(TAG, "Could not find Wifi database table");
                  e.printStackTrace();
            } catch (ParseException e) {
                  Log.d(TAG, "Error retrieving old Wifi Results ");
                  e.printStackTrace();
            }

            // UI Stuff
            multiTouchView                    =                    ((MultiTouchView)
findViewById(R.id.project_site_resultview));
            multiTouchView.setRearrangable(false);

            multiTouchView.addDrawable(map);

            Log.d(TAG, "Init Complete");

      }
```

```
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.projec_activity, menu);
		return true;
	}

	@Override
	protected void onDestroy() {
		super.onDestroy();
	}

	@Override
	protected void onPause() {
		Log.d(TAG, "On Pause'd");
		super.onPause();
		multiTouchView.unloadImages();
		map.unload();

		setWalkingAndScanning(false, false);
		saveProject();
		bssidsCursor.close();
		datasourceLocation.close();
		datasourceBssidResult.close();

	}

	private void saveProject() {
		Log.d(TAG, "On saveProject");
		if (stepsLoc == null) {
		} else {
			for (Location locS : stepsLoc) {
				if (locS != null) {
					// Set arguments
					locationCopy = new LocationCopy();
					// locationCopy.setId(locS.getId());
					locationCopy.setX(locS.getX());
					locationCopy.setY(locS.getY());
					locationCopy.setAccurancy(locS.getAccurancy());
					locationCopy.setTimestamp(locS.getTimestamp());

					// Add to database
					try {
						locationCopy = datasourceLocation
								.createLocation(locationCopy);
					} catch (ParseException e) {
						e.printStackTrace();
						Log.e(TAG,
								"Error     adding     data     to
step/location database");
					}
				}
			}
		}
		Log.d(TAG, "Location database updated");
	}

	@Override
	protected void onResume() {
		super.onResume();
		Log.d(TAG, "setting context");
		multiTouchView.loadImages(this);
		map.load();
```

```
		try {
			datasourceLocation.open();
			datasourceBssidResult.open();
		} catch (SQLException e) {
			Log.d(TAG, "onResume error while opening DB");
			e.printStackTrace();
		}

	}

	protected void setBackgroundImage(String path) {

		try {
			Bitmap bmp = BitmapFactory.decodeFile(path);
			site.setBackgroundBitmap(bmp);
			map.setBackgroundImage(bmp);
			site.setSize(bmp.getWidth(), bmp.getHeight());
			map.setSize(bmp.getWidth(), bmp.getHeight());
			user.setRelativePosition(bmp.getWidth() / 2, bmp.getHeight() / 2);
			multiTouchView.invalidate();

		} catch (Exception e) {
			Log.e(TAG, "could not set background", e);
			Toast.makeText(
					context,

	getString(R.string.project_site_set_background_failed,
						e.getMessage()),
Toast.LENGTH_LONG).show();
		}
	}

	@Override
	public void invalidate() {
		if (multiTouchView != null) {
			multiTouchView.invalidate();
		}
	}

	protected void startWifiBackgroundScan() {

		try {
			// we first stop the old receiver, so we wont receive duplicate
			// results
			// stopWifiScan();

			if (wifiBroadcastReceiver != null) {
				// wifiBroadcastReceiver.
			}

			startWifiScan();
			// Toast.makeText(this, R.string.project_site_wifiscan_started,
			// Toast.LENGTH_SHORT).show();
		} catch (WifiException e) {
			Log.e(TAG, "could not start wifi scan", e);
			Toast.makeText(
					this,

	getString(R.string.project_site_wifiscan_start_failed,
						e.getMessage()),
Toast.LENGTH_LONG).show();
		}
```

```
}

/**
 * start the wifi scan
 */
protected void startWifiScan() throws WifiException {
	Log.d(TAG, "starting WiFi Scan");

	wifiBroadcastReceiver = WifiScanner.startScan(this, this);
	ignoreWifiResults = false;
}

/**
 * stop the wifi scan, if in progress
 */
protected void stopWifiScan() {
	hideWifiScanDialog();

	if (wifiBroadcastReceiver != null) {

		WifiScanner.stopScanner(this, wifiBroadcastReceiver);
		wifiBroadcastReceiver = null;

	}
	// stop scan
	// oh, wait, we can't stop the scan, it's asynchronous!
	// we just have to ignore the result!
	ignoreWifiResults = true;

}

/**
 * hide the wifi scan dialog if shown
 */
protected void hideWifiScanDialog() {
	// if (scanningImageView != null) {
	// ((AnimationDrawable) scanningImageView.getDrawable()).stop();
	// // scanningImageView = null;
	// }
	//
	// if (scanAlertDialog != null) {
	// scanAlertDialog.cancel();
	// // scanAlertDialog = null;
	// }
}

@Override
public void onLocationChange(Location loc) {
	// info from StepDetectionProvider, that the location changed.
	user.setRelativePosition(loc.getX(), loc.getY());
	map.addStep(new PointF(loc.getX(), loc.getY()));
	stepsLoc.add(loc);
	messageHandler.sendEmptyMessage(MESSAGE_REFRESH);
}

@SuppressWarnings("deprecation")
public boolean onOptionsItemSelected(MenuItem item) {

	switch (item.getItemId()) {

	case R.id.menuItemStart:
		Log.d(TAG, "Menu Start/Stop selected");
		if (item.getTitle().equals("Start")) {
```

```
                        item.setTitle("Stop");
                } else {
                        item.setTitle("Start");
                }
                setWalkingAndScanning(!walkingAndScanning, true);
                walkingAndScanning = !walkingAndScanning;
                return false;

        case R.id.menuItemStop:
                Log.d(TAG, "Menu Stop selected");
                return false;

        case R.id.menuItemCheckAccuracy:
                Log.d(TAG, "Menu Accuracy selected");
                scaleOfMap();
                return false;

        case R.id.menuItemDelLastStep:
                Log.d(TAG, "Menu Del Last Step selected");
                delLastStep();
                return false;

        case R.id.menuItemSetThreshold:
                Log.d(TAG, "Menu Threshold selected");
                showDialog(DIALOG_SET_THRESHOLDS);
                return false;

        case R.id.menuItemSetWifiRate:
                Log.d(TAG, "Menu Wifi Rate selected");
                showDialog(DIALOG_SET_WIFI_SCAN_RATE);
                return false;

        case R.id.menuItemDeleteData:
                stepsLoc = null;
                datasourceLocation.deleteAll();// Delete locations
                datasourceBssidResult.deleteAll();// Delete all Wifi data
                Log.d(TAG, "All data deleted");
                Toast.makeText(this,          "All          data          deleted",
Toast.LENGTH_SHORT).show();
                super.onBackPressed();
                return false;

