ECEN 404

Electrical Design Laboratory - Fall 2024

Progress Report

Hemaya: Non-invasive multi-sensor wearable wristband for fatigue prevention

Team 2:

Ahmad Al-Ibrahim - 231001659

Ibrahim Al-Naimi - 528001587

Noof Al- Al-Meghessib - 430006372

Aisha Al-Suwaidi - 630006277

Noora Al-Muhannadi - 731006160

Course Instructor:

Dr. Khalid Qaraqe

Electrical Engineering Department, Texas A&M University at Qatar

Mentor:

Dr. Lilia Aljihmani

Lab Instructor:

Dr. Wesam Mansour

Electrical Engineering Department, Texas A&M University at Qatar

Submission Date

7th November 2024

“On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work.”

Abstract

Fatigue among construction workers is a significant concern, resulting in lower productivity and increased safety hazards. Present techniques for fatigue detection frequently prove to be invasive, unreliable, or challenging to execute in demanding work conditions. In response to this issue, our initiative, ‘Hemaya’, introduces a non-invasive, multi-sensor wearable wristband designed to measure fatigue via critical physiological metrics: heart rate, oxygen saturation, and tremor detection. Through the integration of these sensors with real-time data processing and a machine learning model, the wristband facilitates precise fatigue detection, notifying supervisors through a cloud-connected system. This solution improves safety in the workplace by providing a reliable, continuous monitoring system designed for construction workers in harsh environments.

Table of Contents

1. Introduction & Project Overview

1.1 Introduction

Fatigue among construction workers is a significant concern that can result in severe accidents, decreased productivity, and long-term medical conditions. Due to the physically intense and frequently hazardous nature of construction work, early identification of fatigue is crucial for protecting worker safety. 'Hemaya', a non-invasive multi-sensor wearable wristband, has been developed to address this difficulty. The device tracks physiological metrics like heart rate, oxygen saturation (SpO2), and tremors, which are critical indications of fatigue. ‘Hemaya’ offers real-time fatigue detection by consistently monitoring these signals and sending notifications to supervisors via a cloud-based system. This facilitates immediate action to prevent accidents and reduce health concerns.

This progress report outlines advancements in the project, including the design and integration of sensors, the development of circuits, and the manufacturing of hardware components. Collecting and analyzing data from sensors contribute to a machine-learning model for precise fatigue detection. The testing and troubleshooting procedures, outcomes from initial trials, and difficulties faced are addressed. Following that, actions for more modification and potential enhancements are defined to optimize the system's efficacy in practical construction settings.

1.2 Problem Statement

In the construction field, fatigue is a significant element that negatively impacts worker safety and performance. The unpredictable work conditions, extended hours, and physical demands associated with construction work frequently result in worker fatigue, which, if unregulated, may lead to accidents and reduced efficiency. Current methods for identifying fatigue are insufficient for the dynamic and challenging conditions of construction sites, where continuous monitoring is difficult. Consequently, there is an urgent need for a non-invasive, real-time system to monitor physiological markers to

identify early signs of exhaustion, facilitating immediate action to prevent accidents and enhance overall job site safety.

1.3 Proposed Solution

A non-invasive multi-sensor wristband, 'Hemaya,' has been developed to reduce fatigue in construction workers. This wearable device monitors essential physiological indicators including heart rate, oxygen saturation (SpO2), and tremors, which are vital indications of fatigue. Through continuous monitoring of these factors, the wristband can identify early indicators of fatigue and transfer the data to a cloud-based system for immediate analysis. When fatigue levels exceed set thresholds, automatic notifications are sent to supervisors, facilitating immediate action. This proactive strategy seeks to improve worker safety, decrease the probability of fatigue-related accidents, and increase overall productivity on construction sites.

2. Simulation Results, Visual Prototyping, and Analysis

2.1 Upper-Level Modeling

Figure 1 illustrates the upper-level modeling diagram, which outlines a system designed to monitor and evaluate fatigue levels. The key components of the designed system consist of two sensors. The first sensor is the pulse oximeter MAX30102 which measures the heart rate and the SpO2. The second sensor is the ADXL335 accelerometer which measures the tremor levels. The sensors and the 6V rechargeable battery are connected to the Arduino Uno R4 WiFi microcontroller to collect real-time data and upload them to the cloud for further analysis. This is where the Machine Learning part comes into function. The ML model evaluates the data and compares it to the set threshold for each parameter used. If the SpO2 level is below 95%, the model will indicate fatigue. In addition, the threshold for heart rate will be calculated based on the user's age. The calculated threshold indicates if there is fatigue. Similarly, with the tremor, if it is detected to be between 10 to 20 Hz and amplitude, then the model will detect fatigue. Furthermore, an alarm system is integrated with the mobile application to

send an alert and immediately notify the supervisor in case of any fatigue detections.

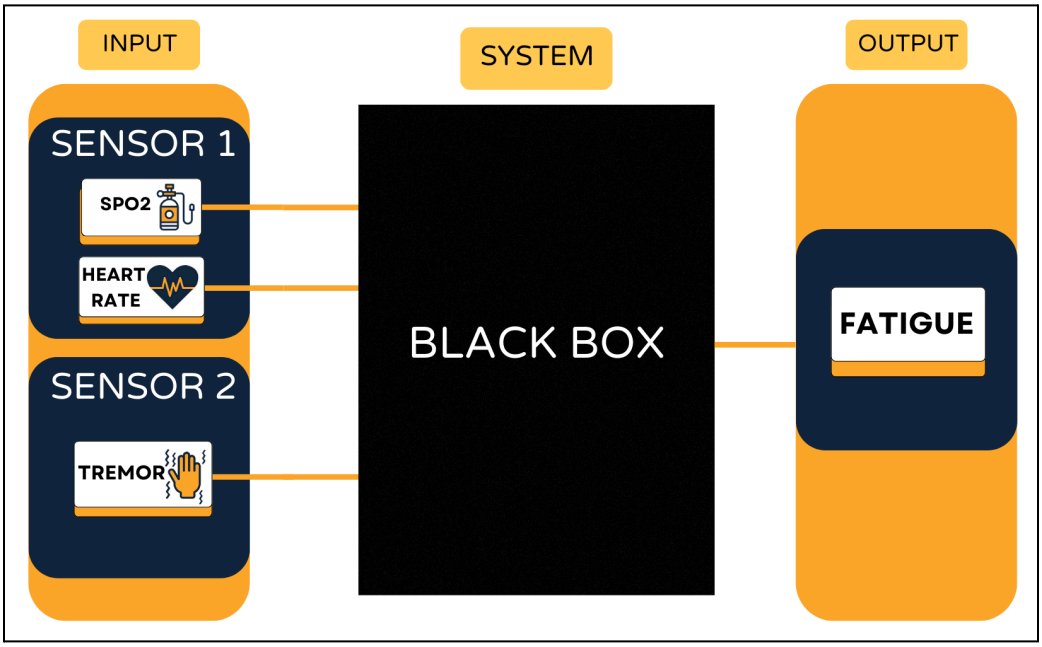


Figure 1: *Upper-Level Functional Modeling Diagram*

2.2 Detailed Functional Modeling of the System

Figure 2 shows the wearable wristband-based fatigue monitoring detailed functional modeling system diagram. First, sensors such as the ADXL335 accelerometer for tremor detection and the MAX30102 for heart rate and oxygen saturation are used. An Arduino Uno R4 WiFi gathers and transmits data to a cloud platform for processing. Before being analyzed using machine learning models against set thresholds for SpO2, heart rate, and tremor levels, the data is subjected to validation, noise filtering, and standardization. Supervisors are notified when fatigue is detected by an app that is connected to the system. This facilitates immediate action with real-time health data.

