A FRAMEWORK FOR SCREENING AND CLASSIFYING OBSTRUCTIVE SLEEP APNEA USING SMARTPHONES

by

Mamoun Al-Mardini

A Thesis Presented to the Faculty of the
American University of Sharjah
College of Engineering
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates
June 2013

Approval Signatures

We, the undersigned, approve the Master's Thesis of Mamoun Al-Mardini

Thesis Title: A Framework for Screening and Classifying Obstructive Sleep Apnea Using Smartphones.

Signature **Date of Signature**

_____ _____
Dr. Fadi Aloul
Associate Professor, Department of Computer Science and Engineering
Thesis Advisor

_____ _____
Dr. Assim Sagahyroon
Professor, Department of Computer Science and Engineering
Thesis Co-Advisor

_____ _____
Dr. Tamer Shanableh
Associate Professor, Department of Computer Science and Engineering
Thesis Committee Member

_____ _____
Dr. Sameh Al Natour
Assistant Professor, Department of Management Information Systems
Thesis Committee Member

_____ _____
Dr. Assim Sagahyroon
Head, Department of Computer Science and Engineering

_____ _____
Dr. Hany El-Kadi
Associate Dean, College of Engineering

_____ _____
Dr. Leland Blank
Interim Dean, College of Engineering

_____ _____
Dr Khaled Assaleh
Director of Graduate Studies

Acknowledgment

First and foremost, I would like to express my gratitude to Allah, who always gives me the strength and patience to fulfill my dreams.

Further, I would like to express my highest appreciation to my advisers, Dr. Fadi Aloul and Dr. Assim Sagahyroon for their time, efforts, and invaluable support in every step during my thesis.

Moreover, I would like to thank Dr. Luai Al-Husseini/Al-Khalidi hospital for giving us the opportunity to run the tests on patients in the sleeping center.

Finally, my greatest appreciation and thanks go to my parents, brother, and sister for their unequivocal support and encouragement. Special thanks to my sister Dr. Diala Al-Mardeeni for helping me in understanding the medical part of this thesis.

To my lovely parents, brother, and sister

for their endless love and support

Abstract

Obstructive sleep apnea (OSA) is a serious sleep disorder which is characterized by frequent obstruction of the upper airway, often resulting in oxygen desaturation. The serious negative impact of OSA on human health makes monitoring and diagnosing it a necessity. Currently, polysomnography is considered the golden standard for diagnosing OSA, which requires an expensive attended overnight stay at a hospital with considerable wiring between the human body and the system. In the proposed research, we implement a reliable, comfortable, inexpensive, and easily available portable device that allows users to apply the OSA test at home without the need for attended overnight tests. The design takes advantage of a smatrphone's built-in sensors, pervasiveness, computational capabilities, and user-friendly interface to screen OSA. We use three main sensors to extract physiological signals from patients which are (1) an oximeter to measure the oxygen level, (2) a microphone to record the respiratory efforts, and (3) an accelerometer to detect the body's movements. The collected signals are then analyzed on the phone to deduce if the patient is suffering from OSA. In the proposed system, we have developed an Android application that is able to record and extract the physiological signals from the patients and analyze them solely on the smartphone without the need for any external resources. The smartphone is able to analyze the oximeter and accelerometer reading. Most health applications use smartphones to collect physiological readings, and then off load them to an external server for analysis. However, in this work we developed an integrated environment that collects and processes data on the smartphone, including the signal processing functions that analyze the recorded respiratory efforts. Finally, we examine our system's ability to screen the disease when compared to the golden standard by testing it on 17 samples. The results showed that 100% of patients were correctly identified as having the disease, and 85.7% of patients were correctly identified as not having the disease. These preliminary results demonstrate the effectiveness of the proposed system as compared to the golden standard and emphasize the important role of smartphones in healthcare.

Search Terms: Obstructive sleep apnea, screening, smartphones, android, physiological signals, oximeter, signal processing.

Table of Contents

Abstract 6

Chapter 1: Introduction 13

Chapter 2: Literature Review 16

2.1 Role of Smartphones in Healthcare 16

2.2 Role of Smartphones in Sleeping Disorders Diagnosis 18

2.3 Obstructive Sleep Apnea (OSA) Portable Devices 21

2.4 Methods Used to Extract Physiological Signals 24

Chapter 3: Methodology of Diagnosing Obstructive Sleep Apnea (OSA) 26

3.1 Pretest Probability of Obstructive Sleep Apnea (OSA) 27

3.2 Methods for Extracting Physiological Signals 28

3.2.1 Extracting Respiratory Efforts 28

3.2.2 Extracting Oxygen Saturation 29

3.2.3 Extracting Body Movements 29

3.3 Methods to Analyze Collected Physiological Signals 29

3.3.1 Oxygen Saturation 30

3.3.2 Body Movements 31

3.3.3 Respiratory Efforts 34

3.4 Determining the Final Diagnostic 37

3.4.1 Diagnostic Based on Partial Data Set 37

3.4.2 Diagnostic Based on Complete Data Set 37

3.5 OSA Severity 39

3.6 Accuracy Assessment 39

3.7 Validation 39

Chapter 4: System Architecture 40

4.1 Hardware Components 41

4.1.1 Pulse Oximeter 41

4.1.2 Smartphone 42
4.1.3 Microphone 43
4.1.4 Accelerometer 43
4.1.5 Bluetooth Interface 44
4.2 System Software 44
4.2.1 Android Software Development Kit (SDK) 44
4.2.2 Matlab 44
Chapter 5: Implementation 46
5.1 Software Development Environment 46
5.2 Application Components 46
5.2.1 DataCollector Activity 47
5.2.1.1 BluetoothManager Class 48
5.2.1.2 AudioRecorder Class 51
5.2.1.3 AccelerometerRecorder Class 52
5.2.2 Pretest Activity: 53
5.2.3 Diagnose Activity 54
5.2.3.1 Analyzing Oxygen Saturation and Accelerometer Readings 54
5.2.3.2 Analyzing Respiratory Efforts 55
5.2.3.2.1 Infinite Impulse Response (IIR) Filter 56
5.2.3.2.2 Butterworth Filter 56
5.2.4 Report Activity 56
5.3 Problems Faced During Implementation 57
5.3.1 Problem 1 – Running the Application for Long Periods 58
5.3.2 Problem 2- Processing Large Files 60
5.3.3 Problem 3-Bluetooth-Related Issues 61
5.4 Performance Tips 62
5.4.1 Avoid Creating Unnecessary Objects 62

5.4.2 Static Over Virtual 62
Chapter 6: Presenting and Discussing the Results 63
6.1 Patient Results 63
6.1.1 Results Based on Partial Data Set 63
6.1.2 Results Based on Complete Data Set 65
6.1.3 Comparison between the Partial and Complete Data Set Diagnostics 66
6.2 Analysis of the Respiratory Efforts and Oxygen Saturation 68
6.2.1 Respiratory Efforts Signal 69
6.3 Comparison between the Proposed System and the Golden Standard 71
6.4 Impact of Each Physiological Signal on the Final Diagnostic 72
6.5 Accuracy Factors 74
6.5.1 Sensitivity (True Positive Rate) 75
6.5.2 Specificity (True Negative Rate) 75
6.5.3 Positive Predictive Value 76
6.5.4 Negative Predictive Value 77
6.6 Comparison with Published Results 78
Chapter 7: Conclusion 79
References 80
Appendix 86
Appendix A: Application Code 86

List of Figures

Figure 1. An example of the wires connected to the patient for a polysomnography (PSG) test. Image from 14
Figure 2. Visual illustration of the smartphone's role in healthcare. 19
Figure 3. Visual illustration for the various types of obstructive sleep apnea (OSA) portable devices. 23
Figure 4. Illustration of the combination of the populations of patients with and without obstructive sleep apnea (OSA) with respect to the apnea/hypopnea index (AHI) cutoff, high pretest probability, true-positive, true-negative, and false-positive results. Reproduced from 28
Figure 5a. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on pulse oximeter and body movements. 32
Figure 5b. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on pulse oximeter and body movements. 33
Figure 6a. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on respiratory efforts. 35
Figure 6b. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on respiratory efforts. 36
Figure 7. Flowchart explaining the steps followed when diagnostic is based on a complete data set. 38
Figure 8. System architecture showing the main components and the data flow between them. 40
Figure 9. Smartphone penetration in selected countries in 2012. This chart covers males and females aged 18 and up 43
Figure 10. Main activities of the application and how they interact with each other. 47
Figure 11. Main parts of the DataCollector activity. 48
Figure 12. Settings used by the oximeter for sending and receiving 48
Figure 13. Oximeter data packet format 49
Figure 14. Pretest Activity. 53
Figure 15. Main components of the Diagnose activity. 54
Figure 17. Report Activity. 57
Figure 18. An ANR dialog displayed to the user when the application becomes unresponsive. 58
Figure 19. Comparison between the partial and complete data set diagnostic (patients). 67
Figure 20. Comparison between the partial and complete data set diagnostics (non-patients). 68
Figure 21. Respiratory effort for one of the patients during one hour. 69

Figure 22. Energy value during an apnea event. 70
Figure 23. Oxygen saturation during an apnea event. 70
Figure 24. Comparison between the proposed system and the golden standard. 72
Figure 25. Accuracy factors calculation 78

List of Tables

Table 1. A chronology of smartphone usage in healthcare applications..... 20
Table 2. A chronology of the obstructive sleep apnea (OSA) portable devices. 24
Table 3. Meaning of the oximeter data packet abbreviations 50
Table 4. Results of the obstructive sleep apnea (OSA) patients (partial data set). 64
Table 5: Results of the non-obstructive sleep apnea (OSA) patients (partial data set). 65
Table 6. Results of the obstructive sleep apnea (OSA) patients (complete data set). 65
Table 7. Results of the non-obstructive sleep apnea (OSA) patients 66
(complete data set). 66
Table 8. Comparison between the partial and complete data set diagnostic (patients). 67
Table 9. Comparison between the partial and complete data set diagnostics (non-patients). 68
Table 10. Comparison between the proposed system and the golden standard. 71
Table 11. Comparison between the physiological signals, complete data set, and golden standard. 73
Table 12. Root mean square error (RMSE) for all data sets. 74
Table 13. A comparison between obstructive sleep Apnea (OSA) systems. 78

Chapter 1: Introduction

Obstructive sleep apnea (OSA) is a common disorder affecting 2-4% of the adult population [1], and it is considered to be the most prevalent sleeping disorder. There are three types of sleep apnea, namely: central, obstructive, and mixed sleep apnea. Central Sleep Apnea (CSA) occurs when the brain sporadically fails to trigger the breathing muscles in the chest, while obstructive sleep apnea (OSA) occurs when air is physically blocked from flowing into lungs during sleep intermittently. Mixed Sleep Apnea (MSA) is a combination of both central and obstructive sleep apnea. Of the three, OSA is the most common while CSA and MSA are significantly rarer. In most cases, the patient is unaware of the breath stoppages because the body does not trigger a full awakening. As per the American Academy of Sleep Medicine (AASM), OSA is defined as a cessation in the airflow lasting for more than two breaths [2].

The National Sleep Foundation (NSF) reported that for adults to function healthily they should obtain seven to eight sleeping hours every night [3]. Frequent obstructions of airflow during this period have a considerable influence on the performance of the human during the daytime. OSA causes excessive sleepiness, non-restorative sleep, high blood pressure, cardiovascular diseases, memory loss problems, erectile dysfunction, personality changes, and depression [4-5]. Besides daytime tiredness, OSA patients may experience job impairment and automobile accidents.

Attended overnight polysomnography (PSG) is considered the golden standard for OSA diagnosis. To get an overnight OSA test, the patient should stay in a specialized sleep laboratory for more than one night, with 22 wires attached to his/her body in order to record and analyze several neurologic and cardio respiratory signals as shown in Figure 1. This brings great anxiety to the patients and they may not be able to sleep well during the night. Moreover, very few hospitals can accommodate the PSG test and it is rarely found in rural areas, which makes it unavailable for everyone and costly. Because of the aforementioned complications of using PSG, a need has arisen for portable devices with acceptable accuracy, high levels of usability, and depend on acquiring fewer biological signals [2], [5-8].

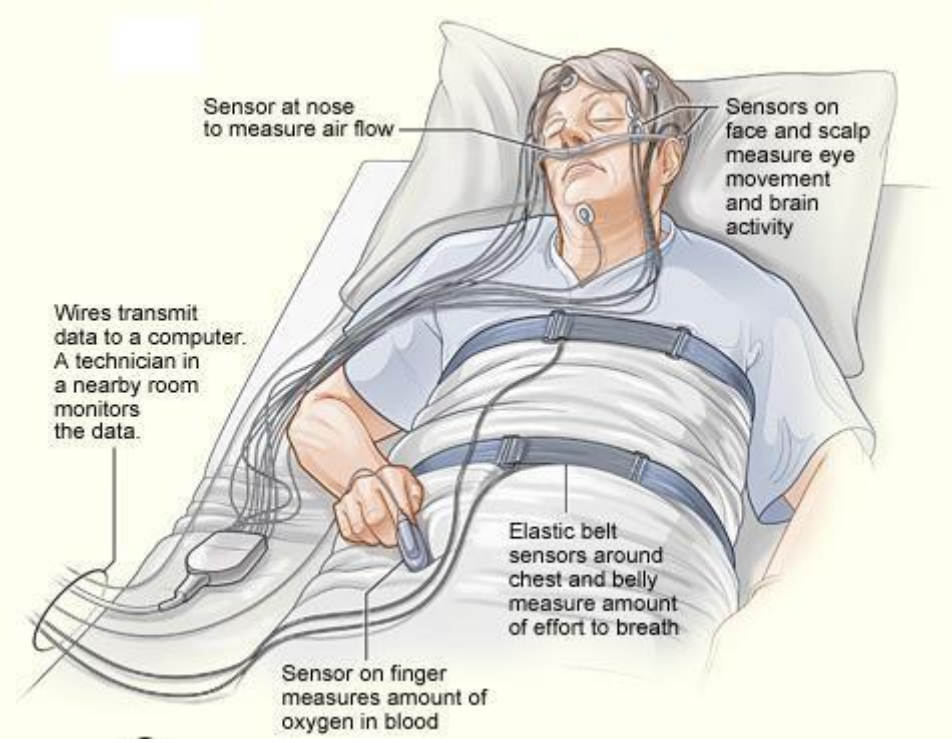

Figure 1. An example of the wires connected to the patient for a polysomnography (PSG) test. Image from [10].

According to the American Academy of Sleep Medicine (AASM), sleep apnea-diagnosing devices fall into four distinct categories [2]:

Category 1: Attended PSG that records at least seven biological parameters.
Category 2: Unattended PSG that records at least seven biological parameters.
Category 3: A portable device that records at least four biological parameters.
Category 4: A portable system that records one or two biological parameters using an oximeter as one of the parameters.

The proposed system focuses on implementing a portable device that is inexpensive, reliable, and accurate when compared with PSG. The proposed design makes use of the pervasiveness of smartphones by designing an Android application that is able to extract the biomedical readings from the patient and analyze them to screen and classify OSA. This screening is considered a preliminary test used to identify people at

high risk of having the disease who may need further confirmatory diagnostic tests at a hospital [11]. In the proposed system, we record three biological parameters to extract physiological signals from patients: (1) an oximeter to measure the oxygen level, (2) a microphone to record the respiratory efforts, and (3) an accelerometer to detect the body's movements. Therefore, the proposed system belongs to category 4 according to the AASM classification.

In the thesis, we develop an algorithm that combines and analyzes the physiological readings to reliably infer if the patient suffers from OSA. This algorithm draws from conclusions made in published literature, experimental data and tests collected in this work, and finally the consultation of a physician expert who has been actively involved in this project. All readings are analyzed solely on the smartphone without the usage of any external resources. Finally, we examine our system's ability to screen the disease as compared to the golden standard. We performed the test on 17 subjects, 8 of whom had already been diagnosed with OSA, 2 who had the symptoms, but were not diagnosed, and 7 subjects who are healthy with no symptoms. The results showed that 100% of patients were correctly identified as having the disease, and 85.7% of patients were correctly identified as not having the disease. Therefore, 14.3% of patients that did not have the disease were incorrectly identified as having the disease.

The rest of this thesis is organized as follows: Chapter 2 includes a literature review of related research. Methods for extracting the physiological signals and the proposed algorithms will be outlined in Chapter 3. Chapter 4 presents the proposed system architecture. Implementation of the application is explained in Chapter 5. Chapter 6 is a presentation and discussion of the results. Finally, Chapter 7 concludes this thesis.

Chapter 2: Literature Review

Rapid advances in technology have enhanced the capability of smartphones and added powerful features to them. Now, one can find phones with high computing capabilities, large capacity memories, built-in sensors, Bluetooth, Wi-Fi interfaces, and high resolution displaying options. Besides the technology, open standards play a significant role in increasing the importance of smartphones and encouraging developers to implement applications in different areas including communication, social, education, and healthcare. Kailas et al. [9] mentioned that there are already more than 7,000 documented applications concerned with healthcare, and with the increase in deployment of mobile phones, the role of smartphones in healthcare applications is expected to become significantly more pronounced in the coming years. The key features that give smartphones the advantage over the existing healthcare systems include portability, ease of use, and availability among people. Furthermore, smartphones may be used when clinicians are far away from their patients, or in rural areas where computer is not applicable [8][12].

In this section, we will review related work about OSA, and we will highlight the role of smartphones in healthcare in general and in OSA in particular.

2.1 Role of Smartphones in Healthcare

To detect abnormal Cardiovascular Disease (CVD), Jin et al. [13] and Oresko et al. [14] developed cell phone-based platforms that continuously record the electrical activity of the heart over a period of time (ECG) and analyze it in real-time.

Moreover, on CVD, Chen et al. [15] highlighted the fact that Heart Rate (HR) and Heart Rate Variability (HRV) can be accurately assessed from acoustic recordings of heart sounds using only a cell phone and a hands-free kit.

In other research efforts [16], the aim was to develop a system that is able to diagnose pneumonia, which is defined as inflammation within and around the alveolar spaces of the lungs. Pneumonia is considered as one of the world's leading killers,

especially in Africa. The researchers created software that can be installed on smartphones to analyze oximeter readings which are obtained from an inexpensive fingertip sensor.

Moving to respiratory diseases, Zhang et al. [17] developed a breathing bio-feedback system using a smartphone and a breathing sensor. Bio-feedback is considered as an alternative technique in which one can control his/her body functions to achieve the desired results, such as reducing pain. The system acquires breathing using a respiratory sensor, and transmits the signal to a smartphone using Bluetooth technology. The connected sensor provides information (feedback) for the body (bio), and the smartphone receives the breathing signals and generates a visual breathing feedback display to the user.

Another study by Scully et al. [18] measured the respiratory rate using the embedded camera in the smartphone. The proposed system used the smartphone to record and analyze the varying color signals of a fingertip placed in contact with the camera. In this research, the authors re-implemented the same algorithm used by the pulse oximeter, but using the functionalities of the smartphone.

Sudden Infant Death Syndrome (SIDS) [19] can also be detected by using the smartphone's built-in microphone, accelerometers, and other sensors to monitor the infant's heartbeat and respiration. The developed application is able to learn by itself through running machine-learning algorithms, and adapt itself as the environment changes around it. This application opened the door for smartphones to act as companion, helper, coach, and guardian.

2.2 Role of Smartphones in Sleeping Disorders Diagnosis

Sleeping disorders play a significant role in individuals' activities during the daytime, and can lead to complications that make the patient suffer from other diseases. For this reason, sleeping disorders attracted researchers in healthcare technology, and in the last few years, some applications have been developed on smartphones to monitor people during sleep. Figure 2 presents a visual illustration of the smartphone's role in healthcare, and Table 1 lists the history of smartphone utilization in healthcare. The following are some of these applications.

Bai et al. [20] proposed a novel approach to predict the sleep quality of individuals by using mobile sensors to collect users' context data including their daily activity, living environment, and social activity information. The approach was implemented on an Android-based phone, and three embedded sensors were used to collect data. These sensors included a Global Positioning System (GPS) sensor to track the person's position during the day, a microphone to record sounds from the user's surroundings, and an accelerometer to record the posture of the of subject including walking, running, sitting and standing.

In [21], Comtois et al. discuss a preliminary investigation of acquiring and processing breathing signals during sleep using a smartphone running an Android operating system. The hardware consists of a smartphone connected to an analog electronic stethoscope through an audio cable. The stethoscope is placed over the suprasternal notch located inferior to the larynx in order to get breathing sounds, and then sends the audio to the phone to do the analysis which is done on the phone itself, without the intervention of any external server. The purpose of the paper was to show that the phone is capable of performing some basic signal processing without diagnosing any disease or involvement of healthcare professionals.

Chen et al. [22] implemented a smartphone application that makes use of the embedded sensors in the phone to monitor different aspects of the patient before, during, and after the treatment. Five aspects were chosen to determine the patient's daytime activities: diet, fitness, stress, sleep, and life events. Four built-in sensors were used to

collect the aforementioned aspects: a GPS, accelerometer, camera, and microphone. For example, the accelerometer was used to detect movements of the patient during the day and while sleeping to estimate day activities and sleep quality.

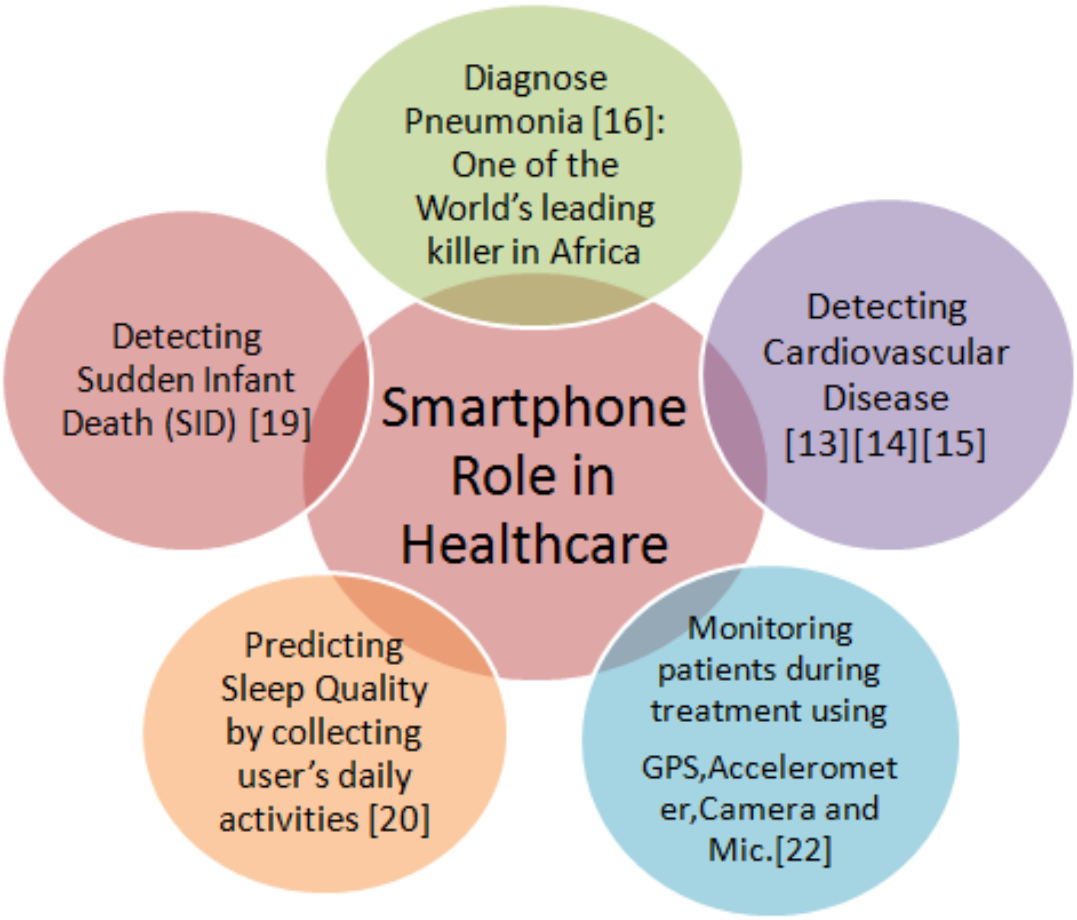

Figure 2. Visual illustration of the smartphone's role in healthcare.

