11-2015

Wireless Sensor Platform for Pulse Oximetry

Anusha Aithal
aa9115@rit.edu

Wireless Sensor Platform for Pulse Oximetry

By
Anusha Aithal

A Thesis Submitted in Partial Fulfillment of the requirements for the Degree of Master of Science in Microelectronic Engineering

Approved by:

____

Dr. Lynn Fuller, Professor
Thesis Advisor, Department of Electrical and Microelectronic Engineering

____

Dr. David Borkholder, Professor
Committee Member, Department of Microsystems Engineering

____

Dr. Robert Pearson, Professor
Committee Member, Department of Electrical and Microelectronic Engineering

____

Dr. Sohail Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
NOVEMBER 2015

Acknowledgements

I would like to express my sincere gratitude to the following people:

Dr. Lynn Fuller, for his constant encouragement and for giving me the opportunity to work on this project.

Members of my committee, Dr. Robert Pearson and Dr. David Borkholder, for their insightful input.

Dr. Daniel Phillips and Ms. Michelle Horan for letting me borrow their equipment.

My fellow students Kavya Duggimpudi and Karine Florent for their support.

And lastly, I would like to dedicate this work to my family.

Abstract

Pulse rate and oxygen saturation are two important clinical measurements that indicate the state of a person's essential body functions. Oxygen saturation is the measurement of oxygenated hemoglobin in arterial blood i.e. it indicates the level of oxygen in the blood. Pulse oximeters, consisting of LEDs and photodetectors, offer a simple and low cost means of monitoring both pulse rate and blood oxygen saturation non-invasively.

The primary objective of this project was to develop a wireless platform for MEMS devices. For this project, a pulse oximeter was also developed as a demonstration vehicle for this wireless platform. A microcontroller and a Bluetooth module was used to transmit the data from the sensor to the smartphone and an Android program was developed as a part of the project to connect with the Bluetooth module and receive, plot and save the data. Once the sensor and Android application were developed, the pulse rate and oxygen saturation measurements were compared to measurements taken by a commercial pulse oximeter to determine the accuracy of the device. The sensor was able to accurately measure with an average error percentage of $\pm 2.86\%$ and $\pm 1.08\%$ for pulse rate and oxygen saturation respectively.

Table of contents

List of figures

List of tables

List of Abbreviations

HbO_2	- Oxygenated hemoglobin
SpO_2	- Oxygen Saturation
IDE	- Integrated Development Environment
PPG	- Photoplethysmogram
SPP	- Serial Port Profile
SDK	- Software Development Kit
APK	- Android Application Package
API	- Application Programming Interface

Chapter 1

Literature Review

Oxygen is one of the most important elements needed to sustain life. It is important to almost every single cell in the body. It is involved in the process of cellular respiration, where carbohydrates, fats and proteins are broken down to retrieve energy. A decline in the levels of oxygen could affect metabolism significantly. A protein in the red blood cells called hemoglobin (Hb) is responsible for most (about 98%) of the oxygen transport in the body. Oxygen binds to the protein, together forming oxyhemoglobin or HbO_2. Oxygen saturation (SpO_2) is a measure of the percentage of HbO_2 in arterial blood.

1.1 Introduction

A pulse oximeter is a device that can measure oxygen saturation non-invasively. Prior to the invention of the pulse oximeter, SpO_2 was measured by performing a blood gas analysis on samples of blood drawn from the patient. This method does not provide real time feedback and is an invasive procedure. For a healthy individual with sufficient oxygen in their blood stream, the tongue and lips appear pink. However, when the oxygen levels are low, they appear blue. This bluish appearance of skin and mucus membranes is cyanosis. Hypoxemia, or abnormally low concentrations of oxygen in the blood, used to be assessed primarily by looking for cyanosis. Using cyanosis for the clinical assessment of hypoxemia is extremely unreliable because several factors including skin color and room lighting can affect the detection of cyanosis. Additionally, cyanosis is only visible when the concentration of deoxygenated hemoglobin is above a certain limit. So a severely anemic patient may never show signs of cyanosis even when extremely hypoxic. Thus, a tool to constantly monitor oxygen saturation levels is very important.

A pulse oximeter measures the oxygen saturation levels and the pulse rate from a photoplethysmogram (PPG). Photoplethysmography was first described by Alrick Hertzman in 1937. He gave it the term “plethysmos”, derived from the Greek work for fullness, since he believed he was measuring the fullness of the tissue when he measured the amount of light it absorbed [1,2]. Most pulse oximeters display the plethysmograph trace in addition to the arterial oxygen saturation and the heart rate. The most important function of the trace is to judge if the oximeters is functioning correctly. However, a normal plethysmograph trace does not imply that the SpO2 value is correct [3].

A pulse oximeter consists of a red LED (~650 nm), infrared LED (~940nm) and a photodetector. The PPG is obtained by measuring the change in the light absorbed by the blood and the ratio of absorption of red to infrared light is used to calculate the SpO_2. Pulse oximeters are used extensively in ICUs, and during surgeries and other procedures involving sedation.

1.2. Historical Review

The relationship between the absorption of light and the concentration of the absorbent was first described by Johann Heinrich Lambert in 1760. This was further investigated by August Beer, who published the Beer-Lambert law in 1851 [3]. The Beer-Lambert law is the linear relationship between the absorbance of light and the amount of absorbing material. Following the invention of the spectrometer in1860 by Kirchhoff and Bunsen, Hoppe-Seyler showed that oxygen changes the color of a component in blood. He coined the term “hemoglobin” and later showed that oxygen and hemoglobin form a loose, dissociable compound he called oxyhemoglobin. Stokes showed that this colored substance was the carrier of oxygen in the blood [4-6].

The first device to measure the oxygen saturation in blood by illuminating it with different wavelengths of light was developed in the 1935 by Karl Matthes, a physician. This device measured ear oxygen saturation and used a second wavelength (the first being red and the second being green or infrared) to compensate for blood volume and tissue pigment. Millikan made the first portable, light-weight oximeter in the early 1940’s to train World War II pilots for aviation. The Hewlett-Packard ear oximeter, developed in the early 1970's, was the forerunner of pulse oximetry. It used eight different wavelengths derived from an incandescent source. The different wavelengths were for different absorbing substance (skin, oxygenated hemoglobin, deoxygenated hemoglobin etc.) and the model assumed that each absorbing substance acted independent of the others. Again, this model did not differentiate between arterial and venous blood. While it was a remarkable advance in oxygen saturation monitoring, the device had several disadvantages including the large, bulky earpiece and the need for regular recalibration [3,4,6,7].

In the early 1970's, Takuo Aoyagi's research involved using dye dilation to measure cardiac output. He used two wavelengths, 805 nm and 900 nm to maximize the dye sensitivity and minimize the oxygen saturation sensitivity. The ratio of the two optical densities was used to get the dye curve, which was expected to correspond with the dye concentrations in the blood. However, he noticed pulsatile variations in the output, making it difficult for him to measure the cardiac output. He then used the wavelengths 900 nm and 630 nm. The 900 nm wavelength was used to avoid interference by the dye and the 630 nm wavelength was chosen to maximize the hemoglobin extinction change caused by oxygen saturation change. He then used the AC/DC ratio of the two wavelengths to calculate the oxygen saturation, hence developing the pulse oximeter [6, 8-10]. Aoyagi and his associates released the pilot model of the pulse oximeter in 1974. Biox made the first commercial oximeter in 1981, followed by Nellcor, who released their product in 1983. It was recognized as the standard for inoperative monitoring by the Association of Anesthetics of Great Britain and Ireland in 1988 and the American Society of Anesthesiologists in 1990 [6].

1.3. Theory

1.3.1. Oxygen Transport in the Body

Oxygen is vital for survival. The absence of oxygen for a prolonged period of time will cause cells to die. Thus, oxygen circulation is an important indicator of a person's health. The respiratory system and the circulatory system are responsible for the oxygen supplied to the cells. Ventilation is the process by which air moves into and out of the lungs. Ventilation is based on the principle that air flows from a region of higher pressure to a region of lower pressure. During inspiration, the muscles between the ribs contract and pull upwards, while the diaphragm contracts and pulls downward, causing the thoracic cavity to expand. At this point, the atmospheric pressure is greater than the pressure inside the lungs, causing air to flow in. During expiration, the muscles relax, causing the volume to decrease and the pressure in the lungs to increase. As the pressure inside the lungs increases beyond the atmosphere pressure, the air flows out of the lungs [11, 12].

The nasal passage warms, filters and moistens the inhaled air before it reaches the lungs. The nasal passage has microscopic hair called cilia which, together with the mucus, filters dust particles out of the inhaled air. The mucus also moistens the air, while the capillaries under the mucus layer warm it. The air then passes through the pharynx (or the throat) to the larynx, which is also referred to as the voice box. Both air and food go through the pharynx. When food is swallowed, a part of the larynx called the epiglottis closes the entrance to the trachea, hence preventing the entry of food particles into the lungs.

The larynx leads into the trachea or the windpipe, which is kept open by ringed cartilages. The trachea also has mucus and cilia to filter out further pollutants and particles. The trachea splits into two bronchi, each of which leads to a lung. The bronchi branches out into bronchial tubes, which divide further into smaller structures called bronchioles as shown in figure 1.1. The bronchioles end in small balloon like structures called the alveoli. The alveoli is where the gas exchange takes place [11, 13].

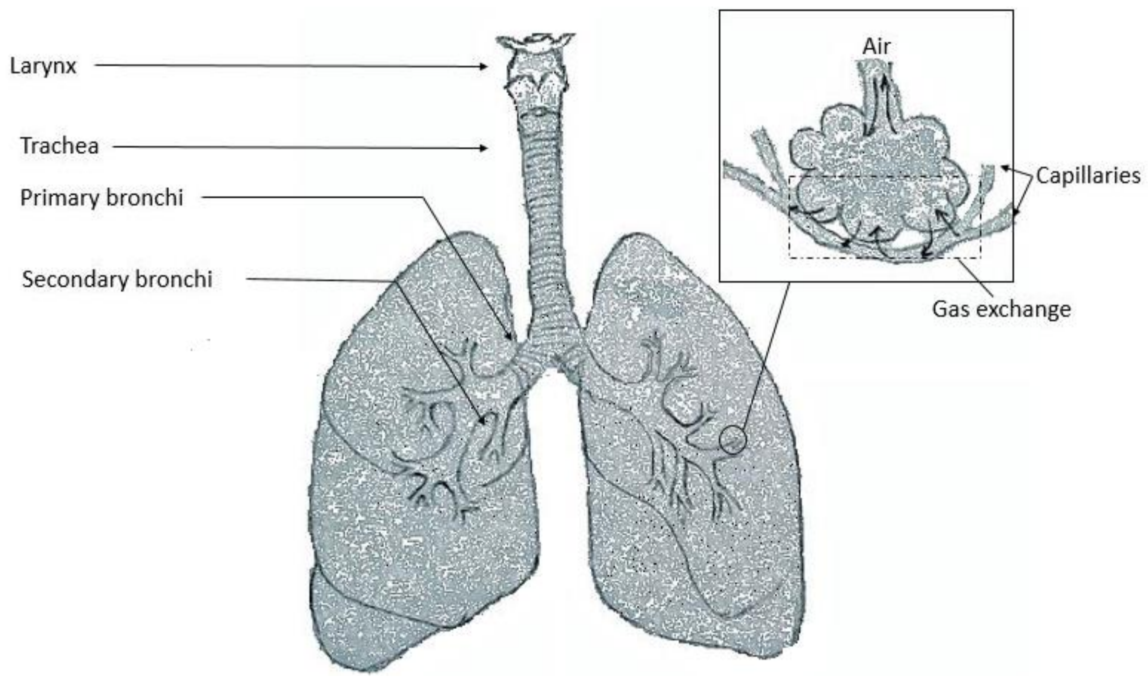


Figure 1.1. The lungs

The alveoli are surrounded by capillaries and the walls of the alveoli are very thin, allowing the exchange of oxygen and carbon dioxide between the alveoli and the capillaries. The partial pressure of oxygen is higher, and that of carbon dioxide is lower in the alveoli than the blood, causing oxygen to diffuse into the blood while the carbon dioxide moves into the alveoli. Most of the oxygen that enters the bloodstream binds to the hemoglobin in the blood, while a small amount dissolves in the plasma. The blood from the capillary then enters the heart through the pulmonary vein and is circulated throughout the body.

The heart serves as a pumping mechanism for the blood. In the pulmonary circulation, oxygen depleted blood is pumped out of the right ventricle of the heart. The pulmonary artery transports this blood to the lungs, where carbon dioxide is released from the blood into the alveoli through the capillaries and oxygen enters the bloodstream. This oxygenated blood is then transported to the left atrium of the heart by the pulmonary vein.

In the systematic circulation, oxygen rich blood is pumped out of the left ventricle via the systemic arteries. The artery branches out into smaller arteries and finally into capillaries. The oxygen diffuses through the walls of the capillaries to the oxygen depleted tissues, while the waste substances and carbon dioxide diffuses into the blood from the tissues. This blood, which is now low in oxygen, is collected in veins and transported back to the heart [11, 13, 15]. Figure 1.2 shows this entire process.

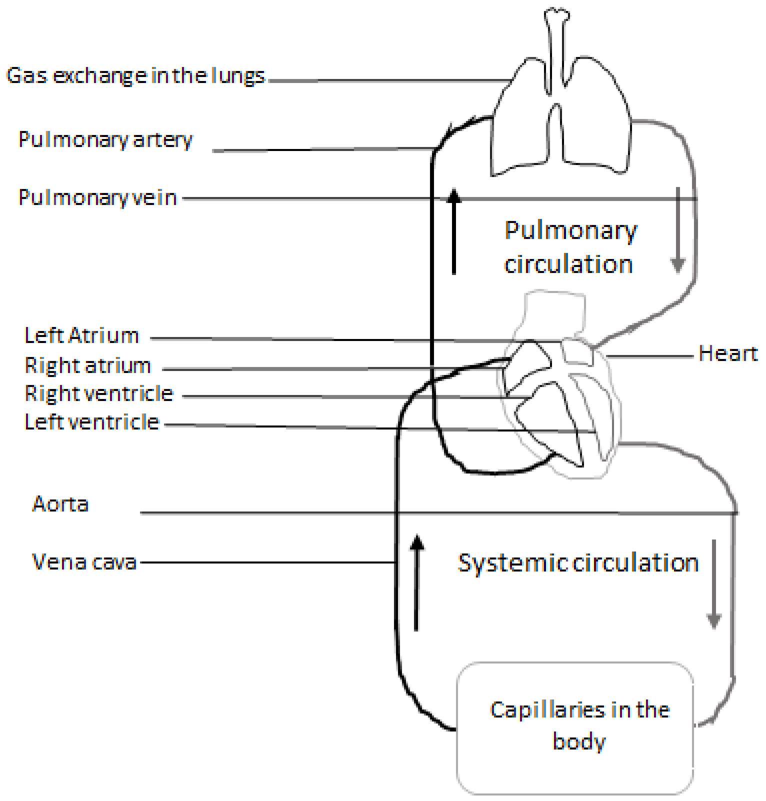


Figure 1.2. Oxygen transport in the blood

The amount of oxygen that gets circulated depends on three factors: the amount of oxygen that binds to the hemoglobin in the blood, the concentration of hemoglobin in the blood and the cardiac output [16]. Cardiac output is the total volume of blood pumped through the circulatory system in one minute. Hemoglobin (abbreviated as Hb) is the protein in red blood cells that carries oxygen. Each hemoglobin molecule has four binding sites for oxygen molecules. A saturated hemoglobin molecules has four oxygen molecules bound to it. Every 100 ml of blood contains about 15g of hemoglobin and each gram of hemoglobin can bind to up to 1.39ml of oxygen so 100 ml of blood can contain up to 20.4ml of oxygen [16, 17].

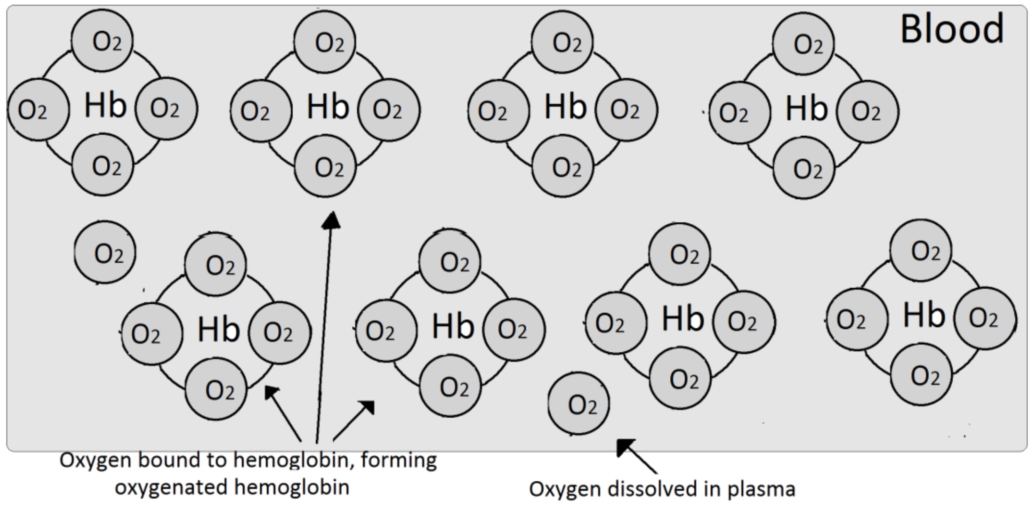

Blood
Hb
O_2
Oxygen bound to hemoglobin, forming oxygenated hemoglobin
Oxygen dissolved in plasma

Figure 1.3. Oxygen bound to hemoglobin

1.3.2. Photoplethysmography

A plethysmograph is used to measure a change in volume within an organ or the whole body. A PPG is an optically obtained plethysmogram. It is a pulsatile waveform that can be used to non-invasively monitor the cardiac rhythm. Light incident on the body can be absorbed by arterial blood, venous blood, skin, tissue, pigments and bones but the light absorbed by arterial blood changes as the volume of blood in the artery changes. There is more blood in the arteries during the systolic phase of the cardiac cycle than in the diastolic phase. Thus, a PPG sensor can optically monitor the blood flow by measuring the light transmitted through or reflected off the tissue. Figure 1.4 shows the absorption of light by tissue over time. The PPG waveform consists of a small AC signal riding over a large DC signal. This DC signal is actually a slowly varying signal and changes with respiration.

