Fall Detection Using a Smartphone

Written by Ian James Daniel Gonzales
(ian.gonzales@hig.no)

Master's Thesis
Master of Science in Media Technology
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2011

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Fall Detection Using a Smartphone

Ian James Daniel Gonzales

1st July 2011

Revision History

Version	Note	Last Updated
0.1	Chapter 2 started. Taxonomy of falls finished and added.	1st March 2011
0.2	Study of falls finished and added	3rd March 2011
0.3	Principles and Algorithm for Fall detection finished and added	4th March 2011
1.0	Commercial fall detection systems finished and added, Evaluation of fall detection system added but not finished	5th March 2011
2.0	Chapter 1, 3 and 4 finished and added. Chapter 2 augmented, modified and finished. Appendix A and B added.	3rd June 2011
3.0	Chapter 5, 6 and 7 finished and added. Appendix C added. Final version.	27th June 2011
3.1	URL addresses rechecked and updated.	1st July 2011

Table 1: Revision History of This Paper

Abstract

A working fall detection system is of medical interest and significance for people who has risk of falling, their family, healthcare workers and the entire health care community. We always say that prevention is better than cure or treatment. This is also true for falling, and that is why fall prevention research and studies are very important. Ultimately though, we cannot prevent all falls! As such, the next best thing is an immediate response and assistance for the person who fell and possibly injured himself.

This paper is a study of fall detection systems. We will take an academic view on what they are and their nature as an assistive tool. We will make our own fall detection system using a smartphone as the device platform. Three specific accelerometer-based algorithms will be tested and compared to each other. By the end of the study, we will present a prototype of our system and its performance.

Preface

This paper is a master's thesis in Media Technology, a faculty of Gjøvik University College in Norway. The time boundary for this study is a complete one school semester, which roughly translates to six months.

0.1 Acknowledgment

I express my gratitude to the following persons for helping me with my experiments: Jurate, Elina, Christian and specially Mario who has helped me the most. I would also like to show my gratitude and appreciation to all other brave participants who helped me out with my experiment and offered themselves in the name of science and academic pursuit. I know it was not an easy job participating. Special thanks to Hilde Bakke for being devoted in helping the master students in general as well as for helping me out with my experiment. Thank you Prof. Slobodan Petrovic for your participation as well.

I would also like to give credits to my teachers who inspired me through my studies in Gjøvik University College - Prof. Faouzi Alaya Cheikh, Sule Yildirim Yayilgan and Simon J R McCallum.

Lastly I would like to say thank you to my supervisor, Prof. Patrick Bours. I hope I did not give you too much of a headache.

0.2 Dedication

I would like to dedicate this thesis to my late mother Dolores Gonzales and my sister Patrisha. You are a constant inspiration to me. Thank you!

Ian James Daniel Gonzales, 1st July 2011

Contents

List of Figures

List of Tables

1 Introduction

While the prevention of fall is undoubtedly a big and important research field in health care, the study in detection of fall has in the past three to four years increased in magnitude and importance as well. This is because ultimately we cannot prevent all falls, but we can try to detect fall accidents and give a quick response and aid.

The study in both prevention and detection of falls is mainly focused but not limited to falls that occur in elderly population. Studies show that one out of three adults age 65 and older falls at least once a year. Falls can lead to moderate to severe injuries, such as hip fractures and head traumas, and can even increase the risk of early death. It can also result in psychological problems from fear of falling [1] [2]. Falls are also the leading cause of injury deaths among people 65 and older. Some of these deaths can be prevented with immediate medical attention, and thus a good fall detection system is important.

1.1 Topic Covered

This paper will discuss what kind of fall detection systems we already have and their general characteristics. There are for the most part two leading approaches for making fall detection systems. The first one is the image-based system, and the second one is the accelerometer-based systems In this paper our focus will be the latter. Aside from the classification of fall detection systems, we will also see how one can typically evaluate such systems. This paper also tries to scour and present its readers commercial fall detection systems in contrast to those being studied and developed by research groups. In chapter 3, we will present this paper's main topic and own suggestion for implementing a fall detection system using a smartphone. For our prototype, three specific algorithms were tested.

1.2 Keywords

fall detection, panic button, mobile phone, pervasive computing, mobile computing, accelerometers

1.3 Problem Description

As previously mentioned, we do have existing fall detection systems. The first arguably system is the panic button as shown in figure 1. It can be a medical alarm bracelet or necklace which has a clear button. It comes with a base unit which calls a predefined response center and/or responder when an alarm is issued. The obvious problem and flaw with this is that, it is the user who has to push the panic button and issue an alarm to get help. This will not work if the user has become immobile or unresponsive when the accident occurs.

(a) Picture taken from `http://www.medicalalerts22.info/`

Figure 1: A typical panic button for the elderly

There is also the issue of mobility. If the system is using a base unit, like the one we mentioned earlier, then it is tied to one specific area - usually indoors. There are cases when fall accidents happen outside as well. Cold countries like Norway experience increased amount of fall accidents due to the roads being slippery. Existing commercial fall detection systems are very expensive and not thoroughly tested for its effectiveness and ease of use.

Another issue is that almost all of these fall detection systems only target the elderly population and see the elderly population as one unit. They do not account for other group of people who might have need for such system despite not being part of the elderly population. An example is people with epilepsy or neurodegenerative disease. Athletes can have interest in using such systems, or solitary adventurers out in the nature. Identifying the users of these systems and making these systems adaptable for different users is also important. There might be differences between how different individuals fall. This should be taken into account.

These issues will be discussed in the succeeding chapters. The proposed system of this paper will hopefully shed light to a system with better functionalities than its predecessors.

1.4 Justification, Motivation and Benefits

A working fall detection system with shown improvement over its predecessor - the panic button - is of medical interest and significance. Although the main target for such systems is the elderly

population, other people with a disorder that cause falling can use this kind of system. Such a system will not only be helpful for these people but also for their families, caregivers and perhaps the health care service community in general. This can only be true if the proposed system is reliable, adaptable, effective and cheap.

In the following chapters we will see some known approaches in this fall detection research study. We will try to comment on each approach and mention their strengths and limitations. This paper hopes to achieve an improvement of these previous studies.

Prior to picking out this topic as the author's master thesis, there were two things that inspired and motivated the author into pursuing this track. First is interest in mobile development. The second is an aspiration to contribute to the health care community.

1.5 Research Questions

We hope to find answers to the following research questions after our study is done. They can also be seen as information we need in order to solve our research problem stated in section 1.3. These questions were made at the very beginning of this study. They are subject to be changed and modified when more knowledge in the field has been obtained.

How would one define a "fall" in terms of a fall detection system?
The first few research articles we have read about fall detection study, indicate that the term fall can be vague when we dissect it's meaning - when used on fall detection and perhaps fall prevention systems. After doing this study, we hope to give this a more concrete explanation. We think that it is important to differentiate a normal fall (from ADL), fall accident (hurtful fall) and serious fall accident (need assistance/without recovery).

Is it possible to make a pervasive, reliable and effective fall detection system that could be adaptable for different types of user?
Not only do we believe it is possible, it is in fact our goal to achieve so.

What is the best algorithm and/or approach for detecting fall?
We don't have a concrete hypothesis for this yet, but we think that it is not only one algorithm but rather a combination of two or more algorithms.

There are now countless of algorithms to be comparing with, as will be explained in Chapter 2. We will narrow our choices to acceleration-based methods.

What constitute to be a good fall detection system?
We will propose a criteria for how well different fall detection systems are.

1.6 Methodology

To be able to answer the previous research questions, we have to use scientific methodologies which is briefly discussed in this section.

- *Literature Studies*. Literature work gives the basis of what kind of system and algorithms one can make. The majority of relevant findings from literature work we have used is presented in chapter 2.
- *Experiment and Data Analysis*. An experiment for data gathering is needed to be done before making any system. This is presented in section 3.4.
- *Creation and Testing of the Fall Detection System*. A prototype is made based on the experiment. Test and evaluation of the prototype were also done. This is presented in chapter 4. The testing part of the prototype can be seen as a continuation of the experiment. In fact they are similar in nature (see chapter 5 for further discussion).

An in-depth approach of how we proceeded to make our fall detection system is presented in chapter 3.

1.7 Planned Contribution

We will make a fall detection system implemented in a mobile phone. This new system will in several ways be an improvement over its predecessors. We will come-up with a good taxonomy and terminology related to the topic of fall detection. Lastly, our comprehensive coverage of the science of falling as well as fall detection systems will be helpful for future researchers in this field.

1.8 Report's Outline

The rest of this master's thesis is divided as follows; 1) chapter 2 provides the readers with the related work about the research field in fall detection as well as falls in general, 2) chapter 3 will present the readers our own approach for making a fall detection system, 3) chapter 4 shows a prototype of the system we have made and an evaluation of it, 3) chapter 5 is a further discussion about problems, assumptions, considerations and decisions we have had through the course of our study, 5) chapter 6 gives a conclusion for our study, and finally 6) chapter 7 is about future work that could be done on top of our study.

2 Related Work

2.1 Study of Falls

DEFINITION OF FALL

Many research groups have tried to define what a *fall* is. This is because fall for one specific group of users, e.g. elderly people, can be different to another, e.g. outdoor adventurers. One goal for the research groups' experimental set-ups is to distinguish Activity of Daily Livings *ADLs*, from that of an actual fall. Now as mentioned, a fall for elderly people might be different for another, so it goes without saying that a fall detection/prevention system for a group of people might not be suited for another.

This paper contends that the term "fall" is vague. Fall detection systems might therefore be understood very differently according to how they define and detect falls. In fact according to The World Health Organization (WHO), there is no universal consensus on the definition of falls [3]. This leaves room for many different interpretations of a fall, and consequently brings into question the validity of the studies. That being said, most researchers [4] start with the following definition of what a fall is and modify it - either by broadening and extending it or restricting it - to suit their system;

> "Fall is inadvertently coming to rest on the ground, floor or other lower level, excluding intentional change in position to rest in furniture, wall or other objects." [3]

TYPES OF FALLS

In our study, we distinguish between two types of falls. The first one is *accidental falls* which are caused by external (environmental and behavioral) factors. The second one is *non-accidental falls* which are caused by internal (biological) factors.

It is important to differentiate between these two as the physics behind them are quite different. (Accidental falls are associated with more activity and external forces.) As such, a fall detection system targeting one group might not be suitable for the second group. In fact, most researches about fall detection and prevention are only about non-accidental falls. This is because fall detection and prevention studies are directly connected to elderly people, which has grown in importance internationally in the last two to three decades. Experimental and Commercial fall detection systems are therefore primarily made for and to be used by elderly people. While elderly people can experience both non-accidental and accidental falls, usually what is being recorded and observed are non-accidental falls due to, e.g., frailty or sickness.

For accidental falls, injury specialist differentiate between *same-level falls* and *elevated falls*. Same-level falls are more frequent but are less severe in terms of suffered injury. Elevated falls are lower in frequency but have a high severity [5].

Among same-level falls, we find [5];

- *Slip and fall*. This is caused by slippery surface (icy, wet or freshly waxed surfuces). The less traction between the surface and a person the more tendency for the person to slip and fall.
- *Trip and fall*. Trip occurs when the front foot strikes an object and is suddenly stopped. The upper body is then thrown forward and a fall occurs. As little as 3/8" rise in a walkway can cause a person to "stub" a toe resulting in a trip and fall.
- *Step and fall*. The front foot lands on a surface lower than expected, such as when unexpectedly stepping off a curb in the dark. Forward falls will happen most likely. A second type of step and fall occurs when one steps forward or down and either the inside or outside of the foot lands on an object higher than the other side. The ankle turns and one tends to fall forward or sideways.

The most common elevated falls are; falls from ladders, vehicles and equipment, loading docks, stairs, stools and chairs.

Non-accidental falls are not so much about how the falls happened, but rather how they are much alike to falls that might happen to elderly people. This means research groups studying this type of falls are trying to simulate falls from elderly patients. Factors that are considered for fall simulations are;

- Direction of falls: forward, lateral and backward
- Body position (sitting, lying<bed>, standing, leaning, etc.) and orientation (knees apart, with forward arm protection, using a walker or rollator, etc.) before fall
- Body position (kneeling, sitting, leaning, etc.) and orientation (ending lying flat, ending in lateral position) after fall
- Speed of falls
- Surface to hit: hard, soft
- Activity or Responsiveness before fall
- Recovery or Activity/Inactivity after fall
- Actions and activities that might look like falls but are not, e.g. jumping

These factors are used to form different plausible fall scenarios which are later used for both data collection, prior to training the fall detection system, and test and evaluation of finished

fall detection systems. The very minimum is to simulate directional falls: forward, lateral, and backward falls.

CAUSE OF FALLING

There are several risk factors which contribute to a higher rate of falling. In general we divide these factors into intrinsic (biological) and extrinsic (environmental and behavioral). We can then use these factors to find groups of people that are likely to fall, and hence make a thorougher and more specific fall (detection) research.

Biological factors are [3] [6];

1. Age. The older a person, the more frequent falls and fall related injuries and mortality happen.
2. Medical Conditions like Diabetes, Parkinson's Disease (PD), patients who suffered stroke, Incontinence(inability of the body to control the excretory functions), Alzheimer Disease
3. Muscle Weakness. A decline in muscle strength can interfere with balance. People with some sort of muscle weakness has five times more tendency to fall. Furthermore persons with lower extremity weakness, usually measured by knee extension, ankle dorsiflexion, and chair stands, have a 1.8-fold increased risk of falling.
4. Visual Impairment. Decrease visual acuity increase the risk of multiple falls.
5. Cognitive impairment and confusion even at relatively modest levels can increase the risk of falling.
6. Foot problems. This include severe bunion, toe deformity, ulcer and deformed nails have a 2-fold increased risk of falling. Problem with ankle, hallux valgus deformity, impaired tactile sensitivity, decreased toe strength, foot pain, all impair balance and increase the risk of falling.
7. BMI. Low Body mass index is associated with increased risk of falling. Low body weight and unintentional weight loss due to malnutrition are a particular problem.
8. Presence of chronic neurological disorder such as epilepsy.

Diseases and medical conditions that have one or more of the factors mentioned above are likely to cause falls. The more risk factors a patient have, the higher the probability that patient will fall. The same is true for the next type of factors.

Behavioral factors are [3] [6];

1. Sedentary behavior. Those who have previously experienced falls, tend to be less active which causes muscle atrophy, which in turn increases the risk of falling.

2. Medication intake. Taking in multiple types (more than 4) of medicine, irregardless of what they are, increases risk of falls.
3. Drugs that act on the central nervous system like antidepressants, sedatives, tranquilizers, and benzodiazepines give a higher risk of falling.
4. Alcohol Misuse. There is a direct correlation between alcohol consumption and falls.

Beside the two mentioned risk factors, researchers have also found some trends to the distribution of falls. They are as follows [3] [6];

- Time. Most falls occur during the day. Only 20% happen at night (9pm-7am).
- Climate/Weather. Falls occur more on places and conditions where it is more colder. For non-accidental falls, this might be because of the mild hypothermia that slows reaction time. Also places where it snows sometimes can get very slippery causing accidental falls.
- Location. There are slightly more falls that occur outdoors than indoors.
- Sex. Women falls and suffer injury more often than men. Men have higher death rates because of falling (even though women fall more often). This leads us to believe that the physics of men falling might be different than the physics of women falling.
- Race. In USA, Caucasians have higher fall rates than Afro-Americans.
- Socioeconomic status. Women living alone tend to fall more often. Also people who has limited access to health and social services, low income, little education and poor housing environments are associated with higher risk of chronic disease which may be associated with an increased risk of falling
- Depression is associated with increased risk of falls.

Since there are many risk factors associated with falls, we conclude that there are also many different groups of people (and not just the elderly) that suffer from falling. Each of these groups might show a different type of falls in terms of fall patterns, behavior and physics (see TYPES OF FALLS 2.1).

PHASES OF FALLING

According to [4], there are four phases of falling. These are the *prefall phase*, the *critical phase*, the *postfall phase* and the *recovery phase*. We have previously mentioned that when researchers try to simulate falls, they make fall scenarios from a set of factors about falls. These factors can be found on some of these phases (see TYPES OF FALLS 2.1).

In the *prefall phase* the person performs usual activities of daily living (ADL) (in accidental falls the activities might vary a lot depending on what the targeted group is), with occasional sudden movements, like sitting or lying down rapidly. Fall detection systems must be able to

distinguish this phase from the next one.

The *critical phase* consists of the sudden movement of the body toward the ground, ending with a vertical shock on the ground. The duration of this phase is extremely short, usually between 300-500ms [4]. [7] determines it to be 400-800ms.

During the *postfall phase*, the person usually remains inactive, frequently lying on the ground. The postfall phase shouldnt last too long to reduce the consequences of the fall.

Eventually the *recovery phase* is either intentional - the person stands up on his own - or with help from another person.

We will use these phases for further discussion in section 2.3.

2.2 Taxonomy of Fall Detection Systems

Irregardless of what group of people a fall detection system is made for, a fall detection system can either be categorized in where and how it is situated/located OR what kind of underlying method is used behind the system.

FIRST CATEGORIZATION

The first category has the following classifications; *wearable systems* and *ambiance systems*. *Wearable systems or endosensors* are systems where the device is being worn by the user. Normally, their purpose is to measure the posture, motion or a combination of both, of the user. Typically used sensors are accelerometers and gyroscopes.

Wearable devices are usually placed either on the head, trunk, waist, wrist, knee or thigh of the body. Several research groups, [8], [9], [10], have found that placing it on the waist is the best place to put the device on. This is because it provides reliable indications of full-body movement, in addition to its ease of acceptance by allowing attachment to an existing waist band. The waist is also the more acceptable option as most people are already used to wearing accessories on their waist, e.g. a belt.

Advantages of wearable devices are they are cheap, easy to be set up and operated, and usually light-weight. Disadvantage is that with this approach, we assume that the worn device keeps a fixed relation with the wearer, and this condition is easy to be broken. Most wearable devices can also be quite intrusive which elderly people does not appreciate. Also the general comment from practicing doctors is that most of the patients have low will to wear device for detecting falls because they feel well before fall occurs. Lastly, wearable systems are exposed to getting damaged and broken upon fall.

Ambiance systems or exosensors are systems in the environment or the very near vicinity of the user. They can be multiple sensors on the floor or on the bed. They can for example detect presence of users by pressure sensors or body heat by infrared sensors.

An advantage with ambiance systems is that they are less intrusive. On the other hand, since most ambiance systems are hidden from the user, some users don't feel secure that the there is a fall detection system working, because they can not see the monitoring device. Ambiance systems can also only be used indoors making it limited for other applications and users. The difficulty of setting-up ambiance systems can vary a lot, but in general it takes time and effort because of their varying complexity. Different ambiance systems differ in addressing the problem of multiple presence (other people, pets, moving things, noise, or other external factor) in the room, but it is more likely that ambiance systems just assume that these noises are not present. In general, wearable systems are also more popular and acceptable by users than ambiance systems.

[11] suggests to make a third group which is *camera-based systems*. Strictly speaking a camera-based system can be wearable, e.g. in the walker of an elderly patient [12], or in the user's environment which is the more normal approach. The reason they want it categorized by its own is that there is an own focus study with this approach. Among the ambiance systems, camera-based ambiance system is one of the most reliable ones.

In this study we will also refer to camera-based systems as *image processing-based systems* and *visual-based systems*. We interchange using the terms camera-based, image processing-based and visual-based throughout this paper, but we usually mean the same thing. We vary the use of the terms depending on what we want to emphasize. Using the term camera-based emphasizes the use of video camera as the fall detection device, while using the term image processing-based emphasizes the use of image processing techniques in the fall detection algorithm, and using the term vision-based emphasizes the perception of sight or being seen.

Several different algorithms on how to calculate fall through image processing techniques are developed. For example, according to [13], the camera can use a people-detection and tracking algorithm. With this tracking algorithm, the centroid height of the person is determined. If the height is below a threshold, then the person is in a "fallen" state.

Three types of image processing-based algorithms are discussed in [11]. They are;

1. *Inactivity Detection Algorithm*. This relies on the fact that a fall will end in an inactivity period on the floor. The algorithm overhead tracks person so to obtain the motion traces of the person. Then it classifies the activities based on the motion traces and context information. Inactivity is one of the classes, and inactivity will be said to be a fall it if occurs in certain context.

2. *Shape Change Analysis Algorithms*. They use the principle that the shape of falling person will change from standing to lying.

3. *3D head motion analysis Algorithm*. They use the principle that vertical motion is faster than horizontal motion in a fall. The fall detection component computes the vertical and horizontal velocity of the head and then uses two appropriate thresholds to distinguish falling from walking.

These algorithms are similar to the ones we will discuss in section 2.3.2.

Advantages of camera-based fall detection systems are: they can be used to detect multiple events simultaneously, they (ambiance-based systems) are less intrusive to users because they are not worn by users, and lastly the recorded video can be used for remote and post verification and analysis. Video cameras' prices are also decreasing in the past years so that they are not as expensive as before. Patients who want extra surveillance prefer this method.

Disadvantages with camera-based methods are: firstly, while camera-based techniques are well established in controlled environments(laboratory, scene), they must be modified in uncontrolled environments where one controls neither the lighting nor the framing, secondly, as the subject moves in a 3 dimensions space, it is also necessary to call upon more complex techniques, namely use of 2 (or more) cameras ("stereovision") [14], thirdly, the acceptance of this technology can pose a major problem when it comes to the user's privacy since the cameras are usually placed in the user's bedroom and bathroom, fourthly, aside from [12], most camera-based systems are ambiance-based systems which means that they don't offer mobility.