        }
        return false;
    }

    /**
     * Function to delete last step
     */
    private void delLastStep() {
        stepsLoc.remove(stepsLoc.size() - 1);

        Vector<PointF> oldDSteps = new Vector<PointF>();
        boolean stepFlag = false;
        for (Location dloc : stepsLoc) {
            if (dloc != null) {
                oldDSteps.add(new PointF(dloc.getX(), dloc.getY()));
                stepFlag = true;
            } else {
                stepFlag = false;
            }
        }
        if (stepFlag) {
```

```
                        map.setSteps(oldDSteps);
                        LocationCopy temp = new LocationCopy(oldDSteps.lastElement().x,
                                    oldDSteps.lastElement().y);
                        user.setRelativePosition(temp.getX(), temp.getY());// Last known

                        // location
                        messageHandler.sendEmptyMessage(MESSAGE_REFRESH);
                        // Log.d(TAG, "Old Steps loaded successfully ");
                        stepFlag = false;
                }

        }

        protected void setWalkingAndScanning(boolean shouldRun, boolean ui) {
                if (!shouldRun) {
                        // stop!

                        if (stepDetectionProvider.isRunning())
                                stepDetectionProvider.stop();
                        if (scheduledTask != null) {
                                scheduledTask.cancel(false);
                                scheduledTask = null;
                        }
                        stopWifiScan();

                        // if(ui)
                        // ((Button)
                        //
findViewById(R.id.project_site_step_detect)).setText(R.string.project_site_start_step_
detect);
                        //
                        // persistScanResults(ui);

                } else {
                        // start
                        unsavedScanResults = new ArrayList<WifiScanResult>();

                        if (!stepDetectionProvider.isRunning()) {
                                stepDetectionProvider.start();
                        }

                        if (scheduledTask == null) {
                                scheduledTask                                                =
scheduler.scheduleWithFixedDelay(wifiRunnable,
                                                5,   (this.schedulerTime   <=   0   ?   1   :
this.schedulerTime),
                                                TimeUnit.SECONDS);
                        }
                        // if(ui)
                        // ((Button)
                        //
findViewById(R.id.project_site_step_detect)).setText(R.string.project_site_stop_step_d
etect);
                }
        }

        @Override
        public void onScanFinished(WifiScanResult wr) {
                hideWifiScanDialog();
                if (!ignoreWifiResults) {
                        try {

                                Log.d(TAG, "received a wifi scan result!");
```

```
                        ignoreWifiResults = true;

                        wr.setProjectLocation(site);

                        if (walkingAndScanning) {
                               unsavedScanResults.add(wr);
                        } else {

                        }

                        new MeasuringPointDrawable(this, map, wr);

                        // StringBuffer sb = new StringBuffer();
                        HashMap<String,   Integer>   ssids   =   new   HashMap<String,
Integer>();
                        // if(wr.getBssids()!=null)

                        List<TrainingData>          tempTest          =          new
ArrayList<TrainingData>();

                        for (BssidResult result : wr.getBssids()) {
                               ssids.put(
                                             result.getSsid(),
                                             (ssids.get(result.getSsid())  ==  null  ?
1 : ssids
                                                           .get(result.getSsid())    +
1));
                               // BssidResult result = it.next();
                               // Logger.d("ScanResult: " + result.toString());
                               // sb.append(result.toString());
                               // sb.append("\n");

                               // Add to WiFi Database
                               try {
                                      result = datasourceBssidResult
                                                    .createBssidResult(result);
                               } catch (ParseException e) {
                                      e.printStackTrace();
                                      Log.e(TAG,   "Error   adding   data   to   Wifi
database");
                               }

                               // Add to temp test list
                               tempTest.add(new          TrainingData(result.getX(),
result.getY(),
                                             result.getLevel(), result.getBssid()));

                               // System.out.println(result.getBssid() + " "
                               // + result.getSsid() + " " + result.getLevel()
                               // + "dBm " + result.getFrequency() + "MHz "
                               // + result.getCapabilities() + " x=" + result.getX()
                               // + " y=" + result.getY() + "\n");

                        }

                        // Test using tempTest for ML Algo and add step to stepsLoc

                        // Add def values
                        Iterator it2 = uniqueBssids.iterator();
                        while (it2.hasNext()) {
                               String tempUBssid = it2.next().toString();
                               int T = 0;
                               int F = 0;
```

```
                                    for (int j = 0; j < tempTest.size(); j++) {
                                          if
(tempTest.get(j).getBssid().equals(tempUBssid)) {
                                                T++;
                                          } else {
                                                F++;
                                          }
                                    }
                                    if (T == 1) {
                                          // This means BSSID present

                                    } else {
                                          if (T == 0 && F > 0) {
                                                tempTest.add(new
TrainingData(tempTest.get(0)
                                                            .getX(),
tempTest.get(0).getY(), 0,
                                                            tempUBssid));
                                          }

                                    }

                              }

                              // Sort List
                              double[][] T = new double[1][];
                              T[0] = sortTrainingBSSIDs(tempTest);

                              // for (int i = 0; i < T[0].length; i++) {
                              // System.out.println(T[0][i]);
                              // }
                              // Test for ML
                              double predictedXY[][] = testML(T);
                              if (predictedXY != null) {
                                    System.out.println("Predicted     X     =     "     +
predictedXY[0][0]
                                                + " Y = " + predictedXY[0][1]);

                                    Location lastLoc = stepsLoc.lastElement();

                                    // new logic for comparing last predicted value
                                    // to avoid convergence
                                    if (counterFirstStep == 0) {

                                    } else {
                                          double distanceLastPred = Math.sqrt(Math.pow(
                                                      (predictedXY[0][0]  -  lastPredX),
2)
                                                      +  Math.pow((predictedXY[0][1]  -
lastPredY), 2));
                                          if (distanceLastPred > 20) {
                                                double distance = Math
                                                            .sqrt(Math.pow(

