Respiratory rate estimation on embedded system

Isabel Morales, Leonardo Martínez Hornak, Alfredo Solari, and Julián Oreggioni

***Abstract*—We present the design, implementation, and results of an algorithm for respiratory rate estimation using respiratory-induced frequency, intensity, and amplitude variation calculated from the infrared (IR) channel of the SEN-15219 board for photoplethysmography (PPG) acquisition. First, the algorithm was developed in Python (on a PC) using synthetic signals and publicly available respiration and PPG data. We also include a graphical user interface to process data from sensors and display vital signs. Later, we ported the algorithm to an MSP432P401R microcontroller to complete our wearable prototype. Results are promissory and show that respiratory rate estimation can be performed on the selected platform with our proposed Fourier Product (FP) method, which results in a Mean Absolute Error of 4.1 using 16-seconds windows of IR-PPG signals.**

***Index Terms*—Respiratory rate estimation, photoplethysmography, signal processing, low-power embedded system**

I. Introduction

Photoplethysmography (PPG) is a technique that uses a light source, mainly red (R) and infrared (IR) light, capable of transmitting different wavelengths and a photodetector. The transmitted light is absorbed and spread throughout human tissue; thus, the reflected light variations correspond to blood volume changes [1]. Electronic devices can provide robust measurements of blood oxygen saturation level (SpO2) and heart rate (HR, expressed in bpm, beats per minute) using PPG in a non-invasively manner [1].

PPG signal can also be used to estimate the respiratory rate (RR, expressed in rpm, respiration per minute), an important physiological indicator. Nonetheless, PPG acquisition poses several challenges, including variability with skin color, age, sex, obesity, artifacts, ambient light, and the finger pressure applied at the measurement point [2].

A good estimation of RR can be obtained using ECG [3], [4]; however, ECG is not a comfortable signal to be acquired in a wearable device. Some state-of-the-art implementations use deep learning techniques to precisely estimate the RR using PPG [5]–[8]. However, it is not feasible to incorporate these techniques in a low-resource microcontroller and obtain adequate autonomy. Finally, some works report estimations of RR using processing techniques feasible to be implemented in a low-power embedded system [9]–[12]. However, to the best of our knowledge, none of them have actually implemented an algorithm in an embedded system, such as those required in a wearable device.

The motivation of this work is to generate an estimation of the RR using non-invasive acquisition methods to contribute to developing a wearable solution. Then, this work presents the implementation of an algorithm for RR estimation that is a variant of already proposed ones in [11], [12]. The main contribution of this work is to implement and characterize a state-of-the-art RR estimation running in a low-resource embedded system. In the near future, we aim to incorporate this algorithm into a wearable vital signs monitoring device (see Fig. 1).

This work was partially funded by CSIC, Universidad de la República, Uruguay (Udelar).

Isabel Morales, Leonardo Martínez Hornak, Alfredo Solari, and Julián Oreggioni are with Facultad de Ingeniería (Udelar). E-mail: imorales@fing.edu.uy.

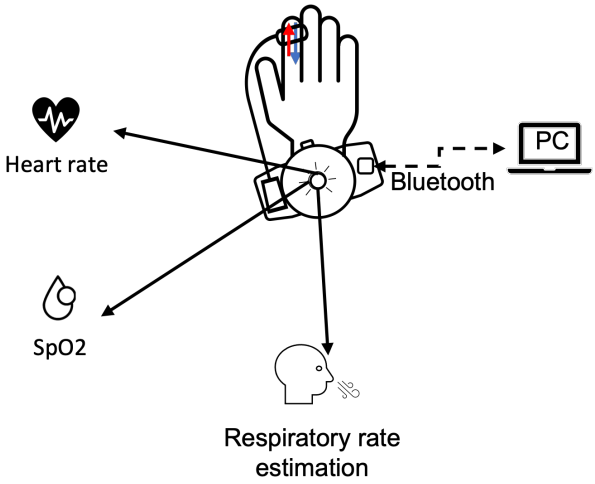

Fig. 1. Wearable vital signs monitoring device.

