Project Report

The project report submitted in partial fulfilment of the

requirements for the award of the degree of

The Professional Master Of

Embedded And Cyber-Physical Systems

ECPS205: Wireless Sensors and Actuator Networks

"Design and Implementation of a Wireless Pulse Monitoring System"

Under the guidance of

Prof. Mahya Safavi

Name of Candidates	UCI Student ID
Sahil Kakadiya	93282494
Vaibhav Gajera	89125790
Aneri Hiren Desai	45409124
Prateeksha Ranjan	28112912

❖ OBJECTIVE

- Develop a wireless pulse monitoring system using two Raspberry Pi units. This system will capture pulse signals through a sensor, transmit the data over Bluetooth, and display the pulse data along with other relevant health metrics on a GUI at the receiving end.

❖ Project Demonstration Link

- Google Drive Demo Link: https://drive.google.com/file/d/18IRrI4MpeUbWc3CTB0dZtUQCziCY4v9t/view?usp=sharing

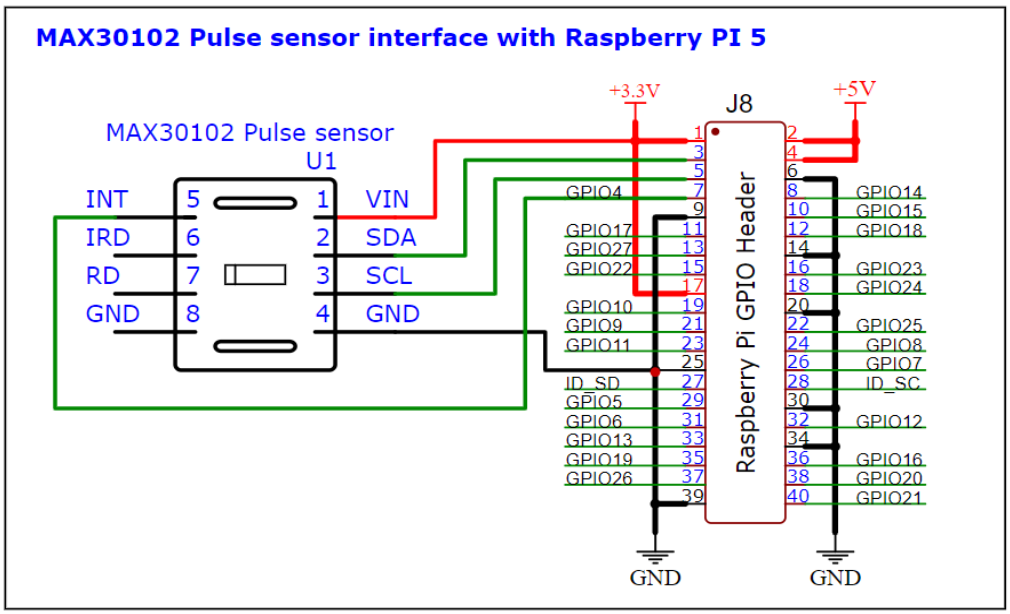
❖ PROJECT SETUP

➢ Hardware Setup (Transmitter Side):

- Connect the PPG Sensor to the Transmitter Raspberry Pi 5

  The connections between the PPG sensor and the Raspberry Pi GPIO pins are configured as follows:

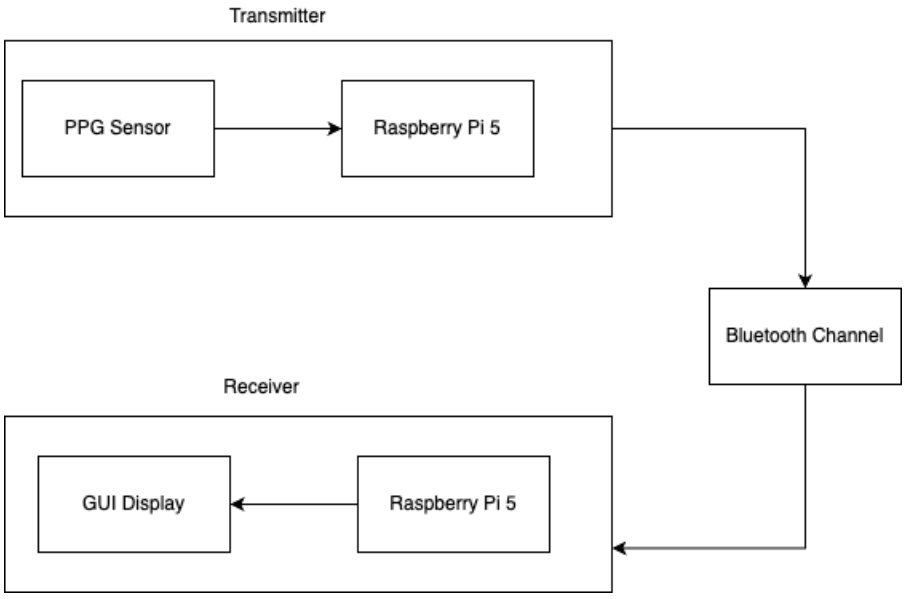
PPG Sensor	Raspberry Pi 5
GND	GND
VIN	3V3
SDA	SDA1
SCL	SCL1
INT	GPIO4



➢ Bluetooth Communication

- The transmitter Raspberry Pi 5 is configured to communicate with the receiver Raspberry Pi 5 using a Bluetooth channel. This connection enables the transmission of sensor data to the receiver for further processing.

❖ Data Flow



➢ Data Collection (Transmitter Side)

- The PPG sensor reads physiological signals and transmits the data to the connected transmitter Raspberry Pi through the configured GPIO pins.
- The Raspberry Pi processes the raw sensor data in JSON format and prepares it for transmission over Bluetooth.

➢ Data Transmission

- The processed data is sent to the receiver Raspberry Pi 5 through the established Bluetooth channel.

❖ Data Processing And Visualization (Receiver Side)

➢ Data Reception

- The receiver Raspberry Pi 5 receives the transmitted data from the transmitter Pi via Bluetooth.

➢ Data Processing

- The data is further processed on the receiver to extract relevant features or perform necessary computations.

➢ GUI Visualization

- The processed data is displayed visually on a graphical user interface (GUI).

❖ Graphical User Interface (GUI) Components

The GUI is designed to facilitate interaction with the transmitter Raspberry Pi and provides real-time visualization of sensor data. The components of the GUI are detailed below:

➢ Button Event Handler

- **Connect Button**
  - Sends a POST request to /connect to establish a Bluetooth connection with the server.
  - If successful, the connect button is disabled, and the disconnect button and start button are enabled.
- **Disconnect Button**
  - Sends a POST request to /disconnect to disconnect the Bluetooth connection.
  - If successful, it re-enables the connect button and disables the others (disconnect, start, stop).
- **Start Button**
  - Sends a POST request to /start to start the data reception from the server.
  - If successful, it disables the start button and enables the stop button, initiating the data fetching process.
- **Stop Data**
  - Sends a POST request to /stop to stop data reception from the server.
  - If successful, it disables the stop Button and re-enables the start Button.

➢ Graph Plot

- **Purpose:** Provides a real-time visualization of the beats per minute (BPM) received from the PPG sensor plotted against time.
- **Feature:** Allows users to monitor and analyze the data dynamically.

➢ Fetching Pulse Data

- The fetchPulseData function fetches pulse data from the backend by sending a GET request to /get_pulse_data.
- The backend responds with JSON data containing pulse metrics (e.g., pulse, beats per minute, impulses per minute, root mean square, and heart rate standard deviation).

- The pulse data is displayed in the HTML table and updated every 2 seconds (due to the setTimeout function).

➢ Chart Update:

- The time of data retrieval (now) is added to the chart labels.
- The beats per minute (beats_per_minute) is added to the chart's data array.
- To avoid chart overflow, only the latest 30 data points are retained by removing the oldest data points.
- The chart is updated using pulseChart.update() to reflect new data points.

➢ VitalsBlock

The vitals block displays essential parameters derived from the PPG sensor readings:

- **BPM (Beats per Minute):** Indicates the heart rate in beats per minute.
- **IPM (Impulses per Minute):** Reflects the number of impulses detected per minute.
- **HRSTD:** Represents the standard deviation of heart rate variability.
- **RMSSD (Root Mean Square of Successive Differences):** A measure used to evaluate the variation in heart rate over successive beats, indicative of heart rate variability (HRV).