      (predictedXY[0][0] - lastLoc.getX()),
                                                                                  2)
                                                                                  + Math.pow(

      (predictedXY[0][1] - lastLoc

            .getY()), 2));
```

```
                                                Log.d(TAG, "Distance of predicted = " +
distance);
                                                Toast.makeText(this,
                                                            "Distance of predicted = "
+ distance,

      Toast.LENGTH_SHORT).show();
                                                if (distance > getMlMinThreshold()
                                                            &&       distance       <
getMlMaxThreshold()) {
                                                      // Log.d(TAG,
                                                      //    "Min/Max    Threshold    =
"+getMlMinThreshold()+"/"+getMlMaxThreshold());
                                                      // Add predicted step
                                                      Log.d(TAG,    "Adding    predicted
step");
                                                      onLocationChange(new Location(
                                                                  (float)
predictedXY[0][0],
                                                                  (float)
predictedXY[0][1]));
                                                } else {

                                                }

                                          }
                                    }
                                    counterFirstStep = counterFirstStep + 1;
                                    lastPredX = predictedXY[0][0];
                                    lastPredY = predictedXY[0][1];
                              }

                              // UI stuff
                              user.bringToFront();
                              multiTouchView.invalidate();

                              // Toast.makeText(
                              // this,
                              // this.getString(R.string.project_site_wifiscan_finished,
                              // ssids.size(), wr.getBssids().size()),
                              // Toast.LENGTH_SHORT).show();

                        } catch (SQLException e) {
                              Log.e(TAG, "could not update wifiscanresult!", e);
                              Toast.makeText(
                                          this,

      this.getString(R.string.project_site_wifiscan_failed,
                                                      e.getMessage()),
Toast.LENGTH_LONG).show();
                        }

                  }
      }

      @Override
      public void onScanFailed(Exception ex) {
            hideWifiScanDialog();
            if (!ignoreWifiResults) {

                  Log.e(TAG, "Wifi scan failed!", ex);
                  Toast.makeText(
                              this,
```

```
                                this.getString(R.string.project_site_wifiscan_failed,
                                        ex.getMessage()),
Toast.LENGTH_LONG).show();

                }

        }

        @Override
        public void onClick(View v) {
                switch (v.getId()) {

                case R.id.start_wifiscan_button:
                        Log.d(TAG, "start a wifiscan");
                        try {
                                startWifiScan();
                                Toast.makeText(getApplicationContext(),  "Doing  Wifi  Scan
...",
                                                Toast.LENGTH_LONG).show();

                        } catch (WifiException e) {
                                Log.e(TAG, "could not start wifi scan!", e);
                                Toast.makeText(this,
                                                R.string.project_site_wifiscan_start_failed,
                                                Toast.LENGTH_LONG).show();
                        }

                        break;
                }
        }

        @SuppressWarnings("deprecation")
        @Override
        protected Dialog onCreateDialog(int id) {
                switch (id) {
                case DIALOG_SET_THRESHOLDS:
                        AlertDialog.Builder         thresholdDialog         =         new
AlertDialog.Builder(this);

                        thresholdDialog.setTitle("Set Thresholds");
                        thresholdDialog.setMessage("Set    the    Min(1)    and    Max(2)
thresholds");

                        // Set an EditText view to get user input
                        final EditText minInput = new EditText(this);
                        final EditText maxInput = new EditText(this);
                        final LinearLayout lila1 = new LinearLayout(this);

                        lila1.setOrientation(LinearLayout.VERTICAL);
                        minInput.setSingleLine(true);
                        minInput.setRawInputType(InputType.TYPE_CLASS_NUMBER
                                        | InputType.TYPE_NUMBER_FLAG_DECIMAL);

                        maxInput.setSingleLine(true);
                        maxInput.setRawInputType(InputType.TYPE_CLASS_NUMBER
                                        | InputType.TYPE_NUMBER_FLAG_DECIMAL);

                        lila1.addView(minInput);
                        lila1.addView(maxInput);

                        thresholdDialog.setView(lila1);

                        thresholdDialog.setPositiveButton("OK",
```

```
                                new DialogInterface.OnClickListener() {
                                        public void onClick(DialogInterface dialog,
                                                        int whichButton) {

                                                try {
                                                        float          valueMin          =
Float.parseFloat(minInput

        .getText().toString());
                                                        float          valueMax          =
Float.parseFloat(maxInput

        .getText().toString());

                                                        setMlMinThreshold(valueMin);
                                                        setMlMaxThreshold(valueMax);

                                                        // Destroy activity

                                                        // Log.d(TAG, "Thresholds set to
="
                                                        // + getMlMinThreshold() + ","
                                                        // + getMlMaxThreshold());
                                                        Toast.makeText(
                                                                        context,
                                                                        "Thresholds  set  to
="
                                                                                        +
getMlMinThreshold() + ","
                                                                                        +
getMlMaxThreshold(),

        Toast.LENGTH_SHORT).show();
                                                        dialog.cancel();

                                                } catch (NumberFormatException nfe) {
                                                        Log.w(TAG,  "Wrong  number  format
format!");
                                                        Toast.makeText(context,   "Not   a
number !",

        Toast.LENGTH_SHORT).show();
                                                }
                                        }
                                });

                        thresholdDialog.setNegativeButton("Cancel",
                                        new DialogInterface.OnClickListener() {
                                                public void onClick(DialogInterface dialog,
                                                                int whichButton) {
                                                        // Canceled.
                                                }
                                        });

                        return thresholdDialog.create();

                case DIALOG_SET_WIFI_SCAN_RATE:
                        AlertDialog.Builder          wifiRateDialog          =          new
AlertDialog.Builder(this);

                        wifiRateDialog.setTitle("Set Wifi Rate");
                        wifiRateDialog.setMessage("Set the Wifi Scan rate");
```

```
                    // Set an EditText view to get user input
                    final EditText scanRate = new EditText(this);
                    scanRate.setRawInputType(InputType.TYPE_CLASS_NUMBER
                                | InputType.TYPE_NUMBER_FLAG_DECIMAL);

                    wifiRateDialog.setView(scanRate);

                    wifiRateDialog.setPositiveButton("OK",
                                new DialogInterface.OnClickListener() {
                                      public void onClick(DialogInterface dialog,
                                                  int whichButton) {