SECOND CATEGORIZATION

Image processing-based fall detection systems can either be an ambiance system or a wearable device as previously mentioned. This means that the taxonomy of fall detection systems does not end with our first classification.

In accordance to another research group, [13], we make another way to categorize fall detection devices based on what the underlying method behind the system was used - *acceleration-only methods, non-acceleration methods, and hybrid or mix methods*.

In *acceleration-only methods* a sensor or multiple of sensors detecting acceleration/deacceleration are used. This method has become rather popular and perhaps the first approach in fall detection systems aside from the panic button because of the simple fact that all falls have a sudden acceleration change. Being able to properly and accurately detect this change is the goal of the further research with this approach. They give fairly good results compared to non-acceleration methods. According to [13], acceleration is a critical role in elderly mobility monitoring, and should be used in fall detection. Compared to visual and acoustic sensors, accelerometers consume less energy and much easier to integrate into wearable mobility monitoring devices.

Because our own fall detection system will be an acceleration-based type, we will present a handful of specific equations and algorithms that is based on acceleration in section 2.4.

The research group in [15] uses rapid acceleration change to detect shock. While their system is not a fall detection system but rather a security and health emergency system, both systems are similar. In section 2.3 we will see that one type of algorithm fall detection systems can use, is based on impact which can be measured by accelerometers. This is exactly the same thing as what shock is for the previously mentioned group. Pattern recognition technique (statistical signal processing methods) is used. The group announced an *accuracy* (see sec 2.5) of 91.1%.

One of the earlier implementation of acceleration-based fall detection system can be found in [16]. It is a wearable, acceleration-based device. The system measures velocity, shock and inactivity. First, a high velocity towards the ground has to be detected. Then, within 3 seconds an impact has to be detected, or the event will be discarded. After the impact the general activity will be observed for 60 seconds. If during this interval at least 40 seconds of inactivity are recorded, the system will generate an audible alarm. It only has a *sensitivity* (see sec 2.5) of 65%.

Any other fall detection systems that do not rely on acceleration, fall into the category of *non-acceleration method*.

We have the *audio-based approach* where audio sensors are used to record sound, which is later analyzed, in the user's surrounding. A sudden noise will be considered as a fall, i.e. basing on the assumption that falls tend to make loud sound upon impact. The problem comes when there are other sounds and noises in the room, thus giving a high false positive rate. [17] provides a more concrete example. The paper used an array of acoustic sensors aligned vertically, which is mounted on targeted places on the patient's room (ambiance system). It has less false positives because it filters out some of the noise. Then height calculation is also involved to remove false alarms. The array of sensors compute the height of the sound produced. If the signal is computed to be less than two feet, it will be considered as a fall, and it will be sent for further analysis before finally stating that it is a fall. Even with the improvement added by using height computation, the false positive rate is still high. When tested, their system gave five false alarms per hour.

Angular-based (body angle/orientation, angular velocity and angular acceleration) approach using gyroscopes is also possible, although in practice this is usually done with combination of accelerometers. [18] uses a bi-axial gyroscope worn on the trunk of the user. Experimentally, it is one of the most successful fall detection system since it has a result of 100% *specificity* and 100% *sensitivity* (More about sensitivity and specificity in section 2.5). This is of course only measured under controlled environments with selected number of falls to simulate.

Visual-based approach, as mentioned before, is also under this category. This is perhaps the most popular among non-acceleration methods.

Other less popular non-acceleration methods are; *pressure-based approach* like in [19], *vibration-based approach* like in [20], and *infrared(body-heat)-based approach* like [21]. *Velocity-based approach* also exists, but this usually falls under acceleration methods, since velocity is calculated from the acceleration. Non-acceleration method is not really effective and commercially successful. The most promising of these are visual-based and angular-based approaches.

Basing on previous studies and results, researchers have tried to improve fall detection systems mainly either by a) using several of the same sensors placed on different body location e.g. one on the thigh and another on the trunk like in [22] and [9], or b)using different sensors. It has been observed that by adding more sensors (either of the same or different kinds), it usually leads to better and more accurate data and thus result. This is what the third approach, the *mix/hybrid* methods, is all about. In practice one of the sensors used is an accelerometer.

Terminology	**Explanation**
Direct Detection	*Detection of a sudden postural variation*
Accelerometric	Detection of sudden movements and shocks to the body
Statimetric	Detection of body height above ground level
Topemetric	Detection of body spatial position
Eidolimetric	Video monitoring of the body
Peripheral Detection	*Detection of behavioral/contextual variations*
Behavioral	Monitoring of behavioral sequences
Geotaximetric	Monitoring of spatial localization of the person
Orthostatimetric	Monitoring of the body orientation (verticality)
Kinesimetric	Monitoring of vital signs
Combinatorial Detection	*Combination of direct and peripheral detection*
Bidimentional	One direct and one peripheral parameter
Tridimentional	One direct and two peripheral parameters
Multidimentional	One direct and three or more peripheral parameters
Axis	Says how many physical dimension axis are monitored (x-, y- z-axis)
Monoaxial	Only one axis is monitored, usually the vertical axis
Multiaxial	Monitors multiple axis usually giving better results
Other terminologies	
Temporal	Monitoring of temporal sequences like time when user is not moving
Shock	Measures shock impact
Image	Image Processing in Measurement
Physiological Parameter	Measurement of physiological parameters
Geolocalisation	Geographical localization of user/device
Cardiac Ballistics	Measuring cardiac movements of the heart
Data Fusion	Combining data from multiple sources and gather that information in order to achieve inferences

Table 2: Terminology for the classification of fall detection

An example is using two tri-axial accelerometers and gyroscopes on two different body parts like what they have done in [9]. Not only can you measure the linear acceleration, but also the angular velocity with this combination of sensors. Their algorithm is divided into three steps: a)activity intensity analysis, b)posture analysis and c)transition analysis. This just means that the algorithm checks for sudden acceleration change, the body's start and final posture/position and the transitions that happen in between. If certain criteria are met then the system will know that there is a fall. This system has fast response time and low computational cost. The specificity of the system is 92% from 72 records, and sensitivity is 91% from 70 records.

Other examples can be combining accelerometer with a microphone for audio sensing or a magnetometer like in [8] which has 3.1% false negative (not detecting real fall) and 7.7% false positive rate (false alarms), or combining the use of infrared sensors and pressure sensor on the floor like in [23]. The latter achieved a *reliability* (see sec 2.5) of 90%. Yet another example is combining accelerometers with air pressure sensor as done with [24]. The air sensor is used to sense the altitude of the device. Unfortunately, the utility of this approach is limited due to practical consideration.

[4] offers a table of terminologies based on commercial fall detection products as well as experimental done by other research groups. This can be useful if one wants to be specific with describing fall detection systems (see table 2). Evaluation of these systems, both commercial and those in research groups, would have great use for these.

2.3 Principles and Algorithms for Fall Detection

There are as [14] presents two approaches to go about and detect fall. The first one, which is also the more commonly used approach, is the *analytical method*, and the second one is with *machine learning techniques*. An example of machine learning approach is presented by [25]. Their system is a visual-based system. By using markers, they tag different points on the body and use these markers as reference points. They found that the reference points and the angles being formed between them, is a very reliable data source for extracting features. Among the different machine learning algorithms, SVM (Support Vector Machine) was reported to have performed the best, followed by Random Forest.

2.3.1 Machine Learning

In Computer Science, the branch of artificial intelligence in which a computer generates rules underlying or based on raw data that has been fed into it is called *Machine Learning*. It involves developing algorithms that would enable computers to learn complex patterns and make intelligent decisions based on that. It is a big branch in the field of Computer Science that covers several topics. We are mostly interested in pattern recognition.

Machine learning can broadly be categorized into two fields, Unsupervised Learning and Su-

pervised Learning. In the prior, the machine tries to identify groups of similar data from a larger dataset. In other words, it tries to form clusters of data based on some criteria such as cost functions. The machine has no prior knowledge of classes the data belongs to, it only tries to identify natural clusters or groups of data. Supervised learning on the other hand learns from the test set which contains classified data and predicts the classes of unseen data [26]. For fall detection systems it is more natural to use supervised learning techniques.

Without any analytical model, one can still carry out an "intuitive" approach to the development of machine learning based fall detection systems starting from observation (a training period) and then classification. Yet it is necessary to set criteria for classification that are sufficiently significant and independent (discriminating). If one proceeds through a supervised training period, one can train a neural network, which will then be used to automatically classify future situations. Only the situations met during training can be classified, the others being mixed in a class labeled "others" (to stumble, to slip, etc.). If the training is "unsupervised", a class "fall" is likely to be isolated if the training period is long enough, even infinite if the event of fall is rare. Moreover the first fall event is likely to be missed since its class is still unknown before its first occurrence [14].

For supervised learning it is important that the quality of the training data is sufficient, exhaustive and accurate. That is why, in the training period, it is important to be able to simulate falls as close as to how the real falls of the group of users who will use the fall detection system.

Among the number of machine learning algorithms out there, it can be confusing picking the right one for certain applications. There are even research studies comparing a lot of them to try and say which is the “best” one [27] [28]. In the end it is left to a developer to choose an appropriate method on his own discretion. For sensitive applications such as a fall detection system, it is advisable to test and compare a number of algorithms to see which one performs the best.

[26] is a paper comparing five different supervised learning methods for a fall detection system. These are Naive Bayesian Classifier, Radial basis Function (RBF), Support Vector Machine, C4.5 and Ripple Down Rule Learner. They use eight scenarios (four falls, three ADLS, and one near fall) to gather data from and test the algorithms against. Naive Bayesian Classier, giving the highest accuracy of 97.32% and taking the least time for building the model, was chosen as the best one in the end.

One of the earlier fall detection system using machine learning algorithm is [7]. The group integrated a device with tri-axial accelerometer, a single chip modem, a MCU device and some other peripherals, to a cellphone. They used this for fall detection with the use of 1-class SVM for preprocessing the signals and KFD (Kernel Fisher Discriminant) and k-NN (Nearest Neighbor) algorithm for precise classification. They reported a 93.3% accuracy for a limited number of scenarios.

2.3.2 Analytical Method

Analytical method is the “manual” or empirical study of data sets and setting-up your own parameters (thresholds) based on them. There is a need for a thorough study of falls and phases of falls for better understanding and result. What this study focuses on, is presented on this section.

When a fall detection system starts, it should right away detect ADLs and would not get these confuse with falling. This happens in the prefall phase.

In the early detection of the critical phase of the fall, rapid acceleration can also signify fall. During a fall there is a temporary period of “free fall”, during which the vertical speed increases linearly with time due to gravitational acceleration. If one measures the vertical speed of controlled movements of the person (to rise, to bend down, to sit down), one can then discriminate these speeds from those occurring during a fall, which would exceed an appropriate threshold. The difficulty lies in the choice of this threshold, if it is too low the device may also detect negative events (“false positives”), but if the threshold is too high it may not detect positive events (“false negative”) [14]. More about false positive and false negatives in section 2.5. A learning period may be used to overcome this difficulty. This learning period may be “supervised” by asking the wearer to carry out a series of voluntary movements in order to mark the normal speed of execution.

Fall detection systems can use fall acceleration, fall velocity, body orientation, body posture, angular acceleration and angular velocity in the critical phase as the determining factor(s) for distinguishing falls from non-fall activities.

Eventually, the detection of the very early phase of the fall could allow launching an airbag or a safety system that could dampen the fall or even save the user. However, this sole approach is still highly experimental and in research.

At the end of the critical phase, the body frequently hits the ground or an obstacle. The mechanical consequence of this is a sudden polarity inversion of the acceleration vector in the direction of the trajectory; this is commonly referred to as the “impact shock”. Fall detection in this phase can be done by pressure sensors, shock sensors or by accelerometer. An accelerometer will register an intense inversion of the polarity of the acceleration vector in the direction of the trajectory [14]. The sound produced upon impact can also be measured like what has been done in [17].

Some systems use this approach, but the difficulty of this approach is to determine the direction of the trajectory, which is obviously variable from one fall to another. However, we know that most falls occur in the “sagittal” plane (forward or backward) as the fall often follows a voluntary movement, which is mainly performed in the “sagittal” plane (to sit down, to rise, to

walk, to bend down). A second difficulty is the location of the sensor on the body relative to the point of impact. Depending on whether or not the sensor is near the point of impact, the signature of the signal recorded at the time of the shock can be significantly different, and it thus becomes more difficult to recognize a fall when it occurs, thus leading to a significant number of "false positives" [4]. That being said, we hope to find patterns through our experimental set-up that will hopefully give us enough data so that a fall detection system can use this method viably.

The simplest approach to detect a fall is by detecting the lying position, e.g. from a horizontal inclination sensor (a mercury contact or a ball trapped in a guide). This is based on the fact that most falls end up in this lying positing but ultimately not all falls, [9] [14]. This is what can be called as indirect detection of the lying posture during postfall. Another approach is to detect when the feet are no longer in contact with the ground. Just detecting if the user is on lying position to signify fall is of course flawed. Napping and sleeping would cause "false positives".

In the postfall phase, a fall detection system can use the body orientation, body postures, and the difference or transition between the body orientation or posture in prefall and postfall phases as determining factor(s) for a fall.

In the recovery phase, the lack of movement, activity or responsiveness can be used to detect fall as, after a "serious" fall where the person may be seriously injured, they frequently remain immobilized in a posture. You allocate some time to observe that the person is actually immobile by using sensors like accelerometers to register how much movement there is [14]. The drawback with this is that intervention or call for help will also take some time to come depending on the delay time you put on the system. Falls that are followed by epileptic attacks will also register a lot of movement. The time a fall detection system should wait before triggering an alarm, based on the recovery phase, either depends on the gravity of the fall (if in critical phase, a vertical shock showing-off-the-charts measurements was measured, then this might be a sign of a very serious fall) or the user using the system (a user more likely to recover and stand-up after a fall should be given a little more time to do so than a user who cannot). A good system should lower this to maximum of one minute.

As previously mentioned, studying the body's postures and not only its orientation can signify that a fall has happened. This is the approach of [9]. Orientation sensors - amongst which a gyroscope would be the best - can be used for this. Infrared sensors, cameras and other sensors can also be used to sense immobility. A smart phone with its range of sensors (preferable a gyroscope, accelerometer and orientation sensor) would be suitable for the postfall phase analysis - actually not only this last part but for all the mentioned principles above.

If the system knows more about the user then it can predict how the user will respond on this phase upon falling. A context-aware system is much more preferable. If the system knows that the user of the system is likely to show activity upon falling in the recovery phase, i.e. epileptic attacks after a fall or a patient who could move the upper part of the body but not stand-up,

then it will not confuse the registered activity as a sign of recovery and will continue to issue an alarm. Also if the system knows, through context, that the user is likely to take more or less time before recovery or standing-up on his own, the system can then vary the length of wait time before issuing an alarm.

Combining the detection principles and algorithms from the different fall phases prove to give more accurate results. Using multiple detection algorithms per fall phase tend to increase accuracy too. The trade-off is more processing resources used by the fall detection system.

2.4 Acceleration-based Equations and Algorithms

To have an overall sense of acceleration, the device's total acceleration from the X, Y and Z axis are combined like most previous researchers have done in [8], [29], [22] and others. This is called the *RSS (Root-of-Sum-Squares)*. Different literatures may refer to this in different manner, but in general they have this equation eq. 2.1. Other studies use the linear acceleration or the actual acceleration of the person. This is sometimes referred to as the dynamic total acceleration, eq. 2.2.

$$\mathrm{RSS} = \sqrt{x^2 + y^2 + z^2} \tag{2.1}$$

$$\mathrm{RSS}_d = \sqrt{x^2 + y^2 + z^2} - 9.81\mathrm{m/s}^2 \tag{2.2}$$

The simplest algorithm is to just check the acceleration in the early part of the critical phase. Standing upright will give a total acceleration (RSS) of 1G, eq. 2.3. This value plummets down towards 0G in free fall until finally reaching impact. A threshold is set based on a training session where data from falls are compared to data from ADLs or other non-fall activities. This value is sometimes called the *Lower Fall Peak (LFP)*. Finally deciding upon the threshold will give the *Lower Fall Threshold (LFT)*. This approach is also called *thresholding*.

$$1\mathrm{G} = 9.81\mathrm{m/s}^2 \tag{2.3}$$

Upon impact, the user receives an impact force from the ground. This force leads to the vertical acceleration value increasing quickly towards the opposite direction of the acceleration due to gravity. Measuring this will give the *Upper Fall Peak (UFP)*. One can then proceed and make an *Upper Fall Threshold (UFT)* based on a training period. One can also use UFT_d (Dynamic Upper Fall Threshold) which is the UFT based on RSS_d.

These two different algorithms were tested in [22]. UFT gives a better specificity and thus reliability than LFT. The more the user is active and doing more varied activities, the lower the

system's specificity is. This means plenty of false alarms. It is therefore a common practice to combine both the LFT and UFT to make a *profile*. This is called the *profiling algorithm*.

A more elaborate profiling scheme is to also measure the *falling-edge time* (t_{FE}) and *rising-edge time* (t_{RE}) as done in [10]. t_{FE} is the time from when the RSS signal last goes below the LFT until it reaches the UFT. t_{RE} on the other hand is the time for when LFT is last exceeded until UFT is exceeded. t_{RE} is always a subset and smaller than t_{FE}. To clarify, imagine that in t_0 the RSS signal first goes below LFT and stays under it until $t_{0+200ms}$. From $t_{0+200ms}$ the RSS value starts to rise up until UFT is reached. UFT is reached in $t_{0+500ms}$. $t_{0+500ms}$ - t_0 is the t_{FE}, while $t_{0+500ms}$ - $t_{0+200ms}$ is the t_{RE}. One can then check for the profile LFT + t_{FE}, + UFT or LFT + t_{RE}, + UFT. Make note that both LFT and UFT are a prerequisite for using t_{FE} and t_{RE}. Thus, profiling algorithms that only use LFT and UFT can be further expanded to also use t_{FE} or t_{RE}.

In [30], they have made a similar profiling algorithm. Using a 1.5 sec window, they partitioned the RSS data with 50% overlap. (Sometimes the sliding window is shorter and does not overlap like in [31]). The algorithm finds the LFP and UFP in the current time window and take the difference between the two giving the $RSS_{UFP\text{-}LFP}$, eq. 2.4. If both the UFP and LFP reach their respective thresholds and the LFP happens before the UFP, then a fall is flagged.

$$RSS_{UFP\text{-}LFP} = t_{UFP} - t_{LFP}, \text{where t = time of occurrence} \tag{2.4}$$

To improve on the specificity of a fall detection system, many algorithms also check for the body posture. Most of these algorithms assume that in the post-fall phase and before the recovery phase, the body is in the lying *posture*. One can then also make a threshold similar to LFT and UFT when the body is lying, the *Upper Lying Peak (ULP)* and the *Lower Lying Peak (LLP)*. This was done in [32]. The algorithm first checks that both LFT and UFT are achieved. Then after 10 seconds if both the *Upper Lying Threshold* and *Lower Lying Threshold* were also achieved then a fall is flagged. This profile is then LFT + UFT + wait 10 sec + LLT + ULT.

Another way to measure the critical phase of a fall is by measuring the *velocity* usually by using the formula eq. 2.5.

$$\int RSS_d \, dt \tag{2.5}$$

The velocity threshold is based on the integral from the time of the start of a fall until impact [10] [29]. The velocity threshold can then also be added in a profiling algorithm or stand alone by itself.

One can also use the *fall index*, eq. 2.6, or the *Z2*, eq. 2.7, to make fall thresholds. In [31] they make use of both in three different types of algorithm. *Algorithm one* checks for both impact and posture. The threshold for impact is based on either, Z2, UFT, UFT_d, or $RSS_{UFP\text{-}LFP}$ + Monitoring of Posture. *Algorithm two* uses LFT + UFT (within time frame of 1 second) or

threshold based on Z2 + Monitoring of Posture. *Algorithm three* uses LFT + threshold based on velocity + UFT (within time frame of 1 second) or threshold based on Z2 + Monitoring of Posture. Posture was detected 2 seconds after the impact using the vertical-axis acceleration, based on the average acceleration in a 0.4 seconds time interval. The lying posture is usually lower than 0.4G. Their findings show that the first algorithm has the best sensitivity (=97%) if UFT or Z2 based threshold + Posture is used. The fall index was used for comparison.

$$FI_i = \sqrt{\sum_{k=x,y,z} \sum_{i-19}^{i} ((A_k)_i - (A_k)_{i-1})^2}, \tag{2.6}$$

where x, y, z = the acceleration from the X, Y and Z axis

$$Z2 = \frac{RSS^2 - RSS_d^2 - G^2}{2G} \tag{2.7}$$

Posture can also be measured using the *scalar product* of two acceleration vectors and thereby finding the angle between them. In [10] this can either be a) between the *reference gravity vector* and the *current gravity vector*, eq. 2.8 or b) between the vertical axis (Ay) and the gravity, eq. 2.9. The reference gravity vector is taken from the average of the tri-axial accelerometer signal recorded when the sensor is attached and the subject is in standing position for 5 seconds. In [24] and [33] this is between two consecutive RSS value, eq. 2.10.

$$\Theta = \arccos \frac{referencegravity \bullet currentgravity}{\|referencegravity\| x \|currentgravity\|} \tag{2.8}$$

$$\varphi = \arccos \frac{A_y \bullet gravity}{\|A_y\| x \|gravity\|} \tag{2.9}$$

$$\Phi = \arccos \frac{RSS_t \bullet RSS_{t+1}}{\|RSS_t\| x \|RSS_{t+1}\|} \tag{2.10}$$

Some devices already have orientation sensors [1] or the more accurate gyroscopes which then can be used instead. The research group in [8] both uses equations 2.4 and 2.11. In equation 2.11, z is the vertical axis (up and down), y is the horizontal axis(left and right) and x is the front-back axis.

$$Orientation = |A_x \sin\Theta_z + A_y \sin\Theta_y + A_z \cos\Theta_y \cos\Theta_z|, \tag{2.11}$$

where $\Theta_x, \Theta_y, \Theta_z$ = the data from the orientation sensor

[1] Traditionally, before the lunch of Apple's iPhone 4, most orientation sensors are also based on the accelerometers. Today however, more and more smartphones have integrated gyroscope which then can be used in combination with the accelerometer to more accurately depict the orientation of the phone.

