Wearable Respiratory and Blood Oxygen Monitoring System

for optimization of blood oxygen Levels

Catharine Tian
Eric Ho
Kelsie Yamano
Shayan Alipourjeddi
Sunny Li

6~12%

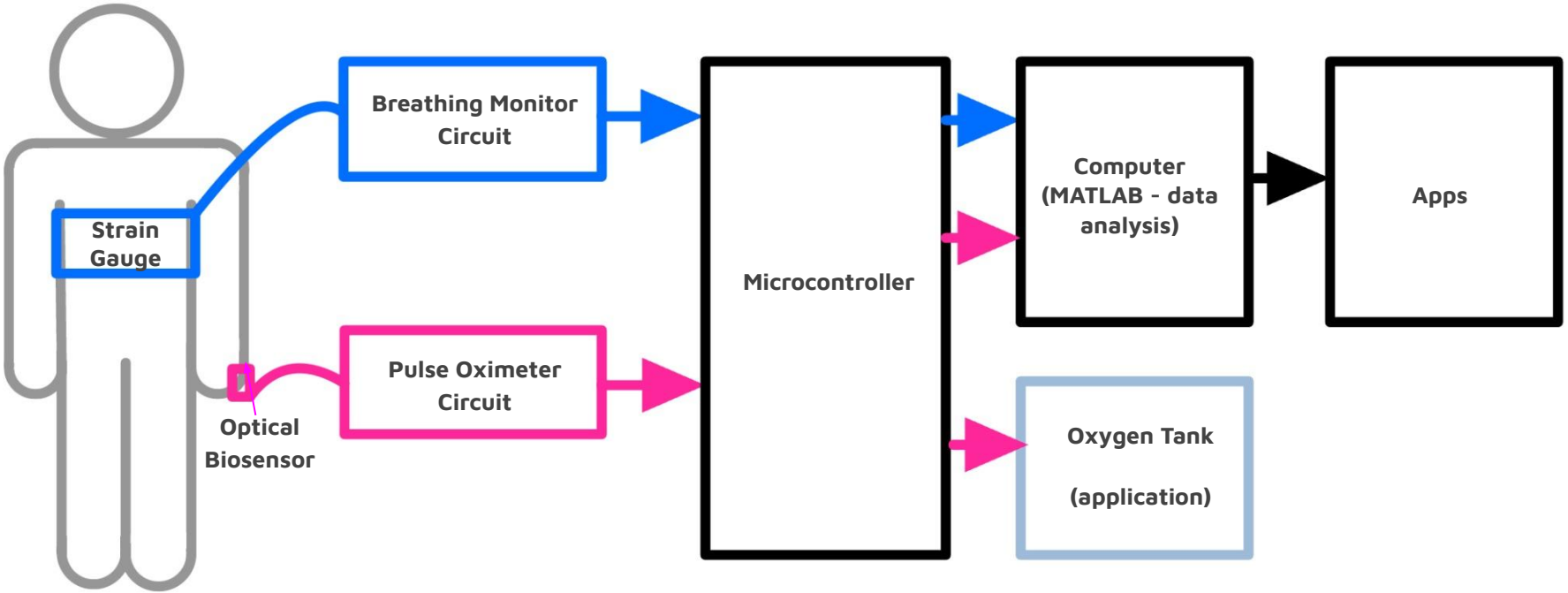
...of the population experience **chronic breathing pattern disorders**, approximately.

- Critical role in ICU for adjusting ventilator settings and ensuring optimal oxygenation
- Fitness & Training

Goals

- Wearable respiratory monitor
  - Breathing rate tracking
  - Breathing depth tracking
- Pulse oximeter
  - Blood oxygen level tracking
  - Monitor oxygen level when it drops below a personalized threshold
- Send all tracking data to phone/computer

Overview

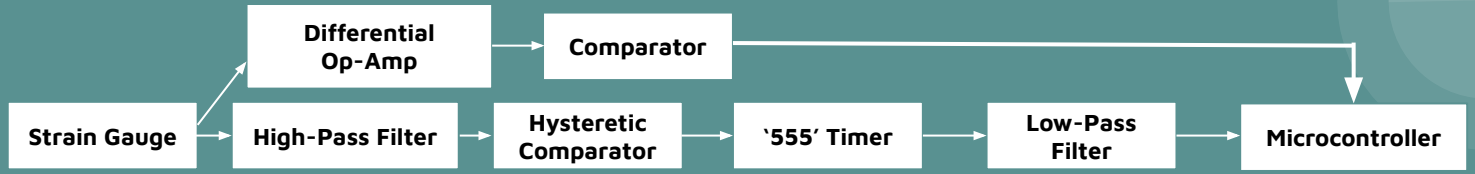

Strain Gauge
Optical Biosensor
Breathing Monitor Circuit
Pulse Oximeter Circuit
Microcontroller
Computer (MATLAB - data analysis)
Apps
Oxygen Tank
(application)

