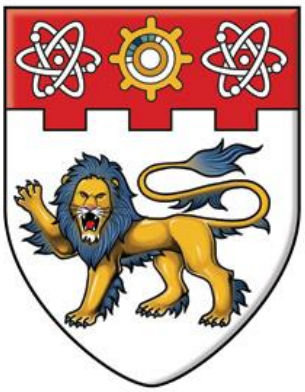
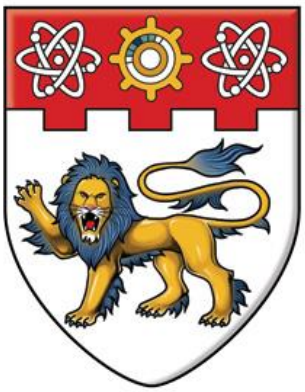


NANYANG TECHNOLOGICAL UNIVERSITY

HUMAN ACTIVITY RECOGNITION AND TRACKING

June Quak Ren Feng
(U1121476E)

School of Computer Engineering
2014



NANYANG TECHNOLOGICAL UNIVERSITY

SCE13-0344
HUMAN ACTIVITY RECOGNITION AND TRACKING

Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Computer Science of the Nanyang Technological University

by

June Quak Ren Feng
(U1121476E)

Supervised by: Professor Tan Ah Hwee

School of Computer Engineering
2014

Abstract

As the population in Singapore grows older, the health-related problems trend will also be seen more commonly. As there will be an increasing number of elderly people with health-related problems likely to need assistant in daily living, the number of caretakers will soon be outnumbered. Therefore, it is necessary to build smart systems to aid the elderly people in their daily living environment.

In order to provide unobtrusive monitoring, every day devices such as smartphones could be used. The more advanced way of data collection for body movement is the use of smartphone sensors; and the common way to build a model for motion recognition includes the use of Hidden Markov Model (HMM).

HMM was used in many motion recognition applications. However, many of these applications only included continuous motions (Running, Walking, Standing, and Sitting) recognition. This report focuses on building of a model that is able to recognize continuous and transitional motions.

In this project, a model was developed with HMM to classify temporal sequences of sensory data. Human motions are categorized into two categories, the lower-level motions and higher-level motions. Lower-level motions include Running, Walking, Standing, Sitting and Lying; higher-level motions include Sit down, Stand up, Lie down, and Fall. A rule-based system was built upon the HMM to achieve transitional (higher-level) motions recognition. Experiments were conducted in real-time to demonstrate the performance of the motion recognizer.

The HMM which can recognize lower-level motions had an accuracy of 99.98% and the incorporated rule-based system for recognizing higher-level motions had the accuracy of 76.25%.

Acknowledgement

I am grateful to have Professor Tan Ah Hwee as my Final Year Project Supervisor. He has given me incredible amount of support and constant guidance during the project period. A weekly meeting was coordinated by him and the project team to ensure that I progressed well every week. I am thankful to Professor Tan especially for the amount of time spent where he would review my progress, help with the problems I encountered and provide valuable suggestions. I also appreciate the time spent explaining any doubts I had and areas of the algorithms that I did not understand. I believe without the weekly meeting and his constant guidance, this project will not be complete.

I would also like to extend my gratitude to Wang Di, the project manager in charge of me, for his guidance and support throughout the course of the project. He has given me many valuable suggestions, constant advice, and has fixed several other separate meetings towards the end of the project to ensure that I am kept to schedule. I am also thankful to Wang Di for the time he took to go through technical details and reports. The aids he has given me were critical to the success of this project.

Lastly, I am also grateful to the project team of LILY center who have provided me with support during the weekly meeting.

Table of Contents

List of Figures

List of Tables

List of Equations

1. Introduction

1.1 Background & Motivation

Singapore will be facing a shrinking and ageing population [1] [2] in years to come with increasing life expectancy and low birth rates trends. Figure 1 shows an overview of the expected population size and citizen's median age from 2012 to 2060.

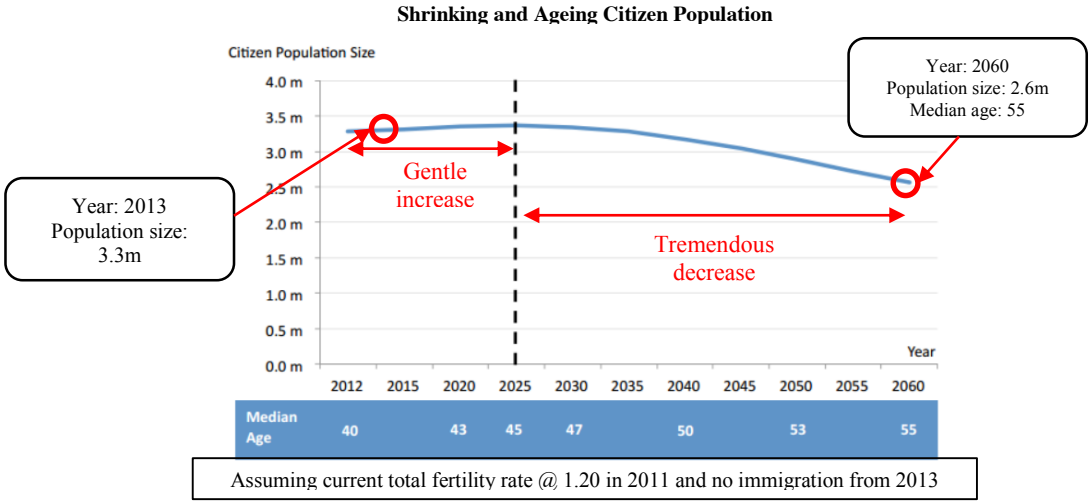


Figure 1 Overview of Singapore's Expected Population Size from 2012 to 2060

Taking a closer look at the number of citizens of different age segments in Figure 2, the number of elderly people in 2050 is expected to be three times more than that in 2012.

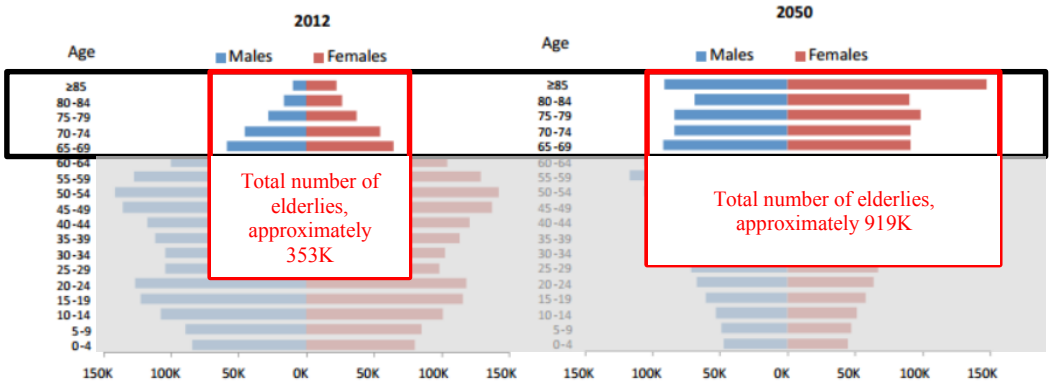


Figure 2 Singapore Citizen Age Profile

Due to an aging population, the health-related problems trend will also be seen more commonly. Hence, there will be an increasing number of elderly people with health-related problems likely to need assistance in daily living. According to a newspaper article [3] dated April 2012, the number of elderly people living alone will increase from 35,000 in 2012, to 83,000 in 2030. With that, the number of caretakers will soon be outnumbered by the rapid increase in the number of elderly people. Hence, it is important to design systems and devices so that the elderly people would be able to stay alone for long hours and simultaneously be monitored and taken care of.

There are many tools [4] available to ensure the safety of the elderly people living alone. However most of these tools require user-interaction (elderly people pressing button) to collect notifications to alert the others. One of the better tools that provides visual view of the surroundings would be live streaming video through cameras that are installed inside a house [12]. In the past years, live streaming videos would have to be kept in view 24/7 and they were unable to send alerts when important events (elderly people fell down) happened at home. In recent years, these live streaming systems are made smarter and some of these systems could send out video clips and images of the important events to the respective people monitoring the homes. However, live streaming systems would raise privacy concerns to the people being monitored in the case of data being stolen.

As technologies advance, smartphones have gradually played an important role in one's lives. There are plenty of things one can do with mobile phones at anywhere. They are now something that one cannot live without, because of that, phone manufacturers are coming up with better smartphones and these smartphones are getting more powerful each year. Smartphones are now able to provide many sensing capabilities that can provide information of the surroundings they are in. With the help of these smartphones, one can easily be able to track and know what the elderly people are doing at home, and whether they have encountered any problems that require close attention.

1.2 Project Objectives and Tasks

The project's main objective is to create a smartphone application that will make use of the data retrieved from several sensors embedded in the phone and be able to recognize the motions of an individual as fast as possible. A user model is to be set up in order to be able to recognize each individual motions.

The motions are categorized into two categories, lower-level and higher-level motions. The lower-level motions consist of continuous motions like "Running", "Walking", "Standing", "Sitting", "Lying" while the higher-level motions like "Fall", "Sit down", "Lie down", "Stand up" consists of transitions of two different lower-level motions.

1.3 Scope and Limitations

The scope of this project is the creation of a system for detecting the various physical motions of an individual. Hidden Markov Model is used in this project for motion recognition for lower-level motions. Higher-level motions are determined by a combination different lower-level motions. This project aims to use every day devices to classify activities regardless where the person is within a home environment. Activities like climbing the stairs, cycling, and driving are not considered in this project as these activities would not be likely to be carried out in a home environment.

1.4 Overview of Report

This report comprises of 8 chapters. They are organized as follows:

Chapter 1 presents the background and objectives of this project;
Chapter 2 provides the literature review of the current human activity recognition systems;
Chapter 3 introduces the methodologies and implementations of different methods used for activity recognition;
Chapter 4 evaluates the performance of experimental results and analysis;
Chapter 5 presents the 3D visualization for the different sensors readings for different motions;
Chapter 6 presents the conclusion and future enhancements that could be made in future;
Chapter 7 provides the bibliography of the various sources that were referred to;
Chapter 8 provides the appendices for this report.

2. Literature Review

There have been many research conducted for human activity recognition on smart devices, which include smartphones. There are various methods in which the researchers have looked into, from data collection steps, data pre-processing steps, classification steps and to making sense of the data collected to recognize motions.

Firstly, the data collection step includes considerations for the kind of data in which one would like to collect. The two most common kind are the vision-based sensor information, and wearable-based sensor information. Vision-based sensor information includes the use of Global Positioning System (GPS) and live video streaming systems [12] to collect the environment information the user is in. The more commonly used devices for collecting wearable-based sensor information include wearable sensor-based system, and smartphones sensors. Of course, to measure the movement of one individual, the position of the devices have to be considered thoroughly as different parts of the body will return different kinds of movement acceleration which will greatly affect the data collected.

Secondly, the commonly used methods in data pre-processing steps are features extractions from sensory data and sampling data at fixed sampling rate [13]. Features such as mean, median, and standard deviation were extracted out of the raw sensory data and were pre-processed at fixed sampling rate.

There are many methods by which classifications of sensory data are achieved. The more commonly used methods are the K-Nearest Neighbor [5], Naïve Bayes [5], Support Vector Machine [13] and Hidden Markov Model (HMM) [10][15]. Using the basic methodologies of the commonly used methods, people have come up with advanced and improved methods like AL-HMM [14], and combinations of methods [5] to achieve classifications with higher accuracy.

In one study [16], the motion recognition step was analyzed using body movement information collected by the video sensors. Several features were extracted out of the data collected and HMM was used to perform motion recognition. However, the usage of video is unfeasible because the area of sensing covered would be low and it will require high cost if many video sensors were used.

More recently, as technology advances, smartphone sensors and accelerometer became widely used by many researchers [17]. This is because accelerometer on the smartphones have become very accurate features in motion recognition.

Using these sensory data of smartphones, another study [8] uses yet another methodology to determine the actual motion of an individual. Decision Tree (DT) was used and various features such as Signal Magnitude Area (SMA), Signal Magnitude Vector, orientations, and tilt angle of the smartphone were extracted from the sensory data and were used as parameters to the DT. The performance of using DT with SMA has achieved a 90.8% accuracy.

However most papers like [5] defined the motions as a unique set of activities like Standing, Walking, and Running. They are unable to detect transitional period between the activities. This is one of the problems that this project aims to resolve.

3. Methodologies

This chapter describes the architecture and algorithms of different techniques used in this project. It also includes detailed descriptions on the implementations of various steps to completing the main goal of the project. The various steps include the implementation of Android application, data collection and pre-processing process, implementation of HMM, implementation of rule-based system to recognize motion transitions, and lastly testing of the entire application.

3.1 Overview of Algorithms

This section will provide an overview to the HMM architecture and Viterbi algorithm which were used to recognize user's motions; K-means algorithm which was used in the implementation of HMM in Jahmm [18]; Signal Magnitude Area (SMA) feature for conducting higher-level motion recognition; and Cross-validation technique for measuring the performance for the HMM.

3.1.1 Hidden Markov Model

User's motions are considered as the temporal sequences because the motions happen in the space of time. As this project focuses on recognizing motions, the exact motions are unobserved (hidden) and motions can be estimated based on the observable behavior of the sensors embedded on the smartphone. HMM is used because it can represent a temporal sequence problem as graphical models.

HMM [21] is a model with Markov (memory-less) property. HMM models a process where one can make predictions for the future of the process based on the present state, and that the future and past are independent of each other. HMM [6] consists of the hidden states, observations, initial state probabilities distributions, state transition probability matrix and observation probability matrix. Figure 3 shows the general architecture of HMM.

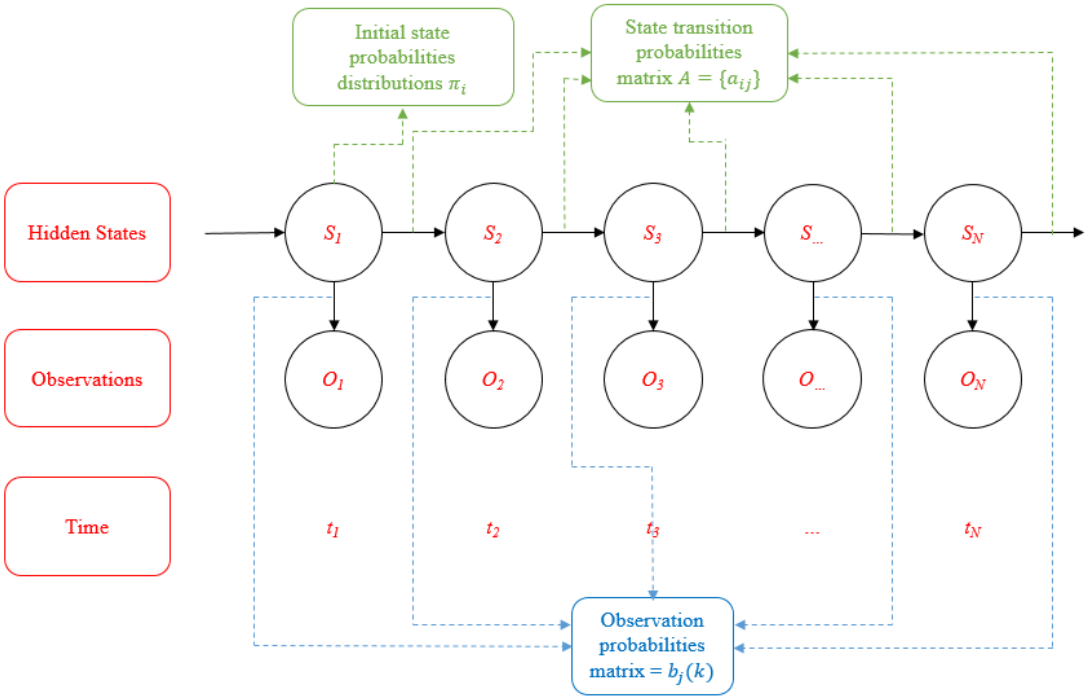


Figure 3 General Architecture of HMM

The set of states $S = \{s_1, s_2, s_3, \ldots, s_N\}$ with N distinct states includes the initial state (S_1), and the final state (S_N). Each state in S is a hidden state at each time t and the actual state q at t is defined as $q_t = s_t$ [15]. Each state in S consists of M distinct observation symbols which can be denoted as $O = \{v_1, v_2, v_3, \ldots, v_M\}$. In the theory of HMM, the observable variable $o_t = v_k, 1 \leq k \leq M$ at t depends only on the hidden state variable S_i at that time.

