EE381

Electronic Circuit Laboratory Project Report

Design and Implementation of Heart Beat/Pulse Sensor

Section: A

Table No. 13

Name: Mudit Jain (220674)

Group Member: Monika (220669)

Problem Statement

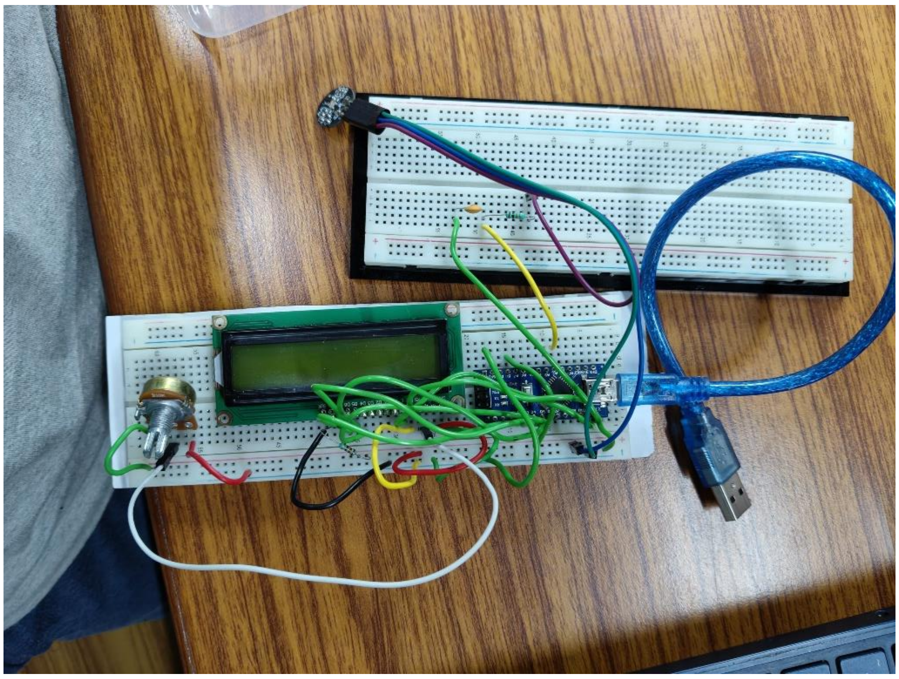
Pulse monitoring is essential in both healthcare and fitness settings. While commercial pulse oximeters are widely available, there is demand for low-cost, DIY systems that can accurately detect and display a person's heart rate in real time using readily available components. The challenge lies in denoising raw sensor data, detecting valid peaks reliably, and displaying readable BPM data on an LCD screen.

This project addresses these challenges by developing an Arduino-based pulse rate monitoring system with real-time BPM calculation, noise reduction, and display via a 16x2 LCD.

Hardware Components:

- Arduino Nano
- Pulse Sensor
- 16x2 LCD Display (I2C-based)
- Breadboard & Jumper Wires
- Power Source (USB)

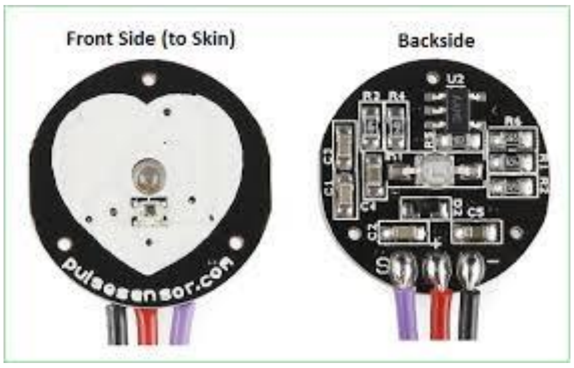
Circuit



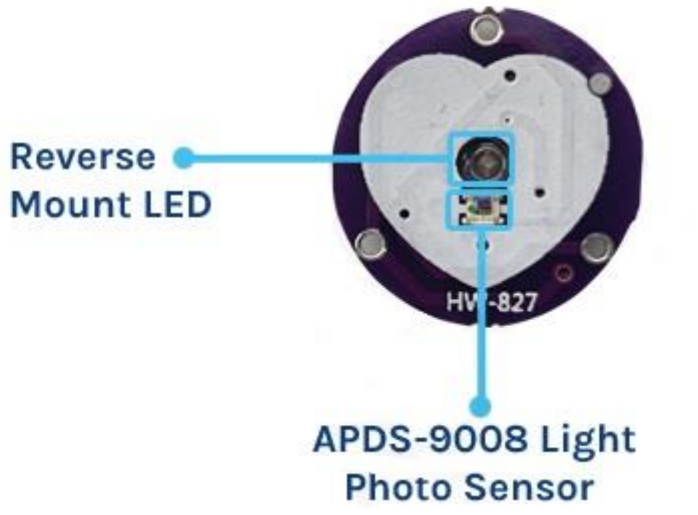
Overview:

Pulse Sensor: -

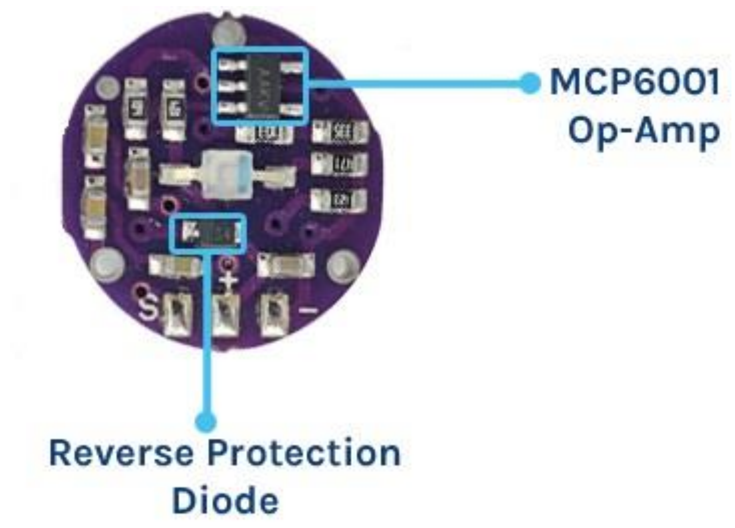
The **Pulse Sensor** is a plug-and-play **heart-rate sensor for Arduino**.



The essence is an integrated **optical amplifying circuit** and **noise eliminating circuit** sensor. Clip the **Pulse Sensor** to your earlobe or fingertip. Then it into your **Arduino**, you are now ready to read **heart rate**.



The front of the sensor comes with the heart logo. This is where you place your finger. On the front side, you will see a small round hole, from where the green LED shines. Just below the LED is a small **ambient light photosensor** APDS9008 which adjust the brightness in different light conditions.



On the back of the module you will find **Op-Amp IC**, a few resistors, and capacitors. This makes up the **R/C filter** network. There is also a **reverse protection diode** to prevent damage if you connect the power leads reverse.

Pulse Sensor Technical Specifications: -

Physical Characteristics-

- **Dimensions**: Approximately 0.625” (15.875mm) in diameter
- **Weight**: Lightweight, usually around a few grams
- **Material**: Biocompatible materials for safe skin contact

Electrical Characteristics-

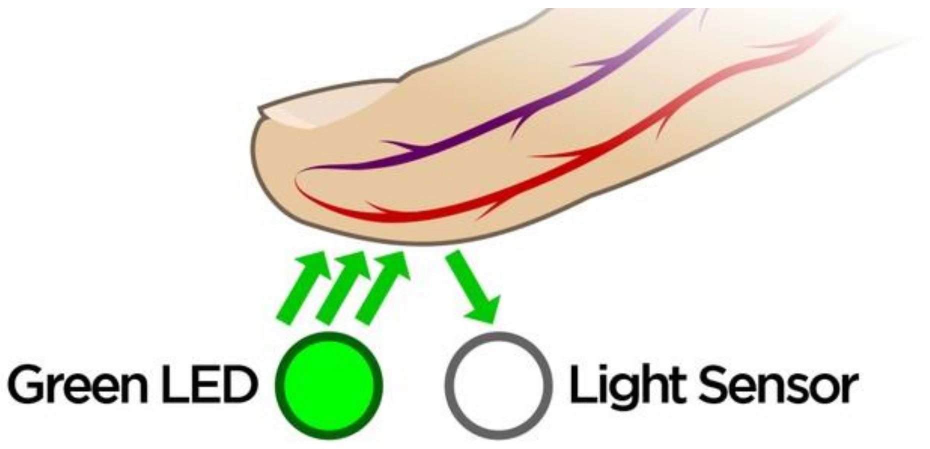
- **Operating Voltage**: 3V – 5.5V
- **Current Consumption**: Typically, around 4mA
- **Output Signal**: Analog (0.3V to VCC)
- **Signal Range**: 0-1023 (10-bit ADC output of Arduino)

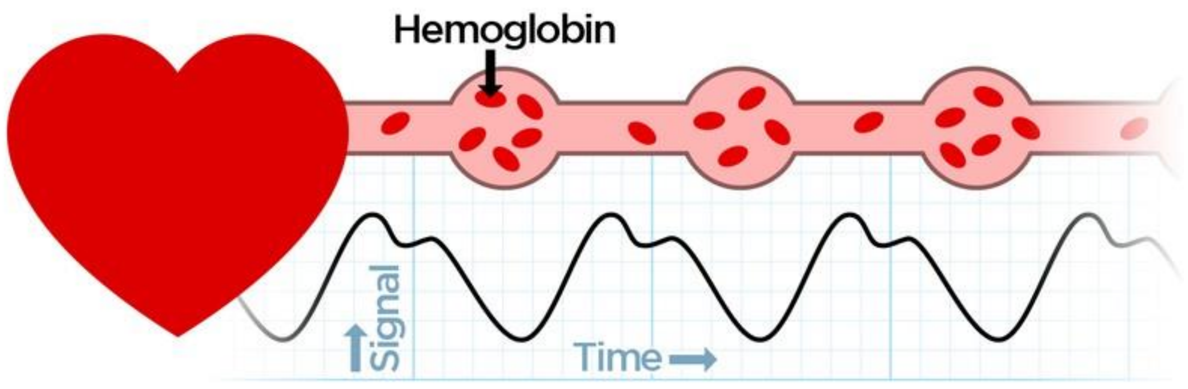
Sensing Technology-

- **Sensor Type**: Photoplethysmogram (PPG)
- **Wavelength**: Typically, around 565nm (Green LED)

Working of the Pulse Sensor: -

The Pulse Sensor works on the principle of Photoplethysmography (PPG), which is a non-invasive method for measuring changes in blood volume under the skin. The sensor essentially consists of two main components: a light-emitting diode (LED) that shines light into the skin and a photodetector that measures the amount of light that is reflected back. Here’s a detailed explanation of its working:

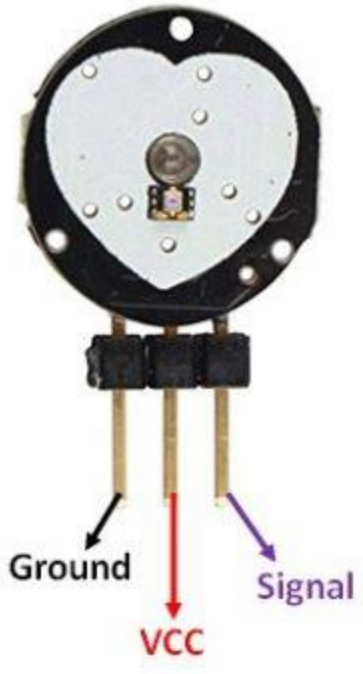




1. **Light Emission**: A green LED emits light into the skin.
2. **Reflection & Detection**: The light interacts with blood and is partially reflected back, captured by a photodetector.
3. **Heart Rate**: Changes in reflected light create a waveform that correlates with heartbeats.
4. **Oxygen Level**: The amount of reflected light also indicates blood oxygen levels, as oxygenated blood absorbs more green light.
5. **Signal Filtering**: A Low Pass Filter cleans up the noisy, raw signal from the photodetector.
6. **Amplification**: An operational amplifier boosts the filtered signal for better accuracy.
7. **Data Reading**: Finally, an Arduino reads the amplified signal and software algorithms translate it into heart rate and potentially blood oxygen levels.

Pulse Sensor Pinouts: -

The pulse sensor has three pins: VCC, GND & Analog Pin.



Interfacing Pulse Sensor with Arduino: -

Let us interface the Pulse Sensor with Arduino and start measuring the Pulse Rate/Heart-Beat/BPM Value.

16x2 Liquid Crystal LCD:

Display the calculated heart rate (BPM) and heartbeat detection status. This allowed offline, portable viewing of results directly on the device, without relying on Bluetooth or serial monitor.

- The LCD is operated using 4-bit mode (only 6 Arduino pins).
- It shows:
    - Line 1: Current BPM
    - Line 2: Status message like "Pulse Detected"

The Liquid Crystal library handles communication with the LCD via digital I/O pins.

Final Pin Connections:

Component	Arduino Pin	Description
Pulse Sensor Signal	A2	Analog Input
Pulse Sensor VCC	5V	Power Supply
Pulse Sensor – GND	GND	Ground
LCD RS	D12	Register select
LCD E	D11	Enable Pin
LCD D4	D5	Data line 4
LCD D5	D4	Data line 5
LCD D6	D3	Data line 6
LCD D7	D2	Data line 7
LED	Built-in (D13)	Blinks on Pulse detection
Potentiometer (LCD)	Centre to V0	Controls LCD contrast

Software and Implementation:

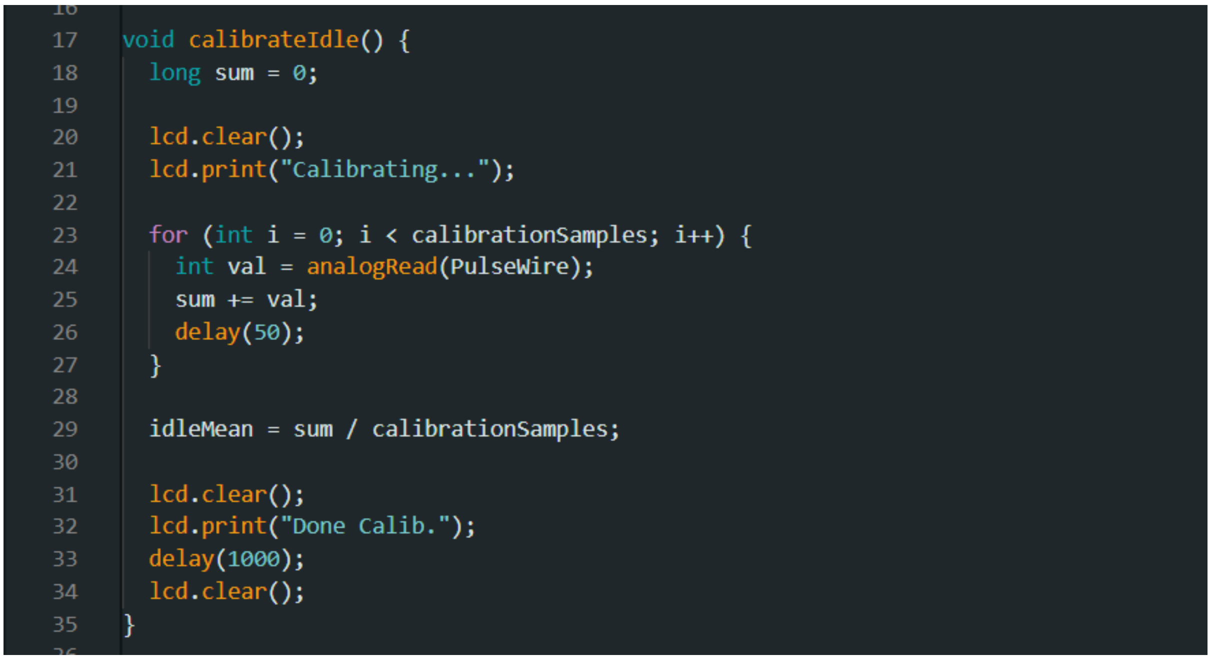
1. Signal Denoising

The raw signal from the pulse sensor is analog and often noisy. To smooth out fluctuations and avoid false beat detections:

- A **moving average filter** is applied to the analog readings.
- The signal is sampled at a high rate (e.g., every 2 ms).
- The signal is compared against a **calibrated idle baseline** to ignore small fluctuations.

