# Chapter 1

# Introduction

## 1.1 Introduction

Monitoring our body's vital signs, such as our heart rate and oxygen levels, is crucial for detecting health problems early on. However, traditional monitoring systems can be challenging to use and expensive, often requiring frequent visits to healthcare providers. This can be difficult for people who live far away or have trouble getting around.

To address this problem, we have developed a smart health monitoring system that uses the power of the internet to monitor vital signs in real-time, no matter where we are! Our system utilizes sensors to collect information about vital signs like heart rate and oxygen levels, and then sends that data to an app that healthcare providers can use to monitor our health. They can check our information at any time, even if they are not with us!

In today's world, where people live in different parts of the world and travel frequently, it is essential to have a healthcare system that is accessible and easy to use. Traditional healthcare monitoring systems are often inefficient and expensive, requiring frequent in-person visits to healthcare providers for monitoring and data collection. This approach makes healthcare services less accessible, particularly for those living in rural areas or with limited mobility. The lack of access to timely interventions can result in serious health problems, leading to additional healthcare costs and complications.

Therefore, we have incorporated machine learning into our smart health monitoring system, enabling early detection and intervention for potential health problems. By providing real-time monitoring of body vitals and surrounding parameters, our system facilitates remote viewing and improves decision-making by

healthcare providers.

Overall, this paper showcases how we can leverage special technology, like machine learning and IoT, to revolutionize healthcare, making it more accessible, efficient, and cost-effective for everyone. With our smart health monitoring system, we can catch health problems early and get the help we need to stay healthy, no matter where we are!

## 1.2 Motivation

This project is motivated by the need for a practical and cost-effective remote health monitoring solution that can provide real-time monitoring of patients' health conditions, enabling early interventions and reducing the need for in-person visits. With the growing need for healthcare services, it is important to have a system that is accessible and easy to use, especially for those living in rural areas or with limited mobility. Our project aims to address these challenges by developing a smart health monitoring system based on IoT technology, which provides real-time monitoring of body vitals. By leveraging machine learning techniques, we can provide valuable insights and early detection of health problems, ultimately improving patient outcomes and reducing healthcare costs.

## 1.3 Problem Statement

To address the issue of detecting heart problems early on and providing timely interventions, this project aims to develop a smart health monitoring system using the ESP8266 and MAX3010X sensors. The goal is to create an accessible and cost-effective solution that can monitor vital signs in real-time and classify individuals as having heart problems or not using a logistic regression algorithm. By doing so, individuals can monitor their health remotely and healthcare professionals can receive valuable insights for better patient care. This solution can be especially beneficial for individuals who have limited access to healthcare services or live in remote areas.

## 1.4 Structure of the Thesis

This thesis is organised as follows.

- Chapter 2 discusses previous works and researches in this particular topic.

- Chapter 3 discusses the System Architecture with the components that are used in this project.
- Chapter 4 discusses the internal function of MAX3010X sensor.
- Chapter 5 discusses the methodology used in this project in great details.
- Chapter 6 focuses on the circuit connection of hardware components and its integration with Blynk platform.
- Chapter 7 focuses on observation of pulse rate and SpO2 from MAX30100 sensor.
- Chapter 8 gives us the insight about machine learning model i.e. Logistic Regression Model and the observations we got by applying the model to the data set containing sensor readings.
- Chapter 9 presents the conclusion and future improvements in the mentioned project.

# Chapter 2

# Literature Review

In a study conducted by Pasha [1], a Thingspeak based sensing and monitoring system was designed for IoT with Matlab analysis. The system utilized an Arduino Uno board with its wifi module ESP8266 and room monitoring sen- sors. Programming was done using Arduino IDE. The use of Thingspeak platform was found to greatly enhance data analysis, enabling observation of data over a period of time. However, the platform lacked triggering action. While this system was designed for patient room monitoring, it can be extended to a full-fledged patient monitoring system by incorporating bio-sensors.

In the article "Development of Smart Healthcare Monitoring System in IoT Environment" by M.M. Islam, A. Rahaman, and Islam [2], the authors present a smart healthcare monitoring system that uses ESP32 as the microcontroller along with heart beat and body temperature sensors. The system also integrates room monitoring sensors like DHT11 (humidity sensor) and MQ 135 (CO sensor). The system employs Thingspeak plat- form for data analysis, and the error between the observed and actual data was found to be less than 5 percent. However, this system is designed only for medical staff to observe patient data. The authors propose the development of a system where the IoT platform can trigger an action, such as sending a message or email, if the sensor data exceeds a threshold value.

Ganesh [3] proposes a health monitoring system using Raspberry Pi and IoT. The system monitors health parameters such as pulse rate, blood pressure, and heart rate using sensors. The Pi camera module is also utilized. The system displays and stores the data on the cloud for easy access via an IP address. However, the system only alerts for blood pressure measurement, and it requires using an IP address each time. The proposed solution is to develop an application that alerts for every sensor and eliminates the need for the IP address.

A health monitoring system for elderly people using IoT technology is proposed by Tham et al. [4]. The system uses a MAX30100 sensor for measuring heart rate and Peripheral Capillary Oxygen Saturation Level (SpO2) and Node MCU (ESP8266) as a microcontroller to collect and transfer the data to the cloud. The data collected from the healthy subjects underwent validation through segmentation and filtering. The SpO2 data is computed to IR/RED variables and processed to get SpO2 ratio using empirically derived calibration curves to produce normal and abnormal results. The heart rate data underwent a correlation test between the experimental reading with the reference reading. The developed system showed a strong correlation value (rs=0.993) and the percentage error resulted in less than 3 percentage and 1.03 percentage for SpO2 and heart rate, respectively. The validation results showed that the monitoring system for SpO2 and heart rate using IoT is ready to be used and allows many authenticated users to monitor the patient's condition.

S. Chavda and co-authors [5] proposed a mobile application that employs machine learning algorithms and human body parameters to predict the risk of heart disease. The authors emphasize the importance of early detection in preventing adverse consequences and suggest that unhealthy lifestyle habits increase the likelihood of heart disease. Furthermore, the authors suggest that incorporating real patient data and developing an intelligent system for treatment selection could improve the accuracy of the model. The use of Electrocardiogram parameters can also enhance early detection accuracy, thereby improving the overall prognosis for individuals with heart disease.

# Chapter 3

# System Architecture

## 3.1 How an IoT System Actually Works

The process of collecting data from the environment involves the use of sensors or devices that can capture various types of information, ranging from simple temperature readings to complex video feeds. These sensors can be part of a device that serves other purposes besides sensing, or they can be bundled together to collect multiple types of data.

To transmit the collected data to the cloud, sensors/devices can utilize different methods, including cellular, satellite, WiFi, Bluetooth, LPWAN, connecting through a gateway/router, or directly to the internet via ethernet. Once the data reaches the cloud, it undergoes some form of processing by software. This can be a simple check to ensure the data is within acceptable parameters, or a more complex process, such as using computer vision to identify objects in video footage.
.