	Year	Authors	Goal	Methods
1	2009	Oresko et al [14].	Real-time acquisitioning and displaying of heartbeats on the phone.	A smartphone is used to display the heartbeats collected by a wearablesingle-channel ECG sensor.
2	2009	Black et al [16].	Detecting pneumonia.	A smartphone is used to analyze the oximeter readings.
3	2010	Zhang et al [17].	Displaying breathing feedback to the user.	A smartphone is used to display the breathing feedback collected by an external server.
4	2010	Chen et al [22].	Monitor different aspects of the patient before, during, and after the treatment.	GPS, accelerometer, camera, and microphone are used to collect the user's activities, such as diet, fitness, stress, sleep, and life events.
5	2011	Scully et al [18].	Using a smartphone to replace the pulse oximeter.	A smartphone is used to record and analyze the varying color signals of the fingertip placed in contact with the camera.
6	2012	Siewiorek [19].	Detecting Sudden Infant Death Syndrome.	Machine-learning algorithms are used to analyze the readings of the built-in microphone and accelerometer to detect the infant's heartbeat and respiration.
7	2012	Bai et al [20].	Prediction of sleep quality.	Three sensors (GPS, accelerometer, and microphone) are used to collect the user's activities.
8	2012	Comtois et al [25].	Acquiring breathing signals.	A smartphone is used to display respiratory efforts collected by an analog electronic stethoscope.

Table 1. A chronology of smartphone usage in healthcare applications.

2.3 Obstructive Sleep Apnea (OSA) Portable Devices

HealthGear [23] is a one of the first Windows-based mobile applications that is used to automatically detect sleep apnea events from blood oximetry. The system consists of an oximeter to constantly monitor the user's blood oxygen level (SpO2). The oximeter is connected wirelessly via Bluetooth to a cell phone which stores, transmits, and analyzes the physiological data, and presents it to the user in a graphical format. HealthGear defines a baseline for the level of oxygen in the blood, and once the desaturation reaches this baseline an event will be detected. Also, in order to classify the severity of the apnea events, HealthGear also defines multi-threshold values to detect apnea events and classify them based on the threshold being passed.

Cao et al. [24] also used an oximeter combined with an accelerometer in their research to diagnose sleep apnea. Their work found that body posture can provide complementary information for analyzing the respiratory movement. Other research focused on the triaxial accelerometer to diagnose sleep apnea which has been widely used for physical activity detection [25][26].

An inexpensive OSA screening technique was proposed in [5], where the authors developed an accelerometer-based system by placing an accelerometer on the suprasternal notch. They then analyzed the data using signal processing techniques implemented on a microcontroller.

Rofouei et al. [27] extended the work done in [23-26] and developed a non-invasive, wearable neck-cuff system capable of real-time sleep monitoring and visualization of physiological signals. These signals are generated and recorded continuously from various sensors incorporated inside a soft neck-cuff and sent via Bluetooth to a cell phone which stores the data. The data is then analyzed using a microprocessor for detecting known sleep disorders such as OSA. The data is also stored to build a large database for further investigations such as data mining to establish a wellness baseline and possibly diagnose other diseases.

Alternatively, some research has focused on the recording of the heart's electrical activity over a period of time to diagnose OSA [8], [28].

Some researchers applied data mining techniques to the data recorded by the biomedical sensors, such as [6], where the authors followed data mining approaches to build an accurate classifier able to detect sleep apnea in real-time. The system builds the classifier based on a dataset, and new readings collected from the pulse oximeter are compared against the classifier.

However, Tseng et al. [29] developed an Android smartphone application to diagnose OSA based on some prediction rules derived by employing decision tree algorithms to a large clinical data set. In this work, neither wearable devices nor bio-sensors are used to collect data from the patient; instead an analysis server is used to extract decision rules from a clinical dataset which contains records of 540 OSA patients. Based on data mining techniques, decision rules are developed and sent to the smartphone. The goal of decision tree learning is to create a model that predicts the value of a target variable based on several input variables. To diagnose OSA, the user is required to enter his/her physiological data (e.g., age, gender, neck circumference and body weight) in the graphical user interface of the application. After that, these values are compared with the rules being extracted from the analysis server, and the result displays whether the individual suffers from OSA or not.

Peripheral Arterial Tone (PAT) technology has been used recently in portable devices [30-32]. WatchPAT100 [33] is a four-channel (level 3) screening portable device that uses two noninvasive self-adhesive finger probes to measure oxygen level and Peripheral Arterial Tone (PAT). PAT technology is a unique and relatively new concept of noninvasive measurement of sympathetic activation levels that is accurate for detecting Sleep Disordered Breathing (SDB). WatchPAT consists of three main components: the body, and two extended probes. One probe is the opticopneumatic sensor that detects the PAT signal; the other measures arterial oxygen saturation. The body of the device contains an accelerometer to detect body movements, which is used to differentiate sleep from wakefulness.

Montazeripouragha [34] hypothesized that breathing sound signals of patients with sleep apnea are significantly different from those of non-apneic individuals. In his work he proposed a technique for the assessment of OSA during wakefulness, by recording tracheal breath sounds in supine and upright positions during nose and mouth breathing at medium flow rate of 17 non-apneic individuals and 35 apneic individuals. At the end, he was able to extract the characteristic features from the respiratory sounds, and classify individuals with different OSA severity.

Figure 3 presents a visual illustration for the various types of OSA portable devices, and Table 2 lists the development of OSA portable devices in chronological order.

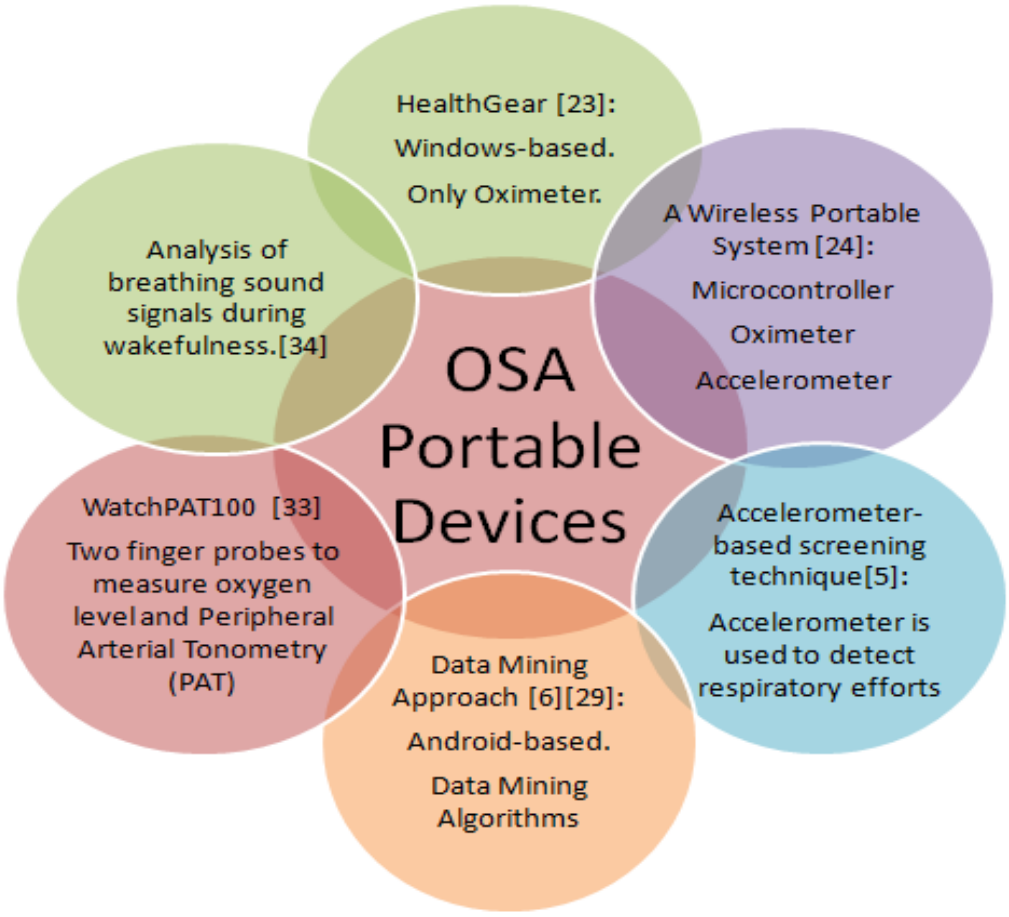

Figure 3. Visual illustration for the various types of obstructive sleep apnea (OSA) portable devices.

	Year	Author	Method
1	2003	Ayas et al [31].	Detects OSA using a wrist-word device with PAT technology.
2	2006	Mack et al [28].	Heart rate activities are recorded using an ECG sensor to detect OSA.
3	2007	Dargie et al [26].	A smartphone is used to analyze the oximeter readings.
4	2010	Bucklin et al [5].	An accelerometer is placed on the neck to detect breathing movements.
5	2011	Roufouei et al [27].	A wearable neck-cuff that contains an oximeter, accelerometer, and microcontroller is used to detect OSA.
6	2012	Cao et al [24].	A microcontroller is used to analyze the oximeter and accelerometer readings.
7	2012	Tseng et al [29].	Data mining techniques are used to detect OSA from oximeter readings.
8	2012	Montazeripouragha [34].	OSA is detected by differentiating the breathing sound signals of apneic and non-apneic individuals while they are awake.

Table 2. A chronology of the obstructive sleep apnea (OSA) portable devices.

2.4 Methods Used to Extract Physiological Signals

As mentioned earlier, polysomnography involves overnight recording of many body functions by using a large number of electrodes. On the other hand, portable devices use a small number of sensors to diagnose OSA. In the proposed design, we use the built-in sensors of the smartphone, so we are limited by a small number of sensors to extract the physiological signals. We take advantage of the smartphone's microphone, and accelerometer. Moreover, we use the Bluetooth interface to get reading from an external device such as a pulse oximeter to extract the oxygen saturation. Therefore, we extract respiratory efforts, oxygen saturation, and body movements using smartphone's

built-in sensors. We next discuss the techniques followed by researchers to extract these signals.

Respiratory efforts and oxygen saturation are the main readings used to diagnose sleep apnea in portable devices. Some approaches use one of them, both, or combine them with other sensors to diagnose sleep apnea.

Different sensors are used to extract respiratory effort from patients, and different approaches are followed by the designers of portable devices to detect respiratory cessations. Some research, such as [15], focused on the nose and mouth to record respiratory airflow. In this research, a device called ApneaLink was used to transmit oronasal airflow via a nasal cannula to a differential pressure transducer attached to the front of the patient's chest. After that the flow measurements were digitalized for storage and later downloaded to a computer. The validity of this approach is limited due to the problems with obstructed nostrils, since partial breathing occurs through the mouth [35].

Another approach followed by researchers is by analyzing respiratory sounds, including breathing and snoring, to devise noninvasive measures of increased respiratory effort from different places in the body, like the chest or throat. For example, in [36], three sound components (cardiac, respiratory, and snoring) were detected by means of a microphone connected to a chestpiece. This chestpiece was applied to the heart region to extract heart rate and respiratory effort for the diagnosis of OSA.

Another study presented by Moussavi et al. [37], [38] focused on recording respiratory sounds from the throat, and then processing the signal to separate cardiac and movement sounds from breathing and snoring sounds. Finally, a fuzzy algorithm was developed to use this information and detect apnea events.

One other approach is the recording of Midsagittal Jaw Movements (MJM, mouth opening) based on magnetic distance determination [39]. Respiration and snoring could be determined by placing a magnetic sensor on the chin and one on the forehead to record the relative jaw movements.

Chapter 3: Methodology of Diagnosing Obstructive Sleep Apnea (OSA)

As discussed in the literature review, there are several systems designed for healthcare purposes. Of the systems that address obstructive sleep apnea (OSA), only a few of them depend on smartphones. In this research, we present a smartphone-based system that is capable of predicting whether an individual suffers from obstructive sleep apnea or not.

The motivations for developing a portable device are the following:

- The popularity of smartphones.
- The complications of using polysomnography (PSG).
- The high cost of sleep disorders on governments, and the benefits of diagnosing the disease in early stages.
- Reducing the costs imposed by the use of PSG.
- Open standards which allow developing customized applications, and the flexibility of downloading them on.

The following is a list of the major tasks to be considered in this research:

- Identify the signs that indicate the probability of OSA, and develop a pretest as a first indication of the presence of the syndrome.
- Identify the methods used to extract physiological signals.
    - How to extract respiratory efforts.
    - How to extract oxygen saturation (SpO2).
    - How to extract body movements.
- Develop a method to analyze the collected physiological signals.
    - Analyze respiratory efforts.
    - Analyze oxygen saturation.
    - Analyze body movements.
- Classify OSA patients based on the severity of their case.
- Implement an Android application to analyze the collected physiological signals.

- Validate the proposed design by running tests on real patients and comparing them against the golden standard.

3.1 Pretest Probability of Obstructive Sleep Apnea (OSA)

Reviews reported by health agencies have revealed that portable devices used to diagnose sleeping disorders result in high relative error rates compared to Polysomnography (PSG). These error rates are either high false-negative (a result that wrongly indicates the absence of a disease) or false-positive (a result that wrongly indicates the presence of a disease). Due to the high false-negative and false positive ratios for portable devices that diagnose sleep apnea, it is recommended to apply a pretest on patients. The importance of a high pretest probability becomes clearer if studies that are focused on the general population are compared with studies focused on clinical populations. A high pretest probability reduces the number of false-positive diagnoses, since only those with a high pretest probability for sleep apnea will use a portable device for the diagnostic. Figure 4 illustrates the concept of false-positive and true-negative for the lowest apnea/ hypopnea index which is equal to 5 [35], [46].

The following is a list of symptoms collected from the complaints of OSA patients [35], [46]:

1) Loud and irregular snoring.
2) Observed or reported nocturnal cessation of breathing.
3) Excessive daytime sleepiness.
4) Unspecific mental problems such as fatigue, low performance, cognitive impairment.
5) Movements during sleep.
6) Morning dizziness, general headache, dry mouth.
7) Impaired sexual functions.
8) Obesity.
9) Arterial hypertension, cardiac arrhythmias.

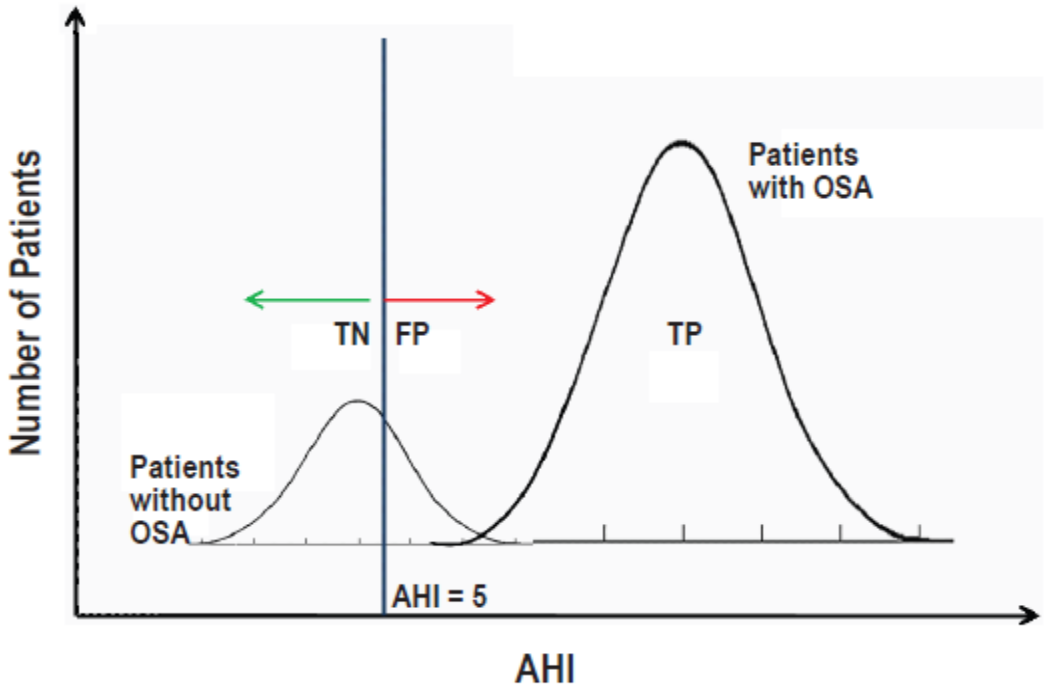

Figure 4. Illustration of the combination of the populations of patients with and without obstructive sleep apnea (OSA) with respect to the apnea/hypopnea index (AHI) cutoff, high pretest probability, true-positive, true-negative, and false-positive results. Reproduced from [46].

3.2 Methods for Extracting Physiological Signals

3.2.1 Extracting Respiratory Efforts

As discussed in the literature review, different methods have been used to extract respiratory efforts, such as placing a thermometer in front of patient's nose or mouth [15], or recording the airflow from the throat [37] or chest [36].

In the proposed design, we use the smartphone's microphone to extract respiratory efforts by placing it close to the throat (where the obstruction happens). Respiratory effort is recorded and stored on the phone's memory.

3.2.2 Extracting Oxygen Saturation

Oxygen saturation level is usually captured by an external pulse oximeter, which measures the oxygen level in the blood. An oximeter is a sensor that is placed on a thin part of the patient's body, usually a fingertip or earlobe, or in the case of an infant, across a foot. More details about the mechanism and our sensor choice will be discussed later in Section 4.1.1.

3.2.3 Extracting Body Movements

Body posture and movements can be detected by continuously taking the directions of the body along all axes. An accelerometer can be used to detect these movements. Researchers typically use an external accelerometer for this purpose. However, in the proposed design, we take advantage of the built-in accelerometer in the smartphone. The smartphone will be placed on the patient's arm, and body movements will be detected by calculating the displacement in the three axes (X, Y, and Z) as will be explained in Section 3.3.2.

3.3 Methods to Analyze Collected Physiological Signals

As discussed earlier in the definition of OSA, an apnea event is detected when there is a cessation of airflow for more than 2 breaths or at least 10 seconds [2]. Based on this definition, we need to detect the obstruction of the airflow, and the accompanied desaturation reflected in the blood. For this reason, we use a microphone that is able to record the respiratory efforts, a pulse oximeter to measure oxygen saturation in the blood, and an accelerometer to detect the movements of the patient. Due to the importance of the pretest in reducing the chances of a false-negative or false-positive, we integrate a pretest in the proposed design as discussed in Section 3.3.1.

3.3.1 Oxygen Saturation

In the proposed implementation, a pulse oximeter is used to measure oxygen level and provide continuous data transmission of a 4 byte data packet sent every second. In the proposed system, we use the Oxygen Desaturation Index (ODI) which is defined as the average number of events per hour [40]. An event is detected if the oxygen level is below the average by 4%, and lasts at least 10 seconds [40].

The following steps (illustrated in Figure 5) are followed to detect an apnea event using SpO2 readings:

- Read oxygen level every second.
- Initial calibration phase:
    - The level of oxygen differs between individuals, so we need to find the average in the first few minutes, and record it.
    - An event is detected if the oxygen level is 4% below the average calculated in the previous step. Therefore, a threshold is defined as *0.96*Average*.

      $X = 0.96 * N$, where X is the threshold and N is the average.
- Moving calibration phase:
    - After the occurrence of an apnea event, the oxygen saturation does not return to the same level just before the event. Therefore, we need to recalculate the average and the threshold again.

      $X\_New = 0.96 * N\_New$, where X_New is the new threshold and N is the new average.
- If the oxygen level detected is lower than the defined threshold, then we have to be sure that the event lasted at least 10 seconds.
- By the end of the sleeping period, ODI is calculated. OSA is confirmed if one of the following conditions is met:
    - ODI>= 10 or
    - (ODI>=5) && (high pretest probability) (Section 3.1).

3.3.2 Body Movements

Based on research reported by [27], [37], body movements may cause variation in the pulse oximetry readings. Therefore, any change in the oxygen level that is associated with body movement is eliminated as illustrated in Figure 5.

The 3-axis accelerometer integrated in the smartphone is used to measure the 3-axis accelerations of the human body along the directions of the three axes. Body positions are categorized into two types: motion and rest. To classify motion and rest, we need to detect the displacement of the three axes (X, Y, and Z). We may consider the displacement of each axis separately to detect motion or use a concept called Signal Vector Magnitude (SVM) proposed in [23] which reflects the motion intensity of the body. SVM is the square root of the sum of the 3-axis accelerations, the amplitude of which mirrors the motion amount of the body. Therefore, rather than dealing with each axis separately to detect motion, we use the SVM concept to combine the values of all axes and detect the motion. In the proposed system, we measured the SVM value of the body in different positions, and noticed the difference between them. For example, the SVM value of the body when the patient is lying on his/her back ranges from {-1,4}. However, the value ranges between {9,15} when the patient is lying on his/her side. Therefore, if the difference is more than 5, then a movement is detected.

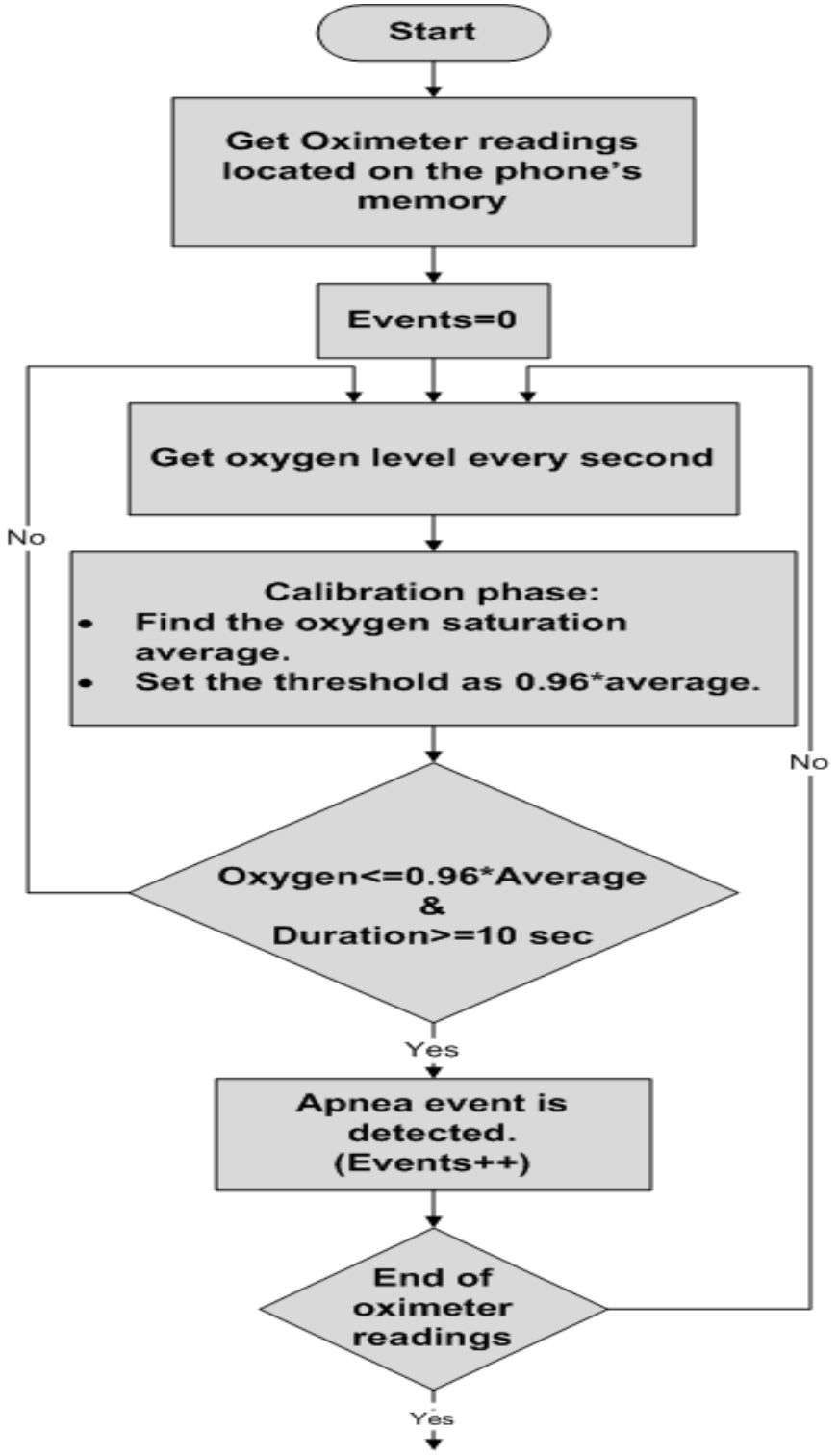

Figure 5a. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on pulse oximeter and body movements.

