DEVELOPMENT AND EVALUATION OF A BLACKBERRY-BASED WEARABLE MOBILITY MONITORING SYSTEM

Hui Hsien Wu

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

Mechanical Engineering
Ottawa-Carleton Institute for Mechanical Engineering
University of Ottawa

Abstract

A Wearable Mobility Monitoring System (WMMS) can be an advantageous device for rehabilitation decision-making. This thesis presents the design and evaluation of a proof-of-concept WMMS that uses the BlackBerry Smartphone platform. A Java program was developed for the BlackBerry 9550, using the integrated tri-axial accelerometer, Global Positioning System sensor (GPS), CMOS digital video camera, and timer to identify change-of-state (CoS) among static states, dynamic states, small activity of daily living (ADL) movements, and car riding. Static states included sitting, lying, standing, and taking an elevator. Dynamic states included walking on level ground, walking on stairs, and walking on a ramp. Small activity of daily living movements included bathroom activities, working in the kitchen, and meal preparation. Following feature extraction from the sensor data, two decision trees were used to distinguish CoS and mobility activities. CoS identification subsequently triggered video recording for improved mobility context analysis during post-processing.

Experimental tests included hardware function, single subject mobility monitoring, and mobility monitoring with five able-bodied subjects. Each subject participated in three trials. Average CoS sensitivity and specificity were 81.57% and 99.31% for the able-bodied subjects. These results provided insight into new algorithms and features that can be used to recognize CoS and activities in real-time and supported continued evaluation of the new WMMS for mobility monitoring.

Table of Contents

List of Tables

List of Figures

Acronyms

ADL	Activities of Daily Living
API	Application Programming Interface
BB	BlackBerry
CoS	Change-of-state
DC	Direct Current circuit
DT	Double Threshold
ECG	Electrocardiography
EEG	Electroencephalography
EMG	Electromyography
FN	False Negative
FP	False Positive
GMMs	Gaussian Mixture Models
GPS	Global Positioning System
IDEEA	Intelligent Device for Energy Expenditure and Activity
LAN	Local Area Network
MS	Maintaining the Same State
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
RIM	Research In Motion
RTSS	Residential Telephone Service Survey
SMA	Signal Magnitude Area
SMV	Signal Magnitude Vector
SR	Sum of Ranges
STD	Standard Deviation
WIMAX	Worldwide Interoperability for Microwave Access
WMMS	Wearable Mobility Monitoring System

Acknowledgement

The thesis gets a lot of support from my advisors, Dr. Edward Lemaire and Dr. Natalie Baddour, who taught and guided me on how to do research both in resources and in information bottlenecks in the search and research breakthroughs.

I want to dedicate this paper to my father Mr. Jia Wang Wu, my mother Miss Mei Mei Chou, my father-in-law Mr. Kun Yao Tsai, my mother-in-law Zhang Yue Zhu, and my brother Mr. Chun Cian Wu for their long-standing support and encouragement, both in emotional, social, and intellectual development. Without their support and care, I could not achieve my goal.

I wish to acknowledge with profound appreciation Research In Motion (RIM) and NSERC for the technical and financial support. Furthermore, the graduate faculty of the University of Ottawa also provided financial support.

I would like to thank Shawn Millar for assistance with data collection. In addition, thanks to my old classmate, Gaëtanne Haché, who gave an orientation and guidance in the research thesis.

I am grateful to all the people at the University of Ottawa Department of Mechanical Engineering and the Ottawa Hospital Rehabilitation Center. For ethics work, thanks to the council of research ethics board, especially to Dr. Nancy Dudek, Miss Dorothyann Curran, Miss Linda Longpré, and Miss Tina Hutchinson for their editing support.

Finally yet importantly, I would like to especially thank my friends and family especially my wife Miss Pei Chuan Tsai and my baby Jeffrey Wu for all assistance during the studying and researching years.

Chapter 1: Introduction

Mobility, defined as the ability to easily move between two separate points, is a significant factor for maintaining quality of life [1]. In a healthcare environment, understanding how a person moves in their daily life is important for making the best clinical decisions. Furthermore, correct mobility assessment is required for effective rehabilitation decision-making and for mobility evaluation in both the community and hospital. The assessment outcomes affect decisions relating to rehabilitation devices, treatment, and environmental modifications.

Wearable Mobility Monitoring, defined as "the use of technology that could be worn on the body to measure mobility", can be used to obtain the required mobility profile. Wearable Mobility Monitoring Systems (WMMS) have laboratory and non-laboratory applications, including real-time activity monitoring, environmental effects, and emergency help [2]. WMMS are a form of a body sensor network that, with appropriate network access, can allow mobility monitoring and analysis anywhere [3].

A Smartphone is a cellular phone with computer-like features, including telecommunications, a camera, and the ability to run sophisticated applications. Smartphones are ubiquitous and multi-functional, incorporating sensors; such as, an accelerometer, GPS, video camera, light sensor, temperature sensor, gyroscope, and/or magnetic sensor. These sensors can be used for activity recognition. Some researchers have used a cell phone to recognize multiple activities via accelerometer only [4], [5].

Wearable video systems with sensors, such as a GPS, electrocardiography (ECG), and accelerometer, have been used to record information about location, movement, and context [6], [7], [8]. In these cases, the contexts were related to food intake and activities, among other parameters. However, these contextual applications did not use video to identify mobility.

Previous research [2] demonstrated the viability of Smartphones as the basis of a WMMS. This preliminary work confirmed that a BlackBerry WMMS, using an external accelerometer and internal digital camera, could identify walking movements (i.e., standing, sitting, lying down) and provide contextual information on the activities through picture analysis. The ability to correctly identify CoS, with few False Positives (FP) and False Negatives (FN), is crucial since appropriate image/video capture depends on real-time CoS determination. Building on this work, this thesis presents a novel WMMS, incorporating new algorithms, internal sensors, and cell phone video for CoS and activity identification. By using only built-in hardware, the system is accessible for consumers. Since, a Smartphone is part of many people's lives, additional hardware costs are eliminated and the consumer will already be familiar with the of use the technology.

1.1 Contributions

The WMMS developed in this thesis monitors an individual's mobility in indoor and outdoor environments, within the user's community. This WMMS uses only built-in hardware supplied with the Smartphone. The WMMS, an activity recognition and mobility analysis system, is unsupervised, ubiquitous, and convenient for research and healthcare applications. The WMMS was designed for independent community ambulation and has minimal space requirements and setup time. Furthermore, the context of the person's mobility activities can be identified, including the environment in which mobility takes place.

While monitoring mobility, the new system saves raw sensor data and features. The raw data could be analysed by other researchers or used in post-processing to improve activity classification. Novel multi-feature algorithms were developed to recognize activities under lower accelerometer sampling rates and sensitivity. Further, this study was also the first to integrate sensors and video analysis for wearable mobility monitoring.

1.2 Scope of the Thesis

The goal of this research was to develop an approach for mobility monitoring that can be used by people with mobility deficits, such as seniors and lower limb rehabilitation patients. Smartphones could monitor a person`s mobility health status, help with rehabilitation decisions, and perform real-time monitoring to improve patient safety, even while alone.

The WMMS evaluates activity states; such as, sitting, standing, lying, walking, climbing, brushing, washing, drying, moving items, and the transitions between states. Additionally, the mobility contexts associated with a CoS, such as a hallway, elevator, kitchen, bathroom, dining room, stairway, inclined plane, and outdoors are often detected and monitored, thus monitoring mobility in 'real-life'.

Mobility CoS was used to trigger video recording in order to improve activity recognition. A CoS is the act of changing from one physical activity behavior to another. Smartphone video was captured when a CoS was identified by the Smartphone's tri-axial accelerometer. Combining activity CoS and video information improves mobility context identification that could lead to better rehabilitation decision-making. Video clips are reviewed manually.

Five able-bodied participants were recruited for the experimental procedure. The participants performed a series of continuous mobility tasks: sitting, lying, standing walking on level ground, climbing up/down stairs, going up/down a ramp, walking outside, taking an elevator, brushing teeth, combing hair, washing hands, drying hands, moving dishes, moving a kettle, moving bread, preparing a meal, and washing dishes. The study was not intended to distinguish all activities, and extended research is required to distinguish CoS and context for different kinds of activities and different populations.

1.3 Rationale

Mobility is important for quality of life. Mobility disabilities and pain are the most prevalent disabilities in Canada [9], which influence accessibility, independence, and social interactions. The number of people with mobility disabilities will continue to increase as our population ages [1].

Currently, most research involving wearable mobility analysis, outside of a laboratory, involves the use of inertial sensors (accelerometers and gyroscopes) or step counters [10], [2], [11]. While reasonable information is provided about whether the person is active or walking/sitting/lying down, these approaches do not provide rich information on the context of the mobility activities. The research proposed in this thesis builds on this literature to create a wearable system that integrates a tri-axial accelerometer, GPS, and video to provide information on both the active state and the environmental context; such as, walking on carpet, tile, or grass; using an elevator; or using an exterior ramp. This is the first system to integrate a video camera, inertial sensor, and GPS for mobility applications.

For mobility-related research, Smartphones have mainly been used as data transmission devices (i.e., receiving data from wearable sensors and transmitting this data to a central repository). Research using the BlackBerry Smartphone platform will be useful for determining the performance of these devices for the demanding mobility-assessment applications, leading to a new version of Smartphones that are integrated into healthcare services. Since Smartphones will eventually become the common technology for mobile communications, mobility assessment could be performed as needed with minimal technology costs to the consumer and healthcare system.

While this thesis focuses on integrated Smartphone sensors, this WMMS could be augmented in the future to enhance health monitoring by including other sensors; such as, ECG, EEG, and EMG. The WMMS might also be used as a life-log, thereby helping people with memory deficits. Mobility identification could also be enhanced to include location-based annotation, that can be used to capture and share information on the

person's community (accessible restaurants, etc,). The WMMS also could be useful for training related to mobility.

1.4 Objectives

The thesis objective is to develop and validate a Smartphone-based WMMS that can monitor mobility at home and in the community. The system must be light and portable, easy to access, contained at one body location, and meet the following objectives:

- Detect, in real-time, a wearer's CoS related to mobility and context for walking, sitting, lying, standing, stairway ascent/descent, and inclined plane ascent/decent.
- CoS should be identified with 95% specificity and 95% sensitivity. Further, video clips analysis should correctly categorize activities and context 95% of the time.

1.5 Thesis Outline

The thesis is in manuscript (research paper) format, including background information in Chapters 1-4 and two manuscripts that have been submitted for publication that describe the innovations and evaluations for the new WMMS.

Chapter 1 introduces the purpose, rationale, initial idea, and several WMMS applications. Chapter 2 is a literature review, including development of other wearable mobility monitors, other mobile mobility monitors, previous WMMS work, and other wearable monitoring research with an accelerometer and/or digital photography, including context identification.

Chapter 3 describes the first WMMS developed at the Ottawa Institute of Rehabilitation Research and Development, including algorithm and system architecture. Chapter 4 shows the new WMMS design, including hardware and software.

Chapter 5 presents the manuscript titled “Activity Change-of-state Identification using a Blackberry Smartphone” that was submitted to Journal of Medical and Biological Engineering (JMBE). This paper assesses the accelerometer and video camera inside the Smartphone and focuses on the development of a CoS determination algorithm for feature calculation. A single-subject evaluation was used to evaluated the algorithm performance and as a basis for optimizing the algorithms and thresholds for higher accuracy.

Chapter 6 presents the manuscript titled “Change-of-state determination to recognize mobility activities using a BlackBerry smartphone” that was submitted to Journal of NeuroEngineering and Rehabilitation (JNER). This paper describes activity classification and test results from able-bodied subjects.

Chapter 7 includes the discussion, conclusion, and future work.

Chapter 2: Literature Review

This section first describes laboratory and non-laboratory mobility monitors. Wearable technology is a necessary extension of non-laboratory mobility monitoring. Wearable monitors are becoming accepted and are gradually developing in functionality (i.e., heart rate monitors, pedometers, personal GPS, etc.). Mobile phones, which are becoming the main telecommunication medium, are an ideal platform for a wearable monitor by providing portable and ubiquitous computing. Lastly, wearable accelerometer monitors and wearable video are reviewed as a method and technology for mobility and context identification.

2.1 Mobility Monitoring in the Community

Mobility monitoring can predict mobility ability, such as the probability of a fall, as an indication of health status by measuring balance and mobility functions. The participants or patients are asked to perform activities, such as sitting, standing, turning, walking, etc. so that their physical function during mobility tests may be better understood. The final medical results are often combined with the person`s medical record, physician's observations, and additional questionnaires related to environmental analysis, physical functions, and social functions [2].

2.1.1 Laboratory Mobility Monitors

A literature review by Haché et al. [2] provides more details on laboratory mobility systems; such as, visual motion tracking systems [12], force plates [13], and pressure mats [14]. These systems can obtain accurate results by using a gait index, balance scale, and/or a mobility scale related to time. However, these systems require additional space, setup time, setup facilities (i.e. stairs and ramp may not exist in the laboratory), and cost (i.e. infrared cameras and medical instruments are not ubiquitous devices). Furthermore, most research related to mobility monitoring did not include an ecological analysis, such as, mobility monitoring at home and in the community.

2.1.2 Non-Laboratory Mobility Monitors

Several kinds of wearable monitoring systems are available for use in and out of the laboratory, with many systems used for long-term and low effort monitoring. Pressure sensing insoles could be combined with wireless transmission to collect information such as pressure, 3D-tilt, and acceleration within a wearable system [15]. Alternatively, Roy et al. [16] used surface electromyography (EMG) combined with acceleration to monitor a stroke patients activities for daily living. An instrument was also developed as an unobtrusive wearable monitor to detect movement disorders in Parkinson's disease during sitting, standing, walking, and lying.

2.1.2.1 Multiple Accelerometers

Several mobility researchers used multiple accelerometers to obtain above 80% accuracy for identifying sitting, standing, lying, and walking. Jani et al. [17] used two tri-axial accelerometers, located at the hip joints, to identify level walking and walking up/down stairs. The researchers used MATLAB software for Independent Component Analysis in feature fraction to enhance accuracy from 70 % to 85 %.

Preece et al. [18] attached three tri-axial accelerometers to the waist, thigh, and ankle, to analyze eight dynamic activities, including level walking and climbing stairs. It was found that a Fast Fourier Transform was better than a discrete wavelet transform for mobility analysis and mean and standard deviation (STD) were better than fractal dimension.

Bao et al. [19] collected data from five bi-axial wireless accelerometers, worn by subjects, to detect body movement such as walking up/down stairs. The system also tried to recognize household activities, including hygiene activities, riding an elevator, cleaning, etc. These sensors were attached to the wrist, mid-sternum, chest, hip, and the shank of the leg. The research compared fourteen classification algorithms, such as neural networks, decision tree, support vector machines, etc, and showed that the decision tree got the best accuracy for overall body motion.

2.1.2.2 Intelligent Device for Energy Expenditure and Activity (IDEEA)

The IDEEA is a commercial product that could detect and identify human movement, including climbing up/down stairs, with high accuracy. A microcomputer was linked with five motion sensors attached to the chest, left thigh, right thigh, left foot, and right foot. Each sensor had a cable connected with the computer. The instrumentation, whose calibration was not required, had a high battery life of about 48 hours. The system focused on gait identification and speed estimation for mobility. The program also showed movement distance and time [20].

2.2 Wearable Monitoring

Wearable monitoring is growing quickly due to advancements in wireless communications, microprocessors, and memory technology. The following subsections will demonstrate that this kind of monitoring has several advantages.

2.2.1 Growth of Wearable Monitoring Systems

Many wearable monitoring systems have emerged in the global marketplace; such as, mobile ECG (Cardio24, www.rdsm.eu/cardio24.html), EEG (TELEMEDX, www.telemedx.com), EMG (BTS FreeEMG 300, www.btsbioengineering.com), and textile sensors (Textronics, www.textronicsinc.com). These commercial instruments focus on blood pressure, heart status, and/or neural signals. Furthermore, wireless local area networks (LAN), such as Wi-Fi and Worldwide Interoperability for Microwave Access (WIMAX), can be built quickly. Wi-Fi was expected to provide higher upload and download speeds than WIMAX to enable transfer of more data and/or information via a mobile phone or computer, especially in an outdoor environment [5], [21]. Table 2-1 shows the specification comparison of WIMAX and Wi-Fi.

Table 2-1: Specification comparison of WIMAX and Wi-Fi [5], [21].

Specification	802.16 (WIMAX)	802.11 a, b, g (Wi-Fi)
Max Throughput	11 ~ 54 Mbps	70 Mbps
Frequency Band	2.4 ~ 5.5 GHz	5 ~ 11 GHz
Max Bit Rate	2 ~ 54 Mbps	15 ~ 134 Mbps
Spectrum	2, 11 GHz	10 ~ 66 GHz

In growth theory, described by Nolan [22], the four stages of technology growth were initiation, expansion, formalization, and maturity. In 2005, wearable mobility monitoring systems were somewhere between initiation and expansion [23]. Acquisition of technology is the major work at the initiation stage, and the technology necessarily focuses on simple functions. The development of new projects rises quickly at the expansion stage.

2.2.2 Advances and Development in Wearable Technology

Paolo [24] addressed the impact of wearable technology using unobtrusive sensors. Textile sensors, accelerometer, EMG sensors, etc. were attached to the body for wearable monitoring in emergency applications, long-term and real-time monitoring outside of a hospital, and treatment of cardio diseases and stroke. Furthermore, Haché et al. [2] proved that wearable technology had low space requirements. Wearable monitoring research is intended to enhance recording of movement, the delivery of information to clinicians in the appropriate way, and the implementation of analysis algorithms for data obtained from these wearable sensors.

2.3 Mobile Phone as a WMMS

2.3.1 Mobile Phone Growth

According to Statistics Canada, mobile phone usage has increased by approximately 1.5 to 8 percent per year until the end of the 2005 [25]. At that time, 50 percent of Canadians used wireless services [26]. A detailed survey of statistical methods

and techniques included sample size, sample design, data processing, and sample weighting guidelines [27].

According to Statistics Canada's Residential Telephone Service Survey (RTSS), by the end of 2006 66.8 percent of Canadians had at least one cell phone [28] and 70.1 percent of Ontario inhabitants had a cell phone (Figure 2.1). In December 2007, the overall percentage increased by 5.3 percent [29]. By the end of the 2008, the percentage increased to 74.3 for all Canadians and 76.8 for Ontario inhabitants [30].

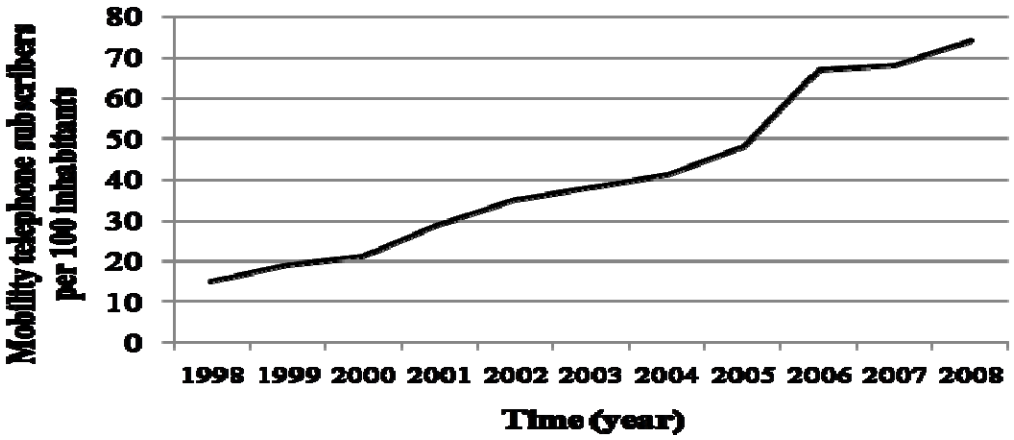


Figure 2.1: Residential mobile phone service survey in Canada [26]-[30].

Globally in 2009, mobile subscriptions were 108.55 per 100 inhabitants in developed countries such as Canada, United States, and France (i.e., some people own more than one cell phone) [31]. Even with the financial crisis of 2007-2010, both developed and developing countries still increased mobile markets. Figure 2.2 shows the number of mobile phone subscribers in developed and developing countries [31], [32].

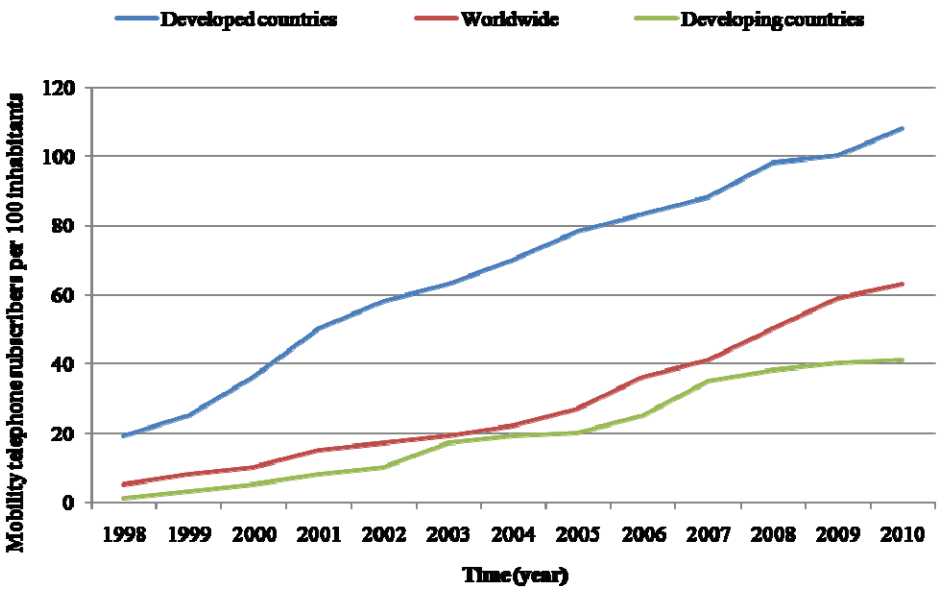


Figure 2.2: Mobile phone subscription in world telecommunication [31], [32].

2.3.2 Smartphone as a Mobility Monitor

Smartphones are now popular, prevalent, and incorporate several sensors; such as, an accelerometer, GPS sensor, video camera, light sensor, temperature sensor, gyroscope, and/ or magnetometer. Brezmes et al. [33] used a Nokia Smartphone's integrated accelerometer to recognize static activities, walking on level ground, and stairs. Three axes accelerations distributed on a Cartesian reference system were used to show the change in user movement. Euclidean distance and k-nearest neighbour algorithms were used to recognize these activities. With the phone in a pocket and a sampling rate of 30 Hz, the accuracy of identification for walking on stairs was 80 %.

Haché et al. [2] used a BlackBerry Smartphone and external sensor board for the WMMS on which this thesis is based. The board contained a tri-axial accelerometer, light sensor, and temperature sensor. The phone was worn on the right pelvis in a smart-holster. Data, sampled at 50 Hz, were processed in 1-second windows. CoS detection algorithms triggered picture capture from the BlackBerry for use in activity recognition; including, sitting, lying, standing, walking on level ground, stairs, ramp, elevator, and a car ride. An

overall sensitivity of 77.7% and specificity of 96.4% were obtained. Activity recognition accuracy for walking on stairs was 54 %.

Kwapisz et al. [11] used an Android Smartphone with a built-in accelerometer to recognize walking on level ground, jogging, stairs, sitting, and standing. The phone was worn in the person's pocket and the sampling rate was 20 Hz. The activity recognition accuracy was approximately 60 % for walking on stairs and between 89.9 % and 98.3 % for other activities. Ganti [4] used a Nokia Smartphone with a built-in accelerometer, microphone, GPS, and GSM to recognize aerobic activities, cooking, deskwork, driving, eating, hygiene activities, meetings, and other activities, such as watching TV. The phone was worn on the waist and the accelerometer sampling rate was 7 Hz. Activity identification accuracy was not described in the paper. The overall precision was 61.15 %. Low sampling frequency and accelerometer sensitivity are potential issues when using a Smartphone-based accelerometer.

Table 2-2: Sensitivity (SE) and frequency (Freq.) comparison for wearable mobility monitor with a tri-axial accelerometer.

Author	Location	Phone based	Features	SE (G)	Freq (Hz)	N	Accuracy	Activities	Method
Z. He [34]	Pocket	No	Discrete cosine transform, the Principal Component Analysis	3	100	1	97%	Still, walk, run, and jump.	Support vector machine
N. Wang [35]	Waist	No	Time-frequency analysis (7 features mode decomposition)	6	50	12	90.90%	Walk up and down slope.	Gaussian mixture model
G. Hache [2]	Waist	No	STD-Y, magnitude area, skewness Y, inclination angle	6	50	5	88%	Stand, sit, lie, walk, stairs, ramp and	Hierarchical decision tree
A. M. Khan [36]	Chest	No	Autoregressive coefficients, signal magnitude area, tilt angle to 3D feature plot	6	20	6	97.65%	Sit, stand, lie, walk, walk on stairs, and run.	Artificial neural nets-hierarchical setting--using
J. R. Kwapisz [11]	Front pant pocket	Android	Average, STD, average absolute difference, average resultant acceleration, signal magnitude area, time between peaks, and binned distribution	2	20	29	50 % stairs, others about 91 ~ 95%	Walk, jog, walk on stairs, sit, and stand.	Multilayer perceptron
Z. Shumei [37]	Waist	Android	Change, max, min, and velocity	1	1	10	81.9% for walk and others are 83.7%	Walk, posture transition, gentle motion, stand, sit, and lie.	Hierarchical classification with multiclass SVM (support
R. K. Ganti [4]	Waist	Nokia N95	Energy expenditure, body orientation(angle), skewness of magnitude of 3D acceleration, acceleration entropy	NA	7	10	Eat and meeting (14%); drive, watch TV, aerobic (80%); desk work, hygiene (50%); cook (100%)	Sit, stand, aerobic, brush teeth, work, lie.	Hidden Markov Model

2.4 Using Accelerometers for Mobility Identification

Mobility analysis via accelerometer involves several steps including placement, data collection, calibration, filtering, feature extraction, and activity classification. Device placement is generally on the waist, which is near the center of mass [38]. This position is also a common location for a mobile phone [2] and many researchers have concluded that the waist is an efficient location for monitoring body activities [38], [39], [40]-[44].

Data collection sampling frequencies are usually between 0.3 and 20 Hz since the frequency of most mobility activities lies within this range [45]. For static movements, the collection frequency can be set to 15 Hz. In the dynamic field, the minimum frequency should be 20 Hz. At the waist position, the sensor range should be ±6g [2], [40].

2.4.1 Filtering Techniques

Noise is generated by the surroundings, such as a vehicle or the sensor rubbing against joints. To accommodate the 0.3 - 20 Hz range for human movement, a high-pass filter was used to remove the gravity direct current (DC) offset from 0.25 ~ 0.5 Hz [2]. This filter separated gravity and human movement accelerations and the method allowed periods of rest and activity to be recognized. The low-pass filter, which could be digital or analog, also eliminated high frequency noise. A low-pass filter removed frequencies greater than 15 or 20 Hz [40]. The combination of low and high pass filters was necessary for identification of walking patterns, where the frequency range was between 8 and 20 Hz.

Huang et al. [46] used Empirical Mode Decomposition to generate an Intrinsic Mode Function in frequency analysis for non-linear and dynamic states. After the shifting process, the Intrinsic Mode Function would appear. The function found the mean between maximum and minimum, which was zero. The number of maxima had to equal to the number of minima.

2.4.2 Signal Processing

WMMS signal processing is required for calibration and feature extraction. Calibration could be automatic or manual [2]. Acceleration calibration includes

calibration of the sensitivity and offset. A summary of various acceleration features from the literature are described below and in section 3.2.

- **Kurtosis**: Kurtosis is the value of the peakedness in a distribution (Equation 2.1). A high value implies that an infrequent large deviation has occurred and that the tail is longer and/or fatter (Figure 2.3). Baek et al. [47] used kurtosis to recognize climbing up/down stairs from walking and running.

$$Kurtosis = \frac{1}{N\sigma^4}\sum_{i=1}^{k}(c_i - m)^4.n_i \qquad (2.1)$$

where N is the total numbers of data points, σ is the standard deviation, m is the mean, c is the center value of the histogram, n the number of data points, and i is the interval

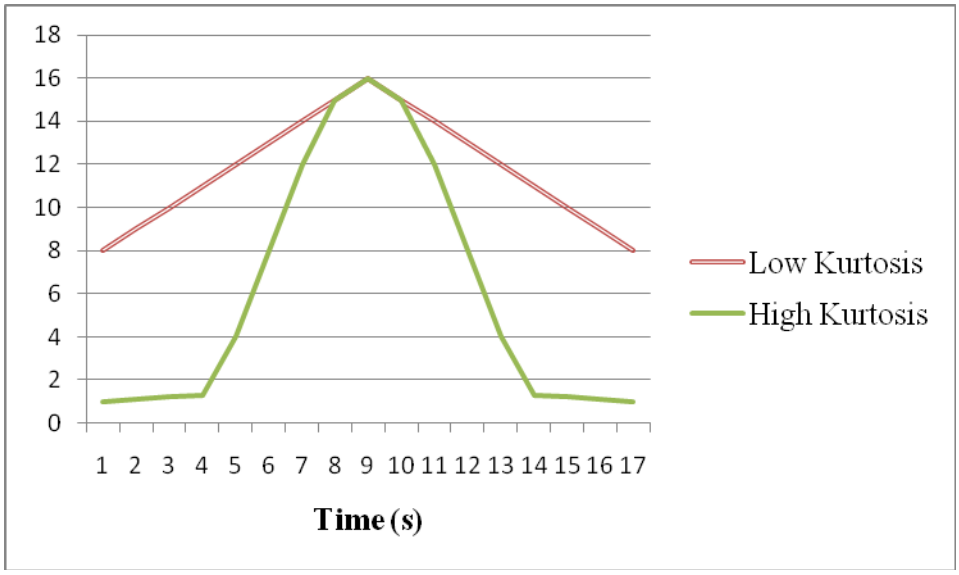


Figure 2.3: High and low kurtosis [48].

- **Correlation**: Correlation [49] measures the dependence between two data sets. This feature helps to identify walking/running from walking on stairs. When the correlation coefficient is positive, both X and Y values are increasing. Conversely, when the value is negative, X increases and Y decreases (Figure 2.4). The range of values is between +1 and -1. Correlation

(Equation 2.3) is defined as covariance (Equation 2.2) divided by the standard deviation for the X and Y data [50].

$$Co\,\mathrm{var}\,iance(X,Y)=\frac{1}{N-1}\sum_{i=1}^{N}(X_i-\overline{X})(Y_i-\overline{Y}) \tag{2.2}$$

where N is the total number and i is the initial value [50]. Correlation is defined as

$$Correlation(X,Y)=\frac{Co\,\mathrm{var}\,iance(X,Y)}{\sigma_X\sigma_Y} \tag{2.3}$$

where σ is the standard deviation [50].

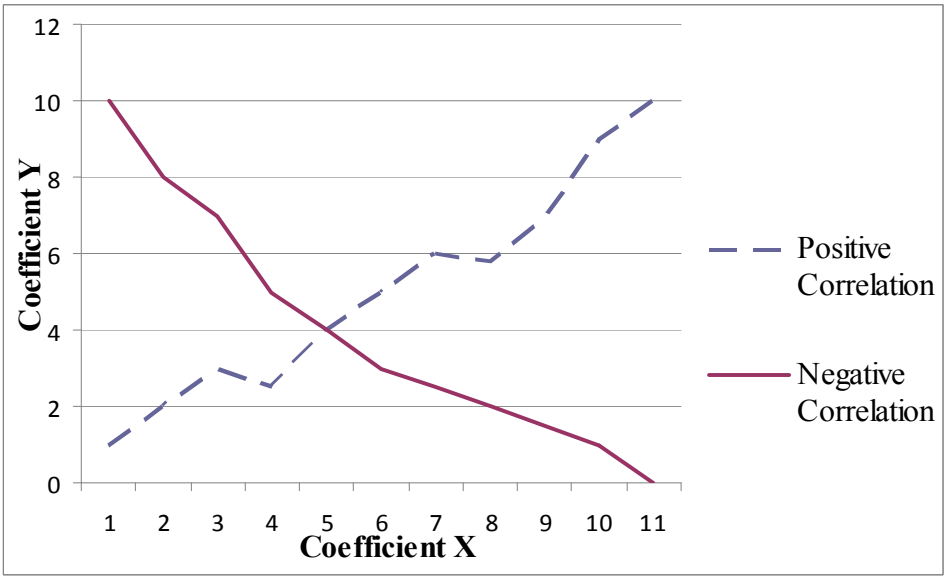


Figure 2.4: Positive and negative correlations [49].

- **Energy**: Energy, or equivalently the magnitude of the discrete signal, is the sum of the squared magnitude of the Fast Fourier Transform (Equation 2.4). Ravi et al. [49] used the Fast Fourier Transform to show magnitude in the frequency domain. The window size is in a range related to the number of samples. The wave shape could be a squared, sine, or cosine wave.

$$Energy = \frac{\sum_{i=1}^{|w|} |X_i|^2}{|w|} \tag{2.4}$$

where w is the window size, i is the initial value, and X is the sum of the Fast Fourier Transform component magnitudes in a window length. Energy can also be calculated directly from the signal itself, without a Fourier transform.

- **Signal Magnitude Vector (SMV)**: SMV (Equation 2.5) is the combination of the root mean square in three axes, but not divided by the total number of samples. Signal Magnitude Vector used to distinguish falling [38].

$$SMV = \sqrt{x_i^2 + y_i^2 + z_i^2} \tag{2.5}$$

where x, y, and z are the accelerations along x, y, z axes, and i is the ith sample of the axis.

2.4.3 Classification

Many activity classifications algorithms have been used to enhance accuracy in mobility assessment [11], [36]. Decision trees are the most common and simple classification method [38], [2], although some researchers have used neural networks [51], [52] or Gaussian Mixture Models (GMMs) [10], [53], [35]. Typical classification approaches are:

- **Hierarchical decision tree**: A hierarchical decision tree is a decision support tool that represents a process with a tree-like graph or model. For signal classification, features are organized within the tree to extract patterns, which are necessary for discrimination and predictive modeling in large databases. The decision tree can lead to activity classification. Haché et al. [2] used a hierarchical decision tree to classify activities based on a number of sensor features. Thresholds were set for each feature within the tree to provide a path to the appropriate activity classification. Karantonis et al. [38] used a real-time movement classification algorithm, which was a kind of hierarchical decision

tree. Each block, which presented the discriminant of an algebraic equation, was divided into two branches in the flowchart (Figure 2.5).

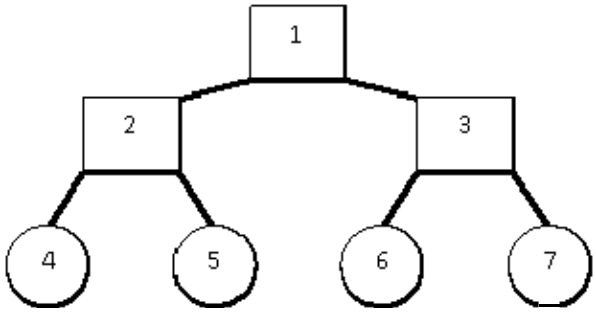


Figure 2.5: Basic structure of the hierarchical decision tree [38]. The squares represent the discriminant of an algebraic equation and the circles represent outputs or activities.

- **Multilayer perceptron classification:** This neural network classification approach is an extension of linear perceptron [51] that incorporated a feed forward Artificial Neural Network with two or more layers of neurons.
- **Neural fuzzy network**: This artificial neural network [52] used signals from a tri-axial accelerometer for mobility analysis. The signal was defined as:

$$s = \sqrt{x^2 + y^2 + z^2} \tag{2.6}$$

where x, y, and z are tri-axial accelerations. Equation 2.7 presents the transfer function of the network,

$$H(z) = 1 + a_1 z^{-1} + ... + a_p z^{-p} \tag{2.7}$$

where H is the transfer function, a is the predictive parameters, and p is the order of the auto regression model, which can be determined by experiment for optimization. The order could be 16, 32, 64, 128, or others. The predictive parameters were selected to minimize the power of the prediction error *e(n)* (Equation 2.8):

$$E[|e(n)|^2] = E[|x(n) - \hat{x}(n)|^2] \tag{2.8}$$

where x is features or input.

Orthogonal theory could minimize the power of *e(n)*:

$$E\{x^*(n-k)[x(n)-\hat{x}(n)]\}=0, \qquad k=1,2,.....,p \tag{2.9}$$

The neural fuzzy model is an Auto-Regressive model that uses a random signal processing approach (Equation 2.10):

$$v_{\min}=r_m+\sum_{k=1}^{p}a_k r_{m-k}+c \tag{2.10}$$

In equation 2.10, v_{min} is the white noise with zero mean, a_k can be calculated, r_m is the autocorrelation function, and m is between zero and p. When |a| is less than one, the output has a white noise input. When "a" is equal to one, the output will be infinite. Therefore, assuming |a| is less than one, the mean r_m can be identical. The constant (c) can be eliminated for simplicity.

When m is greater than zero, v_{min} is eliminated. The formula usually becomes a matrix to solve for a_k (Equation 2.11):

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \end{bmatrix} \begin{bmatrix} r_0 & r_{-1} & r_{-2} & \cdots \\ r_1 & r_0 & r_{-1} & \cdots \\ r_2 & r_1 & r_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.11}$$

The neural fuzzy network has three main layers with the following form (Equation 2.12):

$$y_j=w_{oj}+\sum_{i=1}^{n}w_{ij}x_j \tag{2.12}$$

where y is the output variable, w is the weight factor, and x is the input variable. The structure is shown in Figure 2.6 [52].

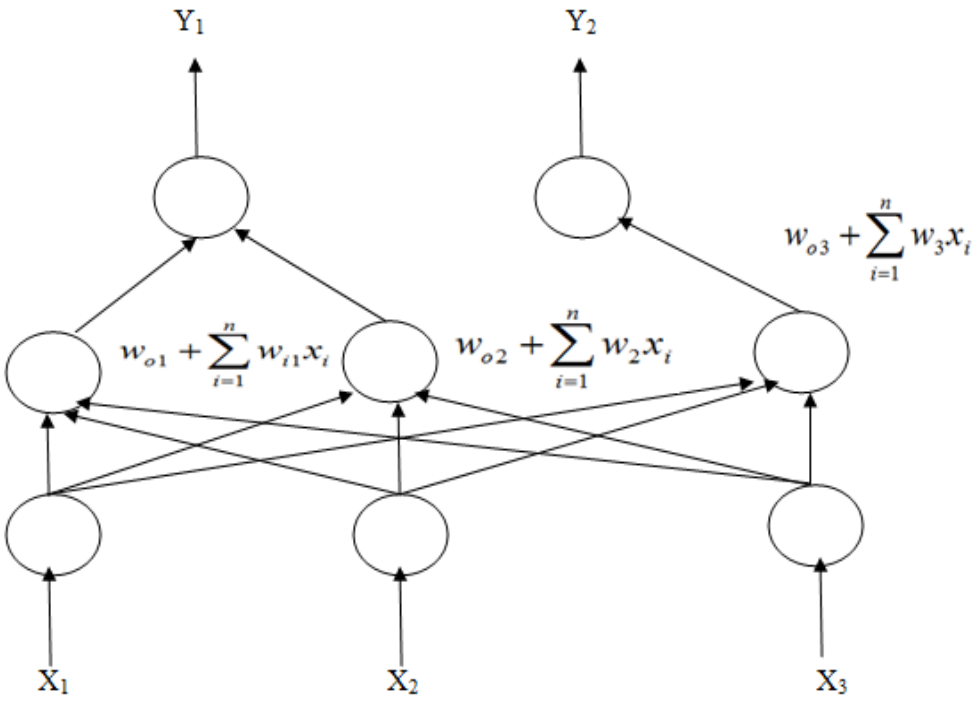


Figure 2.6: The structure of the neural fuzzy network [52].

- **Gaussian Mixture Models (GMMs)**: GMMs are a type of pattern recognition system [35], [53]. A probability density function relates to the multivariable Gaussian mixture density. Several inputs are used to obtain the weights of the distribution by the Expectation-Maximization algorithm. The Expectation-Maximization algorithm is an iterative method that depends on latent variables to find the maximum likelihood estimates of patterns.