Figure 3.1: Four Stages of IoT

After processing, the data is made useful to the end-user through various means. This can involve alerts through email, text, or notification, providing the user with

relevant information, such as a temperature reading that exceeds an acceptable range. Alternatively, users may have access to an interface that allows them to proactively monitor the system. For example, they may be able to view video feeds from multiple locations using a phone app or web browser.

## 3.2 Hardware Components

### 3.2.1 ESP8266

.The ESP8266 is a low-cost Wi-Fi enabled microcontroller that is widely used in IoT applications. It features a single-core 32-bit CPU, a limited number of GPIOs, and no touch sensor, but it offers a compact form factor and low power consumption. The ESP8266 also supports a variety of communication protocols, including SPI, I2C, and UART. Despite its limited hardware capabilities compared to the ESP32, the ESP8266 is still a popular choice for many IoT projects due to its low cost and ease of use.


Figure 3.2: Hardware Components

### 3.2.2 MAX3010X

The MAX30100 is a high-sensitivity optical sensor module designed for heart rate and pulse oximetry (SpO2) monitoring. It uses an LED light source and a photodetector to measure the amount of light absorbed by the blood, allowing it to detect heart rate and oxygen saturation levels in real-time. The MAX30100 is commonly used in wearable health monitoring devices and medical equipment for non-invasive heart rate and SpO2 monitoring.

### 3.2.3 OLED

OLED stands for Organic Light Emitting Diode. It's a display technology that creates light when electricity is applied to organic compounds. OLED displays can be easily connected to an micro controller board and used to display text and graphics.

## 3.3 Software Components

### 3.3.1 Arduino IDE

Arduino IDE is an open-source Integrated Development Environment (IDE) used to develop and program code for Arduino boards. It is a simple and easy-to-use interface that provides a platform for creating and uploading sketches to the Arduino board. The software comes with a set of pre-written libraries and functions, making it easy to program microcontrollers with just a few lines of code. Arduino IDE also supports multiple programming languages, including C and C++, making it a flexible and accessible tool for developers of all levels.

### 3.3.2 Coolterm

Coolterm is a serial port terminal application that enables communication with hardware devices using the serial port of a computer. It is a cross-platform program that supports Windows, macOS, and Linux, making it accessible to a wide range of users. Coolterm allows users to send and receive data in a variety of formats, including ASCII, hexadecimal, and binary. It also provides advanced features such as scripting and automation, making it a useful tool for both hobbyists and professionals working with serial communication.

### 3.3.3 Blynk

Blynk is a mobile application platform that allows developers to create and control Internet of Things (IoT) devices through a simple and intuitive interface. It provides a range of pre-built widgets and functions, making it easy to create custom interfaces for controlling hardware devices. Blynk also offers cloud-based services that enable remote monitoring and control of IoT devices, making it a powerful tool for home automation and other IoT applications. Blynk is compatible with a range of hardware platforms, including Arduino, Raspberry Pi, and ESP8266, making it a flexible and versatile platform for IoT development.

# Chapter 4

# Working Principle of MAX3010X

## 4.1 Transmissive and Reflective Measurement

The MAX30100 is a pulse oximeter and heart-rate sensor that can be used in a variety of medical and fitness applications. It is available in two types: the transmissive type and the reflective type.

The transmissive type uses a light source to transmit light through the patient's finger or earlobe to a photodetector on the other side. The light source is typically an LED that emits light at two different wavelengths, usually red and infrared. The photodetector detects the amount of light that has passed through the tissue and calculates the oxygen saturation and heart rate based on the changes in the amount of light that is absorbed by the blood.

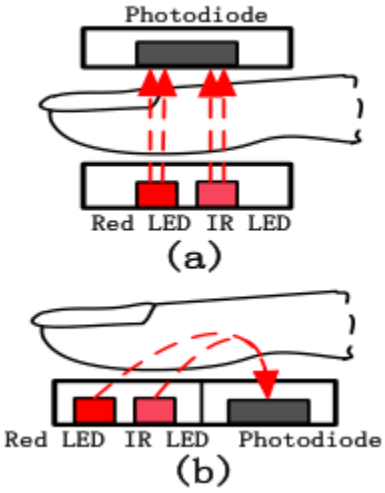


Figure 4.1: Transmissive and Reflective Measurement

The reflective type, on the other hand, uses a light source and a photodetector

on the same side of the patient's finger or earlobe. The light source illuminates the tissue and the photodetector detects the amount of light that is reflected back from the tissue. The oxygen saturation and heart rate are calculated based on the changes in the amount of reflected light that is detected.

In summary, the transmissive type and reflective type MAX30100 sensors are both used for non-invasive measurement of oxygen saturation and heart rate. The choice of which type to use depends on the specific application and the tradeoffs between accuracy and other factors such as cost, power consumption, and ease of use.

In this particular project, the MAX30100 we are using is reflective type measurement sensor.

## 4.2 Reflective Type MAX30100

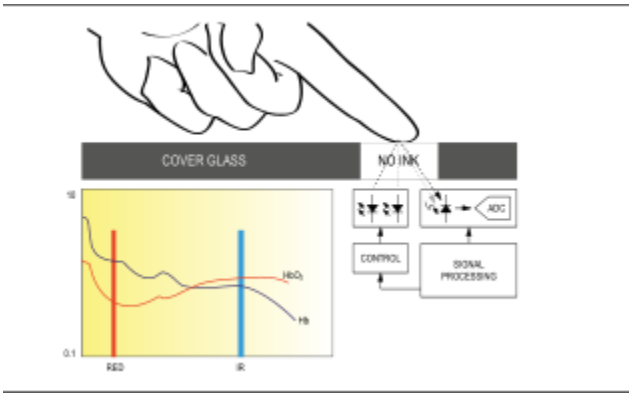


Figure 4.2: MAX30100 Reflective Type

The MAX30100 is a pulse oximeter and heart-rate sensor that consists of several components, including a red LED, an IR LED, a photodiode, an ADC, a signal processing box, and a control box. Here is a brief explanation of the working of each component:

Red LED: The red LED emits light at a wavelength of around 660 nanometers, which is absorbed by oxygenated hemoglobin in the blood. By measuring the amount of light that is absorbed by the blood, the sensor can determine the oxygen saturation levels.

IR LED: The IR LED emits light at a wavelength of around 940 nanometers, which is absorbed by deoxygenated hemoglobin in the blood. By measuring the amount of light that is absorbed by the blood, the sensor can determine the oxygen saturation levels.

Photodiode: The photodiode detects the amount of light that has passed through the tissue or has been reflected back from the tissue. The amount of light that is detected by the photodiode is converted into an electrical signal that can be processed by the ADC.

ADC: The ADC (Analog-to-Digital Converter) converts the analog signal from the photodiode into a digital signal that can be processed by the signal processing box.