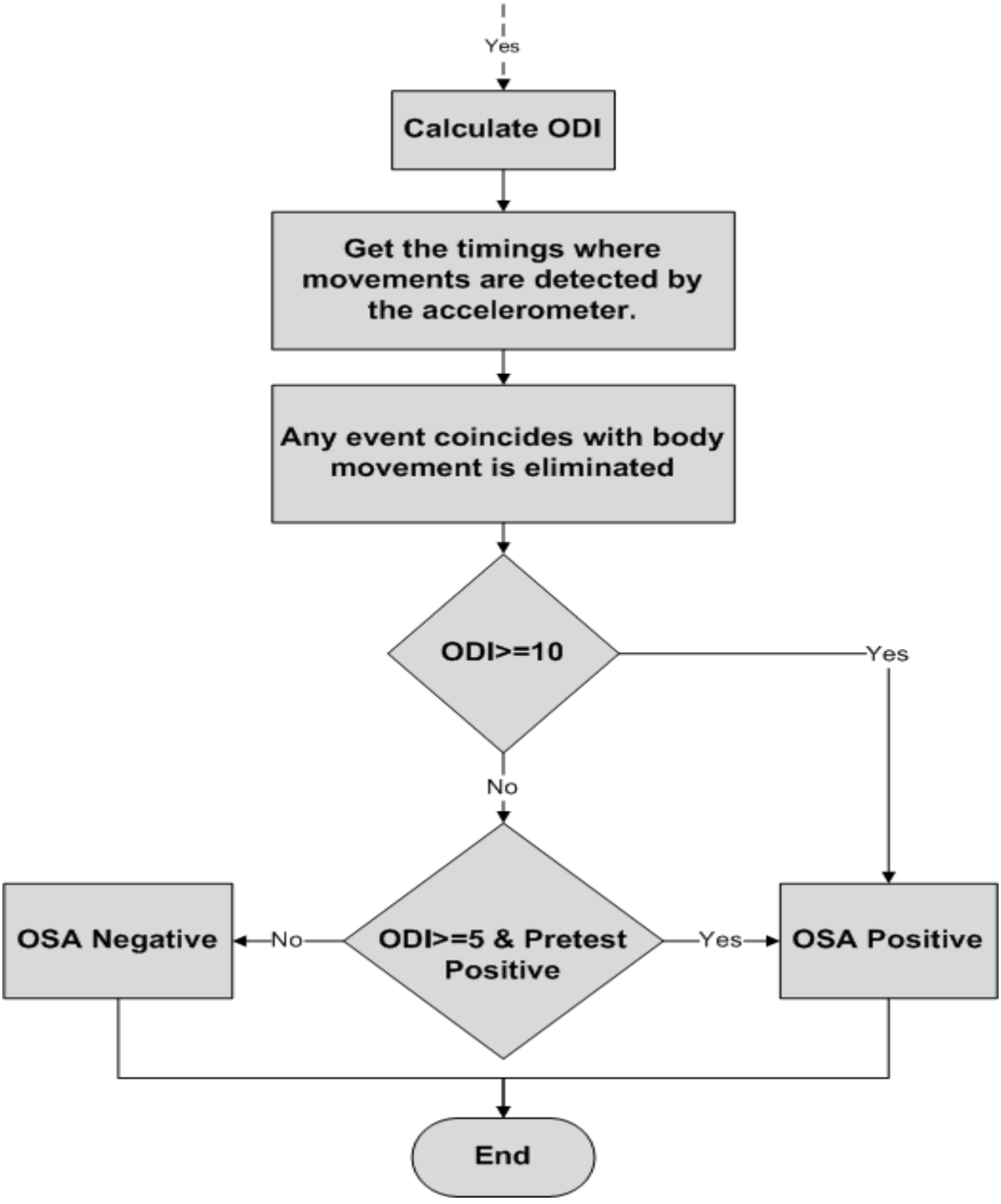

Figure 5b. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on pulse oximeter and body movements.

3.3.3 Respiratory Efforts

In this stage, we record the patient's airflow by using a microphone attached to the patient's throat. Frequency of breathing is different among people but experimentally has been proven to fall between {200,800} Hz [37]. Extracting this range of frequencies allows us to exclude noise and analyze the breathing signal only. Signal processing functions are required to analyze this rage of frequencies. The correctness and accuracy of these functions are first verified using Matlab software running on an external server, and then applied on the smartphone using Android environment. These functions will be discussed in Section 5.2.3.2.

The following steps are followed to detect an apnea event using respiratory effort readings only, as illustrated in Figure 6:

- Read respiratory signal every second.
- Initial calibration phase:
    - The energy of the breathing signal differs between individuals, therefore we need to find the average in the first few minutes, and record it.
    - An event is detected if the energy is below 90% of the average calculated in the previous step. Therefore, a threshold is defined as *0.9*Average*.
      $X = 0.9 * N$, where X is the threshold and N is the average.
    - Threshold value can vary due to the differences in the devices being used and their sensitivity [1].
- If the energy of the breathing signal is detected to be lower than the defined threshold, then we have to be sure that the event lasts at least 10 seconds.
- By the end of the sleeping time, apnea/hypopnea index (AHI) which is defined as the average number of events per hour [2] is calculated. OSA is confirmed if one of the following conditions is met:
    - AHI>= 10 or
    - (AHI>=5) && (high pretest probability) (Section 3.1).

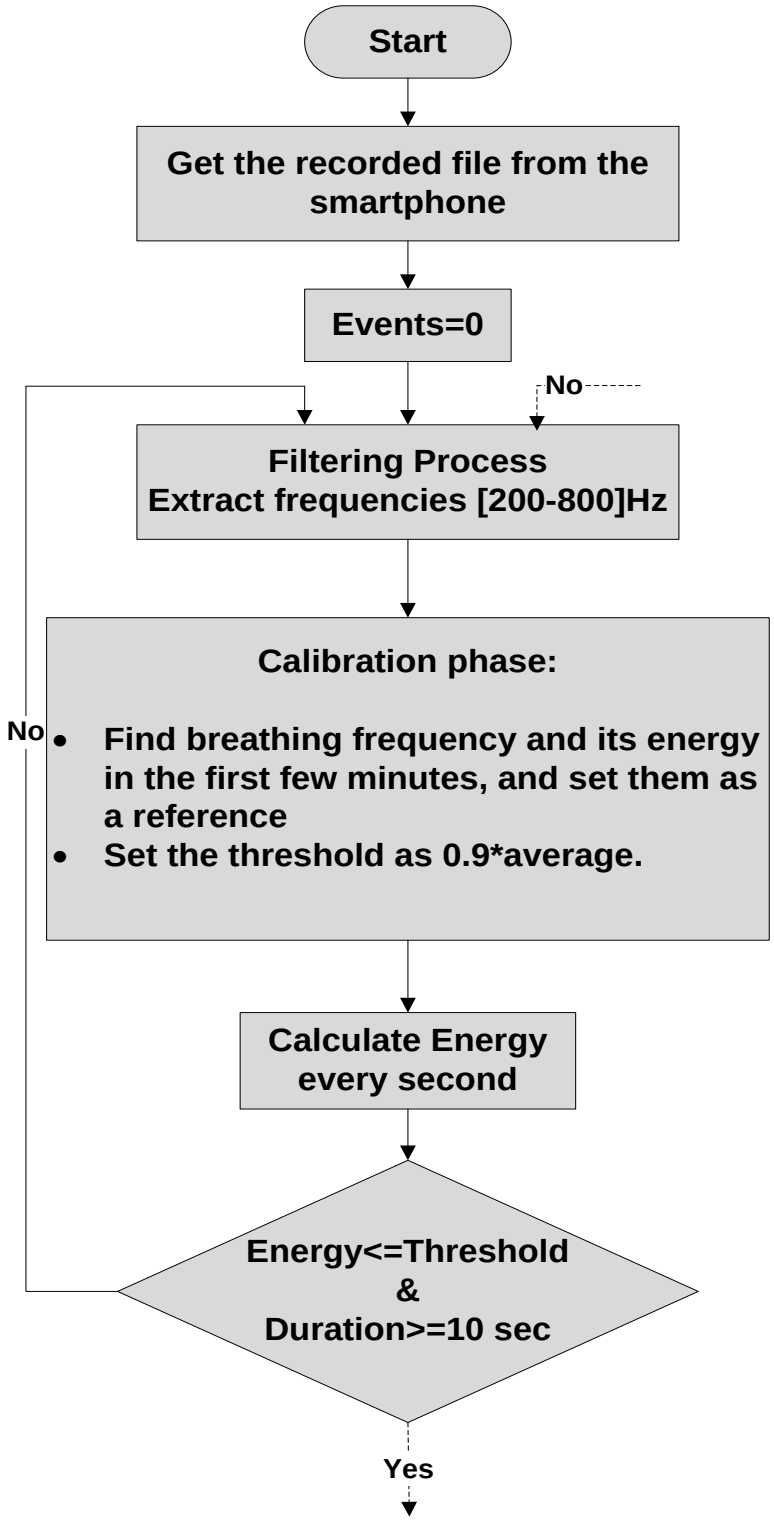

Figure 6a. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on respiratory efforts.

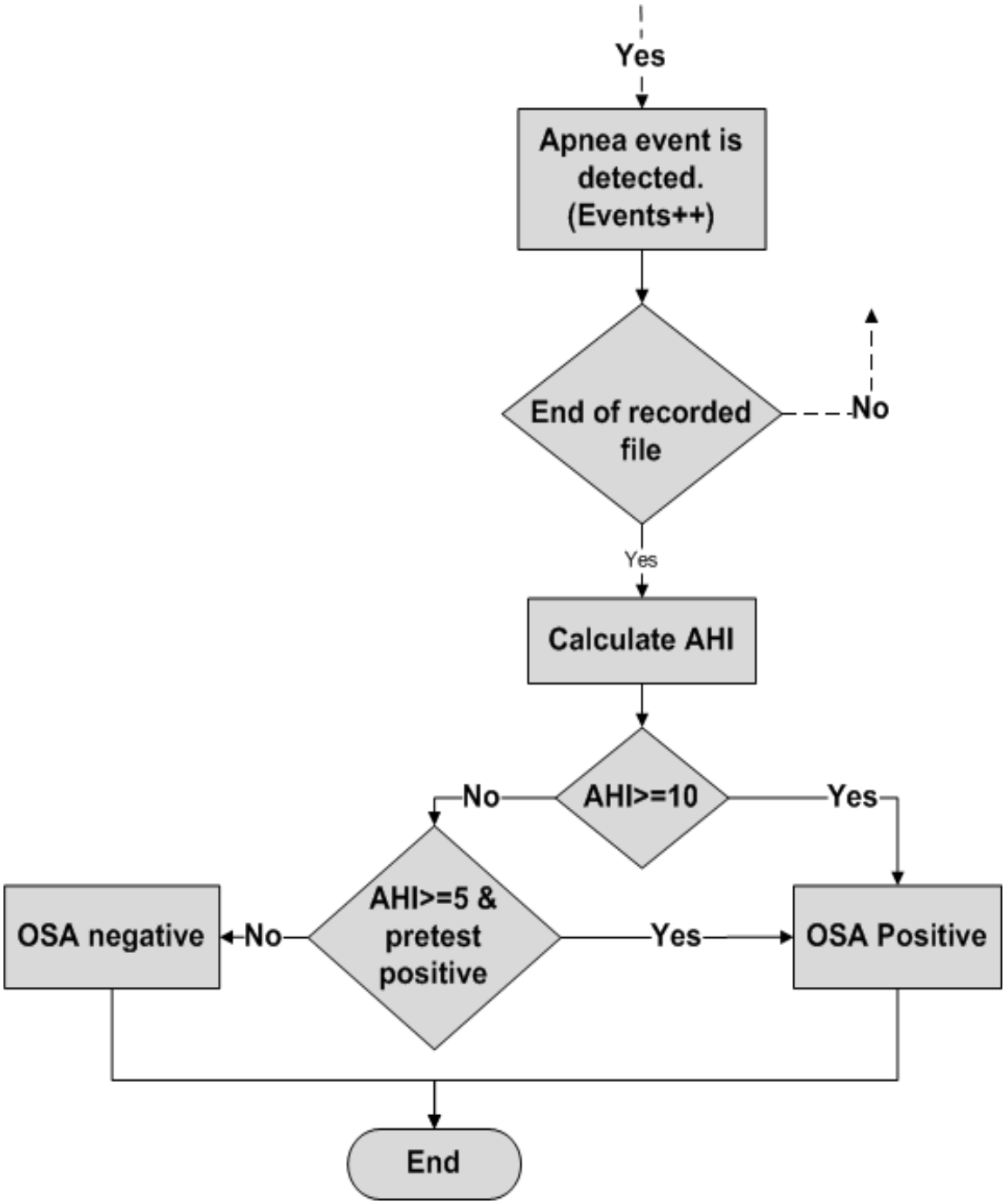

Figure 6b. Flowchart explaining how to diagnose obstructive sleep apnea (OSA) based on respiratory efforts.

3.4 Determining the Final Diagnostic

Respiratory effort analysis requires filtering and calculating the energy every one second as described in Section 3.3.3. Initially, we proposed to send the respiratory efforts data to an external server using a WiFi or 3G connection to run the required analysis using Matlab software. Therefore, we proposed two approaches to obtain the final diagnostic. The first approach is based on two physiological signals: oxygen saturation and body movements. The second approach is based on all the physiological signals, and requires interaction with an external server to analyze the respiratory efforts. Later, we decided to implement the filtering and energy calculation on the smartphone, and exclude the external server from the system architecture. However, we preferred to keep the two proposed approaches to compare the accuracy of having two or three physiological signals on one hand, and to compare the results of analyzing the respiratory efforts on an external server against a smartphone on the other hand.

3.4.1 Diagnostic Based on Partial Data Set

This approach is based on two physiological signals: oxygen saturation and body movements. The benefit of using this approach is that the user is not required to upload the respiratory efforts to an external server or wait for the analysis on the smartphone. Uploading the respiratory efforts to an external server requires WiFi or a 3G connection. Furthermore, analyzing the respiratory efforts on the smartphone takes a long time (around 2 hours). On the other hand, building the diagnostic on two physiological signals is less accurate than on three signals.

3.4.2 Diagnostic Based on Complete Data Set

This approach is based on all physiological signals: respiratory efforts, oxygen saturation, and body movements. Analyzing three physiological signals will yield more accurate results when compared to the previous approach. However, this approach requires a long time either in uploading the respiratory efforts to an external server, or in

analyzing it on a smartphone. Figure 7 displays the flowchart of the steps followed when the diagnostic is based on a complete data set.

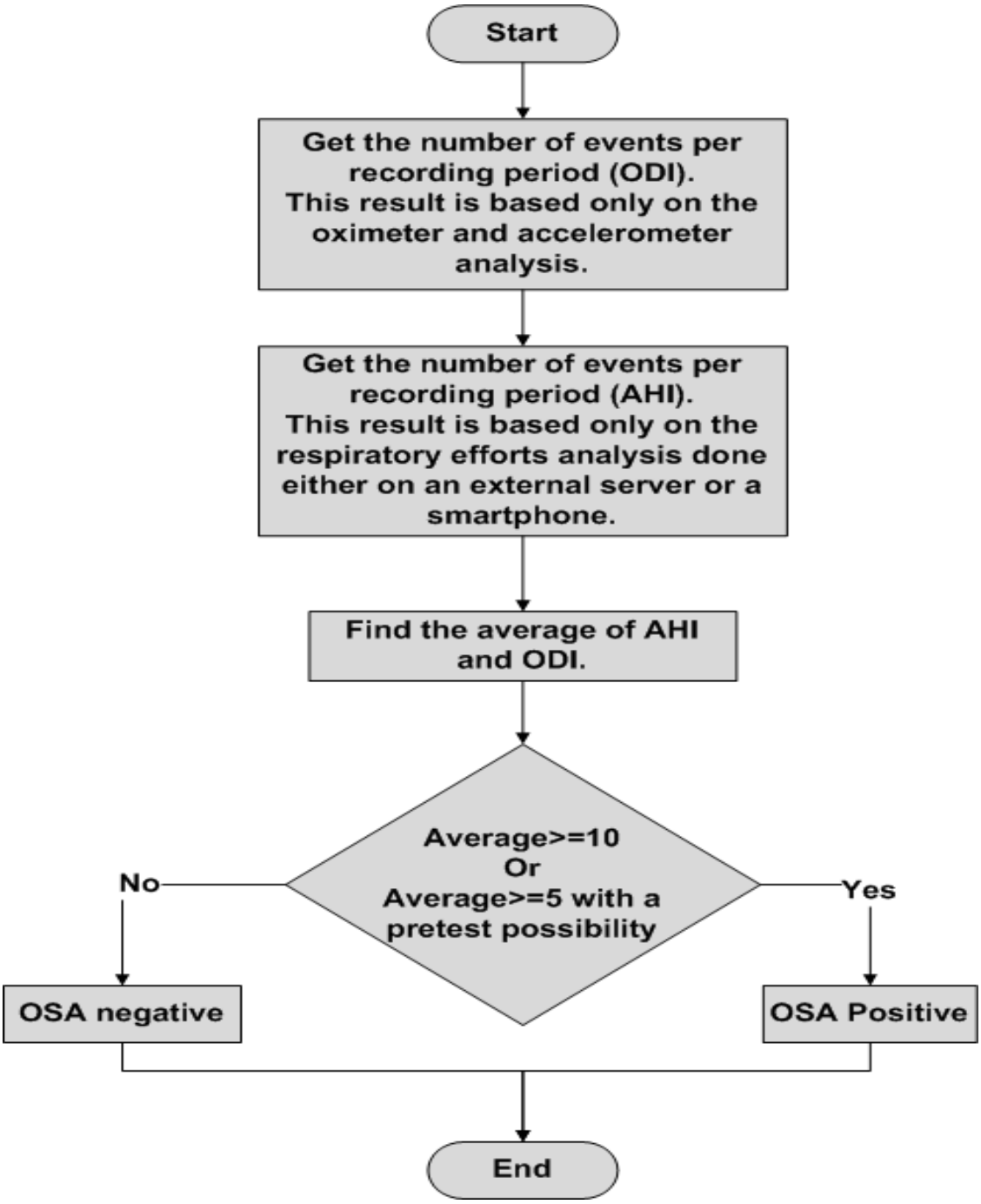

Figure 7. Flowchart explaining the steps followed when diagnostic is based on a complete data set.

3.5 OSA Severity

OSA is classified into three main categories according to its severity: mild, moderate, and severe. The frequency of obstruction events is reported as an apnea/hypopnea index (AHI) or as an oxygen desaturation index (ODI). Both are used synonymously, but ODI is usually used when we are only concerned with oxygen. After finding the number of obstruction events using either one of the approaches described in Section 3.4, we will report the severity of OSA based on the following classification [1]:

- Mild : for AHI or ODI >=5 and <15
- Moderate: for AHI or ODI>=15 and <=30
- Severe: for AHI or ODI >30

3.6 Accuracy Assessment

In either approach, we will get different results for AHI/ODI. In the proposed system, we will compare these results with the actual value derived from the golden standard (PSG), and assess how each approach is close to this reference value. Results are discussed in Sections 6.1.1 and 6.1.2.

3.7 Validation

One of our goals in this research is to develop a reliable system that is able to diagnose OSA. Consequently, we need to validate the proposed design by testing the application on OSA and non-OSA patients, and see how accurate the system is when compared with the golden standard (PSG). All test results and the comparison between the proposed system and golden standard are listed and discussed in Chapter 6.

Chapter 4: System Architecture

In this chapter, we will describe the system architecture and discuss the hardware and software components. As shown in Figure 8, the system consists of four hardware components, which are: a smartphone as a central processing point, an oximeter to record oxygen saturation, a microphone to record respiratory efforts, a built-in accelerometer to record body movements, and a Bluetooth interface that is used as a gateway between the smartphone and the oximeter. On the other hand, the system makes use of two software environments, which are: Android Software Development Kit (SDK) and Matlab. The hardware and software components are explained in the following subsections.

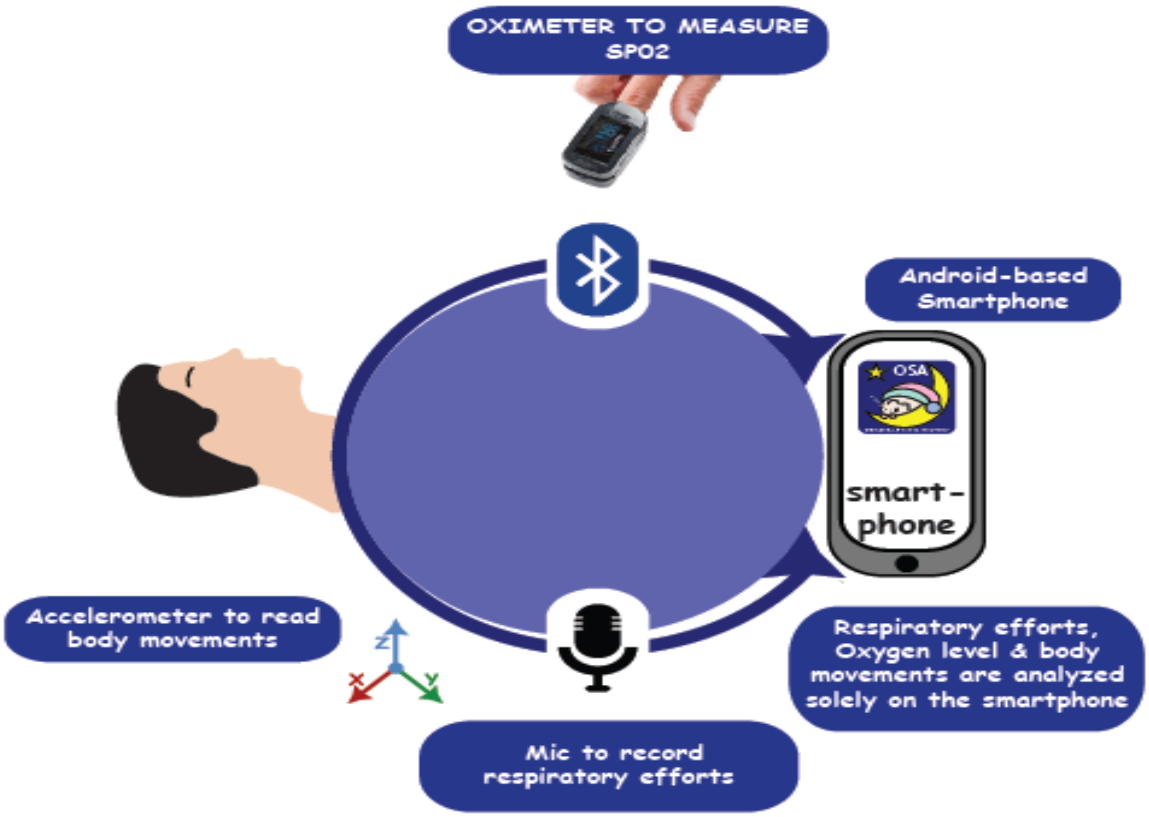

Figure 8. System architecture showing the main components and the data flow between them.

4.1 Hardware Components

As described in Figure 8, we have 5 main components that interact with each other to extract data from the patient, process it, and display results to the user. The following is a brief description of each component.

4.1.1 The Pulse Oximeter

Saturation of peripheral oxygen (SpO2) is an estimation of the oxygen saturation level in the blood. It is usually captured by a pulse oximeter, which is a non-invasive method to measure the percentage of hemoglobin saturation with oxygen. An oximeter is mainly a sensor that is placed on a thin part of the patient's body, usually a fingertip or earlobe, or in the case of an infant, across the foot.

The mechanism of the pulse oximeter in measuring oxygen saturation is based on the absorption of light. Blood with a normal level of oxygen absorbs red light at wavelengths of 660nm, whereas blood with low oxygen saturation absorbs infrared light at wavelengths of 940nm. An oximeter consists of emitters and detectors to emit 600nm and 940nm lights, and to detect the traversed red and infrared lights through the tissues. It measures oxygenation based on the ratio of changing absorbance of the red and infrared lights [24].

In the proposed system, we use a NONIN Model 9560 finger pulse oximeter. This is a small, lightweight (63 grams), portable, wireless device capable of long-term monitoring, all of which make it particularly suitable for home use. The oxygen saturation range (SpO2) measured by the oximeter is 0 - 100% and the accuracy is 70-100% [41]. The cost of this device is about $300.

4.1.2 The Smartphone

A report published by the US National Center for Health Statistics at the CDC (Centers for Disease Control and Prevention) in December 2010 revealed that more than half of Americans aged 25-29 now live in households with mobile phones but no traditional landline telephones [42]. Also, a study done by Google in 2011 and 2012 highlighted that the number of smartphones in different countries is increasing, and it is clearer in developing countries as shown in Figure 9. This penetration has captured the entire age spectrum in different fields such as education, healthcare, and medicine.

High availability, high computing capabilities, large memories, built-in sensors, Bluetooth and Wi-Fi interfaces, and fancy display options are the main reasons we have chosen to use a smartphone as the center of the proposed design.

We use an Android-based smartphone in the system, since we can use open source libraries to develop our customized application. Further, the application can run on all Android models, including Samsung, Sony, HTC, T-Mobile, and Google Nexus. More details about the software will be discussed in the software components section. In the proposed design, we chose Galaxy SII due to its availability in the research lab. This smartphone operates with a Dual-core 1.2 GHz Cortex-A9, with an internal storage of 16 GB/32 GB, 1 GB RAM, and an external storage of up to 32GB.