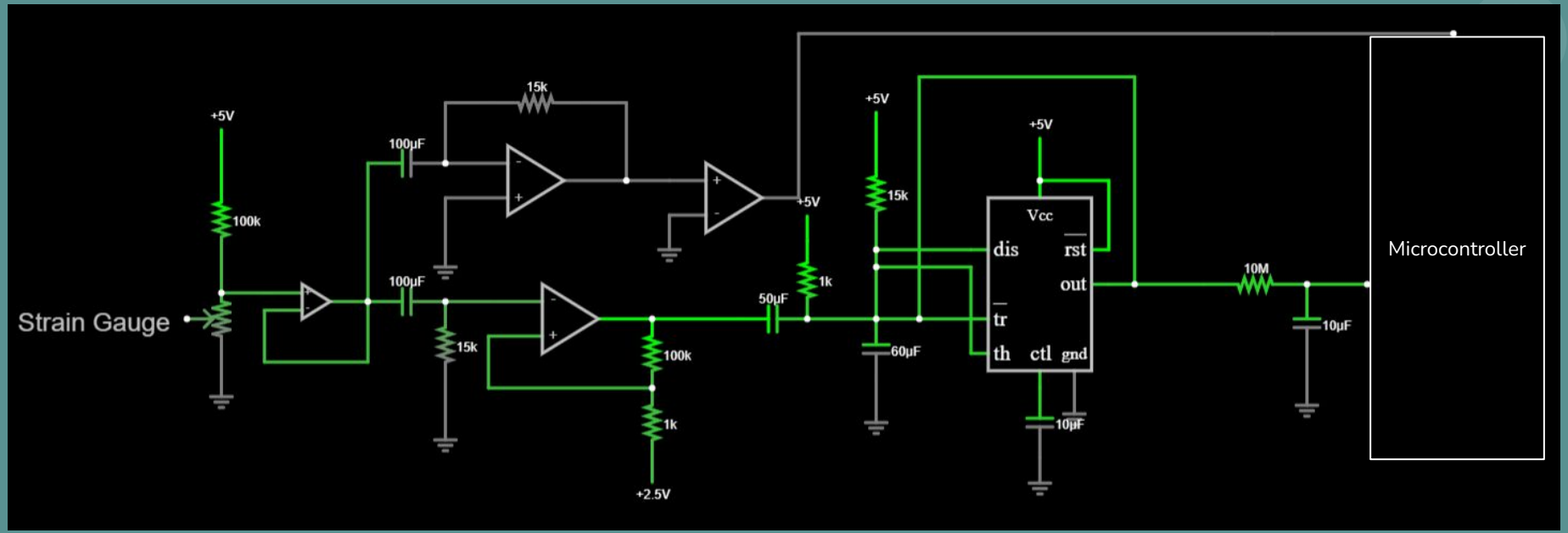
Assumptions

- Validity of Breath Monitor
  - device only measure inhalation and exhalation motion of the lung
  - device movement while running or walking would not trigger a breath
- Ideal Waveform
  - no significant noise interference in signal analysis

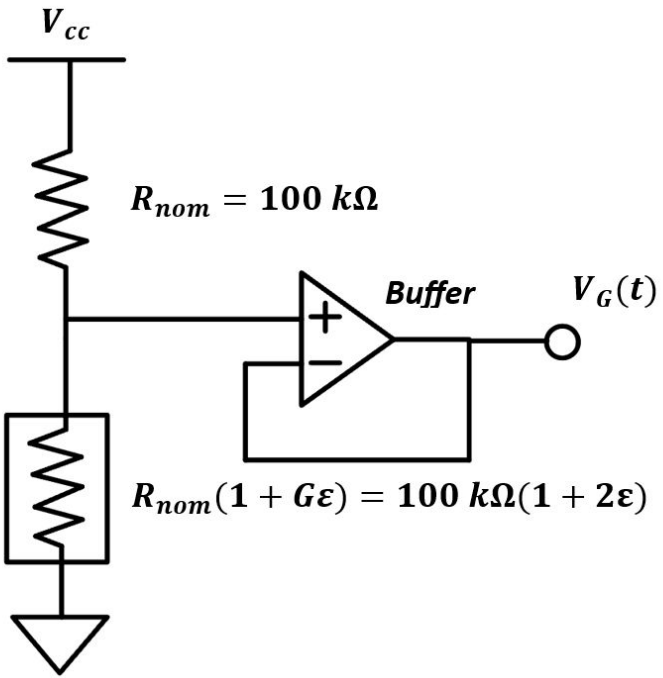
Breathing Monitor

Circuit Overview - Breathing Monitor





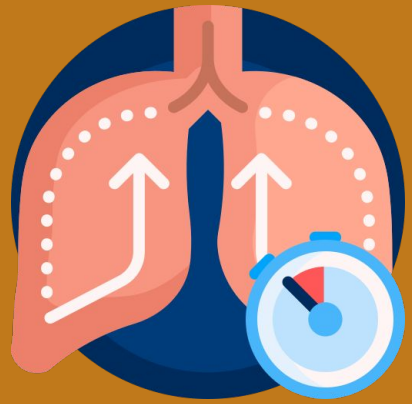
Strain Gauge w/ Voltage Divider



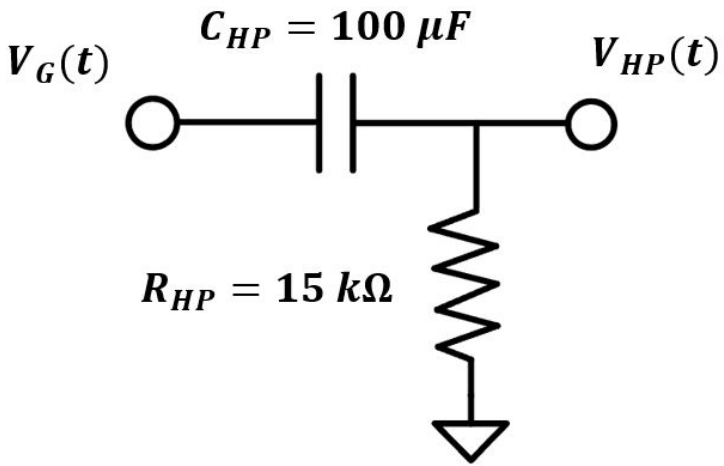
- Purpose: detect the change in lung volume when the person breathes and creates a signal
- Voltage follower is added to prevent loading
  - Buffer can strengthen the signal if the change in strain is small and produces a weak signal
- G = 2: typical for metallic strain gauges
- Vcc = 5V: typical portable battery