Signal processing box: The signal processing box processes the digital signal from the ADC to extract the heart rate and oxygen saturation values. This involves filtering the signal to remove noise and interference, detecting peaks in the signal to identify the heart rate, and using the amplitude and phase differences between the red and IR signals to determine the oxygen saturation.

Control box: The control box manages the operation of the sensor and communicates with the microcontroller or other device that is connected to the sensor. It provides control signals to the LED and photodiode, receives data from the signal processing box, and communicates the results to the external device.

$$SpO2 = \frac{HbO2}{Hb + HbO2} \times 100 \tag{4.1}$$

In this equation, SpO2 represents the peripheral oxygen saturation, HbO2 represents the concentration of oxygenated hemoglobin, and Hb represents the concentration of deoxygenated hemoglobin. The formula calculates the SpO2 value by dividing the concentration of HbO2 by the total concentration of hemoglobin (Hb + HbO2), and then multiplying the result by 100 percentage to obtain a percentage value.

# Chapter 5

# Methodology

## 5.1 A Block Diagram Breakdown

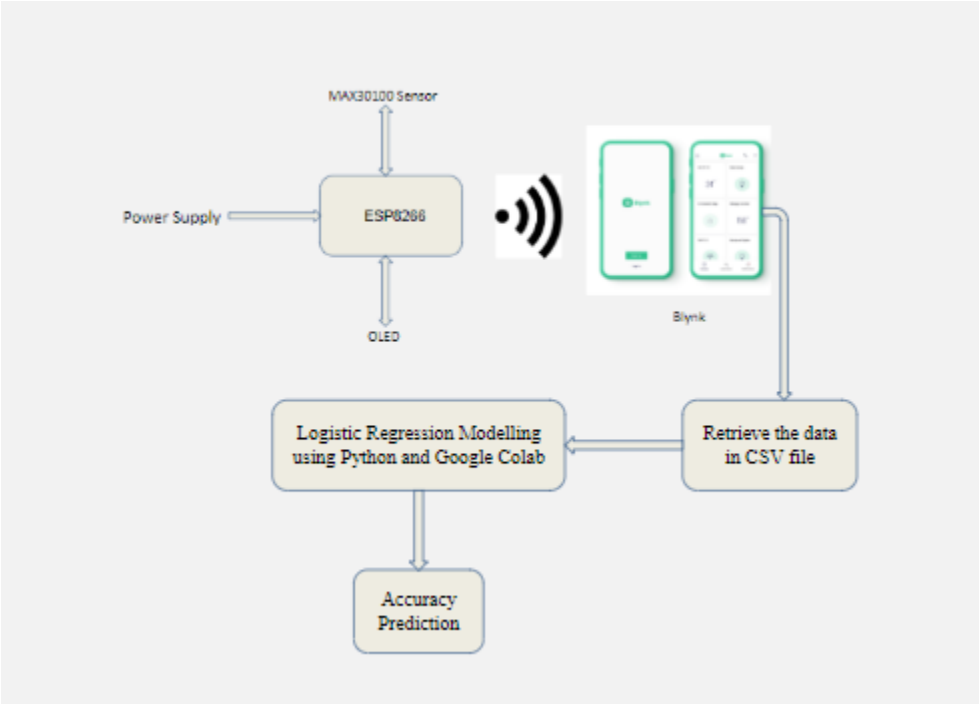


Figure 5.1: Sequence of the whole project

## 5.2 MAX3010X Internal Function

The software design of a system is crucial, and minimizing power consumption is an important consideration. Upon power-up, the software checks a flag to determine the system state. If no measurement is detected, the system enters an ultra-low power sleep mode that consumes very little power. If a measurement is detected, the system collects and processes PPG signals to calculate SpO2 and Pulse Rate

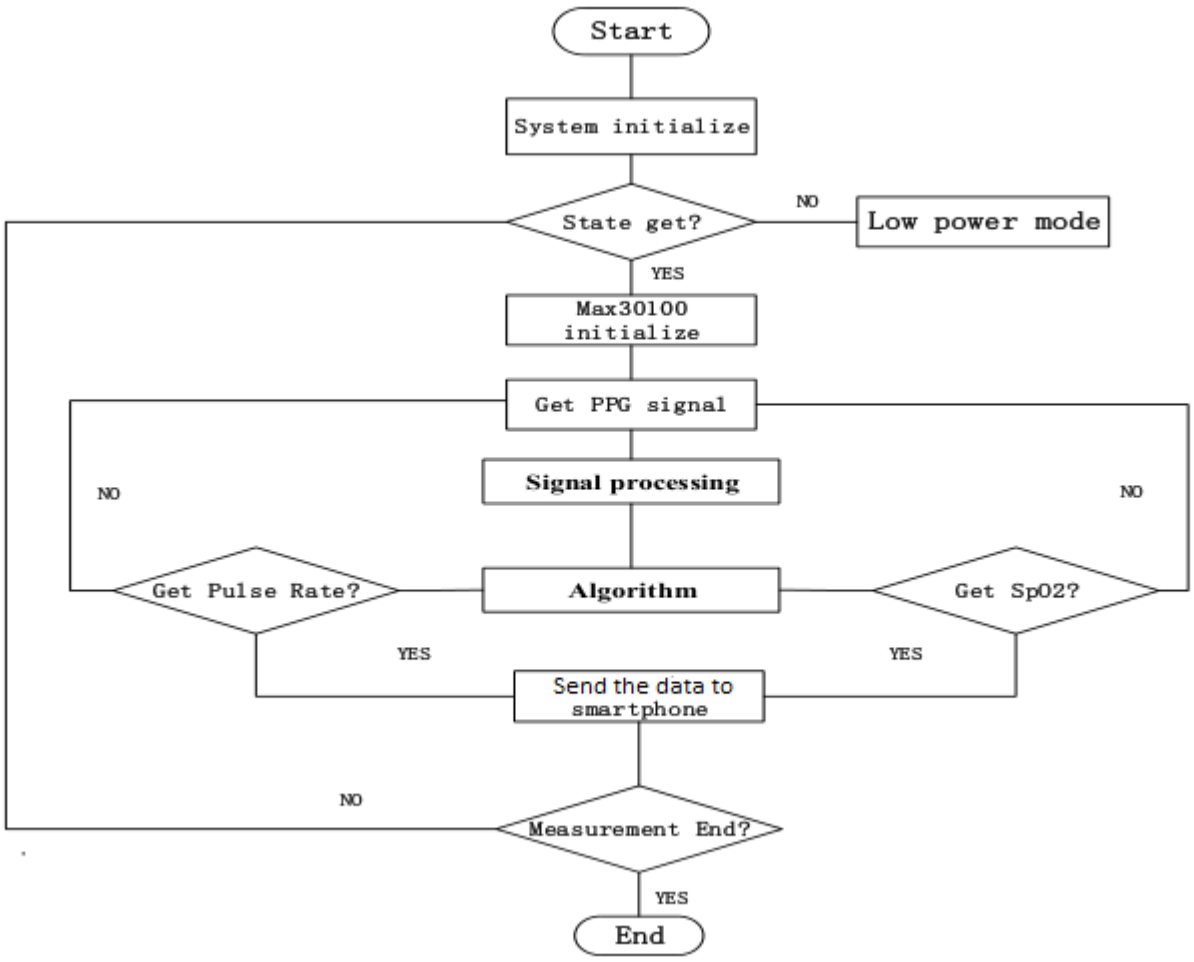


Figure 5.2: Sequence of internal function of MAX3010X

values. Finally, the system sends the data to a smartphone app using the wifi module of ESP8266. The program flow diagram is depicted in Figure 5.1.

