Zero cross algorithm performance on raspberry pi machine for ECG QRS detection

M I Rizqyawan*, M F Amri, A I Simbolon, M A Suhendra, T Haiyunnisa and D E Kusumandari
UPT BPI, Indonesian Institute of Sciences, Bandung 40135, Indonesia

*muha176@lipi.go.id

Abstract. The electrocardiogram (ECG) is one of the most popular instruments in the medical field. Have been developed a smart IoT system for ECG Arrhythmia classification using Android device as the IoT Hub. In order to minimize the computing load on the Android device, some parts of the signal processing are moved to the data acquisition module by using Raspberry Pi 3 as the processor. In the Raspberry machine, a QRS detection system with Zero Cross Algorithm was implemented and its performance was measured. To measure the performance of the algorithm, a timer was placed before and after the execution of the process. The result shows that the algorithm can be run in real-time use in Raspberry Pi 3, although it is a bit slow and produced a high computational load of a single core processor in the machine.

1. Introduction

The electrocardiogram (ECG) is the electrical activity signal of the heart which widely used to diagnose different heart abnormalities [1]. Any difference in the main rhythm of the ECG is called “Arrhythmia”. Automatic detection of arrhythmia [2, 3, 10] is one of the most popular research fields in biomedical signal processing because heart arrhythmias can be used to diagnose heart disease, which is considered one of the deadliest disease. To avoid these deaths, it is necessary to diagnose these arrhythmias accurately and fast [4]. The detection of the QRS complex is the most important task in automatic ECG signal analysis to interpret and decide whether the heartbeat belongs to the normal state or the arrhythmia state [5].

Pan-Tompkins is an algorithm developed by Pan and Willis J. Tompkins that most commonly used to detect the QRS wave of the ECG in real time. This algorithm uses an adaptive threshold after the signal is processed with a bandpass filter, derivative, squaring, and moving average [6]. In order to design an algorithm which is more effective to compute and easily applied to mobile applications, the zero-cross method for QRS detection is proposed. This method uses an addition of high-frequency signal and zero crossing count to detect QRS complex which has a low-frequency oscillation. By employing the zero-crossing count, this method is robust against noise and performs well in the noisy ECG signal [7].

Nowadays, the smartphone has been widely used for heart monitoring system. Stefan Gradl implemented the real-time application for monitoring ECG and providing automated arrhythmia detection on mobile devices [8]. However, due to QRS detection and arrhythmia classification performed by the smartphone, this system caused high computational loads and reduced overall sensitivity for abnormal beat detection. To overcome high computational load, in this paper QRS
detection will be carried by Raspberry Pi 3 as the processor. Raspberry Pi 3 performance in QRS detection system with Zero Cross Algorithm will be evaluated and will be analysed whether it is suitable for real-time use or not.

2. Methodology

We use MIT-BIH arrhythmia database [9] as our testing data for this experiment, using specifically the MLII channel as our raw data. Selected records of the database, named dataset 1 (ds1) in protocol proposed by Chazal et al. [10] is used for the experiment. The algorithm and testing is writes in Python 3 programming language. The processing time of QRS detection of each processing windows is measure in order to examine the computation load required by the algorithm. The processing time is measure by setting up a timer just before and after the process of each steps of QRS detection.

The testing program then loaded and executed on the Raspberry Pi device. We use Raspberry Pi 3 model B as the device for this experiment. It has Quad Core 1.2GHz 64bit CPU with 1GB RAM. The device equipped with some features to support the IoT system such as wireless LAN and Bluetooth Low Energy (BLE) on board, 4 USB 2 ports and HDMI port. It is also contained Micro SD port for device storage the operating system and data. The operating system used in this experiment is Raspbian version 9 with Python version 3.5.3 to run the testing program. The Raspberry Pi 3 can be seen in figure 1.

![Raspberry Pi 3](image)

*Figure 1. Raspberry Pi 3 used in the experiment.*

3. Zero Cross Algorithm

The processing unit on Raspberry that we used on this paper has low computing ability compared to the CPU with big computation processor or GPU. The major factor we use raspberry because we need to build it a small device for a mobile monitoring system that affordable but has a big capability. Because of the limited resources from the Raspberry, we need an algorithm that can meet the requirements, such as low computational load and less time consuming but not compromising on the performance. We used Zero Crossing Count [7] for QRS detection method on our system. The method can be separated into three steps of computation, there are feature extraction, event detection, and temporal localization.

