Wearable Device for TeleHealth Monitoring

Mouhamad Baker Hmayed, Ali Yaakoub
DIBRIS, Università degli Studi di Genova

***Abstract*—The Fitness and Healthcare Watch Tracker is a telemedicine device designed for remote real-time health monitoring. Continuously tracks heart rate (BPM), oxygen saturation SpO_2, and body temperature, transmitting the data to the Arduino IoT Cloud for remote access by healthcare professionals. Built using the Arduino ESP32S3 with an integrated LCD, the device uses the MAX30102 pulse oximeter and the LM35 temperature sensor for accurate measurements. Data are stored and can be sent as an Excel report by email every 24 hours. The MQTT protocol ensures efficient real-time communication, enabling continuous monitoring through a cloud-based dashboard. This project provides a low-power, IoT-enabled solution for accessible, real-time healthcare tracking.**

***Index Terms*—Wearable Device, Telemedicine, IoT, SP32S3, MAX30102, LM35.**

I. Introduction

The goal of this telemedicine device, named the Fitness and Healthcare Watch Tracker, is to provide remote real-time health care monitoring. This device is responsible for monitoring BPM (beats per minute), temperature, and SpO_2 levels in real-time. The data is sent to the IoT Arduino Cloud, where doctors can view the data in real-time plots. Additionally, the data can be saved every 24 hours and sent via email as an Excel file.

II. Methodology

A. Hardware Design

The project was developed using the Arduino ESP32S3 1.28 connected to an LCD. We chose this Arduino model because the heart rate sensor we selected works perfectly with it, allowing us to visualize all the data on the watch screen. This type of Arduino has six free general-purpose input/output (GPIO) pins, I^2C communication pins, and ADC pins. The Arduino can also provide both 3.3V and 5V power.

We used two sensors in our project. The first is the MAX30102, chosen for its high accuracy in measurements. This sensor uses four pins: one pin (V_{in}) is connected to the 5V pin of the Arduino, the ground pin, and the two main pins SDA (data line) and SCL (clock line). The sensor communicates using the I^2C protocol, where SDA transfers data between the MAX30102 sensor and the ESP32S3 micro-controller, and SCL keeps the communication timing synchronized. We used pin 15 for SCL and pin 16 for SDA after defining them as I^2C pins in the code. The ESP32-S3 Touch LCD 1.28" from Waveshare uses SPI (Serial Peripheral Interface) communication, not I²C. SPI is much faster than I^2C, making it ideal for displays that require quick updates. The connection involves key SPI pins like MOSI (data), SCK (clock), and CS (chip select), along with DC (data/command) and RST (reset) for control. Unlike I^2C, which is better for low-speed sensors, SPI enables smooth, flicker-free graphics on the LCD. This is why the ESP32-S3 communicates with the GC9A01 display driver via SPI instead of I^2C for better performance For the temperature sensor we used the LM35 which has three pins, one pin is powered by 3.3V from the Arduino, one pin is grounded, and the third pin (V_{out}) is connected to pin 17 of the Arduino after defining it as an ADC pin. This pin converts the ADC value to voltage and then to Celsius, displaying it on the screen. A small 3.7V lithium battery is planned to power the Arduino.

B. Software and Communication Implementation

For the IoT part, we used the Arduino IoT Cloud. After creating our code, we connected our device to the cloud and saved the "SecretDeviceKey", which will be shared only with the physician and the hospital for patient privacy. We also defined the Wi-Fi SSID and password to connect the micro-controller to the internet, enabling it to send and receive data from the Arduino IoT Cloud. The micro-controller needs internet access to send health data to the cloud, receive data or control commands remotely (e.g. from a mobile app or web dashboard), and sync data in real-time for remote monitoring. The Arduino IoT Cloud in our project relies on MQTT as the primary communication protocol. MQTT uses minimal data for communication, making it ideal for low-power and low-bandwidth devices like our micro-controller (ESP32S3, Arduino). Health monitoring requires fast and continuous updates, and with this protocol a dashboard can subscribe to the data without overloading the micro-controller. MQTT has "Quality of Service" (QoS) levels, ensuring data delivery even with unstable connections (e.g. Wi-Fi drops). Due to its lightweight, efficiency, and real-time communication features, MQTT is the best protocol for our project to achieve real-time monitoring.

Libraries used:

Wire.h: Enables communication via I^2C (for MAX30102).

MAX30105.h: Library for the MAX30102 pulse-oximeter sensor.

SPI.h: Supports SPI communication (screen display).

Adafruit_GFX.h: Graphics library for display functions.