2.5 Evaluation of Fall Detection Systems

When we evaluate fall detection sensors we need to test in practice how well they work. That is, we need to see that real falls will be registered.

False positive, False Negative, Accuracy, Sensitivity and Specificity have been used earlier without really giving any explanation. We will now give them definitions and how they relate to each other.

According to [14] there are four possible cases that could happen when we test fall detection systems. In our test we count the number of occurrence for each case and make note of it, making a statistical record. The four cases are as follows;

1. True Positive (TP): a fall occurs and the device detects it.
2. False Positive (FP): the device announces a fall but the fall did not happen.
3. True Negative (TN): no falls occur and the device does not say that it does.
4. False Negative (FN): a fall occurs but the device does not detect it

Based on the four cases above, we have four criteria to evaluate fall detection systems. There is no consensus among research groups on which one(s) to use, but sensitivity and specificity are the most important to take note of.

- $Sensitivity = \frac{TP}{(TP+FN)}$: This is the capacity to detect fall.
- $Specificity = \frac{TN}{(TN+FP)}$: This is the capacity to detect only a fall.
- $Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)}$: Gives an overall correctness for fall detection.
- $Kappa = \frac{(A-C)}{(1-C)}$: A = Accuracy
- $C = \frac{(TP+FN)}{(TP+TN+FP+FN)} x \frac{(TP+FP)}{(TP+TN+FP+FN)} x \frac{(FN+TN)}{(TP+TN+FP+FN)} x \frac{(TN+FP)}{(TP+TN+FP+FN)}$

Perfect sensors will give 100% (=1) on both sensitivity and specificity. In general we want to avoid false positives and false negatives. Arguably we can say that false negatives are worst to have than false positives. We say that it is more acceptable that the device gives us fake alarms where there are no real falls than the device not alerting us for true falls. This is mentioned here now because what happens most of the time is that when we adjust the parameters of our system so that we increase either of the criteria, the reverse effect will usually be seen on the other criteria. In other words sometimes increasing sensitivity of the device will decrease its specificity and vice versa. This effect can also be seen in the data analysis in section 3.5.

In chapter 3, we will take a deeper look on how to make an experimental set-up to gather these statistical data, which will then tell us the performance of our system. General experimental set-ups much like ours are done by research groups to acquire these data.

According to [4], besides the intrinsic performance of the fall detector, other criteria must be taken in consideration when comparing the different approaches. Such criteria relate to the form factor, the acceptance, the usability, and the level of accomplishment.

The form factor (mass and volume of the sensor) is particularly important if the sensor is carried by the wearer.

The acceptance of the sensor is usually lower if the sensor is carried on the person (endosensor) rather than installed in his environment (exosensors). It is also influenced by the "visibility": a visible sensor (on the wrist, the belt, the chest...) may stigmatize the handicap or dependence. This can also be the case with external sensors visibly placed in the domestic environment. On the other hand some users want to have visible sensors to make them feel more secure. Acceptance is really subjective for different users.

The user friendliness depends on the autonomy, the range, and the packaging. The autonomy is estimated time between two successive maintenance interventions, that is, to change/reload batteries. The range is the maximum distance between the person and the sensor so that the latter can efficiently perform measuring. In the case of a carried sensor, communicating wirelessly, it is the distance between the base station and the sensor. The type of packaging concerns the biological innocuity, for instance biocompatibility, or the risk of injury for sensors directly in contact with the skin.

The level of "accomplishment" was added because some promising systems are still being tested while others are already available on the market place.

Non-fall tests (ADLs for non-accidental falls and carefully chosen activities for accidental falls) should be measured by real potential users of the targeted population of the system, i.e. elderly people. A minimum of five participants should perform all the series of chosen activities at least three times. If we have fifteen chosen activities, we will get a total of 225 (15x5x3) activity simulations. One series of activities should be measured as one. Ideally, it should be up to the participant in which order he or she will do the activities listed in that series. This is to make the scenario more closer to reality. The participant can also change the order of activities in the same series for the next rounds. The participant can also choose the pace to which he or she does the activities.

While the first four criteria are proposed by [4], this paper adds another criteria and that is adaptability. As we first mentioned at the beginning, a good sensor capable of adapting or tailor-suiting different individuals might be desirable.

(See also chapter 5 for further discussion about this topic.)

2.6 Commercial Fall Detection Systems

Most of the available commercial fall detection systems are one way or another a type of a panic-button. The set-up is always the same - a wearable accessory, a base station and a command center. Usually their price range can be from $20 to $25. An example can be found from this site [34]. It provides very little information about the device itself, but it is one of the cheapest out there.

One can also find alarm buttons that does not have a base station and emits a very high noise, 91 dB, when triggered to attract attention to the user. They roughly cost $13. We will not count these further into discussion because we find them lacking in several aspects to be a good fall detection system.

Dedicated assistance companies are the next improvement of the panic button system. The company acts as the command center monitoring the user twentyfour hours a day, every day of the year. They provide better assistance as they are better equipped and trained for emergencies. The downside is that you also pay several times more. They charge their users with monthly subscription fee in addition to a start and equipment fees. A monthly fee can be between $20 and $30 per month depending on what kind of services the company provides. An example is *Alert-1* [35]. In Norway a similar company *Hjelp 24* [36] has the same business model. Some general security companies in Norway also offer fall detection systems in addition to normal home security alarm mechanisms such as *Securitas* [37].

The above mentioned systems lack one vital specification to be a good fall detection system, and that is to have a device that can actually sense fall. They fail in noticing if the user has fallen, if the user is not lucid or conscious. The next type of systems have built-in sensors - or they are sensors themselves - that can sense and detect fall. *HealthSensor a.k.a iLife* from iLife Solutions [38] is the first example. It is a small wearable device that has advance microprocessors and accelerometers. Makers of iLife describe their system to be able to distinguish between serious falls and normal movement (ADL), even rough motions such as jogging or descending stairs. HealthSensor (shown in figure 2) can detect inactivity as well as falls. This can be helpful in some cases. Their website does not provide enough information about its cost, communication capabilities and other technical details.

Brickhouse Alert [39] is another fall sensor. This has the same concept as the company monitoring systems mentioned previously. The only difference is that the device on the user can sense falls making it effective for users who got unconscious upon falling. The company offers an additional service of intrusion/theft detection. Pricing is from $25 to $30. It would seem that the device is also using accelerometers to distinguish falls from normal movements.

Less popular fall sensors are pressure sensors, occupancy sensors, pull-chord sensors, and mobility sensors which can either be camera-based or infrared based. While the previously mentioned sensors are mainly for private users, these less popular sensors are usually used by health

(a) Picture taken from `http://www.ilifesolutions.com/products.html`

Figure 2: iLife HealthSensor

institutions such as hospital, home for the aged and other health care facilities. Pressure sensors as mentioned earlier detects pressure change on floors - in cases where a fall happens. Occupancy sensors are situated on bed or chairs (wheelchairs), so that if the user falls off the chair or bed it will be detected and the system would assume fall. Pull-chords are chords hanging all around the room of the user, so that in case of fall the user can pull on the chord which will then sound an alarm. Finally mobility sensor sets off an alarm if it does not detect some activity in a room within a period of time. Some companies/net stores have gathered all of these different types of devices and market them for sale. *Seekwellness* [40] and *tunstall* [41] are two such companies. Their products' price range can vary a lot.

Typical from these commercial products, are that they haven't been tested properly much more made a proper scientific comparison and study of. Their sensitivity and specificity is therefore impossible to say. Understandably the details of how these devices work internally and how they calculate a fall are well-guarded secrets of their makers. In general wearable sensors are more popular than ambiance systems. Companies that offer subscription to users for daily monitoring and assistance services are more costly but seem to give users more satisfaction by knowing that trained assistance or help is readily available.

There are also some mobile applications, which are usually free, that can be installed on mobile phones to act as fall detection system. Such applications partly depend on which mo-

bile phone they are installed on. Like the commercial devices above, these applications, as far as we are concerned, haven't been thoroughly studied. For iPhones, you can find an app called *Fall Alert* [42] and for Android you can find *mover* [43] and *cradar* [44]. It is not possible to see the codes and check their algorithm without asking the authors, but all of them seem to be accelerometer-based. At this point, we have only partially checked cradar and mover. They seem to use a very simple algorithm to calculate the total acceleration of the device. This rises a big problem of giving many false positives with normal movements.

Studying and evaluating these commercial fall detection systems together with what research groups are developing can give us a good rating system for these systems.

3 Choice of Methods

As we have seen in chapter 2, we have a plethora of fall detection systems if we take into consideration both the hardware, placement, algorithm(s) behind the system and finally the approach that was used. Although we have commented on some of the advantages and disadvantages of these different systems, it is outside the scope of our research to compare and study all of them indepthly. There is a need for a precision in what kind of fall detection system one wish to make and test. This chapter is a discussion of how one can make such decision. We will present in this chapter our own choices and specifications. We will also present our experiment that will later give us the information we need in making any algorithm we wish to implement.

3.1 The Targeted Users and Our Definition of a Fall

It is imperative to properly define both what a fall is, as well as who the targeted users of a fall detection system are, as explained in section 2.1. For the intents and purposes of the system we want to make, we define a fall to be a sudden change of body position coming to rest on the ground, excluding intentional change, followed by inactivity. This means that we only look at the most serious falls where the user lose consciousness after hitting the ground and therefore losing the ability to get help. We assume that a fall that still renders the user mobile and therefore conscious, has not suffered that severe of an injury that he or she cannot communicate to get help.

More specifically, our system first checks a sudden change of acceleration going towards a negative value (or towards 0 if the acceleration is measured and normalized to G-force units). This is an indication of a shift from a normal activity, i.e. ADL, to a fall. This is also the same as entering the critical phase from a pre-fall phase of a fall. Then the system checks for a change of the acceleration towards a positive value (or over 1 if the acceleration is measured and normalized to G-force units). This value becomes higher the harder the impact is. This means that the kind of surface the impact happened on matters a lot. How fast the user is falling down also changes this value.

Since we have defined that our system only recognize serious falls, we can simplify our definition of a fall to being *a fast deacceleration or acceleration towards the ground followed by a hard impact, and finally a moment of inactivity*. We will later put numbers on these values to clarify what we mean exactly by fast deacceleration or a hard impact.

When one has decided upon a specific group of users, e.g. elderly people, all possible falls that can occur for this group should be measured and later tested. Ideally, the measured falls are from the users themselves. One way to do this, is to continuously monitor this group of user for

a long period of time, like what was done in [10]. This poses one big disadvantage though. Even for a period of 1 year, statistically it is hard to observe real falls in real environments. This was true in the case study of [45]. A very long observation time, possibly in the span of ten years or more, is needed to get a statistically significant number of falls to base a fall detection system on. To counter this difficulty, one can instead simulate these expected falls with actors.

The actors that will simulate the expected falls from the targeted user group, should stay true to how the falls are in reality. This is done by either studying video recordings of actual falls from that group like in [45], or reading scientific literatures describing these falls in specifics, or by talking to knowledgeable experts for the targeted users, e.g. geriatric doctors/workers, physical therapists, neurologists, health-care providers, and probably others.

Also depending on what kind of algorithm(s) that will be later used, depicting all other circumstances of how the real falls would be is a necessity. Which means that if your algorithm is measuring impact, it is important to simulate falls hitting on real ground surfaces (usually hard ones) and not use any cushions. This, of course, makes it hard, but not impossible to find qualified enough actors (or rather stuntmen) to simulate the falls on how they should be. A good alternative is therefore using a good dummy representing the typical targeted user .

As we are only testing generic types of falls using the bare minimum (see section 2.1), we will not assume any specific group of users. It will be a mistake to say that our system is made specifically for the elderly people for example. Elderly people fall in so much more ways than just how we will specify. They can fall from bed, chair or while using a rollator/walker, which then produces a somewhat different fall pattern. If needed, a list totaling of nineteen different actual falls observed from a long time study of elderly people, can be found in the research of [45].

3.2 The Chosen Approach

We want to implement our fall detection system as a wearable device, more specifically in a smartphone(see the next section). Furthermore, we will look at acceleration-only methods. There are enough algorithms under this method to be used and compare with. Another reason for this, is that most smartphones (probably all) and some mobile phones already support acceleration-based methods, while only a subset of these phones support non-acceleration methods like magnetic-based or angular-based methods. This means that a successful implementation of our system, will result in better pervasiveness of the product. Acceleration-based systems, more than quite often give better accuracy and results than not basing the system on this approach. This is the same findings as in [13]. Lastly, just using acceleration-only methods is enough to analyze all phases of falling as describe in section 2.1. Because of this, we have purposely compiled a number of specific equations and algorithms used in wearable acceleration-based fall detection systems in section 2.4.

In choosing between analytical approach and machine learning approach, we choose the ana-

lytical approach. This is because machine learning algorithms are mostly designed to run on powerful desktop machines, whereas mobile phones are constrained devices both in terms of computation and energy budget. More lightweight and scalable algorithms are needed for mobile phones in order to preserve the phones user experience (i.e., long battery life and little computation requirements while running a mobile sensing application). There is also a need fo a powerful device to be able to efficiently collect and analyze large scale training data sets as a basis to build more robust classifiers. Even though smartphones are more powerful than the average mobile phones, we still want to conserve its resources as much as possible. A similar assessment was done in [46].

Learning and implementing efficient machine learning algorithms is also usually more difficult. In the course of our literature study, we also have found better performances with systems not using machine learning techniques. This is of course not to say that it is impossible to implement a good fall detection system that uses machine learning techniques.

In analytical approach, one typically sets up an experiment for data gathering. These data are later analyzed and would be the basis of the fall detection system. In theory, one can get these data even without the experiment. This is done by studying previous researches and findings, going through a database which was formed either by statistical agencies or other scientific communities, talking to a panel of experts (focus group) or a combination of any or all the methods just mentioned. We will however stick to performing our own experiment and data gathering. This experiment is presented in section 3.4.

3.3 Why a Smartphone?

We have chosen to implement our system in a smartphone [1][2]. A smartphone is a very powerful device and includes a range of different sensors. The most notable one is the accelerometer. Smartphones are lightweight, highly portable and effectively make our fall detection system *pervasive*. Unlike commercial fall detection devices i.e. the panic button, smartphones can be used both indoors and outdoors. Almost everyone owns a mobile phone and increased popularity of smartphones is also observed. This will enable our system to reach more potential users than if the system is developed on a specially made hardware. Implementing the system in a smartphone combines the fall detection part of the system with the communication part, i.e., upon detecting a bad fall the system calls for help either through SMS, phone call, e-mail or combination of all. Furthermore smartphones offer the possibility of communicating with other devices if needed through e.g. *bluetooth*. Making our system location-aware either by using GPS or W-LAN or both is also feasible. Lastly our chosen OS platform is Android [47]. Applications made for Android can be cheaply made.

[1] For the purpose of this study the choice of the term *smartphone* and not *mobile phone* is deliberate. This is to make room for that the system we expect to make may demand a more powerful platform, i.e. a smartphone, than a traditional mobile phone.

[2] At this point we will not exclude that the system could also be deployable on less powerful mobile phones as we will only be testing accelerometer-based algorithms in this study.

In [48] they even use a phone's camera to take a picture of the patient's surroundings if a fall is detected. The picture taken together with sensor data (based on acceleremoters) and location data (based on the integrated GPS) is then sent to a nearby emergency responder. The picture by itself might not be valuable and perhaps will not show anything important, i.e. the phone is inside the user's pocket, however in other cases the picture might provide additional context information to the responder.

There are several different approaches in fall detections in terms of what kind of equipments and set-ups we could use. Using a smartphone with its integrated sensors (accelerometer, gyroscope, etc.) is only one of them. In most cases combining several equipments/sensors results in getting better performance of the developed system. For the purpose of this paper we envision our first system to just focus on plain fall detection. We will however open the idea for further development of the system by using additional devices [3]. The possibilities and advantages of using the smartphone as the chosen platform, as described in the previous paragraph, will still be open although not a part of the first prototypes of the system.

A major criticism to wearable sensor-based approaches is that, the user may forget to charge or wear the sensor, therefore leaving the system in a non-functioning state. With the integration of accelerometers on smartphones, it has become possible to develop fall detector applications that can run on the smartphones. Since people are more likely to carry their phones with themselves, rather than an additional sensor, smartphones with integrated accelerometers can easily be used for pervasive fall detection. In fact, with mobile phones the usual practice is that users including the elderly people, are used to carrying them making it less likely to forget the fall detection device.

Lastly, fall algorithms made for mobile phones do not usually rely on external servers, making it autonomous.

For the rest of the study, we assume that the user will use the proposed system as if it is a normal application in a mobile phone. This is to stay true on how normal users use their mobile phones. As such we will also be able to evaluate mobile phones as fall detection platform.

3.4 The Experiment

Before making any fall detection system, we need to gather sufficient data which will become the foundation of our fall detection system.

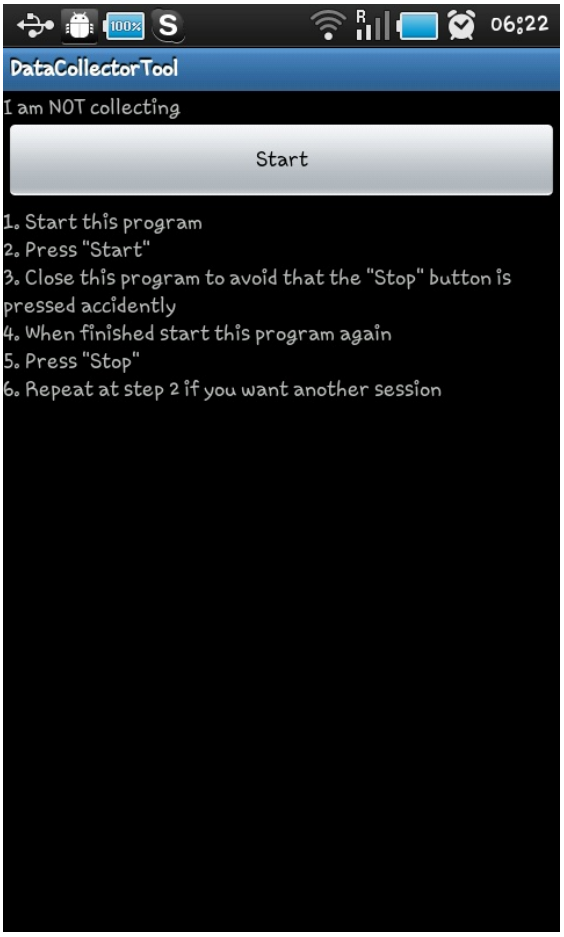
For this purpose, we have acquired the smartphone[4] Dell Streak 5 running Android 2.2 as its operating system. The phone's other technical specifications are as follows;

[3] See section for future work

[4] this phone is also considered to be a *mini tablet*.

- Processor: 1 Ghz
- Dimension: 6 x 3.1 x 0.4 in
- Weight: 218.29 grams
- Memory: 512MB SDRAM
- Communication and Connectivity: GSM/EDGE, UMTS, HSDPA, wifi, bluetooth, USB
- Sensors: Accelerometers(frequency measured to be 46HZ), Ambient Light Sensors, GPS, Magnetometers/e-compass, Capacitive Sensor keys, Proximity Sensors
- Miscellaneous: Camera, Microphone, Speakers, Multi-touch

Figure 3: Data Collector Tool by Mohammad Derawi



Using a program written in Android (fig. 3), we proceeded to measure five types of falls. These are *1)forward fall like a stick, 2)backward fall like a stick, 3)lateral "stick" fall towards*

right, 4)lateral "stick" fall towards left and 5)forward fall collapse (see appendix A). The main difference with falling like a stick and a collapse, is that a "stick" fall does not have knee flexion and the entire body is seemingly rigid - and thus the expression "like a stick" - while falling, while a collapse is non-rigid often much slower than its stick fall counterpart and gives a more unpredictable outcome in terms of speed, body position and orientation and direction of fall.

The program recording the different falls, were recording both acceleration and orientation data in the X, Y and Z axis as well as magnetometer data. Only the acceleration data will be used. Actually, later when we are comparing algorithms we will also use orientation based approach, but this will be based on acceleration data themselves. It will be shown later how you can compute the orientation from the acceleration data. **Note:** The orientation measured by accelerometers should not be confused by rotation measured by gyroscopes.

