Department of Electronic Engineering

INQ2100742 - Scientific Computing with Python

Respiratory Rate Estimation

Author:
Andrea Robinson (2099851) & Rikke Melbø Nielsen (2099691)

Date

Table of Contents

1 Introduction

In this project, the data from an accelerometer and a gyroscope is used to estimate the Respiratory Rate (RR) of a healthy subject. The data is collected using a MuSe (Multi-Sensor miniaturized, low-power, wireless IMU) [1], with the estimation of Respiratory Rate performed on two distinct datasets. In the first dataset, the MuSe is positioned at the center of the sternum, with the subject lying supine, alternately on their left and right sides. The second dataset comprises measurements taken with the sensor placed directly on the subject's chest, over a sweater. The two datasets are referred to as `center_sternum` and `chest_sweater`, respectively.

Given that our group consisted of only two members, we both contributed to all aspects of the development of the code. However, each of us took the lead in certain tasks, which are detailed in the explanation of the code in Section 2.

2 Python implementation of project

2.1 Data preparation

The data was read from text files using the pandas library [6]. We utilized the `read_csv` function with `delim_whitespace=True` to correctly parse our whitespace-delimited text files. We selected the relevant columns using the `.iloc` method, which allows us to access particular segments of the DataFrame. This step ensures that we work only with the data necessary for estimating the Respiratory Rate (RR).

During the data import process, there was an issue with the column names being incorrectly interpreted. Consequently, we assigned new and accurate names to the columns in the refined dataset to correct this error.

Andrea Robinson did the reading of data while Rikke Melbø Nielsen did the plotting of data.

The code for plotting the data comprises two main functions for processing and visualizing accelerometer and gyroscope data from both files, `center_sternum` and `chest_sweater`. The `plot_data` function utilizes Plotly to create the plot. The `create_time_axis` function calculates the end time based on the number of data points and sampling frequency, generating a time axis using NumPy's `linspace` [4].

By looking at the plotted data, it is clear that the measurements have noise at the start and end. To make sure this noise doesn't affect the final estimation of the Respiratory Rate (RR), we removed the first 10% and the last 15% of the data.

In the task of estimating Respiratory Rate (RR) from multi-axis sensor data, we simplifed the problem by focusing on a single axis for both the accelerometer and the gyroscope data. This choice was driven by a desire to reduce the complexity of our analysis, and could help us more accurately figure out the Respiratory Rate (RR).

For the accelerometer, we found that the z-axis provided a more pronounced variation in magnitude when compared to the y- and x-axes. We anticipate that this axis will likely reveal a more distinct pattern or trend that corresponds to the respiratory rate.

Similarly, for the gyroscope data, we applied the same rationale to select the x-axis. Our observations suggested that the x-axis might offer the clearest indication of respiratory-induced movements.

We worked together to decide which measurements to discard and which axes would be best to base our estimation on.

2.2 Frequency Analysis

The implementation of the frequency of the data was done by Andrea Robinson. However, the discussion of the resulting plots was done together.

A Fourier analysis was performed on the collected data to understand the frequency components present in the signals from the accelerometer and gyroscope. This analysis is crucial as it helps in identifying the dominant frequencies within the data that correspond to the respiratory rate.

The frequency analysis was initiated by detrending the data to remove any DC component that could skew the frequency spectrum. This was done using the detrend function from the scipy.signal library [5]. The Fast Fourier Transform (FFT) from the numpy library [3] was then applied to translate the time-domain data into the frequency domain.

We focused on the positive frequencies and converted these to Beats Per Minute (BPM) to relate them to the expected respiratory rate range, simply by multiplying the frequency axes by 60 seconds.

The plots showing the frequency analysis were rather noisy, and a clear peak corresponding to a possible respiratory rate of a human could not be seen. This indicates that the data needs filtering.

2.3 Implementation of the Filter

In this section, we discuss the choices made in designing a filter to extrapolate the respiratory rate signal from the data. The task was implemented by Rikke Melbø Nielsen.

