Akademia Górniczo-Hutnicza w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt zaliczeniowy

Analizator EKG

Autorzy:
Mateusz Baran
Krzysztof Bębenek
Bartłomiej Bułat
Szczepan Czaicki
Tomasz Drzewiecki
Krzysztof Farganus
Łukasz Jaromi
Mateusz Krasucki
Łukasz Krzyżek
Łukasz Kutrzuba
Weronika Łabaj
Paweł Maślanka
Piotr Matuszkiewicz
Norbert Pabian
Łukasz Pękala
Krzysztof Piekutowski
Grzegorz Pietrzyk
Łukasz Podolski
Mikołaj Rzepka
Agata Sitnik
Leszek Sosnowski
Aleksander Steliga
Mateusz Ślażyński
Łukasz Zieńkowski

Opiekun:
mgr inż. Tomasz Pięciak

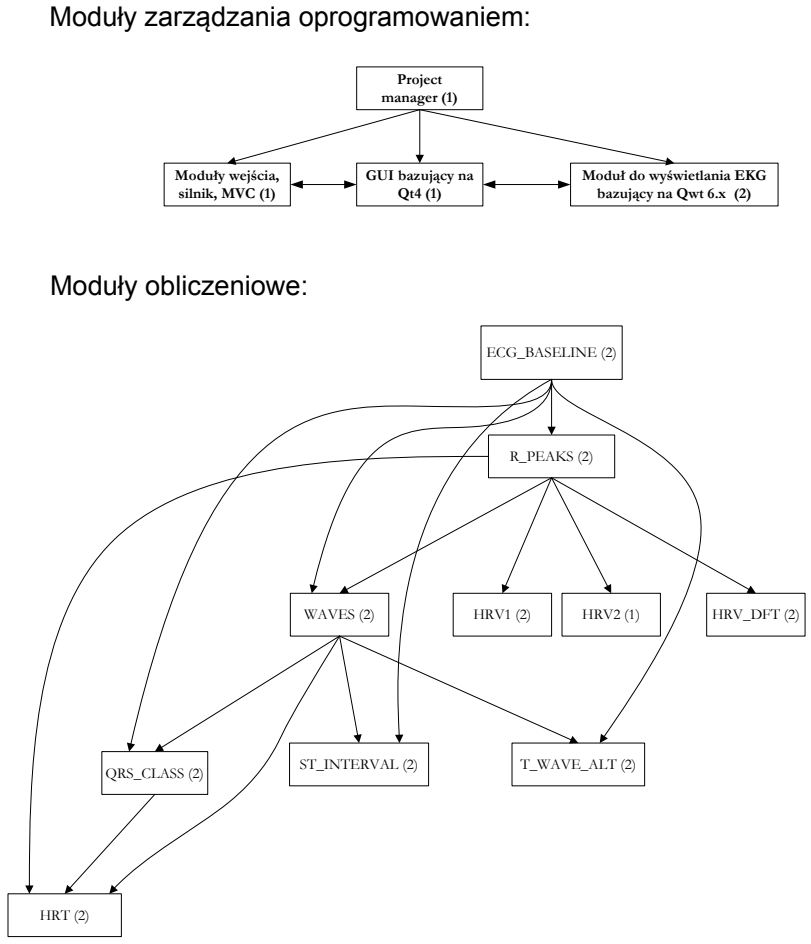
24 stycznia 2013

Spis treści

1 Specyfikacja zadania

Celem projektu jest stworzenie zintegrowanego systemu pozwalającego na przeglądanie i automatyczną analizę sygnału EKG. Sygnał dostarczany jest w formie cyfrowej w standardzie wykorzystywanym w MIT–BIH Arrhythmia Database. Różne etapy przetwarzania, takie jak usuwanie linii bazowej, detekcja załamków R czy klasyfikacja zespołów QRS wykonywana jest przez różne zespoły (szczegółowe opisy specyfikacji modułów: 3), których praca składa się na jeden program.

Moduły przetwarzania integrowane i uzupełniane są modułami kontrolującymi przepływ danych i odpowiadającymi za komunikację z użytkownikiem. Wzajemne zależności pomiędzy modułami prezentuje rys. 1.



Rysunek 1: Zależności pomiędzy modułami projektu.