II. Respiratory rate estimation algorithm

We implemented an algorithm for RR estimation calculating the respiratory-induced frequency variation (RIFV), respiratory-induced intensity variation (RIIV), and respiratory-induced amplitude variation (RIAV) from the PPG signal [11]. We developed a Python version of the algorithm on a PC and validated it using the BIDMC (Beth Israel Deaconess Medical Center, Boston, MA, USA) dataset [13]. Later, the code was migrated to C and ported to a microcontroller. We used the ARM's CMSIS library [14] to implement mathematical functions efficiently.

Algorithm 1 presents our proposal using RIIV as the estimator. To illustrate the functioning of our algorithm, we artificially generated PPG signals (with a fixed HR of 75 bpm) with 3 RR values (12, 18, and 24 rpm). Fig. 2 highlights the main steps of our algorithm. The algorithm for RIFV, RIAV, or other estimators is analogous.

Firstly, the algorithm *standardizes* the signal, subtracting the mean of the signal and dividing it by its standard deviation; then, it filters the raw samples in 16-seconds windows (see Fig. 2-top). The band-pass filter (0.1 - 4 Hz) is divided into two infinite impulse response filters, each of order 4.

Secondly, it detects the peaks of the signal using a non-overlapping 1-second window (SamplingWindow). The de-

Algorithm 1: RR Estimation Algorithm using RIIV

Input: IR PPG raw values ($IRRawSamples$)
Output: Respiration Rate (RR in $_{RR}$)
Result: Respiration rate estimation based on PPG's induced intensity variation due to blood perfusion

```
while RRAlgorithmEnded = 0 do
    IRSamples ← Standardize(IRRawSamples);
    SamplingWindow ← BandPassFilter(IRSamples);
    if WindowCounter < 16 then
        PA ← CalculatePeaks(SamplingWindow);
        AdjustTime(PA);
        WindowCounter++;
    end
    else
        PIA ← AnomalyFilter(PA);
        PIA4 ← ResampleTo4Hz(PIA);
        PIA4FFT ← FFT(PIA4);
        RIIV ← GetAreaOfInterest(PIA4FFT);
        RR ← 60*Max(RIIV);
        RRAlgorithmEnded=1;
    end
end
```

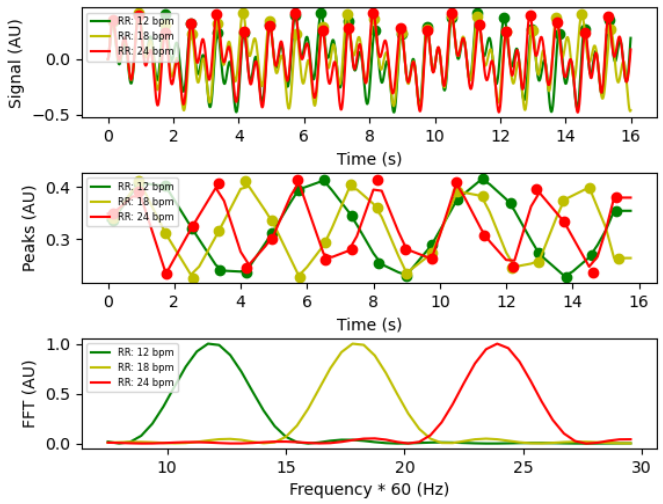

Fig. 2. RR algorithm running in MSP432 processing synthetic PPG signals using RIIV as the estimator. Top: a 16-seconds window of IR PPG samples (standardized and filtered). Center: detected peaks at the corresponding signal intensity value and the interpolation. Bottom: frequency response of the 4 Hz re-sampled intensity variability signal. The maximum corresponds to the RR estimation.