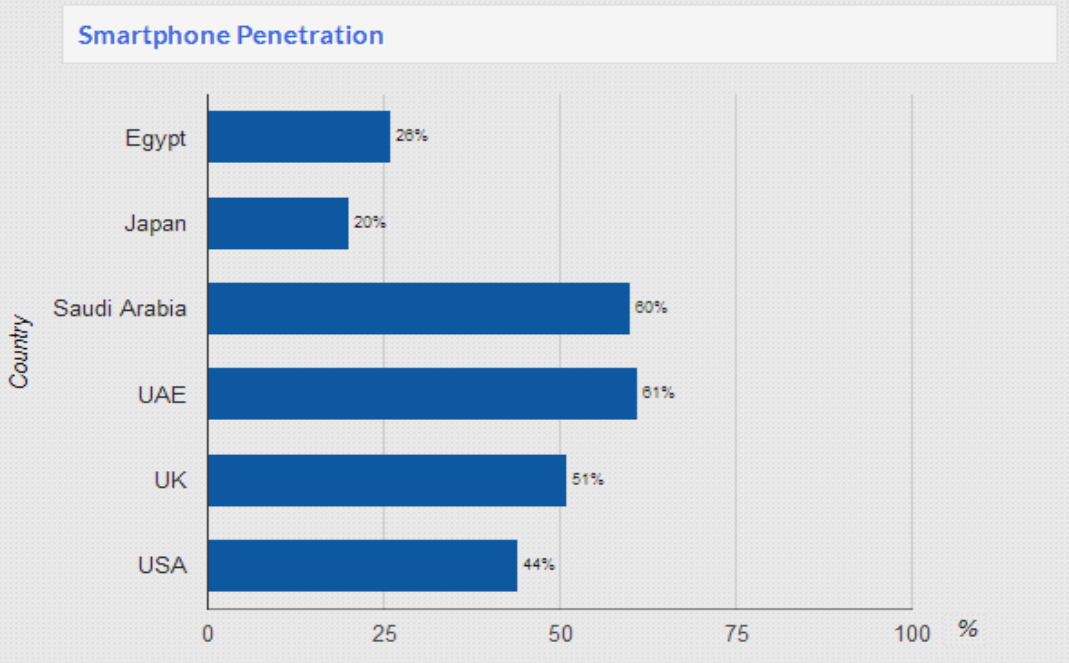

Figure 9. Smartphone penetration in selected countries in 2012. This chart covers males and females aged 18 and up [43].

4.1.3 The Microphone

The frequency response of microphones differs in smartphones due to the different audio circuits designed by manufacturers. This difference lies below 100 Hz, and most of the smartphones are able to detect audio sounds with frequencies of more than 100 Hz. In the proposed design, we are interested in the frequencies that fall in the {200,800} Hz range, and since the smartphone microphone has a frequency response of more than 100 Hz, this makes it suitable for the proposed system.

4.1.4 The Accelerometer

An accelerometer sensor is one of the most commonly used built-in sensors in every smartphone. The accelerometers inside smartphones have different designs, but all of them measure the 3-axis accelerations of the subject body along the directions of the three axes (i.e., the X axis, Y axis and Z axis). Acceleration precision has no effect on our

calculation, since the body will be in a stationary position, and may only make slow movements during the sleeping period.

4.1.5 Bluetooth Communication

The general transmission range of the Bluetooth module is 10m, which is appropriate for home monitoring. We chose Bluetooth because of its popularity and availability in most smartphones. The Bluetooth module used in the oximeter has a compliance version of 2.0 and operating frequency of 2.4 to 2.4835 GHz.

4.2 System Software

Since we are using a phone and an external server to process the data, we will use two developing environments: Android Software Development Kit and Matlab.

4.2.1 Android Software Development Kit (SDK)

The Android SDK provides the API libraries and developer tools necessary to build, test, and debug applications for Android [44]. Android is an open source Linux-based operating system, now considered the most popular OS for mobile phones. This means that if the user changes his/her phone, he/she only has to download the application to the new phone. In the proposed system, we built the application using Java which is a popular and relatively easy-to-use programming language. We also use an Android smartphone with an operating system of version 4.0.4.

4.2.2 Matlab Environment

Matlab is a high-level environment for numerical computation, visualization, and programming. Using Matlab, one can analyze data, develop algorithms, and create models and applications [45]. As mentioned in Section 3.4, we have proposed to use an

external server to analyze the respiratory efforts, then transport everything to the smartphone. Therefore, respiratory efforts can be analyzed on a smartphone or on a server. Analysis on the server is accomplished using Matlab software which provides signal processing built-in functions that ease the analysis of the respiratory efforts. Before transporting the analysis software to the smartphone, we verified its functionality using the Matlab environment.

Chapter 5: Implementation

In Chapters 3 and 4, the algorithms and system architecture needed in the diagnostic process have been discussed and explained in detail. In this chapter, we will describe the main components of the system and explain how each component was developed. Furthermore, we will show how the components interact in the system. Finally, a discussion of the problems faced and the steps taken to overcome them will be presented.

5.1 Software Development Environment

In developing the proposed application, we used the Java programming environment embedded within the Android Software Development Kit (Android SDK). The Android SDK provides comprehensive facilities for developers. It provides the API libraries and tools necessary to build, test, and debug applications for Android-based platforms.

The reason for choosing Android SDK to develop the proposed application is that Android is open-source; this means it uses permissive licensing which allows developers to freely modify and redistribute any software. Additionally, Android operating system has surged in recent years, and its popularity keeps increasing in the market. As of October 2012, more than 700 thousands Android applications have been developed, with over 25 billion downloads [47], [48].

5.2 Application Components

The proposed application consists of four main activities: a DataCollector activity, a Pretest activity, a Diagnose activity, and a Report activity. An “activity” is an application component that provides a Graphical User Interface (GUI) with which a user can interact in order to trigger an action, such as dial the phone, send an email, or take a photo. An application usually consists of multiple activities that are loosely bound to each

other [49]. Figure 10 shows the main activities of the application and how they interact with each other.

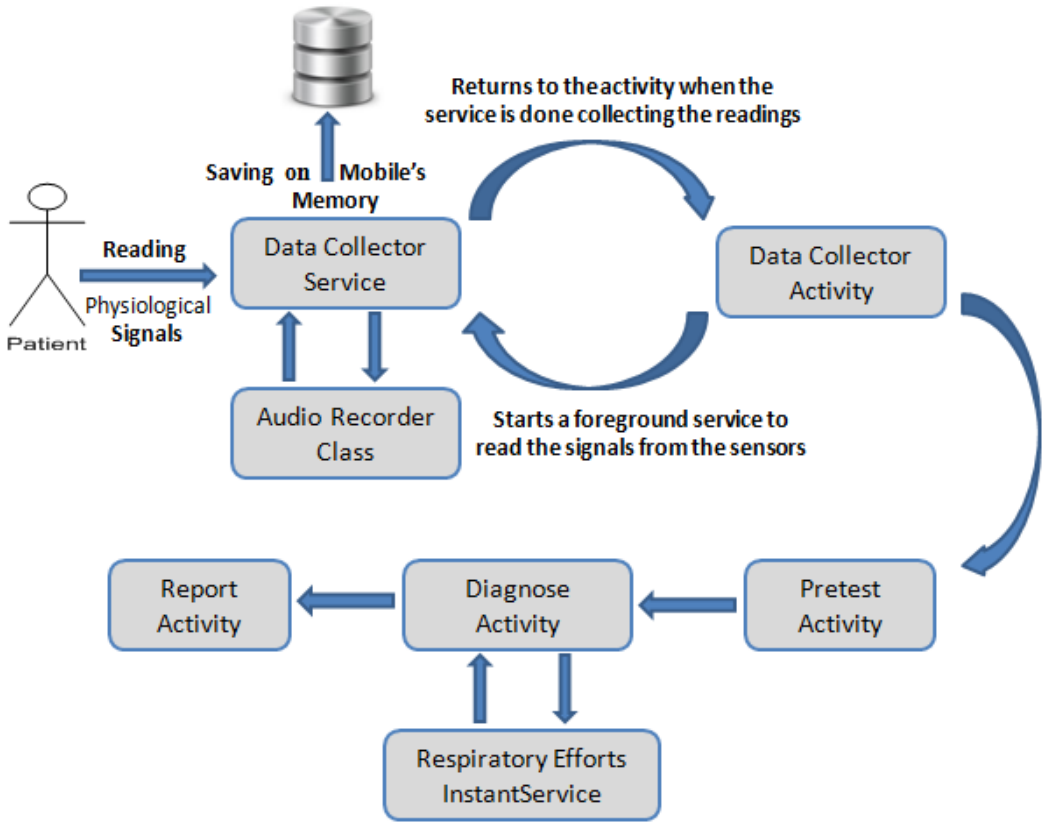

Figure 10. Main activities of the application and how they interact with each other.

5.2.1 DataCollector Activity

The *DataCollector* activity is considered to be the main activity in the application. This activity is the one presented to the user when the application is launched. It is responsible for starting the process of collecting the physiological signals from the patient. Since we aim to collect three readings from patients, the DataCollector activity uses three main classes divided based on the functionality: *AudioRecorder*, *BluetoothManager*, and *AccelerometerRecorder*. These classes are responsible for recording the respiratory efforts, oxygen saturation, and accelerometer values, respectively. Figure 11 shows these three parts.

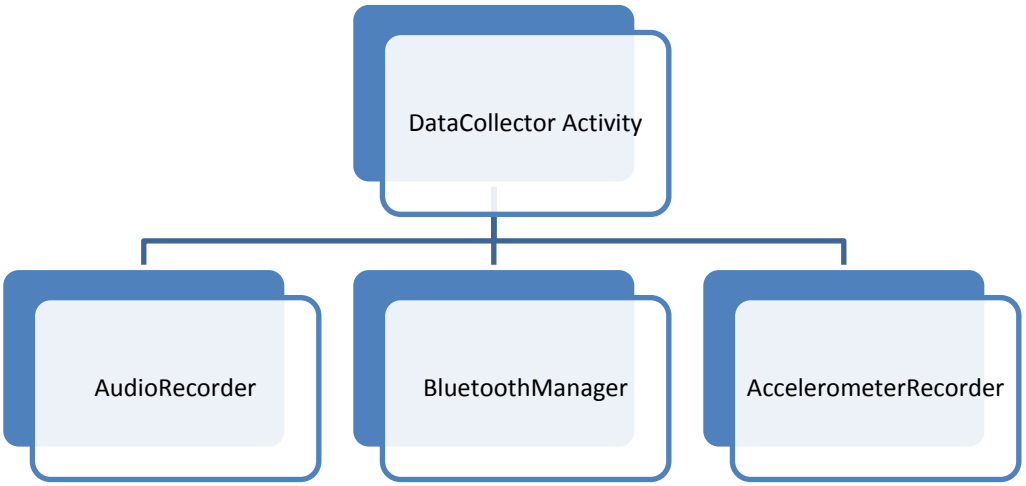

Figure 11. Main parts of the DataCollector activity.

5.2.1.1 BluetoothManager Class

The Bluetooth Manager is responsible for pairing and starting the connection with the oximeter. The oximeter sends the oxygen saturation readings to the smartphone every one second using Bluetooth protocol. The oximeter is considered as a slave device, while the smartphone is considered as a master device. To connect the oximeter to a master device, the master device must initiate the connection by first pairing with the oximeter. The oximeter has a six-digit identification number printed on the battery door. To complete the pairing process, the six-digit number must be provided to the master as the Bluetooth PassKey (Bluetooth PIN). Once the Bluetooth connection is established, the oximeter receives and transmits data using the Serial Port Profile (SPP) with the setting in Figure 12.

Bits per second	Data bits	Parity	Stop bits	Flow Control
9600	8	None	1	None

Figure 12. Settings used by the oximeter for sending and receiving [41].

The oximeter features four data format solutions. They are [41]:

- Data format 13 – provides easy spot-check measurements with the storage and forwarding of measurements.
- Data format 8 – provides real-time oximetry measurements every second.
- Data format 2 – provides real-time oximetry measurements with compressed waveform (8 bit waveform) every 1/75th of a second.
- Data format 7 – provides real-time oximetry measurements with full resolution waveform (16 bit waveform) every 1/75th of a second.

Since we are interested in the oxygen saturation every one second, we have selected the "data format 8" option. Figure 13 shows the data packet for data format 8.

Byte 1 - Status							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
1	R	OOT	LPRF	MPRF	ARTF	HR8	HR7
*__Note__: Bit 7 is always set							
Byte 2 - Heart Rate							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	HR6	HR5	HR4	HR3	HR2	HR1	HR0
*__Note__: Bit 7 is always clear							
Byte 3 - SpO2							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	SP6	SP5	SP4	SP3	SP2	SP1	SP0
*__Note:__ Bit 7 is always clear							
Byte 4 Status2							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	R	SPA	R	SNSF	R	R	LOW
*__Note__: Bit 7 is always clear							

Figure 13. Oximeter data packet format [41].

Parameters	Indicated artifact condition on each pulse
Out Of Track (OOT)	An absence of consecutive good pulse signals. Indicates sustained period of artifact.
Low Perfusion (LPRF)	Amplitude representation of low/no signal quality (holds for entire duration).
Marginal Perfusion (MPRF)	Amplitude representation of low/marginal signal quality (holds for entire duration).
Sensor Alarm (SNSA)	Device is providing unusable data for analysis (set when the finger is removed).
SmartPoint Algorithm (SPA)	High quality SmartPoint measurement.
Low Battery Condition (Low BAT)	Low Batteries. Replace batteries as soon as possible.
Heart Rate (HR8-HR0)	4-beat Pulse Rate average formatted for display.
SpO2(SP6-SP0)	4-beat SpO2 average formatted for display.
Reserved (R) (range:0 or 1)	Reserved for future use.

Table 3. Meaning of the oximeter data packet abbreviations [41].

Table 3 provides an explanation for each of the parameters mentioned in Figure 14.

The following are the steps used to connect to the oximeter:

- Scan for Bluetooth devices.
- Query the local Bluetooth adapter for paired Bluetooth devices.
- Establish RFCOMM channel.
- Transfer data to and from the oximeter

After establishing the connection with the oximeter and while the readings are being received, values are saved in an array list to be processed in the diagnostic activity. Furthermore, the readings are saved in a text file in case the application is closed and analysis is needed in the future.

5.2.1.2 AudioRecorder Class

The AudioRecorder is responsible for recording the respiratory efforts. Respiratory efforts are extracted from the patient by placing a microphone on the patient's throat. In the Android development environment, audio recording could be implemented in two different ways: using the MediaRecorder class or AudioRecord class. Using the MediaRecorder class is very easy but offers less flexibility, while AudioRecord class offers more flexibility but is more complex. MediaRecorder class is used for quick and simple audio recording. It saves the raw data in two different formats: MPEG4 or 3GPP compressed format. On the other hand, the AudioRecord class saves raw data in an uncompressed format; this allows the user to process the audio data, write to a file, and display waveform. Since we need to process and analyze the audio file, we have used the AudioRecorder class. The following are the steps used to record audio using AudioRecorder:

1. Create an instance of AudioRecord.
2. Start recording using the startRecording() method.
3. Read uncompressed data using the AudioRecord.read() method.
4. Stop recording using the stop() method.

The following parameters are used to record an audio file:

- Setting the audio source as Microphone.
- Sample Rate is 44100 Hz.
- Audio encoder format is ENCODING_PCM_16BIT – 16 bit per sample.
- Channel value is CHANNEL_IN_MONO. It is guaranteed to work on all devices.

The raw data is saved in an external memory into chunks of one-hour files. Saving the respiratory efforts for the whole sleeping period in one file has many disadvantages as observed during the implementation phase:

- If an error occurs during the recording process, the whole file could be corrupted and may not be analyzed.

- The sleeping period usually lasts (6 - 8) hours. Recording during this period uses almost (3 - 4) Giga-Bytes of memory. Analyzing such a file exhausts the internal memory and results in low memory error.

For the aforementioned disadvantages, we have saved each one hour in a separate file. Separating the recording file into chunks has the following impacts:

- This separation does not impact the processing time, since each file will be processed separately and at the end, the total processing time will be equal to the processing time for the file before separation.
- Choosing one hour is based on our experiments, as it was found that increasing the recording time to three hours would result in a low memory error. Therefore, the maximum recording time in each file is two hours maximum. The reason for choosing one hour instead of two hours is to minimize the amount of loss in case the file is corrupted, that is, losing one hour of data is better than losing two hours of data.
- Breaking the recording file into chunks may have an impact on the number of apnea events. An apnea event may start at the end of the recording file and end at the beginning of the next file. If each file is analyzed separately, then this apnea event will not be considered in the calculation. For this reason, if an apnea event is started at the end of a file, we check if it continues in the next file and satisfies the conditions of an apnea event (i.e., that it lasts at least 10 seconds).

5.2.1.3 AccelerometerRecorder Class

Smartphones have built-in sensors, and one of these sensors is the accelerometer, which we will use to record the patient's movements. The Android SDK provides a library to control the built-in sensors, which is called "SensorManager." Since we are interested in the accelerometer, we have selected "Type_Accelerometer." The SensorManager library includes a built-in function that is called up whenever there is a change in the sensor readings "onSensorChanged()." Values of the accelerometer sensor are saved in an array list to be used later in the Diagnose activity for analysis.

Furthermore, the readings are saved in a text file in case the application is closed and analysis is needed in the future.

5.2.2 Pretest Activity:

Pretest activity is simply a questionnaire that assesses the possibility of an individual having the disease. The patient can select the symptoms from a checklist as shown in Figure 14. The answers are then forwarded to the Diagnose activity to be used in the analysis process. The patients that are required to do the test in the hospital are those who are obese, complaining from excessive daytime sleepiness, and have been observed to have loud and irregular snoring by their relatives. Reporting these three main conditions increases the possibility of having the disease and sets the pretest value to true. This value is then passed to the Diagnose activity to be used in the diagnosing process as shown in Figures 6 and 7. The pretest has a significant role in reducing the number of false-positive diagnoses as discussed in Section 3.1. Figure 14 shows the Pretest activity.

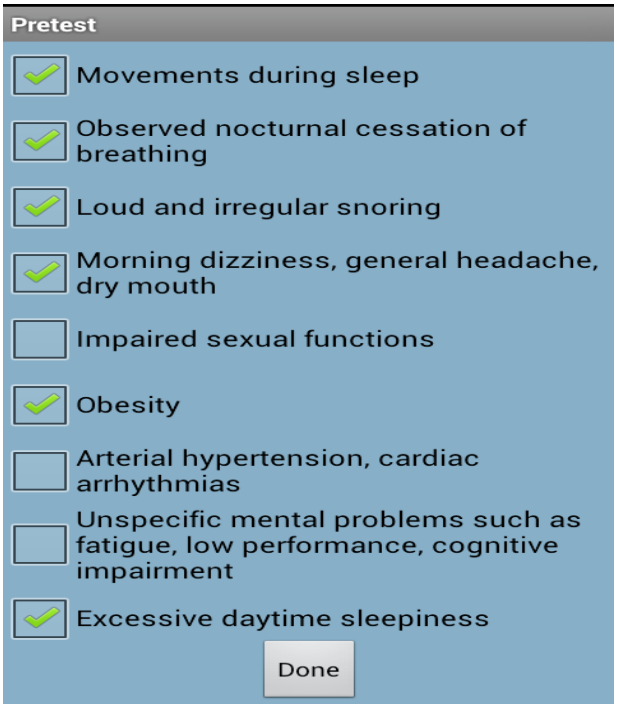

Figure 14. Pretest Activity.

5.2.3 Diagnose Activity

The Diagnose activity is responsible for analyzing the readings collected in the DataCollector activity, and the information passed by the Pretest activity. In this activity, we have implemented the algorithms mentioned in Sections 3.3.2 and 3.3.3. Since we have three main readings from the patient, we have divided this activity into three parts: Analyzing Oxygen Saturation, Analyzing Respiratory Efforts, and Analyzing Accelerometer Readings. Figure 15 shows these three parts.

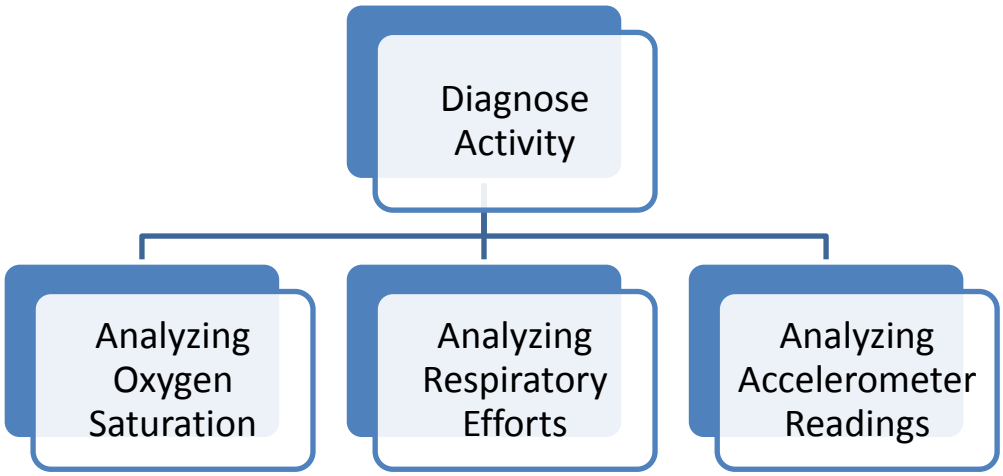

Figure 15. Main components of the Diagnose activity.

5.2.3.1 Analyzing Oxygen Saturation and Accelerometer Readings

Analysis of the oxygen saturation and accelerometer readings is based on the algorithm described in Section 3.3.2. Oxygen saturation and body movement readings are saved in array lists while the DataCollector activity is running. During the analyzing of oxygen saturation and body movements, the following tasks are performed:

- Reading of the oxygen saturation from the array list; a value is recorded every one second.
- Calculation of the oxygen saturation average in the first two minutes.
- Setting the threshold value to 0.98 of the average.
  $X = 0.98 * N$, where X is the Threshold and N is the Average.

- Comparing the readings with the threshold value.
- Reading accelerometer values from the array list. One value is recorded every one second.
- Any oxygen saturation less than the threshold value that coincides with movement is ignored.
- An event is reported if the values are less than the threshold and last at least ten seconds.
- Recalculation of the Average and the Threshold during the sleeping period.
- Calculation of the apnea/hypopnea index, which is the number of events divided by the total period.

5.2.3.2 Analyzing Respiratory Efforts

Analyzing respiratory efforts is based on the algorithm described in Section 3.3.2. The algorithm requires filtering the signal and calculating the energy every second. The Android SDK does not contain a specific library for signal processing functions. Therefore, we have implemented the filtering functions in Android, in order to use them in extracting the required frequencies. Since the frequencies lie between 200 and 800 Hz, we have implemented a band-pass Infinite Impulse Response (IIR) filter. In the following subsections 5.2.3.2.1 and 5.2.3.2.2, we will explain the concept of the IIR filter and the functions used to implement its functionality within the Android environment.

5.2.3.2.1 Infinite Impulse Response (IIR) Filter

The IIR digital filter is also known as a recursive digital filter, because it contains feedback loops. Its output at every time step depends on previous outputs. If X[n] represents the n^{th} input to the filter and Y[n] is the n^{th} output of the filter, then a general IIR filter is implemented as follows:

$$Y[n] = c0 * X[n] + c1 * X[n-1] + \ldots + cM * X[n-M] - (d1 * Y[n-1] + d2 * Y[n-2] + \ldots + dN * Y[n-N]) \quad (1)$$

As seen in (1), the n^{th} output is a linear function of the n^{th} input, the previous *M* inputs, and the previous *N* outputs. The c_k and d_k coefficients are calculated to give the filter a specific frequency response. The number of coefficients *M* and *N* vary based on the type of filter. There are many different types of IIR filters and many different methods to calculate the coefficients. Since we are interested in the Butterworth low-pass filter, we will list the method to calculate the coefficients below [50].

5.2.3.2.2 Butterworth Filter

A Butterworth filter is also known as a maximally flat filter because its frequency response is characterized by no ripple in the pass band and stop band.

5.2.4 Report Activity

The Report activity is the last page displayed to the user, in which all the related details are summarized. This page contains the following information:

- Result: whether or not the user has obstructive sleep apnea (OSA).
- Recording Time: sleeping period measured in hours.
- Lying Down Period: the percentage in which the patient was lying down during the sleeping period.

- Movement Period: the percentage in which the patient was moving during the sleeping period.
- Disease Severity: The category of the disease if the test was positive. Possible values are: severe, moderate, and mild.

Figure 17 shows a sample of the Report activity and how the values are displayed.