A HMM uses three probability distributions. They are the probability distributions over initial states, state transition probabilities distributions of two states, and observation probabilities distributions.

The probability distributions over initial states π are represented in Equation 1.

$$\pi_i = P(q_1 = s_i)$$

Equation 1 Probability Distributions over Initial States

Every state in S shares a mutual state transition probability matrix which is represented by $A = \{a_{ij}\}$ (see Equation 2). The state transition probability matrix describes the probability of a pair of states, for instance, state i going to each of the N possible states j at $t+1$.

$$a_{ij} = P(q_t = s_j | q_{t-1} = s_i), \quad 1 \le i, j \le N$$

Equation 2 State Transition Probability Distributions of Two Consecutive States

Table 1 shows an example of state transition probability matrix. Assume that the states are the human's motions, the probability of an individual walking (S_i) at t to walking (S_j) at $t+1$ is 0.8.

State S_j

State S_i / State/State	Walk	Run
Walk	0.8	0.2
Run	0.1	0.9

Table 1 Example of State Transition Probability Matrix

Every state in S shares a common observation probability distribution matrix which is represented by $b_j(k)$ indicates the probability of state j that would generate an observation $o_t = v_k$.

$$b_j(k) = P(o_t = v_k | q_t = s_j), \quad 1 \le j \le N, 1 \le k \le M$$

Equation 3 Observation Probability Distributions

Table 2 shows an example of observation probability matrix governed by categorical distribution. Assume that the states are the human's motions and observations are the sensory axis X values, the probability of $q_t = "Walk"$ having $o_t = "X = 1.0"$ is 0.3.

Observation States

Hidden states / State/Observation	X=1.0	X=2.0
Walk	0.3	0.7
Run	0.1	0.9

Table 2 Example of Observation Probability Matrix

With the three probability distributions obtained, a complete structure for HMM is formed.

3.1.2 Viterbi Algorithm

Given the structure of HMM, the hidden states were still unachievable. Therefore, Viterbi Algorithm is introduced so that hidden state can be known. The objective of Viterbi Algorithm [10] is to find the single best state sequence $Q = \{q_1, q_2, q_3, \dots, q_t\}$ for a given observation sequence $O = \{o_1, o_2, o_3, \dots, o_t\}$. The highest probability sequence $\delta_t(i)$ will be the output sequence of the algorithm.

Figure 4 shows an illustration of Viterbi Algorithm applied on the HMM. The right hand side of Figure 4 is an expanded version of Figure 3 where there is N possible values for one state Si. The red path is the highest probability sequence illustration.

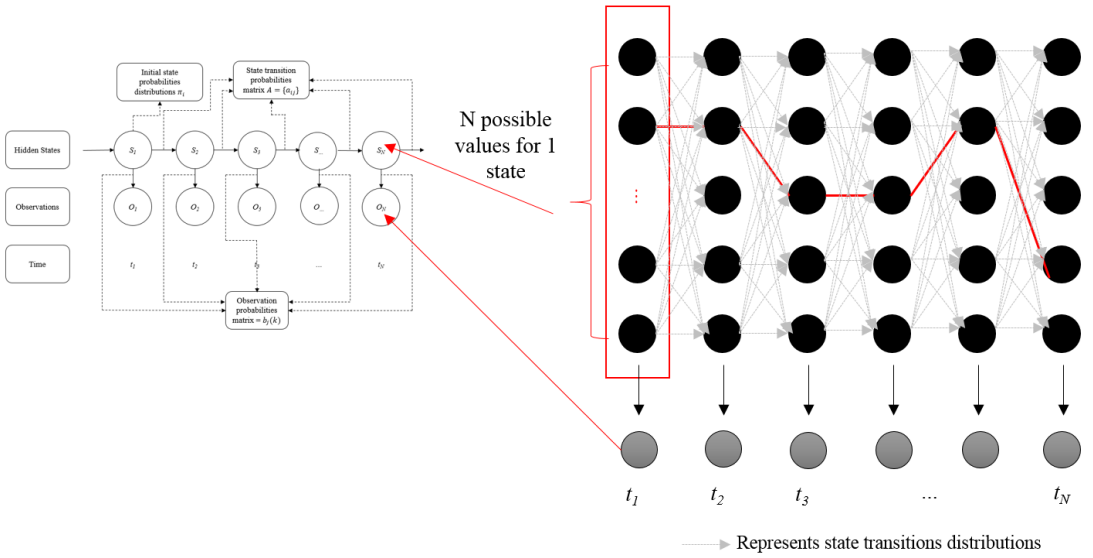


Figure 4 Illustration of Viterbi Algorithm Best State Sequence Path

The initialization step (see Equation 4) to calculate the highest probability sequence is to find the probable sequence at t_1.

$$\delta_1(i) = \pi_i \, . \, b_i(o_1), \quad where \; 1 \leq i \leq N$$

Equation 4 Initialization Step for Viterbi Algorithm

Equation 5 shows the equation to achieve the highest probability sequence where $\delta_t(i)$ accounts for the first t observations and end in state S_i.

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_t} P(q_1, q_2, \dots, q_t = s_i, o_1, o_2, \dots, o_t)$$

Equation 5 Highest Probability Sequence for State i

Equation 6 shows the formula to achieve the maximum probability of each next state. To achieve the next highest probability sequence path at *t+1*, Equation 5 has to be solved inductively as:

$$\delta_{t+1}(j) = \left[\max_{1 \le i \le N} \delta_t(i).a_{ij}\right].b_j(o_{t+1})$$

Equation 6 Cumulative Equation to Get Highest Probability for State j

Since Equation 5 returns the highest probability sequence for every *t*, the following equation shows the best hidden state for every t.

$$\iota_t^* = \underset{1 \le i \le N}{\operatorname{argmax}}[\delta_t(i)]$$

Equation 7 Highest Probability Sequence w.r.t Observation Sequence

3.1.3 K-means Algorithm

This project uses Jahmm [18] to implement HMM. Jahmm is an open source HMM module provided by Java. It made use of traditional K-means algorithm to build the model. K-means algorithm was used as the sensory data collected in the data collection process contained many dimensions with a wide range of values. K-means algorithm [9] was used to cluster points that belongs to the cluster with the nearest mean to reduce number of dimensionality. This forms clusters of data which is easier to classify since there are fewer groups of data compared to many individual values.

3.1.4 Signal Magnitude Area

This project aims to be able to recognize continuous and transitional motions. Hence, motions are divided into two categories, the lower-level motions and the higher-level motions (see Figure 19 and Table 7). Lower-level motions will be classified and recognized by the HMM, while higher-level motions will be determined by a rule-based system together with Signal Magnitude Area (SMA). SMA metric [8] is to distinguish between a resting state and user activity for recognition of basic daily movements. *X(t), y(t),* and *z(t)* are acceleration signals from each axis with respect to time *t*.

$$SMA = \frac{1}{t}\left(\int_0^t |x(t)|dt + \int_0^t |y(t)|dt + \int_0^t |z(t)|dt\right)$$

Equation 8 Signal Magnitude Area

Figure 5 shows an illustration of SMA in pictorial form. The area under axis *X*, *Y*, *Z* are calculated with SMA. SMA provides a good indication for any sudden motions with fast acceleration as it calculates the area (acceleration) under the three axis.

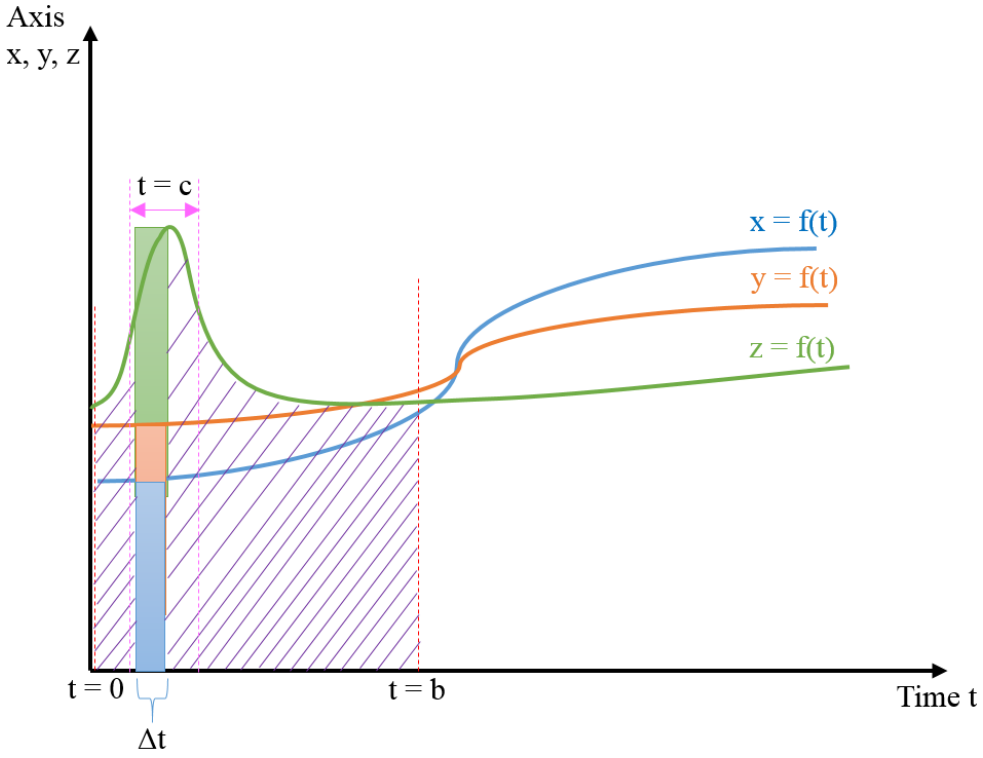


Figure 5 Illustration of Total Area under Curve for Three Axes

For example, if an individual were to make a sudden motion (fell down) at $t = c$, the accelerometer value of *z-axis* is likely to have a noticeable change. The area under the curve will also have a high value. Hence, SMA provides a good indication for any sudden motions with fast acceleration.

3.2 Design and Implementation of Android Application

This section describes the implementation of Android application and the usage of smartphones embedded sensors for data collection process.

3.2.1 Overview of Android Application

As this project used smartphones embedded sensors to collect sensory data for recognizing motions, an Android application was created to serve as a user interface for users to collect sensory data.

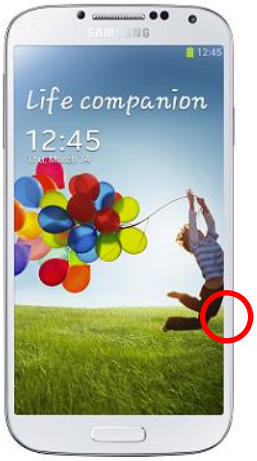
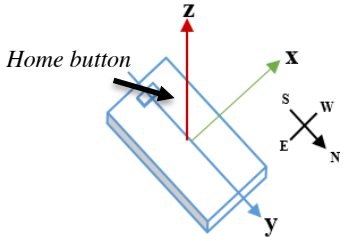
A comparison of different smartphones was conducted at the beginning of this project. The phone that was recommended was Samsung Galaxy S4. Galaxy S4 was selected as it was the most advanced smartphone in the market during the start of this project. It has a big screen and light in weight, which is a plus point to the elderly people. Galaxy S4 consists of twelve sensors and out of which, four (Accelerometer, Linear Accelerometer, Gyroscope and Magnetometer) were used in this project.

An application was created on the Galaxy S4 to collect the data using the smartphone's sensors after smartphone model selection. The application was created with the Android SDK, which provides API libraries and developer tools on the Eclipse software. The platform version 4.2 with API level 14 was used. For each of the four sensors, three axes *X, Y, Z* will have their readings recorded.

3.2.2 Sensors Details

Four sensory data were collected during data collection step and these data were used to classify and recognize motions with HMM. The details of each sensors are as follows.

Types of sensors used	**Detection**
Linear Accelerometer	To measure smartphone's movement and acceleration without Earth's gravity.
Magnetometer (Unit measurement: uT micro-Tesla)	To measure the direction of the magnetic field acting on the smartphone. The magnetometer sensor within the smartphone is very sensitive to interference from local magnetic fields, even if these are very small (such as a metal button on the smartphone's casing).
< Two other sensors, Accelerometer and Gyroscope will be as follows >	

Types of sensors used	Detection
Accelerometer  (Units measurements: m/s²)	Measures the smartphone's movement with Earth's gravity and is used as a Walking Mate. When the phone is lying flat on table, the returned value for all three axis are as follows, X = 0.0 m/s² Y = 0.0 m/s² Z = ~9.8 m/s²  The z axis has a reading of ~9.8 when it is parallel to the ground and motionless. This matches the gravitational acceleration on Earth which is between 9.78 and 9.82 m/s². As the phone changes its orientation, the axis of X, Y, Z changes accordingly.
Gyroscope  (Unit measurements: rad/s)	Measures the inclination of the smartphone. Angular Speed X = 0.0 rad/s Y = 0.0 rad/s Z = 0.0 rad/s Rate of rotation (Angular Speed) around x-axis. Rate of rotation (Angular Speed) around y-axis. Rate of rotation (Angular Speed) around z-axis. Rate of rotation is positive in the counter-clockwise direction.

3.2.3 Design of Android Application

The Android SDK and API libraries allow retrieving and displaying of different sensory data on to the application that has been developed. Figure 6 shows the interface design of the application that collects the sensory data for motion recognition.

Figure 6 Interface Design of Android Application

(a) Users can click on the "Start Recording" button to start recording for motion recognition.

(b) The IP address textbox requires users to input the IP address the server and phone is connected to so that pre-processing step can be initiated.

(c) Users can also fill up the time interval textbox to define how frequently they want to send the sensors to record data at (in seconds). For example, putting "2" at time interval textbox will send the sensory data over to the server every two seconds.

Users would only need to enter the IP address and time interval once. If they restart the application, the entered IP address and time interval will still be there. Users can change IP address and time interval at any time.

The other information that are displayed on the application includes the ambient light sensor, ambient magnetic fields sensor, linear accelerometer, accelerometer, gyroscope, proximity, date and time. These information are for display and will not be further processed.

3.3 Implementation of Data Collection and Pre-processing Processes

This section describes the data collection and pre-processing processes. The collected sensory data from various sensors were pre-processed so that they were well-sampled and consistent to the HMM.

The entire structure of the motion recognition system made use of the client-server architecture. The smartphone acts as the client in the client-server architecture while a PHP server was set-up so that pre-processing step and motion recognition step could be conducted on it.

3.3.1 Data Collection Process

Data collection was carried out in both the training and testing phases of the HMM. It is an essential step as the data collected were used as inputs to the HMM. Figure 7 shows the data collection process for training data for HMM.

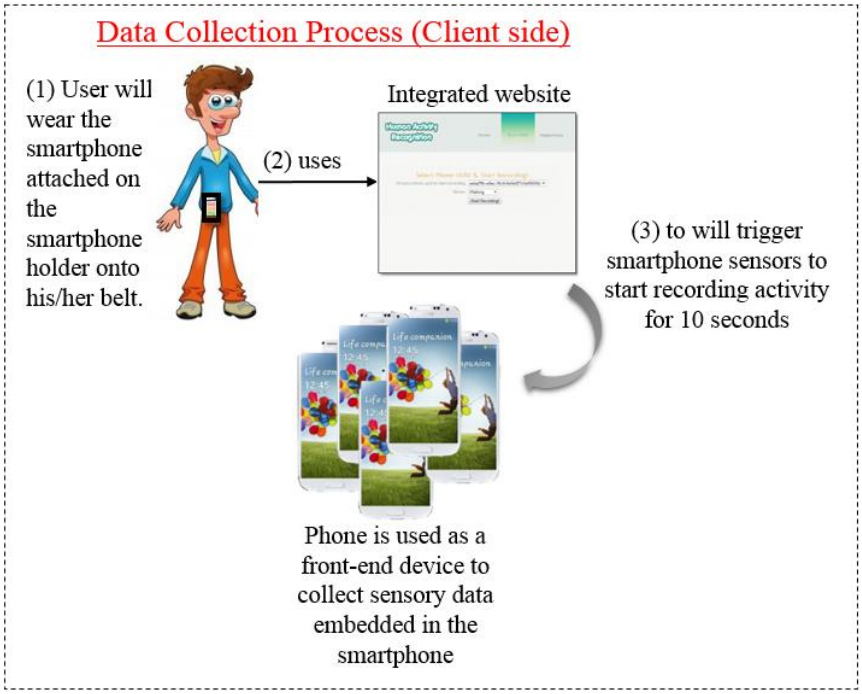


Figure 7 Data Collection Process to Train Data for HMM (on Client Side)

Firstly, before the data collection process begins, the user would have to wear the smartphone that is attached on the smartphone holder onto his/her belt. Figure 8 and Figure 9 shows the Galaxy S4 belt holder and the how it should be worn while data collection process is in progress.