A group of three volunteers ages 25 to 28 years old, 1 female and 2 males is used to simulate the falls. The weight varied from 64kg to 100kg, while the height was from 164cm to 173cm. All of them performed each of the 5 different falls three times, giving a total of 45 (3x3x5) data sets. In addition to the first group, there is a second group which consisted of using a dummy which we calibrated the weight of to alternately change from 36kg to 28kg and finally to 26kg. The dummy will then represent another three individuals.

In addition to performing the different falls, group 1 also performed chosen ADLs and some other non-fall activities to compare data to (see Appendix B). These activities will later be categorized as ADLs, medium activities, and hard activities.

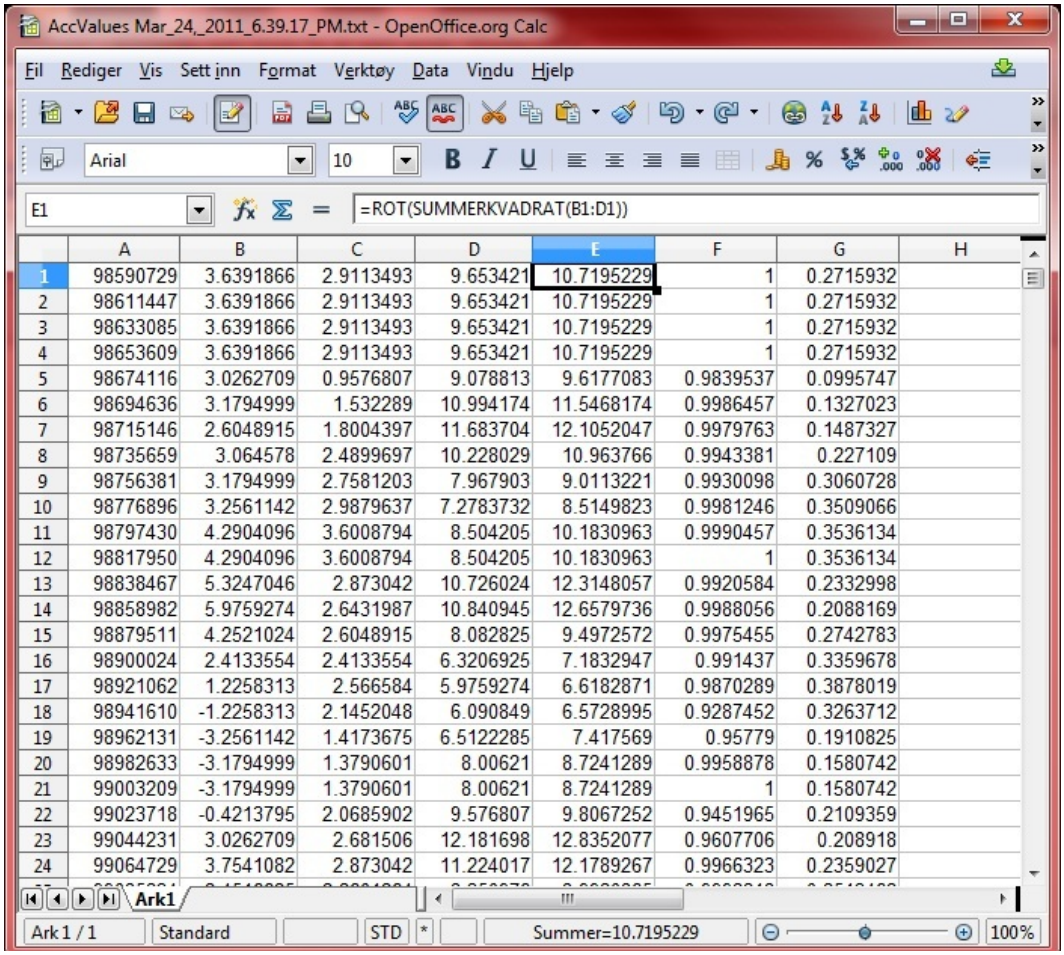
The phone was placed in the users' pants' pockets, where we have assessed to be the most natural place for people to put their mobile phones. In practice the phone's placement is in between the user's waist and the front upper-part of the thigh/femur.

Each fall was measured and recorded individually. The volunteer stands upright steadily (the start position) for at least five seconds. This is to better clarify later, what time the fall has started. Immediately after falling, when the volunteer is already in the post-fall phase, the volunteer is asked to remain immobile on the ground for another five seconds. This again, is to make it easier later to distinguish the phases of the fall in the data analysis. Finally, the volunteer rises up and remain static for another five seconds to mark the recovery phase as well as the end of the significant data.

For the simulation of ADLs and non-fall activities, a similar approach was done. To improve the time of the simulations, we clustered the activities in groups where it is natural. This means for example that activities 1, 2 and 5 were done as one group and in succession. To mark the shift from one activity to another as well as the phases of each activity, the volunteer pauses for five seconds in each phase of activity and in between activities (just like it was done for the fall simulations).

It was noted if there were any anomalies in the experiment.

3.5 Results and Data Analysis



	A	B	C	D	E	F	G	H
1	98590729	3.6391866	2.9113493	9.653421	10.7195229	1	0.2715932	
2	98611447	3.6391866	2.9113493	9.653421	10.7195229	1	0.2715932	
3	98633085	3.6391866	2.9113493	9.653421	10.7195229	1	0.2715932	
4	98653609	3.6391866	2.9113493	9.653421	10.7195229	1	0.2715932	
5	98674116	3.0262709	0.9576807	9.078813	9.6177083	0.9839537	0.0995747	
6	98694636	3.1794999	1.532289	10.994174	11.5468174	0.9986457	0.1327023	
7	98715146	2.6048915	1.8004397	11.683704	12.1052047	0.9979763	0.1487327	
8	98735659	3.064578	2.4899697	10.228029	10.963766	0.9943381	0.227109	
9	98756381	3.1794999	2.7581203	7.967903	9.0113221	0.9930098	0.3060728	
10	98776896	3.2561142	2.9879637	7.2783732	8.5149823	0.9981246	0.3509066	
11	98797430	4.2904096	3.6008794	8.504205	10.1830963	0.9990457	0.3536134	
12	98817950	4.2904096	3.6008794	8.504205	10.1830963	1	0.3536134	
13	98838467	5.3247046	2.873042	10.726024	12.3148057	0.9920584	0.2332998	
14	98858982	5.9759274	2.6431987	10.840945	12.6579736	0.9988056	0.2088169	
15	98879511	4.2521024	2.6048915	8.082825	9.4972572	0.9975455	0.2742783	
16	98900024	2.4133554	2.4133554	6.3206925	7.1832947	0.991437	0.3359678	
17	98921062	1.2258313	2.566584	5.9759274	6.6182871	0.9870289	0.3878019	
18	98941610	-1.2258313	2.1452048	6.090849	6.5728995	0.9287452	0.3263712	
19	98962131	-3.2561142	1.4173675	6.5122285	7.417569	0.95779	0.1910825	
20	98982633	-3.1794999	1.3790601	8.00621	8.7241289	0.9958878	0.1580742	
21	99003209	-3.1794999	1.3790601	8.00621	8.7241289	1	0.1580742	
22	99023718	-0.4213795	2.0685902	9.576807	9.8067252	0.9451965	0.2109359	
23	99044231	3.0262709	2.681506	12.181698	12.8352077	0.9607706	0.208918	
24	99064729	3.7541082	2.873042	11.224017	12.1789267	0.9966323	0.2359027	

Figure 4: Data from the Accelerometers Uploaded in OpenOffice Calc

The gathered data from the Android Data Collector was uploaded to a computer. Each data readings was first transformed to an *OpenOffice Calc* [49] file, which is a spreadsheet for mathematical and statistical computations. The total acceleration of the device, also called the RSS, (see eq 2.1) was computed to form a separate column in the spreadsheet.

A sample data set is shown in figure 4. Column A is the time, each row represents a timestamp which is measured in macro seconds or 1 x 10^{-7} seconds. Column B, C and D are accelerometer data from the phone's X, Y and Z axes respectively. Column E is the computed RSS values. Column E is calculated based on equation 2.10, while Column F is based on equation 2.9. The

values from both column F and G are not yet transformed to their angular values by calling the inverse cosine function. This transformation is done in MatLab.

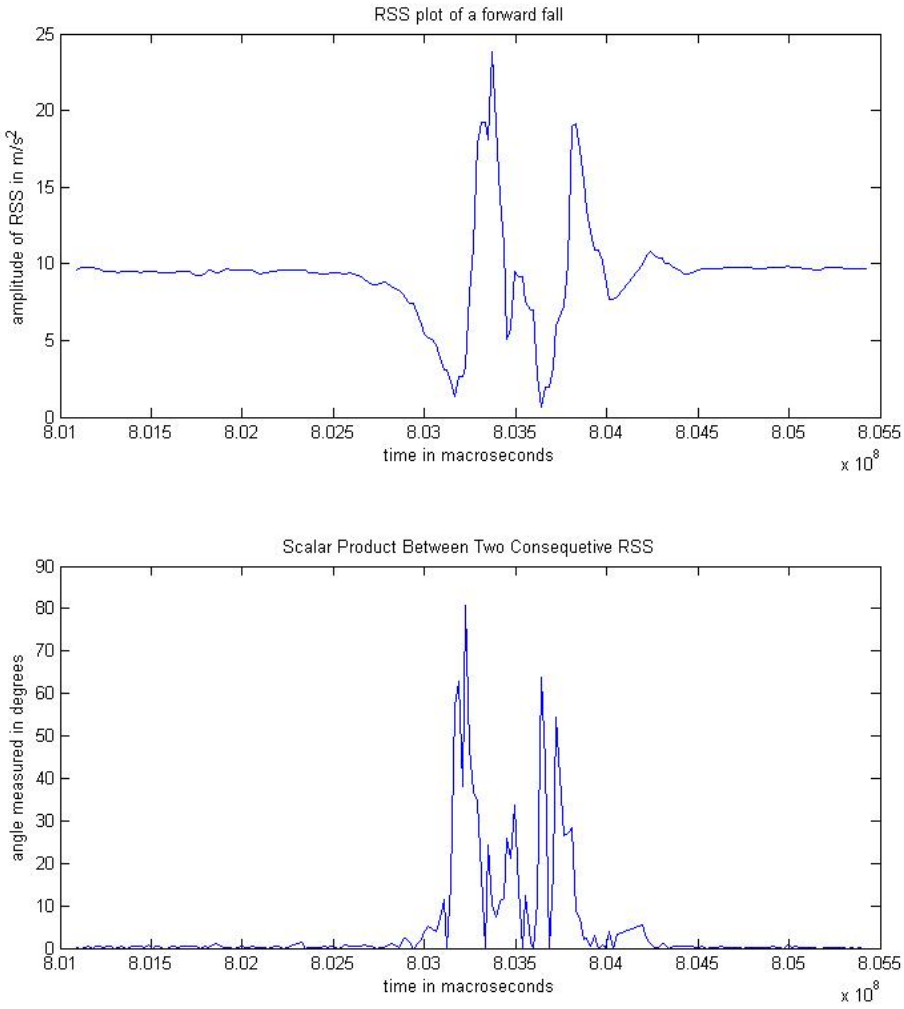


Figure 5: A graph representation of a forward fall

Further data analysis, computations and graphic formations were done in MatLab. Each data-set was studied and segmented to find and distinguish the different phases of a fall (see section 2.1) or an activity if it is an ADL/non-fall activity dataset.

A typical graph for a fall is shown in figure 5. The upper graph is a RSS plot, while the lower

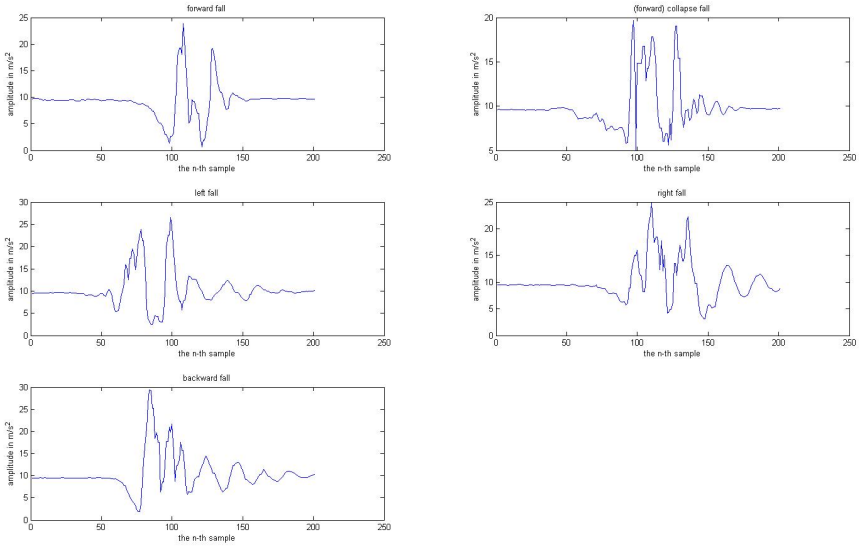


Figure 6: Falls from Group 1

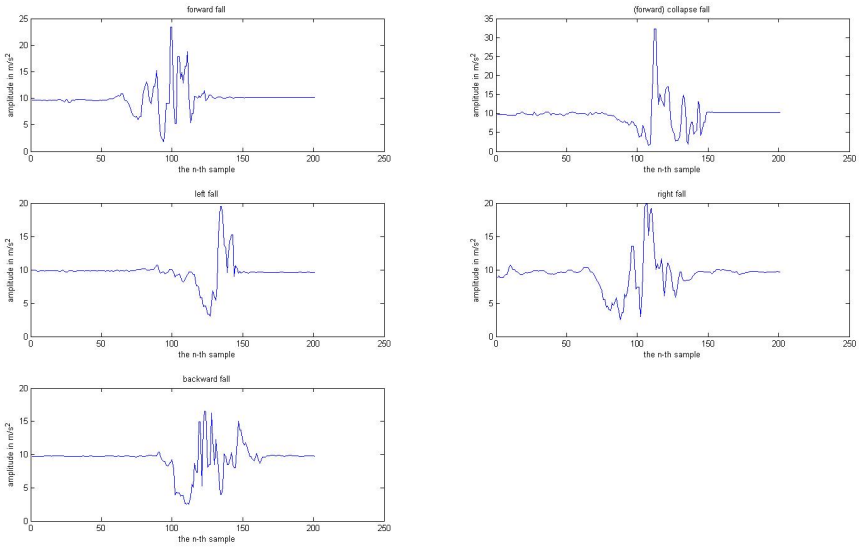


Figure 7: Falls from Group 2

graph shows the scalar product between two consecutive RSS reading [5] (see eq. 2.10). The high peaks of the lower graph directly correspond to the low peaks of the upper graph. This means

[5] using the full acceleration vectors and not only the RSS itself

that we can combine a profiling algorithm that uses LFT and equation 2.10, which will give a higher specificity. Also when at rest or in static posture, the upper graph reads almost 10 which is almost the same as 1G. In the lower graph, static posture is roughly close to 0. This is because in a static position, there is none or minuscule change in the acceleration vectors between two consecutive sensor readings. Thus, we can also use this for inactivity detection. A sample set of falls from group 1 (figure 6) and group 2(figure 7) shows similarity with each other.

In the sample fall, the entire fall happened between timestamp 8.027 and 8.043, which translates to approximately 1.6 seconds in this case. The body first touches the ground in the first negative peak (the impact point). This is where the lower fall peaks (LFP) are based on. Upon impact, the body registers an opposite force giving the upward peak which the upper fall peaks (UFP) are based upon.

The data we are interested in are: the LFPs, UFPs, time between the LFP and UFP (eq. 2.4), falling-edge and rising-edge time (2.4), and the angle between two consecutive measurements (eq. 2.10).

The fact of the matter is that there will be several downward and upward spikes in a fall. Choosing what peaks to represent an individual fall, all depends on what is the focus of your algorithm. An algorithm solely based on UFPs, only needs to look at the highest spike within the fall period (the critical phase of fall), but once a more complex profiling algorithm is used, choosing these peaks would be important.

Similar to the algorithm that was used in [30] and [31], we picked the upper and lower peaks with respect to the time difference between them. It is also a condition that a lower peak comes before an upper peak, as all falls show a prominent downward spike before an upward spike. This first set will the basis for the *first algorithm*.

For the *second algorithm*, the time between the upper and lower peaks are of less importance. The peaks are solely chosen if they represent the maximum peaks during a fall. The only other thing that matters is that the lower peak comes before the upper peak. The formed data set will be the basis for the second algorithm.

As shown in figure 5, a downward spike in a fall in the upper graph, results in having an upward spike in the lower graph (from equation 2.10). With this in mind, we first choose the lowest peak in a fall that gives the highest corresponding angular peak. Then we choose the upper peak directly connected to the previous chosen lower peak. The compiled data from both group 1 and group 2, are shown in tables 3, 4 and 5. This is the basis for the *third algorithm*. **Note**: The value of Φ is only based on the data from the critical phase of a fall.

The computation of the falling-edge time and rising-edge time were added, because we wanted to see if it will help change the sensitivity of the fall detection system. These two time values

Table 3: Compiled Fall Data from Group 1

-	**Min**	**Max**
LFP	1.3408	8.7308
UFP	15.4099	31.0996
$RSS_{UFP\text{-}LFP}$(eq. 2.4)	41.022ms	312.928ms
$\Phi(eq.\ 2.10)$	51.4881 deg	158.0417 deg
Falling-edge time (2.4)	123.128ms	512.887ms
Rising-edge time (2.4)	20.525ms	121.177ms

Table 4: Compiled Fall Data from Group 2

-	**Min**	**Max**
LFP	1.5183	6.3809
UFP	15.4099	32.2497
$RSS_{UFP\text{-}LFP}$(eq. 2.4)	40.979ms	184.564ms
$\Phi(eq.\ 2.10)$	64.2819 deg	165.3800 deg
Falling-edge time (2.4)	130.465ms	512.865ms
Rising-edge time (2.4)	41.006ms	161.188ms

Table 5: Compiled Fall Data from Groups 1 and 2

-	**Min**	**Max**
LFP	1.3408	8.7308
UFP	15.7667	32.2497
$RSS_{UFP\text{-}LFP}$(eq. 2.4)	40.979ms	312.928ms
$\Phi(eq.\ 2.10)$	51.4881deg	165.3800 deg
Falling-edge time (2.4)	123.128ms	512.887ms
Rising-edge time (2.4)	41.006ms	161.188ms

can only be calculated after the lower and upper fall thresholds(LFT and UFT) are decided. This means that they can also be based on and used in the first two algorithms. However, we have not tested adding this in our prototype (see chapter 5 for the reasons).

Here is a recap of the main difference between the three algorithms so far;

1. The thresholds in the first algorithm were chosen in respect to the time between a lower peak and an upper peak. The more these times (from all the falls) are similar to each other, the better.
2. The thresholds in the second algorithm were chosen in respect to their absolute(max and min) peak values.
3. The thresholds in the third algorithm were chosen in respect to the angle between two consecutive acceleration readings.

Results from the ADLs and non-fall activities (see appendix B) were also analyzed. We have grouped these activities into three categories. These are;

1. *Light Activities or the minimum ADL list*. These are what most researches include when they measure ADLs in comparison to falls. Most elderly people should still be capable of accomplishing these activities. This includes sitting on a chair, getting off a chair, lying down on a table (simulation of a bed), standing up from the table and walking.
2. *Medium Activities or the extended ADL list*. These are: sitting on the ground, standing up from the ground, picking up an item on the ground, kneeling, standing-up from kneeling position, lying down on the ground, rotating while lying on the ground, standing up from the ground (from lying position), soft/slow kicks, and finally walking up and down a staircase. These are activities an average person should still be able to do.
3. *Hard Activities or non-fall/non-ADLs*. These are activities a physically active person should be able to do. These are; sitting fast on a chair, standing up fast from a chair, lying down quickly on the ground, push-ups, standing up quickly from the ground, sit-ups, fast kicks, bicycling, going up an elevated platform (0.8-1m in height), jumping down from an elevated platform (0.8-1m in height), vertical jump (from normal standing position), running, forward leap (from a running start), running up and down a staircase, and finally jogging.