2. Idle Value Calibration

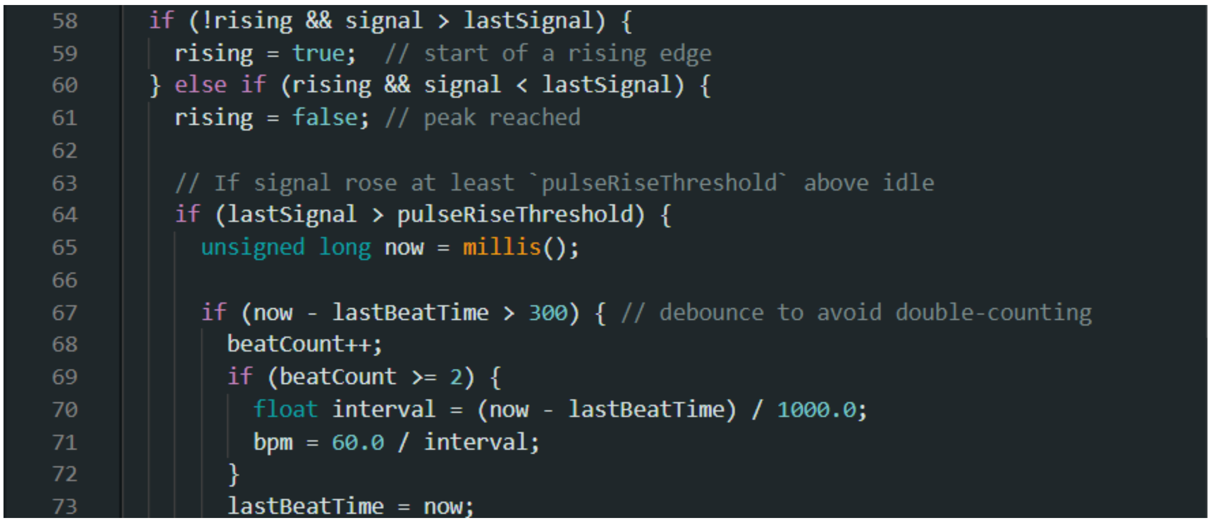
- During startup, the system records around **100 samples** from the sensor.
- These are averaged to compute a **baseline idle voltage**, which serves as a reference.
- This value compensates for environmental light, finger pressure, or sensor drift.



```
void calibrateIdle() {
  long sum = 0;

  lcd.clear();
  lcd.print("Calibrating...");

  for (int i = 0; i < calibrationSamples; i++) {
    int val = analogRead(PulseWire);
    sum += val;
    delay(50);
  }

  idleMean = sum / calibrationSamples;

  lcd.clear();
  lcd.print("Done Calib.");
  delay(1000);
  lcd.clear();
}
```

3. Peak Detection & BPM Calculation

- A **rising edge** is considered when the current value is significantly higher than the idle baseline (e.g., +40).
- A **cool-down period** (~300ms) is enforced to avoid multiple detections per beat.
- **Time between two valid peaks** is used to compute BPM:

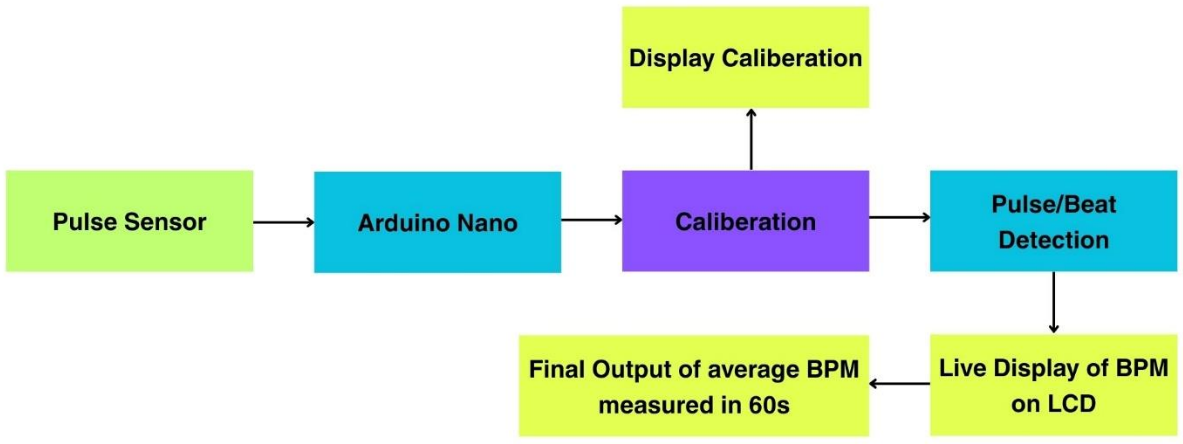


```
  if (!rising && signal > lastSignal) {
    rising = true;  // start of a rising edge
  } else if (rising && signal < lastSignal) {
    rising = false; // peak reached

    // If signal rose at least `pulseRiseThreshold` above idle
    if (lastSignal > pulseRiseThreshold) {
      unsigned long now = millis();

      if (now - lastBeatTime > 300) { // debounce to avoid double-counting
        beatCount++;
        if (beatCount >= 2) {
          float interval = (now - lastBeatTime) / 1000.0;
          bpm = 60.0 / interval;
        }
        lastBeatTime = now;
```