Figure 8 Galaxy S4 Plastic Black Case with Belt Clip

A Samsung Galaxy S4 belt clip was used as the holder to hold on the phone when data collection is in progress. This holder is specially selected so that minimal noise was collected during the data collection process. This holder ensures that the phone is in the same orientation so that recalibration of the smartphone for pre-processing step is not required.

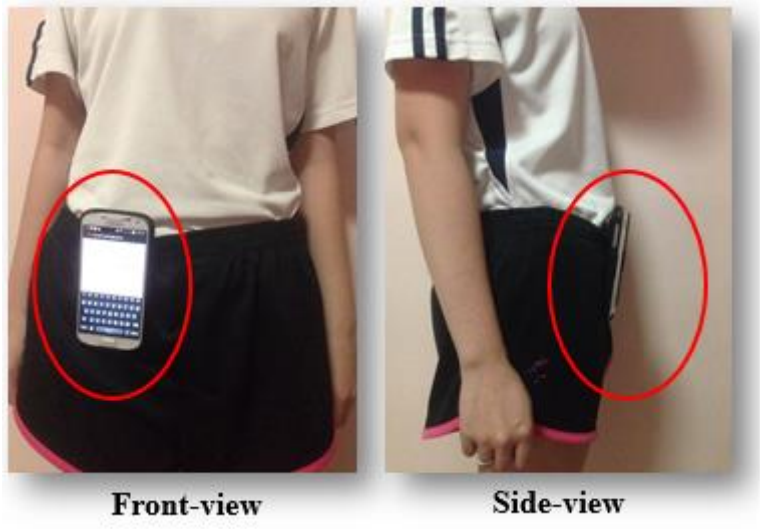


Figure 9 Attaching Phone Holder to the User

Figure 9 shows how the holder is to be worn on the user during both training and testing of HMM data collection process. This holder also reduces noisy data being collected as the phone is not directly rested on the user's body. If the phone is directly rested onto the user's body, the data may be corrupted by user's breathing action according to my own findings.

After placing the smartphone at proper position, data collection process may be initiated. The user may initiate the start of the data collection process by using the integrated website (see Section 3.9). The integrated website will send a trigger to the Android application, the application will continuously store the sensory data for ten seconds while the user does the motion.

The sensors will all be initiated once at the same time for data collection step to be accurate. The sensory data were collected, stored into the smartphone cache and sent to the server for pre-processing step after the ten seconds training interval has ended.

The returned sensory data from the various sensors were not synchronized because the sensory data was only recorded when the sensors changed their sensory values. Thus, the pre-processing step is needed.

3.3.2 Data Pre-processing Process

After each of the ten seconds segments of data collection, the sensory data were sent to a MySQL database on the server. As the various sensory data were not collected at same speed and time, the pre-processing step was needed so that the inputs to the HMM were synchronized and consistent.

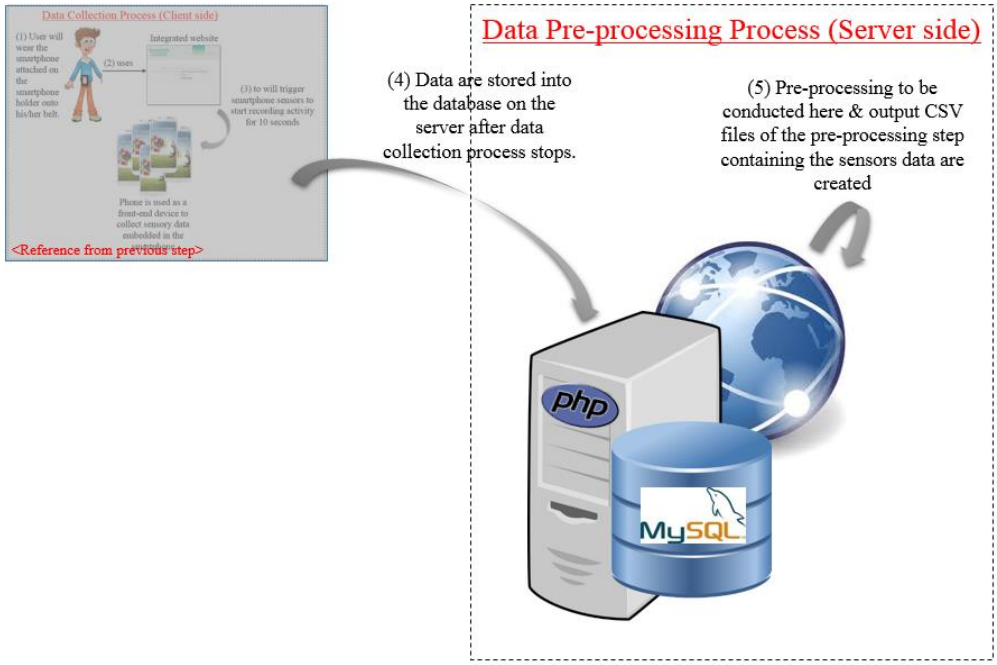


Figure 10 Data Pre-processing Process on Server Side

The data pre-processing consisted of two steps – pre-processing data to regular sampling frequency and at fixed-width sliding windows. In step one, data was pre-processed at regular sampling frequency so that sensory data were consistent. The sampling frequency was created dynamically (easy to change on the integrated website) so that different experiments can be conducted quickly.

An example of a regular sampling frequency would be 100Hz. This means that records were sampled at one hundred sensory data per second, or in another words one sensory data per ten milliseconds. Therefore in ten seconds, there would be one thousand data per ten seconds recording for each sensors.

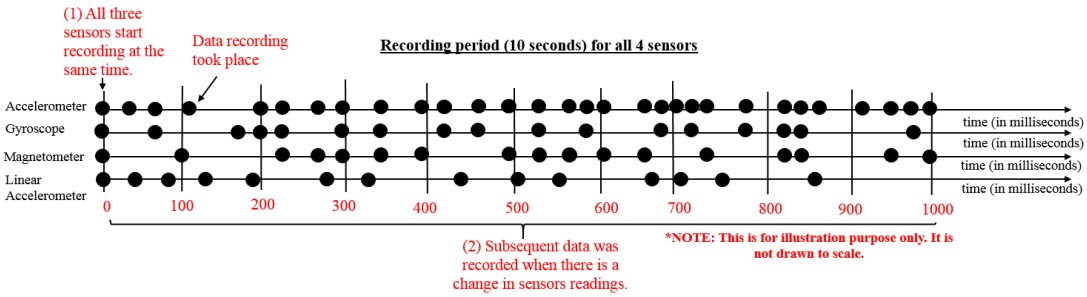


Figure 11 Example of Data Collection over Ten Seconds

In Figure 11, the black dots on the time-line refers to the recording of sensory data at each time interval. For example, at time 0 millisecond (ms) to 100 ms, the Accelerometer value changes thrice and that explained why three black dots can be seen from Figure 11 during that time interval.

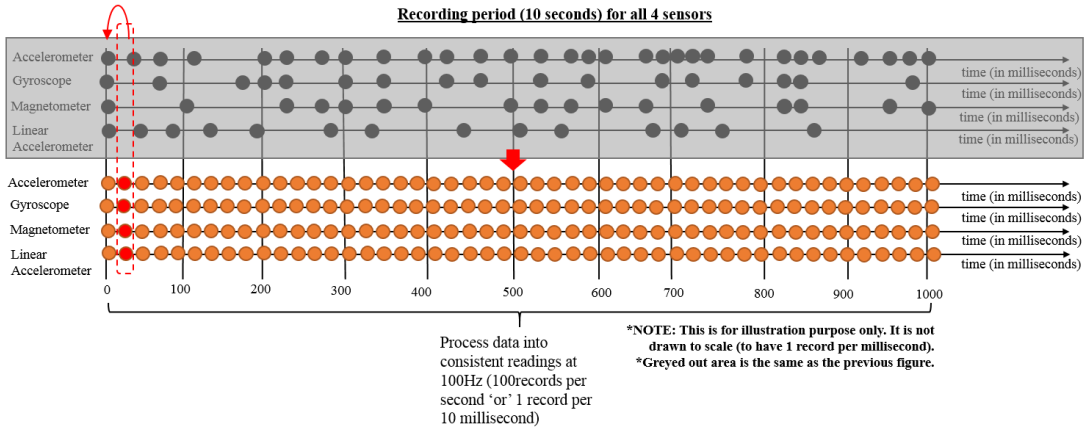


Figure 12 Pre-processing Step One

Figure 12 shows the data pre-processing step one – regular sampling of sensory data. The orange dots refers to the sampled data after pre-processing step one was completed. To achieve a 100Hz frequency rate, if there is no data readings (data value did not change) for the next ten milliseconds (shown in red dots of Figure 12), the sensory readings would follow the previously known ones. An example with data values is shown in Table 3.

Original raw data			Pre-processed data	
Timestamp	Sensor Value		Timestamp	Sensor Value
1391425997242	2.340		1391425997242	2.340
1391425997551	2.356		1391425997252	2.340
* NOTE: Timestamp is represented in UNIX timestamp in this table, and last three digits denotes milliseconds (ms).			1391425997262	2.340
			1391425997552	2.356
			1391425997562	2.356

10ms interval because 100Hz

Table 3 Illustration of Pre-processing Step with Sensors Data

The data was sampled in fixed-width sliding window at pre-processing step two. An illustration for pre-processing step two is as follows.

Table 4 shows the sensory data for five hundreds training motions over ten seconds each in a CSV file format. As it was pre-processed into 100Hz in pre-processing step one, there are one thousand data value for each sensory data over ten seconds. One instance in Table 4 refers to one data recording period over ten seconds. There are five hundred instances as five motions were trained hundred times each.

Sensor X-axis (one axis individually)												
ID/# instances	1	2	.....							1000	Class	
1st instance												Over 10 sec (100Hz)
2nd instance												
⋮					⋮					⋮		
500th instance												

Table 4 One Axis Sensor Data Collected over 10 Seconds at 100Hz

As there were only five hundred instances after one hundred instances of training for every motion, step two of pre-processing step was performed. Data was sampled at fixed-width sliding window (for example, two seconds with 50% overlap).

Sensor X-axis (one axis individually)											
ID/# instance	1	2	.....						200	Class	
1st instance											← 0th to 2nd second (200 readings @ 100Hz)
2nd instance											← 1st to 3rd second (200 readings @ 100Hz)
⋮			⋮						⋮		100instances *5 motions=500
4000th instance											← 500 * 8 = 4000 instances

8 because there will be 8 sets of two seconds interval…

< 0th, 1st, 2nd >, < 1st, 2nd, 3rd >, < 2nd, 3rd, 4th >, < 3rd, 4th, 5th >, < 4th, 5th, 6th >, < 5th, 6th, 7th >, < 6th, 7th, 8th >, < 7th, 8th, 9th > second. Thus, there is 8x more instances.

Table 5 Sampled at Two Seconds Interval and 50% Overlapped

Table 5 shows that data was sampled at fixed-width sliding window of two seconds with 50% overlap. Hence, each instance in Table 4 was chopped into two hundred data values (two seconds interval @ 100Hz) and eight times more instances were seen as compared to the number of instances before pre-processing step two were conducted. This pre-processing step was also useful as it transformed ten seconds of sensory readings into two second instances. This means that real time recognition could be carried out after two seconds of testing, and motions could be recognized as quickly as possible.

3.4 Design and Implementation of Hidden Markov Model

HMM is the technique used to recognize motions in this project. This section describes how HMM was developed. HMM was developed using Jahmm provided by Java. Jahmm takes in a vector of sequential data values as inputs, process them until the highest probability sequence best fits the vector of sequential data values that was passed in. A recognized motion is returned.

3.4.1 Inputs for HMM

The inputs for HMM is the pre-processed CSV file. Figure 13 illustrates the process of creating a HMM after data have been pre-processed.

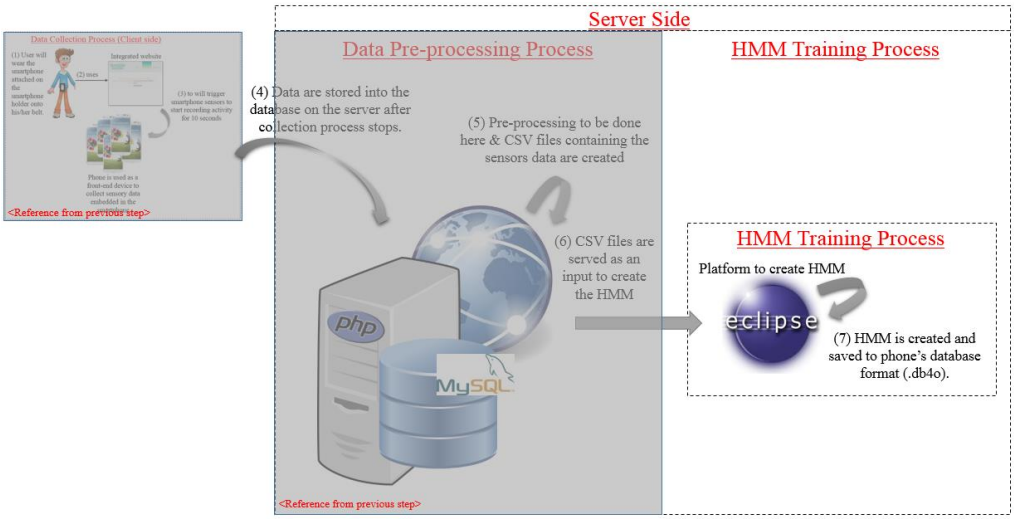


Figure 13 Training of HMM Process

Assume that all three axes of a sensor were put in as one input, each instance in the CSV file will contain the triaxial sensory data as well as the class (motion) shown in Table 6.

Sensor X Y Z all in 1 file as input file

Sensor X at col 0, 3, 6… ID%3==0 where ID start from 0 instead of 1
Sensor Y at col 1, 4, 7… ID%3==1 where ID start from 0 instead of 1
Sensor Z at col 2, 5, 8… ID%3==2 where ID start from 0 instead of 1

ID/# instances	1	2	.....	600	Class	
1st instance						← 0th to 2nd second (600 readings @ 100Hz)
2nd instance						← 1st to 3rd second (600 readings @ 100Hz)
⋮			⋮			100instances*5activities=500
4000th instance						← 500 * 8 = 4000 instances

Table 6 Illustration of Input File for HMM

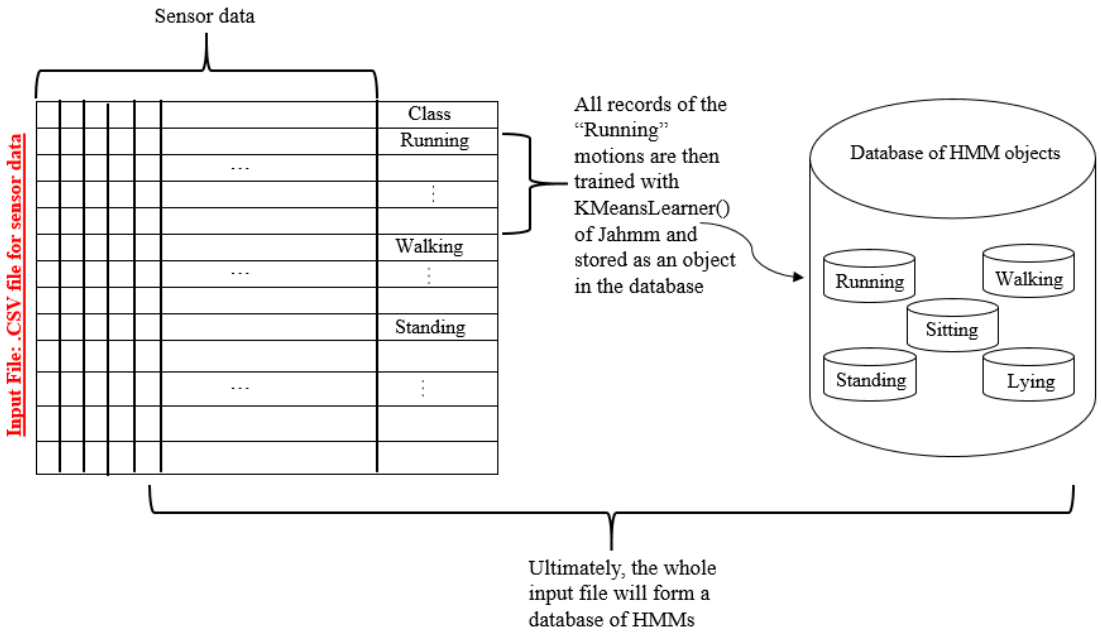


Figure 14 An Overview of How data Are Extracted, Trained and Saved

As the input file contained many rows of data for every motion, these rows belonging to each motion were then extracted out, trained and stored individually, as shown in Figure 14. Every motion trained became an object that was stored into the database of HMMs in .db4o format and every sensor's CSV file trained will be saved into one .db4o format shown in Figure 15. Combined sensors values here refers to the values of Accelerometer, Linear Accelerometer and Gyroscope as they are considered as "more reliable" sensors (see Table 8).