## 5.3 Interaction Between ESP8266, MAX30100, BLYNK

The ESP8266 microcontroller can also be used to communicate with the MAX30100 sensor and Blynk smartphone application. The MAX30100 is connected to the ESP8266 through the I2C interface, allowing the ESP8266 to read the PPG signal from the sensor and control its LED and ADC functions. The ESP8266's microcontroller then processes the sensor data and calculates the SpO2 and Pulse Rate values using a specific algorithm.

Blynk can communicate with the ESP8266 through Wi-Fi or Bluetooth, providing a user interface for the system and displaying the SpO2 and Pulse Rate values in real-time. The ESP8266 sends the data to Blynk using the Blynk API, which can

be implemented using the Arduino IDE or other programming environments.

The communication between the ESP8266, MAX30100, and Blynk is achieved through programming. The ESP8266 is programmed to read data from the MAX30100 sensor, process it, and send it to Blynk. Blynk is programmed to receive the data from the ESP8266 and display it on the user interface. The programming languages used for this task can be Arduino C++, Python, or other languages supported by the ESP8266 and Blynk platforms.

Overall, the communication between the ESP8266, MAX30100, and Blynk is established by leveraging the hardware and software capabilities of each component and programming them to work together seamlessly to achieve the desired functionality.

# Chapter 6

# Circuit Connection and Blynk Integration

## 6.1 Basic Block Diagram

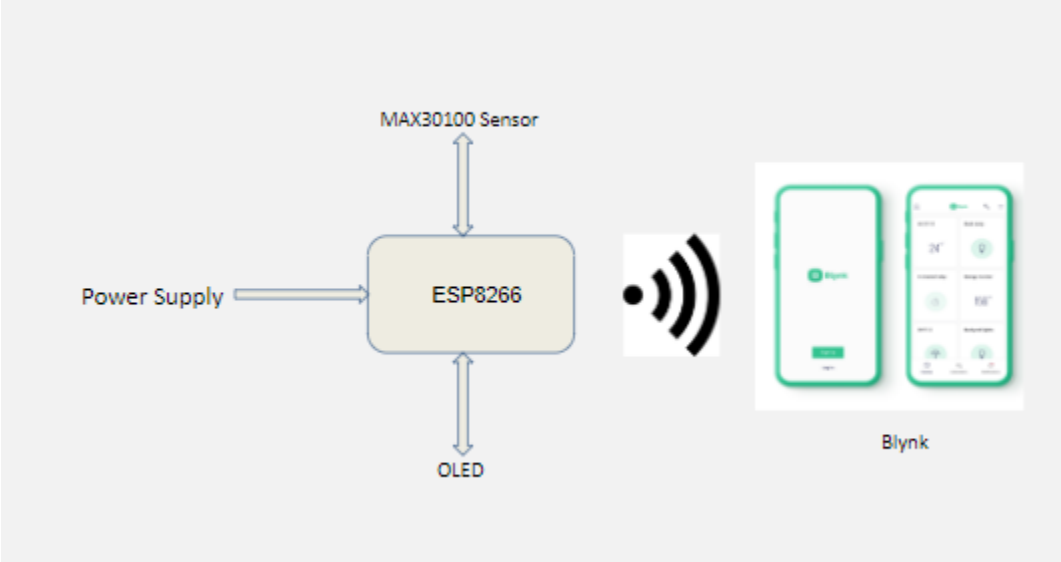


Figure 6.1: Block Diagram

ESP8266 can be powered through a variety of sources depending on the application requirements. Here are some ways to power an ESP8266:

USB: ESP8266 can be powered through a USB port using a USB cable connected to a computer, power bank, or USB charger.

Battery: ESP8266 can be powered by a battery, which can be either a lithium-ion battery or a rechargeable nickel-metal hydride battery.

Power supply module: ESP8266 can also be powered by a power supply module

that converts AC voltage to DC voltage. These modules are commonly used in power supply circuits and can provide a stable voltage output.

When powering ESP8266, it is important to ensure that the voltage and current ratings are within the operating range of the device, which is typically between 2.5V to 3.6V and 80mA to 170mA depending on the operating mode. It is also important to use a power source that is capable of delivering enough current to the device during peak operation to prevent voltage drop or damage to the device.

To connect an ESP8266 with the MAX30100 sensor, you need to use the I2C protocol. The I2C bus uses two wires - SDA (serial data) and SCL (serial clock) - to transmit data between devices. The ESP8266 has built-in hardware support for I2C communication, which makes it easy to interface with the MAX30100 sensor.

To connect an ESP8266 with an OLED display, you can use the I2C or SPI protocol. I2C is generally the preferred protocol as it uses fewer pins and is easier to implement.

To connect an ESP8266 with Blynk, you will need to use the Blynk library in your code. Here are the steps to connect ESP8266 with Blynk:

- Download the Blynk app on your smartphone and create a new account.
- Create a new project in the Blynk app and select ESP8266 as the hardware model.
- Obtain the Auth Token for your project from the Blynk app.
- Include the Blynk library in your Arduino IDE and add the Blynk header file to your code.
- Set up the Wi-Fi connection in your code using the WiFi library.
- Initialize the Blynk object with the Auth Token and Wi-Fi credentials using the Blynk.begin() function.
- Define the functions for your project, such as reading data from sensors or controlling actuators.
- Use the Blynk virtual pins to communicate with the Blynk app. You can use the Blynk.virtualWrite() function to send data to the app and the Blynk.virtualRead() function to receive data from the app.
- Run the code on the ESP8266 and open the Blynk app. You should be able to see the data from your project on the app and control the ESP8266 through

the app.

