# Heartbeat Detection in ECG Signal using Cross-Correlation

Sofia Noemi Crobeddu
student number: 032410554
International Master of Biometrics and Intelligent Vision - Université Paris-Est Cretéil (UPEC)
Assignment 4

## Introduction

In this project an ECG signal is analyzed and it is performed the detection of heartbeats through cross-correlation method and thresholding technique. In order to achieve a more robust exploration, the same process is applied again on different noisy signal, analyzing the different outputs, also in metrics terms, at the variation of the Signal-to-Noise Ratio level.

## 1 ECG Signal Acquisition

The dataset used for this assignment was a sample dataset of an ECG signal. Through the library *pandas* of Python, we can read the data and store them into a dataframe that will be comfortable for the successive analysis. In addiction, a slight modification on the column's name was performed, removing the space at the start of the name.
The dataset contains just the value for each sample of the signal (i.e. it contains a unique column), and it is composed by 1001 samples.
Below it is showed the code performed for this starting step:

```
import pandas as pd
import matplotlib.pyplot as plt

#Loading the signal using pandas
signal = pd.read_csv(r"C:\Users\sofyc\OneDrive\Desktop\UPEC\Data capture and processing\assignment
    4\sample.csv")

#Removing the space before the column's name
signal.columns = signal.columns.str.strip()
```

The Figure 1 shows the ECG signal from data.

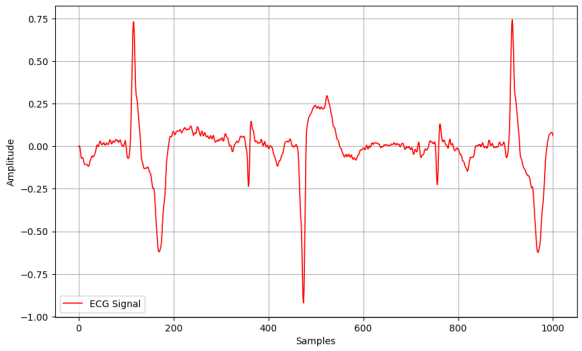


**Fig. 1**: ECG Signal.

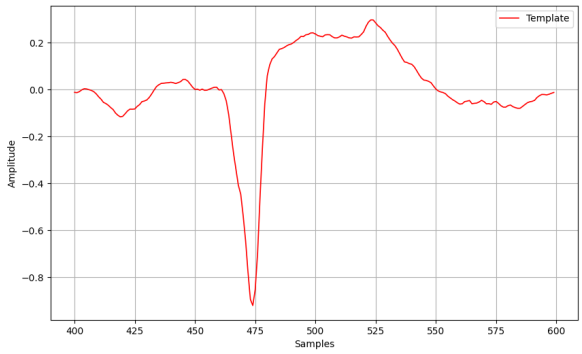


**Fig. 2**: Heartbeat Template: ECG isolated Signal.

## 2 Extraction of the Heartbeat Template

In this section we isolate the heartbeat template, extracting the ECG signal between 400 and 600 samples. It represents the template we will detect also in the next sections with noisy signals. It is basically one single heartbeat (a complete circle of heart), and it is the "ideal" one that permits the comparisons with others heartbeats.
The code to perform this step is the following one:

```
#Heartbeat template: samples between 400 and 600
template = signal[400:600]
```

The Figure 2 shows the template clearly. As we can see the samples in the x axis go from 400 to 600.

## 3 Implementation of the Cross-Correlation

In this section we implement the cross-correlation function to analyze the similarity between the template caught in the previous section, and the ECG signal, for each block of samples. This technique is important to localize similar pattern inside a signal, especially in ECG signals where we should have periodic heartbeat. It is defined as follow:

$$r_{xy}(k) = \sum_{n=-\infty}^{+\infty} x(n)y(n-k)$$

where $r_{xy}(k)$ is the value of the cross-correlation for each shift $k$ between the signal $x$ and the template $y$.
The code for the function of cross-correlation is shown below, and it returns the correlation between the template and the signal. The 'full' mode stands for the calculation of the correlation for each possible shift between the template and the signal. Each combo is usually called "offset" or "lag" and, through the function, we can associate a grade of similarity to each one. The result is stores in the variable *cross_correlation* and is an array. We can identify the heartbeats looking at the points where the cross-correlation amplitude is higher, i.e. where we can see the peaks of the cross-correlation curve. These points represent the ones where there is more similarity between the template and the signal. To interpret better this result, we need to determine the lags before plotting the array. Hence, the array *lags* will contain the shifts for each value of the correlation.
These insights were achieved through the code shown below:

```
from scipy.signal import correlate

#Function to calculate the cross-correlation
def fun_cross_correlation(heart_template, ecg_signal):
    return correlate(heart_template, ecg_signal, mode='full')

#Computing the result
cross_correlation = fun_cross_correlation(template, signal)
```

```
lags = np.arange(-len(signal['Sample Value']) + 1, len(template))
```