                                            try {
                                                  int        valueScanRate        =
Integer.parseInt(scanRate

      .getText().toString());

                                                  setSchedulerTime(valueScanRate);

                                                  // Destroy activity

                                                  // Log.d(TAG, "Wifi Scan Rate set
to ="
                                                  // + getSchedulerTime()+"sec");
                                                  Toast.makeText(
                                                              context,
                                                              "Wifi  Scan  Rate  set
to ="
                                                                          +
getSchedulerTime() + "sec",

      Toast.LENGTH_SHORT).show();

                                                  dialog.cancel();

                                            } catch (NumberFormatException nfe) {
                                                  Log.w(TAG,  "Wrong  number  format
format!");
                                                  Toast.makeText(context,   "Not   a
number !",

      Toast.LENGTH_SHORT).show();
                                            }
                                      }
                                });

                    wifiRateDialog.setNegativeButton("Cancel",
                                new DialogInterface.OnClickListener() {
                                      public void onClick(DialogInterface dialog,
                                                  int whichButton) {
                                            // Canceled.
                                      }
                                });
                    return wifiRateDialog.create();

              default:
                    return super.onCreateDialog(id);
              }
      }

      /**
       * @return the mlMaxThreshold
```

```
 */
public float getMlMaxThreshold() {
	return mlMaxThreshold;
}

/**
 * @param mlMaxThreshold
 *            the mlMaxThreshold to set
 */
public void setMlMaxThreshold(float mlMaxThreshold) {
	this.mlMaxThreshold = mlMaxThreshold;
}

/**
 * @return the mlMinThreshold
 */
public float getMlMinThreshold() {
	return mlMinThreshold;
}

/**
 * @param mlMinThreshold
 *            the mlMinThreshold to set
 */
public void setMlMinThreshold(float mlMinThreshold) {
	this.mlMinThreshold = mlMinThreshold;
}

/**
 * @return the schedulerTime
 */
public int getSchedulerTime() {
	return schedulerTime;
}

/**
 * @param schedulerTime
 *            the schedulerTime to set
 */
public void setSchedulerTime(int schedulerTime) {
	this.schedulerTime = schedulerTime;
}

protected void scaleOfMap() {
	if (scaler == null) {
		scaler = new ScaleLineDrawable(context, map, new OkCallback() {

			@Override
			public void onOk() {
				onMapScaleSelected();
			}

		});
		scaler.getSlider(1).setRelativePosition(map.getWidth() / 2 - 60,
				map.getHeight() / 2);
		scaler.getSlider(2).setRelativePosition((map.getWidth() / 2 + 60),
				(map.getHeight() / 2));
		multiTouchView.invalidate();

	} else {
		onMapScaleSelected();
	}
}
```

```
	// close scaler
	@SuppressWarnings("deprecation")
	protected void onMapScaleSelected() {
		scalerDistance = scaler.getSliderDistance();

		scaler.removeScaleSliders();
		map.removeSubDrawable(scaler);
		scaler = null;
		invalidate();
	}

	/**
	 * Function to print out a 2D array
	 *
	 * @param matrix
	 */
	public void printMatrix(double[][] matrix, String variableName) {
		System.out.println();
		System.out.println("***Matrix " + variableName + "[" + matrix.length
				+ "]" + "[" + matrix[0].length + "]" + " =\n");
		for (int r = 0; r < matrix.length; r++) {
			for (int c = 0; c < matrix[r].length; c++)
				System.out.print(matrix[r][c] + " ");
			System.out.println();
		}
		System.out.println();
	}
}
```

## A.6 KNN.Java

```
package com.colostate.ML.KNN;

import java.util.Arrays;
import java.util.PriorityQueue;

import com.colostate.mecs.vinzzz.model.matrix.MatrixMathematics;

/**
 * Controller class for KNN regression ML Algo
 *
 * @author Viney Ugave(vinzzz@rams.colostate.edu)
 *
 */
public class KNN {

	/**
	 * Number of Inputs
	 */
	private int numInput;
	/**
	 * Number of outputs
	 */
	private int numOutput;
	/**
	 * Number of Samples
	 */
	private int numSamples;

	/**
```

```
 * The value of K (no. of nearest neighbours)
 */
private int K;

// Train specific arrays
/**
 * Input values
 */
private double[][] X;
/**
 * Target values
 */
private double[][] T;

// Test specific arrays
/**
 * Input values
 */
private double[][] Xtest;
/**
 * Target values
 */
private double[][] Ttest;

/**
 * @param numInput
 * @param numOutput
 */
public KNN(int numInput, int numOutput) {
	this.numInput = numInput;
	this.numOutput = numOutput;
	this.K = 1;
}

/**
 * Constructor
 *
 * @param numInput
 * @param numOutput
 * @parar K
 */
public KNN(int numInput, int numOutput, int K) {
	this.numInput = numInput;
	this.numOutput = numOutput;
	this.K = K;
}

/**
 * Function to set training data and context
 *
 * @param Xtrain
 */
public void train(double[][] Xtrain) {

	setNumSamples(Xtrain.length);
	separateAndSetXT(Xtrain, true);

}

@SuppressWarnings("null")
public double[][] test(double[][] testData) {
	separateAndSetXT(testData, false);
	double[][] ytest = new double[Xtest.length][];
```

```
	double[] distance = new double[X.length];
	double[] kDistance = new double[K];

	// for every output sample
	for (int i = 0; i < Xtest.length; i++) {
		for (int j = 0; j < X.length; j++) {
			distance[j] = getDistance(Xtest[i], X[j]);
			System.out.println(distance[j]);
		}

		// find nearest K
		kDistance = findNearestK(distance);

		// Get Indexes of nearest K
		int[] kIndex = findIndexes(kDistance, distance);

		// Get average for regression
		ytest[i] = findAverage(kIndex);

	}
	return ytest;
}

/**
 * Function to find avg
 *
 * @param kInd
 * @return
 */
@SuppressWarnings("null")
private double[] findAverage(int[] kInd) {
	// Hard coded for 2 outputs need to change
	double[] result = new double[2];
	double sumX = 0;
	double sumY = 0;
	for (int i = 0; i < kInd.length; i++) {
		int index = kInd[i];
		sumX = sumX + this.T[index][0];
		sumY = sumY + this.T[index][1];