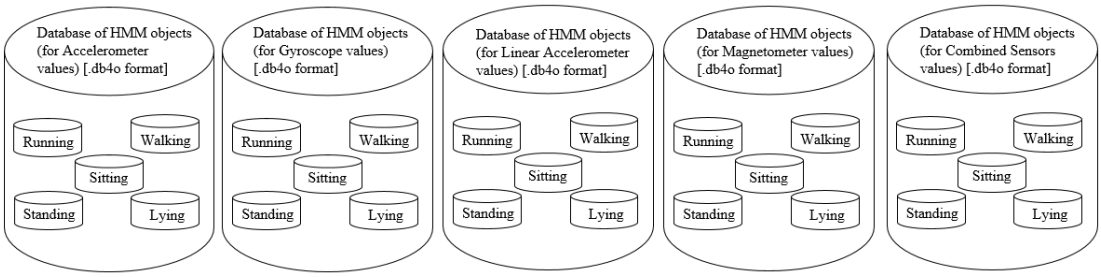


Figure 15 Database of HMM Objects

Hence if for example, four sensors were to be used to recognize motions, each sensor data will need to be compared against the four different databases.

The HMM was created using Jahmm [18] built on Java platform. Jahmm provides the module KmeansLearner which was used to train the HMM.

The KmeansLearner module adopts a traditional K-means algorithm method and finds a suitable HMM that models the set of inputs (observation sequence).

The KmeansLearner module also included the Viterbi algorithm to adjust the parameters of the model in order to maximize $P(O,I|\lambda)$, where O is the observation sequence, I is the sequence given by the Viterbi Algorithm, and λ is the model's parameters.

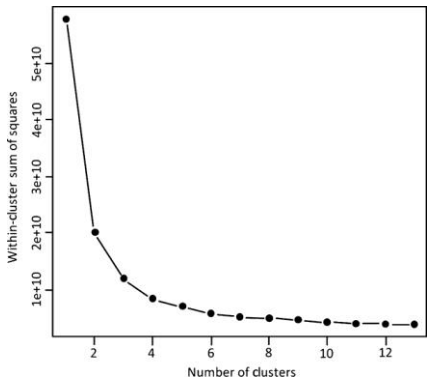


Figure 16 Determining K for K-means

The best number of clusters is at the elbow between the number of clusters and the mean sum of squared error according to [11] and this would allow for the most accurate results without over-fitting the clusters because the smaller the clusters, the smaller the sum of squared error, and the easier it is to over-fit. Hence the K used in this K-means algorithm is specified to be six according to Figure 16 to achieve optimized solution.

Figure 17 shows an overview of Jahmm KmeansLearner module used in this project.

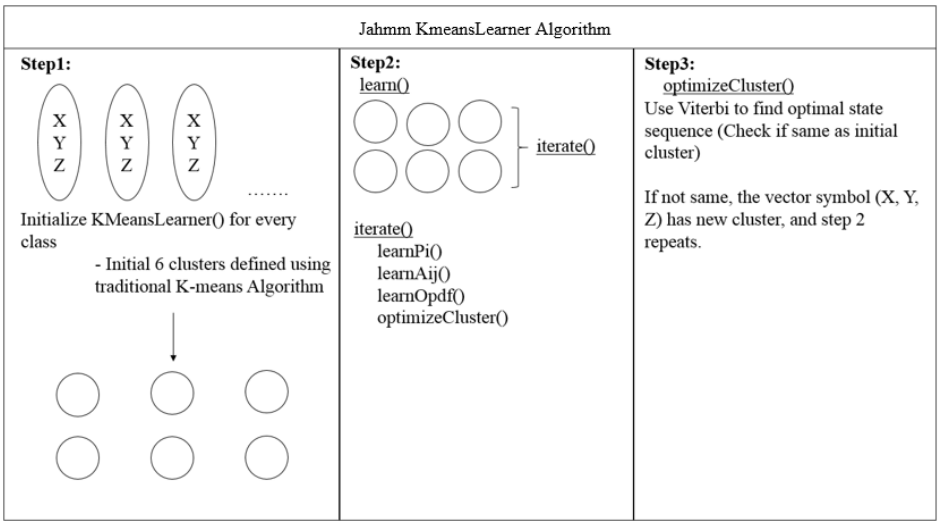


Figure 17 Overview of Jahmm KmeansLearner

3.4.2 Brief Description of Jahmm

This section will bring you through a brief idea how Java implemented it's HMM.

Step1: Clustering of data using K-means algorithm

Firstly, the K-means algorithm will perform one iteration of the K-means algorithm. In each iteration, a new HMM is computed using the current clusters, and the clusters are re-estimated using the HMM.

Step2: Training of HMM

The initial probabilities (learnPi()), state transition probabilities (learnAij()), the mean vector, covariance matrix for each state, the distribution of the probability of observing each symbol on each state were calculated.

learnPi(): Gets the initial probability of every clusters

Every vector belongs to one cluster. For every sequence, get the first vector of sequence and adds one (count) to the cluster which the first vector belongs to. This will get the initial probability of each cluster in the HMM where count divides by the sequence size.

learnAij(): Gets the state transition of clusters

Every vector belongs to one cluster. For every vector in the sequence, calculate the count of the cluster in which the current vector and the next vector in sequence belong to. The state transition probability of the HMM is then set by dividing the count with the sums of the total state transition probabilities.

learnOpdf(): Gets the observation probability distributions associated to the states

This method will build the observation probability distributions associated to the states of the HMM. Each observation vector is given an initial mean and variance of each state.

Step 3: Optimizing HMM

Using the probabilities and matrices calculated, an optimal state sequence *I* is found according to the Viterbi Algorithm for each training. If an observation symbol vector has an optimum state different from its initial state, this observation symbol vector is reassigned to a new state; if any vector was reassigned, the probabilities and matrices have to be re-calculated. Otherwise, the algorithm stops.

After the algorithm stops, the optimized HMMs are returned and stored into database format (.db4o). The optimized HMMs are stored as objects.

3.5 Saving HMM and Java Classes to Server

A Java class containing methods to test temporal sequence of testing data was created. This module was saved in .jar file format so that it can be processed outside Eclipse environment. The saved HMMs and Java classes were stored in the server for motion recognition. Figure 18 shows a complete process of training a HMM, and individual parts of the complete process were discussed (see Sections 3.2, 3.3 and 3.4).

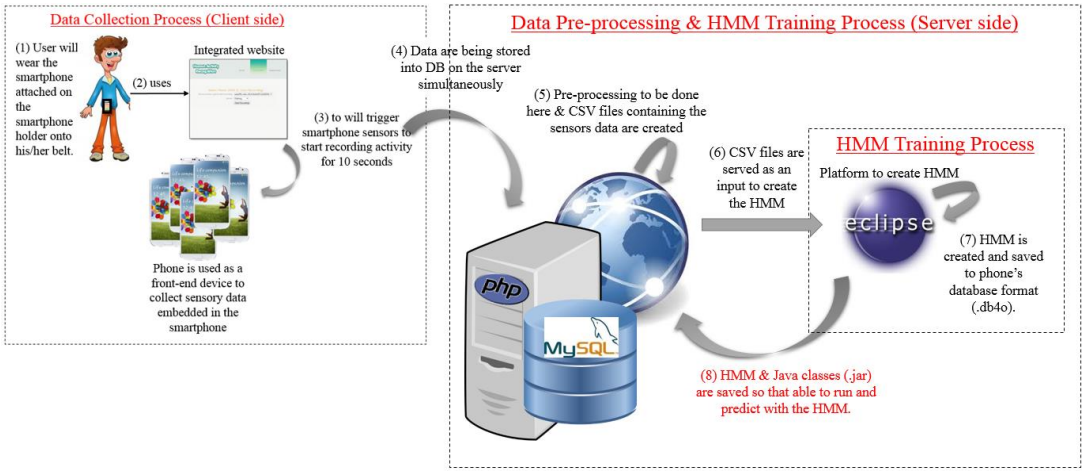


Figure 18 Complete Process of HMM Training

3.6 Enhancing on Motion Transition

The motions are categorized into lower-level motions and higher level motions. The lower-level motions include "Walking", "Running", "Sitting", "Lying", "Standing", and the higher-level motions include "Stand up", "Fall", "Lie down", "Sit down", were built upon the lower-level motions. Figure 19 and Table 7 shows the breakdown of lower-level and higher-level motions.

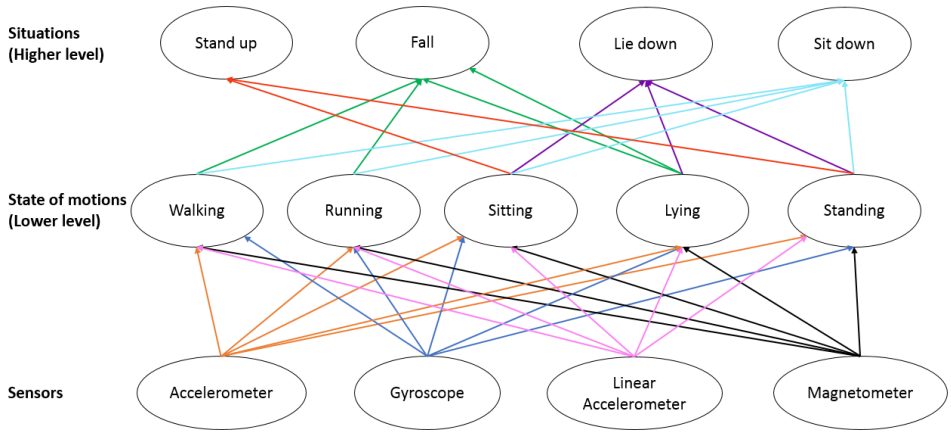


Figure 19 Breakdown of Lower-level Motions to Higher-level Motions

Situation	From Lower-level motion	To Lower-level motion
Stand up	Sitting	Standing
Fall	Running	Lying
	Walking	Lying
	Standing	Lying
Lie down	Sitting	Lying
	Standing	Lying
Sit down	Running	Sitting
	Standing	Sitting
	Walking	Sitting

Table 7 State Transitions in Table Form

In [8], Decision Tree (DT) together with SMA (see Equation 8) were used to recognize the motion "Fall".

Therefore, a rule-based system incorporating DT and SMA was constructed to identify the higher level motions. The rule-based system obtained the motions recognized by the HMM as inputs and conducted a rule-based reasoning to detect the higher-level motions. This means that the rule-based system relied on the correctness of the HMM result. If the HMM results returned a wrong motion, then the rule-based system would be error propagated.

The motions were collected at every time interval the user defined during testing phase. For example, the user defined the time interval to be two seconds. Therefore, for every two seconds, new testing data was recorded and sent to the database simultaneously. Each time the record was received at the server-side, an array of five would be instantiated as shown in Figure 20.

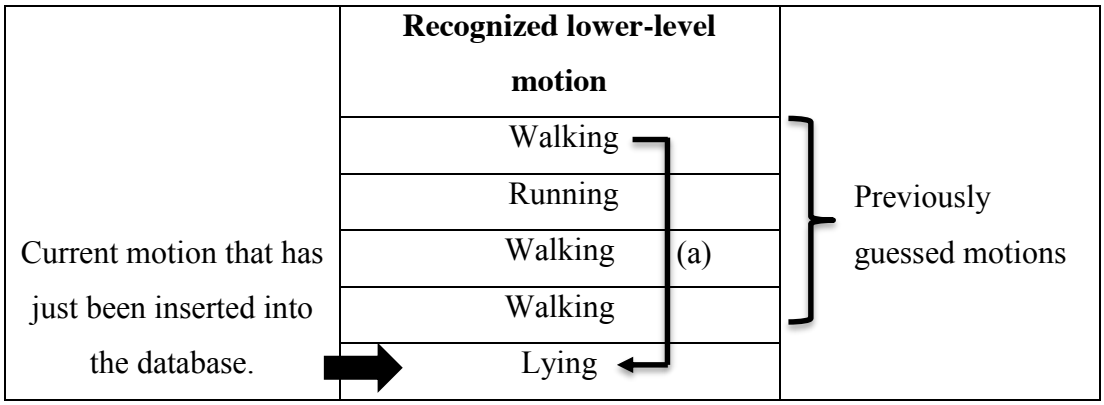


Figure 20 Array of 5 Lower-level Motions to Determine Higher-level Motions

The array consists of four previous records with respect to the current one. The window size is set to five because according to observations, even though the motion transitions happened in less than one second, the sensory data collected will be corrupted for at least three seconds (see Figure 21). Hence in my opinion, keeping window of five motions is an optimal size.

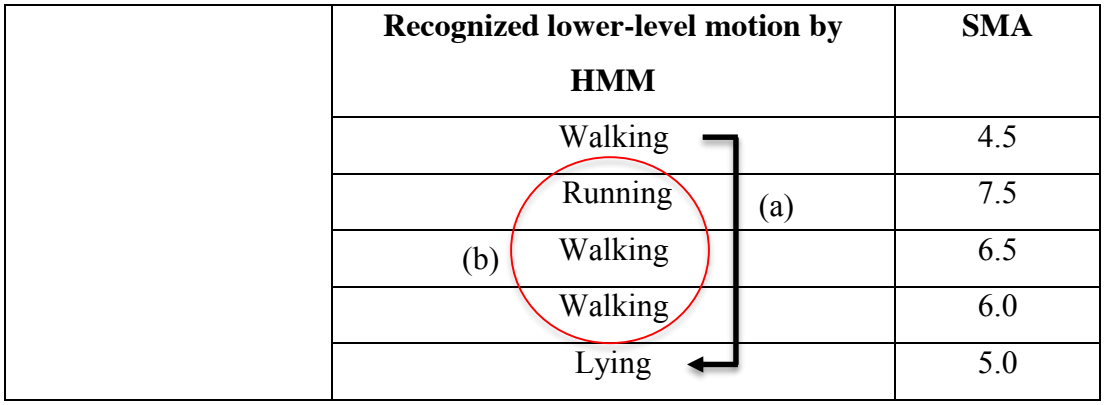


Figure 21 Corrupted Motions for Motions Transition

At each processing step, the first item in the array would be compared against the motion that was just recorded as shown in Figure 21(a).

Motions shown in Figure 21(b) are observed during motion transitions. These motions were known as corrupted motions. Corrupted motions were misclassified motions as they were not trained for the HMM. Therefore, the rule-based system was used here to ensure that the misclassified motions are classified correctly. SMA (see Section 3.1.4) was used to determine the motions.

The comparison logic will be as follows.

An assumption is made here. Transitions from one lower-level motions (Standing, Walking, Running, Sitting, Lying) to another have to be constant for at least the window size defined by the user, two seconds for example. The user should remain in "Walking" position for two seconds, followed by the "Lying" position for two seconds then only will the rule-based system be able to detect accurately as "Fall". This is because the HMM needs time to correctly recognize the lower level motion and return the correct guess, then only will the rule-based system return an accurate motion transitions since the rule-based system relies on correctness of the HMM result.

