Design Project II (CMP/COE 491)

Project Report

An Intelligent Mattress for Diagnosis of Sleep Apnea

American University of Sharjah
Name 1: German Shein (48610)
Name 2: Sankar Sathyanarayanan (48966)
Section: 01
Advisors: Dr. Michel Bernard Pasquier and Dr. Assim Sagahyroon
Date of submission: 12.12.2016 (Fall 2016)

Dr. Michel Bernard Pasquier: _____
Dr. Assim Sagahyroon: _____
German Shein: _____
Sankar Sathyanarayanan: _____

Abstract

Sleep apnea is one of the main concerns of the middle-aged people as it negatively impacts their sleep quality, health in general, and can go undetected until serious damage is done [1]. The purpose of this project is to build a system that allows the users to diagnose this disorder in the home environment with minimal physical attachments and without body invasive measurements. The hardware components of the system are various physiological measurement sensors. These sensors include but are not limited to heartbeat, body temperature and weighing sensors that are connected to microcontrollers. All of the hardware is placed inside the mattress and sends the data to the server. The server will use pulse, breathing recordings and lungs pressure to diagnose sleep apnea. Finally, the system is supposed to store the obtained measurements and diagnosis in a database. The conducted research, literature review and the design description are presented in this report.

Acknowledgements

We would like to thank Dr. Michel Bernard Pasquier and Dr. Assim Sagahyroon for their guidance and support of our work. Our gratitude also extends to the professors from Computer Science & Engineering, and Electrical Engineering departments as well as Psychology professors from Department of International Studies of the American University of Sharjah. Finally, we would like to thank our parents for their endless support through this trying tine.

Table of Contents

List of Figures

1. Introduction

1.1. Background

1.1.1. Sleep Apnea

Sleep Apnea is a sleep disorder, the main symptom of which is the shallow breathing or its complete cessation during sleep for short periods. This disorder has two major types: Obstructive Sleep Apnea (OSA) which is characterized by the blocking of the upper airway and Central Sleep Apnea (CSA) which is characterized by the stopping of the breathing efforts. As a result, affected people suffer from a disturbed sleep and their bodies do not get enough oxygen during a night. Specifically, they have the following symptoms: snoring, sleep deprivation, sleepiness and oxygen saturation. Later on, these symptoms can lead to problems with heart, glucose metabolism and even mental health [1] [2].

Dire consequences that sleep apnea leads to might be a major concern for affected people. They start to seek help from sleep disorder specialist once some of the symptoms become apparent. Among other disorders, a medical professional would check for sleep apnea. The procedure of diagnosis involves sleeping in a hospital environment. In general, there are three ways to detect sleep apnea. The first one detects a lack of oxygen saturation in patient's blood using pulse oximetry [3]. The second one that uses complex process of polysomnography [4]. The third way is an experimental, yet simpler one that detects intensifications of patient's heartbeat rate using electrocardiogram [5].

1.1.2. Pulse Oximetry

Pulse oximetry is a method of monitoring oxygen saturation in patient's blood. If user's oxygen saturation drops by 4 % and goes below 90 % level, then user is having a hypopnea episode [6]. Thee method is non-invasive, but requires wearing a measuring device. The main advantage of this method is that it can be performed at home (with proper setup). However, it requires wearing a device that constantly applies pressure to user's finger [3]. This might prove distracting and affect the results since patient is under the risk to sleep less because of sensory irritation. On top of that, some researchers do not consider it as a more formal method – polysomnography.

1.1.3. Polysomnography

Polysomnography – a complex analysis of sleep that involves the following processes: Electroencephalography (electrical activity of the brain), Electrooculography (electrical activity between the back and the front of the human eye that results from eye's movements), Electromyography (electrical activity of the skeletal muscles) and Electrocardiography (electrical activity of the heart). It is used for a wide range of sleep disorders, not just sleep apnea. While it is very precise, it is impossible to conduct it at home and it presents patients with more sensory irritation that the pulse oximetry as it requires

attachment of 16 to 24 (depends on the desired precision of electrocardiogram) electrodes. It also requires the presence of medical professional nearby [7]. The analysis of the results is also a problem, because some time will be spend ruling out other disorders.

1.1.4. Electrocardiogram

Electrocardiogram (ECG) – a graphical representation of relationship between heart's electrical activity and time. Typically, there are 9/10 regions (sometimes, one of the legs is omitted) where electrodes are placed [8]. Figures 1 and 2 depict the precise locations where the electrodes should be placed.

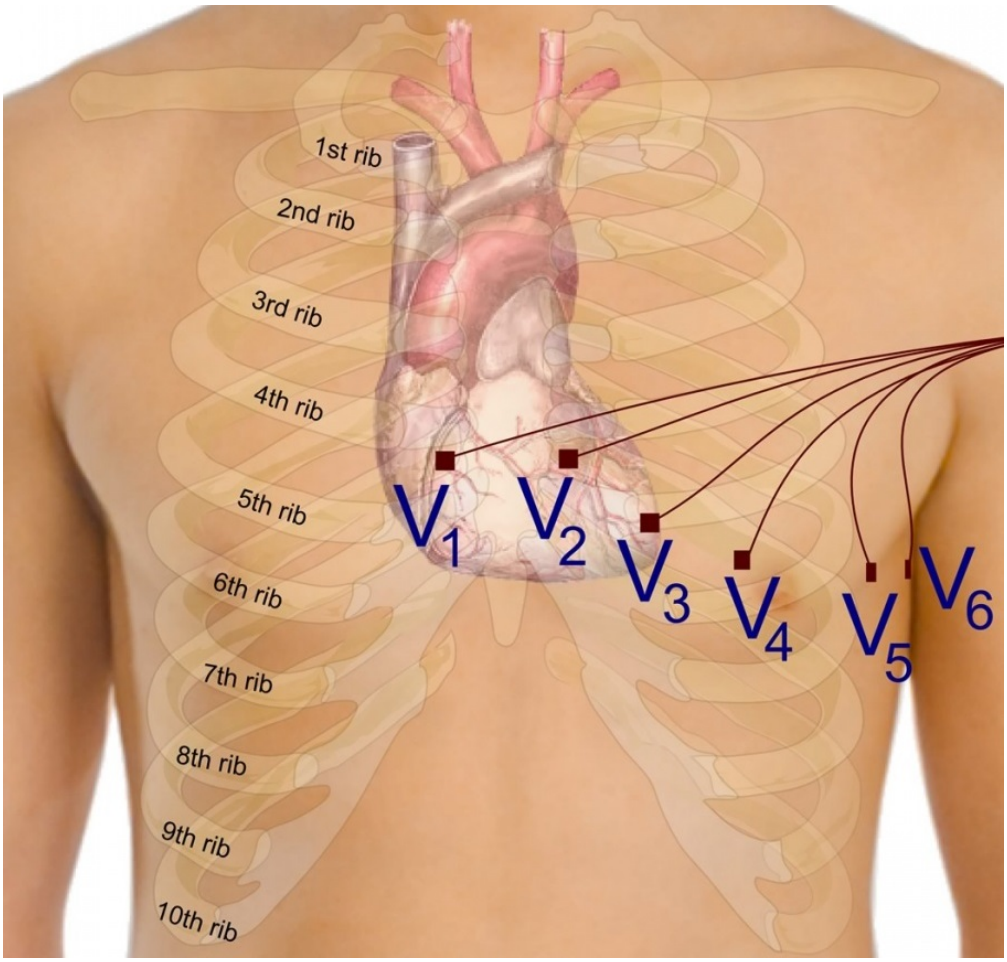

Figure 1. ECG Electrodes Placement in the Chest Region [9]

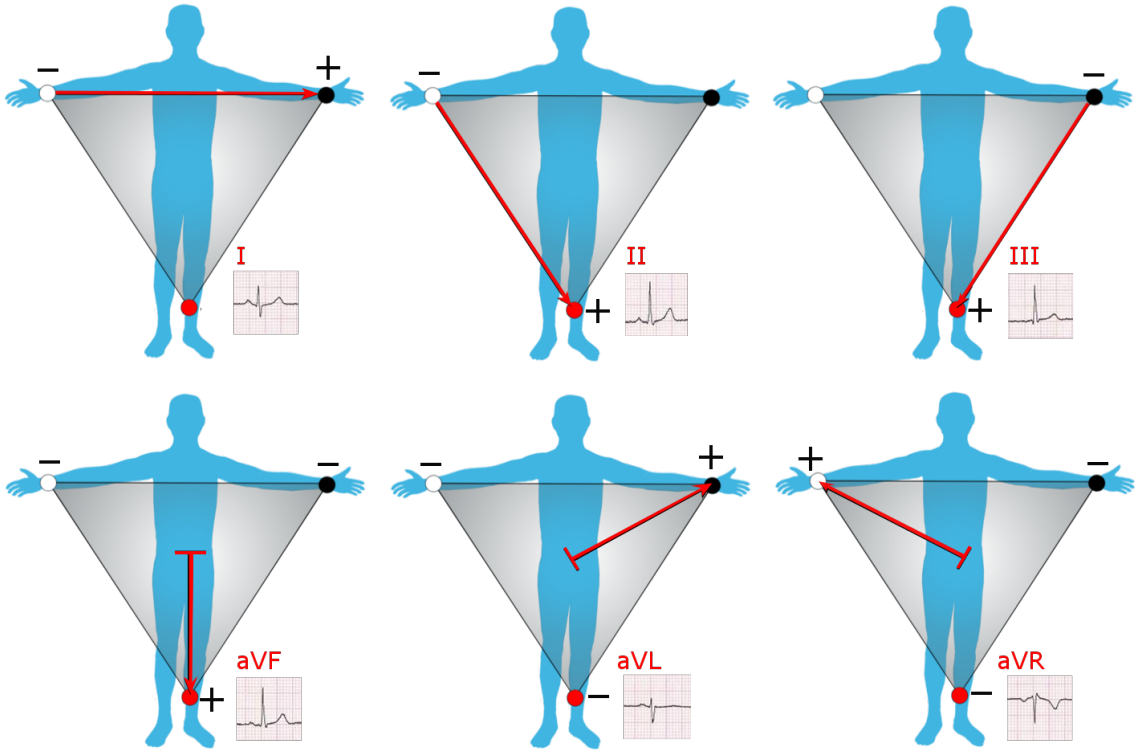

Figure 2. ECG Electrodes Placement on the Limbs [10]

This method is simpler compared to polysomnography. It still presents the same issues, yet reduces the number of connections. Mendez, Bianchi, Matteucci, Cerutti and Penzel have shown that it is an effective diagnostic method. Unfortunately, ECG still requires attachment of sensors to really precise positions of a human body. Even one such placement constricts the movement of a person, making sleep uncomfortable and the system obstructive as figures 1 and 2 demonstrate.

1.1.5. Pulse

The fact that sleep apnea affects the electrical activity of the heart implies that heart's functions at large is affected as well. And indeed, Kamath et al. [32] found that the pulse of person with sleep apnea is increased. Therefore, pulse readings can be used to diagnose the condition. Figure 3 [27] shows all regions where pulse can be measured:

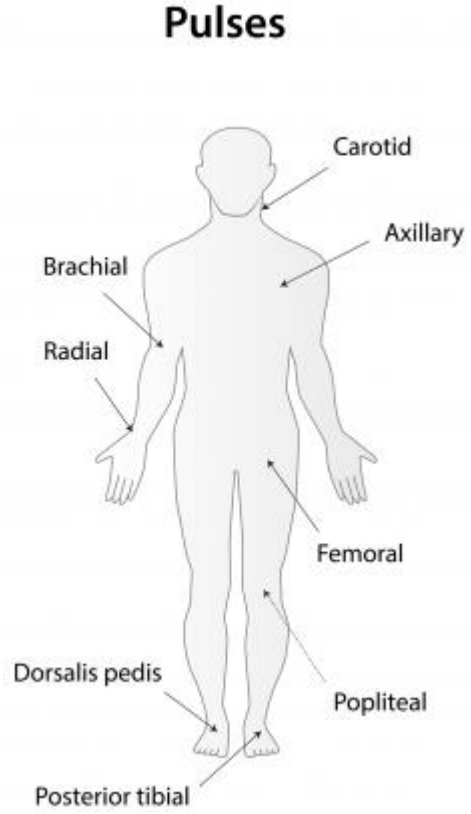

Figure 3. Pulse Measurement Body Regions

Usually, radial, and carotid pulses are measured [35]. However, there are other regions from which the pulse can be collected such as fingertip and earlobe. Though, it is not the pulse that is measured, but rather the intensity of the blood flow from which the heart rate is derived.

The provided information shows that unlike other methods, pulse measurement is extremely flexible as it can be performed from multiple body points. This fact was utilized in our project.

1.1.6. Sleep Stages

There are 4 general sleep stages that humans go through over the course of one sleep: non-rapid eye movement, slow-wave sleep (sometime included within non-rapid eye movement), rapid eye movement and awakening. These stages are usually cycled over the course of sleep with each cycle lasting around 90 minutes. The table below presents the short characterization of each stage as well as their role in identifying sleep apnea:

Stage	Description	Importance	Physiological Characteristics
Non-rapid eye movement (NREM) sleep	The calm sleep stage that serves the purpose of restoring body's resources.	Has a higher apnea-hypopnea index than REM	Decrease of body temperature, heart rate and breathing rate, muscles are allowed to move
Slow-wave sleep (SWS)	Calmest, deepest and most restorative sleep stage	Has a higher apnea-hypopnea index than REM, snoring starts to occur	Decreased response to reflexes.
Rapid eye movement (REM) sleep	Dreaming stage, the most unstable one.	Events of hypopnea and snoring take place at this stage	Decreased reflexes and muscle activity (except respiratory and eye ones), rapid eye movement, unstable heart rate and breathing rate
Awakening	The short stage, body is restored to its normal working mode	Usually comes after REM. Therefore, the last REM during the whole sleep is easier to identify	All characteristics return back to normal

As the table demonstrates, REM sleep stage is the most appropriate to determine whether the person has sleep apnea. Not only the most discernible symptoms appear at this stage, but also the user is also not making any movements.

1.2. Situation

In the USA alone sleep apnea affect around 18 million people. This means roughly one in every 15 Americans suffers from this disorder. Sleep apnea increases the risk of hypertension, stroke and diabetes. In the case of women, it also increases the risk of heart failure. This chronic illness must be treated to protect the sufferer from the potentially more harmful complications that can arise otherwise [13].

Unfortunately, simply detecting sleep apnea is not an easy task by itself. First of all, one has to suspect that he/she has sleep apnea. Secondly, the diagnostic techniques are inconvenient and very few people in our dynamic world are devoted to their health enough to spend several nights at the controlled environment. This situation calls for a convenient solution that can satisfy the inhabitants of today's busy world. For example, researchers at University of Victoria, Canada have designed an intelligent bed, one of the purposes of which is to detect sleep apnea and presented it in "Towards an Intelligent Bed Sensor: Non-intrusive

Monitoring of Sleep Irregularities with Computer Vision Techniques" article [14]. They used optical pressure sensors to monitor movements, breathing and sleep apnea. Their idea influenced our project, yet alternative methods were picked for diagnosis.

1.3. Purpose

An Intelligent Mattress for Diagnosis of Sleep Apnea project aims to create a precise, convenient and mobile diagnosis system that can be used in a home environment by a user without any medical knowledge or experience. The project, in an essence, is a mattress with installed microphones, body temperature, heart rate, pressure, motion and moisture sensors as well as microcontrollers. Additionally, the system is required to be non-obtrusive and non-invasive with no physical attachments. Specifically, all our sensors that are located in the mattress must not be physically wired to a body and must not penetrate it while the user is lying on a mattress. In other words, system's user has to be able to use the system like a regular bed. The challenge presented by the described purpose lies in the fact that none of the diagnostic tools fit the criteria, even the pulse detection. Therefore, users have to put their necks on a designate spot to have their pulse measured. In addition to pulse sensors, microphones will be placed to record breathing and detect snoring or hypopnea during sleeping period. These events will also be confirmed through the pressure sensors since the pressure on the bed will increase. It will serve as an additional diagnostic criterion.

1.4. Scope

The project requires a combined effort of various scientific and engineering disciplines including biomedical engineers, computer engineers and scientists as well as medical doctors. Given the fact that our team possesses only the representative of the last two disciplines, we are required to make additional bioengineering/medical research. All of the chosen components were carefully considered in terms of cost-efficiency, necessity and system robustness. Current expenses include only the prices of the components. The computer engineer of the team concentrated mainly on designing the hardware of the system, while the computer scientist concentrated on the software part.

2. Problem Statement

It is not easy to detect sleep apnea and most sufferers do not realize they have it as doctors can’t diagnose this during routine checkups due to the fact that the symptoms show up in middle of the sleep. There is a need to analyze a person for potential signs of this disorder to avoid these complications. This cannot be done manually as person would have to supervise the patient during his/her sleep and watch out for the potential symptoms like short shallow breaths or snoring. There is a requirement for a device that can substitute a human being and tirelessly monitor a person for possible signs of this illness and report the results when the person wakes up in the morning.

3. Literature Review

3.1. Towards an Intelligent Bed Sensor: Non-intrusive Monitoring of Sleep Irregularities with Computer Vision Techniques

Author(s): Kaveh Malakuti, Alexandra Branzan Albu

The paper reviews one of the previous concepts of the intelligent beds. Like ours, it is non-intrusive and non-invasive. It proposes an approach involves processing of nonvisual input data (pressure signals) with computer vision techniques. Sleep irregularities induce variations in the periodicity of the studied signals. These variations are represented as visual patterns in the inter-frame similarity matrix. The main goal here is the successful application of computer vision techniques for processing non-visual data. However, it uses extensive image processing as it relies on optical pressure sensors. Unfortunately, while their approach certainly fits the criteria, we cannot use it as the diagnosis precision is not the best [14].

3.2. A framework for screening and classifying obstructive sleep apnea using smartphones

Author: Mamoun Al-Mardini

This paper served as the main reference source during our initial research. It directed us towards using microphones and motions sensors in our project. We may also use the author's approach to detect snoring. It also explored the use of the pulse oximetry as a diagnostic method, yet we discarded it due to being obtrusive. The author used Apnea/Hypopnea Index as a measurement of the severity of the disease which is a pretty definite measurement, yet we might not take it as we consider analyzing pulse and breathing patterns which have more ambiguous interpretations [15].

3.3. Design and Development of a Heart Rate Measuring Device using Fingertip

Author(s): M.M.A. Hashem, Rushdi Shams, Md. Abdul Kader, Md. Abu Sayed

This paper discusses the design and development of low power Heart Rate Measuring (HRM) devices. The techniques use optical monitoring to measure heart rate. The paper gives us an overview of a proposed HRM system and its pulse detection aspect. What makes this relevant to our project is that the techniques presented in this paper can help us make a heart rate monitor that can be used at home. This makes it ideal for the bed. It proposes an optical method to detect heart rate which will help us develop a heart rate monitor for just $20 using just LED's and photo sensors.

The pulse detection uses a heartbeat detector which consists of a pulse sensing unit and a display unit. The design and development of a heart rate measuring device is presented that measures the heart rate efficiently in a short time and with less expense without using time consuming and expensive clinical pulse detection systems. The device is able to detect, filter, digitize, and display the heartbeat of a user ergonomically and is proven to show results consistent with medical sensors as the graph below demonstrates [16]:

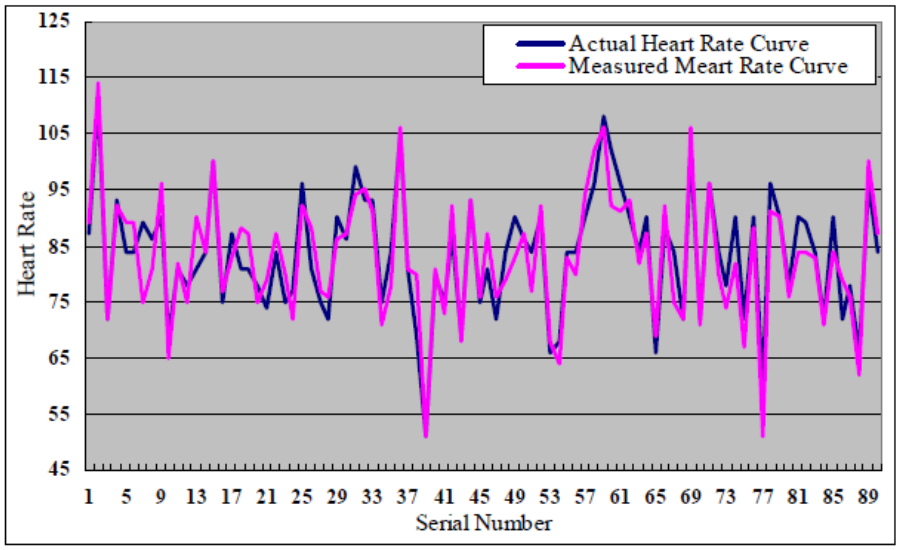

Figure 4. Comparison between Medical and Commercial Heart Rate Sensors [17]

3.4. Sleep Apnea Screening by Autoregressive Models From a Single ECG Lead

Author(s): Martin O. Mendez, Anna Maria Bianchi, Matteo Matteucci, Sergio Cerutti, Thomas Penzel

The authors investigate the use of electrocardiogram (ECG) as a diagnosis tool for obstructive sleep apnea. They looked at RR, QCG intervals and respiratory patterns. The main discovery that is useful to our project is that people with sleep apnea tend to have an ECG with greater amplitudes at the mentioned area and severely dropped breathing with period of its almost complete cessation. The bivariate time-varying autoregressive model, proposed in the study conducted by the authors of this paper to analyze the interrelation of time series during sleep enables fast computation, simple implementation, high signal variation adaptability, high resolution, and extraction of features with large classification power between normal and apneic sleep periods which will go a long way in helping us detect sleep apnea. These findings have influenced our decision to use ECG as our primary diagnostic method [18].

3.5. Apnea Duration and Hypoxemia During REM Sleep in Patients with Obstructive Sleep Apnea

Author(s): Larry J. Findley, Stephen C. Wilhoit, Paul M. Suratt

Findley, Wilhoit and Suratt have made several important considerations in their research. First, they had a control group of health people in their experiments, which is essential to demonstrate that the results are actually valid and differ from the diagnosed patients. Secondly, they looked at lengths of apnea episode in NREM and REM sleep stages. Their evaluation criterion was the oxygen saturation of the patients. Even though, it is different from the previously considered research, taking into account alternative tools is always important, especially since ECG diagnosis is still in experimental phase. The results ultimately show that episodes of apnea (any sleep apnea type) are longer and oxygen saturation is more severe [19].

3.6. Is obstructive sleep apnoea a rapid eye movement-predominant phenomenon?

Author(s): J. A. Loadsman, I. Wilcox

Loadsman and Wilcox arrive at a conclusion that while patients with OSA tend to show their symptoms during REM sleeping phase the correlation seems questionable. They used oxygen saturation as their diagnosis criterion. The importance of this article lies in the fact that it directed us to look at other sleep stages. If currently considered ECG diagnostic technique fails, it is always possible to go back to the oximetry-based diagnosis [20].

3.7. Heart Rate Variability (HRV) Signal Analysis: Clinical Applications

Author(s): Markad V. Kamath, Mari A. Watanabe, Adrian Upton

Detecting sleep stages is an integral part of our project as heart rate varies between sleep stages. To carefully detect sleep apnea, we need to know how heart rate varies with respect to sleep stages and this book gave us this information. Heart rate decreases in general during NREM sleep with respect to resting heart rate and during REM sleep it may go above resting levels. Transition from REM to NREM is characterized by rapid increase in heart rate as well. This information was useful to us in detecting sleep stages [32].

3.8. Comparison of pulse rate variability with heart rate variability during obstructive sleep apnea

Author(s): Ahsan H. Khandoker, Chandan K. Karmakar Marimuthu Palaniswami

This paper was used by us to analyze the heart rate of a patient during NREM. The study based on which this paper was written involved collecting readings from both healthy and sleep apnea patients overnight. Based on the results of this study we learnt that the mean heart rate of sleep apnea is around 75 ± 12 beats per minute. We used this threshold to analyze the heart rate recordings for sleep apnea [33].

3.9. Comparison of Detrended Fluctuation Analysis and Spectral Analysis for Heart Rate Variability in Sleep and Sleep Apnea

Author(s): Thomas Penzel, Jan W. Kantelhardt, Ludger Grote, Jörg-Hermann Peter, Armin Bunde

This paper was used to determine the mean heart rate of sleep apnea patients during REM sleep. The study based on which this paper was written involved collecting readings from 14 healthy subjects, 33 patients with moderate, and 31 patients with severe sleep apnea. Based on the results of this study we could estimate that the mean heart rate for sleep apnea patients during REM sleep to be around 60 to 80 beats per minute. We used this threshold to analyze the heart rate recordings for sleep apnea [34].

4. Requirements Specifications

4.1. Functional requirements

The system's primary purpose is to determine whether a person has sleep apnea or not. The list below breaks this purpose down into more specific goals that have to be achieved if successful performance is desired:

1) A user should be able to sign up/login into the system through the mobile application.
2) The system should diagnose sleep apnea:
    a) The system should take the following physiological measurements of a user lying on a mattress:
        I) Pulse.
        II) Body temperature.
        III) Intensity of sweating.
        IV) Intensity of breathing (determined through the measurement of the lung's movement pressure on the mattress).
        V) Snoring and/or breathing cessation episodes.
        VI) Spontaneous body movements.
    b) The system should send the measurements to a server.
    c) The system should understand whether the user is asleep and distinguish all sleep stages a user undergoes.
    d) The system should use the readings to determine which sleep apnea symptoms a person demonstrates.
    e) The system should store the measurements and the result in the database.
3) A user should see the diagnosis and relevant medical information through a mobile application.

4.2. Non-functional requirements

The point of any commercially available is not only about whether it performs its duty but also about how it performs it and how is presented. Nobody will use computers that require an hour to launch an operating system. Likewise, our system needs not only to diagnose sleep apnea, but also make user's interaction with it easy. Therefore, our system should be:

1) Accessible – the system is related to every human's basic need of sleeping and almost every human being uses mattresses.
2) Comfortable – users have to be able to sleep on the modified mattress and not feel the difference between it and a regular mattress.
3) Fault tolerant – If something goes wrong the system still has to be able to display error messages or abort if necessary.

4) Fire resistant – the mattress is created from generally inflammable textile and the hardware components are under the risk of overheating due to prolonged time of operation (8 hours per day on average).
5) Non-invasive – the sensors must not penetrate human body in any way. The invasive procedures are painful and might damage the sleep quality.
6) Non-obtrusive – the sensors can only interfere with the body through the mattress. Specifically, as a user lies down the sensors come in contact with the body. However, they are not going to be attached and should not be detected by user, since it leads to sensory irritation which in turn damages sleep quality.
7) Reliable – the diagnosis and measurements have to be accurate since personal health is a serious issue and any false diagnosis can lead to financial losses and damage to mental and emotional well-being of a user.
8) Robust – the system should deal with garbage data and overloading of CPU and microcontrollers.
9) Secure – users' data has to remain confidential due to legal and ethical reasons.
10) Usable – the system will be used by people who are not guaranteed to be tech savvy (like senior citizens), therefore the system should be as simple as possible to learn. In theory, they should have the basic skills of operating with smartphone such as opening the apps and going through user authorization.

4.3. Design Objectives

1) The system has to take the following measurements: heart beat rate, body temperature, lungs' pressure. Additionally, it has to detect unexpected movements and if patient is sweating. All sensors have to collect the data without physical attachments and body-invasive procedures.
2) The system has to correctly identify the sleep stages. The software can only operate based on the raw data collected by sensors. Moreover, this data has to be filtered against noises and irrelevant measurements.
3) The system has to detect the sleep apnea. The sensory information will fed to diagnostic software. It will look at extreme levels of physiological readings and treat them as symptoms of the disorder.

4.4. Use-case Diagrams

The functionality of our system can be represented in three use-case diagram. The first diagram shows the gist of setup's operation: user lies down on a mattress, thus providing data for diagnosis, which is sent to a mobile application by the end of the night:

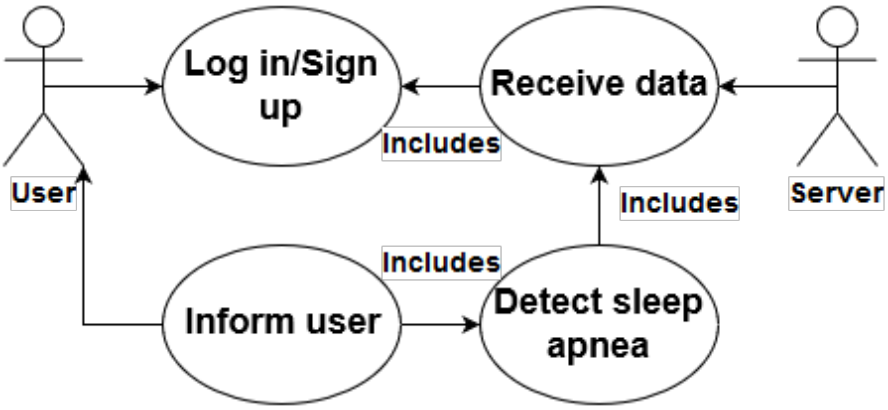

Figure 5. Use-case Diagram 1

The second use-case diagram shows operation of hardware in details. Basically, once the user lies down on a mattress, the pressure sensors will detect that there is user to collect data from (for simplicity, this little step is avoided in the diagram) and microcontroller starts the process of data collection. The collected data is sent to a server for evaluation/diagnosis:

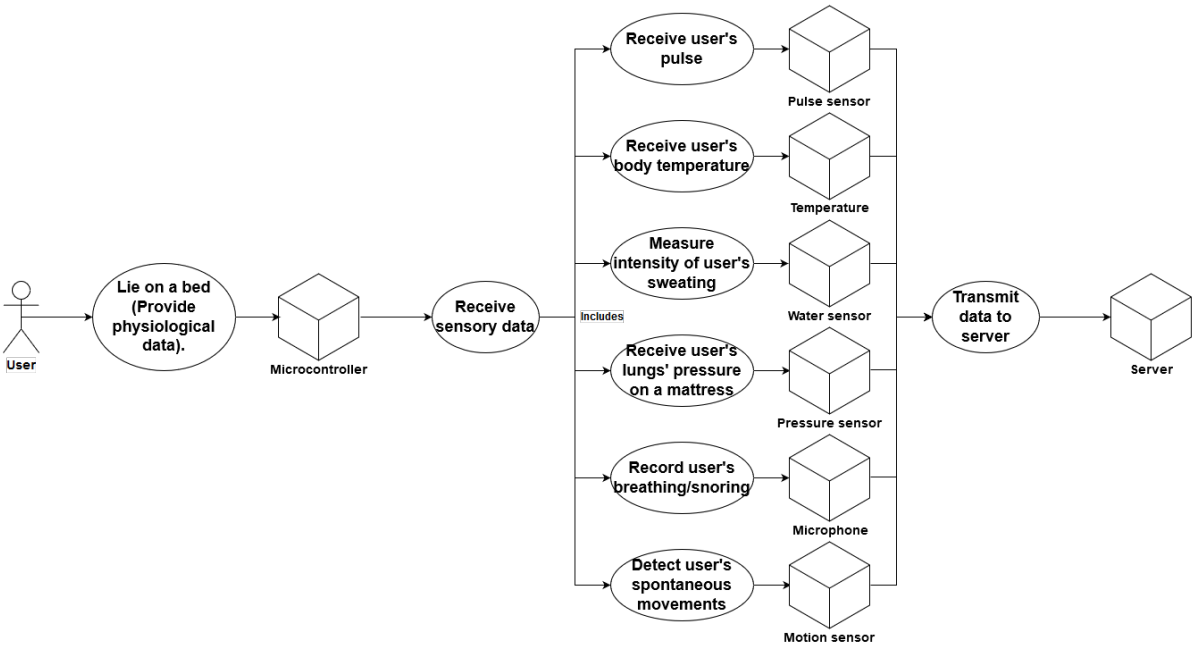

Figure 6. Use-case Diagram 2

The final use-case diagram (figure 7) shows every component of the system and each one's main responsibility. User only interacts only with mattress and a website. The first one receives data, the second gets the results.