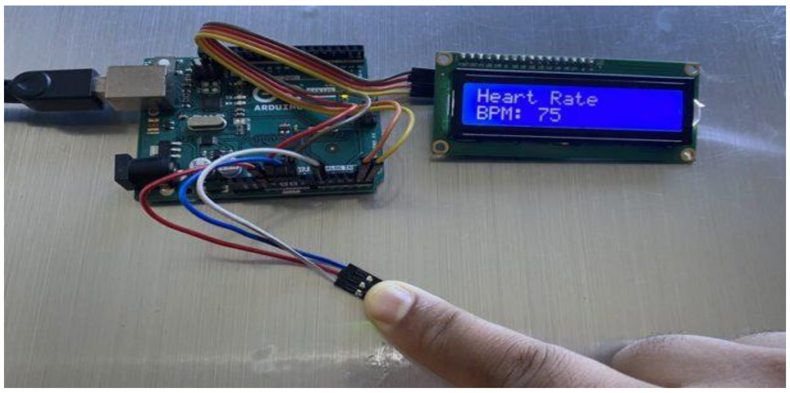
4. Displaying BPM on LCD

- A 16x2 LCD (with I2C interface) displays:
  - Real-time BPM
  - “No pulse detected” message if inactive
- LCD refreshes every few hundred milliseconds to avoid flicker.

Block Diagram:



Testing & Observations:



Noise Analysis and Signal Behavior:

During the initial testing phase of the pulse sensor, we observed significant noise in the analog signal output. The raw waveform fluctuated even when no finger was placed on the sensor, indicating interference and ambient signal pickup. Several key patterns emerged during this phase:

Idle Drift:
When no finger was placed, the analog values still varied around a central value, typically between **500–530**. This drift made it hard to determine whether a heartbeat signal was truly present.

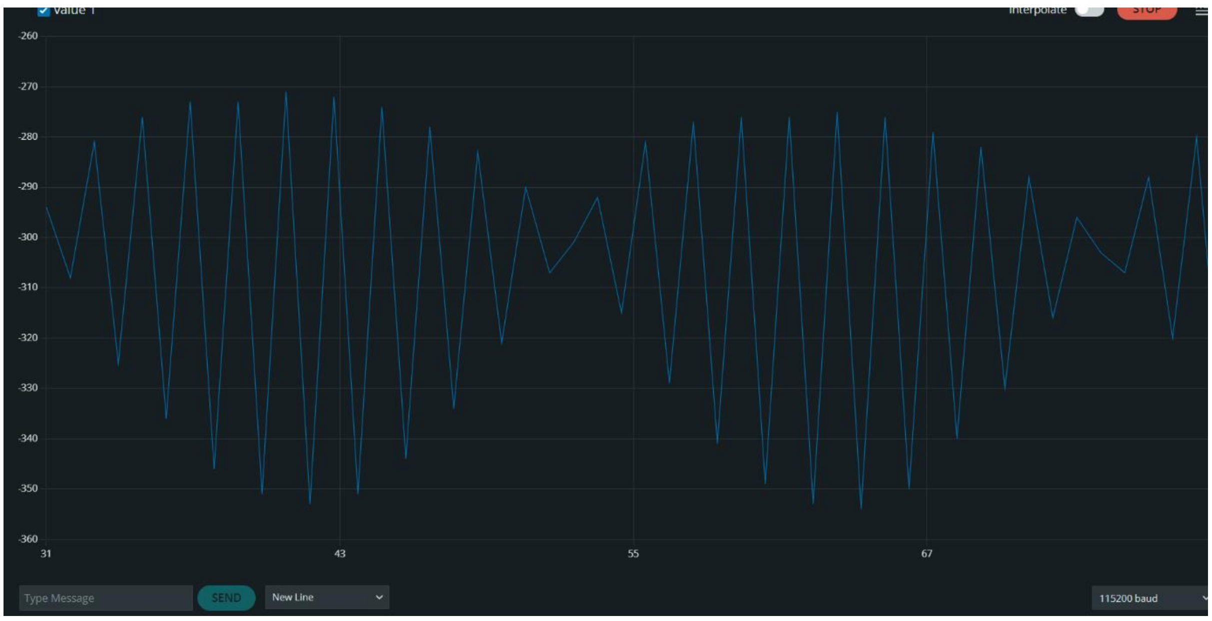
Sudden Spikes Without Finger:
We noticed sharp spikes in values due to **electromagnetic interference** or unstable contacts. These could falsely trigger heartbeat detection if not filtered.

Unstable Peaks During Early Contact:
Immediately after placing a finger, the waveform took 3–4 seconds to stabilize. Early data showed inconsistent peak height and timing.

Consistent Peaks Once Stabilized:
After calibration and smoothing, the waveform clearly exhibited periodic peaks that matched the rhythm of actual heartbeats.

Type of Noise	**Likely Cause**
High-frequency jitter	Electrical noise from jumper wires or USB power fluctuations
Idle drift	Sensor's analog offset & temperature variation
False peaks	Slight hand movements, sensor pressure variation
Low amplitude	Poor finger placement or insufficient skin contact

Noise Plot:



You can see the beat like pattern which is periodic in nature with an offset. Such characteristics helped us to device a strategy for denoising the signal.