$$V_G(t) = \frac{1 + G\varepsilon}{2 + G\varepsilon} V_{cc}$$

Breathing Monitor
- Breathing Rate



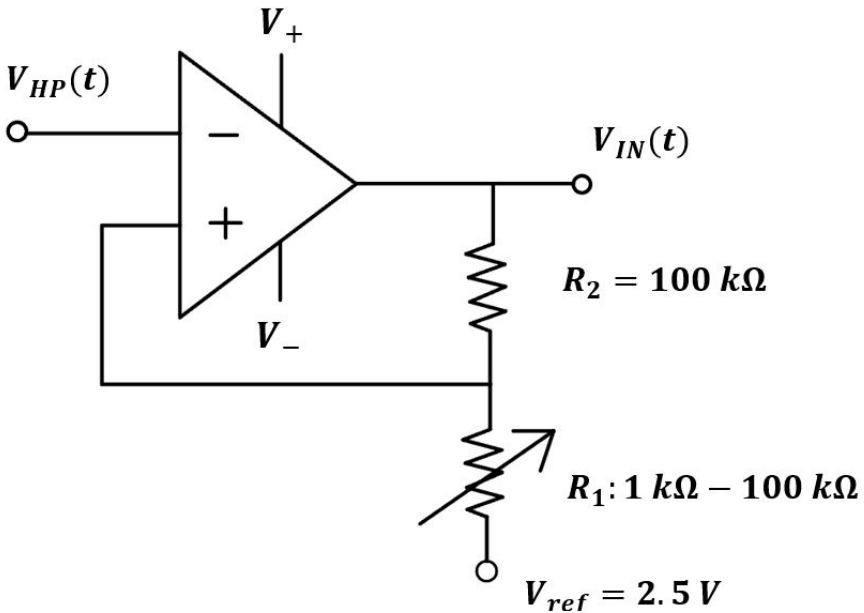
High Pass Filter



- Purpose: conditioning the signal that is generated from the strain gauge
- Typical breath rate: 12-20 breaths per minute
- Estimate min breath rate: fc = 6 breaths per minute → 0.1 Hz

$$\tau_{HP} = R_{HP} \cdot C_{HP} = \frac{1}{\omega} = \frac{1}{2\pi f_c} = \frac{1}{2\pi \cdot 0.1} \approx 1.59s$$

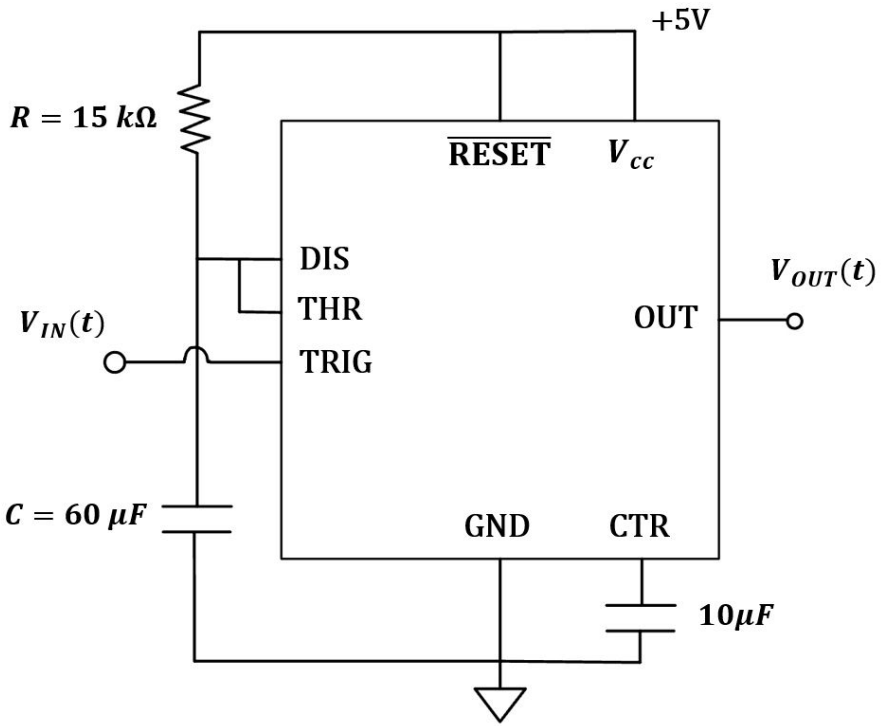
Hysteretic Comparator



- Purpose: convert the wave coming out of the high pass filter and into a square wave signifying either an inhalation (down) or exhalation (up)
- V+ and V- are close to Vref so a really small change in breath can be detected by the comparator
  - Dynamic range
- Potentiometer R1: ranging in 1kΩ - 100kΩ

$$V_{IN}(t) = \begin{cases} V^+ = 5V\ for\ V_{HP}(t) < \dfrac{R_1V^+ + R_2V_{ref}}{R_1 + R_2} \\ V^- = -5V\ for\ V_{HP}(t) > \dfrac{R_1V^- + R_2V_{ref}}{R_1 + R_2} \end{cases}$$

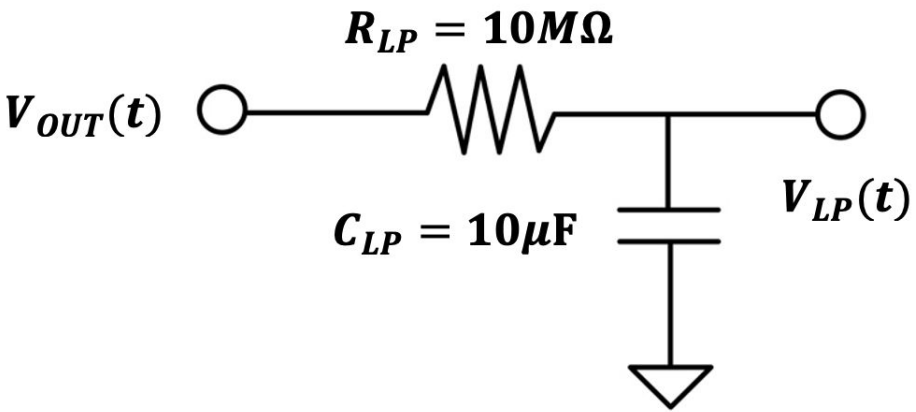
Monostable '555' Timer



- Purpose: turns the square wave from the comparator into an equidistant square wave that represents the breathing rate

$$T = \ln(3)RC = 1.1 \cdot 15k \cdot 60\mu \approx 1s$$

Low-Pass Filter



- Purpose: turns the signal from the monostable ‘555’ timer into a DC signal, which becomes the input to the microcontroller
- Resistor & capacitor values were chosen based on desired cutoff frequency
  - Want a cutoff frequency that is close to zero in order for output to be a DC wave

$$\tau_{LP} = R_{LP} \cdot C_{LP} = \frac{1}{\omega} = \frac{1}{2\pi f_c} = \frac{1}{2\pi \cdot 0.0016} \approx 100s$$

Analog to Digital

- Device converts DC voltage value to breathing rate using the following table:

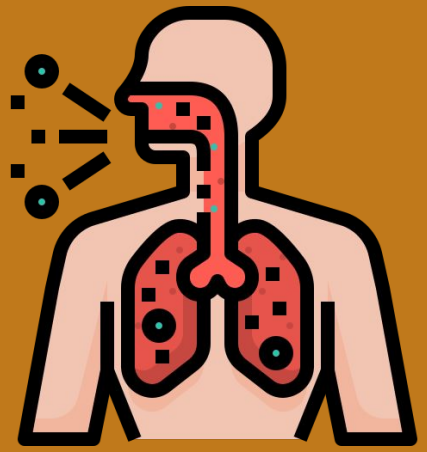
Voltage	Breaths Per Minute
0.500 – 0.582	6
0.583 – 0.666	7
0.667 – 0.749	8
0.750 – 0.832	9
0.833 – 0.915	10
0.916 – 0.999	11
1.000 – 1.082	12
1.083 – 1.166	13
1.167 – 1.249	14
1.250 – 1.332	15
1.333 – 1.415	16
1.416 – 1.499	17
1.500 – 1.582	18
1.583 – 1.666	19

Voltage	Breaths Per Minute
1.667 – 1.749	20
1.750 – 1.832	21
1.833 – 1.915	22
1.916 – 1.999	23
2.000 – 2.082	24
2.083 – 2.166	25
2.167 – 2.249	26
2.250 – 2.332	27
2.333 – 2.415	28
2.416 – 2.499	29
2.500 – 2.582	30
2.583 – 2.666	31
2.667 – 2.749	32
2.750 – 2.832	33

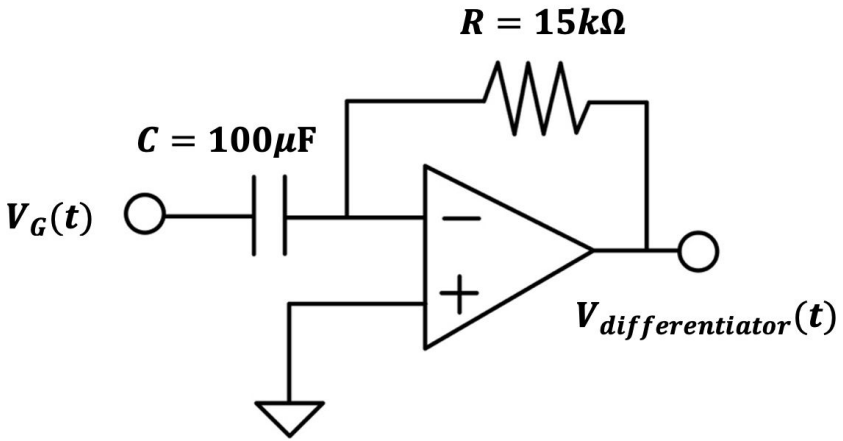
Voltage	Breaths Per Minute
2.833 – 2.916	34
2.917 – 2.999	35
3.000 – 3.082	36
3.083 – 3.166	37
3.167 – 3.249	38
3.250 – 3.332	39
3.333 – 3.416	40
3.417 – 3.499	41
3.500 – 3.582	42
3.583 – 3.666	43
3.667 – 3.749	44
3.750 – 3.832	45
3.833 – 3.916	46
3.917 – 3.999	47

Voltage	Breaths Per Minute
4.000 – 4.082	48
4.083 – 4.166	49
4.167 – 4.249	50
4.250 – 4.332	51
4.333 – 4.416	52
4.417 – 4.499	53
4.500 – 4.582	54
4.583 – 4.666	55
4.667 – 4.749	56
4.750 – 4.832	57
4.833 – 4.916	58
4.917 – 4.999	59
5.000 – 5.082	60

Breathing Monitor - Breathing Depth



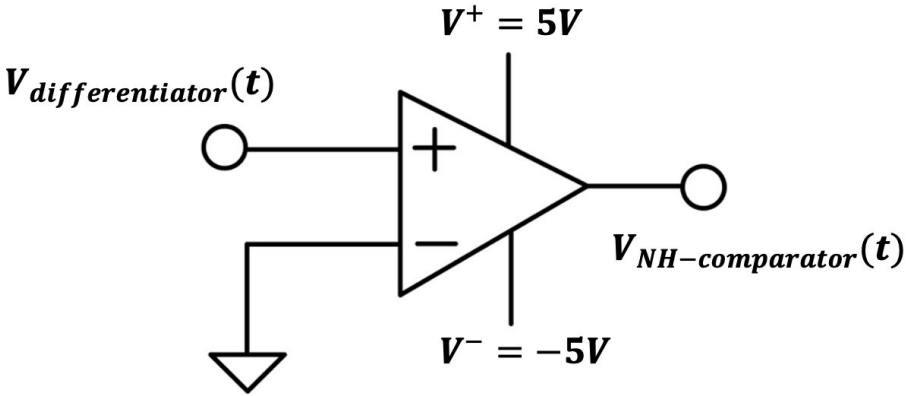
Differential Amplifier



- Purpose: calculates the derivative of the signal, which allows us to know the maximum and minimum values of the signal from the zeros of derivative
- Built-in differentiator high-pass filter component

$$V_{differentiator}(t) = -j\omega RC \cdot V_G(t) = -RC \cdot \frac{dV_G(t)}{dt}$$