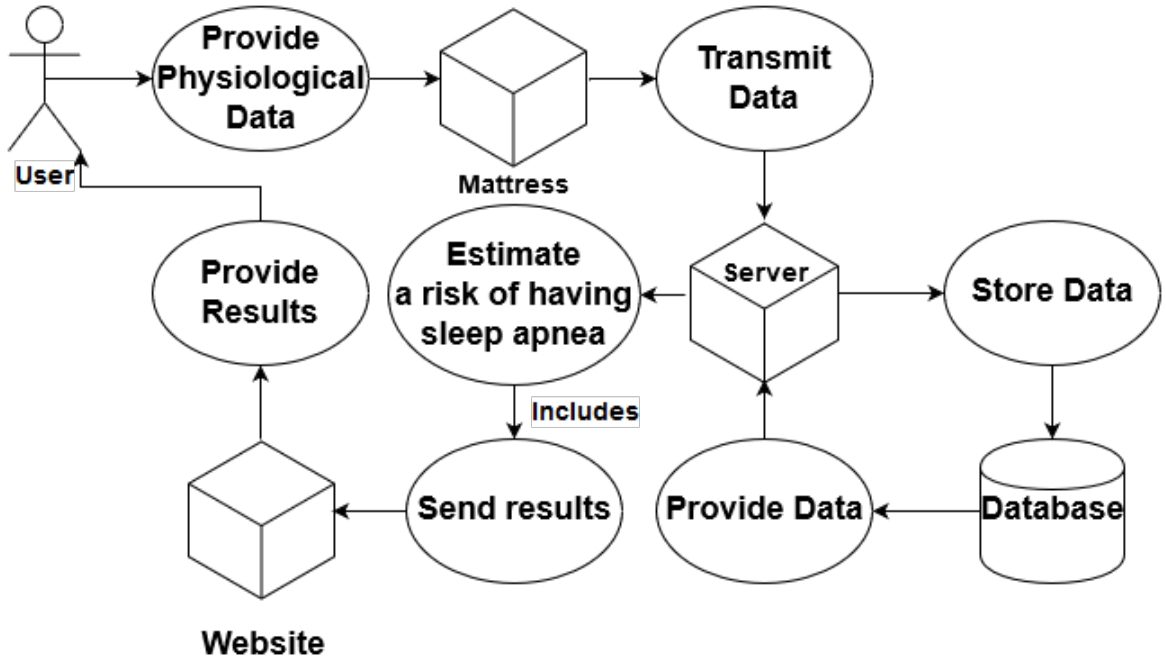

Figure 7. Use-case Diagram 3

5. Technical approach and Solution

5.1. System Overview

At the first session with the system, users have to create an account that they will log into every time before using the system. During the sleep, sensors in bed take necessary physiological measurements. The microcontrollers send the data to the server to be evaluated and stored once the diagnosis is produced. At the end of the night, the server performs the following actions:

1) Determines when the user fell asleep and woke up.
2) Breaks the sleeping period into sleep stages based on body temperature (helped by measurement of sweating intensity) and pulse changes.
3) Picks the REM sleep stages (the stage during which sleep apnea's symptoms can be observed) to detect the following symptoms of sleep apnea:
    a) Common between Central Sleep Apnea and Obstructive Sleep Apnea:
        I) Radical heart rate increase.
        II) Cessation of lung movement.
    b) Obstructive Sleep Apnea-specific – snoring.
    c) Central Sleep Apnea-specific – shallow or non-existent breathing.
4) Based on the quantity of the symptoms, the risk of having sleep apnea is calculated.
5) The evaluation is sent to a website.

Figure 8 shows the algorithm through a flow chart.

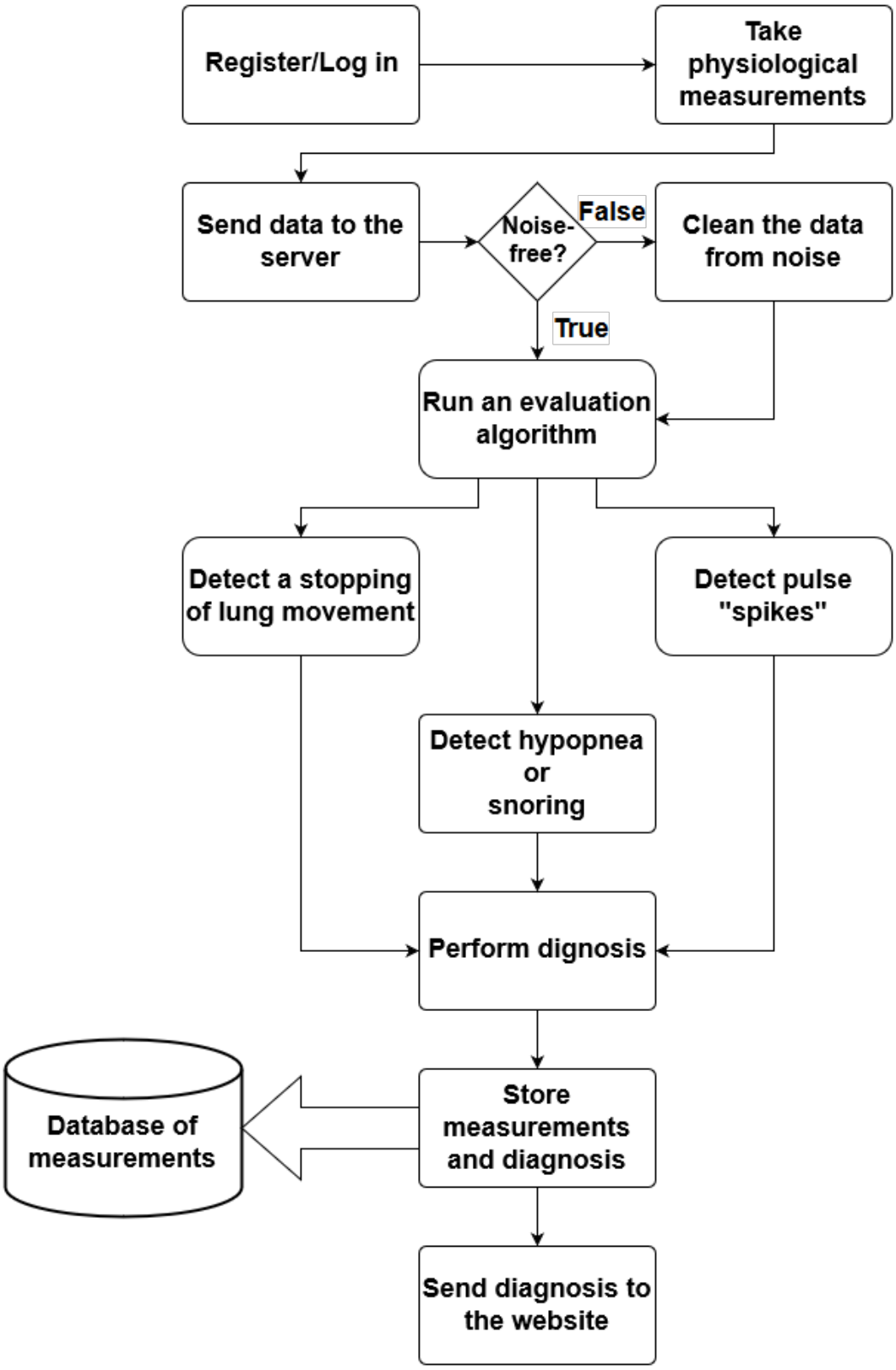

Figure 8. System's Flow Chart

The system has 3 key components:

1) Website – entry and leaving point, a way to distinguish users and inform them.
2) Mattress – the main measuring equipment.
3) Server – the main data processor.

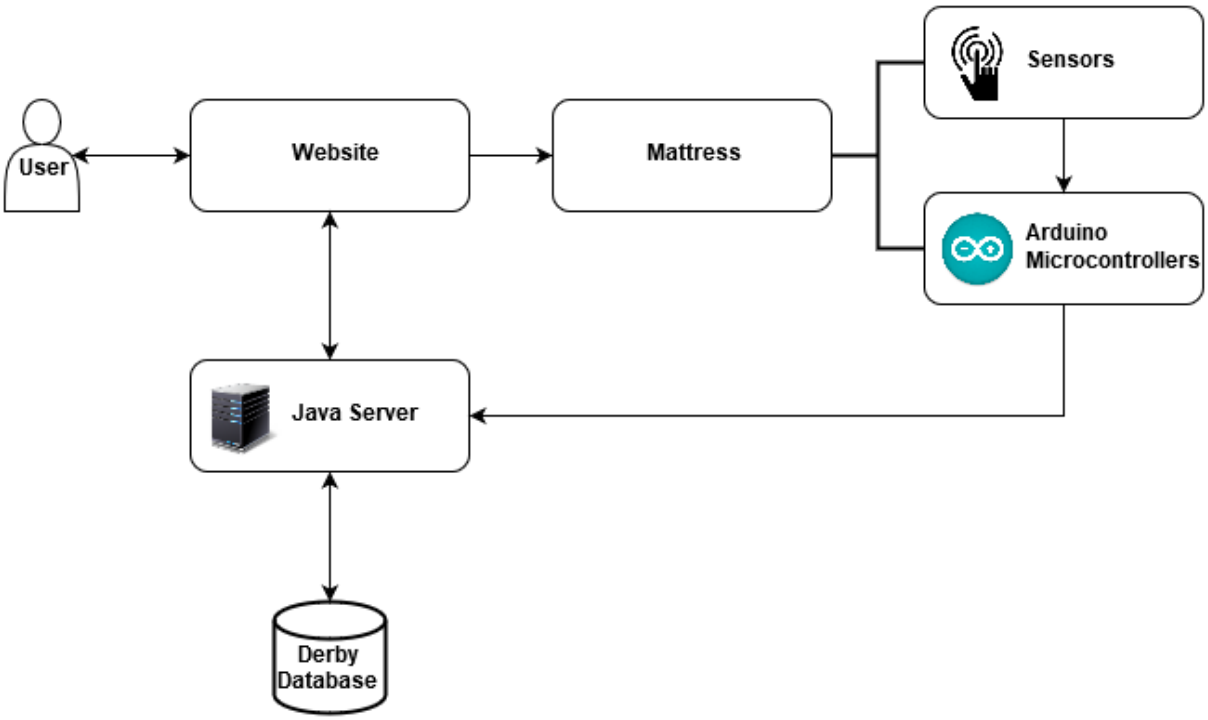

Figure 9. System's Architecture

Figure 9 shows the interaction of these components. Technically, it is just a more abstract representation of a flow chart. Keep in mind though, that there is a bi-directional communication between some components, because lots of data needs to be updated and/or presented to the user. For example, database is constantly flooded with sensory readings, while the user provides data and in return gets diagnosis.

5.2. Microcontrollers, Sensors and Mattress

Two Arduino Mega 2560 microcontrollers are used. This microcontroller was chosen for 2 reasons: it has a sufficient for our project amount of digital and analog input pins, and it is only needed to perform sensory data retrieval and sending. A Raspberry PI microcontroller family was considered. However, it was discarded because many of its components (memory, CPU, working OS) simply will not be used as they are not enough to perform sensory data processing later on.

There are several important criteria to evaluate the sleep quality and diagnose sleep apnea. Firstly, our system has to determine whether the user is asleep. Secondly, it has to associate the received data the sleeping stages. The following table shows what sensors are

required to identify each particular sleeping stage. It is also important to know that a human's brain goes through these stages several times during sleep. This fact will be considered.

Stage	Sensors
Non-rapid eye movement (NREM) sleep	Temperature sensors, heart rate sensors, microphones, motion sensors, pressure sensors, sweat sensors
Slow-wave sleep (SWS)	Temperature sensors, heart rate sensors, microphones, motion sensors, pressure sensors, sweat sensors
Rapid eye movement (REM) sleep	Heart rate sensors, microphones, motion sensors, pressure sensors, sweat sensors
Awakening	Temperature sensors, heart rate sensors, microphones, motion sensors, pressure sensors

Our system has 40 sensors in total: 32 analog and 8 digital. Sensors have the following quantities and ultimate purposes:

1) Analog:
    a) Temperature sensors (8) – measure body temperature of user to distinguish sleep stages the user undergoes.
    b) Sweat sensors (8) – acts as an indirect support to temperature sensor.
    c) Pulse sensors (6) – detect pulse increase during sleep apnea episodes.
    d) Pressure sensors (6) – detect the cessation of lung movement during sleep apnea episodes.
    e) Microphones (4) – detect snoring or breathing cessation during sleep apnea episodes.
2) Digital – Motion sensors (8) – detect spontaneous body movements to confirm that person is not in REM stage.

Microcontrollers are placed inside the mattress in order to conceal it and all important connections from a user. Sensors are placed in a direct contact with the user's body. However, the shapes of temperature and motion sensors as well as microphones are not comfortable to sleep on, so a layer of polymer around the sensors, only leaving necessary parts exposed. Additionally, there is a limited amount of body points (fingertips, wrists, neck and earlobes) from which pulse sensors can perform accurate measurements. This limitation and constraints of non-obtrusiveness and non-invasiveness force us to pick neck as a point of pulse measurement. Props were added to a mattress with pulse sensors on then and user has to lie on one of these props in order to have the pulse measure. However, we did try to give some freedom to users by creating a horizontal symmetric outline of sensor placements so that it would not matter where users chooses to put their heads. The positions of all sensors are shown in figure 10.

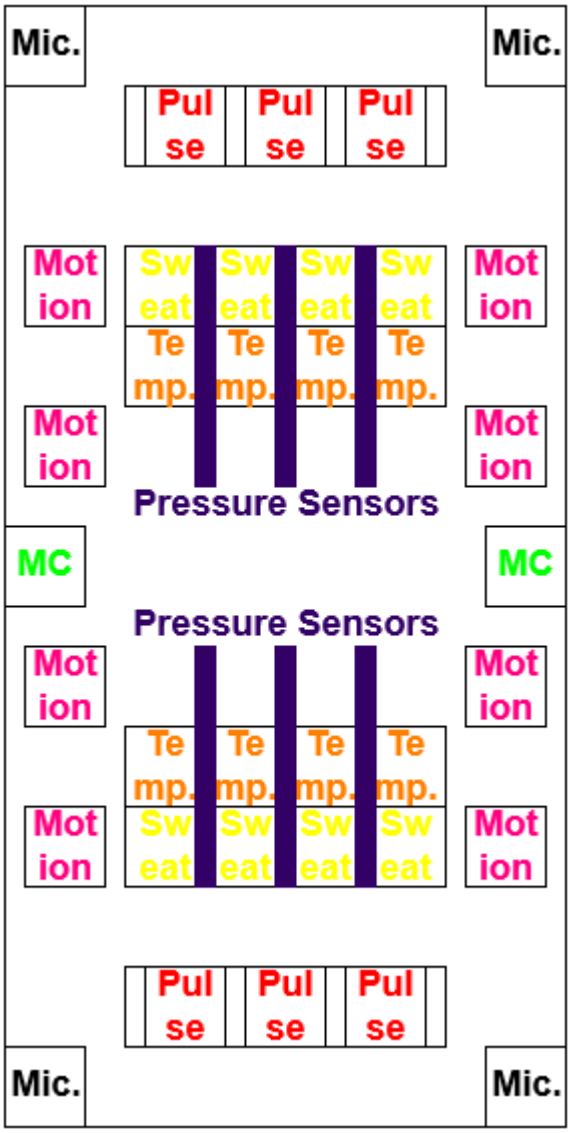

Figure 10. Components' Placement Schematic

5.3. Server

Sensory data provided by microcontrollers will be received by the server. Communication between server and microcontrollers will be established through XBee modules. The server will serve the following purposes:

1) Receiving/storing data.
2) Performing diagnosis.
3) Sending final evaluation to user.

5.4. Website

The website is a starting and finishing point of this system. Before going to bed, a user has to log into the system so that it can distinguish the data from other users. At the end of sleep, a user will be able to see the time of sleep, see if there are abnormalities in measurements and see a diagnosis or a message about being at risk of having sleep apnea. Therefore, the website serves only the purposes of user identification and conveying the information.

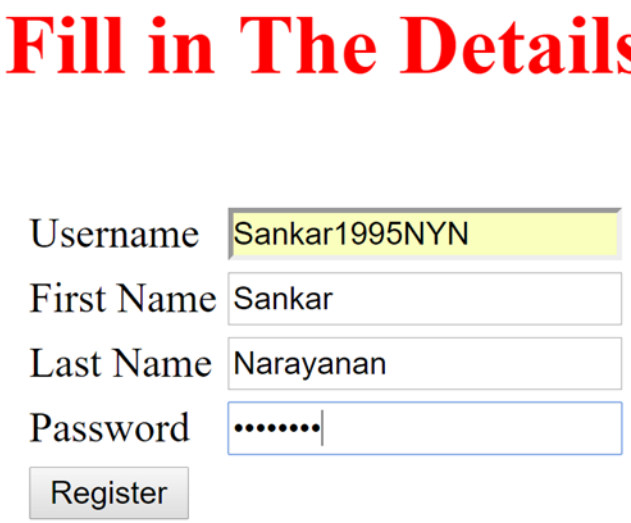

Figure 11. Registration Page

Figure 12. Login Page

Your Results

Username	Time Stamp	Diagnosis
Saurabh	2016-12-07 13:27:11.705	High Risk

Figure 13. Results Page

5.5. Database

The database consists of eight tables. Its primary role is to store the data form the sensors for later processing which will take place when the patient awakens. The patient's information is stored in the patient table. A patient is identified by his/her unique username. The pulse sensor readings are stored in heart rate, the pressure sensor readings are stored in pressures, temperature sensor readings in temperatures, moisture sensor readings in in sweats, microphone readings in sounds. In each of these tables a sensor reading is identified by the username of the user and moment (Timestamp of the moment when the reading was taken). The number of columns in the sensor tables depend on the number of sensors. In addition to this there is also a result table in which the results of all users are stored. A result is identified by the username of the patient it belongs to and moment (Timestamp of the moment when the reading was taken). The results can be viewed by the patient in the website. The schematic for the database is shown in figure 11:

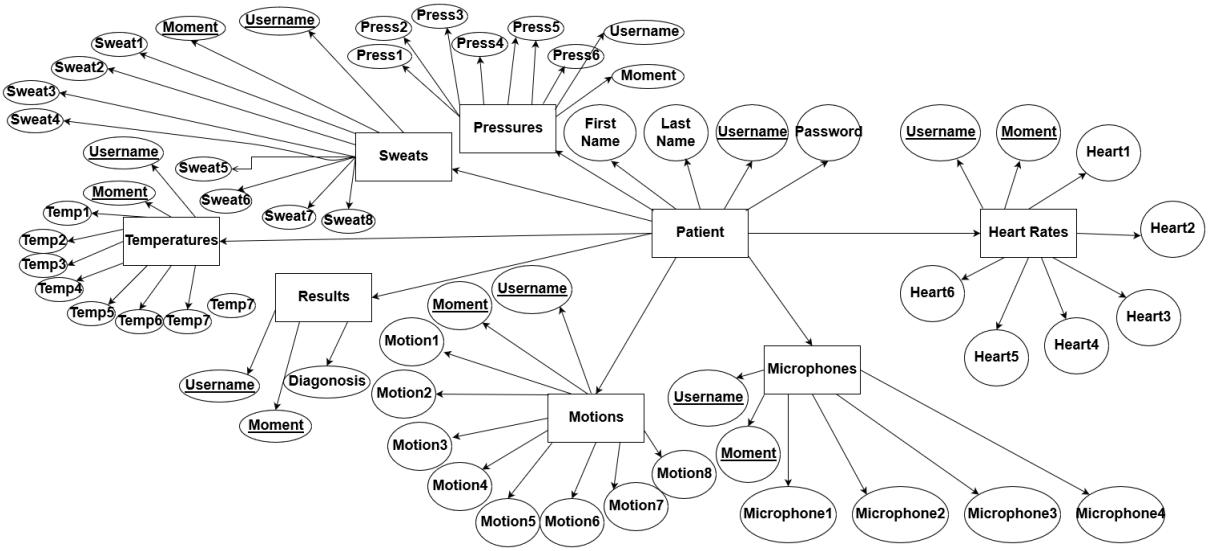

Figure 14. Database Schema

5.6. Design Alternatives

Since the conception of the project's idea, multiple design ideas have to come to our minds. They were either further elaborated on or discarded. While the general concept of an unobtrusive monitoring mattress stayed the same, the details greatly varied. These details include the diagnostic methods, sensors to be used and how to use these sensors. First of all, there are many ways to diagnose sleep apnea: from simple pulse oximetry to complex polysomnography. These methods have varying reliability and validity, both of which tend to suffer in methods that are user-friendly. Secondly, there is a variety of physiological readings that can be assessed in home environment: body temperature, pulse, blood pressure, blood oxygen saturation etc. Of course, with unlimited resources very complex readings can be obtained but that kind of machinery is a system on its own and most probably will not be compatible with our system. Thirdly, not all information about the body's state has to be directly associated with the physiological readings. There are indirect methods that can provide a necessary part of a picture. For example, breathing can be obviously analyzed through microphones. It does require an extended work on a software side, but it can provide a lot of information not only about person's lungs' state, but also about heart's condition and sleep quality if assessed properly. With this knowledge in mind we pondered about three design concepts. Two of them were discarded but the third one was picked and later modified into our final design. For convenience, all concepts will be named after the main diagnostic criterion that is used in each of them and they will be supplied with a list of advantages and disadvantages.

5.6.1. Blood Pressure-based Diagnostic System

This idea was heavily influenced by the Master's thesis of American University of Sharjah graduate student Mamoun Al-Mardini, particularly by his use of pulse oximetry. While it was a tempting approach, it required an attachment of a device to a user's body, which went against our principle of non-obtrusiveness and absence of physical attachments. On top of that, we could not just replicate someone else's approach. Therefore, a similar but different approach had to be found.

Research of alternatives showed that blood pressure increases during apnea episodes. That is not surprising, since breathing pause is small event that affect not only an entire respiratory system, but also a cardiovascular one.

Unfortunately, this approach was abandoned due to two problems. The first problem is the lack of a blood pressure sensor that would not interfere with user's comfort during sleeping. A typical blood pressure monitor gathers data through an arm cuff that squeezes an arm tightly. A person might get used to it eventually but it severely restricts the users' freedom in bed and might wake them up during "light" sleep stages. A touch-based blood pressure sensor was found but only in a form of a patent. The second problem is that increased blood pressure by itself is not a symptom but rather a single consequence of regular blood flow disruption. By itself, it would not enough to detect sleep apnea episodes especially in users with hypertension.

1) Advantages:
    a) Reliable.
    b) Simple.
2) Disadvantages:
    a) Obtrusive and/or invasive.
    b) Large gap between readings (~7 seconds).
    c) Ambiguous results for people with other health complications.

5.6.2. Lung Movement-based Diagnostic System

An unorthodox approach which was conceived during the first discussions with advisors. The basic idea was to monitor lung movements through motion sensors to detect increases in breathing intensity and breathing cessation episodes. It showed great promise because sleep apnea has unique patterns of breathing and our results would have had high validity. Further research, though, showed that motion sensors alone would only be able to detect whether the lungs are moving (not to mention, that they would have high amount of errors). In order to enhance this approach, accelerometers had to be installed as they can show how the fast the lungs are moving, therefore describing breathing's intensity. Accelerometers had to be placed directly on person's chest, which obviously violates the non-obtrusiveness constraint. Basic constraints, reliance on single criterion and contradicting evidence regarding similar approaches butchered the idea.

1) Advantages:
    a) Reliable.
    b) Valid.
    c) Innovative?
2) Disadvantages:
    a) Unreliable (depends on many external factors).
    b) Obtrusive.
    c) Lack of research support.

5.6.3. ECG-based Diagnostic System

Predecessor of current system. During apnea episodes, ECG's amplitude increases significantly. Analysis of ECG is the current growing approach of sleep apnea diagnosis alongside with pulse oximetry. While the results are less reliable than polysomnography, it is the only considered approach that is used in specialized medical centers. The idea was to replicate the procedure in the home environment. Unfortunately, this idea faced two major complications.

First complication was the positioning of the sensors. There is a huge variety of positions where ECG electrodes can be placed but the amount of the positions is still limited and it is impossible to predict how the body will be positioned on the bed.

Second complication stems from necessity of direct attachment of electrodes to body. It is not enough just to touch one of them, because that way the signal will not be stable. On top of that, the conductive gel has to be applied to the spots to which electrodes are attached to. Therefore, even if an attachment part can be potentially avoided, it would still be necessary for users to apply a conductive gel on most of their torsos every night. While technically it does not violate non-obtrusiveness constraint, it causes a great deal of discomfort through sensory irritation.

The complications proved to be an obstacle not worthy overcoming, but they did push us to research further. The first logical an alternative to ECG is another important heart-related measurement – the pulse. It was found that pulse reacts to apnea episodes roughly the same way as ECG – a rapid increase in beats per minute, beyond the values that heart regularly produces. From this discovery, our current system was developed.

1) Advantages:
    a) Reliable.
    b) Valid.
    c) Relatively simple.
2) Disadvantages:
    a) Obtrusive.
    b) Expensive to replicate.

6. Progress since Design Project I

Our project has significantly evolved since the time of Senior Design I. Additional research, implementation work and testing have altered a lot of our initial ideas and ambitions.

On the hardware side, it was the ECG component that had to go. Initially, it was supposed to be used as a main diagnostic criterion due to a huge research support. Because of its inconvenience, it was necessary to look for alternatives, which was found in pulse-based sleep apnea detection.

On the software side, the machine learning aspect of the project and mobile application were removed. Machine Learning was planned to be used for more precise and sophisticated diagnostic algorithms. However, time constraints and slower progress did not leave much time for testing and data gathering, which were crucial to set up this kind of system. Mobile application was instead transformed into a website which serves the same purpose but is easier to implement.

7. Project Management

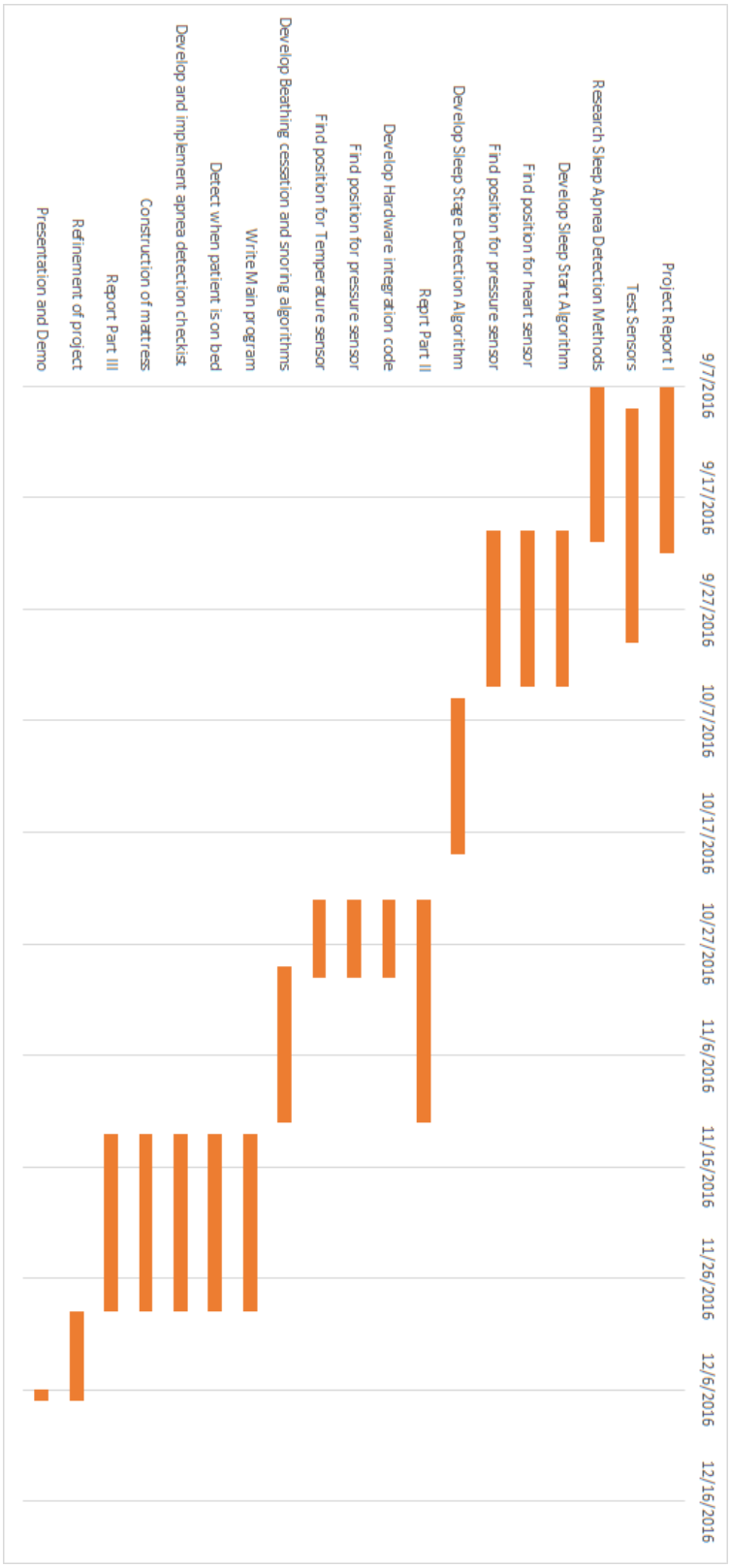

Figure 15. Gantt Chart

8. Implementation

8.1. Hardware

8.1.1. Mattress Reconfiguration

In order to preserve the user's body from harm and discomfort, most of the components had to be concealed. First of all, microcontrollers and breadboards were placed inside the mattress. It was cut at two places at the edges of the middle horizontal line of the mattress. Four more holes were cut, two of each diagonally near each of the microcontroller to place breadboards at.