  Allen et al. [53] used the GMMs-based classification that was trained by the Expectation-Maximization algorithm. The method, which represents a probability density function, uses Bayesian theory. The theory adapts shapes to the mean of the GMMs components in computing likelihood. Classes are divided into walking on level ground, walking up/ down slope, and climbing up/down stairs.

2.5 Context Identification by a Wearable Video Camera

Wearable Video is very easy for people to obtain and use. Many people own a mobile phone with a digital camera and/or video recorder to record their experiences. Several researchers have developed wearable video systems with GPS and/or motion sensors to record location, movements, and/or context [6] [7], [8]. However, several bottlenecks of wearable video include retrieval issues, synchronization, light quality, low visual resolution, artifacts, and automatic annotation.

2.5.1 Life Log

Byrne et al. [6] recorded visual lifelogs using a SenseCam, which includes a wearable camera, tri-axial accelerometer, passive infrared sensor, and light sensor. The accelerometer detected wearer movement. The infrared sensor then detected subjects in front of the wearer. The light sensor detected changes in light level. A road, staircase, door, or grass, were recognized using manual judgment with feature extraction.

Ryoo and Bae [8] created a lifelog system that combined a Personal Digital Assistant (PDA) with several sensors; such as an accelerometer, GPS, ECG, and temperature sensor. The ECG sensor measured heart rate, heart rate variability, and stress level. The GPS obtained local location and universal time. The accelerometer recognized postures and activities and also detected falls. Their paper focused on instrument design and did not provide any accuracy analysis or experiments related to mobility and physiology.

Tancharoen et al. [7] also designed a lifelog system that combined a notebook personal computer (PC) with a microphone, accelerometer, GPS, skin temperature sensor, galvanic skin response sensor, heat flux sensor, step counter, brain wave analyzer, and wearable camera. The microphone detected voice and talking scenes via audio signals. The bi-axial accelerometer recorded maximum, minimum, average, and mean absolute difference accelerations to identify running, walking, and sitting. The GPS recorded latitude, longitude, speed, direction, and relative time. The Galvanic Skin Response sensor measured skin conductivity between two points by electrical conductivity. The heat flux and temperature sensors measured metabolism status and energy expenditure by heat exchange rate and temperature. The brain wave analyzer combined wearable

electroencephalography (EEG) with the portable computer to summarize the interesting features from the brain response. The wearable camera detected face status during talking. The overall purpose was to create and index a lifelog video to search information and to create a retrieval system.

2.5.2 Objective Dietary Assessment

Competency et al. [25] developed a food intake monitor that contained wearable video, reference light, accelerometer, microphone, and GPS. The accelerometer recorded physical activity. The microphone could identify eating scenes by ambient sound. Overall information would then transfer to the registered dietician's computer for analysis. The light was provided by laser diodes and light-emitting diode as a dimensional reference to calculate food portion size and volume to determine nutrients and calories.

2.6 Conclusion

Mobility affects quality of life. A reliable and long-term mobility monitor can provide better information on a person's mobility to help clinicians make good decisions for rehabilitation. Given current technology, the potential exists to make advances in the wearable mobility monitoring area. In addition, as mobile phones and wireless networks become mature and highly prevalent, WMMS development will become easier and smoother. Systems that are suitable for a home and/or community environment could causes less interference with people's lives, as compared with other intrusive systems.

Information on relevant wearable mobility monitors in the literature can help when developing a better WMMS. Statistical methods, time domain, and decision trees are useful for our research. Furthermore, signal processing, such as sample frequency and signal enhancement, will be pursued in the analysis.

Wearable video is a viable approach for identifying context. This approach can help resolve the problem caused by unclear or poorly positioned still images, which were an issue of the WMMS by Haché, et al. [2]. Audio information can also help to recognize activities.

Chapter 3: Overview of the First BlackBerry WMMS

The WMMS developed in this thesis benefitted from research by Haché et al. [2] on an initial BlackBerry-based system. This first version WMMS is reviewed in this chapter.

3.1 System Architecture

Research by Haché et al. [2] showed that synchronized and external sensors in a "Smart-holster" (Figure 3.1), combined with Smartphone GPS and camera images, could be used as a WMMS. The Smart-holster incorporated an accelerometer, Bluetooth sensor, temperature sensor, humidity sensor, and light sensor. The board was combined with a BlackBerry Bold 9000 that had an embedded camera. The tri-axial accelerometer had a ±6g sensitivity and 50 Hz sampling rate.

Figure 3.1: System architecture of the first BlackBerry WMMS [2].

3.2 Signal Processing

The Smartphone preprocessed sensor data before generating real-time features from the acceleration and GPS signals [2]. A rotation angle concept [54] was used as a simple calibration method. For example, in static conditions, if the vertical acceleration component is vertical to the Earth, the output should be equal to gravity (1g). If the shaft rotates 180 degrees, the output should be equal to negative gravity (-1g).

The following features were used to recognize mobility states:

- **Y acceleration standard deviation (STD-Y)**: STD-Y defines static or dynamic movement states (Equation 4.1). Y direction depends on the phone's orientation.

$$STD-Y=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-m)^2} \tag{3.1}$$

  Here, y_i is the Y-acceleration value, m is the Mean-Y, and N is the data window size. The X and Z acceleration standard deviations (STD-X, STD- Z) formulas were the same as equation 4.1. STD-X, Y, and/or Z features recognized dynamic and stairs states [2], [11], [49]

- **Signal Magnitude Area (SMA)**: SMA is the sum of tri-axial acceleration magnitudes over time, as defined in Equation 4.2. This feature distinguishes periods of activity and rest via a threshold value. When the Signal Magnitude Area was greater than the threshold value, the user was in the dynamic state. The time interval should be more than one second.

$$SMA=\frac{1}{t}(\int_0^t|x(t)|\,dt+\int_0^t|y(t)|\,dt+\int_0^t|z(t)|\,dt) \tag{3.2}$$

  where x, y, and z are the accelerations along x, y, z axes.

- **Skewness-Y**: Skewness measures the asymmetrical probability distribution, as given in Equation 4.3. This value can distinguish a signal spike. Baek et al.

[47] used skewness to distinguish walking and running from climbing up and down stairs.

$$Skewness = \frac{1}{N\sigma^3}\sum_{i=1}^{k}(c_i - m)^3.n_i \tag{3.3}$$

Here, N is a total number of samples, k is the number of intervals, i is the interval, c_i is the axis acceleration at point i, m is the mean, and n_i is the number of samples in interval i.

- **Inclination angle**: Inclination angle is defined as the angle between the positive z-axis and gravity related to the y-axis. Postural orientation relates to body tilt. This value can distinguish sitting and standing states [38], [2]. In Equation 4.4, Az is the average z-axis acceleration and Ay is the average y-axis acceleration. The range of the original value is from 180 to -180 degrees. The additional 180 degrees is for convenience so that the total range is between 0 and 360 degrees.

$$\theta = \arctan(\frac{Az}{Ay}) \tag{3.4}$$

- **GPSspeed**: GPSspeed was used to detect if a person was in a moving vehicle.
- **Light intensity**: The intensity detected indoor/outdoor conditions.

3.3 Change-of-State and Classification

All features were produced by the BlackBerry and input into a decision tree algorithm, which in turn determined if a CoS had occurred. A picture was taken by the BlackBerry's camera at each CoS. All features were calculated once per second. Feature data, time stamp, and image name were saved to a feature log. The static digital photograph was also saved on a 1 GB SD card.

A hierarchical decision tree with six features (Figure 3.3), combined with a double threshold (DT) algorithm, was used to classify signals and recognize activities such as sitting, standing, lying, walking on level ground, walking on stairs, and car riding [2].

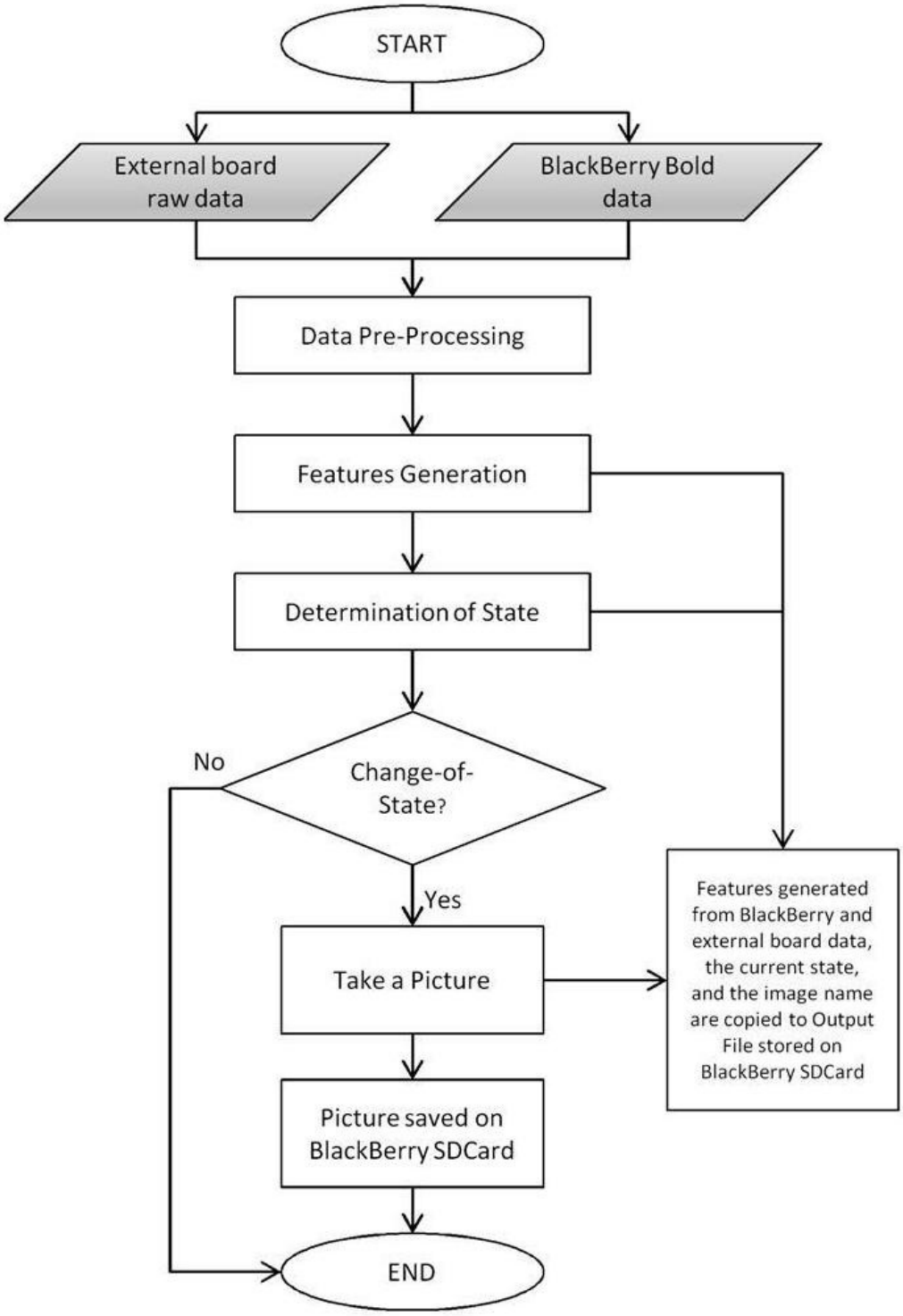


Figure 3.2: Haché et al. WMMS algorithm and signal processing [2].

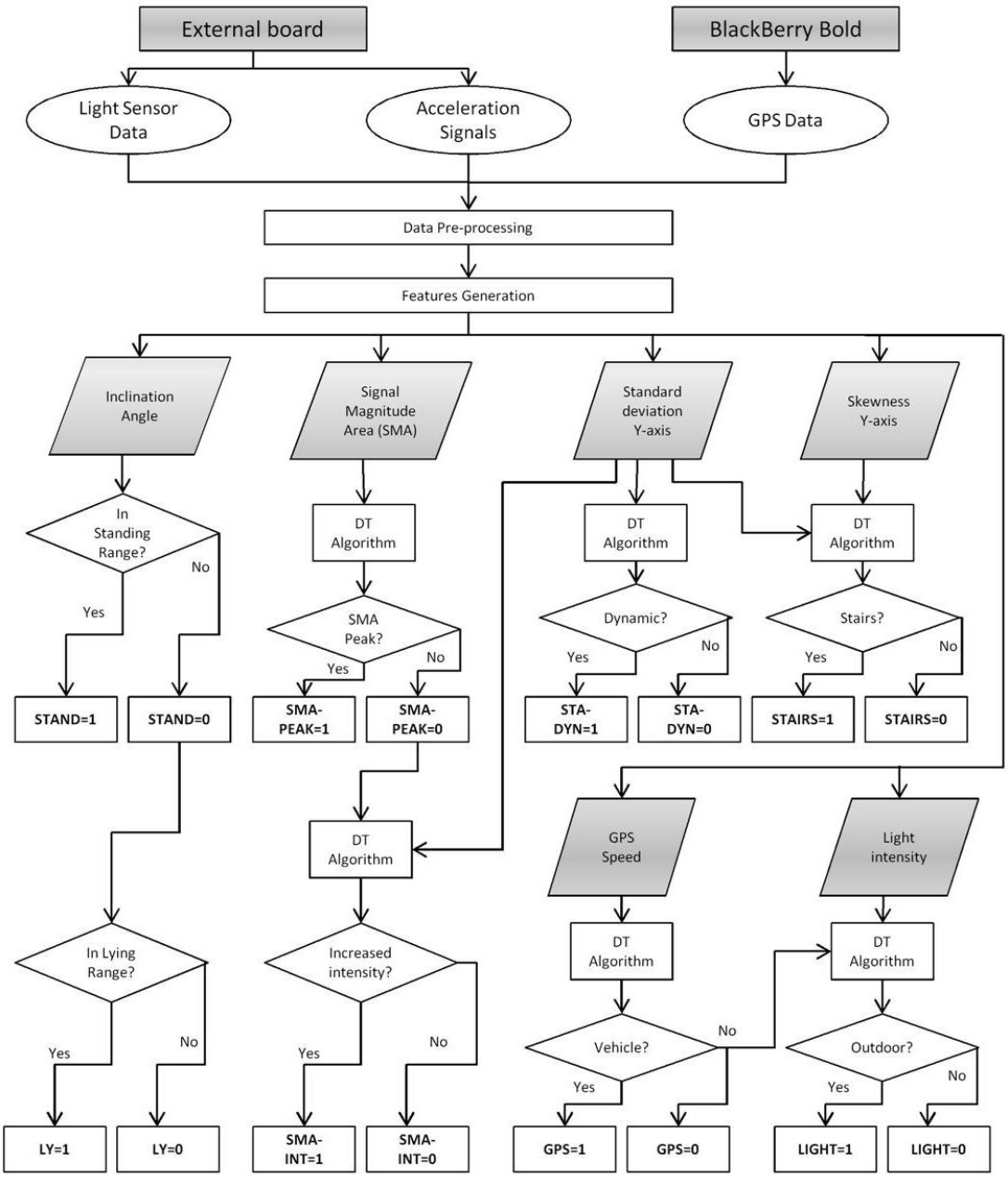


Figure 3.3: Haché et al. WMMS state determination algorithm [2].

3.4 Hardware Limitation

During picture capture, no acceleration and other sensor data could be received or processed, for 0.7 to 0.9 seconds [2]. Furthermore, the BlackBerry Bold 9000 could take up to half an hour to detect GPS satellites.

Battery usage, which is an issue for long term monitoring, was 29% per hour. For long-term monitoring, at least 24 hours battery capacity is required. The camera view and light sensor could sometimes be accidentally covered by a coat, especially during winter.

3.5 Data Analysis

Feature logs were analyzed by data window using Excel and raw data were used to calculate sensitivity and specificity, given in Equation 4.5 and 4.6 [2].

$$Sensitivity\ \ (\%) = \frac{Number\ \ of\ \ TruePositives}{Number\ \ of\ \ TruePosives + Number\ \ of\ \ FalseNegatives} \times 100 \quad (3.5)$$

$$Specificity\ \ (\%) = \frac{Number\ \ of\ \ TrueNegatives}{Number\ \ of\ \ TrueNegatives + Number\ \ of\ \ FalsePositives} \times 100 \quad (3.6)$$

All data were analysed to identify whether the state was a true positive (system identifies a CoS when a CoS occurred), true negative (system does not identify a CoS when no CoS occurred), false positive (system identifies a CoS when no CoS occurred), false negative (system does not identify a CoS when a CoS occurred).

Chapter 4: WMMS Design

The overall evolution of the wearable mobility monitor is explained in Figure 4.1. The first three steps were discussed in the literature review (Section 2.1.1) and the fourth step was explained in section 2.3.2. The fifth step consists of the first BlackBerry WMMS [2]. This chapter presents the design and hardware limitations of the new WMMS developed in this thesis.

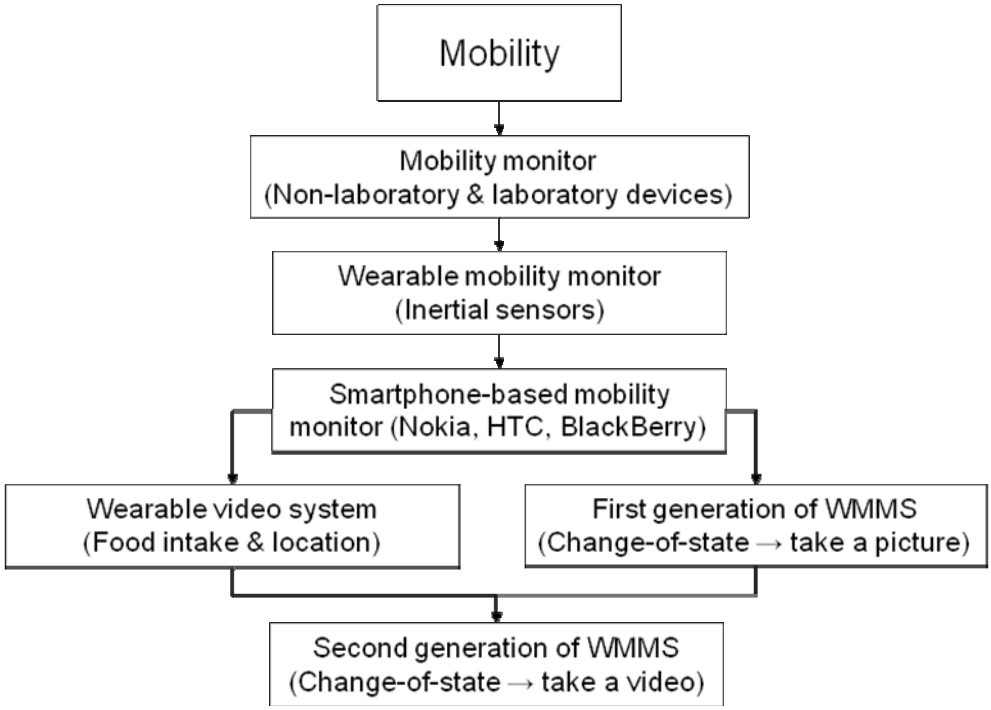


Figure 4.1: The evolution of wearable mobility monitoring system.

4.1 System Architecture

The new WMMS is shown in Figure 4.2, with the Smartphone located at the waist to record accelerations, GPS data, video, and audio. This information is then transferred to a computer for analysis. Subjects could perform any activity, anywhere during mobility monitoring. BlackBerry Smartphones provide an ideal platform for the

WMMS with capabilities for recording, processing, storing, and transmitting acceleration, GPS, and video information. The system can subsequently send mobility data to a hospital external server for assessment of how people with disabilities move outside the healthcare clinic. Received data at the server could be further analyzed, and feedback could then be given back to the user, if required. New BlackBerry devices that integrate accelerometers and video capture capabilities provide an opportunity for mobility assessment using only integrated sensors. Using the Blackberry 9550-Storm2, data were collected with the phone positioned in a passive holster.

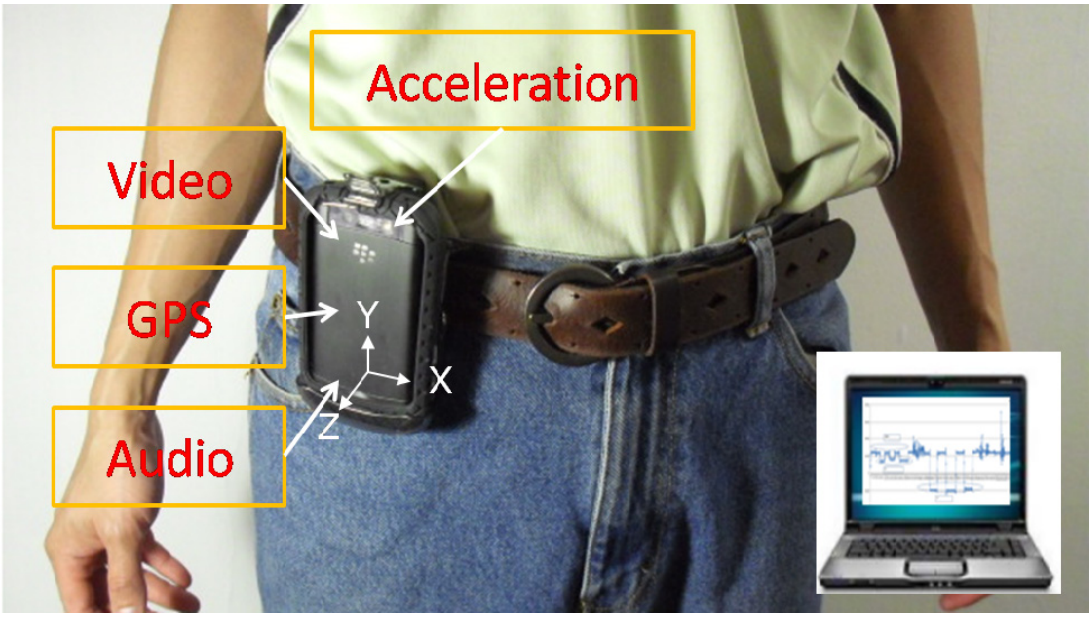


Figure 4.2: WMMS 2.0 system.

4.2 Design Criteria

The WMMS design criteria must meet the objectives of the research. The system has to be reliable and easily available. The design criteria can be considered within the context of hardware, software, and signal processing.

4.2.1 Hardware Specification

The WMMS needs to be light, have long battery life and large memory, and be a small, wearable device. The current research uses a BlackBerry Smartphone with a built-in accelerometer and onboard video camera.

4.2.1.1 Sensors and Devices Comparison

Since the WMMS needs to be worn on the body, weight and size are a great concern. The system needs onboard computer-like capabilities and the ability to capture video. Smartphones have PDA-like functions and additional communications capabilities. Building on the use of a BlackBerry in the previous WMMS prototype, a BlackBerry Smartphone was the first choice as the platform for the new WMMS. The comparison between various BlackBerry devices is shown in Table 4-1 [55]. From this comparison, BlackBerry 9800 was the best choice because higher video resolution and quicker processing time. However, this BlackBerry device (October 2010) was not released on time to be usable for this research. Therefore, the BlackBerry 9550 was chosen as the hardware platform of the WMMS.

Table 4-1: Hardware comparison of BlackBerry (BB) Smartphones [55].

Devices	BB 9000	BB 9550	BB 9800
GPS with A-GPS (assisted GPS)	Y	Y	Y
Video	2.0 MP	3.15 MP	5 MP
Accelerometer	No	2g	2g
CPU (MHz)	624	624	624
RAM	128 MB	256 MB	512 MB
SD Card (up to)	16 GB	32 GB	32 GB
Operation temperature	0 ~ 40 °C	0 ~ 40 °C	0 ~ 40 °C
Weight	136 grams	160 grams	161.1 grams
Volume	142.56 cm^2	97.96 cm^2	100.48 cm^2
Operating system version	4.7	5.0	6.0

4.2.1.2 Acceleration

The accelerometer produces two kinds of acceleration information. One is orientation, which is in three dimensions relative to the handset and the ground. Orientation is difficult to use for activity classification because the BB operating system only provides discrete values of 0, 1, 2, 3, 4, 5, and 10. The “0” value represents the left side of the BlackBerry Smartphone moving upwards. “1” represents the right side of the Smartphone moving upwards. “2” represents the top side of the Smartphone moving upwards. “3” represents the bottom side of the Smartphone moving upwards. “4” represents the front side of the Smartphone facing the sky. “5” represents the backside of the phone facing the ground. “10” represents an unknown orientation [22].

Figure 4.3 presents a test of orientation output during walking, sitting, and standing. The signal cannot be used since the accuracy obtained is not able to identify specific activities but can roughly determine if the state or activity had changed.

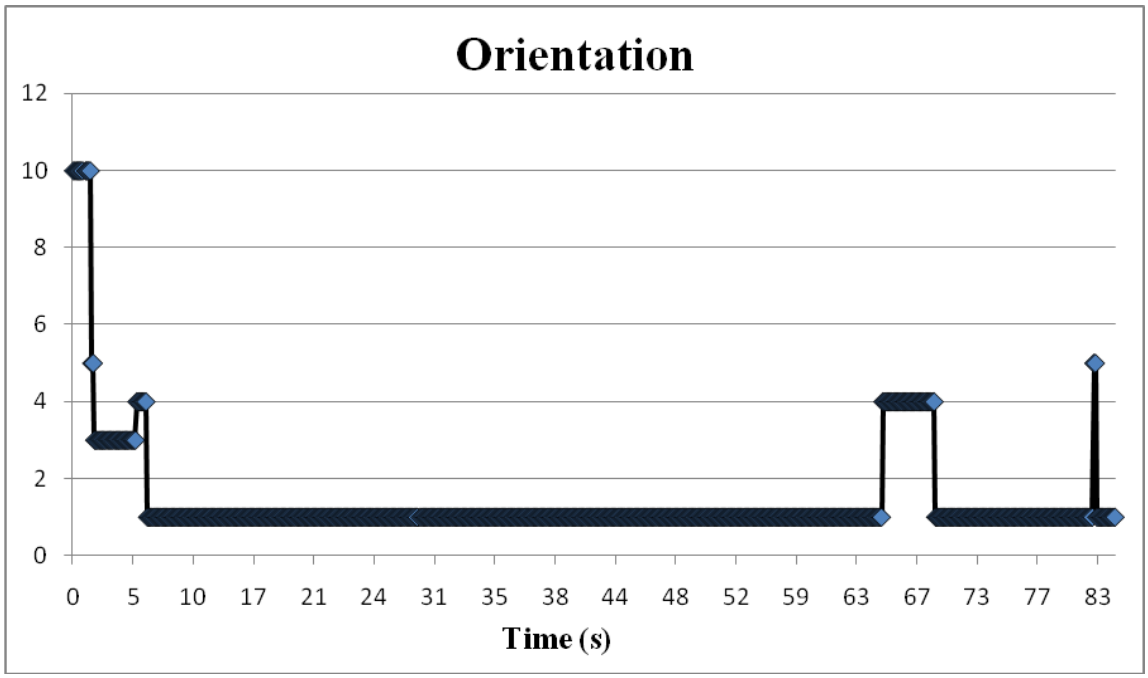


Figure 4.3: Orientation for sitting and standing states.

The other acceleration information produced by the BlackBerry is raw acceleration data. Figure 4.4 illustrates the relationship between the acceleration axes. Positive X indicates that the BlackBerry Smartphone is moving toward the left side of the phone and negative X indicates that the Smartphone is moving towards the right side of the phone. Positive Y indicates that the phone is moving toward the top side of the phone and negative Y indicates that the phone is moving toward the bottom side of phone. Positive Z indicates that the phone is moving toward the front side of the phone and negative Z indicates that the phone is moving toward the backside of the phone. Accelerometer sensitivity is ± 2000 cm/s^2 (2g). The average sampling frequency of accelerometer is 20 Hz and the maximum sampling

Figure 4.4: Blackberry acceleration vectors.

frequency, which can only be maintained for a few seconds without data loss, is 30 Hz. The accelerometer is a three-axis microelectromechanical system.

4.2.1.3 Limitations of the Tri-axial Accelerometer

The accelerometer has two limitations. The first is the rotation effect [56]. The acceleration is directionally related to any movement and must be fixed in a position on the body throughout the motion. Accelerometer rotation causes miscalculation of vertical acceleration. The WMMS uses a holster to fix the device on the front and right pelvis.

The second limitation is errors due to vibration while riding in a moving vehicle. Bouten et al [39][38] found that vibration from vehicles generated high accelerations. These high values were not from voluntary human movement. Using GPS features instead of acceleration features during a car ride could decrease false positives from the vibration of vehicle to detect the change of car ride start and stop.

4.2.2 Software Development

Software selection is the first step for software design. Software programs were required to obtain raw data. Signal processing was necessary for filtering and calibration, reducing the false information, such as artefact effects, signal noise, etc. Finally, the analysis was useful to distinguish from among a variety of activities.

4.2.2.1 Software Selection

Editing BlackBerry programs requires three software programs. BlackBerry Operating System (OS) 5.0 was used in the implementation of programs. Eclipse SDK was the chosen programming platform. BlackBerry Desktop Manager was used to transfer programs to the Smartphone. In Table 4-2 different operating system versions had many different programming library features such as acceleration, audio, and video control [55]. The highest compatible operating system version was 5.0 for BlackBerry 9550, (April 2011). This version was chosen for design and development.

Table 4-2: Functions comparison of BlackBerry operating system versions [55].

Functions	OS 4.7	OS 5.0	OS 6.0
Video	Y	Y	Y
Audio	Y	Y	Y
GPS	Y	Y	Y
Acceleration	N	Y	Y
Auto Flash in Video	N	N	Y

The Eclipse programming platform was the best supported tool for Blackberry development, is a free and open resource, is a multi-language software environment, and includes an integrated development environment (IDE) and plug-in system (Table 4-3 [57]). Java is the main language on the Eclipse platform and BlackBerry interface.

Table 4-3: Functions comparison of Eclipse SDK and Sun SDK [57].

Functions	Sun EE 5.0 SDK	Eclipse plug-in BlackBerry 5.0
Auto editing	Y	Y
Signature tool	Y	Y
BlackBerry Project model	N	Y

4.2.2.2 Software Design Procedures

The WMMS was developed using Eclipse 3.5, BlackBerry Java SDK 5.0, and BlackBerry operating system 5.0, EXCEL, and MATLAB. Accelerations, GPS location, and video were collected at the phone's maximum sampling frequency. Excel was used to analyze thresholds and algorithm structure. After results were verified in Excel, the algorithm and thresholds settings were programmed into Eclipse and then transferred to the BlackBerry operating system, as shown in Figure 4.5.Whenever the thresholds needed to be changed, the results could be quickly obtained from Excel without data download and program transfer time. The software development procedures and guidelines are shown in Appendix A.

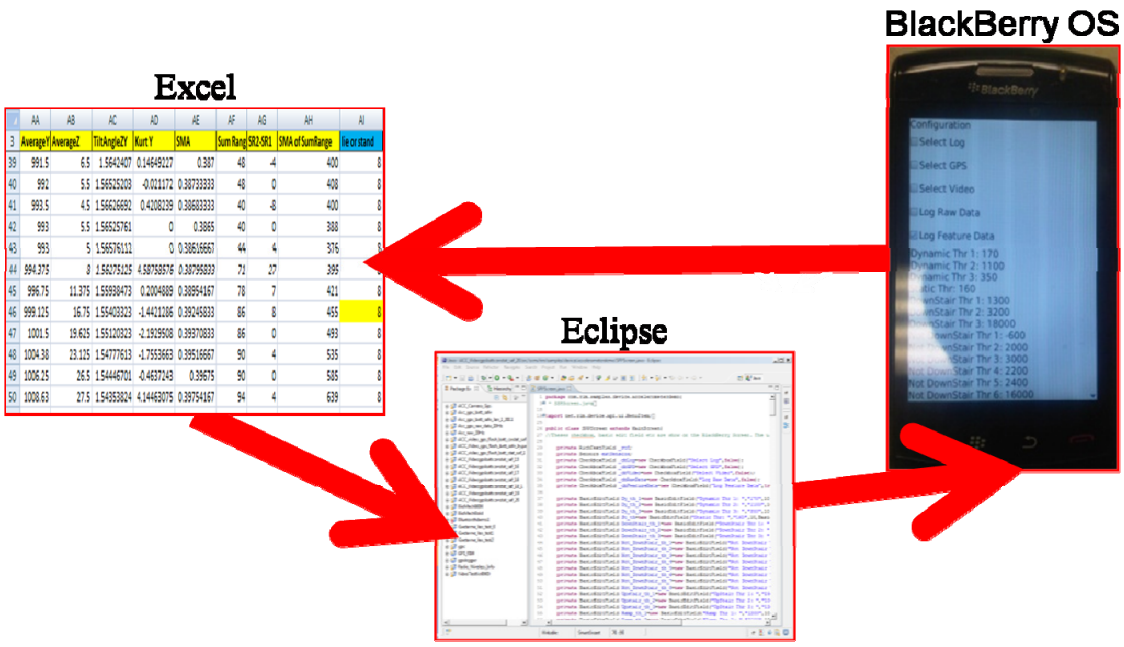


Figure 4.5: Main design flow of the new WMMS.

4.2.2.3 Main Structure of Eclipse Program

The main structure of the WMMS program is shown in Figure 4.6. The flow chart is similar to the previous WMMS [2]. The new WMMS adds raw data and feature log generation from the onboard acceleration and Smartphone video functions. Mobility Context, which is an extension of the user interface (UI) application, links with the APPScreen. The AppScreen is an entrance script to run SPPScreen and to turn on the network, such as Bluetooth. The SPPScreen, which is a configuration script, loads the changeable configuration setting and shows measurement models and configuration selections. The script executes the Sensors script, which is the main script. Before running the Sensors script, the configuration setting must be set. The setting could be by default or selections by the user. The Sensors script is linked with GPSdata, BBCamera, LogFile, and OutputObject. Furthermore, object name helps to identify parameters and variables, so OutputObject is created.

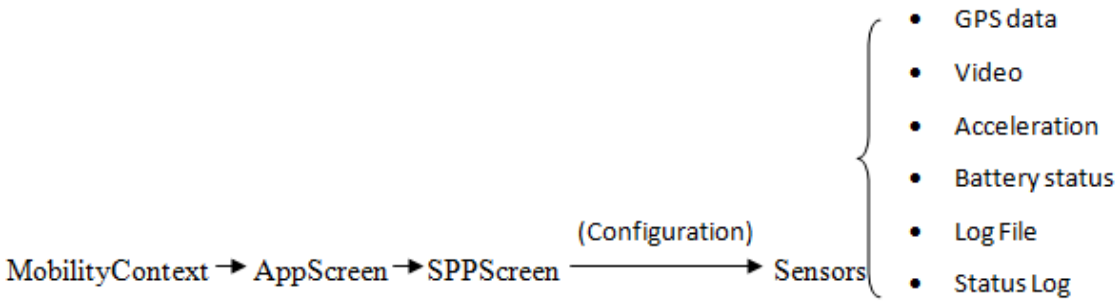


Figure 4.6: The main structure of WMMS program.

The time-out threshold had to be set after the sensors stopped transmitting data, to avoid missing data. The main command, which is Thread, increases the time-to-wait value.

The script MobilityContext pushes the screen from the application. The UiApplication should be imported first. The main structure is shown in Appendix B.1.

The data can be divided into raw data and output data from log files. Output data includes the feature log, the results of CoS, and activity recognition summary. Sensors was the main program that collected data from the accelerometer and the program linked with GPS, audio, and video programs.

4.2.2.4 Basic Program Structure

The section presents the programming methodology for obtaining raw data.

a. Acceleration

Research In Motion (RIM) provided the programming interface for obtaining acceleration raw data [55]. Orientation needed to be set, and the channel had to be opened to obtain acceleration. Orientation was the direction relative to the ground. The function accelerometerSensor.openRawDataChannel() opened the accelerometer channel and raw data was captured using getAccelerometerData(). Data could be displayed on the screen or stored on SD card or the default phone memory. The main structure is shown in Figure 4.7 and the detailed code is shown in Appendix B.2.

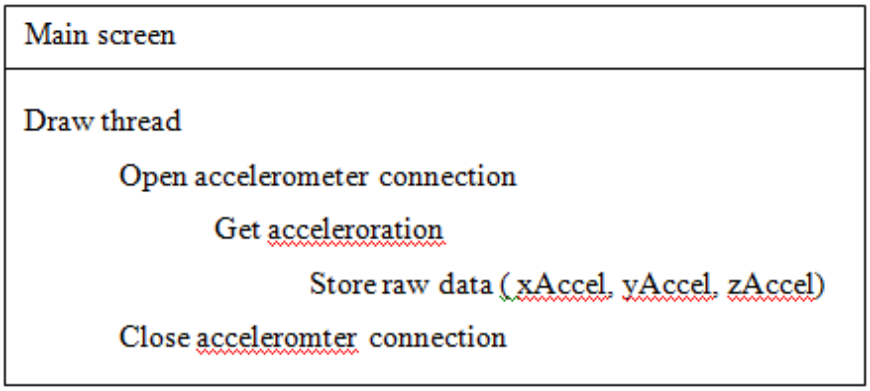


Figure 4.7: Acceleration data collection program structure [55].

b. GPS

RIM provided the programming interface to obtain raw GPS data [55]. Location provider was the starting point to represent information related to location. Location listener received the specific information. Location class represented the standard set of location information: coordinate time stamp, position, speed, and heading. The main structure is shown in Figure 4.8. The detailed code is shown in Appendix B.3.

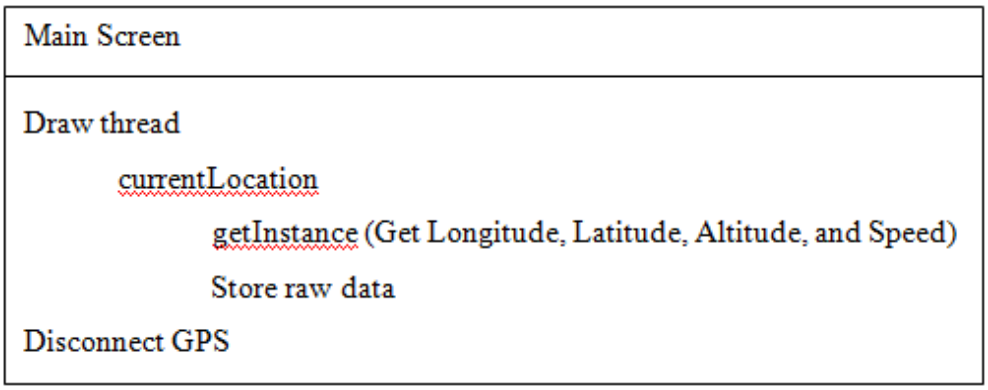


Figure 4.8: GPS data collection program structure [55].

c. Video Recording

RIM provided the programming interface to obtain video clips [55], but the structure did not work well for the CoS triggered WMMS application. Thus, a new function stream and thread were required. Video recording is similar to audio recording because the programming flow is the same. Creating a video media player captures video, with the video encoding parameter available to obtain video property values.

VideoControl controlled video objects and RecordControl controlled recording activity for these objects. OutputStream associated the recorded stream and recording flow. File format and path could be set prior to recording. When OutputStream started to record data, data would be written in a proprietary path and format, such as, .sbv or .3GP. The main structure is shown in Figure 4.9. The detailed code is shown in Appendix B.4.

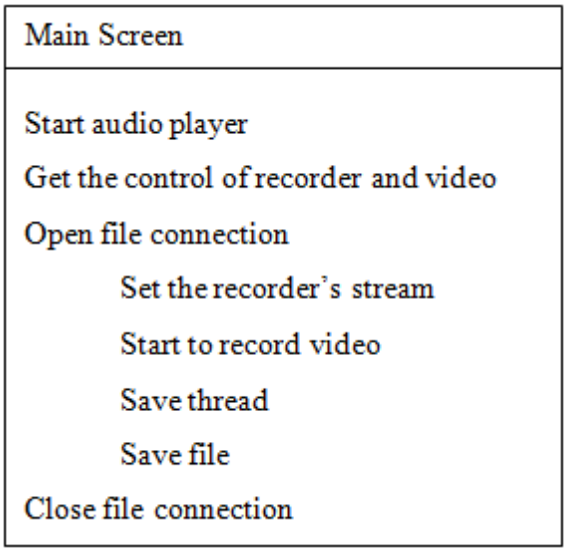


Figure 4.9: The main video recording program structure [55].

d. Safety Setting

Since the WMMS design was tested on five able-bodied people, safety considerations were a necessary step. When any experiment is run for a long time, the battery can become hot. The safety restrictions thus ensure operation within a safe range [2]. The detailed code is given in Appendix B.5.