This design ensures efficient data transmission and comprehensive visualization for user interaction and analysis.

❖ Server Side Code Flow:

➢ Initialization:

- The Bluetooth Pulse Server is created, which in turn initializes the Bluetooth Connection Manager. The server starts the accept_connection thread to listen for incoming Bluetooth connections.

➢ Bluetooth Connection:

- When a Bluetooth client connects to the server, the Bluetooth Connection Manager accepts the connection, and the on_connect_callback (defined in Bluetooth Pulse Server) is invoked, triggering start_data_collection.

➢ Data Reception:

- The Bluetooth Connection Manager listens for incoming data from the connected client through the listen_for_data method. The data_received_callback function in Bluetooth Pulse Server processes the data.
- If the data is a command (START_SYNC, STOP_SYNC, or ACK_ACK), it triggers synchronization logic:
  - START_SYNC: Acknowledged with ACK, and the server waits for ACK_ACK. If received, it starts the pulse data transmission.
  - STOP_SYNC: Acknowledged similarly, but stops data transmission upon receiving ACK_ACK.

➢ Pulse Data Transmission:

- Once synchronization (START_SYNC) is successful, the stream_pulse_data method starts. This continuously reads data from the MAX30102 sensor, calculates the heart rate and SpO2, and transmits the pulse data back to the client.
- The pulse data is sent as a JSON string (containing heart rate, SpO2, etc.) every second, while transmit_data is True.

➢ Client Disconnects:

- If the client disconnects or sends a STOP_SYNC command, data collection stops, and the server is ready to accept a new client connection.

❖ Client-Side Code Workflow

➢ Initialization:

- The Bluetooth Client class is instantiated with details like the target device name, port, and address. The Bluetooth client prepares itself to communicate with the Bluetooth server.
- The command_queue holds commands sent by the Flask server via API calls (connect, start, stop, disconnect).

➢ Command Handler:

- The client starts a background thread (command_handler) that listens for commands from the queue. It processes commands like connect, start, stop, and disconnect sequentially.
- Commands are added to the queue by Flask endpoints (/connect, /start, /stop, /disconnect).

➢ Handling Commands:

- **Connect:**
  - The connect command triggers the client to discover and pair with the Bluetooth server. If successful, it establishes a connection.
- **Start:**
  - The start command initiates data reception. It sends START_SYNC, waits for ACK, then sends ACK_ACK, and starts the data reception thread.
- **Stop:**
  - The stop command stops data reception. It sends STOP_SYNC, waits for ACK, and then sends ACK_ACK.
- **Disconnect:**
  - The disconnect command stops data reception (if active) and disconnects from the Bluetooth server.
- **Data Reception:**
  - The data_reception_loop method continuously listens for pulse data from the Bluetooth server, processes it, and stores it in the pulse_data variable.
  - This data is available to be fetched by the frontend through the /get_pulse_data API endpoint.
- **Communication:**
  - The client uses the bluetooth.BluetoothSocket to send and receive commands/data to/from the Bluetooth server.
  - Commands such as START_SYNC, STOP_SYNC, and data responses are exchanged over the Bluetooth connection.

- **Stopping and Disconnecting:**
  - The client properly shuts down the connection by stopping the data thread and closing the Bluetooth connection when the disconnect or exit command is processed.

❖ Sensor Data Reading

➢ Data Reading from Sensor (MX30102)

- **Sensor Read Loop:**
  - The read_sensor function contains a loop that runs for a number of iterations equal to WINDOW_SIZE. Each iteration attempts to read data from the sensor.

- **red, ir = m.read_sequential():**
  - The m.read_sequential() function reads the red and infrared (IR) light data from the sensor. These are key measurements used to calculate heart rate and SpO2.

  - The red light is used to detect oxygen levels in the blood, while infrared light helps in assessing heart rate.

➢ Calculation of Heart Rate and SpO2

- **Heart Rate and SpO2 Calculation:**
  - After acquiring the data, the code calls hrcalc.calc_hr_and_spo2(ir, red), which uses the infrared (ir) and red light data to calculate:

  - **Heart Rate (hr):** Measured in beats per minute (BPM).

  - **SpO2:** The oxygen saturation level in the blood, typically expressed as a percentage.