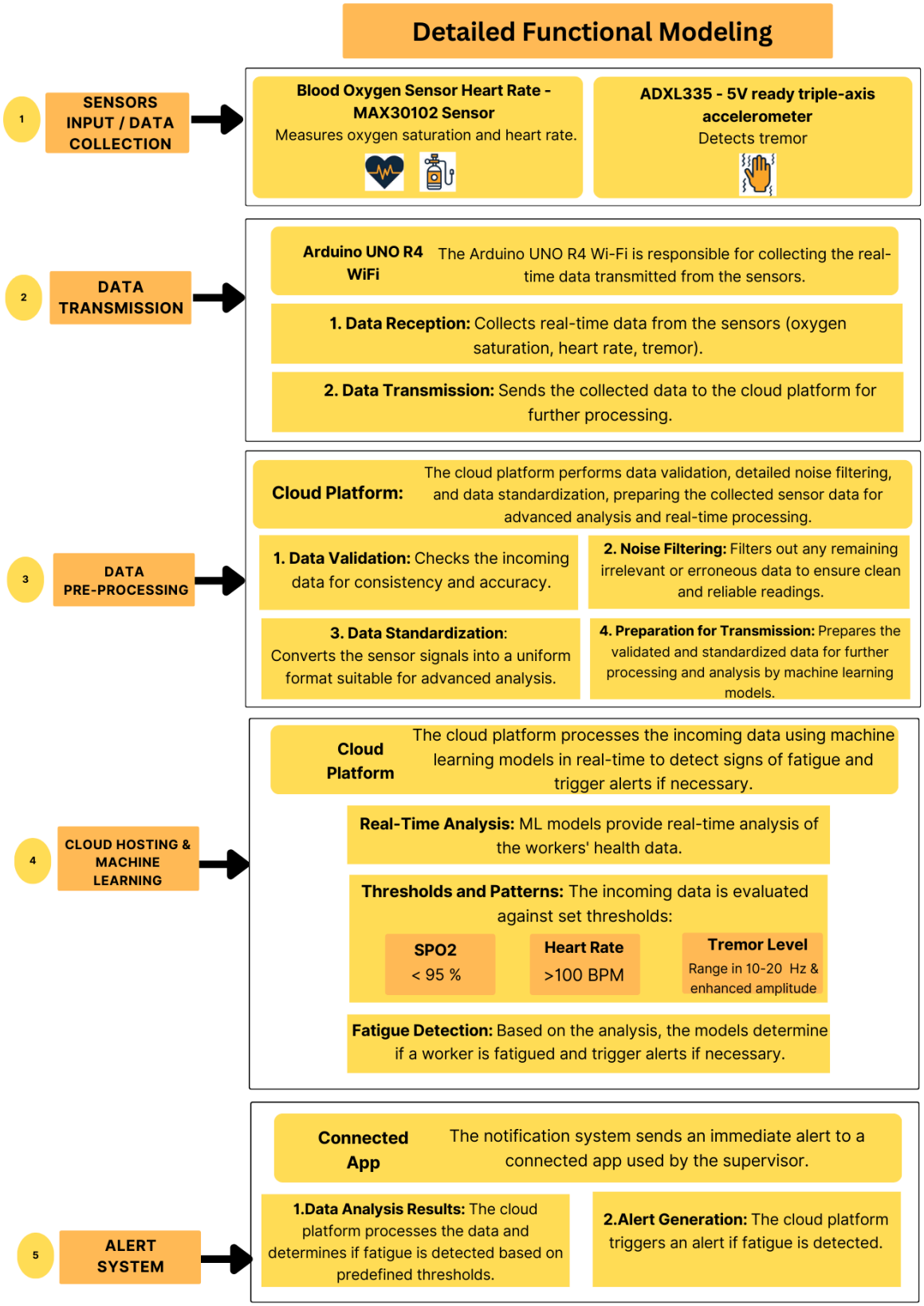


***Figure 2**: Functions Detailed Description*

Function 1: Sensors Input / Data Collection

This process includes utilizing sensors to gather information from employees. The sensors referenced are the Blood Oxygen Sensor Heart Rate Click GY MAX30102 Sensor, which can measure blood oxygen saturation and heart rate, and the ADXL335. 5V ready triple-axis accelerometer, which detects tremors or involuntary movements indicating fatigue.

Function 2: Data Transmission

The Arduino Uno R4 WiFi microcontroller functions as the central hub for collecting data from the sensors. It acquires and organizes sensor data, converting raw readings into usable values such as oxygen saturation percentages. This preparation ensures that the data is ready for further analysis and processing.

Function 3: Data Pre-Processing

After gathering the data, the cloud platform continues pre-processing it. This includes verifying the data's reliability, preparing it to eliminate any irregularities, and removing any noise to guarantee the data's precision.

Function 4: Cloud Hosting and Machine Learning

Within the cloud platform, machine learning models analyze the data to detect signs of fatigue. These models learn to recognize fatigue by examining patterns in oxygen saturation, heart rate, and tremor levels. For example, a drop in oxygen saturation below 95% may indicate fatigue in the worker. The threshold for heart rate is calculated based on the user's age [1]. A significant increase in heart rate during strenuous activity, approaching or surpassing their age-adjusted threshold, could indicate strain and potential fatigue [2]. Additionally, tremor levels, considering both amplitude and frequency in the 10-20 Hz range, may indicate muscle tiredness or neurological concerns [3].

Function 5: Alert System

The last feature is the system, which activates when the machine learning models identify fatigue according to the set thresholds. Notifications are transmitted via a linked app to the supervisor. This quick alert enables action to avoid accidents or additional health concerns caused by fatigue.

2.3 Hardware Setup

The following figures show the system's hardware configuration setup includes a 7.4V battery, an Arduino Uno R4 WiFi microcontroller, and two essential sensors: the MAX30102 and ADXL335. The battery provides power to the Arduino by linking its positive terminal to the **Vin pin** and its negative terminal to a **GND pin**, providing steady power distribution to the entire system. The MAX30102 sensor, utilized for monitoring SpO2 and heart rate, is interfaced with the Arduino through the I2C communication protocol, with the **SDA** and **SCL pins** on the sensor connected to the respective **A4 (SDA)** and **A5 (SCL) pins** on the Arduino. The ADXL335 accelerometer, which measures tremor levels along the x, y, and z axes, is interfaced with the analog **pins A0**, **A1**, and **A2** on the Arduino corresponding to each axis. The configuration features clear labeling and a color-coded, facilitating the clear identification of each pin's function and ensuring effective data transmission from the sensors to the Arduino for processing. This structured configuration facilitates continuous monitoring and reliable data transmission to the cloud for further analysis and alert generation.

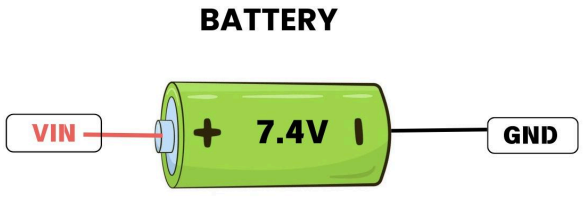


***Figure 3**: 7.4 Battery connection to Vin and GND terminals*

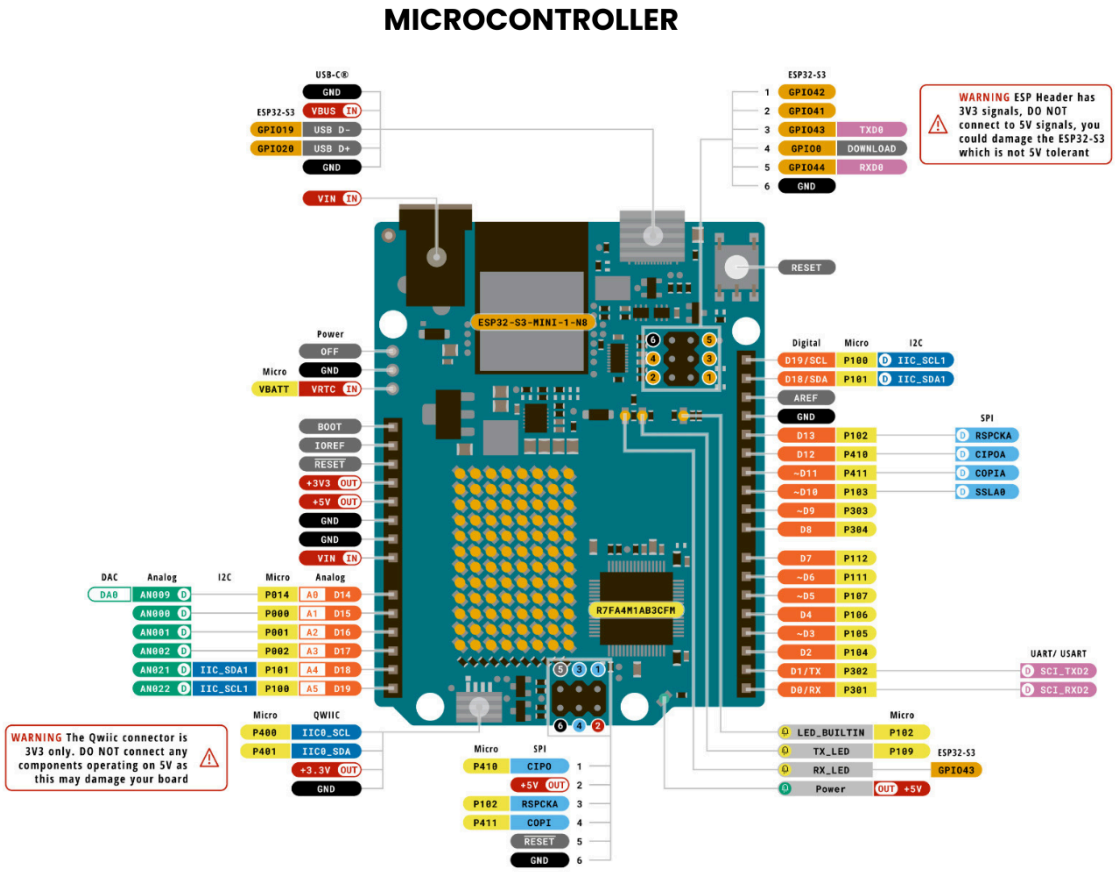


Figure 4: *Pinout diagram of Arduino Uno R4 WiFi, displaying power, analog, digital I2C ,and SPI pins.*

Figure 5: *Pin function legend with color codes*

SENSORS

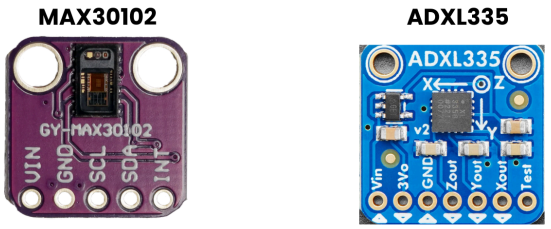


Figure 6: *MAX30102 (SpO2 and heart rate) and ADXL335 (accelerometer) sensors*

2.4 Analysis of Designed Circuits

The analysis of the designed circuit centers on the interaction of components to facilitate precise fatigue monitoring. The circuit integrates the MAX30102 sensor for measuring SpO2 and heart rate alongside the ADXL335 accelerometer for detecting tremors along the x, y, and z axes. The sensors are linked to the Arduino Uno R4 WiFi microcontroller, which functions as the central processing unit. The MAX30102 communicates with the Arduino using the I2C protocol, facilitating efficient data transmission via the SDA and SCL pins, whereas the ADXL335 uses analog connections through the A0, A1, and A2 pins for each axis. The 7.4V battery provides power to the Arduino via the Vin pin, while ground connections provide stable current flow within the circuit. Uploading sensor data to the cloud facilitates real-time processing and machine learning analysis, enabling the system to precisely and rapidly detect fatigue signs. This configuration effectively integrates power efficiency and performance, making it suitable for constant monitoring in challenging construction settings.