For "Fall":

If (first item in array == 'Walking' or 'Running' and last item in array == 'Lying'){

- check out the SMA peak (highest number for all SMA in array) of all items in the array.

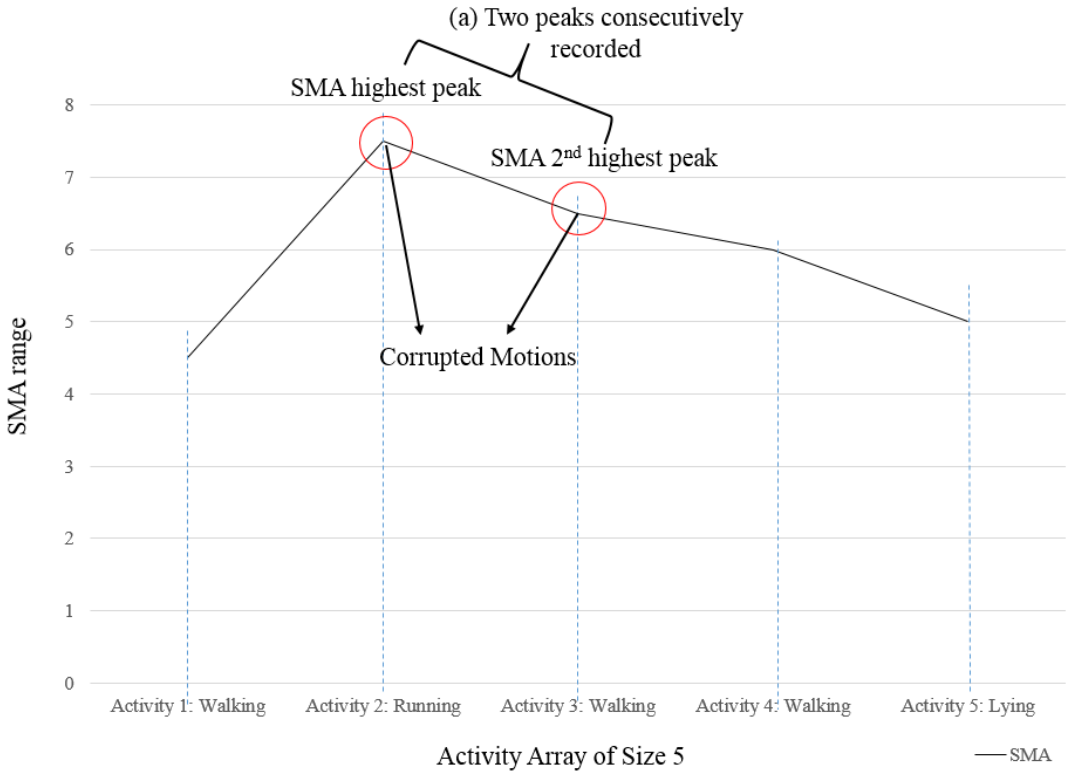


Figure 22 SMA Peaks for Higher-level Motions

- if the two peaks happen to be in one after another [8], then a "Fall" is given to the motions with the SMA peaks

}

Sometimes there may still be some noise due to sensor lags. Hence, smoothen the motion for "Fall"

If (first item in array == 'Fall' and last item in array == 'Lying'){

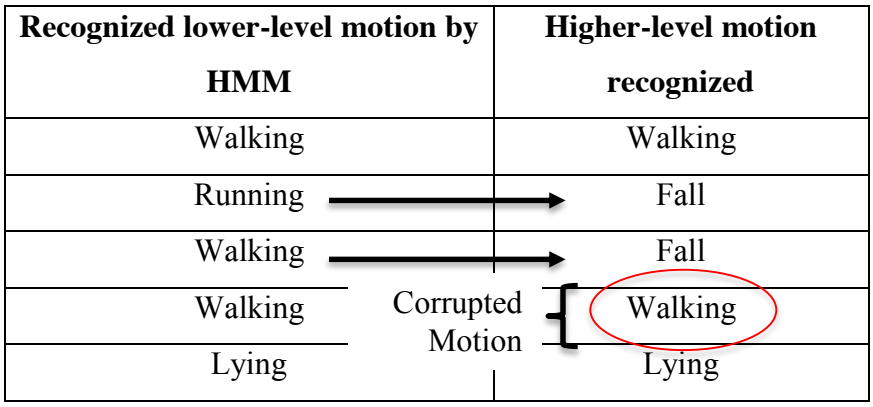


Figure 23 Corrupted Motion Exists Even After Rule-based System Analysis

- check if any array in between the two motions are not of the two motions. If it is not, then it is classified as 'Fall'

}

--- Using the illustrations on Figure 22 and 23, the rest of the motions are also processed that way ---

For "Sit Down":

If (first item in array == 'Standing' or 'Walking' or 'Running' and last item in array == 'Sitting'){

- check if any array in between the two motions are not of the two motions. If it is not, then it is classified as 'Sit Down'

}

For "Lie Down":

If (first item in array == 'Sitting' or 'Standing' and last item in array == 'Lying'){

- check if any array in between the two motions are not of the two motions. If it is not, then it is classified as 'Lie down'

}

For "Stand up":

If (first item in array == 'Sitting' and last item in array == 'Standing'){

- check if any array in between the two motions are not of the two motions. If it is not, then it is classified as 'Stand up'

}

As data are collected in a "clean" manner with little noise, it is then very sensitive to outliers. Hence, when it detect a sudden change in sensor readings, it might recognize a wrong motion. Therefore, smoothing of motions is needed.

Smoothing low-level motion (Lying)

If (last item in array == 'Lying' and last second item in array != 'Lying' and last third item in array == 'Lying'){

- classify the last second item as 'Lying'

}

Smoothing low-level motion (Sitting)

If (last item in array == 'Sitting' and last second item in array != 'Sitting' and last third item in array == 'Sitting'){

- classify the last second item as 'Sitting'

}

Smoothing low-level motion (Standing)

If (last item in array == 'Standing' and last second item in array != 'Standing' and last third item in array == 'Standing'){

- classify the last second item as 'Standing'

}

Smoothing low-level motion (Running)

If (last item in array == 'Running' and last second item in array = 'Walking' and last third item in array == 'Running'){

- classify the last second item as 'Running'

}

Smoothing low-level motion (Walking)

If (last item in array == 'Walking' and last second item in array = 'Running' and last third item in array == 'Walking'){

- classify the last second item as 'Walking'

}

3.7 Cross-validations

After training the HMMs, cross-validation was used for estimating the performance of the built HMM. This was implemented via the Java platform through the Eclipse environment. Figure 24 shows the whole process of how data are cross-validated.

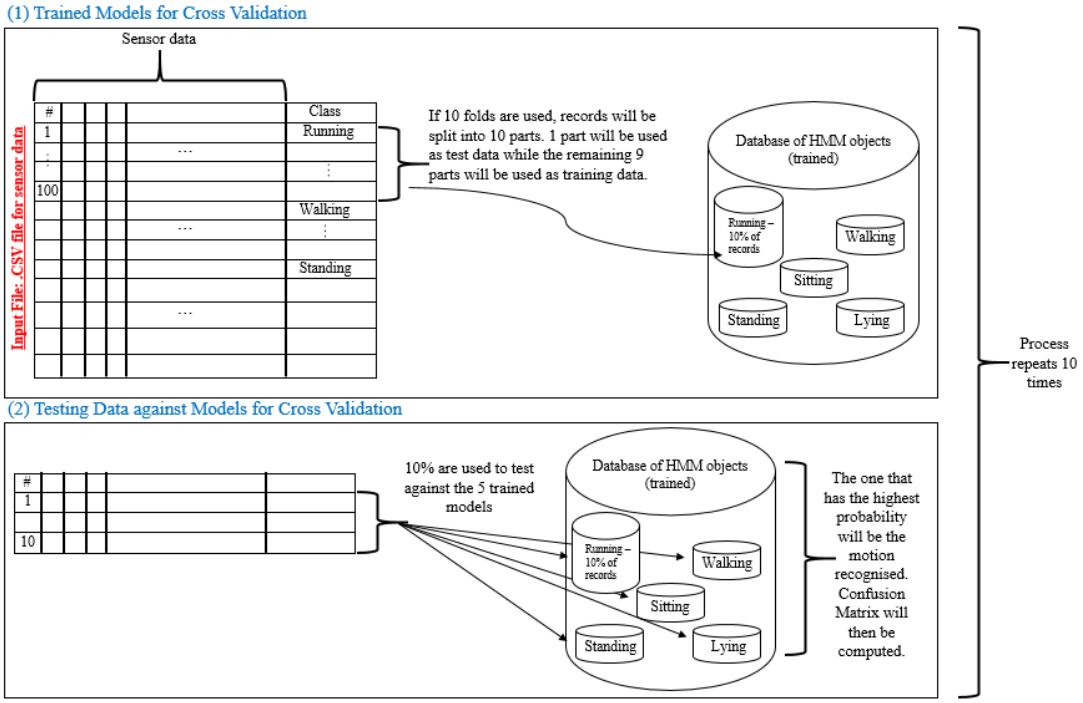


Figure 24 Cross-Validation Process

3.8 Confusion Matrix

Figure 25 shows a confusion matrix on the of Accelerometer sensory data pre-processed at fixed-width sliding window of two seconds interval and sampled at 100Hz. This confusion matrix was built with the results from ten-fold cross-validation. Rows represents the actual class and columns represents predicted class.

Confusion Matrix (2Seconds_100Hz) - Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	797	3	0	0	0	99.63
	Running	0	800	0	0	0	100.00
	Sitting	0	32	768	0	0	96.00
	Standing	0	14	0	758	28	94.75
	Walking	0	0	0	0	800	100.00
	Overall Accuracy: 98.10%						

Figure 25 Example of Confusion Matrix on the Lower-level motions

Figure 25 shows that "Running" motion is easily confused since it is always recognized as "Running" for "Lying", "Sitting" and "Standing" motions. This may be due to firstly, the huge range of accelerometer values that "Running" motion has, and secondly, the way each motion was trained. The motions were trained in a "clean" manner such that when the individual was training the motion "Lying", he/she was lying and facing up. Therefore, if during testing phase, that individual were to toss and turn, the HMM would not be able to recognize the motion.

3.9 Motion Testing in Real-time

This section discusses the methodology of conducting real time motion testing.

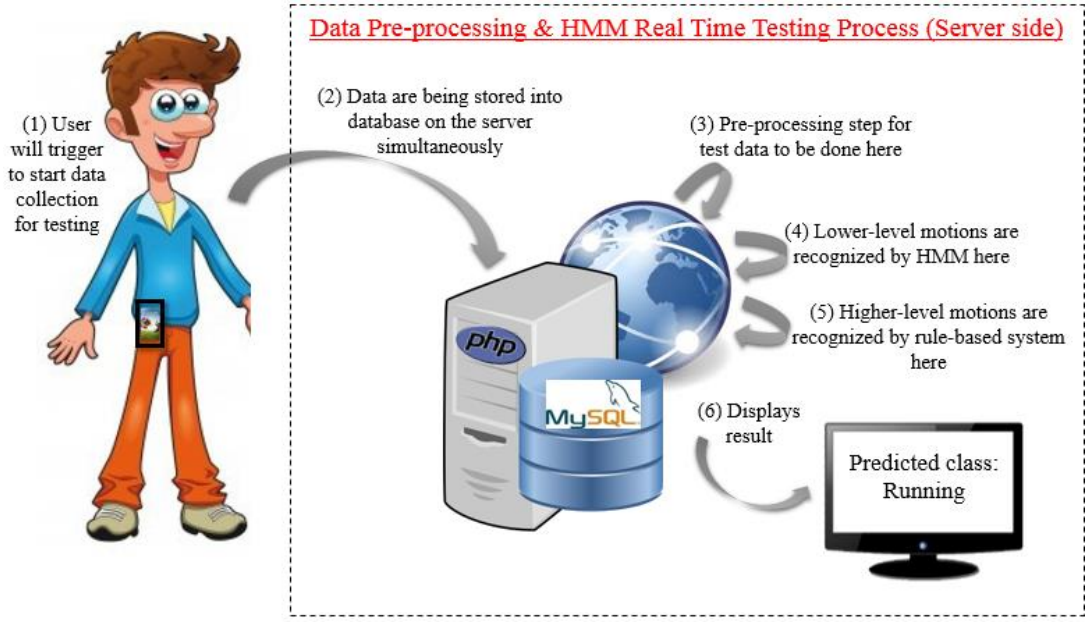


Figure 26 Overview of Real-time Testing Process

To do real time testing, the 'Start Recording' button on the application would be triggered by the user and the application would then start to record the sensory data based on the time interval defined by the user. For instance, if the user defined the time interval as two seconds, the sensory data would be sent to the server in two second segments. To simultaneously recognize the user's activity, a batch file which consisted of the pre-processing and motion recognition steps for the testing data was run on the server. The pre-processing step for testing data was similar to the training step so that testing data corresponded to the trained model. The recognized motion was then updated to the database and displayed on the integrated website.

3.10 Implementation of Integrated Website for Training and Testing of Models

A website was created to serve as an integrated site for training and testing of HMM. The website is currently running in the local server and can be put into public domain easily.

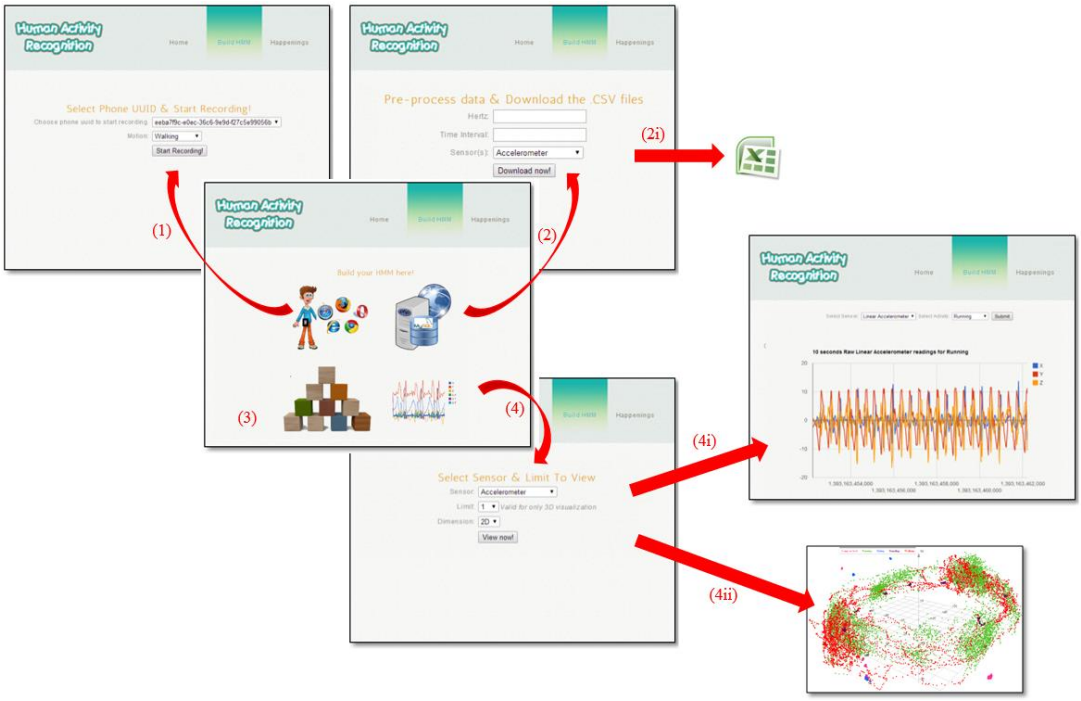


Figure 27 Integrated Website for Building of HMM

Figure 27 shows the integrated website for building of the HMM. Step (1) shows the website interface for training sensory data for future HMM. The integrated website triggers the smartphone to record the sensory data for consecutive ten seconds. This sensor data is then sent back to the server to be part of the trained data. Step (2) shows the website interface for pre-processing of sensory data according to user's definition. Sliding window size and sampling frequencies are defined here. The output files for pre-processing step are the different .CSV files. At step (3), user would be able to train a new HMM upon a click on the button shown. Users can also view the sensory data in 2D and 3D space (for visualization purpose).