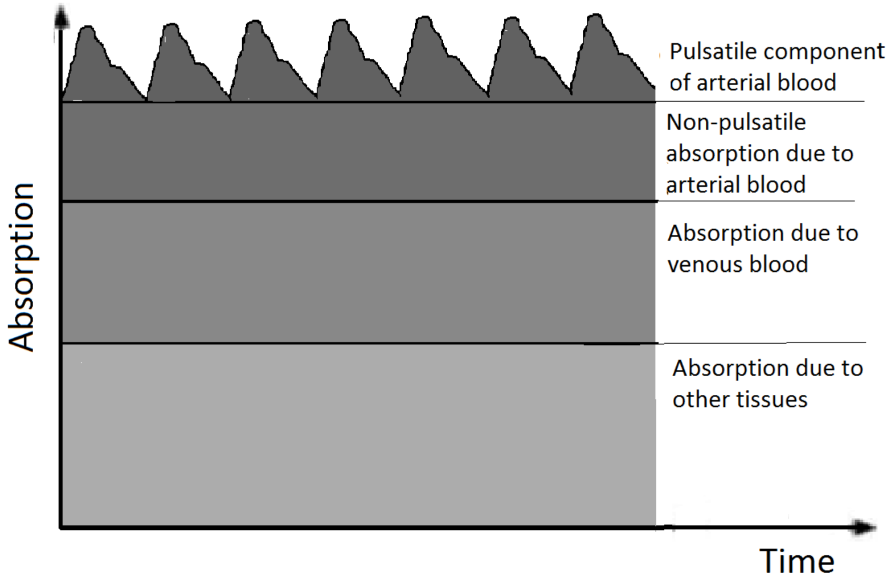

Pulsatile component of arterial blood
Non-pulsatile absorption due to arterial blood
Absorption due to venous blood
Absorption due to other tissues
Absorption
Time

Figure 1.4. Variation in attenuation due to arterial blood

1.3.3. Oxygen Saturation

Oxygen saturation can be defined as the ratio of oxygenated hemoglobin to the total hemoglobin (oxygenated and deoxygenated) in the blood. If all the binding sites on the hemoglobin have oxygen molecules attached to them, the hemoglobin is said to have 100% saturation.

$$\mathrm{SpO_2(\%)} = \frac{\mathrm{HbO_2}}{\mathrm{HbO_2} + \mathrm{Hb}} * 100 \tag{1.1}$$

Oxygen saturation is the fifth most important vital sign after pulse rate, body temperature, blood pressure, and respiration. Most healthy individuals have an arterial oxygen saturation level of 95%-100% at sea level. Since the saturation levels depend on partial pressure of gases, extreme altitude can affect the saturation levels. Venous blood normally has a saturation of around 75%. The oxygen saturations level remains roughly constant in time but health problems like lung diseases or smoking can cause a decrease in these levels. For example, 94% can be considered normal for a heavy smoker. If the arterial oxygen saturation level is below 90%, it is considered low and requires treatment [17, 19].

Hemoglobin forms reversible and unstable bonds with oxygen to form HbO_2. In its oxygenated state, it appears a bright red color and in its reduced state, it appears a darker purplish in color. Beer-Lambert's law can be used to calculate the absorbance of monochromatic light by a transparent substance through which it passes as shown in the equation below [3].

$$I_t = I_o * 10^{-\varepsilon c d} \tag{1.2}$$

Where I_t is the intensity of transmitted light, I_o is the intensity of incident light, ε is the extinction coefficient (the fraction of light absorbed at a certain wavelength), c is the concentration of the absorbing substance, and d is the length of the path through the sample.

Two independent equations can be derived to describe the absorption of Hb and HbO_2 at two different wavelengths, i.e. red and infrared. These equations can then be solved to find the oxygen saturation, as this depends on the Hb – HbO_2 ratio. Hence, upon solving these equations, the SpO_2 can be calculated using the following equation [20].

$$SpO_2 = A + B * R \tag{1.3}$$

Where A and B are constants and R is the ratio of the optical density ($\log(I_o/I_t)$) of the oxygenated and the deoxygenated hemoglobin.

However, the Beer-Lambert law only applies to monochromatic radiation through a homogenous substance. There must be only one absorbent and there should be no reaction between the absorbent and the solvent. Blood is a non-homogenous substance and the amount of light absorbed changes as the volume of the blood changes. While the principle of pulse oximetry relies on the Beer-Lambert law, the non-homogenous nature of blood makes it clear that the device needs to be calibrated [3]. The calibration of a pulse oximeter will be discussed in the following section.

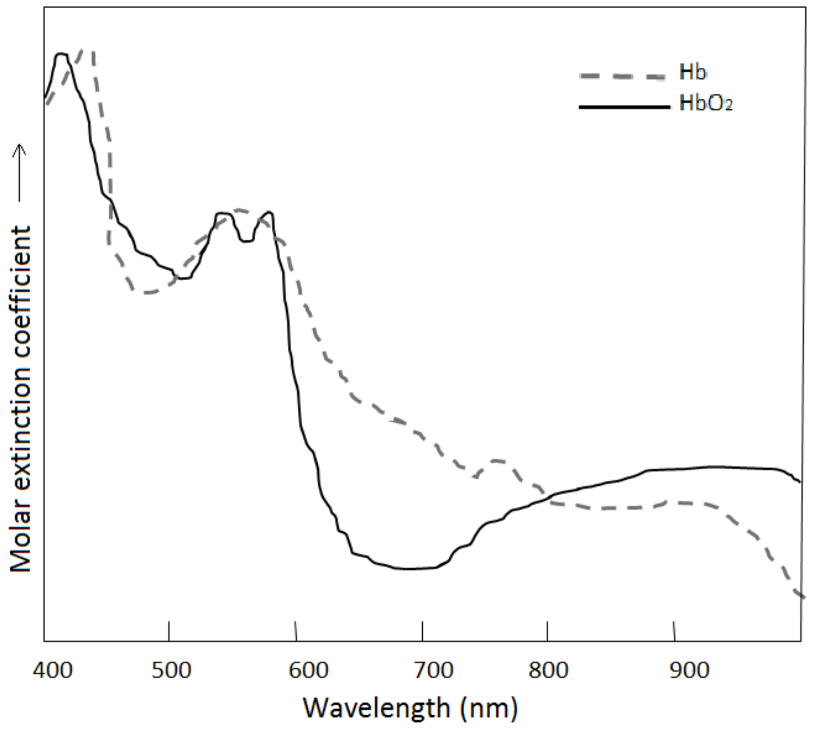


Figure 1.5. Example absorption spectrum of Hb and HbO_2

A pulse oximeter consists of a red LED, an infrared LED, and a photodetector. In order to get the SpO_2, the red and the infrared LEDs must be at a wavelength where the extinction coefficients of Hb and HbO_2 are different. Figure 1.5 shows the absorption spectrum of oxygenated and deoxygenated hemoglobin in the visible – near infrared region. The isobestic points are the wavelengths at which the extinction coefficients of the two substances are equal. Typically, pulse oximeters used a red LED at around 660 nm and an infrared LED at 880 nm – 940 nm. The range of wavelengths that can be used in 600 nm -1300nm. At wavelengths shorter than 600 nm, the skin pigment, melanin, absorbs strongly and at wavelengths longer than 1300 nm, the water present in the tissues absorbs strongly [3].

The sensor containing the LEDs and the photodetector is placed in close contact with the skin, typically at the fingertip or at the wrist and SpO2 is calculated from the ratio of absorption of the two wavelengths. The detector can be placed such that the sensor works in reflection or transmission mode.

1.3.4. Transmittance and reflectance pulse oximetry

In reflectance mode pulse oximetry, the LEDs and the photodetector are on the same side of the probe. These sensors can be placed at the fingertip or at the ear lobe. The photodiode measures the light back-scattered by the bone, tissues and blood vessels. In transmission mode pulse oximetry, the detector is placed on the opposite side of the sensor from the LEDs. These sensors can be placed at the fingertip, wrist, chest, forehead, etc. While a device using transmission mode can provide good measurements, the number of sites where it can be placed is limited. It must be placed at a point through which light can easily be transmitted. The reflectance based device, however, does not have this problem and can be placed in areas where light cannot easily transmit through as well.

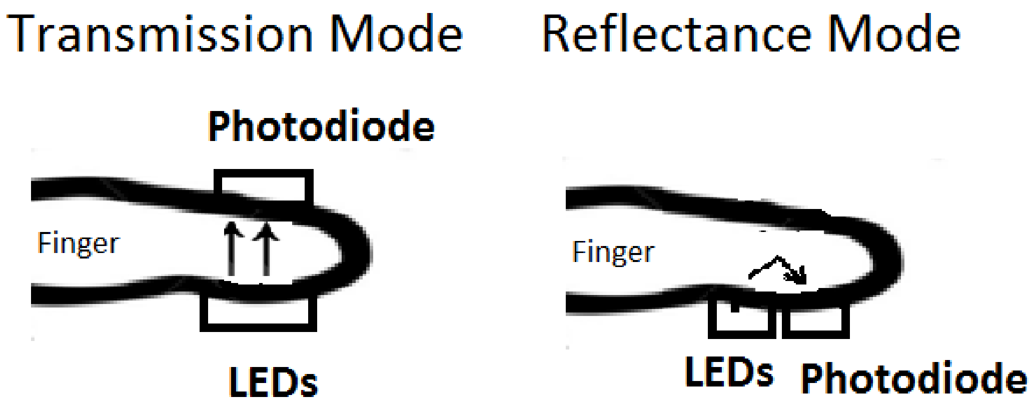


Figure 1.6. Transmittance pulse oximetry and reflectance pulse oximetry

1.3.5. Calibrating a pulse oximeter

The Beer-Lambert law was used to calibrate pulse oximeters originally, but as stated in the previous section, the law does not apply to pulse oximetry due to the presence of multiple absorbents and the scattering nature of blood. Pulse oximeters are now calibrated using an empirical method where a given ratio, R, is used to estimate the oxygen saturation levels [11]. R is a ratio of the amplitude of AC and DC signals from the red and the infrared LED.

$$R = \left(\frac{\left(\frac{AC}{DC}\right)_{RED}}{\left(\frac{AC}{DC}\right)_{IR}} \right) \tag{1.4}$$

Traditionally, pulse oximeters were calibrated by the *in vivo* method. This method used a CO-oximeter for comparison. There are four common types of hemoglobin: oxygenated hemoglobin (HbO_2), reduced hemoglobin (Hb), carboxyhemoglobin (COHb), and methemoglobin (MetHb). The CO-oximeter analyzes the concentration of the different types of hemoglobin using as few as four wavelengths of light [11].

The procedure involved drawing blood samples from the radial artery and then analyzing these samples to ascertain the levels of COHb and MetHb. This is then used to calculate the oxygen/air mixture required to bring the oxygen saturation to 100%. Then they breathe an oxygen/air combination with progressively less oxygen and more nitrogen, making the patient progressively more hypoxic and at each stage, samples of arterial blood is collected from the patient and analyzed using the CO-oximeter, creating a calibration curve for oxygen saturation [3, 11].

The *in vivo* method was used to calibrate pulse oximeters until 1993 [11]. There are several problems associated with this method, the biggest being that the methods involves deliberately desaturating a healthy volunteer. The range over which the device is calibrated is limited. The volunteer cannot be desaturated below 85% without risking hypoxic brain damage, so the calibration curve must be extrapolated for SpO_2 values below 85%.

The *in vitro* calibration method typically involves whole blood (anticoagulated with heparin) pumped through a model finger. A pump mixes in a known mixture of oxygen, nitrogen and carbon dioxide and a drive system is used to simulate the pulsatile nature of arterial blood. The oxygen saturation is measured by the pulse oximeter being calibrated, a CO-oximeter, and the blood samples are tested in a blood gas analyzer for pH and pCO [11]. The calibration method used for this project is discussed in section 5.1.

Chapter 2

Hardware Design and Circuitry

The pulse oximeter design is comprised of 5 parts: the LED driving circuit, the photodiode and a current to voltage converter, filtering and amplification, a low pass filter, and microcontroller with Bluetooth module. This chapter discusses the design and implementation of the electronic circuitry, while the next chapter discusses the microcontroller and the Bluetooth module.

2.1. Design Specification

The design specifications for the sensor and the hardware are listed below in table 5.1. The sensor specifications were determined from commercial device specifications. The specifications of several commercial pulse oximeters were examined and the following specification were chosen, however, the entire desired range of both the pulse rate as well as the oxygen saturation levels will not be tested for this project.

Table 2.1. Design Specifications

Hardware Specifications	
Red LED	Maximum current allowed = 30 mA Power supply = 9 V Current limiting resistor = 330Ω
Infrared LED	Maximum current allowed = 50 mA Power supply = 9 V Current limiting resistor = 220Ω
Non-inverting amplifier	Gain ~ 85 Single supply amplifier used Power supply +9 V
Low pass filter	Cutoff frequency = 15 Hz
Sensor Specification	
Effective measuring range	Pulse rate: 35 – 250 bpm Oxygen saturation: 0 – 100%
Accuracy	Pulse rate: ±3% Oxygen saturation: ±3%

2.2. LED deriving circuit

The sensor consists of an IR LED (SEP8736) at 880 nm, a red LED (VCC 5322F1) at 650 nm and a photodiode (PDB-C 156) with a spectral range of 400 nm to 1100 nm. The LED driving circuit is part “A” in the figure below. The “Control-Red” and the “Control-IR” signals come from the Arduino. The Arduino is used to switch the LEDs on and off, allowing the photodetector to measure the absorbance at both wavelengths independently. Resistors R1 and R2 are used to control the current through the LEDs and ensure that this is under the maximum allowed.

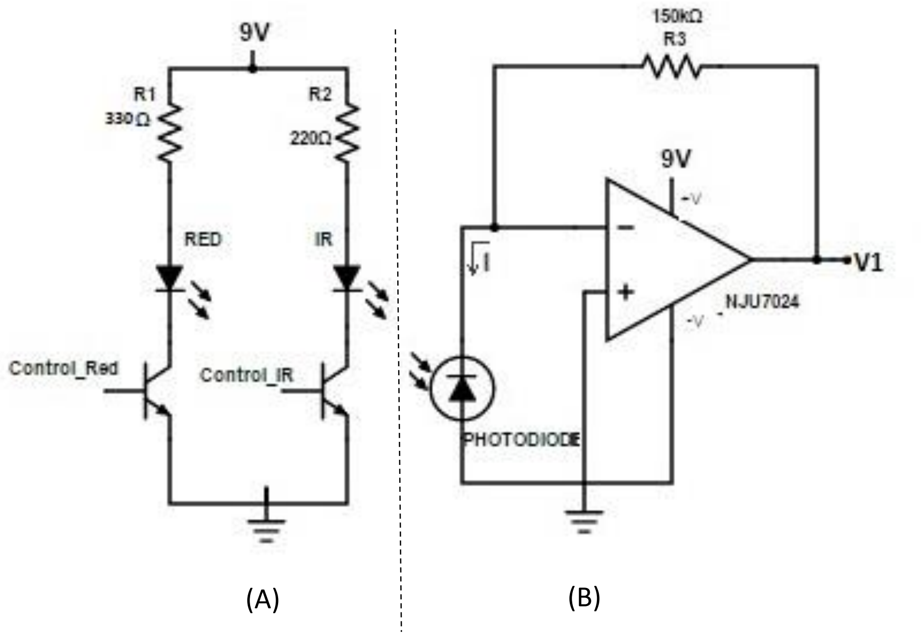


Figure 2.1. LED driving circuit and photodetector unit

2.3. Current – voltage converter

Part “B” comprises of the photodiode followed by a current-to-voltage converter. This circuit simply converts and amplifies the current output of the photodiode to a voltage. The resulting output voltage, V1 is as expressed in equation 2.1. Resistor R3 determines the gain of this stage. The DC voltage required for the oxygen saturation calculation is obtained from the end of this stage, so V1 is also an input to the Arduino.

$$V1 = I\ (R3) \tag{2.1}$$

2.4. Filtering and amplification

Part “C” passes the output of the current-voltage converter through a 1μF capacitor to filter out the DC voltage. Resistors R4 and R5 are then used to offset the ac signal halfway between ground and 9V. The signal is amplified by a single supply amplifier (NJU7024) in the non-inverting configuration and then passed through capacitor C2, which is a coupling capacitor that is used to block DC. This is followed by a low pass filter with a cutoff frequency of 15Hz to filter out any noise from the sensor. The waveform out of the amplifier is centered at 4.5 V, and since the maximum allowed input voltage to the Arduino is 5 V, this DC is eliminated and resistors R10 and R11 are used to provide a lower DC offset to the AC waveform such that the waveform is always below 5 V. Finally, the diode D2 is used to limit the negative voltage input to the microcontroller to a maximum of -0.7 V.

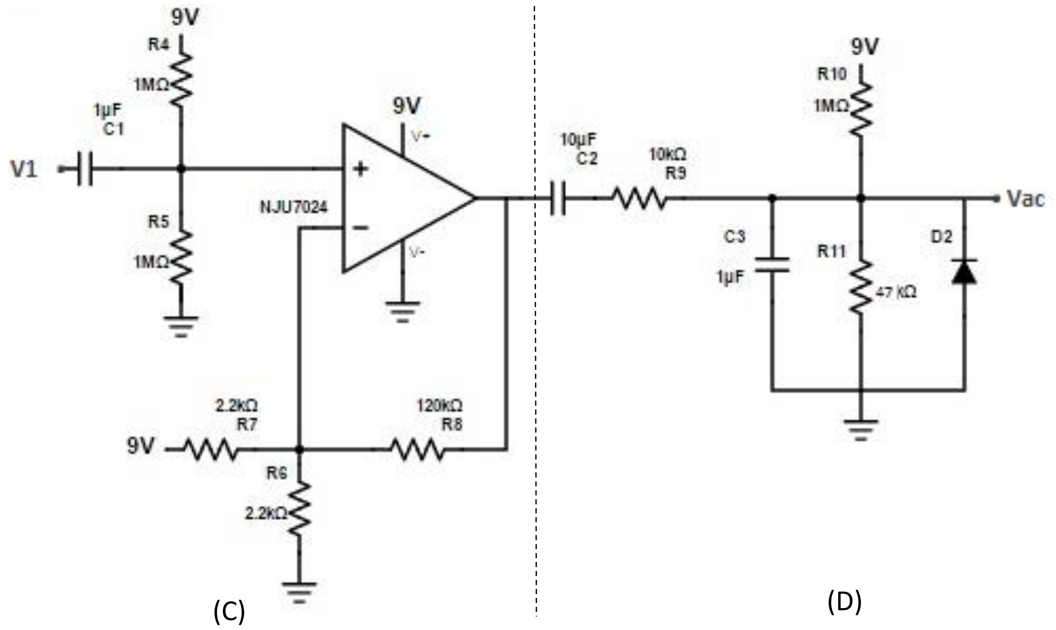


Figure 2.2. Filtering and amplification

The microcontroller and Bluetooth module are used to calculate the pulse rate and the oxygen saturation level and transmit the data to the smartphone will be discussed in the following chapter.