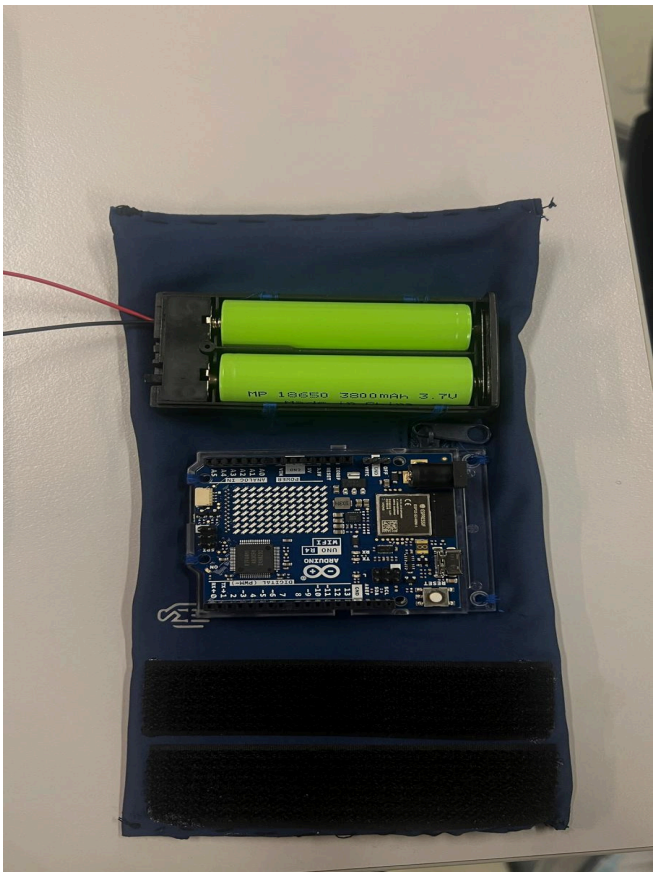
2.5 Visual Prototyping

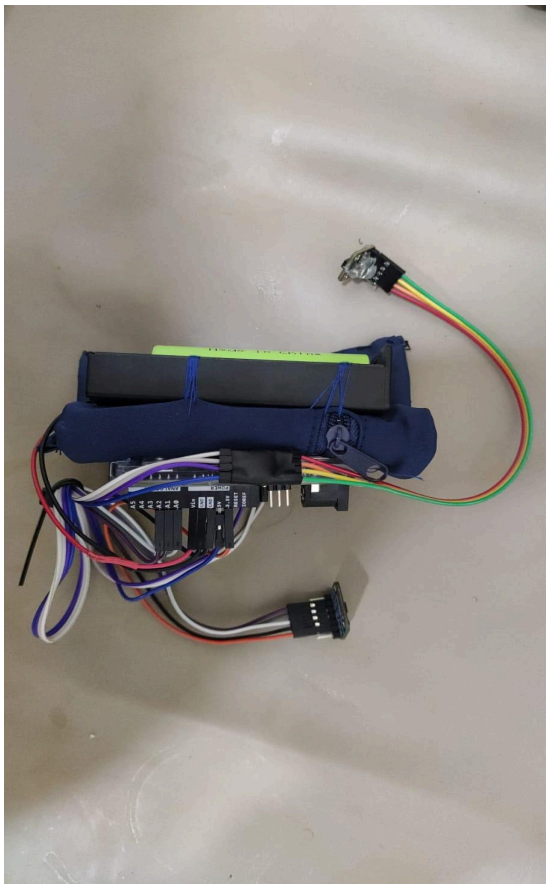
The prototype is represented in **Figure 3**. The wristband has been meticulously designed to incorporate vital electrical components, achieving a harmonious balance between functionality and wearer comfort. The Arduino Uno R4 microcontroller, functioning as the device's central processing unit, is firmly affixed to the top of the wristband. Accompanying it is a battery pack with two cylindrical cells, guaranteeing the gadget sufficient power for real-time monitoring without the need for regular recharge. The elevated positioning of the bulkier components minimizes skin contact, hence improving comfort and facilitating a sleek appearance on the wrist's outside.

The MAX30102 and ADXL335 sensors are directly stitched into the fabric on the bottom of the wristband, with their surfaces exposed to ensure skin contact. The MAX30102 sensor is an optical device specifically designed to quantify heart rate and SpO2, essential indicators for evaluating cardiovascular and respiratory health. Positioning it on the

inside side of the wristband grants direct access to blood flow, hence enhancing the precision of its measurements. The ADXL335 accelerometer, located on the inner side, continuously observes wrist movement and identifies vibrations. This sensor is especially useful for collecting data on fatigue or involuntary muscular movement, as tremor frequency may signify physical depletion or neurological reactions.

Through the process of stitching these sensors onto the underside of the wristband, they are able to retain continuous touch with the skin, which guarantees the capture of an excellent level of data. The wearable is kept discrete, functional, and comfortable thanks to this smart arrangement, which also ensures that the necessary hardware is installed without any awkwardness. A non-invasive design is the goal of the design, which will allow users to wear the wristband for extended periods of time without experiencing any discomfort while simultaneously collecting real-time health data in an effective manner.





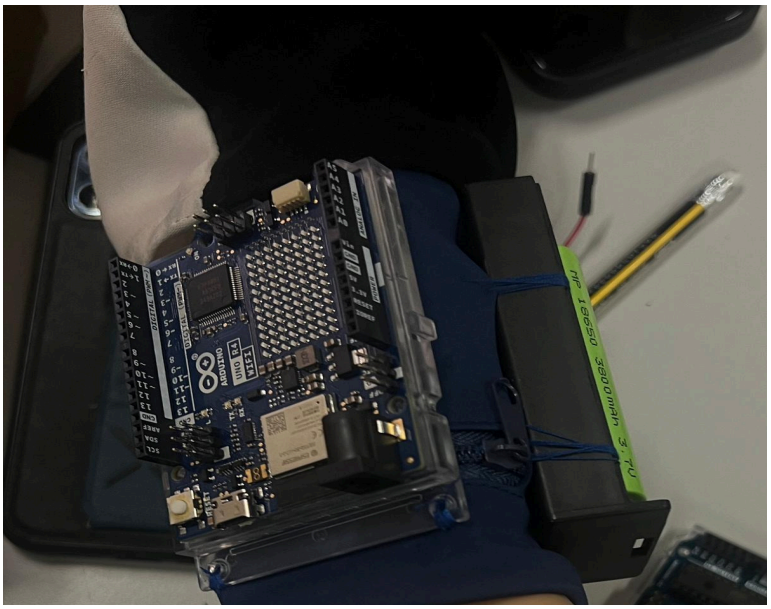


Figure 3: The Visual Prototype Design.

2.6 Arduino Code for SpO2, Heart Rate, and Tremor Sensors

A specific Arduino code was built to read data from the sensors and transfer it to ThingSpeak for remote monitoring, as shown in **Figures 4 and 5**. This code is designed to set up the sensors, verifying their proper connection via I2C communication, and to ensure each sensor is accurately configured for precise data collecting. The main objective of this configuration is to gather real-time data, including heart rate and SpO2 (oxygen saturation), with movement data reflecting tremors and to provide remote access to this information.

The Arduino code comprises numerous essential functions that oversee various operation components. A function is designated to establish the Wi-Fi connection by utilizing the specified SSID and password credentials to connect the device to a wireless network. This step is crucial for facilitating communication with ThingSpeak, the platform designated for data upload. Upon connection, the Arduino is capable of incessantly monitoring and transmitting data.

A separate function in the code facilitates data transmission to ThingSpeak via HTTP requests. This function formulates the request via the ThingSpeak API, incorporating the requisite Write API Key and channel details. The data is formatted and transmitted to designated fields inside the ThingSpeak channel, facilitating organized and systematic distant data storage.

Supplementary-specific capabilities are incorporated for the acquisition and documentation of data from each sensor. The MAX30105 sensor, utilized for heart rate and SpO2 measurements, is initialized and programmed to modify LED intensities for the best efficiency. It aggregates data in batches, which are analyzed to determine heart rate and SpO2 levels, assuring precision before transmission to ThingSpeak. The ADXL335 accelerometer, utilized for tremor monitoring, acquires raw data across three axes (x, y, and z). The data is analyzed to calculate the Root Mean Square (RMS) values, which offer a consistent assessment of tremor severity and frequency.

