NTNU
Kunnskap for en bedre verden

Department of ICT and Natural Sciences

AIS2201 - Digital Signal Processing

Portfolio Project Description
Part 2: Algorithm Development

Kai Erik Hoff

Date

Overview

This document describes part 2 out of 3 for the portfolio project for AIS2201 - Digital Signal Processing, autumn 2025. The focus of part 2 of the project is to adapt signal processing principles learned throughout the course to the particular problem encountered in the project.

Table of Contents

1 Checklist from part 1 of the project

Were you able to do the following?

1. Use data from a source file to generate simulated noisy ECG signals, where noise level and interference type can be adjusted.
2. Pass the simulated ECG signals as input to the Python function representing the DSP system to be tested, and record the function's output.
3. Compare the DSP system's output to a reference, calculate a metric for the DSP system's performance, and provide informative visual representations for use in system evaluation.
4. Complete a status meeting with lecturer to receive feedback on your solution to part 1.

If not: **do not** proceed with part 2 of the project until the above list is completed.

2 Part 2: Developing a DSP system for heartbeat detection

Now that we have a clear idea of what type of interference can be expected in an ECG signal and what challenges need to be overcome, it is time to begin the development of the heartbeat detection algorithm itself. The testing routine created as part 1 of the project will serve as a useful tool not only in evaluating system performance, but also as a means of pinpointing potential weaknesses in the algorithm.

Proposed DSP systems should adhere to the system specifications in the project overview, with noisy ECG signal and sample rate f_s as input parameters, and calculated heartbeat durations as output. A primitive algorithm for heartbeat detection which adheres to this specification is shown in appendix A, and may serve as a basis for further modifications.

2.1 Level 1: interference filtering and automatic thresholding

A solution to level 1 should use suitably designed digital filters to suppress (to some degree) each of the three main types of interference: baseline drift, white noise, and 50 Hz narrowband interference. Both FIR and IIR filters may be suitable for this task, and it is up to each student to determine which types of filter to use, although IIR filters have not been covered in detail in this course.

Once the various types of interference present in the signal have been suppressed, the system should ideally be able to pinpoint the timing of each heartbeat in the recorded ECG signal. The example program introduced in part 1 of the project accomplishes this by detecting threshold crossings, but the threshold must be set manually. A proposed solution for level 1 of the project should be able to calculate a potentially suitable threshold value based on the input signal. Students are also encouraged to explore potential improvements to the threshold crossing algorithm such as hysteresis.

2.2 Level 2: advanced detection methods

Until now, we have not spent much time considering the theory and composition of an ECG signal, beyond having a significant peak for each heartbeat. As a rule, ECG signals *tend* to have the general shape shown in figure 1, with the R wave having significantly higher magnitude than other parts of the ECG signal.

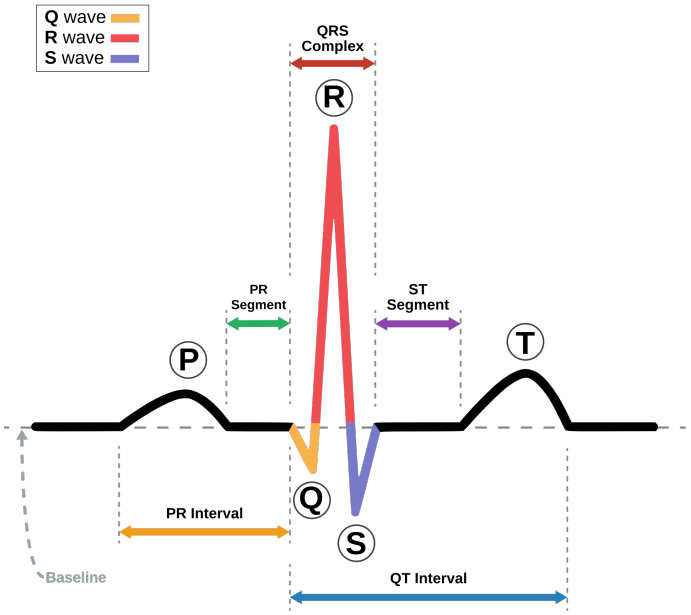

Figure 1: Annotated ECG Pulse

The signal shape in figure 1 is far from guaranteed however, and deviations from this strict pattern are to be expected. A fairly common phenomenon observed in ECG signals is for instance a T-wave whose amplitude can be equal to or even higher than the R-wave. Figure 2 shows an example of such an ECG signal, which is recorded in the files `ecg_data_2.mat` and `ecg_data_single_2.mat`. For such signals, a simple heartbeat detection system which relies on detecting threshold crossings will be unreliable, and a more sophisticated approach is needed.

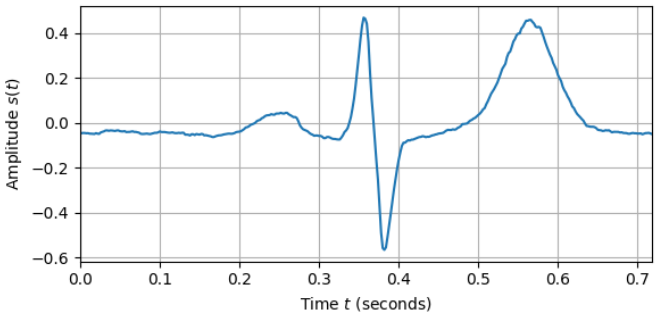

Figure 2: Single ECG Pulse with prominent T-wave

A solution to level 2 should, in addition to interference filtering as described in level 1, employ a more robust approach to heartbeat detection which aims to detect a specific *signal pattern* in the signal rather than rely simplistic approaches such as threshold crossings. It is up to the student to determine which DSP principles may be employed to accomplish this, but a good place to start would be cross-correlation or autocorrelation.

3 Project Report

It is recommended to keep the report up-to-date as the project progresses. By the end of Part 2: Algorithm Development you should be able to fill in most of chapters 2, 3 and 4:

- **Chapter 3: Proposed Algorithm**

  Present your heart rate detection algorithm. Take note that the report **must** include system block diagram that represents your DSP system. Remember that you are supposed to present the ***theoretical/mathematical basis*** of your proposed system, simply attaching screenshots of python code is ***not*** sufficient.

- **Chapter 4: Simulation and Results:**

  Explain your choice of simulated test signals, and how the DSP system performance is to be quantified. Where system evaluation is dependent on analyzing plots, provide an explanation of how the plots are to be interpreted.

  Present the evaluation results for your system along with your interpretation of the results. If your development process involved more in-depth troubleshooting using e.g. spectral analysis of signals at various points in the system, this should also be included here.

It is recommended to refer to the report template [1] for a more thorough description of what the different chapters should contain.

[1] Overleaf project report template

Appendix

A Threshold crossing algorithm

```
import numpy as np
def get_crossing_intervals(x, fs, threshold=0.0):
    """
    Function to detect positive threshold crossings in a discrete-time signal.
    Parameters:
     x - discrete signal samples
     fs - sampling frequency
     threshold - threshold value, defaults to 0.0.
    Returns:
     intervals - a list of time durations between each threshold crossing
                 registered
    """
    N = len(x)
    intervals = []
    above_threshold = True
    last_crossing = 0.0
    for n in range(N):
        if x[n] > threshold and not above_threshold:
            above_threshold = True
            current_crossing = n/fs
            intervals.append(current_crossing - last_crossing)
            last_crossing = current_crossing
        elif x[n] <= threshold and above_threshold:
            above_threshold = False
    return intervals
```