Indian Institute of Technology, Gandhinagar

ES333 - Project Report

PREDICTING HEART ATTACKS USING STM32 MICROCONTROLLER

Authors:
Aditi Dey - 20110007
Asma Vindhani - 20110227
Kankshi Komre - 20110092
Prakriti - 20110142

Under the guidance of:
Prof. Jhuma Saha
Ayush Srivastava

Index

Introduction

Heart disease is a leading cause of death worldwide, and early detection of heart attacks is crucial to improving patient outcomes. In recent years, there has been growing interest in

developing portable and non-invasive heart attack predictors that can be used to detect signs of a heart attack in real time. One such approach is to use microcontrollers, which are small and low-cost devices that can be programmed to perform a wide range of tasks.

A heart attack predictor using a microcontroller is a system that can be used to detect early signs of a heart attack by analyzing real-time data from various sensors. The microcontroller is programmed to use an algorithm that takes into account various physiological parameters, such as heart rate, blood pressure, and oxygen saturation, to predict the likelihood of a heart attack. The system can provide an alert when the risk of a heart attack exceeds a certain threshold, allowing patients to seek medical attention immediately.

The heart attack predictor using a microcontroller has several advantages over traditional methods of heart attack detection, including portability, non-invasiveness, and low cost. This makes it an attractive option for use in remote or low-resource settings, where access to medical equipment and personnel may be limited.

In this project, we aim to design and implement a heart attack predictor using a microcontroller. Our goal is to develop a system that is accurate, reliable, and easy to use and that can provide early warning signs of a heart attack to patients and medical professionals alike.

Hardware

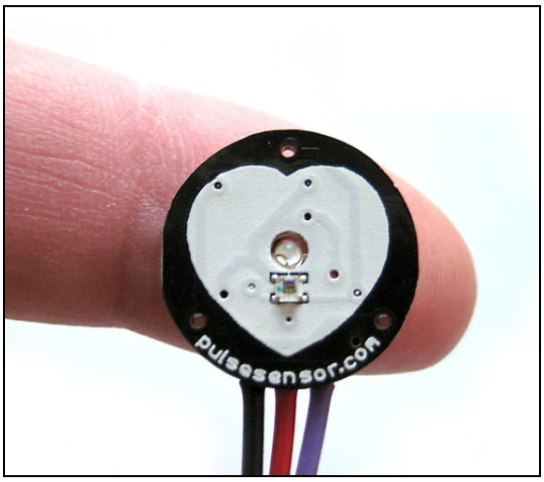
We are implementing this project using STM32F439ZI microcontroller. In the microcontroller, we have used its User LED's for the results. There is also an enable signal for the complete system.

Under this section, we are listing all the components required and their configuration on the microcontroller.

Materials Used

1) STM32F439ZI Discovery Board
2) Pulse Sensor (REES52)
3) 2 LED's
4) External USB (C-type)
5) Breadboard
6) Jumper Wires

Pulse Sensor



REES52 Pulse Sensor
Image Source: https://pulsesensor.com/

The Pulse sensor is connected to the STM32 board to take the user's pulse as analog input. Analog values of pulses are transmitted through ADC in continuous conversion mode to the STM32 board. Continuous conversion mode allows us to read and convert the data from the sensor continuously instead of reading discontinuous inputs per run. These digital values of pulses are transmitted to the serial terminal using an external USB via a buffer. The serial terminal is used to observe the digital values of the input in a terminal.