	}

	result[0] = new Double(sumX / kInd.length);
	result[1] = new Double(sumY / kInd.length);
	return result;
}

/**
 * Function to get index of the K nearest Neighbours
 *
 * @param kDistance
 * @param distance
 * @return
 */
private int[] findIndexes(double[] kDis, double[] dis) {
	int[] index = new int[kDis.length];
	for (int i = 0; i < kDis.length; i++) {
		for (int j = 0; j < dis.length; j++) {
			if (kDis[i] == dis[j]) {
				index[i] = j;

			}
		}
```

```
		}

		return index;
	}

	/**
	 * Function to get nearest K distance
	 *
	 * @param distance
	 * @return
	 */
	private double[] findNearestK(double[] dist) {
		double[] sortedDistance = Arrays.copyOf(dist, dist.length);
		Arrays.sort(sortedDistance);
		double[] low = Arrays.copyOfRange(sortedDistance, 0, this.K);
		return low;
	}

	/**
	 * Function to find Euclidean distance between arguments
	 *
	 * @param ds
	 * @param ds2
	 * @return
	 */
	private double getDistance(double[] ds, double[] ds2) {
		if (ds.length == ds2.length) {
			double sum = 0;
			for (int i = 0; i < ds2.length; i++) {
				sum = sum + Math.pow(ds[i] - ds2[i], 2);
				// System.out.println("ds= "+ds[i]+" ds2= "+ds2[i]);

			}
			sum = Math.sqrt(sum);
			return sum;
		} else {
			System.out.println("Dimensions of Train and Test dont match");
			return 0;
		}
	}

	/**
	 * Function to separate inputs and outputs from a 2d array
	 *
	 * @param XT
	 *            2d array containing i/p & o/p like {[x1,x2,x3,.....y1,y2],
	 *            [x11,x22,x33,....y21,y23]}
	 * @param isTrain
	 *            true if training, false if test
	 */
	private void separateAndSetXT(double[][] XT, boolean isTrain) {
		int nSamples = getNumSamples();
		if (isTrain) {
			double[][] tempX = new double[nSamples][this.numInput];
			double[][] tempT = new double[nSamples][this.numOutput];

			for (int i = 0; i < nSamples; i++) {
				System.arraycopy(XT[i], 0, tempX[i], 0, this.numInput);
				System.arraycopy(XT[i], this.numInput, tempT[i], 0,
						this.numOutput);
			}

			setX(tempX);
```

```
			setT(tempT);

		} else {
			nSamples = XT.length;// For Indoor Nav
			double[][] tempX = new double[nSamples][this.numInput];
			double[][] tempT = new double[nSamples][this.numOutput];

			for (int i = 0; i < nSamples; i++) {
				System.arraycopy(XT[i], 0, tempX[i], 0, this.numInput);
				System.arraycopy(XT[i], this.numInput, tempT[i], 0,
							this.numOutput);
			}

			setXtest(tempX);
			setTtest(tempT);

		}

	}

	/**
	 * @return the numInput
	 */
	public int getNumInput() {
		return numInput;
	}

	/**
	 * @param numInput
	 *            the numInput to set
	 */
	public void setNumInput(int numInput) {
		this.numInput = numInput;
	}

	/**
	 * @return the numOutput
	 */
	public int getNumOutput() {
		return numOutput;
	}

	/**
	 * @param numOutput
	 *            the numOutput to set
	 */
	public void setNumOutput(int numOutput) {
		this.numOutput = numOutput;
	}

	/**
	 * @return the numSamples
	 */
	public int getNumSamples() {
		return numSamples;
	}

	/**
	 * @param numSamples
	 *            the numSamples to set
	 */
	public void setNumSamples(int numSamples) {
		this.numSamples = numSamples;
```

```
	}

	/**
	 * @return the x
	 */
	public double[][] getX() {
		return X;
	}

	/**
	 * @param x
	 *            the x to set
	 */
	public void setX(double[][] x) {
		X = x;
	}

	/**
	 * @return the t
	 */
	public double[][] getT() {
		return T;
	}

	/**
	 * @param t
	 *            the t to set
	 */
	public void setT(double[][] t) {
		T = t;
	}

	/**
	 * @return the xtest
	 */
	public double[][] getXtest() {
		return Xtest;
	}

	/**
	 * @param xtest
	 *            the xtest to set
	 */
	public void setXtest(double[][] xtest) {
		Xtest = xtest;
	}

	/**
	 * @return the ttest
	 */
	public double[][] getTtest() {
		return Ttest;
	}

	/**
	 * @param ttest
	 *            the ttest to set
	 */
	public void setTtest(double[][] ttest) {
		Ttest = ttest;
	}

}
```

## A.7 NeuralNetwork.Java

```
package com.colostate.ML.NeuralNetwork;

import java.util.Random;

import com.colostate.mecs.vinzzz.model.matrix.IllegalDimensionException;
import com.colostate.mecs.vinzzz.model.matrix.Matrix;
import com.colostate.mecs.vinzzz.model.matrix.MatrixMathematics;

/**
 * Neural Network class that implements a simple forward pass back propagation
 * neural net as explained by Dr Chuck Anderson in :
 * http://www.cs.colostate.edu/
 * ~anderson/cs545/index.html/doku.php?id=notes:notesneuralnet1
 *
 * @author Viney Ugave(vinzzz@rams.colostate.edu)
 */
public class NeuralNetwork {
	/**
	 * Number of Inputs
	 */
	private int numInput;
	/**
	 * Number of hidden layers
	 */
	private int numHidden;
	/**
	 * Number of outputs
	 */
	private int numOutput;
	/**
	 * Number of Samples
	 */
	private int numSamples;
	/**
	 * Learning rate rhoh
	 */
	private double rhoh;
	/**
	 * Learning rate rhoo
	 */
	private double rhoo;

	// Train specific arrays
	/**
	 * Input values
	 */
	private double[][] X;
	/**
	 * Target values
	 */
	private double[][] T;

	// Test specific arrays
	/**
	 * Input values
	 */
	private double[][] Xtest;
	/**
```

```
 * Target values
 */
private double[][] Ttest;

// Weight Matrices
/**
 * Weight matrix for hidden layer
 */
Matrix V;
/**
 * Weight matrix for output layer
 */
Matrix W;