## 6.2 Circuit Connection

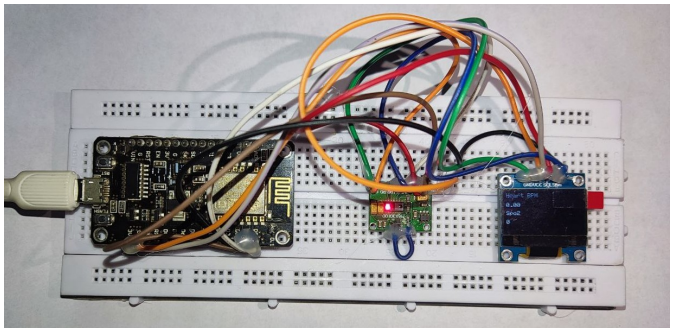

Figure 6.2: Circuit Connection

Here is a figure of our circuit connection.

| ESP32 | MAX30100 |
|---|---|
| 3.3V | VCC |
| GND | GND |
| GPIO#4 (SDA) | SDA |
| GPIO#5 (SCL) | SCL |

Table 6.1: Connection between ESP8266 and MAX30100

| ESP8266 | OLED Display |
|---|---|
| 3.3V | VCC |
| GND | GND |
| GPIO#5 (SDA) | SDA |
| GPIO#4 (SCL) | SCL |

Table 6.2: Connection between ESP8266 and OLED Display

In table 6.1 and 6.2, we can see the pin configuration for the above circuit.

The ESP8266 connects to the WiFi network using the provided credentials and initializes the Blynk library with the authentication token and WiFi object. The Blynk.run() function is called repeatedly in the loop() function to handle Blynk events and maintain the connection to the Blynk server. Note that you need to include the necessary libraries and replace the placeholder values for the WiFi credentials and authentication token with your own values.

# Chapter 7

# Observations From Sensor

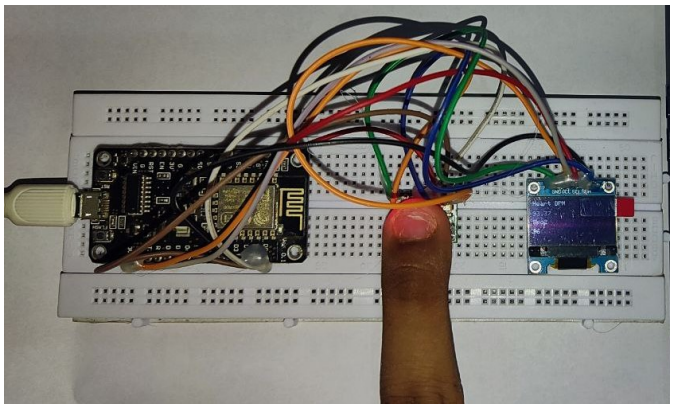

Figure 7.1: Taking Reading From MAX30100

To take the reading from MAX30100, the finger should be placed on the sensor module's finger clip. The clip should be placed on the fingertip, ensuring that the LED lights and the photodiode sensor are in contact with the finger. It's important to keep the finger still while taking the reading, as any movement can cause inaccurate readings.

After taking the reading from MAX30100 and sending the data to Blynk, you can visualize the SpO2 and pulse rate values on the Blynk app in real-time. You can create a custom interface on the Blynk app to display the data using various widgets such as gauges, value displays, and graphs.

We can retrieve the sensor data using Blynk API and a python script. Otherwise we can create a CSV file of sensor data using Coolterm, which will record the data of sensor from serial monitor of Arduino IDE.

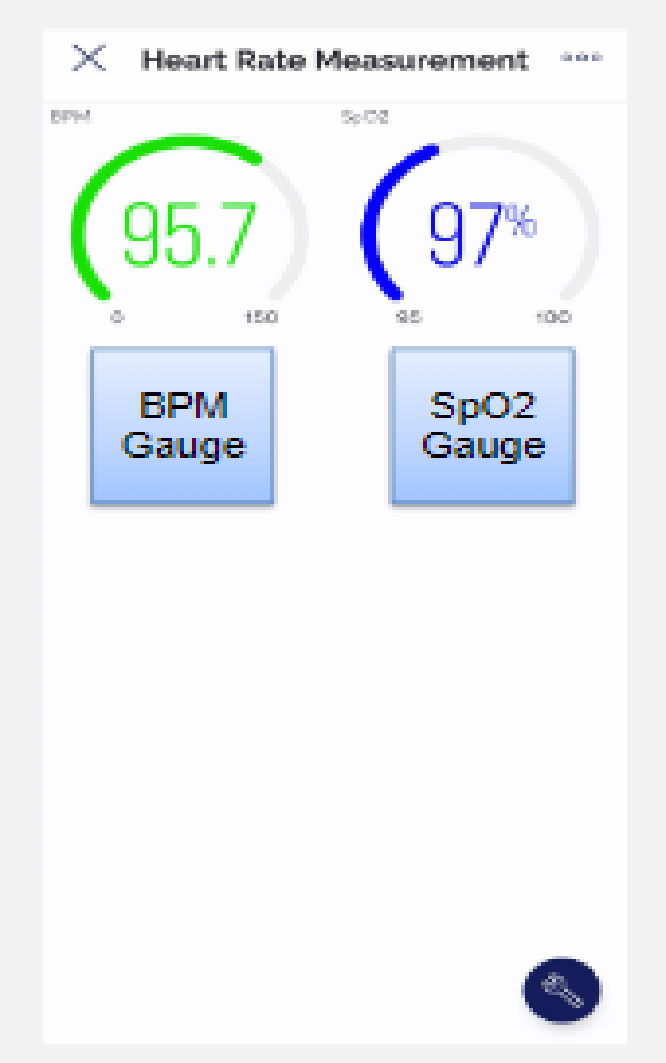


Figure 7.2: Blynk Interface

# Chapter 8

# Machine Learning-Based Heart Disease Classification using Logistic Regression

## 8.1 What is Machine Learning and Logistic Regression

Machine learning is a type of artificial intelligence that allows computers to learn and make predictions or decisions based on data, without being explicitly programmed. Instead of following specific instructions, machine learning algorithms learn patterns in the data and use these patterns to make predictions or decisions.

Logistic regression is a type of machine learning algorithm that is commonly used for binary classification problems, where the goal is to predict whether an outcome is one of two possible values (e.g. yes or no, true or false, normal or abnormal).

## 8.2 Steps of Logistic Regression

1. Data Pre-processing step: This step involves preparing the dataset for logistic regression by cleaning, transforming, and scaling the data to make it suitable for analysis. It also involves splitting the data into training and testing sets.
2. Fitting Logistic Regression to the Training set:Logistic regression works by fitting a logistic curve to the data, which is a mathematical function that maps the input variables to a probability value between 0 and 1.The logistic curve is then used to make predictions about the likelihood of an outcome being one of the two possible values, based on the values of the input variables.
3. Predicting the test result:The model can then be used to make predictions for new individuals, by plugging in their values for the input variables and using the logistic curve to calculate the probability of the desired result.

4. Test accuracy of the result: The accuracy of the predicted results is evaluated using a confusion matrix, which is a table that shows the number of true positives, true negatives, false positives, and false negatives. The accuracy of the model is determined by calculating metrics such as precision, recall, and F1 score.

5. Visualizing the test set result: The final step is to visualize the results of the logistic regression model on the test set.