The goal is to remove frequencies related to the pulse, typically in the range of 40 bpm to 100 bpm. Initially, we intended to analyze the frequency spectrum, but due to noisy signals, we opted to design the filter based on the anticipated pulse and respiratory rate per minute (RPM), expected to be within the range of 12-18, as indicated by [2]. One of our attempts involved implementing a lowpass filter with cutoff frequencies set at 30 and 40 bpm. However, it became evident that a bandpass filter, with cutoff frequencies corresponding to a minimum bpm of 8 and a maximum bpm of 30, outperformed in capturing the characteristics resembling the respiratory rate.

The functions used for filter implementation are listed in Listing 1.

Listing 1: Python functions used for filter implementation.

```
hz(bpm)
sos_butterworth_filter(data, order, min_bpm, max_bpm, fs)
```

The function `hz(bpm)` is employed to convert bpm to hertz. This conversion is useful for the filtering process done by the bandpass since the python built in-function `scipy.signal.butter` which is used inside the function `sos_butterworth_filter(data, order, min_bpm, max_bpm, fs)` takes frequency in Hz as parameters. The function textttsos_butterworth_filter(data, order, min_bpm, max_bpm, fs) for applying a bandpass Butterworth filter.

The function `hz(bpm)` serves to convert beats per minute (bpm) to hertz. This choice is motivated by the desire to work with bpm because the pulse rate and respiratory rate per minute (RPM) are typically expressed in beats per minute. Using bpm ensures consistency in the representation of these rates, making it easier to interpret and compare them directly. It is necessary to convert bpm to Hz for the bandpass filtering process since the Python built-in function `scipy.signal.butter`, employed within
`sos_butterworth_filter(data, order, min_bpm, max_bpm, fs)`, takes frequency in Hz as parameters.

Detrending the data was essential to eliminate a transient response observed in the filtered signals, preventing the introduction of unwanted artifacts. This process ensures a more accurate representation of the respiratory rate by eliminating the baseline mean.

Plotting the Fast Fourier Transform (FFT) spectrum again with the filtered data displayed a distinct peak, representing the respiratory rate per minute (RPM). The frequency peaks in the filtered signals aligned well with the expected values and we were able to extract a meaningful respiratory rate signal from the given data.

2.4 Extracting the Respiratory Rate per Minute and Comparing the Two Methods for Obtaining the Respiratory Rate

The estimation of the RPM using the seismocardiography (SCG) method yields values of approximately 12.6 RPM from gyroscope readings in the x-direction and 23.26 RPM from accelerometer readings in the z-direction. These estimations are derived from identifying peaks in the filtered frequency spectrum of the data. Notably, the expected respiratory rate for a healthy adult at rest typically falls within the range of 12-18 RPM. The gyroscope-derived estimate aligns well with this expected range, while the accelerometer-derived estimate is outside.

For the ballistocardiography (BCG) method, the estimated respiratory rate per minute is 9.04 RPM, as obtained from both gyroscope readings in the x-direction and accelerometer readings in the z-direction. This lower estimate suggests potential statistical variations in the sensor readings, especially considering it is lower than the expected range for a healthy individual.

Analyzing the data from the `chest_sweater` file proved more straightforward in extracting the RPM, both before and after filtering, due to a distinct and clear peak in the frequency spectrum. In contrast, the data from the `center_sternum` file, when using the SCG method, exhibited higher noise levels, requiring more extensive filtering to reveal the peak in the frequency spectrum. This difference is because the `chest_sweater` file shows a clearer pattern in the original data.

Despite the initial challenges, the SCG method, post-filtration, provided a more realistic estimation of the RPM compared to the BCG method, aligning more closely with the expected respiratory rate values.

3 Bibliography

[1] 221e.com. *Muse*. URL: https://www.221e.com/muse-miniaturized-multi-sensor-imu (visited on 18/01/2024).

[2] Cleveland Clinic. *Vital Signs*. 2023. URL: https://my.clevelandclinic.org/health/articles/10881-vital-signs (visited on 18/01/2020).

[3] NumPy Developers. *Discrete Fourier Transform*. URL: https://numpy.org/doc/stable/reference/routines.fft.html (visited on 18/01/2024).

[4] NumPy Developers. *numpy.linspace*. URL: https://numpy.org/doc/stable/reference/generated/numpy.linspace.html (visited on 24/01/2024).

[5] SciPy v1.11.4 Manual. *scipy.signal.detrend*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.detrend.html (visited on 18/01/2024).

[6] pandas.org. *pandas.read_csv*. URL: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html (visited on 18/01/2024).