// Matrices needed for calculation
/**
 * Output Matrix from Hidden layer
 */
Matrix Z;
/**
 * Output Matrix from Output layer
 */
Matrix Y;
/**
 * Error in Output
 */
Matrix E;

private static Random rnd;
private MatrixMathematics mMath;

/**
 * Constructor for NeuralNetwork
 *
 * @param numInput
 *            Number of i/p
 * @param numHidden
 *            Number of hidden layers
 * @param numOutput
 *            Number of o/p
 */
public NeuralNetwork(int numInput, int numHidden, int numOutput) {
       this.numInput = numInput;
       this.numHidden = numHidden;
       this.numOutput = numOutput;
       this.rhoh = 0.00000001;
       this.rhoo = 0.1;
       this.rnd = new Random(0);
}

/**
 * Constructor for NeuralNetwork
 *
 * @param numInput
 *            Number of i/p
 * @param numHidden
 *            Number of hidden layers
 * @param numOutput
 *            Number of o/p
 * @param rhoh
 *            Learning rate
 * @param rhoo
 *            Learning rate
```

```
 */
public NeuralNetwork(int numInput, int numHidden, int numOutput,
            double rhoh, double rhoo) {
      this.numInput = numInput;
      this.numHidden = numHidden;
      this.numOutput = numOutput;
      this.rhoh = rhoh;
      this.rhoo = rhoo;
      this.rnd = new Random(0);
}

/**
 * Function that inits the weights i.e V and W
 */
private void initWeights() {

      // Init Matrix Math operator
      mMath = new MatrixMathematics();

      // Calculate learning rate
      setRhoh(this.rhoh / (getNumSamples() * getNumOutput()));
      setRhoo(this.rhoo / getNumSamples());

      // Init Weights
      double[][] tempV = new double[this.numInput + 1][this.numHidden];
      double[][] tempW = new double[this.numHidden + 1][this.numOutput];

      // Initialize weights to uniformly distributed values between small
      // normally-distributed between -0.1 and 0.1
      double lo = -0.1;
      double hi = 0.1;
      // for V
      for (int i = 0; i < tempV.length; i++) {
            for (int j = 0; j < tempV[0].length; j++) {
                  tempV[i][j] = (hi - lo) * rnd.nextDouble() + lo;
            }
      }
      // for W
      for (int i = 0; i < tempW.length; i++) {
            for (int j = 0; j < tempW[0].length; j++) {
                  tempW[i][j] = (hi - lo) * rnd.nextDouble() + lo;
            }
      }

      // set the weight matrices
      this.V = new Matrix(tempV);
      this.W = new Matrix(tempW);

}

/**
 * Function to train Neural Net
 *
 * @param Xtrain
 *            2d double array containing the training set
 * @param epochs
 *            number of iterations to run as to minimize the error
 */
public void train(double[][] Xtrain, int epochs) {

      setNumSamples(Xtrain.length);
      separateAndSetXT(Xtrain, true);
      initWeights();
```

```
            // Training Input

            Matrix X = new Matrix(getX());
            Matrix X1 = new Matrix(getX()).insertColumnWithValue1();

            // Training Target
            Matrix T = new Matrix(getT());

            for (int i = 0; i < epochs; i++) {
                  // Output Matrices
                  Z = new Matrix(X1.getNrows(), V.getNcols());// 1 already added in
X
                  Y = new Matrix(Z.getNrows(), W.getNcols());

                  // Error Matrice
                  E = new Matrix(Y.getNrows(), Y.getNcols());

                  try {
                        // Forward pass on training data
                        Z = mMath.multiply(X1, V);
                        Z = Z.tanH();

                        Matrix               Z1               =               new
Matrix(Z.getValues()).insertColumnWithValue1();
                        Y = mMath.multiply(Z1, W);

                        // Error in output
                        E = mMath.subtract(Y, T);

                        // Backward pass - the backpropagation and weight update
steps

                        // 1.Calculating V
                        Matrix temp = Z.squareTheMatrix();// (1-Z**2)
                        temp = temp.subtractFromConstant(1);
                        Matrix   temp1   =   mMath.transpose(mMath.createSubMatrix(W,
0));// W[1:,:].T
                        Matrix temp2 = mMath.multiply(E, temp1);// np.dot( error,

      // W[1:,:].T)
                        Matrix   temp3   =   mMath.multiplyElements(temp2,   temp);//
np.dot(

                        // error,

                        // W[1:,:].T)

                        // *

                        // (1-Z**2)
                        Matrix    temp4    =    mMath.multiply(mMath.transpose(X1),
temp3);// np.dot(

                              // X1.T,

                              // np.dot(

                              // error,

                              // W[1:,:].T)

                              // *
```

```
                                    // (1-Z**2))
                        Matrix temp5 = temp4.multiplyByConstant(getRhoh());
                        V = mMath.subtract(V, temp5);// V = V - rh * np.dot( X1.T,
                                                                                //
np.dot( error, W[1:,:].T) *
                                                                                // (1-
Z**2))

                        // 2.Calculating W
                        Matrix  temp6  =  mMath.multiply(mMath.transpose(Z1),  E);//
np.dot(

                              // Z1.T,

                              // error)
                        Matrix temp7 = temp6.multiplyByConstant(getRhoo());// ro *

                        // np.dot(

                        // Z1.T,

                        // error)
                        W = mMath.subtract(W, temp7);// W = W - ro * np.dot( Z1.T,
                                                                                //
error)

                              // System.out.println("Iteration"+i);
                              // printMatrix(E.getValues(),"E");
                        } catch (IllegalDimensionException e) {
                              e.printStackTrace();
                        }
                  }
                  System.out.println("Training Complete! ");

      }

      public double[][] test(double[][] testData) {
            separateAndSetXT(testData, false);
            Matrix Ytest = new Matrix(Y.getNrows(), Y.getNcols());
            Matrix Xtest1 = new Matrix(getXtest()).insertColumnWithValue1();

            // Forward pass
            Matrix temp8 = mMath.multiply(Xtest1, getV());// np.dot(Xtest1,V)
            temp8 = temp8.tanH();// np.tanh(np.dot(Xtest1,V))
            Matrix  temp9  =  new  Matrix(temp8.insertColumnWithValue1().getValues());//
addOnes(np.tanh(np.dot(Xtest1,V)))
            Ytest            =            mMath.multiply(temp9,            getW());//
np.dot(addOnes(np.tanh(np.dot(Xtest1,V))),
                                                                                // W)