The primary loop in the code incessantly observes and gathers data from both sensors at predetermined intervals. Upon collecting and processing this data, it is transmitted to ThingSpeak, rendering it available for remote observation and study. The code is organized into separate functions for Wi-Fi connectivity, data transfer, and sensor data processing, resulting in an efficient and modular system that facilitates straightforward debugging and updates as required. This architecture guarantees dependable, real-time data acquisition and remote surveillance, rendering the wearable system an effective instrument for ongoing health and mobility monitoring. In the appendix section A, the remaining code will be displayed.

```
#include <WiFi.h>
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "spo2_algorithm.h"

// Wi-Fi credentials
const char* ssid = "Ibrahim Iphone";
const char* password = "12345678";

// ThingSpeak settings
unsigned long myChannelNumber = 2649311; // Your ThingSpeak Channel ID
String apiKey = "UQYE6K9L0JGBCVXY";     // Your Write API Key
const char* server = "api.thingspeak.com";
WiFiClient client;

// MAX30105 sensor object
MAX30105 particleSensor;

// Pin definitions for accelerometer
const int xInput = A0;
const int yInput = A1;
const int zInput = A2;

// Constants for tremor calculation
const int sampleSize = 10;
const int windowSize = 50;
int xWindow[windowSize];
int yWindow[windowSize];
int zWindow[windowSize];
int windowIndex = 0;

// Buffer for SpO2 and heart rate
uint32_t irBuffer[100];  // Infrared LED sensor data
uint32_t redBuffer[100]; // Red LED sensor data
int32_t spo2;            // SPO2 value
int8_t validSPO2;        // Valid SPO2 value
int32_t heartRate;       // Heart rate
int8_t validHeartRate;   // Valid heart rate

// Heart rate tracking variables
const byte RATE_SIZE = 4;
byte rates[RATE_SIZE];  // Array of heart rates
byte rateSpot = 0;
long lastBeat = 0;
float beatsPerMinute;
int beatAvg;
```

Figure 4: The Arduino Code

```
Data sent to ThingSpeak!
SPO2: 96
Smoothed Heart Rate: 30
ThingSpeak Response: HTTP/1.1 200 OK
ThingSpeak Response:
Date: Wed, 06 Nov 2024 10:28:08 GMT
ThingSpeak Response:
Content-Type: text/plain; charset=utf-8
ThingSpeak Response:
Content-Length: 4
ThingSpeak Response:
Connection: close
ThingSpeak Response:
Status: 200 OK
ThingSpeak Response:
Cache-Control: max-age=0, private, must-revalidate
ThingSpeak Response:
Access-Control-Allow-Origin: *
ThingSpeak Response:
Access-Control-Max-Age: 1800
ThingSpeak Response:
X-Request-Id: e774a9e4-cb96-458d-adb6-4f9ba7911d73
ThingSpeak Response:
Access-Control-Allow-Headers: origin, content-type, X-Requested-With
ThingSpeak Response:
Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH
ThingSpeak Response:
ETag: W/"31eb7c0492d494a5022cc02a7d7e443c"
ThingSpeak Response:
X-Frame-Options: SAMEORIGIN
ThingSpeak Response:
```

Figure 5: Output of the Arduino Code

2.7 Sensors Input and Data Collection

The Sensors Input / Data Collection section outlines the primary sensors employed in this wearable monitoring system, intended to monitor vital health metrics and identify movement anomalies. These sensors are essential for real-time monitoring of an individual's physiological condition, as they deliver continuous, comprehensive data on heart rate and SpO2, in addition to bodily movements, such as the tremor. The gathered data serves as the foundation for sophisticated analysis in subsequent phases of the system, facilitating the identification of possible indicators of exhaustion.

The initial sensor referenced is the MAX30102; this sensor uses optical methods to assess two vital health parameters: blood oxygen saturation and heart rate. Blood oxygen saturation levels indicate the efficiency of oxygen transport in the bloodstream, which is essential for detecting any respiratory problems. The heart rate data offers information into cardiovascular function, which may fluctuate with exercise level, stress, or exhaustion. Collectively, these metrics provide an extensive overview of the user's health condition and assist in detecting any abrupt alterations that may signify physical stress or exhaustion.

The second sensor in this configuration is the ADXL335 triple-axis accelerometer, a 5V-compatible device that quantifies motion along three axes (x, y, and z). This accelerometer identifies subtle movements, enabling it to perceive tremors or minor, repetitive motions in the user's body. By examining the patterns and frequency of these movements, the system can detect anomalies that may indicate Fatigue or muscular tension. For example, erratic or inconsistent movement patterns may signify the physical onset of fatigue, as the body finds it challenging to regulate movements when fatigued.

2.8 Machine Learning Integration for Fatigue Detection

The code depicted in **Figure 6** is intended to enable the analysis and identification of fatigue using health data acquired from sensors, transmitted to ThingSpeak, and processed via machine learning algorithms. The main aim of this code is to facilitate real-time monitoring and fatigue detection utilizing essential parameter indicators, including blood oxygen saturation, heart rate, and tremor. The code initiates a connection with ThingSpeak and acquires a history dataset including 1500 readings, thus establishing a solid data foundation for analysis.

The preliminary phase in data processing involves clearing the dataset by eliminating rows with missing values, hence augmenting the analysis's dependability. The purified data is further divided into three different variables: SpO2, heart rate, and tremor. A critical data quality assessment is conducted to guarantee the inclusion of only legitimate readings, such as excluding SpO2 values below 89%, as exceedingly low values are deemed unreliable and may skew the results. The code subsequently calculates the average values for each indicator to establish a foundational comprehension of the historical data and evaluate its overall attributes.

To identify potential fatigue, the code employs predefined thresholds: a SpO2 level below 95%, a heart rate over 100 beats per minute , and tremor frequency between 10 and 20 Hz. The code employs K-Fold Cross-Validation, an effective technique for model assessment, to divide the historical data into training and testing sets across several folds. This technique evaluates the performance and generalizability of the Random Forest models employed for fatigue classification, with accuracies computed and averaged across all folds to determine the models' efficiency.

The machine learning segment of the code uses ensemble learning techniques, particularly Random Forests, to develop predictive models for each statistic. Random Forests are adept at this task due to their capacity to manage non-linear interactions and deliver high accuracy in classification challenges. The models are trained on normalized historical data, with binary labels denoting whether each reading resides within

fatigue-inducing ranges. The code subsequently acquires real-time data from ThingSpeak, normalizes it with the same methodology as the historical data, and employs the trained models to identify indicators of fatigue. The fatigue analysis evaluates whether live SpO2 values are below the 95% threshold, heart rate readings surpass 100 bpm, and tremor frequency remains within the permitted range. The findings of this analysis are presented, indicating whether the existing data imply fatigue and providing alerts if required.

Name

Fatigue SPO2 / HR / TREMOR

MATLAB Code

```
% Define your ThingSpeak channel ID and read API key
channelID = 2649311;  % Your channel ID
readAPIKey = 'XD9591D3JH3PIZ15';  % Your Read API Key
fields = [1, 2, 3];  % Fields: SpO2 (Field 1), HeartRate (Field 2), and Tremor (Field 3)

% Retrieve the last 1500 historical readings available from ThingSpeak
[data, timestamps] = thingSpeakRead(channelID, 'Fields', fields, 'NumPoints', 1500, 'ReadKey', rea

% Check if historical data is empty
if isempty(data)
    disp('No historical data available from ThingSpeak.');
else
    % Remove rows with missing values before extracting data into separate variables
    data = rmmissing(data);  % Clean the entire dataset first

    % Now extract the data into separate variables
    SpO2_historical = data(:, 1);
    HeartRate_historical = data(:, 2);
    Tremor_historical = data(:, 3);

    % Check for valid historical data (exclude SpO2 < 89%)
    valid_idx_historical = SpO2_historical >= 89;
    SpO2_historical = SpO2_historical(valid_idx_historical);
    HeartRate_historical = HeartRate_historical(valid_idx_historical);
    Tremor_historical = Tremor_historical(valid_idx_historical);

    % Check for data contents and size
    disp(['Size of valid SpO2 (historical): ', num2str(size(SpO2_historical))]);
    disp(['Size of HeartRate (historical): ', num2str(size(HeartRate_historical))]);
    disp(['Size of Tremor (historical): ', num2str(size(Tremor_historical))]);

    % Calculate average for historical data and check for NaN
    SpO2_avg_historical = mean(SpO2_historical, 'omitnan');
    HeartRate_avg_historical = mean(HeartRate_historical, 'omitnan');
    Tremor_avg_historical = mean(Tremor_historical, 'omitnan');

    % Display the averages for historical data
    disp(['Average of SpO2 (historical): ', num2str(SpO2_avg_historical)]);
    disp(['Average of HeartRate (historical): ', num2str(HeartRate_avg_historical)]);
    disp(['Average of Tremor (historical): ', num2str(Tremor_avg_historical)]);

    % Define fatigue detection thresholds
    SpO2_threshold = 95;  % Fatigue if SpO2 < 95%
    HeartRate_threshold = 100;  % Fatigue if HeartRate > 100 bpm
    Tremor_min_threshold = 10;  % Minimum threshold for Tremor
    Tremor_max_threshold = 20;  % Maximum threshold for Tremor
```

Figure 6: Machine Learning Code for Fatigue Detection

2.9 Alert System (App and Cloud Integration)

The app's user interface was developed using Thunkable and integrated with Xcode migrator tools, leveraging Apple ID and Team ID configurations in Thunkable to ensure compatibility with iOS. The interface was designed to provide an intuitive and user-friendly experience, allowing users to access various functionalities seamlessly. The main features include a login screen, real-time health monitoring, calorie calculation, and data visualization.