Figure 28 shows a page where the real time motion recognition system will return its result. This page refreshes itself after every few seconds so as to keep the records updated.

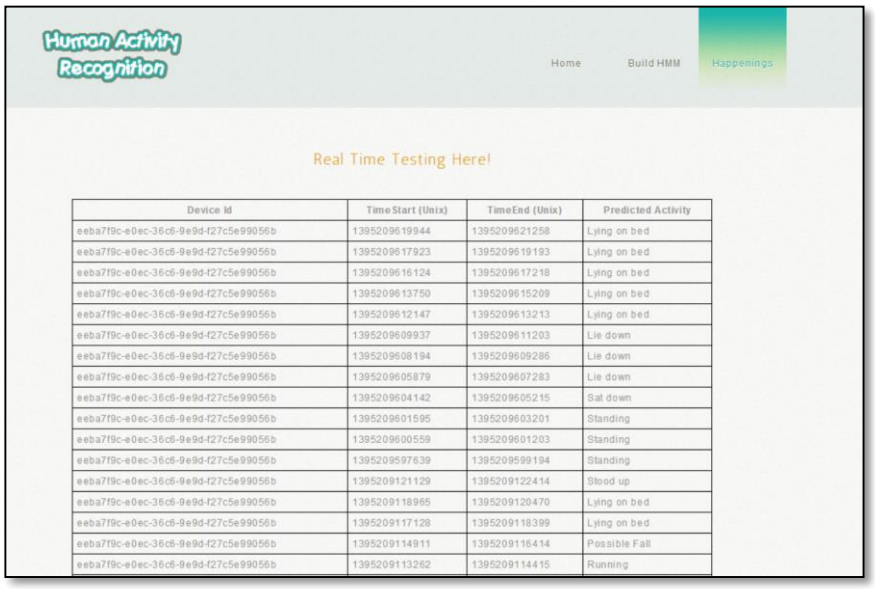


Figure 28 Webpage Where Real-time Testing of Data Is Displayed

4. Experimental Results and Analysis

This chapter describes the experiments that have been conducted and analysis of the experiments will be discussed. The experiments carried out include analysing the performance of various HMMs, analysing summations of probabilities for individual HMMs, analysing best combinations for pre-processing steps, and lastly, the analysis of real-time testing in terms of performance and time taken.

4.1 Performance of HMM

Data for four sensors were collected, trained and tested. This experiment was conducted to determine which sensors gave the best performance.

The inputs for this experiment were the sequence of vectors of axes *X*, *Y*, *Z* sensory data, one class – motion, and various HMM models stored in .db4o format (see Figure 15 and Table 6). All training data were 50% overlapped and pre-processed into different sampling frequency and sliding window interval shown in Table 8. The method used to determine the performance of the various HMM was ten-fold cross-validation.

(Refer to Appendix A for the insights of individual Cross-Validation matrix)

Sensors Models/Time & Sampling Frequency	**2 seconds**		**1 second**		**0.5 second**	
	50Hz	**100Hz**	**50Hz**	**100Hz**	**50Hz**	**100Hz**
Accelerometer	98.10%	98.10%	97.87%	-	97.55%	-
Gyroscope	89.13%	88.02	84.91%	-	79.68%	-
Linear Accelerometer	99.70%	**99.70%**	**99.47%**	-	98.66%	-
Magnetometer	60.30%	59.78%	58.61%	-	56.73%	-
Combined Sensors	**99.98%**	98.98%	99.07%		**99.11%**	
	***Combined sensors** include Accelerometer + Gyroscope + Linear Acc.					

Table 8 Comparison of Sensors Models Accuracy

Table 8 shows the accuracy of the four individual sensors used in this project. The table consists of the various sensors models that have been trained, the time interval for the fixed-width of sliding window is sampled at, and the sampling frequencies for every data collected.

From the sensors models accuracy returned by ten-fold cross-validation performance measure, the linear accelerometer reading for all different sliding window's time intervals and sampling frequencies showed the most promising accuracy. Follow by accelerometer, and then gyroscope. Magnetometer readings had very low accuracy as compared to the other three sensors. This might have been be due to the fact that Magnetometer readings were highly affected by the magnetic fields and instruments around the smartphone during the data collection process.

Table 8 also shows the accuracy of "Combined Sensors". Combined sensors include Accelerometer, Gyroscope and Linear Accelerometer as they were the more reliable sensors as seen from Table 8. Magnetometer was not included as it would have negatively affected the accuracy of the combined sensors model given the Magnetometer's individually poor result. Hence, the three reliable sensors with axes *X*, *Y*, *Z* were combined into one input vector of nine axes, trained and tested, and the result is as shown in Table 8.

4.2 Performance of Working with Individual HMMs

There are several ways to determine the best motion using the different sensors models. In this experiment, probabilities of each sensors models were summed up and the highest probability motion was determined as shown in Table 9.

The inputs for this experiments were the pre-processed real-time testing data with fixed width of two seconds sliding window, sampled at 50Hz. The tested motion was "Sitting" and the returned probabilities were in Natural logarithm form.

Sensory Values ↓ Models →	Probabilities					
	Lying	**Running**	**Sitting**	**Standing**	**Walking**	**Probable Motion**
Accelerometer	Infinity	-442.7	-517.7	Infinity	-852.1	Running
Linear Acc.	-94.7	-597.2	423.4	295.1	-185.8	Sitting
Gyroscope	1051.1	-427.5	1139.3	1098.2	-134.8	Sitting
Overall Prob.	Infinity	-1467.4	1045	Infinity	-1172.7	

Table 9 Example of Probabilities of Different Sensors Model for One Motion

Table 9 shows the returned probabilities of "Sitting" motion tested against three reliable sensors. As discussed in Section 3.4.1, different sensors have their own sensory HMM (Three different HMM in this case). Within each HMM, there were five other objects. For every testing data set collected, different sensory data collected were tested against five objects in the three HMMs. For example, Accelerometer testing data of one motion was compared against five objects (Lying, Running, Sitting, Standing and Walking) in the Accelerometer HMM. The returned value of negative infinity implied that the probability of motion being "Lying" was 0, and for the rest of the values, the higher the value, the more probable the motion class would be. Therefore in the case of Table 9, the returned probable motion was set to be "Running" as "Running" displayed the highest probability for Accelerometer HMM. However, this was not case as the tested motion was actually "Sitting".

Therefore, the sum of probabilities for different sensor models may obtain a more accurate result instead of data from individual sensors.

4.3 Performance of Pre-processing Step

As sensory data collected may not be synchronized for different sensors at different training and testing periods, it would not be accurate to train on these raw data. Hence, pre-processing for training and test data to ensure a consistent frequency was needed.

In order for the test data to be synchronized with the trained models, the pre-processing step performed on the training data would need to be done similarly on the testing data. For example if training data was pre-processed using fixed-width sliding window of two seconds interval, sampled at 50Hz with 50% data overlapping, the test data would also need to be pre-processed at same time interval and sampling rate.

Referencing back to Table 8, the best optimal combination of sensors, time interval for sliding window, sampling frequency was the combined sensors, with two seconds of sliding window at 50Hz with 50% data overlapping with an accuracy of 99.98%.

In practise, having to choose the optimal combination of sensors was not straightforward. This was because if one second sliding window was used, the motions would be recognized twice as fast as compared to a two second sliding window. However, the combined sensors accuracy for one second 50Hz did not perform as well as Linear Accelerometer's one second 50Hz model, and because the rule-based system was built upon the returned result, it was better to use the model that can return a better accuracy result.

Another reason for using a combined sensors model instead of individual sensor models was the time taken to recognize one motion. If testing data was to be tested against the various individual sensor models, the time taken would be three times (if three individual sensors are used) longer as compared against a combined model. However, although the time taken is much lesser with the combined model, the disadvantage of using a combined model was that if the model recognized the motion wrongly, it had no other models to compare with.

4.4 Real-time Testing of Lower-level Motions

Besides cross-validating the performance of models, real-time testing was also conducted to measure the performance of the models. This experiment was conducted to determine the accuracy of the mode when real-time testing was performed.

The inputs for this experiment were the combined sensors model together with the pre-processed real-time testing data sampled at 50Hz with one second sliding window interval. The motions collected and tested were continuous motions without state transitions. The test size was one hundred samples for each motion and the user was a twenty-three year old female.

Results for one hundred testing samples		
Motion	**Accuracy**	**Mistaken Motion(s)**
Lying	100%	-
Sitting	99%	Running
Standing	100%	-
Running	99%	Walking
Walking	95%	Running

Table 10 Performance for Recognition of a Twenty-three Year Old Female

The returned result for this experiment was quite accurate during real-time testing. This may have been because real-time testing was done on the same person who have trained this combined sensors HMM.

4.5 Real-time Testing of Lower-level Motions on Another User

Real-time testing on the same user who trained the model showed that the model had a high accuracy of accuracy when tested on the same user. This experiment was conducted to measure the performance for testing HMM on another user. The inputs for this experiment were the combined sensors model, pre-processed real-time testing data sampled at 50Hz with one second sliding window interval. The motions tested were continuous motions without state transitions. The tested size was one hundred samples for each motion and the user was a fifty year old female.

Results for one hundred testing samples		
Motion	**Accuracy**	**Mistaken Motion(s)**
Lying	98%	Running
Sitting	85%	Running
Standing	97%	Walking
Running	99%	Walking
Walking	76%	Running

Table 11 Performance for Recognition of a Fifty Year Old Female

Real-time testing of lower-level motions for another user was carried out. Table 11 shows the accuracy of each motion respectively.

From the result in Table 11, and compared against the accuracy of Table 10, the combined sensors model was concluded to be optimized to the user it was trained on as the model did not perform as well for the other user.

The "Sitting" motion was often mistaken for the "Running" motion. This was because of the way the data was trained. The data collection process was carried out such that when the motion was trained, it was trained with minimal movements in the other parts of the body. For example, when the "Sitting" motion was trained, the user only sat there with minimum movement in the upper body. Thus, if the user moved slightly in his/her upper body while sitting down, the motion would have a high chance of being misclassified.

4.6 Real-time Testing of Higher-level Motions

This experiment is to determine the performance of the rule-based system. The inputs for the experiments were the combined sensors model and the pre-processed real-time testing data sampled at 50Hz with fixed-width of sliding window at two second intervals. The different higher-level motions were the motion transitions in Table 7. The experiment was tested on twenty samples for each motion transitions and was tested on a twenty-three year old female. The actual times when motion transitions took place were recorded and compared against the time when the server first recognized the motion transitions. If the server updated the motion transitions at the testing data with timestamp ± 2 seconds, the motion transitions were considered correctly classified.

The rule-based system was built upon the HMM. Hence it relied on the correctness of individual motions recognized by the HMM. If the HMM was unable to provide an accurate motion, then the rule-based system would be error propagated. If the HMM recognized the lower-level motions accurately, the rule-based system performed well on the motion transitions. The accuracy of each motion transition is shown in Table 12.

Results for twenty testing samples			
Higher-level Motion	**From motion**	**To motion**	**Accuracy**
Sit down	Standing	Sitting	90%
	Walking	Sitting	75%
	Running	Sitting	85%
Stand up	Sitting	Standing	75%
Fall	Walking	Lying	75%
	Running	Lying	60%
	Standing	Lying	*
Lie down	Sitting	Lying	70%
	Standing	Lying	80%
Average of all higher-level motions except *			**76.25%**
* To be implemented in future with location and duration.			

Table 12 Performance of Rule-based System for Higher-level Motions

The higher-level motion "Fall: Running to Lying" motion had a lower accuracy of being recognized. This was because when performing the "Fall Down" motion, the phone's position shifted. Thus HMM was unable to recognize a "Lying" position after the fall, and hence the rule-based system was unable to recognize the motion transition as a "Fall". Therefore, in order to increase the recognition accuracy, "Lying" motion would have to be trained with more variations.

In addition, "Fall" from "Standing" to "Lying" could not be recognized at the moment as it shared similar "from" and "to" motions with "Lie down". The main reason why the rule-based system could not differentiate the "Lying" motion was that the sensors could not recognized if the subject was "Lying-on-bed" or "Lying-on-floor". Thus, in order to determine this motion, other information such as location sensor and duration between motions can be used. However, due to time-constraint, this experiment could not be conducted in this project.

4.7 Time Taken for Different Tasks

This experiment is to determine if different computer systems with different specifications will affect the performance of each task. Initially, all training and testing processes were conducted on the laptop. However, the returned time taken for each task on the laptop was massive. Thus, this experiment was inspired and two systems (a laptop and a desktop computer) were used in this experiment. The specifications for the two systems are shown in Table 13.

Desktop Computer Specifications	Laptop Specifications
CPU Speed: 3.20GHz RAM: 8.00 GB No. of cores: 4	CPU Speed: 1.80GHz RAM: 4.00 GB No. of cores: 2

Table 13 Systems Specifications

4.7.1 HMM Training and Performance Measures Process

HMM training process will have to be re-conducted to account for any additions in new training data for the HMM. Table 14 shows the time taken to train one HMM and the time taken to measure the performance of one HMM using cross-validation method.

Systems	Time taken to train one combined sensors HMM
Laptop	21 minutes
Desktop Computer	8 minutes
Systems	**Time taken to measure performance of one HMM (Cross-Validation Process)**
Laptop	> 180 minutes
Desktop Computer	45 minutes

Table 14 Time Taken for HMM Training and Performance Measure Process

The results in Table 14 shows that if one wishes to re-conduct the training and performance measure process, he/she is recommended to use a desktop computer instead of a laptop of similar specifications.

4.7.2 Real-time Testing Process for Lower-level Motions

From the training process, desktop computer has definitely shown better performance than laptop. Hence, this section only describes the time taken for real-time testing process on a desktop computer as it is believed that the laptop would not be able to handle pre-processing and recognition without generating a backlog.

In UNIX time stamp				In seconds		
Recording start time	Recording end time	Time testing data transmitted to server	Time lower-level motion is recognized	Transmission Latency	Time for pre-processing & recognition process	Total time taken from end of recording to recognition process is completed
1395425641	1395425641	1395425642	1395425643	1	1	2
1395425642	1395425643	1395425644	1395425645	1	1	2
1395425644	1395425645	1395425646	1395425647	1	1	2
1395425646	1395425647	1395425648	1395425649	1	1	2
1395425648	1395425649	1395425650	1395425651	1	1	2
1395425650	1395425651	1395425652	1395425653	1	1	2
1395425652	1395425653	1395425654	1395425655	1	1	2
1395425654	1395425655	1395425656	1395425657	1	1	2
1395425656	1395425657	1395425658	1395425659	1	1	2
1395425658	1395425659	1395425660	1395425661	1	1	2
1395425660	1395425661	1395425662	1395425663	1	1	2
1395425662	1395425663	1395425664	1395425665	1	1	2
1395425664	1395425665	1395425666	1395425667	1	1	2
1395425666	1395425667	1395425668	1395425669	1	1	2
1395425668	1395425669	1395425670	1395425671	1	1	2
1395425670	1395425671	1395425672	1395425673	1	1	2
1395425672	1395425673	1395425674	1395425675	1	1	2
1395425674	1395425675	1395425676	1395425677	1	1	2
1395425676	1395425677	1395425678	1395425679	1	1	2

Table 15 Real-time Testing for Lower-level Motions on a Desktop Computer

Table 15 shows the time taken for real-time testing process conducted on the desktop computer. The time taken for testing data to be sent from the smartphone to the server (transmission latency) took about one second and recognition of motions process took another one second. This shows that lower-level motions were recognized in two seconds after the testing data has been recorded on the smartphone.

The transmission latency for real-time testing process will vary under different data transmission rates, and performance analysis on the time taken for pre-processing and recognition process are to be discussed in the next section.

4.7.2.1 Performance Analysis on the Time Taken for Pre-processing and Recognition Processes

An experiment of twenty samples was carried out to determine the time taken to load and test a HMM for lower-level motions. Before each testing step, HMM has to be

loaded from the disk. Table 16 shows the different time stamps of the testing process for twenty testing samples.