2.5. Electrical Simulation

LT Spice IV was used to simulate the circuitry in figure 2.2. A voltage source was used to simulate the output of the current to voltage converter. Below are the images showing the schematic and the simulated waveforms and the frequency response. Signal V1 from figures 2.1 and 2.2 was modeled as a 2mV sine wave at 2Hz with a DC offset of 3V. A single supply operational amplifier was used to simulate the performance of the NJU7024. As desired, the final waveform (Vout in figure 2.4) is centered at 0.4 V.

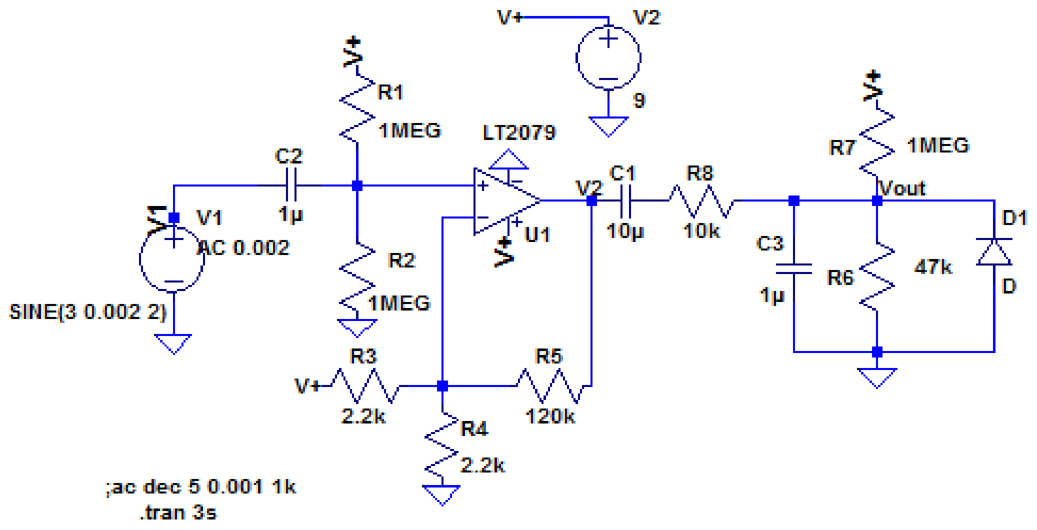


Figure 2.3. Schematic of simulated circuit

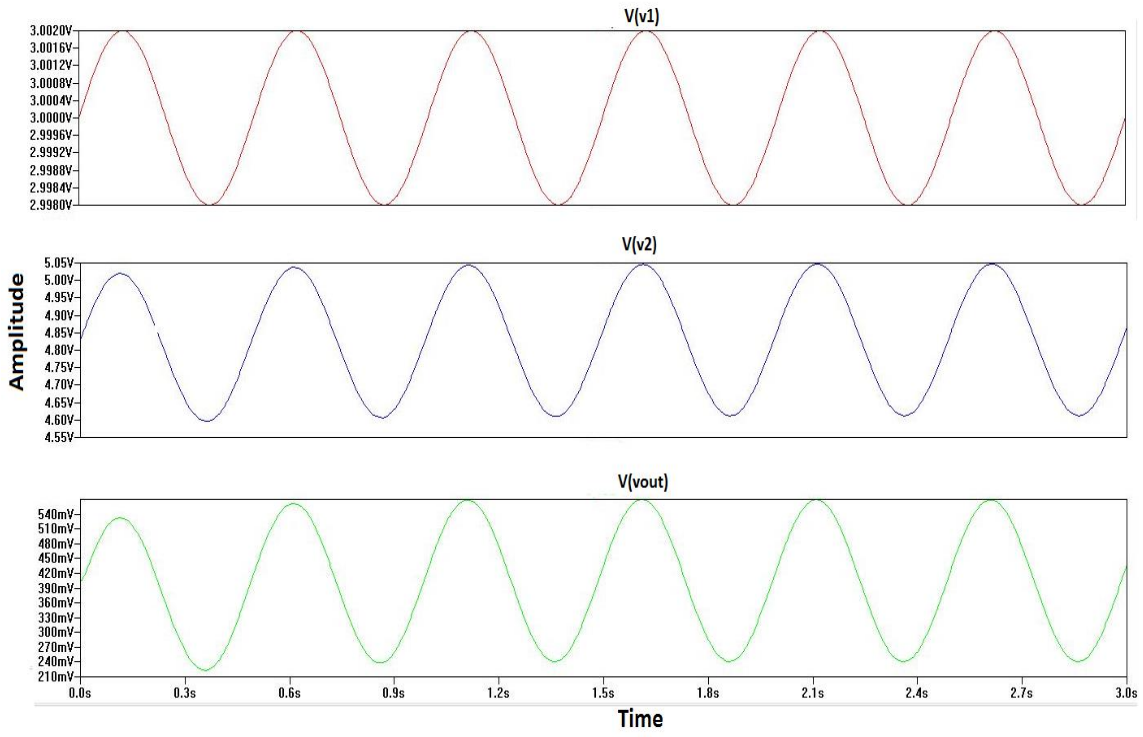


Figure 2.4. Simulated waveforms at V1, V2 and Vout

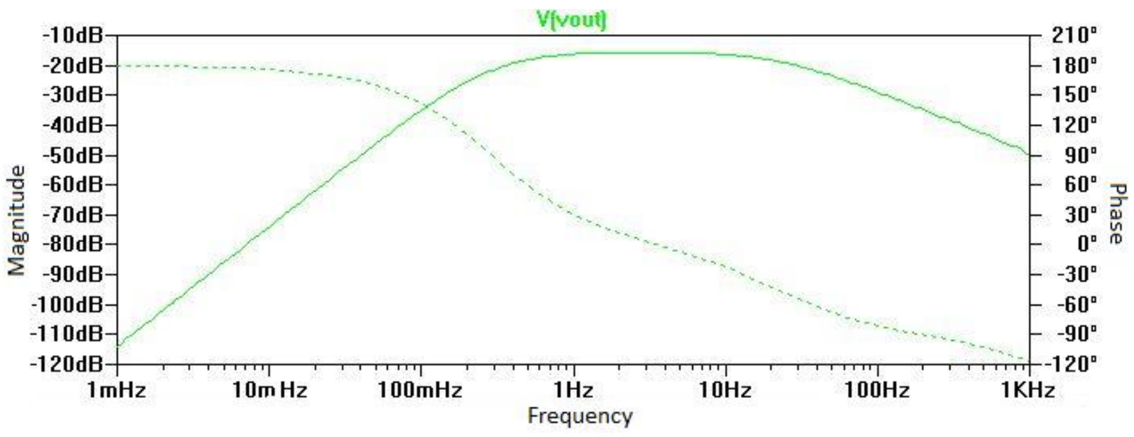


Figure 2.5. Simulated frequency response.

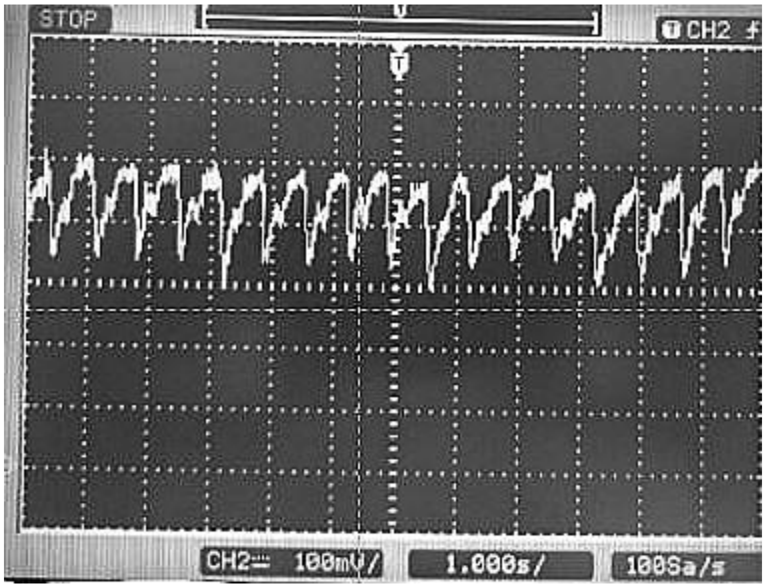


Figure 2.6. Screen capture of the output PPG waveform from the sensor

Figure 2.6 is a screen capture of a PPG waveform from the oscilloscope. The output from the sensor is a 200 mV peak to peak signal centered at 0.4 V, compared to the 300 mV peak to peak output obtained from the simulation. The amplitude and frequency of sensor output will vary from person to person, however, the signal will always be centered at 0.4 V as designed using the electronics.

Chapter 3

Arduino Programming

Arduino is an open source platform that consists of a programmable microcontroller on a board with I/O pins, and a software or an IDE (Integrated Development Environment) that can be used to write the program, upload it to the board and even view the results using a serial monitor. The Arduino Uno is the most used and most documented board in the Arduino family. It is also the most robust board made by Arduino [21]. An Arduino program is called a sketch. The Arduino IDE comes with several example sketches which can directly be compiled and uploaded to the board. In addition to the several websites and blogs that discuss projects on the Arduino, there are forums on the Arduino website itself that help answer questions and solve problems, making it a great tool for people of all skill levels.

For this project, the Arduino is used to receive data from the sensor, calculate the pulse rate and the oxygen saturation values and then send this data to the Android application using a Bluetooth module. An Arduino Uno board was used with a SparkFun BlueSMiRF Bluetooth module. This chapter discusses how to use the BlueSMiRF module and briefly explains the Arduino program.

3.1. Introduction to Bluetooth

Bluetooth is a wireless communication system that can be used to send and receive data over a short distance. Created in 1994 by Ericsson, it was originally viewed as a wireless alternative to RS-232 data cables. Bluetooth devices operates in an unlicensed frequency band at 2.4 GHz. This band ranges from 2.4GHz to 2.485 GHz and is called the industrial, scientific and medical (ISM) band. It is unlicensed and available in most countries. The technology uses a spread spectrum, frequency hopping, full-duplex signal at 1600 hops/sec [22].

The technology gets its name from the 10th century Danish king, Harold Bluetooth. The standard for Bluetooth is maintained by the Bluetooth Special Interests Group (SIG). The Bluetooth technology is built into several devices, including smartphones, laptops, tablets, cars, medical devices and even toothbrushes. The development of this technology was instrumental in reducing the requirement for wired connections, and allows users to share data, music, pictures and other information wirelessly between paired devices [22].

3.2. Bluetooth Module

SparkFun makes several different Bluetooth modems including Bluetooth Mate and BlueSMiRF. For this project, the BlueSMiRF Gold modem was used. The SparkFun Bluetooth modems can be used to communicate with any other Bluetooth device that supports SPP, including a computer or a smartphone that supports Bluetooth. If the Bluetooth module is being used with a computer that does not already have Bluetooth, a Bluetooth USB module can be plugged into the USB slot of the computer.

Bluetooth modules can have a range of 1m – 100m depending on the type of device. A class 3 Bluetooth module typically has a range of 1 meter, while a class 2 has a range of 10 m and a class 1 has a range of 100 meters. The BlueSMiRF used is a class 1 device.

A voltage from 3.3V to 6V can be used to power the device. There is a linear 3.3V regulator on board since the maximum operating voltage of the module is 3.3V. The ground pin (GND) is the 0 reference voltage, common to all the devices connected to the Bluetooth modem. The TX-O and the RX-I pins are transmit and receive pins respectively. The receive pin receives data serially and must be connected to the transmit pin of the Arduino. Similarly, the transmit pin of the Bluetooth module is used to send or transmit data serially and must be connected to the receive pin of the Arduino. The VCC, GND, TX-O and RX-I are the pins that are required for all serial communication using the modem. The RTS-O pin is the request to send pin. This is used for hardware flow control in some serial interfaces. The CTS-I or the clear to send pin is another signal to control the serial flow [23]. Neither of these pins are critical for simple serial interfaces and will be shorted together for this project. Figure 3.1 shows the different parts of the Bluetooth modem. The figure shows that there are two LEDs on the board. The stat LED is red and the connect LED is green.

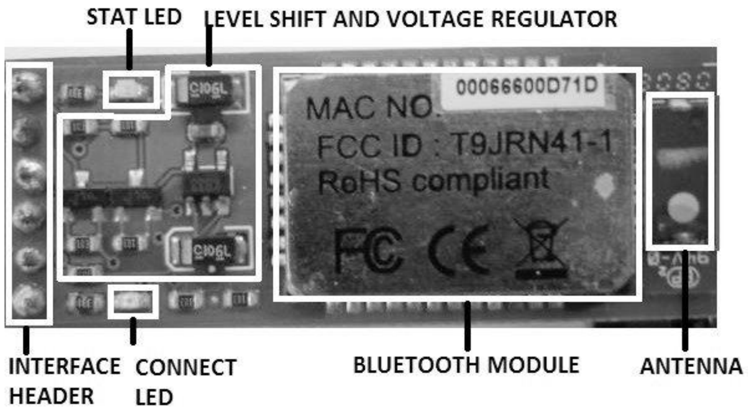


Figure 3.1. Parts of the SparkFun BlueSMiRF

3.3. Serial Communication

The Arduino hardware has built-in support for serial communication. Pins 0 and 1 on most Arduino boards are the serial receive and transmit pins. These are also connected to the computer via the USB connection. If these pins are used to communicate with the computer, they cannot be used as digital input or output pins. The begin() function is used to set the serial communication data rate in bits per second. For communicating with a computer, only the following rates cate be used: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. However, other baud rates can be specified if the device that the Arduino is communicating with requires it [24]. If the serial ports on the board are being used, Serial.begin(rate) can be called to set a particular baud rate. Since the serial pins are also connected to the computer, the serial monitor option in the toolbar menu of the Arduino IDE can be opened. Once the baud rate on the serial monitor is set to the same rate as that set using begin(), the serially transmitted data can be viewed.

In addition to the built-in serial pins, the Arduino also supports the SoftwareSerial library that can allow serial communication on any other digital pins on the Arduino board. For this project, pins 2 and 3 of the Arduino board are used for serial communication. Pin D2 of the Arduino is connected to the TX-O pin of the Bluetooth modem and the pin D3 of the Arduino board is connected to the RX-I pin of the Bluetooth modem. This means that pin D2 must be declared as the receive pin and pin D3 must be declared as the transmit pin in the Arduino code. Additionally, the VCC pin of the Bluetooth modem is connected to 5V pin and GND pin goes to the GND pin on the Arduino board.

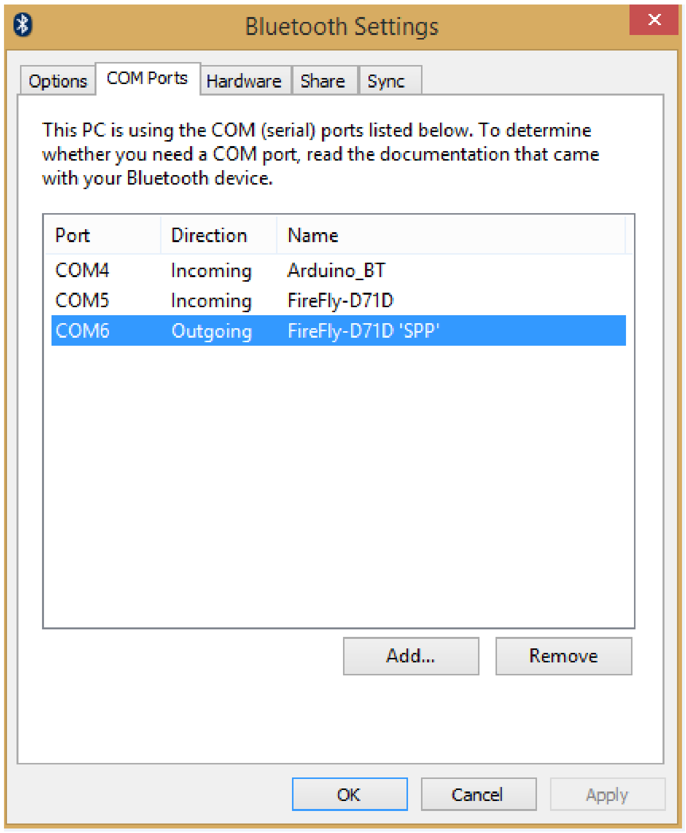
3.4. Setting up the Bluetooth modem

The Bluetooth modem can be powered up by powering the Arduino it is connected to. Once the modem is powered, it can be used to wirelessly connect with any other Bluetooth device that supports SPP (most Bluetooth devices do). Once the two devices are paired, the Bluetooth modem can be configured to operate at the desired baud rate. The default baud rate of the BlueSMiRF Bluetooth modem is 115200 bps.

The Bluetooth modem can operate in two modes – command mode and data mode. The command mode is used to configure the Bluetooth module. The device must be in command mode if the name, pin code or the baud rate of the device is to be changed. In command mode, the module can also scan for and connect to available devices within its range. In the data mode, the module is simply used to transmit and receive data.

The status LED indicates is the device is in command mode. When the device is powered up, a timer called the configuration time starts counting. The default period for the configuration timer is 60 seconds but this can be changed to a different duration (or even turned off!). Once the timer has run out, the user cannot enter the command mode until the device is powered up again. If the red status LED blinks at the rate of 10 blinks per second, this indicated that the device is in command mode. If it blinks at 2 blinks per second, the device is not in command mode but the configuration timer is still counting and the user can still enter command mode. If it blinks at one blink per second, the configuration timer has run out and the command mode can only be entered when the device is powered up again.

The Arduino was used as the medium between the computer and the Bluetooth module to send and receive commands. To connect with the module from a computer, first power up the Arduino and the BlueSMiRF. Next, ensure that the Bluetooth on the computer is on and then scan for the BlueSMiRF. The default name of the device will most likely be on the package of the device. The device used was called "FireFly-D71D. Once the device has been located, select the pairing option. At this point, the user will be prompted to enter the pin/pairing code. The default code for all the RN-41 and RN-42 devices is 1234. The device is now ready to use but the user will require the COM port in order to serially communicate with the device using the serial monitor on the Arduino IDE. When Windows installs the drivers for a new Bluetooth device, it also creates a COM port for it. The COM ports can be viewed by opening up device manager and looking in the "Ports" tree or by opening the Bluetooth settings and look in the COMS tab as shown below.