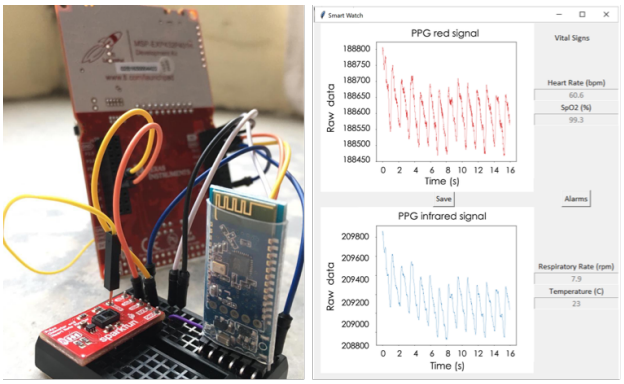

Fig. 3. Device prototype, which consists of an embedded system and a GUI.

tected peaks are those that have at least 40 % of the maximum peak-to-peak amplitude in each window. Then, it generates a peak array (PA) that stores its time and intensity (see peaks marked with colored dots in Fig. 2-top).

Next, the algorithm executes an *anomaly filter* in the PA to eliminate peaks detected at a non-expected time. Peaks that generate an HR lower than 90 % or higher than 115 % of the subject's HR are considered *anomalous* and are discarded. Therefore, a peak intensity array (PIA) is obtained. The PIA is re-sampled to 4 Hz [11]. Due to being unequally spaced, linear interpolation is performed. Fig. 2-center shows the interpolation performed on the PIA.

Finally, the algorithm performs a 512-points FFT (Fast Fourier Transform) on the re-sampled data. The maximum power registered in the area of interest, between 8 and 28 rpm, corresponds to the RIIV, which is our estimation of the RR (see Fig. 2-bottom).

III. Embedded System

The hardware design was previously reported in [15] (see Fig. 3). The SEN-15219 board from Sparkfun acquires the HR, SpO2 (blood oxygen saturation level), and PPG data. The core of the device is the MSP432P401R from Texas Instruments, a 32-bit ARM-Cortex M4F microcontroller with a hardware floating-point unit. It works in two modes: i) low power mode 0 (LPM0) to save energy during the wait times in the communication with the SEN-15219 board, and ii) active mode at 48 MHz. The device wirelessly transmits IR and R PPG signals via an HC-06 Bluetooth module. We used the TI v20.2.6 LTS compiler from Texas Instruments with level 2 (–opt_level = 2) optimizations (global).

The embedded software follows a Function-Queue-Scheduling architecture [16]. We modified the embedded software reported in [15], adding the estimation of RR (Fig. 4 illustrates the software workflow). After system initialization, which includes the configuration of the SEN-15219 board and microcontroller peripherals, the device enters LPM0. Then, the embedded system performs the same tasks periodically. Every 40 ms, it obtains PPG (IR and R), HR, and SpO2 samples from the SEN-15219 board. When 200 ms elapsed, it sends the PPG signal values to the BT module, which are forwarded to the PC for real-time visualization. Finally, every 1 second, it acquires and sends HR, SpO2, and temperature to the PC.

We developed a graphical user interface (GUI) to support our research. The application runs on *Windows 10* and depends on *Pybluez* [17], a Python library for Bluetooth communication. Fig. 3 shows the main view of the GUI. The application enables receiving, storing, processing, and analyzing sensor data. In addition, the application allows the display of raw and processed data from the R and IR channels of the PPG sensor and the configuration of alarms for the future wearable.

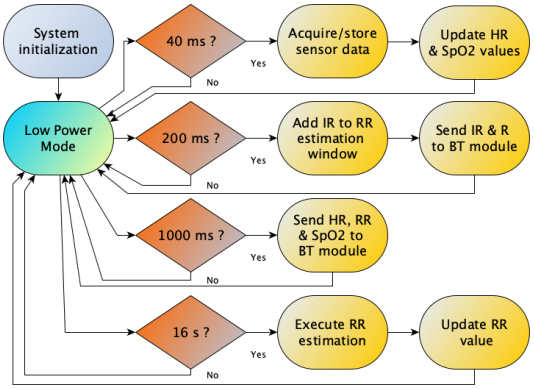

Fig. 4. Embedded software workflow

IV. Experimental Results

In order to assess our algorithm using a controlled setup, the software module responsible for receiving the sample data from the SEN-15219 board via I2C is replaced by a Test Double [18]. The Test Double module supplies data that the microcontroller reads directly from its memory.