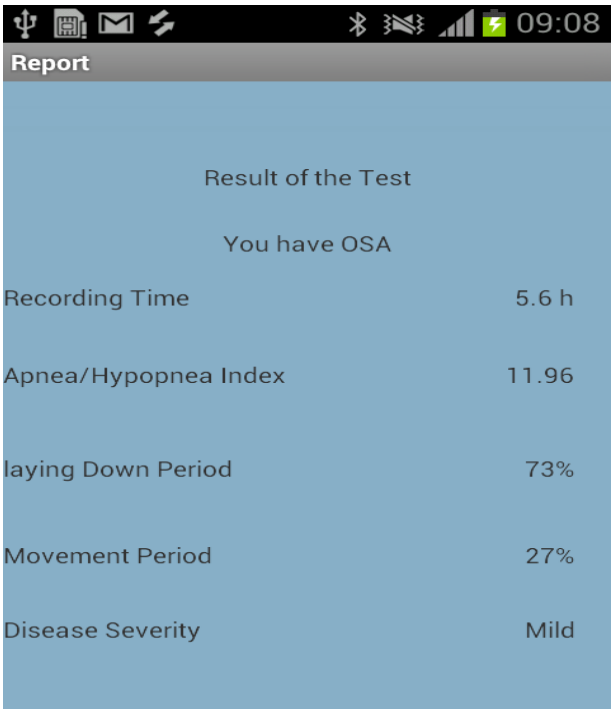

Figure 17. Report Activity.

5.3 Problems Faced During Implementation

Since the application must run for a long time, during which we use the I/O ports and memory excessively, we must take the smartphone's performance under consideration. Choosing the right algorithm and data structure should take high priority in order to improve performance and utilize the available resources. During the implementation phase, we faced some problems due to processing and memory limitations. For the benefit of future implementations, we have listed tips below to ease the implementation process.

5.3.1 Problem 1 – Running the Application for Long Periods

Running the application for a long time may impact the performance of the application and it may become unresponsive. In the developed application, the smartphone collects the physiological readings from the patient for long periods (6-8 hours) without user interaction. By the end of this period, the application will most likely become unresponsive when the user tries to interact with the Graphical User Interface (GUI), such as by pressing a button. This may happen when some long operation takes place in the "main" thread, which prevents the application to process any GUI events in the application.

In the Android operating system, the system warns users when applications are unresponsive for a period of time by displaying a dialog that says "<your app> isn't responding," such as the dialog in Figure 18. At this point, the system offers the user two options: either to quit the application or to wait [52].

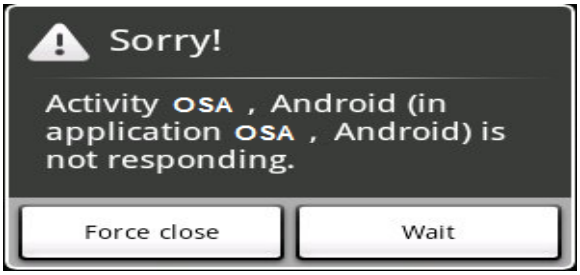

Figure 18. An ANR dialog displayed to the user when the application becomes unresponsive.

In general, Android OS displays an "Application Not Responding" ANR dialog if an application cannot respond to user input in a period of 10 seconds. This may happen if some of the Input/Output operations are blocked in the main thread, so the system will not be able to process incoming user inputs. Android OS displays the ANR dialog for a particular application when one of the following conditions is met [52]:

- The application did not respond to an input event (such as key press or screen touch events) within 5 seconds.
- A BroadcastReceiver hasn't finished executing within 10 seconds.

In order to avoid having unresponsive applications , users must do one of the following:

1. Implement operations that take a long time as a service instead of as an activity. A "service" is an application component that is used to perform a longer-running operation without interaction with the user or to provide functionality for other applications. There are two types of services, Foreground and Background services. The background service may be killed by the system if it is under heavy memory pressure. On the other hand, foreground services will almost never be killed by the system in case of low memory [53].

2. Do not perform the work in the "User Interface" (UI) thread, but instead create a Worker thread and do most of the work there. Implementing the work in the Worker thread keeps the UI thread running and prevents the system from concluding that the application has frozen. Android applications in general run entirely on a single thread by default (the "UI thread" or "main thread"). This means that any operation performed in the UI thread that takes a long time to complete can trigger the ANR dialog. For that reason, any operation that runs in the main thread should do as little work as possible on that thread. Shifting the job to the worker thread allows the application to handle the applications that take a long time, and then send the result back to the UI thread. To handle complex interactions between the UI and Worker thread, a Handler is considered in the Worker thread to process messages delivered from the UI thread. Extending the AsyncTask class is the best way for such situations, which simplifies the execution of Worker thread tasks that require interacting with the UI. The AsyncTask allows the application to perform asynchronous work in the UI. It performs the blocking operations in a worker thread and then publishes the results on the UI thread, without requiring thread handling [54].

3. IntentService is a base class for Services that handle asynchronous requests (expressed as Intents) off the main thread by way of Intent requests. Each intent is added to the IntentService's queue and handled consecutively. Instead of launching an AsyncTask and managing it each time there is additional processing,

one can just implement his/her own service, create an intent, package it up with the appropriate data needed for processing, and start the service. The result of the processing can be sent back to the application by broadcasting an Intent object and using a broadcast receiver to grab the result and use it within the application [55].

In the developed application, we have implemented a foreground service that is called by the main activity to collect the physiological signals. On the other hand, we first used the AsyncTask in the Diagnose class to process the wave files, but it required a long processing time. For this reason, we switched to the IntentService which decreased the processing time to one fourth that of the AsyncTask. This allows the application to run for a long time with a decrease in the processing overhead. Moreover, if one of the operations takes a long time, it will not affect the response of the application and it will not hang.

5.3.2 Problem 2- Processing Large Files

Recording the respiratory efforts during the sleeping period (6-8 hours) results in a file with an average size of 4 GB. To analyze the file, we read it in a vector format in Matlab, and then perform the required analysis (filtering and energy calculations). Running the Matlab code results in a memory error due to the large size of the recorded file: "Out of memory. Type HELP MEMORY for your options." In order to overcome the memory problem, we took the following points under consideration:

- Increase the RAM size.
- Increase the virtual memory dedicated to the Matlab program.
- If running Windows, try to run the code on a 64-bit version.
- If none of the above recommendations help in solving the memory problem, as we faced in the implementation process, try to split the recorded file into segments of one or two hours. When you analyze the recorded files, read and analyze each file separately. Splitting the recorded file can be done during the

recording process or after recording. In the developed application, we have used the former way to overcome this problem.

5.3.3 Problem 3-Bluetooth-Related Issues

Android 4.0 (API level 14) introduces support for the Bluetooth Health Device Profile (HDP). This profile allows developers to create applications that use Bluetooth to communicate with health devices that support Bluetooth, such as oximeters, thermometers, heart-rate monitors, blood meters, and scales.

Using the HDP profile requires implementing four main components, which are:

- **A source:** the health device that is used to collect and transmit medical data (in our case, the oximeter) to the smartphone.
- **A sink:** the smartphone that receives the medical data.
- **Registration:** refers to registering a sink (smartphone) for a particular health device (oximeter).
- **Connection:** opening a channel between the health device (oximeter) and the smartphone.

We implemented the concepts of the HDP profile and ran the sample code provided by the Android SDK Manager. Nevertheless, we have faced many problems when testing it. While debugging the application, it was found that the health device was registered with the smartphone, and then a channel was established between them. After that, the channel started transmitting the readings from the health device to the smartphone. Suddenly, and after 10 seconds, the channel stopped responding and did not transmit the readings to the smartphone. Researching about the problem revealed that many developers faced the same problem, and the issue was reported to the Android development team.

In order to overcome this problem, we used the basic concept of Bluetooth in Android, which is based essentially on four major tasks: setting up Bluetooth, finding devices that are available in the local area, connecting devices, and transferring data between devices. More details about these steps can be found in [56].

5.4 Performance Tips

Besides the problems faced during the implementation phase and the solution proposed in Section 5.3, we have devised some performance tips to enhance the performance of the application. The following are some of these tips:

5.4.1 Avoid Creating Unnecessary Objects

Allocating memory is a very important design consideration in this application's performance. It is recommended to avoid creating objects in the application, since object creation is never free. Allocating more objects in the application enforces the process of periodic garbage collection. Fewer objects create mean less-frequent garbage collection, which has a direct impact on user experience [57].

5.4.2 Static Over Virtual

In the developed application, we defined the variables and methods as static. This allows us to access them directly without the need for creating objects. Invocations of static variables and methods are about 15-20% faster when compared to virtual variables [57].

Chapter 6: Presenting and Discussing the Results

In this chapter, we will present and discuss the results obtained from testing the application using 17 samples (10 patients and 7 non-patients). Initially, we will list the results of all test samples obtained from running the test based on partial data set and then on complete data set (as described in Sections 3.4.1 and 3.4.2, respectively). Next, we analyze the respiratory efforts, oxygen saturation, and body movements for one of the patients. Third, the results will be compared against the golden standard results. Fourth, we will compare the results of analyzing the respiratory efforts obtained from the external server and smartphone. Finally, we will calculate some of the clinical test measurements including: Sensitivity, Specificity, Positive Predictive Value, and Negative Predictive Value, which provide indications about the reliability and accuracy of the proposed system.

6.1 Patient Results

We performed the test on 17 subjects, 8 of whom had already been diagnosed with OSA, 2 who had the symptoms, but were not diagnosed, and 7 who were healthy with no symptoms. The results are based on partial and complete data sets as described in Sections 3.4.1 and 3.4.2, respectively. The first approach involves the oxygen saturation and body movements to diagnose obstructive sleep apnea (OSA). However, the second approach involves all the physiological readings (respiratory efforts, oxygen saturation, and body movements) to diagnose OSA.

6.1.1 Results Based on Partial Data Set

The results displayed in this subsection are based only on the oxygen saturation and body movements as described in Section 3.4.1. The results of the tests are summarized in Tables 4 and 5. These tables show values for the following parameters:

- Apnea/hypopnea index (AHI), which is defined as the average number of apnea events per hour.

- Lowest oxygen saturation each patient experienced during the sleeping period.
- Longest obstructive sleep apnea event reported during the sleeping period.
- Recording period.
- Severity of the disease, whether it is mild, moderate, or severe.
- Correctly Classified, which indicates whether the patient has been diagnosed and classified in the correct severity or not, by comparing the results with the golden standard. More details about the comparison between the proposed system and the golden standard are summarized in Table 9.

Name	AHI	Lowest Oxygen Saturation (%)	Longest Event (sec)	Recording Period(hours)	Severity	Correctly Classified
Patient 1	8.1	90	12	5.7	Mild	True
Patient 2	11	89	15	7.3	Mild	True
Patient 3	20.57	83	70	6.37	Moderate	True
Patient 4	15.45	85	15	4.4	Moderate	True
Patient 5	21.2	86	61	4.19	Moderate	True
Patient 6	65.6	52	115	6.1	Severe	True
Patient 7	41.3	76	80	5.2	Severe	True
Patient 8	91.8	77	69	4.9	Severe	True
Patient 9	5.1	89	15	4.5	Mild	-
Patient 10	10	86	17	5.1	Mild	-

Table 4. Results of the obstructive sleep apnea (OSA) patients (partial data set).

As seen from Table 4, all patients were correctly diagnosed and classified. Four of them were diagnosed as having mild OSA, three as having moderate OSA, and three as having severe OSA. The last two patients (patient 9 and 10) suffered from the symptoms, but didn’t take the test at the hospital.

Name	AHI	Lowest oxygen saturation (%)	Longest Event (sec)	Recording time(hours)	Correctly Classified
Person 1	0	95	-	6.4	True
Person 2	0	95	-	4.5	True
Person 3	1.3	94	11	4.3	True
Person 4	2.1	93	11	4.8	True
Person 5	10.1	85	15	8	False
Person 6	1.6	94	11	6.3	True
Person 7	0.75	95	10	4	True

Table 5: Results of the non-obstructive sleep apnea (OSA) patients (partial data set).

As shown in Table 5, seven normal persons (non-patients) were tested. Six of them were classified correctly, while one of them was classified as having the disease, though he did not (i.e., a false positive).

6.1.2 Results Based on Complete Data Set

The results displayed in this subsection are based on all the physiological signals (oxygen saturation, body movements, and respiratory efforts) as described in Section 3.4.2. The results of the tests are summarized in Tables 6 and 7.

Name	AHI	Lowest Oxygen Saturation (%)	Longest Event (sec)	Recording Period (hours)	Severity	Correctly Classified
Patient 1	8	90	12	5.7	Mild	True
Patient 2	10.8	89	15	7.3	Mild	True
Patient 3	19.8	83	70	6.37	Moderate	True
Patient 4	15	85	15	4.4	Moderate	True
Patient 5	19.6	86	61	4.19	Moderate	True
Patient 6	62.8	52	115	6.1	Severe	True
Patient 7	39.7	76	80	5.2	Severe	True
Patient 8	81.1	77	69	4.9	Severe	True
Patient 9	5	89	15	4.5	Mild	-
Patient 10	9.8	86	17	5.1	Mild	-

Table 6. Results of the obstructive sleep apnea (OSA) patients (complete data set).

As we can see in Table 6, all patients were correctly classified. Four of them were diagnosed as having mild OSA, three as having moderate OSA, and three as having severe OSA.

Name	AHI	Lowest Oxygen Saturation (%)	Longest Event (sec)	Recording Time(hours)	Correctly Classified
Patient 1	0	95	-	6.4	True
Patient 2	0	95	-	4.5	True
Patient 3	1.1	94	11	4.3	True
Patient 4	1.8	94	11	4.8	True
Patient 5	9.4	85	15	8	False
Patient 6	1.5	94	11	6.3	True
Patient 7	0.4	95	10	4	True

Table 7. Results of the non-obstructive sleep apnea (OSA) patients (complete data set).

As shown in Table 7, seven normal persons (non-patients) applied the test. Six of them were classified correctly. One of them was classified as having the disease, but later it was found that he did not (wrongly classified). Figure 22 shows a graphical view of the AHI value for each of the five non-patients.

6.1.3 Comparison between the Partial and Complete Data Set Diagnostics

When we compare the AHI results in Tables 4-5 and Tables 6-7, we can see that the results are very close. The partial diagnostic resulted in higher AHI values compared with the complete diagnostic. We believe that the reason is due to the number of physiological factors included in the test. Having more factors to decide the occurrence of an apnea event can minimize the AHI value, since all readings (respiratory efforts, oxygen saturation, and body movements) must report the apnea event (cessation of breath). For example, if respiratory efforts reported an apnea event, but there was no associated oxygen desaturation, the apnea event is not counted. As a result, the AHI value will be smaller. Tables 8 and 9 show a comparison between the partial and complete data set diagnostics, and these comparisons are illustrated in Figures 19 and 20.

Name	Partial Data Set		Complete Data Set	
	AHI	Severity	Severity	AHI
Patient 1	8.1	Mild	Mild	8
Patient 2	11	Mild	Mild	10.8
Patient 3	20.57	Moderate	Moderate	19.8
Patient 4	15.45	Moderate	Moderate	15
Patient 5	21.2	Moderate	Moderate	19.6
Patient 6	65.6	Severe	Severe	62.8
Patient 7	41.3	Severe	Severe	39.7
Patient 8	91.8	Severe	Severe	81.1
Patient 9	5.1	Mild	Mild	5
Patient 10	10	Mild	Mild	9.8

Table 8. Comparison between the partial and complete data set diagnostic (patients).

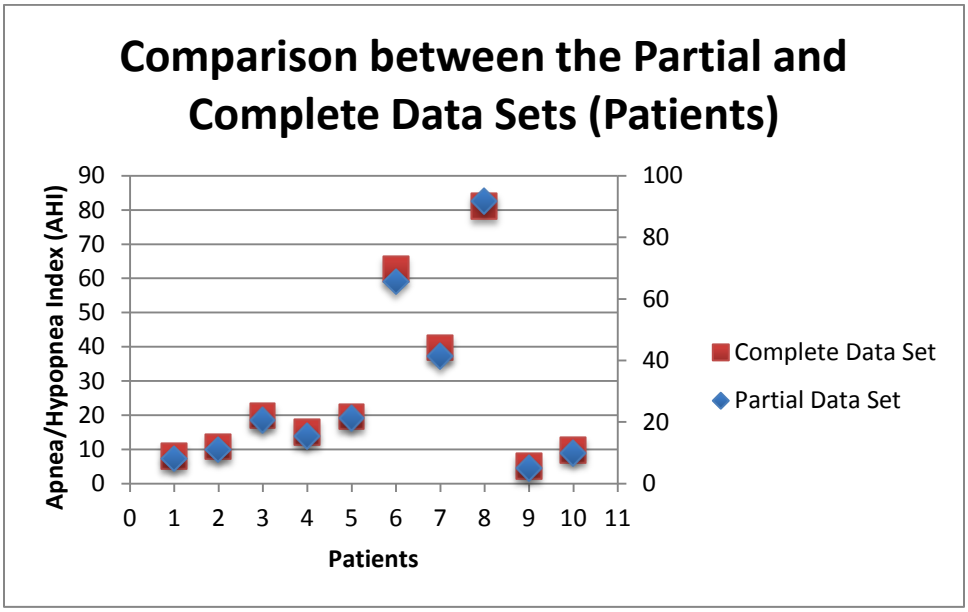

Figure 19. Comparison between the partial and complete data set diagnostic (patients).

Name	Partial Data Set	Complete Data Set
	AHI	AHI
Patient 1	0	0
Patient 2	0	0
Patient 3	1.3	1.1
Patient 4	2.1	1.8
Patient 5	10.1	9.4
Patient 6	1.6	1.5
Patient 7	0.75	0.4

Table 9. Comparison between the partial and complete data set diagnostics (non-patients).

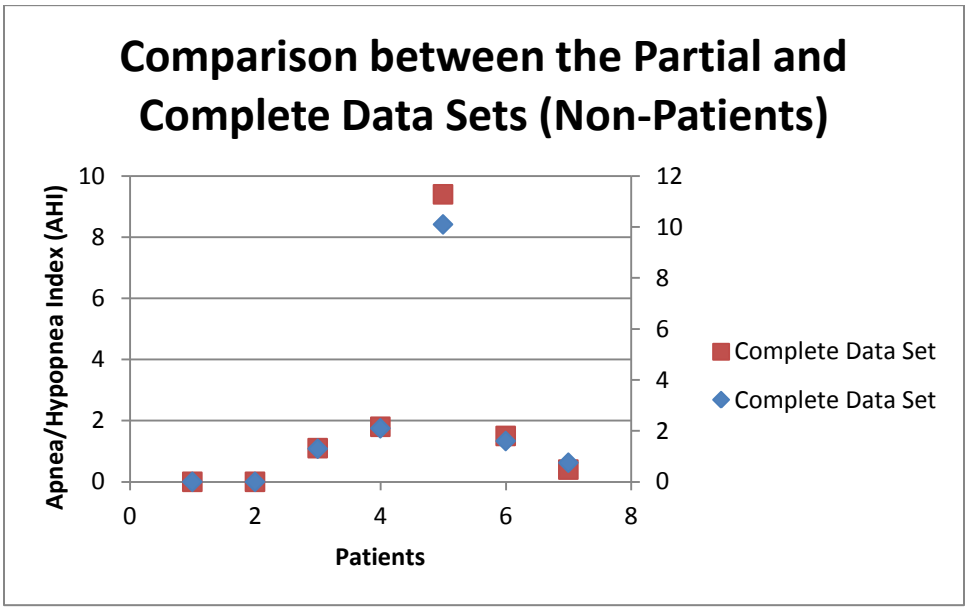

Figure 20. Comparison between the partial and complete data set diagnostics (non-patients).

6.2 Analysis of the Respiratory Efforts and Oxygen Saturation

In this section, we will use data from a patient to present the occurrence of an apnea event. Furthermore, we will show one of the apnea events and discuss the readings during this event.

6.2.1 Respiratory Efforts Signal

Figure 21 shows the respiratory efforts signal for one of the subjects in our work during a one-hour time window. The Y-axis represents the energy of the signal in each second, and the X-axis represents the time in seconds. The spikes in the graph represent the snoring periods, and the dotted black line represents the average threshold value that is calculated during this hour.

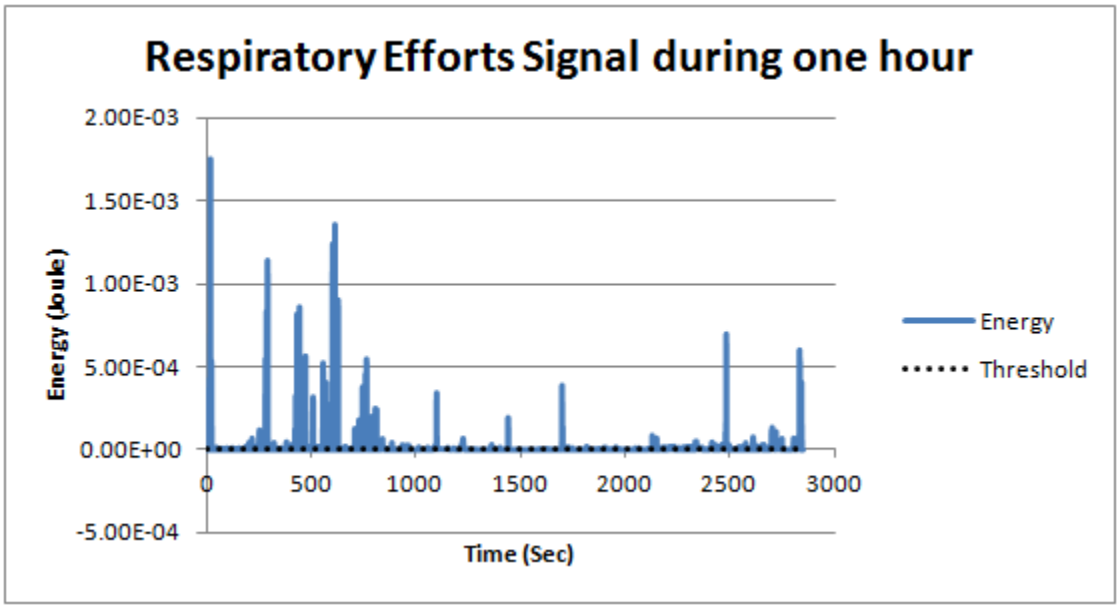

Figure 21. Respiratory effort for one of the patients during one hour.

Figure 22 shows the energy of the respiratory effort signal for one of the subjects in our work during one minute time window. The spikes in the graph represent the snoring periods. As we can see, the snoring period is followed by a decrease in the oxygen level, and the energy values are less than the threshold value which is represented by the red line.

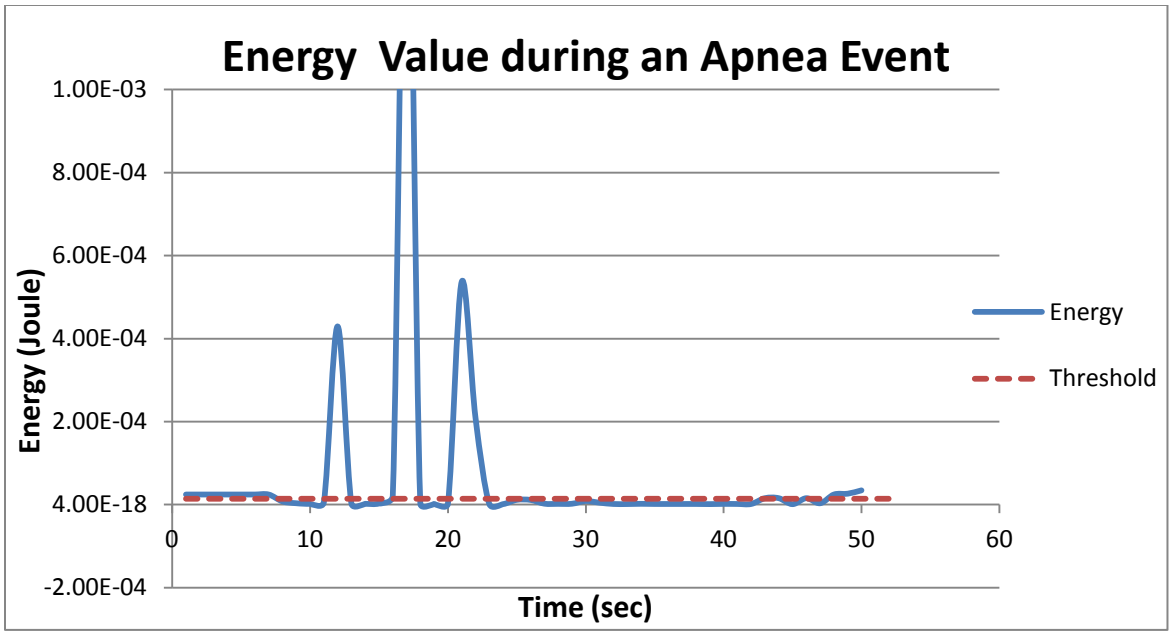

Figure 22. Energy value during an apnea event.

Figure 23 shows the oxygen level in the blood for one of our tested samples, which is extracted from the oximeter every second. The readings correspond to the respiratory efforts displayed in Figure 26. When the patient started to snore, a decrease in the oxygen saturation followed. After a few seconds, the oxygen saturation reached a level less than the threshold value that lasted for more than 10 seconds. Having the energy of the respiratory efforts and the oxygen saturation less than the threshold values is reported as an apnea event.