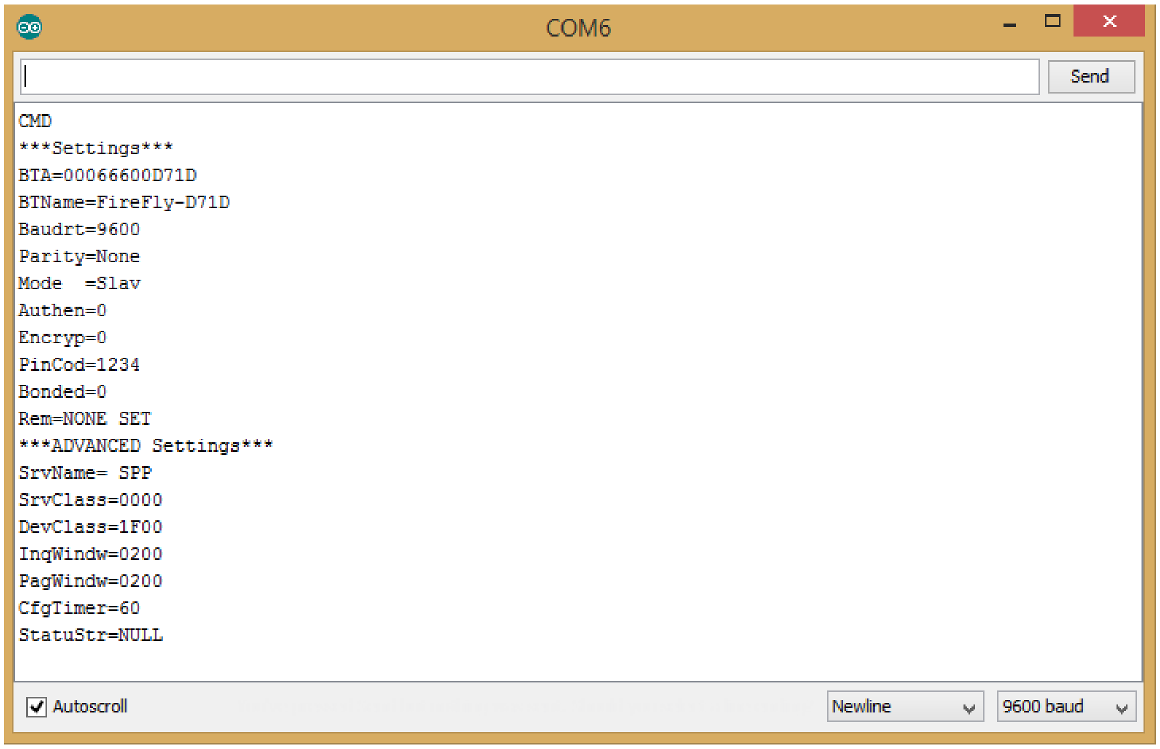
This PC is using the COM (serial) ports listed below. To determine whether you need a COM port, read the documentation that came with your Bluetooth device.

Port	Direction	Name
COM4	Incoming	Arduino_BT
COM5	Incoming	FireFly-D71D
COM6	Outgoing	FireFly-D71D 'SPP'

Figure 3.2. Bluetooth COM ports

Once the connection between the two devices is set up and the COM port is identified, the Arduino can be programmed to assist the user to enter the command mode. The SparkFun website has a tutorial on how to use the BlueSMiRF, which includes an Arduino sketch to enter the command mode (available at learn.sparkfun.com/tutorials/using-the-bluesmirf). The code uses the SoftwareSerial library to assign serial transmit and receive functions to two digital pins and then calls the begin() function with the baud rate of 115200 since this is the default rate.

After the sketch is uploaded, select the COM port associated with the Bluetooth module from the list of ports in the Tools menu and then open up the serial monitor, again, from the Tools dropdown menu. In the serial monitor window, set the baud rate to 9600 and from the drop-down menu next to the baud rate, select “No line ending”. The command mode can be entered by typing in '$$$' into the serial monitor. If the module successfully enters the command mode, it responds with CMD and the stat LED starts blinking much faster. Once it is in the command mode, you can view or change the settings. Settings can be viewed or changed after setting the line ending dropdown to "Newline". Typing in "D" and then "E" in the serial monitor returns some basic information on the device including its name, address and baud rate.



```
CMD
***Settings***
BTA=00066600D71D
BTName=FireFly-D71D
Baudrt=9600
Parity=None
Mode  =Slav
Authen=0
Encryp=0
PinCod=1234
Bonded=0
Rem=NONE SET
***ADVANCED Settings***
SrvName= SPP
SrvClass=0000
DevClass=1F00
InqWindw=0200
PagWindw=0200
CfgTimer=60
StatuStr=NULL
```

Figure 3.3. Device information extracted using command mode

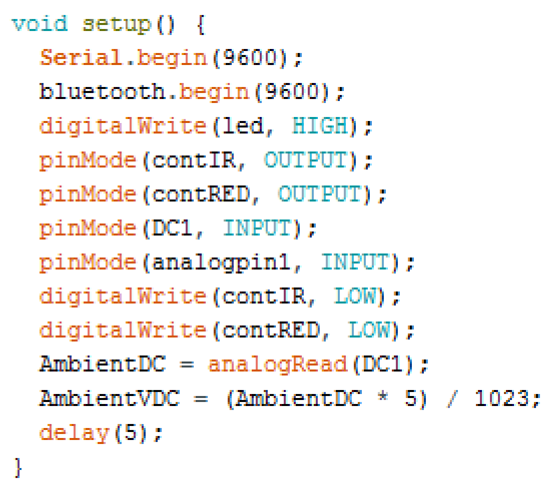
Since the default baud rate of the device is 115200, it needs to be changed to 9600 for this project. The baud rate can be changed to 9600 by sending "SU,96" to the module through the serial monitor. If the baud rate has been successfully changed, the module responds with "AOK" and the device is now ready to be used for the project. The device name and pin code can be changed as well, but it was left at the default setting for this project.

3.5. Arduino Code

The Arduino code is quite simple. First, the SoftwareSerial library must be included so that the Bluetooth device can use any (assigned) digital pins to exchange data with the Arduino. The pins used to transmit and receive data must then be declared along with the serial object. In this case, the serial object has been named “bluetooth”.

```
#include <SoftwareSerial.h>

int bluetoothTx = 2;  // TX-O pin of bluetooth modem, Arduino D2
int bluetoothRx = 3;  // RX-I pin of bluetooth modem, Arduino D3

SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);
```

The variables used in the code are declared and initialized, if necessary, before the setup() function is called. The setup() function is called when the sketch starts and runs only once each time the Arduino is powered up while the loop() function, which follows the setup() function, continuously loops and runs until the Arduino is powered off. The setup() function can be used to initialize variables, set up the pin modes or start the library functions. In this code, it is used to begin serial communication with the Bluetooth module and the computer, set up pin modes, and measure the ambient light when both the red and the infrared LEDs are off. The analogRead maps a voltage from 0 to 5 V and maps it to an integer value between 0 and 1023. The code converts this back to a value between 0 and 5 before transmitting the value to the smartphone.



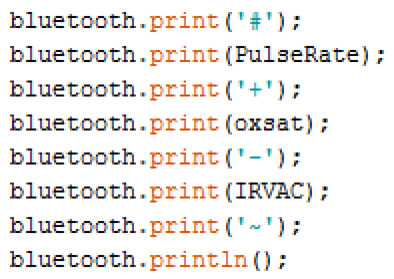
```
void setup() {
  Serial.begin(9600);
  bluetooth.begin(9600);
  digitalWrite(led, HIGH);
  pinMode(contIR, OUTPUT);
  pinMode(contRED, OUTPUT);
  pinMode(DC1, INPUT);
  pinMode(analogpin1, INPUT);
  digitalWrite(contIR, LOW);
  digitalWrite(contRED, LOW);
  AmbientDC = analogRead(DC1);
  AmbientVDC = (AmbientDC * 5) / 1023;
  delay(5);
}
```

The loop() function is used to alternatively switch between the red and the infrared LEDs, measure the AC and DC voltages, calculate the pulse rate and the oxygen saturation and transmit this along with the AC waveform from the infrared LED to the smartphone. Equation 3.1 is used to calculate the oxygen saturation value.

$$SpO_2 = A + B * \left(\frac{\left(\frac{AC}{DC}\right)_{RED}}{\left(\frac{AC}{DC}\right)_{IR}} \right) \tag{3.1}$$

Where A is 104.86 and B is -23.288. The device was calibrated to obtain this relationship. The calibration is discussed in the results section.

The pulse rate was calculated from the AC waveform of the infrared LED. Since this waveform is centered at 0.40V (set using R6 and R7 in figure 2.2), the Arduino was used to check for the when the waveform increased beyond 0.40 V. The function millis() returns the number of milliseconds since the Arduino board began running the current program. This was used to save a time stamp at each crossing and the time stamps were then used to calculate the period of the waveform.



```
bluetooth.print('#');
bluetooth.print(PulseRate);
bluetooth.print('+');
bluetooth.print(oxsat);
bluetooth.print('-');
bluetooth.print(IRVAC);
bluetooth.print('~');
bluetooth.println();
```

At the end of each iteration, the AC voltage measured from the IR LED, the calculated pulse rate and the oxygen saturation are transmitted to the smartphone. To help the Android identify the beginning and ending of each transmission, the first character transmitted is "#" and the last character transmitted is "~". Additionally, to help the Android sort between the pulse rate, the oxygen saturation and the AC voltage reading, more special characters are used as shown above. The Android code, discussed in the next chapter, uses these characters to identify the data from the input stream.

Chapter 4

Building and Implementing an Android Application

The aim of the android application is to receive the sensor data via Bluetooth and display it. So, the application should be able to check if the device has a Bluetooth module that can be used, turn this module on, connect with the Arduino, receive the data, and finally display and plot the data. The following chapter briefly explains the Android programming fundamentals and explains the objectives that were implemented. Reference [25] was used to understand the theory explained in this chapter.

4.1. Android Programming Fundamentals

Android applications are written in Java programming language. The Android SDK or the Android Software Development Kit is a set of software development tools that compiles the Java code along with any data and resource files like libraries, pictures, audio clips, etc., to generate a single .apk file. The .apk (Android Package) is the file format that is used to distribute or install the application onto Android devices.

The Android operating system works like a multi-user Linux system, where each application is a different user. The applications are each assigned a unique Linux user ID and only the assigned user ID can access the permissions and data associated with that app. However, it is possible for two different apps to have the same Linux user ID, allowing them to access each other's files. An application must request permission to use the data and resources on the device like the camera, Bluetooth, user's contact list and messages etc. These permissions are granted by the user when the app is being installed.

4.1.1. Application Components

a. Activity

An activity is a single screen with a user interface. In the app written for this project, for example, one activity displays the home screen, another displays the Bluetooth paired devices and yet another activity displays the results from the sensor. Even though the activities run in a sequence in the app, they can function independently of each other. A different app can also access these activities, if the app allows it. For example, the camera or the microphone app could open any messaging app to share pictures or recordings.

b. Services

A service does not provide any user interface. It runs in the background without blocking user interaction with another activity. For example, the media player app allows music to continue playing even after the user opens a different app.

c. Content Provider

The content provider allows the app to store data. If the content provider allows it, other apps can view or even modify this data. For example, the Android system has a content provider that manages the user's contact list information. With the appropriate permissions, other apps can view, edit or add to the list.

d. Broadcast Receivers

The broadcast receiver receives system-wide broadcasts. Several broadcasts, like those announcing the battery is low or that an image has been captured, come from the system itself but apps can also initiate broadcasts. Broadcast receivers do not have a user interface but apps can create status bar notifications to notify the user when an event occurs.

4.1.2. Intent

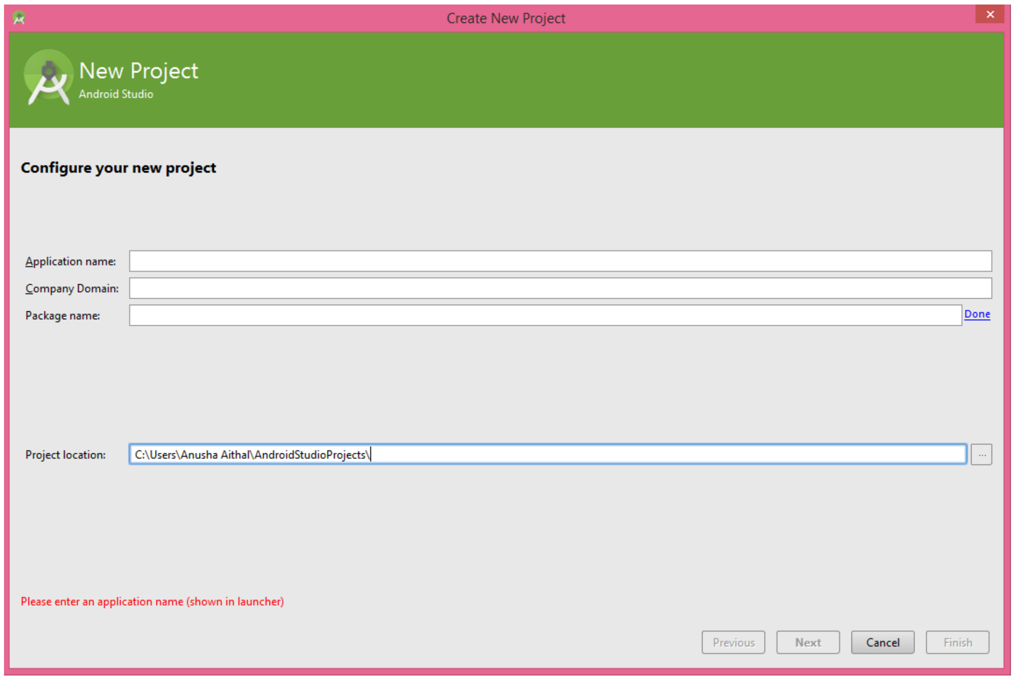
An intent is an asynchronous message that is used to start three of the four android components, namely, activities, services and broadcast receivers. At runtime, an intent can bind various components to each other, even components that are not a part of your app. There are separate methods for starting each component. An activity can be started by passing an Intent (a messaging object) to startActivity() or startActivityForResult() (when you want the activity to return a result). To start a service, the Intent must be passed to startService() and a broadcast can be initiated by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

4.2. Android Studio

Most Android developers use Java to develop their app. Eclipse, NetBeans and IntelliJ IDEA are amongst the most common editors used. For this project, Android Studio, which is based on IntelliJ IDEA was used. Android Studio is the official IDE for Android Application Development. The first step to developing an app is downloading the editor (Android Studio in this case) and then downloading the SDK tools and platform using the SDK manager. The SDK manager can be launched from the File menu (File > Settings > Appearance & Behavior > System Settings > Android SDK), the Tools menu (Tools > Android > SDK Manager), or by simply clicking the SDK Manager Icon on the menu bar. The SDK manager installs the required packages and the latest tools by default. In addition, there is a checkbox next to each available SDK platform or tool. These can be added or updated by simply checking the boxes, and clicking **Apply** and **OK.** For this project, the latest available packages at the time were installed, in addition to these packages, packages and tools available for Android Versions 2.2 and 2.3.3 (API levels 8 and 10 respectively) were installed since the Android Version of the phone used to test the applications was 2.3.6. Once the editor and the SDK tools are installed, the app can be developed.

4.2.1. Creating a Project

Every application has an application name, a domain name and a package name. The application name is the name that appears to the users. The company domain name is a qualifier that will be appended to the package name. It has to have at least two identifiers. Android Studio typically remembers this domain name for every project you create. The package name needs to be unique if the application is to be released to the market. For example, if the application name is “My Application” and the company domain name is “companyname.com”, then the package name defaults to “com.companyname.myapplication” but this can be edited independent of the domain and application name. Once the application, domain and package names are set, you need to select the form factors your applications can run on and the minimum SDK. Since we require this application to run only on phones, the **Phone and Tablet** option was selected and API level 8 was set as the minimum SDK. The minimum SDK is the earliest version of Android that can run the application. By targeting API level 8, the app will run on almost all the devices active on Google Play.



Configure your new project

Application name:
Company Domain:
Package name:

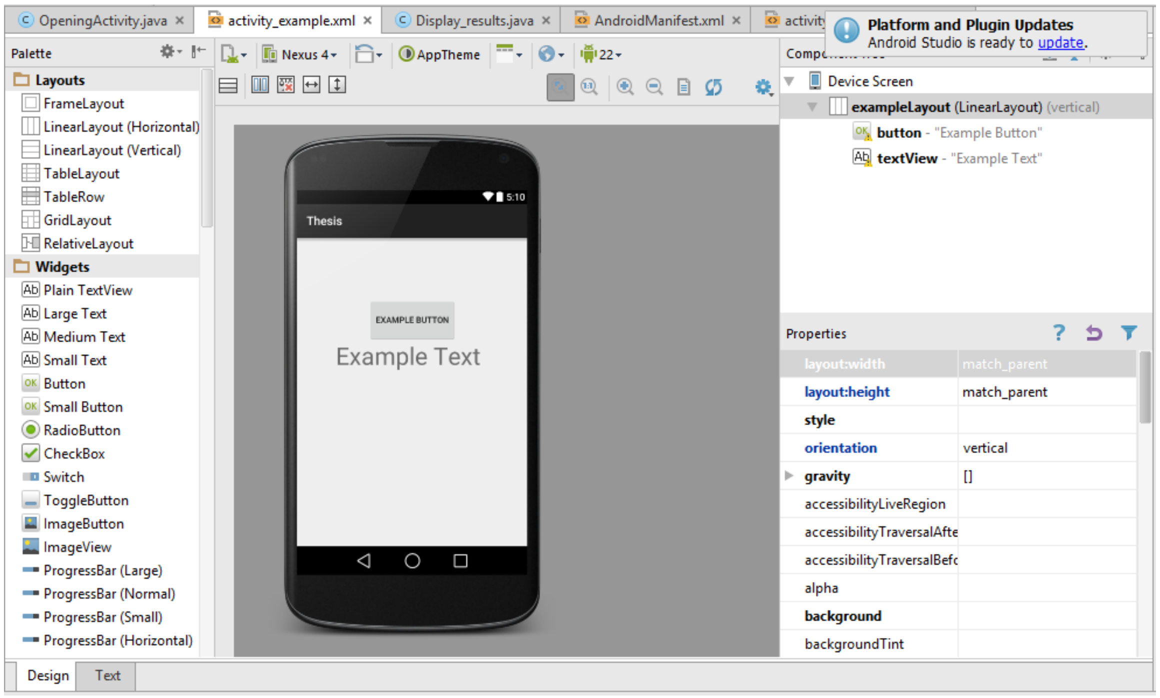
Project location: C:\Users\Anusha Aithal\AndroidStudioProjects\

Please enter an application name (shown in launcher)

Figure 4.1. Configuring a new Android Studio Project

4.2.2. Layout XML File

A layout file can be used to define the visual structure for the user interface of the activity. The layout file can be found in the "res" folder and it can be edited either using the graphical design assistant, where items can be dragged and placed as required, or by typing it out in XML (Extensible Markup Language). The application could also include code to modify the screen objects at runtime. The figure below shows the layout of an activity called "activity_example" and it includes a linear layout with a TextView and a Button.