In (hrs:min:sec.ms)			In milliseconds	
Time before loading model	Time after loading model and before recognition process	Time after motion recognition process	Time to load a model	Time to test against HMM (Recognition Process)
16:52:03.008	16:52:03.551	16:52:03.553	543	2
16:52:05.398	16:52:05.859	16:52:05.862	461	3
16:52:07.069	16:52:07.559	16:52:07.562	490	3
16:52:09.068	16:52:09.543	16:52:09.545	475	2
16:52:11.078	16:52:11.551	16:52:11.554	473	3
16:52:13.099	16:52:13.611	16:52:13.614	512	3
16:52:15.161	16:52:15.652	16:52:15.655	491	3
16:52:17.106	16:52:17.561	16:52:17.564	455	3
16:52:19.192	16:52:19.669	16:52:19.672	477	3
16:52:21.212	16:52:21.705	16:52:21.708	493	3
16:52:23.091	16:52:23.564	16:52:23.566	473	2
16:52:25.145	16:52:25.613	16:52:25.617	468	4
16:52:27.242	16:52:27.730	16:52:27.734	488	4
16:52:29.091	16:52:29.562	16:52:29.565	471	3
16:52:31.216	16:52:31.679	16:52:31.682	463	3
16:52:33.111	16:52:33.588	16:52:33.591	477	3
16:52:35.361	16:52:35.857	16:52:35.860	496	3
16:52:37.194	16:52:37.682	16:52:37.685	488	3
16:52:39.117	16:52:39.606	16:52:39.609	489	3
16:52:41.147	16:52:41.616	16:52:41.618	469	2

Table 16 Time Taken for Loading and Testing of HMM on a Desktop Computer

With Table 16, performance analysis is carried out. Table 17 shows the performance analysis of this experiment.

Performance Analysis on Loading and Testing of HMM	**In milliseconds**	
	Average	**Standard Deviation**
Time to load a model	483	19
Time to test against HMM – recognition	3	5

Table 17 Performance Analysis on a Desktop Computer

Table 17 shows that the time taken to test the testing data against the HMM to recognize a motion was 3 milliseconds on average, and the time taken for loading the HMM onto the disk was 483 milliseconds.

As discussed in Table 15, the time taken for pre-processing step and recognition step was around one second, and Table 17 shows the time taken for recognition step was about 483 milliseconds on average. Thus concluding that the time taken for pre-processing step and the updating of motions into the database took the remaining 600 milliseconds.

This experiment showed that the actual time for HMM recognition was almost instantaneous. Loading of the HMM into the disk each time a testing process was conducted was one of the factors that slowed down the entire process.

In order to speed up the process of motion recognition, one way is to omit working on the server database. However, Jahmm model generator does not work well. When loading the saved HMM for testing process on the smartphone, the returned results were completely inaccurate. Hence, a server database had to be used alternatively.

4.7.3 Real-time Testing Process for Higher-level Motions

An experiment of ten motion transitions was conducted to analyse on the performance of the rule-based system implemented. The rule-based system was built on the motion results returned by the HMM.

In Unix Timestamp				In milliseconds	
Recording Start Time	Recording End Time	Time Sent To Server	Time Motion Is Updated	Total Time Taken	Recognized Motion
1395577608	1395577609	1395577609	1395577609	0	Sitting
1395577609	1395577611	1395577611	1395577612	1	Sitting
1395577612	1395577613	1395577613	1395577619	6	Stood up
1395577614	1395577615	1395577615	1395577615	0	Standing
1395577616	1395577617	1395577617	1395577617	0	Standing
1395577617	1395577619	1395577619	1395577619	0	Standing
1395577619	1395577621	1395577621	1395577621	0	Standing
1395577622	1395577623	1395577623	1395577623	0	Standing
1395577624	1395577625	1395577625	1395577625	0	Standing
1395577626	1395577627	1395577627	1395577633	6	Sat down
1395577627	1395577629	1395577629	1395577629	0	Sitting
1395577630	1395577631	1395577631	1395577631	0	Sitting
1395577631	1395577633	1395577633	1395577633	0	Sitting
1395577634	1395577635	1395577635	1395577642	7	Stood up
1395577635	1395577637	1395577636	1395577637	1	Standing
1395577637	1395577639	1395577639	1395577639	0	Standing
1395577639	1395577641	1395577641	1395577642	1	Standing
1395577642	1395577643	1395577643	1395577643	0	Standing
1395577643	1395577645	1395577645	1395577651	6	Sat down
1395577646	1395577647	1395577647	1395577647	0	Sitting
1395577647	1395577649	1395577649	1395577649	0	Sitting
1395577649	1395577651	1395577651	1395577651	0	Sitting
1395577651	1395577653	1395577653	1395577653	0	Sitting
1395577654	1395577655	1395577655	1395577655	0	Sitting
1395577656	1395577657	1395577657	1395577663	6	Stood up
1395577658	1395577659	1395577659	1395577659	0	Standing
1395577660	1395577661	1395577661	1395577661	0	Standing
1395577661	1395577663	1395577663	1395577663	0	Standing
1395577663	1395577665	1395577665	1395577671	6	Sat down
1395577665	1395577667	1395577667	1395577667	0	Sitting
1395577668	1395577669	1395577669	1395577669	0	Sitting
1395577670	1395577671	1395577671	1395577671	0	Sitting
1395577671	1395577673	1395577673	1395577673	0	Sitting
1395577673	1395577675	1395577675	1395577681	6	Stood up
1395577676	1395577677	1395577677	1395577677	0	Standing
1395577677	1395577679	1395577679	1395577679	0	Standing
1395577680	1395577681	1395577680	1395577687	7	Sat down
1395577681	1395577683	1395577683	1395577687	4	Sat down
1395577683	1395577685	1395577685	1395577685	0	Sitting
1395577686	1395577687	1395577687	1395577687	0	Sitting
1395577688	1395577689	1395577689	1395577695	6	Stood up
1395577689	1395577691	1395577691	1395577695	4	Stood up
1395577691	1395577693	1395577693	1395577693	0	Standing
1395577693	1395577695	1395577695	1395577695	0	Standing
1395577696	1395577697	1395577697	1395577697	0	Standing

Figure 29 Performance of Rule-based System

Figure 29 shows the performance of recognition of higher-level. Each lower-level motion took an average of one second to be processed as seen in Section 4.7.2.1. However, Figure 29 shows that higher-level motion took about six to seven seconds on average to be classified as an array of five was used in the implementation (see Section 3.6).

Each time a record was inserted into the database, the rule-based system would initiate the classification method. However, the newer record was always the last in the array of five motions. The classification method would have to classify its first four array items before classifying the fifth item (state transition motion). Hence, time was lost during the waiting of classifications of the previous motions.

Figure 29 also shows the smoothing process (see Section 3.6) of the rule-based system. Smoothing process compares three items in the array, (the previous array item, current array item, and the next array item). Time was also lost during the process of waiting for the classifications of all the array items that were needed for classifications.

4.8 Comparisons with related work

4.8.1 Hidden Markov Model vs K-Nearest Neighbor

In [5], the performance of two classification methods, Naïve Bayes and Clustered K-Nearest Neighbor (KNN), for activity recognition on smart phones were analyzed. Clustered KNN is an enhancement on the usual KNN algorithm with additional pre-processing steps on the different features before its classification step and this improved algorithm has shown a more promising result than Naïve Bayes. Table 18 shows a comparisons of methodologies and different motions available for recognition.

	KNN [5]	**HMM (this project)**
Means of collecting data	Android application and smartphone sensors	Android application and smartphone sensors
Pre-processing step (Sampling frequencies)	Window size of one second and sampling at 50ms	Window size of two seconds and sampling at 50Hz (5ms)
Analysis tool	KNN	HMM
Accuracy	92% (Tested by five people)	99.98% (Self-testing)
Motions recognizable	Running, Walking, Standing, Sitting	Running, Walking, Standing, Sitting, Lying
Higher-level Motions	N.A	Fall, Sit down, Stand up, Lie down Average of (76.25%)

Table 18 Comparisons of HMM with KNN

While the method of data collections was similar, the capabilities of the two models varied greatly. The KNN recognized fewer motions than HMM and also was unable to detect transitions between the main motions. This could be due to the lower sampling frequency resulting in less time to detect the transitions.

However, the main reason for its lower accuracy and motions recognized was most probably due to the intrinsic properties of the classification algorithms. KNN produces distinct clusters for each activity and each activity is fitted into the closest cluster. Therefore, in cases where the motions are not distinct, the accuracy suffers. As transitions would combine a mix of motions, it was unable to recognize them. HMM on the hand, generated multiple clusters each individually represented a state of the motion. A sequence of motions were then assessed to find likely sequences. This sequences could combine multiple states and the probabilities would then be listed accordingly.

5. 3D Visualization of Data

This section shows 3D visualization of raw sensory data for different sensors and different lower-level motions. These 3D visuals were built with MathBox API [20]. These 3D visuals were compiled into the integrated website (see Section 3.9).

5.1 Visualization of All Lower-level Motions for Different Sensors

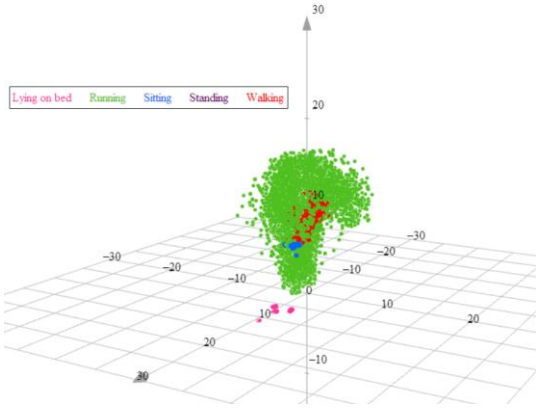


Figure 30 Accelerometer Values for Five Different Motions

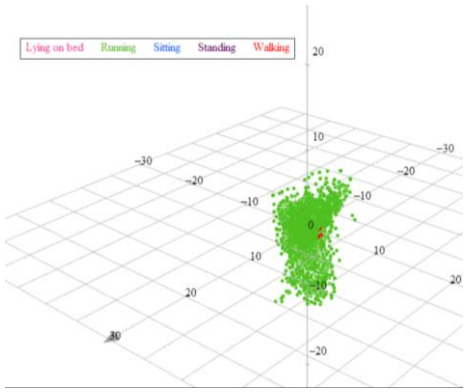


Figure 31 Gyroscope Values for Five Different Motions

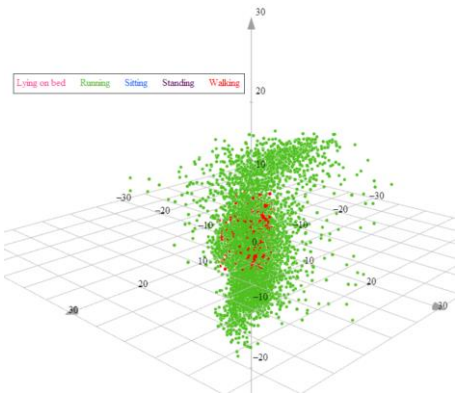


Figure 32 Linear Accelerometer Values for Five Different Motions

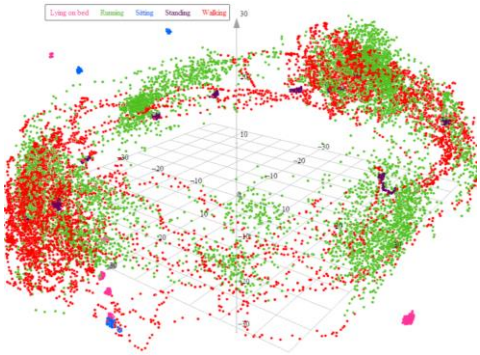


Figure 33 Magnetometer Values for Five Different Motions

Figure 33 shows the Magnetometer readings for five different motions. This figure provides many points of interests as it showed how Magnetometer reading behaved. Magnetometer readings not clustered together like the other sensors. This might be the reason why Magnetometer performed poorly (see Section 4.1) during motion recognition process.

5.2 One Record of Each Motion (With Path Lines)

Figures 34, 35, 36 and 37 shows the raw sensory data of every individual motions.

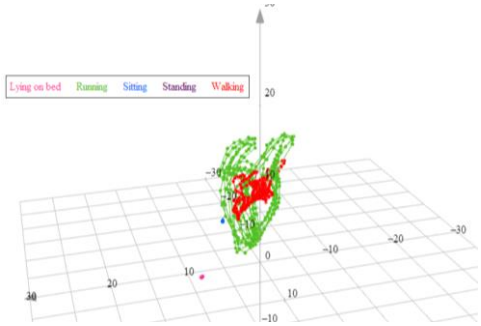


Figure 34 Accelerometer Values With Path Lines

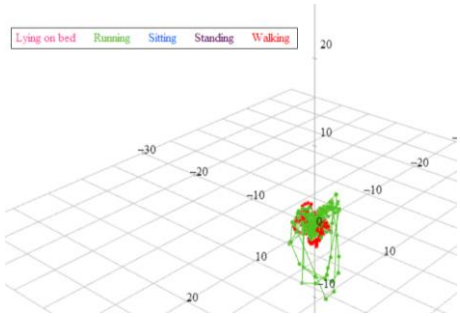


Figure 35 Gyroscope Values With Path Lines

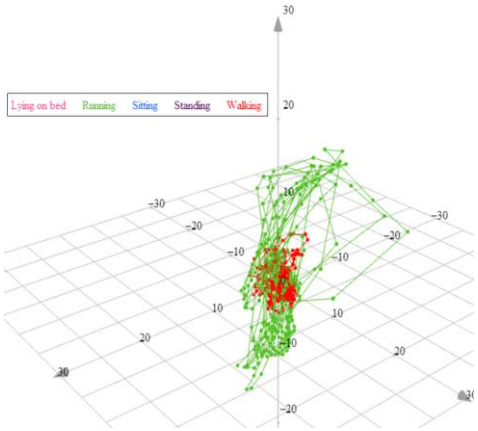


Figure 36 Linear Accelerometer Values With Path Lines

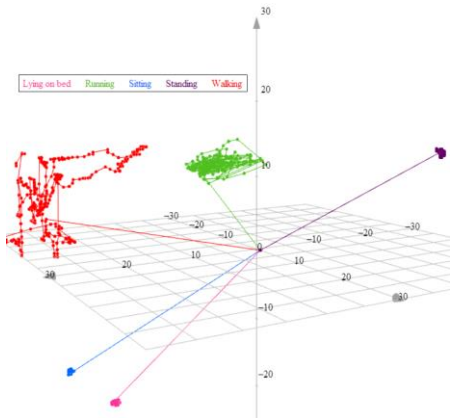


Figure 37 Magnetometer Values With Path Lines

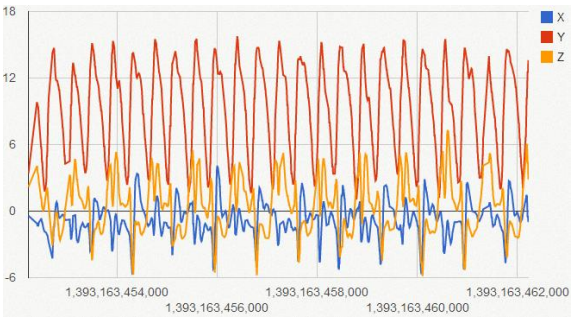


Figure 38 Accelerometer Values for Running Motion in 2D

The behaviour of the different sensory data could be inferred from the figures. Each sensory data were collected consecutively. 3D visualizations might provide a more in-depth analysis of the axes data instead of 2D charts show in Figure 38. *Note that for the five clusters in Figure 37, they are not actually connected to the origin.

5.3 “Lying-on-floor” vs “Lying-on-bed”

The raw sensory data for both the “Lying-on-floor” and “Lying-on-bed” motions were plotted in the 3D shown in Figures 39, 40 and 41. These two motions were specifically plotted because both motions were in “Lying” position. “Lying” position was important as it was one of the two positions (see Section 3.6) to determine a “Fall” and “Lie down”. Hence, if the two motions could be differentiated, “Fall” and “Lie down” motions could be differentiated.