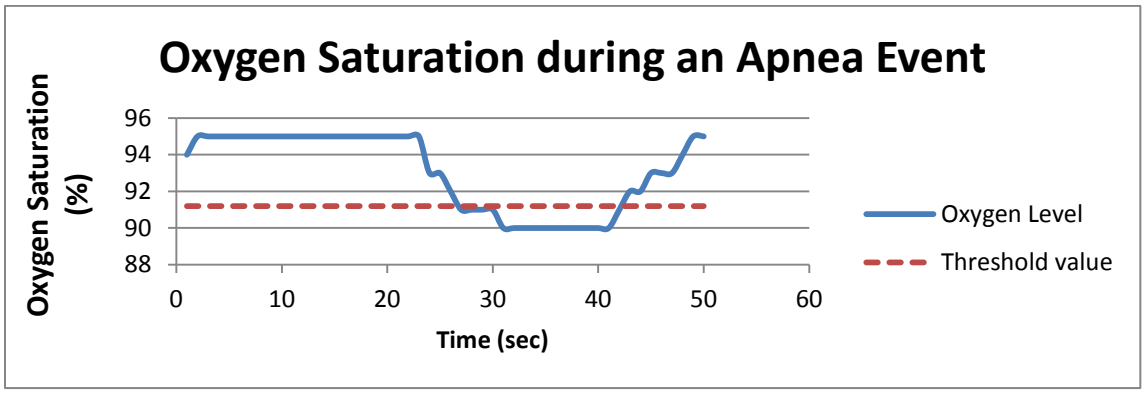

Figure 23. Oxygen saturation during an apnea event.

6.3 Comparison between the Proposed System and the Golden Standard

Comparing the results obtained from the proposed system against the golden standard is very important, since it gives us indication about the accuracy and the reliability of the proposed system. Table 10 lists the AHI values and the severities of the patients obtained from the proposed system and the golden standard.

Name	Proposed System (Complete Data Set)		Golden Standard		Error ^{2}
	AHI	Severity	Severity	AHI	
Patient.1	8	Mild	Mild	5.2	7.8
Patient.2	10.8	Mild	Mild	14	10.2
Patient.3	19.8	Moderate	Severe	30.2	108.2
Patient.4	15	Moderate	Moderate	17	7.8
Patient.5	19.6	Moderate	Moderate	29	88.4
Patient.6	62.8	Severe	Severe	80.7	320.4
Patient.7	39.7	Severe	Severe	60.6	436.8
Patient.8	81.1	Severe	Severe	89.9	77.4

Table 10. Comparison between the proposed system and the golden standard.

As shown in Table 10, the AHI values obtained from the proposed system are close to the values obtained using the golden standard. The severities of the patients were correctly classified, except for the third patient. The proposed system showed that the patient had moderate OSA, while the golden standard showed that the patient had severe OSA. Column 6 shows the square of the error, which is defined as the difference between the squares of the actual and expected values. The sum of these errors is used in calculating the Root Mean Square Error (RMSE) as seen in Equations (2) and (3):

$$RMSE = \sqrt{MSE} \tag{2}$$

$$MSE = \frac{1}{m} \times \sum_{i=1}^{m} \mathrm{E}i \tag{3}$$

The RMSE is found to be 32.5. The smaller the RMSE, the more accurate the system is considered to be. A value of 32.5 is considered high, but in the medical field, we care about having a correct diagnosis more than the difference between the proposed

system and the golden standard. Figure 24 shows a graphical comparison of the AHI values between the proposed system and the golden standard.

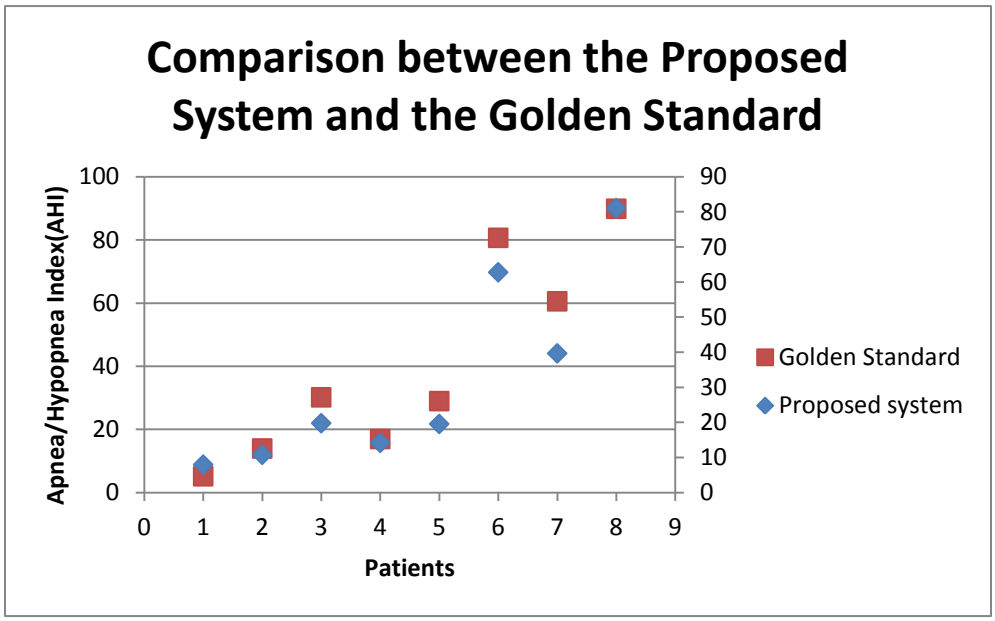

Figure 24. Comparison between the proposed system and the golden standard.

6.4 Impact of Each Physiological Signal on the Final Diagnostic

In this section, we will assess the accuracy of the classification using partial data sets. The oxygen saturation and body movements are considered one set (partial data set 1), and respiratory effort represents the other set (partial data set 2). While the complete data set represents the case when all physiological signals are used. Table 11 lists the results obtained from each set besides the results of the complete data set and the golden standard.

Name	Partial Data Set 1 (Oxygen + Body Movements)		Partial Data Set 2 (Respiratory Effort)		Complete Data Set (Oxygen + Body Movement + Respiratory Effort)		Golden Standard	
	AHI	Severity	AHI	Severity	Severity	AHI	Severity	AHI
Patient 1	8.1	Mild	7.9	Mild	Mild	8	Mild	5.2
Patient 2	11	Mild	10.5	Mild	Mild	10.8	Mild	14
Patient 3	20.57	Moderate	19	Moderate	Moderate	19.8	Severe	30.2
Patient 4	15.45	Moderate	14.5	Mild	Moderate	15	Moderate	17
Patient 5	21.2	Moderate	17.9	Moderate	Moderate	19.6	Moderate	29
Patient 6	65.6	Severe	60	Severe	Severe	62.8	Severe	80.7
Patient 7	41.3	Severe	38.1	Severe	Severe	39.7	Severe	60.6
Patient 8	91.8	Severe	70.4	Severe	Severe	81.1	Severe	89.9

Table 11. Comparison between the physiological signals, complete data set, and golden standard.

As seen in Table 11, both data sets correctly diagnosed the patients as having obstructive sleep apnea (OSA). However, the severity of patient 3 was incorrectly identified in both data sets. Furthermore, the severity of patient 4 was incorrectly identified in partial data set 2. The values of the partial data set 1 are closer to the golden standard values than the values obtained from the partial data set 2. Partial data set 1 consists of two physiological signals that determine the final diagnostic, while partial data set consists of only one physiological signal. Therefore, partial data set 1 performs better than partial data set 2 in diagnosing OSA. Moreover, the accuracy of the simple microphone used by the smartphone might have impact on the accuracy of detecting apnea events in partial data set 2. Table 12 shows the error[2] and the root mean square error (RMSE) for all data sets.

Name	Partial Data Set 1	Partial Data Set 2	Complete Data Set	Golden Standard	Error^2		
	AHI	AHI	AHI	AHI	Partial Data Set 1	Partial Data Set 2	Complete Data Set
Patient 1	8.1	7.9	8	5.2	8.41	7.29	7.8
Patient 2	11	10.5	10.8	14	9	12.25	10.2
Patient 3	20.57	19	19.8	30.2	92.7	125.44	108.2
Patient 4	15.45	14.5	15	17	2.4	6.25	7.8
Patient 5	21.2	17.9	19.6	29	60.84	123.2	88.4
Patient 6	65.6	60	62.8	80.7	228	428.49	320.4
Patient 7	41.3	38.1	39.7	60.6	372.49	506.25	436.8
Patient 8	91.8	70.4	81.1	89.9	3.61	380.25	77.4
Total					777.45	1589.42	1057
RMSE					27.9	39.9	32.5

Table 12. Root mean square error (RMSE) for all data sets.

6.5 Accuracy Factors

When evaluating the accuracy of a clinical test, the terms sensitivity, specificity, positive predictive value (PPV), and negative predictive value (NPV) are used. Sensitivity and specificity do not depend on the disease prevalence in the population of interest. On the other hand, PPV and NPV are dependent on the pervasiveness of the disease in the population of interest.

The following terms are fundamental the understanding of factors that affect the accuracy of the clinical tests [57-59]:

- True positive: Sick people correctly diagnosed as sick.
- False positive: Healthy people incorrectly diagnosed as sick.
- True negative: Healthy people correctly diagnosed as healthy.
- False negative: Sick people incorrectly diagnosed as healthy.

6.5.1 Sensitivity (True Positive Rate)

"Sensitivity" evaluates the percentage of actual positives which are correctly identified as such (the percentage of sick people who are correctly diagnosed as having the disease). A test with 100% sensitivity correctly diagnoses all patients with the disease. A test with 80% sensitivity identifies 80% of patients who have the disease (true positives), but 20% of patients who have the disease are undetected (false negatives). Sensitivity is defined as seen in Equations 4 and 5 [57-59]:

$$Sensitivity = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Negatives} \quad (4)$$

$$Sensitivity = Probability\ of\ a\ positive\ test, given\ that\ the\ patient\ is\ ill. \quad (5)$$

6.5.2 Specificity (True Negative Rate)

"Specificity" evaluates the percentage of negatives which are correctly identified as such (the percentage of healthy people who are correctly diagnosed as not having the disease). A test with 100% specificity can correctly diagnose all patients as not having the disease. A test with 80% specificity correctly identifies 80% of patients as not having disease. In other words, 80% test negative (true negatives), but 20% who don't have the disease incorrectly test positive (false positives). Specificity is defined as seen in Equations 6 and 7 [57-59].

$$Specificity = \frac{Number\ of\ True\ Negatives}{Number\ of\ True\ Negatives + Number\ of\ False\ Positives} \quad (6)$$

$$Specificit = Probability\ of\ a\ negative\ test, given\ that\ the\ patient\ is\ well. \quad (7)$$

In the proposed system, we measure the number of true positives, false positives, true negatives, and false negatives. After that we calculate the sensitivity and specificity as shown below:

- Number of true positives: 8 sick people correctly diagnosed as sick.
- Number of false positives: 1 healthy individual incorrectly diagnosed as sick.
- Number of true negatives: 6 healthy people correctly diagnosed as healthy.
- Number of false negatives: 0 sick people incorrectly diagnosed as healthy.

$$Sensitivity = \frac{8}{8+0} = 100\%$$

$$Specificity = \frac{6}{6+1} = 85.7\%$$

The values 100% and 85.7% for the sensitivity and specificity, respectively, mean that 100% of patients were correctly identified as having the disease, and 85.7% of patients were correctly identified as not having the disease. On the other hand, 14.3% of patients that did not have the disease were incorrectly identified.

Besides the sensitivity and specificity measured above, we calculated the accuracy of the severity classification, i.e., how accurately the system classifies the sick people into the correct severity (mild, moderate, and severe). Among the 8 patients that diagnosed as having the disease, 7 of them were classified correctly.

$$Accuracy\ of\ the\ severity\ classification = \frac{7}{8} = 87.5\%$$

6.5.3 Positive Predictive Value

The positive predictive value (PPV) of a test is the percentage of positive test results that are true positives. This percentage is dependent on the prevalence of the disease in the population of interest and is useful to clinicians since it answers the

question: “How likely is it that this patient has the disease given that the test result is positive?” [58]. The PPV is defined as shown in Equation (8).

$$PPV = \frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Positives} \quad (8)$$

6.5.4 Negative Predictive Value

The negative predictive value (NPV) of a test is the percentage of negative test results that are true negatives. This percentage is dependent on the prevalence of the disease in the population of interest and is useful to clinicians since it answers the question: “How likely is it that this patient does not have the disease given that the test result is negative?” [58]. The NPV is defined as shown in Equation (9).

$$NPV = \frac{Number\ of\ True\ Negatives}{Number\ of\ True\ Negatives + Number\ of\ False\ Negatives} \quad (9)$$

In the proposed system, we measured the positive predictive value and the negative predictive value for the tests done on the 13 persons. The results are reported below:

$$PPV = \frac{8}{8+1} = 88.9\%$$

$$NPV = \frac{6}{6+0} = 100\%$$

Figure 25 shows a summary of the accuracy factors explained in Sections 6.5.1 - 6.5.4.

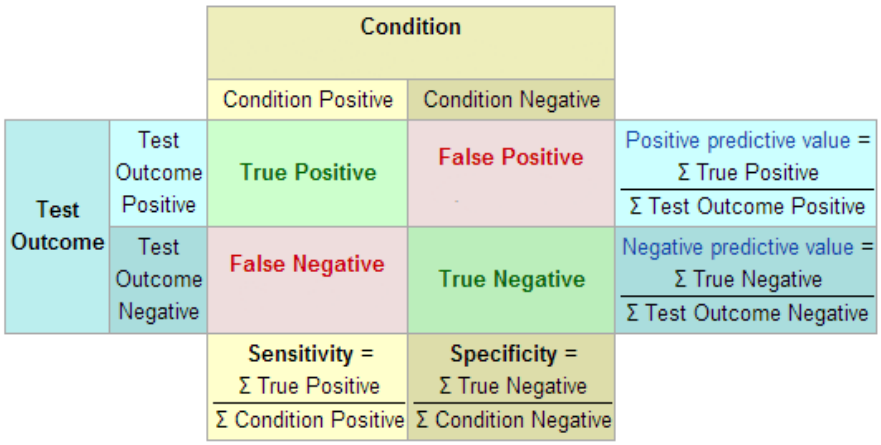

Figure 25. Accuracy factors calculations [59].

6.6 Comparison with Published Results

Several portable devices have been used to diagnose obstructive sleep apnea (OSA) as explained in Chapter 2. In this section, a comparison between results reported using these portable devices and the proposed system in this work is presented. Some authors calculated the sensitivity and specificity, while others just measured the accuracy of their developed systems. Table 13 displays the sensitivity, specificity, and accuracy of previous published work as well as results obtained using our system.

Previous Work	Number of Participants	Sensitivity	Specificity	Accuracy
Burgos et al [6]	70	96%	96.1%	93%
Oliver et al [23]	20	-	-	100%
Rofouei et al [27]	3	Only one subject was compared against the golden standard, and it was correctly diagnosed.		
Mack et al [28]	-	89.2%	94.6%	-
Tseng et al [29]	540	-	-	96.85%
Mont. et al [34]	52	88%	83.3%	84%
Yadollahi et al [37]	40	88.9%	92.3%	92.5%
Senny et al [39]	34	86.1%	87.4%	73.1%
Chung et al [40]	475	92.3%	83.3%	95%
Proposed system	15	100%	85.7%	94.1

Table 13. A comparison between obstructive sleep Apnea (OSA) systems.

Chapter 7: Conclusion

Obstructive sleep apnea (OSA) is a potentially serious sleep disorder, which is characterized by repetitive pauses in breathing during sleep. Several types of sleep apnea exist, but the most common type is obstructive sleep apnea. Polysomnography (PSG) is the golden standard in diagnosing OSA, but it requires an attended overnight test in the hospital, and imposes high cost and discomfort to the patients. In an effort to relieve patients from the requirements dictated by PSG, researchers have experimented with various home-bound and inexpensive techniques for detecting OSA. Due to the popularity of smartphones and their powerful computational capabilities, we propose an OSA detection technique that makes use of the built-in sensors in the smartphone. This detection platform extracts and analyzes physiological signals including oxygen saturation, body movements, and respiratory effort with the final output being a reliable screening and classification of OSA.

Following a comprehensive review of published work in this area, we proposed two approaches that build upon and enhance previous published work. To ascertain the diagnostic result, we compared our results with the golden standard and were in continuous consultation with a sleeping-disorder specialist as well.

The smartphone application developed performed as expected in recording and extracting the physiological signals from the patients and analyzing the collected data solely on the smartphone without the need for external resources. Hence, a self-contained, reliable and portable solution.

Upon examining our system's ability to screen the disease as compared to polysomnography, we tested the application on 17 subjects. The results showed that 100% of patients were correctly identified as having the disease, and 85.7% of patients were correctly identified as not having the disease. These results demonstrate the effectiveness of the system as compared to the golden standard and emphasize the important role that smartphones can play in healthcare in the future.

References

[1] Berger, M.; Oksenberg, A.; Silverberg, S.; Arons, E.; Radwan, H.; Iaina, A., "Avoiding the supine position during sleep lowers 24 h blood pressure in obstructive sleep apnea (OSA) patients," *Journal of human hypertension*, vol. 11, no. 10, pp. 657-664, June 1997.

[2] Quan, S.; Gillin, J. C.; Littner, M. R.; Shepard, J. W., "Sleep-related breathing disorders in adults: Recommendations for syndrome definition and measurement techniques in clinical research," *Sleep*, vol. 22, no. 5, pp. 662-689, 1999.

[3] National Sleep Foundation (NSF). [Online]. http://www.sleepfoundation.org/article/hot-topics/let-sleep-work-you. [Accessed 18 5 2013].

[4] Ancoli-Israel, S.; DuHamel, E. R.; Stepnowsky, C.; Engler, R.; Cohen-Zion, M.; Marler, M.;, "The relationship between congestive heart failure, sleep apnea, and mortality in older men," *CHEST, Official Publication of the American College of Chest Physicians*, vol. 124, no. 4, pp. 1400-1405, 2003.

[5] Bucklin, C.L.; Das, M.; Luo, S.L., "An inexpensive accelerometer-based sleep-apnea screening technique," *Proceedings of the IEEE National* on *Aerospace and Electronics Conference (NAECON)*, pp. 396-399, July 2010.

[6] Burgos, A.; Goñi, A.; Illarramendi, A.; Bermudez, J., "Real-Time Detection of Apneas on a PDA," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 4, pp. 995-1002, July 2010.

[7] Hamada, M.; Masahiro, I., "Home monitoring using portable polygraphy for perioperative assessment of pediatric obstructive sleep apnea syndrome," *The Tokai journal of experimental and clinical medicine*, vol. 37, no. 3, p. 66, 2012.

[8] Gao, R.; Yang, L.; Wu, X.; Wang, T.; Lu, S.; Han, F., "A phone-based e-health system for OSAS and its energy issue," *IEEE International Symposium on Information Technology in Medicine and Education (ITME)*, vol.2, pp. 682-686, 3-5 Aug. 2012.

[9] Kailas, A.; Chia-Chin Chong; Watanabe, F., "From Mobile Phones to Personal Wellness Dashboards," *IEEE Pulse Magazine on Medicine and Biology Society*, vol. 1, no. 1, pp. 57-63, Aug. 2010.

[10] National Heart, Lung, and Blood Institute. [Online]. http://www.nhlbi.nih.gov/health/dci/images/sleep_studies.jpg. [Accessed 6 5 2013].

[11] Phillips, G.; Felix, L.; Galli, L.; Patel, V.; Edwards, P., "The effectiveness of M-health technologies for improving health and health services: a systematic review protocol," *BMC research notes*, vol. 3, no. 1, p. 250, 2010.

[12] Myers, T.; Mosby, *Mosby's medical dictionary*: Mosby Inc, 2006.

[13] Jin, Z.; Sun, Y; Cheng, A.C., "Predicting cardiovascular disease from real-time electrocardiographic monitoring: An adaptive machine learning approach on a cell phone," *Proceedings of the IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBC)*, pp. 6889-6892, 3-6 Sept. 2009.

[14] Oresko, J. J.; Jin, Z.; Cheng, J.; Huang, S.; Sun, Y.; Duschl, H.; Cheng, A. C., "A Wearable Smartphone-Based Platform for Real-Time Cardiovascular Disease Detection Via Electrocardiogram Processing," *IEEE Transactions on Information Technology in Biomedicine*, vol.14, no.3, pp. 734,740, May 2010.

[15] Oresko, J. J.; Jin, Z.; Cheng, J.; Huang, S.; Sun, Y.; Duschl, H.; Cheng, A. C., "Intelligent heartsound diagnostics on a cell phone using a hands-free kit," *Proceedings of the AAAI on Artificial Intelligence for Development* (*AI-D)*, pp. 26-31, 2010.

[16] Jim Black. (2009) Microsoft Research. [Online]. http://research.microsoft.com/en-us/collaboration/focus/health/smartphone_clinical_diagnosis.pdf. [Accessed 20 12 2012].

[17] Zhang, Z.; Wu, H.; Wang, W.; Wang, B., "A smartphone based respiratory biofeedback system," *Proceedings of the IEEE 3rd International Conference on Biomedical Engineering and Informatics (BMEI)*, vol. 2, pp. 717-720, 16-18 Oct. 2010.

[18] Scully, C.; Lee, J; Meyer, J.; Gorbach, A.M.; Granquist-Fraser, D.; Mendelson, Y.; Chon, K.H., "Physiological Parameter Monitoring from Optical Recordings With a Mobile Phone," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 2, pp. 303-306, Feb. 2012.

[19] Siewiorek, D., "Generation smartphone," *Spectrum, IEEE* , vol. 49, no. 9, pp. 54-58, September 2012.

[20] Bai, Y.; Xu, B.; Ma, Y.; Sun, G.; Zhao, Y., "Will you have a good sleep tonight? sleep quality prediction with mobile phone*". Proceedings of the 7th International Conference on Body Area Networks. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)*, pp. 124-130, 2012.

[21] Comtois, G.; Salisbury, J.I.; Sun, Y "A smartphone-based platform for analyzing physiological audio signals," *Proceedings of the IEEE 38th Annual Northeast Bioengineering Conference (NEBEC) on Bioengineering*, pp. 69-70, 16-18 March 2012.

[22] Chen, N.; Rabb, M.; Lee, Y.; Schatz, B., "Feasibility of Long-term Monitoring of Everyday Health Through Smartphones," [Online]. https://www.ideals.illinois.edu/handle/2142/18804. [Accessed 14 6 2013].

[23] Oliver, N.; Flores-Mangas, F., "HealthGear: automatic sleep apnea detection and monitoring with a mobile phone," *Journal of Communications,* vol. 2, no. 2, pp. 1-9, 2007.

[24] Cao, Z.; Zhu, R.; Que, R., "A Wireless Portable System with Micro Sensors for Monitoring Respiratory Diseases," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 11, pp. 3110-3116, Nov. 2012.

[25] Lee, Y. D.; Jung, S. J.; Seo, Y. S.; Chung, W. Y., "Measurement of Motion Activity during Ambulatory Using Pulse Oximeter and Triaxial Accelerometer," *Proceedings of the IEEE Third International Conference on Convergence and Hybrid Information Technology*, vol. 1, pp. 436-441, 11-13 Nov. 2008.

[26] Dargie, W., "Analysis of Time and Frequency Domain Features of Accelerometer Measurements," *Proceedings of IEEE 18th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1-6, Aug. 2009.

[27] Rofouei, M.; Sinclair, M.; Bittner, R.; Blank, T.; Saw, N.; DeJean, G.; Heffron, J., "A Non-invasive Wearable Neck-Cuff System for Real-Time Sleep Monitoring," *Proceedings of the IEEE International Conference on Body Sensor Networks (BSN)*, pp. 156-161, 23-25 May 2011.

[28] Mack, D.C.; Alwan, M.; Turner, B.; Suratt, P.; Felder, R.A., "A Passive and Portable System for Monitoring Heart Rate and Detecting Sleep Apnea and Arousals: Preliminary Validation," *Proceedings of the IEEE 1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare (D2H2)*, pp. 51-54, 2-4 April 2006.

[29] Tseng, M. H.; Hsu, H. C.; Chang, C. C.; Ting, H.; Wu, H. C.; Tang, P. H., "Development of an Intelligent App for Obstructive Sleep Apnea Prediction on Android Smartphone Using Data Mining Approach," *Proceedings of the IEEE International Conference on Ubiquitous Intelligence & Computing / Autonomic & Trusted Computing (UIC/ATC)*, pp. 774-779, 2012.

[30] Pillar, G.; Bar, A.; Shlitner, A.; Schnall, R.; Shefy, J.; Lavie, P., "Autonomic arousal index: an automated detection based on peripheral arterial tonometry." *Sleep,* vol. 25, no. 5, pp. 543-549, 2002.

[31] Ayas, N. T.; Pittman, S.; MacDonald, M.; White, D. P., "Assessment of a wrist-worn device in the detection of obstructive sleep apnea." *Sleep,* vol. 4, no. 5, pp. 435-442, 2003.