  - **RMSSD:** The root mean square of successive differences of heart rate measurements, a standard metric for variability in heart rate.

These calculations require processing the sensor data to determine peaks (heartbeats) and calculate oxygen levels using algorithms like the ones used in pulse oximeters.

➢ Validating Readings

- **Validity Check:**
  - The system checks if the calculated values for heart rate and SpO2 are valid by evaluating the flags hr_valid and spo2_valid.
  - If both flags are True, the readings are valid, and the system proceeds to further analysis.
  - If either value is invalid, the system prints an error message ("Invalid readings. Please try again.") and returns None.

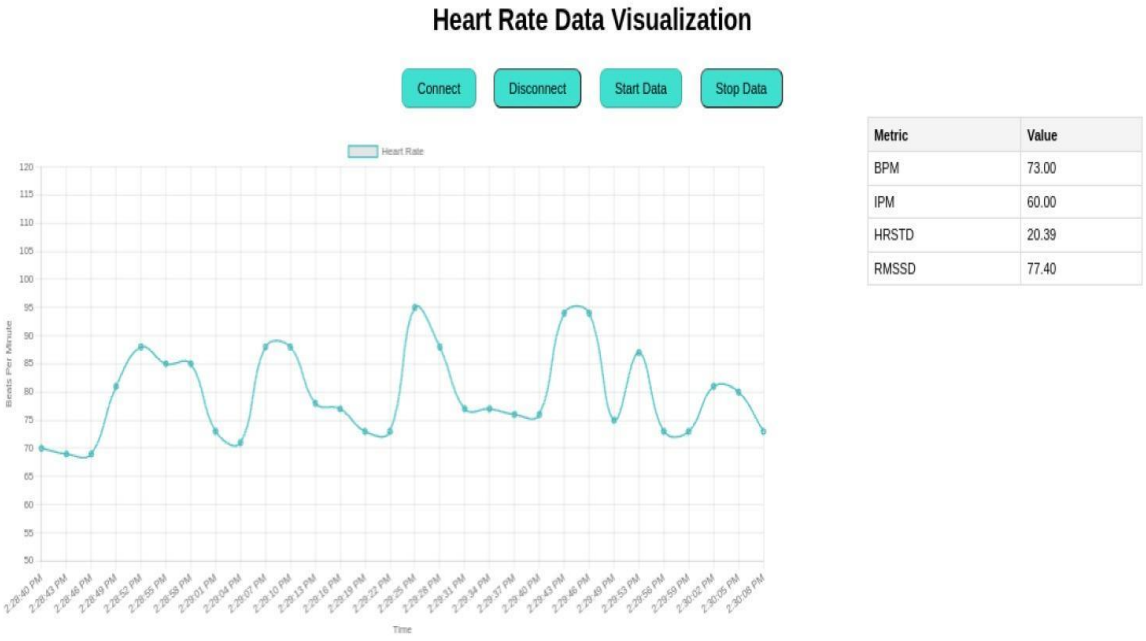
➢ Heart Rate Standard Deviation (HRSTD) Calculation

- **HRSTD Queue:**
  - The hrstd_queue is used to store a series of heart rate values over a window of time (defined by WINDOW_SIZE).
- **HRSTD Calculation:**
  - If the queue contains more than two data points, it calculates the standard deviation (np.std(hrstd_queue)) of the heart rate values in the queue. This value is used to assess variability in the heart rate over time.
  - If there are fewer than two values in the queue, HRSTD is set to None.
- **Generating Pulse Data JSON**

  If all the required metrics are valid:
  - The function generates a JSON object (pulse_data_json) containing the following data:
  - **Pulse**: A random pulse value between 60 and 100 (simulated data for testing).
  - **Impulses Per Minute**: A random number between 50 and 120 (simulated data).
  - **Beats Per Minute**: The calculated heart rate (HR).
  - **Root Mean Square**: The RMSSD value.
  - **HRSTD:** The calculated heart rate standard deviation (HRSTD).

- **Handling Invalid Data**
  - If any of the calculated values are invalid (either HR or SpO2), the function prints an error message and returns None.

❖ Observation:



Metric	Value
BPM	73.00
IPM	60.00
HRSTD	20.39
RMSSD	77.40