4.2.3 Signal Processing

The new WMMS algorithm is presented in Figure 4.10, including signal processing, feature extraction, and signal classification. The WMMS saved the feature log and raw data (produced once every 0.125 second) since future work might use the raw data to develop new algorithms. The data were obtained directly from the Smartphone. The new algorithm determined changes of state in order to trigger a video clip. All data were stored on a 16G SD card.

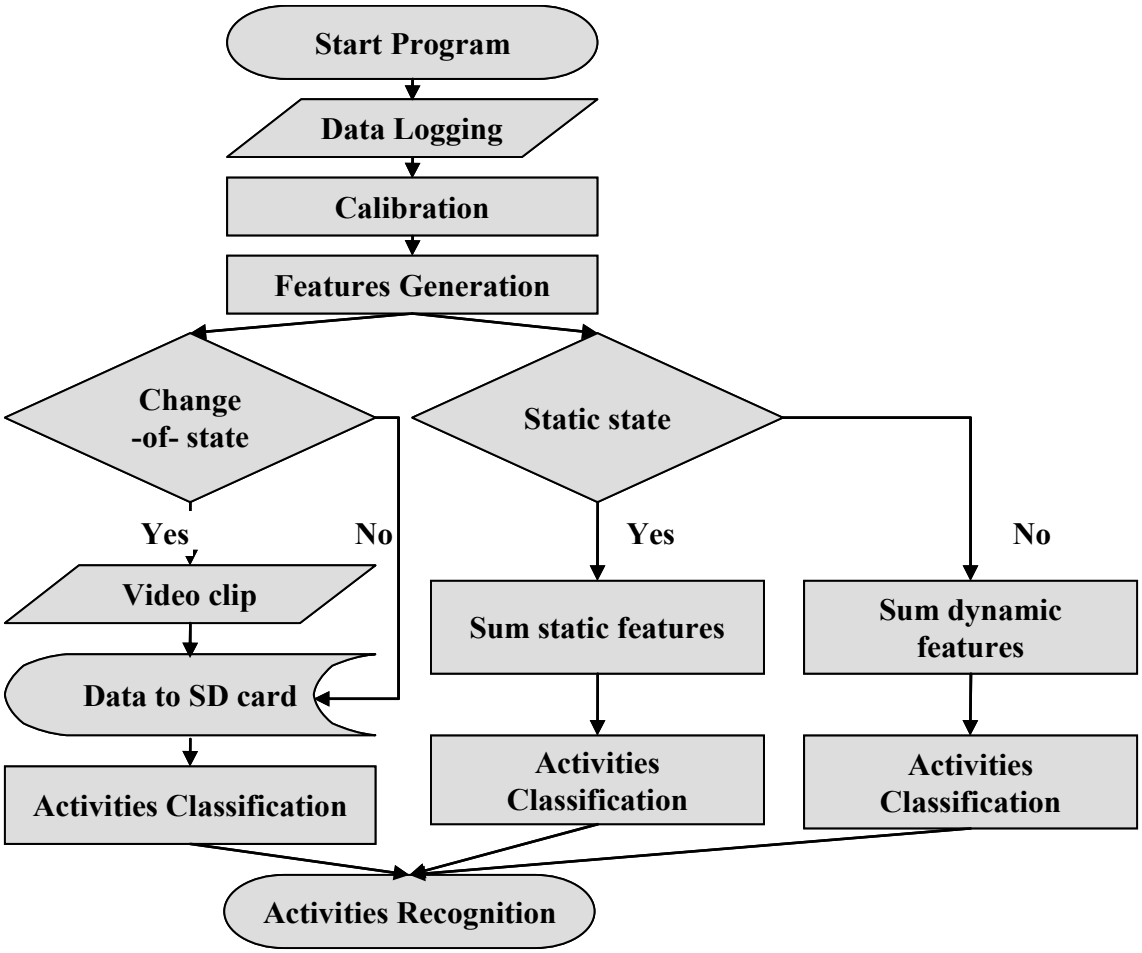


Figure 4.10: WMMS II algorithm and signal processing.

4.2.3.1 Signal Pre-processing Analysis

Accelerometer signals usually require calibration for direct current circuit offset and signal drift. The signal, which was generated by sensors with direct current components, could be easily calibrated since there are no phase effects related to alternate current components.

According to the rotation angle concept (see Section 3.1), the phone is laid flat on a desktop and records for two hours. Subsequently, the phone is rotated 180 degrees on the desktop and records for two hours. Accelerations along X, Y, and Z directions are tested by the same method. Data analysis showed drift and standard deviation of -0.00011g and 0.0046g for the x-axis, -0.0015g and 0.0046g for the y-axis, and 0.0016g and 0.0054g for the z-axis every hour. These drift rates would affect inclination angle (Section 4.2.3.2) by

less than 2 degrees per hour. When the worst drift was calculated, each hour has 0.15 degree drift for inclination angle. Therefore, the WMMS system would require accelerometer recalibration every 12 hours. Further, accelerations had a direct current circuit offset (X-axis: 0.015g, Y-axis: 0.035g, and Z-axis: 0.027g), as shown in Figure 4.11. Based on these effects, a compensation value is set in the program.

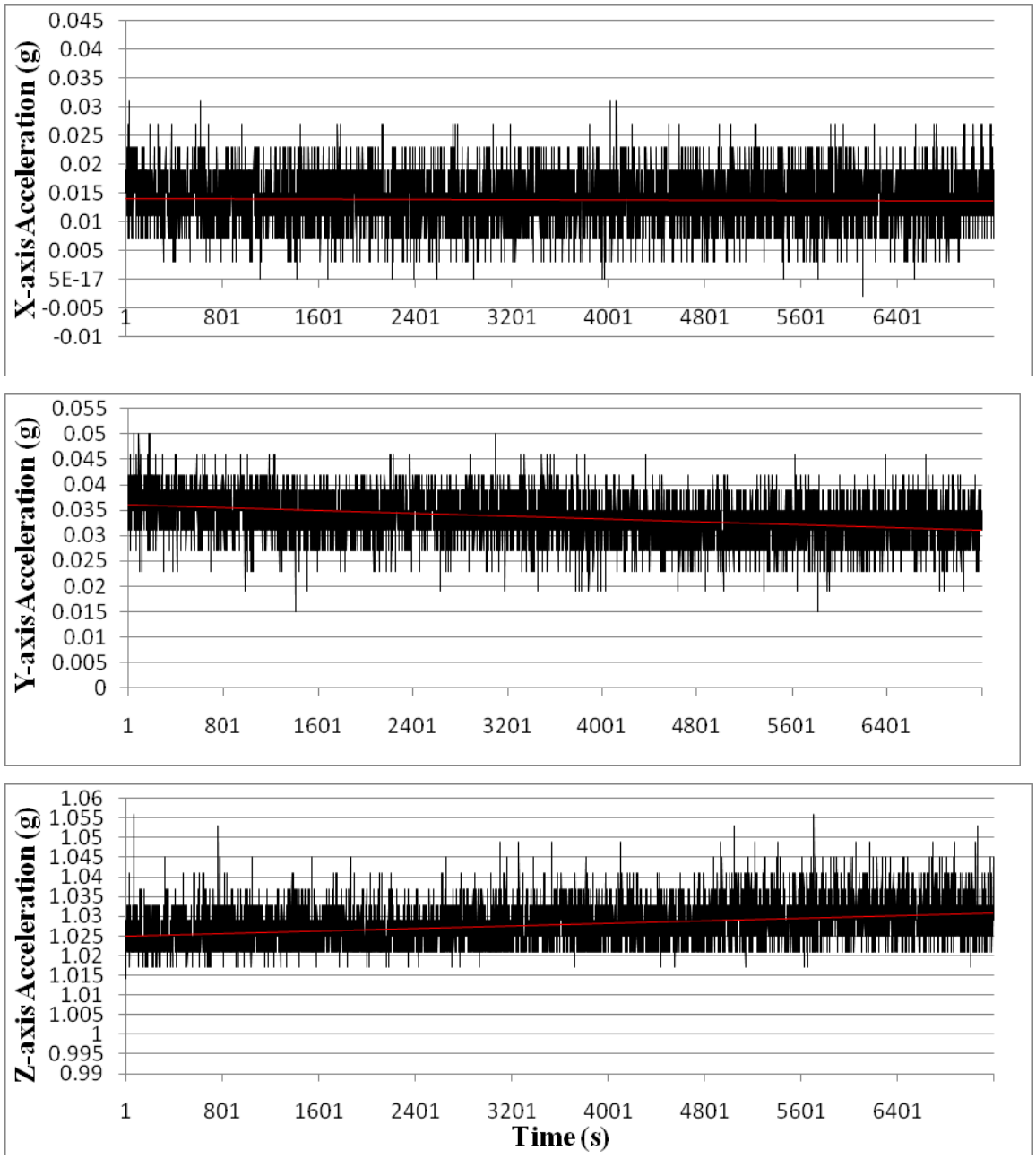


Figure 4.11: Drift and offset for acceleration-X, Y, and Z during two hours.

Signal "features" can be extracted from the raw acceleration data. Mean values recognize absolute location and orientation. Standard deviation can evaluate static and dynamic states. Video information, which includes audio data, can easily recognize a dynamic state. GPS data can obtain an outdoor location. These features could be used to distinguish mobility activities.

4.2.3.2 Feature Extraction

Features were developed based on those used in first version WMMS, with the exception of Signal Magnitude Area and Skewness-Y. Signal Magnitude Area (Equation 4.2) and Skewness-Y (Equation 4.3) could not be used in the new WMMS [2] to identify climbing stairs because the BB accelerometer had a smaller range than the accelerometer used in the Smartholster of the first version WMMS. Signal Magnitude Area could only reach 0.6 with the ±2g accelerometer because of signal clipping, whereas Signal Magnitude Area would reach 2 with the ±6g accelerometer (Figure 4.12).

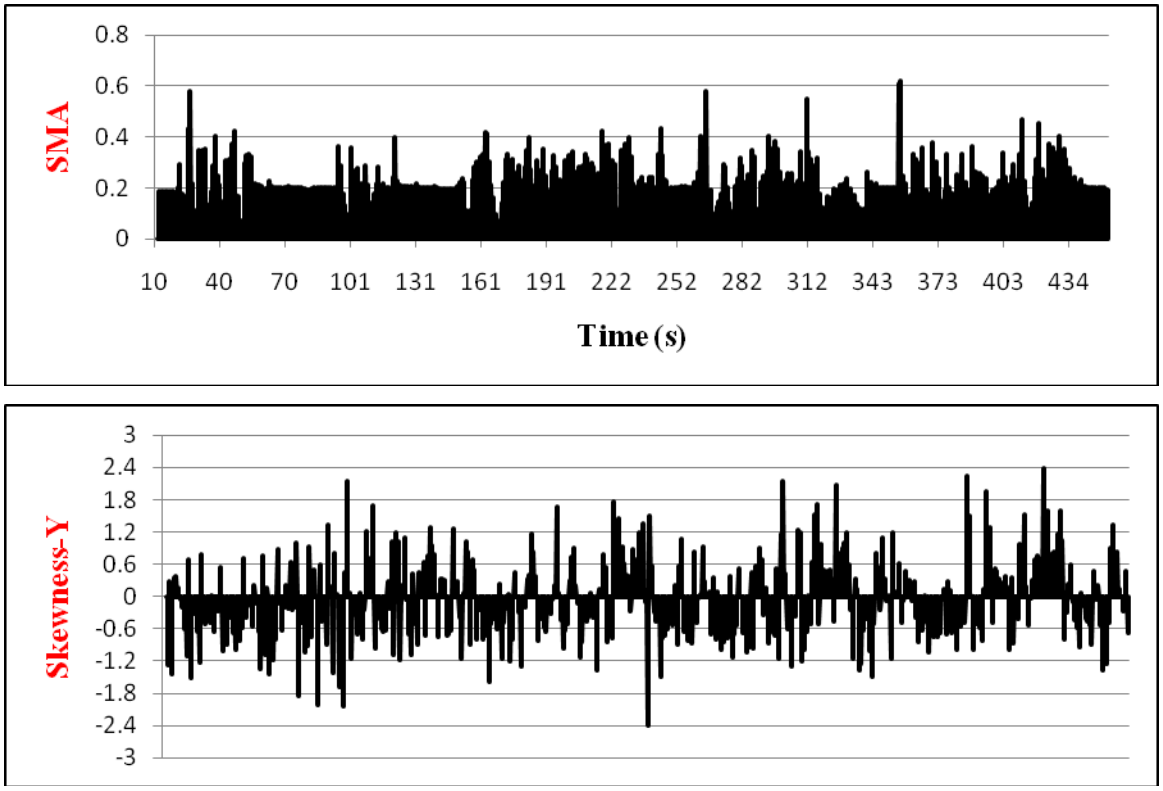


Figure 4.12: Signal Magnitude Area and Skewness-Y for BlackBerry 9550 for walking on stairs, riding an elevator, walking on level ground, and standing.

According to Equations 4.2 and 4.3, smaller accelerations will generate smaller Signal Magnitude Area and Skewness-Y, so the variation of Signal Magnitude Area and Skewness-Y becomes small and thus less meaningful.

- **Inclination angle**: Inclination angle (Equation 4.4) distinguished sitting, lying, and standing for the classification of static activities [2].
- **Y acceleration standard deviation (STD-Y)** (Section 2.4.2): The standard deviation of the y- acceleration over a 1-second window defined static and dynamic movement states [2], see also Equation 4.1. STD-Y was chosen to identify dynamic states because the Y direction had more variation than the X and Z directions during dynamic states, and the variation of mobility had more changes in the Y direction overall (Figure 4.13).

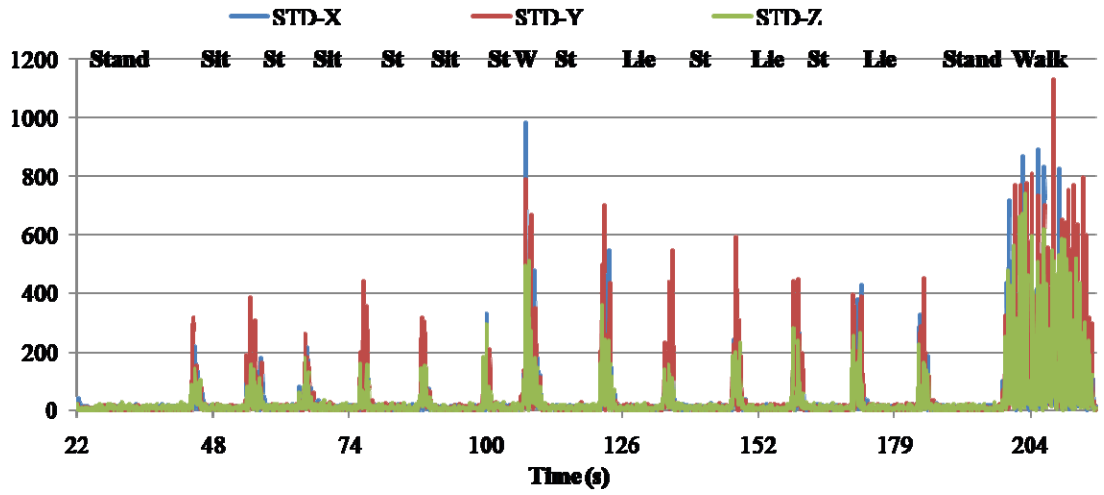


Figure 4.13: STD-Y V.S. STD-X and Z for standing (St) and walking.

4.2.3.3 Determination of CoS

In this study, a CoS was defined as a change between mobility states. The WMMS was to recognize CoS for the following activities:

- Start/Stop moving (e.g., walking)
- Going up or down stairs/ramp
- Postures change (e.g., standing, sitting, and lying)

- Posture transitions (e.g., stand-to-sit, sit-to-stand, stand-to-lie, lie-to-stand)
- Environment change (e.g., elevator)
- Small movement change (e.g., brushing teeth, combing hair, washing hands, drying hands, moving plates, moving a kettle, toasting bread, preparing a meal, and washing dishes)
- Car ride start/stop

4.2.3.4 Signal Classification

If the activity classification can be visually determined from a graph of raw data, the accuracy and sensitivity will be close to one hundred percent. The disadvantage of this approach is the long time and experience required making a good judgment. Therefore, using a threshold to judge activities can save time and requires no prior experience. However, accuracy and sensitivity needed to be improved with this approach.

4.3 Hardware Limitation

The limitations of the onboard accelerometer are that acceleration cannot be recorded during video recording, low accelerometer sensitivity, and lower sampling frequency than desirable. These limitations are due to the BlackBerry operating system and accelerometer specifications. The issues were presented at the RIM seminar in Waterloo in December 2010 [58].

For the BlackBerry 9000 and 9550, acceleration cannot be recorded for the duration of a multimedia recording, a minimum about 2.9 seconds for the WMMS (Table 4-4). This issue limits the algorithm's ability to judge CoS when activities change continuously within 4 seconds, including stop time and the time for feature calculation. For example, if a person sits in a chair and then stands up within 4 seconds, the standing will not be detected or identified by the accelerometer-based algorithm. However, the video clip may be used to identify the extra movements.

Table 4-4: Timing log of acceleration and video collection.

Real-time (ms)	Accumulated Time (second)	Take a video clip	Stop time (Second)
1290448701112	37	N	
1290448701232	37	N	0.12
1290448701353	37	Y	0.12
1290448704281	40	N	2.93
1290448704401	40	N	0.12
1290448722491	58	N	
1290448722612	58	N	0.12
1290448722732	58	N	0.12
1290448722853	59	Y	0.12
1290448725717	61	N	2.86
1290448725837	62	N	0.12
1290448725957	62	N	0.12
1290448726078	62	N	0.12
1290448791862	128	N	
1290448791982	128	N	0.12
1290448792103	128	N	0.12
1290448792223	128	Y	0.12
1290448795181	131	N	2.96
1290448795301	131	N	0.12

Accelerometer sampling rate was evaluated while manually shaking the Smartphone for approximately two minutes. Ten trials were recorded for a range of sampling frequencies, 4, 8, 10, 15, 20, 25, 30, 35 Hz. The maximum sampling rate was 30 Hz for accelerations only; however, only 5 seconds of accelerometer data would be collected before no additional data was received; therefore, the maximum reliable acceleration sampling frequency was 20 Hz. The reason is that BlackBerry accelerometer directly sends acceleration out after opening accelerometer channel. The sampling rate is controlled by waiting time (i.e., 50 ms is the waiting time for 20 Hz).

Standard deviation, miss count, and average frequency are important variables for system reliability. Five trials were executed by sitting, standing, lying, and walking with video recording (8 Hz), video control setting (10 Hz), and accelerometer only (20 Hz). The

detailed procedure is at Appendix E.1. For accelerometer only, the standard deviation of the sampling rate was 1 Hz, and the averaged missing count was 0.1 to 0.2 percent. Further, when video control was enabled, a requirement for video capture, the maximum sampling frequency was 8.3 Hz, standard deviation was 0.5 Hz, and the averaged missing count was 0.3 to 0.5 percent. The low available frequency was limited by BlackBerry OS 5.0 because the original design used accelerations only to detect screen rotation and not for any application that required detailed acceleration data. This issue is also prevalent in other Smartphones, such as some Nokia and Android-based devices [11], [4].

Issues were also found with the video player and camera flash control. The video clip format is .sbv, which can only be played by a BlackBerry 9700 and/or a newer Smartphone. Other multimedia players, such as windows media player, KM player, BlackBerry 9550 video player, etc., only showed video images without sound. This issue should be resolved with future operating system versions.

The Smartphone has a light sensor to detect light intensity and has a flash. However, the flash could not be controlled in automatic mode through Java Application Programming Interface (API) 5.0. If the automatic mode worked, the flash function would help to identify a dark environment. A programming interface was not available to access the phone's light intensity information. If the intensity level could be logged, the data could be used to recognize indoor and outdoor environments.

4.4 System Evaluation Summary

With the basic program structures (Section 4.2.2.3) and signal analysis procedures in place (Section 4.2.3), the next steps were evaluation and editing. A newer version of the BlackBerry OS might be helpful to solve some issues, such as the auto flash function of video control and the light meter detection.

Chapter 5: Change-of-State Determination

This chapter focuses on the development and testing of CoS determination algorithms for feature calculation. The content is a manuscript that was submitted to the Journal of Medical and Biological Engineering: Wu, H.H., Lemaire, E.D., Baddour, N., Activity Change-of-state Identification using a BlackBerry Smartphone, Journal of Medical and Biological Engineering, submitted for publication, 2011.

5.1 Introduction

Mobility, defined as the ability to move easily between two separate points, is a significant factor for maintaining quality of life [1]. In a healthcare environment, understanding how people move in their daily life is important for making the best clinical decisions and to evaluate mobility in both the community and hospital. Mobility assessment outcomes affect decisions on rehabilitation devices, treatment, and environmental modifications.

During a clinician-patient encounter, mobility status is typically self-reported. This can provide biased information that does not accurately reflect the person's mobility outside of the clinic. Functional scales (CBMS [59] , Berg Balance Scale [60], Dynamic Gait Index [54], COVS [61], etc.) can be used in the clinic to improve the clinician's understanding of a person's mobility status; however, these tests do not describe how a person moves in the community [62], [63]. Ideally, mobility measures would be performed in the community to provide valid and useful information for healthcare providers and researchers [23] .

Wearable Mobility Monitoring Systems (WMMS) combine sensors, computing, and multimedia technologies that can be worn on the body to measure mobility and can be used to generate person's mobility profile. WMMS have clinical and non-clinical

applications that include real-time activity monitoring, environmental effects, and emergency help [2]. WMMS are a form of a body sensor network that, with appropriate network access, can allow mobility monitoring and analysis anywhere [3]. The physical rehabilitation sector, people with mobility disabilities, and researchers would benefit from WMMS applications.

A Smartphone is a cellular telephone with computer-like features that include cameras, telecommunications, and the ability to run sophisticated applications. Smartphones, are ubiquitous and multi-functional, incorporating sensors such as accelerometers, GPS, video camera, light sensor, temperature sensor, gyroscope, and magnetometer. These sensors can be used for activity recognition. Most research has used cell phone platforms to recognize multiple activities by accelerometer only [2], [4], [11]. Smartphone accelerometer and/or microphone signals have been used for activity classification, but the recognition accuracy for activities, such as walking on stairs, was low.

Wearable video systems with sensors, such as a GPS, ECG, and accelerometer, have been used to record information about location, movement, and contexts such as food intake and activity [6], [7], [8] with context identification for food intake and activities. These contextual applications did not use image analysis to help with context identification. Byrne et al. [6] frequently recorded static images using a SenseCam, which included a wearable camera, tri-axial accelerometer, passive infrared sensor, and light sensor. The accelerometer was used to detect movement in order to trigger picture capture when changing from static to dynamic states. Manual judgment was used for feature extraction and context identification (i.e., road, staircase, door, or grass). Other Lifelog research [3], [11] used GPS, environment change (heat flux or temperature), and voice as an index to find the appropriate images to review. These sensor-video systems were not designed for mobility assessment.

Previous research has demonstrated the viability of a Smartphone as the basis of a WMMS [2]. This prior work confirmed that the BlackBerry WMMS, using an external accelerometer and internal digital camera, could identify walking movements, standing, sitting, and lying down and provide contextual information on the activities through

qualitative image analysis. The ability to correctly identify CoS, with few false positives and false negatives, is crucial since appropriate image/video capture depends on real-time CoS determination. The BlackBerry platform has the sensors, multitasking operating system, processing power, storage, security, and acceptance in the healthcare field that are required for the target application.

This paper presents the development of a WMMS that incorporates new algorithms and uses only the internal sensors and on board Smartphone video for CoS identification. By using only built-in hardware, the system is accessible for consumers. Since a Smartphone is part of many people's lives, there is no cost associated with additional hardware and the consumer will be familiar with the technology. In this article, only the ability of the new WMMS to identify changes of state is investigated. The research intends to recognize activities with context. In accelerometer only respect, stairs, ramp, elevator activities are poor to identify. Therefore, using a change-of-state related to accelerometer only judgment triggers video recording and then classifies activities from video.

5.2 Methods

The new WMMS was evaluated using the Blackberry 9550 (Storm2) Smartphone, using only the internal sensors and multimedia capabilities to identify mobility CoS. A CoS occurs when a person changes their mobility state; such as, standing to walking, walking to stair ascent, sitting to standing, etc. In the final application, the CoS would trigger the phone's video camera to record the activity from the user's perspective. The CoS, sensor features and video data would be used in post-processing to identify each activity and the context of the activity.

5.2.1 System Architecture

The WMMS application was developed using Eclipse 3.5, BlackBerry Java SDK 5.0, and BlackBerry OS 5.0. Acceleration, GPS location, and video were collected at the phone's maximum sampling frequencies. During video capture, the maximum acceleration sampling frequency was 8 Hz for all axes. Accelerometer sensitivity was ± 2g and the BlackBerry system divided this range into ± 2000 units. The highest video

resolution of the 3.15 MP camera was 480x352 pixels and the video frame rate was 30 frames per second. One set of GPS data was extracted for each 1-second data window (1Hz). To simulate event-based video capture, three-second video clips were captured every ten seconds. In future prototypes, video would only be captured after a CoS occurred. During video capture, no accelerometer or GPS data were available, a limitation imposed by the BlackBerry system used in this study.

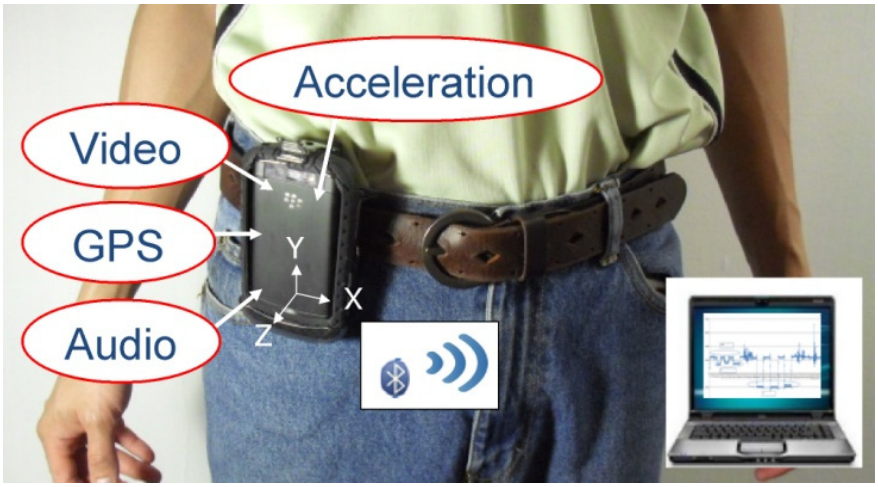


Figure 5.1: Smartphone in holster with sensor orientation.

Before the WMMS trials, the BlackBerry was set flat on a level table and 20s of raw acceleration data were capture. Acceleration offsets were calculated for each axis.

The Blackberry Storm2 was positioned in a passive holster with the camera facing outward (Figure 5.1). All data were simultaneously collected by the WMMS application and stored on a 16GB SD card. Following tethered data transfer to a computer, MATLAB was used to extract features from the 3-axis accelerometer data. A decision tree (Figure 5.2) was programmed in Microsoft Excel to identity CoS. Excel was used for all statistical analyses.

5.2.2 Feature Calculations

Features that were sensitive to changes in mobility status were extracted from the 3-axis accelerometer and GPS data,. All data were processed in 1-second windows, with 8 samples per window. The features used in the CoS algorithms are:

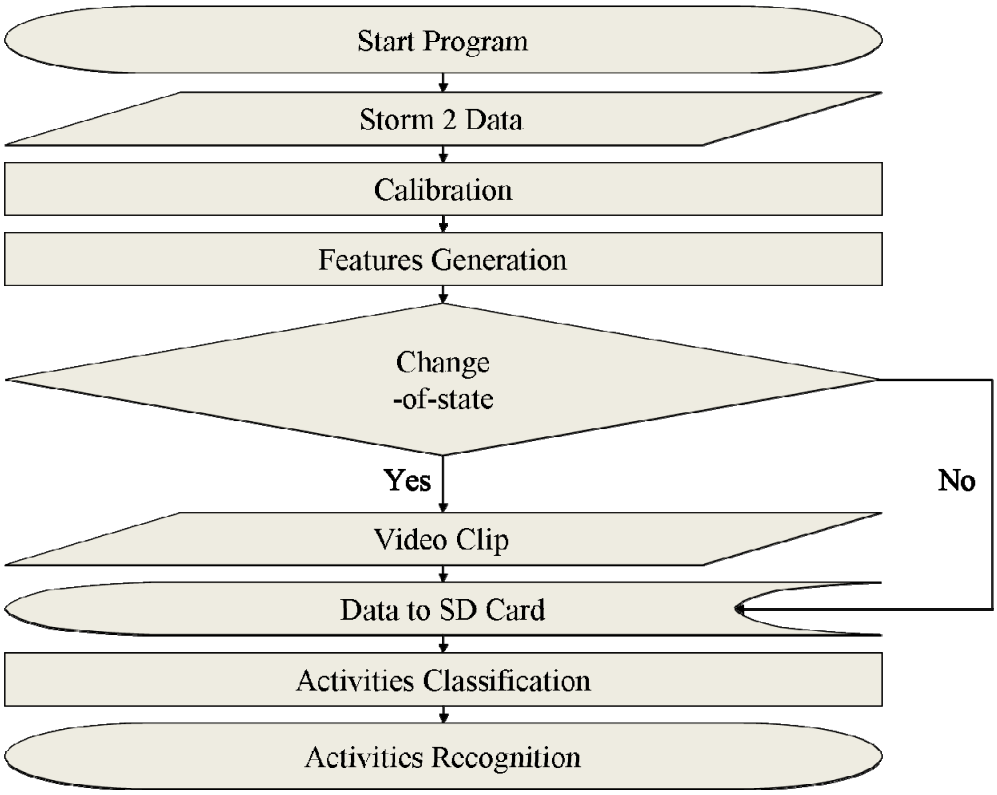


Figure 5.2: WMMS algorithm and signal processing.

- **Y acceleration mean (Mean-Y)**: Mean-Y is used to recognize standing.
- **Y acceleration standard deviation (STD-Y)**: STD-Y defines static or dynamic movement states and helps identify elevator use. The X and Z acceleration standard deviations (STD-X, STD-Z), whose formulas are the same as Equation 5.1, are included in the model to improve true positive identification between static and dynamic states.

$$STD\ Y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - m)^2} \tag{5.1}$$

where y_i is the ith Y-acceleration value, m is the mean Y-acceleration, and N is the data window size.

- **Y-acceleration range (Range Y):** Range Y is used for stairs and ramp identification. When possible, range is used instead of standard deviation to reduce computational load and improve real-time capabilities. $MaxY$ and $MinY$ are the maximum and minimum values within the data window.

$$Range\ Y = (MaxY - MinY) \tag{5.2}$$

- **Sum of ranges (SR)**: SR is used to distinguish walking on level ground, walking on stairs, and ramp states. Range X, Y, and Z are the range values from the three acceleration axes.

$$SR = (Range\ X + Range\ Y + Range\ Z) \tag{5.3}$$

- **Signal Magnitude Area of SR (SMA-SR)**: SMA-SR, the sum of all SR values within the data window, identifies stair navigation. Since the time interval is constant, SMA-SR becomes an efficient estimation of the SR integral. For this paper, N is set to eight (i.e., previous eight data windows, relating to the previous eight seconds of data).

$$SMASR = \sum_{i=1}^{N} SR_i \tag{5.4}$$

- **Difference in SR values (DiffSR)**: DiffSR is the difference between sum-of-ranges for the current window (*SR2*) and previous window (*SR1*). DiffSR is used to recognize mobility CoS over time, especially for stairs descent identification.

$$DiffSR = (SR2 - SR1) \tag{5.5}$$

- **Range X plus Range Z (Rxz)**: Rxz identifies stair navigation.

$$Rxz = (RangeX + RangeZ) \tag{5.6}$$

- **Sum of GPS speeds (SGPSspeed)**: SGPSspeed is the sum of GPS speeds within the previous 20 data windows and is used to identify vehicle mobility. The larger data window decreases false positives when a car stops at a stop sign or traffic light (Figure 5.3).

$$SGPSspeed = \sum_{i=1}^{N} GPSspeed \tag{5.7}$$

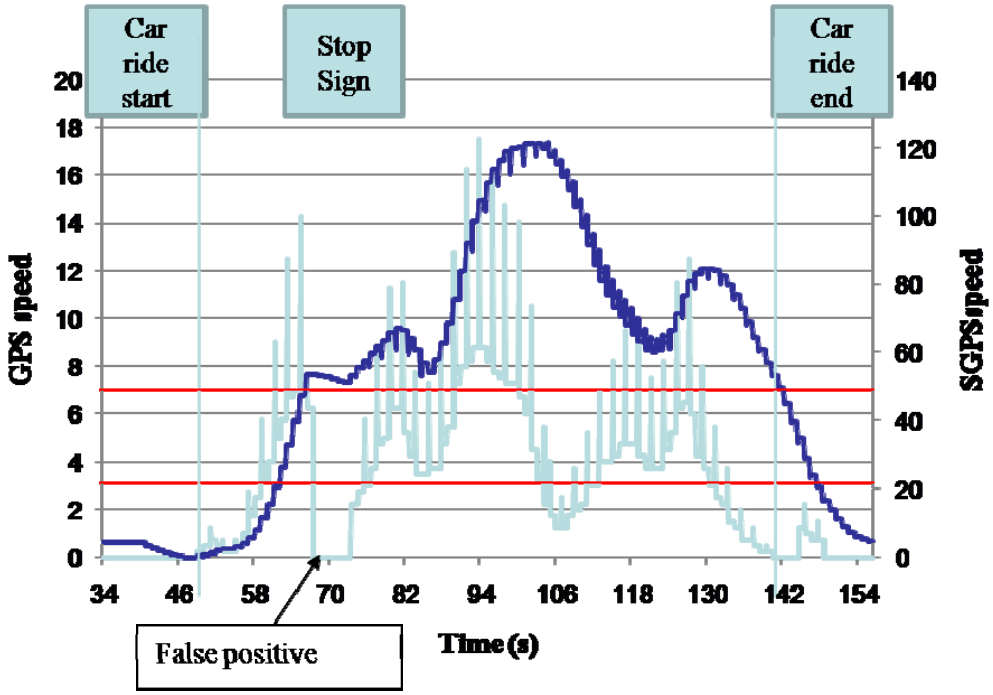


Figure 5.3: SGPSspeed for the reduction of false positives. Upper line is the high threshold, and the lower line is the low threshold.

5.2.3 Change-of-State Identification

A decision tree with pre-set thresholds was used for feature analysis (Figure 5.4). These thresholds were initially defined by inspection of output graphs from Excel, using single-subject data (i.e., separate data set collected using the same subject). Subsequently, thresholds were iteratively modified and adjusted based on algorithm performance until appropriate activity classification was achieved. Double thresholds (Table 5-1) were required to reduce CoS false positives (i.e., both upper and lower threshold must be passed before a CoS is triggered). For example, accelerometer signals fluctuate about the upper threshold value during walking; therefore, a CoS is only identified when the lower threshold is passed. Since the objective of this algorithm was to identify changes of state, identification of level ground walking was not required since this is a linking activity between mobility events. Level ground walking falls within the dynamic state, but other activities such as jumping or skipping would also be categorized as dynamic within the current model.

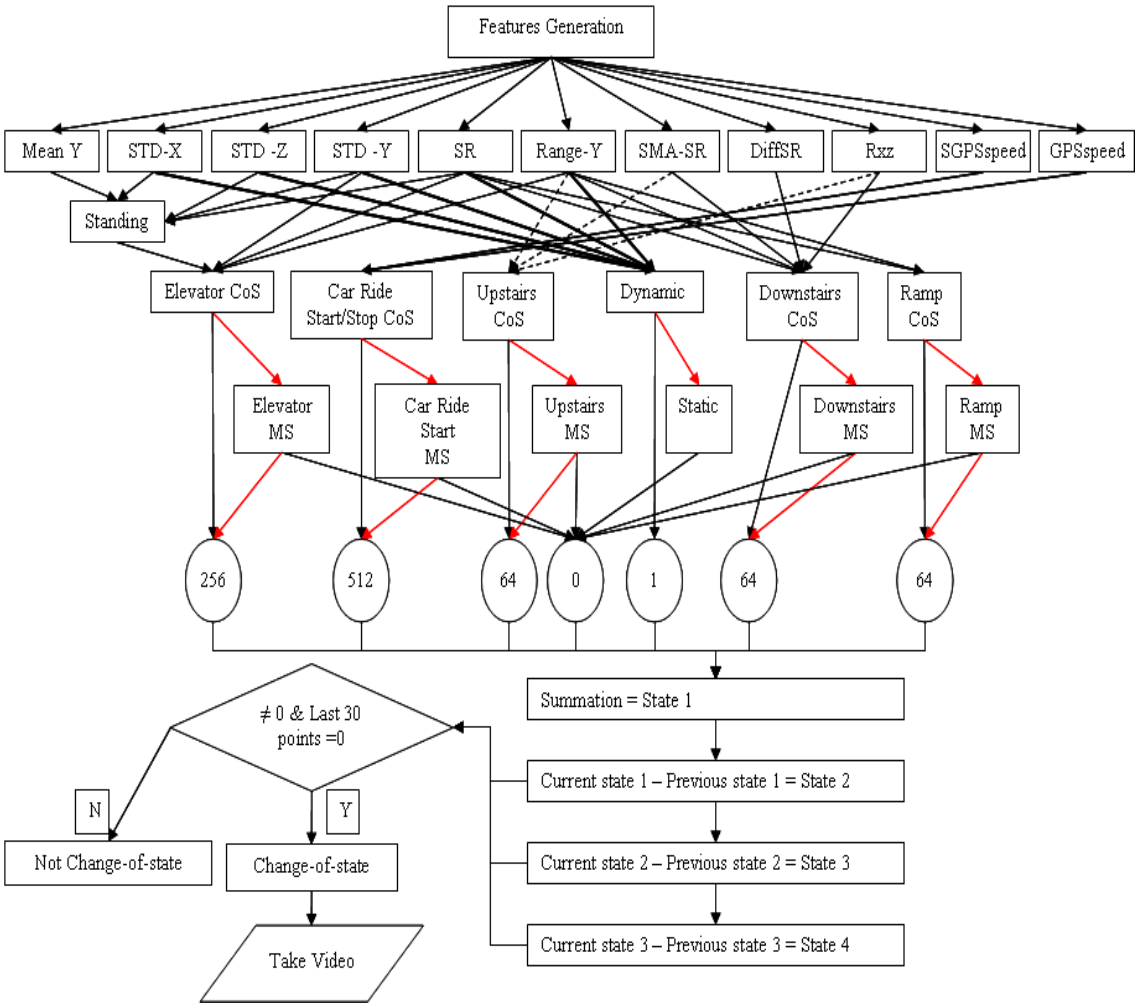


Figure 5.4: State determination algorithm. CoS is change-of-state and MS is maintaining the previous state. The values for state identification and activity classification are: “256” for riding an elevator, “512” for car ride start/end, “64” for walking on stairs or walking on ramp, “1” for dynamic state, “0” for non-defined state.

When the CoS determination is TRUE, the features connect to an output value to enable subsequent activity classification: “256” for riding an elevator, “512” for car ride start/end, “64” for climbing stairs or walking on ramp, “1” for dynamic state, “0” for non-defined state. The static state includes sitting, standing, and lying. A change-of-state will trigger video recording.

Further, for FALSE state decisions, the activity is considered the same as the previous state. The eleven features were combined in various groupings to identify the current

mobility state. The combination approaches are listed in Table 5-1. In future versions of the software, optimization could be introduced that removes features that are not needed for some users. For example, if a user just has a small X-direction body sway during walking on stairs, Rxz could be removed for the downstairs judgment. Processing efficiency would be improved because fewer features are calculated.