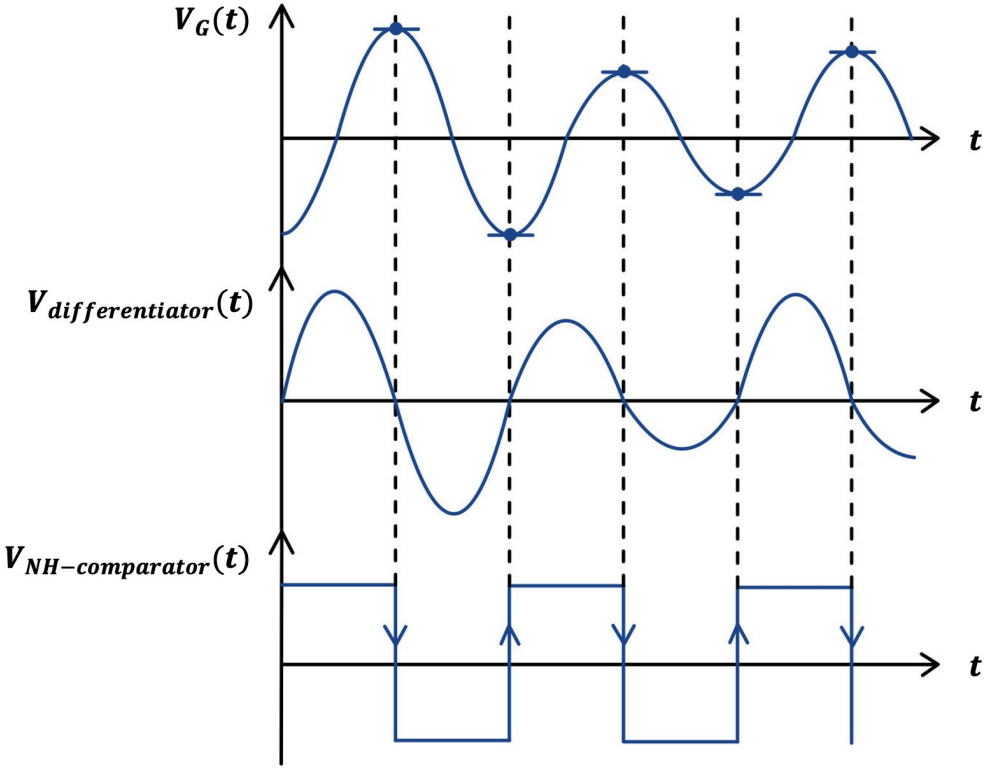
Non-hysteretic comparator



- Purpose: takes the output of the differential amplifier and turns it into a square wave, indicating peak voltage times from the strain gauge for the microcontroller
- Rising/falling edge: time at inhalation/exhalation time

$$V_{NH-comparator}(t) = \begin{cases} V^+ = 5V \ for \ V_{differentiator}(t) > 0 \\ V^- = -5V \ for \ V_{differentiator}(t) < 0 \end{cases}$$

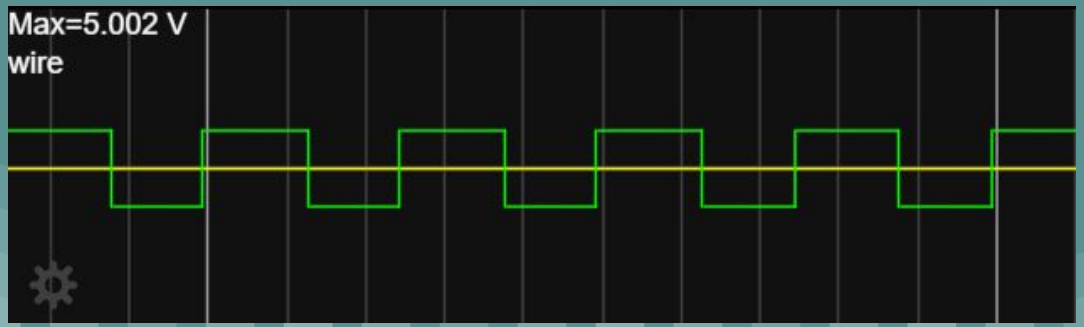
Breathing Depth Waveform Analysis



Waveform Analysis

Branch 1 - Breathing Rate

Branch 2 - Breathing Depth



Algorithm of Microcontroller from Breathing Monitor to Computer

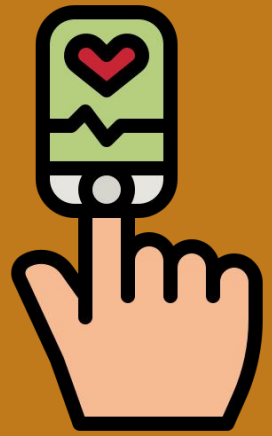
1. Write ON, start of breath recording operation enabled:
    a. Set time sampling rate to at every 0.001 seconds
2. Initialize variables
    a. last_voltage = 0; last_voltage_sample = 0
    b. Create an empty matrix for peak_times [] and peak_voltages [] and store in MATLAB
    c. Do an initial min and max breathing depth trial to determine lung volume associated with voltage values
3. The edge going down on the square wave in the waveform shows the time that the microcontroller needs to go back to the smoothed input curves available from the low pass filter and take a voltage sample.
4. For each samples:
    a. If last_voltage is positive and current_voltage is negative
        i. Record time corresponding to current_slope in peak_times []
        ii. Record the voltage measurement at this time in peak_voltages []
    b. If last_voltage is negative and current_voltage is positive
        i. Record time corresponding to current_slope in valley_times []
        ii. Record the voltage measurement at this time in valley_voltages []
    c. Set last_voltage = current_voltage
5. Subtract the absolute value of the first component in the valley_voltage [] with the absolute value first component in the peak_voltage [] and store in magnitude_of_breath_voltage []
6. Based on initializations determine the magnitude_of_breath [] in liters (tidal volume) from magnitude_of_breath_voltage []
7. Overlay magnitude_of_breath [] values as a square wave with pulse oximeter curve for analysis

Arduino code from Breathing Monitor to Matlab

```ino
// Libraries
#include <Arduino.h>

// Constants
const int ANALOG_PIN = A0;
const float SAMPLING_RATE = 0.001; // Sampling rate in seconds (0.001 seconds = 1kHz)
const int THRESHOLD = 512; // Threshold for detecting peaks and valleys
const int MATRIX_SIZE = 100; // Size of the arrays for peak and valley times and voltages

// Variables
float last_voltage = 0;
float peak_times[MATRIX_SIZE];
float peak_voltages[MATRIX_SIZE];
float valley_times[MATRIX_SIZE];
float valley_voltages[MATRIX_SIZE];
float magnitude_of_breath_voltage[MATRIX_SIZE];
float magnitude_of_breath[MATRIX_SIZE];

// Function to initialize variables and arrays
void initialize() {
  last_voltage = 0;
  memset(peak_times, 0, sizeof(peak_times));
  memset(peak_voltages, 0, sizeof(peak_voltages));
  memset(valley_times, 0, sizeof(valley_times));
  memset(valley_voltages, 0, sizeof(valley_voltages));
  memset(magnitude_of_breath_voltage, 0, sizeof(magnitude_of_breath_voltage));
  memset(magnitude_of_breath, 0, sizeof(magnitude_of_breath));
}

void setup() {
  // Start serial communication
  Serial.begin(9600);
  initialize();
}
```

```ino
void loop() {
  // Set time sampling rate
  delayMicroseconds(SAMPLING_RATE * 1000000); // Delay in microseconds

  // Read analog input
  int current_voltage = analogRead(ANALOG_PIN);

  // Detect peaks and valleys
  if (last_voltage > THRESHOLD && current_voltage < -THRESHOLD) {
    // Record peak
    peak_times[0] = millis(); // Record time
    peak_voltages[0] = last_voltage; // Record voltage
  } else if (last_voltage < -THRESHOLD && current_voltage > THRESHOLD) {
    // Record valley
    valley_times[0] = millis(); // Record time
    valley_voltages[0] = current_voltage; // Record voltage

    // Calculate magnitude of breath
    magnitude_of_breath_voltage[0] = abs(valley_voltages[0]) - abs(peak_voltages[0]);
    // Determine magnitude_of_breath from magnitude_of_breath_voltage
    // ... Implement this calculation based on your specific requirements

    // Overlay magnitude_of_breath values for analysis
    // ... Implement this overlaying process as needed
  }

  // Update last_voltage
  last_voltage = current_voltage;
}
```