2 Specyfikacja techniczna rozwiązania

2.1 Wykorzystane narzędzia

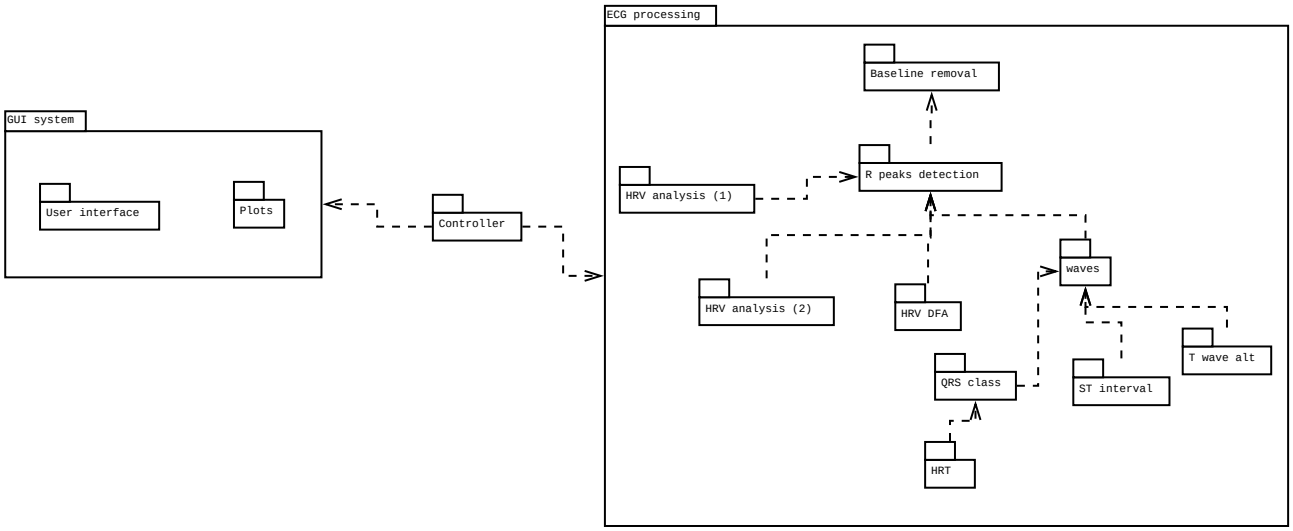
Podczas realizacji projektu wykorzystywane były różne narzędzia do tworzenia i prototypowania rozwiązań. Wstępne projekty przygotowywane były w programie Matlab, zaś ostateczny kod powstawał w języku C++ (standard '03 z elementami standardu C++11 obsługiwanymi przez wspierane kompilatory). Minimalne wymagania kompilacji projektu są następujące:

- Jeden z kompilatorów:
  - Microsoft Visual Studio 2010
  - GCC 4.5
- Biblioteki:
  - Boost 1.51
  - Qt 4.8
  - Qwt 6.01
  - gsl 1.15
  - WFDB 10.5.16 (zawarta w źródłach projektu)
  - FFTW 3.3.2

Do wersjonowania i śledzenia błędów wykorzystywaliśmy platformę Github wraz z rozproszonym systemem kontroli wersji Git. Posiada on zaawansowane możliwości wspierające pracę grupową nad projektem, co szczególnie przydaje się, gdy liczba osób jest duża.

2.2 Projekt systemu

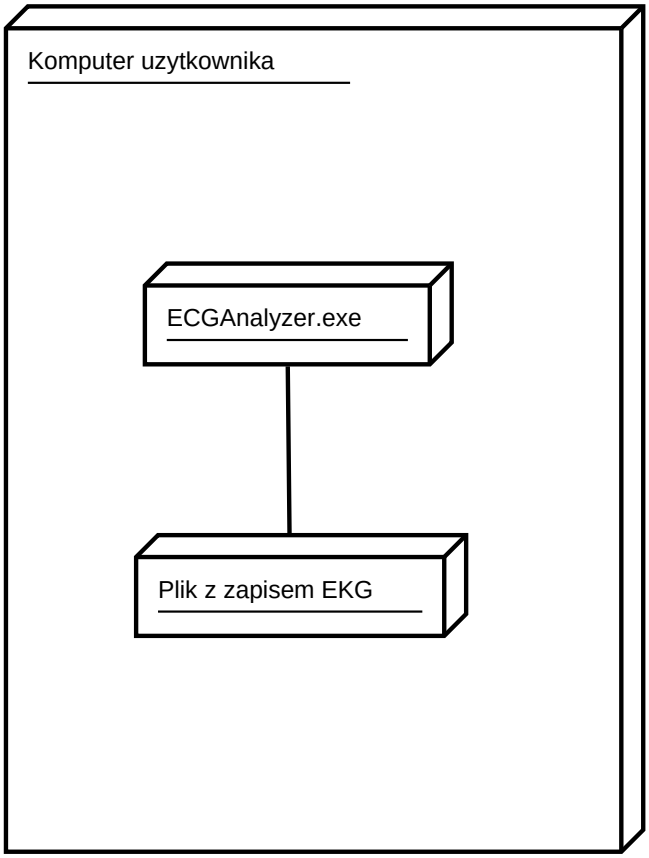
Program został wykonany w architekturze MVC – istnieje ścisły podział na część wyświetlającą interfejs użytkownika, moduły przetwarzania sygnału oraz kontroler łączący te dwa elementy, co obrazuje diagram pakietów 2.2. Zastosowano obiektowe podejście przy projektowaniu hierarchii klas realizujących przetwarzanie (rys. **??**), umożliwiające bezproblemową wymianę implementacji dowolnego modułu na inną.