Table 5-1: Feature combination methods with threshold values for CoS and maintaining the previous state (MS). “256” is riding an elevator, “512” is car ride start/end, “64” is walking on stairs or walking on a ramp, “1” is a dynamic state, “0” is a non-defined state.

Activity	Combination Method	Output
Standing	Mean-Y>960 and Mean-Y<1200 and STD-X<20 and STD-Y<80 and STD-Z<20 and SR<350	TRUE/FALSE
Elevator CoS	Standing=TRUE and (STD-Y>45 or (SR>160 and Range Y>50))	TRUE=256
Elevator MS	STD-Y<50	TRUE= In elevator (0) FALSE=Previous state
Car ride CoS	GPSspeed>2.74	TRUE=512
Car ride MS	SGPSspeed<50	TRUE= In car ride (0) FALSE=Previous state
Upstairs CoS	SMA-SR>22000	TRUE= 64
Upstairs MS	Range Y<500 or (Rxz<1000 and SMA-SR<15000)	TRUE=Going upstairs (0) FALSE=Previous state
Dynamic	(SR>1100 and Range Y>350) and (STD-X>160 or STD-Y>160 or STD-Z > 160)	TRUE=1
Static	STD-Y<70	TRUE= 0 FALSE=Previous state
Down stairs CoS	Range Y>1300 or SR>3200 or SMA-SR>24750	TRUE= 64
Down stairs MS	(DiffSR<-600 and SMA-SR<3000 and SR<2000) or (SR<2200) or (SR<2400 and SMA-SR<18000 and (Rxz < 830 or Range Y<500))	TRUE=Going downstairs (0) FALSE=Previous state
Ramp CoS	Range Y>1200 or SR>2500	TRUE=64
Ramp MS	Range Y<600	TRUE=On ramp (0) FALSE=Previous state

Output values were summed for each window. A CoS is identified when output sums for the current and three previous windows (i.e., four windows in total) are each not equal to zero (Figure 5.4).

5.2.4 Evaluation Protocol

One able-bodied subject performed a series of consecutive movements: standing, sitting, lying down, taking an elevator, walking on stairs, walking on a ramp, and starting/ending a car ride. Walking on level ground occurred between each activity. Three trials were captured for all activities while accelerometer, GPS, and cell phone video were recorded. Each activity took approximately 10 seconds to complete. For sensitivity and specificity analysis, a project assistant manually identified CoS timing during post-processing using the accelerometer output and task sequence timing from the video. The detailed procedure is at Appendix E.2.

In addition to the movement analysis, additional tests were performed by a) continuously recording video for 30 minutes, b) continuously recording accelerometer output for 12 hours, and c) continuous recording accelerometer and GPS with 3 seconds of video recorded every 10 seconds (45 minutes test duration). Three trials were performed for each test. GPS delay was assessed during the mobility tasks by measuring the time between walking from indoors to outdoors and the first GPS signal. Each video clip was qualitatively assessed in terms of quality, camera movement/shaking, context, image direction, and sound content. The Smartphone video was played back on a BlackBerry 9700 device for analysis.

5.3 Results

WMMS data, video clips, and data calibration were successfully captured/calculated simultaneously with the BlackBerry multitasking environment. The Storm2 9550 accelerometer sampling rate, with BlackBerry video control running, averaged 8.25 Hz (STD=0.49) [58].

By combining and weighting the range, sum, and covariance statistics, good CoS classification was possible for standing, sitting, and lying (Table 5-2), with only two false

positives. Walking CoS accuracy reached 96%. For specificity, 52 false positives out of 6851 true negatives occurred during walking (i.e., walking CoS was identified while person was still in the walking state). Many of these false positives could be removed in post-processing.

Table 5-2: CoS sensitivity (SE) and specificity (SP) at the transition between activities (total of three trials), TP = true positive, FN = false negative, FP = false positive, TN = true negative.

Change-of-State	TP	FN	FP	TN	SE	SP
Standing	48	0	1	7092	100.00%	99.99%
Sitting	15	0	1	1644	100.00%	99.94%
Lying	9	0	0	1471	100.00%	100.00%
Walk	62	1	52	6851	98.41%	99.25%
Elevator	0	6	6	1269	0.00%	99.53%
Upstairs	6	3	18	1352	66.67%	98.69%
Downstairs	7	2	19	1183	77.78%	98.42%
Ramp	3	6	22	1176	33.33%	98.16%
Car ride start/end	6	0	0	3002	100.00%	100.00%
Average	157	18	119	25040	89.71%	99.53%

For elevators, the CoS algorithm triggered when the person stopped in front of the elevator; however, the activity was classified as standing instead of taking an elevator. When walking to standing in front of elevator, the change-of-state algorithm triggered video recording. Since a video clip would be captured, the video can be used to correct the status from standing to the elevator classification. A CoS could also be triggered when walking into the elevator or when the elevator moves up/down, providing additional video clips for activity classification.

The sensitivity for walking on stairs was between 66% and 77%. Downstairs walking was easier to classify, with distinct changes to the accelerometer signals (Figure 5.5). Ramp navigation was not well recognized, with a sensitivity of 33%. Change-of-state of starting and ending a car ride was successful in all six cases.

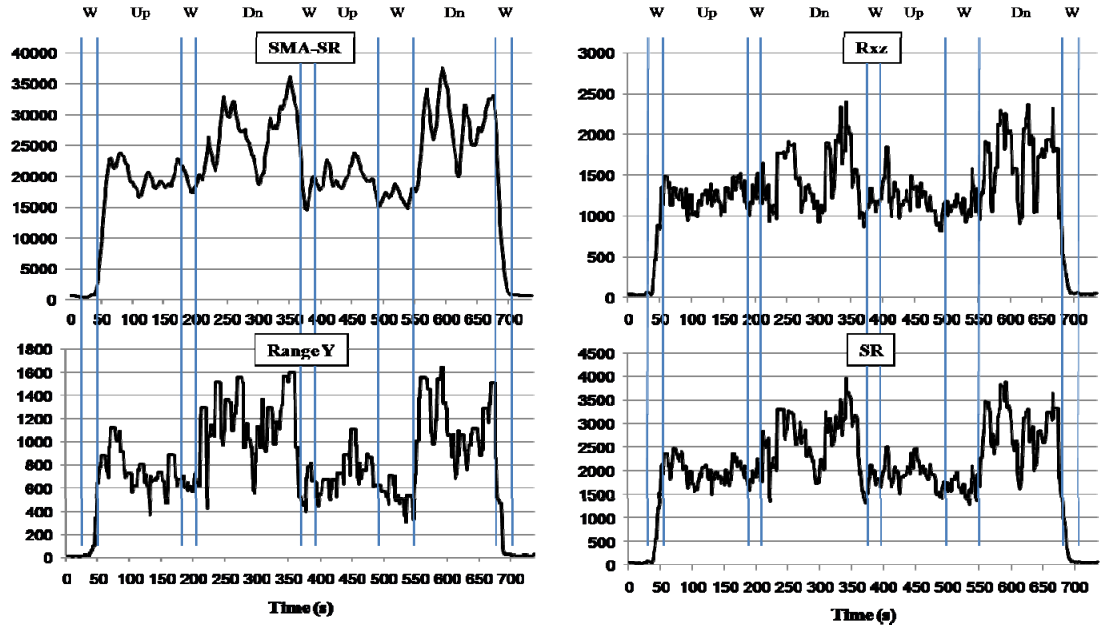


Figure 5.5: SMA-SR, Range Y, Rxz, and SR for walking (W), walking upstairs (Up), and walking downstairs (Dn).

For this study, CoS was evaluated at the transition period between activities, without considering CoS events during an activity. If this transition period is extended to include a CoS determination during the activity, the sensitivity increased to 100% for all activities and false negatives decreased to zero (Table 5-3). Since the objective is to categorize mobility activities, with video post-processing, a video clip would be captured for each mobility task, but not necessarily at the transition point. Therefore, correct classification for each activity would be expected with the video-enabled system. The detailed result is shown in Table D-1.

Since 3D accelerometer signals are similar between walking on level ground and walking on a ramp, unique features are difficult to find. In cases where the person slowed down as they moved from level ground to ramp ascent, the Range Y lower threshold was able to trigger a CoS. Since this method triggered many false positives, a combination of Range Y and SR was used to decrease the false positives for ramp navigation (Figure 5.6).

Table 5-3: CoS sensitivity (SE) and specificity (SP) without consideration of transition timing, TP = true positive, FN = false negative, FP = false positive, TN = true negative.

Change-of-state	TP	FN	FP	TN	SE	SP
Standing	48	0	1	7092	100%	100%
Sitting	16	0	1	1644	100%	100%
Lying	9	0	0	1471	100%	100%
Walk	63	0	52	6851	100%	99%
Elevator	6	0	1	1274	100%	100%
Upstairs	9	0	18	1352	100%	99%
Downstairs	9	0	19	1183	100%	98%
Ramp	9	0	22	1176	100%	98%
Car ride start/end	6	0	0	3002	100%	100%
Total	175	0	114	25045	100%	100%

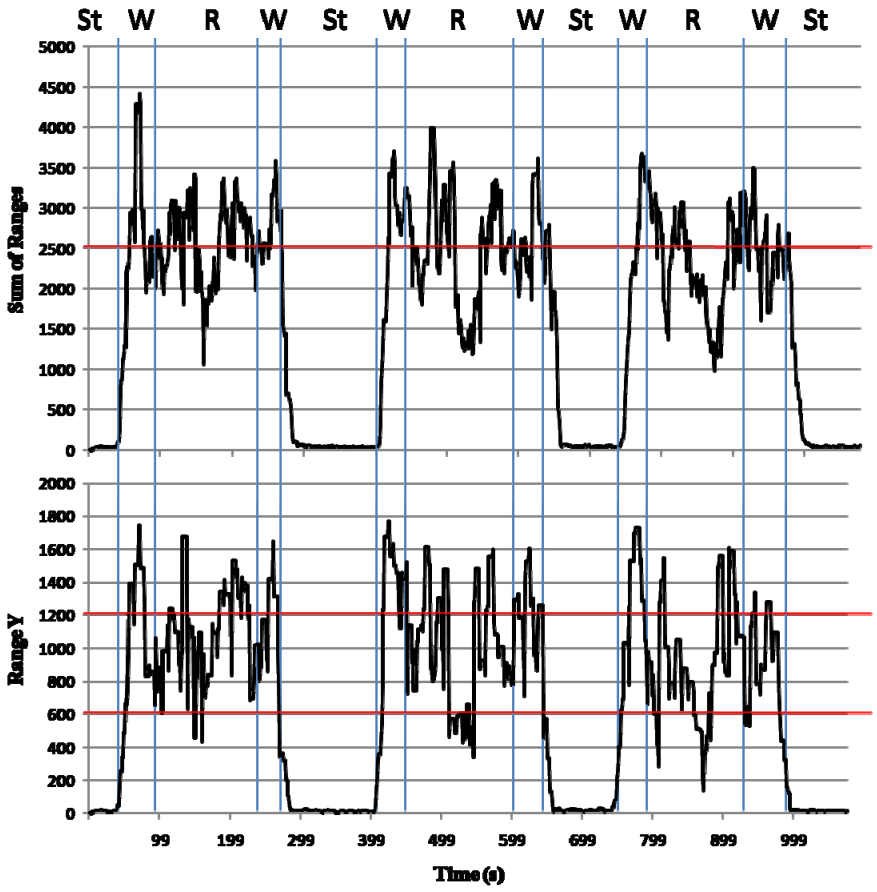


Figure 5.6: Sum of ranges and Range Y for standing (St), walking (W), and ramp walking (R).

Video was collected to help identify the activity and context (Table 5-4). Based on qualitative assessment, video provided better results than still-photos because of the audio information and multiple-consecutive-images. Based on the audio information, an evaluator could qualitatively identify walking on level ground as compared with walking on stairs. Even though the video shook for all dynamic states, the video was useful for post-processing. The camera direction allowed an evaluator to distinguish walking upward/downward and static activities, such as, sitting, lying, and standing. Each activity showed different environments or contexts; including, the inside of a car, stairs, handrail, ceiling, and type of flooring.

Table 5-4: Video clip analysis.

Activities	Context	Shaking images (Y/N)	Image direction	Sound content
Standing	Wall	N	Forward	Quiet
Sitting	Ceiling, wall	N	Forward, upward	Quiet
Lying	Ceiling	N	Upward	Quiet
Riding an elevator	Elevator	N	Forward	Elevator
Walking on flat ground	Ground	Y	Forward	Walking
Climbing upstairs	Stairs, hand rail	Y	Forward, upward	Climbing stairs
Climbing downstairs	Wall	Y	Forward, downward	Climbing stairs
Walking on up ramp	Wall, ceiling, hand rail	Y	Forward & upward	Walking
Walking on down ramp	Wall, ceiling, hand rail	Y	Forward & downward	Walking
Car ride start/end	Inside of car	N	Forward & upward	Car engine

For battery level, 28 minutes of continuous video recording used 10% of the battery capacity. Therefore, approximately four hours of continuous video could be recorded. Acceleration-only recording used 10% of the battery capacity over 4.3 hours; therefore, accelerometer-only could record for 43 hours. The WMMS application, with all sensors and video recording at 10s intervals, required 2% battery capacity over a range of 157 to

301 seconds. Thus, the WMMS could function for 2.18 to 4.18 hours. A larger capacity battery would be required for long-term mobility testing, or the device would require charging during the day. Furthermore, the BlackBerry 9550 had a GPS delay between two and five minutes.

5.4 Discussion

This preliminary assessment of a new BlackBerry Smartphone-based mobility analysis system showed that the WMMS was able to identify a mobility CoS at the transition point between most activities. When the entire activity period was examined, instead of just the transition period, the algorithm was able to identify a CoS for all activities. Since CoS will trigger video clip capture in the final WMMS, video data of each mobility activity will be available for post-processing. These video clips should improved activity categorization accuracy.

For elevators, the algorithm triggered a CoS when the person stopped in front of the doors, which would result in video clip capture, but the activity was classified as standing. During use, elevator acceleration was not sufficient to enable correct classification via the accelerometer. However, video clip analysis should enable correct classification of the elevator state.

Climbing stairs identification improved over previous wearable mobility research that used a tri-axial accelerometer [2], [11] from 60% to 79%. Further, since able-bodied walking gait may not change substantially when ascending or descending a ramp, body-worn accelerometers are often unable to detect a CoS. New altimeter sensors could be used to identify changes in body vertical displacement, thereby helping to characterize incline gait or elevator use. However, this would require external electronics and wireless communications to work with the BlackBerry Smartphone system.

The project methodology only considered a true positive to be at the transition period between tasks, therefore false positives were recorded when a ramp CoS was triggered later in the ramp ascent. In practical use, triggering a ramp CoS at any point in the task would provide the required information.

The algorithms of Haché, et al. [2], Baek, et al. [47], and Allen, et al. [10], used single accelerometer features to identify a movement activity. For example, signal magnitude area recognized stair ascent or walking and standard deviation identified static/dynamic transitions. The new WMMS combined acceleration features to improve CoS sensitivity and specificity for many mobility tasks. Combining features also enabled the CoS algorithms to work appropriate at low sampling rates and lower accelerometer ranges.

For recognizing sit-stand and stand-sit, STD-Y, STD-X and STD-Z output were combined instead of using STD-Y on its own. Since one of three features must be true to trigger a CoS, sensitivity increased from 79 % to 100% without decreasing specificity (False positives = 0).

The SR feature is a calculation-efficient estimate of the acceleration curve integral within the data window. The SR curve was similar to the STD-Y curve, but SR values were several times larger. Therefore, SR thresholds were easier to define. By combining SR and STD-Y, the number of false negatives was reduced, thereby enhancing activity classification. Further, STD-Y on its own produced more false positives than the combination of SR and STD-Y. When SR is removed from the algorithm, false positives for walking and static activities increased from 5 to 7. Figure 5.5 shows that SR could also be used to identify downstairs walking.

Range Y has been used as a feature to identify various activities [11]. However, when a small threshold range was used to enhance detection, the number of false positives increased to an unacceptable level. Increasing the threshold range increased the number of false negatives. Similar results were found when using SR on its own for stair CoS identification. Since SMA-SR provided a smoother curve with similar shape, double thresholds were more clearly defined. Therefore, SMA-SR was combined with SR and Range Y to improve appropriate CoS triggering. DiffSR was effective for locating the first step from level ground to the first step of stair ascent. Rxz was also included to deal with side-to-side accelerations. A combination that included Rxz and DiffSR decreased false negatives during stair navigation.

Only Range Y, Rxz, and SMA-SR were required to detect a CoS for stair ascent. Combining features increased accuracy from 70% to 72% for walking on stairs. When SMA-SR was removed from the algorithm, false positives increased.

The standard battery capacity was insufficient for day-long mobility monitoring. A high capacity battery theoretically would increase the maximum usage time from 4.18 hour to 8 hours. However, regular charging would be required for long-term mobility monitoring. Battery and Smartphone system design improvements in future devices should also extend the data collection time.

The GPS delay will affect various mobility classifications. For driving, automatic identification would be delayed since GPS speed is the main indicator for this activity (i.e., the activity would be classified as sitting or lying). Post analysis of the BlackBerry video could help to correct for this delay since the car would be shown in the video clip when entering the vehicle. If GPS is being used to analyze outdoor movement patterns, this delay would result in an unrecoverable data gap. Other wireless location methods could be combined in the future to provide seamless information.

While the results of this preliminary investigation are promising, this was a single-case study that demonstrated the potential for this change-of-state approach. Further testing with a larger sample (i.e., more subjects and repeated measures) is required to demonstrate widespread application. The thresholds were optimized for the subject based on a previous data set of the subject performing discrete activities. Therefore, a threshold calibration procedure may be required to tune the algorithm for other individuals.

5.5 Conclusion

This study verified that the new BlackBerry WMMS algorithm and hardware platform was effective at detecting changes-of-state over a range of mobility activities. The methods were effective at the 8 Hz data rate and 2g accelerometer sensitivity. In this single-subject analysis, the new WMMS was better at CoS classification than the previous BlackBerry WMMS [2]. GPS delay was also smaller with the new Smartphone. While these results were often better than previous studies, some of this improvement

could be attributed to the threshold optimization that was performed for the subject. The default battery capacity remains inadequate for long-term mobility monitoring, but a standard battery could be used if charged at midday and a high capacity battery could be used if charged in the evening. Subsequent validation studies will investigate the activity classification accuracy, with and without BlackBerry video clip assistance, with a larger sample size.

Chapter 6: Activity Classification

This chapter describes the activity classification algorithms and results of tests on able-bodied subjects. The content of this chapter is the text of a manuscript that was submitted to the Journal of NeuroEngineering and Rehabilitation: Wu, H.H., Lemaire, E.D., Baddour, N., Change-of-state Determination to Recognize Mobility Activities using a BlackBerry Smartphone, Journal of NeuroEngineering and Rehabilitation, submitted for publication, 2011.

6.1 Introduction

Understanding how a person moves within their daily lives is important for healthcare decision-making. Typically, a person's mobility status is self-reported in the clinic, thereby introducing error and increasing the potential for biased information. Functional scales can be administered in the clinic to help gain an understanding of a person's mobility status [59], [60], [61], [62]; however, these tests do not measure how a person moves when they leave the clinic. A quantitative method of characterizing how a person moves in the community would provide valid and useful information for healthcare providers and researchers.

Wearable systems have been developed to evaluate mobility in any environment or location. These portable systems are worn on the body and collect mobility information within the home and community. Examples include accelerometer-based systems [36], **Error! Reference source not found.**, [35], [34] portable EMG [16], and foot pressure devices [64]. Some wearable systems can be used for long-term and easy to use monitoring.

Smartphones are now popular and prevalent. These phones are multitasking computing platforms that can incorporate accelerometers, GPS, video camera, light sensors, temperature sensors, gyroscopes, and magnetometers [65], [66]. Smartphones provide an

ideal interface for mobility assessment in the community since they are small, light, easily worn, and easy to use for most consumers.

As shown in Table 6-1, cell phones have been used to recognize multiple activities by analyzing the phone's accelerometer data. These systems used internal or external sensors. Most systems only used the phone as a wireless data transfer device, not as a wearable computing platform. These systems used an external computer to analyze features. Further, low frequency and low sensitivity outcomes reduced the viability of the internal-accelerometer-only approaches for mobility analysis.

Table 6-1: Tri-axial accelerometer sensitivity, data acquisition frequency, and sensor placement comparison

Author	Placement	Internal sensor	Sensitivity (G)	Frequency (Hz)
Z. He [34]	Pocket	No	3	100
N. Wang [35]	Waist	No	6	50
G. Haché [2]	Waist	No	6	50
A. M. Khan [36]	Chest	No	6	20
J.R. Kwapisz [11]	Front pant pocket	Yes	2	20
Z. Shumei [37]	Waist	Yes	1	1
R.K. Ganti [4]	Waist	Yes	NA	7

For lifelog applications, which log digital memories, wearable video systems were developed with sensors (GPS, electrocardiogram, video clips, and acceleration) to record location and context of a person's daily life [6], [7], [8]. Lifelogs have several issues, including information retrieval, synchronization, light quality, low visual resolution, artifacts, and automatic annotation.

A previous project [2] showed that sensors located in a "Smart-holster" could be combined with a Blackberry Smartphone to provide image-assisted mobility assessment. Accelerometer, temperature sensor, humidity sensor, light sensor, and Bluetooth were integrated into the Smartphone holster. Software was written for the BlackBerry 9000, which had an embedded camera and GPS, as a WMMS. A hierarchical decision tree combined with a double threshold (DT) algorithm classified signals to recognize CoS and activities. The BlackBerry automatically took a digital picture at each CoS. The

preliminary work confirmed that the WMMS, using acceleration and pictures, could identify walking movements, standing, sitting, and lying down, and provide context for these activities.

While the Smart Holster system made advances in the mobility monitoring area, improvements could be made in terms of the CoS algorithms, use of video instead of still images, and classification methods for other activities of daily living. New BlackBerry devices that integrate an accelerometer, GPS, and video capture provide an opportunity for mobility analysis using only integrated sensors. This study evaluated the sensitivity and specificity of a new BlackBerry-based WMMS that only used internal sensors and cell phone video camera to identify mobility activities.

6.2 Methods

6.2.1 WMMS System

The prototype WMMS was developed for BlackBerry OS 5.0 on the Storm2 9550 Smartphone. The WMMS used the BlackBerry 9550 integrated accelerometer, GPS, camera, and SD memory card for mobility data collection. The three-axis accelerometer sensitivity was ± 2g. The 3.15 MP camera had a maximum video resolution of 480x352 pixels and video frame rate of 30 frames per second [55]. During video capture, no accelerometer or GPS data were available. With video control active, the accelerometer sampling rate was approximately 8 Hz [58]. For the WMMS, GPS data were sampled at 1 Hz. As shown Figure 6.1, the WMMS is worn in a holster on the right and front pelvis.

The new WMMS software [67], sampled time, accelerations, and GPS location and saved the raw data to a 16Gb SD-card. Accelerations were processed to calibrate the axis orientation, using the gravity rotation method [54]. The processed acceleration was input into a feature extraction algorithm for analysis with 1-second data windows. Following feature extraction, the features were analyzed in a decision tree (Figure 6.2) to determine if a CoS had occurred. If a CoS was identified, a three-second video clip was captured. The second decision tree (Figure 6.3) was used to categorize the activity. Time, features,

and activity classification were saved to an output file on the SD card. The program code is shown in Appendix C.

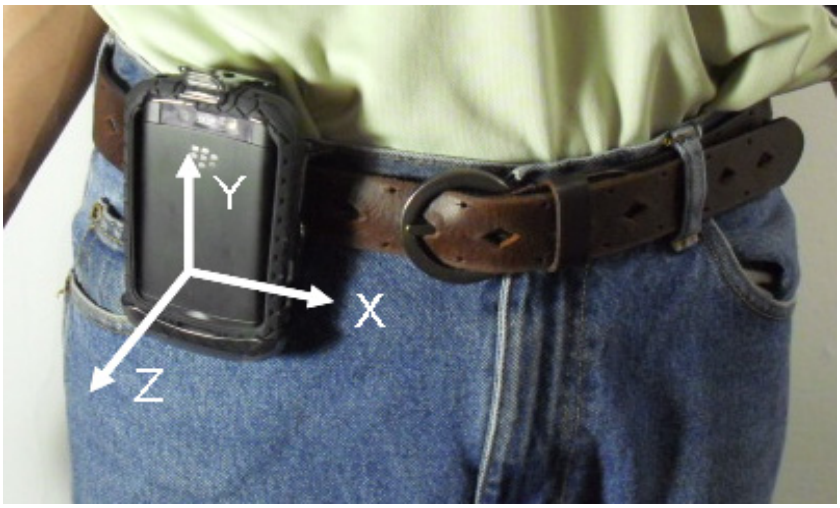


Figure 6.1: Smartphone and holster's location.

6.2.2 Accelerometer Features

Acceleration features were identified that were sensitive to changes in mobility status [68]. These included mean Y-axis acceleration, standard deviation (STD) (eq. 6.1) in X,Y,Z axes, Y-axis range (Range-Y) (eq. 6.2), Sum of Ranges (SR) (eq. 6.3), Signal Magnitude Area (SMA) of SR (eq. 6.4), difference of Sum of Ranges (DiffSR) (eq. 6.5), range of X and Z (Rxz) (eq. 6.6). The inclination angle (eq. 6.7) could distinguish sitting, lying, and standing for the classification of static activities.

6.2.3 Change-of-state / Classification

The CoS state algorithm (Figure 6.2) uses pre-set thresholds for feature analysis. Each feature was independently analyzed using single or double-thresholds and scored as true or false. These Boolean values were combined to recognize a static state, taking an elevator, walking-related movements, and small activities of daily living (ADL) movements, such as meal preparation, hygienic activities, or working in the kitchen. Small movements are between static and walking-related activities, including movement while sitting and standing and various ADL. A unique number was assigned for each activity. A CoS was triggered when the current and previous three windows did not have

the same activity value (i.e., if activity classification for all four windows is not equal to zero, a CoS is recorded).

$$STD-Y=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-m_y)^2} \tag{6.1}$$

$$Range-Y=(MaxY-MinY) \tag{6.2}$$

$$SR=(Range-X+Range-Y+Range-Z) \tag{6.3}$$

$$SMA-SR=\sum_{i=1}^{N}SR_i \tag{6.4}$$

$$DiffSR=(SR2-SR1) \tag{6.5}$$

$$Rxz=(Range-X+Range-Z) \tag{6.6}$$

$$Inclination\quad Angle=\arctan(\frac{m_z}{m_y})(^\circ) \tag{6.7}$$

m_y = mean Y-acceleration, y_i = individual acceleration value, N = samples per data window, *SR2* = sum of ranges in current window, *SR1* = sum of ranges in previous window, m_z = mean Z-acceleration.

6.2.4 Activity Classification

The decision tree for activity classification (Figure 6.3) used the same features and thresholds as the CoS algorithm to classify eight activities; sitting, standing, lying, riding an elevator, small stand-movements, small sit-movements, small lie-movements, and walking. Each activity was assigned a specific number for programming identification. The Dynamic feature was used to separate the analysis into static and dynamic activity groupings. The specific numbers for activities in each group were summed and then combined into groups: walking activities (level ground walking, ramps, and stairs), elevator, sit, stand, and lie. The small movement categories included ADL and movements while sitting and standing.

Following data collection, the video clips were reviewed to help classify activities. Clips were played back on a BlackBerry 9700 phone. Video and audio were available for qualitative assessment of the current activity.

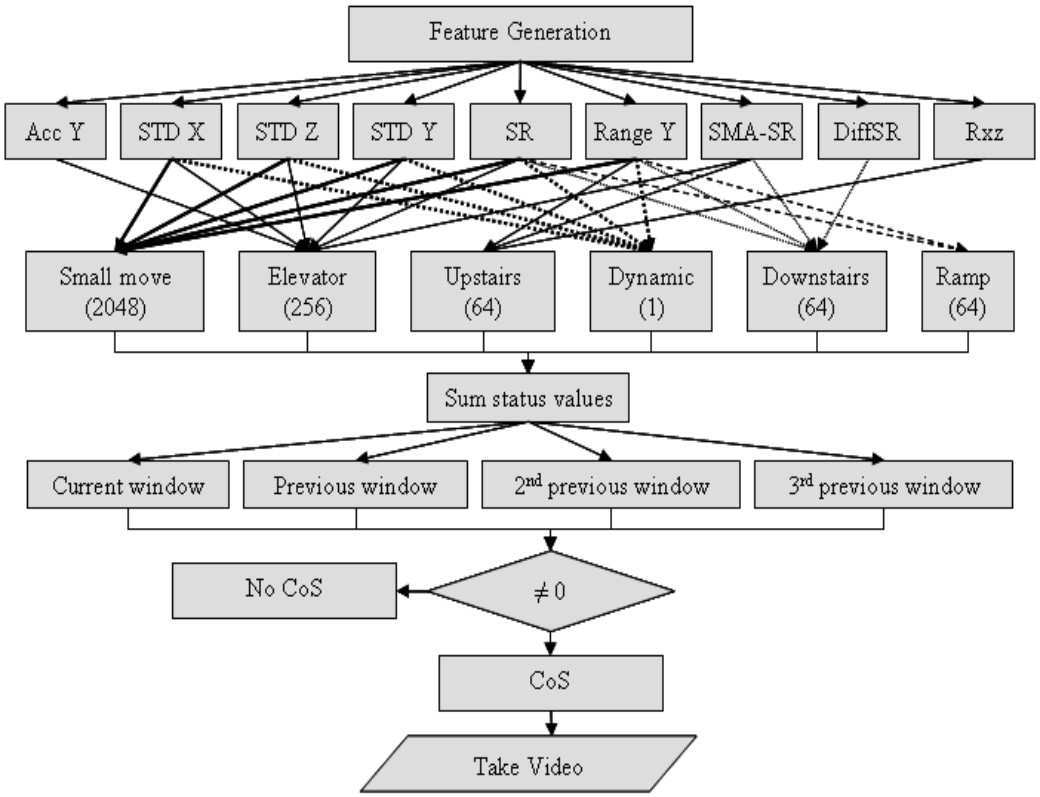


Figure 6.2: CoS determination algorithm.

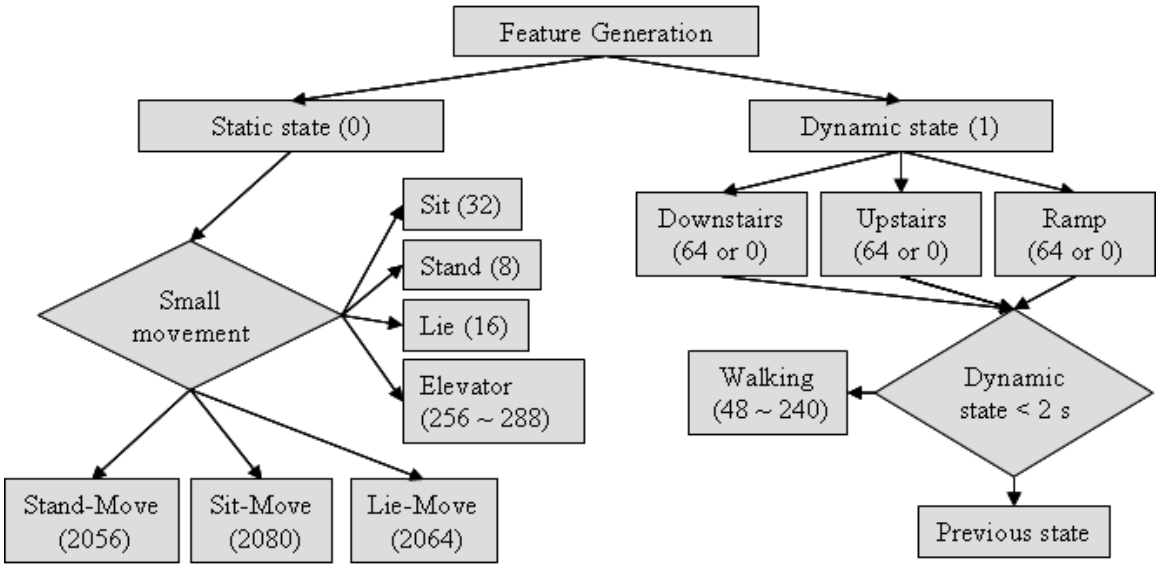


Figure 6.3: Activity classification from accelerometer data.

6.2.5 Test Procedure

A convenience sample of five able-bodied subjects (four males and one female; age: 35.2 ± 8.72 years; height: 174.58 ± 10.75 cm; weight: 66.46 ± 9.7 kg) were recruited from the Ottawa Hospital Rehabilitation Center (TOHRC, Ottawa, Canada). A form that included informed consent, information sheet, and media agreement was obtained for each subject. Subjects who had injuries or gait deficits were excluded.

Data collection took place within TOHRC (hallways, elevator, one bedroom apartment, stairs, Rehabilitation Technology Laboratory) and outside on a paved pathway.

Participants wore the WMMS on their right-front waist, in a regular BlackBerry holster attached to a belt, with the camera pointing forward. No additional instructions were given for positioning the WMMS. The participants were asked to follow a predetermined path and completed a series of mobility tasks. These tasks included standing, walking, sitting, riding an elevator, brushing teeth, combing hair, washing hands, drying hands, setting dishes, filling the kettle with water, toasting bread, a simulated meal at dining table, washing dishes, walking on stairs, lying on a bed, walking on a ramp, and walking outdoors. Each activity in the sequence took approximately 10 to 20 seconds to complete. Each trial had a total of 41 changes of state.

A digital video camcorder was used as the gold standard to record each trial. Three trials were captured for each task sequence. Activity timing was obtained from the camcorder recording for comparison with the WMMS output. Start and end points of each trial were identified by shaking the Smartphone for 2 seconds. The camcorder also provided contextual information for analysis. Finally, data were imported from the SD card into Microsoft Excel for statistical analysis. The detailed procedure is in Appendix E.3.

6.3 Results

The results can be analyzed in terms of CoS and activity classification. For CoS identification at the transition point between activities (Table 6-2), sensitivities for standing, sitting, lying, and taking an elevator were between 97% and 100%. Walking-related CoS, such as stairs and ramps, had 67% to 73% sensitivity. Sensitivity related to

CoS between walking and small movements (i.e., brushing teeth, etc.) were between 40% and 93%. The CoS results for daily living activities were poorer (below 27%) since the continuous series of small movements produced acceleration features that were similar between activities. Detailed results are shown in Table D-2.

Table 6-2: Average true positives (TP), false negatives (FN) and sensitivity (SE) of CoS at the transition between activities. Standard deviations are in brackets.

	Changes-of-State	TP	FN	SE
Static	Stand ↔ Walk	6.8 (1.9)	0 (0.0)	100.00%
	Lie ↔ Walk	2.8 (0.4)	0.2 (0.4)	100.00%
	Elevator ↔ Walk	3 (0.0)	0 (0.0)	98.30%
	Sit ↔ Walk	6.6 (0.9)	0 (0.0)	96.70%
Walking related	Walk ↔ Stairs	5.8 (0.4)	0 (0.0)	73.30%
	Walk ↔ Ramp	5.6 (0.5)	0.2 (0.4)	66.70%
Daily living with walking	Toast bread → Walk	2.4 (0.5)	0.6 (0.5)	93.30%
	Dry hands → Walk	0.4 (0.5)	2.6 (0.5)	83.30%
	Prepare a meal ↔ Walk	0.4 (0.5)	2.6 (0.5)	86.70%
	Wash dishes ↔ Walk	0.8 (0.8)	2.2 (0.8)	66.70%
	Walk → Brush teeth	2.6 (0.5)	0.4 (0.5)	80.00%
	Set Dishes → Move a kettle	1.2 (0.8)	1.8 (0.8)	73.30%
	Move kettle → Toast bread	2.2 (0.8)	0.8 (0.8)	73.30%
	Walk → Set dishes	1.6 (1.5)	1.4 (1.5)	40.00%
Daily living	Wash hands → Dry hands	2.8 (0.4)	0.2 (0.4)	26.70%
	Brush teeth → Comb hair	2.2 (0.4)	0.8 (0.4)	13.30%
	Comb hair → Wash hands	2.8 (0.4)	0.2 (0.4)	13.30%

Table 6-3 shows the specificity results for CoS identification during each activity, using the CoS algorithm (Figure 6.2). The number of false positives was less than 12% for all measures, with half the measures reporting less than 5% false positives. Walking produced the most false positives, identifying a walking-CoS when no CoS occurred, for 324 out of 2700 cases (12%). Large standard deviations were found for some measures due to differences in timing for each activity (i.e., people who walked slower took more time and had more data points for analysis, leading to more true negatives). The detailed result is shown in Table D-3.

Table 6-3: Average false positives (FP), true negatives (TN) and specificity (SP) of CoS during activities. Standard deviations are in brackets.

	Changes-of-State	FP	TN	SP
Static	Lie	0 (0.0)	53.2 (16.5)	100.00%
	Sit	0.2 (0.4)	47.6 (20.1)	99.58%
	Stand	3.8 (3.3)	163.4 (84.5)	97.73%
	Take an elevator	5.4 (3.4)	115.2 (36.8)	95.52%
Walking related	Ramp	1.6 (1.3)	23 (11.5)	93.50%
	Stairs	6.8 (4.0)	63.8 (20.7)	90.37%
	Walk	64.8 (19.3)	475.2 (169.9)	88.00%
Daily Living	Dry hands	0.2 (0.4)	15.2 (2.0)	98.70%
	Set dishes	0.4 (0.5)	12 (6.3)	96.77%
	Wash dishes	1.6 (1.3)	32.8 (9.8)	95.35%
	Comb hair	1.4 (1.1)	28 (6.1)	95.24%
	Toast bread	1 (1.2)	18.2 (8.3)	94.79%
	Wash hands	1.2 (0.8)	19.6 (9.7)	94.23%
	Brush teeth	2.6 (1.1)	29 (7.4)	91.77%
	Prepare a meal	4.6 (1.5)	40 (9.9)	89.69%
	Move a kettle	2.4 (2.1)	18 (5.9)	88.24%

Better activity classification results were achieved when using both acceleration features and video clips for all activities except sitting, as compared to using the accelerometer only (Table 6-4). Confusion matrix table is shown in Table D-4 and Table D-5. The accelerometer-only analysis used the activity classification algorithm in Figure 6.3. Acceleration-only had 4% greater sensitivity than acceleration-and-video for sitting because one CoS was missed, which resulted in no associated video data. Using the accelerometer only, none of the ADL was identified; however, no false positives were reported, resulting in a high specificity. Detailed results are shown in Table D-6 and Table D-7.

Since accelerometer data was unavailable during video capture, no acceleration could be recorded for a minimum of 2.9 seconds after a CoS was identified. For example, if a person sits in a chair, stands up, and sits down again within 4 seconds (3 seconds of video capture and 1-second data window), the CoS will not be detected or identified. However,

digital video would be available during this period for activity classification. The average accelerometer sampling frequency was 7.88 ± 1.39 Hz.

Table 6-4: Sensitivity (SE) and specificity (SP) comparison of activity classification for accelerometer only and accelerometer with video clips results.

	Activity	Acc. Only		Acc. + Video	
		SE (%)	SP (%)	SE (%)	SP (%)
Static	Stand	98	99	100	100
	Lie	96	100	100	100
	Elevator	2	100	100	100
	Sit	96	99	92	100
Walking-related	Walk	95	92	96	93
	Stairs	45	73	86	99
	Ramp	22	98	50	100
Daily Living	Dining activity	0	100	94	100
	Bathroom activity	0	100	84	100
	Kitchen activity	0	100	70	100
	Move a kettle	0	100	37	100
	Wash dishes	0	100	32	100
	Toast bread	0	100	31	100
	Prepare a meal	0	100	21	100
	Dry hands	0	100	15	100
	Set dishes	0	100	7	100
	Wash hands	0	100	5	100
	Brush teeth	0	100	0	100
	Comb hair	0	100	0	100

In various circumstances, in particular when video images were dark, video analysis became difficult. The BlackBerry 9550 included a light sensor, but the sensor output was unavailable with Java API 5.0. Light intensity level could be beneficial for recognizing indoor and outdoor environments.