Pulse Oximeter



Pulse Oximeter

Functions

- Measure arterial blood oxygen saturation
- Hemoglobin absorbance difference [5]

[Beer-Lambert's Law]

Advantages

- Non-invasive
- Accurate
- Real-time monitor
- Portable

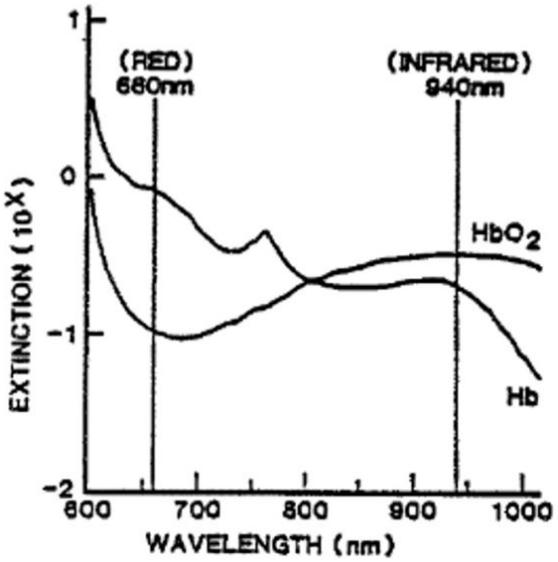
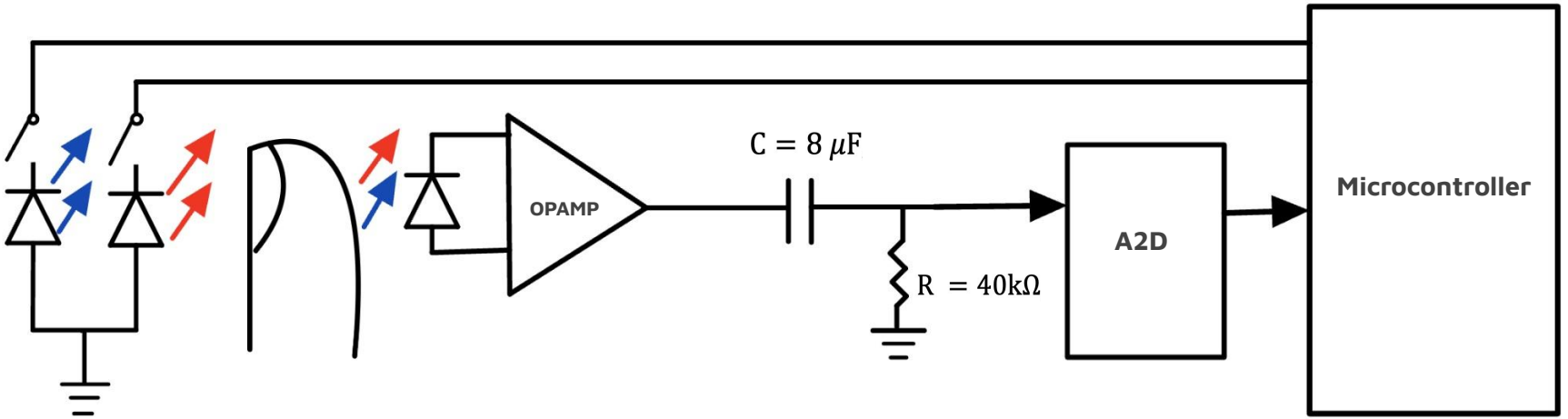


Fig 1. Oxyhemoglobin (HbO_2) and reduced hemoglobin (Hb) exhibit markedly different absorption (extinction) characteristics relative to red light at 660 nm and infrared light at 940 nm. (Courtesy of Ohmeda)

Pulse Oximeter

Components

- Red LED light (650nm) [2]
- Infrared LED light (910nm) [2]
- Photodetector
- High Pass
- Microcontroller



Pulse Oximeter

Lowering noise, improving SNR

High-pass Filter

Purpose

- Remove noise from:
  - Respiration (0.2 ~ 0.4 Hz)
  - Motion artifacts (>0.1Hz)
  - Ambient light
- Passes unattenuated AC signal

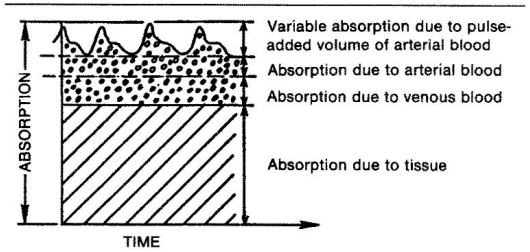


Fig 3. Tissue composite shows dynamic and static components affecting light absorption. (Courtesy of Ohmeda)

Parameters

- Cut-off frequency: fc = 0.5 Hz [2]

$$f_c = \frac{1}{2\pi RC} = 0.5\text{Hz} \qquad R = 40\text{k}\Omega,\ C = 8\,\mu\text{F},\ f_c = 0.497\text{Hz}$$