            System.out.println("Testing Complete!");
            return Ytest.getValues();

      }

      /**
       * Function to separate inputs and outputs from a 2d array
       *
       * @param XT
       *            2d array containing i/p & o/p like {[x1,x2,x3,.....y1,y2],
       *            [x11,x22,x33,....y21,y23]}
       * @param isTrain
```

```
 *            true if training, false if test
 */
private void separateAndSetXT(double[][] XT, boolean isTrain) {
	int nSamples = getNumSamples();
	if (isTrain) {
		double[][] tempX = new double[nSamples][this.numInput];
		double[][] tempT = new double[nSamples][this.numOutput];

		for (int i = 0; i < nSamples; i++) {
			System.arraycopy(XT[i], 0, tempX[i], 0, this.numInput);
			System.arraycopy(XT[i], this.numInput, tempT[i], 0,
					this.numOutput);
		}

		setX(tempX);
		setT(tempT);

	} else {
		nSamples = XT.length;// For Indoor Nav
		double[][] tempX = new double[nSamples][this.numInput];
		double[][] tempT = new double[nSamples][this.numOutput];

		for (int i = 0; i < nSamples; i++) {
			System.arraycopy(XT[i], 0, tempX[i], 0, this.numInput);
			System.arraycopy(XT[i], this.numInput, tempT[i], 0,
					this.numOutput);
		}

		setXtest(tempX);
		setTtest(tempT);

	}

}

public double Accuracy(double[][] tar, double[][] pred) {
	Matrix Target = new Matrix(tar);
	Matrix Ypredicted = new Matrix(pred);

	if (Target.getNrows() == Ypredicted.getNrows()
			&& Target.getNcols() == Ypredicted.getNcols()) {
		try {
			Matrix error = mMath.subtract(Target, Ypredicted);
			error = error.squareTheMatrix();
			double err = mMath.Mean(error);
			err = Math.sqrt(err);

			return err;

		} catch (IllegalDimensionException e) {
			e.printStackTrace();
		}
	}
	return -1;
}

public double Accuracy(Matrix Target, Matrix Ypredicted) {

	if (Target.getNrows() == Ypredicted.getNrows()
			&& Target.getNcols() == Ypredicted.getNcols()) {
		try {
			Matrix error = mMath.subtract(Target, Ypredicted);
			error = error.squareTheMatrix();
```

```
                    double err = mMath.Mean(error);
                    err = Math.sqrt(err);

                    return err;

              } catch (IllegalDimensionException e) {
                    e.printStackTrace();
              }
       }
       return -1;
}

/**
 * @return the numInput
 */
public int getNumInput() {
       return numInput;
}

/**
 * @param numInput
 *            the numInput to set
 */
public void setNumInput(int numInput) {
       this.numInput = numInput;
}

/**
 * @return the numHidden
 */
public int getNumHidden() {
       return numHidden;
}

/**
 * @param numHidden
 *            the numHidden to set
 */
public void setNumHidden(int numHidden) {
       this.numHidden = numHidden;
}

/**
 * @return the numOutput
 */
public int getNumOutput() {
       return numOutput;
}

/**
 * @param numOutput
 *            the numOutput to set
 */
public void setNumOutput(int numOutput) {
       this.numOutput = numOutput;
}

/**
 * @return the numSamples
 */
public int getNumSamples() {
       return numSamples;
}
```

```
/**
 * @param numSamples
 *            the numSamples to set
 */
public void setNumSamples(int numSamples) {
	this.numSamples = numSamples;
}

/**
 * @return the rhoh
 */
public double getRhoh() {
	return rhoh;
}

/**
 * @param rhoh
 *            the rhoh to set
 */
public void setRhoh(double rhoh) {
	this.rhoh = rhoh;
}

/**
 * @return the rhoo
 */
public double getRhoo() {
	return rhoo;
}

/**
 * @param rhoo
 *            the rhoo to set
 */
public void setRhoo(double rhoo) {
	this.rhoo = rhoo;
}

/**
 * @return the x
 */
public double[][] getX() {
	return X;
}

/**
 * @param x
 *            the x to set
 */
public void setX(double[][] x) {
	X = x;
}

/**
 * @return the t
 */
public double[][] getT() {
	return T;
}

/**
 * @param t
```

```
 *            the t to set
 */
public void setT(double[][] t) {
	T = t;
}

/**
 * @return the xtest
 */
public double[][] getXtest() {
	return Xtest;
}

/**
 * @param xtest
 *            the xtest to set
 */
public void setXtest(double[][] xtest) {
	Xtest = xtest;
}

/**
 * @return the ttest
 */
public double[][] getTtest() {
	return Ttest;
}

/**
 * @param ttest
 *            the ttest to set
 */
public void setTtest(double[][] ttest) {
	Ttest = ttest;
}

/**
 * @return the v
 */
public Matrix getV() {
	return V;
}

/**
 * @param v
 *            the v to set
 */
public void setV(Matrix v) {
	V = v;
}

/**
 * @return the w
 */
public Matrix getW() {
	return W;
}

/**
 * @param w
 *            the w to set
 */
public void setW(Matrix w) {
```

```
			W = w;
		}

		/**
		 * Function to print out a 2D array
		 *
		 * @param matrix
		 */
		public void printMatrix(double[][] matrix, String variableName) {
			System.out.println();
			System.out.println("***Matrix " + variableName + "[" + matrix.length
					+ "]" + "[" + matrix[0].length + "]" + " =\n");
			for (int r = 0; r < matrix.length; r++) {
				for (int c = 0; c < matrix[r].length; c++)
					System.out.print(matrix[r][c] + " ");
				System.out.println();
			}
			System.out.println();
		}

}
```

## A.8 Regression.Java

```
package com.colostate.ML.LinearRegression;

import com.colostate.mecs.vinzzz.model.matrix.Matrix;
import com.colostate.mecs.vinzzz.model.matrix.MatrixMathematics;
import com.colostate.mecs.vinzzz.model.matrix.NoSquareException;
/**
 * Controller class for Linear Regression
 * @author viney
 *
 */
public class Regression {