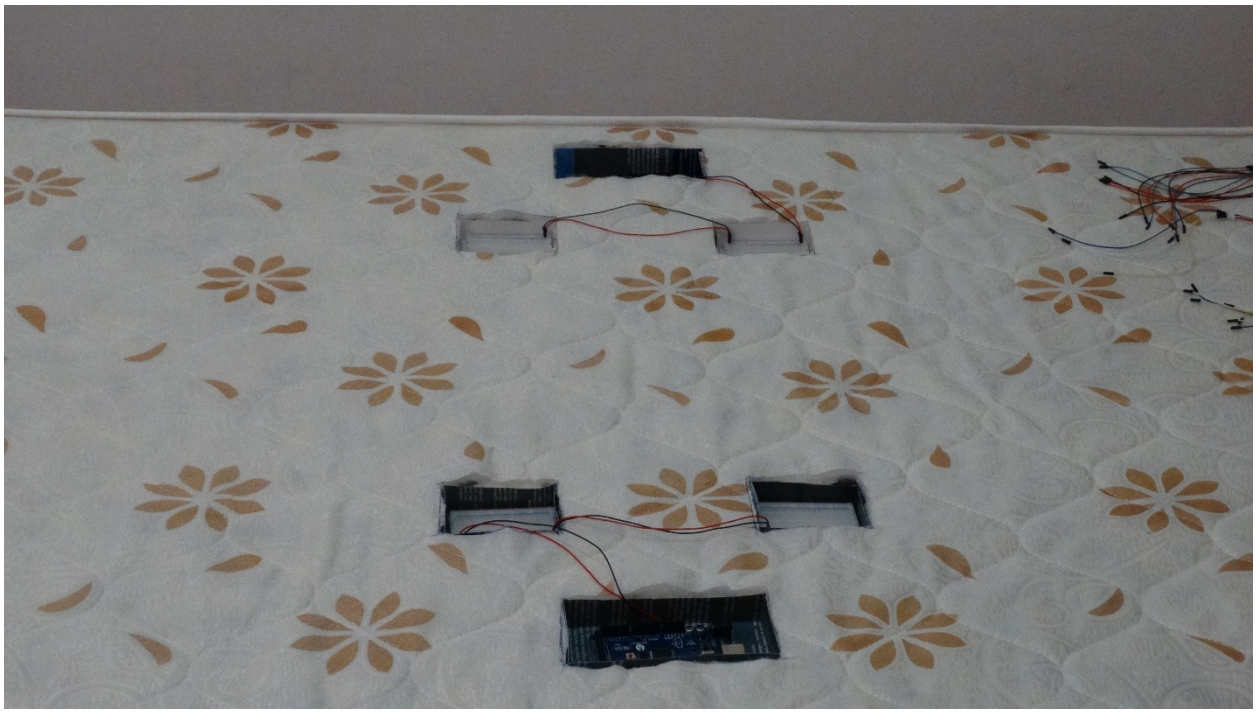

Figure 16. Placement of microcontrollers and breadboards inside the mattress

To preserve the rest of the mattress, a polymer cover was bought and placed over the mattress. It was used to cover the wires leading to the sensors as well. Once the position were determined the sensors were attached to the cover or pockets were cut out if the sensor contained pointy/sharp parts. The holes were drilled near each sensor to lead the wires underneath the cover. Additionally, a block was placed to place a user's neck upon because it contains heart rate sensors:

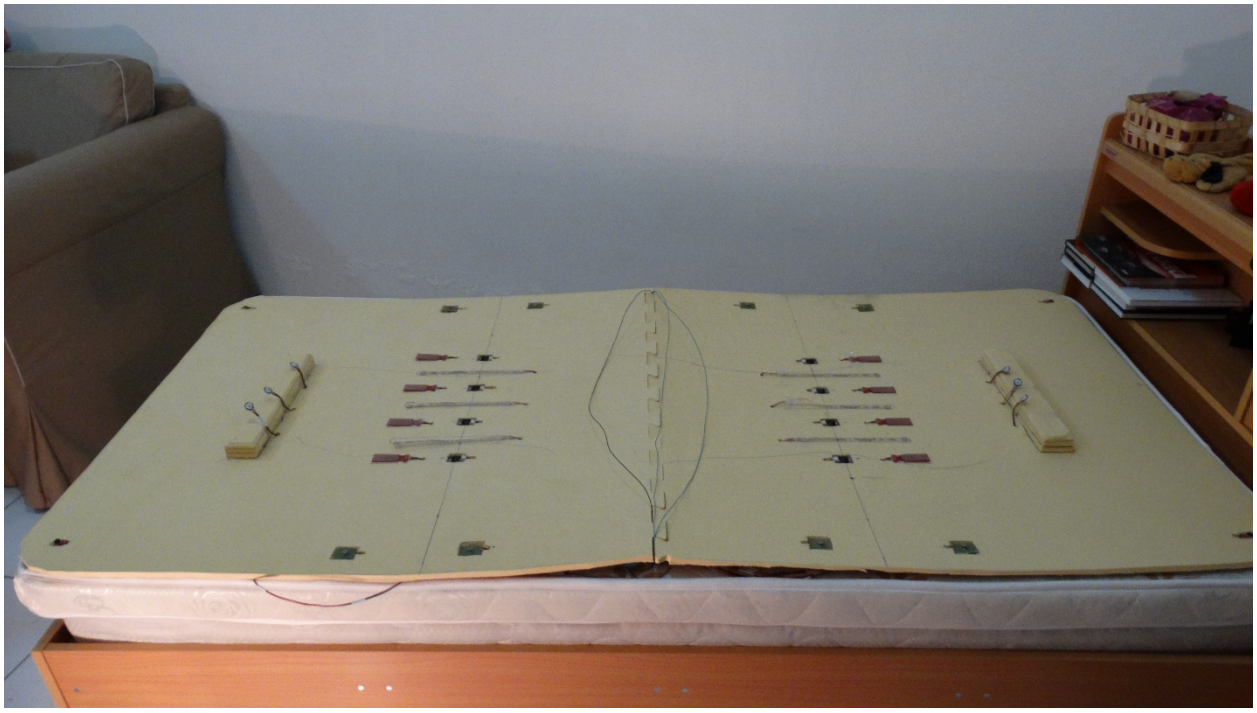

Figure 17. Sensors' Placement

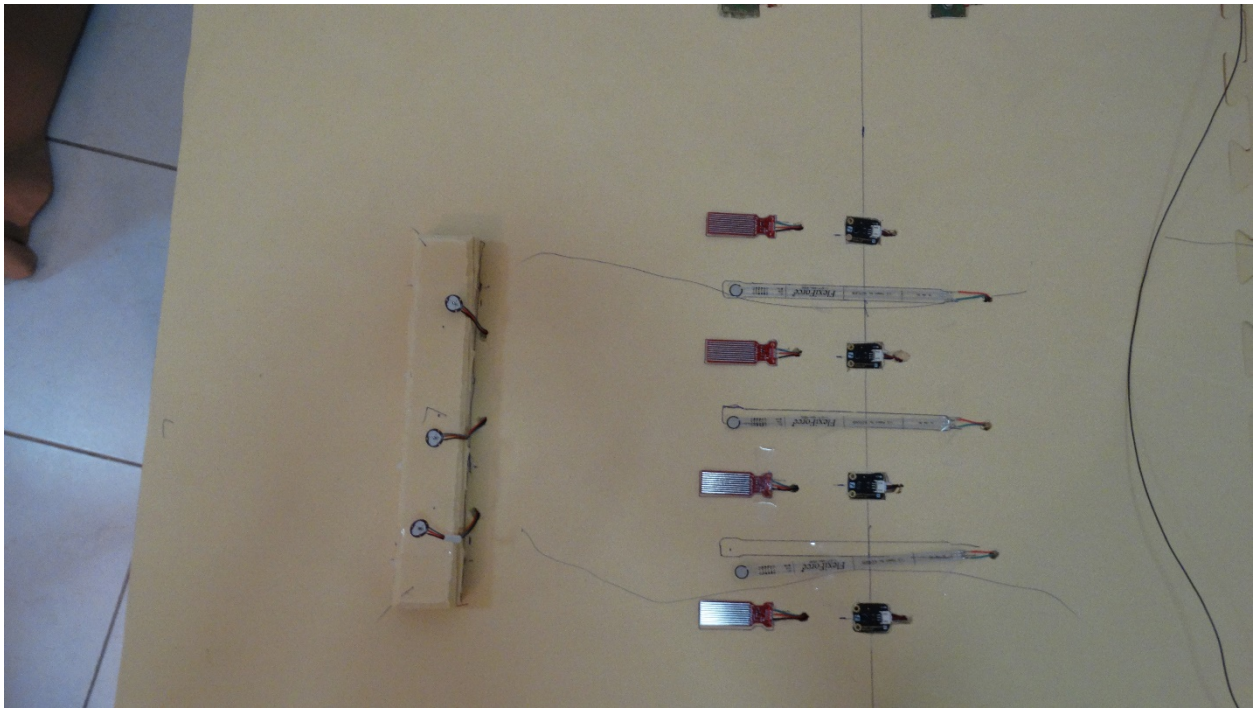

Figure 18. Sensors' Placement (Lower Half of the Mattress)

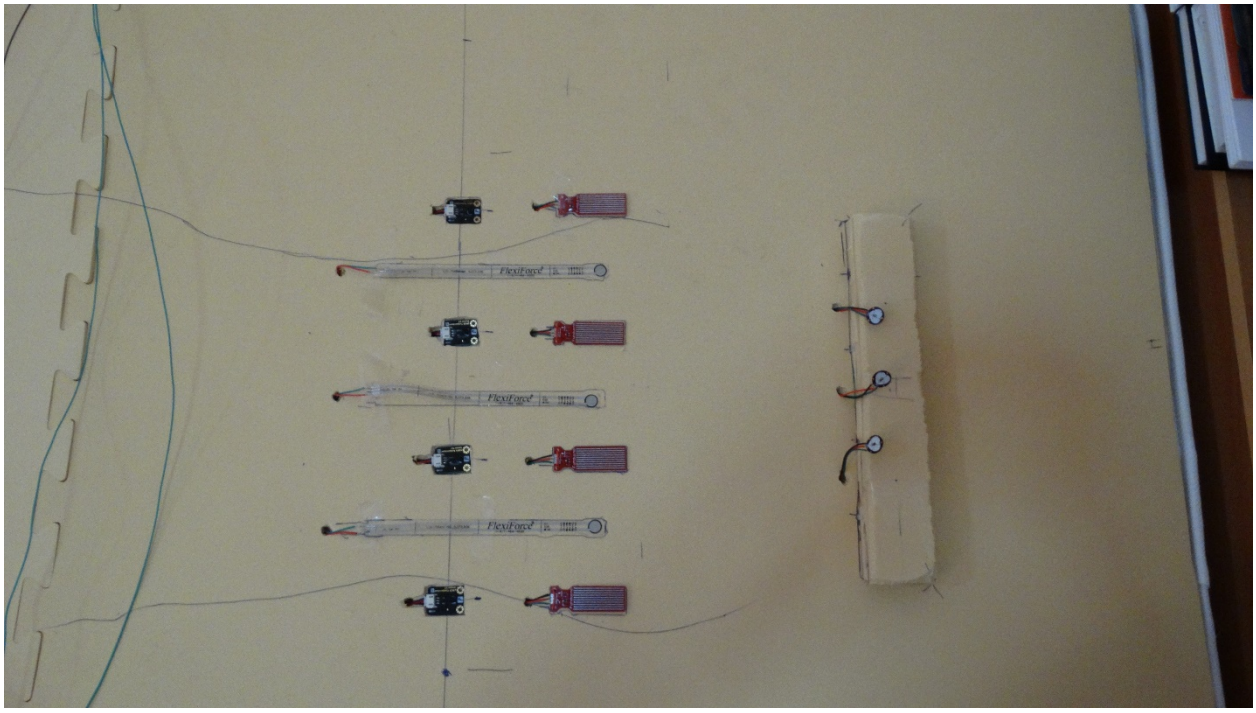

Figure 19. Sensors' Placement (Upper Half of the Mattress)

8.1.2. Sensors' Calibration

An Intelligent Mattress for Diagnosis of Sleep Apnea uses 6 types of sensors: 5 of them receive analogue data and 1 of them receives digital (Boolean) value. Analogue sensors measure amount of sweat produced by user, pressure force applied by lungs and diaphragm, pulse of the user, sound levels of breathing and user's body temperature. Digital sensor includes only motion sensor.

Most of the sensors are read by microcontroller in a straight-forward manner. However, heart rate sensor and microphones require additional processing of incoming data. The system sends data to a server every 100 milliseconds.

8.1.2.1. Heart Rate Sensors' Calibration

Analogue data received by heart rate sensor is not a pulse by itself, rather a raw signal that represents the intensity of blood flow in observed capillary tissues. Ten intervals between each heartbeat (known as inter-beat intervals are recorded) each chosen period of time and are then used to calculate the pulse (beats per minute) through the equation 7.2.1.1:

$$P = \frac{60000}{\sum_{n=0}^{9} I_n} \quad (7.2.1.1)$$

P represents pulse of the user and is measured in beats per minute (BPM), I represents Inter-beat Interval (measured in milliseconds) and 60000 is an amount of milliseconds in a minute [30].

A sample code provided by creators of the sensors is designed for only one sensor only. Multiple sensors (in this case, six) require a larger sampling frequency. The code performs all pulse calculation within an interrupt service routine of Arduino microcontroller. It depends on of a timers to determine the sampling period. Specifically, the timer is given a pre-scaler value and a value until which to count. These values determine the sampling frequency through an equation 7.2.1.2:

$$C = \frac{16*10^6}{(P*f_S)} - 1 \text{ (7.2.1.2)}$$

C represents the value up until which the counter is incrementing its values, $16 * 10^6$ is microcontroller's clock frequency, P is a pre-scaler and f_S is a sampling frequency. The full algorithm [30] is shown in figure 14. The code for this algorithm can be found in the appendix.

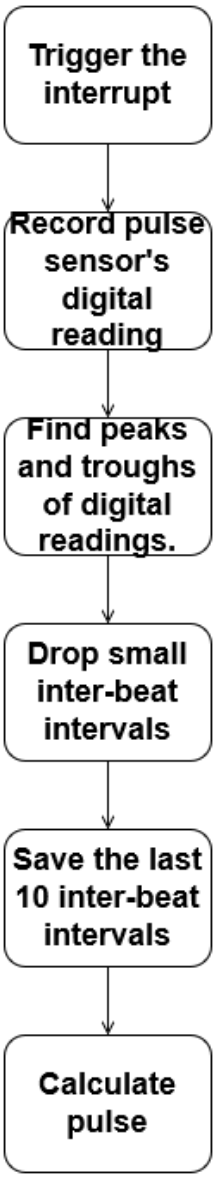

Figure 20. Pulse's Calculation Algorithm Flow Chart

8.1.2.2. Microphones' Calibration

Microphones also depend on a timer, since it records only a sample of the sound. Within this period a minimal and maximal digital values of sound are found and sound wave amplitude is calculated using the difference between the two. Sampling period was chosen to be the same as the period of sending the data to the server. Figure 17 demonstrates how the algorithm works.

A sampling period of 100 milliseconds for two reasons. First of all, instances of breathing and snoring may take up to several seconds depending on the sleep stage, condition of lungs etc., therefore a smaller period provides unnecessary data and a larger period leaves some data unaccounted for. Secondly, the other data needs to be more instantaneous (such as heart rates and lung/diaphragm pressure forces) for the correct diagnosis and needs to be sent as frequently as possible. Therefore, 100 milliseconds is a golden mean, since it is frequent enough for other sensors and does not create sound data loss (as it coincides exactly with the moment of sending data to the server).

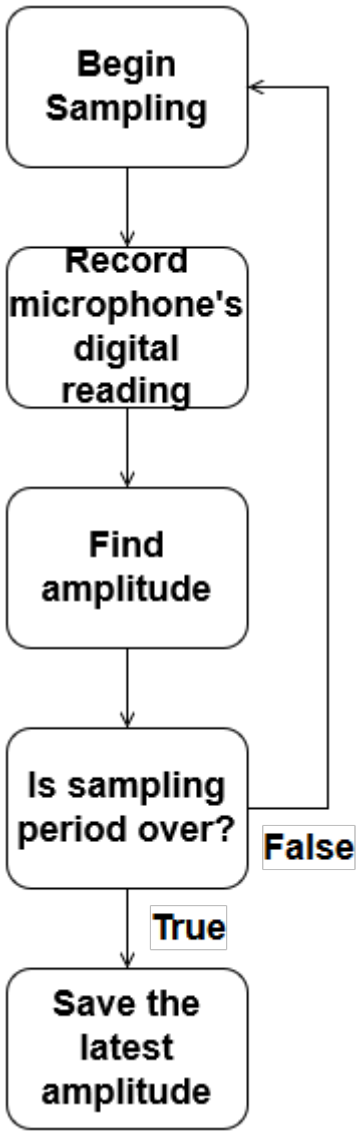

Figure 21. Microphone's Calculation Algorithm Flow Chart

8.1.3. Data Communication

Due to a large amount of sensors (40), more than one microcontroller had to be used. Even though a single Arduino Mega 2560 has more than 40 pins, most of them are digital and only 16 are analogue, while 32 were needed. Therefore, two microcontrollers had to be used. This fact introduced a need of data communication between the two microcontrollers.

Arduino Mega 2560 has 4 serial communication ports. Serial Communication Port 0 (referred to without 0, this notation is used for convenience) is used in communication with PC, therefore it was left alone. Serial Communication Port 1 was used to established data exchange between to microcontrollers:

1) Pin TX1 (slave) was connected to pin RX1 (master).
2) Pin RX1 (slave) was connected to pin TX1 (master).
3) GND (slave) was connected to GND (master).

The first microcontroller is used only for data collection and sends the acquired readings to a second one that records its portion of data and sends the complete set to a server through an XBee module for wireless communication. Initially, both microcontrollers were thought to perform the same tasks with only data destination being different. Later it was found that serial communication is severely restricted by three factors:

1) The size of data stream is 64 bytes.
2) Delay is required in order for all data to be sent (time of delay depends on the amount of data).
3) Heart rates and sound levels calculations are time consuming processes that might not be finished by the time the data needs to be sent, so the garbage data is provided instead.

The first problem was solved by sending the sensory data as bytes. None of the readings exceeded 10 bits in size, which means that the number can be split in just two bytes. Therefore, 20 sensors would provide 40 bytes of data. The second problem was solved by introducing a 100 milliseconds delay to the slave microcontroller code. The third problem partially stems from the second one. Therefore, in order not to increase the delay, these calculations are done by the master microcontroller instead. Additionally, some garbage data still makes it through and sometimes the sent data gets swapped. In that case, the received numbers are replaced by zeros and the serial communication is restarted.

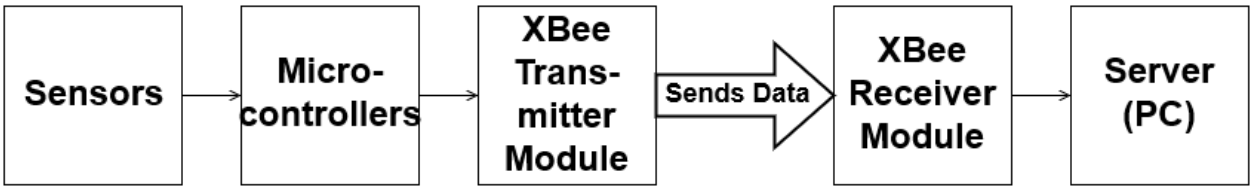

Figure 22. Data Flow Diagram

8.1.4. Receiving Data by Server

The data is received through an Arduino Receiver class. The class is a thread that works the entire time the software is run. Every 100 milliseconds it receives the data from 40 sensors. Once it receives the readings, it converts them into appropriate units and stores them in a database to be used later in the diagnostic procedure. Since the project employs the use of XBee modules, the class has to request the use of a serial port. The setting for it are the following:

1) Baud Rate = 9600 bps.
2) Data Bits = 8 bits.
3) Stop Bits = 1 bit.
4) Parity Bits = 0.

Since the data is sent as a long string, it was safer to receive the data byte-by-byte. Once all the bytes are received, the string is split into tokens that hold the sensory data.

8.1.5. Data Conversion

Sensors provide the data in the digital form. Initially, the conversion to analogues values was to be done on the microcontrollers themselves, but due to observed problems with serial communication, it was thought that the safer option is to do it on the server's side that has much larger memory and computational capacity on its side. Therefore, only heart rates (full conversion) and digital sound amplitudes (partial conversion) are done by microcontroller due to dependence on a timer. The rest of the data is sent in a digital form, which represents the output voltage. The equation 7.5.1 shows conversion of sensor data from digital form to voltage:

$$V_{Out} = V_{In} * \frac{x}{1023} \text{ (7.5.1)}$$

V_{Out} represents output voltage, V_{In} represents input voltage and x represents the sensory data in digital form and 1023 is a constant that represents the maximal digital value of the sensor. Note: in case of Arduino Mega 2560, V_{In} is usually equal to either 3.3 V or 5 V. In order to get a larger resolution, 5 volts were chosen. Equation 7.5.1 is a core equation for the conversion of most of the sensors.

$$t = 100 * V_{Out} \text{ (7.5.2)}$$

Equation 7.5.2 is an equation for the calculation of temperature recorded by the thermometer, where t represents the observed temperature, 100 – the resolution of the sensor with the unit of $\frac{^{\circ}C}{V}$ and V_{Out} – output voltage. The temperature is kept in Celsius in order to adhere to the metric system.

Lungs and Diaphragm Pressure Force is one of the most complex values to compute. First of all, the pressure sensor's voltage is extracted through an equation 7.5.3:

$$V_{Out} = \frac{R_C}{R_{FS}+R_C} * V_{In} \text{ (7.5.3)}$$

$$R_{FS} = R_C * \left(\frac{V_{In}}{V_{Out}} - 1\right) \text{ (7.5.4)}$$

Equation 7.5.4 is used to calculate a resistance of a pressure sensor. R_{FS} is Force Sensitive Resistance (FSR) (resistance of a pressure sensor), R_C is a constant resistance of the voltage divider, V_{In} is an input voltage, V_{Out} is an output voltage. R_{FS} is later converted to electrical conductance of FSR:

$$G_{FS} = \frac{1}{R_{FS}} \text{ (7.5.5)}$$

Equation 7.5.5 is used to calculate electrical conductance of a pressure sensor. G_{FS} is electrical conductance of Force Sensitive Resistor and R_{FS} is Force Sensitive Resistance. As per equations provided by the designer of the sensor, there are two possible final calculations of the pressure force depending on the values of the FSR.

If pressure sensor's resistance is less than or equal to 600 **Ω**, then the equation 7.5.6 should be applied [31]:

$$F = \frac{G_{FS}-0.00075}{0.00000032639} * g \text{ (7.5.6)}$$

If pressure sensor's resistance is more than 600 **Ω**, then the equation 7.5.7 should be applied [31]:

$$F = \frac{G_{FS}}{0.000000642857} * g \text{ (7.5.7)}$$

F represents the pressure force applied by the lungs and diaphragm. G_{FS} is electrical conductance of Force Sensitive Resistor, 0.00075 is an offset conductance given by the sensor's creator, 0.00000032639 is a constant with a unit of $\frac{S}{kg}$ and is provided by the sensor's creator, g is gravitational acceleration, which is equal to 9.81 $\frac{m}{s^2}$ and $\frac{N}{kg}$.

Equation 7.5.8 is used to calculate sound level detected by a microphone. Due to time constraints, it was impossible to implement a more sophisticated sound processing. Therefore, it is assumed that the only sound detected by microphones would be the breathing sound. Hence, only a sound level needs to be recorded.

$$L = 20 * \log\left(\frac{V_{Out}}{0.0001}\right) \text{ (7.5.8)}$$

L is a sound level represented through decibels, 20 is a standard constant used in sound pressure levels calculation and V_{Out} is output voltage.

$$\varphi = \frac{x}{1023} * 100\ \%\ (7.5.9)$$

Equation 7.5.9 is used to calculate the sweat intensity of a user. Unlike other sensors, there is no equation for this sensor. Therefore, the conversion was inspired by the relative humidity equation. It is not necessary to extract voltages for this equation, since the final value has no units anyway. Sweat intensity is calculated by dividing the measured sweat intensity by maximal digital value of the sweat intensity: φ represents the sweat intensity and x represents the sensory data in digital form and 1023 is a constant that represents the maximal digital value of the sensor.

8.1.6. Power Demands of the System

Each microcontroller in the system requires 12 Volts. Current that flows through V_{CC} and GND pins is equal to 200 mA, and each digital pins needs 40 mA of current [28]. V_{CC}, both grounds and 6 digital I/O pins are used (4 for motion sensors, 2 for serial communication) Therefore the power consumption of a single microcontroller is equal to: P = V * I = 12 * (0.2 * 3 + 0.04 * 6) = 10.08 W, same as a regular tablet [29]. Both microcontrollers require the double of that amount which is equal to 20.16 W. To keep our device wireless we used two blocks of 8 AA batteries that supply 12 Volts to the system. They can keep it working for 16 hours without.

8.2. Software

8.2.1. Role of Software

While the hardware which mainly comprises of the sensors, microcontrollers and interconnecting wires is responsible for gathering and transmitting data, the software helps to analyze this data and tries to estimate the probability that the user has sleep apnea. This is done through several custom-made algorithms which were developed to analyze the data arriving from the sensors. The chief algorithms are those which make key decisions like detecting breathing pauses, identifying sleep stages (REM/NREM), detecting start of patient's sleep, detecting time of awakening, analyzing microphone readings for snoring, analyzing microphone readings for breathing pauses, analyzing heart rates for abnormalities. This section of the report will explain each of these algorithms using flow charts.

8.2.2. Overview of software implementation

Diagnostic software has been implemented using Java. We chose Java because it is unparalleled when it comes to support for object oriented-programming. Java classes and functions provide a great amount of support for data abstraction which helps us to simplify

the coding effort. The diagnostic software is implemented in the SleepAnalyzer class. Each algorithm is implemented is implemented as a function which will be called at the appropriate time.

The system will keep waiting till it receives data. Once data is received at the end of a person's sleep, it will be fed to several diagnostic algorithms. As mentioned previously there are separate algorithms for dealing with pulse analysis, sleep stage detection, sleep start detection, determining time of awakening, snoring detection and detecting breathing pauses. The following flowcharts illustrates the overall functioning of the system.

8.2.3. Sleep Start Detection Algorithm

It is important to know when the user has fallen asleep as this is the exact instant when the analysis regarding sleep apnea should begin. The main challenge faced during the development of this algorithm was that we were restricted to pulse, pressure, moisture and temperature sensors. Generally, electroencephalograms which measure brain activity are used to detect sleep stages and transition from wakefulness to sleep. However, we did not have access to this kind of sophisticated equipment. Also, pulse and breathing do-not change much when our body begins its initial descent into sleep. However, we discovered that as a person enters the first NREM sleep stage, their body temperature decreases by 2 degree Fahrenheit. Thus, we developed an algorithm which captures the normal body temperature of person while he/she is awake and scans the overnight temperature sensor readings for a drop larger than 2 degrees Fahrenheit when compared to temperature while awake. The exact moment when this drop is observed is considered as the instant when the person entered the first NREM sleep stage at the beginning of sleep. Figure 18 shows the flow chart of this algorithm.