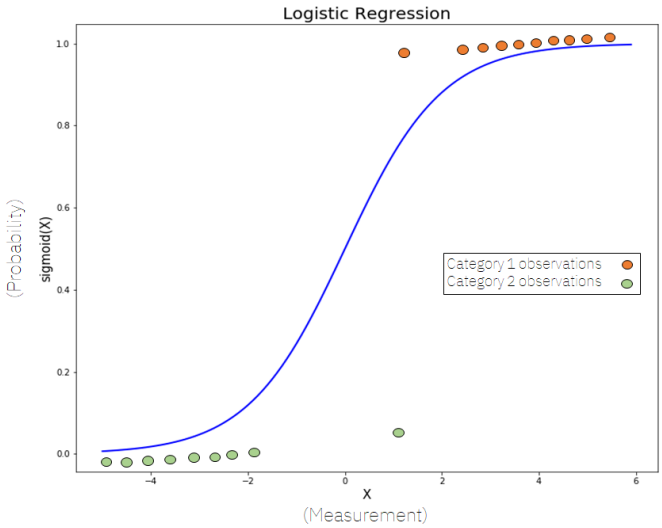


Figure 8.1: Logistic Curve

## 8.3 Logistic Regression Model Using Python

### 8.3.1 Confusion Matrix, Recall, Precision, f1 score

| Abbreviation | Meaning |
|---|---|
| TP | True Positive - The number of correct positive predictions |
| TN | True Negative - The number of correct negative predictions |
| FP | False Positive - The number of incorrect positive predictions |
| FN | False Negative - The number of incorrect negative predictions |

Table 8.1: Meaning of TP, TN, FP, and FN

Confusion matrix is a popular evaluation metric for classification models, including logistic regression. It is used to evaluate the performance of a model by comparing its predicted values with the actual values. Confusion matrix helps us to understand how well our model is performing in terms of its true positive, true negative, false positive, and false negative predictions. These metrics are important in assessing the accuracy of a logistic regression model, particularly in scenarios where

the cost of mis-classifying the different classes is different.

Recall (also called sensitivity) is the proportion of actual positive cases that are correctly identified by the model. It is calculated as the ratio of true positives to the sum of true positives and false negatives. A high recall value indicates that the model is good at identifying positive cases. $R = \frac{TP}{TP+FN}$

Precision is the proportion of predicted positive cases that are actually positive. It is calculated as the ratio of true positives to the sum of true positives and false positives. A high precision value indicates that the model is good at avoiding false positives. $P = \frac{TP}{TP+FP}$

F1 score is defined as the harmonic mean of recall and precision.Note that F1 score can be interpreted as a weighted average of precision and recall, where the weight is given by the harmonic mean. This means that F1 score takes into account both precision and recall, and is a useful metric for evaluating the overall performance of a binary classifier. $F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

### 8.3.2 Python code and Output

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
df = pd.read_csv('/content/drive/MyDrive/Major Project Dataset - Sheet1.csv')
```

Figure 8.2: Loading of Libraries

This code snippet is reading a CSV file named "Major Project Dataset - Sheet1.csv" from a Google Drive folder. The file is being read as a pandas dataframe using the "read_csv()" function. Then, the "train_test_split()" function from scikit-learn is imported to split the dataset into training and testing sets. The logistic regression model is imported from scikit-learn's linear model module.The accuracy score, confusion matrix, and classification report metrics are imported from the metrics module of scikit-learn.Overall, the code seems to be setting up the necessary libraries and functions for implementing logistic regression on the given dataset.

The code given below performs logistic regression on a dataset with features 'Age', 'Pulse Rate', and 'Sp02' to predict the presence of heart condition based on Sp02 and Pulse Rate readings. The dataset is split into training and testing sets, and

the logistic regression model is fit on the training set. The accuracy, confusion matrix, and classification report are printed for the model's performance on the test set.

```
df['Heart Condition'] = (df['Sp02'] < 90) | ((df['Pulse Rate'] > 100) | (df['Pulse Rate'] < 60))
X = df[['Age', 'Pulse Rate', 'Sp02']]
y = df['Heart Condition']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Loading...

Figure 8.3: Accuracy Prediction

```
Accuracy: 0.875
Confusion Matrix:
 [[3 0]
 [1 4]]
Classification Report:
              precision    recall  f1-score   support

       False       0.75      1.00      0.86         3
        True       1.00      0.80      0.89         5

    accuracy                           0.88         8
   macro avg       0.88      0.90      0.87         8
weighted avg       0.91      0.88      0.88         8
```

Figure 8.4: Output of Accuracy Prediction

In the given code, the dataset is split into training and testing data using the train_test_split function from sklearn. The test_size parameter is set to 0.2, which means that 20% of the data is randomly selected for testing, and the remaining 80% is used for training the model. In this case, since the dataset contains 36

rows, 28 rows are used for training the model, and 8 rows are used for testing the model's performance.

```
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

Figure 8.5: Code for plotting confusion matrix

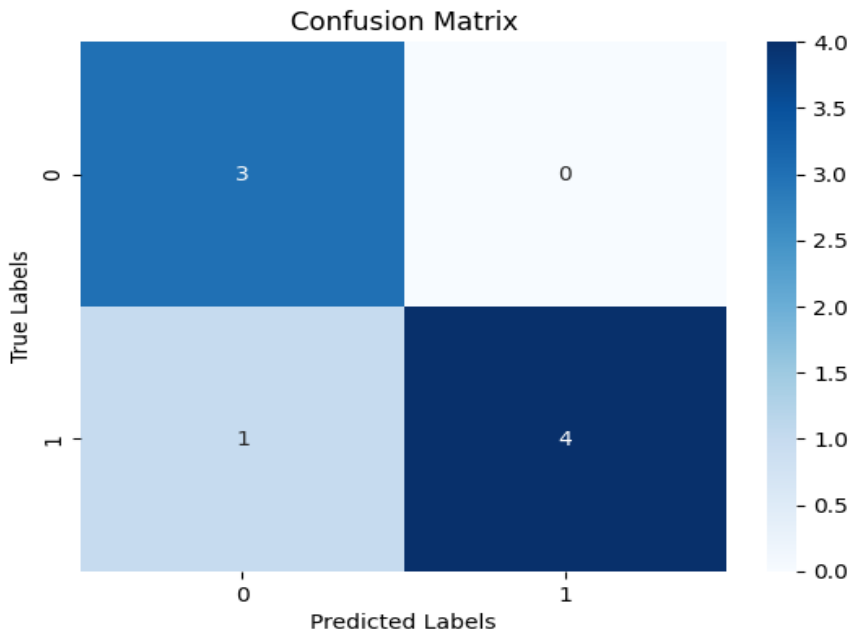


Figure 8.6: Confusion Matrix Plot

In machine learning, the ROC curve (Receiver Operating Characteristic curve) is a graphical representation of the performance of a binary classification model. It is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The TPR is the proportion of actual positive cases that are correctly identified as positive, while the FPR is the proportion of actual negative cases that are incorrectly identified as positive. The ROC

curve helps in choosing an appropriate threshold value that balances the trade-off between TPR and FPR, based on the specific needs of the model.