Inclination angles (Figure 6.4) for lying down ranged from 81 to 115 degrees (average = 99 ± 8.9 degrees), sitting from 136 to 175 degrees (average =157 ± 9.38 degrees), and standing from 179 to 208 degrees (average = 192 ± 7.5 degrees). Since the Smartphone

was worn on the waist, the Smartphone position could occasionally shift due to motion of the belt, pants, or holster.

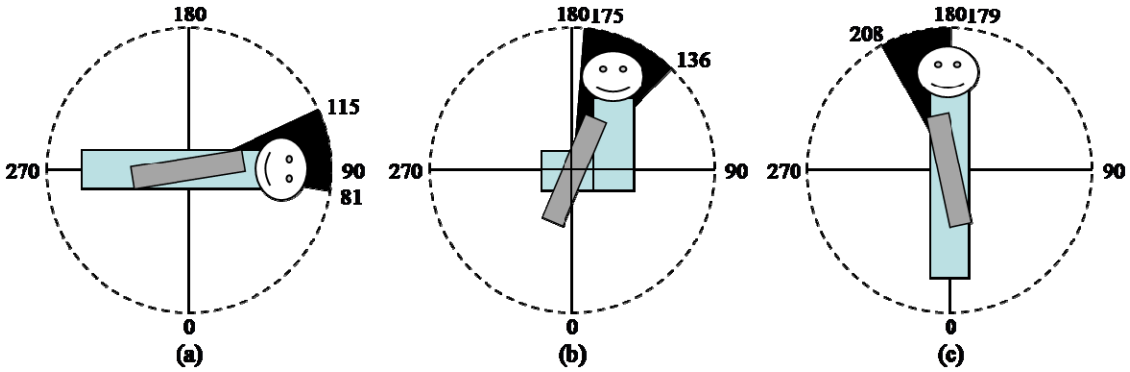


Figure 6.4: Body and BlackBerry (dark box) inclination angles for a) lying, b) sitting, and c) standing position.

As shown Figure 6.5, a combination of SR (SR > 100) and static status (STD-Y < 160) were used to identify small movements. Discrete small movements were easily distinguished; however, a CoS could be missed for a continuous series of small movements. For example, brushing teeth and then combing hair.

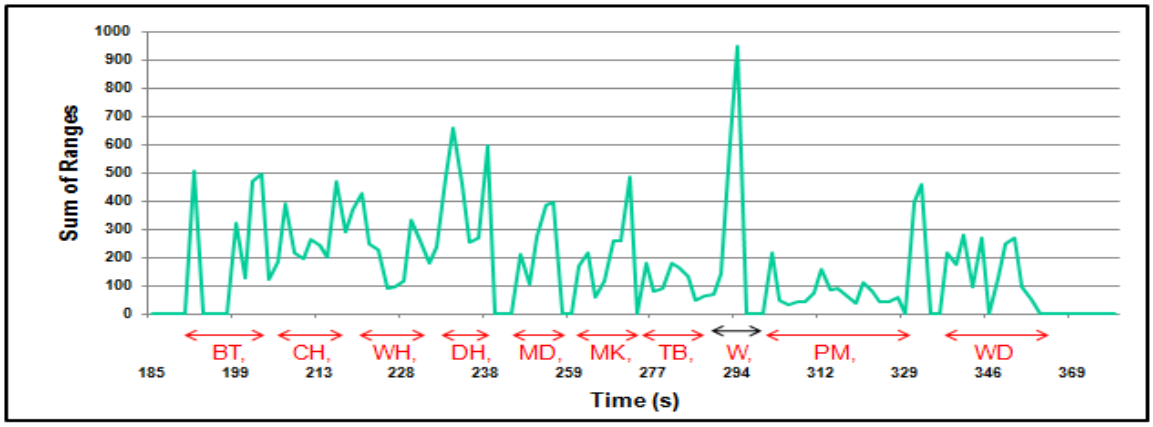


Figure 6.5: Sum of Ranges to distinguish small movements: brushing teeth (BT), combing hair (CH), washing hands (WH), drying hands (DH), moving dishes (MD), moving a kettle (MK), toasting bread (TB), preparing a meal (PM), and washing dishes (WD).

Sum of ranges was more sensitive than STD-Y for distinguishing mobility states (Figure 6.6). Further, the SMA-SR curves were smoother than the sum of ranges curves. The smooth SMA-SR curve was better at defining thresholds to classify walking on stairs and walking on level ground.

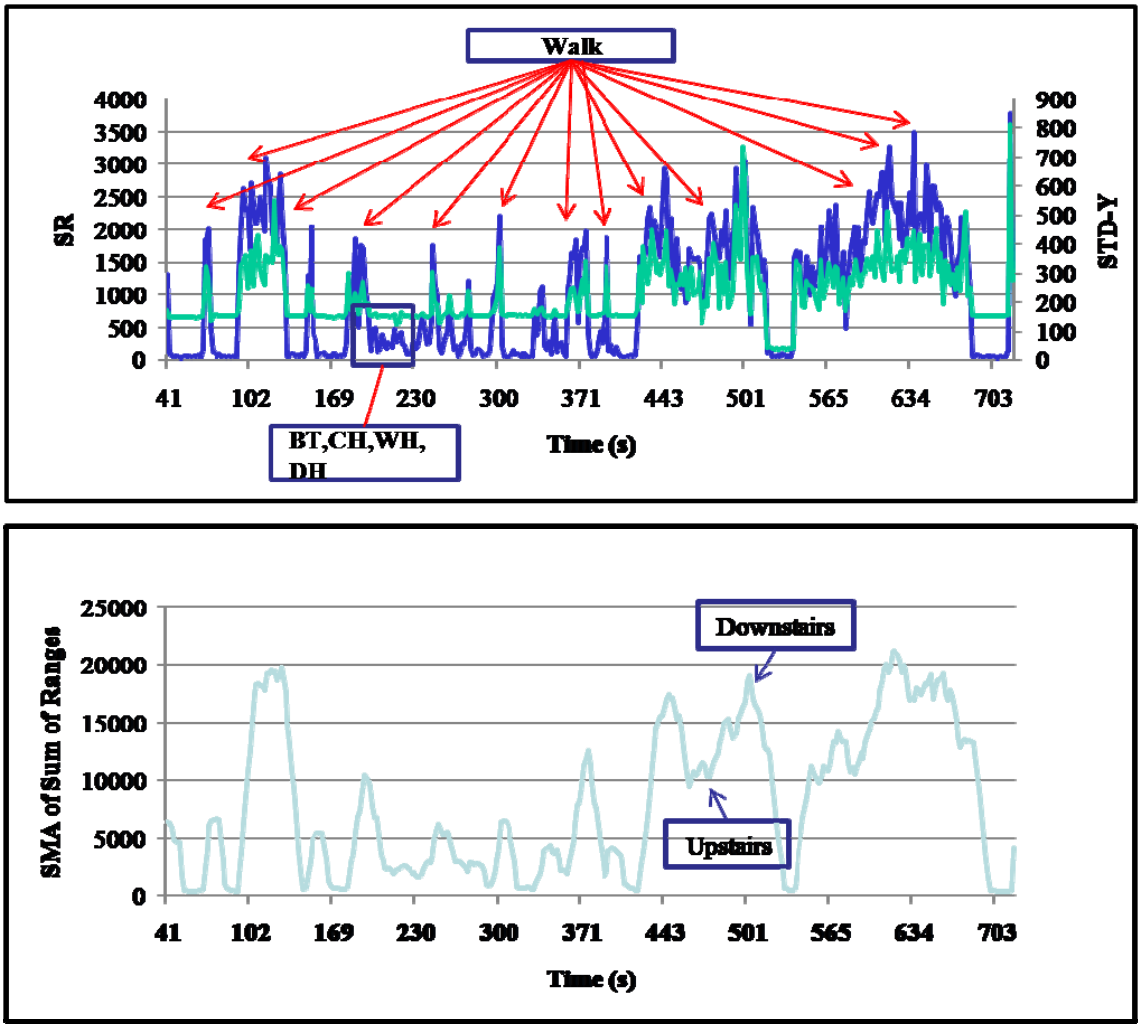


Figure 6.6: Sum of ranges (SR) and STD-Y to distinguish walking and static states (BT =brushing teeth, CH=combing hair, WH=washing hands, and DH=drying hands). SMA-SR identifies walking on stairs and walking on level ground.

Video was a valuable medium for evaluating mobility activities since audio information was available and multiple-images could be used in the analysis (Table 7.5). The sense of

motion also provided useful information for categorizing the activity and context. Still images are also viable for analysis but provide inferior information compared with moving video, as shown in Figure 6.7 for walking.

Table 6-5: Video clip analysis.

Activities	Context	Shaking images (Y/N)	Image direction	Sound content
Stand	Wall, ground	N	Forward	Quiet
Sit	Ceiling, wall	N	Forward, upward	Quiet
Lie	Ceiling	N	Upward	Quiet
Ride an elevator	Elevator	N	Forward	Elevator
Walk on level ground	Ground	Y	Forward	Walking
Climb upstairs	Stairs, hand rail	Y	Forward, upward	Climbing stairs
Climb downstairs	Wall	Y	Forward, downward	Climbing stairs
Up ramp	Wall, ceiling, hand rail	Y	Forward, upward	Walking
Down ramp	Wall, ceiling, hand rail	Y	Forward, downward	Walking
Brush teeth	Washbasin	Y	Forward	Flushing
Comb hair	Washbasin	N	Forward	Quiet
Wash hands	Washbasin	Y	Downward	Flushing
Dry hands	Towel rack, towel	Y	Forward	Quiet
Set dishes	Table	Y	Forward	Dish collisions
Move kettle	Kettle, kitchen background	Y	Forward	Quiet or filling water
Toast bread	Toaster, Table	N	Forward	Quiet
Meal setup	Table	N	Forward & upward	Quiet or filling water
Wash dishes	Sink, dishes	Y	Forward	Washing, flushing, dish collisions

Activity recognition for static and walking states was confirmed by video (Figure 6.7). Occasionally, video showed a moving-hand during walking, as the arm swung during locomotion. Since the phone was located on the right waist, the video usually showed

more right side than left side. For ramp navigation, Smartphone video was not able to show the ramp; however, the video did allow the evaluator to determine that the person was ascending or descending. Similarly, video was unable to show the stairs when descending, but movement and sound of the footfalls on the stairs allowed the evaluator to categorize the stair descent activity.

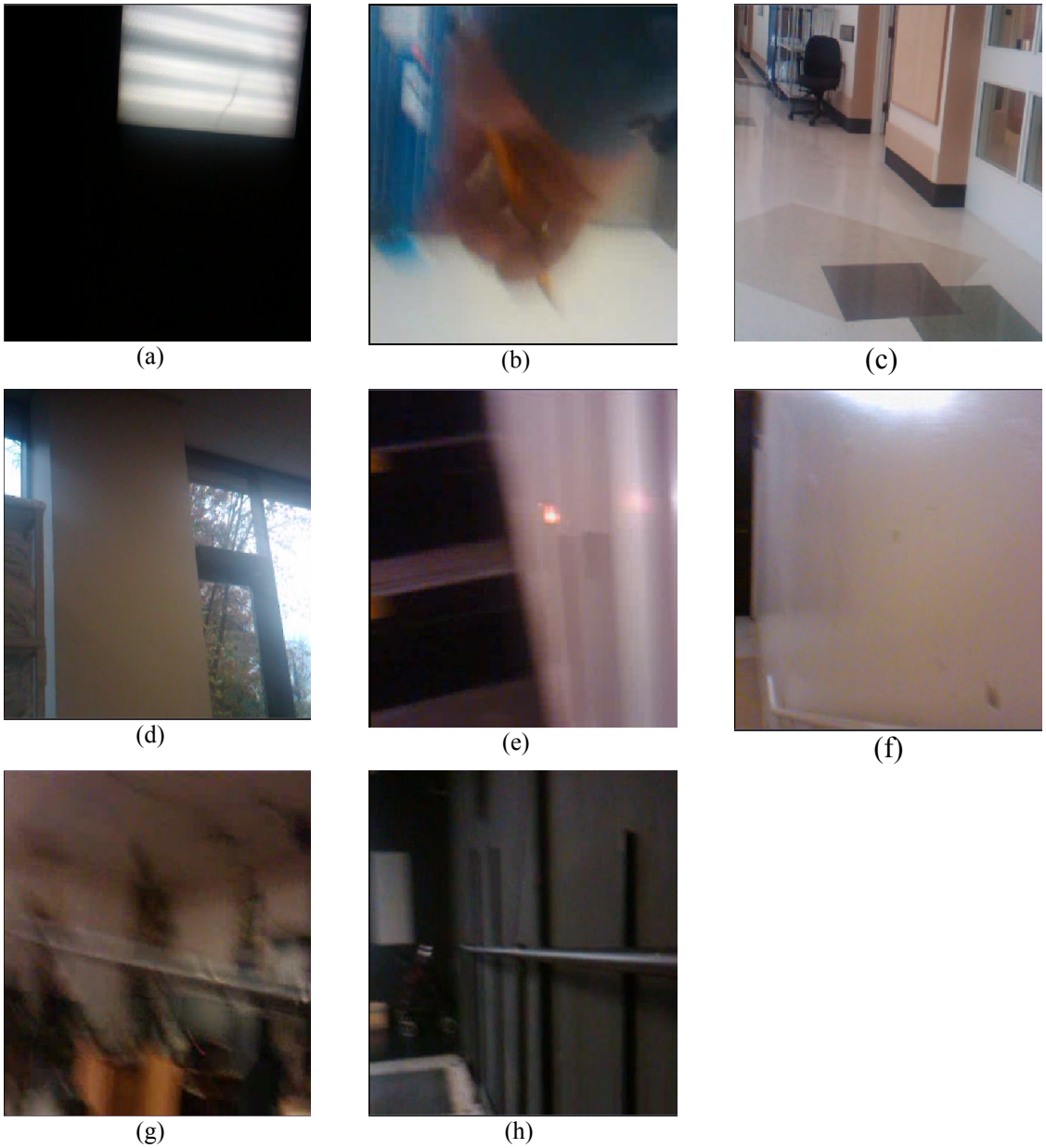
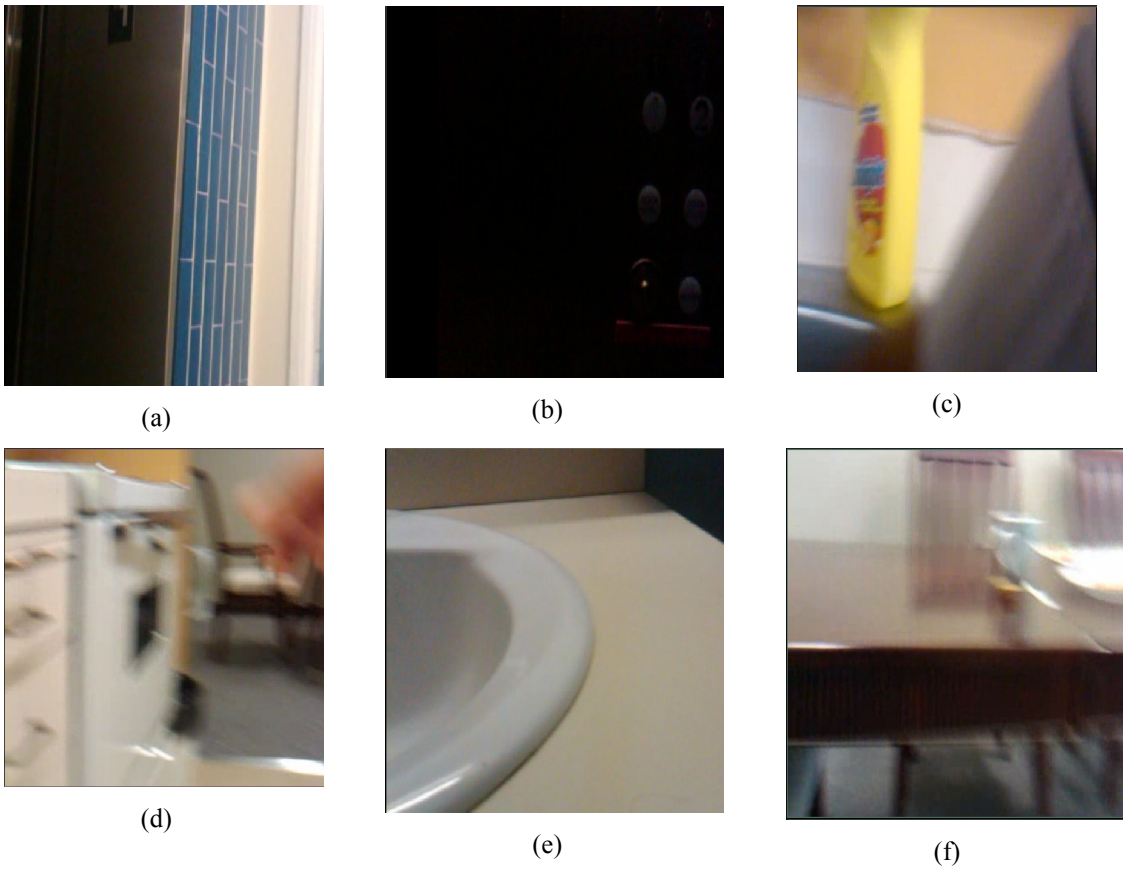


Figure 6.7: Images from BlackBerry video for a) lying, b) walking, c) standing, d) sitting, e) climbing upstairs, f) walking downstairs, g) walking up ramp, h) walking down ramp.

The elevator CoS was usually triggered by a transition from walking to standing, rather than movement of the elevator up and down. The activity can be classified by seeing the elevator door, or the inside of the elevator, in the video clip (Figure 6.8). The inside of elevator was unclear and dark, but the elevator sound was clearly defined. Further, the bathroom, kitchen, and dining room activities were clearly identified from video images.



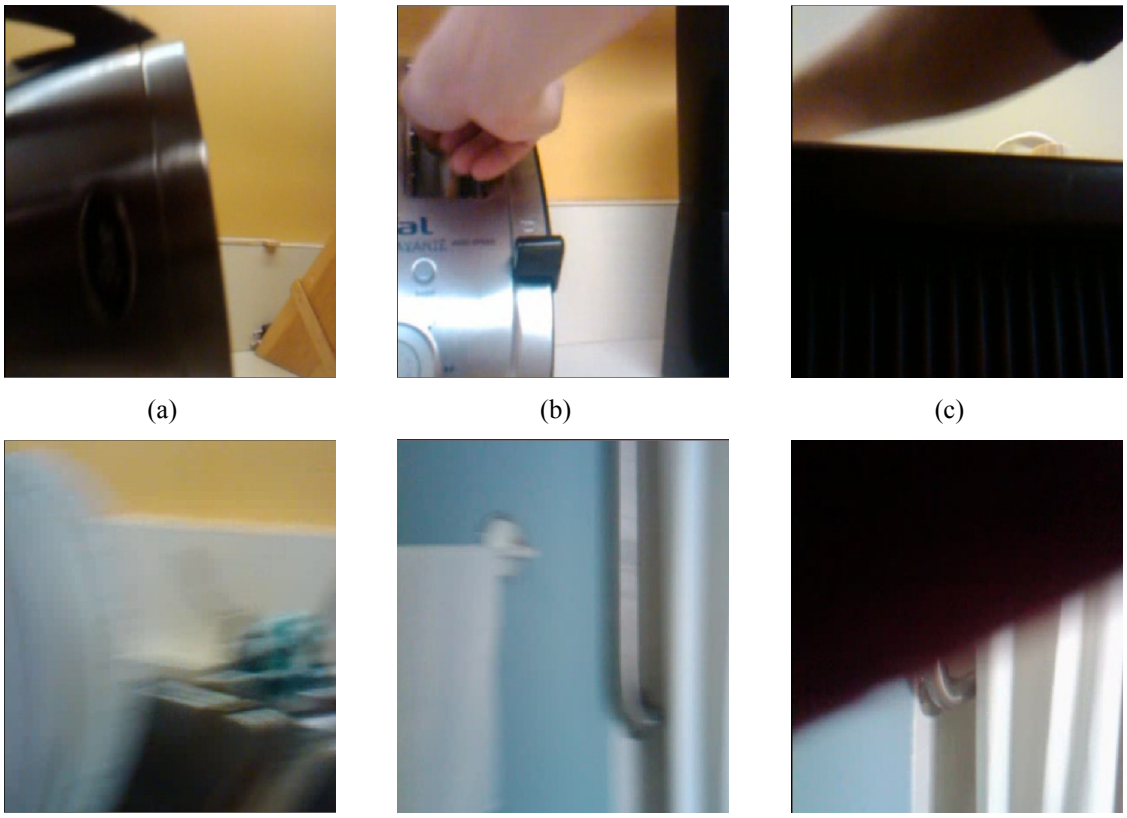
(a) (b) (c)

(d) (e) (f)

Figure 6.8: Images from BlackBerry video for a) elevator door b) the inside of elevator, c) kitchen sink, d) kitchen cabinet and oven, e) bathroom sink, f) dining table and meals.

Small movements, such as brushing teeth, combing hair, and moving plates, could not be identified because these activities were out of the Smartphone camera range. Some clips could clearly identify moving a kettle, toasting bread, meal preparation, and washing dishes (see Figure 6.9). Further, drying hands and meal preparation needed continuous video images to identify the activities (i.e., single images could not identify these activities). Moreover, video from people with shorter lower bodies could not be used for activity classification when a tabletop or kitchen counter obstructed the phone, located at

waist height. For example, moving a kettle and toasting bread were not visible in the video field for these participants.



(a) (b) (c)

(d) (e) (f)

Figure 6.9: Images from BlackBerry video for small movements: a) Moving a kettle, b) toasting bread, c) meal preparation, d) washing dishes, e) towel and towel rack, f) drying hands.

6.4 Discussion

This study demonstrated that a combination of BlackBerry accelerometer signal analysis and cell phone video assessment provides a viable solution for identifying many mobility activities and the context of these activities. Since an available Smartphone was worn in a typical manner, at the waist in a holster, and no external sensors or hardware was required, the WMMS could be easily implemented in the community.

The ability to appropriately identity a CoS is critical for a WMMS that uses video. The WMMS successfully recognized CoS for both static and walking activities. Accurate and

real-time CoS identification triggers video clips at the appropriate time to enable post-processing. By taking the initial activity classification from the sensor data and refining the classification decision using the BlackBerry video, superior activity recognition results were obtained.

Static activity (sitting, standing, and lying) classification accuracy improved from outcomes in the literature (below 94.6%, [66], [10], [2]) to 97.3%. The results would have been 100% except one CoS was missed during a walk-to-sit CoS. More thorough biomechanical analysis is required to identify the movement pattern that can create this outlying error. Standing and lying down were recognized with 100% accuracy.

The WMMS produced better sensitivity for walking as compared with the previous system [2]. Some previous research studies had better walking accuracy (above 96%) [55], [10] but only distinguished differences between very distinct mobility states, such as, running, jumping, and no-movement. Accuracy of these systems may reduce if they were categorizing similar activities, such as running on level ground and running up an incline.

Stair navigation identification improved from 60% [2], [11], in the literature, to 86% with the new WMMS. However, the result did not reach the target (95%). Acceleration patterns between walking on level ground and stairs are similar, and the stair slope does not greatly change accelerations at the waist.

CoS identification for ramp walking was enhanced from 43.3% for the previous system [2] to 66% because the new WMMS used a specific ramp judgment algorithm within the decision tree. Ramp classification also improved from 16.6% [2] to 50%. Accelerometer signals for able-bodied people may not be appropriate for ramp CoS identification; however, accelerometer signals from people with mobility disabilities many be sufficiently distinct to enable ramp CoS triggering. During ramp descent, errors may also occur if a walking CoS is triggered before the person finishes descent since the waist-mounted video would not show the ramp bottom in the video field. An altitude sensor may help to detect the change in body vertical position during ramp and stair navigation, thereby providing more accurate CoS triggering.

Specificity is also important for the WMMS since identifying a CoS when no change occurred (false positive) results in inappropriate video capture that affects storage capacity, battery life, and increases the workload for video post-processing. Most prior research did not report specificity or false positive results [4], [11], [37]. For the two studies that reported false positives, the maximum false positive rate was above 12% for walking-related activities [2], [42]. Improved threshold calibration methods for each individual may help to decrease false positives and false negatives.

The Blackberry device was able to process the features and algorithms in real-time and save outcome data for each 1-second window. Most previous mobility research used cell phones to collect raw data and then analyzed features and algorithms offline [4], [11], [37], [69]. For the new WMMS, threshold settings, decision trees, feature extraction, and timing were executed on the Smartphone in real time. Further research will develop an application that can directly provide mobility results on-screen for a user.

The combination of accelerometer and video camera was superior to using the accelerometer data alone to provide useful mobility information. Bathroom, kitchen, and dining room activities could be recognized by video in 70 % to 94 % of the cases. Although the combination of accelerometer and video could not identify small ADL movements clearly, this method had better accuracy than the accelerometer only, which could not recognize these ADLs. Further, CoS for small movements could not be clearly identified. A higher accelerometer sampling rate may allow for more complex signal analysis that could improve small movement CoS identification. Additional features, such as signal magnitude area, skewness, **Error! Reference source not found.**, energy, correlation [49], and kurtosis [47], might help to recognize the small movement CoS, but these features needs higher sampling rate or accelerometer sensitivity.

From subjective analysis of the image results, video was better than still images for refining activity classification and recognizing context; such as, flooring type, bathroom/kitchen, and outdoors. Still images had limitations in dark areas, such as an elevator [2], but video clips included audio data that helped identify an elevator and other states. Since video had continuous images to distinguish upward and downward

movement, these clips were useful for stairs and ramp navigation. Although video cameras could potentially recognize small ADL movements, the waist-mounted Smartphone location limited the camera to detecting activities that were roughly occurring at waist height.

While BlackBerry video proved useful, the current implementation requires human time to review and classify the activities. While this may be appropriate for research and specific clinical applications, automated video analysis would be required to have widespread use of the video data for activity classification.

6.5 Conclusion

The new BlackBerry-based WMMS successfully used integrated sensors and Smartphone video to recognize a variety of mobility activities. By combining and weighting sum, range, and covariance statistics, the WMMS was able to recognize standing, sitting, lying, riding an elevator, walking on level ground, ramps, stairs, and ADL (washing hands, drying hands, setting dishes, moving a kettle, toasting bread, preparing a meal, and washing dishes). Static activities, walking on level ground, walking on stairs, walking on a ramp, and riding an elevator had higher sensitivities than previous studies, but the overall CoS identification and activity classification performance could be improved by further research.

Adding additional sensors, increasing the accelerometer sampling rate/sensitivity, and adding new threshold calibration methods can be considered in future work. The classification of other small movement ADL activities also requires further research to increase sensitivity and specificity. For this WMMS to be recognized for use with rehabilitation and mobility disabilities, further evaluation research is required with a variety of patient populations.

Chapter 7: Conclusion

Real-time mobility states, including knowledge of the environment, are necessary as a reference for health monitoring, especially in field of rehabilitation. When clinicians obtain accurate mobility information related to non-laboratory environments, such as in the community and home, they might make better clinical decisions for people with mobility deficits. The WMMS developed in this thesis is designed to provide clinicians with this kind of information.

A BlackBerry Smartphone is an efficient and low-cost device for the WMMS application. The phone provides multi-tasking functions and integrated accelerometer, GPS, and video camera. The cell phone computing platform can perform data collection, signal post-processing, and data storage.

The new WMMS has the potential to recognize changes-of-state and activities, as well as the context in which these take place. The combination of acceleration, GPS, and video is effective for mobility monitoring. The features, thresholds, and new algorithms help improve sensitivity results. Video clearly distinguishes all activities, at the cost of requiring human intervention during data post-processing.

Hardware limitations, which have limited the design of some features and algorithms in this version of the WMMS, should be solved with future BlackBerry Smartphone releases. Although static states (i.e. standing, sitting, lying, and taking an elevator), walking on level ground, walking on stairs, and car riding classification have better results than the previous WMMS, sensitivity and specificity still require further study to reach a higher percentage. The Smartphone approach can also be used to recognize small activities of daily living movement (i.e. brushing teeth, combing hair, washing hands, drying hands, setting dishes, moving a kettle, toasting bread, preparing a meal, and washing dishes) and walking on a ramp, however their lower sensitivity should be improved in future work.

7.1 Future Work

Future hardware development might include integrating additional sensor in the WMMS. For example, an altitude sensor might better recognize walking on stairs, walking on ramps, and taking an elevator because the sensor can distinguish changes in elevation over 1-2 m. The current WMMS version had lower accelerometer sensitivity, a lower sampling rate, and a pause in the acceleration recording during video capture. These issues were due to the design of the BlackBerry hardware and operating system and have been discussed with the manufacturer of the BlackBerry, Research In Motion. A 6-gravity sensitivity accelerometer and 50 Hz (ideally 100 Hz) sampling rate would greatly assist the WMMS by allowing for more complex features. Furthermore, continuous, uninterrupted accelerometer data would also help to define activities and changes-of-state.

Signal post processing research could decrease false positives and increase true positives for overall activities. Automated threshold calibration, which would efficiently enhance sensitivity and specificity for each user, requires more study, and automated offset and drift calibration would be developed in a fixed period for long-term monitoring. Further, movements associated with the small activities of daily living also require further study to understand signal features related to these activities. Advanced feature development and algorithms might improve movement classification. Periodic accelerometer calibration may also be required for long-term monitoring.

Automated context or acceleration recognition would decrease time consumption for the mobility analysis. Further research might find more indexes to group outcomes, which can save processing/analysis time for Smartphone devices and increase accuracy for activity recognition. When a specific environment is defined, better feature grouping would only use the appropriate measures for activity and context identification, thereby improving application efficiency.

WMMS system evaluation with people who have a mobility disability is important to ensure that the system performs well with pathological gait. Research participant recruitment requires cooperation with clinicians and rehabilitation consultants. Future work will involve testing the WMMS with people with amputations, hemiplegia, and the elderly.

References

[1]. A. E. Patla and A. Shumway-Cook, "Dimensions of mobility: Defining the complexity and difficulty associated with community mobility," *Journal of Aging and Physical Activity*, vol. 7, pp. 7-19, 1999.

[2]. G. Haché, E. Lemaire, and N. Baddour, "Wearable mobility monitoring using a multimedia Smartphone platform," *IEEE Transactions on Instrumentation and Measurement,* pp. 1-9, 2010.

[3]. C. N. Scanaill, S. Carew, P. Barralon, N. Noury, D. Lyons, and G. M. Lyons, "A review of approaches to mobility telemonitoring of the elderly in their living environment," *Annals of Biomedical Engineering*, vol. 34, pp. 547-563, 2006.

[4]. R. K. Ganti, S. Srinivasan, and A. Gacic, "Multisensor fusion in Smartphones for lifestyle monitoring," in *Proceedings of the International Conference on Body Sensor Networks*, pp. 36–43, 2010.

[5]. S. Jindal, A. Jindal, and N. Gupta, "Grouping WI-MAX, 3G and WI-FI for wireless broadband," in *Proceedings of the First IEEE and IFIP International Conference in Central Asia on Internet*, 2005.

[6]. D. Byrne, A. R. Doherty, C. G. M. Snoek, G. J. F. Jones, and A. F. Smeaton, "Everyday concept detection in visual lifelogs: validation, relationships and trends," *Multimedia Tools Appl,* vol. 49, pp. 119-144, 2010.

[7]. D. Tancharoen, T. Yamasaki, and K. Aizawa, "Practical experience recording and indexing of life log video," in *Proceedings of the 2nd ACM Workshop on Continuous Archival and Retrieval of Personal Experiences,* pp. 66, 2005.

[8]. D. W. Ryoo and C. Bae, "Design of The Wearable Gadgets for Life-Log Services based on UTC," *IEEE Transactions on Consumer Electronics,* vol. 53, pp. 1477-1482, 2007.

[9]. Statistics Canada. (2007). *Participation and Activity Limitation Survey 2006: Tables*. Ottawa: Statistics Canada. Online Available: Accessed: 20 Oct. 2010.

[10]. F. R. Allen, E. Ambikairajah, N. H. Lovell, and B. G. Celler, "Classification of a known sequence of motions and postures from accelerometry data using adapted Gaussian mixture models," *Physiological Measurement,* vol. 27, pp. 935-951, 2006.

[11]. J. R. Kwapisz. G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," in *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*, pp. 10–18, 2010.

[12]. Vicon Motion Systems. Available: http://www.vicon.com Accessed: 06 July 2011.

[13]. Advanced Mechanical Technology Inc. *Model BP400600,* AMTI, Online. Available: http://amti.biz/ Accessed: 06 July 2011.

[14]. Novel. Product Information: System I emed, Novel, Online. Available: http://www.novel.de/productinfo/systems-emed.htm Accessed: 06 July 2011.

[15]. H. Jagos, J. Oberzaucher, M. Reichel, W. L. Zagler, and W. Hlauschek, "A multimodal approach for insole motion measurement and analysis," *Procedia Engineering,* vol. 2, pp. 3103-3108, 2010.

[16]. S. H. Roy, M. S. Cheng, S.-S. Chang, J. Moore, G. De Luca, S. H. Nawab, and C. J. De Luca, "A combined sEMG and accelerometer system for monitoring functional activity in stroke," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol.17, issue 6, pp. 585, 2009.

[17]. J. Mantyjarvi, J. Himberg, and T. Seppanen, "Recognizing human motion with multiple acceleration sensors," in *IEEE International Conference on Systems Man and Cybernetics,* pp. 747-752, 2001.

[18]. S. J. Preece, J. Y. Goulermas, L. P. Kenney, and D. Howard, "A comparison of feature extraction methods for the classification of dynamic activities from accelerometer data," *IEEE Trans. Biomed. Eng.,* vol. 56, pp. 871-879, Mar 2009.

[19]. L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," *Pervasive Computing,* 2004, pp. 1-17.

[20]. K. Zhang, P. Werner, M. Sun, F. X. Pi-Sunyer, and C. N. Boozer, "Measurement of human daily physical activity," *Obesity,* vol. 11, pp. 33-40, 2003.

[21]. M. F. Finneran. (2004, June 1). *WiMax versus wi-fi*. pp. 24. Online Available: http://ocw.upm.es/ingenieria-telematica/redes-y-servicios-de-radio/contenidos/rsrd-OCW/web-05-06/Finneran.pdf Accessed: 30 June 2010.

[22]. R. L. Nolan, "Managing the computer resources: A stage hypothesis", *Communications of the ACM*, vol.16, Number 7, July 1973.

[23]. P. H. Y. Ooi, G. Culjak, and E. Lawrence, "Wireless and wearable overview: Stages of growth theory in medical technology applications," in *Proceedings of the International Conference on Mobile Business*, pp. 536, 2005.

[24]. P. Bonato. (2005, 25 February). "Advances in wearable technology and applications in physical medicine and rehabilitation." *Journal of NeuroEngineering and Rehabilitation* 2005(2:2), Online Available: http://www.jneuroengrehab.com/content/2/1/2 Accessed: 27 Nov 2010.

[25]. C. Competency, N. Genomics, and P. F. Acids, "A wearable electronic system for objective dietary assessment," 2009.

[26]. Statistics Canada. (2006, April, 5). Residential telephone service survey. *Residential Telephone Service Survey*. Pp. 1. Online Available: http://www.statcan.gc.ca/daily-quotidien/060405/dq060405b-eng.htm Accessed: 30 June, 2010.

[27]. Statistics Canada, "Microdata User Guide," *Residential Telephone Service Survey,* vol. 2010, pp. 61, May 2004.

[28]. Statistics Canada. (2007, May, 4). Residential telephone service survey. *Residential Telephone Service Survey*. Pp. 1. Online Available: http://www.statcan.gc.ca/daily-quotidien/070504/dq070504a-eng.htm Accessed: 30 June 2010.

[29]. Statistics Canada. (2008, April, 23). Residential telephone service survey. *Residential Telephone Service Survey*. Pp. 1. Online Available: http://www.statcan.gc.ca/daily-quotidien/080423/dq080423d-eng.htm Accessed: 30 June 2010.

[30]. Statistics Canada. (2008, December). Residential telephone service survey. *Residential Telephone Service Survey*. Pp. 1. Online Available: http://www.statcan.gc.ca/daily-quotidien/090615/dq090615c-eng.htm Accessed: 30 June 2010.

[31]. H. Ketabdar and T. Polzehl, "Fall and emergency detection with mobile phones," in *Proceeding of the Eleventh International ACM SIGACCESS Conference on Computers and Accessibility*, pp. 241-242, 2009.

[32]. International Telecommunication Union. *The ICT development index*. Measuring the Information Society. pp. 13-18. Online Available: http://www.itu.int/ITU-D/ict/publications/idi/2009/index.html Accessed 16 March 2009.

[33]. T. Brezmes, J. L. Gorricho, and J. Cotrina, "Activity recognition from accelerometer data on a mobile phone," *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pp. 796-799, 2009.

[34]. Z. He and L. Jin, "Activity recognition from acceleration data based on discrete consine transform and SVM," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 5041-5044, 2009.

[35]. N. Wang, E. Ambikairajah, S. J. Redmond, B. G. Celler, and N. H. Lovell, "Classification of walking patterns on inclined surfaces from accelerometry data," in *Proceedings of the IEEE Digital Signal Processing 16th International Conference on Digital Object Identifier*, pp. 1-4, 2009.

[36]. A. M. Khan, Y.-K. Lee, S. Y. Lee, and T.-S. Kim, "A triaxial accelerometer-based physical activity recognition via augmented signal features and a hierarchical recognizer," *IEEE Transactions on Information Technology in Biomedicine,* vol. 14, pp. 1166-1172, 2010.

[37]. Z. Shumei, P. McCullagh, C. Nugent, and Z. Huiru, "Activity monitoring using a smart phone's accelerometer with hierarchical classification," in *Proceedings of the Intelligent Environments (IE) Sixth International Conference*, pp. 158, 2010.

[38]. D. M. Karantonis, M. R. Narayanan, M. Mathie, N. H. Lovell, and 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.

[39]. C.V.C. Bouten, K.T.M. Koekkoek, M. Verduin, R. Kodde, and J.D. Janssen, "A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity," *IEEE Transactions on Biomedical Engineering*, vol. 44, pp. 136-147, 1997.

[40]. M. J. Mathie, "Monitoring and interpreting human movement patterns using a triaxial accelerometer," *PhD Doctorate Thesis, University of New South Wales, School of EE and Telecom,* 2003.

[41]. M. J. Mathie, A. C. F. Coster, N. H. Lovell, B. G. Celler, S. R. Lord and A. Tiedemann, "A pilot study of long-term monitoring of human movements in the home using accelerometry," *J. Telemed. Telecare,* vol. 10, pp. 144-151, 2004.

[42]. M. J. Mathie, A. C. F. Coster, N. H. Lovell, and B. G. Celler, "Detection of daily physical activities using a triaxial accelerometer," *Medical and Biological Engineering and Computing*, vol 41, no. 3, pp. 296-301, 2003.

[43]. M. Sekine, T. Tamura, M. Akay, T. Fujimoto, T. Togawa and Y. Fukui, "Discrimination of walking patterns using wavelet-based fractal analysis," *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 10, pp. 188-196, 2002.

[44]. M. Sekine, T. Tamura, T. Togawa and Y. Fukui, "Classification of waist-acceleration signals in a continuous walking record," *Medical Engineering and Physics,* vol. 22, pp. 285-291, 2000.

[45]. N. J. Mansfield, "Human response to vibration," in Anonymous CRC, pp. 1-11, 2005.

[46]. N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N. C. Yen, C. C. Tung, and H. H. Liu, "The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis," in *Proceedings of the Mathematical, Physical and Engineering Sciences,* vol. 454, pp. 903-995, 1998.

[47]. J. Baek, G. Lee, W. Park, and B. J. Yun, "Accelerometer signal processing for user activity detection," in *Proceedings of the Knowledge-Based Intelligent Information and Engineering Systems,* pp. 610-617, 2004.

[48]. D. Joanes and C. Gill, "Comparing measures of sample Skewness and kurtosis," *Journal of the Royal Statistical Society: Series D (the Statistician),* vol. 47, pp. 183-189, 1998.