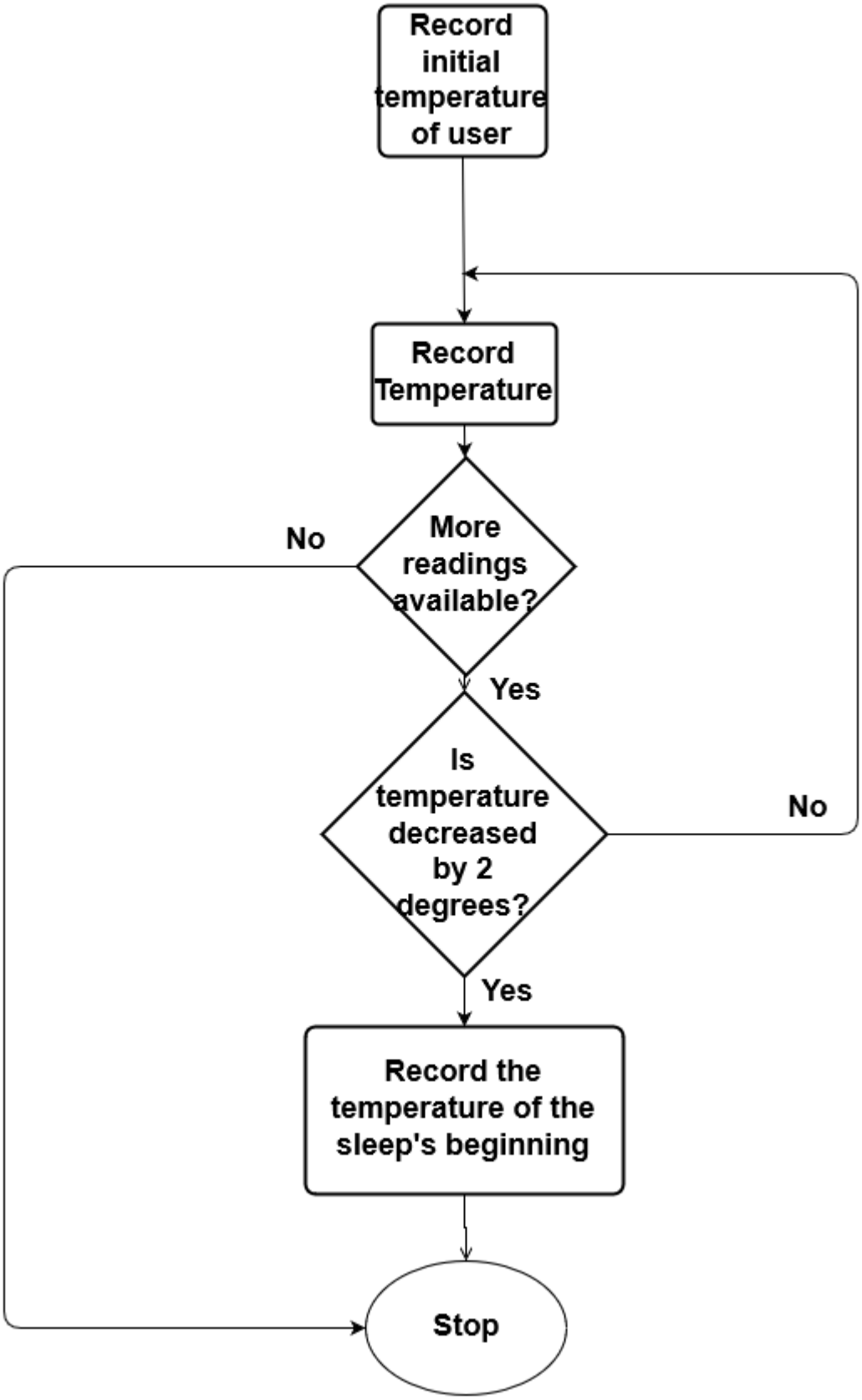

Figure 23. Sleep Start Detection Algorithm Flow Chart

8.2.4. Sleep Stage Detection Algorithms

The parameters that we are using to detect sleep apnea include pulse, temperature, lungs and diaphragm pressure, sweat intensity. They vary based on the specific stage of sleep that person is in. For example, the increase in heart rate associated with an episode of sleep apnea could also be attributed to REM sleep. To account for this, we are diving the sleeping time of the patient into REM/NREM sleep stages. This is mainly done with the help of pulse sensors and motion sensors. During REM sleep, unless a person is suffering from a medical disorder, there should be no physical movement exhibited. Also, we estimated through research that a person's heart rate should increase by a factor of 7 percent or more in rem sleep when compared to their resting heart rate. Our algorithm works by progressing through all the overnight pulse recordings and looks for the seven percent increase in heart rate to detect transition from NREM to REM. It categorizes other intervals as NREM intervals. It also uses motion sensors along with heart rate sensors to detect the sleep stages.

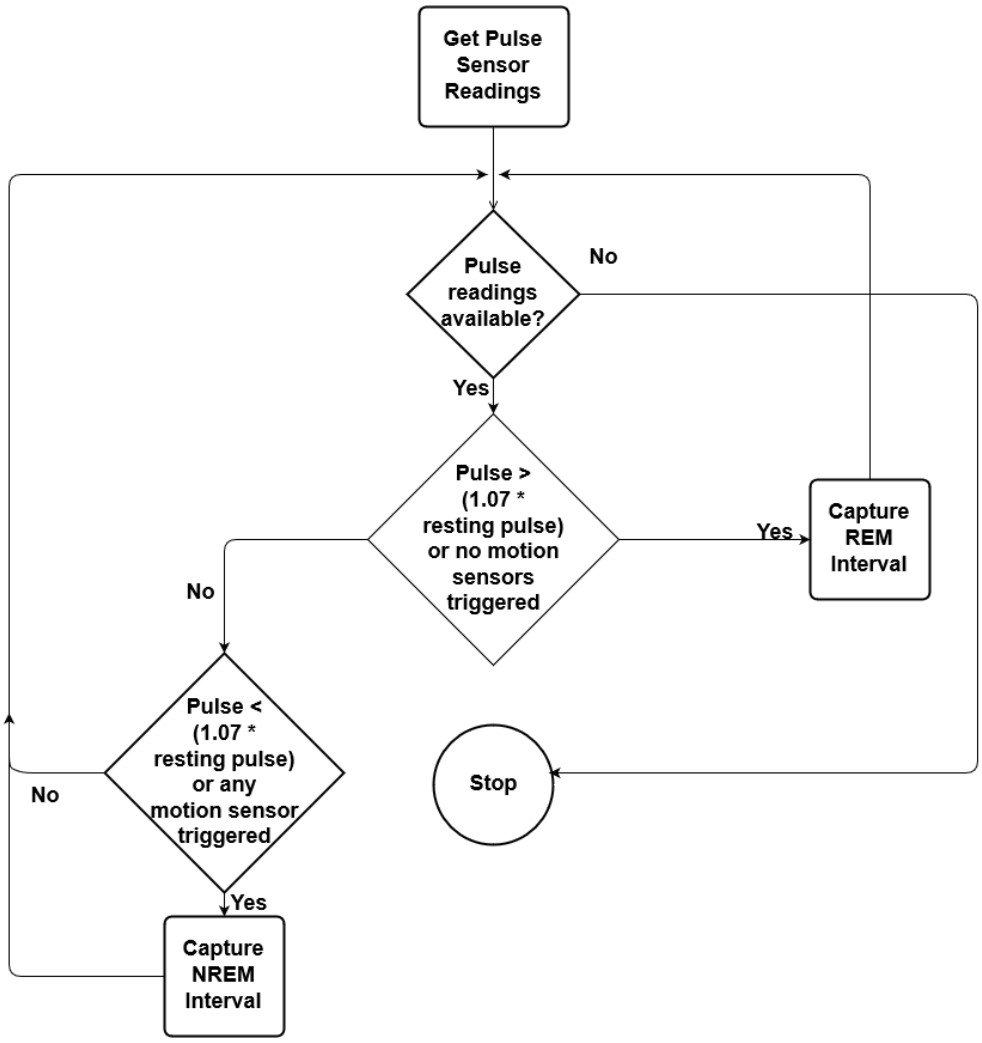

Figure 24. Sleep Stage Detection Algorithm Flow Chart

8.2.5. Sleep Apnea Detection Through Sound Analysis

The purpose for recording sound using microphones is twofold. We will detect snoring when the sound decibels reach a value larger than 70. We will also detect breathing cessations and shallow breathing by looking for very low values of sound decibels in the range of 0 to 20. The algorithm for detecting obstructive sleep apnea will look for a minimum of 5 snoring episodes in an hour and the algorithm for detecting destructive sleep apnea will look for a minimum of 5 breathing cessation episodes in an hour. In either case, more than 5 episodes suggests a likelihood of sleep apnea.

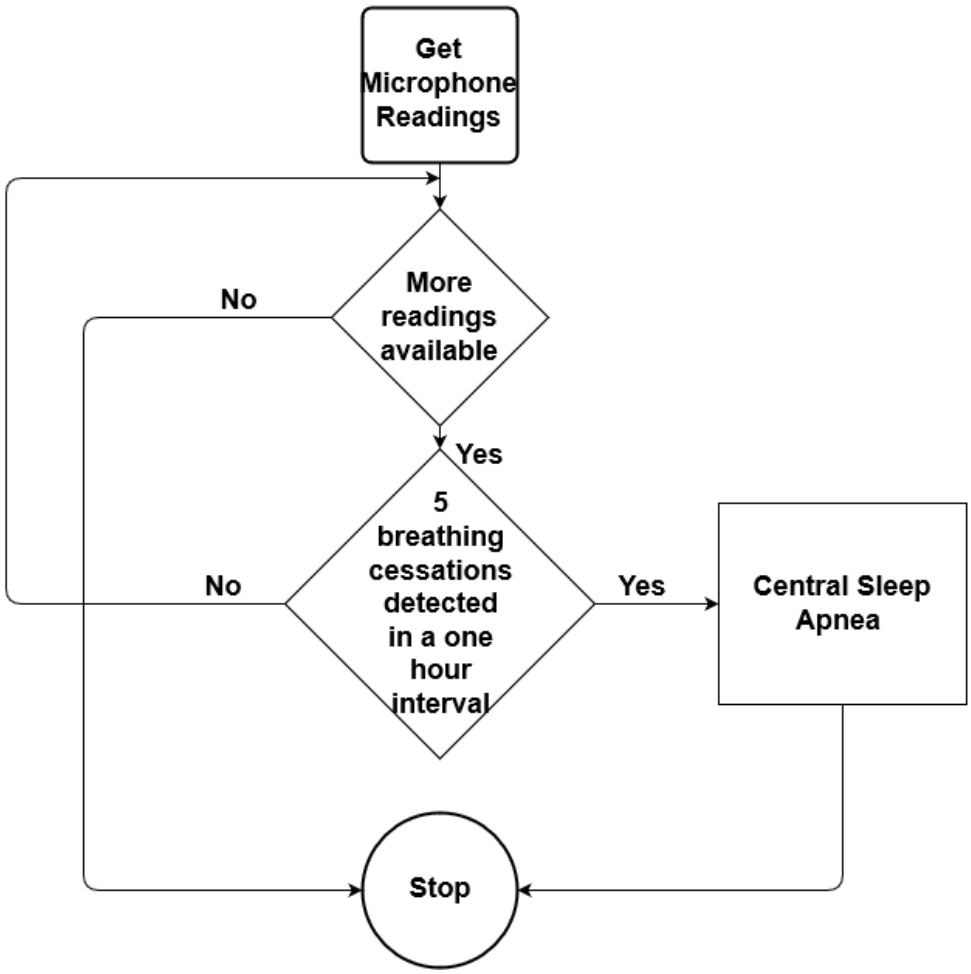

Figure 25. Breathing Cessations Detection (Sound) Algorithm Flow Chart

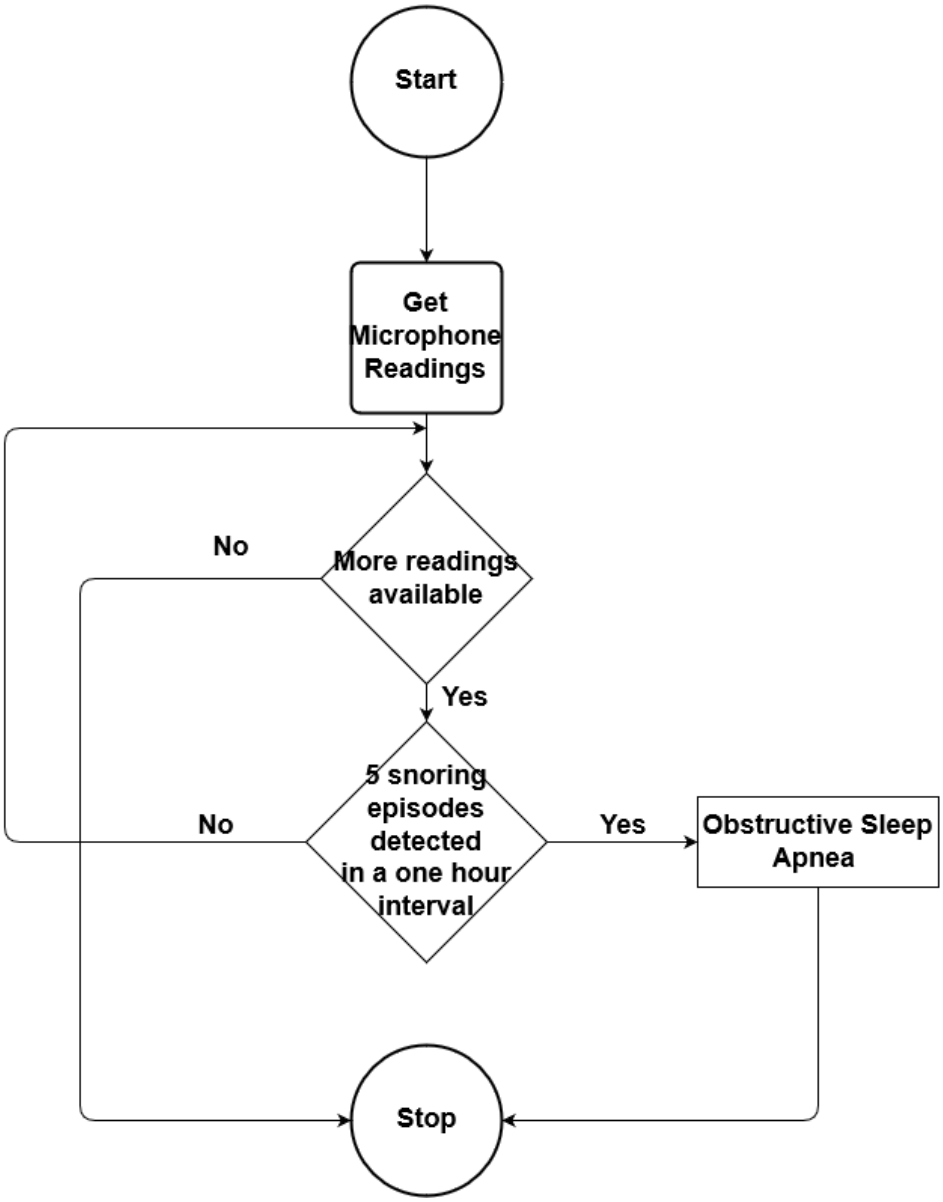

Figure 26. Snoring Detection (Sound) Algorithm Flow Chart

8.2.6. Sleep Apnea Detection Through Pulse Analysis

To detect whether a patient is at risk of having sleep apnea we developed an algorithm for analyzing the readings obtained from the heart rate sensors overnight. Since there is great amount of difference between the NREM and REM heart rate of a person, we need to develop separate algorithms for analyzing these two stages.

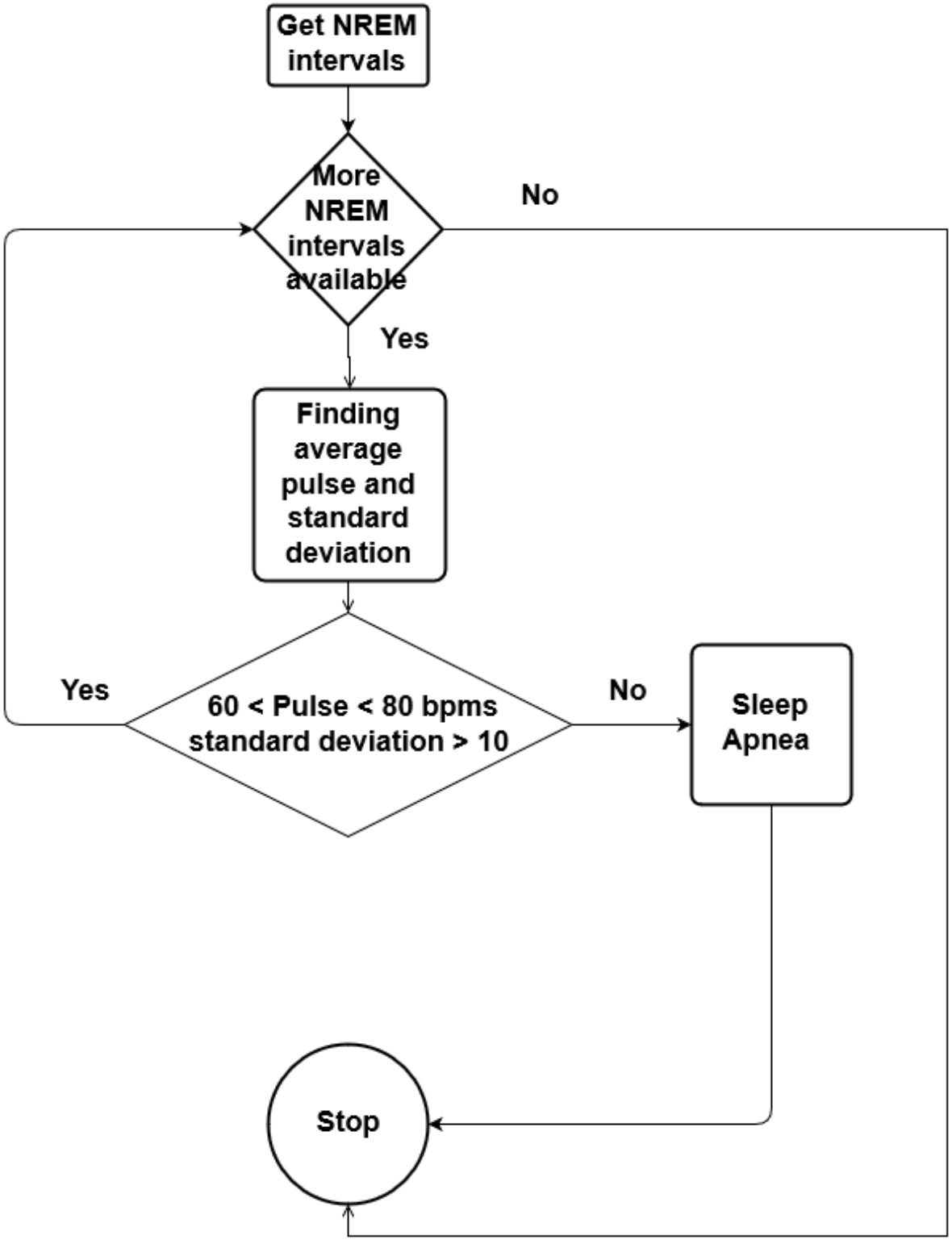

Figure 27. Pulse Increase Detection during NREM Algorithm Flow Chart

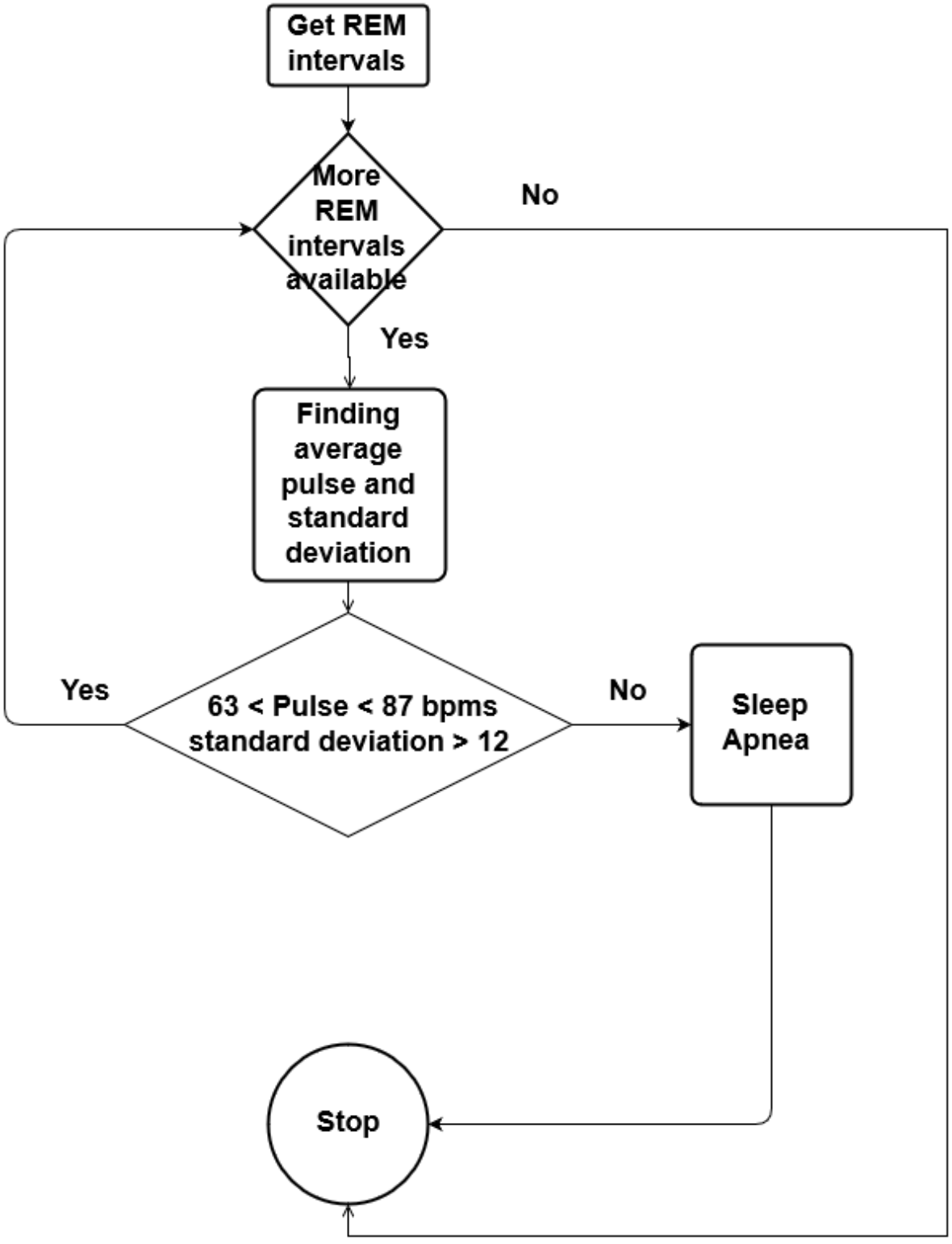

Figure 28. Pulse Increase Detection During REM Algorithm Flow Chart

Both algorithms analyze the REM/NREM intervals provided by the sleep stage detector algorithm. The REM Pulse analyzer algorithm calculates the mean and standard deviation for the heart rates belonging to each REM interval. It then checks if the mean pulse rate is between 60 and 80 and if the standard deviation greater than 10. The NREM pulse analyzer also calculates the mean and standard deviation for the heart rates belonging to each

NREM interval and checks if mean pulse rate is between 63 and 87 and if the standard deviation greater than 12. For his/her pulse to be indicative of sleep apnea, the patient should satisfy either the REM or the NREM criteria.

8.2.7. Sleep Apnea Risk Assessment

There are three criteria for sleep apnea: pulse increase, snoring or shallow breathing and cessation of lung movement. Instead of giving a definite diagnosis, our system looks for any of these symptoms and counts how many are present. If there all of these are present, user is advised to contact medical professional. Otherwise, 0 symptoms represent no risk, 1 – minimal, 2 – medium. The algorithm is shown in figure 25:

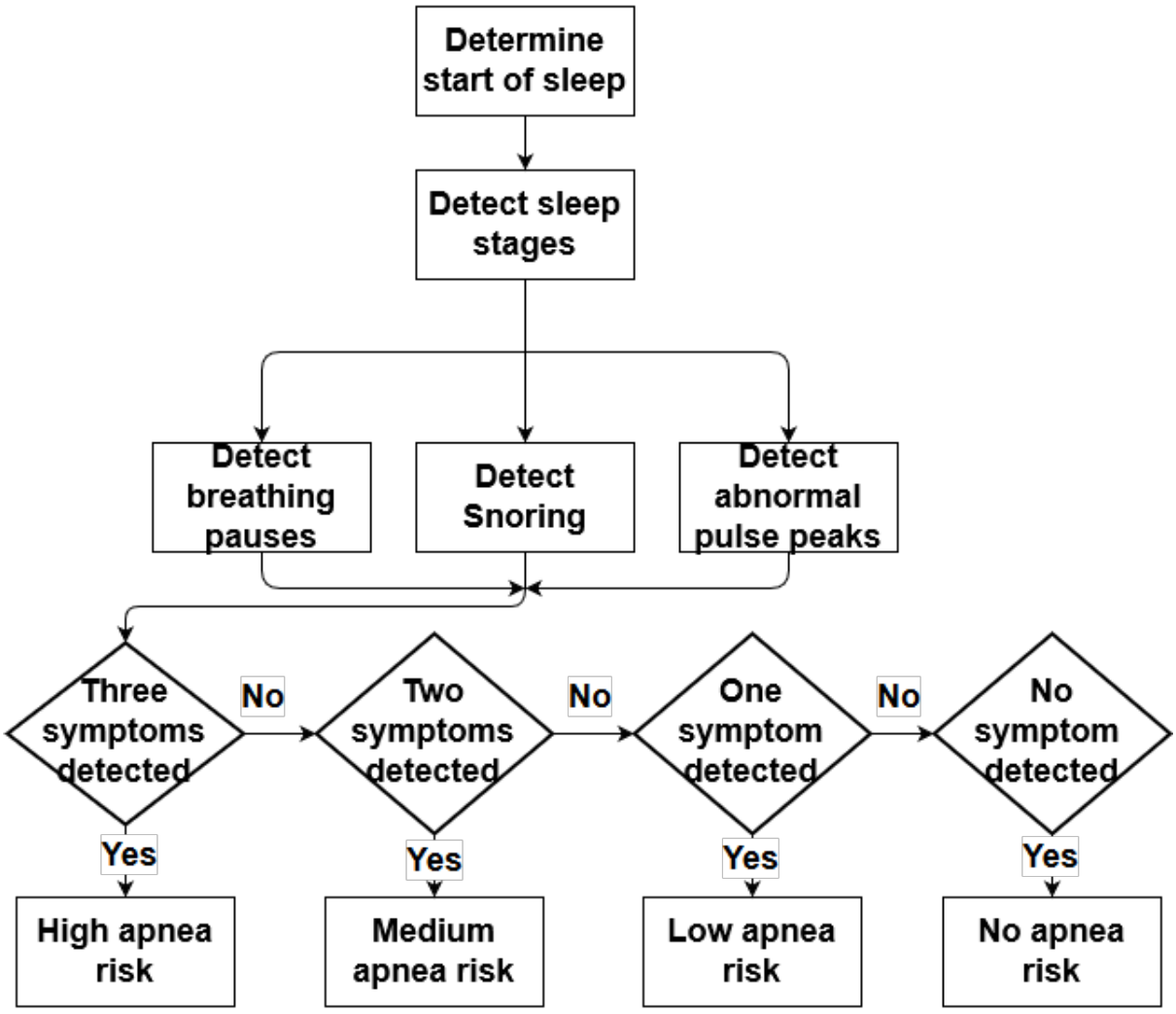

Figure 29. Sleep Apnea Risk Assessment Algorithm Flow Chart

9. Testing

9.1. Hardware

9.1.1. Heart Rate Sensors

Heart rate sensor tests were divided in three parts. The first part included testing the different body locations that can be used to measure the heart rate. These locations were fingertip, wrist, chest, spine and neck. The first two locations would make the sensors obtrusive while the third and the fourth ones are both accessible only 50 % of the time (plus, it was later found that they are not supposed to work here, since heart rate sensors use a photo-element to work and need to have a clear visibility of capillaries). The results are shown below. Clearly, chest and spine (even if the heart rate seems legitimate in this case, it is purely coincidental) are excluded immediately, but the rest show promise and consistent results. Unfortunately, the heart rate sensors produce a lot of noise if the visual contact is broken or if poorly isolated parts of the sensor come in contact with the other material. All these fluctuations are shown in the graphs in the form of extreme values.