Denoising Technique:

To address these issues, we implemented the following:

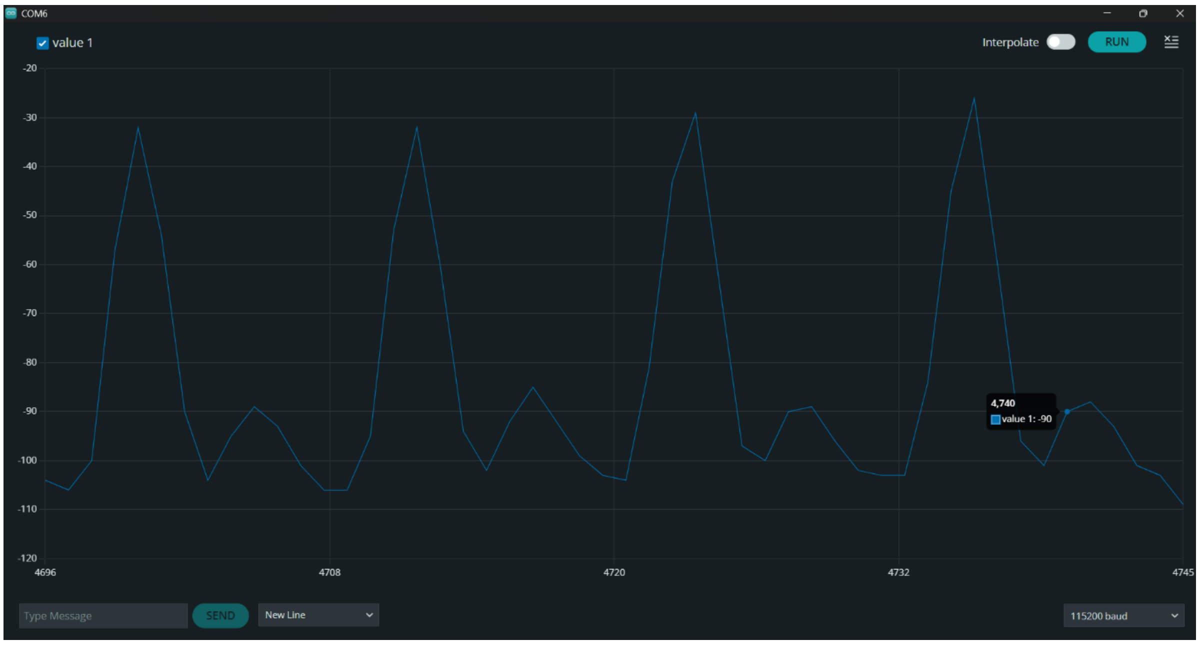
- **Calibration Phase**:
  On startup, the system samples 50 readings to compute the average **idle signal (mean value)**, which is then subtracted from all future readings to center the signal around zero.

- **Signal Smoothing**:
  Applied a **simple averaging filter** to reduce jitter and smooth the waveform:

  avgSignal = 0.6 * avgSignal + 0.4 * signal;

- **Threshold-Based Detection**:
  Pulse peaks were only considered valid if they exceeded a certain **amplitude threshold** (e.g., >70), minimizing false triggers.
- **Peak Lockout Window**:
  After each detected peak, we ignored further pulses for 300ms, preventing multiple detections of the same beat.

Output after Denoising:



It can be observed we have peaks resembling a beat followed by smaller peak which shows hearts diastole beat.

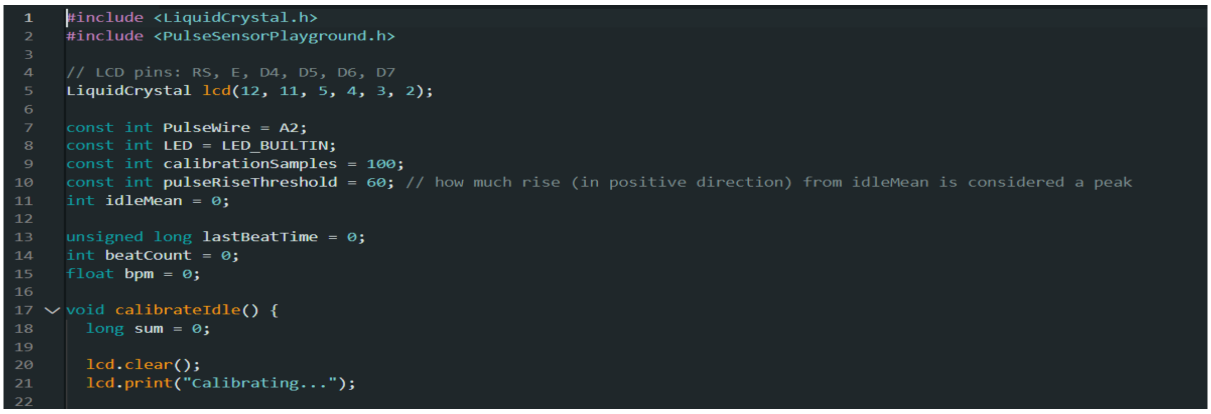
Key Results:

Metric	Result
Idle Calibration Time	~0.5 seconds
BPM Detection accuracy	~ ±5 BPM for steady hands
Minimum Detection interval	~300ms
Display Latency	~100ms
Sensor Noise Resilience	High (Post filtering)
Power Consumption	Low (USB-powered or battery-capability

Conclusion:

This project successfully demonstrates a robust, real-time pulse monitoring system using Arduino. Signal processing techniques like idle calibration and peak detection with debounce filtering helped minimize false readings. Displaying BPM on an LCD makes this setup completely standalone, without any dependency on mobile or cloud systems. It is ideal for low-cost, offline pulse monitoring use-cases in education, DIY health kits, or prototyping.