		/**
		 * Number of Inputs
		 */
		private int numInput;
		/**
		 * Number of outputs
		 */
		private int numOutput;
		/**
		 * Number of Samples
		 */
		private int numSamples;

		// Train specific arrays
		/**
		 * Input values
		 */
		private double[][] X;
		/**
		 * Target values
		 */
		private double[][] T;

		// Test specific arrays
		/**
```

```
 * Input values
 */
private double[][] Xtest;
/**
 * Target values
 */
private double[][] Ttest;

// Weight Matrices
/**
 * Weight matrix
 */
Matrix W;

private MatrixMathematics mMath;


/**Constructor
 * @param numInput
 * @param numOutput
 */
public Regression(int numInput, int numOutput) {
	this.numInput = numInput;
	this.numOutput = numOutput;
}

/**
 * Function to train the regression model
 *
 * @param Xtrain
 *            2d double array containing the training set
 */
public void train(double[][] Xtrain) {
	setNumSamples(Xtrain.length);
	separateAndSetXT(Xtrain, true);

	// Training Input
	Matrix X = new Matrix(getX());
	Matrix X1 = new Matrix(getX()).insertColumnWithValue1();

	// Training Target
	Matrix T = new Matrix(getT());

	try {
		Matrix temp = mMath.transpose(X);//X.T
		Matrix temp1 = mMath.multiply(temp, X);//np.dot(X.T, X)
		Matrix temp3 = mMath.multiply(temp, T);//np.dot(X.T,T)
		Matrix temp4 = mMath.inverse(temp1);//np.dot(X.T,X)- inverse
		W =mMath.multiply(temp4, temp3);


	} catch (NoSquareException e) {
		e.printStackTrace();
	}

}

public double[][] test(double[][] testData) {
	separateAndSetXT(testData, false);

	Matrix Xtest =new Matrix(getXtest());

	Matrix Ytest=mMath.multiply(Xtest, W);
```

```
	return Ytest.getValues();

}
/**
 * Function to separate inputs and outputs from a 2d array
 *
 * @param XT
 *            2d array containing i/p & o/p like {[x1,x2,x3,.....y1,y2],
 *            [x11,x22,x33,....y21,y23]}
 * @param isTrain
 *            true if training, false if test
 */
private void separateAndSetXT(double[][] XT, boolean isTrain) {
	int nSamples = getNumSamples();
	if (isTrain) {
		double[][] tempX = new double[nSamples][this.numInput];
		double[][] tempT = new double[nSamples][this.numOutput];

		for (int i = 0; i < nSamples; i++) {
			System.arraycopy(XT[i], 0, tempX[i], 0, this.numInput);
			System.arraycopy(XT[i], this.numInput, tempT[i], 0,
					this.numOutput);
		}

		setX(tempX);
		setT(tempT);

	} else {
		nSamples = XT.length;// For Indoor Nav
		double[][] tempX = new double[nSamples][this.numInput];
		double[][] tempT = new double[nSamples][this.numOutput];

		for (int i = 0; i < nSamples; i++) {
			System.arraycopy(XT[i], 0, tempX[i], 0, this.numInput);
			System.arraycopy(XT[i], this.numInput, tempT[i], 0,
					this.numOutput);
		}

		setXtest(tempX);
		setTtest(tempT);

	}

}




/**
 * @return the numInput
 */
public int getNumInput() {
	return numInput;
}


/**
 * @param numInput the numInput to set
 */
public void setNumInput(int numInput) {
	this.numInput = numInput;
}
```

```
/**
 * @return the numOutput
 */
public int getNumOutput() {
	return numOutput;
}


/**
 * @param numOutput the numOutput to set
 */
public void setNumOutput(int numOutput) {
	this.numOutput = numOutput;
}


/**
 * @return the numSamples
 */
public int getNumSamples() {
	return numSamples;
}


/**
 * @param numSamples the numSamples to set
 */
public void setNumSamples(int numSamples) {
	this.numSamples = numSamples;
}


/**
 * @return the x
 */
public double[][] getX() {
	return X;
}


/**
 * @param x the x to set
 */
public void setX(double[][] x) {
	X = x;
}


/**
 * @return the t
 */
public double[][] getT() {
	return T;
}


/**
 * @param t the t to set
 */
public void setT(double[][] t) {
	T = t;
```

```
	}


	/**
	 * @return the xtest
	 */
	public double[][] getXtest() {
		return Xtest;
	}


	/**
	 * @param xtest the xtest to set
	 */
	public void setXtest(double[][] xtest) {
		Xtest = xtest;
	}


	/**
	 * @return the ttest
	 */
	public double[][] getTtest() {
		return Ttest;
	}


	/**
	 * @param ttest the ttest to set
	 */
	public void setTtest(double[][] ttest) {
		Ttest = ttest;
	}


	/**
	 * @return the w
	 */
	public Matrix getW() {
		return W;
	}


	/**
	 * @param w the w to set
	 */
	public void setW(Matrix w) {
		W = w;
	}

}
```

# ABBREVIATIONS

| | |
|---|---|
| 3D | 3-Dimensional |
| 3G/4G | 3rd and 4th Generation (of cellular mobile networks) |
| AP | Access Point |
| API | Application Programming Interface |
| ARM | Advanced RISC Machine |
| FCC | Federal Communications Commision |
| LBS | Location Based Services |
| CPU | Central Processing Unit |
| GNSS | Global Navigation Satellite System |
| GPS | Global Positioning System |
| GLONASS | Globalnaya navigatsionnaya sputnikovaya sistema |
| HTC | High Tech Computer corporation |
| KNN | K-Nearest Neighbor |
| LBA | Location-Based Application |

| | |
|---|---|
| MEMS | MicroElectroMechanical System |
| MHz/GHz | MegaHertz/GigaHertz |
| mW | milliWatts |
| ANN | Artifitial Neural Network |
| OS | Operating System |
| PC | Personal Computer |
| Li | Lithium |
| LTE | Long-Term Evolution |
| QoS | Quality of Service |
| RAM | Random Access Memory |
| mAh | milliAmpere Hour |
| RFID | Radio Frequency Identification |
| RSSI | Received Signal Strength Indicator |
| MAC | Media Access Control |
| SMS | Short Message Service |
| SVM | Support Vector Machine |
| UI | User Interface |

| | |
|---|---|
| VRL | Variable Rate Logging |
| WiFi | Wireless Fidelity |