[49]. N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *Proceedings of the National Conference on Artificial Intelligence,* pp. 1541, 2005.

[50]. K. G. Jöreskog, "Structural analysis of covariance and correlation matrices," *Psychometrika,* vol. 43, pp. 443-477, 1978.

[51]. N. Lovell, N. Wang, E. Ambikairajah, and B. G. Celler, "Accelerometry based classification of walking patterns using time-frequency analysis," in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* pp. 4899-4902., 2007.

[52]. S. H. Liu and Y. J. Chang, "Using accelerometers for physical actions recognition by a neural fuzzy network," *Telemedicine and e-health,* vol. 15, pp. 867-876, 2009.

[53]. F. R. Allen, E. Ambikairajah, N. H. Lovell, and B. G. Celler, "An adapted Gaussian mixture model approach to accelerometry-based movement classification using time-domain features," in *Proceedings of the Engineering in Medicine and Biology Society Conference*, IEEE, 2006.

[54]. STMicroelectronics, *MEMS Inertial Sensor - High Performance 3-Axis ±2/±6g Ultracompact Linear Accelerometer*, LIS344ALH Datasheet, Rev. 3. Geneva, Switzerland: STMicroelectronics, 2008.

[55]. BlackBerry Storm Series Support. (2010). BlackBerry official site. Online Available: http://us.blackberry.com/support/devices/blackberrystorm.jsp Accessed: 06 July. 2010.

[56]. K. Sato, S. L. Smith, and W. A. Sands, "Validation of an accelerometer for measuring sport performance," *Journal of Strength and Conditioning Research*, vol. 23, pp. 341-347, 2009.

[57]. Eclipse.org. (2010). Eclipse official site. Online Available: http://www.eclipse.org/ Accessed: 27 Oct. 2010.

[58]. H. H. Wu, E. D. Lemaire, and N. Baddour, “Using the BlackBerry to Assess Mobility for People with Disabilities.” presented at *RIM Research Day*. Waterloo, Canada, December 2010.

[59]. J.-A. Howe, E. L. Inness, A. Venturini, J. I. Williams, and M. C. Verrier, "The Community balance and mobility scale - A balance measure for individuals with traumatic brain injury," *Clinical Rehabilitation*, vol. 20, pp. 885-895, 2006.

[60]. L. Blum and N. Korner-Bitensky, "Usefulness of the Berg Balance Scale in stroke rehabilitation: A systematic review," *Physical Therapy*, vol. 88, pp. 559-566, 2008.

[61]. L. Seaby and G. Torrance “Reliability of a physiotherapy functional assessment used in a rehabilitation setting,” *Physiotherapy Canada*, vol. 41, pp.264-271, 1989.

[62]. T. Herman, N. Inbar-Borovsky, M. Brozgol, N. Giladi, and J. M. Hausdorff, "The dynamic gait index in healthy older adults: The role of stair climbing, fear of falling and gender," *Gait and Posture*, vol. 29, pp. 237-241, 2009.

[63]. M. Ermes, J. Pärkkä, J. Mäntyjärvi and I. Korhonen, “Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions,” *IEEE Trans Inf Technol Biomed*, vol. 12, pp. 20-26, 2008.

[64]. Tekscan Inc. F-Scan Lite VersaTek System, Clinical and Research Solutions, Available: http://www.tekscan.com/medical/system-fscan-lite1.html Accessed: 06 July 2011.

[65]. Apple-iPhone4, Apple inc, Online Available: http://www.apple.com/iphone/ Accessed: 06 July 2011.

[66]. BlackBerry Smartphone, Research In Motion, Online. Available: http://us.blackberry.com/Smartphones/Accessed: 06 July 2011.

[67]. BlackBerry JDE 5.0.0 API reference, Research In Motion Limited. Online Available: http://www.blackberry.com/developers/docs/5.0.0api/index.html Accessed: 06 July. 2011.

[68]. H. H. Wu, E. D. Lemaire, and N. Baddour, “Using the BlackBerry to assess mobility for rehabilitation,” in *Proceedings of the Medical Informatics on Canadian Medical and Biological Engineering Society (CMBEC) Conference*, 2011.

[69]. J. Yang, “Toward physical activity diary: Motion recognition using simple acceleration features with mobile phones,” in *Proceedings of the ACM International Conference on Multimedia,* pp. 1-9, 2009.

[70]. BlackBerry Signing Authority Tool for Software Download, Research In Motion. (2010). Online Available: https://www.blackberry.com/Downloads/entry.do?code=D82118376DF344B0010F53909B961DB3 Accessed: 27 Oct. 2010.

Appendix A: Software Development and Guidelines

a. Procedures to Pre-edit Eclipse Program

Before editing the WMMS program, the latest version of Eclipse SDK (3.5.2) was installed. First, a new BlackBerry project was created, and then template programs were imported from the folders of BlackBerry JDE, see Figure A.1.

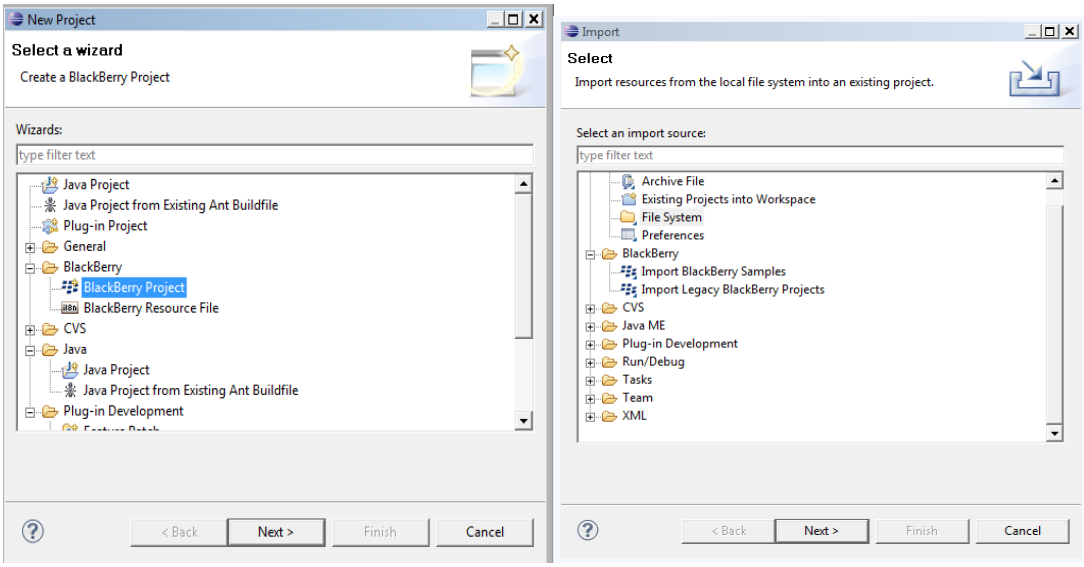


Figure A.1: How to create a BlackBerry project in Eclipse

The BlackBerry Java Plug-in is included the BlackBerry Java SDK 5.0. The 5.0, which has video codes, is compatible with the BlackBerry Storm 2 (9550). In addition, "Add Libraries" imports the required jar files, and then BlackBerry JRE 5.0.0. has to be selected for every BlackBerry Application, Figure A.2.

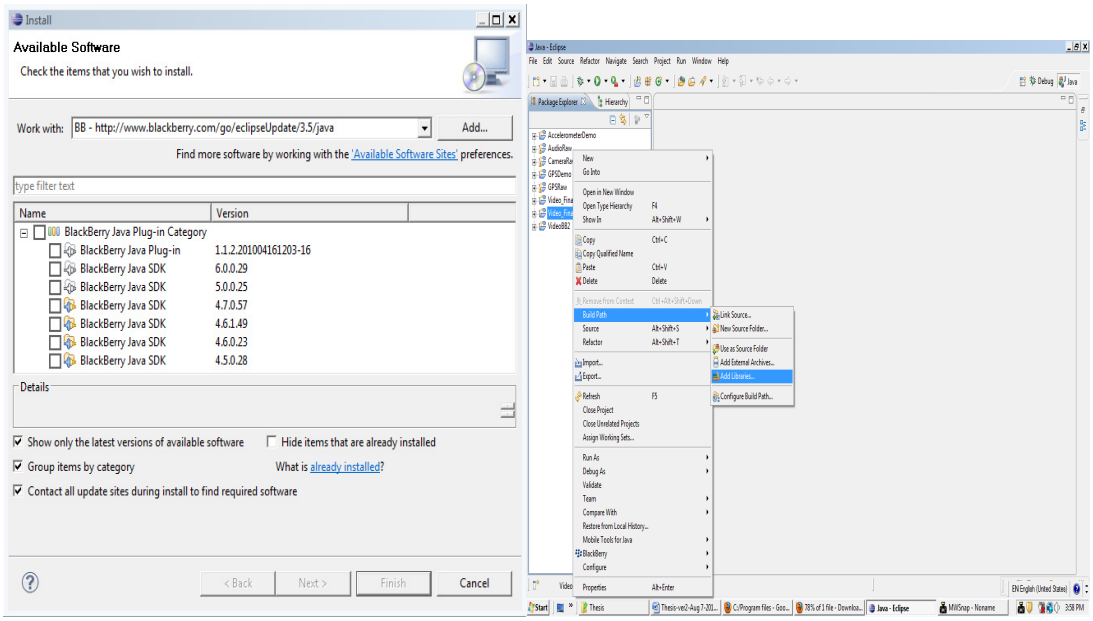


Figure A.2: How to install Eclipse plug in BlackBerry Java SDK and its library

b. Procedures to Transfer Programs for Requirements of a BlackBerry Smartphone

Under Package Explorer, right-click a project is for the selection sheet. After choosing debug as BlackBerry device, the jar, jad, debug, cod, alx, cso, and rapc would be produced automatically with some error dialogs in Figure A.3. The jad and jar files were MIDlet. The alx file told the loader where the application file was located on PC. The code file had to be signed by a signature tool.

c. Procedures to Get the Security Application Program Interface (API)

The application needs to be signed using a Personal Identification Number (PIN). The signature tool is provided by RIM and requires the payment of an administration fee of $20.00 USD. First, the code signing registration form has to be downloaded from the BlackBerry website [70], and then the form has to be completed. After payment, RIM sends the .csl files by email. After getting the .csi files, the signature tool, which is in the BlackBerry JDE folder, can sign the code files. Furthermore, when the code files are clicked, the signature tool pops out. The Request item has to be clicked to get signed status in Figure A.4, and the Internet has to be available.

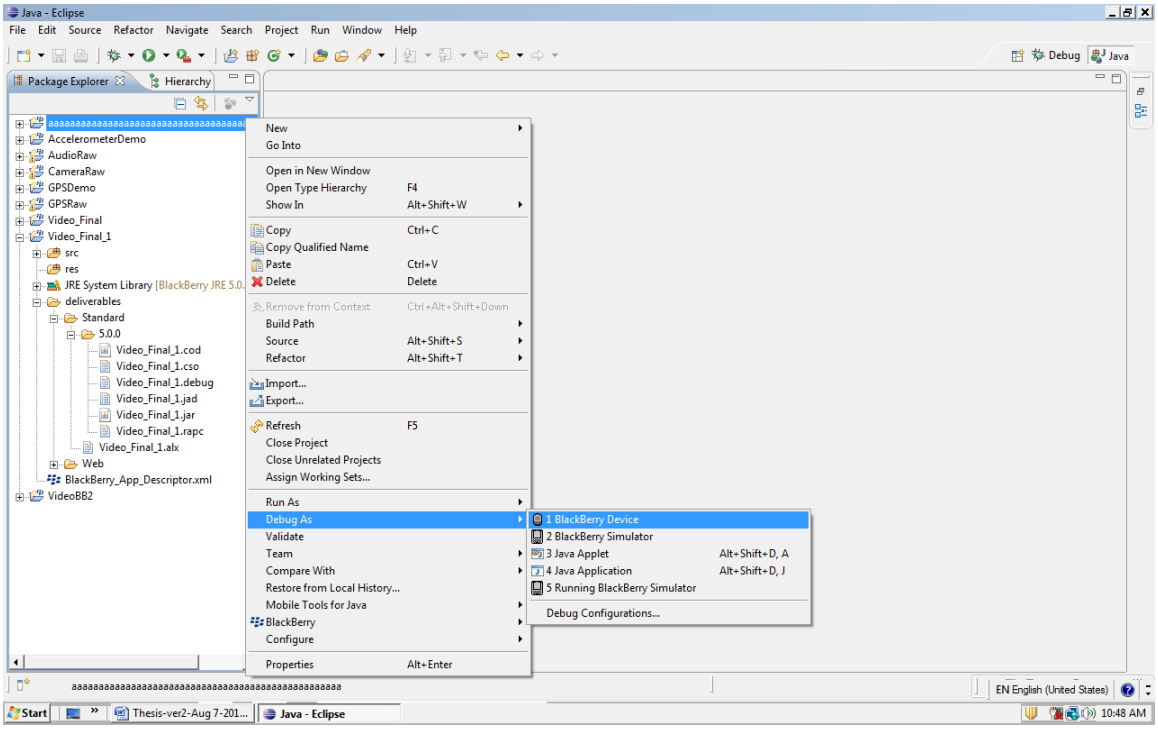


Figure A.3: How to create .alx file of BlackBerry application

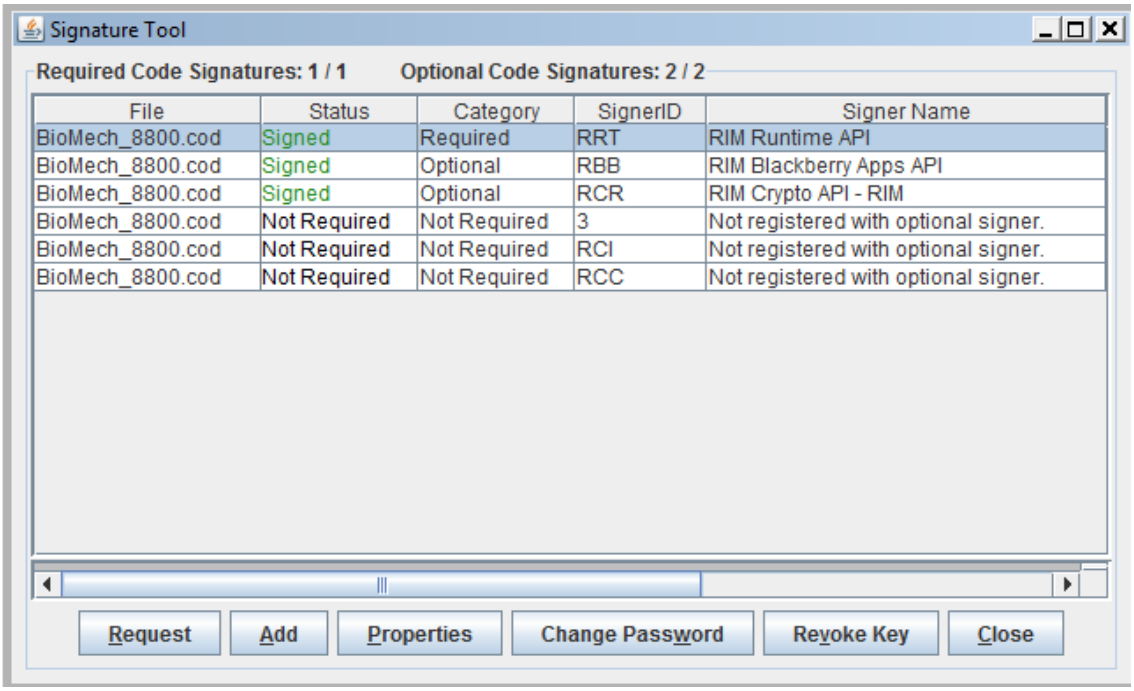


Figure A.4: The interface of the signature tool

d. Procedures to Export Programs from the PC to a BlackBerry Smartphone

Before installing the programs to the phone, BlackBerry Desktop Software 6.0.0.40 was installed. Backing up was the first step, whenever the software program was edited or changed. Next, the phone had to be connected to the PC by via USB. Subsequently, the desktop software could load applications, and the programmer could import files (.alx files) to the BlackBerry.

e. Procedure to Analyze Raw Data by Excel

After executing and downloading raw data from the Smartphone's SD card, raw data needed to be converted, such as .3gp files. The Blackberry Video Creator, (1.13.0.9) converts .3gp to .avi files, and can be downloaded from the website, http://www.seabyrdtech.com/bbvideo. The limit of Excel is 1,048,576 rows and 16,384 columns for importing raw data, and the data limitation is 320,000 to draw a chart. After opening Microsoft Office Excel 2007, clicking Office Button opens files to import txt raw data, Figure A.5. Drift, threshold, and range are analyzed with a graph. Before any features were implemented into the phone, Excel was used to decide suitable features and algorithms. After obtaining reasonable results, the features and algorithm were programmed in Java for the BlackBerry.

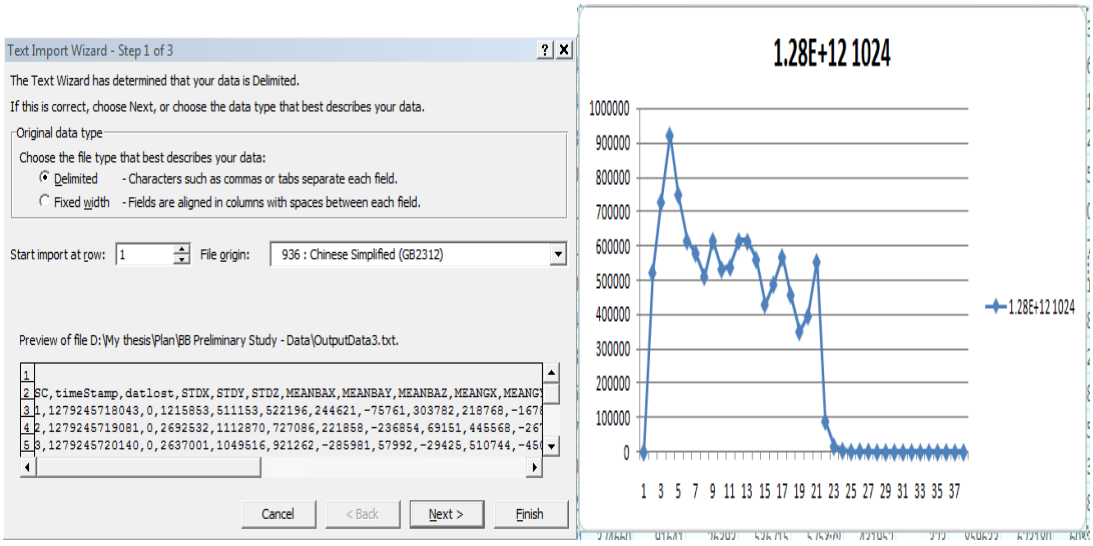


Figure A.5: Procedure to analyze raw data by Excel 2007

f. Procedure to analyze raw data by MATLAB

After opening MATLAB 7.0, clicking File imports txt data into the workspace. In the workspace, the plot function can be used to analyze threshold, range, and drift in several charts, Figure A.6. In the beginning of the software design phase, MATLAB was used to analyze features.

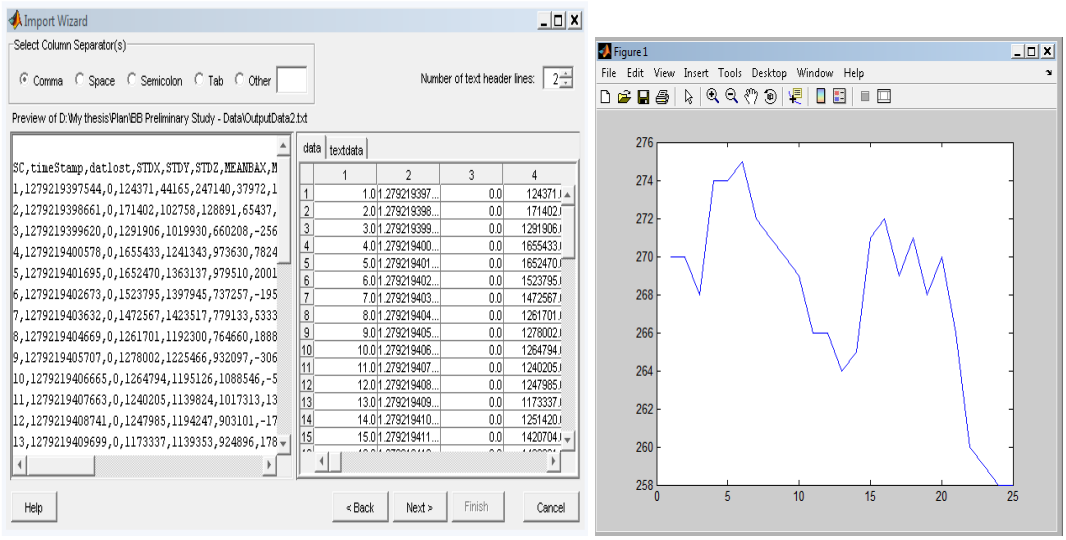


Figure A.6: Procedure to analyze raw data by MATLAB 7.0

Appendix B: JavaScript of Sensors Data

B.1 Script of MobilityContext

```
public class App extends UiApplication {
  public static void main(String args) {
    App app = new App();
    app.enterEventDispatcher();
  }
  private App() {
    pushScreen(new NextProgram());
  }
}
```

AppScreen was to retrieve UiApplication. The main structure was at below.

```
UiApplication.getUiApplication().pushScreen(new SPPScreen());
```

SPPScreen was to set configuration.

```
public class SPPScreen extends MainScreen{
   private RichTextField _rtf;
   private Sensors extSensors;
private void showAllConfigField(){
            add(_cbLog);
          add(_cbGPS);
    }
private void deleteAllConfigField(){
                if (_cbLog.isVisible())
                {
                        delete(_cbLog);
                      delete(_cbGPS);
                }
         }
private void updateConfiguration(){
      String tempstr=_beDelay.getText();
      byte tempByte=Byte.parseByte(tempstr);
      config.setSamplingDelay(tempByte);
      config.setFlagGPS(_cbGPS.getChecked());
      config.setFlagLog(_cbLog.getChecked());
      config.setFlagFullProcessing(_cbFullProcessing.getChecked());
      tempstr=_beLPFfreq.getText();
      float tempfloat=Float.parseFloat(tempstr);
   config.setLPFfreq(tempfloat);
    }
}
```

B.2 Detail Code of Accelerations Collection

```
Public class AccelerometerDemo extends MainScreen{
 private RawThread _thread;
 private Channel _accChannel;
 private AccelerometerData accData;
      private static final int DEFAULT_ORIENTATION =
      Display.DIRECTION_NORTH;
 public AccelerometerDemo(){
   Ui.getUiEngineInstance().setAcceptableDirections( DEFAULT_ORIENTATION
   );
   }
 private class RawThread extends Thread{
  public void run(){
   AccelerometerChannelConfig channelConfig = new
   AccelerometerChannelConfig( AccelerometerChannelConfig.TYPE_RAW );
   channelConfig.setSamplesCount(1);
   _accChannel = AccelerometerSensor.openRawDataChannel(
   AccelerometerDemo.this );
   while( _running ){
      accData = _accChannel.getAccelerometerData();
    short xAccel = accData.getLastXAcceleration();
       logfile.writeStringToLog(xAccel);
   }//end of while
  }//end of run
 }//end of thread
 private MenuItem _startMenuItem = new MenuItem("StartMeasurement" , 0, 0){
  public void run(){
     _thread = new RawThread();
     _thread.start();
  }
 };
 private MenuItem _stopMenuItem = new MenuItem("StopMeasurement" , 0, 0){
  public void run(){
     keepWritingOutput=false;
     _thread._running = false;
  }
 };
   }
```

B.3 Detail Code of GPS Data Collection

```
Private class RawThread extends Thread{
      private boolean currentLocation() {
                                    boolean retval = true;
                          try {
                          LocationProvider lp = LocationProvider.getInstance(null);
                          if (lp != null) {
                          lp.setLocationListener(new LocationListenerImpl(), interval, 1, 1);
                                    } else {
                          retval = false;
                                    }
                          } catch (LocationException e) {
                          System.out.println("Error: " + e.toString());
                          }
                    return retval;
            }//end of location
}//end of thread
public class LocationListenerImpl implements LocationListener {
      private double latitude;
      private double longitude;
      private String satCountStr;
      private double heading;
      private double altitude;
      private double speed;
      public void locationUpdated(LocationProvider provider, Location location) {
       if (location.isValid()) {
       heading = location.getCourse();
       longitude = Location.getQualifiedCoordinates().getLongitude();
       latitude = location.getQualifiedCoordinates().getLatitude();
       altitude = location.getQualifiedCoordinates().getAltitude();
       speed = location.getSpeed();
       logfile.writeStringToLog(appStartTime1);
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(longitude);
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(latitude);
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(altitude);
       logfile.writeStringToLog(",");
       logfile.writeStringToLog(heading);
       logfile.writeStringToLog(",");
```

```
        logfile.writeStringToLog(speed);
        logfile.writeStringToLog("\n");
      }
  }
    public void providerStateChanged(LocationProvider provider, int newState) {
    }
    public double getHeading() {
      return heading;
    }
    public double getAltitude() {
      return altitude;
    }
    public String getSatCount() {
      return satCountStr;
    }
    public double getLatitude() {
      return latitude;
    }
    public double getLongitude() {
      return longitude;
    }
    public double getSpeed() {
      return speed;
    }
}
```

B.4 Detail Code of Video Recording

```
Public class VideoRecordingApplication extends MainScreen{
  private static final String STREAM_VIDEO_FILE =
  "file:///SDCard/BlackBerry/videos/bb2_final_1.3gp";
  String _encoding = "encoding=video/3gpp&mode=standard";
  private static Player _player;
  private static VideoControl _vc;
  private RecordControl _rc;
  private static OutputStream _out;
  private static FileConnection _fc;
  public VideoTime videoThread=new VideoTime();
  public VideoRecordingApplication(){
    _player = javax.microedition.media.Manager.createPlayer( "capture://video?" +
    _encoding );
    _player.start();
    _vc = (VideoControl)_player.getControl( "VideoControl" );
     _rc = (RecordControl)_player.getControl( "RecordControl" );
  }
  addMenuItem( new MenuItem( "Start Record", 0, 0 ) {
```

```
        public void run() {
   startThread();
        }
}
addMenuItem( new MenuItem( “Commit Record”, 0, 0 ){
 public void run(){
  commit();
 }
 public void startThread(){
     videoThread.start();
 }
 public class VideoTime extends Thread {
     boolean takingVideo;
   public VideoTime(){
   takingVideo=true;
   }
 public void run(){
             while (takingVideo){
                     startRecord();
             }
        public void stopTakingVideo(){
             takingVideo=false;
   }
}
  private void startRecord(){
    _fc = (FileConnection)Connector.open( STREAM_VIDEO_FILE );
    if( !_fc.exists() ) {
    _fc.create();
    }
    _fc.truncate( 0 );
    _out = _fc.openOutputStream();
    _rc.setRecordStream( _out );
  }
     _rc.startRecord();
}
 private void commit(){
          videoThread.stopTakingVideo();
    _rc.commit();
    _out.close();
    _fc.close();
}
   }
```

B.5 Detail Code of Safety Setting

```
BattTemp = DeviceInfo.getBatteryTemperature();
if (BattTemp > 38){
  keepWritingOutput=false;
  _thread._running = false;
  _thread.notifyAll();
  _thread = null;
  logfile.writeStringToLog("emergency stop because of temp\n");
  _screen.closeVideo();
}
```

B.6 Detail Code of Excel program

"AI" represents cell, and "AI" represents thresholds. 32 (Sitting), 16 (Lying), and 8 (Standing) represent states.

```
=IF(AND(AI8>$AI$4,AI8<$AI$5),
32,(IF(AND(AI8>$AI$2,AI8<$AI$3,AL8<$AK$5),16,8)))
```

Sample code of Eclipse is shown at below. "&&" means and, and "||" means or.

```
//Lie, Stand, or Sit**********************************
if (TiltAngZY > Thr_st_l && TiltAngZY < Thr_st_h){
    ststate=32; //Sitting state
}
else if (TiltAngZY > Thr_ly_l && TiltAngZY < Thr_ly_h){
    ststate=16; //Lying state
}
else {
    ststate=8; //Standing state
}
```

B.6 Sample Code for Continuous Data Window

```
DiffSR=(short)(SumofRange-prevSR);
SMASR=(short)(SumofRange+prevSR+prevSR2+prevSR3+prevSR4+prevSR5+p
      revSR6+prevSR7);
    prevSR7=prevSR6;
    prevSR6=prevSR5;
    prevSR5=prevSR4;
    prevSR4=prevSR3;
    prevSR3=prevSR2;
    prevSR2=prevSR;
  prevSR=SumofRange;
```