Final Code:

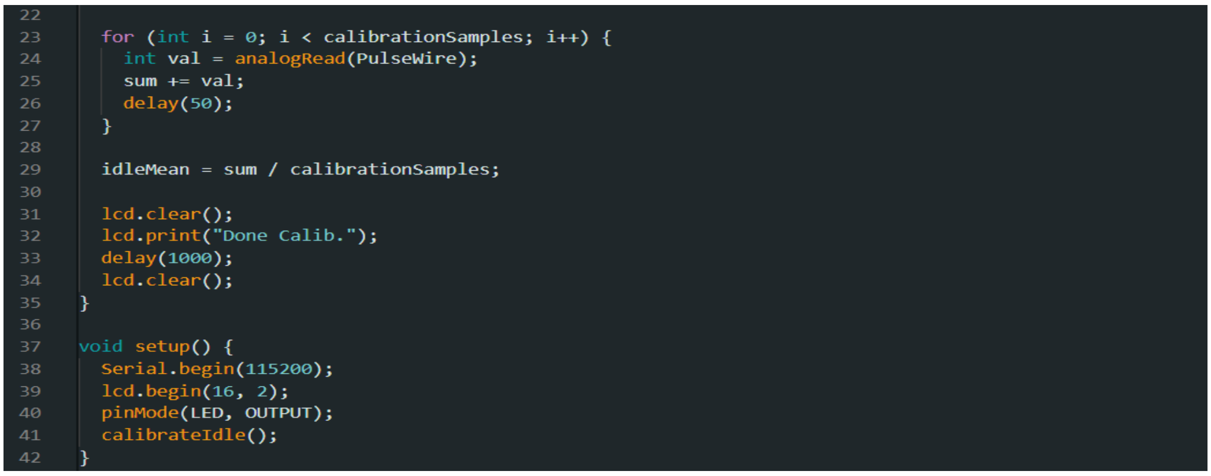


```
#include <LiquidCrystal.h>
#include <PulseSensorPlayground.h>

// LCD pins: RS, E, D4, D5, D6, D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int PulseWire = A2;
const int LED = LED_BUILTIN;
const int calibrationSamples = 100;
const int pulseRiseThreshold = 60; // how much rise (in positive direction) from idleMean is considered a peak
int idleMean = 0;

unsigned long lastBeatTime = 0;
int beatCount = 0;
float bpm = 0;

void calibrateIdle() {
  long sum = 0;

  lcd.clear();
  lcd.print("Calibrating...");

```

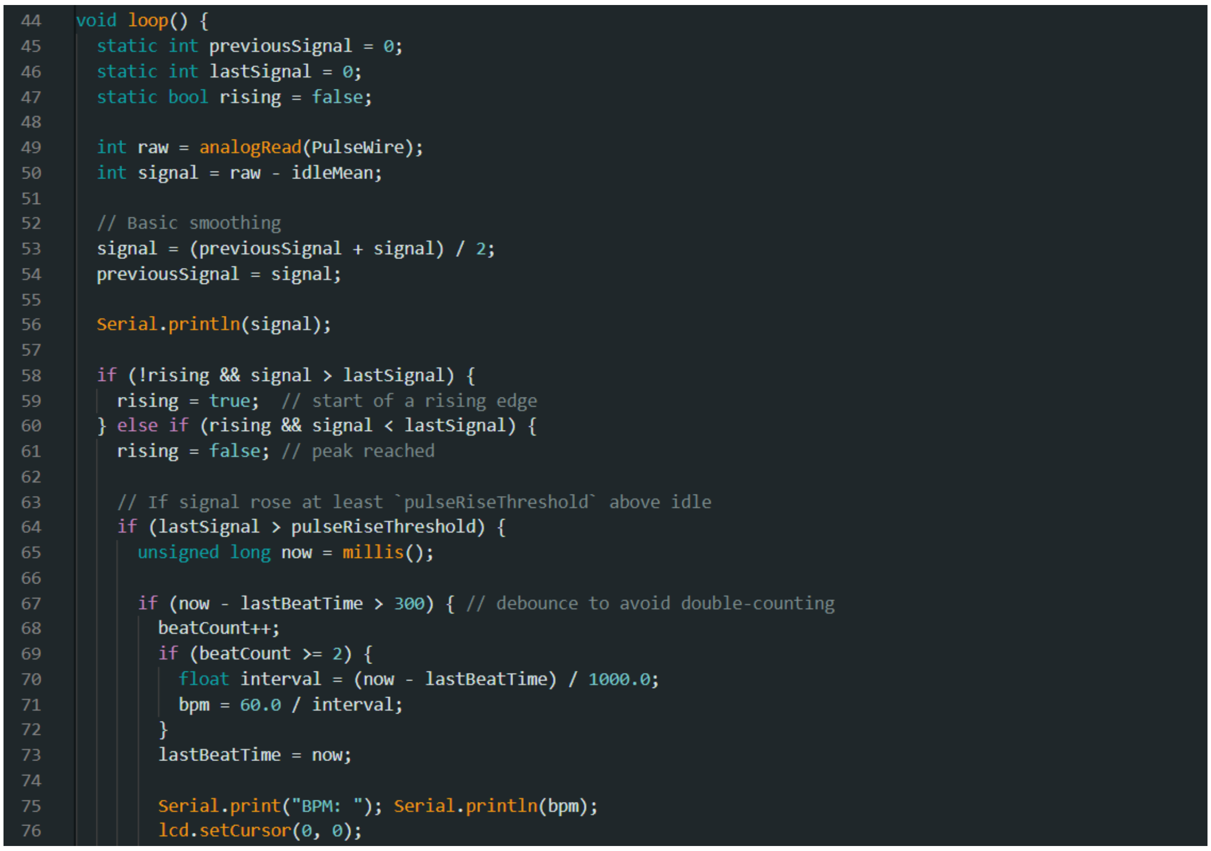
calibrate() function:

- Reads multiple values (100 samples) from the pulse sensor without a finger on it.
- Calculates the average idle value, i.e., what the sensor outputs when there's no pulse.
- This helps in differentiating real pulses from background noise.