The feature extraction aims to generate feature signal, which would be used as the input of event detection process. The event detection block searches the possible location of QRS complex by comparing the feature signal with the threshold. Temporal localization searches the actual peak location in the block of event detected in the previous process.
3.1 Feature Extraction

The first process in this step is filtering the raw signal. The first treatment is a Bandpass filter that filled with a cut-off frequency of 18 Hz and 35 Hz. Then the output will be processed with the nonlinear transform, by squaring the filtered signal, multiplied by its signum. This process follows the equation (1).

\[ y(n) = \text{sign}(x_f(n)) \cdot x^2(n) \]  

where \( y(n) \) is the transformed signal and \( x_f(n) \) denotes the bandpass filtered signal. This process will separate the important features of the signal, and the rest of other features will be transformed close to the x-axis.

The second step is adding the high-frequency sequence to the output signal. But for this process, we must calculate the amplitude of the high-frequency sequence before adding it. The calculated amplitude of the high-frequency sequence can be obtained by equation (2).

\[ K(n) = \lambda_k K(n-1) + (1 - \lambda_k) |y(n)c| \]  

where the estimated amplitude symbolled with \( K(n) \), \( \lambda_k \) is the forgetting factor, \( y(n) \) is the nonlinear transformed signal, and \( c \) implied the constant gain. After the amplitude is calculated, then we derived the high-frequency sequence by equation (3).

\[ b(n) = (-1)^n \cdot K(n) \]  

where \( b(n) \) is the high-frequency sequence and \( K(n) \) is the amplitude. The next step is adding the high-frequency sequence to the nonlinear transformed signal and make the zero-cross in equation (4).

\[ z(n) = y(n) + b(n) \]  

where \( z(n) \) is the signal after the high-frequency addition, \( y(n) \) is the nonlinear transformed signal, and \( b(n) \) is the high-frequency sequence. The whole process of the high-frequency addition can be seen in figure 2.

The last step in the feature extraction is zero crossing count. The zero crossing is counted by using equation (5).

\[ D(n) = \lambda_D D(n-1) + (1 - \lambda_D) d(n) \]  

where \( \lambda_D \) is forgetting factor and \( d(n) \) is the count of zero-crosses at the segment.

3.2 Event Detection

The main purpose of this process is to provide the predicted area of QRS complex by comparing the extracted featured signal with the threshold. The mathematic calculation of threshold is shown by equation (6).

\[ \Theta(n) = \lambda_\Theta \Theta(n-1) + (1 - \lambda_\Theta) D(n) \]  

where is \( \lambda_\Theta \) forgetting factor, \( D(n) \) is the feature signal and \( \Theta(n) \) is the adaptive threshold. Using this method, the feature signal that lower than the calculated threshold, the event is marked as the start point of the process. The algorithm would continue the process and when getting the feature signal that higher than the threshold, it is counted as the end of the process.

3.3 Temporal Localization

The R peak is the highest peak or lowest peak on the QRS complex. In this section, R-peak would be detected. To locate the R-peak, the temporal Localization will calculate the maximum and minimum of the signal in the predicted area of QRS complex that obtained before. If the maximum value is
higher than the absolute value of the minimum, then the temporal location of the maximum value is considered as the R-wave location. But if otherwise, the temporal location of the minimum value is considered as the R-wave location.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{flowchart.png}
\caption{Flowchart of the High-Frequency Addition. The blue box indicates the amplitude estimation. The yellow box indicates the high-frequency sequence. And the green box is the high-frequency addition.}
\end{figure}