Boxplots are used to show the rest of the statistical results of the data. This is because in this case it is easier to see the overlaps between the falls and the activities. Figure 8 shows boxplots
6

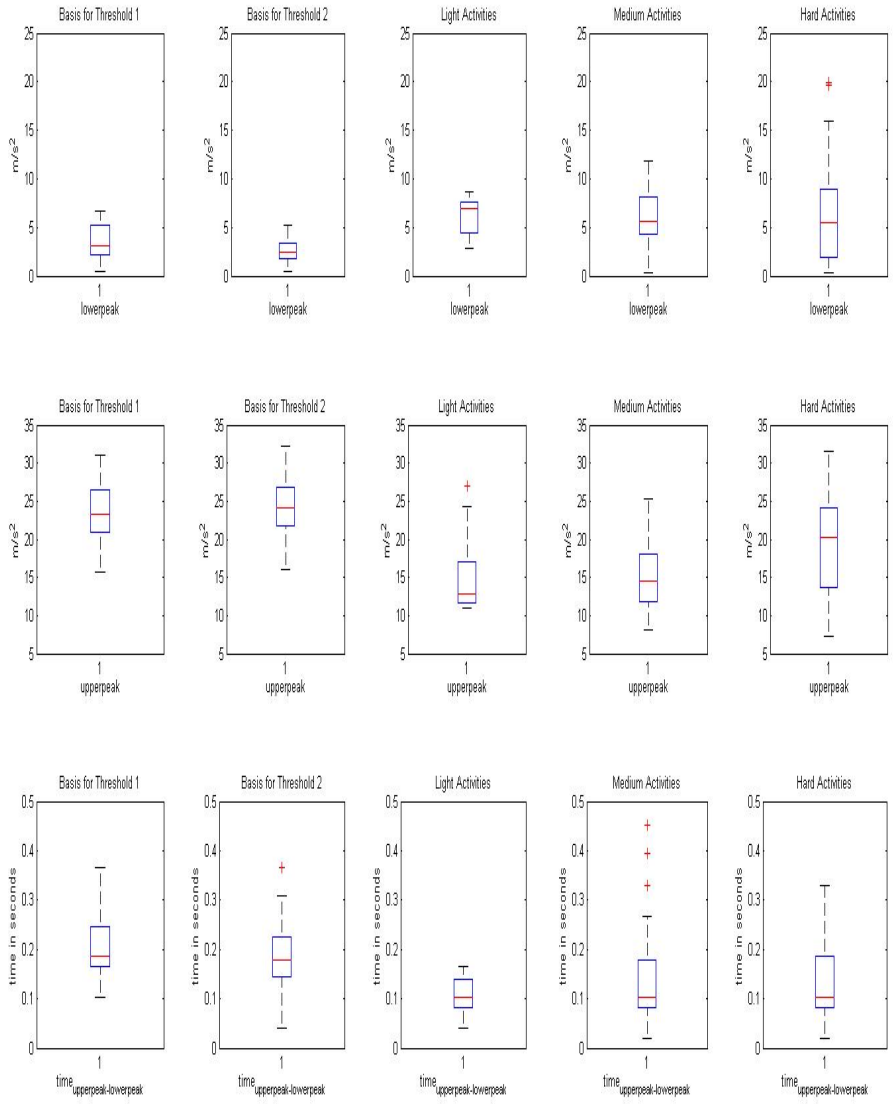


Figure 8: Boxplots from both the Data Set of Falls and ADLs/non-fall activites

4 Prototype

Basing on the values from the previous experiment (see section 3.4), we analyze the resulting data to get our own thresholds. With these thresholds and a number of chosen equations (see section 2.4), we will build our own algorithms and try to evaluate their performance (see section 2.5).

4.1 Creating the Prototype

A prototype of our system is depicted in figure 9. As mentioned, it is developed and tested in Android for API versions 7 and higher. The application was run and tested in three different mobile phones; Dell Streak, HTC Hero and Samsung Galaxy S. The application consists mainly of three parts;

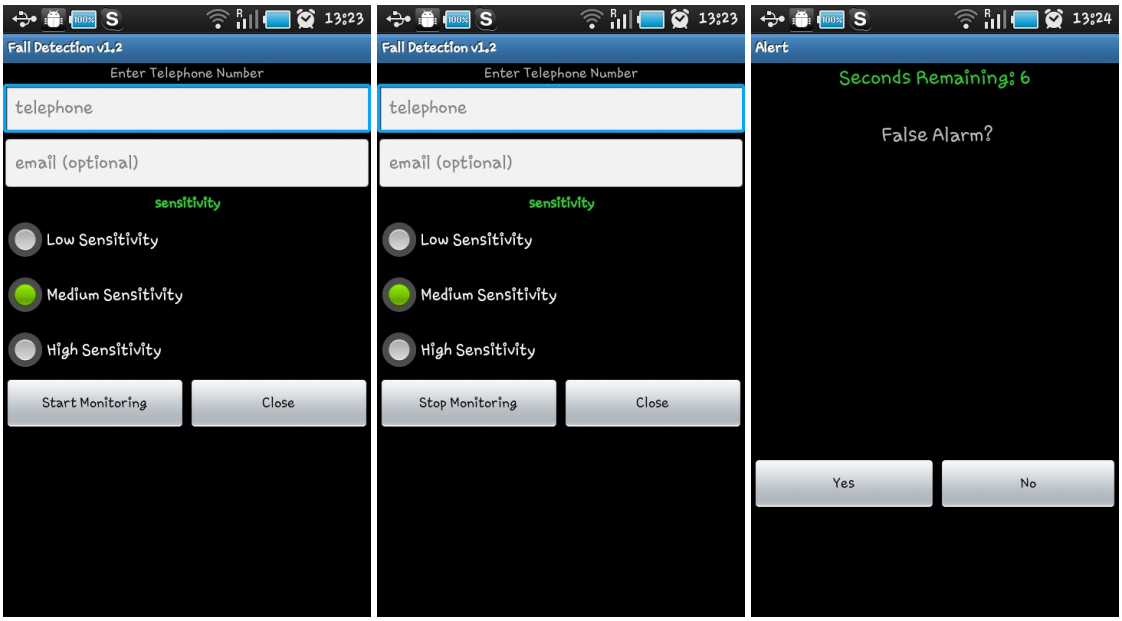
1. *FallDetection.java* : This is an activity class that shows the GUI-interface of the application. Users can choose the sensitivity level*, telephone number and an optional e-mail address for which a fall alarm will be sent to. This is also where the user starts and stops the application for fall monitoring.

2. *SensorService.java* : This class starts a service which is similar to a Daemon process that runs in the background. The service implements a class listener to access the mobile phone's accelerometer (and possibly other sensors). The actual fall detection algorithm is also found in this class. Implementing a service conserves much of the phone's resources.

3. *Alert.java* : This is another activity class which shows the user a notification if a fall is detected. In here, an alert message can be sent out either via SMS, e-mail, phone call or any combination of them. In the current prototype, only outgoing SMS and phone calls were implemented.

* Because the prototype was only being tested by the research group and not for any actual users yet, we have disabled the sensitivity options, as well as hardcoding a phone number to send a SMS and call to in case of fall alarms. The sensitivity options are meant to change the threshold levels depending on the type of user. Low sensitivity would use the thresholds which are primarily based on Hard Activities (see section 3.4) intended for active users. Medium sensitivity is based on Medium Activities intended for the average user. High sensitivity is based on Light Activities and is intended for people with limited daily movements such as an elderly person. For the rest of the discussion, we will assume this test version.

Upon opening the application, the user is presented with the interface shown in figure 9a. At this point, the fall detection mechanism has yet to be started. Pushing the "Start Monitor" button



(a) Start of Application

(b) Turning on The Fall Detection Mechanism

(c) Fall was Detected and Started an Alarm With a Counter

Figure 9: Fall Detection v1.2

will start the fall detection mechanism, and the button is changed to “Stop Monitoring”.

As mentioned, the version that was used in the rest of our study was intended for ease of testing and not yet for real users. To be able to compare our chosen algorithms, three different services are started instead of one. Each service represents a different algorithm as explained in section 3.4.

If one or more of the services, which is now monitoring the user, detect a fall they will issue their own alarm notification. Thus in the test version, if a fall triggers all 3 services, 3 different alarms will be issued. Each alarm notification will also show from which service it was generated from.

An alarm notification (fig. 9c) will give a user 20 seconds to respond. Failing to respond to this will force the system to send an alert message to the predefined phone number. By default, the alert message is sent via SMS and a phonecall. If the user becomes unconscious and not able to talk, then the alert message is still conveyed via the SMS.

If the user falls but remains mobile and conscious, and the system detects the fall, the user can choose if he or she will still send an alert message depending on how badly hurt he or she is. The user may want to ask for help even though he or she is still conscious. In this case, the

user can either wait until the timer runs out or push the “no” button. Pushing the “yes” button will cancel the alert notification and the system will not send any alert message.

Appendix C provides the most important codes used in the application. A pseudo code of the first algorithm that is used in service 1 is shown in algorithm 4.1.

Algorithm 4.1 Algorithm 1

```
while service_is_running do
  Check for Lower Fall Threshold
  if RSS ≤ 6.67 then
    LFT_exceeded = true
  end if
  Check for Upper Fall Threshold
  if RSS ≥ 15.76 then
    UFT_exceeded = true
  end if
  Check That Both Thresholds are Exceeded
  if LFT_exceeded = true and UFT_exceeded = true then
    Check That LFT happens before UFT and the Time in Between
    if UFT_timestamp − LFT_timestamp ≥ 0.102sec and UFT_timestamp −
    LFT_timestamp ≤ 0.366sec then
      Assume Fall
      Fall_detected = true
    end if
  end if
  Check Inactivity Period After Assumed Fall
  if Fall_detected = true and Inactivity_period > 0.520sec and Inactivity_period <
  3.500sec then
    Issue an Alert Notification
  end if
end while
```

A pseudo code of the first algorithm that is used in service 2 is shown in algorithm 4.2.

A pseudo code of the first algorithm that is used in service 3 is shown in algorithm 4.3.

All three algorithms employ an inactivity detection as mentioned earlier. We chose to use the scalar product between 2 consecutive readings to measure inactivity. If in at least 5 consecutive acceleration readings, the scalar product is lower than 3 degrees, depicting none or minuscule change between the readings, then a period of inactivity is detected. In our case, this is set to at least be 0.087 sec. We chose such short inactivity time so the system will not prolong sending an alert issue if it is a real fall. This is because, there will be a last confirmation of inactivity in the form of the 20 seconds alarm notification.

Algorithm 4.2 Algorithm 2

```
while service_is_running do
  Check for Lower Fall Threshold
  if RSS ≤ 5.26 then
    LFT_exceeded = true
  end if
  Check for Upper Fall Threshold
  if RSS ≥ 16.08 then
    UFT_exceeded = true
  end if
  Check That Both Thresholds are Exceeded
  if LFT_exceeded = true and UFT_exceeded = true then
    Check That LFT happens before UFT and the Time in Between
    if UFT_timestamp − LFT_timestamp ≥ 0.041sec and UFT_timestamp −
    LFT_timestamp ≤ 0.366sec then
      Assume Fall
      Fall_detected = true
    end if
  end if
  Check Inactivity Period After Assumed Fall
  if Fall_detected = true and Inactivity_period > 0.520sec and Inactivity_period <
  3.500sec then
    Issue an Alert Notification
  end if
end while
```

Algorithm 4.3 Algorithm 3

```
while service_is_running do
  Check for Lower Fall Threshold and Orientation
  if RSS ≤ 8.73 and Orientation ≥ 48.39 then
    LFT_exceeded = true
  end if
  Check for Upper Fall Threshold and Orientation
  if RSS ≥ 15.76 and Orientation ≥ 5 then
    UFT_exceeded = true
  end if
  Check That Both Thresholds are Exceeded
  if LFT_exceeded = true and UFT_exceeded = true then
    Check That LFT happens before UFT and the Time in Between
    if UFT_timestamp − LFT_timestamp ≥ 0.041sec and UFT_timestamp −
    LFT_timestamp ≤ 0.313sec then
      Assume Fall
      Fall_detected = true
    end if
  end if
  Check Inactivity Period After Assumed Fall
  if Fall_detected = true and Inactivity_period > 0.520sec and Inactivity_period <
  3.500sec then
    Issue an Alert Notification
  end if
end while
```

4.2 Testing the Prototype and its Results

We have used two ways for testing the prototype;

- *Controlled test*. This is similar to the previous experiment where both fall simulations and ADL/non-fall activity simulations are performed by a group of volunteers.
- *Daily "continuous" test*. A group of users is asked to use the system in their daily activities as much as possible. During the logged hours the system is being used, the number of times the system issues an alarm is counted. It is also noted if the alarm was a false alarm or a true detected fall.

A different set of volunteers was used for testing this prototype. The group consisted of 2 females and 6 males with ages ranging from 22 to 54, height ranging from 164cm to 185cm, and weight ranging from 65kg to 90kg. A dummy weighing now 25kg was also used in the test. The system was tested against a total of 120 falls and 696 ADLs/non-fall activities. Table 6 shows the sensitivity measurement for the 3 algorithms that were tested, divided to the different types of falls.

Table 6: Sensitivity Table

-	**Forward Fall**	**Left Fall**	**Right Fall**	**Backward Fall**	**Collapse**
Algorithm 1	56.0%	52.0%	27.3%	50.0%	29.2%
Algorithm 2	44.0%	40.0%	9.1%	41.7%	20.8%
Algorithm 3	56.0%	24.0%	13.6%	41.7%	16.7%

The specificity levels for the 3 different algorithms are shown in table 7.

Table 7: Specificity Table

-	**Light Activities**	**Medium Activities**	**Hard Activities**
Algorithm 1	100.0%	99.1%	99.7%
Algorithm 2	100.0%	99.5%	99.7%
Algorithm 3	100.0%	99.1%	100.0%

Combining all the data from the test, an overall evaluation of the system based on the 3 algorithms can be calculated. The results are shown in table 8.

Table 8: Overall Results of the 3 Algorithms

-	**Volunteers**		**Dummy**		**All**	
-	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
Algorithm 1	41.2%	99.6%	55.6%	-	43.3%	99.6%
Algorithm 2	31.3%	99.7%	33.3%	-	31.7%	99.7%
Algorithm 3	32.4%	99.7%	22.2%	-	30.8%	99.7%

For the daily "continuous" test, only two users have submitted back their evaluation. Approximately 425 hours were logged between the 2 users. During this time, algorithm 1 had a total of 8

alarms, algorithm 2 had a total of 6 alarms, and algorithm 3 had 8 alarms. All alarms were false, that is no real fall happened, although in one of the alarms, it was the case of the user suddenly throwing himself on the bed which looked for all purposes like a fall.

Further discussion about the experiment, prototype and results can be found in chapters 5 and 6.

4.3 Previous Versions and Tests

Prior to making and testing our final prototype as described in this chapter, preliminary tests and earlier prototypes were made. Assessment of these earlier studies would be the foundation for the final prototype.

The first prototype that was made, did not have any inactivity detection but uses all 3 algorithms mentioned in section 3.4. A limited testing of the first prototype using the dummy, showed that all three algorithms gave 100% (11 true positives/11 real falls) in sensitivity. However, the specificity was in an untolerable level. During a test period of approximately 2 hours, a false alarm was issued by the application multiple times when the test subject was walking, jumping, running or making other sudden movements. These results were in direct accordance with the data from the previous experiment and were to be expected.

For our first attempt in implementing a system with inactivity detection, only algorithm 3 (described in section 3.5) was used in addition to the inactivity detection algorithm. To compensate for the low specificity levels from the previous prototype, we have raised the thresholds into higher levels. Testing this prototype version was done in a similar fashion as the final prototype.

A group of 2 females and 1 male different from group 1 that was used in the experiment, volunteered for testing this prototype. The age varied from 23 to 27, the height from 159cm to 174cm, and the weight from 45kg to 80kg. The dummy, estimated now to weigh 25 kg, was also used in the test. The results of the test showed that the system achieved a sensitivity of 62.2% (23 true positives/37 real falls) and a specificity of 96.7% (176 true negative/ 182 total activities). Also in approximately 145 logged hours of daily “continuous” testing, only 2 false alarms were detected. The first one was when a user was playing basketball and the second one was when a user was kneeling on the ground and wiping(cleaning) the floor. No real falls occured during this period.

5 Problems and Discussion

This chapter will present our assumptions, considerations, decisions and the discussion that followed. Chapter 3 already shows some of these discussions when we presented our choice of methods.

5.1 Discussion About the Experiment

As far as we have gathered, there are 4 main ways to conduct an experiment to both train (gather data for) and test a fall detection system. The first one is to gather all data in one experiment. The gathered data will then be divided in 2; one half is used as training data and will be the basis for any thresholds, and the other half is used for testing. Thus a sufficient amount of data is needed.

A second method for the experiment is to separate the experiment for data gathering and the experiment for testing. A third method is to have an experiment for data gathering and use a special software like *LabVIEW* [50], to generate new random data based on some parameters and some training data, for testing.

The third method was used in [31], where they only had 3 volunteers: 2 males (42 and 48 years) and 1 female (38 years) for 3 types of fall simulations. Each fall simulation was done at least twice. A second group of 1 male (22 years) and 1 female (38 years) were used to perform a list of ADLs. In testing the prototype for its sensitivity, LabView was used, which simulated other falls based on the fall samples from the experiment.

Lastly, one can reuse training data from previous studies or relevant database and test the final product with real people. In [30], they have used an algorithm which they claim to be an improvement from a previous research. For testing their system, they set up two different approaches. The first one is by using 10 young subjects to perform both ADL and falls. In the second approach, the 10 young subjects performed the falls while 10 elderly subjects performed the ADLs. It is unclear if the thresholds values were based on the fourth method or the first method.

In our study, we chose the second approach. Both third and fourth methods are a little bit controversial, and we did not have any access to previous dataset nor familiarity with using specialized softwares such as LabVIEW. The choice then fell between the first and second option.

By using the first method, there will only be a need to have 1 experiment for gathering of data. The problem with using this is that in the testing phase of the system, only the algorithm is

being tested. It will not provide test results on how the system as a whole (algorithm, hardware, users, situations, real-life usage, etc.) functioned. Aside from this, it was very difficult to gather volunteers for the fall simulations because of the nature of the experiment. We would not be able to gather a sufficient amount of participants to have enough data at once for both training and testing a prototype unless waiting for a longer period of recruitment. The first option is best if one only wants to compare different algorithms with no particular regard for the chosen platform.

The experiment for data gathering posed a lot of complications. Firstly, the participants were asked to do real falls. This means that they were in real danger of injuries. In fact, a couple of our participants did suffer some minor injuries and most of them complained about bruising and muscle pains the following day.

Again, because of the nature of the experiment only few people volunteered to participate. All throughout our study, only 14 people participated. Among these participants, only 5 people gave convincing real-like falls as how we explained it to them. Normally, for testing fall simulations, one seeks the aid of a professional(stuntmen, judo athletes, or other fall experts) either for supervision and instruction or for doing the fall simulation himself. For example, in the research study of [29], only 1 gymnast was used for fall simulations both in the training and testing part of the system. Another healthy elderly woman (83 years old) was used to measure the ADLs. Also in [17], only one stunt actor was used for the simulations. The stunt actor was instructed on doing five falls by the research team's nursing collaborators. There were no data for a test of the prototype they have made. Even with a larger participation rate (10), like in the research study of [18], a supervision of a physical education professional was needed. Suffice to say, we did not have this resource nor could we afford to wait for a longer time for people to participate.

In our own literature study, the most participants a research group could gather for fall simulations are 10 people, mostly young and healthy. Like in the previous example, [10] had a group of 10 healthy male performing 8 chosen falls and 4 ADLs, while a second group of 10 elderly people (73-93 years) with 4 women and 5 men performed a list of ADLs. All these were done in the training period. It was not clear what data they used for testing their system. Even though they had a second group of 10 elderly people, in reality fall thresholds are based primarily on actual fall data. ADL data are used to adjust the threshold and for comparison reasons, i.e. testing the specificity.

The difficulty of a proper experiment set-up is reflected by the whole research community of fall detection. In the study of [51] and [52], only a total of five young (<30 years) healthy male subjects were recruited for the study. They performed both falls and a list of ADLs. Likewise, only 2 male students participated in [13]. In [45], they got their dataset from actual fall data of 12 elderly people who have been observed in 309 patient days. Since the resulting set of fall data was too small, accurate fall simulations based on a video recording showing 19 actual falls, were also done. The research group was less concerned about the variation and quantity of the test

group, but how accurate they could simulate falls based on real falls of their targeted users.

We figured during the course of our study to use a dummy for fall simulation, like how it was done in [8]. In that study, they used a mannequin for both the training and test period. They used the first experimental approach of only doing 1 experiment and dividing the data set into two halves. When they have finally evaluated the performance of their system, it was only based on test results from the mannequin. The mannequin was also in reality far from closely anatomically resembling a human being .

Originally, when we decided to use a dummy, the plan was to get a crash dummy which as far as we have gathered the best one resembling a human being, both in its mass, built, joint location, flexibility, etc. In the end, the nursing department at our university provided us with an old CPR dummy. The problem with this dummy is that it was not sturdy enough, being old and not built for falling. Its body mass was also low. We could somehow adjust its body mass by adding weights inside the dummy, but it won't take long time before the dummy falls apart upon impact. This meant that we also could not get sufficient data just from the dummy. Ultimately, we were not satisfied in using just this dummy to base our data on. In the end, this is why we decided to use both a dummy and volunteers for data gathering. We figured that both data set would compliment each other, and we would be able to compare the results from each group too.

Yet another complications come from the chosen platform itself - the smartphone. As mentioned in section 2.2, wearable devices are at risk of getting damaged and broken. This risk was in fact higher in the experiment and testing phase where multiple falls were purposely simulated. During our experiment, one Dell Streak unit was broken after multiple trials. Because of this, we have to change to a different mobile unit. After some testing and calibration, we have deduced that the accelerometer of the new device to have the same sensitivity and calibration as in the Dell Streak's. As such, we did not have to normalize our training data. This is however a step, one ought to look at if the system will eventually be mass produced to function in any existing mobile phone with accelerometers.