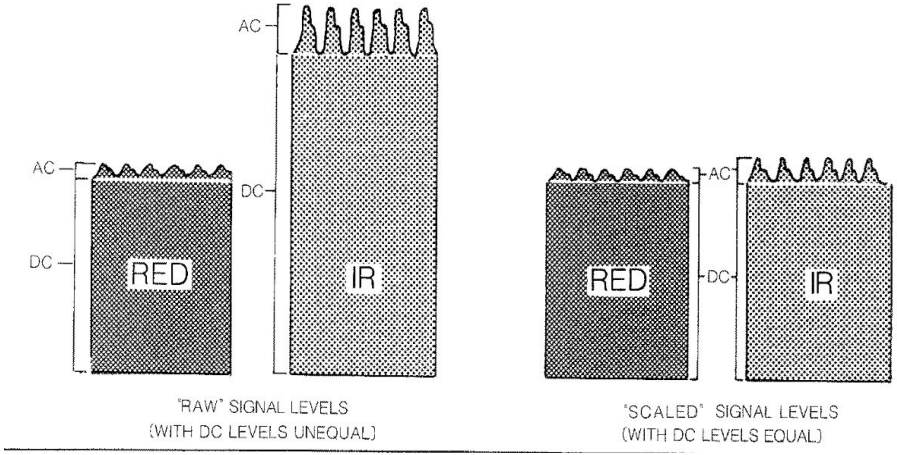
Pulse Oximeter

Scaling

- Infrared light signal normalized by red light signal

  IR(DC) = R (DC)

  IR(AC) = IR(AC)/R(AC) [5]



Computation

- SaO2 = R/IR (of AC signal)

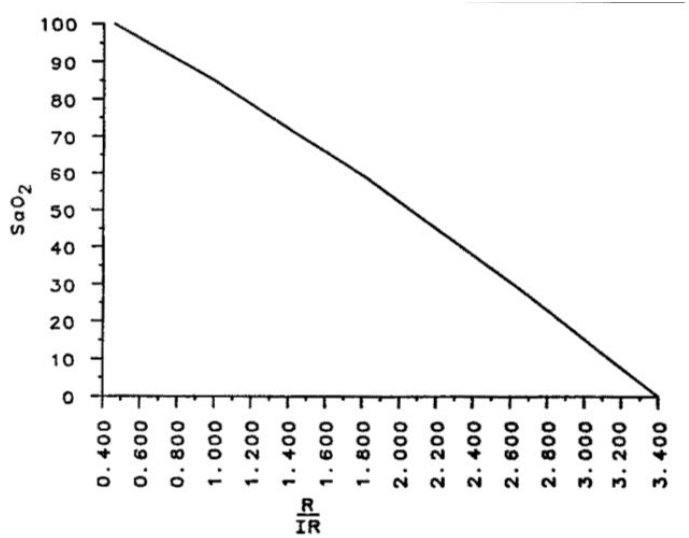


Fig 5. Relationship of red (R)/infrared (IR) numeric ratio value to arterial oxygen saturation (SaO_2).

Algorithm of Microcontroller from Pulse Oximeter to Computer

1. Initialize
    a. Set up pulse oximeter sensor with red LED, infrared LED, and photodetector
    b. Set up microcontroller with necessary input and output pins
    c. sampling_rate = 0.001 seconds
2. Begin loop: start reading raw data from photodetector
3. Data processing
    a. Process raw data to calculate oxygen saturation (%)
    b. Store calculated oxygen saturation value associated with each time in a two matrices in MATLAB (oxygen_saturation []; time [])
4. Plot data on same graph as breath monitor data with a secondary “y” axis and time continuing the same as the “x” axis
    a. Oxygen saturation vs time
    b. Breath depth (liters) vs time
5. Display update: update display with newly plotted graph
6. Repeat and continue reading data

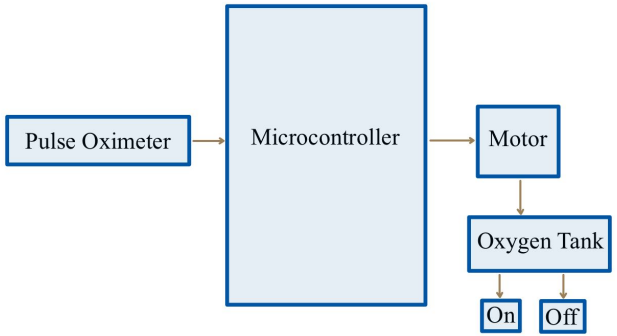
Arduino Code from Pulse Oximeter to Computer

```ino
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_MAX30105.h>
#include <SoftwareSerial.h>

Adafruit_MAX30105 particleSensor;
SoftwareSerial mySerial(2, 3); // RX, TX

unsigned long samplingInterval = 1000; // Sampling interval in milliseconds (1 second)
unsigned long previousMillis = 0;

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30105 was not found. Please check wiring/power.");
    while (1);
  }
  particleSensor.setup();
  particleSensor.setPulseAmplitudeRed(0x0A);
  particleSensor.setPulseAmplitudeGreen(0); // Disable green LED

  // Initialize MATLAB communication
  mySerial.println("Initializing MATLAB...");
}

void loop() {
  unsigned long currentMillis = millis();

  // Check if it's time to sample
  if (currentMillis - previousMillis >= samplingInterval) {
    previousMillis = currentMillis;

    // Read raw data from photodetector
    uint32_t IRValue = particleSensor.getIR();
```

```ino
    // Data processing
    float oxygenSaturation = processRawData(IRValue);

    // Store data in MATLAB matrices
    sendDataToMATLAB(oxygenSaturation, currentMillis);

    // Display update
    updateDisplay(oxygenSaturation, currentMillis);
  }
}

float processRawData(uint32_t IRValue) {
  // Implement your data processing algorithm here to calculate oxygen saturation
  // For example:
  // float oxygenSaturation = calculateOxygenSaturation(IRValue);
  // return oxygenSaturation;
}

void sendDataToMATLAB(float oxygenSaturation, unsigned long time) {
  // Send data to MATLAB through serial communication
  mySerial.print("oxygen_saturation[");
  mySerial.print(time);
  mySerial.print("] = ");
  mySerial.print(oxygenSaturation);
  mySerial.println(";");

  mySerial.print("time[");
  mySerial.print(time);
  mySerial.print("] = ");
  mySerial.print(time);
  mySerial.println(";");
}

void updateDisplay(float oxygenSaturation, unsigned long time) {
  // Implement code to update the display with newly plotted graph
  // For example:
  // Plot data on the same graph as breath monitor data with a secondary “y” axis
  // using appropriate libraries and methods
```