STM32 Configuration

Component	Port & Pin configuration	Pin Mode
Pulse Sensor	PA0	Input
Enable	PB8	Input

External LED1	PE9	Output
External LED2	PF13	Output

ADC Settings:
In Pin PA0, we have used ADC1_IN0 for ADC conversion and we have enabled the continuous conversion mode. The following is the code for conversion and reading of values.

```
HAL_ADC_Start(&hadc1); // starting the ADC

HAL_ADC_PollForConversion(&hadc1, 100); // poll for conversion

uint16_t adc_val = HAL_ADC_GetValue(&hadc1); // Read analog value from ADC
```

Then the values are transmitted through external USB.

```
sprintf(buffer, "%d\n", value);
                    CDC_Transmit_FS((uint8_t*) buffer, strlen(buffer));
```

Methodology for Data Analysis

Method 1: Prediction using MMSE

Once we had the input heartbeat, we conducted a statistical technique called Maximum mean square error (MMSE) which is commonly used in data analysis. MMSE is used to estimate a parameter or predict a value based on observed data. It minimizes the expected value of the squared difference between the predicted value and the true value of the parameter.

1. MMSE calculation: We consider two arrays, the base array (which we take as reference, and the beats array, and we calculate the MMSE of the two arrays using this function:

```
float mmse(float x, float y, int n) {
    float sum = 0.0, diff;
    diff = x - y;
    return (diff * diff)/n;
}
```

2. We have used the following Base array as the reference for comparison:

```
    float base[600] = { 80.00, 69.30, 69.00, 70.93, 72.87, 68.40, 71.33,
83.57, 85.20, 84.07, 84.47, 87.17, 87.10, 85.33, 84.17, 86.30, . . . . .
```

```
            84.70, 86.10, 89.23, 84.37, 84.63, 84.30, 83.00 };
```

3. We make the beats array by taking input from the sensor:

```
int curr = 0;
while (curr < 600 && HAL_GPIO_ReadPin(GPIOB, input_Pin) == 1) {

    HAL_ADC_Start(&hadc1); // start the adc
    HAL_ADC_PollForConversion(&hadc1, 100); // poll for conversion
    // Read analog value from ADC
    uint16_t value = HAL_ADC_GetValue(&hadc1);

    beats[curr] = (float) value / 10.0;
```

4. Analysis: We have kept a counter variable high, and everytime the MMSE value crosses a given threshold value, we increment this counter variable.

```
int i = 0;
            while (i < 600 && HAL_GPIO_ReadPin(GPIOB, input_Pin) == 1) {
                float mmse_curr = mmse(base[curr], beats[curr], 600);
                if (mmse_curr > 10.0) {
                    high++;
                }
                i++;
            }
```

5. Heart Health Estimation: If this counter variable crosses a certain threshold, this means that the heart rate is varying too randomly, and there is a risk of heart attack.

```
if (HAL_GPIO_ReadPin(GPIOB, input_Pin) == 1) {
                if (high > 550) {
                    HAL_GPIO_WritePin(GPIOB, red_Pin, SET);
                    HAL_GPIO_WritePin(GPIOB, green_Pin, RESET);
                    HAL_GPIO_WritePin(GPIOB, blue_Pin, RESET);
                } else {
                    HAL_GPIO_WritePin(GPIOB, green_Pin, SET);
                    HAL_GPIO_WritePin(GPIOB, red_Pin, RESET);
                    HAL_GPIO_WritePin(GPIOB, blue_Pin, RESET);
                }
```

Results:

We light the On Board Red LED if there is a risk of heart attack, and the On Board Green LED if there isn't any risk.

Method 2: Threshold Detection

We know that the heart rate of a person is between 60-100 bpm. Since the sensor present with us can give us some imprecise value so we have kept a range of 50-110 bpm.

Every time the current heart rate goes down 50, then limit_cross_low increases its value by 1. If the lower limit is crossed 1 or 2 times, then it's fine, but if it crosses more than 10 times, then it might be that it has actually crossed the threshold.

```
if (beats[curr] < 50.0) {
   limit_cross_low++
}
```

Every time the current heart rate goes up 110, then limit_cross_high increases its value by 1. If the upper limit is crossed 1 or 2 times, then it's fine, but if it crosses more than 10 times, then it might be that it has actually crossed the threshold.

```
if (beats[curr] > 110.0) {
   limit_cross_high++
}
```

Therefore, when the heart rate goes down below 50 bpm, then an external green led glows. When the heart rate goes above 110 bpm, then an external red led glows.