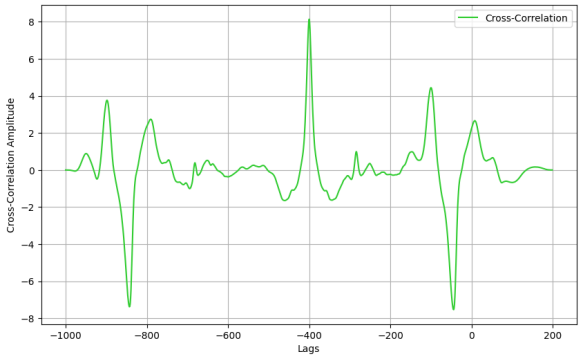


**Fig. 3**: Cross-Correlation between ECG Signal and Heartbeat Template.

The Figure 3 shows the Cross-correlation between the original signal and the heartbeat template. As we can notice, in the x axis the lags have also negative values. This is due to the fact that in some shifts the signal $x$ and the template $y$ overlap just in part. Negative lags indicate that the template is ahead of $x$, while positive lags represent the template in late in respect to the signal.

# 4 Application of Thresholding for Detection

Going on with the analysis, we can now detect the peaks of the cross-correlation through the Thresholding technique (as we did for example in assignment 1 but for the R peaks detection). In this case we use it to detect relevant peaks that will be isolated from the ones not significant. The threshold is calculated as a percentage (arbitrary) of the maximum of the cross-correlation. In this specific case, after some trials, I decided to set the threshold to the 30% of the maximum, since it lets to detect 5 evident peaks that we can see from Figure 4.
The commands to perform this analysis are shown below:

```
import numpy as np.
from scipy.signal find_peaks

#We define a threshold as the 30% of the maximum of the cross-correlation
threshold = 0.3 * np.max(cross_correlation)

#Transforming in 1D array
cross_correlation = np.array(cross_correlation).flatten()

#Peak detection algorithm
peaks, _ = find_peaks(cross_correlation, height=threshold)
```

The value of the threshold is around 2.44 and it is tracked in yellow line in the Figure 4. After this, as in assignment 1, to catch the peaks we use the function *find_peaks* from *Scipy Signal* to detect the picks that exceed the threshold, identifying the positions of the potential heartbeats. Before doing this, we need to transform the cross-correlation array in a vector.
Peaks detected are signed with an x in Figure 4.

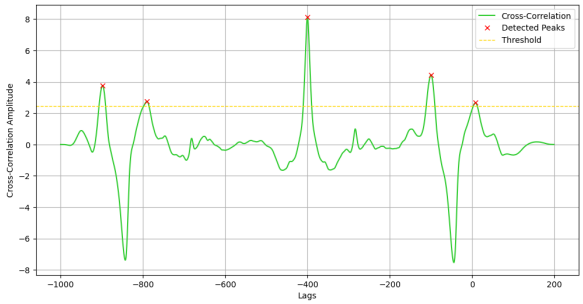


**Fig. 4**: Cross-Correlation with Detected Peaks.

# 5 Simulation of Noise and Variation of the SNR

In this section we simulate White Gaussian Noise and we add it to our ECG signal. It is a common practice useful to test the robustness of algorithm detection as the one we applied with the thresholding technique. In this case we simulate this noise varying the level of Signal-to-Noise Ratio (SNR) between 10, 20 and 30 dB. For completeness of information, SNR represents the ratio between signal power and noisy power.
To implement this process, I created a function called *add_noise* where first, we need to calculate the power of the signal (i.e. the mean of the squared signal), and after we find the noise power defined as:

$$\text{Noise power} = \frac{\text{Signal power}}{10^{\frac{SNR}{10}}}$$

This formula comes from the one defining the SNR in decibel:

$$SNR = 10\log_{10}(\frac{\text{Signal power}}{\text{Noise power}})$$

After this, we can generate easily the noise using the function for normal random values from *Numpy* library, since the noise mean is 0. The output that we can obtain from the function is the ECG signal with noise added.
Figures 5, 6, 7 show the output for a SNR equal to 10 dB, 20 dB and 30 dB respectively. In the first one we have a high level of noise, and for this reason it is more difficult to analyze the ECG signal, as we can see from the graph. With a higher SNR (20 and 30), we can identify more clearly the heartbeat, since the noise is reduced.
The code implemented is the following ones:

```
#Function to add Gaussian noise to the ECG signal
def add_noise(signal, SNR_given):
    #Calculate the signal power
    signal_power = np.mean(signal**2)
    #Calculate noise power based on the desire SNR
    noise_power = signal_power / (10**(SNR_given/10))
    #Generating Gaussian noise N(mean=0, sd=sqrt(noise_power))
    noise_gauss = np.random.normal(0, np.sqrt(noise_power), signal.shape)
    #Return the signal with noise
    return(signal + noise_gauss)
```

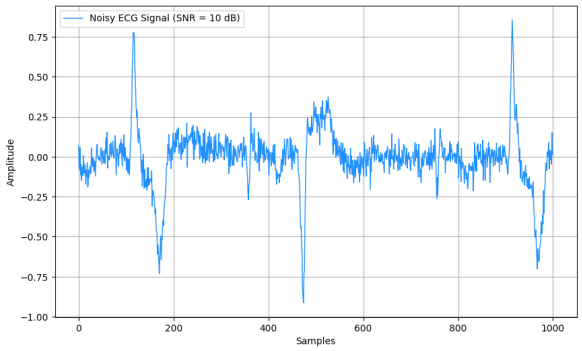


**Fig. 5**: Noisy ECG Signal at SNR = 10 dB.

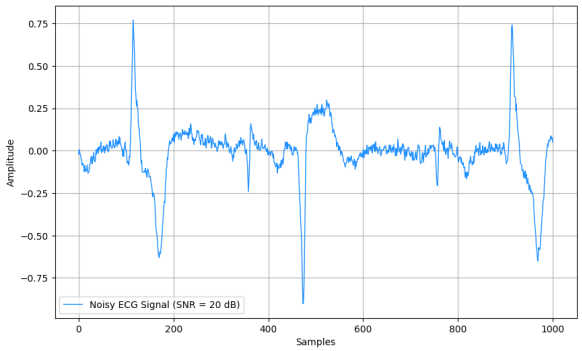


**Fig. 6**: Noisy ECG Signal at SNR = 20 dB.

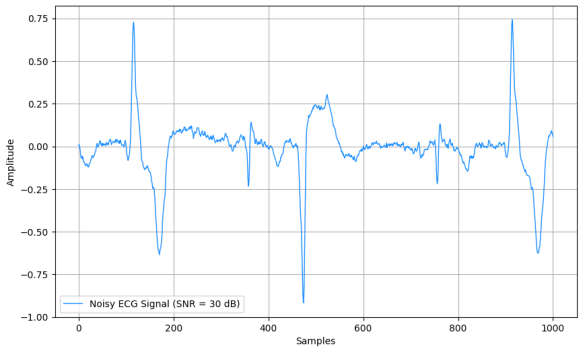


**Fig. 7**: Noisy ECG Signal at SNR = 30 dB.

# 6 Detection of Heartbeat in Noisy Signals using Thresholding

After finding the signal with noise for each SNR level, we can repeat the same process we did for the original ECG signal. Hence, we apply again the cross-correlation function between the template and every signal with noise of different SNR levels setting the threshold as the 30% of the maximum of the cross-correlation, as we did in section 4. So, in this case, we will have three different thresholds, one per each signal with noise. Through this process we can detect the position of potential heartbeats. For monitoring the accuracy of our analysis, and also to avoid the repetition of the same passages in

the next section 7, we can calculate the True positive (TP), False Positive (FP) and False Negative (FN):

- TP represent the number of heartbeat detected correctly, i.e. the peaks' positions detected that coincide with the real peaks of the signal;
- FP represent the number of incorrect detections (the extra detections), i.e. the peaks that were caught but that are not actually heartbeat and they were may caused by noise;
- FN represent the number of missing detections, i.e. the real peaks that were not caught because of noise.