Adafruit_GC9A01A.h: Library for the GC9A01A TFT display.

thingProperties.h: Manages Arduino IoT Cloud communication.

We used the Arduino IDE to write our whole code with both electronic and IoT parts, and the most important commands are :

Wire.begin(SDA_PIN, SCL_PIN) // Initializes I^2C communication using defined SDA/SCL pins.

particleSensor.setup();
particleSensor.setPulseAmplitudeRed(0x10);
particleSensor.setPulseAmplitudeIR(0x10); // Configures MAX30102 and sets LED brightness for red & IR light.

initProperties();
ArduinoCloud.begin(ArduinoIoTPreferredConnection); // Connects to the Arduino IoT Cloud.

if(currentMillis - lastBPMUpdate >= 10) { lastBPMUpdate = currentMillis;
long irValue = particleSensor.getIR(); // Checks the infrared (IR) value from MAX30102 every 10ms.

if(irValue > 2500) long delta = millis() - lastBeat;
if(delta > 120) lastBeat = millis(); beatsPerMinute = 60000.0 / (delta * 0.5);
rates[rateSpot++] = (byte)beatsPerMinute; rateSpot %= RATE_SIZE; else beatsPerMinute = 0; // If a finger is detected (IR > 2500), it calculates BPM based on the time interval between beats.

if(currentMillis - lastTempUpdate >= 500) { lastTempUpdate = currentMillis;
int rawADC = analogRead(LM35_PIN);
float voltage = (rawADC / 4095.0) * 3.3; // Reads the LM35 temperature sensor every 500ms.
temperature = voltage * 100.0; // Converts ADC value to voltage, then to Celsius.
onTemperatureChange();

long redValue = particleSensor.getRed();
long irValue = particleSensor.getIR();
if (irValue > 2500) { float ratio = (float)redValue / (float)irValue;
spo2 = 104 - (17 * ratio);} else { spo2 = 0; // Reset SpO_2 when no finger detected }

tft.setTextColor(GC9A01A_RED); String bpmText = "BPM: " + String(bPM);
tft.setCursor(centerX - bpmText.length() * 12 / 2, centerY - 40);
tft.print(bpmText); // Displays BPM in red on the watch screen.

#ifndef THINGPROPERTIES_H
#define THINGPROPERTIES_H #include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>
const char DEVICE_LOGIN_NAME[] = "ed1f09f3-d970-4d1b-a27e-83901e19c122";
const char DEVICE_KEY[] = "H60YSI7do74#MM420wkD6oWXk";
const char SSID[] = "ali11";
const char PASS[] = "ali121312";
WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);

The header file thingProperties.h connects the device to Wi-Fi using the provided SSID and password, then authenticates it with the Arduino IoT Cloud using a unique device key. It defines cloud variables (bPM, SpO_2, temperature) to store real-time health data, which syncs automatically with the cloud. This allows remote monitoring via a dashboard or mobile app for real-time tracking.

III. Results and Discussion

Using IoT ARDUINO Cloud we created a percentage gauge for SpO_2 and a real-time plotter for both SpO_2 and BPM,on the dashboard allowing the physician to check them continuously.
The heart rate measurement in this project is done using the MAX30102 pulse oximeter sensor. The sensor emits infrared (IR) and red light onto the finger, and the amount of light absorbed by the blood varies with the pulsation of the heart. The IR value is read every 10ms, and if a finger is detected (IR $>$ 2500), the time interval (delta) between two consecutive beats is calculated. Using this time, the heart rate is determined using the formula:

$$BPM = \frac{6000}{delta * 0.5}$$

The temperature measurement in this project is done using the LM35 temperature sensor, which outputs a voltage proportional to the temperature. Every 500ms, the microcontroller reads the sensor's analog voltage using the ADC (Analog-to-Digital Converter). The raw ADC value is converted into voltage using the formula:

$$voltage = \frac{rawADC}{4095.0} * 3.3$$

Since the LM35 provides 10mV per °C, the temperature in Celsius is calculated as:

$$temperature = \frac{voltage}{100}$$

For SpO_2 the sensor emits red and infrared (IR) light onto the finger, and the amount of light absorbed varies based on oxygen levels in the blood every 500 ms.
ratio= Red Light Value / IR Light Value Using an approximate formula, SpO_2 is estimated:

$$SPO2 = 104 - (17 * ratio)$$

IV. Conclusion

The data can be managed using SQL EDR to store it within the hospital system, enabling remote patient monitoring post-therapy while they remain at home. Further development of the device could include an emergency notification system to alert healthcare providers if any vital sign readings become abnormal.