Upon logging in, users are directed to a dashboard to monitor heart rate, SpO_2, and tremor data in real-time. The dashboard displays these metrics through easy-to-read gauges, providing a quick overview of vital signs that are essential for fatigue monitoring. Additionally, a calorie calculator tool allows users to estimate calories burned based on their heart rate, weight, age, and duration of activity. This feature helps users understand the relationship between their physical activity levels and energy expenditure.

The app also includes a data analysis section where users can view graphical representations of their SpO_2, heart rate, and tremor data over time. These graphs provide valuable insights into physiological trends and are particularly useful for identifying fatigue patterns. For ease of use, a "Download all Data" button allows users to save their data for further analysis or sharing with health professionals. Overall, the app UI is structured to support real-time monitoring, user engagement, and data accessibility, making it a comprehensive tool for tracking and managing fatigue.

2.10 Mechanical Structures

The mechanical design of the device prioritizes effective sensor placement to ensure accurate data collection. Proper sensor positioning is essential to capture reliable measurements for SpO_2, heart rate, and tremor, which are crucial for monitoring fatigue in real-time. Each sensor is strategically placed to optimize contact with the skin, reducing the likelihood of data interference or signal loss. This careful placement enhances the device's ability to consistently track physiological parameters accurately, even in demanding environments like construction sites.

Another key aspect of the design is its adjustable fit, which allows the device to comfortably accommodate users of different wrist sizes. An adjustable band ensures that the sensors maintain close contact with the skin, improving data accuracy and user comfort. This feature is particularly important for prolonged use, as it minimizes discomfort and enables the device to be worn securely throughout a work shift. The combination of optimal sensor placement and an adjustable fit creates a user-friendly and efficient device suited to the needs of construction workers.

3. Functional Prototyping, Testing, Troubleshooting

3.1 Functional Prototyping: Hardware and Software

During the functional prototyping phase, multiple troubleshooting steps were taken to ensure smooth hardware and software integration. One of the initial challenges was data retrieval, where the application failed to retrieve data from ThingSpeak due to an incorrect API key configuration. This issue was resolved by double-checking the ThingSpeak channel settings and updating the API keys for accurate data retrieval.

Another issue involved handling missing data, where certain sensor readings were absent, leading to errors in analysis. To address this, we implemented a data cleaning process using MATLAB's missing () function, effectively removing any missing values before analysis.

To enhance the accuracy of machine learning predictions, especially during K-Fold Cross-Validation, adjustments were made to the model parameters, and different thresholds were tested. Additionally, data normalization was applied to handle varying ranges in heart rate data, resulting in improved classification accuracy.

In terms of UI and display, the WebView component's display was misaligned, with gauges appearing too small on mobile devices. This was corrected using HTML and CSS adjustments to ensure the data was centered and the gauges were adequately sized for mobile viewing.

Lastly, real-time updates presented a challenge, as the app occasionally lagged while

fetching data from ThingSpeak. To balance real-time performance with system efficiency, data-fetching intervals were optimized, and the request frequency was minimized to ensure smooth operation.

3.2 Testing of Sensors

The testing phase for sensors focused on validating the reliability, accuracy, and seamless integration of SpO_2, heart rate, and tremor measurements, which are essential for real-time fatigue monitoring. To achieve this, several testing goals were established.

First, we assessed the accuracy of data retrieval by verifying the app's ability to reliably pull the latest sensor data from ThingSpeak, ensuring consistent and accurate readings for key metrics such as SpO_2, heart rate, and tremor. This involved testing the retrieval of multiple data points to confirm that the data was both up-to-date and correctly displayed within the app.

Additionally, we evaluated the machine learning predictions generated by our fatigue detection model, using predefined thresholds for each metric to assess the model's effectiveness in identifying fatigue states. By running the model with both historical and live sensor data, we were able to fine-tune its predictive accuracy, ensuring that it could accurately detect fatigue indicators based on real-time data input.

The app's performance was another key focus, as we measured its responsiveness when fetching real-time data and updating the user interface, particularly the gauge displays and charts. This step was crucial for ensuring that the app could maintain smooth operation without lag, providing users with reliable, immediate feedback.

To ensure a seamless user experience, we tested the app on multiple devices, including both Android and iOS platforms, verifying that it maintained consistent performance and a user-friendly layout across all devices. Finally, cross-platform compatibility and error-handling tests were conducted to confirm that the app could appropriately handle missing data and display clear error messages when necessary. These tests confirmed that

the sensor data integration was robust, laying a solid foundation for reliable real-time fatigue monitoring.

3.3 Troubleshooting

This project involved tackling two significant obstacles, each of which taught us valuable lessons in determination and problem-solving. The first major challenge we faced was with the code itself. Initially, we encountered an error where the heart rate, which is a key parameter we were measuring for the construction workers. This was frustrating, as heart rate data is essential for tracking worker health and safety, which is a central part of our project. To address this, we requested assistance from professors and teaching assistants, dedicating time to fully understand the root of the problem. With their support, we eventually corrected the error, ensuring that we started seeing some results.

However, just as we thought we had resolved the issues, a second challenge appeared. Although the code was running, the output values were unstable, with the numbers fluctuating and sometimes producing incorrect readings. We realized that this inconsistency could be due to a range of errors, from minor coding bugs to data variability. After some analysis, we decided to modify the code to filter out unreliable data by setting specific thresholds. This allowed us to focus only on the values that met our reliability criteria, improving the accuracy of our results.

These barriers, while challenging, were valuable learning experiences. Not only did they help us develop technical skills in coding and data analysis, but they also strengthened our critical thinking abilities. By working through these issues, we learned the importance of patience, resourcefulness, and teamwork in problem-solving.

3.4 Experimental Results

Created at	Entry id	Spo2	HR	Tremor
2024-11-05T05:26:10+03:00	9029	99	73	4.13
2024-11-05T05:26:17+03:00	9030	99	79	4.17
2024-11-05T05:26:24+03:00	9031	100	78	4.14
2024-11-05T05:26:31+03:00	9032	100	75	4.11
2024-11-05T05:26:37+03:00	9033	100	79	4.08
2024-11-05T05:26:44+03:00	9034	99	84	4.08
2024-11-05T05:26:51+03:00	9035	99	87	4.14
2024-11-05T05:26:57+03:00	9036	99	82	4.13
2024-11-05T05:27:04+03:00	9037	98	82	4.10
2024-11-05T05:27:11+03:00	9038	99	81	4.16
2024-11-05T05:27:18+03:00	9039	96	82	4.05
2024-11-05T05:27:24+03:00	9040	99	79	4.00
2024-11-05T05:27:31+03:00	9041	100	88	4.03
2024-11-05T05:27:38+03:00	9042	100	87	4.16
2024-11-05T05:27:45+03:00	9043	89	91	4.02
2024-11-05T05:27:51+03:00	9044	99	92	4.13
2024-11-05T05:27:58+03:00	9045	100	91	4.14
2024-11-05T05:28:05+03:00	9046	99	89	4.20
2024-11-05T05:28:12+03:00	9047	0	89	4.00
2024-11-05T05:28:18+03:00	9048	89	84	4.05
2024-11-05T05:28:25+03:00	9049	98	81	4.08
2024-11-05T05:28:32+03:00	9050	63	82	4.05
2024-11-05T05:28:39+03:00	9051	99	87	4.00
2024-11-05T05:28:45+03:00	9052	98	89	4.03

Figure 7:The readings at the first run

Figure 8: The graph of Spo2 output

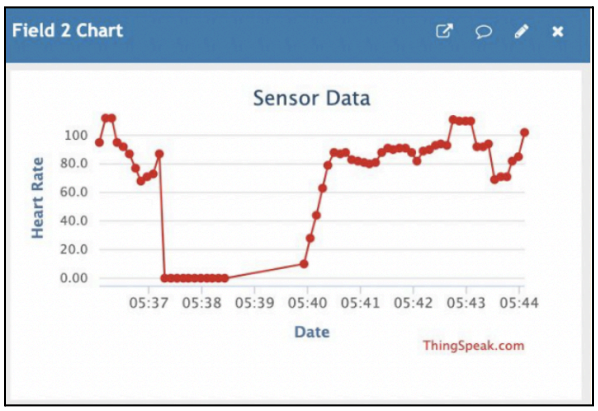


Figure 9: The graph of Heart Rate output

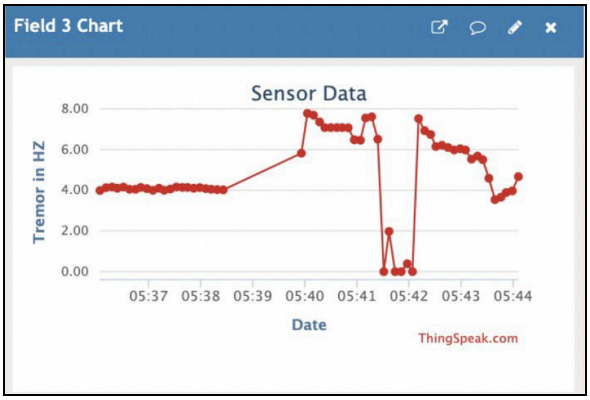


Figure 10: The graph of Tremor output

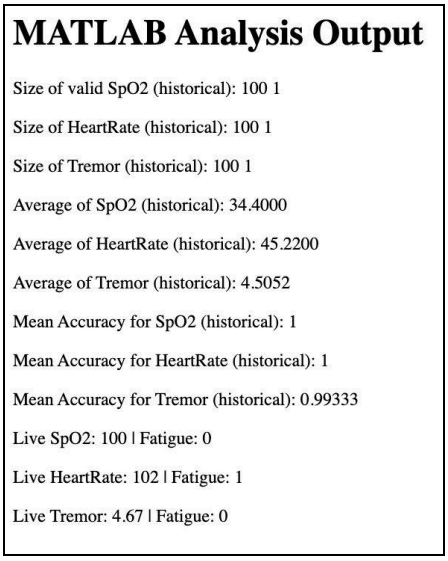


Figure 11: The ML MATLAB analysis output

Figure 12: The output of the application platform

4. Link Between Projects and ECEN Courses

During our academic journey at Texas A&M University Qatar, we have gained a comprehensive understanding of engineering knowledge and practical skills through a curriculum designed to prepare students to tackle real-world problems. We completed

numerous courses that aided us with the foundational knowledge and skills required to efficiently carry out our senior design project of a non-invasive multi-sensor wearable wristband for fatigue prevention. The main courses that have significantly contributed to our project are the following: ECEN 210, ECEN 214, ECEN 446, and ECEN 449.