[32] Bar, A.; Pillar, G.; Dvir, I.; Sheffy, J.; Schnall, R. P.; Lavie, P., "Evaluation of a portable device based on peripheral arterial tone for unattended home sleep studies." *CHEST, Official Publication of the American College of Chest Physicians*, vol.123, no.3, pp. 695-703, 2003.

[33] Itmar. Itamar Medical. [Online]. http://www.itamar-medical.com/. [Accessed 20 12 2012].

[34] Montazeripouragha, A.; Moussavi, Z., *Acoustical analysis of respiratory sounds for detection of obstructive sleep apnea*. Dissertation, University of Manitoba: MSpace, 16-Mar-2012.

[35] Penzel, T.; Blau, A.; Garcia, C.; Schöbel, C.; Sebert, M.; Fietze, I., "Portable monitoring in sleep apnea," *Current Respiratory Care Reports*, vol. 1, no. 2, pp. 139-145, 2012.

[36] Kaniusas, E.; Pfutzner, H.; Saletu, B., "Acoustical signal properties for cardiac/respiratory activity and apneas," *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 11, pp. 1812-1822, Nov. 2005.

[37] Yadollahi, A.; Moussavi, Z., "Acoustic obstructive sleep apnea detection," *Proceedings of the IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBC)*, pp. 7110-7113, 2009.

[38] Huq, S., Moussavi, Z., "Acoustic breath-phase detection using tracheal breath sounds," *Springer Transactions on Medical and Biological Engineering and Computing*, vol. 50, no. 3, pp. 297-308, 2012.

[39] Senny, F.; Destine, J.; Poirrier, R., "Midsagittal Jaw Movement Analysis for the Scoring of Sleep Apneas and Hypopneas," *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 1, pp. 87-95, Jan. 2008.

[40] Chung, F.; Liao, P.; Elsaid, H.; Islam, S.; Shapiro, C. M.; Sun, Y., "Oxygen desaturation index from nocturnal oximetry: A sensitive and specific tool to detect sleep-disordered breathing in surgical patients," *Anesthesia & Analgesia*, vol. 114, no. 5, pp. 993-1000, 2012.

[41] Nonin Pulse Oximeter. [Online]. http://www.nonin.com/.[Accessed 20 12 2012].

[42] CBCNews [Online].
http://www.cbc.ca/news/story/2010/12/21/con-cell-survey.html.
[Accessed 20 12 2012].

[43] Google. [Online].
http://www.thinkwithgoogle.com/insights/featured/our-mobile-planet-2012/.

[Accessed 20 12 2012].

[44] Android. [Online]. http://www.android.com/.[Accessed 20 12 2012].

[45] Matlab. [Online]. http://www.mathworks.com/products/matlab/.
[Accessed 20 12 2012].

[46] Collop, N. A.; Tracy, S. L.; Kapur, V.; Mehra, R.; Kuhlmann, D.; Fleishman, S. A.; Ojile, J. M., "Obstructive sleep apnea devices for out-of-center (OOC) testing: technology evaluation," *Journal of clinical sleep medicine (JCSM), official publication of the American Academy of Sleep Medicine,* vol. 7, no. 5, pp. 531, 2011.

[47] Businessweek. [Online]. http://www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices.[Accessed 18 5 2013].

[48] Android Official Blog. [Online]. http://officialandroid.blogspot.ca/2012/09/google-play-hits-25-billion-downloads.html.[Accessed 18 5 2013].

[49] Android Developers. [Online].
http://developer.android.com/guide/components/activities.html.
[Accessed 18 5 2013].

[50] Shenoi, A., *Introduction to digital signal processing and filter design*: Wiley-Interscience, 2005.

[51] Extrom Laboratories LLC. [Online]. http://www.exstrom.com/journal/sigproc/. [Accessed 18 5 2013].

[52] Android Developers. [Online]. http://developer.android.com/training/articles/perf-anr.html. [Accessed 18 5 2013].

[53] Android Developers. [Online]. http://developer.android.com/reference/android/app/Service.html. [Accessed 18 5 2013].

[54] Android Developers. [Online]. http://developer.android.com/guide/components/processes-and-threads.html. [Accessed 18 5 2013].

[55] Android Developers. [Online]. http://developer.android.com/reference/android/app/IntentService.html. [Accessed 18 5 2013].

[56] Android Developers. [Online]. http://developer.android.com/guide/topics/connectivity/bluetooth.html. [Accessed 18 5 2013].

[57] Android Developers. [Online]. http://developer.android.com/training/articles/perf-tips.html. [Accessed 18 5 2013].

[58] Lalkhen, A.; McCluskey, A., "Clinical tests: sensitivity and specificity." *Continuing Education in Anaesthesia, Critical Care & Pain*, vol. 8, no. 6, pp. 221-223, 2008.

[59] Mangrulkar, R.; Gruber, S., "Patients and Populations: Medical Decision-Making", *open.michigan.* [Online]. http://open.umich.edu/education/med/m1/patients-pop-decision-making/fall2011. [Accessed 13 6 2013].

Appendix

Appendix A: Application Code

Main Activity

```
public class MainActivity extends Activity {
public final static String EXTRA_MESSAGE = "com.oximeter.MESSAGE ";

	private ArrayAdapter<String> adapter;
	public TextView lbl;
	public static final ArrayList<Integer> LIST = new ArrayList<Integer>();
	public static  final ArrayList<Double> accLIST = new ArrayList<Double>();
	public static int count=1;


	@Override
	protected void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		lbl = (TextView) findViewById(R.id.lbl);

		findViewById(R.id.btnConnect).setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {

				if (BtManager.BT_ADAPTER != null) {
					if (BtManager.BT_ADAPTER.isEnabled()) {
						doScanning();
					} else {
						Intent enableIntent = new Intent(
						BluetoothAdapter.ACTION_REQUEST_ENABLE);
						startActivityForResult(enableIntent, 0);
					}

				}

			}
		});

		findViewById(R.id.btnClose).setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {

				//accLIST = new ArrayList<Float>(BtManager.AccLIST);
				Intent i = new Intent(MainActivity.this, BtManager.class);
				stopService(i);
				Toast.makeText(MainActivity.this, "Disconnected",
						Toast.LENGTH_SHORT).show();



				stopService(new Intent(MainActivity.this, BtManager.class));
				Intent intent = new Intent(MainActivity.this, Pretest.class);
				startActivity(intent);
			}
		});

		IntentFilter in = new IntentFilter(BtManager.ACTION);
		registerReceiver(MyReciever, in);
	}

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

```
private void doScanning() {
		Intent i = new Intent(MainActivity.this, BtManager.class);
		stopService(i);

		ArrayList<String> list = new ArrayList<String>();
		adapter = new ArrayAdapter<String>(this,
					android.R.layout.simple_spinner_item, list);
		adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
		showList();
		IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
		registerReceiver(newDeviceReceiver, filter);
		filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
		this.registerReceiver(newDeviceReceiver, filter);

		Toast.makeText(this, "Finding Bluetooth devices...", Toast.LENGTH_LONG)
					.show();
		BtManager.BT_ADAPTER.startDiscovery();
}


private final BroadcastReceiver newDeviceReceiver = new BroadcastReceiver() {
		@Override
		public void onReceive(Context context, Intent intent) {

			String action = intent.getAction();
			if (BluetoothDevice.ACTION_FOUND.equals(action)) {
				BluetoothDevice device = intent
							.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
				String name = device.getName() + " (" + device.getAddress()
							+ ")";
				if (adapter.getPosition(name) == -1)
					adapter.add(name);
			} else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED
					.equals(action)) {
				if (adapter.getCount() == 0) {
					Toast.makeText(MainActivity.this, "No device found!",
							Toast.LENGTH_LONG).show();
					// finish();
				} else
					Toast.makeText(MainActivity.this, "Searching finished!",
							Toast.LENGTH_LONG).show();
			}
		}
};


@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

		super.onActivityResult(requestCode, resultCode, data);
		if (resultCode == RESULT_OK)
			doScanning();
		else {
			Toast.makeText(this, "Bluetooth is turned off.", Toast.LENGTH_LONG)
					.show();
		}
}


private void showList() {

		Builder b = new Builder(this);
		b.setTitle("Select Device:");
		b.setAdapter(adapter, new DialogInterface.OnClickListener() {

			@Override
			public void onClick(DialogInterface dialog, int which) {
				BtManager.BT_ADAPTER.cancelDiscovery();
				dialog.dismiss();
				String s = adapter.getItem(which);

				String mac = s.substring(s.lastIndexOf("(") + 1, s.length() - 1);
				PreferenceManager
							.getDefaultSharedPreferences(MainActivity.this).edit()
							.putString("MAC", mac).commit();

				Intent i = new Intent(MainActivity.this, BtManager.class);
				stopService(i);
				startService(i);
```

```
                }
        });
        b.setNeutralButton("Cancel", new DialogInterface.OnClickListener() {

                @Override
                public void onClick(DialogInterface dialog, int which) {
                        BtManager.BT_ADAPTER.cancelDiscovery();
                        dialog.dismiss();
                }
        });

        b.create().show();
}

private ProgressDialog dia;
private BroadcastReceiver MyReciever = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
                if (BtManager.ACTION.equalsIgnoreCase(intent.getAction())) {
                        int status = intent.getIntExtra(BtManager.STATUS, 0);
                        switch (status) {
                        case BtManager.CONNECTING:
                                dia = ProgressDialog.show(MainActivity.this, null,
                                                "Connecting");
                                break;
                        case BtManager.CONNECTED:
                                if (dia != null && dia.isShowing())
                                        dia.dismiss();
                                Toast.makeText(MainActivity.this,
                                                "Bluetooth devices paired.", Toast.LENGTH_LONG)
                                                .show();
                                break;
                        case BtManager.FAILED:
                                if (dia != null && dia.isShowing())
                                        dia.dismiss();
                                Toast.makeText(MainActivity.this,
                                                "Unable to connect the device!", Toast.LENGTH_LONG)
                                                .show();

                                break;
                        case BtManager.DATA:
                                String val = intent.getStringExtra(BtManager.VALUE);
                                if (!val.equalsIgnoreCase("Data= NO DATA")) {
                                        LIST.add(Integer.parseInt(val.replace("Data=", "")));

                                }
                                lbl.setText(val);
                                break;

                        case BtManager.DATA2:
                                int val2 = intent.getIntExtra(BtManager.VALUE,0);
                                        count=val2;

                        case BtManager.DATA3:
                                double val3 = intent.getDoubleExtra(BtManager.VALUE, 0);
                                accLIST.add(val3);

                                break;

                        default:
                                break;
                        }
                }
        }
};}
```

BtManager

```
public class BtManager extends Service implements SensorEventListener
{
	public static final String ACTION="com.oximeter.BtManager";
	public static final String STATUS="status";
	public static final int CONNECTING=1;
	public static final int CONNECTED=2;
	public static final int FAILED=3;
	public static final int DATA=4;
	public static final int DATA2=5;
	public static final int DATA3=6;
	public static final String VALUE="value";
	public static final BluetoothAdapter BT_ADAPTER = BluetoothAdapter.getDefaultAdapter();
	public final ArrayList<Byte> LIST=new ArrayList<Byte>();
	private DataOutputStream fout;
	private BluetoothSocket socket;

	////////////////////////////////Accelerometer parameters:
	SensorManager sm;
	TextView Rss;
	public static double  Result;
	double xSensor;
	double ySensor;
	double zSensor;
	double sum;
	public static ArrayList<Double> AccLIST=new ArrayList<Double>();//accelerometer list
	private DataOutputStream foutacc;


	////////////////////////////////

	public  void connect()
	{
		Intent i=new Intent(ACTION);
		i.putExtra(STATUS, CONNECTING);
		sendBroadcast(i);

		String mac=PreferenceManager.getDefaultSharedPreferences(this).getString("MAC", "no-data");
		Log.e("MAC", mac);
		try
		{
			BluetoothDevice device = BtManager.BT_ADAPTER.getRemoteDevice(mac);
			socket = null;
			socket = device.createRfcommSocketToServiceRecord(UUID
					.fromString("00001101-0000-1000-8000-00805F9B34FB"));
			BtManager.BT_ADAPTER.cancelDiscovery();

			Method m = BluetoothDevice.class.getMethod("convertPinToBytes",
					new Class[] { String.class });
			byte[] pin = (byte[]) m.invoke(device, "1234");
			m = device.getClass().getMethod("setPin", new Class[] { pin.getClass() });
			m.invoke(device, pin);
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}

		try
		{
			Log.e("SERVICE", "CONNECTING");
			socket.connect();

			i=new Intent(ACTION);
			i.putExtra(STATUS, CONNECTED);
			sendBroadcast(i);

			Log.e("SERVICE", "CONNECTED");

			doStart();

		}
		catch (IOException e)
		{
			e.printStackTrace();
			try
			{
```

```
                                        if (socket != null)
                                                socket.close();
                                }
                                catch (IOException e2)
                                {
                                        e2.printStackTrace();
                                }
                                Log.e("SERVICE", "FAILED");
                                i=new Intent(ACTION);
                                i.putExtra(STATUS, FAILED);
                                sendBroadcast(i);

                        }

        }

        public void startListen(final InputStream in)
        {
                        Log.e("SERVICE", "LISTEN"+socket);
                        LIST.clear();
                        //AccLIST.clear();
                        new Thread(new Runnable()
                        {
                                public void run()
                                {
                                        while (socket!=null)
                                        {
                                                long diff=System.currentTimeMillis()-startTime;
                                                if(diff>60*60*1000)
                                                {
                                                        stopRecording();
                                                        count++;
                                                        MainActivity.count++;
                                                        startTime=System.currentTimeMillis();
                                                        startRecording();

                                                        Intent i=new Intent(ACTION);
                                                        i.putExtra(STATUS, DATA2);
                                                        i.putExtra(VALUE, count);
                                                        sendBroadcast(i);
                                                }
                                                Log.e("SERVICE", "LISTEN=WHILE");
                                                byte[] buffer = new byte[32];
                                                try
                                                {
                                                        in.read(buffer);

                                                        final byte spo2 = buffer[2];
                                                        System.out.println("MESSAGE: " + new String(buffer));
                                                        System.out.println("SPO2: " + spo2);

                                                        if (spo2 != 127)
                                                        {

                                                                LIST.add(spo2);
                                                                if(fout==null)
                                                                        fout=new DataOutputStream(new
                                        FileOutputStream(new File(Environment.getExternalStorageDirectory(),"Oximeter.txt")));
                                                                String str=spo2+", ";
                                                                fout.writeUTF(str);
                                                        }

                                                        /////accelerometer part

                                                        //AccLIST.add(Result);
                                                        MainActivity.accLIST.add(Result);
                                                        if(foutacc==null)
                                                                foutacc=new DataOutputStream(new FileOutputStream(new
                                        File(Environment.getExternalStorageDirectory(),"Accelerometer.txt")));
                                                        String str1=Result+", ";
                                                        foutacc.writeUTF(str1);


                                                        Log.e("SERVICE", "DISPLAYED");
                                                        String str;
                                                        if (spo2 != 127)
                                                                str="Data=" + spo2;
                                                        else
                                                                str="Data= NO DATA";
```

```
                                                Intent i=new Intent(ACTION);
                                                i.putExtra(STATUS, DATA);
                                                i.putExtra(VALUE, str);
                                                sendBroadcast(i);


                                                Intent i2=new Intent(ACTION);
                                                i2.putExtra(STATUS, DATA3);
                                                i2.putExtra(VALUE, Result);
                                                sendBroadcast(i2);

                                                Thread.sleep(1000);

                                        }
                                        catch (Exception e)
                                        {
                                                e.printStackTrace();
                                        }
                                }
                        }
                }).start();


}

private void sendData(final OutputStream out)
{

        try
        {
                out.write(new byte[] { 0x02, 0x70, 0x02, 0x02, 0x08, 0x03 });
                out.flush();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
}


public void close()
{

        Log.e("SERVICE", "CLOSE");
        try
        {
                if (socket != null)
                        socket.close();
                socket=null;
                if(fout!=null)
                {
                        fout.flush();
                        fout.close();
                }
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
}

@Override
public IBinder onBind(Intent intent)
{
        return null;
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
        return START_STICKY;
}

@Override
public void onCreate()
{
        super.onCreate();

        Notification notification = new Notification(R.drawable.ic_launcher, "Oximeter",
                System.currentTimeMillis());
        Intent notificationIntent = new Intent(this, MainActivity.class);
```

```
		PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
		notification.setLatestEventInfo(this, "Oximeter","Service started", pendingIntent);
		startForeground(100, notification);
		Log.e("SERVICE", "CREAT");

		sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
		Sensor S = sm.getSensorList(Sensor.TYPE_ACCELEROMETER).get(0);
		sm.registerListener(this, S, SensorManager.SENSOR_DELAY_NORMAL );

		connect();

}
@Override
public void onDestroy()
{

		super.onDestroy();
		stopRecording();
		Log.e("SERVICE", "DESTROY");
		stopForeground(true);
		close();

}

private long startTime;
private void doStart()
{
		Log.e("SERVICE", "STARt"+socket);
		try
		{
			if(socket!=null)
			{
				startTime=System.currentTimeMillis();
				startRecording();
				sendData(socket.getOutputStream());
				startListen(socket.getInputStream());
			}
		}
		catch (IOException e)
		{
			e.printStackTrace();
			stopSelf();
		}

}

public static int count=1;
private ExtAudioRecorder recorder = null;
private void startRecording() {
   recorder = ExtAudioRecorder.getInstanse(false);
   File f=new File(Environment.getExternalStorageDirectory(), "external_sd/AudioRecorder");

  //  f.mkdirs();
   //recorder.setOutputFile(new File(f, "audio.wav").getAbsolutePath());

   f.mkdirs();
   recorder.setOutputFile(new File(f, "audio"+count+".wav").getAbsolutePath());


   try {
      recorder.prepare();
      recorder.start();
   } catch (IllegalStateException e) {
      e.printStackTrace();
   } catch (Exception e) {
      e.printStackTrace();
   }
}

private void stopRecording()
{
		try
		{
			if(recorder!=null)
			{
				recorder.stop();
				recorder.reset();
				recorder.release();
			}
		}catch (Exception e) {
```

```
                        e.printStackTrace();
                }
        }


        ////////////////////////////Functions related to the accelerometer:


        @Override
        public void onAccuracyChanged(Sensor arg0, int arg1) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onSensorChanged(SensorEvent e) {
                // TODO Auto-generated method stub

                xSensor = e.values[0];
                ySensor = e.values[1];
                zSensor = e.values[2];
                sum=(double) (Math.pow(xSensor, 2)+Math.pow(xSensor, 2)+Math.pow(xSensor, 2));
                Result=Math.sqrt(sum);

        }
}
```

ExtAudioRecorder

```
public class ExtAudioRecorder
{
        private final static int[] sampleRates = {44100, 22050, 11025, 8000};

        public static ExtAudioRecorder getInstanse(Boolean recordingCompressed)
        {
                ExtAudioRecorder result = null;

                if(recordingCompressed)
                {
                        result = new ExtAudioRecorder(          false,

        AudioSource.MIC,

        sampleRates[3],

        AudioFormat.CHANNEL_CONFIGURATION_MONO,

        AudioFormat.ENCODING_PCM_16BIT);
                }
                else
                {
                        int i=0;
                        do
                        {
                                result = new ExtAudioRecorder(          true,

        AudioSource.MIC,

        sampleRates[i],

        AudioFormat.CHANNEL_CONFIGURATION_MONO,

        AudioFormat.ENCODING_PCM_16BIT);

                        } while((++i<sampleRates.length) & !(result.getState() == ExtAudioRecorder.State.INITIALIZING));
                }
                return result;
        }


        public enum State {INITIALIZING, READY, RECORDING, ERROR, STOPPED};

        public static final boolean RECORDING_UNCOMPRESSED = true;
        public static final boolean RECORDING_COMPRESSED = false;
        private static final int TIMER_INTERVAL = 120;
        private boolean         uncomp;
        private AudioRecord     audioRec = null;
```

```
private int             Amplit= 0;
private String          filePath = null;
private State           state;
private RandomAccessFile randomAccessWriter;
private short           num_Channels;
private int             s_Rate;
private short           b_Samples;
private int             buffer_Size;
private int             audio_Source;
private int             aFormat;
private int             framePeriod;
private byte[]          buffer;
private int             payloadSize;

public State getState()
{
        return state;
}


private AudioRecord.OnRecordPositionUpdateListener updateListener = new AudioRecord.OnRecordPositionUpdateListener()
{
        public void onPeriodicNotification(AudioRecord recorder)
        {
                audioRec.read(buffer, 0, buffer.length);
                try
                {
                        randomAccessWriter.write(buffer);
                        payloadSize += buffer.length;
                        if (b_Samples == 16)
                        {
                                for (int i=0; i<buffer.length/2; i++)
                                {
                                        short curSample = getShort(buffer[i*2], buffer[i*2+1]);
                                        if (curSample > Amplit)
                                        {
                                                Amplit = curSample;
                                        }
                                }
                        }
                        else
                        {
                                for (int i=0; i<buffer.length; i++)
                                {
                                        if (buffer[i] > Amplit)
                                        {
                                                Amplit = buffer[i];
                                        }
                                }
                        }
                }
                catch (IOException e)
                {
                        Log.e(ExtAudioRecorder.class.getName(), "Error ");

                }
        }

        public void onMarkerReached(AudioRecord recorder)
        {

        }
};

public ExtAudioRecorder(boolean uncompressed, int audioSource, int sampleRate, int channelConfig, int audioFormat)
{
        try
        {
                uncomp = uncompressed;

                        if (audioFormat == AudioFormat.ENCODING_PCM_16BIT)
                        {
                                b_Samples = 16;
                        }
                        else
                        {
                                b_Samples = 8;
                        }
```

```
                                        if (channelConfig == AudioFormat.CHANNEL_CONFIGURATION_MONO)
                                        {
                                                num_Channels = 1;
                                        }
                                        else
                                        {
                                                num_Channels = 2;
                                        }

                                        audio_Source = audioSource;
                                        s_Rate   = sampleRate;
                                        aFormat = audioFormat;

                                        framePeriod = sampleRate * TIMER_INTERVAL / 1000;
                                        buffer_Size = framePeriod * 2 * b_Samples * num_Channels / 8;
                                        if (buffer_Size < AudioRecord.getMinBufferSize(sampleRate, channelConfig, audioFormat))
                                        {
                                                buffer_Size = AudioRecord.getMinBufferSize(sampleRate, channelConfig, audioFormat);
                                                framePeriod = buffer_Size / ( 2 * b_Samples * num_Channels / 8 );
                                        Log.w(ExtAudioRecorder.class.getName(), "Increasing buffer size to " +
                                        Integer.toString(buffer_Size));
                                        }

                                        audioRec = new AudioRecord(audioSource, sampleRate, channelConfig, audioFormat, buffer_Size);

                                        if (audioRec.getState() != AudioRecord.STATE_INITIALIZED)
                                                throw new Exception("AudioRecord initialization failed");
                                        audioRec.setRecordPositionUpdateListener(updateListener);
                                        audioRec.setPositionNotificationPeriod(framePeriod);

                                Amplit = 0;
                                filePath = null;
                                state = State.INITIALIZING;
                        } catch (Exception e)
                        {
                                state = State.ERROR;
                        }
}


public void setOutputFile(String argPath)
{
                try
                {
                        if (state == State.INITIALIZING)
                        {
                                filePath = argPath;

                        }
                }
                catch (Exception e)
                {

                        state = State.ERROR;
                }
}


public int getMaxAmplitude()
{
                if (state == State.RECORDING)
                {


                                        int result = Amplit;
                                        Amplit = 0;
                                        return result;


                }
                else
                {
                        return 0;
                }
}
```