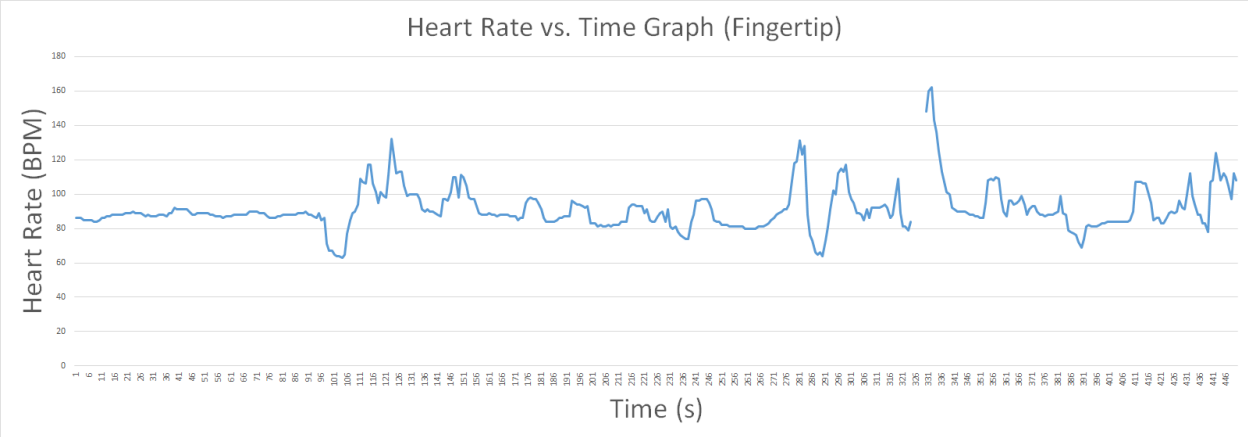

Figure 30. Pulse Sensor Test 1

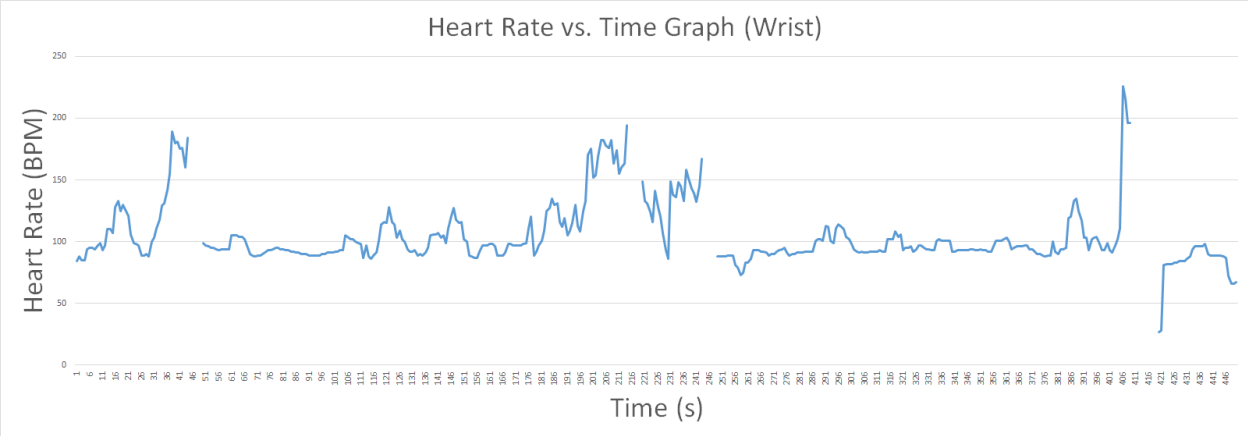

Figure 31. Pulse Sensor Test 2

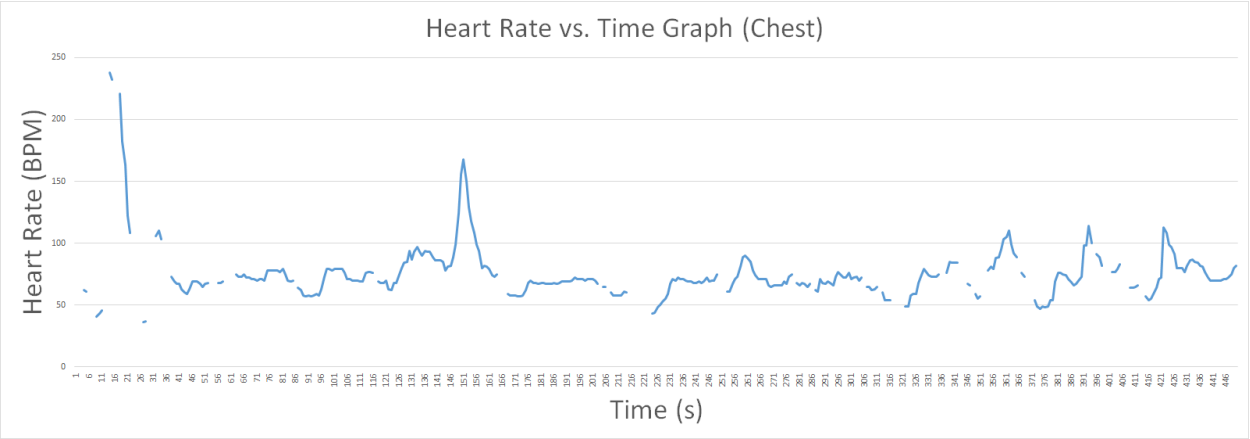

Figure 32. Pulse Sensor Test 3

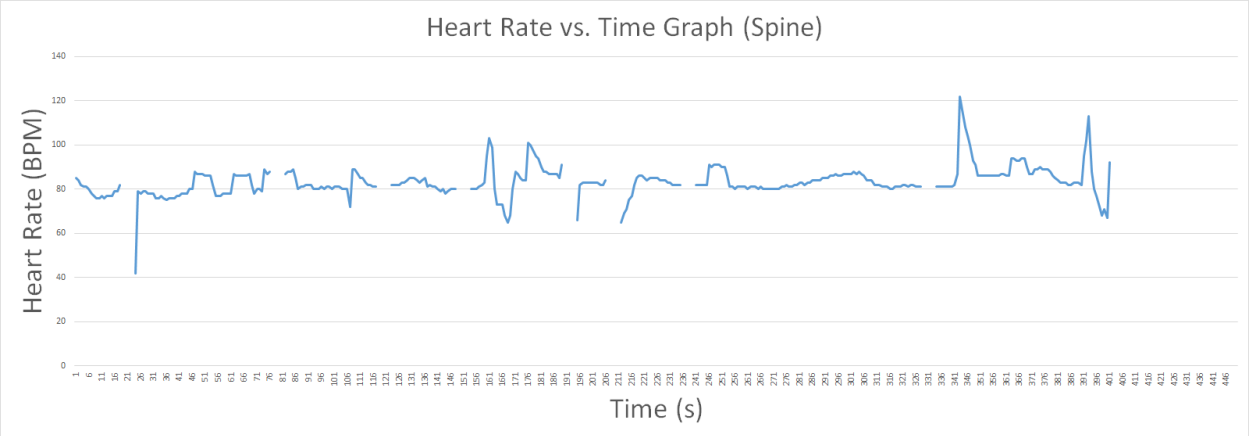

Figure 33. Pulse Sensor Test 4

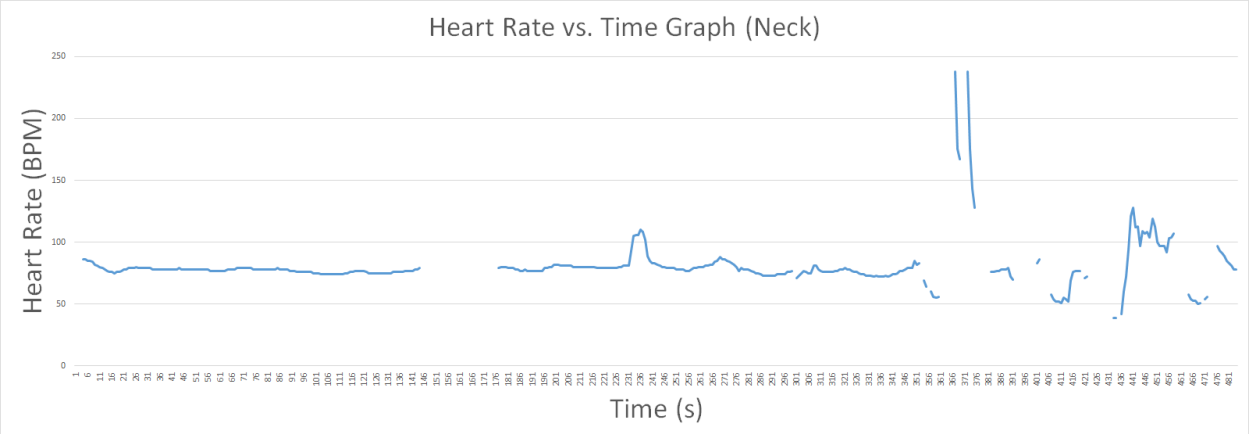

Figure 34. Pulse Sensor Test 5

The described tests show that the only location that will keep the system relatively non-obtrusive and will allow the production of valid and reliable results is the neck. Therefore, we introduced another constraint: a user has to lie on a prop that contains three heart rate sensors.

The second part of testing included comparing how the sensors react to different sides of the neck. Noise aside, the results are similar to each other.

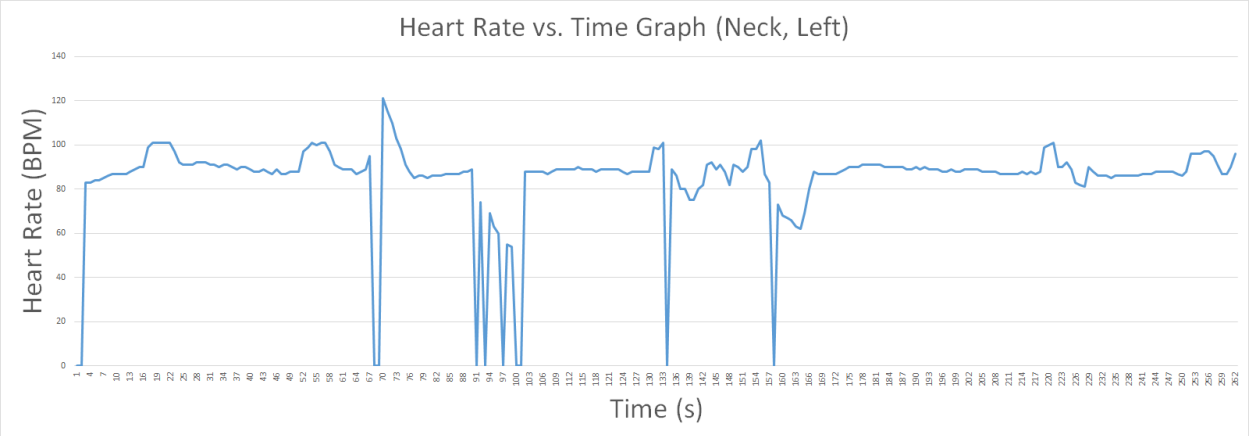

Figure 35. Pulse Sensor Test 6

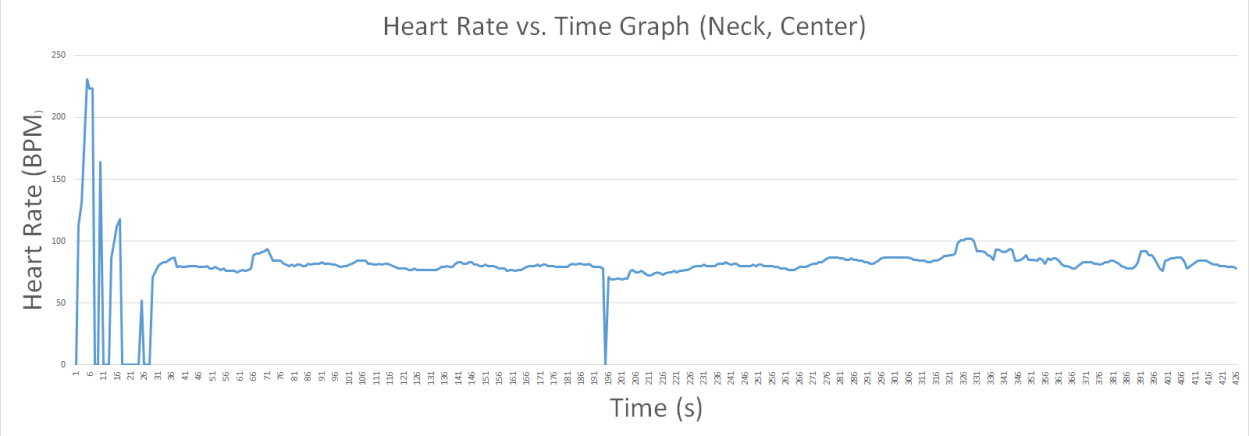

Figure 36.Pulse Sensor Test 7

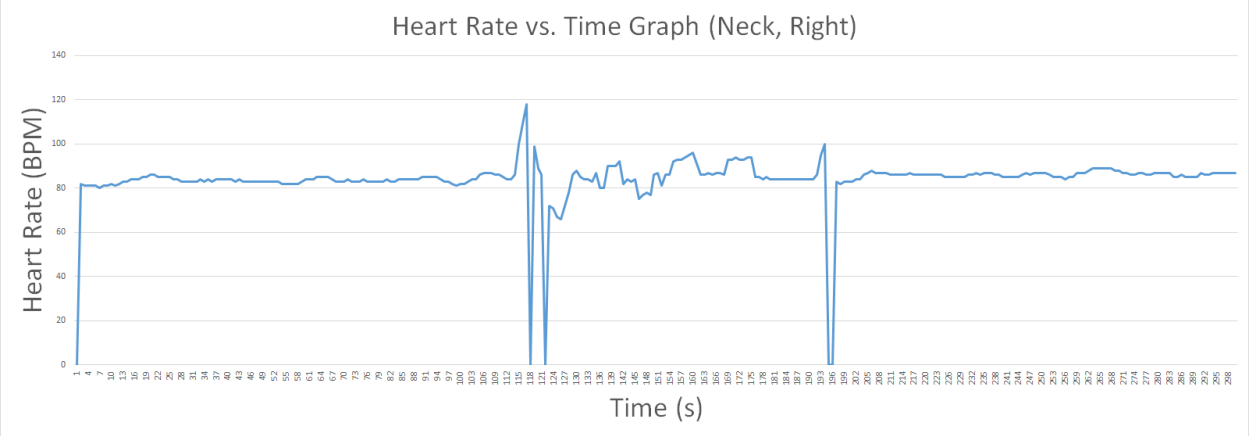

Figure 37. Pulse Sensor Test 8

9.1.2. Temperature Sensors

The most important factor that needed to be determined regarding the temperature sensors was how they react to different parts of the body. Specifically, the tests were designed to determine which areas of the body had a faster thermal conductance, whether it would be affected by placing the thermometers inside the mattress and whether it would be slowed by the clothes and/or mattress covers. All tests were performed until the sensors reached the normal body temperature of 36 ± 1 °C and held it stably with the accuracy of reading confirmed with pharmaceutical thermometer.

9.1.2.1. Without Clothes On

The first series of tests shows how the sensors react to various areas of the body without clothes. In the first test, the test subject was lying on his stomach and imitating movements of a sleeping person. This imitation served as a mean to randomize which part of the body contacts the sensor. It took 551 seconds to reach the stable reading of normal body temperature (which at the time of the test was 36.2 °C).

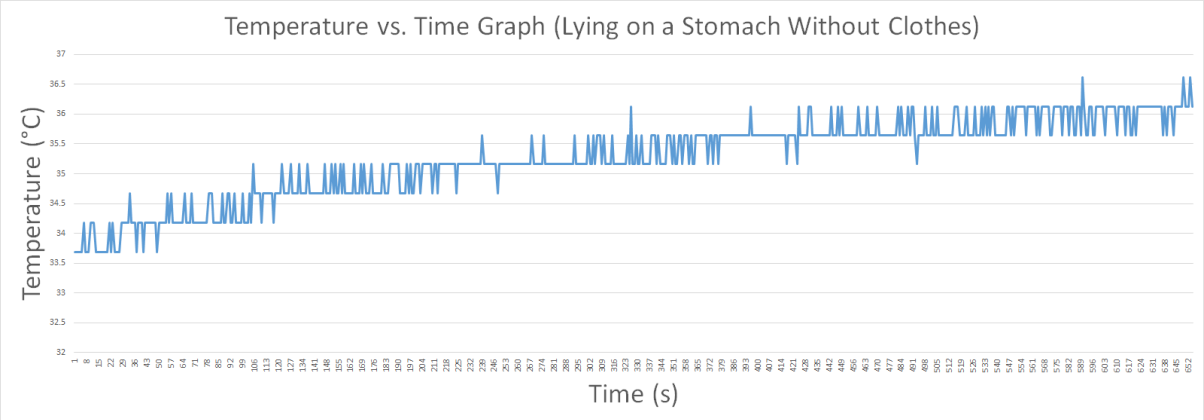

Figure 38. Thermometer Test 1

The second test is identical in its design to the first one, but in this one the subject was lying on his back. The stable normal body temperature reading was achieved after 458 seconds (which at the time of the test was 35.9 °C).

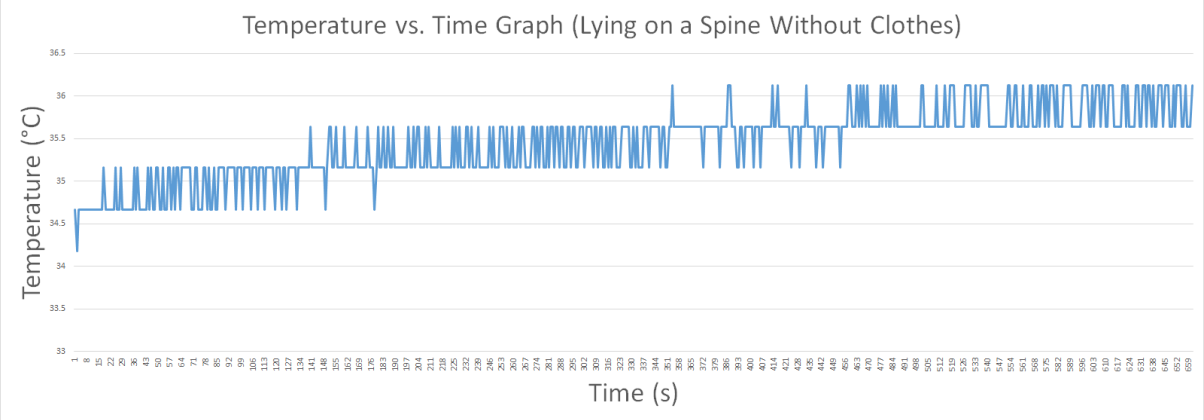

Figure 39. Thermometer Test 2

The final test incorporated four of the temperature sensors that measure temperature of the upper half of user's body. The lower four were discarded as it is highly unlikely that user's legs would have a continuous stable contact with either of them. Testing subject was lying on a spine with two of the sensors touching his body. The third sensor was placed in a "reference" position – under the armpit. Area under armpits, anus and mouth are the areas with highest thermal conductance and are utilized by medical professionals during the measurements of body temperatures. The fourth sensor served as a control sensor, meaning that it did come in contact with body at all. The sensors were cooled down and took longer to heat up. This condition was factored in. Predictably, once one of the sensors was put in a reference position, it took 172 (can be seen in figure 36: the rapid increase in the reference curve) seconds to get to the regular body temperature, whereas the first and the second sensors touching the random body parts needed 1220 and 1370 seconds respectively. Control sensor's readings never exceed 31.25 °C. Subject's body temperature was equal to 35.6 °C at the time of the test.

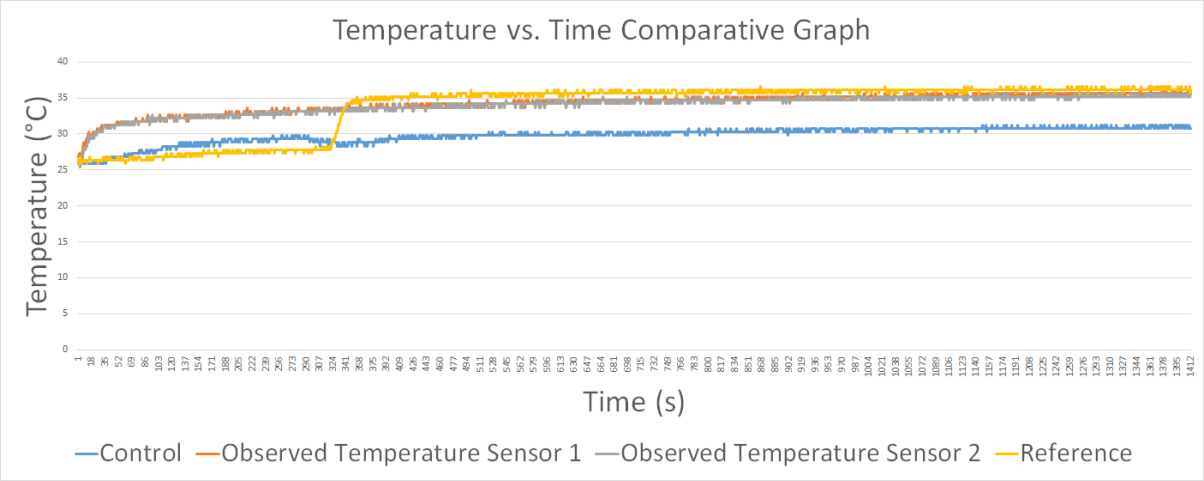

Figure 40. Thermometer Test 3

9.1.2.2. With Clothes On

Next, the series of tests shows how the sensors react to various areas of the body with clothes was done. Just like in the first series of tests, the test subject would on stomach and spine, and he would imitate the movements of a sleeping person. In the first case, it took 500 seconds to reach the normal body temperature (which at the time of the test was 36.0 °C) and in the second one – 541 seconds with the body temperature being 35.5 °C.

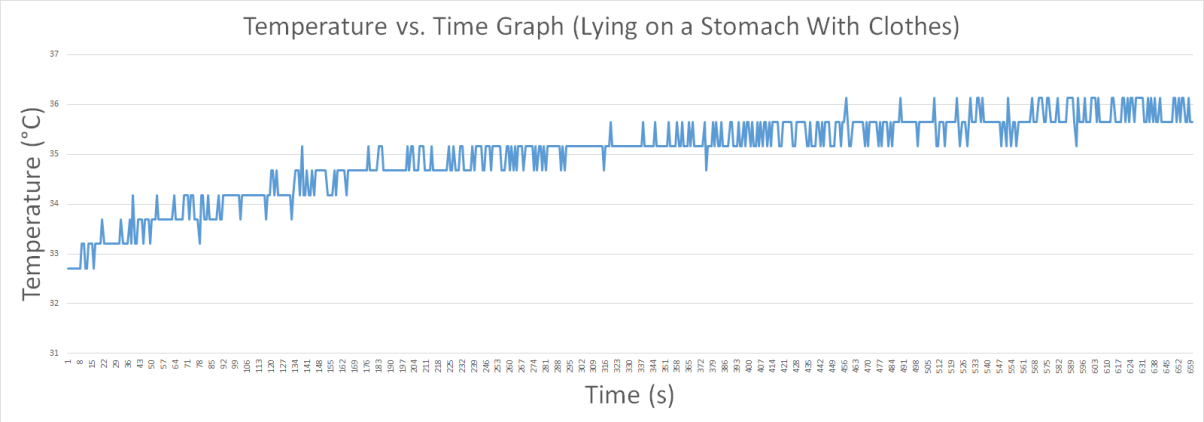

Figure 41. Thermometer Test 4

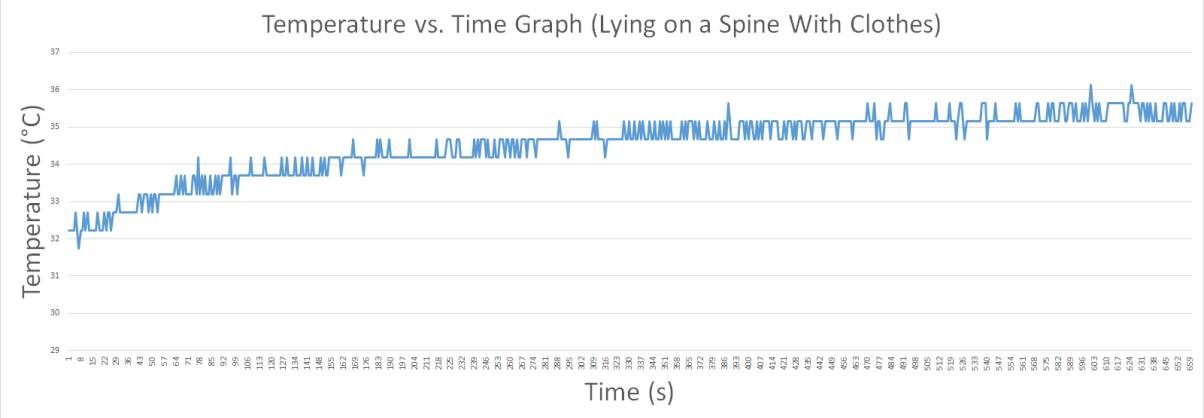

Figure 42. Thermometer Test 5

9.1.2.3. Thermometer Placed Inside the Mattress

Test with thermometer placed inside the mattress was done with clothes on and with randomized body positions (the subject would change them at his own volition). The stable normal body temperature reading was achieved after 2272 seconds (which at the time of the test was 36.3 °C).

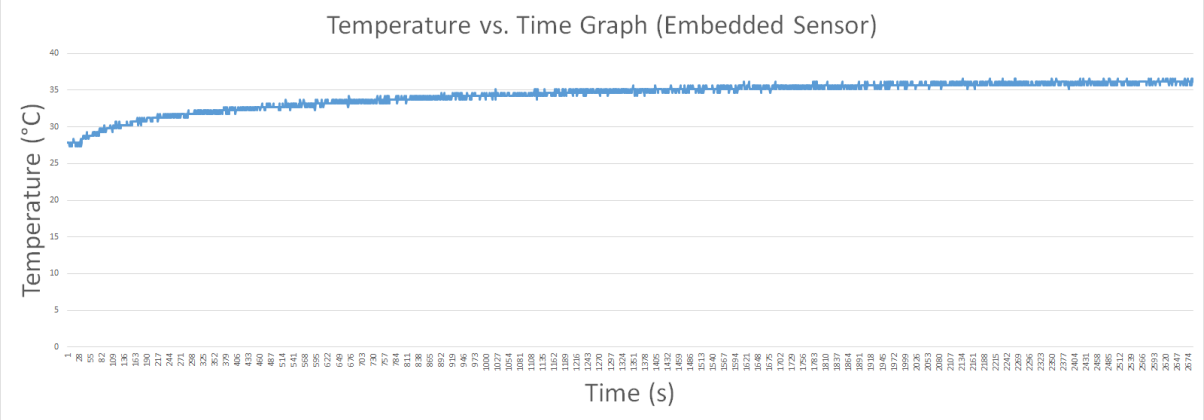

Figure 43. Thermometer Test 6

9.1.2.4. Discussion

The above tests clearly show a significant problem: the best locations to get body temperature are not convenient to have temperature sensors placed in them. Placing a sensor in these locations would severely restrict the movements of a person and/or provide discomfort during sleep. Fortunately, most of the people do not fall asleep immediately. And even when they fall asleep, their body temperature drops and it becomes easier to capture the moment when the person travels to the Kingdom of Morpheus. Therefore, it was decided that the temperature sensors should be placed along the anticipated location of user's torso (slightly above the horizontal axis).

9.1.3. Pressure Sensors

The pressure sensor is actually a Force Sensing Resistor (FSR). Because the microcontroller reads and outputs voltage, not the resistance, it was necessary to put the sensor in an electric circuit that would give voltage. The most basic setup that suits this purpose is a voltage divider, which has two resistors: FSR and a constant resistor. Without any applied pressure, FSR's resistance approaches infinity (its value is somewhere in the GΩ). Depending on the placement of FSR, two possible equations can be derived:

$$V_{Out} = \frac{R}{FSR+R} * V_{In} \quad (8.1.3.1)$$

$$V_{Out} = \frac{FSR}{R+FSR} * V_{In} \quad (8.1.3.2)$$

In the first configuration (shown in figure 40) output voltage is calculated though equation 8.1.3.1. There the absence pressure produces 0 Volts at the output, while maximal possible pressure produces 5 Volts. The second configuration (shown in figure 41) is characterized by equation 8.1.3.2. It is reverse coded: 5 Volts implies that there is no pressure and 0 Volts signifies that maximal pressure is applied.

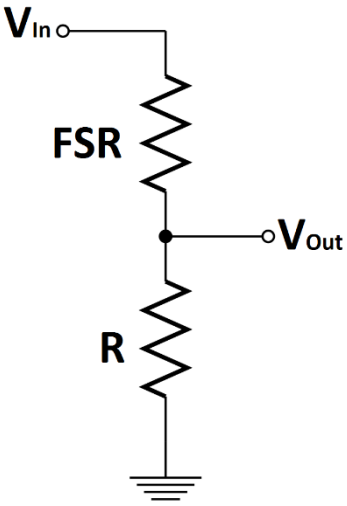

Figure 44. Voltage Divider Configuration 1

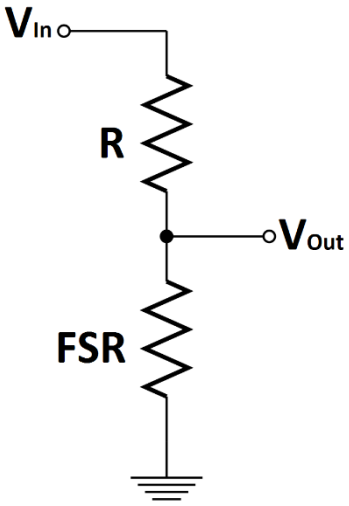

Figure 45. Voltage Divider Configuration 2 (Reverse Coded Results)

Pressure sensor was tested with three resistors: 100 kΩ (tested using the first configuration), 1 MΩ (tested using the second configuration) and 10 MΩ (tested using both configurations). 100 kΩ resistor case was used to determine whether pressure sensor detects lung and diaphragm pressure during breathing cessation, rapid breathing and snoring. In the first case, the pressure would stay constant, in the second – the amplitude would increase and snoring was not detected since it is not a result of lung movement complications. Unfortunately, the 100 kΩ resistor makes FSR oversensitive and, therefore, leads to irrelevant data. To combat this problem, larger sensors were tested and it was found that 10 MΩ was enough to prevent the oversensitivity problem, yet still sense the lung and diaphragm movement. The data presented in the graphs in a digital form. The 1 MΩ and 10 MΩ are only tested for sensitivity to preserve time.

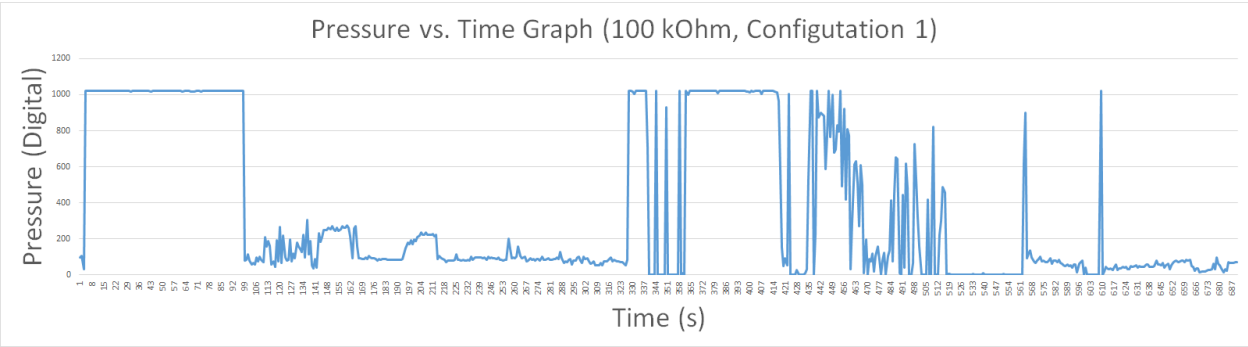

Figure 46. Pressure Sensor Test 1

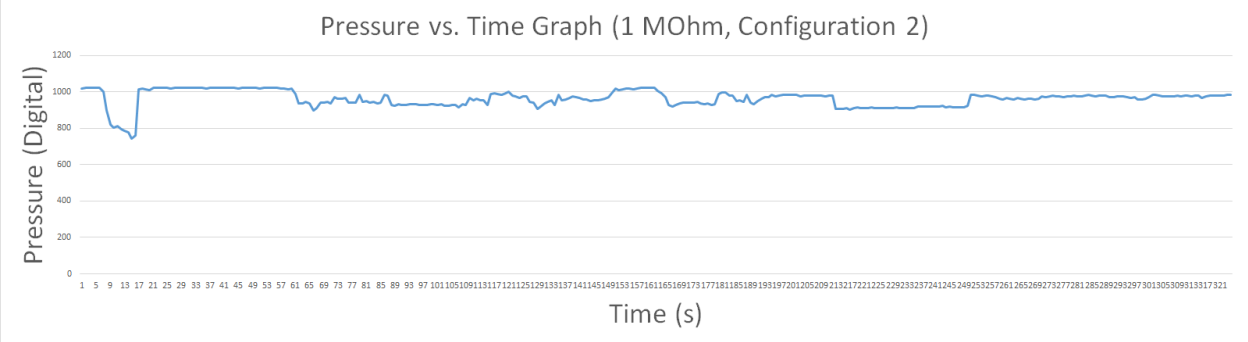

Figure 47. Pressure Sensor Test 2

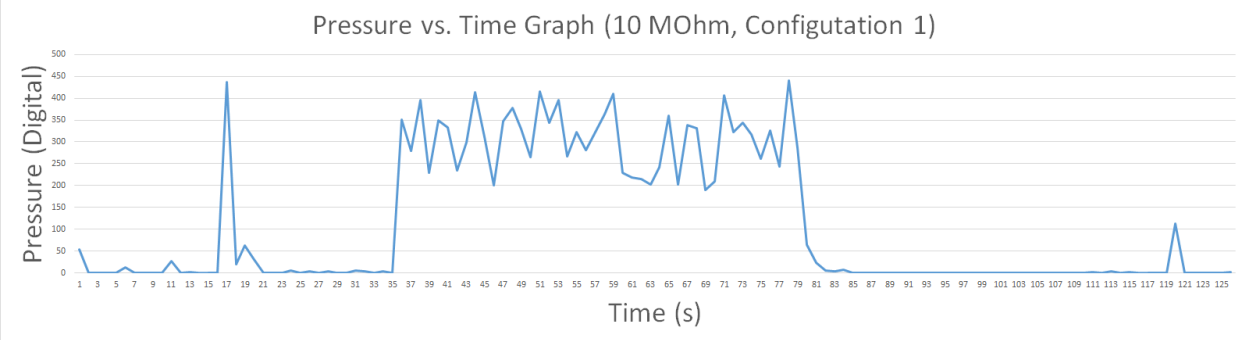

Figure 48. Pressure Sensor Test 3

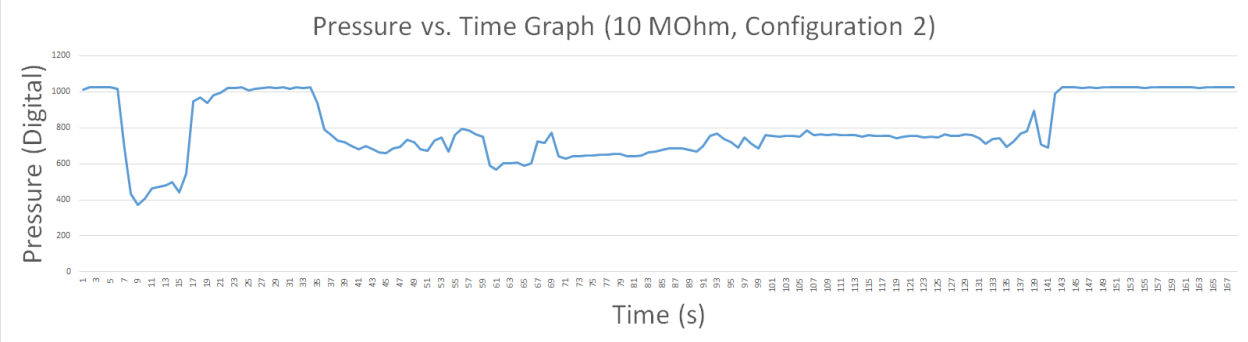

Figure 49. Pressure Sensor Test 4

Because it is important that the pressure sensors record the movement of lungs and diaphragm, the placement positions have to allow the sensors to perform their duty and accommodate for user's movements throughout the night. Fortunately, the area of lungs and diaphragm movement is almost equal to that of the torso and a person is forced to lie on the props with heart rate sensors on them. Therefore, the pressure sensors are placed between temperature sensors.