4.1 ECEN Course 1

ECEN 210, a Computer Programming and Algorithms course, introduced us to C/C++ during our sophomore year. This course focused on the fundamentals of the C language, such as modular programming and functions; arrays and matrices; pointers and strings; simple data structures; searching, sorting, and numerical algorithms [4]. The project relies heavily on the C language as a base for the Arduino Integrated Development Environment (IDE) we are employing and programming the sensors to allow us to collect real-time data. In addition, ECEN 214 is an Electrical Circuit Theory course that provided us with a comprehensive understanding and covered the essential topics related to electrical circuits, such as circuit laws, network reduction, nodal analysis, and mesh analysis, energy storage elements [4]. The practical skills learned in ECEN 214 will help us design and implement circuits that seamlessly integrate and function successfully between our sensors and Arduino board.

4.2 ECEN Course 2

The ECEN 446 Information Theory, Inference and Learning Algorithms course provides essential concepts and techniques that are invaluable for implementing the machine learning algorithm in our safety wristband project. Specifically, knowledge gained from neural networks and support vector machines will be pivotal in developing the algorithm to assess the health status of construction workers using sensor data. Clustering methods will aid in categorizing different fatigue levels by identifying similar patterns in oxygen saturation levels, heart rate, and tremor measurements. Maximum likelihood estimation will refine model parameters to optimize health status predictions based on collected data. Monte Carlo methods and important sampling techniques will enhance probabilistic analysis and improve model performance across diverse conditions [4]. Additionally, applying data compression strategies will be critical for efficiently managing and

transmitting large volumes of sensor data to the Arduino microprocessor via wireless connectivity. By integrating these advanced machine learning and statistical inference techniques from the course, our project aims to enhance workplace safety and productivity by accurately monitoring and assessing construction workers' health status in real-time.

4.2 ECEN Course 3

The ECEN 449 Microprocessor Systems Design course played a crucial role in implementing key aspects of our safety wristband project, with a strong focus on integrating microprocessor and Arduino functionalities. The course's instruction on microprocessors and single-board computer hardware assisted in designing an effective interface between the sensors and the Arduino microprocessor, ensuring efficient data collection from oxygen saturation levels, heart rate, and tremor sensors. Understanding chip-select equations for memory board design and mastering interfacing protocols like serial and parallel I/O streamlined the integration of these sensors with Arduino. Additionally, knowledge of ROM, static, and dynamic RAM circuits was pivotal for optimizing data storage and processing capabilities on the Arduino, facilitating real-time monitoring of construction workers' health status. The proficiency in assembly language programming gained through the course enabled us to efficiently implement the machine-learning algorithm on the Arduino, ensuring precise assessment of fatigue levels. By applying these advanced concepts and techniques, our project successfully developed a robust safety wristband that enhances workplace safety by continuously monitoring and tracking fatigue levels among construction workers.

5. Progress Compared to the Proposed Timeline

Task Completed	ECEN 404 : Senior Projec Timeline Chart												
Task in Progress	Hemaya : Non-invasive multi-sensor wearable wristband for fatigue prevention												
Task Remaining	August	September				October			November				December
Weeks	4	1	2	3	4	1	2	4	1	2	3	4	1
Tasks													
Discussion Meetings													
Presentation 1													
Demo Day 1													
Discussion Meetings													
Presentation 2													
Update Website													
Discussion Meetings													
Presentation 3													
Final Report													
Demo Day (Working system & Final Presentation)													
Mentor Evaluation													
Peer Evaluation													

Figure 13: The progress timeline of our project

The entire schedule for the Fall 2024 semester is shown in the table above. It contains all of the assignments and tasks that the course instructor has given, along with the week in which the deadlines were established. The table helps us by providing instructions on the tasks we must do by the deadline to move on to the next phase. The blue blocks show the completed tasks, the green blocks in the table show each assignment that is currently being worked on, and the red blocks show the remaining tasks. As demonstrated in the lab sessions, the project's progress and the suggested time frame are nearly comparable. The only little modification needed is to make sure the fatigue monitor is fully functional when getting real-time data from the channel.

6. Discussion and Future Recommendations

Discussion

The Hemaya non-invasive, multi-sensor wristband represents significant progress in fatigue prevention technology. By integrating multiple sensors that monitor oxygen saturation, heart rate variability, and tremor, Hemaya provides a comprehensive tool for assessing and managing fatigue. Its real-time data collection facilitates immediate feedback, enhancing users' awareness of their physical state and enabling timely fatigue management strategies. The non-invasive nature of the wristband ensures minimal disruption to users' daily activities. Furthermore, integrating a mobile application for personalized insights and recommendations boosts user engagement and promotes healthier habits.

However, several challenges must be addressed for optimal functionality. Hemaya relies heavily on sensor precision and data interpretation algorithms for reliability. Individual physiological variations and environmental conditions can impact data accuracy. Moreover, consistent and long-term user engagement is vital for collecting meaningful data, which requires strategies to encourage users to wear the device regularly.

6.1 Discussion of Project Successes and Challenges

Innovative Technology Integration: One major success of the Hemaya project was the seamless integration of advanced sensors into a compact, non-invasive wristband. By incorporating oxygen saturation, heart rate variability, and tremor, the device provides real-time physiological data, offering valuable insights into users' health and fatigue levels.

User-Centric Design: The project prioritized user comfort and practicality, resulting in a lightweight, adjustable wristband that was well-received during testing. Participants highlighted the comfort of wearing the device for extended periods, underscoring the effectiveness of the user-focused design.

Positive User Feedback: User testing produced encouraging results, with participants praising the device's functionality and the accompanying mobile app. Personalized recommendations and practical insights were particularly appreciated, showcasing Hemaya's potential to empower

users in their health and wellness journey.

Challenges:

- Sensor Accuracy and Calibration: Maintaining the accuracy and consistency of sensor readings proved challenging due to individual physiological differences and varying environmental conditions. Regular calibration and future advancements in sensor technology will be essential for improvement.
- User Engagement: Sustaining long-term user commitment was another challenge. While initial feedback was positive, maintaining daily usage requires strategies such as gamification and incentives.
- Data Privacy Concerns: Addressing privacy concerns regarding the collection and management of sensitive health data was vital. Establishing trust through transparency and robust data security measures remains a priority.

6.2 Verification

Verification is essential for ensuring that Hemaya meets its design and operational goals. The process includes comprehensive sensor calibration, which is conducted under controlled conditions to ensure accuracy in oxygen saturation, heart rate, and tremor. Data accuracy is verified by comparing sensor outputs with gold-standard medical equipment.

Key performance indicators (KPIs) for evaluation include:

- Sensor accuracy and precision
- User feedback and satisfaction ratings
- Battery performance over time
- Frequency and severity of any adverse events during trials

The results of these verification activities will guide any necessary refinements and inform decisions for product rollout and future updates.

6.3 Future Recommendations

Enhancing Sensor Technology: Future research should focus on improving sensor accuracy and reliability, particularly in varied conditions. Collaborations with sensor manufacturers and biomedical engineers could yield significant advancements.

Algorithm Refinement: Leveraging machine learning algorithms to better analyze user data and enhance fatigue prediction models will be crucial. This can lead to more tailored and accurate recommendations.

User Education and Training: Educating users on the importance of managing fatigue and the correct use of the wristband can boost engagement. Training materials should emphasize the benefits of consistent device usage and data interpretation.

6.4 Improvements and Optimizations

Advanced Calibration Techniques: Implementing adaptive calibration methods that adjust to individual user baselines can enhance sensor accuracy. **Sensor Fusion Algorithms:** Developing algorithms that combine data from multiple sensors for a comprehensive analysis can improve fatigue detection. **Energy Efficiency:** Utilizing low-power sensors and innovative battery technology can extend battery life. **Adaptive Power Management:** Intelligent power-saving modes can optimize energy consumption based on user activity. **Gamification Features:** Adding gamification elements to the mobile app, such as challenges and rewards, can encourage consistent use. **Personalized Notifications:** Tailored alerts and reminders based on user behavior can increase engagement.

7. Conclusion

The development of the Hemaya multi-sensor wearable wristband marks a significant step forward in health technology, particularly in the prevention and management of fatigue. By combining cutting-edge sensors with an intuitive, user-centric design, Hemaya empowers users with real-time insights into their health, promoting timely interventions and supporting a proactive approach to fatigue management. User feedback during testing highlighted its potential to improve health awareness and facilitate healthier lifestyle decisions. This aligns with current trends in health technology that prioritize data-driven wellness and personalized solutions.