```
!pip install --upgrade scikit-learn
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

Figure 8.7: ROC Code

A good model will have a curve that is closer to the top left corner, indicating higher TPR and lower FPR.

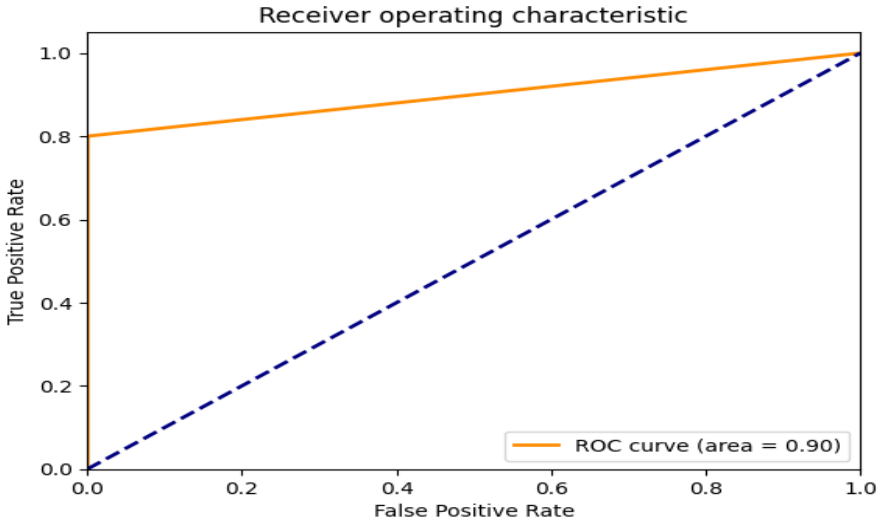


Figure 8.8: ROC Plot

The code below computes the correlation matrix for the variables in the DataFrame df, and then plots a heatmap of the correlation matrix using the seaborn library. The heatmap has annotated values that represent the correlation coefficients between pairs of variables. The cmap='coolwarm' argument sets the color scheme

```
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation matrix plot')
plt.show()
```

Figure 8.9: Code for Correlation matrix

of the heatmap to be blue and red.The correlation coefficient ranges from -1 to 1, where -1 indicates a perfect negative correlation (as one variable increases, the other variable decreases), 0 indicates no correlation, and 1 indicates a perfect positive correlation (as one variable increases, the other variable also increases).

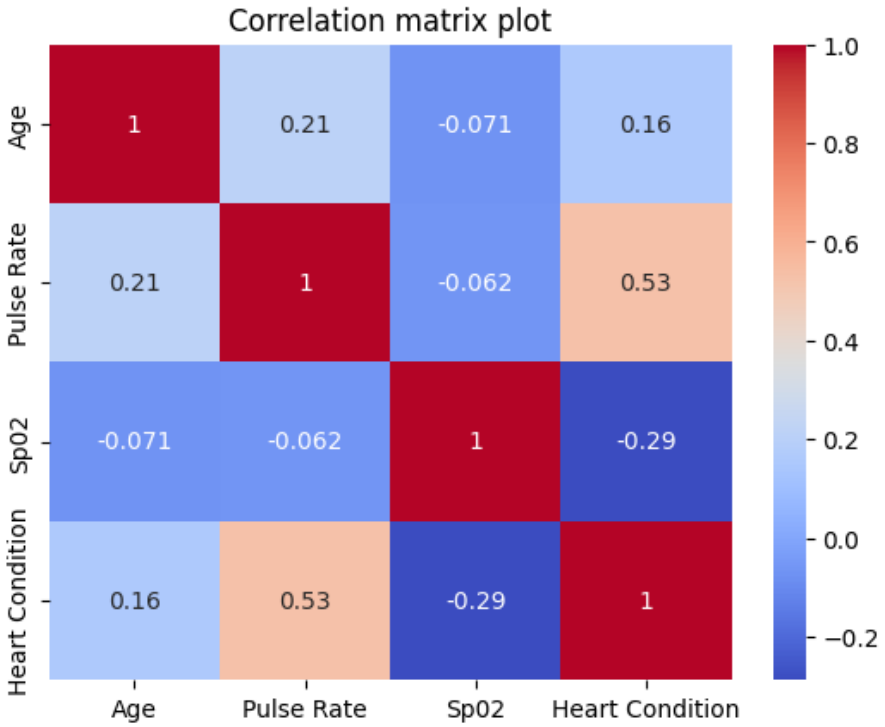


Figure 8.10: Plot for Correlation matrix

# Chapter 9

# Conclusion and Future Work

Based on the analysis of the dataset using logistic regression, we can conclude that the model was able to achieve a good accuracy of 87.5% in predicting heart conditions based on pulse rate and Sp02 readings. The confusion matrix and classification report helped us understand the number of true positives, true negatives, false positives, and false negatives, which are important metrics for evaluating the performance of a classification model.

In terms of future work, one potential area of improvement could be to collect more data and include additional features that might be relevant for predicting heart conditions. Additionally, we could explore other machine learning algorithms such as decision trees, random forests, or support vector machines, and compare their performance to logistic regression. It is also important to validate the performance of the model on an independent dataset to ensure its generalizability.

Overall, the project provides a promising framework for using IoT devices and machine learning techniques for healthcare applications. By developing accurate and reliable predictive models, we can potentially improve the diagnosis and management of various health conditions, leading to better health outcomes for individuals.

# Bibliography


[1] S. Pasha, "Thingspeak Based Sensing and Monitoring System for IoT with Matlab Analysis," *International Journal of New Technology and Research*, vol. 2, no. 6, pp. 66-69, Jun. 2016.

[2] M. M. Islam, A. Rahaman, and M. R. Islam, "Development of Smart Healthcare Monitoring System in IoT Environment," *SN Computer Science*, vol. 1, no. 11, p. 185, Oct. 2020.

[3] E. N. Ganesh, "Health monitoring system using Raspberry Pi and IOT," *Oriental Journal of Computer Science and Technology*, vol. 12, no. 1, pp. 08-13, 2019.

[4] O. Y. Tham, M. A. Markom, A. H. A. Bakar, E. S. M. M. Tan and A. M. Markom, "IoT Health Monitoring Device of Oxygen Saturation (SpO2) and Heart Rate Level," in *2020 1st International Conference on Information Technology, Advanced Mechanical and Electrical Engineering (ICITAMEE)*, Yogyakarta, Indonesia, 2020, pp. 128-133, doi: 10.1109/ICITAMEE50454.2020.9398455.

[5] Chavda, S., Kumbhar, R., Karande, S.and Bangal, S. (2022). Early Detection of Heart Diseases using Decision Tree Classification. *International Journal of Emerging Technologies and Innovative Research*, 9(1), 213-221.