The LCD says “Calibrating…” and then “Done Calib.”



```

  for (int i = 0; i < calibrationSamples; i++) {
    int val = analogRead(PulseWire);
    sum += val;
    delay(50);
  }

  idleMean = sum / calibrationSamples;

  lcd.clear();
  lcd.print("Done Calib.");
  delay(1000);
  lcd.clear();
}

void setup() {
  Serial.begin(115200);
  lcd.begin(16, 2);
  pinMode(LED, OUTPUT);
  calibrateIdle();
}
```



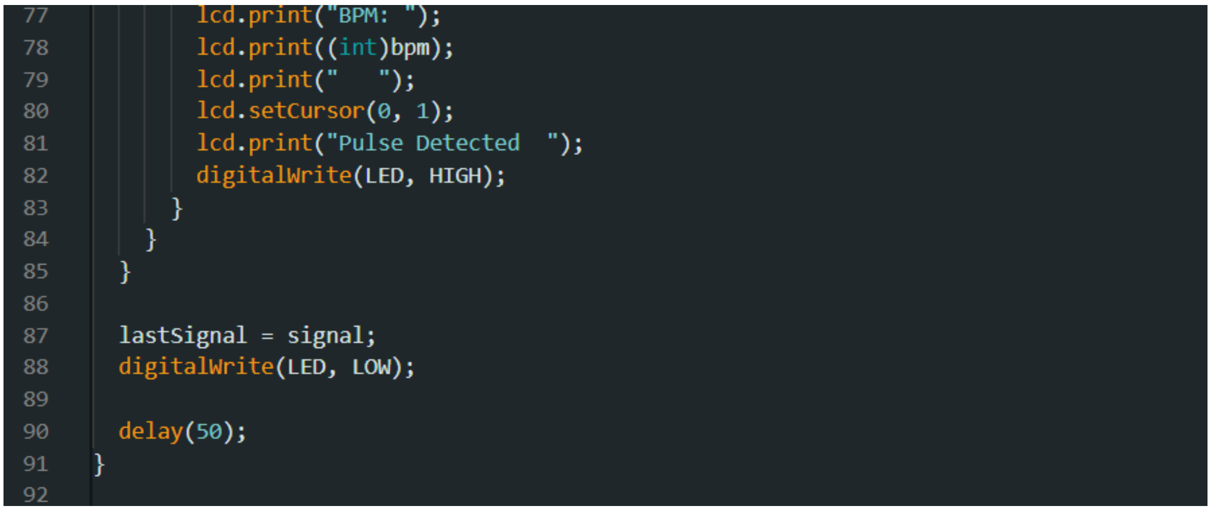
```
void loop() {
  static int previousSignal = 0;
  static int lastSignal = 0;
  static bool rising = false;

  int raw = analogRead(PulseWire);
  int signal = raw - idleMean;

  // Basic smoothing
  signal = (previousSignal + signal) / 2;
  previousSignal = signal;

  Serial.println(signal);

  if (!rising && signal > lastSignal) {
    rising = true;  // start of a rising edge
  } else if (rising && signal < lastSignal) {
    rising = false; // peak reached

    // If signal rose at least `pulseRiseThreshold` above idle
    if (lastSignal > pulseRiseThreshold) {
      unsigned long now = millis();

      if (now - lastBeatTime > 300) { // debounce to avoid double-counting
        beatCount++;
        if (beatCount >= 2) {
          float interval = (now - lastBeatTime) / 1000.0;
          bpm = 60.0 / interval;
        }
        lastBeatTime = now;

        Serial.print("BPM: "); Serial.println(bpm);
        lcd.setCursor(0, 0);
```

- It reads the current pulse value.
- Subtracts the idle value so only variations due to heartbeat remain.
- Then smooths the signal to reduce noise.
- It looks for a **rising edge** (when signal increases) and then a **falling edge** (when it starts to drop).
- If the highest point (peak) exceeds a threshold (60), it's considered a **valid heartbeat**.

If a valid heartbeat is found:

- The time between this and the last beat is measured.

- BPM is calculated using BPM = 60 / time_between_beats.
- **BPM is displayed** on the LCD.
- “Pulse Detected” appears briefly on the second line.
- The onboard LED flashes with each heartbeat.



```
          lcd.print("BPM: ");
          lcd.print((int)bpm);
          lcd.print("    ");
          lcd.setCursor(0, 1);
          lcd.print("Pulse Detected  ");
          digitalWrite(LED, HIGH);
        }
      }
    }

    lastSignal = signal;
    digitalWrite(LED, LOW);

    delay(50);
  }
```

- Updates lastSignal for the next loop iteration.
- Turns the LED off after a short pulse.
- Adds a **small delay (50ms)** to control data rate.

----------THANK YOU---------