IoT Project Summary Report

Appnea

Introduction

Background: Sleep apnea is a medical condition that affects a significant portion of the population, leading to serious health complications if left untreated. Despite its prevalence, many remain undiagnosed due to the lack of accessible, user-friendly detection methods. In our project, we define an apnea event as a time span of at least 8 seconds without a sign of breathing.

Objective:
We propose a smart monitoring system that leverages modern technology to monitor vital signs indicative of sleep apnea or hypopnea. By tracking breathing rate during sleep, our system aims to identify abnormalities that signal the condition, all from the comfort of one's home.

Market analysis:

Currently, many available solutions are either too specialized for home use or come with a steep price, limiting accessibility. For instance, the WatchPAT ONE, though innovative, is priced at $190, making it less attainable .

Our system not only focuses specifically on sleep apnea but also aims to be a more affordable, home-based solution.

System Architecture and Design

ESP-32: The system's hardware component is spearheaded by an ESP-32 microcontroller, which is connected to our sensor and transmits Bluetooth messages to our app.

Accelerometer: Allows tracking of chest movements.

Pulse Oximeter(oxygen saturation sensor): It monitors the oxygen levels in the blood, and BPM providing crucial data for apnea detection.

Android Application: The Android application serves as the user interface, a comfortable and interactive app through which users interact with and receive insights from their collected health data.

Python Integration: By leveraging Python, the system can conduct sophisticated data processing tasks such as signal filtering and peak detection.

Arduino IDE code recap

Multi threading- In order to sample from both the IMU and the saturation sensor simultaneously, we utilizing the multi threading capabilities of the ESP-32. We dedicated a separate thread for sampling from the saturation sensor as follows:

```
// Create a task that will run the getSpO2Task function on core 0
xTaskCreatePinnedToCore(
  getSpO2Task,   /* Task function. */
  "GetSpO2Task", /* Name of task. */
  10000,         /* Stack size of task */
  NULL,          /* parameter of the task */
  1,             /* priority of the task */
  NULL,          /* Task handle to keep track of created task */
  0);            /* pin task to core 0 */
```

In this task, we used the manufacturer’s algorithms as provided in Arduino IDE example libraries.

Android Studio code recap

The 3 most important modules are:
MainActivity.java - Serves as the main entry point of the app, handling the setup of the app's main interface and initial permissions.

TerminalFragment.java - Manages the terminal interface for serial communication, processes incoming data, Visualizes data using a line chart, handles Python script integration for data analysis.

LoadCSV.java - Allows users to analyze their recorded data, showing charts for both HR and spO2, displays a list of all apneas events that were detected during the period.

Finding apnea events – general workflow

- We try to find new apnea events every fixed number of minutes.
- We keep track of IMU data during that time (then reset).
- Whenever the fixed time span ends, we call our algorithm.
- We start by applying a low pass filter to the magnitude of the acceleration values.
- Using the filtered data, we try to find peaks – each peak is a sharp movement of the chest – a sign of breathing.

•

Data Preprocessing Techniques:

- Noise Reduction: A low-pass filter is applied to the collected data, particularly effective for motion data from accelerometers and gyroscopes. This step is crucial for mitigating the impact of sensor noise and external vibrations, ensuring that subsequent analyses are based on accurate representations of the user's movements. Since the frequency of breathing during rest is 0.2-0.4 hz, we chose a cutoff frequency that's slightly higher – 0.7, to eliminate as much noise as possible.

```
def butter_lowpass(cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def butter_lowpass_filter(data, cutoff, fs, order=5):
    # Ensure each element is converted to a float
    if not isinstance(data, list):
        data = list(data)
    data = [float(i) for i in data]
    data = np.array(data)

    b, a = butter_lowpass(cutoff, fs, order=order)
    y = lfilter(b, a, data)
    return y.tolist()  # Convert the filtered data back to a list for compatibility
```

Finding peaks – details

Without any parameters to find_peaks, any local maximum is a peak. This could make peaks out of noise – we want only peaks from breathing!

The parameters define what we consider a peak:

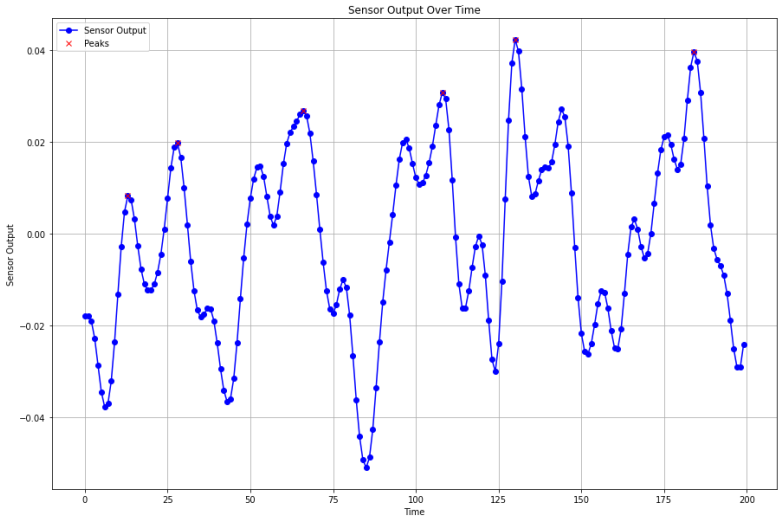
Prominence: defined as the height of a peak relative to the lowest contour line surrounding that peak and not containing any higher peak. This helps filter out minor peaks that are probably caused by noise. By trial and error on captured data, we adjusted this parameter so that it will not detect "noise peaks", and on the other hand will not ignore "real peaks".

Distance: Makes sure peaks are not too close to each other. In our case it means that there must must be a difference of at least 0.8 seconds between peaks.

```
# Detect peaks - adjust parameters as necessary
peaks, _ = find_peaks(magnitudes_list, prominence=0.03, distance=8)

# Calculate intervals between peaks (in terms of data points)
intervals = np.diff(peaks) * sample_interval

# Initialize the list to hold the timestamps and durations of apnea events
apnea_events = []

# Define the apnea threshold (e.g., 8 seconds for adults)
apnea_threshold = 8

# Loop through intervals to find apnea events, calculate their exact times, and durations
for i, interval in enumerate(intervals):
    if interval > apnea_threshold:
        event_time = start_time + timedelta(seconds=peaks[i] * sample_interval)
        # Format the timestamp and duration as strings and append as a sublist
        apnea_events.append([event_time.strftime('%Y-%m-%d %H:%M:%S'), str(interval)])

return apnea_events
```

Visualizations:

An example of typical IMU data received during normal breathing



An example of apnea event detection used to spot Apnea\Hypopnea from the SpO2 sensor