[6] J. Wan, Y. Zou, Y. Li and J. Wang, "Reflective type blood oxygen saturation detection system based on MAX30100," in *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, Shenzhen, China, 2017, pp. 615-619, doi: 10.1109/SPAC.2017.8304350.

[7] M. M. Khan, S. Mehnaz, A. Shaha, M. Nayem, and S. Bourouis, "IoT-based smart health monitoring system for COVID-19 patients," *Computational and Mathematical Methods in Medicine*, vol. 2021, p. 8591036, Mar. 2021.

# Appendix A

# Code

## A.1 Code for Sensor Readings

Here is the code used for reading sensor data:

```
#define BLYNK_TEMPLATE_ID "TMPL3ljr9QBKw"
#define BLYNK_TEMPLATE_NAME "blood␣pressor"
#define BLYNK_AUTH_TOKEN "W3uQCoIWOTO_j01xd2Aa0XQafYMDP-red↪ S_"
#include <Wire.h>

#include "MAX30100_PulseOximeter.h"

#define BLYNK_PRINT Serial

#include <Blynk.h>

#include <ESP8266WiFi.h>

#include <BlynkSimpleEsp8266.h>

#include "Adafruit_GFX.h"

#include "OakOLED.h"

#define REPORTING_PERIOD_MS 1000

OakOLED oled;
```

```
23
24
25 char auth[] = BLYNK_AUTH_TOKEN;
26
27 char ssid[] = "bx";  // type your wifi name
28 char pass[] = "12345678";  // type your wifi password
29
30 BlynkTimer timer;
31
32
33
34 // Connections : SCL PIN - D1 , SDA PIN - D2 , INT PIN -
     red↪  D0
35
36 PulseOximeter pox;
37
38
39
40 float BPM, SpO2;
41
42 uint32_t tsLastReport = 0;
43
44
45
46 const unsigned char bitmap [] PROGMEM =
47
48 {
49
50   0x00, 0x00, 0x00, 0x00, 0x01, 0x80, 0x18, 0x00, 0x0f,
       red↪ 0xe0, 0x7f, 0x00, 0x3f, 0xf9, 0xff, 0xc0,
51
52   0x7f, 0xf9, 0xff, 0xc0, 0x7f, 0xff, 0xff, 0xe0, 0x7f,
       red↪ 0xff, 0xff, 0xe0, 0xff, 0xff, 0xff, 0xf0,
53
54   0xff, 0xf7, 0xff, 0xf0, 0xff, 0xe7, 0xff, 0xf0, 0xff,
       red↪ 0xe7, 0xff, 0xf0, 0x7f, 0xdb, 0xff, 0xe0,
55
56   0x7f, 0x9b, 0xff, 0xe0, 0x00, 0x3b, 0xc0, 0x00, 0x3f,
```

```
    red↪ 0xf9, 0x9f, 0xc0, 0x3f, 0xfd, 0xbf, 0xc0,
57
58    0x1f, 0xfd, 0xbf, 0x80, 0x0f, 0xfd, 0x7f, 0x00, 0x07,
    red↪ 0xfe, 0x7e, 0x00, 0x03, 0xfe, 0xfc, 0x00,
59
60    0x01, 0xff, 0xf8, 0x00, 0x00, 0xff, 0xf0, 0x00, 0x00,
    red↪ 0x7f, 0xe0, 0x00, 0x00, 0x3f, 0xc0, 0x00,
61
62    0x00, 0x0f, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
    red↪ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
63
64  };
65
66
67
68  void onBeatDetected()
69
70  {
71
72    Serial.println("Beat␣Detected!");
73
74    oled.drawBitmap( 60, 20, bitmap, 28, 28, 1);
75
76    oled.display();
77
78  }
79
80
81
82  void setup()
83
84  {
85
86    Serial.begin(115200);
87  Blynk.begin(auth, ssid, pass);
88    oled.begin();
89
90    oled.clearDisplay();
```

```
91
92    oled.setTextSize(1);
93
94    oled.setTextColor(1);
95
96    oled.setCursor(0, 0);
97
98
99
100   oled.println("Initializing␣pulse␣oximeter..");
101
102   oled.display();
103
104
105
106   pinMode(16, OUTPUT);
107
108   Blynk.begin(auth, ssid, pass);
109
110
111
112   Serial.print("Initializing␣Pulse␣Oximeter..");
113
114
115
116   if (!pox.begin())
117
118   {
119
120     Serial.println("FAILED");
121
122     oled.clearDisplay();
123
124     oled.setTextSize(1);
125
126     oled.setTextColor(1);
127
128     oled.setCursor(0, 0);
```

```
129
130       oled.println("FAILED");
131
132       oled.display();
133
134       for (;;);
135
136     }
137
138     else
139
140     {
141
142       oled.clearDisplay();
143
144       oled.setTextSize(1);
145
146       oled.setTextColor(1);
147
148       oled.setCursor(0, 0);
149
150       oled.println("SUCCESS");
151
152       oled.display();
153
154       Serial.println("SUCCESS");
155
156       pox.setOnBeatDetectedCallback(onBeatDetected);
157
158     }
159
160
161
162     // The default current for the IR LED is 50mA and it
        red↪ could be changed by uncommenting the following
        red↪  line.
163
164     //pox.setIRLedCurrent(MAX30100_LED_CURR_7_6MA);
```

```
165
166
167
168 }
169
170
171
172 void loop()
173
174 {
175
176   pox.update();
177
178   Blynk.run();
179
180 //timer.run();
181
182   BPM = pox.getHeartRate();
183
184   SpO2 = pox.getSpO2();
185
186
187     if (millis() - tsLastReport > REPORTING_PERIOD_MS)
188
189   {
190
191     Serial.print("Heart␣rate:");
192
193     Serial.print(BPM);
194
195     Serial.print("␣SpO2:");
196
197     Serial.print(SpO2);
198
199     Serial.println("␣%");
200
201
202
```

```
203     Blynk.virtualWrite(V1, BPM);
204
205     Blynk.virtualWrite(V2, SpO2);
206
207
208
209     oled.clearDisplay();
210
211     oled.setTextSize(1);
212
213     oled.setTextColor(1);
214
215     oled.setCursor(0, 16);
216
217     oled.println(pox.getHeartRate());
218
219
220
221     oled.setTextSize(1);
222
223     oled.setTextColor(1);
224
225     oled.setCursor(0, 0);
226
227     oled.println("Heart␣BPM");
228
229
230
231     oled.setTextSize(1);
232
233     oled.setTextColor(1);
234
235     oled.setCursor(0, 30);
236
237     oled.println("Spo2");
238
239
240
```

```
241       oled.setTextSize(1);
242
243       oled.setTextColor(1);
244
245       oled.setCursor(0, 45);
246
247       oled.println(pox.getSpO2());
248
249       oled.display();
250
251
252
253       tsLastReport = millis();
254
255     }
256
257 }
```

Listing A.1: Code