To maintain its competitive edge and relevance in the market, Hemaya must continuously address challenges such as sensor accuracy, user engagement, and data privacy. Iterative improvements based on user feedback, technological advances, and interdisciplinary collaboration will enhance its functionality and user experience. By focusing on these areas, Hemaya can become a leader in wearable health solutions, offering a reliable tool that improves safety and productivity, especially in high-risk environments like construction sites.

The journey of Hemaya demonstrates a commitment to innovative problem-solving and user-focused development. With ongoing research and development, the potential of this device extends beyond fatigue management to other areas of health monitoring and prevention. By empowering users to take control of their well-being and fostering greater awareness of fatigue, Hemaya can contribute to building a healthier, more resilient society.

In summary, Hemaya showcases the transformative power of wearable technology in health monitoring. It has the potential to significantly improve quality of life by providing accessible, non-invasive monitoring that supports users in achieving better health outcomes. With strategic advancements in sensor accuracy, user engagement strategies, and data protection, Hemaya is well-positioned to make a meaningful impact in health technology.

8. References

[1] R. Health, “3 Easy Steps to Find Your Target Heart Rate Range,” Feb. 28, 2017. https://riverview.org/blog/fitness-2/3-easy-steps-to-find-your-target-heart-rate-range/#:~:text=Oncechanges_in_tremor_caused_by_physical_efforts_of_different_volume_and_intensitychanges_in_tremor_caused_by_physical_efforts_of_different_volume_and_intensity (accessed Jul. 04, 2024).

[2] “Calculators: Heart online,” Calculators | Heart Online, https://heartonline.org.au/resources/calculators/target-heart-rate-calculator (accessed Jul. 4, 2024).

[3] (PDF) fatigue-induced changes in tremor caused by physical efforts of different volume and intensity, https://www.researchgate.net/publication/268295263_Fatigue-induced_changes_in_tremor_caused_by_physical_efforts_of_different_volume_and_intensity (accessed Jul. 4, 2024).

[4] “Course descriptions,” Texas A&M University at Qatar, https://www.qatar.tamu.edu/academics/ecen/academics/course-descriptions (accessed Jul. 3, 2024).

9. Appendices

Appendix A: Arduino Code for SpO2, Heart Rate, and Tremor Sensors

```
#include <WiFi.h>
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "spo2_algorithm.h"
// Wi-Fi credentials
const char* ssid = "Ibrahim Iphone";
const char* password = "12345678";
// ThingSpeak settings
unsigned long myChannelNumber = 2649311; // Your ThingSpeak Channel
ID
String apiKey = "UQYE6K9L0JGBCVXY";     // Your Write API Key
const char* server = "api.thingspeak.com";
WiFiClient client;
// MAX30105 sensor object
MAX30105 particleSensor;
// Pin definitions for accelerometer
const int xInput = A0;
const int yInput = A1;
const int zInput = A2;
// Constants for tremor calculation
const int sampleSize = 10;
const int windowSize = 50;
int xWindow[windowSize];
int yWindow[windowSize];
int zWindow[windowSize];
int windowIndex = 0;
// Buffer for SpO2 and heart rate
uint32_t irBuffer[100];  // Infrared LED sensor data
uint32_t redBuffer[100]; // Red LED sensor data
int32_t spo2;            // SPO2 value
int8_t validSPO2;        // Valid SPO2 value
int32_t heartRate;       // Heart rate
int8_t validHeartRate;   // Valid heart rate
// Heart rate tracking variables
const byte RATE_SIZE = 4;
byte rates[RATE_SIZE];  // Array of heart rates
byte rateSpot = 0;
long lastBeat = 0;
float beatsPerMinute;
int beatAvg;
unsigned long lastReadingTime = 0;
```

```
      const unsigned long READ_INTERVAL = 3000; // 3 seconds
      void setup() {
       Serial.begin(115200);
       // Initialize MAX30105 sensor
       //Serial.println("Initializing MAX30105...");
       if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
         Serial.println("MAX30105 was not found.");
         while (1); // Halt if sensor not found
       }
       Serial.println("MAX30105 Initialized.");
        particleSensor.setup();
       particleSensor.setPulseAmplitudeRed(0x0A);  // Lower intensity for
red LED
       particleSensor.setPulseAmplitudeGreen(0);   // Turn off green LED
       // Setup Wi-Fi connection
       Serial.println("Connecting to WiFi...");
       WiFi.begin(ssid, password);
       while (WiFi.status() != WL_CONNECTED) {
         delay(1000);
         Serial.println("Connecting...");
       }
       Serial.println("Connected to WiFi");
       // Print IP Address
       Serial.print("IP Address: ");
       Serial.println(WiFi.localIP());
      }
      void loop() {
       unsigned long currentTime = millis();
        // Get tremor data
       Serial.println("Reading tremor data...");
       int xRaw = ReadAxis(xInput);
       int yRaw = ReadAxis(yInput);
       int zRaw = ReadAxis(zInput);
       xWindow[windowIndex] = xRaw;
       yWindow[windowIndex] = yRaw;
       zWindow[windowIndex] = zRaw;
       windowIndex = (windowIndex + 1) % windowSize;
       float tremorX = calculateRMS(xWindow, windowSize);
       float tremorY = calculateRMS(yWindow, windowSize);
       float tremorZ = calculateRMS(zWindow, windowSize);
        //Serial.print("Tremor X: "); Serial.println(tremorX);
      // Serial.print("Tremor Y: "); Serial.println(tremorY);
       //Serial.print("Tremor Z: "); Serial.println(tremorZ);
       // Read heart rate and SpO2 every 5 seconds
       if (currentTime - lastReadingTime >= READ_INTERVAL) {
         lastReadingTime = currentTime;
         long irValue;
```

```
        rateSpot = 0;
        // Fill the buffer for SpO2 and heart rate calculations
        for (int i = 0; i < 100; i++) {
          while (particleSensor.available() == false) {
            particleSensor.check(); // Wait until the sensor has data
          }
          redBuffer[i] = particleSensor.getRed();
          irBuffer[i] = particleSensor.getIR();
          particleSensor.nextSample();  // Move to the next sample
        }
        // Calculate heart rate and SpO2 using the algorithm
        maxim_heart_rate_and_oxygen_saturation(irBuffer, 100, redBuffer,
&spo2, &validSPO2, &heartRate, &validHeartRate);
        // Heart rate beat detection with debouncing (min 250 ms between
beats)
        for (int i = 0; i < 100; i++) {
          irValue = irBuffer[i];
          if (checkForBeat(irValue)) {
            long delta = millis() - lastBeat;
            if (delta > 250) {  // Debounce: only consider a beat if more
than 250ms have passed
              lastBeat = millis();
              beatsPerMinute = 60 / (delta / 1000.0);
              if (beatsPerMinute < 255 && beatsPerMinute > 30) {  // Only
accept reasonable heart rate values
                rates[rateSpot++] = (byte)beatsPerMinute;
                if (rateSpot == RATE_SIZE) rateSpot = 0;
                // Calculate average heart rate
                beatAvg = 0;
                for (byte x = 0; x < RATE_SIZE; x++) {
                  beatAvg += rates[x];
                }
                beatAvg /= RATE_SIZE;
              }
            }
          }
        }
        // Check if heart rate and SpO2 are valid
        if (validHeartRate) {
          beatsPerMinute = heartRate;
        }
        if (validSPO2) {
          Serial.print("SPO2: ");
          Serial.println(spo2);
        } else {
          Serial.println("Invalid SPO2 reading.");
        }
```

```
      // Print heart rate and SpO2
      //Serial.print("Heart Rate: "); Serial.println(beatsPerMinute);
      //Serial.print("Average Heart Rate: "); Serial.println(beatAvg);
      //Serial.print("SpO2: "); Serial.println(spo2);
      // Send data to ThingSpeak using HTTP requests
      if (WiFi.status() == WL_CONNECTED) {
        if (client.connect(server, 80)) {
          String url = "/update?api_key=" + apiKey;
          url += "&field1=" + String(spo2);  // Field 1: SpO2
          url += "&field2=" + String(beatsPerMinute);  // Field 2:
Heart Rate Average (BPM)
          url += "&field3=" + String(tremorX);  // Field 3: Tremor X
          url += "&field4=" + String(tremorY);  // Field 4: Tremor Y
          url += "&field5=" + String(tremorZ);  // Field 5: Tremor Z
          client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                       "Host: " + server + "\r\n" +
                       "Connection: close\r\n\r\n");
          // Check the response
          while (client.available() == 0) {
            if (!client.connected()) {
              Serial.println("Connection failed.");
              return;
            }
          }
          // Print the response from ThingSpeak
          while (client.available()) {
            String line = client.readStringUntil('\r');
            Serial.print("ThingSpeak Response: ");
            Serial.println(line);
          }
          client.stop();
          Serial.println("Data sent to ThingSpeak!");
        }
      } else {
        Serial.println("WiFi not connected!");
      }
     }
    }
    // Helper functions
    int ReadAxis(int axisPin) {
     long reading = 0;
     analogRead(axisPin);
     delay(1);
     for (int i = 0; i < sampleSize; i++) {
       reading += analogRead(axisPin);
     }
     return reading / sampleSize;
```

```
}
float calculateRMS(int arr[], int size) {
 long sum = 0;
 for (int i = 0; i < size; i++) {
   sum += sq(arr[i]);
 }
 return sqrt(sum / (float)size);
}
```