In signal processing, especially with ECG signals, the calculation of the True Negatives (TN) is usually very complicated to have. TN in this context in fact represent the points where there was no heartbeat. So, in this case, it is not necessary.
Moreover, in the same code we also calculate the accuracy metric, indicating the ratio between correct detections and the total detections and errors, and defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TN are not calculated for this specific situation, but we can still have a final metric of Accuracy that gives us a quantification of our analysis.
Another metric calculated is the Precision metric, indicating the ratio between correct real heartbeats detected and the total of heartbeat, and defined as:

$$Precision = \frac{TP}{TP + FP}$$

The third metric calculated is the Recall, defined as:

$$Recall = \frac{TP}{TP + FN}$$

These metrics vary between 0 and 1.
The commands to perform this analysis are shown below:

```
#Application of cross-correlation function and thresholding function
cross_corr_noisy_signals = []
threshold_noise = []
peaks_noise = []
lags_noise = []
true_positive_noise = []
false_positive_noise = []
false_negative_noise = []
accuracies_noise = []

#We calculate also precision and recall for the next point 7.
precision_noise = []
recall_noise = []

for i in SNR_levels:
    #Creating the signal with noise
    noisy_signal = add_noise(signal['Sample Value'], i)

    #Cross-correlation calculation for the noisy signal
    c = fun_cross_correlation(np.array(template).flatten(), noisy_signal)
    cross_corr_noisy_signals.append(c)

    #Threshold calculation
    t = 0.3 * np.max(c)
    threshold_noise.append(t)

    #Peaks detection
    p, _ = find_peaks(c, height=t)
    peaks_noise.append(p)

    #Calculation of lags
    l = np.arange(-len(noisy_signal) + 1, len(template))
    lags_noise.append(l)
```

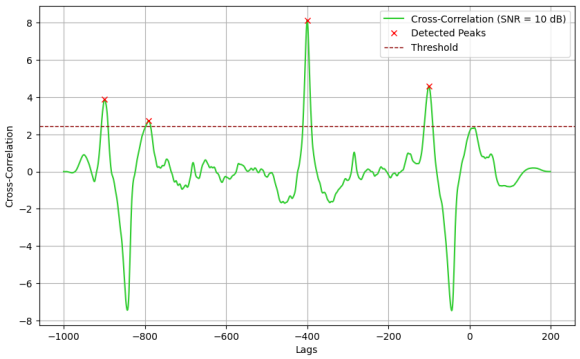


**Fig. 8**: Cross-Correlation between Template and Noisy Signal at SNR = 10 dB.

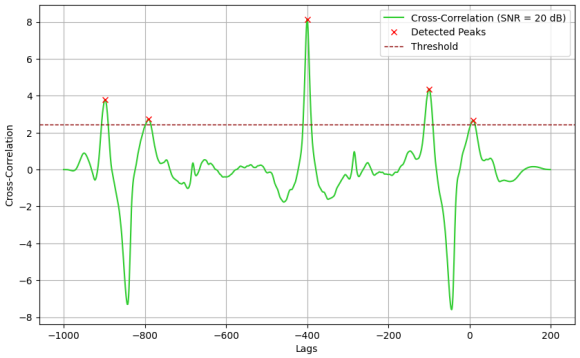


**Fig. 9**: Cross-Correlation between Template and Noisy Signal at SNR = 20 dB.

```
34
35      #True Positive, False Positive, False Negative calculation
36      tp = len(set(peaks) & set(p)) #Correct detections
37      fp = len(set(p) - set(peaks)) #Extra detections
38      fn = len(set(peaks) - set(p)) #Missing detections
39
40      true_positive_noise.append(tp)
41      false_positive_noise.append(fp)
42      false_negative_noise.append(fn)
43
44      #Accuracy calculation
45      acc = tp / (tp + fp + fn) if (tp + fp + fn) > 0 else 0
46      accuracies_noise.append(acc)
47
48      #Precision calculation
49      pr = tp / (tp + fp) if (tp + fp) > 0 else 0
50      precision_noise.append(pr)
51
52      #Recall calculation
53      rec = tp / (tp + fn) if (tp + fn) > 0 else 0
54      recall_noise.append(rec)
```

The Figures 8, 9, 10 below, show the peaks detection in the cross-correlation between the template and each signal with noise. As we can see there are 5 detected peaks in each graph, and the visual difference between the three cross-correlations is not so evident. Cross-correlation, in fact, shows high variations for lower SNR (such as 2 or 4 dB) compared to higher values of SNR like 30, 50 or 70 dB. This can be explained by two factors principally:

- logarithmic scale of SNR measured in decibel;
- the impact of the noise. For low values of SNR, in fact, the noise has a power similar or higher than the signal power, and of course this makes difficult to identify clearly the signal. With large values of SNR the noise becomes irrelevant.