Wearable fall detection systems are assumed to be used in a fixed location. This is because in its development phase, the gathered data are also based on a fixed spot, e.g. thigh, chest, head, with the use of a strap or some other device to gain stability. In our study, we wanted to see how well a mobile device function as a fall detection system if it is used as how it would normally be used, meaning that we assume that users will still put their mobile device in their pockets. The problem with this of course is that the instability of the device would give random noises. As we have for example found out, pockets that are too loose will give more noise to the phone, i.e upon impact the phone will be shaken more and give bouncing effects in oppose to if the phone sits steadily in a tight pocket. Pockets can sit more in the front of the thigh or the side of the thigh. Some body parts provide better and accurate readings than others as we have mentioned in section 2.2. Also, a user can use an entirely different pocket, e.g. on the front of the shirt, to put the phone in. In this case, the phone will be sitting higher than what is expected. One can ac-

count for this, by offering the user extra options. It is also feasible to change the threshold levels dynamically, but this would require implementing additional algorithm for this, like clustering algorithms used in machine learning techniques. For our study, we have simplified our assumptions and did not take this into consideration as this will be a different focus of development. Noise filtering algorithms can also be used, but for the same reason as before, we have simplified our assumptions and did not use any preprocessing of the raw data. Our focus is immediate response, and a system that will not eat much of the phone's resources that it disregards the phone's other functionalities.

The orientation of the phone itself can also affect how the system works. For example, if one uses equation 2.9 or 2.11 as a part of the algorithm, one needs to take into account that the axes of the phone changes depending on its orientation, i.e. if the phone is in standing position, lying, etc. One can adjust for this by sensing when the phone is in static mode for a longer period of time, and defining that orientation as the start position. One can then use the proper axis for the algorithm. In our system, we have only used the total acceleration of the phone. Meaning, whatever the orientation of the phone is, it will still give the same total acceleration, i.e RSS.

As mentioned, during the course of our experiment we broke one mobile unit and had to replace it with another. Although we have assessed that the new phone's accelerometers had the same sensitivity and range as the previous one, we found out one major flaw with it. It seemed like sometimes the phone stops reading acceleration data. This might be due to the fact that the phone was old and that it had some internal faults. Due to this, we have observed that the system (regardless of the 3 algorithms) gave a way lower percentage for sensitivity for lateral falls (towards right or left). For 4 of the volunteers, the system never detected one single fall among either all right falls or left falls. We found out that this was when the user is falling laterally on the side where the phone sat in. It would seem like even though the accelerometers could read the data, some other internal parts of the phone does not do well with the force of the direct impact.

Collapse falls also gave a very low sensitivity percentage. When the volunteers were asked to do collapse falls, they tend to give a very slow soft fall. Ideally, collapse falls should have been represented in all directions like in the case of the stick falls, but in our experiment it was not possible to do so. This was because we have not procured good enough volunteers that could simulate these types of falls. The volunteers who did try these falls, were not convincing enough with their simulated fall, i.e. the falls did not seem real.

Alas we could also not force the volunteers to perform all types of falls as many as possible. We let the participants evaluate which falls they are comfortable to perform, and if they are able to give more fall simulations than the minimum of 3 per type of fall. Some participants did not finish the minimum of three, while others dropped doing a certain type of fall completely. In our original set-up we did not want to use any softening cushion as we are using an algorithm that measures the impact value. The softening cushion will give a different data set than what

one would normally get if a person falls on a hard surface, which is more normal for real falls. Either the upper fall peaks are lower or bouncing effects (noise) will be introduced. This was also a reason why we considered using a dummy, since we could drop the dummy on a hard surface, but as mentioned earlier our dummy was not good enough to be the sole source of our data.

When we have asked our participants to do the fall simulations, we first suggested to use a gym mat (see appendix A for the actual gym mat). This is made of rubber that was approximately a bit over 2 cm thick. After some tests, we found out that this actually did not help much. Frontal falls were somehow doable for the participants, even though they still exposed the volunteers to pain and injury. Collapse was the easiest to do, but then again most participants were doing it not that realistically. When we asked the participants, they did not want to continue doing the falls on the mat. At this moment, we have decided to use a better softening cushion anyway. We would first ask the participants to do some trial falls on the gym mat if they think they are able to do the falls on the mat. If they can not, we offer them a softer cushion. Among the 14 participants only 4 participants did the fall simulations on the gym mat, and not all types of falls were done on it too. Backward stick falls were not actual at all to do on the mats, as head injury is more likely to happen in a backward fall.

In the training session, all three volunteers did some frontal fall simulations on the gym mat while the rest was done on a mattress which was somehow bouncy. For the testing phase, a different less bouncy mattress was used in combination with the gym mat or outside in the grass field. Our problem now is that we have used different surfaces. For pre-impact algorithms this does not matter, but in all of our three algorithms we were using a post impact algorithm when we check for the upper fall peaks. We contemplated at this point to change our algorithm to just use a pre-impact acceleration based algorithm, i.e. just testing for LFT. But as explained earlier, this is unreliable and will produce plenty of false alarms. This will also mean, that we would need to go back to the drawing board and remake our system with a different algorithm. Unfortunately, we did not have time for this. We decided to pursue with the test and analyze the results anyway.

5.2 Discussion About the Prototype

Before even setting-up our first experiment and creation of a prototype, we originally planned to use a smartphone with both gyroscopes and accelerometers. In our hypothesis based on the literature study we have done, an approach using data from both gyroscopes and accelerometers seemed to perform the best. We failed in procuring a device with both gyroscope and accelerometers. Our university did not have access to one such device at that point. As far as we have understood, gyroscopes are also new as an integrated technology in mobile phones, and that iPhone 4 was indeed the first mobile phone to have integrated this. **Note:** Gyroscope should not be confused with orientation sensors!

Thus, we limited our research to just acceleration-based algorithms. It then became our focus

to make an acceleration-based fall detection system to be as simple as possible to conserve the phone's resources so that less powerful phones can also use the system. This lead us to choose the 3 algorithms we have now. In the very first prototype, algorithm 1 and 2 did not use any orientation calculation by using equation 2.8, as such it was much simpler than algorithm 3, but because we decided to implement an inactivity detection, they too had to use it. Arguably an inactivity detection algorithm can be implemented differently, but choosing what kind of inactivity detection algorithm to use is less important.

The computation of the falling-edge time and rising-edge time were added, because we wanted to see if it will help change the sensitivity of the fall detection system. These two time values can only be calculated after the lower and upper fall thresholds(LFT and UFT) are decided. This means that they can also be based on and used in the first two algorithms. In fact, we made a prototype that used this. We have later found out that the code for the algorithm we made was not entirely correct so that we omitted the test results for it. Implementing this was more difficult than the other 3 algorithms. We however managed to figure the threshold levels for the falling-edge time and the rising-edge time based on the experimental results. One only needs to make a correct code for it.

In the first prototypes, the alarm notification was set to 10 seconds. We found out that this was too quick in case of a false alarm. For our study, we consider the alarm notification as the second part of the inactivity detection. If we assume that the main fall detection algorithm to have good specificity percentage, then one would not need to delay an alert issue too long. Arguably, one can say that this is not needed at all if the specificity of a system is very good. Usually though even good systems employ this, because "one can never be too sure". The alternative for example is to adjust the inactivity detection algorithm to detect a prolonged inactivity period (usually over 1 min based on other systems) before directly issuing a fall alert via SMS, phone call or whatever communication technology one has decided to use. The problem with this, is that there will be cases when a serious fall occurs but the user will still show signs of activity, e.g. twitching reflexes or external factor effects. The system will think that the user is fine because of the registered activity while the truth is that the user can be badly injured. In the end, we compromised with using a 20 seconds notification before issuing a fall alert. This time is subject to debate though, but we have assessed this time to be not too long to delay a fall alert in case of real falls, but also not that quick so that the user can cancel it in case of fall alarms.

5.3 Discussion About the Results

Test results show that all 3 algorithms performed poorly in the sensitivity test. This is not in acceptable levels. For the most part this was because of the problems and complications we have had with the experiment, hardware and equipments, participation numbers and credibility, and other things we have mentioned in the two previous sections. We believe that if the experiment was done without the mentioned complications, the sensitivity levels would not be as low as what we have got. We probably could have done better by either changing our algorithm to cope

with the circumstances we had or redo the experiment as we first conceived it to be, but alas we did not have time to change or modify the algorithm because of the time constraint of this study nor did we have time and resources to do the experiment, creation of prototype and testing it one more time.

That being said, the test data still gave us relevant results which we will present in the conclusion.

6 Conclusion

We realize now that fall detection research and study is a complicated field that needs more allocated resources to have a meaningful study. Going back to our original research questions, we will now try to answer them based on what we have learned in our research study. There is a need to redefine or make a more precise questions for most of these research questions. This is because the research questions were made before we started our research study. Our knowledge on how feasible one can answer each question was at that point limited or non-existent.

How would one define a "fall" in terms of a fall detection system?

We have learned that the term "fall" is in fact vague in how it is used for most of existing research studies about fall detection systems. What this means is that, most fall detection systems have not properly defined the types of falls they have used, usually only simulating directional falls. This in itself does not need to bad though. If the goal is only to compare a number of algorithms against each other then this is completely fine. When one, however, makes a fall detection system for a specific group, e.g. elderly people, and based their system on a few generic fall simulations, then the resulting system will mostly be unreliable. Thus, if one aims to make a fall detection system for a specific group of users, it is better to use real fall data from the targeted users. In cases where this is not possible, then proper simulation of expected falls from that group should be done instead.

Further, we have learned that there are actually several group of users that falls under fall-risk groups. As far as we are concerned, it is primarily only the elderly population that current fall detection systems target.

Section 2.1 explains the most important factors to be considered when one is making a fall detection system. These factors are the type of falls, the cause of falls, the mechanics and phases of a fall and different fall-risk groups

For our own system, section 3.1 answers how we have defined a fall. We have not assumed any targeted group of users as we had to limit our research focus to investigating if a smartphone could be a good fall detection system.

Is it possible to make a pervasive, reliable and effective fall detection system that could be adaptable for different types of user?

We are redefining this question to be: *Will a smartphone prove to be a pervasive, reliable, effective fall detection system that could be adaptable for different user?*

The nature of mobile phones already support pervasiveness and adaptability. By making the fall detection system a mobile phone application, i.e. *app*, it can then be easily spread to anyone with mobile phone. While it is true that we programmed our system in Android systems, translating the application to other platforms should not be that tedious of a task as long as you already set-up the threshold levels and still have the results from the experiment.

Since we have not configured our system to cope for different users, we were not able to test adaptability of the system for different users. Based on our experiment though, we see that one can at least change the sensitivity level of the system based on how active the user is. The more active a person is, the less sensitive should the system be, i.e. it should not confuse running or jumping as a fall. Full adaptability is something a bit more complicated and will be presented in the next chapter.

For the chosen algorithms and the system that we ended up with, we could say that it is not reliable since among the 3 algorithms, the best one only showed 43.3% sensitivity. This is inspite of the specificity being in a very good level. It is true though, that this result was affected negatively by the unforseen complications we have outlined in the last chapter. The system's reliability is poor mostly because of the implemented algorithms. This is not the same as saying that smartphones in general are unreliable fall detection systems. They do have their own flaws though.

A smartphone in itself is not a suitable platform to be used for falling as most of them are fragile. In our experiment, the phones were exposed to forces they were not intended to handle. We have learned that it is better to have the phone in a protective casing and also in a fixed location. Although this might stigmatize the user and will beat the purpose of being able to use a fall detection system as how one normally would use a mobile phone, this will still increase its reliability and performance. Protective casing should also have been used in the training phase, i.e. data gathering phase of the experiment or alternatively use a more robust accelerometers which could tolerate falls, and just normalize the data later to be used on a mobile phone.

While reliability of the system goes to the sensitivity and specificity of a system, effectiveness is what a system does after sensing a fall. In this case, it is very effective if it senses a fall. The phone already has built in communication means and does not rely on a base station. It is also not limited to just use one connectivity means. The phone also only monitors one user at a time. The data noise one can get, is if the phone is loosely attached to the user. This is easily fixed by using straps. Some users will not like this because the system becomes more intrusive, while others will not mind this at all.

We conclude that most mobile phones, saved some which are specially made for physically active users, are just not designed to tolerate hard impacts. One can alleviate this by using protective casings, but this does not promise complete protection. If the cost of saving money by using a fall detection system in a smartphone than the more expensive commercial products,

outweighs the cost of breaking the phone after several falls, then investing in this kind of system is justified. This means that a fall detection system made in a mobile phone is probably most useful for when the target group is not expected to fall too often, e.g. outdoor adventurers or healthy elderly people below 70. Also, if one has already a smartphone that can run the application, then investing in it should be cheaper than the alternative, but if one decides to buy a smartphone only to use its fall detection capabilities, then this might be not that good of an idea. Lastly, mobile phone based fall detection systems are still the best (and probably the only) alternative for fall detection system that can be used outdoors - this in itself is a form of adaptability.

What is the best algorithm and/or approach for detecting fall?

As we cannot compare all known approaches and algorithms that can be used in making a fall detection system, we are redefining this question to be: *Given the three accelerometer-based algorithms in section 3.5, which is the best one?*

Both in using the system against actual people and dummies and in the different types of falls, algorithm 1 performed the best.

In section 5.1 we stated that most lateral falls on either left or right side, did not produce an alarm in any of the 3 algorithms, and that this was probably because of some internal hardware flaw of the device. While we still believe that, it is also possible that the algorithm is just not suited to detect lateral falls very well. Most of the volunteers in the testing phase, even when asked to do a stick fall, would still bend or do some form of flexion when doing the lateral falls. The profiling algorithm only measures acceleration change and not orientation or posture change, so we have enough reason to think that also the algorithm is just not strong if a lateral “non-stick” fall occurs.

As a last note, we ought to mention that from our literature study, it appears that combination of gyroscope and accelerometers as well as including all phases of a fall for fall detection, is possibly the best approach. By using both accelerometers and gyroscopes, one can accurately measure (and recreate) the 6-degree freedom of movement, i.e. 1) going up and down in the Y-axis, 2) going left and right in the X-axis, 3) going forward(front) and backward(back) in the Z-axis, 4) rotation on the vertical axis (yaw), 5) rotation in the lateral axis (pitch) and 6) rotation on the longitudinal axis (roll). Simply put, it will be able to accurately measure all kinds of movement a human person can do. Also in chapter 3, we presented our concrete approach for making a system. This is partly based on the fact that we believed that approach to be our best option for this study. Finally, an approach or a system for detecting fall might be good for a group of people and circumstances but not for the other. It is up to the developers team to properly assess and define these circumstances and group of people and then decide which approach to use.

What constitute to be a good fall detection system?

Section 2.5 provides details on how one can evaluate a fall detection system. This evaluation is objective in its nature. Sensitivity says the ability of a system to detect falls, while specificity is the ability of the system to only detect falls.

Given that the targeted users as well as the use-area of the system are defined, a perfect system will give 100 % on both sensitivity and specificity. So far no system has achieved this, saved some few which only performed a limited list of falls and ADLs in a controlled environment. As a sort of rule of thumb, anything above 90% is still considered to be good. The lower these values become, the less reliable a system becomes.

Sensitivity and specificity need to be both taken into consideration in evaluating a system, but sensitivity should be given more priority.

A more subjective evaluation of a fall detection system can also be added. This will be briefly mentioned in the *future work* chapter.

We have also provided the readers how we concretely evaluated and tested the system we have made in chapters 3 and 4.

Other conclusions and realization after the study

Since there are many risk factors associated with falls, we conclude that there are also many different groups of people (and not just the elderly) that suffer from falling. Each of these groups might show a different type of falls in terms of fall patterns, behavior and physics.

A fall detection system might suit for one person but not for the other. This all depends on the circumstances, type of user and the user's preferences.

Fall experiments are very difficult. Consider the following. There are a total of 19 different fall scenarios for elderly people. These falls should be simulated at least 3 times each. If one also considers doing the fall scenarios on different surface, e.g. soft vs hard surface, then for every volunteer he or she needs to simulate at least 19x3x2 (=114) falls. If the system assumes several targeted users, to offer adaptability, then different sets of fall scenarios are added. This is so far only the fall simulations. ADLs and non-fall activities are also to be simulated.

7 Future Work

In our study we have only used and tested 3 specific acceleration-based algorithms implemented in a smart phone. Section 2.4 provides other concrete acceleration-based algorithms which all can be tested and implemented in a smartphone. They can be used as they are presented or just use them as a basis for an improved algorithm. This is something that could be looked at in the future.

Although our chosen algorithms performed poorly, we cannot conclude that it is in general this type of algorithm that is bad, because of the problems and complications we have had. One ought to look at them one more time, provided better resources in the form of participants, equipments and time.

Also, still within the boundaries of a smartphone, we suggest looking at a comparison for a system that uses gyroscope+accelerometer, just gyroscope, and just accelerometer, and if time constraints still permits it, a combination with magnetometers too.

We have stated that a smartphone provides many possibilities and yet we have barely used its potential. One can make a location-aware fall detection system, so that also the user's location will be sent in the outgoing alert message. One can also look at if sending a photo and audio recording as part of the outgoing alert message, would also provide better context to assess the situation of the user and see the severity of a fall. Since one can connect a phone to a more powerful stationary computer, e.g. via bluetooth, one can for example perform a more advanced machine learning algorithm in the PC if the user is indoors. If the user goes outdoors, the mobile device shifts to a more resource saving system in the mobile phone itself.

If one has managed to implement location-aware system as well as context through picture and sound, one can proceed to make a fully context-aware fall detection system. This means that the system will adjust its thresholds and thus sensitivity depending on the context. Context can be about the user - i.e. the user's height, weight, body mass, age, gender, activeness, risk group - location, time, or other factors that can affect falls as described in section 2.1. This is a very ambitious and large study though. One first need to investigate if these contexts do in fact affect the way a person falls. The research group needs to find the relationship between a person's profile (e.g. age, height, weight) and the measured data. A system like this will give full adaptability for different users.

A better more unified framework for both experimental set-up as well as evaluation of fall detection system is also needed. The purpose of the framework is to provide uniformity and standard for evaluation of all fall detection systems. The problem with today's evaluation, is that

even if a research group say that they have achieved a 100% in sensitivy or specificity, we have no way in knowing if they have sufficient and realistic enough activity and fall scenarios unless we thoroughly study their work. It is even worse for commercial products where tests are rarely published. The number of times each activity or fall is tested varies much too. This is also true for the number of participants as well as if the participants represent the targeted group of the system (or perhaps even the general population) well.

[4] suggests one framework, but we think that this could be improved and be more detailed. The framework should also be divided into an objective evaluation, much like what we have done - testing the sensitivity and specificity through controlled testing and continuous daily use. The second part is a subjective evaluation in the form of surveys and interviews of actual users of the system as well as anyone affected by the system, e.g. the user's family or health care provider. The subjective evaluation could show what kind of user will prefer what kind of system, e.g. wearable vs. ambiance. The GUI interface or else design of the system can be tested through the subjective evaluation too. The feedback from this evaluation can further help improve the design of the system.

Research in accidental falls (see section 2.1) needs to be better addressed, i.e. fall simulations that includes trip and fall, slip and fall and so on and so forth.

Bibliography

[1] C, T. & D, S. 2004. What are the main risk factors for falls among older people and what are the most effective interventions to prevent these falls? Available in"http://www.euro.who.int/__data/assets/pdf_file/0018/74700/E82552.pdf" by the World Health Organization, Last Visited at 01-07-2011.

[2] Center for Disease Control, P., for Injury Prevention, N. C., & Control. 2010. Falls among older adults. "http://www.cdc.gov/homeandrecreationalsafety/falls/adultfalls.html". Last Visited at 01-07-2011.

[3] Yoshida, S. 2007. A global report on falls prevention epidemiology of falls. Available in"http://www.who.int/ageing/projects/1.Epidemiologyoffallsinolderage.pdf" by the World Health Organization, Last Visited at 01-07-2011.

[4] Noury, N., Rumeau, P., Bourke, A., ÓLaighin, G., & Lundy, J. 2008. A proposal for the classification and evaluation of fall detectors. *IRBM*, 29(6), 340 – 349.

[5] Charles Brown, Carol Lehtola, W. J. B. 2009. Preventing injuries from slips, trips and falls. Available in"http://edis.ifas.ufl.edu/as042" by the University of Florida, Last Visited at 01-07-2011.

[6] Stalenhoef, P. A., and. J André Knottnerus and. Luc P de Witteb, J. P. D., & Crebolder, H. F. 2000. The construction of a patient record-based risk model for recurrent falls among elderly people living in the community. Available in"http://fampra.oxfordjournals.org/content/17/6/490.abstract" by Oxford University, Last Visited at 01-07-2011.

[7] Zhang, T., Wang, J., Liu, P., & Hou, J. 2006. Fall detection by embedding an accelerometer in cellphone and using kfd algorithm. In *International Journal of Computer Science and Network Security*, Vol. 6 no. 10 pp 227–284.

[8] Dai, J., Bai, X., Yang, Z., Shen, Z., & Xuan, D. 29 2010. Perfalld: A pervasive fall detection system using mobile phones. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*.

[9] Li, Q., Stankovic, J., Hanson, M., Barth, A., Lach, J., & Zhou, G. 2009. Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information. In *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, 138 –143.

[10] Bourke, A., van de Ven, P., Gamble, M., O'Connor, R., Murphy, K., Bogan, E., McQuade, E., Finucane, P., ÓLaighin, G., & Nelson, J. 2010. Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities. *Journal of Biomechanics*, 43(15), 3051 – 3057.

[11] Yu, X. 2008. Approaches and principles of fall detection for elderly and patient. In *e-health Networking, Applications and Services, 2008. HealthCom 2008. 10th International Conference on*, 42 –47.

[12] Ng, S., Fakih, A., Fourney, A., Poupart, P., & Zelek, J. 2009. Towards a mobility diagnostic tool: Tracking rollator users' leg pose with a monocular vision system. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, 1220 –1225.