Platform and Plugin Updates
Android Studio is ready to update.

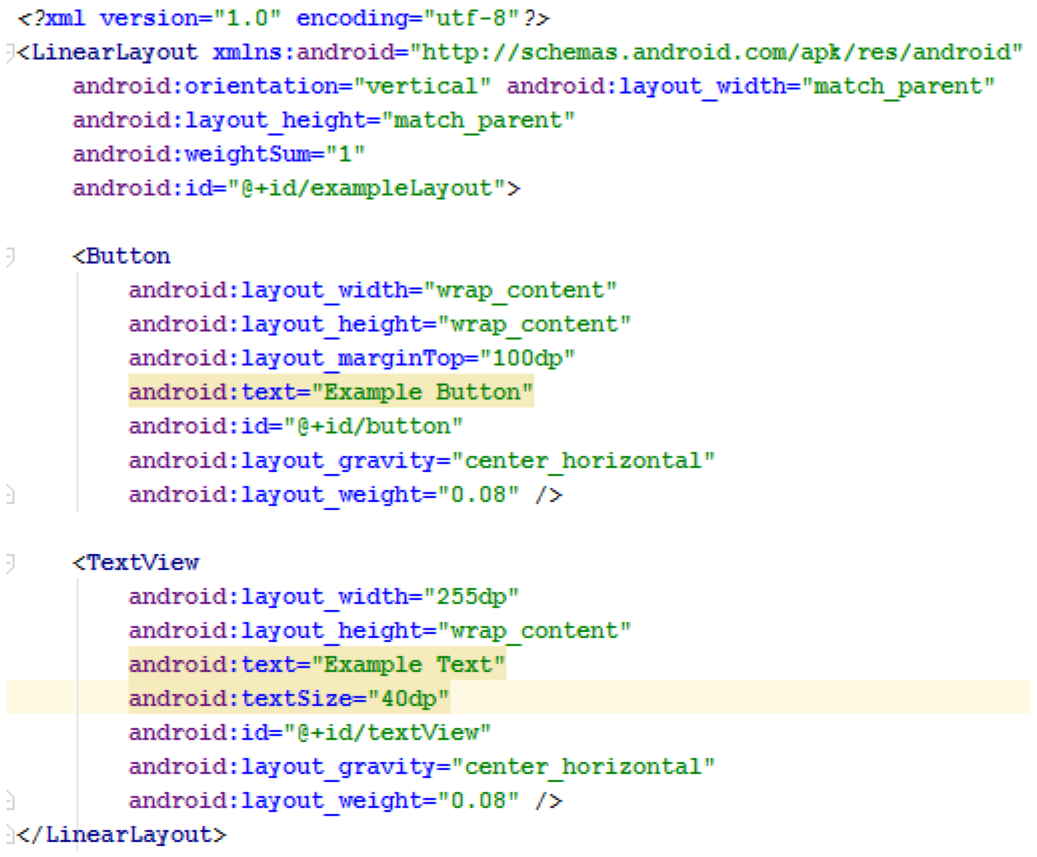
Device Screen
exampleLayout (LinearLayout) (vertical)
button - "Example Button"
textView - "Example Text"

Properties

layout:width	match_parent
layout:height	match_parent
style	
orientation	vertical
gravity	[]
accessibilityLiveRegion	
accessibilityTraversalAfte	
accessibilityTraversalBefo	
alpha	
background	
backgroundTint	

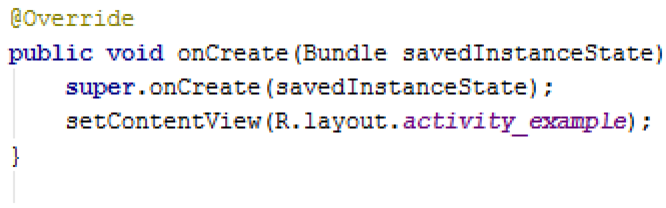
Figure 4.2. Example of an Activity Layout

Every layout must contain one root element, which must be a View or ViewGroup object. This element acts like the basic building block for the UI components. Once this has been defined, additional objects and widgets can be added. In this example, a vertical linear layout is used as the root element. The android:orientation field in the code below can be horizontal or vertical. The vertical linear layout arranges its children components in a single column.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="1"
    android:id="@+id/exampleLayout">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:text="Example Button"
        android:id="@+id/button"
        android:layout_gravity="center_horizontal"
        android:layout_weight="0.08" />

    <TextView
        android:layout_width="255dp"
        android:layout_height="wrap_content"
        android:text="Example Text"
        android:textSize="40dp"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal"
        android:layout_weight="0.08" />
</LinearLayout>
```

The layout file can be loaded on to the application code in the Activity.onCreate() implementation. Since we named our layout activity_example, this can be done be calling setContentView() and passing the reference R.layout.activity_example to it.



```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_example);
}
```

Any of the View objects can have an id associated with them. In the example code, all the components have a unique ID. These objects can be accessed in the main java code by referencing the resource ID. For example, in the java code, the text in the TextView object can be edited by creating an instance of the object as shown below.

```
TextView text1 = (TextView) findViewById(R.id.textView);
text1.setText("Text has been edited");
```

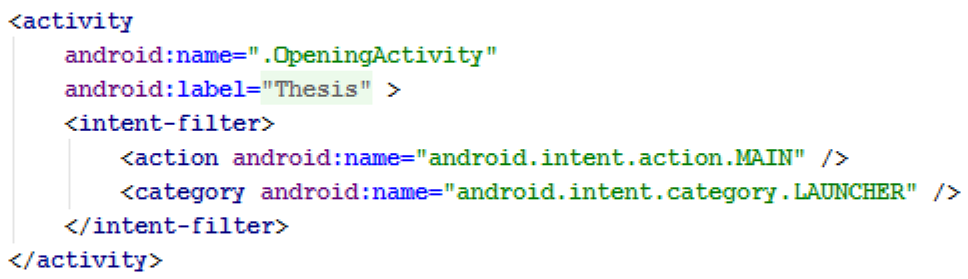
While the user interface can be set using in layout XML file, the application itself is primarily written in java. The java files are located in the “src” or “java” folder. The sections below provide more details on how the java file was written to reach the objectives of the application.

4.2.3. The Manifest File

Every app has a manifest file. All the components of the application have to be declared in this file. The manifest file includes a lot of information on the app including the minimum API level, the API libraries used, all the permissions the app needs and the hardware and software features required by the app like Bluetooth services or the camera.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Since this app uses Bluetooth and writes to the external storage device, these permissions have to be included in the AndroidManifest.xml file as shown above. The Bluetooth permission BLUETOOTH is required to perform any Bluetooth communication, such as requesting a connection, accepting a connection, and transferring data. The BLUETOOTH_ADMIN permission is required if the app is to change the Bluetooth settings or initiate discovery. If you use the BLUETOOTH_ADMIN permission, you also need to include the BLUETOOTH permission.

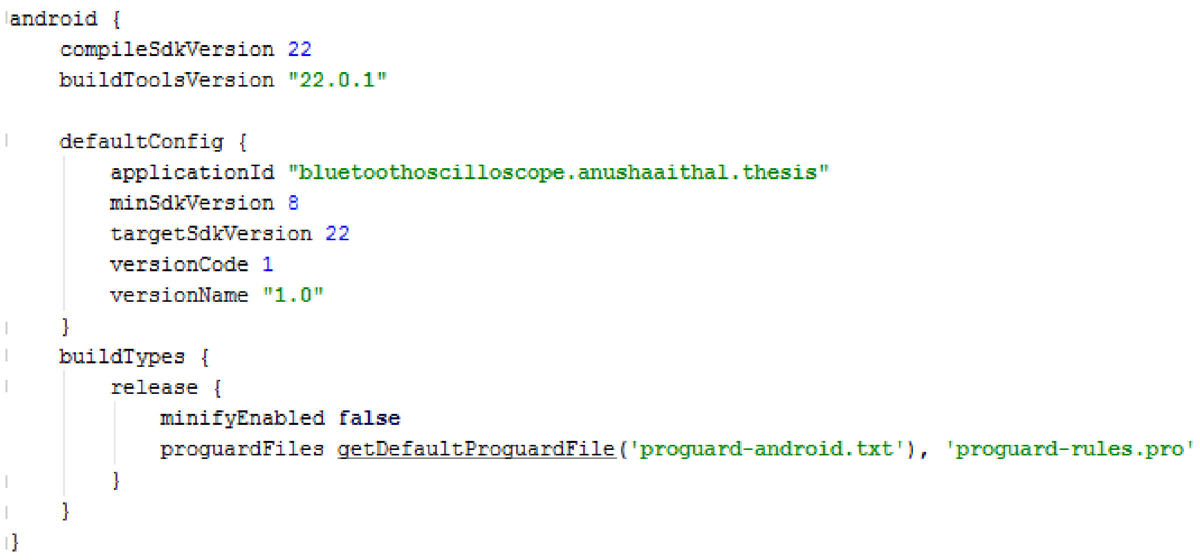


```
<activity
    android:name=".OpeningActivity"
    android:label="Thesis" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

The Manifest file uses <activity>, <service>, <receiver> and <provider> to declare the activity, service, broadcast receiver and content provider components respectively. When declaring an activity, android:name attribute specifies the fully qualified class name of the activity subclass and the android:label attributes specifies a string to use as the user-visible label for the activity. An intent filter can also be optionally included during the activity declaration that declare the capabilities of the app so it can also respond to intents from other apps. The intent filter is declared by including an <intent-filter> element as a child of the component declaration element as shown above. The <action> element declares the intent action accepted, in the name attribute. The <category> element declares the intent category accepted, in the name attribute. If the category is launcher, the activity is the initial activity. So in this case, when the app is first opened, the activity called "OpeningActivity" is launched first.

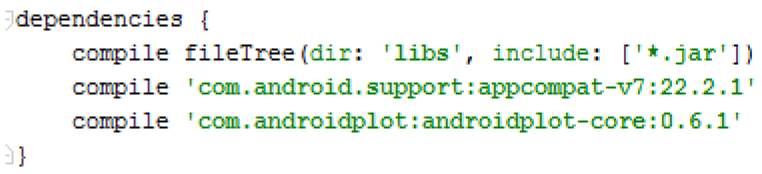
4.2.4. Gradle

Android Studio uses Gradle to compile and build the app. There is a separate build.gradle file for each module of the project as well as one for the entire project. The build.gradle file for the app is where the dependencies are listed.



```
android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "bluetoothoscilloscope.anushaaithal.thesis"
        minSdkVersion 8
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

The build.gradle file lists the version of the SDK the project is compiled against. This is typically the latest version available, unless changed by the user. It should be Android 4.1 or greater. The application ID is the full package name. The minimum SDK version is the earliest Android version that the app supports. The target SDK version is the highest Android version the app has been tested against.



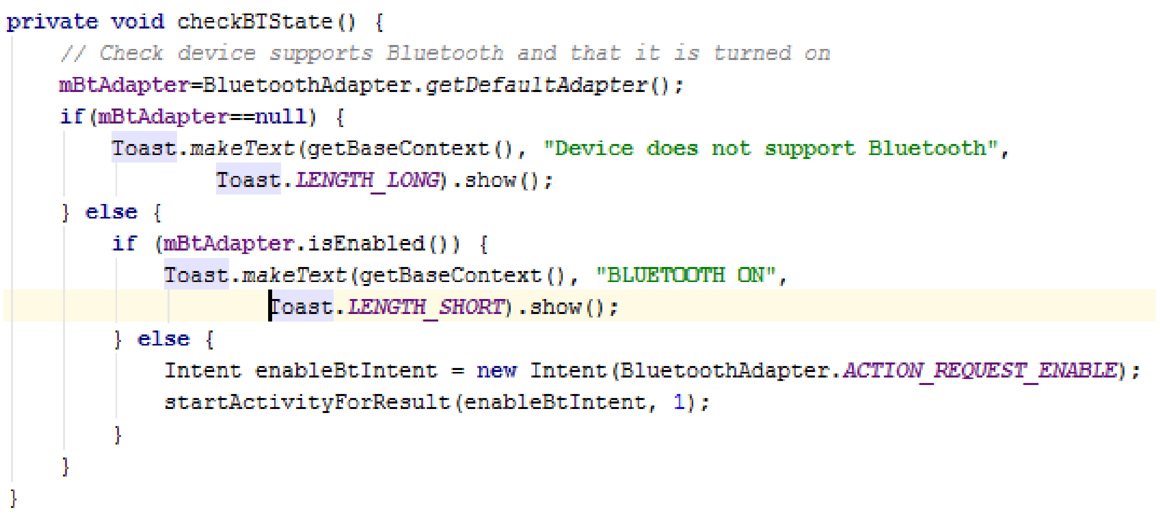
```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.androidplot:androidplot-core:0.6.1'
}
```

The gradle file also lists all the dependencies. The dependency element is declared after the android element. The com.android.support:appcompat-v7:22.2.1 declares a dependency on version 22.2.1 of the Android Support Library. The Android Support Library is available in the Android Repository package of the Android SDK. This is one of the packages that must be downloaded and installed. The compile fileTree(dir: 'libs', include: ['*.jar']) tells the build system that any JAR file inside app/libs is a dependency and should be included in the compilation classpath and in the final package. To plot the data in real time, an additional library called AndroidPlot was used. AndroidPlot is a free tool to graph data and will be discussed later in this chapter. To include this library, the .jar file was downloaded and included in the libs folder of the project (C:\Users\Username\AndroidStudioProjects\Appname\app\libs). Additionally, the .jar file should be included in the dependencies element. Whenever any changes are made to the build.gradle file, Android Studio requires a project sync to import the configuration changes.

4.3. Bluetooth

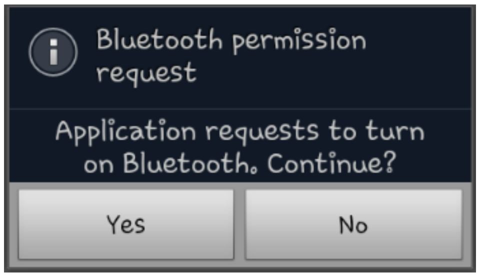
4.3.1. Setting up Bluetooth

In order to use any of the Bluetooth functions, the Bluetooth permissions must be set up as mentioned earlier and the android.bluetooth package must be included in the Java code. Once this is included, the first step would be to check if the device supports Bluetooth. If it does not support Bluetooth, then all features will be disabled and the app cannot be used. If the device does support Bluetooth, the next step would be to check if the Bluetooth is on/enabled. If Bluetooth is supported but disabled, the user can be prompted to enable it without leaving the application.



```java
private void checkBTState() {
    // Check device supports Bluetooth and that it is turned on
    mBtAdapter=BluetoothAdapter.getDefaultAdapter();
    if(mBtAdapter==null) {
        Toast.makeText(getBaseContext(), "Device does not support Bluetooth",
                Toast.LENGTH_LONG).show();
    } else {
        if (mBtAdapter.isEnabled()) {
            Toast.makeText(getBaseContext(), "BLUETOOTH ON",
                    Toast.LENGTH_SHORT).show();
        } else {
            Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent, 1);
        }
    }
}
```

The BluetoothAdapter is required for any Bluetooth activity. There is one Bluetooth adapter for the entire system. Calling the getDefaultAdapter() function returns this adapter and the application can interact with the Bluetooth adapter using this object. If the function returns a null, it means that the system does not support Bluetooth and the app can't proceed any further. Once we have the Bluetooth adapter, we can check if it is enabled by calling the isEnabled() function. If it returns false, the Bluetooth adapter is disabled. To request that the Bluetooth be enabled, startActivityForResult() must be called with an ACTION_REQUEST_ENABLE action intent. This causes a dialog box to appear that requests the user to enable Bluetooth as shown in figure 4.3.


Bluetooth permission request

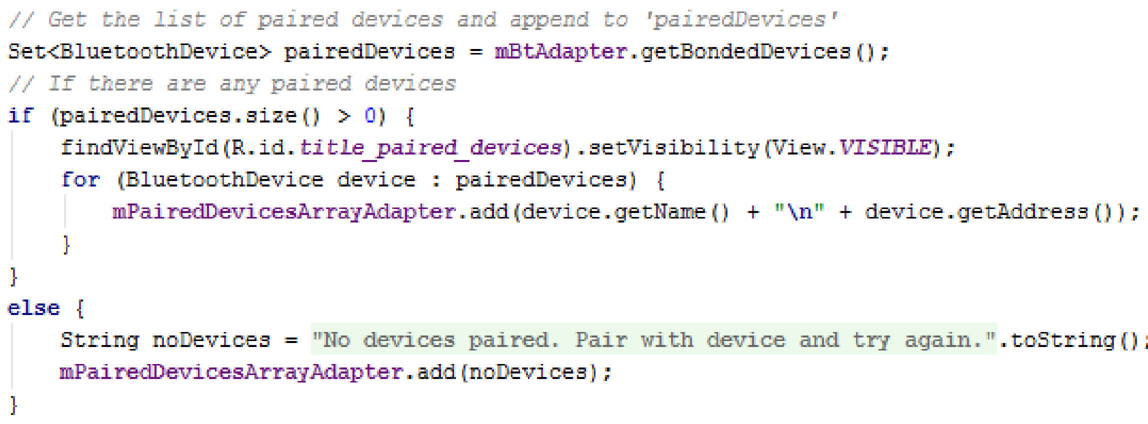
Application requests to turn on Bluetooth. Continue?

Figure 4.3. Enable Bluetooth dialogue

4.3.2. Finding and connecting with other devices

BluetoothAdapter can be used to find new devices or query the list of paired devices. The device discovery or inquiry is a procedure when the device scans the area for other Bluetooth devices within its range and requests information about each one of them. If the device is discoverable, that is, if the device settings enable it to be visible/discoverable, it responds to this request by sending its Bluetooth name, class and unique MAC address back to the device that is currently scanning. Using this information, the first device can initiate a connection with the second. When a connection is made for the first time, a pairing request is automatically sent. Once the devices are paired, the basic Bluetooth information is saved and a scan would not have to be performed to connect with this device at a later time.

To get a list of the paired devices, getBondedeDevices() can be called. This will return a set of BluetoothDevices that represent the paired devices. BluetoothDevice represents a remote Bluetooth device. This allows you to query information on the device, like name, class and address, and it allows you to connect to a device. If there are no paired devices, then the app displays a message that says "No devices paired. Pair with device and try again".