Results:
We lit an external green LED if the lower limit is cross, and an external Red LED if the upper limit is crossed. The lower and higher limits in most heart attack sensors are taken to be 80 and 110 respectively, but the output from our sensor was going as low as 60, hence the green LED was always glowing.

Software Development

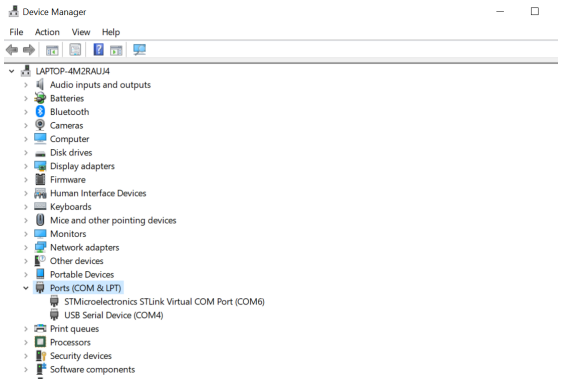
Details

For the front-end development, we used HTML, CSS, and JavaScript, while the Flask framework in Python was used for the backend. The full Python code files for this section can be found at this link.

Static: contains animation.gif that is shown on the website.
Templates: contains the html files rendered by the framework.
App.py: server file of Flask framework
Code1.py: contains code for receiving serial data.
Data.pkl: stores the array of received data
Data1.pkl: stores the user data.

Backend code

We initially created a Python file to read the serial data coming from the STM board using USB. Once USB is configured on the STM board, we can see the two ports in the device manager.



Then to receive data on serial data from the STM, we can use the code given below. Serial is a Python library used to access serial ports on your device that is configured to be in Application mode.

```
ser = serial.Serial('COM4', 115200, timeout=1)
a = 0
arr = []
while (a < 600):
    print("Sample number ", end="")
    print(a, end=" ")
    print(" : ", end="")
```

```
        data = ser.readline().decode()
        a = a+1
        if (data == b''):
            value = 0
        else:
            try:
                value = float(data)
            except:
                value = 0
        arr.append(value)
    return arr
```

Then we can store the incoming data in pickle files using the following code. Pickle is a python library that allows flexibility when deserializing objects. You can easily save different variables into a Pickle file and load them back in a different Python session.

Now to store data in pickle (.pkl) files, we can use the following code:

```
arr = get_values()
with open('data.pkl', 'wb') as f:
    pickle.dump(arr, f)
```

To actually import the data and process it, to plot the ecg signal, we had the following process.

1) Configuring flask app

```
from data import arr
from data import arr2
from flask import Flask, render_template

app = Flask(__name__)

if __name__ == '__main__':
    app.run(debug=True)
```

2) Reading data from pickle files

```
import pickle
with open('data.pkl', 'rb') as f:
    arr = pickle.load(f)
with open('data1.pkl', 'rb') as q:
    arr2 = pickle.load(q)
```

3) Rendering data: Rendering the data on the '/' endpoint. Index.html is rendered on accessing this endpoint.

```
@app.route('/')
def index():
    chart_data = arr[50:150]
    chart_labels = []
    for i in range(50,150):
        chart_labels.append(i)
    heartRate = int(sum(arr)/600.0)
    return render_template('index.html', chart_data=chart_data,
chart_labels=chart_labels, name=arr2[0], age=arr2[1], rate=heartRate)
```