We assessed several calculation methods to select the best RR estimator. The evaluation was carried out on 10,000 randomly selected 16-seconds windows samples from the BIDMC dataset [13]. Firstly, we considered RIFV, RIAV, and RIIV as RR estimators. Secondly, the mean of RIFV, RIIV, and RIAV (Mean). Thirdly, the product of the Fourier transforms of RIFV, RIAV, and RIIV and then taking the maxima (Fourier product or FP). FP produces a better result than mixing them linearly or non-linearly after the maximum frequency of each signal is found. Figs. 5 and 6 present the histograms to asses the absolute error.

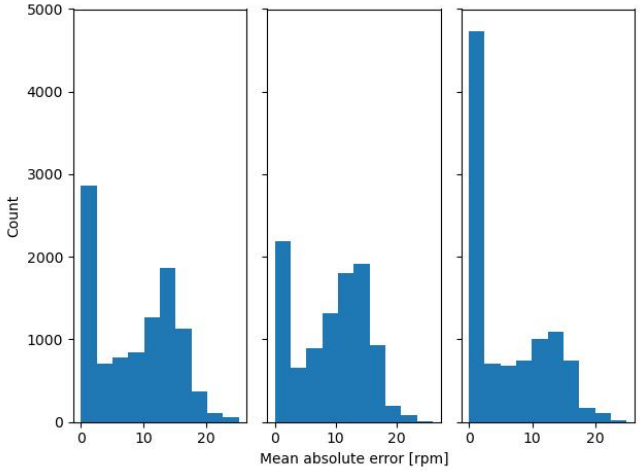

Fig. 5. Absolute error for different RR estimators over 10,000 samples from [13]. Left to right: RIFV, RIAV, RIIV.

To calculate the MAE (Mean Absolute Error), we consider Eq. 1.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} y_i - y_{REF_{RR}} \quad (1)$$

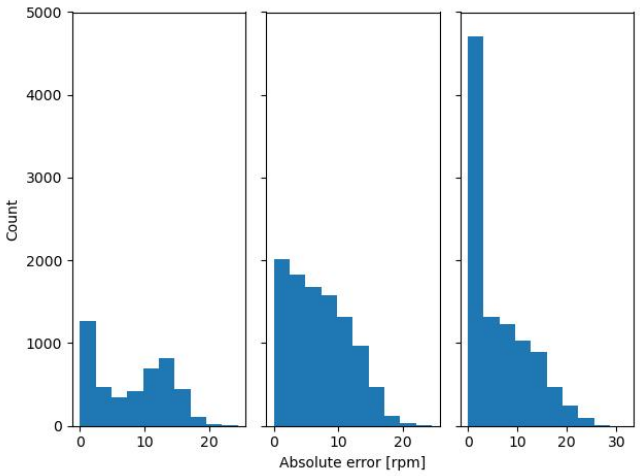

Fig. 6. Absolute error for different RR estimators over 10,000 samples from [13]. Left to right: Karlen et al. [11], Mean, and FP.

where n is the number of aleatory samples taken from all 53 patients of the BIDMC dataset, y_i is our RR estimation, and $y_{REF_{RR}}$ is the dataset RR measured at 1 Hz.

Table I summarizes the results obtained when using these different methods. FP shows a better MAE, and the Mean the best STD (standard deviation of absolute error). [11] shows similar results; however, that approach discarded 3152 samples (more than 30 %). Our proposal obtained similar results to [11], while no samples were discarded.