Algorithm from Microcontroller to Oxygen Tank

1. If the pulse oximeter data goes below the threshold, the microcontroller will send a signal to the oxygen tank telling it to turn on the motor that turns on the oxygen tank and allows the oxygen levels to return to normal.
2. Initialize
    a. Set lower_threshold_value = 95% (pulse oximeter returns a value for % of oxygen in blood; every person's threshold may be different for recovery application)
    b. Set upper_threshold_value = 99% (for athletics application)
    c. Set motor_status = OFF
3. Begin loop: read oxygen saturation sample from pulse oximeter every 0.001 seconds
4. Check Oxygen Saturation
    a. If oxygen_saturation < threshold_value
        i. If motor_status == OFF
            1. Set motor_status == ON
    b. Else if oxygen_saturatio= threshold_value
        i. If motor_status == ON
        ii. Set motor_status = OFF
5. Repeat for every sample to control oxygen levels



Arduino Code from Microcontroller to Oxygen Tank

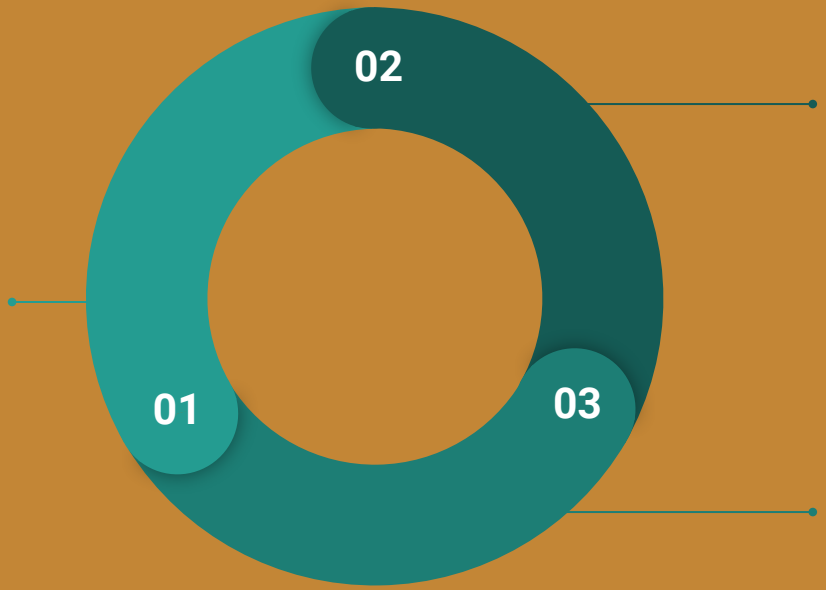
```ino
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_MAX30105.h>

#define LOWER_THRESHOLD 95
#define UPPER_THRESHOLD 99

#define MOTOR_PIN 10

Adafruit_MAX30105 particleSensor;

void setup() {
  Serial.begin(9600);

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30105 was not found. Please check wiring/power.");
    while (1);
  }
  particleSensor.setup();
  particleSensor.setPulseAmplitudeRed(0x0A);
  particleSensor.setPulseAmplitudeGreen(0); // Disable green LED

  // Initialize motor pin
  pinMode(MOTOR_PIN, OUTPUT);
  digitalWrite(MOTOR_PIN, LOW); // Initially turn off the motor
}

void loop() {
  // Read oxygen saturation sample from pulse oximeter every 0.001 seconds
  uint32_t oxygenSaturation = readOxygenSaturation();

  // Check oxygen saturation
  checkOxygenSaturation(oxygenSaturation);

  delay(1); // Delay to achieve approximately 1 ms sampling rate
}
```

```ino

uint32_t readOxygenSaturation() {
  // Read oxygen saturation sample from pulse oximeter
  return particleSensor.getIR();
}

void checkOxygenSaturation(uint32_t oxygenSaturation) {
  // Check if oxygen saturation is below the lower threshold
  if (oxygenSaturation < LOWER_THRESHOLD) {
    // If motor is currently off, turn it on
    if (!digitalRead(MOTOR_PIN)) {
      digitalWrite(MOTOR_PIN, HIGH);
      Serial.println("Motor turned ON");
    }
  }
  // Check if oxygen saturation is above or equal to the upper threshold
  else if (oxygenSaturation >= UPPER_THRESHOLD) {
    // If motor is currently on, turn it off
    if (digitalRead(MOTOR_PIN)) {
      digitalWrite(MOTOR_PIN, LOW);
      Serial.println("Motor turned OFF");
    }
  }
}
```

Application Overview

Blood oxygen levels become too low

Various factors such as fatigue and poor circulation cause blood oxygen levels to drop.



Pulse oximeter detects change

The pulse oximeter takes in data with every breath and is combined with a circuit that sets off a trigger when below a healthy blood oxygen threshold of 95%.

Oxygen tubes increase oxygen output in liters per minute

Once this trigger is activated, oxygen tubes are activated to increase oxygen dispensal until the pulse oximeter reading returns to above the threshold.

Advantages/Applications

Athletic Performance in Elite Running

- Altitude training simulation to reduce oxygen intake per breath
    - Physiological effect: increases quantity of red blood cells over time which increases transport of oxygen to muscle cells (more cellular respiration) [4]
- Circuit provides a signal to an oxygen mask that controls the level of oxygen available while running on a treadmill
    - Negative feedback loop: blood oxygen levels above set level → mask reduces the amount of oxygen via microcontroller

Hospital Stay Surgery Recovery

- After surgery, patients walk around the hospital to promote blood flow and recovery
- Device can be connected to oxygen tubes controlled via a microcontroller to ensure blood oxygen levels remain above threshold. [3]
- Allows recovery while keeping patient safe

Limitations

- Movement while running could trigger a false breath and affect the breathing rate
- More efficient noise elimination methods

References

[1] https://pubs.aip.org/aip/rsi/article/83/10/104708/359654/Optimal-filter-bandwidth-for-pulse-oximetry

[2] https://www.ee.columbia.edu/~kinget/EE6350_S14/PPG6350_Web/files/Datasheet%20PPG.pdf

[3] https://iopscience.iop.org/article/10.1088/1742-6596/2318/1/012022/pdf

[4] https://utswmed.org/medblog/high-altitude-training/

[5] https://link.springer.com/article/10.1007/BF01617328

Thank you to Professor Cauwenberghs, Samira, Vikrant, and Adyant for all the help!