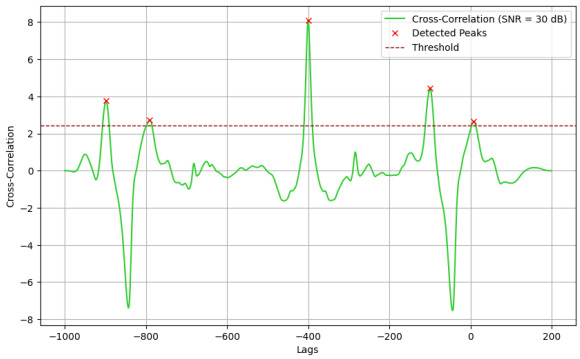


**Fig. 10**: Cross-Correlation between Template and Noisy Signal at SNR = 30 dB.

# 7 Analysis of the Detection Performance with Thresholding

In this final section we evaluate the outputs of the metrics calculated in section 6. After putting the outputs in a dictionary and transforming it into a dataframe (see code below), we reach the results shown in Table 1.
With a SNR of 10 dB, we have a high noise, so the number of correct detections (TP) is limited, just 2, while false heartbeat detections that exceed the threshold (TP) are 2, and the missed heartbeats below the threshold, and so not detected, are 3. For this reason we have an accuracy of 29%, a precision of 50% and a recall of 40%. This low values suggest that because of noise, the system can detect just a part of the heartbeats and, within these, some are not real. For SNR=20 dB we have a similar situation but still slightly better since the TP are 3 and FN are 2, while FP remains 2. Moreover the accuracy reaches the 43%, the precision and the recall are 60%. Finally, for a SNR of 30 dB, we reach a total accuracy, precision and recall.
It is important to specify that these values changes every time we run again the code to add noise to the signal, since the noise is generated randomly.

```
#We have already calcuated the results in point 6.
#Results table
res = {
    'SNR level': SNR_levels,
    'True positives': true_positive_noise,
    'False positives': false_positive_noise,
    'False negatives': false_negative_noise,
    'Accuracy': accuracies_noise,
    'Precision': precision_noise,
    'Recall': recall_noise
}
res_df = pd.DataFrame(res)
```

| SNR level | True positives | False positives | False negatives | Accuracy | Precision | Recall |
|---|---|---|---|---|---|---|
| 10 | 2 | 2 | 3 | 0.29 | 0.5 | 0.4 |
| 20 | 3 | 2 | 2 | 0.43 | 0.6 | 0.6 |
| 30 | 5 | 0 | 0 | 1.00 | 1.0 | 1.0 |

**Table 1**: Performance metrics varying SNR level.

Figure 11 shows the performance of the metrics compared to the SNR level. As we can see there is a positive relation between SNR level and metric performance, and this is due to the fact that a higher SNR represents less noise power.

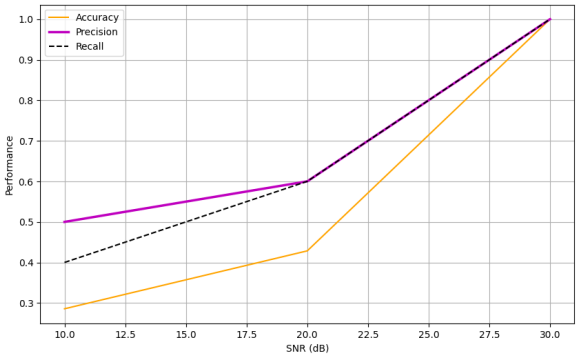


**Fig. 11**: Performance Metrics vs SNR Levels.

# References


[1] contributors, W.: Cross-correlation — Wikipedia, The Free Encyclopedia (2023). https://en.wikipedia.org/wiki/Cross-correlation

[2] WaveWalker DSP: Cross-Correlation Explained with Real Signals (2021). https://www.wavewalkerdsp.com/2021/12/01/cross-correlation-explained-with-real-signals/

[3] contributors, W.: Signal-to-noise ratio — Wikipedia, The Free Encyclopedia (2023). https://en.wikipedia.org/wiki/Signal-to-noise_ratio

[4] Kulas, B.: Working with ECG Heart Rate Data on Python (2023). https://bartek-kulas.medium.com/working-with-ecg-heart-rate-data-on-python-7a45fa880d48

[5] Kumar, P.: Evaluation Metrics in Machine Learning (2023). https://www.linkedin.com/pulse/evaluation-metrics-machine-learning-kumar-p/

[6] Nait-Ali, A.: Correlation in Digital Signal and Image Processing. Lecture slides, Université Paris-Est Crétéil (UPEC) - Faculté des Sciences et Technologie (2024)

[7] Nait-Ali, A.: Session: Digital Signal Processing. Lecture slides, Université Paris-Est Crétéil (UPEC) - Faculté des Sciences et Technologie (2024)