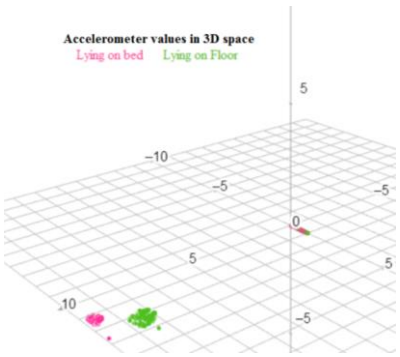


Figure 39 Accelerometer Values in 3D Space for Lying-on-bed on-floor

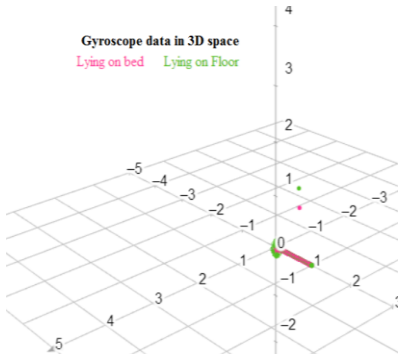


Figure 40 Gyroscope Values in 3D Space for Lying-on-bed on-Floor

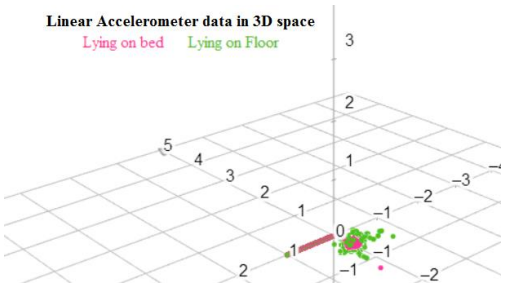


Figure 41 Linear Acc. Values in 3D Space for Lying-on-bed and on-floor

Accelerometer sensory data showed two different clusters between the two motions. This might be because of the smartphone’s position during these two testing processes. The phone might be shifted when it was tested on ground and on bed. This was evident by the technology of the Accelerometer sensors were built upon, which include Earth Gravity (see Section 3.2.2). Therefore, whether on ground or on bed, the gravity towards an axis always holds, and not change. Hence, the only reason for the difference was the angle the phone is tilted at.

6. Conclusion

6.1 Summary of Work Done

The following tasks were accomplished over the course of the project.

Firstly, studies and research were conducted on the latest smartphones available in the market and comparisons were made among the phone's specifications, sensors availability, and abilities.

Secondly, a smartphone application was developed to retrieve and display various sensors data. A locally-stored database was created to store sensors data upon triggers from the user. As motions recognition using Hidden Markov Model was needed, studies on the structure, processes and implementations of the Hidden Markov Model were conducted.

In this project, motions are categorized into lower-level motions, and higher-level motions. The lower-level motions include "Walking", "Running", "Sitting", "Lying", "Standing", and the higher-level motions include "Stand up", "Fall", "Lie down", "Sit down".

This project applied Hidden Markov Model, a probabilistic model, to represent and recognize motions based on observed sensory temporal sequences. The HMM was constructed using Jahmm provided by Java. An enhancement with a rule-based system on the already-built Hidden Markov Model was made to recognize higher level motions.

Lastly, testing of individual motions were conducted and performance were analyzed.

6.2 Future Work

While the current system is complete and fully-capable for its scope, future enhancement could be made to expand the scope and provide quality of life improvements.

The data collection step could be created more naturally so that motion training samples could be collected continuously instead of the existing ten seconds data collection period. This would speed up the data collection process by 50% as currently a five second warning interval is given before the start of every data collection period resulting in a loss of five seconds for each ten seconds of data collected.

The next improvement that could be made would be supporting individual models for each individual. As training data is currently tagged with a unique identifier for each individual, this would be relatively easy to implement as compared to the other improvements mentioned. Individualized models might allow for greater accuracy as the motions of each individual are unique.

Motion testing have been conducted and motions could be recognized almost instantaneously by the HMM. However, due to the limitations of Jahmm model generator, the HMM could not be integrated into the Android application. Thus, motion recognition had to be done on the server, and that had resulted in some transmission latency which cannot be avoided. The transmission latency had unfortunately caused the motion recognition to be delayed by about two seconds. Thus, integration of HMM into the Android application so that real-time motion testing can be recognized immediately. Jahmm has limited support for saving and loading of models. The models that were loaded were inaccurate and the results were different from the generated models. As a result, an object database, db4o had to be used for the storage of the models. However this was not an ideal workaround as the android application was unable to load the multi-dimensional models used. As such, any future enhancement to enabling classification support on the mobile device should focus on fixing Jahmm's model storage modules.

However, due to the speed of the HMM classification, it would also be feasible for the classification to be carried out on the server with several modifications. Currently, the main bottleneck in the processing of results is that the HMM models need to be constantly loaded from disk as an instance of the Java is created for each classification. This could be changed to have the Java program be loaded once and run as a server accepting accelerometer values and returning the classifications. This would eliminate the need for the models to be loaded constantly and improve overall speed by one hundred times.

Motions in this project are trained with minimal movements on the upper body, and that if upper body was moved vigorously during motions recognition, the motion may be misclassified. Thus, motions should be trained while adding more variations. For example, having a conversation with another individual while collecting the "Sitting" motion as one may get excited about a topic and begin gesturing his hands while sitting down.

Location sensors could be incorporated in this project to provide a more accurate picture of an individual's motions. They could be used to accurately track the position of the smartphone in 3D space. This would allow the application to track change in location over time as opposed to the current force over time. This enhancement would be useful especially for detecting falling motions. The falling motions are now detected when there is a large change in acceleration between the "Standing, Walking, Running" and "Lying" motions. However, a "Lie down" motions would have a similar acceleration force and lower-level motion transitions. Thus, a change in position over time would be a better measurement for a fall motion.

The rule-based system used for detecting higher-level motions are currently hand-crafted. Rules in this rule-based system were decided based on careful analysis of the training data, while the rules were useful for this training data, this may not be the case for different individuals. As such, an automated system for generating rules would ensure the rule based classification system remains accurate for different individuals.

Lastly, “null” motions should be considered in the HMM such that when unable to detect a motion accurately, the system would return “null” instead of mistaken motion like “Running” for the actual “Sitting” motion when upper body movements was vigorously moved. When “null” motions were detected, a reclassification could be done. The reclassification method will allow for the creation of a more uniquely identifiable sets of activities. This will allow more details of the individual’s motion instead of lumping them into unclassified motions.

References

[1] National Population and T. Division, "Our Demographic Challenges and What These Mean to Us," National Population and T. Division, 2012.

[2] National Population and T. Division, "A Sustainable Population for A Dynamic Singapore," National Population and T. Division, 2013, p. 10.

[3] J. Tai and L. K. Lim, "Seniors living alone may rise to 83,000 by 2030," in *The Straits Times*, 2012.

[4] N. Bernstein, "Aging in Place Gadgets - Gadgets That Help You Stay Home Longer," 2014.

[5] M. Kose, O. D. Incel, and C. Ersoy, "Online Human Activity Recognition on Smart Phones" Bogazici University, Istanbul, Turkey 2012.

[6] J. Ratajczak, "Learning behavioural patterns in a mobile context using smartphones," Master's thesis [Academic thesis], Department of Informatics and Mathematical Modeling (IMM), Technical University of Denmark, 2011.

[7] Y. Wu, A. Ganapathiraju, and J. Picone, "Baum-Welch Re-estimation of Hidden Markov Model," Institute for Signal and Information Processing1999.

[8] S. M. D. M. Karantonis, , M. R. Narayanan, M. Mathie, S. M. N. H. Lovell, and M. B. G. Celler, "Implementation of a Real-Time Human Movement Classifier Using a Triaxial Accelerometer for Ambulatory Monitoring," *IEEE Transactions on Information Technology in Biomedicine,* vol. 10, 2006.

[9] V. Faber, "Clustering and the Continuous k-Means Algorithm," *Los Alamos Science,* p. 142, 1994.

[10] J. Yang, Y. Xu, and C. S. Chen., "Human Action Learning via Hidden Markov Model," *IEEE Transactions On System Man And Cybernetics – Part A: Systems And Humans,* vol. 27, 1997.

[11] Izenman AJ. "Modern multivariate statistical techniques: regression, classification, and manifold learning". New York, NY: Springer; 2008.

[12] Honeywell International, Inc. (2012). *Protect what matters*. Available: http://homesecurity.honeywell.com/home_security.html#security-systems

[13] D. Anguita, A. Ghio, L. Oneto, X. Parra., and J. L. Reyes-Ortiz, "Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine," International Workshop of Ambient Assisted Living, 2012.

[14] B. Cheng, Y. Tsai, G. Liao, and E-S. Byeon, "HMM Machine Learning and Inference for Activities of Daily Living Recognition," 2009.

[15] H. Fang, R. Srinivasan, and D. J. Cook, "Feature Selection for Human Activity Recognition in Smart Home Environments," 2012.

[16] R. Cilla, M. A. Patricioemail, J. Garcíaemail, A. Berlangaemail, and J. M. Molinaemail, "Recognizing Human Activities from Sensors Using Hidden Markov Models Constructed by Feature Selection Techniques," 2009.

[17] J. R. P. Siirtola, "Recognizing Human Activities User-independently on Smartphones Based on Accelerometer Data," 2012.

[18] J-M. François. (2006). *Jahmm - Hidden Markov Model (HMM)*. Available: http://jahmm.googlecode.com/

[19] C. Li and G. Biswas, "Clustering Sequence Data using Hidden Markov Model Representation."

[20] N. Soares, J. Sundström, W. Pimenta, M. Bartelme, and H. Ferreira. *MathBox*. Available: https://github.com/unconed/MathBox.js

[21] The Editors of the Encyclopædia, and Britannica. (2014). Markov Process. Available: http://global.britannica.com/EBchecked/topic/365797/Markov-process

Appendix A

Ten-fold Cross-Validation Confusion Matrix for Different Sensors

This section shows the different confusion matrix for different sensors at various sampling frequencies.

Confusion Matrix for Sliding Window 0.5 Seconds @ 50Hz

Confusion Matrix (0.5Seconds_50Hz) - Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	3465	35	0	0	0	99.00
	Running	0	3490	0	0	10	99.71
	Sitting	0	108	3392	0	0	96.91
	Standing	0	59	0	3316	125	94.74
	Walking	0	92	0	0	3408	97.37
	Overall Accuracy: 97.55%						

Confusion Matrix (0.5Seconds_50Hz) - Gyroscope							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	2363	0	367	770	0	67.51
	Running	0	3409	0	0	91	97.40
	Sitting	233	0	2786	481	0	79.60
	Standing	728	0	748	2022	2	57.77
	Walking	0	136	0	0	3364	96.11
	Overall Accuracy: 79.68%						

Confusion Matrix (0.5Seconds_50Hz) – Linear Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	3472	0	9	1	18	99.20
	Running	0	3399	0	0	101	97.11
	Sitting	32	0	3459	8	1	98.83
	Standing	0	0	2	3494	4	99.83
	Walking	0	59	0	0	3441	98.31
	Overall Accuracy: 98.66%						

Confusion Matrix (0.5Seconds_50Hz) – Magnetometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	1457	287	1176	241	339	41.63
	Running	108	2196	122	208	866	62.74
	Sitting	229	435	1913	199	724	54.66
	Standing	68	183	157	1649	1443	47.11
	Walking	39	389	98	261	2713	77.51
	Overall Accuracy: 56.73%						

Confusion Matrix (0.5Seconds_50Hz) – All Sensors							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	3458	42	0	0	0	98.8
	Running	0	3500	0	0	0	100
	Sitting	0	46	3454	0	0	98.69
	Standing	0	0	0	3447	53	98.49
	Walking	0	14	0	0	3486	99.6
	Overall Accuracy: 99.11%						

Confusion Matrix for Sliding Window One Seconds @ 50Hz

Confusion Matrix (1Seconds_50Hz) – Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	1683	17	0	0	0	99.00
	Running	0	1697	0	0	3	99.82
	Sitting	0	50	1650	0	0	97.06
	Standing	0	26	0	1610	64	94.71
	Walking	0	21	0	0	1679	98.76
	Overall Accuracy: 97.87%						

Confusion Matrix (1Seconds_50Hz) - Gyroscope							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	1242	0	110	348	0	73.06
	Running	0	1697	0	0	3	99.82
	Sitting	60	0	1458	182	0	85.76
	Standing	255	0	290	1154	1	67.88
	Walking	0	34	0	0	1666	98.00
	Overall Accuracy: 84.91%						

Confusion Matrix (1Seconds_50Hz) – Linear Accelerometer.							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	1687	0	4	0	9	99.24
	Running	0	1691	0	0	9	99.47
	Sitting	16	0	1681	3	0	98.88
	Standing	0	0	0	1699	1	99.94
	Walking	0	3	0	0	1697	99.82
	Overall Accuracy: 99.47%						

Confusion Matrix (1Seconds_50Hz) – Magnetometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	707	175	585	111	122	41.59
	Running	56	1074	42	97	431	63.18
	Sitting	108	245	915	94	338	53.82
	Standing	31	128	62	804	675	47.29
	Walking	7	153	15	43	1482	87.18
	Overall Accuracy: 58.61%						

Confusion Matrix (1Seconds_50Hz) – All Sensors							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	1683	17	0	0	0	99.00
	Running	0	1700	0	0	0	100.00
	Sitting	0	22	1680	0	0	98.71
	Standing	0	0	0	1660	40	97.65
	Walking	0	0	0	0	1700	100.00
	Overall Accuracy: 99.98%						

Confusion Matrix for Sliding Window Two Seconds @ 50Hz

Confusion Matrix (2Seconds_50Hz) - Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	793	7	0	0	0	99.13
	Running	0	798	0	0	2	99.75
	Sitting	0	23	777	0	0	97.13
	Standing	0	6	0	757	37	94.63
	Walking	0	1	0	0	799	99.88
	Overall Accuracy: 98.10%						

Confusion Matrix (2Seconds_50Hz) - Gyroscope							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	615	0	25	160	0	76.88
	Running	0	799	0	0	1	99.88
	Sitting	10	0	731	59	0	91.38
	Standing	87	0	86	626	1	78.25
	Walking	0	6	0	0	794	99.25
	Overall Accuracy: 89.13%						

Confusion Matrix (2Seconds_50Hz) – Linear Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	795	0	1	0	4	99.38
	Running	0	799	0	0	1	99.88
	Sitting	8	0	792	0	0	99.00
	Standing	0	0	0	800	0	100.00
	Walking	0	0	0	0	800	100.00
	Overall Accuracy: 99.70%						

Confusion Matrix (2Seconds_50Hz) – Magnetometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	315	77	293	40	75	39.38
	Running	6	523	26	50	195	65.38
	Sitting	52	101	444	30	173	55.50
	Standing	11	56	38	383	312	47.88
	Walking	2	49	1	3	745	93.13
	Overall Accuracy: 60.30%						

Confusion Matrix (2Seconds_50Hz) – All Sensors							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	790	10	0	0	0	98.75
	Running	0	800	0	0	0	100.00
	Sitting	0	10	790	0	0	98.75
	Standing	0	0	0	779	21	97.38
	Walking	0	0	0	0	800	100.00
	Overall Accuracy: 99.98%						

Confusion Matrix for Sliding Window Two Seconds @ 100Hz

Confusion Matrix (2Seconds_100Hz) - Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	797	3	0	0	0	99.63
	Running	0	800	0	0	0	100.00
	Sitting	0	32	768	0	0	96.00
	Standing	0	14	0	758	28	94.75
	Walking	0	0	0	0	800	100.00
	Overall Accuracy: 98.10%						

Confusion Matrix (2Seconds_100Hz) - Gyroscope							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	613	0	26	161	0	76.63
	Running	0	800	0	0	0	100.00
	Sitting	6	0	734	60	0	91.75
	Standing	124	0	93	582	1	72.75
	Walking	0	8	0	0	792	99.00
	Overall Accuracy: 88.02%						

Confusion Matrix (2Seconds_100Hz) – Linear Accelerometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	796	0	0	0	4	99.50
	Running	0	800	0	0	0	100.00
	Sitting	7	0	793	0	0	99.13
	Standing	0	0	0	799	1	99.88
	Walking	0	0	0		800	100.00
	Overall Accuracy: 99.70%						

Confusion Matrix (2Seconds_100Hz) – Magnetometer							
		Predicted Class					
Actual Class		Lying	Running	Sitting	Standing	Walking	Accuracy %
	Lying	315	76	293	38	78	39.38
	Running	3	553	27	50	167	69.13
	Sitting	53	178	425	31	113	53.13
	Standing	10	61	51	364	314	45.50
	Walking	2	57	2	5	734	91.75
	Overall Accuracy: 59.78%						

- THE END -