TABLE I
RR estimation results

Estimator	MAE (rpm)	STD (rpm)
RIFV	4.98	4.24
RIIV	4.38	4.40
RIAV	6.00	4.23
Mean	4.42	3.45
FP	**4.13**	**3.97**
Reproduction of Karlen et al [11]	4.51	3.78
Reproduction of Uguz [12]	4.89	4.32

Table II presents results on some particular patients from the BIDMC dataset using our algorithm performed in Python (running on the PC) and C (running on the microcontroller). The absolute errors of these experiments are below 4 rpm.

TABLE II
RR estimation algorithm results.

		RR estimation (rpm)			
BIDMC	Ref	PC - Python	Abs. error	Micro - C	Abs. error
3	17.0	15.7	1.31	16.0	1.00
5	10.0	9.80	0.18	9.80	0.20
7	20.0	16.6	3.43	17.0	3.00
8	21.0	19.4	1.60	19.2	1.80
11	14.0	12.6	1.40	12.9	1.10
20	16.0	18.1	2.10	17.1	1.10
MAE (rpm)		1.70		1.40	

The implementation of the algorithm on the microcontroller requires 93.3 kB of flash memory (36.5 % of total memory) and 13.5 kB of RAM. The execution time is around 22 ms

TABLE III
RESPIRATORY RATE ESTIMATION ALGORITHMS COMPARISON USING THE BIDMC DATASET

Algorithm	Complexity	Respiratory range (rpm)	Discard samples	Window	MAE (rpm)	STD (rpm)
This Work (FP)	low	8-28	no	16 s	4.13	3.97
Karlen et al [11] made for this work	low	8-28	yes	16 s	4.51	3.78
Karlen et al [11] made in [10]	low	4-65	yes	32 s	5.80	>7.80
Uguz [12] made for this work	low (autoregressive)	8-28	no	16 s	4.89	4.32
Pimentel et al [10]	low (autoregressive)	4-65	no	32 s	4.00	>3.70
Bian et al [5]	high	4-60	no	60 s	2.60	0.40
RRWaveNet [7]	high	?-65	no	16 s	1.87	0.95

at 48 MHz, which makes it possible to estimate RR while sending raw samples to the PC for visualization.

We measure the RMS current drained from the power supply using a standard 3.5-digit multi-meter and Energy Trace from Texas Instruments. The energy consumption of the system was measured considering two operation modes. In *continuous mode*, where the device acquires data and transmits it to a PC, it consumes 128.8 mWh. In *intermittent mode*, where every 256 seconds, a timer wakes the system up from low power mode 3 (LPM3) to acquire data and transmit them for 32 seconds. In this mode, the device consumes 38.9 mWh. The most power-hungry module is the SEN-15219 board (56 mWh, 43.4 %), mainly due to the power needed for the LEDs, followed by the Bluetooth module (43 mWh, 33.3 %) and, finally, the microcontroller (30 mWh, 23.3 %). The autonomy, considering a 235 mAh coin cell lithium CR2032 battery, is 5.5 hours in continuous mode and 18.2 hours in intermittent mode.

V. STATE-OF-THE-ART COMPARISON

Table III shows that implementations with high-complex techniques can estimate the RR using PPG with reasonable precision [5], [7]. However, it is not feasible to incorporate these techniques in a low-resource microcontroller and achieve high autonomy. On the other hand, the proposals that use feasible low-power embedded system processing techniques [10]–[12] are not so precise.

The estimator proposed in [11] is the mean value of RIFV, RIAV, and RIIV when its STD is less than 4 rpm (discarding the rest). Our implementation of [11] achieves a good STD; however, more than 30 % of the samples were discarded. Finally, [10] achieves the best MAE; however, the memory required to process a 32-seconds window is inadequate for a low-resource embedded system.

VI. CONCLUSIONS

This work has shown the feasibility of implementing an algorithm for RR estimation in an MSP432 microcontroller. Preliminary results show that the mean of RIIV, RIAV, RIFV, and the proposed FP method are reasonable estimators of the RR.

Estimating RR accurately from PPG is still an open problem. Even highly sophisticated techniques which are not portable to a microcontroller do not achieve excellent results. The proposed FP method is simple and provides acceptable results. Nonetheless, the RIIV is not so unfavorable and is easier to implement.