Appendix B: Machine Learning Integration for Fatigue Detection

```
% Define your ThingSpeak channel ID and read API key
channelID = 2649311;  % Your channel ID
readAPIKey = 'XD9591D3JH3PIZ15';  % Your Read API Key
fields = [1, 2, 3];  % Fields: SpO2 (Field 1), HeartRate (Field 2), and
Tremor (Field 3)

% Retrieve the last 1500 historical readings available from ThingSpeak
[data, timestamps] = thingSpeakRead(channelID, 'Fields', fields,
'NumPoints', 1500, 'ReadKey', readAPIKey);

% Check if historical data is empty
if isempty(data)
   disp('No historical data available from ThingSpeak.');
else
   % Remove rows with missing values before extracting data into
separate variables
   data = rmmissing(data);  % Clean the entire dataset first

   % Now extract the data into separate variables
   SpO2_historical = data(:, 1);
   HeartRate_historical = data(:, 2);
   Tremor_historical = data(:, 3);

   % Check for valid historical data (exclude SpO2 < 89%)
   valid_idx_historical = SpO2_historical >= 89;
   SpO2_historical = SpO2_historical(valid_idx_historical);
   HeartRate_historical = HeartRate_historical(valid_idx_historical);
   Tremor_historical = Tremor_historical(valid_idx_historical);

   % Check for data contents and size
   disp(['Size of valid SpO2 (historical): ',
num2str(size(SpO2_historical))]);
```

```
   disp(['Size of HeartRate (historical): ',
num2str(size(HeartRate_historical))]);
   disp(['Size of Tremor (historical): ',
num2str(size(Tremor_historical))]);

   % Calculate average for historical data and check for NaN
   SpO2_avg_historical = mean(SpO2_historical, 'omitnan');
   HeartRate_avg_historical = mean(HeartRate_historical, 'omitnan');
   Tremor_avg_historical = mean(Tremor_historical, 'omitnan');

   % Display the averages for historical data
   disp(['Average of SpO2 (historical): ',
num2str(SpO2_avg_historical)]);
   disp(['Average of HeartRate (historical): ',
num2str(HeartRate_avg_historical)]);
   disp(['Average of Tremor (historical): ',
num2str(Tremor_avg_historical)]);

   % Define fatigue detection thresholds
   SpO2_threshold = 95;  % Fatigue if SpO2 < 95%
   HeartRate_threshold = 100;  % Fatigue if HeartRate > 100 bpm
   Tremor_min_threshold = 10;  % Minimum threshold for Tremor
   Tremor_max_threshold = 20;  % Maximum threshold for Tremor

   % Create binary labels for fatigue based on the thresholds
   SpO2_labels_historical = SpO2_historical < SpO2_threshold;
   HeartRate_labels_historical = HeartRate_historical >
HeartRate_threshold;
   Tremor_labels_historical = (Tremor_historical > Tremor_min_threshold)
& (Tremor_historical < Tremor_max_threshold);

   % Normalize the historical data
   SpO2_historical_norm = (SpO2_historical - min(SpO2_historical)) /
(max(SpO2_historical) - min(SpO2_historical));
   HeartRate_historical_norm = (HeartRate_historical -
min(HeartRate_historical)) / (max(HeartRate_historical) -
min(HeartRate_historical));
   Tremor_historical_norm = (Tremor_historical - min(Tremor_historical))
/ (max(Tremor_historical) - min(Tremor_historical));

   % K-Fold Cross-Validation for historical data
   k = 5;  % Number of folds
   cv_SpO2_historical = cvpartition(numel(SpO2_historical_norm),
'KFold', k);
   cv_HeartRate_historical =
cvpartition(numel(HeartRate_historical_norm), 'KFold', k);
```

```
   cv_Tremor_historical = cvpartition(numel(Tremor_historical_norm),
'KFold', k);
   accuracies_SpO2_historical = zeros(k, 1);
   accuracies_HeartRate_historical = zeros(k, 1);
   accuracies_Tremor_historical = zeros(k, 1);

   % Train and test model using K-Fold Cross-Validation for historical
data
   for i = 1:k
       % Split the data into training and testing sets for SpO2
       XTrain_SpO2_historical =
SpO2_historical_norm(training(cv_SpO2_historical, i), :);
       YTrain_SpO2_historical =
SpO2_labels_historical(training(cv_SpO2_historical, i));
       XTest_SpO2_historical =
SpO2_historical_norm(test(cv_SpO2_historical, i), :);
       YTest_SpO2_historical =
SpO2_labels_historical(test(cv_SpO2_historical, i));

       % Train the model using Random Forest for SpO2
       SpO2_model_historical = fitcensemble(XTrain_SpO2_historical,
YTrain_SpO2_historical, 'Method', 'Bag');

       % Evaluate the model
       predictions_SpO2_historical = predict(SpO2_model_historical,
XTest_SpO2_historical);
       accuracies_SpO2_historical(i) = sum(predictions_SpO2_historical
== YTest_SpO2_historical) / length(YTest_SpO2_historical);

       % Repeat for HeartRate
       XTrain_HeartRate_historical =
HeartRate_historical_norm(training(cv_HeartRate_historical, i), :);
       YTrain_HeartRate_historical =
HeartRate_labels_historical(training(cv_HeartRate_historical, i));
       XTest_HeartRate_historical =
HeartRate_historical_norm(test(cv_HeartRate_historical, i), :);
       YTest_HeartRate_historical =
HeartRate_labels_historical(test(cv_HeartRate_historical, i));

       HeartRate_model_historical =
fitcensemble(XTrain_HeartRate_historical, YTrain_HeartRate_historical,
'Method', 'Bag');
       predictions_HeartRate_historical =
predict(HeartRate_model_historical, XTest_HeartRate_historical);
       accuracies_HeartRate_historical(i) =
sum(predictions_HeartRate_historical == YTest_HeartRate_historical) /
length(YTest_HeartRate_historical);
```

```
        % Repeat for Tremor
        XTrain_Tremor_historical =
Tremor_historical_norm(training(cv_Tremor_historical, i), :);
        YTrain_Tremor_historical =
Tremor_labels_historical(training(cv_Tremor_historical, i));
        XTest_Tremor_historical =
Tremor_historical_norm(test(cv_Tremor_historical, i), :);
        YTest_Tremor_historical =
Tremor_labels_historical(test(cv_Tremor_historical, i));

        Tremor_model_historical = fitcensemble(XTrain_Tremor_historical,
YTrain_Tremor_historical, 'Method', 'Bag');
        predictions_Tremor_historical = predict(Tremor_model_historical,
XTest_Tremor_historical);
        accuracies_Tremor_historical(i) =
sum(predictions_Tremor_historical == YTest_Tremor_historical) /
length(YTest_Tremor_historical);
    end

    % Display mean accuracy across folds for historical data
    SpO2_meanAccuracy_historical = mean(accuracies_SpO2_historical);
    HeartRate_meanAccuracy_historical =
mean(accuracies_HeartRate_historical);
    Tremor_meanAccuracy_historical = mean(accuracies_Tremor_historical);

    disp(['Mean Accuracy for SpO2 (historical): ',
num2str(SpO2_meanAccuracy_historical)]);
    disp(['Mean Accuracy for HeartRate (historical): ',
num2str(HeartRate_meanAccuracy_historical)]);
    disp(['Mean Accuracy for Tremor (historical): ',
num2str(Tremor_meanAccuracy_historical)]);
end

% Retrieve live data
[new_data, ~] = thingSpeakRead(channelID, 'Fields', fields, 'NumPoints',
1, 'ReadKey', readAPIKey);

% Check if live data is empty
if isempty(new_data)
    disp('No live data available from ThingSpeak.');
else
    % Extract live data
    SpO2_live = new_data(1);
    HeartRate_live = new_data(2);
    Tremor_live = new_data(3);
```

```
    % Check for valid live data (exclude SpO2 < 89%)
    if SpO2_live >= 75
        % Normalize the live data
        SpO2_live_norm = (SpO2_live - min(SpO2_historical)) /
(max(SpO2_historical) - min(SpO2_historical));
        HeartRate_live_norm = (HeartRate_live -
min(HeartRate_historical)) / (max(HeartRate_historical) -
min(HeartRate_historical));
        Tremor_live_norm = (Tremor_live - min(Tremor_historical)) /
(max(Tremor_historical) - min(Tremor_historical));

        % Perform fatigue analysis only if SpO2 is valid
        SpO2_fatigue = SpO2_live < SpO2_threshold;
        HeartRate_fatigue = HeartRate_live > HeartRate_threshold;
        Tremor_fatigue = (Tremor_live > Tremor_min_threshold) &&
(Tremor_live < Tremor_max_threshold);

        % Display live data results
        disp(['Live SpO2: ', num2str(SpO2_live), ' | Fatigue: ',
num2str(SpO2_fatigue)]);
        disp(['Live HeartRate: ', num2str(HeartRate_live), ' | Fatigue:
', num2str(HeartRate_fatigue)]);
        disp(['Live Tremor: ', num2str(Tremor_live), ' | Fatigue: ',
num2str(Tremor_fatigue)]);
    else
        disp('Put your finger SpO2 is below " Normal" 75 %, skipping
fatigue analysis for live data.');
    end
end
```