```
public void prepare()
{
	try
	{
		if (state == State.INITIALIZING)
		{

			if ((audioRec.getState() == AudioRecord.STATE_INITIALIZED) & (filePath != null))
			{
				// write file header

				randomAccessWriter = new RandomAccessFile(filePath, "rw");

				randomAccessWriter.setLength(0);
				randomAccessWriter.writeBytes("RIFF");
				randomAccessWriter.writeInt(0);
				randomAccessWriter.writeBytes("WAVE");
				randomAccessWriter.writeBytes("fmt ");
				randomAccessWriter.writeInt(Integer.reverseBytes(16));
				randomAccessWriter.writeShort(Short.reverseBytes((short) 1));
				randomAccessWriter.writeShort(Short.reverseBytes(num_Channels));
				randomAccessWriter.writeInt(Integer.reverseBytes(s_Rate));
		randomAccessWriter.writeInt(Integer.reverseBytes(s_Rate*b_Samples*num_Channels/8));
		randomAccessWriter.writeShort(Short.reverseBytes((short)(num_Channels*b_Samples/8)));
				randomAccessWriter.writeShort(Short.reverseBytes(b_Samples));
				randomAccessWriter.writeBytes("data");
				randomAccessWriter.writeInt(0);

				buffer = new byte[framePeriod*b_Samples/8*num_Channels];
				state = State.READY;
			}
			else
			{

				state = State.ERROR;
			}


		}
		else
		{
			release();
			state = State.ERROR;
		}
	}
	catch(Exception e)
	{

		state = State.ERROR;
	}
}

public void release()
{
	if (state == State.RECORDING)
	{
		stop();
	}
	else
	{
		if ((state == State.READY) & (uncomp))
		{
			try
			{
				randomAccessWriter.close(); // Remove prepared file
			}
			catch (IOException e)
			{
				Log.e(ExtAudioRecorder.class.getName(), "closing error");
			}
			(new File(filePath)).delete();
		}
	}


		if (audioRec != null)
		{
			audioRec.release();
		}}
```

```
	public void reset()
	{
		try
		{
			if (state != State.ERROR)
			{
				release();
				filePath = null;
				Amplit = 0;


				audioRec = new AudioRecord(audio_Source, s_Rate, num_Channels+1, aFormat, buffer_Size);


				state = State.INITIALIZING;
			}
		}
		catch (Exception e)
		{
			state = State.ERROR;
		}
	}


	public void start()
	{
		if (state == State.READY)
		{


				payloadSize = 0;
				audioRec.startRecording();
				audioRec.read(buffer, 0, buffer.length);

			state = State.RECORDING;
		}

	}


	public void stop()
	{
		if (state == State.RECORDING)
		{

				audioRec.stop();

				try
				{
					randomAccessWriter.seek(4);
					randomAccessWriter.writeInt(Integer.reverseBytes(36+payloadSize));

					randomAccessWriter.seek(40);
					randomAccessWriter.writeInt(Integer.reverseBytes(payloadSize));

					randomAccessWriter.close();
				}
				catch(IOException e)
				{
				Log.e(ExtAudioRecorder.class.getName(), "I/O exception occured while closing output file");
				state = State.ERROR;
				}

			state = State.STOPPED;
		}

	}


	private short getShort(byte argB1, byte argB2)
	{
		return (short)(argB1 | (argB2 << 8));
	}

}
```

Pretest

```
public class Pretest extends Activity{

	CheckBox CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8, CH9;

	public int PretestCounter=0;
	public int SpecialCounter=0;
	String message;
	//public static  ArrayList<Byte> mm;
	public final static String EXTRA_MESSAGE = "com.oximeter.MESSAGE";

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);
		setContentView(R.layout.pretest);
		//Intent intent = getIntent();
		//mm = intent.getByteArrayExtra(MainActivity.ll);
		 ((Button)findViewById(R.id.btnDone)).setOnClickListener(btnClick);
		Initialize();
	}

	private void Initialize() {
		// TODO Auto-generated method stub
		CH1=(CheckBox)findViewById(R.id.checkBox1);
		CH2=(CheckBox)findViewById(R.id.checkBox2);
		CH3=(CheckBox)findViewById(R.id.checkBox3);
		CH4=(CheckBox)findViewById(R.id.checkBox4);
		CH5=(CheckBox)findViewById(R.id.checkBox5);
		CH6=(CheckBox)findViewById(R.id.checkBox6);
		CH7=(CheckBox)findViewById(R.id.checkBox7);
		CH8=(CheckBox)findViewById(R.id.checkBox8);
		CH9=(CheckBox)findViewById(R.id.checkBox9);

		CH1.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
					SpecialCounter++;
				}
			}

		});
		CH2.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
				}
			}

		});
		CH3.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
					SpecialCounter++;
				}
			}

		});
		CH4.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
					SpecialCounter++;
		}}});
```

```
		CH5.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
					SpecialCounter++;
				}
			}

		});
		CH6.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
				}
			}

		});
		CH7.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
				}
			}

		});
		CH8.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
				}
			}

		});
		CH9.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					PretestCounter+=1;
					SpecialCounter++;
				}
			}

		});

	}
	private View.OnClickListener btnClick = new View.OnClickListener() {
		public void onClick(View v) {
			switch(v.getId()){
				case R.id.btnDone:{

					if(PretestCounter>5 || SpecialCounter>4)
						message="Yes";
					else
						message="No";
					Intent intent = new Intent(Pretest.this, Diagnose.class);

					intent.putExtra(EXTRA_MESSAGE, message);
					startActivity(intent);
					break;
				}


			}}};}
```

Diagnose

```
public class Diagnose extends Activity{

	CheckBox CH1, CH2, CH3;
	public static TextView Result,Severity, Index;
	public static int ODI_Sum=0;
	public static int p=1;
	public static String Severity2; //to be used in the report class
	public static String result; //to be used in the report class
	private boolean OximeterOnly = false;
	private boolean RespiratoryOnly = false;
	private boolean Both = false;
	private static String message;
	private static ArrayList<String> LIST=new ArrayList<String>();

	//variables to be used by the oximeter algorithm:
	double Sum=0;
	double Average=0;
	double Threshold;
	double EventCounter=0;
	double Events=0;
	double AHI=0;
	public static double ODI=0;
	public static double NumberOfHours;
	public static double AverageIndices=0;

	int test=0;
	String[] inputArray;
	String delimiter = ", ";
	String input="";


	double message2;

	private ResponseReceiver receiver;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);

		IntentFilter filter = new IntentFilter(ResponseReceiver.ACTION_RESP);
		 filter.addCategory(Intent.CATEGORY_DEFAULT);
		  receiver = new ResponseReceiver();
		  registerReceiver(receiver, filter);

		//get the intent from the Pretest class.....Intent contain the pretest probability as Yes or No....value is passed in message string
		Intent intent = getIntent();
		message = intent.getStringExtra(Pretest.EXTRA_MESSAGE);


		NumberOfHours=(double)MainActivity.LIST.size()/(60.0*60.0);

		setContentView(R.layout.diagnose);

		//initialize the checkboxes. These boxes are used to enable the user to choose which analysis he/she wants
		SetCheckBoxHandler();

		//initialize the button.
		SetButtonHandler();

		//the Button is only enabled if one of the checkboxes is enabled.
		enableButton(R.id.btnDiagnose,false);
		enableButton(R.id.btnReport,false);


		CH1.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					OximeterOnly=true;
					enableButton(R.id.btnDiagnose,true);
				}
			}});
```

```
		CH2.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					RespiratoryOnly=true;
					enableButton(R.id.btnDiagnose,true);
				}
			}

		});
		CH3.setOnClickListener(new OnClickListener() {

			@Override
			public void onClick(View v) {
				// TODO Auto-generated method stub
				if (((CheckBox) v).isChecked()) {
					Both=true;
					enableButton(R.id.btnDiagnose,true);
				}
			}

		});

}
private void SetButtonHandler() {
		// TODO Auto-generated method stub
		 ((Button)findViewById(R.id.btnDiagnose)).setOnClickListener(btnClick);
		 ((Button)findViewById(R.id.btnReport)).setOnClickListener(btnClick);
}

private void SetCheckBoxHandler() {
		// TODO Auto-generated method stub

		CH1=(CheckBox)findViewById(R.id.cbOximeter);
		CH2=(CheckBox)findViewById(R.id.cbRespiratory);
		CH3=(CheckBox)findViewById(R.id.cbBoth);
		Result = (TextView) findViewById(R.id.tvResult);
		Severity = (TextView) findViewById(R.id.tvSevirity);
		Index = (TextView) findViewById(R.id.textView2);
}
private void enableButton(int id,boolean isEnable){
		((Button)findViewById(id)).setEnabled(isEnable);

}

private View.OnClickListener btnClick = new View.OnClickListener() {
		public void onClick(View v) {
			switch(v.getId()){
				case R.id.btnDiagnose:{
					if(CH1.isChecked())
					{
						OximeterAlgorithm();
						Index.setText(Double.toString(AHI));
						message2=AHI;

					}
					else if(CH2.isChecked())
					{


						processingRespiratoryFiles();



					}
						/*else if(CH3.isChecked())
					{
						processingRespiratoryFiles();

					}*/


					enableButton(R.id.btnReport,true);

					break;
				}
				case R.id.btnReport:{
```

```
                                                Intent intent = new Intent(Diagnose.this, Report.class);
                                                intent.putExtra("recTime", message2);

                                                startActivity(intent);}}}};


//////////////Respiratory efforts Algorithm//////////////
public  void processingRespiratoryFiles() {
                if(p<=MainActivity.count)
                {
                filePath = Environment.getExternalStorageDirectory() + "/external_sd/AudioRecorder/audio"+p+".wav";

                File filein = new File(filePath);
                if (filein.exists())
                {
                                Intent msgIntent = new Intent(this, RespiratoryAnalysisService.class);
                        msgIntent.putExtra(RespiratoryAnalysisService.PARAM_IN_MSG, filePath);
                        startService(msgIntent);


                }
                }
                else
                {
                                ODI_Sum=0;
                RespiratoryHandling(ODI_Sum,0);
                Index.setText(Double.toString(ODI_Sum));
                message2=ODI_Sum;
                }


}


//////Oximeter Algorithm/////////
private void OximeterAlgorithm()
{
                Events=0;
                Average=0;
                Sum=0;

                //Calibration phase
                for(int i=3; i<33; i++)
                {
                                Sum=Sum+MainActivity.LIST.get(i);
                }
                Average=Sum/30;
                Threshold=0.96*Average;// the threshold is set to be less than the average by 4%
                for(int i=3; i<MainActivity.LIST.size(); i++)//Start from three, to avid any wrong values in the beginning
                {
                                EventCounter=0;
                                while(MainActivity.LIST.get(i)<=Threshold)
                                {
                                                EventCounter++;
                                                if(i!=(MainActivity.LIST.size()-1))
                                                i++;
                                                else
                                                                break;
                                }
                                if(EventCounter>=10)//this means that the apnea lasts more than 10 sec
                                                Events++;
                }

                AHI=Events/NumberOfHours;



                if(CH3.isChecked()==false)
                {
                if(AHI>=10 ||(AHI>=5 && message=="Yes"))//the message is the yes if the pretest shows +ve value
                {
                                DisplayResult("Positive");
                                if(AHI>=5 && AHI<15)
                                {
                                                DisplaySevirity("Mild");
                                }
                                else if(AHI>=15 && AHI<30)
                                {
                                                DisplaySevirity("Moderate");
                                }
```

```
                    else if(AHI>=30)
                    {
                              DisplaySevirity("Severe");
                    }
          }
          else
          {

                    DisplayResult("Negative");
          }
          }



}


private static void DisplayResult(String s)
{
          if(s=="Positive")
          {
                    result="you have OSA";
                    Result.setText("you have OSA");
          }
          else if(s=="Negative")
          {
                    result="you DON'T have OSA";
                    Result.setText("you DON'T have OSA");
          }
}
private void DisplaySevirity(String s)
{
          if(s=="Mild")
          {
                    Severity2="Mild";
                    Severity.setText("Mild");
          }
          else if(s=="Moderate")
          {
                    Severity2="Moderate";
                    Severity.setText("Moderate");
          }
          else if(s=="Severe")
          {
                    Severity2="Severe";
                    Severity.setText("Severe");
          }
}
private  void RespiratoryHandling(double ODI,double AHI)
{
          if(CH3.isChecked()==false)
          {
          if(ODI>10 ||(ODI>5 && message=="Yes"))
          {
                    DisplayResult("Positive");
                    if(ODI>=5 && ODI<15)
                    {
                              DisplaySevirity("Mild");
                    }
                    else if(ODI>=15 && AHI<30)
                    {
                              DisplaySevirity("Moderate");
                    }
                    else if(ODI>=30)
                    {
                              DisplaySevirity("Severe");
                    }
          }
          else
          {
                    DisplayResult("Negative");
          }
          }
          else
          {
                    AverageIndices=(AHI+ODI)/2;
                    if(AverageIndices>10 ||(AverageIndices>5 && message=="Yes"))
                    {
                              DisplayResult("Positive");
                              if(AverageIndices>=5 && AverageIndices<15)
```

```
                                        {
                                                DisplaySevirity("Mild");
                                        }
                                        else if(AverageIndices>=15 && AverageIndices<30)
                                        {
                                                DisplaySevirity("Moderate");
                                        }
                                        else if(AverageIndices>=30)
                                        {
                                                DisplaySevirity("Severe");
                                        }

                                }
                                else
                                {
                                        DisplayResult("Negative");
                                }
                        }
}

 public class ResponseReceiver extends BroadcastReceiver {
     public static final String ACTION_RESP = "com.oximeter.intent.action.MESSAGE_PROCESSED";
     @Override
     public void onReceive(Context context, Intent intent) {

       // Update UI, new "message" processed by SimpleIntentService

       String text = intent.getStringExtra(RespiratoryAnalysisService.PARAM_OUT_MSG);

       //ODI=(double) Double.parseDouble(text) /NumberOfHours;
       Events= Double.parseDouble(text);
       ODI=Events/NumberOfHours;
                                        ODI_Sum+=ODI;

                                        if(p==2)
                                                p+=2;
                                        else
                                                p++;
       processingRespiratoryFiles();

     }}}
```

RespiratoryAnalysisService

```
public class RespiratoryAnalysisService extends IntentService{

        public static final String PARAM_IN_MSG = "imsg";
        public static final String PARAM_OUT_MSG = "omsg";
        double mean_energy;
        double ThresholdR ;
        int event = 0;
        double sum =0.0;
        int ord=5;
        public static String filePath;
        private int input_ptr;
        private int output_ptr;
        private double[] input_sig;
        private double[] output_sig;

        // coefficients of the filter of order 5.
        private double[] acoeffecient = { 1.0, -9.70759194981593, 42.4255663720555,
                                -109.924013757227, 186.990630312725, -218.214580502029,
                                176.920871031747, -98.4037968389088, 35.9342553349938,
                                -7.77959077393888, 0.758250770398488 };

        private double[] bcoffecient = { 1.26227039651684e-07, 0.0,
                                -6.31135198258419e-07, 0.0, 1.26227039651684e-06, 0.0,
                                -1.26227039651684e-06, 0.0, 6.31135198258419e-07, 0,
                                -1.26227039651684e-07 };

        public RespiratoryAnalysisService() {
                super("RespiratoryAnalysisService");
                // TODO Auto-generated constructor stub
        }


        @Override
```

```
protected void onHandleIntent(Intent intent) {
		// TODO Auto-generated method stub

		String msg = intent.getStringExtra(PARAM_IN_MSG);//getting the audio file path from the diagnose activity
		try {
				event = 0;
				sum =0.0;

				byte[] header = new byte[44];

				File filein = new File(msg);
				FileInputStream fin = new FileInputStream(filein);
				BufferedInputStream bin = new BufferedInputStream(fin);
				DataInputStream data_in_stream = new DataInputStream(bin);

				//Extracting information from the header
				data_in_stream.read(header);

				int fs = (header[24] & 0xff) | (header[25] & 0xff) << 8
						| (header[26] & 0xff) << 16 | (header[27] & 0xff) << 24;
				double Ts = 1.0 / fs;
				// Number of Channels
				int numChannels = (header[22] & 0xff) | (header[23] & 0xff) << 8;
				// Bits Per Sample
				int bitsPerSample = (header[34] & 0xff) | (header[35] & 0xff) << 8;

				// Sample Rate
				int numBytes = (header[40] & 0xff) | (header[41] & 0xff) << 8
						| (header[42] & 0xff) << 16 | (header[43] & 0xff) << 24;
				int N = numBytes / numChannels / (bitsPerSample / 8);


				// Calibration phase
				// Find average energy for the first 30 seconds
				double energy = 0.0;
				reset_buffer(acoeffecient.length, bcoffecient.length);
				for (int i = 0; i < 30; i++) {
					sum = readEnergyFiltered(data_in_stream, fs, numChannels);
					energy += sum * Ts;
				}
				if(Diagnose.p==1)
				{
				 mean_energy = energy / 30.0;
				 ThresholdR = 0.9 * mean_energy;
				}
				reset_buffer(acoeffecient.length, bcoffecient.length);
				data_in_stream.close();
				bin.close();
				fin.close();

				fin = new FileInputStream(filein);
				bin = new BufferedInputStream(fin);
				data_in_stream = new DataInputStream(bin);
				data_in_stream.read(header);

				N = N / fs;
				int j = 1;
				while (j <= N) {
					int counter = 0;
					//int trigger = 0;
					energy = readEnergyFiltered(data_in_stream, fs, numChannels) * Ts;
					if (energy >= 0.) {


						while ((energy < ThresholdR)) {
							counter++;
							energy = readEnergyFiltered(data_in_stream, fs, numChannels) * Ts;

							if (energy < 0)
								break;

						}
					}
```

```
                                        if (counter > 10)
                                                event++;

                                        if (counter > 0)
                                                j += counter;
                                        else
                                                j++;
                                }
                                data_in_stream.close();
                                bin.close();
                                fin.close();

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

Intent broadcastIntent = new Intent();
broadcastIntent.setAction(ResponseReceiver.ACTION_RESP);
broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
broadcastIntent.putExtra(PARAM_OUT_MSG, Integer.toString(event));
sendBroadcast(broadcastIntent);

        }


        public void reset_buffer(int alength, int blength) {
                input_sig = new double[blength + 1];
                output_sig = new double[alength + 1];

                input_ptr = 0;
                output_ptr = 0;
        }


        public double readEnergyFiltered(DataInputStream data_in_stream, int length,
                                int numChannels) {
                double energy = 0.0;

                try {
                        byte[] byteBuffer = new byte[length * 2 * numChannels];


                        int r = data_in_stream.read(byteBuffer);

                        if (r == -1)
                                return -1.0;
                        r = r / (2 * numChannels);

                        double y;

                        for (int i = 0; i < r; ++i) {

                                short sample = (short) ((byteBuffer[2 * numChannels * i] & 0xff) | (byteBuffer[2
                                                * numChannels * i + 1] & 0xff) << 8);
                                input_sig[input_ptr] = (double) sample / 32768.0;


                                y = bcoffecient[0] * input_sig[0];

                                for (int j = 1; j < ord+1; j++)
                                {
                                        y=0;
                                        for(int k=0; k<j+1;k++)
                                        y += bcoffecient[k] * input_sig[j-k];
                                for (int k = 0; k < j; k++)
                                        y -= acoeffecient[k+1] * output_sig[j-k-1];

                                }
                                output_sig[output_ptr] = y;
                                for(i=ord+1;i<r+1;i++)
                                {
                                        output_sig[i]=0;
                                        for(int j=0;j<ord+1;j++)
                                                output_sig[i]+=         bcoffecient[j]*input_sig[i-j];
                                        for(int j=0;j<ord;j++)
                                                output_sig[i]-=         acoeffecient[j+1]*output_sig[i-j-1];
                                }

                                energy += output_sig[output_ptr] * output_sig[output_ptr];
```

```
                                        output_ptr++;
                                        input_ptr++;
                                        }

                        return energy;
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return -1.0;
        }

}
```

Report Activity

```
public class Report extends Activity {

        TextView RecordingTime,Sevirity, AHI,Supine,Movements, Result;
        double message2;
        double SupineCounter, MovementsCounter, PreviousValue, CurrentValue, SupinePercentage, MovementsPercentage;



        @Override
        protected void onCreate(Bundle savedInstanceState) {
                // TODO Auto-generated method stub
                super.onCreate(savedInstanceState);
                //get the recording period using intent:
                                        Bundle extras = getIntent().getExtras();
                                        message2 = extras.getDouble("recTime");
                setContentView(R.layout.report);
                ViewTextHandler();

                Result.setText(Diagnose.result);

                AccelerometerAnalysis();
                message2=roundTwoDecimals(message2);
                // String.format("%.2f", AHI);
                //Displaying the values:
                double NoH=roundTwoDecimals(Diagnose.NumberOfHours);
                RecordingTime.setText(Double.toString(NoH)+ " h");
                AHI.setText(Double.toString(message2));

                if(Diagnose.Severity2==null)
                        Sevirity.setText("N/A");
                else
                        Sevirity.setText(Diagnose.Severity2);

                SupinePercentage=(SupineCounter/(MainActivity.accLIST.size()-2))*100;//this equation divides the number of movement
                if(SupinePercentage==0.0)
                        Supine.setText("No");
                else
                {
                        SupinePercentage=roundTwoDecimals(SupinePercentage);
                        Supine.setText(Double.toString(SupinePercentage)+"%");
                }


                MovementsPercentage=(MovementsCounter/(MainActivity.accLIST.size()-2))*100; //this equation divides the number of
                if(MovementsPercentage==0.0)
                        Movements.setText("No");
                else
                {
                        MovementsPercentage=roundTwoDecimals(MovementsPercentage);
                        Movements.setText(Double.toString(MovementsPercentage)+"%");
                }

        }
        private void AccelerometerAnalysis() {
                // TODO Auto-generated method stub
                PreviousValue=MainActivity.accLIST.get(1);
                for(int i=2; i<MainActivity.accLIST.size(); i++)
                {
                        CurrentValue=MainActivity.accLIST.get(i);
                        if(Math.abs(CurrentValue-PreviousValue)>5.0)
```

```
                                {
                                        MovementsCounter++;//MovementsCounter holds the number of seconds for Movements
                                }
                                else
                                {
                                        SupineCounter++;//SupineCounter holds the number of seconds for Supine position
                                }
                                PreviousValue=CurrentValue;
                                }//end for loop

                        }


        private void ViewTextHandler() {
                        // TODO Auto-generated method stub


                        AHI = (TextView) findViewById(R.id.textView7);
                        RecordingTime = (TextView) findViewById(R.id.textView1);
                        Sevirity=(TextView) findViewById(R.id.textView10);
                        Supine=(TextView) findViewById(R.id.textView8);
                        Movements=(TextView) findViewById(R.id.textView9);
                        Result=(TextView) findViewById(R.id.tvRes);
        }
        double roundTwoDecimals(double d)
        {
           DecimalFormat twoDForm = new DecimalFormat("#.##");
           return Double.valueOf(twoDForm.format(d));
        }


}
```

Matlab Code

```
event=0;
[x, fs] = wavread('C:\2.wav');% to read the wav file. x:includes the samples, fs: sample frequency number of samples per unit of time (usually seconds)

Ts=1/fs;%  sampling period or sampling interval, which is the time between samples.
N = length(x);% N is the number of samples
slength = N*Ts;%slength is the length of the sound file in seconds


%Calibration phase: find the average energy in the first 30 seconds:

% Reading the first 30 seconds
calibration_samples = 30 * fs;% first 30 seconds
[x2, Fs] = wavread('C:\2.wav', calibration_samples);
Tss=1/Fs;
% 2)filtering the frequency to be only between [200-800] Hz
[b,a]=butter(5,[200 800]/(Fs/2));
y2=filter(b,a,x2);


%This loop is to find the average energy of the samples for the first 30 seconds
startSample=1;
endSample=Fs;
energy=0;
 for i=1:30

   energy=energy+sum(y2(startSample:endSample).^2)*Tss;
   startSample=endSample+1;
   endSample=endSample+Fs;

 end
 mean_energy=energy/30;
 Reference_Energy=mean_energy;% this is a reference energy level for the patient. This value is different among individuals.
 Threshold=0.9*Reference_Energy;


% Now filtering the whole recorded file to be between [200-800] Hz
[b,a]=butter(5,[200 800]/(fs/2));
y=filter(b,a,x);
N = length(y);
N=N/fs; % how many iteration we need
% Algorithm: in each second, the energy is calculated, and compared with the
% threshold value. if (energy<=threshold), then we need to check the next 9
% seconds, because apnea event is detected when the value of the energy is less than the threshold for at least 10 seconds
```

```
startSample=1;
endSample=fs;
energy=0;
j=1;
 while( j<=N)
   counter=0;
   energy=sum(y(startSample:endSample).^2)*Ts;

   if (energy<=Threshold)
      counter=counter+1;

      for k=1:10

         startSample=endSample+1;
         endSample=endSample+fs;
         energy=sum(y(startSample:endSample).^2)*Ts;

           if (energy<=Threshold)
              counter=counter+1;

           else
              break;
           end %end inner if
      end % end inner for



   end % end outer IF
    if(counter>=10)% If the counter is more than or equal to 10 then an event is detected
      event=event+1;

    end
   if(counter>0)
      j=j+counter;% Increment by the counter value, because if we entered the inner loop then we will move in the sound samples, and this is represented
in the counter value.
   else
      j=j+1;
   end
   startSample=endSample+1;
   endSample=endSample+fs;

 end % end outer For
```

Vita

Mamoun Al-Mardini was born on May 13, 1986, in Jerash, Jordan. He was educated in local public schools and graduated from the Model High School, Irbid, in 2004.

Mr. Al-Mardini graduated from the Jordan University of Science and Technology in Irbed, Jordan, in 2009. His degree was a Bachelor of Science in Computer Engineering.

In 2009, Mr. Al-Mardini worked as a customer support engineer at Cisco Technical Assistance Center in Amman, Jordan. In 2011, he joined the American University of Sharjah (AUS) to pursue his M.S. degree, and was granted a graduate teaching assistantship. He was awarded the Master of Science degree in Computer Engineering in 2013. Mr. Al-Mardini is a member of the Jordanian Engineers Association.