ACKNOWLEDGMENT

The authors would like to thank to Núcleo de Ingeniería Biomédica, Comisión Sectorial de Investigación Científica (CSIC, Udelar), and the Uruguayan chapter of the IEEE Circuits and Systems Society.

REFERENCES

[1] J. G. Webster, *Design of pulse oximeters*. CRC Press, 1997.
[2] J. Fine, et al, "Sources of inaccuracy in photoplethysmography for continuous cardiovascular monitoring," *Biosensors*, vol. 11, no. 4, p. 126, 2021.
[3] J. Fan, S. Yang, J. Liu, Z. Zhu, J. Xiao, L. Chang, S. Lin, and J. Zhou, "A high accuracy & ultra-low power ecg-derived respiration estimation processor for wearable respiration monitoring sensor," *Biosensors*, vol. 12, no. 8, p. 665, 2022.
[4] P. H. Charlton, T. Bonnici, L. Tarassenko, D. A. Clifton, R. Beale, and P. J. Watkinson, "An assessment of algorithms to estimate respiratory rate from the electrocardiogram and photoplethysmogram," *Physiological measurement*, vol. 37, no. 4, p. 610, 2016.
[5] D. Bian, et al, "Respiratory rate estimation using PPG: a deep learning approach," in *International conference of the IEEE engineering in Medicine & Biology Society (EMBC)*, 2020, pp. 5948–5952.
[6] V. Ravichandran, et al, "Respnet: A deep learning model for extraction of respiration from photoplethysmogram," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2019, pp. 5556–5559.
[7] P. Osathitporn, et al, "RRWaveNet: A compact end-to-end multi-scale residual CNN for robust PPG respiratory rate estimation," *arXiv preprint arXiv:2208.08672*, 2022.
[8] R. Dai, C. Lu, M. Avidan, and T. Kannampallil, "Respwatch: Robust measurement of respiratory rate on smartwatches with photoplethysmography," in *International Conference on Internet-of-Things Design and Implementation*, 2021, pp. 208–220.
[9] T. Iqbal, et al, "Photoplethysmography-based respiratory rate estimation algorithm for health monitoring applications," *J. of med. and biol. eng.*, vol. 42, no. 2, pp. 242–252, 2022.
[10] M. A. F. Pimentel, et al, "Toward a robust estimation of respiratory rate from pulse oximeters," *IEEE Trans. Biomed. Eng.*, vol. 64, no. 8, pp. 1914–1923, 2017.
[11] W. Karlen, S. Raman, J. M. Ansermino, and G. A. Dumont, "Multiparameter respiratoy rate estimation from the photoplethysmogram," *IEEE Trans. on Biomed. Eng.*, vol. 60, no. 7, pp. 1946–1953, 2013.
[12] D. Uguz, "Design of a multipurpose photoplethysmography sensor to assist cardiovascular and respiratory diagnosis," in *21th Int. Student Conf. on Elect. Eng.*, vol. 21, 2017, pp. 1–7.
[13] A. Goldberger, et al. (2021, 05) BIDMC PPG and Respiration Dataset, PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation. [Online]. Available: https://physionet.org/content/bidmc/1.0.0/
[14] ARM. (2022, 06) CMSIS-DSP Software Libray Version 1.8.0. [Online]. Available: https://github.com/ARM-software/CMSIS_5
[15] L. Martínez Hornak, I. Morales, A. Solari, and J. Oreggioni, "Wearable device prototype for vital signs monitoring," in *XII Congreso Argentino de Sistemas Embebidos, UNLP, La Plata, Buenos Aires, Argentina*. CASE, 2022, pp. 70–72.
[16] D. E. Simon, "Survey of software architectures," in *An Embedded Software Primer*. Addison-Wesley Professional, 1999, pp. 115–136.
[17] Karulis. (2021, 08) Pybluez: Python extension module allowing access to system Bluetooth resources. [Online]. Available: https://pybluez.github.io/
[18] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.