4. Result
The testing is conducted using Linux terminal from Raspbian OS. The result of the testing is shown in table 1.

| Process                                    | Min (ms) | Max (ms) | Average (ms) |
|--------------------------------------------|----------|----------|--------------|
| Bandpass Filter                            | 51.4     | 90       | 52.5         |
| Nonlinear                                  | 0.67     | 0.96     | 0.79         |
| High-frequency Addition                    | 298.1    | 315.7    | 307.1        |
| Zero Crossing Count                        | 466.5    | 487.3    | 478.3        |
| Event Detection                            | 219.3    | 252.7    | 230.8        |
| Total Process of Algorithm                 | 1,044.7  | 1,095.6  | 1,069.5      |

The result from table 1 then converted to real-time factor proposed by Sufi et al. [1] to measure the real-time capability of the method using equation (7).

\begin{equation}
R_f = \frac{P_i}{Measurement\ Window}
\end{equation}
where $R_f$ is real-time factor value, and $P_t$ is processing time acquired from table 1. The real-time factor can be seen in table 2. Figure 3 shows the processing time of each sub-process of the Zero Cross Algorithm.

**Table 2.** The real-time factor of the Total Process of Zero Cross Algorithm.

| Zero Cross Algorithm (Real-time factor) |
|----------------------------------------|
| Min                                    | 0.0174 |
| Max                                    | 0.0182 |
| Average                                | 0.0178 |

As can be seen, the zero-cross count step takes the most amount of time.

As can be seen, the zero-crossing count sub-process takes most of the processing time in the algorithm with the minimum time of 466.5ms. It is still longer than the maximum value of the second longest processing sub-process, which is High-frequency Addition with 315.7ms.

Please note that the implementation of the algorithm is far from being optimized, as Köhler mentioned in the original paper that the most computationally expensive stages are the bandpass filter and event detection [7]. Therefore, there is still an improvement room on the overall processing time of the algorithm by optimizing the implementation of high-frequency addition and zero-crossing count stages.

As the whole algorithm itself, the algorithm takes 1,069.5ms to process the 60s of data in processing window. While the algorithm generally takes a lot of time (around 1s), it is still considered suitable for real-time use. At maximum, the real-time factor of the algorithm is 0.0182, which is still much less than 1. According to [1], whenever $R_f$ is much less than 1, the algorithm is classified as real-time.

5. **Conclusion**

This paper presents Zero Cross algorithm performance on Raspberry Pi 3 device for ECG QRS detection. The result shows that this method is suitable for real-time use of QRS detection in the raspberry device. Although the algorithm generates a high computing load on the device and seems to take a lot of processing time, the real-time factor shows that the algorithm fulfils the requirement of real-time use by 0.0182 at maximum which is less than 1. Furthermore, there is a lot of improvement room to make by optimizing the implementation of the algorithm, especially in the zero-crossing count and high-frequency addition stages. This result also shows that it is possible to build the part of ECG signal processing unit into the embedded system using Raspberry Pi and send the feature vector to the
cloud where the classification is performed. This will eliminate the needs of the smartphone as stated in our initial aims.

Acknowledgements
This research was supported by the Insinas Ristek funded by Minister of Research, Technology, and Higher Education, Indonesia.

References
[1] Sufi F, Fang Q and Cosic I 2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society p 3697
[2] Luz E, Schwartz W, Cámara-Chávez G and Menotti D 2016 Computer Methods and Programs in Biomedicine 127 144
[3] Escalona-Moran M, Soriano M, Fischer I and Mirasso C 2015 IEEE Journal of Biomedical and Health Informatics 19 892
[4] Karimifard S, Ahmadian A, Khoshnevisan M and Nambakhsh M S 2006 28th IEE EMBS Annual International Conference p 1367
[5] Park K S, Cho B H, Lee D H, Song S H, Lee J S, Chee Y J, Kim I Y and Kim S I 2008 Computers in Cardiology 35 229
[6] Pan J and Tompkins W 1985 IEEE Transactions on Biomedical Engineering 32 230
[7] Köhler B, Hennig C and Orglmeister R 2003 Progress in Biomedical Research 8 138
[8] Gradl S, Kugler P, Lohmuller C and Eskofier B 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society
[9] Moody G and Mark R 2001 IEEE Engineering in Medicine and Biology Magazine 20 45
[10] deChazal P, O'Dwyer M and Reilly R 2004 IEEE Transactions on Biomedical Engineering 51 1196