[13] Perry, J., Kellog, S., Vaidya, S., Youn, J.-H., Ali, H., & Sharif, H. 2009. Survey and evaluation of real-time fall detection approaches. In *High-Capacity Optical Networks and Enabling Technologies (HONET), 2009 6th International Symposium on*, 158 –164.

[14] Noury, N., Fleury, A., Rumeau, P., Bourke, A., Laighin, G., Rialle, V., & Lundy, J. 2007. Fall detection - principles and methods. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, 1663 –1666.

[15] Ketabdar, H. Detecting physical shock by a mobile phone and its applications in security and emergency.

[16] Degen, T. & Jaeckel, H. 2003. Speedy: a fall detector in a wrist watch. In *In Proceedings. Seventh IEEE International Symposium on Wearable Computing*, 184–189.

[17] Popescu, M., Li, Y., Skubic, M., & Rantz, M. 2008. An acoustic fall detector system that uses sound height information to reduce the false alarm rate. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, 4628 –4631.

[18] Bourke, A. & Lyons, G. 2008. A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. *Medical Engineering Physics*, 30(1), 84 – 90.

[19] Aud, M. A., Abbott, C. C., Tyrer, H. W., Neelgund, R. V., Shriniwar, U. G., Mohammed, A., & Devarakonda, K. K. 2010. Smart carpet: Developing a sensor system to detect falls and summon assistance. *J Gerontol Nurs*, 36(7), 8–12.

[20] Alwan, M., Rajendran, P., Kell, S., Mack, D., Dalal, S., Wolfe, M., & Felder, R. 0 2006. A smart and passive floor-vibration based fall detector for elderly. In *Information and Communication Technologies, 2006. ICTTA '06. 2nd*.

[21] A. Sixsmith, N. Johnson, & R. Whatmore. 2005. Pyroelectric ir sensor arrays for fall detection in the older population. *J. Phys. IV France*, 128, 153–160.

[22] Bourke, A., O'Brien, J., & Lyons, G. 2007. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. *Gait Posture*, 26(2), 194 – 199.

[23] Tzeng, H.-W., Chen, M.-Y., & Chen, J.-Y. 2010. Design of fall detection system with floor pressure and infrared image. In *System Science and Engineering (ICSSE), 2010 International Conference on*, 131 –135.

[24] Salomon, R., Luder, M., & Bieber, G. 29 2010-april 2 2010. ifall - a new embedded system for the detection of unexpected falls. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, 286 –291.

[25] Mitja, L. & Bostjan, K. Fall detection and recognition with machine learning. Technical report, Jožef Stefan Institute, Department of Intelligent Systems, 2009. "http://dis.ijs.si/mitjal/documents/Fall_detection_and_activity_recognition_with_machine_learning-Informatica-09.pdf", Last Visited at 01-07-2011.

[26] Ralhan, A. S. A study on machine learning algorithms for fall detection and movement classification. Master's thesis, University of Saskatchewan, "http://library2.usask.ca/theses/available/etd-12222009-144628/", 2009. Last Visited at 01-07-2011.

[27] Caruana, R. & Niculescu-mizil, A. 2006. An empirical comparison of supervised learning algorithms. In *In Proc. 23 rd Intl. Conf. Machine learning (ICML06)*, 161–168.

[28] Caruana, R., Karampatziakis, N., & Yessenalina, A. 2008. An empirical evaluation of supervised learning in high dimensions. In *In International Conference on Machine Learning (ICML)*, 96–103.

[29] Lindemann, U., Hock, A., Stuber, M., Keck, W., & Becker, C. 2005. Evaluation of a fall detector based on accelerometers: a pilot study. *Med Biol Eng Comput*, 43(5), 548–51.

[30] Jantaraprim, P., Phukpattaranont, P., Limsakul, C., & Wongkittisuksa, B. may 2010. Improving the accuracy of a fall detection algorithm using free fall characteristics. In *Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTICON), 2010 International Conference on*, 501 –504.

[31] Kangas, M., Konttila, A., Lindgren, P., Winblad, I., & Jämsä, T. 2008. Comparison of low-complexity fall detection algorithms for body attached accelerometers. *Gait Posture*, 28(2), 285 – 291.

[32] Nguyen, T.-T., Cho, M.-C., & Lee, T.-S. sept. 2009. Automatic fall detection using wearable biomedical signal measurement terminal. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, 5203 –5206.

[33] Salomon, R., Lux0308 andder, M., & Bieber, G. july 2010. ifall - case studies in unexpected falls. In *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, 1645 –1650.

[34] 2011. Survival store. Available in "http://survivalstore.com/r6s15lcb2.html", Last Visited at 01-07-2011.

[35] 2011. Alert 1. Available in "http://www.alert-1.com/", Last Visited at 01-07-2011.

[36] 2011. Hjelp 24. Available in "http://www.hjelp24.no/alarmsentral/", Last Visited at 01-07-2011.

[37] 2011. Securitas. Available in "http://www.securitas.com/no/no/Kundesegmenter/Helse-og-omsorg/", Last Visited at 01-07-2011.

[38] 2011. ilife. Available in "http://www.ilifesolutions.com/products.html", Last Visited at 01-07-2011.

[39] 2011. Brickhouse. Available in "http://www.brickhousealert.com/howitworks.html", Last Visited at 01-07-2011.

[40] 2011. Seekwellness. Available in "http://www.seekwellness.com/LPs/fall-alarms.htm", Last Visited at 01-07-2011.

[41] 2011. Tunstall. Available in "http://www.tunstall.co.uk/Products-and-services/Product-overview-", Last Visited at 01-07-2011.

[42] 2011. Fall alert. Available in "http://www.macworld.com/appguide/app.html?id=416637&expand=false", Last Visited at 01-07-2011.

[43] 2011. Mover. Available in "http://www.androidzoom.com/android_applications/health/mover-m_nly.html", Last Visited at 01-07-2011.

[44] 2011. Cradar. Available in "http://actionxl.com/CRADAR.html", Last Visited at 01-07-2011.

[45] Boyle, J. & Karunanithi, M. 2008. Simulated fall detection via accelerometers. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, 1274 –1277.

[46] Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., & Campbell, A. sept. 2010. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9), 140 –150.

[47] 2011. Android. Available in "www.android.com", Last Visited at 01-07-2011.

[48] Hansen, T. R., Eklund, J. M., Sprinkle, J., Bajcsy, R., & Sastry, S. 2005. Using smart sensors and a camera phone to detect and verify the fall of elderly persons. In *In European Medicine, Biology and Engineering Conference*.

[49] 2011. Oppenoffice:calc. Available in "http://www.openoffice.org/product/calc.html", Last Visited at 01-07-2011.

[50] 2011. Labview. Available in "http://www.ni.com/labview/whatis/", Last Visited at 01-07-2011.

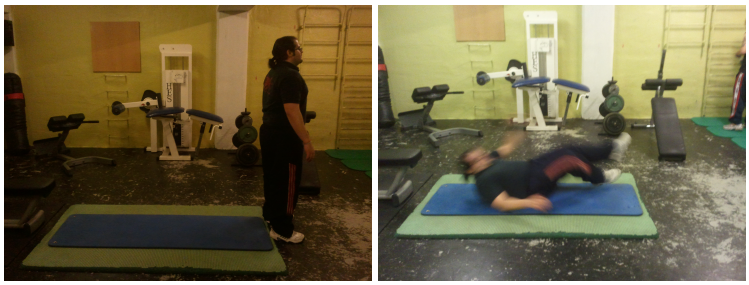
[51] Bourke, A. K., O'Donovan, K. J., Nelson, J., & OLaighin, G. M. aug. 2008. Fall-detection through vertical velocity thresholding using a tri-axial accelerometer characterized using an optical motion-capture system. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, 2832 –2835.

[52] Bourke, A., O'Donovan, K., & ÓLaighin, G. 2008. The identification of vertical velocity profiles using an inertial sensor to investigate pre-impact detection of falls. *Medical Engineering Physics*, 30(7), 937 – 946.

A Fall Scenarios

1) Forward Fall Like a Stick

2) Backward Fall Like a Stick



3) Lateral “Stick” Fall to the Left

4) Lateral “Stick” Fall to the Right


5) Forward Fall Collapse



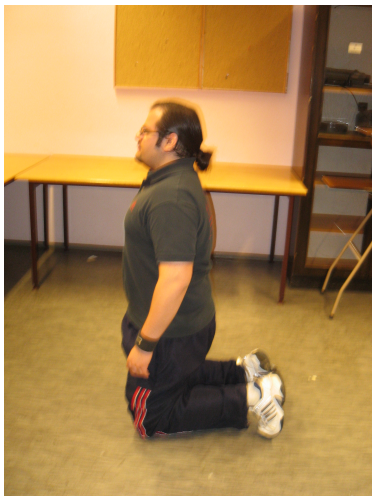
B ADLs and Non-fall Activities

1)From a standing position, sit down (normal speed) on chair. Remain sitting for 5 sec then stand up.

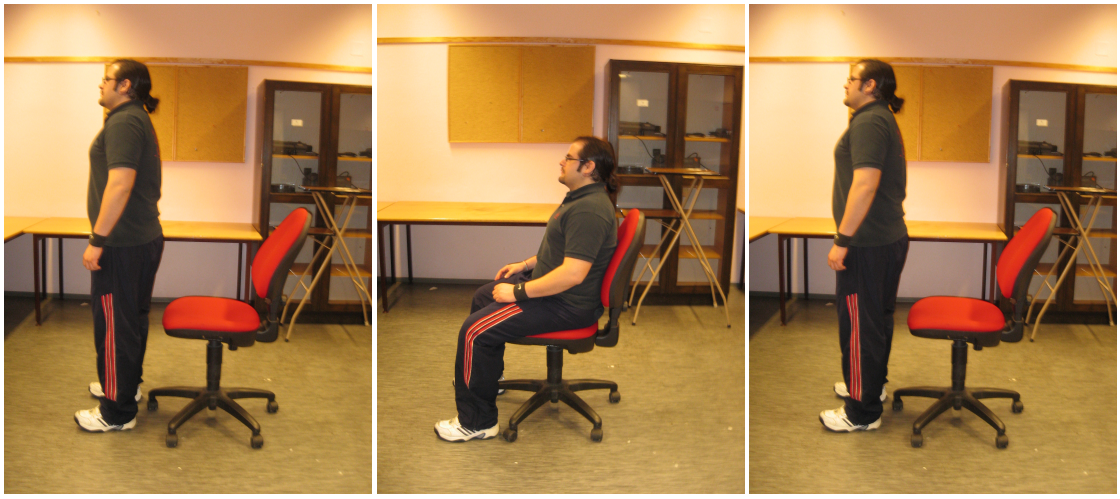
2)From standing position, sit down (normal speed) on the floor . Remain sitting for 5 sec then stand-up.

3)From standing position, bend-down and pick-up item on the floor. Rise-up.

4)From standing position kneel down and remain kneeling for 5 sec then rise-up.



5)From standing position, quickly sit down on a chair. Stay seated for 5 sec then rise up quickly.



6)From standing position, quickly fall down facing the floor. Assume the position of someone who will do push-up. Remain in that position for 5 sec. Do 3-5 push-ups. Quickly rise up.

7)From standing position lie down on the floor and remain lying there for 5 sec. Rotate 180 degrees so you are facing the ground. Stay on that position for 5 sec.

Rotate 180 degrees to go back to original position. Assume sit-up position and remain in that position for 5 sec. Do 3-5 sit-ups and stand up right after.

8)From standing position, kick (normal speed) a punching bag. Remain standing for 5 sec. Kick the bag again (fast/hard).

9)Mount the bicycle and start cycling and do so for 5 sec.

10)From point A, walk (normal speed) to point B (approximately 8-10m).


11)From standing position, lie down on table and remain lying there for 5 sec then stand-up.

12)Climb up on the platform(table in this case). And remain standing there for 5 sec.

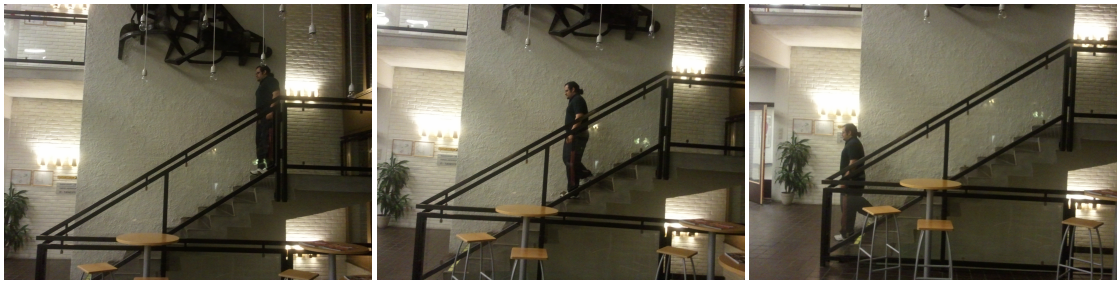
Jump down from the platform.

13)From standing position, jump up vertically.

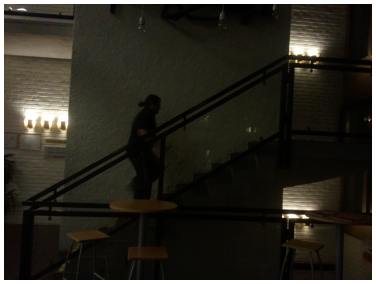
14)From standing position, start to run from point A to point B. Upon reaching point B and without stopping your momentum, jump as high and as far as you can.

15)Run from point A to point B.

16)From standing position, walk (normal speed) up the stairs. Wait 5 sec on top of the stairs then walk down. Remain standing for 5 sec.



From standing position, run-up the stairs. Wait 5 sec on the top of the stairs then run down.



17)From point A, jog all the way to point B.

C Codes

C.1 FallDetection.java

```
package falldetection.hig.no;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.PowerManager;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioGroup;

public class FallDetection extends Activity{

private static final String TAG = "Main";
private static final boolean debug = true;

  private static EditText telephone;
  private static EditText email;
  private static RadioGroup sex;
  private static RadioGroup sensitivity;
  private static Button monitor;
  private static Button close;
  private ComponentName service;

  public static Boolean serviceStarted;

  private PowerManager pm;
private PowerManager.WakeLock w1;

  /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
     if (debug==true) Log.i(TAG, "oncreate started");
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        telephone = (EditText) findViewById(R.id.telephone);
        email = (EditText) findViewById(R.id.email);
        sensitivity = (RadioGroup) findViewById(R.id.sensitivity);
        monitor = (Button) findViewById(R.id.monitor);
        close = (Button) findViewById(R.id.close);
        serviceStarted = false;

        pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
w1 = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, TAG);

if (debug==true) Log.i(TAG, "oncreate finished");
    }

@Override
    protected void onResume()
    {
if (debug==true) Log.i(TAG, "onResume started");

super.onResume();

if (debug==true) Log.i(TAG, "onResume finished");
    }

@Override
public void onRestart() {
if (debug==true) Log.i(TAG, "onRestart started");

super.onRestart();

if (debug==true) Log.i(TAG, "onRestart finisehd");
}

@Override
protected void onPause()
{
if (debug==true) Log.i(TAG, "onPause started");
super.onPause();
/*unbindService(authConnection);*/
```

```
if (debug==true) Log.i(TAG, "onPause finished");
}

    @Override
protected void onStop(){
     if (debug==true) Log.i(TAG, "onStop started");

super.onStop();

if (debug==true) Log.i(TAG, "onStop finished");
}

    @Override
    public void onBackPressed() {
    // do something on back.
    return;
    }


    @Override
    protected void onActivityResult (int requestCode, int resultCode, Intent data){
     if (debug==true) Log.i(TAG, "onActivityResult started");

     if (debug==true) Log.i(TAG, "onActivityResult finished");
    }

    @Override
protected void onDestroy(){
     if (debug==true) Log.i(TAG, "onDestroy started");
//stopService(new Intent(this, SensorService.class));
super.onDestroy();
this.finish();
if (debug==true) Log.i(TAG, "onDestroy finished");
}

    /*@Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
     savedInstanceState.putString(key, value);
    }*/

    public void close(View v){
     if (debug==true) Log.i(TAG, "close");
```

```
     this.finish();
    }

    public void startMonitor(View v){
     if (debug==true) Log.i(TAG, "startMonitor");

        if(serviceStarted == true)
{
     if (debug==true) Log.i(TAG, "service stopped");

     new Thread(new Runnable() {
             public void run() {

     serviceStarted = false;
     //cs.stopCollecting();
     stopService(new Intent(FallDetection.this, SensorService3.class));
     stopService(new Intent(FallDetection.this, SensorService2.class));
     stopService(new Intent(FallDetection.this, SensorService.class));
     if (w1.isHeld()) w1.release();

             }
         }).start();

monitor.setText("Start Monitoring");
}
else
{
if (debug==true) Log.i(TAG, "service started");

new Thread(new Runnable() {
public void run() {

serviceStarted = true;
if (!w1.isHeld()) w1.acquire();
startService(new Intent(FallDetection.this, SensorService3.class));
startService(new Intent(FallDetection.this, SensorService2.class));
startService(new Intent(FallDetection.this, SensorService.class));

             }
         }).start();
monitor.setText("Stop Monitoring");
}
```

```
     if (debug==true) Log.i(TAG, "startMonitor finished");
      }

}
```

C.2 SensorService.java

```
package falldetection.hig.no;

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

import android.app.Service;

import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

/**
 *
 * @author Ian James Daniel Gonzales
 * v1.3
 * This service uses algorithm 3
 * and with inactivity detection of at least 0.5 sec.
 */
public class SensorService extends Service{

private static final String tag = "Service";
private static final boolean debug = true;

/*Default thresholds */
//private static final float UFT_max = 29.462866;
private static final float UFT_min = (float) 15.7600;
//private static final float LFT_max_low = (float) 8.0335653;
private static final float LFT_max = (float) 8.7300;
```

```
//private static final float LFT_min = 1.3408;
//private static final float ori_max = 158.0417;
private static final float ori_min = (float) 48.3900;
//private static final long time_delay_max = 1994180;
//private static final long time_delay_min = 524195;
private static final long tre_max = 313000;
private static final long tre_min = 40900;

/*Sensitive setting threshold*/
/*private static final float UFT_sensitive_min = 15.4099;
private static final float LFT_sensitive_max = 8.7308;
private static final float ori_sen_min = 45.6950;
private static final long time_delay_sen_max = 349130;
private static final long time_delay_sen_min = 2083468;*/

/*private static float []x;
private static float []y;
private static float []z;
private static float []rss;
private static float []orientation;*/
private static float curr_rss;
private static float prev_rss;
private static float curr_x;
private static float prev_x;
private static float curr_y;
private static float prev_y;
private static float curr_z;
private static float prev_z;
private static float curr_ori;
private static float prev_ori;
private static int counter;
private static int counter_2;
private static long timestamp_fall_detected, LFT_timestamp, UFT_timestamp;

private static Vector lft_timestamps;
private static SensorManager sm;
private static List<Sensor> sensorAcc;
private static SensorListener sl;

private static boolean isRunning = false;
public static boolean aboveAccThreshold = true;
private static boolean detectedFall = false;
```

```
private static boolean LFT_exceeded = false;
private static boolean UFT_exceeded = false;
//private static boolean limbo = false;
//private static boolean curr_on, prev_on = false;

public SensorService() {
super();
if (isRunning == false) {
isRunning = true;
}
}

@Override
public void onCreate(){
if (debug==true) Log.i(tag, "onCreate");

counter=0;
counter_2=0;
timestamp_fall_detected=0;

 curr_rss = 0;
 prev_rss = 0;
 curr_x = 0;
 prev_x = 0;
 curr_y = 0;
 prev_y = 0;
 curr_z = 0;
 prev_z = 0;
 curr_ori = 0;
 prev_ori = 0;

super.onCreate();
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
if (debug==true) Log.i(tag, "onStartCommand");

sm = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
sensorAcc = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
sl = new SensorListener(this);
```

```
if(sensorAcc.size() > 0)
{
Sensor sensor = sensorAcc.get(0);
sm.registerListener(sl, sensor, SensorManager.SENSOR_DELAY_NORMAL);
}

return super.onStartCommand(intent,flags,startId);
}

@Override
public void onDestroy() {
if (debug==true) Log.i(tag, "onDestroy");
sm.unregisterListener(sl, sensorAcc.get(0));
sm.unregisterListener(sl);
isRunning = false;
super.onDestroy();
}

/**
 * Calculates the scalar product of two vectors
 * @param a x-axis for vector 1
 * @param b y-axis for vector 1
 * @param c z-axis for vector 1
 * @param d x-axis for vector 2
 * @param e y-axis for vector 2
 * @param f z-axis for vector 2
 * @param rss1 size of vector 1
 * @param rss2 size of vector 2
 * @return angle in degrees
 */
float orientation(float a, float b, float c, float d, float e, float f, float rss1, float rss2
float ori =  (float) Math.toDegrees(Math.acos(((a*d)+(b*e)+(c*f))/(rss1*rss2)));
if(ori>=0.0) return ori;
else return (float) 0.0;
}

private class SensorListener implements SensorEventListener
{
SensorListener(Context context) {

}
```

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
// TODO Auto-generated method stub

}

@Override
public void onSensorChanged(final SensorEvent se) {
new Thread(new Runnable() {
public void run() {
//if (FallDetection.serviceStarted == true){

if (se.sensor.getType()==Sensor.TYPE_ACCELEROMETER){

curr_x=se.values[0];
curr_y=se.values[1];
curr_z=se.values[2];
curr_rss=(float)Math.sqrt((curr_x*curr_x)+(curr_y*curr_y)+(curr_z*curr_z));