9.1.4. Moisture Sensors

Moisture sensors are used to measure the amount of sweat produced by a user during sleep and apnea episodes. A single test was done to determine whether the sensor detect sweat when:

1) A nude sweaty body touches the sensor directly.
2) A nude sweaty body touches the sensor through a sheet.
3) A clothed sweaty body touches the sensor through a sheet.

Even though, the sensitivity decreased with each additional layer placed over the sensor, it still detected the sweat and could differentiate the intensity as the graph clearly shows. The intensity was regulated by gradually wiping the sweat from the sensor.

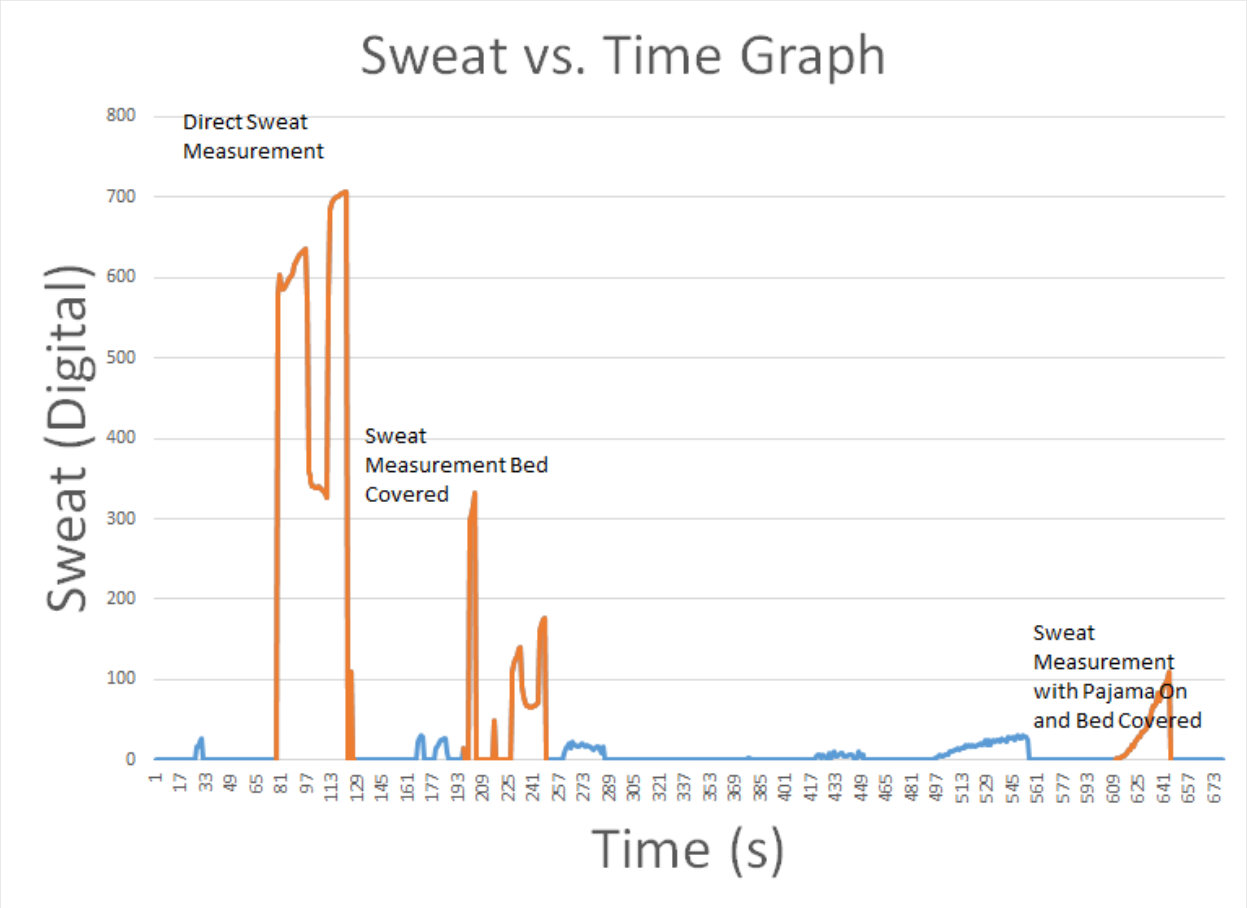

Figure 50. Moisture Sensor Test

9.1.5. Motion Sensors

Motion sensors testing is a simple one and was done by waving a hand in front of the sensor on different distances with no obstructions and with a sheet. Value of 1 represents the absence of movement, value of 0 represents movement. The table shows the sample results of the sensor's action:

No obstructions	
No motion	1

Hand waved in front of the sensor (distance is negligible)	0
Hand waved in front of the sensor (1 meter away)	0
Blocked by a sheet	
No motion	1
Hand waved in front of the sensor (distance is negligible)	0
Hand waved in front of the sensor (1 meter away)	0

9.2. Software

Testing was performed for each of the individual functions to identify whether these functions perform the intended action:

9.2.1. Sleep Start Detection Algorithm

Test Number: 1
Title: Sleep Start Detection Algorithm.
Function/s: findsleepStart ();
Description: The first algorithm that was tested was the algorithm to detect the start of a person's sleep. A dummy array of temperature readings was created and fed to the function which implements these algorithms. The algorithm could detect the exact moment when the user's temperature dropped by a factor of 2 degrees Fahrenheit.
Test case/s: Array of Temperatures: x = {70, 65, 76, 61, 45, 67};
Output:

```
Initial temperature in fahrenheit: 168.8
Array of temperatures in fahrenheit: 158.0 149.0 168.8 141.8 113.0 152.60000000000002
start of sleep 3
true
```

Figure 51. Sleep Start Detection Algorithm Test Case Outcome

Note: First two readings were ignored as bogus readings to simulate sensor warmup time.

9.2.2. Sound Analysis of Obstructive Sleep Apnea Algorithm

Test Number: 2
Title: Sound Analysis of Obstructive Sleep Apnea Algorithm
Function/s: analyzesoundForOSA ().
Description: An array containing 7200 dummy sound readings (all of which were 0 initially) was created. A few consecutive readings were changed to 80 and above to simulate snoring

and fed to the OSA sound analysis algorithm. After the changed readings were set back to 0 and fed again to the algorithm to simulate the negative test case.
Test case/s: Array of Sound Readings: sounds [7200].
Positive test case: Set few sets of consecutive values in array to 80 and above.
Negative test case: Reset the changed values back to 0.
Post Test Actions taken: None
Output:

```
Positive test cases true
Negative test cases false
```

Figure 52. Sound Analysis of Obstructive Sleep Apnea Algorithm Test Case Outcome

9.2.3. Sound Analysis of Central Sleep Apnea Algorithm

Test Number: 3
Title: Algorithm for Sound Analysis of Central Sleep Apnea
Function/s: analyzesoundForCSA ().
Description: An array containing 7200 dummy sound readings (all of which were 90 initially) was created. A few consecutive readings were changed to 0 and above to simulate breathing cessation and fed to the CSA sound analysis algorithm. After that the changed readings were set back to 90 and fed again to the algorithm to simulate the negative test case.
Test case/s: Array of Sound Readings: sounds [7200].
Positive test case: Set few sets of consecutive values in array to 0.
Negative test case: Reset the changed values back to 90.
Output:

```
Positive test cases true
Negative test cases false
```

Figure 53. Sound Analysis of Central Sleep Apnea Algorithm Test Case Outcome

9.2.4. Sleep Stages Determining Algorithm

Test Number: 4
Title: Sleep Stages Determining Algorithm
Function/s: detectSleepStages()
Description: A stream of pulse and motion sensor readings were fed to the sleep stage detection algorithm. The algorithm then analyzes the data and uses it to capture the REM and NREM intervals.
Test case/s:

```
double [] pulses = {20, 23, 60, 25, 67, 90, 34, 23};
boolean [] m1 = {false, false, true, false, false, false, false, false};
boolean [] m2 = {false, false, false, false, false, false, false, false};
boolean [] m3 = {false, false, true, false, false, false, false, false};
```

```
boolean [] m4 = {false, false, true, false, false, false, false, false};
boolean [] m5 = {false, false, true, false, false, false, false, false};
boolean [] m6 = {false, false, true, false, false, false, false, false};
boolean [] m7 = {false, false, true, false, false, false, false, false};
boolean [] m8 = {false, false, true, false, false, false, false, false};
```

Output:

```
Non-Rem Intervals
0.0 3.0
6.0 7.0
Rem Intervals
4.0 5.0
```

Figure 54. Sleep Stages Determining Algorithm Test Case Outcome

9.2.5. NREM and REM Pulse Analysis Algorithm

Test Number: 5
Title: NREM and REM Pulse Analysis Algorithm
Function/s: analyzeNRemPulses (), analyzeRemPulses ()
Description: The algorithms for analyzing pulses during the NREM and REM intervals were put to test by feeding an array of NREM and REM object interval objects. Both the algorithms should process their respective intervals. The ArrayLists were already filled by the sleep stage detection algorithms.
Test case/s: array of pulses, array of NREM intervals, array of rem intervals.
Output:

```
Likelyhood of apnea during NREM: false
Likelyhood of apnea during REM: true
```

Figure 55. NREM and REM Pulse Analysis Algorithm Test Case Outcome

10. Standards

Our system adheres to the following standards:

1) IEEE 802.15.4 – Wireless Sensor/Control Networks [21]. This standard offers the fundamental lower network layers for a type of wireless personal area network (WPAN). The primary goal of this kind of network is low-cost, low-speed communication between devices. The focus is also on low cost communication of nearby devices, which are close to each other. The power consumption of this set-up is very low. This protocol provides for a cheap and cost-effective means of communication, which will help us, stay within the project budget.
2) IEC 62353 – Standards for the Safety and Efficacy of Medical Electrical Equipment [22]. This standard attempts to cancel out any potential hazards in any electrical medical equipment. This standard requires testing to be performed after the initial start-up, after the repair and periodically. The testing must be done by a qualified electrician and proper documentation of the system must be provides about the device. Our electrical equipment is not as complex as what can be found in a commercial-grade device. Hence, we will easily be able to perform the tests and repairs in a qualified electronics repair shop. We decided to follow this standard during the implementation of our project because our smart bed is a medical device that directly interacts and records data from the patient. Therefore, safety and security is paramount.

11. Project Global, Economic and Societal Impact

This project may potentially have an impact on a field of user-friendly mobile medicine and non-obtrusive medical monitoring because it presents a convenient diagnosis system that can be used without prior medical knowledge in the home environment. Such system may help the medical professionals as they can concentrate on treating the disorders rather than go through a lengthy and prone-to-error process of diagnosis. The workload redistribution will not make the job of the doctors easier but it will make their work more effective and may contribute to the increased well-being of the world's population. Moreover, since the diagnosis is removed from the cost of the medical services, the individual medical expenses are decreased. This can significantly improve the financial situation for many people. However, there can be even more long-term effects. Our diagnosis system, for example, deals with sleep apnea. This disorder, if it goes undiagnosed and untreated, can later result in major psychological (depression, worsening of memory) and physiological (diabetes, increased risks of stroke and heart attack) damage. While the severe forms of the disorder like obstructive sleep apnea are easily spotted without any diagnosis, there is a significant portion of the affected people who have a mild form of the disorder and are unaware of it. Therefore, our system cannot only help the medical industry and the economic situation of the population but also prevents the major health problems in the future.

12. List of Components

12.1. Hardware

Component	Quantity	Purpose/Justification of use	Total Price (in USD)
SainSmart Arduino Mega2560	2	To receive data from sensors and send it to a server	31.98
SainSmart XBee Shield	1	To integrate XBee transmitter with microcontroller, which enabled wireless communication	9.99
XBee Module - Series 1 - 1mW	2	To send data to the server	39.92
Pulse Heart Rate Sensor	6	To measure user's heart rate	149.70
DFRobot LM35 Analog Linear Temperature Sensor	8	To measure user's body temperature	36.00
SparkFun Electret Microphone Breakout	4	To record breathing of a user	23.80
PIR (motion) sensor	8	To detect spontaneous body movements of a user	79.60
Flexiforce Pressure Sensor - 100lbs.	6	To measure the pressure applied by various regions of user's body	119.70
SainSmart Water Sensor	8	To detect sweating	71.84
Breadboard - Self-Adhesive	4	To hold all connections of electric circuit	19.80
10 MΩ Resistor	6	To make pressure sensor produce output	0.06
Wires	N/A		71.70
Mattress for one	1	To keep microcontrollers in and provide environment for user to sleep in	54.79
Polymer cover	2	To cover microcontrollers, wires and enclose/attach sensors to.	64.66

Mattress cover	1	To cover polymer, sensors and add comfort for user	41.10
8 AA Battery Holder (12 V)	2	To provide power for the system	2.74
PC	1	To act as a server, receive data, perform diagnosis	N/A
USB Cable	1	To connect XBee Explorer to PC	3.16
XBee Explorer	1	To integrate XBee receiver with PC	19.96

The total price of the components sums up to $840.41 (prices of tools and backup microcontrollers are not included). It is important to note that this is the price of all components for a prototype of the system. If this product is to be launched in mass production the price of this intelligent mattress would be much lower.

12.2. Software

Component	**Purpose/Justification of use**
Arduino IDE	Create and upload the code for microcontrollers
NetBeans IDE	Develop server application
Apache Derby Database Client	Create a database of readings and users

13. Glossary [23] [24] [25] [26]

Algorithm – Self-contained step-by-step set of operations to be performed.

Brachial Pulse – The pulse of the brachial artery, located in the upper arm, can be felt on the elbow's ventral aspect.

Carotid Pulse – pulse felt by placing index and middle fingers on the side of the neck near the windpipe.

Central Sleep Apnea – disorder in which breathing repeatedly stops and starts during sleep

Cross-platform (aka **multiplatform**) – refers to the capability of software or hardware to run identically on different platforms.

Diaphragm – dome-shaped sheet of muscle that is inserted into the lower ribs.

Electrocardiography – recording of the moment-to-moment electromotive forces of the Heart as projected onto various sites on the body's surface, delineated as a scalar function of Time.

Electroencephalography – recording of electric currents developed in the Brain by means of Electrodes applied to the Scalp, to the surface of the Brain, or placed within the substance of the Brain.

Electromyography – recording of the changes in electric potential of Muscle by means of surface or Needle Electrodes.

Electrooculography – recording of the average amplitude of the resting potential arising between the Cornea and the Retina in Light and Dark Adaptation as the eyes turn a standard distance to the right and the left.

Force-Sensing Resistor – material whose resistance changes when a force or pressure is applied.

Hypertension – high blood pressure.

Microcontroller – a highly integrated chip that contains all the components comprising a controller. Typically this includes a CPU, RAM, some form of ROM, I/O ports, and timers.

Non-rapid eye movement sleep – a period of sleep characterized by decreased metabolic activity, slowed breathing and heart rate, and the absence of dreaming.

Obstructive Sleep Apnea – disorder that causes complete or partial obstructions of the upper airway.

Oximetry – the determination of Oxygen-Hemoglobin saturation of Blood either by withdrawing a sample and passing it through a classical photoelectric oximeter or by Electrodes attached to some translucent part of the body like finger, earlobe, or Skin fold.

Polysomnography – simultaneous and continuous monitoring of several parameters during Sleep to study normal and abnormal Sleep. The study includes monitoring of Brain Waves, to assess Sleep Stages, and other physiological variables such as Breathing, Eye Movements, and Blood Oxygen levels which exhibit a disrupted pattern with Sleep disturbances.

Pulse – the regular expansion of an artery caused by the ejection of blood into the arterial system by the contractions of the heart.

QRS Complex – combination of three of the graphical deflections seen on a typical electrocardiogram.

R-R Interval – the time elapsing between two consecutive R waves in the electrocardiogram.

Radial Pulse – pulse felt on wrist just under the thumb.

Rapid eye movement sleep – a recurrent period of sleep, typically totaling about two hours a night, during which most dreaming occurs as the eyes move under closed lids and the skeletal muscles are deeply relaxed.

Sensor – a device that responds to a physical stimulus (as heat, light, sound, pressure, magnetism, or a particular motion) and transmits a resulting impulse (as for measurement or operating a control).

Server – the main computer in a network which provides files and services that are used by the other computers.

Sleep Apnea – brief periods of recurrent cessation of breathing during sleep that is caused especially by obstruction of the airway or a disturbance in the brain's respiratory center and is associated especially with excessive daytime sleepiness.

Slow-wave Sleep – a state of deep usually dreamless sleep that occurs regularly during a normal period of sleep with intervening periods of REM sleep and that is characterized by delta waves and a low level of autonomic physiological activity.

Stroke – Interruption of reduction of blood supply to brain.

14. Conclusion

An Intelligent Mattress for Diagnosis of Sleep Apnea was a challenging project. It demanded a lengthy research in multiple areas. Not only had it enhanced our knowledge of programming, hardware, networking and anatomy, but it also put our conflict resolution and team building skills at test. Amount of work, complexity of tasks and deadlines anxiety led to many stressful situation that had to be resolved effectively in order not to harm the project. Even though the project had several setbacks and some ideas were discarded, it still provided us with unique experience and knowledge.

15. References

[1] K. P. Strohl, "Sleep apnea in adults", Uptodate.com, 2016. [Online]. Available: http://www.uptodate.com/contents/sleep-apnea-in-adults-beyond-the-basics. [Accessed: 06- Apr- 2016].

[2] F. Farzana, "5 Important Factors That Link Sleep Apnea With Diabetes", Diabetes Treatment Guide, 2016. [Online]. Available: http://www.diabetestreatmentguide.org/5-important-factors-that-link-sleep-apnea-with-diabetes/. [Accessed: 07- Apr- 2016].

[3] M. Jennings, "How Pulse Oximetry Overnight Helps in the Treatment of Sleep Apnea", Adjustablebedsforyou.info, 2015. [Online]. Available: http://adjustablebedsforyou.info/2015/10/29/how-pulse-oximetry-overnight-helps-in-the-treatment-of-sleep-apnea/. [Accessed: 07- Apr- 2016].

[4] M. Kapoor and G. Greenough, "Home Sleep Tests for Obstructive Sleep Apnea (OSA)", The Journal of the American Board of Family Medicine, vol. 28, no. 4, pp. 504-509, 2015.

[5] "Detecting and quantifying apnea based on the ECG", Physionet.org, 2016. [Online]. Available: https://www.physionet.org/challenge/2000/. [Accessed: 26- Apr- 2016].

[6] U. Magalang, J. Dmochowski, S. Veeramachaneni, A. Draw, M. Mador, A. El-Solh and B. Grant, "Prediction of the Apnea-Hypopnea Index From Overnight Pulse Oximetry*", Chest, vol. 124, no. 5, pp. 1694-1701, 2003.

[7] "12-Lead ECG Placement", Emtresource.com, 2014. [Online]. Available: http://www.emtresource.com/resources/ecg/12-lead-ecg-placement/. [Accessed: 26- Apr- 2016].

[8] "12-Lead ECG Placement", Emtresource.com, 2014. [Online]. Available: http://www.emtresource.com/resources/ecg/12-lead-ecg-placement/. [Accessed: 07- Apr- 2016].

[9] OSCE Skills, Precordial leads in ECG [Online]. Available at: http://www.osceskills.com/resources/Precordial-leads-in-ECG.jpg.

[10] Npatchett, Derivation of the limb leads [Online]. Available at: https://upload.wikimedia.org/wikipedia/commons/1/19/Limb_leads_of_EKG.png [27 March 2015].

[11] “An Introduction to Machine Learning Theory and Its Applications: A Visual Tutorial with Examples,” Total Engineering Blog. [Online]. Available at:

https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer. [Accessed: 26-Apr-2016].

[12] K. Foster, R. Koprowski and J. Skufca, "Machine learning, medical diagnosis, and biomedical engineering research - commentary", BioMedical Engineering OnLine, vol. 13, no. 1, p. 94, 2014.

[13] "Sleep Apnea - 18 million Americans have it", Sleepdex.org, 2016. [Online]. Available: http://www.sleepdex.org/apnea.htm. [Accessed: 26- Apr- 2016].

[14] K. Malakuti and A. B. Albu, “Towards an Intelligent Bed Sensor: Non-intrusive Monitoring of Sleep Irregularities with Computer Vision Techniques,” 2010 20th International Conference on Pattern Recognition, 2010.

[15] M. Al-Mardini, F. Aloul, A. Sagahyroon and L. Al-Husseini, "Classifying obstructive sleep apnea using smartphones", Journal of Biomedical Informatics, vol. 52, pp. 251-259, 2014.

[16] M. M. A. Hashem, R. Shams, M. A. Kader, and M. A. Sayed, “Design and development of a heart rate measuring device using fingertip,” International Conference on Computer and Communication Engineering (ICCCE'10), 2010.

[17] M.M.A. Hashem, Rushdi Shams, Md. Abdul Kader, Md. Abu Sayed, “Design and Development of a Heart Rate Measuring Device using Fingertip”, Computer and Communication Engineering (ICCCE), 2010 International Conference on 11-12 May 2010, Malaysia, 2010, p. 5. DOI: 10.1109/ICCCE.2010.5556841

[18] M. Mendez, A. Bianchi, M. Matteucci, S. Cerutti and T. Penzel, "Sleep Apnea Screening by Autoregressive Models From a Single ECG Lead", IEEE Transactions on Biomedical Engineering, vol. 56, no. 12, pp. 2838-2850, 2009.

[19] L. Findley, S. Wilhoit and P. Suratt, "Apnea Duration and Hypoxemia During REM Sleep in Patients with Obstructive Sleep Apnea", Chest, vol. 87, no. 4, pp. 432-436, 1985.

[20] J. Loadsman and I. Wilcox, "Is obstructive sleep apnoea a rapid eye movement-predominant phenomenon?", British Journal of Anaesthesia, vol. 85, no. 3, pp. 354-358, 2000.

[21] “IEEE 802.15.4”, Wikipedia, 2016. [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.15.4.

[22] IEEE, “IEC 62353: Standards for the Safety and Efficacy of Medical Electrical Equipment”.

[23] Merriam-Webster Dictionary. Available at: http://www.merriam-webster.com/.

[24] Webopedia Online Tech Dictionary. Available at: http://www.webopedia.com/.

[25] Medical Dictionary Online. Available at: http://www.online-medical-dictionary.org/.

[26] The Free Dictionary. Available at: http://www.thefreedictionary.com/.

[27] wiseGEEK. Pulses [Online]. Available at: http://images.wisegeek.com/small/pulses-labeled-on-model.jpg

[28] Arduino. "Arduino Pin Current Limitations." [Online]. Available at: http://playground.arduino.cc/Main/ArduinoPinCurrentLimitations

[29] T. Courtney, "How Much Power Do Common Devices Consume?" [Online]. Available at: https://www.gozolt.com/blog/power-devices-consume/, Mar. 30, 2015

[30] J. Murphy, Y. Gitman. "Pulse Sensor Amped. Arduino Code v1.2 Walkthrough." [Online]. Available at: http://pulsesensor.com/pages/pulse-sensor-amped-arduino-v1dot1

[31] J. Lindblom. "Example sketch for SparkFun's force sensitive resistors" [Source Code]. Available at: https://learn.sparkfun.com/tutorials/force-sensitive-resistor-hookup-guide, Apr. 28, 2016

[32] M. V. Kamath, M. Watanabe, and A. Upton. "Heart rate variability (HRV) signal analysis: clinical applications." CRC Press, 2012.

[33] A. H. Khandoker, C. K. Karmakar, and M. Palaniswami. "Comparison of pulse rate variability with heart rate variability during obstructive sleep apnea." Medical engineering & physics, vol. 33, no. 2, pp. 204-209, 2011.