```
// Get the list of paired devices and append to 'pairedDevices'
Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
// If there are any paired devices
if (pairedDevices.size() > 0) {
    findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
    for (BluetoothDevice device : pairedDevices) {
        mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
else {
    String noDevices = "No devices paired. Pair with device and try again.".toString();
    mPairedDevicesArrayAdapter.add(noDevices);
}
```

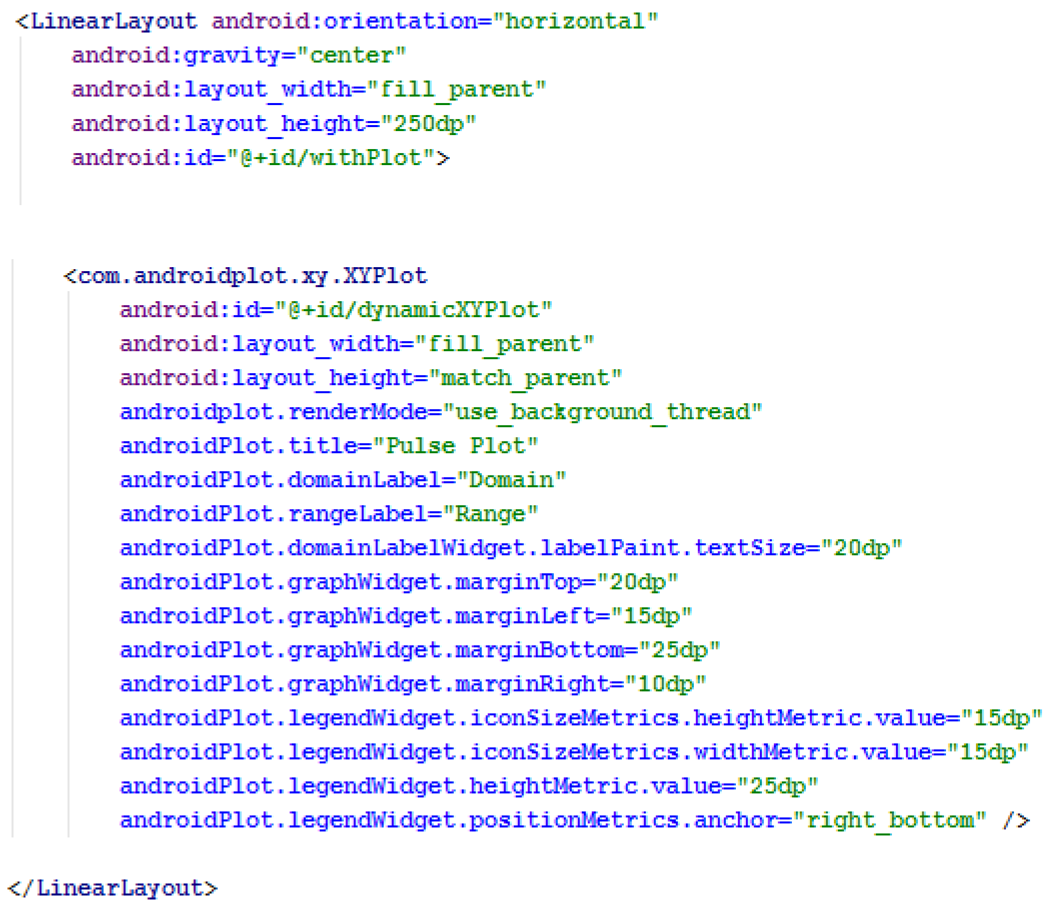
Once the list of paired devices has been displayed, the user can select any of the devices listed and the app attempts to connect to the device. If the device is within range and discoverable, the app connects the two devices and moves on to the next screen where the received data is displayed. If the device chosen is not available, the app displays a message to let the user know that the connection was not successful.

To scan for new devices, startDiscovery() can be called. This process first returns with a Boolean indicating if the process has been started successfully, and then it retrieves the names of the Bluetooth devices found. The application must have a BroadcastReceiver for the ACTION_FOUND event to receive information about the devices discovered.

4.4. Plotting data in real time

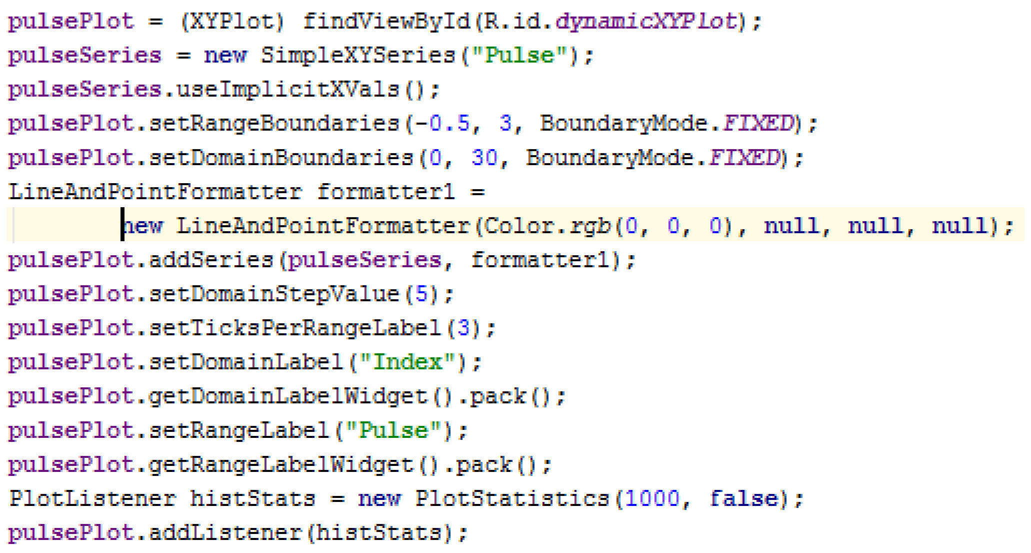
To plot the data, an additional library called AndroidPlot was included as mentioned previously. AndroidPlot is an API for creating charts within an Android application. It is compatible with all versions of Android from 1.6 onward and is used by over 500 applications on Google Play. It is a free tool that can be used to make line, bar, scatter, step and pie charts and it allows both static and dynamic charting [26].

To create a plot in the application using AndroidPlot, the widget must first be declared in the Manifest file and the Gradle. It must also be defined in the XML file, where the size of the plot and the tags are set as shown below.



```
<LinearLayout android:orientation="horizontal"
    android:gravity="center"
    android:layout_width="fill_parent"
    android:layout_height="250dp"
    android:id="@+id/withPlot">


    <com.androidplot.xy.XYPlot
        android:id="@+id/dynamicXYPlot"
        android:layout_width="fill_parent"
        android:layout_height="match_parent"
        androidplot.renderMode="use_background_thread"
        androidPlot.title="Pulse Plot"
        androidPlot.domainLabel="Domain"
        androidPlot.rangeLabel="Range"
        androidPlot.domainLabelWidget.labelPaint.textSize="20dp"
        androidPlot.graphWidget.marginTop="20dp"
        androidPlot.graphWidget.marginLeft="15dp"
        androidPlot.graphWidget.marginBottom="25dp"
        androidPlot.graphWidget.marginRight="10dp"
        androidPlot.legendWidget.iconSizeMetrics.heightMetric.value="15dp"
        androidPlot.legendWidget.iconSizeMetrics.widthMetric.value="15dp"
        androidPlot.legendWidget.heightMetric.value="25dp"
        androidPlot.legendWidget.positionMetrics.anchor="right_bottom" />

</LinearLayout>
```

The AndroidPlot classes must be included in the java program at the beginning of the code.



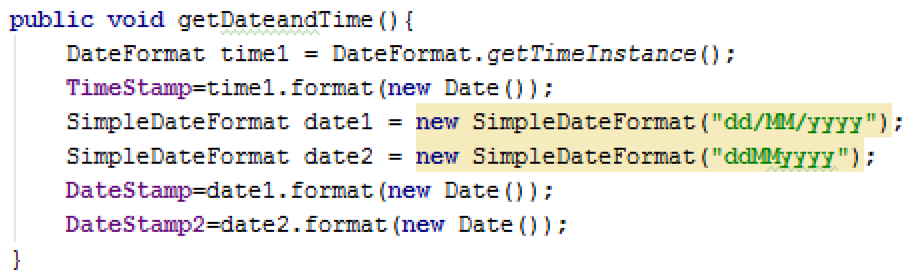
```
pulsePlot = (XYPlot) findViewById(R.id.dynamicXYPlot);
pulseSeries = new SimpleXYSeries("Pulse");
pulseSeries.useImplicitXVals();
pulsePlot.setRangeBoundaries(-0.5, 3, BoundaryMode.FIXED);
pulsePlot.setDomainBoundaries(0, 30, BoundaryMode.FIXED);
LineAndPointFormatter formatter1 =
        new LineAndPointFormatter(Color.rgb(0, 0, 0), null, null, null);
pulsePlot.addSeries(pulseSeries, formatter1);
pulsePlot.setDomainStepValue(5);
pulsePlot.setTicksPerRangeLabel(3);
pulsePlot.setDomainLabel("Index");
pulsePlot.getDomainLabelWidget().pack();
pulsePlot.setRangeLabel("Pulse");
pulsePlot.getRangeLabelWidget().pack();
PlotListener histStats = new PlotStatistics(1000, false);
pulsePlot.addListener(histStats);
```

4.5. Saving the Oximeter Data

The sensor data is stored on the external memory of the phone. The purpose of this file is to have a record of all the values measured by the sensor, to be viewed at a later point if necessary. The app saves it as a .txt file but it can easily be opened in Microsoft Excel, making it easier to go through. Any irregularity in the pulse rate or oxygen saturation levels can be identified using this document if the sensor was on when the event occurred. To save the file, the app first creates a folder called “PulseOximeter” if it does not already exist and then it creates a file called “OximeterData” followed by the date. The data is then saved on this file.

4.5.1. Getting date and time

To get the current date and time, Android classes java.util.Date, java.text.SimpleDateFormat and java.text.DateFormat need to be included.

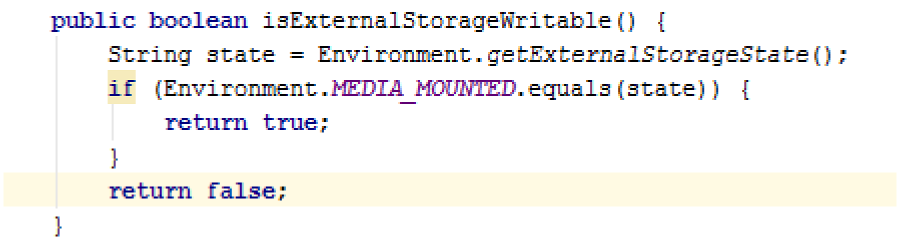


```java
public void getDateandTime(){
    DateFormat time1 = DateFormat.getTimeInstance();
    TimeStamp=time1.format(new Date());
    SimpleDateFormat date1 = new SimpleDateFormat("dd/MM/yyyy");
    SimpleDateFormat date2 = new SimpleDateFormat("ddMMyyyy");
    DateStamp=date1.format(new Date());
    DateStamp2=date2.format(new Date());
}
```

The time gets saved in the hh:mm:ss format. The date is saved in two different formats because the file name cannot have special characters in it.

4.5.2. Creating and saving to file

Before creating a file, we need to make sure that the required permissions are listed in the Android Manifest file. WRITE_EXTERNAL_STORAGE and READ_EXTERNAL_STORAGE are the permissions required to write to and read files on the external storage respectively. Since we will only be writing to the file, we do not need to list the permission required to read from external storage. Next, we need to check if the external storage is available to read/write.

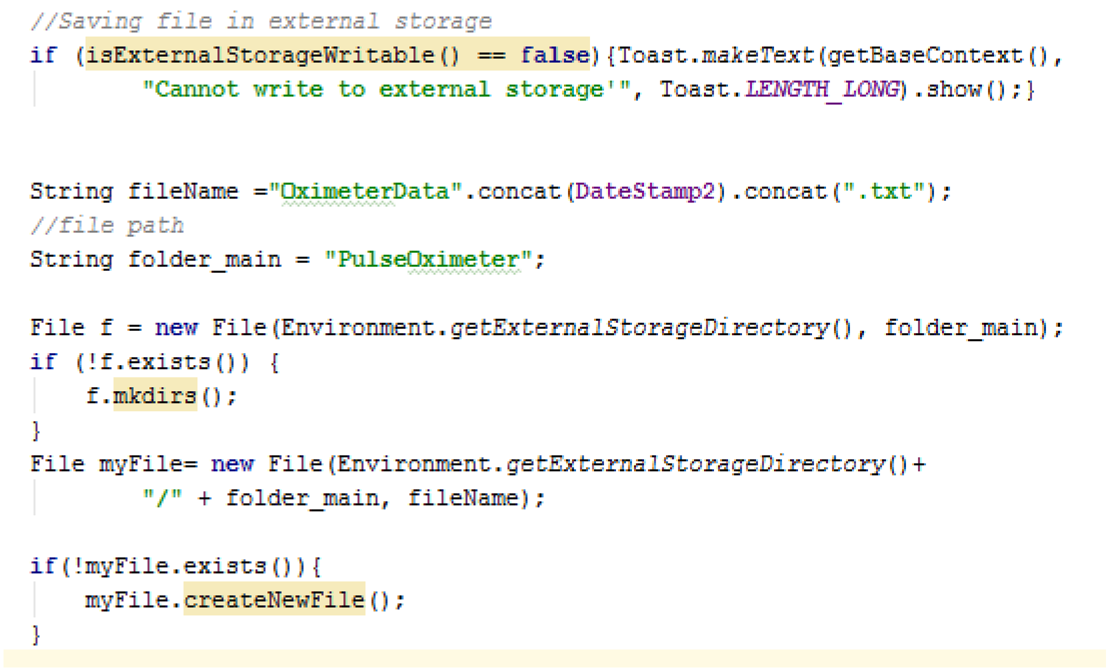


```java
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

If the above getExternalStorageState() returns MEDIA_MOUNTED, then the external storage is available to write. Files can be stored as public files or private files. Private files are treated as files that belong to the app. These files are accessible to the user and other apps but these files are such that they generally do not provide anything of value to the other apps. These files get deleted when the user uninstalls the app. A public file, on the other hand, remain on the system even after app has been uninstalled. These files are available to the user and the other apps. They can include picture and media files, downloaded documents, etc.

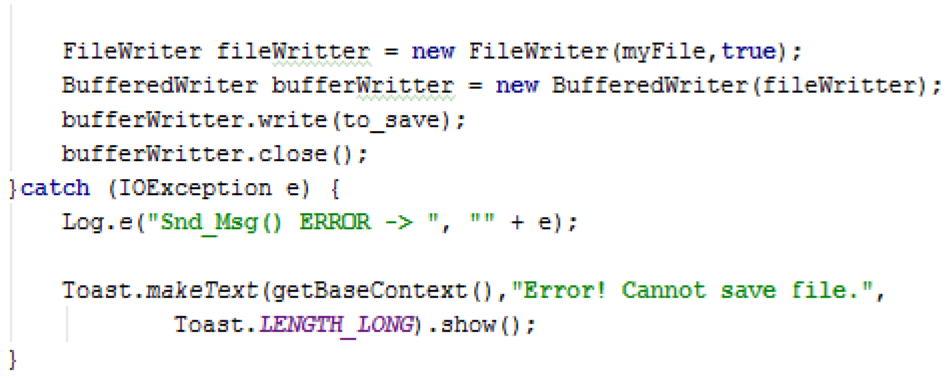
To save as a public file, two methods can be used. GetExternalStoragePublicDirectory() returns the top level public external storage directory for shoving files of a particular type based on the argument passed. The method takes an argument depending on the type of file you want to save. For example, if the app were to save a picture, the argument DIRECTORY_PICTURES would need to be included so that the file can be logically sorted in the appropriate directory. Since this app is not saving pictures, music or videos, it uses the method getexternalStorageDirectory(). This method is also used to save public files, but it returns the root external directory and not a sub-folder within the external storage.

The following few lines of code first checks if the external storage is available to write. It then checks if the folder "PulseOximeter" exists in the root external storage directory. If it does not exist, it creates a folder by that name. It then checks if a .txt file with the name "OximeterData" followed by the current date exists in this directory. If it doesn't, it creates a new file with this name.



```
//Saving file in external storage
if (isExternalStorageWritable() == false){Toast.makeText(getBaseContext(),
        "Cannot write to external storage'", Toast.LENGTH_LONG).show();}


String fileName ="OximeterData".concat(DateStamp2).concat(".txt");
//file path
String folder_main = "PulseOximeter";

File f = new File(Environment.getExternalStorageDirectory(), folder_main);
if (!f.exists()) {
    f.mkdirs();
}
File myFile= new File(Environment.getExternalStorageDirectory()+
        "/" + folder_main, fileName);

if(!myFile.exists()){
    myFile.createNewFile();
}
```

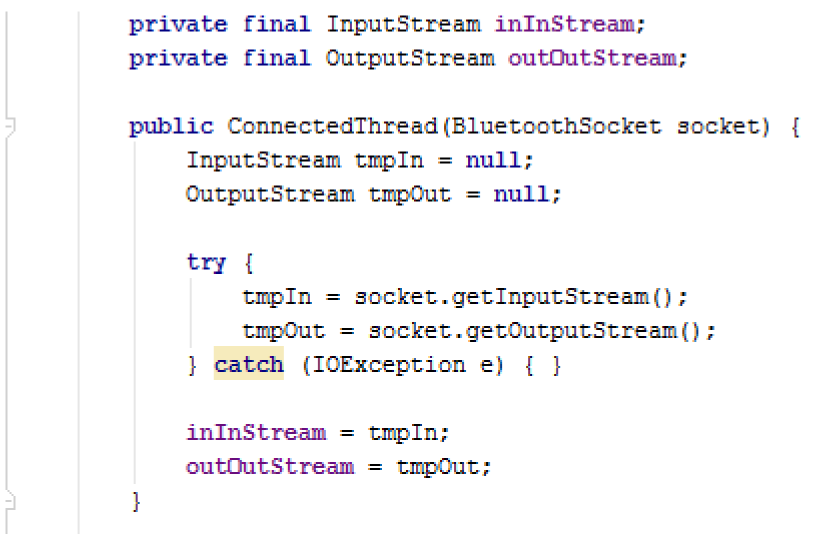
Next, the app must save the data to this file. The FileWriter and BufferedWriter classes is used to write to/append to the file as shown below. "to_save" is the string that is passed to the saved function. The content of this string will be discussed later in this chapter. Finally, if there was any error in saving the data, i.e. if for some reason the path name is incorrect or the folder or file could not be created, the "Error! Cannot save file" message is displayed on the screen.