Appendix C: WMMS Program Code

Final Eclipse program for BlackBerry 9550 with operating system 5.0. The sequence of following scripts depends on the timing of trigger (Section 4.2.2.3).

```
package com.rim.samples.device.accelerometerdemo;

/*
* MobilityContext.java
*
* Copyright 1998-2008 Research In Motion Ltd. All Rights Reserved
*
*/
import net.rim.device.api.ui.UiApplication;
public class MobilityContext extends UiApplication {
  public static void main (String args) {
    MobilityContext app = new MobilityContext();
    app.enterEventDispatcher();
 }
  private MobilityContext() {
    pushScreen (new AppScreen());
    }
}

package com.rim.samples.device.accelerometerdemo;
/*
* AppScreen.java
*
* Copyright 1998-2008 Research In Motion Ltd. All Rights Reserved
*/
import net.rim.device.api.system.AccelerometerSensor;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.MenuItem;
import net.rim.device.api.ui.Ui;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.container.MainScreen;
final class AppScreen extends MainScreen {
private static final int DEFAULT_ORIENTATION = Display.DIRECTION_NORTH;
  public AppScreen() {
```

```
    setTitle("Mobility-Context Application");
    if( AccelerometerSensor.isSupported() ){
      // Initialize UI
      // Prevent UI from rotating our screen.
      addMenuItem(_turnon_multisensors);

      Ui.getUiEngineInstance().setAcceptableDirections( DEFAULT_ORIENTATION );
    }
    else{
      UiApplication.getUiApplication().invokeLater(new Runnable(){
        public void run(){
      Dialog.alert("This device does not support accelerometer.");
          System.exit(0);
        }
      });
    }
  }
/*
* AppScreen.java
*
* © Edward Lemaire, The Ottawa Hospital Rehabilitation Centre, 2011
* Confidential and proprietary.
*/
  ////////////////////////////////////////////////////////
  //        Custom Item             //
  ////////////////////////////////////////////////////////
  private MenuItem _turnon_multisensors = new MenuItem("Turn On Multi_Sensors",
30, 30) {
    public void run() {
      UiApplication.getUiApplication().pushScreen(new SPPScreen());
      close(); //close the current screen
    }
  };
}



package com.rim.samples.device.accelerometerdemo;
/*
 * SSPScreen.java
 *
 * Copyright © 1998-2008 Research In Motion Ltd.
 *
*/
import net.rim.device.api.ui.MenuItem;
import net.rim.device.api.ui.component.BasicEditField;
import net.rim.device.api.ui.component.RichTextField;
```

```
import net.rim.device.api.ui.component.Status;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.component.CheckboxField;
public class SPPScreen extends MainScreen{
//Theses checkbox, basic edit field etc are show on the BlackBerry Screen. The user
could modified some parameters from there.
    private RichTextField _rtf;
    private Sensors extSensors;
private CheckboxField _cbLog=new CheckboxField("Select Log",false);
private CheckboxField _cbGPS=new CheckboxField("Select GPS",false);
private CheckboxField _cbVideo=new CheckboxField("Select Video",false);
private CheckboxField _cbRawData=new CheckboxField("Log Raw Data",false);
private CheckboxField _cbFeatureData=new CheckboxField("Log Feature
Data",true);
private BasicEditField Dy_th_1=new BasicEditField("Dynamic Thr 1:
","170",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Dy_th_2=new BasicEditField("Dynamic Thr 2:
","1100",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Dy_th_3=new BasicEditField("Dynamic Thr 3:
","350",10,BasicEditField.FILTER_INTEGER);
private BasicEditField St_th=new BasicEditField("Static Thr:
","160",10,BasicEditField.FILTER_INTEGER);
private BasicEditField DownStair_th_1=new BasicEditField("DownStair Thr 1:
","1300",10,BasicEditField.FILTER_INTEGER);
private BasicEditField DownStair_th_2=new BasicEditField("DownStair Thr 2:
","3200",10,BasicEditField.FILTER_INTEGER);
private BasicEditField DownStair_th_3=new BasicEditField("DownStair Thr 3:
","18000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_1=new BasicEditField("Not DownStair
Thr 1: ","-600",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_2=new BasicEditField("Not DownStair
Thr 2: ","2000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_3=new BasicEditField("Not DownStair
Thr 3: ","3000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_4=new BasicEditField("Not DownStair
Thr 4: ","2200",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_5=new BasicEditField("Not DownStair
Thr 5: ","2400",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_6=new BasicEditField("Not DownStair
Thr 6: ","16000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_7=new BasicEditField("Not DownStair
Thr 7: ","830",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Not_DownStair_th_8=new BasicEditField("Not DownStair
Thr 8: ","500",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Upstair_th_1=new BasicEditField("UpStair Thr 1:
","2500",10,BasicEditField.FILTER_INTEGER);
```

```
private BasicEditField Upstair_th_2=new BasicEditField("UpStair Thr 2:
","1000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Upstair_th_3=new BasicEditField("UpStair Thr 3:
","13000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Ramp_th_1=new BasicEditField("Ramp Thr 1:
","1200",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Ramp_th_2=new BasicEditField("Ramp Thr 2:
","2500",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Ramp_th_3=new BasicEditField("Ramp Thr 3:
","600",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Elevator_th_1=new BasicEditField("Elevator Thr 1:
","40",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Elevator_th_2=new BasicEditField("Elevator Thr 2:
","180",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Elevator_th_3=new BasicEditField("Elevator Thr 3:
","960",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Elevator_th_4=new BasicEditField("Elevator Thr 4:
","1200",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Elevator_th_5=new BasicEditField("Elevator Thr 5:
","158",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Elevator_th_6=new BasicEditField("Elevator Thr 6:
","50",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Movestatic_th_h=new BasicEditField("Movestatic Thr h:
","100",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Movestatic_th_l=new BasicEditField("Movestatic Thr l:
","70",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Thr_ly_l=new BasicEditField("Threshold lying low:
","50",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Thr_ly_h=new BasicEditField("Threshold lying high:
","120",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Thr_st_l=new BasicEditField("Threshold standing low:
","130",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Thr_st_h=new BasicEditField("Threshold standing high:
","188",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Thr_car_1=new BasicEditField("Threshold Car ride:
","3",10,BasicEditField.FILTER_INTEGER);
private BasicEditField takingvideotime=new BasicEditField("Taking video time:
","3000",10,BasicEditField.FILTER_INTEGER);
private BasicEditField Car_stop_count=new BasicEditField("Car stop count:
","19",10,BasicEditField.FILTER_INTEGER);
/** The field containing the feed from the camera. */
    private Configuration config=new Configuration();
    public SPPScreen()
    {
            _rtf = new RichTextField("Start...");
            extSensors=new Sensors();
```

```
            add(_rtf);
            addMenuItem(_startMeasurement);
            addMenuItem(_config);
    }
    private void showAllConfigField(){
            add(_cbLog);
            add(_cbGPS);
            add(_cbVideo);
            add(_cbRawData);
            add(_cbFeatureData);
            add(Dy_th_1);
            add(Dy_th_2);
            add(Dy_th_3);
            add(St_th);
            add(DownStair_th_1);
            add(DownStair_th_2);
            add(DownStair_th_3);
            add(Not_DownStair_th_1);
            add(Not_DownStair_th_2);
            add(Not_DownStair_th_3);
            add(Not_DownStair_th_4);
            add(Not_DownStair_th_5);
            add(Not_DownStair_th_6);
            add(Not_DownStair_th_7);
            add(Not_DownStair_th_8);
            add(Upstair_th_1);
            add(Upstair_th_2);
            add(Upstair_th_3);
            add(Ramp_th_1);
            add(Ramp_th_2);
            add(Ramp_th_3);
            add(Elevator_th_1);
            add(Elevator_th_2);
            add(Elevator_th_3);
            add(Elevator_th_4);
            add(Elevator_th_5);
            add(Elevator_th_6);
            add(Movestatic_th_h);
            add(Movestatic_th_l);
            add(Thr_ly_l);
            add(Thr_ly_h);
            add(Thr_st_l);
            add(Thr_st_h);
            add(Thr_car_1);
            add(takingvideotime);
            add(Car_stop_count);
```

```
}
private void deleteAllConfigField(){
        if (_cbLog.isVisible())
        {
                delete(_cbLog);
                delete(_cbGPS);
                delete(_cbVideo);
                delete(_cbRawData);
                delete(_cbFeatureData);
                delete(Dy_th_1);
                delete(Dy_th_2);
                delete(Dy_th_3);
                delete(St_th);
                delete(DownStair_th_1);
                delete(DownStair_th_2);
                delete(DownStair_th_3);
                delete(Not_DownStair_th_1);
                delete(Not_DownStair_th_2);
                delete(Not_DownStair_th_3);
                delete(Not_DownStair_th_4);
                delete(Not_DownStair_th_5);
                delete(Not_DownStair_th_6);
                delete(Not_DownStair_th_7);
                delete(Not_DownStair_th_8);
                delete(Upstair_th_1);
                delete(Upstair_th_2);
                delete(Upstair_th_3);
                delete(Ramp_th_1);
                delete(Ramp_th_2);
                delete(Ramp_th_3);
                delete(Elevator_th_1);
                delete(Elevator_th_2);
                delete(Elevator_th_3);
                delete(Elevator_th_4);
                delete(Elevator_th_5);
                delete(Elevator_th_6);
                delete(Movestatic_th_h);
                delete(Movestatic_th_l);
                delete(Thr_ly_l);
                delete(Thr_ly_h);
                delete(Thr_st_l);
                delete(Thr_st_h);
                delete(Thr_car_1);
                delete(takingvideotime);
                delete(Car_stop_count);
        }
```

```
}
private void updateConfiguration(){
        config.setFlagGPS(_cbGPS.getChecked());
        config.setFlagLog(_cbLog.getChecked());
        config.setFlagVideo(_cbVideo.getChecked());
        config.setFlagRawDataOnly(_cbRawData.getChecked());
        config.setFlagFeatureData(_cbFeatureData.getChecked());
        String tempstr=Dy_th_1.getText();
        int tempint=Integer.parseInt(tempstr);
        config.setDyth1(tempint);
        tempstr=Dy_th_2.getText();
        tempint=Integer.parseInt(tempstr);
        config.setDyth2(tempint);
        tempstr=Dy_th_3.getText();
        tempint=Integer.parseInt(tempstr);
        config.setDyth3(tempint);
        tempstr=St_th.getText();
        tempint=Integer.parseInt(tempstr);
        config.setStth(tempint);
        tempstr=DownStair_th_1.getText();
        tempint=Integer.parseInt(tempstr);
        config.setDownStairth1(tempint);
        tempstr=DownStair_th_2.getText();
        tempint=Integer.parseInt(tempstr);
        config.setDownStairth2(tempint);
        tempstr=DownStair_th_3.getText();
        tempint=Integer.parseInt(tempstr);
        config.setDownStairth3(tempint);
        tempstr=Not_DownStair_th_1.getText();
        tempint=Integer.parseInt(tempstr);
        config.setNotDownStairth1(tempint);
        tempstr=Not_DownStair_th_2.getText();
        tempint=Integer.parseInt(tempstr);
        config.setNotDownStairth2(tempint);
        tempstr=Not_DownStair_th_3.getText();
        tempint=Integer.parseInt(tempstr);
        config.setNotDownStairth3(tempint);
        tempstr=Not_DownStair_th_4.getText();
        tempint=Integer.parseInt(tempstr);
        config.setNotDownStairth4(tempint);
        tempstr=Not_DownStair_th_5.getText();
        tempint=Integer.parseInt(tempstr);
        config.setNotDownStairth5(tempint);
        tempstr=Not_DownStair_th_6.getText();
        tempint=Integer.parseInt(tempstr);
        config.setNotDownStairth6(tempint);
```

```
tempstr=Not_DownStair_th_7.getText();
tempint=Integer.parseInt(tempstr);
config.setNotDownStairth7(tempint);
tempstr=Not_DownStair_th_8.getText();
tempint=Integer.parseInt(tempstr);
config.setNotDownStairth8(tempint);

tempstr=Upstair_th_1.getText();
tempint=Integer.parseInt(tempstr);
config.setUpstairth1(tempint);
tempstr=Upstair_th_2.getText();
tempint=Integer.parseInt(tempstr);
config.setUpstairth2(tempint);
tempstr=Upstair_th_3.getText();
tempint=Integer.parseInt(tempstr);
config.setUpstairth3(tempint);
tempstr=Ramp_th_1.getText();
tempint=Integer.parseInt(tempstr);
config.setRampth1(tempint);
tempstr=Ramp_th_2.getText();
tempint=Integer.parseInt(tempstr);
config.setRampth2(tempint);
tempstr=Ramp_th_3.getText();
tempint=Integer.parseInt(tempstr);
config.setRampth3(tempint);
tempstr=Elevator_th_1.getText();
tempint=Integer.parseInt(tempstr);
config.setElevatorth1(tempint);
tempstr=Elevator_th_2.getText();
tempint=Integer.parseInt(tempstr);
config.setElevatorth2(tempint);
tempstr=Elevator_th_3.getText();
tempint=Integer.parseInt(tempstr);
config.setElevatorth3(tempint);
tempstr=Elevator_th_4.getText();
tempint=Integer.parseInt(tempstr);
config.setElevatorth4(tempint);
tempstr=Elevator_th_5.getText();
tempint=Integer.parseInt(tempstr);
config.setElevatorth5(tempint);
tempstr=Elevator_th_6.getText();
tempint=Integer.parseInt(tempstr);
config.setElevatorth6(tempint);
tempstr=Movestatic_th_h.getText();
tempint=Integer.parseInt(tempstr);
config.setMovestaticthh(tempint);
```

```
            tempstr=Movestatic_th_l.getText();
            tempint=Integer.parseInt(tempstr);
            config.setMovestaticthl(tempint);
            tempstr=Thr_ly_l.getText();
            tempint=Integer.parseInt(tempstr);
            config.setThrlyl(tempint);
            tempstr=Thr_ly_h.getText();
            tempint=Integer.parseInt(tempstr);
            config.setThrlyh(tempint);
            tempstr=Thr_st_l.getText();
            tempint=Integer.parseInt(tempstr);
            config.setThrstl(tempint);
            tempstr=Thr_st_h.getText();
            tempint=Integer.parseInt(tempstr);
            config.setThrsth(tempint);


            tempstr=Thr_car_1.getText();
            tempint=Integer.parseInt(tempstr);
            config.setThrcar1(tempint);
            tempstr=takingvideotime.getText();
            tempint=Integer.parseInt(tempstr);
            config.setTakingVideoTime(tempint);
            tempstr=Car_stop_count.getText();
            tempint=Integer.parseInt(tempstr);
            config.setCarstopcount(tempint);
            }
    //Close application. When close button on BB is press, this method is called.
    public void close(){
            if (extSensors != null){
                    extSensors.stopMeasurement();
       try {
                     Thread.sleep(500);
                     } catch (InterruptedException e1) {
                     e1.printStackTrace();
                     }
      extSensors.closeGPS();
       try {
                     Thread.sleep(500);
                     } catch (InterruptedException e1) {
                     e1.printStackTrace();
                     }
                    extSensors.closeSP();
            }
            else {
                    Status.show("ExtSensors is null\n");
```

```
        }
    try {
                Thread.sleep(500);
                } catch (InterruptedException e1) {
                e1.printStackTrace();
                }
        System.exit(0);
}
//////////////////////////////////////////////////////////
//      Menu Items          //
//////////////////////////////////////////////////////////
private MenuItem _takeVideo = new MenuItem("Take Video", 90, 90)
{
        public void run() {
                if (extSensors.video!=null)
                {
                        _rtf.setText("Taking a Video Clip");
                        extSensors.video.startRecording();
                }
                else {
                        _rtf.setText("There is no Video Camera");
                }
        }
};

private MenuItem _stopVideo = new MenuItem("Stop Video", 90, 90)
{
        public void run() {
                        _rtf.setText("Stoping Camera");
                        extSensors.stopVideo();
        }
};

private MenuItem _closeVideo = new MenuItem("Save and Close Video", 90, 90)
{
        public void run() {
                if (extSensors!=null){
                        _rtf.setText("Saving and Closing Camera");
                        extSensors.closeVideoFull();
                        delete(extSensors.video.videoField);
                        removeMenuItem(_takeVideo);
                        removeMenuItem(_closeVideo);
                }
                else {
                        _rtf.setText("No Video Camera to close");
                }
```

```
            }
        };
        private MenuItem _startMeasurement =new MenuItem("Start Measurement",
    90,90)
        {
            public void run (){
                deleteAllConfigField();
                updateConfiguration();
                boolean flagVideo=config.getFlagVideo();
                if (flagVideo)
                {
                    if (extSensors.video==null){
                        _rtf.setText("Starting Video");
                        extSensors.startVideo();
                        addMenuItem(_takeVideo);
                        addMenuItem(_stopVideo);
                        addMenuItem(_closeVideo);
                        add(extSensors.video.videoField);
                        extSensors.video.startThread();
                    }
                    else {
                    }
                }
                else if (!flagVideo){
                    if (extSensors.video!=null){
                        _rtf.setText("Closing Video");
                        extSensors.closeVideoFull();
                        removeMenuItem(_takeVideo);
                        removeMenuItem(_closeVideo);
                    }
                }
                if (extSensors!=null){
                    _rtf.setText("Getting Motion Data");
                    extSensors.startMeasurement(config);
                }
                else if (extSensors==null){
                    _rtf.setText("No external Sensors");
                }
                addMenuItem(_stopMeasurement);
                removeMenuItem(_startMeasurement);
                removeMenuItem(_config);
            }
        };

        private MenuItem _stopMeasurement =new MenuItem("Stop Measurement",
    90,90)
```

```
        {
                public void run (){
                        _rtf.setText("Stop Measurement");
                        if (extSensors!=null){
                                extSensors.stopMeasurement();
                        }
                        else if (extSensors==null){
                                _rtf.setText("No external Sensors to stop");
                        }
                        addMenuItem(_startMeasurement);
                        addMenuItem(_config);
                        removeMenuItem(_stopMeasurement);
                }
        };
        private MenuItem _config =new MenuItem("Configuration", 90,90)
        {
                public void run (){
                        deleteAll();
                        add(_rtf);
                        _rtf.setText("Configuration");
                        showAllConfigField();
                        removeMenuItem(_config);
                }
        };
}


package com.rim.samples.device.accelerometerdemo;
public class Configuration {
private boolean flagLog; //If true, means log file is created
private boolean flagGPS; //if true, means GPS data is collected.
private boolean flagVideo; //If true, means Video function is on.
private boolean flagRawDataOnly; //If true, means raw data is logged.
    private byte samplingDelay; //Value of the sampling delay
    private boolean flagFeatureData;
    private int Dy_th_1;
    private int Dy_th_2;
    private int Dy_th_3;
    private int St_th;
    private int DownStair_th_1;
    private int DownStair_th_2;
    private int DownStair_th_3;
    private int Not_DownStair_th_1;
    private int Not_DownStair_th_2;
    private int Not_DownStair_th_3;
    private int Not_DownStair_th_4;
    private int Not_DownStair_th_5;
```

```
private int Not_DownStair_th_6;
private int Not_DownStair_th_7;
private int Not_DownStair_th_8;
private int Upstair_th_1;
private int Upstair_th_2;
private int Upstair_th_3;
private int Ramp_th_1;
private int Ramp_th_2;
private int Ramp_th_3;
private int Elevator_th_1;
private int Elevator_th_2;
private int Elevator_th_3;
private int Elevator_th_4;
private int Elevator_th_5;
private int Elevator_th_6;
private int Movestatic_th_h;
private int Movestatic_th_l;
private int Thr_ly_l;
private int Thr_ly_h;
private int Thr_st_l;
private int Thr_st_h;
private int Thr_car_1;
private int takingvideotime;
private int Car_stop_count;
public Configuration() {
}
public void setFlagLog(boolean flagValue){
        flagLog=flagValue;
}
public boolean getFlagLog(){
        return flagLog;
}
public void setFlagGPS(boolean flagValue){
        flagGPS=flagValue;
}
public boolean getFlagGPS(){
        return flagGPS;
}
public void setFlagVideo(boolean flagValue){
        flagVideo=flagValue;
}
public boolean getFlagVideo(){
        return flagVideo;
}
public void setFlagRawDataOnly(boolean flagValue){
        flagRawDataOnly=flagValue;
```

```
}
public boolean getFlagRawDataOnly(){
        return flagRawDataOnly;
}
public void setSamplingDelay(byte bValue){
        samplingDelay=bValue;
}
public byte getSamplingDelay(){
        return samplingDelay;
}
public void setFlagFeatureData(boolean flagValue){
        flagFeatureData=flagValue;
}
public boolean getFlagFeatureData(){
        return flagFeatureData;
}
public void setDyth1(int threshold){
        Dy_th_1=threshold;
}
public int getDyth1(){
        return Dy_th_1;
}
public void setDyth2(int threshold){
        Dy_th_2=threshold;
}
public int getDyth2(){
        return Dy_th_2;
}
public void setDyth3(int threshold){
        Dy_th_3=threshold;
}
public int getDyth3(){
        return Dy_th_3;
}
public void setStth(int threshold){
        St_th=threshold;
}
public int getStth(){
        return St_th;
}
public void setDownStairth1(int threshold){
        DownStair_th_1=threshold;
}
public int getDownStairth1(){
        return DownStair_th_1;
}
```

```
public void setDownStairth2(int threshold){
        DownStair_th_2=threshold;
}
public int getDownStairth2(){
        return DownStair_th_2;
}
public void setDownStairth3(int threshold){
        DownStair_th_3=threshold;
}
public int getDownStairth3(){
        return DownStair_th_3;
}
public void setNotDownStairth1(int threshold){
        Not_DownStair_th_1=threshold;
}
public int getNotDownStairth1(){
        return Not_DownStair_th_1;
}
public void setNotDownStairth2(int threshold){
        Not_DownStair_th_2=threshold;
}
public int getNotDownStairth2(){
        return Not_DownStair_th_2;
}
public void setNotDownStairth3(int threshold){
        Not_DownStair_th_3=threshold;
}
public int getNotDownStairth3(){
        return Not_DownStair_th_3;
}
public void setNotDownStairth4(int threshold){
        Not_DownStair_th_4=threshold;
}
public int getNotDownStairth4(){
        return Not_DownStair_th_4;
}
public void setNotDownStairth5(int threshold){
        Not_DownStair_th_5=threshold;
}
public int getNotDownStairth5(){
        return Not_DownStair_th_5;
}
public void setNotDownStairth6(int threshold){
        Not_DownStair_th_6=threshold;
}
public int getNotDownStairth6(){
```

```
        return Not_DownStair_th_6;
}
public void setNotDownStairth7(int threshold){
        Not_DownStair_th_7=threshold;
}
public int getNotDownStairth7(){
        return Not_DownStair_th_7;
}
public void setNotDownStairth8(int threshold){
        Not_DownStair_th_8=threshold;
}
public int getNotDownStairth8(){
        return Not_DownStair_th_8;
}
public void setUpstairth1(int threshold){
        Upstair_th_1=threshold;
}
public int getUpstairth1(){
        return Upstair_th_1;
}
public void setUpstairth2(int threshold){
        Upstair_th_2=threshold;
}
public int getUpstairth2(){
        return Upstair_th_2;
}
public void setUpstairth3(int threshold){
        Upstair_th_3=threshold;
}
public int getUpstairth3(){
        return Upstair_th_3;
}
public void setRampth1(int threshold){
        Ramp_th_1=threshold;
}
public int getRampth1(){
        return Ramp_th_1;
}
public void setRampth2(int threshold){
        Ramp_th_2=threshold;
}
public int getRampth2(){
        return Ramp_th_2;
}
public void setRampth3(int threshold){
        Ramp_th_3=threshold;
```

```
}
public int getRampth3(){
        return Ramp_th_3;
}
public void setElevatorth1(int threshold){
        Elevator_th_1=threshold;
}
public int getElevatorth1(){
        return Elevator_th_1;

}
public void setElevatorth2(int threshold){
        Elevator_th_2=threshold;
}
public int getElevatorth2(){
        return Elevator_th_2;
}
public void setElevatorth3(int threshold){
        Elevator_th_3=threshold;
}
public int getElevatorth3(){
        return Elevator_th_3;
}
public void setElevatorth4(int threshold){
        Elevator_th_4=threshold;
}
public int getElevatorth4(){
        return Elevator_th_4;
}
public void setElevatorth5(int threshold){
        Elevator_th_5=threshold;
}
public int getElevatorth5(){
        return Elevator_th_5;
}
public void setElevatorth6(int threshold){
        Elevator_th_6=threshold;
}
public int getElevatorth6(){
        return Elevator_th_6;
}
public void setMovestaticthh(int threshold){
        Movestatic_th_h=threshold;
}
public int getMovestaticthh(){
        return Movestatic_th_h;
```

```
}
public void setMovestaticthl(int threshold){
        Movestatic_th_l=threshold;
}
public int getMovestaticthl(){
        return Movestatic_th_l;
}
public void setThrlyl(int threshold){
        Thr_ly_l=threshold;
}
public int getThrlyl(){
        return Thr_ly_l;
}
public void setThrlyh(int threshold){
        Thr_ly_h=threshold;
}
public int getThrlyh(){
        return Thr_ly_h;
}
public void setThrstl(int threshold){
        Thr_st_l=threshold;
}
public int getThrstl(){
        return Thr_st_l;
}
public void setThrsth(int threshold){
        Thr_st_h=threshold;
}
public int getThrsth(){
        return Thr_st_h;
}
public void setThrcar1(int threshold){
        Thr_car_1=threshold;
}
public int getThrcar1(){
        return Thr_car_1;
}
public void setTakingVideoTime(int threshold){
        takingvideotime=threshold;
}
public int getTakingVideoTime(){
        return takingvideotime;
}
public void setCarstopcount(int threshold){
        Car_stop_count=threshold;
}
```

```
    public int getCarstopcount(){
            return Car_stop_count;
    }
}


package com.rim.samples.device.accelerometerdemo;

/**
* AccelerometerDemo.java
* Copyright © 1998-2010 Research In Motion Ltd. All Right Reserved
*/
/*
* Sensors.java
*
* © Edward Lemaire, The Ottawa Hospital Rehabilitation Centre, 2011
* Confidential and proprietary.
*/
import com.rim.samples.device.accelerometerdemo.LogFile;
import net.rim.device.api.math.Fixed32;
import net.rim.device.api.system.*;
import net.rim.device.api.system.AccelerometerSensor.*;
import net.rim.device.api.ui.component.*;
class Sensors
{
    //Object
    public BBVideo video;
    private Configuration configuration; //=new Configuration();
    private LogFile logfile1;
    private FeatureFile featurefile1;
    private GPSdata gps;
  //thread
    private WriteToFileRawDataThread _writingRawDataThread;
    private WriteToFileFeatureDataThread _writingFeatureDataThread;
    //Flag Variable
    private static boolean logging=false;
    private static boolean flagRawData=false;
    private static boolean flagFeatureData=false;
    private static boolean flagVideo=false;
    private static boolean flagGPS=false;
    //Channel
    private Channel _accChannel;
    //AccelerometerData
    private AccelerometerData accData;
    public static volatile long appStartTime1=0;
    public static volatile long appStartTime2=0;
    public static volatile long feature_realTime=0;
```

```
public static volatile float FeatureTime =0;
public static volatile float Time =0;
public static volatile float Time2 =0;
public static volatile float Time3 =0;
public static volatile float Time4 =0;
public static volatile float Time5 =0;
public static volatile float Time6 =0;
public static volatile float Time7 =0;
public static volatile float Time8 =0;
public static volatile float Time9 =0;
public static volatile float Time10 =0;
public static volatile float PrevTime =0;
public static volatile long realTime =0;
public static volatile long realTime1 =0;
public static volatile long realTime2 =0;
public static volatile long realTime3 =0;
public static volatile long realTime4 =0;
public static volatile long realTime5 =0;
public static volatile long realTime6 =0;
public static volatile long realTime7 =0;
public static volatile long realTime8 =0;
public static volatile long realTime9 =0;
public static volatile long realTime10 =0;
public static volatile long endTime =0;
float Rate1;
float Rate2;
float Rate3;
float Rate4;
float Rate5;
float Rate6;
float Rate7;
float Rate8;
public static volatile long end =0;
int frequency=8;
int frequency_count;
int feature_frequency=8;
int feature_frequency_count;
//Data field
short xAccel;
short yAccel;
short zAccel;
short xAccel2;
short yAccel2;
short zAccel2;
short xAccel3;
short yAccel3;
```

```
short zAccel3;
short xAccel4;
short yAccel4;
short zAccel4;
short xAccel5;
short yAccel5;
short zAccel5;
short xAccel6;
short yAccel6;
short zAccel6;
short xAccel7;
short yAccel7;
short zAccel7;
short xAccel8;
short yAccel8;
short zAccel8;
short xAccel0;
short yAccel0;
short zAccel0;

short xAccel20;
short yAccel20;
short zAccel20;
short xAccel30;
short yAccel30;
short zAccel30;
short xAccel40;
short yAccel40;
short zAccel40;
short xAccel50;
short yAccel50;
short zAccel50;
short xAccel60;
short yAccel60;
short zAccel60;
short xAccel70;
short yAccel70;
short zAccel70;
short xAccel80;
short yAccel80;
short zAccel80;
short prevxAccel;
short prevyAccel;
short prevzAccel;
short prevxAccel2;
short prevyAccel2;
```

```
    short prevzAccel2;
    short prevxAccel3;
    short prevyAccel3;
    short prevzAccel3;
    short prevxAccel4;
    short prevyAccel4;
    short prevzAccel4;
    short prevxAccel5;
    short prevyAccel5;
    short prevzAccel5;
    short prevxAccel6;
    short prevyAccel6;
    short prevzAccel6;
    short prevxAccel7;
    short prevyAccel7;
    short prevzAccel7;
    short prevxAccel8;
    short prevyAccel8;
    short prevzAccel8;
  //Feature
    short xAccelMax;
    short yAccelMax;
    short zAccelMax;
    short xAccelMin;
    short yAccelMin;
    short zAccelMin;
    short RangeX0;
    short RangeY0;
    short RangeZ0;

    short RangeX;
    short RangeY;
    short RangeZ;
    short RangeXZ;
    float MeanX;
    float MeanY;
    float MeanZ;
    double varianceX;
    double STDX;
    double varianceY;
    double STDY;
    double varianceZ;
    double STDZ;
    int TiltAngZY;
    short SumofRange;
    short SumofRange0;
```

```
    short prevSR;
    short prevSR2;
    short prevSR3;
    short prevSR4;
    short prevSR5;
    short prevSR6;
    short prevSR7;
    short SMASR;
    short DiffSR;
    //Calibration value
    int XzeroOffset=11;
    int YzeroOffset=35;
    int ZzeroOffset=17;
    //Battery level & signal intensity
    int BattLevel;
    int BattTemp;
    int BattVol;
    int Radio1;
    int Wireless1;
    int GPRS;
    //For GPS
    double gpsLong;
    double gpsLat;
    float gpsAlt;
    double gpsHead;
    float gpsSpeed;
  float prevgpsSpeed;
    float SumofGPSSpeed;
    //Threshold variable
    int Dy_th_1;
    int Dy_th_2;
    int Dy_th_3;
    int St_th;
    int DownStair_th_1;
    int DownStair_th_2;
    int DownStair_th_3;
    int Not_DownStair_th_1;
    int Not_DownStair_th_2;
    int Not_DownStair_th_3;
    int Not_DownStair_th_4;
    int Not_DownStair_th_5;
    int Not_DownStair_th_6;
    int Not_DownStair_th_7;
    int Not_DownStair_th_8;
    int Upstair_th_1;
    int Upstair_th_2;
```

```
    int Upstair_th_3;
    int Ramp_th_1;
    int Ramp_th_2;
    int Ramp_th_3;
    int Elevator_th_1;
    int Elevator_th_2;
    int Elevator_th_3;
    int Elevator_th_4;
    int Elevator_th_5;
    int Elevator_th_6;
    int Movestatic_th_h;
    int Movestatic_th_l;
    int Thr_ly_l;
    int Thr_ly_h;
    int Thr_st_l;
    int Thr_st_h;
    int Thr_movest_l;
    int Thr_movest_h;
    int Thr_car_1;
    int Car_stop_count;
    //For classification
short state=0;
short ststate=0;
short prevststate=0;
short dstate=0;
short prevdstate=0;
short sdstate=0;
short prevsdstate=0;
short moveststate=0;
short prevmoveststate=0;
short prevmoveststate2=0;
short ustate=0;
short prevustate=0;
short rstate=0;
short prevrstate=0;
short estate=0;
short prevestate=0;
short TakeVideo=0;
short prevTakeVideo=0;
short cstate=0;
short prevcstate=0;
short prevState=0;
short prevState2=0;
short prevState3=0;
short changeOfState=0;
short changeOfState2=0;
```

```
 short changeOfState3=0;
 short ExpectedAct=0;
 short PrevExpectedAct=0;

 //Count
 int TakeVideoCount=0;
 int VideoTimeCount=0;
 int VideoStartCount=0;
 int CarStopCount=0;
 int CarRideCount=0;
// Constructor
public Sensors(){
}
public void closeSP()
{
  configuration=null;
}
public void startMeasurement(Configuration config){
    configuration=config;
    logging=configuration.getFlagLog();
    flagRawData=configuration.getFlagRawDataOnly();
    flagFeatureData=configuration.getFlagFeatureData();
    flagVideo=configuration.getFlagVideo();
    flagGPS=configuration.getFlagGPS();
    initializeBBThread();
    if (logging){
            logfile1=new LogFile();
            logfile1.writeStringToLog("start thread, delay:
"+configuration.getSamplingDelay());
            logfile1.closeLogFile();
    }
    else{
            logfile1=null;
    }
    if (flagGPS){
            startGPS();
    }
    else {
            if (gps!=null){
                    closeGPS();
            }
    }
 openThreads();
    startThreads();

}
```

```
public void stopMeasurement(){
   try{
            closeVideoFull();
            stopThreads();
   }
   catch (Exception e){
            Status.show("erro in closing"+e.toString());
   }
   if( _writingRawDataThread!= null ){
   synchronized( _writingRawDataThread ){
            _writingRawDataThread._running = false;
            _writingRawDataThread.notifyAll();
            _writingRawDataThread = null;
            logfile1.writeStringToLog("stop thread\n");
                     logfile1.writeStringToLog("End Raw Time1,");
                     end=System.currentTimeMillis();
                     endTime=(end-appStartTime1)/1000;
        logfile1.writeStringToLog(endTime);
            logfile1.closeLogFile();
            logfile1=null;
   }
    }
}
private void initializeBBThread()
{
   //Get threshold values.
   Dy_th_1=configuration.getDyth1();
   Dy_th_2=configuration.getDyth2();
   Dy_th_3=configuration.getDyth3();
   St_th=configuration.getStth();
   DownStair_th_1=configuration.getDownStairth1();
   DownStair_th_2=configuration.getDownStairth2();
   DownStair_th_3=configuration.getDownStairth3();
   Not_DownStair_th_1=configuration.getNotDownStairth1();
   Not_DownStair_th_2=configuration.getNotDownStairth2();
   Not_DownStair_th_3=configuration.getNotDownStairth3();
   Not_DownStair_th_4=configuration.getNotDownStairth4();
   Not_DownStair_th_5=configuration.getNotDownStairth5();
   Not_DownStair_th_6=configuration.getNotDownStairth6();
   Not_DownStair_th_7=configuration.getNotDownStairth7();
   Not_DownStair_th_8=configuration.getNotDownStairth8();
   Upstair_th_1=configuration.getUpstairth1();
   Upstair_th_2=configuration.getUpstairth2();
   Upstair_th_3=configuration.getUpstairth3();
   Ramp_th_1=configuration.getRampth1();
   Ramp_th_2=configuration.getRampth2();
```

```
        Ramp_th_3=configuration.getRampth3();
        Elevator_th_1=configuration.getElevatorth1();
        Elevator_th_2=configuration.getElevatorth2();
        Elevator_th_3=configuration.getElevatorth3();
        Elevator_th_4=configuration.getElevatorth4();
        Elevator_th_5=configuration.getElevatorth5();
        Elevator_th_6=configuration.getElevatorth6();
        Movestatic_th_h=configuration.getMovestaticthh();
        Movestatic_th_l=configuration.getMovestaticthl();
        Thr_ly_l=configuration.getThrlyl();
      Thr_ly_h=configuration.getThrlyh();
      Thr_st_l=configuration.getThrstl();
      Thr_st_h=configuration.getThrsth();
      Thr_car_1=configuration.getThrcar1();
      Car_stop_count=configuration.getCarstopcount();
     }
     private void startThreads()
     {
        if (flagRawData){
                _writingRawDataThread.start();
        }
        if (flagFeatureData){
                _writingFeatureDataThread.start();
        }
     }
     private void openThreads()
     {
       _accChannel =
    AccelerometerSensor.openRawDataChannel( Application.getApplication() );
        if (flagRawData){
        _writingRawDataThread = new WriteToFileRawDataThread();
        }
        else {
        _writingRawDataThread=null;
        }
        if (flagFeatureData){
        _writingFeatureDataThread = new WriteToFileFeatureDataThread();
        }
        else {
                _writingFeatureDataThread=null;
        }
     }
    //Tilt angle calculation using the two axes method and atan2.     private int
    TiltAnglezy2(float fvalz, float fvaly){
        int nz=(int)(fvalz*10000);
        int ny=(int)(fvaly*10000);
```

```
    nz=Fixed32.tenThouToFP(nz);
    ny=Fixed32.tenThouToFP(ny);
    int ang=Fixed32.atand2(nz, ny);
    ang=Fixed32.toRoundedInt(ang);
    ang=ang+180; //Added offset to have the range from 0 to 360 instead of -180 to
180. 0 degree is when person is head down, feet up.              return ang;
    }
 /**
  * A thread class to handle screen updates.
  */
 private class WriteToFileRawDataThread extends Thread{
  private boolean _running;
  public void run(){
 _running = true;
    logfile1=new LogFile();
    frequency_count=(frequency -1 );
    appStartTime1=System.currentTimeMillis();
    logfile1.writeStringToLog("start thread,,,,,,,,,Window size,");
    logfile1.writeStringToLog(frequency);
    logfile1.writeStringToLog(",");
    logfile1.writeStringToLog("Hz");
    logfile1.writeStringToLog(",");
    logfile1.writeStringToLog(frequency_count);
    logfile1.writeStringToLog("\n");
    logfile1.writeStringToLog("startRawTime1,");
    logfile1.writeStringToLog(appStartTime1);
    logfile1.writeStringToLog("\n");
    logfile1.writeStringToLog("RealTime,Time,RawX,RawY,RawZ,Rate,Longitude,
Latitude,Altitude,Heading,Speed,BattLevel,Batt_Temp,Batt_V,RadioS,GPRSS,Wirel
essS\n");
            /**
       * Gets the latest acceleromenter data.
       * @throws IOException
       */
    while( _running ){
            // Real device, call the API for samples.
     try {
                    Thread.sleep(100);
            } catch (InterruptedException e1) {
                    e1.printStackTrace();
            }
     accData = _accChannel.getAccelerometerData();
     realTime = System.currentTimeMillis();
     Time=(float)(realTime - appStartTime1)/1000;
     xAccel = accData.getLastXAcceleration();
     yAccel = accData.getLastYAcceleration();
```

```
zAccel = accData.getLastZAcceleration();
Rate1=(1/(Time-PrevTime));
//Origianl calibration
xAccel0=(short) (xAccel-XzeroOffset);
yAccel0=(short) (yAccel-XzeroOffset);
zAccel0=(short) (zAccel-XzeroOffset);
logfile1.writeStringToLog(realTime);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Time);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(xAccel0);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(yAccel0);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(zAccel0);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Rate1);
logfile1.writeStringToLog("\n");
try {
        Thread.sleep(100);
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }
    accData = _accChannel.getAccelerometerData();
    realTime2 = System.currentTimeMillis();
Time2=(float)(realTime2-appStartTime1)/1000;
Rate2=(1/(Time2-Time));
xAccel2 = accData.getLastXAcceleration();
yAccel2 = accData.getLastYAcceleration();
zAccel2 = accData.getLastZAcceleration();
xAccel20=(short) (xAccel2-XzeroOffset);
yAccel20=(short) (yAccel2-XzeroOffset);
zAccel20=(short) (zAccel2-XzeroOffset);
logfile1.writeStringToLog(realTime2);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Time2);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(xAccel20);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(yAccel20);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(zAccel20);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Rate2);
logfile1.writeStringToLog("\n");
try {
```

```
                Thread.sleep(100);
  } catch (InterruptedException e1) {
                e1.printStackTrace();
  }
        accData = _accChannel.getAccelerometerData();
        realTime3 = System.currentTimeMillis();
Time3=(float)(realTime3-appStartTime1)/1000;
Rate3=(1/(Time3-Time2));
xAccel3 = accData.getLastXAcceleration();
yAccel3 = accData.getLastYAcceleration();
zAccel3 = accData.getLastZAcceleration();
xAccel30=(short) (xAccel3-XzeroOffset);
yAccel30=(short) (yAccel3-XzeroOffset);
zAccel30=(short) (zAccel3-XzeroOffset);
logfile1.writeStringToLog(realTime3);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Time3);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(xAccel30);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(yAccel30);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(zAccel30);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Rate3);
        logfile1.writeStringToLog("\n");
 try {
                 Thread.sleep(100);
                } catch (InterruptedException e1) {
                 e1.printStackTrace();
                }
        accData = _accChannel.getAccelerometerData();
                realTime4 = System.currentTimeMillis();
 Time4=(float)(realTime4-appStartTime1)/1000;
 Rate4=(1/(Time4-Time3));
 xAccel4 = accData.getLastXAcceleration();
 yAccel4 = accData.getLastYAcceleration();
 zAccel4 = accData.getLastZAcceleration();
 xAccel40=(short) (xAccel4-XzeroOffset);
 yAccel40=(short) (yAccel4-XzeroOffset);
 zAccel40=(short) (zAccel4-XzeroOffset);
 logfile1.writeStringToLog(realTime4);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Time4);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(xAccel40);
```

```
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(yAccel40);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(zAccel40);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Rate4);
        logfile1.writeStringToLog("\n");


try {
                 Thread.sleep(100);
                } catch (InterruptedException e1) {
                 e1.printStackTrace();
                }
                accData = _accChannel.getAccelerometerData();
                realTime5 = System.currentTimeMillis();
                Time5=(float)(realTime5-appStartTime1)/1000;
        Rate5=(1/(Time5-Time4));
        xAccel5 = accData.getLastXAcceleration();
        yAccel5 = accData.getLastYAcceleration();
        zAccel5 = accData.getLastZAcceleration();
        xAccel50=(short) (xAccel5-XzeroOffset);
        yAccel50=(short) (yAccel5-XzeroOffset);
        zAccel50=(short) (zAccel5-XzeroOffset);
        logfile1.writeStringToLog(realTime5);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Time5);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(xAccel50);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(yAccel50);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(zAccel50);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Rate5);
        logfile1.writeStringToLog("\n");
try {
                 Thread.sleep(100);
                } catch (InterruptedException e1) {
                 e1.printStackTrace();
                }
                accData = _accChannel.getAccelerometerData();
                realTime6 = System.currentTimeMillis();
        Time6=(float)(realTime6-appStartTime1)/1000;
Rate6=(1/(Time6-Time5));
        xAccel6 = accData.getLastXAcceleration();
```

```
        yAccel6 = accData.getLastYAcceleration();
        zAccel6 = accData.getLastZAcceleration();
        xAccel60=(short) (xAccel6-XzeroOffset);
        yAccel60=(short) (yAccel6-XzeroOffset);
        zAccel60=(short) (zAccel6-XzeroOffset);
        logfile1.writeStringToLog(realTime6);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Time6);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(xAccel60);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(yAccel60);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(zAccel60);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Rate6);
        logfile1.writeStringToLog("\n");

try {
                 Thread.sleep(100);
                } catch (InterruptedException e1) {
                 e1.printStackTrace();
                }
                accData = _accChannel.getAccelerometerData();
                realTime7 = System.currentTimeMillis();
Time7=(float)(realTime7-appStartTime1)/1000;
Rate7=(1/(Time7-Time6));
        xAccel7 = accData.getLastXAcceleration();
        yAccel7 = accData.getLastYAcceleration();
        zAccel7 = accData.getLastZAcceleration();
        xAccel70=(short) (xAccel7-XzeroOffset);
        yAccel70=(short) (yAccel7-XzeroOffset);
        zAccel70=(short) (zAccel7-XzeroOffset);
        logfile1.writeStringToLog(realTime7);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Time7);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(xAccel70);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(yAccel70);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(zAccel70);
        logfile1.writeStringToLog(",");
        logfile1.writeStringToLog(Rate7);
                logfile1.writeStringToLog("\n");
try {
```

```
            Thread.sleep(100);
           } catch (InterruptedException e1) {
            e1.printStackTrace();
           }
           accData = _accChannel.getAccelerometerData();
           realTime8 = System.currentTimeMillis();
Time8=(float)(realTime8-appStartTime1)/1000;
 Rate8=(1/(Time8-Time7));
xAccel8 = accData.getLastXAcceleration();
yAccel8 = accData.getLastYAcceleration();
 zAccel8 = accData.getLastZAcceleration();
xAccel80=(short) (xAccel8-XzeroOffset);
yAccel80=(short) (yAccel8-XzeroOffset);
zAccel80=(short) (zAccel8-XzeroOffset);
logfile1.writeStringToLog(realTime8);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Time8);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(xAccel80);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(yAccel80);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(zAccel80);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(Rate8);
PrevTime=Time8;

if (flagGPS){
        gpsLong=gps.getLongitude();
        gpsLat=gps.getLatitude();
        gpsAlt=gps.getAltitude();
        gpsHead=gps.getCourse();
        gpsSpeed=(float) gps.getGPSspeed();
}
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(gpsLong);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(gpsLat);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(gpsAlt);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(gpsHead);
logfile1.writeStringToLog(",");
logfile1.writeStringToLog(gpsSpeed);
  //Get BatteryLevel & signal intensity
BattLevel = DeviceInfo.getBatteryLevel();
```

```
                BattTemp = DeviceInfo.getBatteryTemperature();
                BattVol = DeviceInfo.getBatteryVoltage();
                Radio1= RadioInfo.getSignalLevel();
                GPRS = GPRSInfo.getCellInfo().getRSSI();
                Wireless1=WLANInfo.getWLANState();
                logfile1.writeStringToLog(",");
                logfile1.writeStringToLog(BattLevel);
                logfile1.writeStringToLog(",");
                logfile1.writeStringToLog(BattTemp);
                logfile1.writeStringToLog(",");
                logfile1.writeStringToLog(BattVol);
                logfile1.writeStringToLog(",");
                logfile1.writeStringToLog(Radio1);
                logfile1.writeStringToLog(",");
                logfile1.writeStringToLog(GPRS);
                logfile1.writeStringToLog(",");
                logfile1.writeStringToLog(Wireless1);
                logfile1.writeStringToLog("\n");
                if (BattTemp > 38){
  _writingRawDataThread._running = false;
  _writingRawDataThread.notifyAll();
  _writingRawDataThread = null;
  logfile1.writeStringToLog("emergency stop because of temp\n");
  video.closeVideo_full();
                }
    }//end of while
  }//enf of run
  public void stop(){
                        _running=false;
     }
 }//end of thread
/**
 * A thread class to handle screen updates.
 */
private class WriteToFileFeatureDataThread extends Thread{
  private boolean _runningFeature;

  public void run(){
_runningFeature = true;
featurefile1=new FeatureFile();
feature_frequency_count=(feature_frequency -1 );
appStartTime2=System.currentTimeMillis();
featurefile1.writeStringToLog("start thread,,,,,,,,,Window size,");
featurefile1.writeStringToLog(feature_frequency);
featurefile1.writeStringToLog(",");
featurefile1.writeStringToLog("Hz");
```

```
  featurefile1.writeStringToLog(",");
  featurefile1.writeStringToLog(feature_frequency_count);
  featurefile1.writeStringToLog("\n");
  featurefile1.writeStringToLog("startFeatureTime,");
  featurefile1.writeStringToLog(appStartTime2);
  featurefile1.writeStringToLog("\n");
featurefile1.writeStringToLog("RealTime,Time,SumofGPSspeed,MaxX,MaxY,MaxZ
,MinX,MinY,MinZ,RangeX,RangeY,RangeZ,RangeXZ,AVEX,AVEY,AVEZ,STDX,
STDY,STDZ,TiltAngZY,SumofRange,DiffSR,SMASR,LyorSt,StorDy,Movestatic,D
ownStair,UpStair,Ramp,Elevator,CarRide,State,ChangeofState,ChangeofState2,Chan
geofState3,ExpectAct,TakeVideo,TakeVideoCount,CarStopCount,CarRideCount\n");
     while( _runningFeature ){
      try {
                        Thread.sleep(100);
                    } catch (InterruptedException e1) {
                        e1.printStackTrace();
                }
      accData = _accChannel.getAccelerometerData();
      xAccel = accData.getLastXAcceleration();
      yAccel = accData.getLastYAcceleration();
      zAccel = accData.getLastZAcceleration();
      //Origianl calibration
      xAccel0=(short) (xAccel-XzeroOffset);
      yAccel0=(short) (yAccel-XzeroOffset);
      zAccel0=(short) (zAccel-XzeroOffset);
      try {
                Thread.sleep(100);
                      } catch (InterruptedException e1) {
                       e1.printStackTrace();
                      }
               accData = _accChannel.getAccelerometerData();
      xAccel2 = accData.getLastXAcceleration();
      yAccel2 = accData.getLastYAcceleration();
      zAccel2 = accData.getLastZAcceleration();
      xAccel20=(short) (xAccel2-XzeroOffset);
      yAccel20=(short) (yAccel2-XzeroOffset);
      zAccel20=(short) (zAccel2-XzeroOffset);
      try {
                Thread.sleep(100);
               } catch (InterruptedException e1) {
                e1.printStackTrace();
               }
               accData = _accChannel.getAccelerometerData();
      xAccel3 = accData.getLastXAcceleration();
      yAccel3 = accData.getLastYAcceleration();
      zAccel3 = accData.getLastZAcceleration();
```

```
xAccel30=(short) (xAccel3-XzeroOffset);
yAccel30=(short) (yAccel3-XzeroOffset);
zAccel30=(short) (zAccel3-XzeroOffset);
try {
          Thread.sleep(100);
         } catch (InterruptedException e1) {
          e1.printStackTrace();
         }
         accData = _accChannel.getAccelerometerData();
xAccel4 = accData.getLastXAcceleration();
yAccel4 = accData.getLastYAcceleration();
zAccel4 = accData.getLastZAcceleration();
xAccel40=(short) (xAccel4-XzeroOffset);
yAccel40=(short) (yAccel4-XzeroOffset);
zAccel40=(short) (zAccel4-XzeroOffset);
try {
          Thread.sleep(100);
         } catch (InterruptedException e1) {
          e1.printStackTrace();
         }
         accData = _accChannel.getAccelerometerData();
xAccel5 = accData.getLastXAcceleration();
yAccel5 = accData.getLastYAcceleration();
zAccel5 = accData.getLastZAcceleration();
xAccel50=(short) (xAccel5-XzeroOffset);
yAccel50=(short) (yAccel5-XzeroOffset);
zAccel50=(short) (zAccel5-XzeroOffset);
try {
          Thread.sleep(100);
         } catch (InterruptedException e1) {
          e1.printStackTrace();
         }
         accData = _accChannel.getAccelerometerData();
xAccel6 = accData.getLastXAcceleration();
yAccel6 = accData.getLastYAcceleration();
zAccel6 = accData.getLastZAcceleration();
xAccel60=(short) (xAccel6-XzeroOffset);
yAccel60=(short) (yAccel6-XzeroOffset);
zAccel60=(short) (zAccel6-XzeroOffset);
try {
          Thread.sleep(100);
         } catch (InterruptedException e1) {
          e1.printStackTrace();
         }
         accData = _accChannel.getAccelerometerData();
```

```
            xAccel7 = accData.getLastXAcceleration();
        yAccel7 = accData.getLastYAcceleration();
        zAccel7 = accData.getLastZAcceleration();
        xAccel70=(short) (xAccel7-XzeroOffset);
        yAccel70=(short) (yAccel7-XzeroOffset);
        zAccel70=(short) (zAccel7-XzeroOffset);
        try {
                 Thread.sleep(100);
                } catch (InterruptedException e1) {
                 e1.printStackTrace();
                }
                accData = _accChannel.getAccelerometerData();
        xAccel8 = accData.getLastXAcceleration();
        yAccel8 = accData.getLastYAcceleration();
        zAccel8 = accData.getLastZAcceleration();
        xAccel80=(short) (xAccel8-XzeroOffset);
        yAccel80=(short) (yAccel8-XzeroOffset);
        zAccel80=(short) (zAccel8-XzeroOffset);
        try {
                 Thread.sleep(100);
                } catch (InterruptedException e1) {
                 e1.printStackTrace();
                }
        if (flagGPS){
         gpsSpeed=(float) gps.getGPSspeed();
        }
                feature_realTime = System.currentTimeMillis();
        FeatureTime=(feature_realTime-appStartTime2)/1000;
        xAccelMax=(short)
Math.max((Math.max(xAccel70,(Math.max(xAccel50,(Math.max(xAccel0,xAccel20)
)))),(Math.max(xAccel80,(Math.max(xAccel60,(Math.max(xAccel30,xAccel40)))))));
        yAccelMax=(short)
Math.max((Math.max(yAccel70,(Math.max(yAccel50,(Math.max(yAccel0,yAccel20)
)))),(Math.max(yAccel80,(Math.max(yAccel60,(Math.max(yAccel30,yAccel40)))))));
        zAccelMax=(short)
Math.max((Math.max(zAccel70,(Math.max(zAccel50,(Math.max(zAccel0,zAccel20))
))),(Math.max(zAccel80,(Math.max(zAccel60,(Math.max(zAccel30,zAccel40)))))));
        xAccelMin=(short)
Math.min((Math.min(xAccel70,(Math.min(xAccel50,(Math.min(xAccel0,xAccel20)))
))),(Math.min(xAccel80,(Math.min(xAccel60,(Math.min(xAccel30,xAccel40)))))));
        yAccelMin=(short)
Math.min((Math.min(yAccel70,(Math.min(yAccel50,(Math.min(yAccel0,yAccel20)))
))),(Math.min(yAccel80,(Math.min(yAccel60,(Math.min(yAccel30,yAccel40)))))));
        zAccelMin=(short)
Math.min((Math.min(zAccel70,(Math.min(zAccel50,(Math.min(zAccel0,zAccel20))))
)),(Math.min(zAccel80,(Math.min(zAccel60,(Math.min(zAccel30,zAccel40)))))));
```

```
        RangeX=(short) (xAccelMax-xAccelMin);
        RangeY=(short) (yAccelMax-yAccelMin);
        RangeZ=(short) (zAccelMax-zAccelMin);
        RangeXZ=(short) (RangeX+RangeZ);
MeanX=(float)((xAccel0+xAccel20+xAccel30+xAccel40+xAccel50+xAccel60+xAc
cel70+xAccel80)/feature_frequency_count);
MeanY=(float)((yAccel0+yAccel20+yAccel30+yAccel40+yAccel50+yAccel60+yAc
cel70+yAccel80)/feature_frequency_count);
MeanZ=(float)((zAccel0+zAccel20+zAccel30+zAccel40+zAccel50+zAccel60+zAcc
el70+zAccel80)/feature_frequency_count);
varianceX=((((xAccel0-MeanX)*(xAccel0-MeanX))+((xAccel20-
MeanX)*(xAccel20-MeanX))+((xAccel30-MeanX)*(xAccel30-
MeanX))+((xAccel40-MeanX)*(xAccel40-MeanX))+((xAccel50-
MeanX)*(xAccel50-MeanX))+((xAccel60-MeanX)*(xAccel60-
MeanX))+((xAccel70-MeanX)*(xAccel70-MeanX))+((xAccel80-
MeanX)*(xAccel80-MeanX)))/feature_frequency_count);
STDX=Math.sqrt((double)varianceX);
varianceY=((((yAccel0-MeanY)*(yAccel0-MeanY))+((yAccel20-
MeanY)*(yAccel20-MeanY))+((yAccel30-MeanY)*(yAccel30-
MeanY))+((yAccel40-MeanY)*(yAccel40-MeanY))+((yAccel50-
MeanY)*(yAccel50-MeanY))+((yAccel60-MeanY)*(yAccel60-
MeanY))+((yAccel70-MeanY)*(yAccel70-MeanY))+((yAccel80-
MeanY)*(yAccel80-MeanY)))/feature_frequency_count);
STDY=Math.sqrt((double)varianceY);
varianceZ=((((zAccel0-MeanZ)*(zAccel0-MeanZ))+((zAccel20-MeanZ)*(zAccel20-
MeanZ))+((zAccel30-MeanZ)*(zAccel30-MeanZ))+((zAccel40-MeanZ)*(zAccel40-
MeanZ))+((zAccel50-MeanZ)*(zAccel50-MeanZ))+((zAccel60-MeanZ)*(zAccel60-
MeanZ))+((zAccel70-MeanZ)*(zAccel70-MeanZ))+((zAccel80-MeanZ)*(zAccel80-
MeanZ)))/feature_frequency_count);
STDZ=Math.sqrt((double)varianceZ);
TiltAngZY=TiltAnglezy2(MeanZ,MeanY);
SumofRange=(short) (RangeX+RangeY+RangeZ);
            featurefile1.writeStringToLog(feature_realTime);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(FeatureTime);
        featurefile1.writeStringToLog(",");
        SumofGPSSpeed=(prevgpsSpeed+gpsSpeed);
            prevgpsSpeed=gpsSpeed;
        featurefile1.writeStringToLog(SumofGPSSpeed);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(xAccelMax);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(yAccelMax);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(zAccelMax);
        featurefile1.writeStringToLog(",");
```

```
        featurefile1.writeStringToLog(xAccelMin);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(yAccelMin);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(zAccelMin);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(RangeX);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(RangeY);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(RangeZ);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(RangeXZ);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(MeanX);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(MeanY);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(MeanZ);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(STDX);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(STDY);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(STDZ);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(TiltAngZY);
        featurefile1.writeStringToLog(",");
        featurefile1.writeStringToLog(SumofRange);
                DiffSR=(short)(SumofRange-prevSR);
SMASR=(short)(SumofRange+prevSR+prevSR2+prevSR3+prevSR4+prevSR5+prev
SR6+prevSR7);
                prevSR7=prevSR6;
                prevSR6=prevSR5;
                prevSR5=prevSR4;
                prevSR4=prevSR3;
                prevSR3=prevSR2;
                prevSR2=prevSR;
                prevSR=SumofRange;
      featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(DiffSR);
      featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(SMASR);
      ClassificationTree();
      }//end of while
    }//enf of run
```

```
    public void stop(){
                    _runningFeature = false;
      }
    }//end of thread
      public void ClassificationTree()
      {
            //Lie, Stand, or Sit**********************************
            if (TiltAngZY > Thr_st_l && TiltAngZY < Thr_st_h){
            //Sitting state
                  ststate=32;
            }
            else if (TiltAngZY > Thr_ly_l && TiltAngZY < Thr_ly_h){
                  ststate=16; //Lying state
            }
            else {
                  ststate=8; //Standing state
            }
      featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(ststate);
      prevststate=ststate;
            //Static or Dynamic*****************************
    if (STDX > Dy_th_1 || STDY > Dy_th_1 || STDZ > Dy_th_1 || ( SumofRange >
Dy_th_2 && RangeY > Dy_th_3 )){
      sdstate = 1;
    }
    else if ( STDY < St_th){
      sdstate = 0;
    }
    else{
      sdstate = prevsdstate;
    }
      featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(sdstate);
      prevsdstate = sdstate;
    //Moving under static state*****************************
    if(sdstate==0 && SumofRange > Movestatic_th_h)
      moveststate=2048;
    else if (sdstate==0 && SumofRange > Movestatic_th_l)
      moveststate=prevmoveststate;
    else{
      moveststate=0;
    }
      featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(moveststate);
    //Downstairs*************************************************
```

```
   if (RangeY > DownStair_th_1 || SumofRange > DownStair_th_2 || SMASR >
DownStair_th_3){
     dstate = 64;
   }
   else if ((DiffSR < Not_DownStair_th_1 && SumofRange < Not_DownStair_th_2
&& SMASR < Not_DownStair_th_6) || (SumofRange < Not_DownStair_th_4) ||
((SumofRange < Not_DownStair_th_5 && SMASR < Not_DownStair_th_6 &&
(RangeXZ < Not_DownStair_th_7 || RangeY < Not_DownStair_th_8)))){
     dstate = 0;
   }
   else{
     dstate = prevdstate;    }
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(dstate);
     prevdstate = dstate;
   //Upstairs**************************************************
   if (RangeY < Not_DownStair_th_8 || (SMASR < Not_DownStair_th_6 &&
RangeXZ < Upstair_th_2)){
     ustate = 0;
   }
   else if(SMASR > Upstair_th_3){
     ustate =64;
   }
   else{
     ustate = prevustate;
   }
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(ustate);
     prevustate = ustate;
   //Ramp******************************************************
   if (RangeY < Ramp_th_3){
     rstate =0;
   }
   else if(RangeY > Ramp_th_1 || SumofRange > Ramp_th_2){
     rstate = 64;
   }
   else{
     rstate = prevrstate;
   }
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(rstate);
     prevrstate = rstate;

   //Elevator**************************************************
   if (STDX < Elevator_th_1 && RangeY < Elevator_th_2 && STDZ <
Elevator_th_1 && MeanY > Elevator_th_3 && MeanY < Elevator_th_4 &&
```

```
SumofRange < Dy_th_3 && ( STDY > Elevator_th_5 || ( SumofRange > Dy_th_1
&& RangeY > Elevator_th_6))){
     estate = 256;
   }
   else if(SumofRange < Elevator_th_6){
     estate =0;
   }
   else{
     estate = prevestate;
   }
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(estate);
     prevestate = estate;
   //Car ride start/stop**************************************
   if (SumofGPSSpeed > Thr_car_1 ){
     cstate=512;
     CarStopCount=0;
     CarRideCount++;
   }else if(SumofGPSSpeed < Thr_car_1 && CarStopCount > Car_stop_count){
     CarStopCount++;
     cstate = 0;
     CarRideCount=0;
   }else{
     CarStopCount++;
     cstate=prevcstate;
   }
     featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(cstate);
     prevcstate = cstate;
   //The summation of states*****************************
state=(short)(ststate+sdstate+moveststate+dstate+ustate+rstate+estate+cstate);
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(state);
   //Change of State*********************************
              changeOfState=(short) (state-prevState);
              changeOfState2=(short) (state-prevState2);
              changeOfState3=(short) (state-prevState3);
              prevState3=prevState2;
              prevState2=prevState;
              prevState=state;
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(changeOfState);
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(changeOfState2);
              featurefile1.writeStringToLog(",");
     featurefile1.writeStringToLog(changeOfState3);
```

```
    //Expected Activities************************************
    if (sdstate==0 && moveststate == 0){
      ExpectedAct=(short) (ststate+estate+cstate);
    }
    else if( prevmoveststate2 > 0 && prevmoveststate > 0 && moveststate > 0 &&
sdstate==0 && prevststate == ststate){
      ExpectedAct=(short)(moveststate+ststate);
    }
    else if((dstate+ustate+rstate)==0 && sdstate > 0 && ststate > 8 ){
      ExpectedAct=1024;
    }
    else if(sdstate == 1 ){
      ExpectedAct=(short) (dstate+ustate+rstate+48);
    }else{
      ExpectedAct = PrevExpectedAct;
    }
                featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(ExpectedAct);
      prevmoveststate2=prevmoveststate;
      prevmoveststate = moveststate;
      PrevExpectedAct=ExpectedAct;
    //Take Video Count*****************************************
    if (((changeOfState > 0 && changeOfState2 > 0 && changeOfState3 > 0 ) ||
(changeOfState < 0 && changeOfState2 < 0 && changeOfState3 < 0 )) &&
prevTakeVideo == 0 && TakeVideoCount>0 && CarRideCount < 2){
       TakeVideo=1;
        }else{
                TakeVideo=0;
        }
                featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(TakeVideo);
      prevTakeVideo=TakeVideo;
    if(TakeVideo==0){
      TakeVideoCount++;
    }
    if (TakeVideo==1 && flagVideo){
                TakeVideoCount=0;
                video.startRecording_test(configuration);
       }
                featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(TakeVideoCount);
                featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(CarStopCount);
                featurefile1.writeStringToLog(",");
      featurefile1.writeStringToLog(CarRideCount);
                featurefile1.writeStringToLog("\n");
```

```
    }
  public void closeVideoFull(){
            if (video!=null){
                    video.closeVideo_full();
                    video=null;
            }
    }
  public void stopVideo(){
                    video.stopRecording();
  }
  public void startVideo(){
    video=new BBVideo();
  }
  public void startGPS(){
    if (gps==null){
            gps=new GPSdata();
            gps.start();
    }
  }
  public void closeGPS(){
    if (gps!=null){
            gps.stop();
            gps=null;
    }
  }
  private void stopThreads()
  {
    if (flagRawData){
            if (_writingRawDataThread != null) {
                    _writingRawDataThread.stop(); //Stop the Thread
            }
    }
    if (flagFeatureData){
            if (_writingFeatureDataThread != null) {
                    _writingFeatureDataThread.stop(); //Stop the Thread
            }
    }
    _writingFeatureDataThread = null;
   _writingRawDataThread =null;
  }
}


package com.rim.samples.device.accelerometerdemo;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
```

```
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
public class LogFile {
    private OutputStreamWriter logFile;
    private OutputStream output1;
    private FileConnection fconn1;
    public LogFile() {
            createLogFile();
    }
 private void createLogFile()
 {
    int _logcounter=0;
    String
FileNamelog=System.getProperty("fileconn.dir.memorycard")+"BlackBerry/docume
nts/";
            try{
                    if (fconn1==null){
                            fconn1 = (FileConnection) Connector.open(FileNamelog+
"Log"+ _logcounter + ".txt");
                    while (fconn1.exists()) {
                            fconn1.close();
                            ++_logcounter;
                            fconn1 = (FileConnection) Connector.open(FileNamelog+
"Log"+_logcounter+ ".txt");
                    }
                    fconn1.create();
                    output1 = fconn1.openOutputStream(999999999);
                    logFile= new OutputStreamWriter(output1);
                    System.out.println("File created");
                    }
            }
            catch(IOException ioFileex)
            {
                    //Catch and re-throw the exception.
                    throw new RuntimeException(ioFileex.toString());
      }
 }
 public void closeLogFile()
 {
    try{
                    logFile.write("Log File was closed properly\n");
                    logFile.flush();
                    output1.close();
                    logFile.close();
                    fconn1.close();
    }
```

```
        catch (IOException ioex) {
         throw new RuntimeException(ioex.toString());
        }
    }
    public void writeStringToLog(String logString)
    {
        try{
                logFile.write(logString);
        }
        catch(IOException ioFileex)
        {
                throw new RuntimeException(ioFileex.toString());
        }
    }
    public void writeStringToLog(int logInt)
    {
        try{
                logFile.write(Integer.toString(logInt));
        }
        catch(IOException ioFileex)
        {
                throw new RuntimeException(ioFileex.toString());
        }
    }
    public void writeStringToLog(double logfloat)
    {
        String temp=Double.toString(logfloat);
        try{
                logFile.write(temp);
        }
        catch(IOException ioFileex)
        {
                throw new RuntimeException(ioFileex.toString());
        }
    }
}


package com.rim.samples.device.accelerometerdemo;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
public class FeatureFile {
    private OutputStreamWriter logFile1;
```

```
    private OutputStream output2;
    private FileConnection fconn2;
    public FeatureFile() {
            createLogFile();
    }
  private void createLogFile()
  {
    int _logcounter=0;
    String
FileNamelog=System.getProperty("fileconn.dir.memorycard")+"BlackBerry/docume
nts/";
            try{
                    if (fconn2==null){
                      fconn2 = (FileConnection) Connector.open(FileNamelog+
"Feature"+ _logcounter + ".txt");                                    while
(fconn2.exists()) {
                     fconn2.close();
                     ++_logcounter;
                     fconn2 = (FileConnection) Connector.open(FileNamelog+
"Feature"+_logcounter+ ".txt");
                    }
                    fconn2.create();
                    output2 = fconn2.openOutputStream(999999999);
                    logFile1= new OutputStreamWriter(output2);
                    System.out.println("File created");
                    }
            }
            catch(IOException ioFileex)
            {
                    throw new RuntimeException(ioFileex.toString());
                    }
  }
  public void closeFeatureFile()
  {
    try{
                    logFile1.write("Log File was closed properly\n");
                    logFile1.flush();
                    output2.close();
                    logFile1.close();
                    fconn2.close();

    }
            catch (IOException ioex) {
                    throw new RuntimeException(ioex.toString());
            }
  }
```

```
  public void writeStringToLog(String logString)
  {
     try{
              logFile1.write(logString);

     }
     catch(IOException ioFileex)
     {
              throw new RuntimeException(ioFileex.toString());
     }
  }
  public void writeStringToLog(double logfloat)
  {
     String temp=Double.toString(logfloat);
     try{
              logFile1.write(temp);
     }
     catch(IOException ioFileex)
     {
              throw new RuntimeException(ioFileex.toString());
     }
  }
}



package com.rim.samples.device.accelerometerdemo;
import java.io.OutputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
import javax.microedition.media.Player;
import javax.microedition.media.control.RecordControl;
import javax.microedition.media.control.VideoControl;
import net.rim.device.api.system.Backlight;
import net.rim.device.api.ui.Field;
public class BBVideo{
       /**
       * A screen on which to display the Video image.
       */
        String _encoding = "encoding=video/3gpp&mode=standard";
             private Player _player;
        private VideoControl _vc;
        private RecordControl _rc;
        private boolean _locationSet = false;
        private OutputStream _out;
        private FileConnection _fc;
        public Field videoField;
```

```
        public VideoTime videoThread=new VideoTime();
        private Configuration configuration; //=new Configuration();
        int takingvideotime;

       /*
       * constructor passed the chosen video encoding property value
       */
          public BBVideo()
          {
           try {
//create a video media player to capture video                    _player =
javax.microedition.media.Manager.createPlayer( "capture://video?" + _encoding );
     // try to start the player
       _player.start();
                  _vc = (VideoControl)_player.getControl( "VideoControl" );
                  _rc = (RecordControl)_player.getControl( "RecordControl" );
videoField = (Field)_vc.initDisplayMode( VideoControl.USE_GUI_PRIMITIVE,
"net.rim.device.api.ui.Field" );
           } catch ( final Exception e ) {
      System.out.println( "Exception in VideoScreen constructor" );
          }
         }
                    public void startThread(){
                      videoThread.start();
              }
              public class VideoTime extends Thread {
                        boolean takingVideo;
                        public VideoTime(){
              takingVideo=true;
         }
         public void run()
         {
           while (takingVideo){
            startRecord();
              }
         }
         public void stopTakingVideo(){
                              takingVideo=false;
              }
       }
             private void startRecord(){
            if( !_locationSet )
            {
               try {
                        int _logcounter=0;
```

```
String
FileNamelog=System.getProperty("fileconn.dir.memorycard")+"BlackBerry/videos/";
        if (_fc ==null){
          _fc = (FileConnection) Connector.open(FileNamelog+ "BB_video"+
_logcounter + ".sbv");
        while (_fc.exists()) {
           _fc.close();
           ++_logcounter;
_fc = (FileConnection) Connector.open(FileNamelog+ "BB_video"+_logcounter+
".sbv");
         }
           _fc.create();
         }
           _fc.truncate( 0 );
    // ready to write video stream
           _out = _fc.openOutputStream();
           } catch ( Exception e ) {
               return;
           }
         _rc.setRecordStream( _out );
         _locationSet = true;
             }
              }
          public void startRecording()
          {
           //save battery and for video recording
               Backlight.enable(true, 10);
           _rc.startRecord();
          }
          public void stopRecording()
          {
           _rc.stopRecord();
          }
          public void startRecording_test(Configuration config)
          {
           Backlight.enable(true, 10);
           configuration=config;
         takingvideotime=configuration.getTakingVideoTime();
           _rc.startRecord();
           try {
    Thread.sleep(takingvideotime);//startRcord + Sleep --> cause accelerometer to
stop issue.
                        } catch (InterruptedException e) {
                               e.printStackTrace();
                        }
                                    _rc.stopRecord();
```

```
        }
        public void closeVideo_full(){
                        commit();
                }
        private void commit()
        {
         videoThread.stopTakingVideo();
          try {
            _rc.commit();
            _locationSet = false;
              try {
                _out.close();
                _fc.close();
              } catch ( Exception e ) {
              }
          } catch ( Exception e ) {
          }
        }
}
```

```
package com.rim.samples.device.accelerometerdemo;
import javax.microedition.location.*;
public class GPSdata implements LocationListener {
  LocationProvider lp;
  Criteria criteria=new Criteria();
  private double lng;
  private double lat;
  private float alt;
  private double head;
  private float speed;
  public String longitude;
  public String latitude;
  public String altitude;
  public String heading;
  public String gpsSpeed;
  private Location location;
  public GPSdata() {
    lp = null;
    criteria.setHorizontalAccuracy(10);
    criteria.setVerticalAccuracy(10);
    criteria.setCostAllowed(false);
  }
  public boolean isConnected() {
    return (lp != null);
```

```
}
public void start() {
  if (lp == null) {
    try {
      lp = LocationProvider.getInstance(null);
      if(lp != null) {
        lp.setLocationListener(this, 9, -1, -1);
      }
     }
     catch(LocationException e) {
     }
  }
}
public void stop() {
  if (lp != null) {
    lp.setLocationListener(null, -1, -1, -1);
    lp.reset();
    lp = null;
  }
 }
 public void locationUpdated(LocationProvider provider, Location gpslocation) {
   location=gpslocation;
   if(location.isValid()) {
    lng = location.getQualifiedCoordinates().getLongitude();
    lat = location.getQualifiedCoordinates().getLatitude();
    alt = location.getQualifiedCoordinates().getAltitude();
    head = location.getCourse();
    speed=location.getSpeed();
  }
  else{
     lng = 0;
     lat = 0;
     alt = 0;
     head = 0;
     speed=0;
    longitude=null;
    latitude=null;
    altitude=null;
    heading=null;
    gpsSpeed=null;
  }
 }
 public void providerStateChanged(LocationProvider provider, int newState) {
        //Nothing    }}
 }
 public double getLongitude()
```

```
    {
        return lng;
    }
    public double getLatitude()
    {
        return lat;
    }
    public float getAltitude()
    {
        return alt;
    }
    public double getCourse()
    {
        return head;
    }
    public float getGPSspeed()
    {
        return speed;
    }
}
```