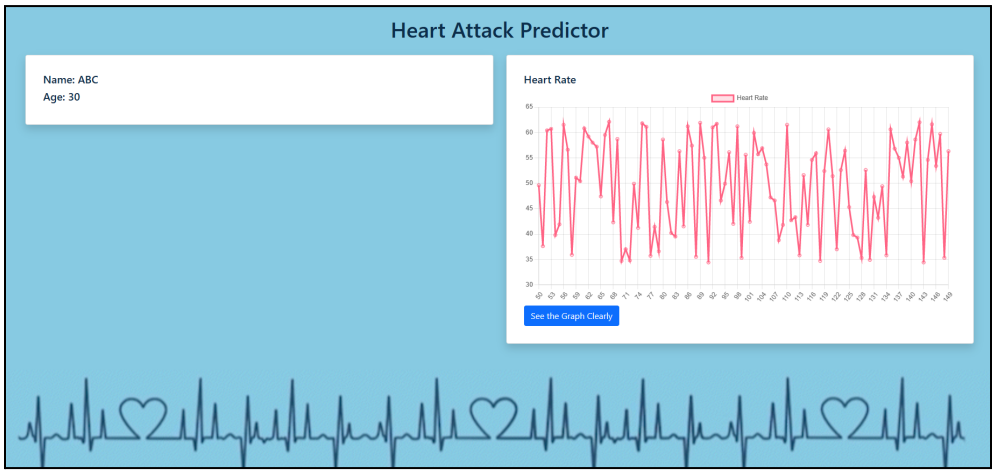
Rendering the data on the '/see-graph' endpoint. Index2.html is rendered on accessing this endpoint.

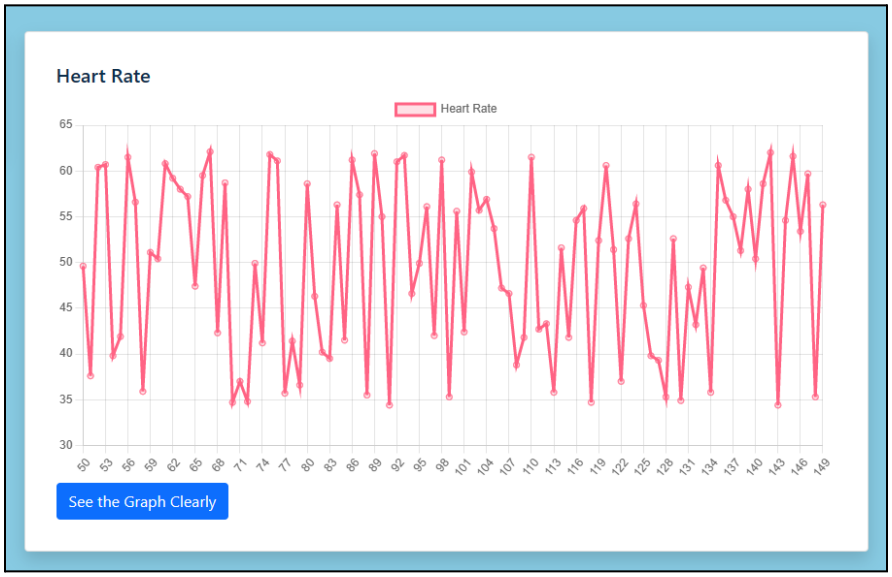
```
@app.route('/see-graph')
def indexx():
    chart_data = arr[50:350]
    chart_labels = []
    for i in range(50,350):
        chart_labels.append(i)

    return render_template('index2.html', chart_data=chart_data,
chart_labels=chart_labels)
```

Frontend code

1) **Index.html:** Contains code for rendering the home page and a button to go on the '/see-graph' endpoint.
2) **Index1.html:** Shows all graphs of all the data points received from the terminal.

Results


Heart Attack Predictor
Name: ABC
Age: 30
Heart Rate
Heart Rate
See the Graph Clearly


Heart Rate
Heart Rate
65
60
55
50
45
40
35
30
50 53 56 59 62 65 68 71 74 77 80 83 86 89 92 95 98 101 104 107 110 113 116 119 122 125 128 131 134 137 140 143 146 149
See the Graph Clearly