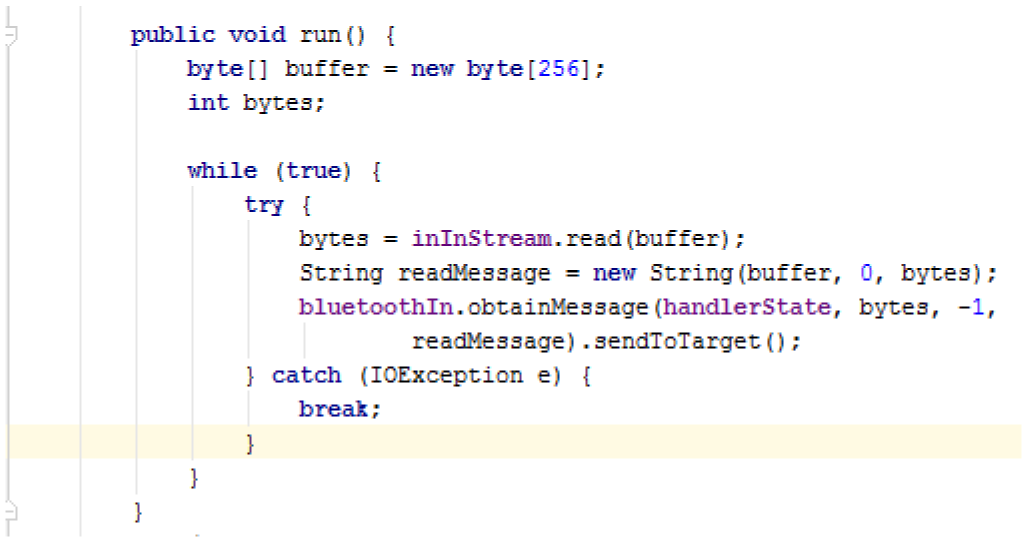
```
    FileWriter fileWritter = new FileWriter(myFile,true);
    BufferedWriter bufferWritter = new BufferedWriter(fileWritter);
    bufferWritter.write(to_save);
    bufferWritter.close();
}catch (IOException e) {
    Log.e("Snd_Msg() ERROR -> ", "" + e);

    Toast.makeText(getBaseContext(),"Error! Cannot save file.",
            Toast.LENGTH_LONG).show();
}
```

4.6. Receiving and sorting data

The BluetoothSocket element represents the Bluetooth socket. This is a point of connection between two Bluetooth devices and allows then exchange data using InputStream and OutputStream. InputStream is used to read data from a source and OutputStream is used to write data to a source. A Handler is used to process the data received. A Handler (bluetoothIn in this code) is associated with a thread, and this collects the data that was received and sends it to the main thread/UI. The obtainMessage returns the message.

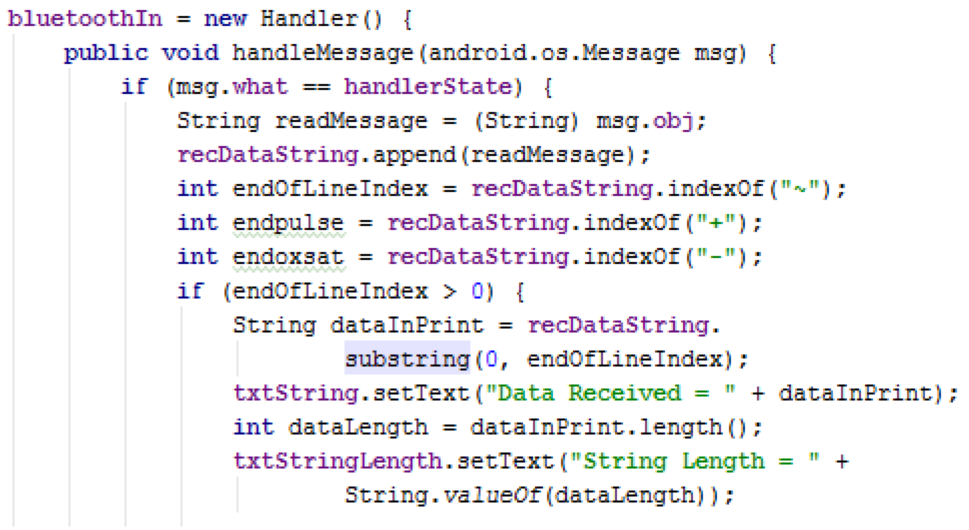


```
private final InputStream inInStream;
private final OutputStream outOutStream;

public ConnectedThread(BluetoothSocket socket) {
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    try {
        tmpIn = socket.getInputStream();
        tmpOut = socket.getOutputStream();
    } catch (IOException e) { }

    inInStream = tmpIn;
    outOutStream = tmpOut;
}
```

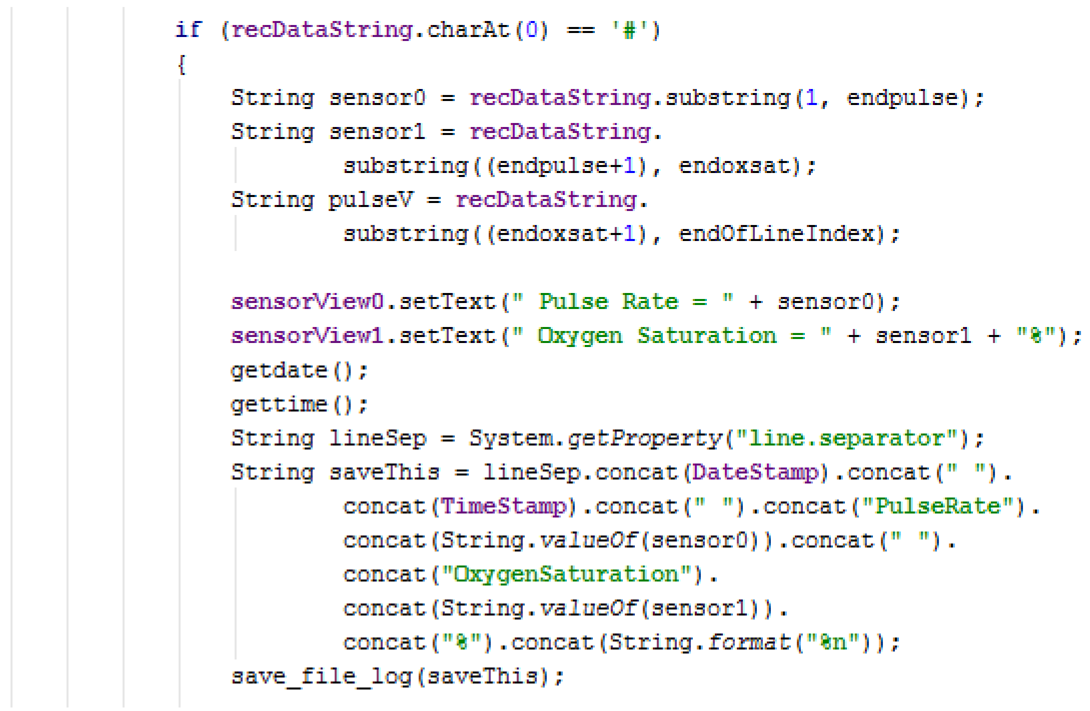


```
public void run() {
    byte[] buffer = new byte[256];
    int bytes;

    while (true) {
        try {
            bytes = inInStream.read(buffer);
            String readMessage = new String(buffer, 0, bytes);
            bluetoothIn.obtainMessage(handlerState, bytes, -1,
                    readMessage).sendToTarget();
        } catch (IOException e) {
            break;
        }
    }
}
```

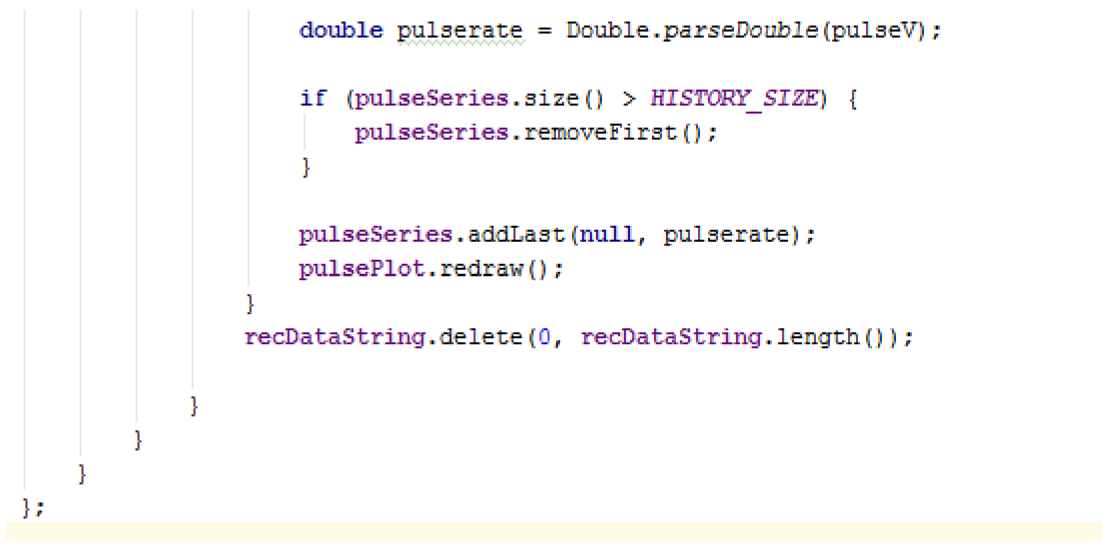
As mentioned in the previous chapter, the data string sent starts with '#', followed by the pulse rate, followed by '+', followed by the oxygen saturation value, followed by '-' and the ac component of the pulse. The string ends with '~'. The symbols '#', '+', '-', and '~' are used to separate out the data and they are then displayed (using setText or pulsePlot) and saved. The string sent to the file save function consists of the date stamp, the time stamp, the pulse rate and the oxygen saturation value. The string ends with a carriage return so that the next set of data gets saved as the next line. To plot the data, the received string is first converted to the data type double. The app then checks if the series has reached its maximum limit. If it has, it deletes the first value. It then appends the latest data received to the end of the series and replots it. Finally, the recDataString field is cleared.



```
bluetoothIn = new Handler() {
    public void handleMessage(android.os.Message msg) {
        if (msg.what == handlerState) {
            String readMessage = (String) msg.obj;
            recDataString.append(readMessage);
            int endOfLineIndex = recDataString.indexOf("~");
            int endpulse = recDataString.indexOf("+");
            int endoxsat = recDataString.indexOf("-");
            if (endOfLineIndex > 0) {
                String dataInPrint = recDataString.
                        substring(0, endOfLineIndex);
                txtString.setText("Data Received = " + dataInPrint);
                int dataLength = dataInPrint.length();
                txtStringLength.setText("String Length = " +
                        String.valueOf(dataLength));
```



```
                if (recDataString.charAt(0) == '#')
                {
                    String sensor0 = recDataString.substring(1, endpulse);
                    String sensor1 = recDataString.
                            substring((endpulse+1), endoxsat);
                    String pulseV = recDataString.
                            substring((endoxsat+1), endOfLineIndex);

                    sensorView0.setText(" Pulse Rate = " + sensor0);
                    sensorView1.setText(" Oxygen Saturation = " + sensor1 + "%");
                    getdate();
                    gettime();
                    String lineSep = System.getProperty("line.separator");
                    String saveThis = lineSep.concat(DateStamp).concat(" ").
                            concat(TimeStamp).concat(" ").concat("PulseRate").
                            concat(String.valueOf(sensor0)).concat(" ").
                            concat("OxygenSaturation").
                            concat(String.valueOf(sensor1)).
                            concat("%").concat(String.format("%n"));
                    save_file_log(saveThis);
```



```
                    double pulserate = Double.parseDouble(pulseV);

                    if (pulseSeries.size() > HISTORY_SIZE) {
                        pulseSeries.removeFirst();
                    }

                    pulseSeries.addLast(null, pulserate);
                    pulsePlot.redraw();
                }
                recDataString.delete(0, recDataString.length());

            }
        }
    }
};
```

Chapter 5

Results and Conclusions

The pulse oximeter was used to measure data continuously in 2-3 minute time spans and the corresponding data measured by a Contec pulse oximeter (model number CMS50D) was recorded simultaneously. Pulse and oxygen saturation measurements were taken in multiple sittings over several days. The measurements were also preformed when the subject was hyperventilating and after vigorous exercise to try to induce a change in the pulse rate and to see how well the device responds to changing pulse rates. Additionally, the oxygen saturation levels were measured as the subject was holding their breath for 35-45 seconds after a forced expulsion. The changing oxygen saturation levels were used to calibrate and test the response of the device.

The AC and DC voltages from the photodiode are measured every 20 milliseconds by the Arduino. To calculate the pulse rate, the Arduino checks for when the AC voltage goes from a point below 0.4 V to a point above 0.4 V. Since the waveform is centered at 0.4 V, every pulse crosses this point twice – once during the rising slope and once during the falling slope. The Arduino notes the time every time the rising slope crosses 0.4 V and uses the time difference between two consecutive points to measure the pulse rate. To calculate the oxygen saturation, the AC peak to peak voltage is measured for both the red and the infrared LEDs by calculating the difference between the maximum and minimum points on the waveform. This AC peak to peak reading, along with the DC measurements for both the LEDs are substituted in equation 5.2 to calculate the R ratio. Both the oxygen saturation data and the pulse rate data displayed by the smartphone (and in the plots below) have been averaged over 6 seconds of measurements.

5.1. Results

Table 5.1 shows the average pulse rate and oxygen saturation measured using both devices, the reference pulse oximeter and the device designed for this project. The table also shows the percentage error for the oxygen saturation which was calculated using equation 5.1. The percentage error was calculated for each data point and the absolute values were then averages to get the values displayed in table 5.1.

$$Percentage\ Error = \frac{Reference\ value - Measured\ value}{Reference\ value} * 100 \tag{5.1}$$

Table 5.1. Accuracy of average pulse rate and oxygen saturation

Reference Pulse Oximeter		Prototype Pulse Oximeter		Percentage Error	
SpO_2 (%)	Pulse Rate (bpm)	SpO_2 (%)	Pulse Rate (bpm)	SpO_2	Pulse Rate
98.67	88.75	99.46	88.19	1.08	2.86

To calibrate the pulse oximeter, the subject held their breath for 35-45 seconds after a forceful expulsion. This was repeated several times and the R (equation 5.2) value for the device was plotted against the oxygen saturation reading from the reference oximeter. The linear fit of this plot was used to calculate the oxygen saturation reported in the following plots. Ideally, to calibrate and test the device, the oxygen saturation levels must be measured over a much wider range (as explained in section 1.3.5), say 70 – 100%, but since neither the breath-holding process not the hyperventilating lowered the oxygen saturation below 95%, the accuracy of the device cannot be tested properly.

$$R = \left(\frac{\left(\frac{AC}{DC}\right)_{RED}}{\left(\frac{AC}{DC}\right)_{IR}} \right) \tag{5.2}$$

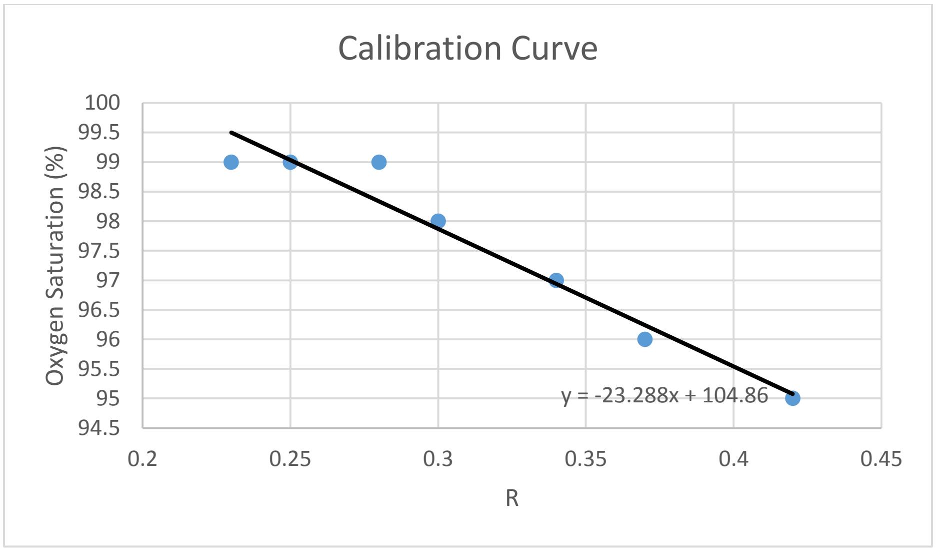


Figure 5.1. Calibration curve

Figure 5.2 shows the comparison of the pulse rates measured by the prototype device and the reference pulse oximeter. As mentioned in table 5.1, the total percentage error for the pulse oximeter measurements is 2.86%. The solid line in the plot is the linear regression line.

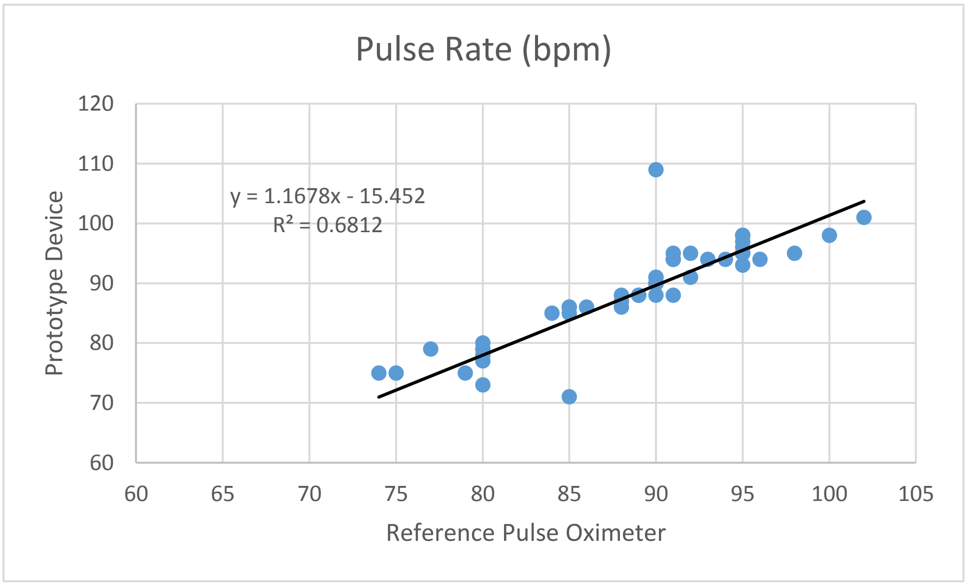


Figure 5.2. Comparison of the pulse rate measured by the reference device and the prototype device

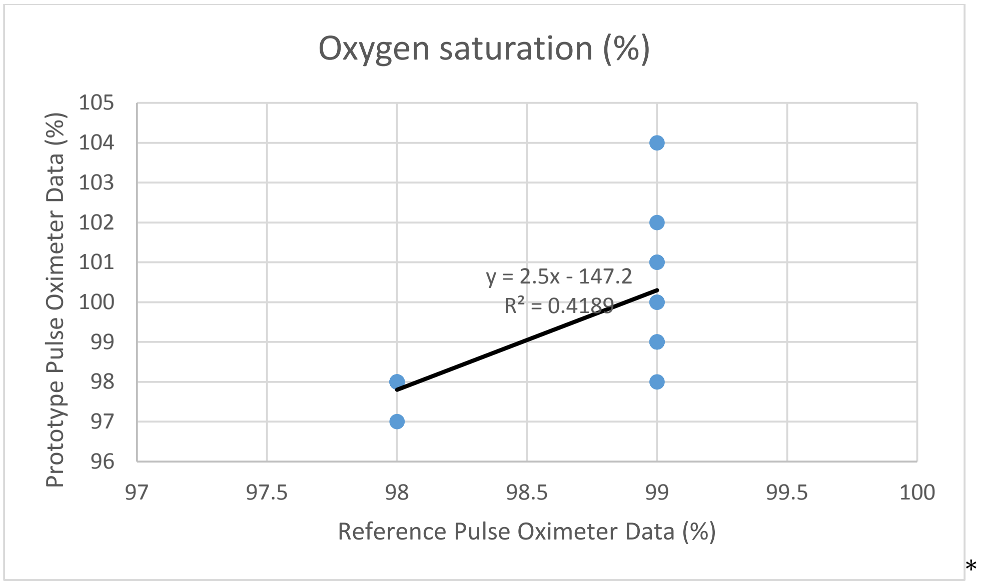


Figure 5.3. Comparison of the oxygen saturation measured by the reference device and the prototype device

Figure 5.3 shows the comparison of the oxygen saturation data measured by the prototype device and the reference pulse oximeter. The solid line represents the linear regression line. The R^2 value is 0.42, indicating that the data measured by the prototype device does not correlate very well with the reference device. However, the resolution of the reference device is single digit, and this could have some impact on the percentage error calculation of the prototype device. A better resolution could possibly give better results.