[34] T. Penzel, J. W. Kantelhardt, L. Grote, J. H. Peter and A. Bunde, "Comparison of detrended fluctuation analysis and spectral analysis for heart rate variability in sleep and sleep apnea," in IEEE Transactions on Biomedical Engineering, vol. 50, no. 10, pp. 1143-1151, Oct. 2003. doi: 10.1109/TBME.2003.817636. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1232484&isnumber=27615

[35] Top End Sports. "Measuring Heart Rate." [Online]. Available at: http://www.topendsports.com/testing/heart-rate-measure.htm

Appendix A. Microcontrollers Arduino Code

1) COE_491_Project_Slave_Microcontroller_Code.ino:

```
void setup ()
{
	pinMode (24, INPUT_PULLUP);
	pinMode (26, INPUT_PULLUP);
	pinMode (28, INPUT_PULLUP);
	pinMode (53, INPUT_PULLUP);
	Serial.begin (9600);
	Serial1.begin (9600);
}

void loop ()
{
	short index, sensory_data;
	for (index = 0; index < 16; index = index + 1)
	{
		sensory_data = analogRead (index);
		Serial1.write (sensory_data >> 8);
		Serial1.write (sensory_data);
	}
	sensory_data = digitalRead (24);
	Serial1.write (sensory_data >> 8);
	Serial1.write (sensory_data);
	sensory_data = digitalRead (26);
	Serial1.write (sensory_data >> 8);
	Serial1.write (sensory_data);
	sensory_data = digitalRead (28);
	Serial1.write (sensory_data >> 8);
	Serial1.write (sensory_data);
	sensory_data = digitalRead (53);
	Serial1.write (sensory_data >> 8);
	Serial1.write (sensory_data);
	delay (100);
}
```

2) Heart_Rate_Calculation_Code.ino:

```
volatile boolean first_heartbeat_flags [6] = {true, true, true, true, true, true},
second_heartbeat_flags [6] = {true, true, true, true, true, true}, pulse_flags [6] =
{false, false, false, false, false, false}, quantified_self_flags [6] = {false, false,
false, false, false, false};
volatile int analog_signals [6], pulse_wave_amplitudes [6] = {100, 100, 100, 100, 100,
100}, pulse_wave_peaks [6] = {512, 512, 512, 512, 512, 512}, pulse_wave_troughs [6]  =
{512, 512, 512, 512, 512, 512}, thresholds [6] = {512, 512, 512, 512, 512, 512};
volatile double interbeat_intervals [6] = {600.0, 600.0, 600.0, 600.0, 600.0, 600.0},
last_heartbeat_times [6] = {0, 0, 0, 0, 0, 0}, latest_interbeat_intervals [6] [10],
pulse_sampling_timers [6] = {0, 0, 0, 0, 0, 0}, sums_of_interbeat_intervals [6] = {0,
0, 0, 0, 0, 0}, times_since_the_last_heartbeat [6];

void Configure_Interruptions ()
{
	TCCR2A = 0x02;
	TCCR2B = 0x06;
	OCR2A = 0x1E;
	TIMSK2 = 0x02;
	sei ();
```

```
}

ISR (TIMER2_COMPA_vect)
{
	int index_1, index_2;
	cli ();
	for (index_1 = 0; index_1 < 6; index_1 = index_1 + 1)
	{
		analog_signals [index_1] = heart_rate_sensory_data [index_1];
		pulse_sampling_timers [index_1] = pulse_sampling_timers [index_1] + 0.5;
		times_since_the_last_heartbeat [index_1] = pulse_sampling_timers
[index_1] - last_heartbeat_times [index_1];
		if (analog_signals [index_1] < thresholds [index_1] &&
times_since_the_last_heartbeat [index_1] > (interbeat_intervals [index_1] / 5) * 3)
		{
			if (analog_signals [index_1] < pulse_wave_troughs [index_1])
			{
				pulse_wave_troughs [index_1] = analog_signals [index_1];
			}
		}
		if (analog_signals [index_1] > thresholds [index_1] && analog_signals
[index_1] > pulse_wave_peaks [index_1])
		{
			pulse_wave_peaks [index_1] = analog_signals [index_1];
		}
		if (times_since_the_last_heartbeat [index_1] > 0)
		{
			if ((analog_signals [index_1] > thresholds [index_1]) &&
(pulse_flags [index_1] == false) && (times_since_the_last_heartbeat [index_1] >
(interbeat_intervals [index_1] / 5) * 3) )
			{
				pulse_flags [index_1] = true;
				interbeat_intervals [index_1] = pulse_sampling_timers
[index_1] - last_heartbeat_times [index_1];
				last_heartbeat_times [index_1] = pulse_sampling_timers
[index_1];
				if (first_heartbeat_flags [index_1])
				{
					first_heartbeat_flags [index_1] = false;
					return;
				}
				if (second_heartbeat_flags [index_1])
				{
					second_heartbeat_flags [index_1] = false;
					for (index_2 = 0; index_2 < 10; index_2 = index_2 +
1)
					{
						latest_interbeat_intervals [index_1]
[index_2] = interbeat_intervals [index_1];
					}
				}
				for (index_2 = 0; index_2 < 9; index_2 = index_2 + 1)
				{
					latest_interbeat_intervals [index_1] [index_2] =
latest_interbeat_intervals [index_1] [index_2 + 1];
					sums_of_interbeat_intervals [index_1] =
sums_of_interbeat_intervals [index_1] + latest_interbeat_intervals [index_1]
[index_2];
				}
				latest_interbeat_intervals [index_1] [9] =
interbeat_intervals [index_1];
```

```
                        sums_of_interbeat_intervals [index_1] =
sums_of_interbeat_intervals [index_1] + latest_interbeat_intervals [index_1] [9];
                        sums_of_interbeat_intervals [index_1] =
sums_of_interbeat_intervals [index_1] / 10;
                        heart_rates [index_1] = (int) (60000 /
sums_of_interbeat_intervals [index_1]);
                        quantified_self_flags [index_1] = true;
                    }
            }
            if (analog_signals [index_1] < thresholds [index_1] && pulse_flags
[index_1] == true)
            {
                    pulse_flags [index_1] = false;
                    pulse_wave_amplitudes [index_1] = pulse_wave_peaks [index_1] -
pulse_wave_troughs [index_1];
                    thresholds [index_1] = pulse_wave_amplitudes [index_1] / 2 +
pulse_wave_troughs [index_1];
                    pulse_wave_peaks [index_1] = thresholds [index_1];
                    pulse_wave_troughs [index_1] = thresholds [index_1];
            }
            if (times_since_the_last_heartbeat [index_1] > 2500)
            {
                    thresholds [index_1] = 512;
                    pulse_wave_peaks [index_1] = 512;
                    pulse_wave_troughs [index_1] = 512;
                    last_heartbeat_times [index_1] = pulse_sampling_timers [index_1];
                    first_heartbeat_flags [index_1] = true;
                    second_heartbeat_flags [index_1] = true;
            }
    }
    sei ();
}
```

3) COE_491_Project_Master_Microcontoller_Code.ino:

```
const int sound_sampling_period = 100;
volatile int heart_rates [6];
volatile short heart_rate_sensory_data [6];

void setup ()
{
    Configure_Interruptions ();
    pinMode (24, INPUT_PULLUP);
    pinMode (26, INPUT_PULLUP);
    pinMode (28, INPUT_PULLUP);
    pinMode (53, INPUT_PULLUP);
    Serial.begin (9600);
    Serial1.begin (9600);
}

void loop ()
{
    boolean need_of_transmission_to_be_restarted = false;
    short index, maximal_signal_level = 0, minimal_signal_level = 1023, readings
[40];
    String output;
    long start_of_sound_sampling_period;
    for (index = 0; index < 20; index = index + 1)
    {
        readings [index] = (Serial1.read () << 8) | Serial1.read ();
```

```
                if (readings [index] > 1023 || readings [index] < 0)
                {
                        need_of_transmission_to_be_restarted = true;
                }
        }
        for (index = 16; index < 20; index = index + 1)
        {
                if (readings [index] > 1)
                {
                        need_of_transmission_to_be_restarted = true;
                }
        }
        for (index = 0; index < 3; index = index + 1)
        {
                readings [index + 20] = analogRead (index);
        }
        for (index = 0; index < 2; index = index + 1)
        {
                start_of_sound_sampling_period = millis ();
                while (millis () - start_of_sound_sampling_period <
sound_sampling_period)
                {
                        readings [index + 23] = analogRead (index + 3);
                        if (readings [index + 23] > maximal_signal_level)
                        {
                                maximal_signal_level = readings [index + 23];
                        }
                        else if (readings [index + 23] < minimal_signal_level);
                        {
                                minimal_signal_level = readings [index + 23];
                        }
                        readings [index + 23] = maximal_signal_level -
minimal_signal_level;
                }
        }
        for (index = 0; index < 2; index = index + 1)
        {
                start_of_sound_sampling_period = millis ();
                while (millis () - start_of_sound_sampling_period <
sound_sampling_period)
                {
                        if (readings [index + 3] > maximal_signal_level)
                        {
                                maximal_signal_level = readings [index + 3];
                        }
                        else if (readings [index + 3] < minimal_signal_level);
                        {
                                minimal_signal_level = readings [index + 3];
                        }
                        readings [index + 3] = maximal_signal_level -
minimal_signal_level;
                }
        }
        for (index = 0; index < 3; index = index + 1)
        {
                heart_rate_sensory_data [index] = readings [index + 9];
        }
        for (index = 3; index < 6; index = index + 1)
        {
                heart_rate_sensory_data [index] = analogRead (index + 6);
        }
        for (index = 0; index < 4; index = index + 1)
```

```
	{
		readings [index + 25] = analogRead (index + 5);
	}
	for (index = 0; index < 4; index = index + 1)
	{
		readings [index + 32] = analogRead (index + 12);
	}
	readings [36] = digitalRead (24);
	readings [37] = digitalRead (26);
	readings [38] = digitalRead (28);
	readings [39] = digitalRead (53);
	if (need_of_transmission_to_be_restarted)
	{
		output = "";
		for (index = 0; index < 39; index = index + 1)
		{
			output = output + 0 + " ";
		}
		output = output + 0;
		Serial.println (output);
		Serial1.end ();
		Serial1.begin (9600);
	}
	else
	{
		output = "";
		for (index = 0; index < 3; index = index + 1)
		{
			readings [index + 9] = heart_rates [index];
		}
		for (index = 3; index < 6; index = index + 1)
		{
			readings [index + 26] = heart_rates [index];
		}
		for (index = 0; index < 39; index = index + 1)
		{
			output = output + readings [index] + " ";
		}
		output = output + heart_rate_sensory_data [39];
		Serial.println (output);
	}
	delay (100);
}
```