Appendix D: Detailed Experimental Results

Table D-1: CoS results for the WMMS algorithm. TP = true positives, FN = false negatives, FP = false positives, and TN = true negatives.

Changes-of-state	Trial 1				Trial 2				Trial 3			
	TP	**FN**	**FP**	**TN**	**TP**	**FN**	**FP**	**TN**	**TP**	**FN**	**FP**	**TN**
Standing	14	0	0	2256	18	0	1	2136	16	0	0	2700
Sitting	6	0	0	739	5	0	1	484	5	0	0	421
Lying	3	0	0	498	3	0	0	507	3	0	0	466
Walk	20	0	14	2127	22	1	18	2302	20	0	20	2422
Elevator	0	2	2	397	0	2	2	422	0	2	2	450
Upstairs	3	0	7	489	2	1	6	437	1	2	5	426
Downstairs	2	1	7	419	2	1	5	392	3	0	7	372
Ramp	1	2	9	412	2	1	6	375	0	3	7	389
Car ride start/stop	2	0	0	1049	2	0	0	977	2	0	0	976
Total	51	5	39	8386	56	6	39	8032	50	7	41	8622

Table D-2: Sensitivity of CoS identification (Detail) for the last version of new WMMS. The sequence is followed by test procedure in 3 trails for each subject. TP = true positives, FN = false negatives, FP = false positives, TN = true negatives, and SE = sensitivity.

Changes-of-state	Subject 1		Subject 2		Subject 3		Subject 4		Subject 5		SE %
	TP	FN	TP	FN	TP	FN	TP	FN	TP	FN	
Stand - Walk	6	0	8	0	7	0	4	0	9	0	100.0
Walk - Sit	3	0	3	0	2	1	3	0	3	0	93.3
Sit - Walk	3	0	3	0	3	0	3	0	3	0	100.0
Walk - Stand	8	0	7	0	6	0	6	0	6	0	100.0
Walk - Elevator	6	0	6	0	6	0	5	0	6	0	100.0
Elevator - Walk	6	0	5	0	6	0	6	0	5	1	96.5
Walk – Brush teeth	3	0	2	1	2	1	3	0	2	1	80.0
Brush teeth – Comb hair	0	3	0	3	1	2	0	3	1	2	13.3
Comb hair – Wash hands	0	3	0	3	1	2	1	2	0	3	13.3
Wash hands – Dry hands	0	3	0	3	2	1	1	2	1	2	26.7
Dry hands – Walk	3	0	2	1	2	1	3	0	3	0	86.7
Walk – Move dishes	2	1	2	1	1	2	1	2	0	3	40.0
Move dishes – Move kettle	3	0	2	1	1	2	2	1	3	0	73.3
Move kettle – Make toast	3	0	0	3	2	1	0	3	3	0	53.3
Make toast - Walk	3	0	3	0	2	1	3	0	3	0	93.3
Walk – Meal setup	2	1	2	1	2	1	3	0	2	1	73.3
Meal setup – Walk	3	0	3	0	3	0	3	0	2	1	93.3
Walk – Wash dishes	1	2	2	1	1	2	2	1	1	2	46.7
Wash dishes – Walk	3	0	2	1	3	0	3	0	2	1	86.7
Walk – Stairs	5	1	6	0	5	1	6	0	3	3	83.3
Stairs – Walk	3	3	5	1	3	3	6	0	2	4	63.3
Walk – Lie	3	0	3	0	3	0	3	0	3	0	100.0
Lie – Walk	3	0	3	0	3	0	3	0	3	0	100.0
Walk – Ramp	1	2	1	2	1	2	3	0	3	0	60.0
Ramp - Walk	1	2	3	0	3	0	3	0	1	2	73.3

Table D-3: Specificity of CoS identification (Detail) for the last version of new WMMS. TP = true positives, FN = false negatives, FP = false positives, TN = true negatives, and SP = specificity.

Subjects	Subject1		Subject2		Subject3		Subject4		Subject5		
Changes-of-state	FP	TN	FP	TN	FP	TN	FP	TN	FP	TN	SP %
Lie	0	57	0	48	0	40	0	41	0	80	100.0
Sit	1	61	0	47	0	34	0	23	0	73	99.6
Dry hands	0	15	1	12	0	17	0	15	0	17	98.7
Stand	0	137	5	171	8	147	5	65	1	297	97.7
Move dishes	0	22	1	14	0	9	1	9	0	6	96.8
Take an elevator	7	89	6	101	5	107	9	99	0	180	95.5
Wash dishes	3	37	1	28	1	27	0	48	3	24	95.4
Comb hair	0	28	1	38	2	28	1	23	3	23	95.2
Toast bread	3	31	0	11	0	15	1	22	1	12	94.8
Wash hands	0	20	2	36	1	12	1	13	2	17	94.2
Ramp	1	29	3	19	0	40	3	12	1	15	93.5
Brush teeth	2	34	4	36	3	32	1	25	3	18	91.8
Stairs	3	61	12	42	5	98	10	57	4	61	90.4
Prepare a meal	4	53	3	30	7	48	4	35	5	34	89.7
Move a kettle	1	24	2	14	5	24	4	17	0	11	88.2
Walk	74	499	75	563	84	667	55	434	36	213	88.0

Table D-4: Confusion matrix of activity result including accelerometer-only.

Accelerometer only	Predicted: Stand	Sit	Lie	Walk	Elevator	Stairs	Ramp	Dry hands	Move dishes	Wash dishes	Comb hair	Toast bread	Wash hands	Brush Teeth	Prepare meal	Move kettle
Actual: Stand	980	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0
Sit	0	293	0	6	0	0	0	0	0	0	0	0	0	0	0	0
Lie	0	0	280	10	0	0	0	0	0	0	0	0	0	0	0	0
Walk	54	16	11	2805	2	0	0	0	0	0	0	0	0	0	0	0
Elevator	525	0	0	34	17	0	0	0	0	0	0	0	0	0	0	0
Stairs	1	0	0	391	0	0	0	0	0	0	0	0	0	0	0	0
Ramp	0	0	0	146	0	0	0	0	0	0	0	0	0	0	0	0
Dry hands	28	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0
Move dishes	30	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
Wash dishes	43	0	0	13	4	0	0	0	0	0	0	0	0	0	0	0
Comb hair	36	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Toast bread	35	15	0	25	14	0	0	0	0	0	0	0	0	0	0	0
Wash hands	36	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
Brush Teeth	53	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0
Prepare meal	0	213	0	7	3	0	0	0	0	0	0	0	0	0	0	0
Move kettle	51	0	0	39	11	0	0	0	0	0	0	0	0	0	0	0

Table D-5: Confusion matrix of activity result including accelerometer with video.

Accelerometer + Video	Predicted: Stand	Sit	Lie	Walk	Elevator	Stairs	Ramp	Dry hands	Move dishes	Wash dishes	Comb hair	Toast bread	Wash hands	Brush Teeth	Prepare meal	Move kettle
Actual: Stand	996	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
Sit	0	280	0	21	0	0	0	0	0	0	0	0	0	0	0	0
Lie	0	0	290	0	0	0	0	0	0	0	0	0	0	0	0	0
Walk	0	0	0	2871	9	61	1	0	0	5	0	0	0	0	3	0
Elevator	0	0	0	0	622	0	0	0	0	0	0	0	0	0	0	0
Stairs	0	0	0	49	0	344	0	0	0	0	0	0	0	0	0	0
Ramp	0	0	0	76	0	0	70	0	0	0	0	0	0	0	0	0
Dry hands	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0
Move dishes	0	0	0	23	0	0	0	0	7	0	0	0	0	0	0	0
Wash dishes	0	0	0	57	0	0	0	0	0	62	0	0	0	0	0	0
Comb hair	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0
Toast bread	0	0	0	0	0	0	0	0	0	0	0	62	0	0	0	0
Wash hands	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0
Brush Teeth	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0
Prepare meal	0	0	0	10	0	0	0	0	0	0	0	0	0	0	52	0
Move kettle	0	0	0	2	0	0	0	0	2	0	0	0	0	0	0	70

Table D-6: Activity classification for accelerometer only. TP = true positives, FN = false negatives, FP = false positives, TN = true negatives, SE = sensitivity, and SP = specificity.

Activity	Subject1				Subject2				Subject3				Subject4				Subject5			
	TP	FN	FP	TN	TP	FN	FP	TN	TP	FN	FP	TN	TP	FN	FP	TN	TP	FN	FP	TN
Stand	165	4	12	1239	199	3	16	1369	193	6	13	1495	109	3	14	1059	320	5	4	1014
Walk	601	26	64	729	643	34	71	839	841	40	91	735	495	34	95	561	262	29	30	1022
Sit	64	1	5	1350	45	5	19	1518	48	2	5	1652	41	4	5	1135	76	0	11	1256
Elevator	6	96	9	1309	1	112	0	1474	5	113	9	1580	0	114	0	1071	0	186	7	1150
Stairs	24	46	300	1050	32	28	333	1194	13	96	214	1384	48	25	313	799	56	15	685	587
Lie	59	1	0	1360	49	2	0	1536	41	2	0	1664	39	5	0	1141	82	1	0	1260
Ramp	1	32	14	1373	3	22	15	1547	0	43	29	1635	10	8	24	1143	17	2	33	1291
Brush teeth	0	39	0	1381	0	47	0	1540	0	38	0	1669	0	33	0	1152	0	25	0	1318
Comb hair	0	31	0	1389	0	42	0	1545	0	33	0	1674	0	27	0	1158	0	29	0	1314
Wash hands	0	21	0	1399	0	41	0	1546	0	28	0	1679	0	24	0	1161	0	22	0	1321
Dry hands	0	18	0	1402	0	16	0	1571	0	20	0	1687	0	18	0	1167	0	20	0	1323
Move dishes	0	25	0	1395	0	18	0	1569	0	12	0	1695	0	13	0	1172	0	9	0	1334
Move kettle	0	28	0	1392	0	19	0	1568	0	32	0	1675	0	24	0	1161	0	14	0	1329
Make toast	0	37	0	1383	0	14	0	1573	0	18	0	1689	0	26	0	1159	0	16	0	1327
Meal setup	0	60	0	1360	0	36	0	1551	0	58	0	1649	0	42	0	1143	0	42	0	1301
Wash dishes	0	43	0	1377	0	32	0	1555	0	31	0	1676	0	51	0	1134	0	30	0	1313

Table D-7: Activity classification for accelerometer with video. TP = true positives, FN = false negatives, FP = false positives, TN = true negatives, SE = sensitivity, and SP = specificity.

Activity	Subject1				Subject2				Subject3				Subject4				Subject5			
	TP	FN	FP	TN	TP	FN	FP	TN	TP	FN	FP	TN	TP	FN	FP	TN	TP	FN	FP	TN
Stand	160	0	0	1260	183	0	9	1395	181	0	0	1526	100	0	0	1085	318	0	0	1025
Walk	501	15	32	872	546	23	75	943	710	39	100	858	441	0	56	688	204	18	72	1049
Sit	61	0	0	1359	47	0	0	1540	27	21	0	1659	42	0	0	1143	73	0	0	1270
Elevator	91	0	0	1329	101	0	2	1484	107	0	0	1600	99	0	0	1086	185	0	9	1149
Stairs	59	3	15	1343	42	0	11	1534	82	17	31	1577	57	0	0	1128	38	26	1	1278
Lie	57	0	0	1363	48	0	0	1539	40	0	0	1667	41	0	0	1144	80	0	0	1263
Ramp	9	22	0	1389	12	9	1	1565	12	30	0	1665	12	0	0	1173	15	0	1	1327
Brush teeth	0	34	0	1386	0	39	1	1547	0	33	0	1674	0	27	0	1158	0	21	0	1322
Comb hair	0	31	0	1389	0	41	0	1546	0	30	0	1677	0	26	0	1159	0	25	0	1318
Wash hands	0	13	0	1407	0	39	1	1547	6	20	0	1681	0	14	0	1171	0	20	0	1323
Dry hands	0	17	0	1403	0	15	0	1572	0	18	0	1689	5	11	0	1169	8	11	0	1324
Move dishes	0	23	0	1397	0	15	1	1571	0	11	0	1696	5	6	2	1172	0	9	0	1334
Move kettle	24	0	0	1396	0	15	0	1572	0	26	0	1681	0	18	0	1167	11	0	0	1332
Make toast	18	13	0	1389	0	14	0	1573	0	16	0	1691	0	25	0	1160	12	0	0	1331
Meal setup	0	54	0	1366	0	31	1	1555	0	49	0	1658	10	25	0	1150	33	2	3	1305
Wash dishes	21	18	0	1381	26	3	1	1557	0	29	0	1678	0	49	0	1136	8	18	4	1313
Bathroom	95	0	0	1325	112	22	2	1451	90	4	0	1613	48	35	0	1102	68	17	0	1258
Kitchen	114	3	0	1303	61	16	2	1508	54	28	0	1625	48	56	1	1080	31	30	2	1280
Dining	50	4	0	1366	25	6	1	1555	49	0	0	1658	35	0	0	1150	33	2	3	1305

Appendix E: Detailed Experimental Procedures

E.1 The detailed test procedure for the hardware test was:

1. From standing position for 30 seconds and then sit for 30 seconds.
2. Standing position for 30 seconds and then sit for 30 seconds.
3. Standing position for 30 seconds and then sit for 30 seconds.
4. Walk 5 meters towards a bed.
5. Standing position for 30 seconds.
6. Lie on the bed for 30 seconds and then stand up for 30 seconds.
7. Lie on the bed for 30 seconds and then stand up for 30 seconds.
8. Lie on the bed for 30 seconds and then stand up for 30 seconds.
9. Walk 25 meters and then stand for 30 seconds.

E.2 The detailed test procedure for the single subject was:

1. From standing position, shake Smartphone for 2 seconds.
2. Walk for 25 meters and then sit for 30 seconds.
3. Stand up & walk 60 meters until an elevator is reached
4. Stand and wait for the elevator and then walk into the elevator
5. Turn around and then take the elevator to the second floor
6. Walk 20 m and then turn around
7. Walk 20 m toward elevator and then stand and wait for the elevator
8. Walk into the elevator and then turn around
9. Take the elevator to the first floor
10. Walk 50 meters towards the stairwell
11. Open door and enter stairwell
12. Walk up stairs (13 steps, landing, 13 steps)

13. Walk toward the exit door
14. Open door and turn right and then walk 15 meters
15. Turn around and then walk 15 meters towards the stairwell
16. Open door and enter stairwell
17. Walk down stairs (13 steps, landing, 13 steps)
18. Walk toward the exit door and then open door and turn right
19. Walk 20 meters inside the Rehab Technology Lab toward a bed
20. Lie on the bed for 30 seconds and then stand up
21. Walk 10 meters and then turn right towards a ramp
22. Walk up 10 meters on the ramp and then turn around
23. Walk down 10 meters on the ramp
24. Walk 30 meters to the starting point
25. Stand for 30 seconds, and shake the Smartphone for 2 seconds.

E.3 The detailed test procedure for the five able-bodied subjects was:

1. From a standing position, shake the Smartphone for 2 seconds
2. Walk for 25 meters and then sit for 20 seconds
3. Stand up & walk 60 meters
4. Stand and wait for the elevator and then walk into the elevator
5. Turn around and then take the elevator to the second floor
6. Walk 2 m to open door and then walk 4 m and enter the bathroom
7. At the sink, simulate brushing teeth for 20 seconds, brushing hair for 20 seconds, wash hands for 15 seconds, dry hands at the towel rack
8. Leave bathroom and walked 4 m into the kitchen
9. Remove three dishes from the cabinet and place them on the counter
10. Pick up a kettle from the stove, fill the kettle with water, replace the kettle on the stove
11. Pick up two slices of bread and placed them in the toaster.
12. Walk 4 m from the kitchen into the dining room and sit at a table with a simulated meal setup

13. Butter two slices of bread, spread salt and pepper over a simulated meal, and pour a glass of water from a pitcher (each task item is placed across the table from the subjects, requiring them to reach their trunk and arms forward to pick up the items)
14. Rise from the chair, walk back into the kitchen, and wash a set of dishes and utensils in the sink
15. Walk 2 m from the kitchen to the elevator
16. Stand and wait for the elevator and then walk into the elevator
17. Turn around and then take the elevator to the first floor
18. Walk 50 meters towards the stairwell and then open door and enter stairwell
19. Walk up stairs (13 steps, landing, 13 steps)
20. Walk toward the exit door and then open door and turn right
21. Walk 15 meters and then turn around
22. Walk 15 meters towards the stairwell
23. Open door and enter stairwell
24. Walk down stairs (13 steps, landing, 13 steps)
25. Walk toward the exit door and then open door and turn right
26. Walk 20 meters inside the laboratory toward a bed
27. Lie on a bed for 20 seconds and then stand up & walk 10 meters
28. Turn right towards a ramp and then walk up 10 meters on the ramp
29. Turn around and then walk down 10 meters on the ramp
30. Pick up and wear a coat and then walk 10 meters towards the exit door
31. Open door and turn right to go outside
32. Walk 150 meters on paved path way to the entrance
33. Turn back to the laboratory.
34. Walk 150 meters on paved path way towards the starting point
35. Open door and turn right and then walk 25 meters to the starting point
36. Stand for 20 seconds, and shake the Smartphone for 2 seconds.