Figure 5.4 shows the comparison of the pulse rates after exercise. These measurements were made after the subject performed squats and jumping jacks. The measurements were taken over a 3 minute period after the exercise. The R^2 value of the plot is much higher than the previous pulse rate plot. This is most likely because the measurements for figure 5.4 were all made over a short time period and the dataset is much smaller, while the dataset plotted in figure 5.2 is much larger and was measured over several days. The pulse rate ranges from 98 to 132 beats per second.

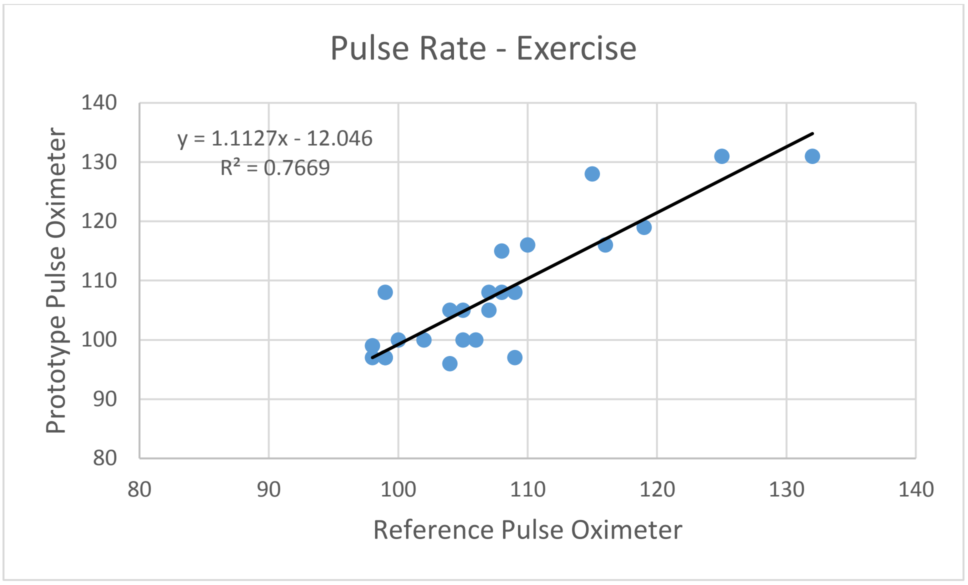


Figure 5.4. Comparison of the pulse rate measured by the reference device and the prototype device after exercise.

Figure 5.5 shows the comparison of the pulse rates during hyperventilation. As the measurements were started, the subject was to hyperventilate by breathing into a paper bag. The device is quite sensitive to motion and since the subject was attempting to hyperventilate during the measurement, the correlation between the measured value and the reference value is lower than expected even though this is a small dataset and was also measured over a short interval of time.

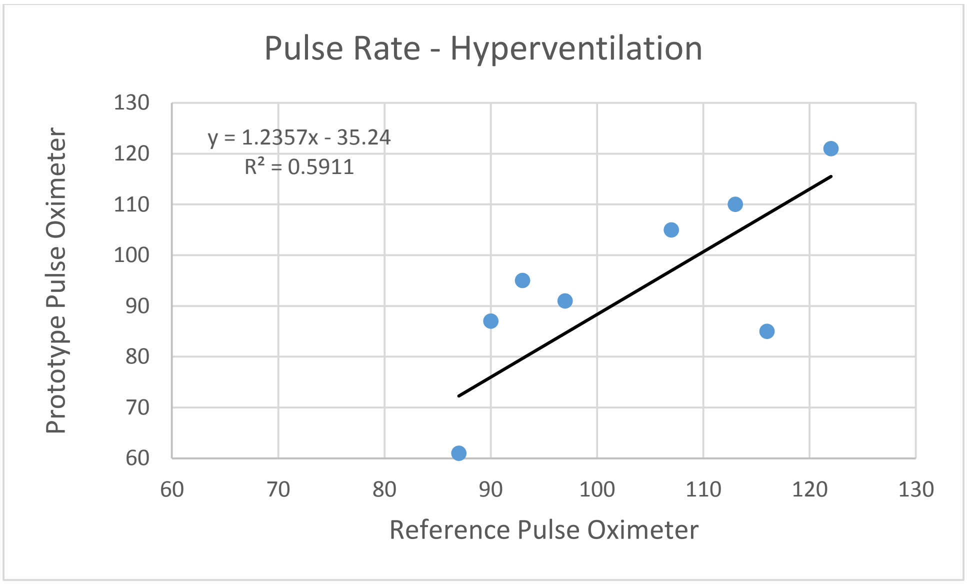
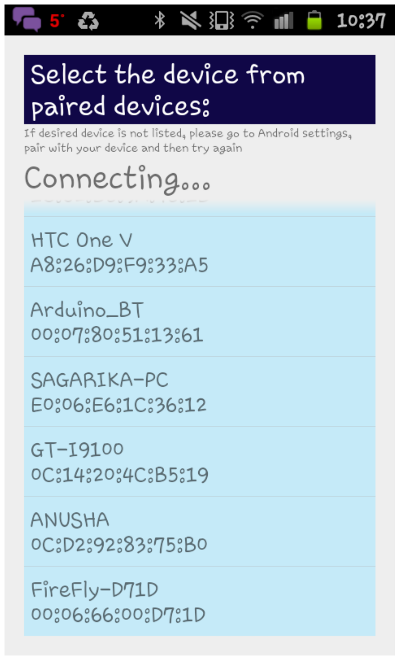


Figure 5.5. Comparison of the pulse rate measured by the reference device and the prototype device for hyperventilation test

Figure 5.6 shows the first page of the Android application after the Bluetooth has been turned on and the required device has been selected. If the connection was not successful, it goes displays a "Connection failure" notification at the bottom of the screen. If the connection was successful, it starts to display the data it receives as shown in figure 5.7.


10:37

Select the device from paired devices:

If desired device is not listed, please go to Android settings, pair with your device and then try again

Connecting...

HTC One V
A8:26:D9:F9:33:A5

Arduino_BT
00:07:80:51:13:61

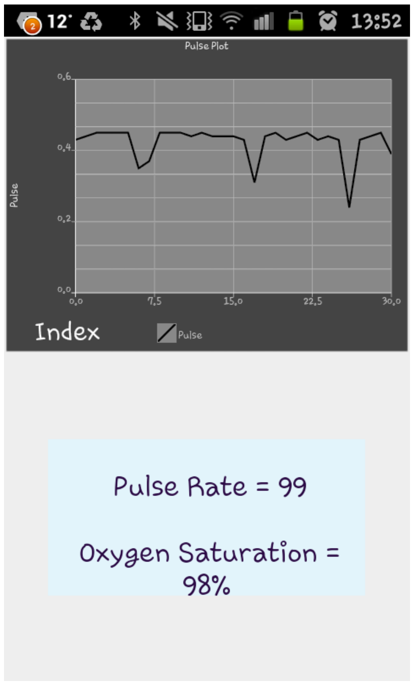
SAGARIKA-PC
E0:06:E6:1C:36:12

GT-I9100
0C:14:20:4C:B5:19

ANUSHA
0C:D2:92:83:75:B0

FireFly-D71D
00:06:66:00:D7:1D

Figure 5.6. List of paired devices on the Android application



Pulse Rate = 99

Oxygen Saturation = 98%

Figure 5.7. Data display screen on the Android application

A pulse oximeter is very sensitive to motion and there are several factors that can influence the performance of the device. For a pulse oximeter that attaches at the finger, the size of the finger, the skin and the presence or absence or scar tissue are some of the factors at effect the signal received by the photodiode. Additionally, if the sensor is attached with too much or too little pressure, the strength of the signal received changes and if the strength of the signal changes constantly, both the pulse rate and the oxygen saturation calculations will be effected. If the sensor is clamped on too tight, this could even cut off the circulation at the finger, so both the sensor placement and the pressure are very important.

5.2. Conclusion and future work

To summarize, the aim of the project was to make a wireless platform and to develop a pulse oximeter to test the wireless platform. A pulse oximeter was designed and built to measure the pulse rate and oxygen saturation at the fingertip. The design consisted of a modified clothespin with a red and an infrared LED on one side and a photodiode on the other, to make a transmission-mode pulse oximeter. An electronic circuit was designed to convert the photocurrent to a voltage, and then filter and amplify this voltage to get the required pulsatile signal. This signal was then processed using an Arduino Uno microcontroller and then sent to a Bluetooth module to be transmitted to a smartphone. An Android application was written to turn on the Bluetooth on the smartphone and connect to a paired device. The app then receives the data transmitted by the Bluetooth module attached to the Arduino and displays, plots and saves the data in a text file.

The pulse rate and the oxygen saturation were calculated in the microcontroller. Since the pulse waveform was centered at 0.4 V, the points at which the rising slope of the waveform crossed 0.4 V was used to calculate the pulse rate. Equation 3.1 was used to calculate the oxygen saturation with A as 104.86 and B as -23.288. The constants A and B were obtained from the calibration curve from figure 5.1.

The average percentage error for the pulse rate is $\pm 2.86\%$ and for the oxygen saturation, the average percentage error is $\pm 1.08\%$. The required accuracy was $\pm$ 3 for both the pulse rate as well as the oxygen saturation but this, unfortunately, was not achieved. While the average error percentage is below 3%, not all the data points measured by the prototype device are within 3 bpm or 3% of the data points measured by the reference device. Hence, the device does not have an accuracy of 3 bpm or 3% for pulse rate and oxygen saturation respectively. The device still needs some improvement to improve the accuracy of both the pulse rate and the oxygen saturation. To correctly calibrate and test the device, both the pulse rate and the SpO_2 would need to change over a much wider range, such a 40 - 120 bpm and 70 - 100%. Since neither the breath-holding process not the hyperventilating lowered the oxygen saturation below 95%, with most of the data points at 98% or 99%, the accuracy of the device cannot be tested properly.

While the hardware and software components of this project have achieved the goals of measuring the pulse rate and oxygen saturation and wirelessly transmitting and displaying this data on a smartphone, there are several improvements that could be made to bring the device up to industry standards. The sensor is mounted on a modified clothespin, making the sensor uncomfortably tight for people with bigger fingers. It also starts to cut off the blood flow when clipped on for several minutes, making it impossible to measure data continuously over a large time interval. The sensor can be redesigned to be more comfortable for people of different sizes and so that it can be used to continuously monitor a subject for any duration of time. It can also be redesigned to be more aesthetically pleasing.

While the sensor itself (the LEDs and the photodiode) was mounted on the clothespin, the electronics used to convert, filter, and amplify the current was implemented on a breadboard since it was much easier make adjustments as required. However, integrating the sensor, the electronics and the microcontroller onto a PCB would make the device more portable, easier to use, and possible more reliable since this would eliminate a small fraction of noise and stray signals. Additionally, this would allow the use of smaller components and minimize the size of the entire design. The calibration process for the oxygen saturation curve must also be improved to get a wider range for oxygen saturation.

The Android application can also be improved upon to run and save data even when the app is not open if the subject desires. Currently, the application pauses if the display of the phone is turned off and it terminates the connection if the app is closed. The current program cannot run as a background application. It can also be set up to display a warning if the pulse rate or the oxygen saturation values are constantly outside a desired range. Additionally, applications can be written to allow smartphones working on Apple or Windows operating systems to perform the same function.

Appendix A

Data Sets

Table A.1. Pulse rate data

Pulse Rate		Pulse Rate	
Reference Device	Prototype Device	Reference Device	Prototype Device
98	95	90	91
79	75	88	88
74	75	88	87
75	75	88	86
77	79	90	90
91	94	90	91
93	94	91	94
94	94	102	101
96	94	95	96
95	97	95	95
89	88	92	95
90	88	92	91
91	88	90	90
84	85	85	86
91	95	80	77
82	56	80	73
85	71	86	86
85	85	80	80
89	88	80	79
90	90	80	78

90	90	95	95
95	98	95	93
95	95	95	98
90	109	85	86
100	98	90	90
80	79	95	95

Table A.2. Oxygen Saturation Data

Reference Device	Prototype Device
99	101
99	100
99	100
98	98
98	98
98	97
98	98
98	98
99	98
99	104
99	101
99	102
99	99
99	99
99	99

Table A.3. Pulse Rate Data after Exercise

Pulse Rate		Pulse Rate	
Reference Device	Prototype Device	Reference Device	Prototype Device
132	131	109	108
125	131	108	115
119	119	105	105
116	116	104	105
115	128	107	105
110	116	102	100
105	100	100	100
104	96	99	97
106	100	98	97
107	108	99	108
108	108	99	97
109	97	98	99

Table A.4. Pulse Rate Data after Hyperventilation

Pulse Rate	
Reference Device	Prototype Device
122	121
116	85
113	110
107	105
97	91

93	95
90	87
88	56
87	61

References

[1]J. Allen, 'Photoplethysmography and its application in clinical physiological measurement',Physiol. Meas., vol. 28, no. 3, pp. R1-R39, 2007.

[2]A. Alian and K. Shelley, 'Photoplethysmography', Best Practice & Research Clinical Anaesthesiology, vol. 28, no. 4, pp. 395-406, 2014.

[3]J. Moyle, *Pulse Oximetry*, 2nd ed. London: BMJ Books, 2002, pp. 1-35.

[4]J. Severinghaus and P. Astrup, 'History of blood gas analysis. VI. Oximetry', *Journal of Clinical Monitoring*, vol. 2, no. 4, pp. 270-288, 1986.

[5]E. Eger, L. Saidman and R. Westhorpe, *The wondrous story of anesthesia*. .

[6]Y. Pole, 'Evolution of the pulse oximeter', *International Congress Series*, vol. 1242, pp. 137-144, 2002.

[7]N. Townsend, ‘Pulse Oximetry’, *Medical Electronics*, Michaelmas term, pp. 32-36, 2001

[8]J. Severinghaus and Y. Honda, 'History of blood gas analysis. VII. Pulse Oximetry', *Journal* of Clinical Monitoring, vol. 3, pp. 135-138, 1987.

[9]J. Goldman, M. Petterson, R. Kopotic and S. Barker, 'Masimo Signal Extraction Technology Pulse Oximetry', *Journal of Clinical Monitoring and Computing*, vol. 16, no. 7, pp. 475-483, 2000.

[10]T. Aoyagi, 'Pulse oximetry: its invention, theory, and future', *Journal of Anesthesia*, vol. 17, no. 4, pp. 259-266, 2003.

[11]J. Webster, *Design of pulse oximeters*. Bristol: Institute of Physics Pub., 1997.

[12]Biologyonline.us, 'Pulmonary Ventilation', 2015. [Online]. Available: http://biologyonline.us/Online%20A&P/AP%201/Northland/AP1lab/Lab%2011/73.htm

[13]'How does the blood circulatory system work?', Institute for Quality and Efficiency in Health Care (IQWiG), 2012.

[14] Wikipedia, 'Lung', 2015. [Online]. Available: https://en.wikipedia.org/wiki/Lung.

[15]Breath2000.org, 'Breathing', 2015. [Online]. Available: http://www.breath2000.org/physiology.html.

[16]Konica Minolta Sensing, Inc., 'How to read SpO2', 2006.

[17]Who.int, 'WHO | Pulse oximetry: training manual', 2015. [Online]. Available: http://www.who.int/patientsafety/safesurgery/pulse_oximetry/tr_material/en/.

[18]T. Tamura, Y. Maeda, M. Sekine and M. Yoshida, 'Wearable Photoplethysmographic Sensors—Past and Present', Electronics, vol. 3, no. 2, pp. 282-302, 2014.

[19]N. Stubán and N. Masatsugu, 'Non-invasive calibration method for pulse oximeters',Periodica Polytechnica Electrical Engineering, vol. 52, no. 1-2, p. 91, 2008.

[20]S. Duun, R. Haahr, K. Birkelund and E. Thomsen, 'A Ring-Shaped Photodiode Designed for Use in a Reflectance Pulse Oximetry Sensor in Wireless Health Monitoring Applications', IEEE Sensors J., vol. 10, no. 2, pp. 261-268, 2010.

[21]Arduino.cc, 'Arduino - ArduinoBoardUno', 2015. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno.

[22]Bluetooth.com, 'How it Works | Bluetooth Technology Website', 2015. [Online]. Available: http://www.bluetooth.com/Pages/How-It-Works.aspx.

[23]Learn.sparkfun.com, 'Using the BlueSMiRF - learn.sparkfun.com', 2015. [Online]. Available: https://learn.sparkfun.com/tutorials/using-the-bluesmirf.

[24]Arduino.cc, 'Arduino - Reference', 2015. [Online]. Available: https://www.arduino.cc/en/Reference/HomePage.

[25]Developer.android.com, 'Android Developers', 2015. [Online]. Available: http://developer.android.com/.

[26]'Androidplot', 2015. [Online]. Available: http://androidplot.com/.