Appendix B. Diagnostic Software Java Code

```
import jssc.SerialPort;
import jssc.SerialPortException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ArduinoReceiver implements Subject, Runnable
{
	private ArrayList <Observer> observers;
	private boolean [] motion_sensors_readings;
	private double [] body_temperatures, lungs_and_diaphragm_pressure_forces,
sound_levels, sweat_content_percentages;
	private int [] heart_rates;
	private String port;
	private Thread thread;

	public ArduinoReceiver (String port)
	{
		observers = new ArrayList <Observer> ();
		motion_sensors_readings = new boolean [8];
		body_temperatures = new double [8];
		lungs_and_diaphragm_pressure_forces = new double [6];
		sweat_content_percentages = new double [8];
		sound_levels = new double [4];
		heart_rates = new int [6];
		this.port = port;
		thread = new Thread (this);
		thread.start ();
	}

	private static StringBuilder ExtractCommand (int command [])
	{
		int index = 0;
		StringBuilder command_string = new StringBuilder ();
		while (command [index] != 13)
		{
			command_string.append ((char) command [index]);
			index = index + 1;
		}
		return command_string;
	}

	public void NotifyObservers ()
	{
		int index;
		for (index = 0; index < observers.size (); index = index + 1)
		{
			Observer observer = observers.get (index);
			observer.Update (motion_sensors_readings, body_temperatures,
					lungs_and_diaphragm_pressure_forces,
sound_levels,
					sweat_content_percentages, heart_rates);
		}
	}

	public void RegisterObserver (Observer observer)
```

```
	{
		observers.add (observer);
	}

	public void RemoveObsever (Observer observer)
	{
		int index = observers.indexOf (observer);
		if (index >= 0)
		{
			observers.remove (index);
		}
	}

	public void ReadSensors () throws SerialPortException
	{
		int index;
		SerialPort serial_port = new SerialPort (port);
		StringBuffer sensory_data_string_buffer = new StringBuffer ();
		try
		{
			serial_port.openPort ();
			serial_port.setParams (9600, 8, 1, 0);
			while (true)
			{
				byte character = serial_port.readBytes (1) [0];
				if (character != '\n')
				{
					sensory_data_string_buffer.append ((char)
character);
				}
				else
				{
					try
					{
						String tokens [] =
sensory_data_string_buffer
							.toString ().split (" ");
						for (index = 0; index < 4; index = index +
1)
						{
							if (Integer.parseInt (tokens [index
+ 16]) == 0)
							{
								motion_sensors_readings
[index] = false;
							}
							else
							{
								motion_sensors_readings
[index] = true;
							}
						}
						for (index = 0; index < 4; index = index +
1)
						{
							if (Integer.parseInt (tokens [index
+ 36].trim ()) == 0)
							{
```

```
                                                motion_sensors_readings
[index + 4] = false;
                                        }
                                        else
                                        {
                                                motion_sensors_readings
[index + 4] = true;
                                        }
                                }
                                for (index = 0; index < 4; index = index +
1)
                                {
                                        body_temperatures [index] = Integer
                                                .parseInt (tokens
[index + 12]) * 500.0 / 1023.0;
                                }
                                for (index = 0; index < 4; index = index +
1)
                                {
                                        body_temperatures [index + 4] =
Integer
                                                .parseInt (tokens
[index + 32]) * 500.0 / 1023.0;
                                }
                                for (index = 0; index < 3; index = index +
1)
                                {
                                        lungs_and_diaphragm_pressure_forces
[index] = 10000000.0 * ((5.0 / (Integer
                                                .parseInt (tokens
[index]) * 5.0 / 1023.0)) - 1.0);
                                        if
(lungs_and_diaphragm_pressure_forces [index] <= 600.0)
                                        {

      lungs_and_diaphragm_pressure_forces [index] = (((1.0 /
lungs_and_diaphragm_pressure_forces [index]) - 0.00075) / 0.00000032639) * 9.81;
                                        }
                                        else
                                        {

      lungs_and_diaphragm_pressure_forces [index] = ((1.0 /
lungs_and_diaphragm_pressure_forces [index]) / 0.000000642857) * 9.81;
                                        }
                                }
                                for (index = 0; index < 3; index = index +
1)
                                {
                                        lungs_and_diaphragm_pressure_forces
[index + 3] = 10000000.0 * ((5.0 / (Integer
                                                .parseInt (tokens
[index + 20]) * 5.0 / 1023.0)) - 1.0);
                                        if
(lungs_and_diaphragm_pressure_forces [index + 3] <= 600.0)
                                        {

      lungs_and_diaphragm_pressure_forces [index + 3] = (((1.0 /
lungs_and_diaphragm_pressure_forces [index + 3]) - 0.00075) / 0.00000032639) *
9.81;
```

```
                                        }
                                        else
                                        {
        lungs_and_diaphragm_pressure_forces [index + 3] = ((1.0 /
lungs_and_diaphragm_pressure_forces [index + 3]) / 0.000000642857) * 9.81;
                                        }
                                }
                                for (index = 0; index < 2; index = index +
1)
                                {
                                        if (Integer.parseInt (tokens [index
+ 3]) == 0)
                                        {
                                                sound_levels [index] = 0;
                                        }
                                        else
                                        {
                                                sound_levels [index] = 20 *
Math
                                                        .log10 ((Integer
        .parseInt (tokens [index + 3]) * 5.0 / 1023.0) / 0.0001);
                                        }
                                }
                                for (index = 0; index < 2; index = index +
1)
                                {
                                        if (Integer.parseInt (tokens [index
+ 23]) == 0)
                                        {
                                                sound_levels [index] = 0;
                                        }
                                        else
                                        {
                                                sound_levels [index + 2] = 20
* Math
                                                        .log10 ((Integer
        .parseInt (tokens [index + 23]) * 5.0 / 1023.0) / 0.0001);
                                        }
                                }
                                for (index = 0; index < 4; index = index +
1)
                                {
                                        sweat_content_percentages [index] =
(Integer
                                                .parseInt (tokens
[index + 5]) / 1023.0) * 100.0;
                                }
                                for (index = 0; index < 4; index = index +
1)
                                {
                                        sweat_content_percentages [index +
4] = (Integer
                                                .parseInt (tokens
[index + 25]) / 1023.0) * 100.0;
                                }
```

```
                                        for (index = 0; index < 3; index = index +
1)
                                        {
                                                heart_rates [index] = Integer
                                                        .parseInt (tokens
[index + 9]);
                                        }
                                        for (index = 3; index < 6; index = index +
1)
                                        {
                                                heart_rates [index] = Integer
                                                        .parseInt (tokens
[index + 29]);
                                        }
                                        NotifyObservers ();
                                        sensory_data_string_buffer.setLength (0);
                                }
                                catch (Exception exception_1)
                                {
                                        for (index = 0; index <
motion_sensors_readings.length; index = index + 1)
                                        {
                                                motion_sensors_readings [index] =
true;
                                        }
                                        for (index = 0; index <
body_temperatures.length; index = index + 1)
                                        {
                                                body_temperatures [index] = 0;
                                        }
                                        for (index = 0; index <
lungs_and_diaphragm_pressure_forces.length; index = index + 1)
                                        {
                                                lungs_and_diaphragm_pressure_forces
[index] = 0;
                                        }
                                        for (index = 0; index <
sound_levels.length; index = index + 1)
                                        {
                                                sound_levels [index] = 0;
                                        }
                                        for (index = 0; index <
sweat_content_percentages.length; index = index + 1)
                                        {
                                                sweat_content_percentages [index] =
0;
                                        }
                                        for (index = 0; index <
heart_rates.length; index = index + 1)
                                        {
                                                heart_rates [index] = 0;
                                        }
                                        NotifyObservers ();
                                        sensory_data_string_buffer.setLength (0);
                                }
                        }
                }
        }
        catch (SerialPortException exception_2)
```

```
                {
                        System.out.println (exception_2);
                }
                serial_port.closePort ();
        }

        public void run ()
        {
                try
                {
                        ReadSensors ();
                }
                catch (SerialPortException exception)
                {
                        exception.printStackTrace ();
                }
        }
}

import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.LinkedList;
import java.util.Properties;
import java.util.Vector;
import java.util.stream.Stream;
import javax.sql.rowset.CachedRowSet;
import javax.sql.rowset.RowSetProvider;

public class Driver
{
        public static boolean awake = true;
        public static boolean onbed = false;
        public static Timestamp onbedts;
        public static Timestamp offbedts;

        public static boolean returnsweat (ArrayList <Double> sweats)
        {
                int sweatcount = 0;
                for (int i = 0; i < sweats.size (); i++)
                {
                        if (sweats.get (i) > 0)
                        {
                                ++sweatcount;
                                if (sweatcount > 5)
                                {
                                        return true;
                                }
                        }
                }
                return false;
        }

        static double [] returnDoubleArray (ArrayList <Double> x)
        {
                double [] darry = new double [x.size ()];
                for (int i = 0; i < x.size (); i++)
                {
```

```
                darry [i] = x.get (i).doubleValue ();
        }
        return darry;
}

static boolean [] returnBooleanArray (ArrayList <Boolean> x)
{
        boolean [] barry = new boolean [x.size ()];
        for (int i = 0; i < x.size (); i++)
        {
                barry [i] = x.get (i).booleanValue ();
        }
        return barry;
}

public static void main (String [] args) throws InterruptedException,
                SQLException
{
        ArduinoReceiver test = new ArduinoReceiver ("COM4");
        int tempcount = 0;
        SensoryData thing = new SensoryData (test);
        double [] latest_temperatures;
        while (!onbed)
        {
                double [] latest_pressures = thing
                                .Get_Lungs_And_Diaphragm_Pressure_Forces ();
                int zerocount = 0;
                latest_temperatures = thing.Get_Body_Temperatures ();
                for (int i = 0; i < 8; i++)
                {
                        if (latest_temperatures [i] == 0)
                        {
                                ++tempcount;
                        }
                }
                for (int i = 0; i < 6; i++)
                {
                        if (latest_pressures [i] == 0)
                        {
                                ++zerocount;
                        }
                }
                if ((zerocount < 6) && (tempcount != 0))
                {
                        onbed = true;
                        System.out.println ("ZEROCOUNT= " + zerocount);
                        onbedts = new Timestamp (new Date ().getTime ());
                        for (int i = 0; i < latest_pressures.length; i++)
                        {
                                System.out.println (latest_pressures [i]);
                        }
                        break;
                }
                System.out.println ("Please lay down");
        }
        System.out.println ("PERSON DETECTED AT " + onbedts);
        while (onbed)
        {
                thing.Output ();
```

```
                Thread.sleep (1000);
                double [] latest_pressures = thing
                                .Get_Lungs_And_Diaphragm_Pressure_Forces ();
                int zerocount = 0;
                tempcount = 0;
                latest_temperatures = thing.Get_Body_Temperatures ();
                for (int i = 0; i < 8; i++)
                {
                        if (latest_temperatures [i] == 0)
                        {
                                ++tempcount;
                        }
                }
                for (int i = 0; i < 6; i++)
                {
                        if (latest_pressures [i] == 0)
                        {
                                ++zerocount;
                        }
                }
                if ((zerocount == 6) && tempcount != 8)
                {
                        onbed = false;
                        System.out.println ("Onbed= false");
                        offbedts = new Timestamp (new Date ().getTime ());
                        break;
                }
        }
        System.out.println ("PERSON GOT OFF AT " + offbedts);
        Timestamp t = java.sql.Timestamp.valueOf ("2016-12-03
19:05:34.922");
        CachedRowSet crs = null;
        String username = "Saurabh";
        crs = RowSetProvider.newFactory ().createCachedRowSet ();
        crs.setUrl ("jdbc:derby://localhost:1527/patient");
        crs.setUsername ("a");
        crs.setPassword ("a");
        ArrayList <Timestamp> times = new ArrayList ();
        ArrayList <Double> pressures = new ArrayList ();
        ArrayList <Double> pressure1 = new ArrayList ();
        ArrayList <Double> pressure2 = new ArrayList ();
        ArrayList <Double> pressure3 = new ArrayList ();
        ArrayList <Double> pressure4 = new ArrayList ();
        ArrayList <Double> pressure5 = new ArrayList ();
        ArrayList <Double> pressure6 = new ArrayList ();
        ArrayList <Double> sounds = new ArrayList ();
        ArrayList <Boolean> motion1 = new ArrayList ();
        ArrayList <Boolean> motion2 = new ArrayList ();
        ArrayList <Boolean> motion3 = new ArrayList ();
        ArrayList <Boolean> motion4 = new ArrayList ();
        ArrayList <Boolean> motion5 = new ArrayList ();
        ArrayList <Boolean> motion6 = new ArrayList ();
        ArrayList <Boolean> motion7 = new ArrayList ();
        ArrayList <Boolean> motion8 = new ArrayList ();
        ArrayList <Double> sweat1 = new ArrayList ();
        ArrayList <Double> sweat2 = new ArrayList ();
        ArrayList <Double> sweat3 = new ArrayList ();
        ArrayList <Double> sweat4 = new ArrayList ();
        ArrayList <Double> sweat5 = new ArrayList ();
```

```
            ArrayList <Double> sweat6 = new ArrayList ();
            ArrayList <Double> sweat7 = new ArrayList ();
            ArrayList <Double> sweat8 = new ArrayList ();
            ArrayList <Double> hearts = new ArrayList ();
            ArrayList <Double> mic1 = new ArrayList ();
            ArrayList <Double> mic2 = new ArrayList ();
            ArrayList <Double> mic3 = new ArrayList ();
            ArrayList <Double> mic4 = new ArrayList ();
            ArrayList <Double> temperatures = new ArrayList ();
            ArrayList <Double> temp1 = new ArrayList ();
            ArrayList <Double> temp2 = new ArrayList ();
            ArrayList <Double> temp3 = new ArrayList ();
            ArrayList <Double> temp4 = new ArrayList ();
            ArrayList <Double> temp5 = new ArrayList ();
            ArrayList <Double> temp6 = new ArrayList ();
            ArrayList <Double> temp7 = new ArrayList ();
            ArrayList <Double> temp8 = new ArrayList ();
            crs.setCommand ("select * from pressures where username= ? and
moment>= ? and moment<= ?");
            crs.setTimestamp (2, onbedts);
            crs.setTimestamp (3, offbedts);
            crs.setString (1, username);
            crs.execute ();
            double ps1 = 0, ps2 = 0, ps3 = 0, ps4 = 0, ps5 = 0, ps6 = 0;
            while (crs.next ())
            {
                  Timestamp t1 = crs.getTimestamp ("moment");
                  ps1 = crs.getDouble ("press1");
                  ps2 = crs.getDouble ("press2");
                  ps3 = crs.getDouble ("press3");
                  ps4 = crs.getDouble ("press4");
                  ps5 = crs.getDouble ("press5");
                  ps6 = crs.getDouble ("press6");
                  pressure1.add (ps1);
                  pressure2.add (ps2);
                  pressure3.add (ps3);
                  pressure4.add (ps4);
                  pressure5.add (ps5);
                  pressure6.add (ps6);
                  times.add (t1);
                  pressures.add (ps1);
                  pressures.add (ps2);
                  pressures.add (ps3);
                  pressures.add (ps4);
                  pressures.add (ps5);
                  pressures.add (ps6);
            }
            for (int i = 0; i < pressure1.size (); i++)
            {
                  System.out.println (pressure4.get (i));
            }
            crs.setCommand ("select * from heartrates where username=? and
moment>= ? and moment<= ? ");
            crs.setTimestamp (2, onbedts);
            crs.setTimestamp (3, offbedts);
            crs.setString (1, username);
            crs.execute ();
            int i = 0;
            while (crs.next () && i < pressure1.size ())
```

```
            {
                  t = crs.getTimestamp ("moment");
                  double hr1 = crs.getDouble ("heart1");
                  double hr2 = crs.getDouble ("heart2");
                  double hr3 = crs.getDouble ("heart3");
                  double hr4 = crs.getDouble ("heart4");
                  double hr5 = crs.getDouble ("heart5");
                  double hr6 = crs.getDouble ("heart6");
                  if ((pressure1.get (i) + pressure4.get (i)) > (pressure2.get
(i) + pressure5
                              .get (i)))
                  {
                        if (pressure1.get (i) > pressure4.get (i))
                        {
                              hearts.add (hr1);
                        }
                        else
                        {
                              hearts.add (hr4);
                        }
                  }
                  else if ((pressure2.get (i) + pressure5.get (i)) > (pressure3
                              .get (i) + pressure6.get (i)))
                  {
                        if (ps2 > ps5)
                        {
                              hearts.add (hr2);
                        }
                        else
                        {
                              hearts.add (hr5);
                        }
                  }
                  else
                  {
                        if (pressure3.get (i) > pressure6.get (i))
                        {
                              hearts.add (hr3);
                        }
                        else
                        {
                              hearts.add (hr6);
                        }
                  }
                  ++i;
            }
            crs.setCommand ("select * from motions where username=? and
moment>= ? and moment<=? ");
            crs.setTimestamp (2, onbedts);
            crs.setTimestamp (3, offbedts);
            crs.setString (1, username);
            crs.execute ();
            i = 0;
            while (crs.next ())
            {
                  t = crs.getTimestamp ("moment");
                  motion1.add (crs.getBoolean ("motion1"));
                  motion2.add (crs.getBoolean ("motion2"));
                  motion3.add (crs.getBoolean ("motion3"));
```

```
                motion4.add (crs.getBoolean ("motion4"));
                motion5.add (crs.getBoolean ("motion5"));
                motion6.add (crs.getBoolean ("motion6"));
                motion7.add (crs.getBoolean ("motion7"));
                motion8.add (crs.getBoolean ("motion8"));
                ++i;
            }
            crs.setCommand ("select * from sweats where username= ? and
moment>= ? and moment<= ?");
            crs.setTimestamp (2, onbedts);
            crs.setTimestamp (3, offbedts);
            crs.setString (1, username);
            crs.execute ();
            i = 0;
            while (crs.next ())
            {
                sweat1.add (crs.getDouble ("sweat1"));
                sweat2.add (crs.getDouble ("sweat2"));
                sweat3.add (crs.getDouble ("sweat3"));
                sweat4.add (crs.getDouble ("sweat4"));
                sweat5.add (crs.getDouble ("sweat5"));
                sweat6.add (crs.getDouble ("HEART6"));
                sweat7.add (crs.getDouble ("sweat7"));
                sweat8.add (crs.getDouble ("sweat8"));
                ++i;
            }
            crs.setCommand ("select * from microphones where username= ? and
moment>= ? and moment<= ? ");
            crs.setTimestamp (2, onbedts);
            crs.setString (1, username);
            crs.setTimestamp (3, offbedts);
            crs.execute ();
            boolean hassweat = (returnsweat (sweat1) || returnsweat (sweat2)
                        || returnsweat (sweat3) || returnsweat (sweat4)
                        || returnsweat (sweat5) || returnsweat (sweat6)
                        || returnsweat (sweat7) || returnsweat (sweat8));
            i = 0;
            while (crs.next ())
            {
                mic1.add (crs.getDouble ("microphone1"));
                mic2.add (crs.getDouble ("microphone2"));
                mic3.add (crs.getDouble ("microphone3"));
                mic4.add (crs.getDouble ("microphone4"));
                ++i;
            }
            crs.setCommand ("select * from temperatures where username= ? and
moment>= ? and moment<= ? ");
            crs.setTimestamp (2, onbedts);
            crs.setTimestamp (3, offbedts);
            crs.setString (1, username);
            crs.execute ();
            i = 0;
            while (crs.next ())
            {
                temp1.add (crs.getDouble ("temp1"));
                temp2.add (crs.getDouble ("temp2"));
                temp3.add (crs.getDouble ("temp3"));
                temp4.add (crs.getDouble ("temp4"));
                temp5.add (crs.getDouble ("temp5"));
```

```
                temp6.add (crs.getDouble ("temp6"));
                temp7.add (crs.getDouble ("temp7"));
                temp8.add (crs.getDouble ("temp8"));
                System.out.println ("Temp1 " + temp1.get (i));
                ++i;
            }
            i = 0;
            while (i < temp1.size ())
            {
                if ((temp1.get (i) + temp2.get (i)) > (temp3.get (i) + temp4
                            .get (i)))
                {
                    if ((temp1.get (i) + temp2.get (i)) > (temp2.get (i) +
temp3
                                    .get (i)))
                    {
                        if (temp1.get (i) > temp2.get (i))
                        {
                            temperatures.add (temp1.get (i));
                        }
                        else
                        {
                            temperatures.add (temp2.get (i));
                        }
                    }
                    else
                    {
                        if (temp2.get (i) > temp3.get (i))
                        {
                            temperatures.add (temp2.get (i));
                        }
                        else
                        {
                            temperatures.add (temp3.get (i));
                        }
                    }
                }
                else
                {
                    if ((temp3.get (i) + temp4.get (i)) > (temp2.get (i) +
temp3
                                    .get (i)))
                    {
                        if (temp3.get (i) > temp4.get (i))
                        {
                            temperatures.add (temp3.get (i));
                        }
                        else
                        {
                            temperatures.add (temp4.get (i));
                        }
                    }
                    else
                    {
                        if (temp2.get (i) > temp3.get (i))
                        {
                            temperatures.add (temp2.get (i));
                        }
                        else
```

```
                        {
                                temperatures.add (temp3.get (i));
                        }
                }
        }
        ++i;
}
System.out.println ("Sizze of tem perature" + temperatures.size ());
double hrts [] = returnDoubleArray (hearts);
boolean mtn1 [] = returnBooleanArray (motion1);
boolean mtn2 [] = returnBooleanArray (motion2);
boolean mtn3 [] = returnBooleanArray (motion3);
boolean mtn4 [] = returnBooleanArray (motion4);
boolean mtn5 [] = returnBooleanArray (motion5);
boolean mtn6 [] = returnBooleanArray (motion6);
boolean mtn7 [] = returnBooleanArray (motion7);
boolean mtn8 [] = returnBooleanArray (motion8);
double swt1 [] = returnDoubleArray (sweat1);
double swt2 [] = returnDoubleArray (sweat2);
double swt3 [] = returnDoubleArray (sweat3);
double swt4 [] = returnDoubleArray (sweat4);
double swt5 [] = returnDoubleArray (sweat5);
double swt6 [] = returnDoubleArray (sweat6);
double swt7 [] = returnDoubleArray (sweat7);
double swt8 [] = returnDoubleArray (sweat8);
double prs1 [] = returnDoubleArray (pressure1);
double prs2 [] = returnDoubleArray (pressure2);
double prs3 [] = returnDoubleArray (pressure3);
double prs4 [] = returnDoubleArray (pressure4);
double prs5 [] = returnDoubleArray (pressure5);
double prs6 [] = returnDoubleArray (pressure6);
double temps [] = returnDoubleArray (temperatures);
double m1 [] = returnDoubleArray (mic1);
double m2 [] = returnDoubleArray (mic2);
double m3 [] = returnDoubleArray (mic3);
double m4 [] = returnDoubleArray (mic4);
for (i = 0; i < mtn1.length; i++)
{
        System.out.println ("MTN1 " + mtn1 [i]);
}
for (i = 0; i < prs2.length; i++)
{
        System.out.println ("PRS2 " + prs2 [i]);
}
SleepAnalyzer analyzer = new SleepAnalyzer ();
analyzer.findsleepStart (temps);
boolean m1OSAanalyze = analyzer.analyzesoundForOBSA (m1);
boolean m2OSAanalyze = analyzer.analyzesoundForOBSA (m2);
boolean m3OSAanalyze = analyzer.analyzesoundForOBSA (m3);
boolean m4OSAanalyze = analyzer.analyzesoundForOBSA (m4);
boolean m1CSAanalyze = analyzer.analyzesoundForCSA (m1);
boolean m2CSAanalyze = analyzer.analyzesoundForCSA (m2);
boolean m3CSAanalyze = analyzer.analyzesoundForCSA (m3);
boolean m4CSAanalyze = analyzer.analyzesoundForCSA (m4);
boolean snoring = false, cessation = false;
if (m1OSAanalyze || m2OSAanalyze || m3OSAanalyze || m4OSAanalyze)
{
        snoring = true;
}
```

```
            if (m1CSAanalyze || m2CSAanalyze || m3CSAanalyze || m4CSAanalyze)
            {
                  cessation = true;
            }
            analyzer.detectSleepStages (hrts, mtn1, mtn2, mtn3, mtn4, mtn5,
mtn6,
                        mtn7, mtn8);
            boolean abNREMPulse = analyzer.analyzeNRemPulses (hrts);
            boolean abREMPulse = analyzer.analyzeRemPulses (hrts);
            ArrayList <Double> press;
            press = new ArrayList ();
            double maxPress;
            for (i = 0; i < pressure1.size (); i++)
            {
                  double [] tempArray = new double [6];
                  tempArray [0] = pressure1.get (i);
                  tempArray [1] = pressure2.get (i);
                  tempArray [2] = pressure3.get (i);
                  tempArray [3] = pressure4.get (i);
                  tempArray [4] = pressure5.get (i);
                  tempArray [5] = pressure6.get (i);
                  maxPress = tempArray [0];
                  for (int j = 1; j < 6; j++)
                  {
                        if (tempArray [j] > maxPress)
                        {
                              maxPress = tempArray [j];
                        }
                  }
                  press.add (maxPress);
            }
            boolean pressureCSA = analyzer
                        .DetectBreathingPausesSymptom (returnDoubleArray
(press));
            boolean soundCSA = false, cessations_present = false;
            if (m1CSAanalyze || m2CSAanalyze || m3CSAanalyze || m4CSAanalyze)
            {
                  soundCSA = true;
            }
            if (soundCSA && pressureCSA)
            {
                  cessations_present = true;
            }
            if (m1OSAanalyze || m2OSAanalyze || m3OSAanalyze || m4OSAanalyze)
            {
                  snoring = true;
            }
            String risk = "";
            boolean [] decarr = new boolean [3];
            decarr [1] = cessations_present;
            decarr [2] = snoring;
            decarr [3] = (abNREMPulse || abREMPulse);
            int count = 0;
            for (i = 0; i < 3; i++)
            {
                  if (decarr [i])
                  {
                        ++count;
                  }
```

```
                }
                if (hassweat)
                {
                        ++count;
                }
                ;
                if (count >= 3)
                {
                        risk = "High Risk";
                }
                if (count == 2)
                {
                        risk = "Medium Risk";
                }
                if (count == 1)
                {
                        risk = "Low Risk";
                }
                if (count == 0)
                {
                        risk = "No Risk";
                }
                crs.setCommand ("insert into results values(?,?,?)");
                crs.setString (1, "Saurabh");
                crs.setTimestamp (2, new Timestamp (new Date ().getTime ()));
                crs.setString (3, risk);
                crs.execute ();
        }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author Sankar
 */
public class NRemInterval
{
        double startTime;
        double endTime;

        public double getStartTime ()
        {
                return startTime;
        }

        public void setStartTime (double startTime)
        {
                this.startTime = startTime;
        }

        public double getEndTime ()
        {
                return endTime;
        }
```

```
	public void setEndTime (double endTime)
	{
		this.endTime = endTime;
	}

	public NRemInterval (double startTime, double endTime)
	{
		this.startTime = startTime;
		this.endTime = endTime;
	}
}

public interface Observer
{
    public void Update (boolean [] motion_sensors_readings, double []
body_temperatures, double [] lungs_and_diaphragm_pressure_forces, double []
sound_levels, double [] sweat_content_percentages, int [] heart_rates);
}

public class RemInterval
{
	public double startTime;
	public double endTime;

	public double getStartTime ()
	{
		return startTime;
	}

	public void setStartTime (double startTime)
	{
		this.startTime = startTime;
	}

	public double getEndTime ()
	{
		return endTime;
	}

	public void setEndTime (double endTime)
	{
		this.endTime = endTime;
	}

	public RemInterval (double startTime, double endTime)
	{
		this.startTime = startTime;
		this.endTime = endTime;
	}
}

import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.sql.rowset.CachedRowSet;
import javax.sql.rowset.RowSetProvider;
```

```
public class SensoryData implements Observer
{
	private boolean [] motion_sensors_readings;
	private boolean [] prev_motion_sensors_readings;
	private double [] body_temperatures, lungs_and_diaphragm_pressure_forces,
			sound_levels, sweat_content_percentages;
	private double [] prev_body_temperatures,
			prev_lungs_and_diaphragm_pressure_forces, prev_sound_levels,
			prev_sweat_content_percentages;
	private int [] heart_rates;
	private int [] prev_heart_rates;
	private Subject mattress;
	private String username;

	public String getUsername ()
	{
		return username;
	}

	public void setUsername (String username)
	{
		this.username = username;
	}

	public SensoryData (Subject mattress)
	{
		motion_sensors_readings = new boolean [8];
		body_temperatures = new double [8];
		lungs_and_diaphragm_pressure_forces = new double [6];
		sweat_content_percentages = new double [8];
		sound_levels = new double [4];
		heart_rates = new int [6];
		prev_motion_sensors_readings = new boolean [8];
		prev_body_temperatures = new double [8];
		prev_lungs_and_diaphragm_pressure_forces = new double [6];
		prev_sweat_content_percentages = new double [8];
		prev_sound_levels = new double [4];
		prev_heart_rates = new int [6];
		this.mattress = mattress;
		this.mattress.RegisterObserver (this);
		for (int i = 0; i < 8; i++)
		{
			if (i < 4)
			{
				prev_sound_levels [i] = 0.0;
			}
			if (i < 6)
			{
				prev_lungs_and_diaphragm_pressure_forces [i] = 0.0;
				prev_heart_rates [i] = 0;
			}
			prev_motion_sensors_readings [i] = false;
			prev_body_temperatures [i] = 0.0;
			prev_sweat_content_percentages [i] = 0.0;
		}
	}

	boolean start = false;
	int temp_zero_count = 0;
```

```
	public void Update (boolean [] motion_sensors_readings,
			double [] body_temperatures,
			double [] lungs_and_diaphragm_pressure_forces,
			double [] sound_levels, double [] sweat_content_percentages,
			int [] heart_rates)
	{
		try
		{
			int index;
			CachedRowSet crs = null;
			int count = 0;
			username = "Saurabh";
			crs = RowSetProvider.newFactory ().createCachedRowSet ();
			crs.setUrl ("jdbc:derby://localhost:1527/patient");
			crs.setUsername ("a");
			crs.setPassword ("a");
			java.util.Date date = new java.util.Date ();
			Timestamp moment = new Timestamp (date.getTime ());
			for (index = 0; index < motion_sensors_readings.length; index
= index + 1)
			{
				this.motion_sensors_readings [index] =
motion_sensors_readings [index];
			}
			for (index = 0; index < body_temperatures.length; index =
index + 1)
			{
				this.body_temperatures [index] = body_temperatures
[index];
			}
			for (index = 0; index <
lungs_and_diaphragm_pressure_forces.length; index = index + 1)
			{
				this.lungs_and_diaphragm_pressure_forces [index] =
lungs_and_diaphragm_pressure_forces [index];
			}
			for (index = 0; index < sound_levels.length; index = index +
1)
			{
				this.sound_levels [index] = sound_levels [index];
			}
			for (index = 0; index < sweat_content_percentages.length;
index = index + 1)
			{
				this.sweat_content_percentages [index] =
sweat_content_percentages [index];
			}
			for (index = 0; index < heart_rates.length; index = index + 1)
			{
				this.heart_rates [index] = heart_rates [index];
			}
			count = 0;
			for (int i = 0; i < 8; i++)
			{
				if (i < 4)
				{
					if (this.sound_levels [i] == 0)
					{
```

```
                                    ++count;
                              }
                        }
                        if (i < 6)
                        {
                              if (this.lungs_and_diaphragm_pressure_forces [i]
== 0.0)
                              {
                                    ++count;
                              }
                              if (this.heart_rates [i] == 0)
                              {
                                    ++count;
                              }
                        }
                        if (this.body_temperatures [i] == 0)
                        {
                              ++count;
                        }
                        if (this.sweat_content_percentages [i] == 0)
                        {
                              ++count;
                        }
                  }
                  System.out.println ("THE COUNT IS: " + count);
                  if (count >= 30)
                  {
                        System.out.println ("useless data");
                        if (start)
                        {
                              crs.setCommand ("insert into  motions
values(?,?,?,?,?,?,?,?,?,?)");
                              crs.setString (1, username);
                              crs.setTimestamp (2, moment);
                              for (int i = 0; i < 8; i++)
                              {
                                    crs.setBoolean (index + 3,
                                                prev_motion_sensors_readings
[i]);
                              }
                              crs.execute ();
                              crs.setCommand ("insert into  temperatures
values(?,?,?,?,?,?,?,?,?,?)");
                              crs.setString (1, username);
                              crs.setTimestamp (2, moment);
                              for (int i = 0; i < 8; i++)
                              {
                                    crs.setDouble (index + 3,
                                                this.prev_body_temperatures
[i]);
                              }
                              crs.execute ();
                              crs.setCommand ("insert into  pressures
values(?,?,?,?,?,?,?,?)");
                              crs.setString (1, username);
                              crs.setTimestamp (2, moment);
                              for (int i = 0; i < 6; i++)
                              {
                                    crs.setDouble (
```

```
                                                index + 3,

      this.prev_lungs_and_diaphragm_pressure_forces [i]);
                                    }
                                    crs.execute ();
                                    crs.setCommand ("insert into  microphones
values(?,?,?,?,?,?)");
                                    crs.setString (1, username);
                                    crs.setTimestamp (2, moment);
                                    for (int i = 0; i < 4; i++)
                                    {
                                          crs.setDouble (index + 3,
this.prev_sound_levels [i]);
                                    }
                                    crs.execute ();
                                    crs.setCommand ("insert into  sweats
values(?,?,?,?,?,?,?,?,?,?)");
                                    crs.setString (1, username);
                                    crs.setTimestamp (2, moment);
                                    for (int i = 0; i < 8; i++)
                                    {
                                          crs.setDouble (index + 3,

      this.prev_sweat_content_percentages [i]);
                                    }
                                    crs.setCommand ("insert into  heartrates
values(?,?,?,?,?,?,?,?,?,?)");
                                    crs.setString (1, username);
                                    crs.setTimestamp (2, moment);
                                    for (int i = 0; i < 8; i++)
                                    {
                                          crs.setDouble (index + 3,
this.prev_heart_rates [i]);
                                    }
                                    crs.execute ();
                              }
                        }
                        else
                        {
                              // System.out.println("can insert into database");
                              start = false;
                              crs.setCommand ("insert into  motions
values(?,?,?,?,?,?,?,?,?,?)");
                              crs.setString (1, username);
                              crs.setTimestamp (2, moment);
                              for (int i = 0; i < 8; i++)
                              {
                                    // System.out.println("CURRENT MOTION SENSOR
READING"+this.motion_sensors_readings[i]);
                                    crs.setBoolean (i + 3,
this.motion_sensors_readings [i]);
                                    prev_motion_sensors_readings [i] =
this.motion_sensors_readings [i];
                              }
                              crs.execute ();
                              crs.setCommand ("insert into  temperatures
values(?,?,?,?,?,?,?,?,?,?)");
                              crs.setString (1, username);
                              crs.setTimestamp (2, moment);
```

```
                        for (int i = 0; i < 8; i++)
                        {
                                crs.setDouble (i + 3, this.body_temperatures
[i]);
                                prev_body_temperatures [i] =
this.body_temperatures [i];
                        }
                        crs.execute ();
                        crs.setCommand ("insert into  pressures
values(?,?,?,?,?,?,?,?)");
                        crs.setString (1, username);
                        crs.setTimestamp (2, moment);
                        for (int i = 0; i < 6; i++)
                        {
                                crs.setDouble (i + 3,

      this.lungs_and_diaphragm_pressure_forces [i]);
                                prev_lungs_and_diaphragm_pressure_forces [i] =
this.lungs_and_diaphragm_pressure_forces [i];
                        }
                        crs.execute ();
                        crs.setCommand ("insert into  microphones
values(?,?,?,?,?,?)");
                        crs.setString (1, username);
                        crs.setTimestamp (2, moment);
                        for (int i = 0; i < 4; i++)
                        {
                                crs.setDouble (i + 3, this.sound_levels [i]);
                                prev_sound_levels [i] = this.sound_levels [i];
                        }
                        crs.execute ();
                        crs.setCommand ("insert into  heartrates
values(?,?,?,?,?,?,?,?)");
                        crs.setString (1, username);
                        crs.setTimestamp (2, moment);
                        for (int i = 0; i < 6; i++)
                        {
                                crs.setDouble (i + 3, this.heart_rates [i]);
                                prev_heart_rates [i] = this.heart_rates [i];
                        }
                        crs.execute ();
                        crs.setCommand ("insert into  sweats
values(?,?,?,?,?,?,?,?,?,?)");
                        crs.setString (1, username);
                        crs.setTimestamp (2, moment);
                        for (int i = 0; i < 8; i++)
                        {
                                // System.out.println("Sweat
percentages"+this.sweat_content_percentages[i]);
                                crs.setDouble (i + 3,
this.sweat_content_percentages [i]);
                                prev_sweat_content_percentages [i] =
this.sweat_content_percentages [i];
                        }
                        crs.execute ();
                }
        }
        catch (SQLException ex)
        {
```

```
                    Logger.getLogger (SensoryData.class.getName ()).log
(Level.SEVERE,
                                    null, ex);
                }
        }

        public boolean [] Get_Motion_Sensors_Readings ()
        {
                return motion_sensors_readings;
        }

        public double [] Get_Body_Temperatures ()
        {
                return body_temperatures;
        }

        public double [] Get_Lungs_And_Diaphragm_Pressure_Forces ()
        {
                return lungs_and_diaphragm_pressure_forces;
        }

        public double [] Get_Sound_Levels ()
        {
                return sound_levels;
        }

        public double [] Get_Sweat_Content_Percentages ()
        {
                return sweat_content_percentages;
        }

        public int [] Get_Heart_Rates ()
        {
                return heart_rates;
        }

        public void Output ()
        {
                int index;
                for (index = 0; index < motion_sensors_readings.length; index =
index + 1)
                {
                        System.out.print (motion_sensors_readings [index]);
                        System.out.print (" ");
                }
                for (index = 0; index < body_temperatures.length; index = index + 1)
                {
                        System.out.print (body_temperatures [index]);
                        System.out.print (" ");
                }
                for (index = 0; index < lungs_and_diaphragm_pressure_forces.length;
index = index + 1)
                {
                        System.out.print (lungs_and_diaphragm_pressure_forces
[index]);
                        System.out.print (" ");
                }
                for (index = 0; index < sound_levels.length; index = index + 1)
                {
```

```
                System.out.print (sound_levels [index]);
                System.out.print (" ");
            }
            for (index = 0; index < sweat_content_percentages.length; index =
index + 1)
            {
                System.out.print (sweat_content_percentages [index]);
                System.out.print (" ");
            }
            for (index = 0; index < heart_rates.length; index = index + 1)
            {
                System.out.print (heart_rates [index]);
                System.out.print (" ");
            }
            System.out.println ();
        }
}

import java.util.ArrayList;

public class SleepAnalyzer
{
        ArrayList <RemInterval> rems = new ArrayList <> ();
        ArrayList <NRemInterval> nrems = new ArrayList <> ();

        private class BreathingCessationEpisode
        {
            public double standard_deviation;
            public int second_of_start, second_of_end;

            public BreathingCessationEpisode ()
            {
                standard_deviation = 0;
                second_of_start = 0;
                second_of_end = 0;
            }

            public BreathingCessationEpisode (double standard_deviation,
                    int second_of_start, int second_of_end)
            {
                this.standard_deviation = standard_deviation;
                this.second_of_start = second_of_start;
                this.second_of_end = second_of_end;
            }
        }

        ArrayList <RemInterval> intervals = new ArrayList <> ();
        private int seconds_of_sleep;

        public SleepAnalyzer ()
        {
            seconds_of_sleep = 0;
        }

        int sleepstart;

        void findsleepStart (double temperature_sensor_readings [])
        {
            int i;
```

```
            int bogusreadings = 1800;
            double initialtemp = (((9.0 / 5.0) * temperature_sensor_readings
[bogusreadings]) + 32.0);
            for (i = bogusreadings + 1; i < temperature_sensor_readings.length;
i++)
            {
                  double ftemp = (((9.0 / 5.0) * temperature_sensor_readings
[i]) + 32.0);
                  if (ftemp <= initialtemp - 1.3)
                  {
                        sleepstart = i;
                        return;
                  }
            }
      }

      public double normalheartrate = 0;

      public int getNormalheartrate ()
      {
            return (int) normalheartrate;
      }

      public void setNormalheartrate (double normalheartrate)
      {
            this.normalheartrate = normalheartrate;
      }

      // void detectSleepStages(double[] heartRates, boolean motion1[],boolean
      // motion2[],boolean motion3[],boolean motion4[],boolean motion5[],boolean
      // motion6[],boolean motion7[],boolean motion8[])
      // {
      // int remstart = -1,nremstart =-1,remend=-1,nremend = -1;
      // boolean inRem =false, inNrem = false;
      // for(int i=sleepstart;i<heartRates.length;i++)
      // {
      // if(heartRates[i]>=(1.07*heartRates[i]))
      // {
      // if(!inRem)
      // {
      //
      //
      // inRem = true;
      // inNrem=false;
      // nremend =i;
      // NRemInterval nrinterval = new NRemInterval(remstart,remend);
      // nrems.add(nrinterval);
      // remstart = i;
      //
      // }
      //
      // }
      // else
      // {
      // if(inRem || i==sleepstart || motion1[i]|| motion2[i]|| motion3[i]||
      // motion4[i]|| motion5[i]|| motion6[i]|| motion7[i]|| motion8[i])
      // {
      // inRem=false;
      // inNrem =true;
```

```
	// if(i>sleepstart)
	// {
	// remend = i-1;
	// RemInterval rinterval = new RemInterval(remstart,remend);
	// rems.add(rinterval);
	// }
	// nremstart=i;
	//
	// }
	//
	// }
	// }
	// }
	void detectSleepStages (double [] heartRates, boolean motion1 [],
			boolean motion2 [], boolean motion3 [], boolean motion4 [],
			boolean motion5 [], boolean motion6 [], boolean motion7 [],
			boolean motion8 [])
	{
		int remstart = -1, nremstart = -1, remend = -1, nremend = -1;
		boolean inRem = false, inNrem = false;
		for (int i = sleepstart; i < heartRates.length; i++)
		{
			if ((heartRates [i] >= (1.07 * heartRates [sleepstart]))
					&& !(motion1 [i] || motion2 [i] || motion3 [i]
							|| motion4 [i] || motion5 [i] ||
motion6 [i]
							|| motion7 [i] || motion8 [i]))
			{
				int savei = i;
				remstart = i;
				System.out.println (heartRates [i]);
				while (((i + 1) < heartRates.length)
						&& ((heartRates [i + 1] >= (1.07 * 60)) &&
!(motion1 [i + 1]
								|| motion2 [i + 1]
								|| motion3 [i + 1]
								|| motion4 [i + 1]
								|| motion5 [i + 1]
								|| motion6 [i + 1] || motion7
[i + 1] || motion8 [i + 1])))
				{
					++i;
					System.out.println (heartRates [i]);
				}
				remend = i;
				if (remend > savei)
				{
					RemInterval rinterval = new RemInterval
(remstart, remend);
					rems.add (rinterval);
				}
			}
			else
			{
				if ((heartRates [i] < (1.07 * heartRates [sleepstart]))
						|| (motion1 [i] || motion2 [i] || motion3
[i]
								|| motion4 [i] || motion5 [i]
|| motion6 [i]
```

```
                                                || motion7 [i] || motion8
[i]))
                        {
                                nremstart = i;
                                int savei = i;
                                while (((i + 1) < heartRates.length)
                                        && ((heartRates [i + 1] < (1.07 *
60)) || (motion1 [i + 1]
                                                || motion2 [i + 1]
                                                || motion3 [i + 1]
                                                || motion4 [i + 1]
                                                || motion5 [i + 1]
                                                || motion6 [i + 1] ||
motion7 [i + 1] || motion8 [i + 1])))
                                {
                                        ++i;
                                }
                                nremend = i;
                                if (nremend > savei)
                                {
                                        NRemInterval nrinterval = new NRemInterval
(nremstart,
                                                        nremend);
                                        nrems.add (nrinterval);
                                }
                        }
                }
        }
    }

    // boolean analyzesoundForOBSA(double[] sounds)
    // {
    // int n=sounds.length;
    // int count=0;
    // for(int i=0;i<n-3600;i++)
    // {
    // if(sounds[i]>=70)
    // {
    // count=0;
    // for(int j=i;j<3600+i;j++)
    // {
    // int savej = j;
    // while((sounds[j]>=70)&&j<(3600 + i))
    // {
    // ++j;
    // }
    // if(j>savej)
    // {
    // ++count;
    // if(count == 5)
    // {
    // return true;
    // }
    // }
    // }
    //
    // }
    // }
    // return false;
```

```
//
// }
boolean analyzesoundForOBSA (double [] sounds)
{
	int n = sounds.length;
	int count = 0;
	int i = sleepstart;
	for (; i < n - 3600; i++)
	{
		for (int j = i; j < 3600 + i; j++)
		{
			int savej = j;
			if (sounds [j] >= 70)
			{
				while (sounds [j] >= 70)
				{
					++j;
				}
			}
			if (j > savej)
			{
				++count;
				if (count >= 5)
				{
					return true;
				}
			}
		}
	}
	// }
	return false;
}

// boolean analyzesoundForCSA(double[] sounds)
// {
// int n=sounds.length;
// int count=0;
// for(int i=0;i<n-3600;i++)
// {
// if(sounds[i]>=0 && sounds[i]<=20)
// {
// count=0;
//
// for(int j=i;j<3600+i;j++)
// {
// int savej = j;
// while((sounds[j]>=0 && sounds[j]<=20)&&j<3600+i)
// {
// ++j;
// }
// if(j>savej)
// {
// ++count;
// if(count>=5)
// {
// return true;
// }
// }
// }
// }
```

```
	//
	// }
	// }
	// return false;
	//
	// }
	boolean analyzesoundForCSA (double [] sounds)
	{
		int n = sounds.length;
		int count = 0;
		int i = sleepstart;
		for (; i < n - 3600; i++)
		{
			for (int j = i; j < 3600 + i; j++)
			{
				int savej = j;
				if (sounds [j] <= 20)
				{
					while (sounds [j] <= 20)
					{
						++j;
					}
				}
				if (j > savej)
				{
					++count;
					if (count >= 5)
					{
						return true;
					}
				}
			}
		}
		// }
		return false;
	}

	public boolean DetectBreathingPausesSymptom (
			double [] pressure_sensors_readings_over_night)
	{
		ArrayList <BreathingCessationEpisode> breathing_cessation_episodes =
new ArrayList <BreathingCessationEpisode> ();
		double [] standard_deviations;
		int index1, index2, index3 = 0, number_of_samples =
pressure_sensors_readings_over_night.length / 5;
		double [][] samples_of_pressure_sensors_readings_over_night;
		if (pressure_sensors_readings_over_night.length % 5 != 0)
		{
			number_of_samples = number_of_samples + 1;
		}
		standard_deviations = new double [number_of_samples];
		samples_of_pressure_sensors_readings_over_night = new double
[number_of_samples] [5];
		for (index1 = 0; index1 < number_of_samples - 1; index1 = index1 +
1)
		{
			for (index2 = 0; index2 < 5; index2 = index2 + 1)
			{
```

```
                        samples_of_pressure_sensors_readings_over_night
[index1] [index2] = pressure_sensors_readings_over_night [index3];
                        index3 = index3 + 1;
                  }
            }
            for (index1 = 0; index1 <
samples_of_pressure_sensors_readings_over_night.length
                        - index3; index1 = index1 + 1)
            {
                  samples_of_pressure_sensors_readings_over_night
[number_of_samples - 1] [index1] = pressure_sensors_readings_over_night [index3];
                  index3 = index3 + 1;
            }
            for (index2 = index1; index2 < 5; index2 = index2 + 1)
            {
                  samples_of_pressure_sensors_readings_over_night
[number_of_samples - 1] [index2] = 0;
            }
            for (index1 = 0; index1 < number_of_samples; index1 = index1 + 1)
            {
                  double [] temporary_array = new double [5];
                  for (index2 = 0; index2 < 5; index2 = index2 + 1)
                  {
                        temporary_array [index2] =
samples_of_pressure_sensors_readings_over_night [index1] [index2];
                  }
                  standard_deviations [index1] = StatisticalFunctions
                              .CalculateStandardDeviation (temporary_array);
                  if (standard_deviations [index1] < 6.0)
                  {
                        breathing_cessation_episodes
                                    .add (new BreathingCessationEpisode (
                                                standard_deviations [index1],
5 * index1,
                                                5 * index1 + 2));
                  }
            }
            if (breathing_cessation_episodes.size () < 5)
            {
                  return false;
            }
            for (index1 = 0; index1 < breathing_cessation_episodes.size () - 4;
index1 = index1 + 1)
            {
                  if (breathing_cessation_episodes.get (index1 +
4).second_of_end
                              - breathing_cessation_episodes.get
(index1).second_of_start <= 3600)
                  {
                        return true;
                  }
            }
            return false;
      }

      /**
       *
       * @param heartrates
       * @return
```

```
 */
int avg = 0;

// public boolean analyzeNRemPulses(int[] heartrates)
// {
// ArrayList<Boolean> analyzedPulses = new ArrayList();
// for(int i=0;i<nrems.size();i++)
// {
// NRemInterval nint;
// nint =new NRemInterval(nrems.get(i).startTime,nrems.get(i).endTime);
//
// double[] hrs= new
// double[(int)((nrems.get(i).startTime-nrems.get(i).endTime)+1)];
// int sum =0;
// int n=(int) ((nrems.get(i).startTime-nrems.get(i).endTime)+1);
//
// for(int j = (int)nrems.get(i).startTime,
// k=0;j<=nrems.get(i).endTime;j++,k++)
// {
// hrs[k]= heartrates[j];
// sum=sum+heartrates[j];
// }
// avg=sum/n;
// if(StatisticalFunctions.CalculateStandardDeviation(hrs)>=12 && (avg>=63
// && avg<=87))
// {
// avg=0;
// return true;
// }
// }
// return false;
//
// }
// public boolean analyzeRemPulses(double[] heartrates)
// {
// ArrayList<Boolean> analyzedPulses = new ArrayList();
// for(int i=0;i<nrems.size();i++)
// {
// RemInterval rint;
// rint =new RemInterval(rems.get(i).startTime,rems.get(i).endTime);
//
// double[] hrs= new
// double[(int)((rems.get(i).startTime-rems.get(i).endTime)+1)];
// int sum =0;
// int n=(int) ((rems.get(i).startTime-rems.get(i).endTime)+1);
//
// for(int j = (int)rems.get(i).startTime,
// k=0;j<=rems.get(i).endTime;j++,k++)
// {
// hrs[k]= heartrates[j];
// sum=(int)(sum+heartrates[j]);
// }
// avg=sum/n;
// if(StatisticalFunctions.CalculateStandardDeviation(hrs)>=10 && (avg>=60
// && avg<=80))
// {
//
// return true;
// }
```

```
	// avg=0;
	// }
	// return false;
	//
	// }
	public boolean analyzeNRemPulses (double [] heartrates)
	{
		int avg = 0;
		ArrayList <Boolean> analyzedPulses = new ArrayList ();
		for (int i = 0; i < nrems.size (); i++)
		{
			NRemInterval nint;
			nint = new NRemInterval (nrems.get (i).startTime,
					nrems.get (i).endTime);
			double [] hrs = new double [(int) ((nrems.get (i).endTime -
nrems
					.get (i).startTime) + 1)];
			int sum = 0;
			int n = (int) ((nrems.get (i).endTime - nrems.get
(i).startTime) + 1);
			for (int j = (int) nrems.get (i).startTime, k = 0; j <= nrems
					.get (i).endTime; j++, k++)
			{
				hrs [k] = heartrates [j];
				sum = (int) (sum + heartrates [j]);
			}
			avg = sum / n;
			if (StatisticalFunctions.CalculateStandardDeviation (hrs) >=
12
					&& (avg >= 63 && avg <= 87))
			{
				avg = 0;
				return true;
			}
		}
		return false;
	}

	public boolean analyzeRemPulses (double [] heartrates)
	{
		ArrayList <Boolean> analyzedPulses = new ArrayList ();
		for (int i = 0; i < rems.size (); i++)
		{
			RemInterval rint;
			int avg = 0;
			rint = new RemInterval (rems.get (0).startTime,
					rems.get (0).endTime);
			double [] hrs = new double [(int) ((rems.get (i).endTime -
rems
					.get (i).startTime) + 1)];
			int sum = 0;
			int n = (int) ((rems.get (i).endTime - rems.get (i).startTime)
+ 1);
			for (int j = (int) rems.get (i).startTime, k = 0; j <=
rems.get (i).endTime; j++, k++)
			{
				hrs [k] = heartrates [j];
				sum = (int) (sum + heartrates [j]);
			}
```

```
                    avg = sum / n;
                    if (StatisticalFunctions.CalculateStandardDeviation (hrs) >=
10
                                && (avg >= 60 && avg <= 80))
                    {
                          return true;
                    }
                    avg = 0;
             }
             return false;
       }
}

public class StatisticalFunctions
{
       public static double CalculateMean (double [] data)
       {
             double mean = 0;
             int index;
             for (index = 0; index < data.length; index = index + 1)
             {
                    mean = mean + data [index];
             }
             mean = mean / (data.length * 1.0);
             return mean;
       }

       public static double CalculateStandardDeviation (double [] data)
       {
             double mean = CalculateMean (data), standard_deviation = 0;
             int index;
             for (index = 0; index < data.length; index = index + 1)
             {
                    standard_deviation = standard_deviation + (data [index] -
mean)
                                * (data [index] - mean);
             }
             standard_deviation = Math.sqrt (standard_deviation
                          / ((data.length - 1) * 1.0));
             return standard_deviation;
       }
}

public interface Subject
{
    public void RegisterObserver (Observer observer);
    public void RemoveObsever (Observer observer);
    public void NotifyObservers () throws InterruptedException;
}
```