/* Orientation between position at timestamp x and timestamp(x-1) */
float ori = orientation(curr_x, curr_y, curr_z, prev_x, prev_y, prev_z, curr_rss, prev_rss);
curr_ori = ori;

prev_x = curr_x;
prev_y = curr_y;
prev_z = curr_z;
prev_rss = curr_rss;

if (debug==true) Log.i(tag, "status per tick:"+counter+":orientation:"+ori+":RSS:"+(curr_rss));

/* This part counts the inactivity period */
if ((ori<3) && (prev_ori<3)){
counter_2++;
if (debug==true) Log.i(tag, "orientation below 3:"+ori);
}else counter_2=0;

prev_ori = curr_ori;

/* An alarm is issued if a fall was detected followed by inactivity period */
if ((counter_2>3) && (((se.timestamp/1000)-timestamp_fall_detected)>520000 && ((se.timestamp/1000)-ti
startActivity(new Intent(SensorService.this, Alert.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
```

```
if (debug==true) Log.i(tag, "new activity:"+((se.timestamp/1000)-timestamp_fall_detected)+":"+

timestamp_fall_detected=0;
counter_2=0;
detectedFall = false;
}

/* LFT check */
if (curr_rss<=LFT_max && ori>=ori_min  && UFT_exceeded==false){
if (debug==true) Log.i(tag, "Lower fall threshall exceeded");

if (LFT_exceeded==false){

lft_timestamps = new Vector(1, 1);
LFT_timestamp=(se.timestamp)/1000;

}

lft_timestamps.add((long)(se.timestamp)/1000);

LFT_exceeded = true;
if (debug==true) Log.i(tag, "LFT:"+((se.timestamp)/1000)+":"+ori+"::"+curr_rss);
}

/* UFT check */
if (curr_rss>=UFT_min  && LFT_exceeded==true && ori>=5 ){
if (debug==true) Log.i(tag, "Upper fall threshold exceeded!");

UFT_exceeded = true;
UFT_timestamp=(se.timestamp)/1000;

Iterator it = lft_timestamps.iterator();

while(it.hasNext()){
Long tmp = (Long) it.next();
if (tmp != 0 && (UFT_timestamp-tmp) >= tre_min && (UFT_timestamp-tmp)<=tre_max) {
detectedFall = true;
timestamp_fall_detected = (se.timestamp)/1000;

if (debug==true) Log.i(tag, "Fall:" +":"+ori+"::"+curr_rss+ "::"+(UFT_timestamp-tmp));
}
}
```

```
counter=0;
LFT_exceeded = false;
UFT_exceeded = false;

if (debug==true) Log.i(tag, "UFT:"+((se.timestamp)/1000)+":"+ori+"::"+curr_rss+":::"+(UFT_timestamp-
}
}
}

}).start();

} // end onSensorChanged()
}

public class MyBinder extends Binder {
public SensorService getService() {
return SensorService.this;
}
}

@Override
public IBinder onBind(Intent arg0) {
// TODO Auto-generated method stub
return null;
}

}
```

C.3 SensorService2.java

```
package falldetection.hig.no;

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

import android.app.Service;

import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
```

```
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

/**
 *
 * @author Ian James Daniel Gonzales
 * v1.3
 * This service uses algorithm 1
 * and with inactivity detection of at least 0.5 sec.
 */
public class SensorService2 extends Service{

private static final String tag = "Service";
private static final boolean debug = true;

/*Default thresholds */
//private static final float UFT_max = 29.462866;
private static final float UFT_min = (float) 15.7600;
//private static final float LFT_max_low = (float) 8.0335653;
private static final float LFT_max = (float) 6.6700;
//private static final float LFT_min = 1.3408;
//private static final float ori_max = 158.0417;
private static final float ori_min = (float) 48.3900;
//private static final long time_delay_max = 1994180;
//private static final long time_delay_min = 524195;
private static final long tre_max = 366000;
private static final long tre_min = 102000;

/*Sensitive setting threshold*/
/*private static final float UFT_sensitive_min = 15.4099;
private static final float LFT_sensitive_max = 8.7308;
private static final float ori_sen_min = 45.6950;
private static final long time_delay_sen_max = 349130;
private static final long time_delay_sen_min = 2083468;*/

private static float curr_rss;
private static float prev_rss;
private static float curr_x;
```

```
private static float prev_x;
private static float curr_y;
private static float prev_y;
private static float curr_z;
private static float prev_z;
private static float curr_ori;
private static float prev_ori;
private static int counter;
private static int counter_2;
private static long timestamp_fall_detected, LFT_timestamp, UFT_timestamp;

private static Vector lft_timestamps;

//private static PowerManager pm;
//private static PowerManager.WakeLock w1;
private static SensorManager sm;
private static List<Sensor> sensorAcc;
private static SensorListener sl;

private static boolean isRunning = false;
public static boolean aboveAccThreshold = true;
private static boolean detectedFall = false;
private static boolean LFT_exceeded = false;
private static boolean UFT_exceeded = false;
//private static boolean limbo = false;
//private static boolean curr_on, prev_on = false;

public SensorService2() {
super();
if (isRunning == false) {
isRunning = true;
}
}

@Override
public void onCreate(){
if (debug==true) Log.i(tag, "onCreate2");

counter=0;
counter_2=0;
timestamp_fall_detected=0;
```

```
 curr_rss = 0;
 prev_rss = 0;
 curr_x = 0;
 prev_x = 0;
 curr_y = 0;
 prev_y = 0;
 curr_z = 0;
 prev_z = 0;
 curr_ori = 0;
 prev_ori = 0;

super.onCreate();
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
if (debug==true) Log.i(tag, "onStartCommand2");

sm = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
sensorAcc = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
sl = new SensorListener(this);

//pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
//w1 = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, tag);

if(sensorAcc.size() > 0)
{
Sensor sensor = sensorAcc.get(0);
sm.registerListener(sl, sensor, SensorManager.SENSOR_DELAY_NORMAL);
}

return super.onStartCommand(intent,flags,startId);
}

@Override
public void onDestroy() {
if (debug==true) Log.i(tag, "onDestroy2");
sm.unregisterListener(sl, sensorAcc.get(0));
sm.unregisterListener(sl);
isRunning = false;
super.onDestroy();
}
```

```
/**
 * Calculates the scalar product of two vectors
 * @param a x-axis for vector 1
 * @param b y-axis for vector 1
 * @param c z-axis for vector 1
 * @param d x-axis for vector 2
 * @param e y-axis for vector 2
 * @param f z-axis for vector 2
 * @param rss1 size of vector 1
 * @param rss2 size of vector 2
 * @return angle in degrees
 */
float orientation(float a, float b, float c, float d, float e, float f, float rss1, float rss2){
float ori =  (float) Math.toDegrees(Math.acos(((a*d)+(b*e)+(c*f))/(rss1*rss2)));
if(ori>=0.0) return ori;
else return (float) 0.0;
}

private class SensorListener implements SensorEventListener
{
SensorListener(Context context) {

}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
// TODO Auto-generated method stub

}

@Override
public void onSensorChanged(final SensorEvent se) {
new Thread(new Runnable() {
public void run() {
//if (FallDetection.serviceStarted == true){

if (se.sensor.getType()==Sensor.TYPE_ACCELEROMETER){

curr_x=se.values[0];
curr_y=se.values[1];
curr_z=se.values[2];
```

```
curr_rss=(float)Math.sqrt((curr_x*curr_x)+(curr_y*curr_y)+(curr_z*curr_z));

/* Orientation between position at timestamp x and timestamp(x-1) */
float ori = orientation(curr_x, curr_y, curr_z, prev_x, prev_y, prev_z, curr_rss, prev_rss);
curr_ori = ori;

prev_x = curr_x;
prev_y = curr_y;
prev_z = curr_z;
prev_rss = curr_rss;

if (debug==true) Log.i(tag, "2status per tick:"+counter+":orientation:"+ori+":RSS:"+(curr_rss)


/* This part counts the inactivity period */
if ((ori<3) && (prev_ori<3)){
counter_2++;
if (debug==true) Log.i(tag, "2orientation below 3:"+ori);
}else counter_2=0;

prev_ori = curr_ori;

/* An alarm is issued if a fall was detected followed by inactivity period */
if ((counter_2>3) && (((se.timestamp/1000)-timestamp_fall_detected)>520000 && ((se.timestamp/1
startActivity(new Intent(SensorService2.this, Alert2.class).setFlags(Intent.FLAG_ACTIVITY_NEW_

if (debug==true) Log.i(tag, "2new activity:"+((se.timestamp/1000)-timestamp_fall_detected)+":"

timestamp_fall_detected=0;
counter_2=0;
detectedFall = false;
}

/* LFT check */
if (curr_rss<=LFT_max  && UFT_exceeded==false){
if (debug==true) Log.i(tag, "2Lower fall threshall exceeded");

if (LFT_exceeded==false){

lft_timestamps = new Vector(1, 1);
LFT_timestamp=(se.timestamp)/1000;
```

```
}

lft_timestamps.add((long)(se.timestamp)/1000);

LFT_exceeded = true;
if (debug==true) Log.i(tag, "2LFT:"+((se.timestamp)/1000)+":"+ori+"::"+curr_rss);
}

/* UFT check */
if (curr_rss>=UFT_min  && LFT_exceeded==true){
if (debug==true) Log.i(tag, "2Upper fall threshold exceeded!");

UFT_exceeded = true;
UFT_timestamp=(se.timestamp)/1000;

Iterator it = lft_timestamps.iterator();

while(it.hasNext()){
Long tmp = (Long) it.next();
if (tmp != 0 && (UFT_timestamp-tmp) >= tre_min && (UFT_timestamp-tmp)<=tre_max) {
detectedFall = true;
timestamp_fall_detected = (se.timestamp)/1000;

if (debug==true) Log.i(tag, "2Fall:" +":"+ori+"::"+curr_rss+ "::"+(UFT_timestamp-tmp));

}
}

counter=0;
LFT_exceeded = false;
UFT_exceeded = false;

if (debug==true) Log.i(tag, "2UFT:"+((se.timestamp)/1000)+":"+ori+"::"+curr_rss+":::"+(UFT_timestamp-
}
}
}

}).start();

} // end onSensorChanged()
}
```

```
public class MyBinder extends Binder {
public SensorService2 getService() {
return SensorService2.this;
}
}

@Override
public IBinder onBind(Intent arg0) {
// TODO Auto-generated method stub
return null;
}

}
```

C.4 SensorService3.java

```
package falldetection.hig.no;

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

import android.app.Service;

import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

/**
 *
 * @author Ian James Daniel Gonzales
 * v1.3
 * This service uses algorithm 2
 * and with inactivity detection of at least 0.5 sec.
 */
```

```
public class SensorService3 extends Service{

private static final String tag = "Service";
private static final boolean debug = true;

/*Default thresholds */
//private static final float UFT_max = 29.462866;
private static final float UFT_min = (float) 16.0800;
//private static final float LFT_max_low = (float) 8.0335653;
private static final float LFT_max = (float) 5.2600;
//private static final float LFT_min = 1.3408;
//private static final float ori_max = 158.0417;
private static final float ori_min = (float) 48.3900;
//private static final long time_delay_max = 1994180;
//private static final long time_delay_min = 524195;
private static final long tre_max = 366000;
private static final long tre_min = 41000;

/*Sensitive setting threshold*/
/*private static final float UFT_sensitive_min = 15.4099;
private static final float LFT_sensitive_max = 8.7308;
private static final float ori_sen_min = 45.6950;
private static final long time_delay_sen_max = 349130;
private static final long time_delay_sen_min = 2083468;*/

private static float curr_rss;
private static float prev_rss;
private static float curr_x;
private static float prev_x;
private static float curr_y;
private static float prev_y;
private static float curr_z;
private static float prev_z;
private static float curr_ori;
private static float prev_ori;
private static int counter;
private static int counter_2;
private static long timestamp_fall_detected, LFT_timestamp, UFT_timestamp;

private static Vector lft_timestamps;

//private static PowerManager pm;
```

```
//private static PowerManager.WakeLock w1;
private static SensorManager sm;
private static List<Sensor> sensorAcc;
private static SensorListener sl;

private static boolean isRunning = false;
public static boolean aboveAccThreshold = true;
private static boolean detectedFall = false;
private static boolean LFT_exceeded = false;
private static boolean UFT_exceeded = false;
//private static boolean limbo = false;
//private static boolean curr_on, prev_on = false;

public SensorService3() {
super();
if (isRunning == false) {
isRunning = true;
}
}

@Override
public void onCreate(){
if (debug==true) Log.i(tag, "onCreate3");

counter=0;
counter_2=0;
timestamp_fall_detected=0;

 curr_rss = 0;
 prev_rss = 0;
 curr_x = 0;
 prev_x = 0;
 curr_y = 0;
 prev_y = 0;
 curr_z = 0;
 prev_z = 0;
 curr_ori = 0;
 prev_ori = 0;

super.onCreate();
}
```

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
if (debug==true) Log.i(tag, "onStartCommand3");

sm = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
sensorAcc = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
sl = new SensorListener(this);

//pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
//w1 = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, tag);

if(sensorAcc.size() > 0)
{
Sensor sensor = sensorAcc.get(0);
sm.registerListener(sl, sensor, SensorManager.SENSOR_DELAY_NORMAL);
}

return super.onStartCommand(intent,flags,startId);
}

@Override
public void onDestroy() {
if (debug==true) Log.i(tag, "onDestroy3");
sm.unregisterListener(sl, sensorAcc.get(0));
sm.unregisterListener(sl);
isRunning = false;
super.onDestroy();
}

/**
 * Calculates the scalar product of two vectors
 * @param a x-axis for vector 1
 * @param b y-axis for vector 1
 * @param c z-axis for vector 1
 * @param d x-axis for vector 2
 * @param e y-axis for vector 2
 * @param f z-axis for vector 2
 * @param rss1 size of vector 1
 * @param rss2 size of vector 2
 * @return angle in degrees
 */
float orientation(float a, float b, float c, float d, float e, float f, float rss1, float rss2){
```

```
float ori =  (float) Math.toDegrees(Math.acos(((a*d)+(b*e)+(c*f))/(rss1*rss2)));
if(ori>=0.0) return ori;
else return (float) 0.0;
}

private class SensorListener implements SensorEventListener
{
SensorListener(Context context) {

}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
// TODO Auto-generated method stub

}

@Override
public void onSensorChanged(final SensorEvent se) {
new Thread(new Runnable() {
public void run() {
//if (FallDetection.serviceStarted == true){

if (se.sensor.getType()==Sensor.TYPE_ACCELEROMETER){

curr_x=se.values[0];
curr_y=se.values[1];
curr_z=se.values[2];
curr_rss=(float)Math.sqrt((curr_x*curr_x)+(curr_y*curr_y)+(curr_z*curr_z));

/* Orientation between position at timestamp x and timestamp(x-1) */
float ori = orientation(curr_x, curr_y, curr_z, prev_x, prev_y, prev_z, curr_rss, prev_rss);
curr_ori = ori;

prev_x = curr_x;
prev_y = curr_y;
prev_z = curr_z;
prev_rss = curr_rss;

if (debug==true) Log.i(tag, "3status per tick:"+counter+":orientation:"+ori+":RSS:"+(curr_rss)
```

```
/* This part counts the inactivity period */
if ((ori<3) && (prev_ori<3)){
counter_2++;
if (debug==true) Log.i(tag, "3orientation below 3:"+ori);
}else counter_2=0;

prev_ori = curr_ori;

/* An alarm is issued if a fall was detected followed by inactivity period */
if ((counter_2>3) && (((se.timestamp/1000)-timestamp_fall_detected)>520000 && ((se.timestamp/1000)-t
startActivity(new Intent(SensorService3.this, Alert3.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK))

if (debug==true) Log.i(tag, "3new activity:"+((se.timestamp/1000)-timestamp_fall_detected)+":"+counte

timestamp_fall_detected=0;
counter_2=0;
detectedFall = false;
}

/* LFT check */
if (curr_rss<=LFT_max  && UFT_exceeded==false){
if (debug==true) Log.i(tag, "3Lower fall threshall exceeded");

if (LFT_exceeded==false){

lft_timestamps = new Vector(1, 1);
LFT_timestamp=(se.timestamp)/1000;

}

lft_timestamps.add((long)(se.timestamp)/1000);

LFT_exceeded = true;
if (debug==true) Log.i(tag, "3LFT:"+((se.timestamp)/1000)+":"+ori+"::"+curr_rss);
}

/* UFT check */
if (curr_rss>=UFT_min  && LFT_exceeded==true){
if (debug==true) Log.i(tag, "3Upper fall threshold exceeded!");

UFT_exceeded = true;
UFT_timestamp=(se.timestamp)/1000;
```

```
Iterator it = lft_timestamps.iterator();

while(it.hasNext()){
Long tmp = (Long) it.next();
if (tmp != 0 && (UFT_timestamp-tmp) >= tre_min && (UFT_timestamp-tmp)<=tre_max) {
detectedFall = true;
timestamp_fall_detected = (se.timestamp)/1000;

if (debug==true) Log.i(tag, "3Fall:" +":"+ori+"::"+curr_rss+ "::"+(UFT_timestamp-tmp));

}
}

counter=0;
LFT_exceeded = false;
UFT_exceeded = false;

if (debug==true) Log.i(tag, "3UFT:"+((se.timestamp)/1000)+":"+ori+"::"+curr_rss+":::"+(UFT_tim
}
}
}
//}
}).start();

} // end onSensorChanged()
}

public class MyBinder extends Binder {
public SensorService3 getService() {
return SensorService3.this;
}
}

@Override
public IBinder onBind(Intent arg0) {
// TODO Auto-generated method stub
return null;
}

}
```

C.5 Alert.java

```
package falldetection.hig.no;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
//import android.content.Intent;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.telephony.SmsManager;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
//import android.widget.Toast;
import android.widget.Toast;

public class Alert extends Activity{

private static final String tag = "Alert";
private static final boolean debug = true;

private boolean cancelled;
private CountDownTimer ct;
private TextView text;
private TextView alert_msg;
private Button alert_yes;
private Button alert_no;
private NotificationManager nm;

 public void onCreate(Bundle savedInstanceState) {
 if (debug==true) Log.i(tag, "Alert Activity Created");
  super.onCreate(savedInstanceState);

  this.runOnUiThread(new Runnable() {
public void run() {
```

```
  setContentView(R.layout.alert);

  text = (TextView) findViewById(R.id.alert);
  alert_msg = (TextView) findViewById(R.id.alert_msg);
  alert_yes = (Button) findViewById(R.id.alert_yes);
  alert_no = (Button) findViewById(R.id.alert_no);
  cancelled =false;

        ct = new CountDownTimer(20000, 1000){
@Override
public void onFinish() {

if (cancelled) finish();
else callHelp();

}

@Override
public void onTick(long millisUntilFinished) {
text.setText("Seconds Remaining: "+ millisUntilFinished/1000);
}
        }.start();

Uri ringURI = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
Notification notification =  new Notification();
notification.sound = ringURI;
long[] vibrate = new long[] {0, 1000, 0, 1000, 0, 1000};
notification.vibrate= (vibrate);
nm = (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
nm.notify(1, notification);

  }
  });

if (debug==true) Log.i(tag, "Alert Activity Finished");
 }

 @Override
 public void onDestroy(){
 nm.cancel(1);
 super.onDestroy();
```

```
 }

 @Override
 public void onResume(){
 if (debug==true) Log.i(tag, "onResume created");

 super.onResume();

 if (debug==true) Log.i(tag, "onResume finished");
 }

 public void yesAlert(View v){
 if (debug==true) Log.i(tag, "yesAlert");

  cancelled = true;
ct.cancel();
ct = null;
finish();
 }

 public void noAlert(View v){
 if (debug==true) Log.i(tag, "noAlert");

  cancelled = false;
  ct.cancel();
  ct=null;
  callHelp();
 }

 /*public void onStop() {
 super.onStop();
 }*/

 public void callHelp(){
 if (debug==true) Log.i(tag, "Call Help");

 text.setText("Calling for assistance");

try {
SmsManager sm = SmsManager.getDefault();
// here is where the destination of the text should go
String number = "0047********";
```

```
//String number = "0047********";
sm.sendTextMessage(number, null, "Emergency I fell!", null, null);

/*Code to make a phone call, currently disabled*/
        //Intent callIntent = new Intent(Intent.ACTION_CALL);
        //callIntent.setData(Uri.parse("tel:0047********"));
        //startActivity(callIntent);

        finish();
    } catch (ActivityNotFoundException e) {
        Log.e("helloandroid dialing example", "Call failed", e);
        finish();
    }

 /*Code to make a send e-mail, currently disabled*/
 /*Intent i = new Intent(Intent.ACTION_SEND);
 i.setType("message/rfc822");
 i.putExtra(Intent.EXTRA_EMAIL  , new String[]{"some.user@email.com"});
 i.putExtra(Intent.EXTRA_SUBJECT, "i fell");
 i.putExtra(Intent.EXTRA_TEXT   , "fall detected algorithm 1");
 try {
     startActivity(Intent.createChooser(i, "Send mail..."));
 } catch (android.content.ActivityNotFoundException ex) {
     Toast.makeText(Alert.this, "There are no email clients installed.", Toast.LENGTH_SHORT).s
 }*/

 cancelled = true;

 }
}
```