**Breathing Monitor**

1. Intro

Chest wall movement can be detected using strain gauges to measure breathing rate. This project aims to demonstrate how this signal could be measured, parsed & processed, and displayed to a user.

1.2. Breadboard/Circuit schematic

A circuit board with wires connected to it

Description automatically generated

Fig 1. Breadboard schematic

A circuit board with wires and numbers

Description automatically generated

Fig 2. Circuit schematic

A simple circuit in which a voltage divider allows a microcontroller (depicted as an Arduino Uno) to measure the change in voltage as resistance changes according to strain gauge deformation (depicted as a flex sensor).

With chest wall movement, resistance of the strain gauge changes. Subsequent change in voltage is read as an analogue signal by the microcontroller.

1.3. Pre-processing

Noise is generated from vibration, activity of other body muscles (including cardiac activity), poor electrode contact, and inducted electrical noise.

Chest wall movement is presumed to be smooth and greater in magnitude relative to sources of noise. Sudden changes in sensor value can be ‘smoothed out’ using a moving average filter, averaging out the last few data points decreasing the influence of sudden movement.

This filter introduces some delay from real time movement to trace signal, however data required to find peak frequency in the frequency domain is preserved.

1.3. Arduino output

Whilst the Arduino IDE can plot sensor data via the serial plotter, its function is limited. Therefore, the Arduino outputs to python via the serial communications function.

This script prints current time and detected sensor value to the serial monitor, which can then be read into python for a live plot.

This data can be saved using python libraries for future analysis.

Pseudocode:  
# Include serial library for serial communications  
# Set specific pin, output mode, and Baud rate (maximum transfer speed)

Print to serial monitor: ‘Time (ms), Sensor value (arb) // Print header labels to serial monitor

Loop:

Float sensorValue  
sensorValue = pin voltage  
  
//Moving average filter; average the last 10 values  
For loop x10:  
 sum = sum + sensorValue  
 averageVaue = sum / 10  
  
// Print time, raw sensor value, and averaged filtered value, and enter new line in serial monitor to separate data points  
Print current time  
Print sensorValue  
Print averageValue  
Print empty line

1.4. Python Libraries

Using the serial library, the appropriate port can be selected, and constants set in order allow the Arduino’s output can be read into python. Matplotlib then allows this data to plotted in real time updating graphs.

The signal can be processed using the NumPy and SciPy libraries. To have a constantly updating processed signal, the Collections module allows use a ring buffer to store the most recent data points to be processed (a list like container in which data can be appended or popped at the start or beginning).

Data can be saved to a CSV file using the CSV module. The Pandas library allows data from CSV files to be viewed/analysed later. Here, it is used to run mock data from a CSV file as a simulation.

Other modules used include Itertools for indexing purposes.

1.5. Python script

A python script can be written which parses data from an Arduino (or in this case as CSV file containing mock noisy data using the Pandas library) and append the signal data to empty lists. A ring buffer (using is used to store the more recent values. A Fast Fourier transform can then be calculated using the SciPy and NumPy libraries for data in this ring buffer.

Matplotlib is then used to plot an updating graph displaying a live trace of sensor data vs time, and a spectrum analyser showing the frequency domain of the signal. Once the ring buffer is full, the FFT calculation can occur, and the highest peak frequency in the spectrum is determined to be the breathing rate and displayed on the graph.

The Pandas library can then be used to save the data received to a CSV file for future reference.

Pseudocode:  
# Include serial, Matplotlib, CSV, SciPy, NumPy, and other needed libraries  
# Identify and set correct port and Baud rate  
  
Open serial communication  
Initialise empty lists; xread (time) and yread (sensor value)  
Set minimum number of data points for FFT calculation  
Initialise ring buffer  
Create and initialise plot  
  
Read and Process data:  
Append data to xread, yread and ring buffer  
// Wait until ring buffer is full to perform FFT, otherwise return calibrating message  
If length of ring buffer is equal or greater than minimum number of data points:  
Calculate FFT   
Calculate power spectrum density  
Find peak frequency  
Return FFT, power spectrum density, peak frequency  
Else:  
Return calibrating message  
  
Update graph:  
Run ‘Read and Process data’  
Plot live trace (xread, yread)  
Plot frequency power (power, frequencies)   
// Detect peak frequency which is presumed to be breathing rate  
Display peak frequency  
  
On close:  
Save data to CSV   
  
Animate and show plot; run ‘Update graph’  
Run on close when closed

1.6. Mock data example  
A screenshot of a computer

Description automatically generated

Fig 3: Mock sinusoidal data

A screenshot of a computer

Description automatically generated

Fig 4: Addition of Gaussian noise

Mock data was generated with the addition of two sinusoids to mimic chest wall movement with interference from some other movement e.g., cardiac activity. Gaussian noise is added to mimic noise from other sources. This is saved to a CSV file which can then be read and parsed.

A graph of a frequency and a frequency

Description automatically generated with medium confidence

Fig 5: Live trace, calibration message whilst ring buffer fills

A graph of a frequency and breathing monitor

Description automatically generated with medium confidence

Fig 6: Spectrum analyser showing power of different frequencies in original signal and peak frequency

Matplotlib displays a live trace. A calibrating message is displayed as the ring fills before the FFT calculation can occur. Once full, the frequency domain is displayed. In this example, the two sinusoid frequencies can be seen, as well as random noise at baseline. The peak frequency is presumed to be the breathing rate.

1.6. Further development  
The filtering method used in the above example is more of a proof of concept rather than being appropriate for real world use. Instead, a more appropriate filter can be designed using functions in the scipy.signal package. This is beyond the scope of this current project.

Bluetooth wireless connectivity from an Arduino to a PC or other device is relatively easy to implement provided an appropriate microcontroller is used, using the same Arduino library.

For end users, python has many toolkits to create a graphical user interface for ease of use. For this purpose, a toolkit such as PySimpleGUI is appropriate. This is not currently implemented due to time constraints.