Rysunek 2: Diagram pakietów.

Samo przetwarzanie realizowane jest wielofazowo – moduły przeliczane są sekwencyjnie na osobnym wątku niż wątek zdarzeń GUI, co pozwala na zatrzymanie zbyt długo trwającej operacji. Zaimplementowano także mechanizm buforowania wyników, dzięki czemu przy zmianie parametrów pewnego modułu nie ma konieczności przeliczania wyników modułów wcześniejszych.

Cała aplikacja napisana jest jako samodzielny, wieloplatformowy program (rys. 2.2) – możliwa jest kompilacja pliku binarnego pod systemami Windows, Linux i MacOS X. Całe przetwarzanie wykonywane jest lokalnie, dzięki czemu nie jest wymagane połączenie z Internetem. Dzięki wybraniu licencji GPL v2+ projekt jest wolny i możliwy jest jego dalszy rozwój i kompilacja na nowe platformy.



Rysunek 3: Diagram wdrożenia.

3 Opisy modułów

3.1 Usuwanie linii bazowej

Autorzy: Weronika Łabaj i Piotr Matuszkiewicz.

3.1.1 Opis zadania

Temat Metody filtracji i detekcji izolinii w sygnale EKG

Opis Występujące zakłócenia sieciowe i mięśniowe, jak również falowanie linii izoelektrycznej w sygnale EKG niejednokrotnie uniemożliwiają właściwą i poprawną analizę sygnału. Celem projektu jest opracowanie i implementacja metod związanych filtracją i detekcją linii izoelektrycznej w sygnale EKG. W szczególności należy rozważyć:

- filtr Butterwortha,
- średnią kroczącą,
- metody nieadaptacyjne: filtr Savitzky-Golay'a

- metody atapdacyjne np. filtr Wienera, LMS

Dane ciąg próbek sygnału EKG z bazy Physionet.org

Szukane moduł programu filtrujący sygnał EKG z zakłóceń sieciowych/mięśniowych oraz usuwający z sygnału falowanie linii izoelektrycznej przy wykorzystaniu różnych metod; w ostatecznym module programu będzie możliwość wyboru algorytmu filtrującego i usuwającego falowanie linii izoelektrycznej

3.1.2 Badania literaturowe

3.1.3 Opis procedur i metod

3.1.4 Warunki testowania

3.1.5 Wyniki

3.2 Wykrywanie załamków R

Autorzy: Paweł Maślanka i Norbert Pabian

3.2.1 Opis zadania

Temat Detekcja załamka R

Opis Detekcja załamków R w przefiltrowanym sygnale cyfrowym EKG, mimo iż jest tematem przebadanym i dobrze znanym, to wciąż nie istnieje złoty standard postępowania. Alternatywą dla powszechnie znanego algorytmu "Pan-Tompkins", opartego o podstawowe przekształcenia matematyczne potęgowania i różniczkowania może okazać się transformata Hilberta odpowiednio przygotowanego sygnału. Celem projektu jest opracowanie i implementacja metod służących do detekcji załamków R w sygnale EKG w oparciu o algorytm "Pan-Tompkins" oraz zespolone przekształcenie Hilberta. Podejmując projekt należy zwrócić uwagę na problematykę wystąpienia szumu i analizy sygnału na końcach przedziału, gdzie rozważania teoretyczne transformaty Hilberta nieco odbiegają od praktycznych.

Dane przefiltrowany ciąg próbek z modułu ECG_BASELINE

Szukane moduł programu wyznaczający numery próbek załamków R oraz zaznaczający na wykresie wykryte załamki R. Numery próbek pozwolą na dalszą analizę ilościową i jakościową sygnału EKG.

3.2.2 Badania literaturowe

TODO BADANIA LITERATUROWE

3.2.3 Opis procedur i metod

Implementacja modułu wykrywania załamków `R` znajduje się w klasie `RPeaksDetector` która rozszerza abstrakcyjny moduł wykrywania załamków `R`. Klasa ta posiada implementację dwóch metod wykrywania załamków `R`:

- Pant-Pompkins
- Hilbert

Moduł na podstawie parametrów jakie otrzymuje z `GUI` wybiera odpowiednią metodę detekcji. Oprócz metody detekcji możliwe jest również ustawienie ręczne parametrów takich jak:

- dla metody Pant-Pompkins
  - próg detekcji

  - szerokość okna całkowania

- dla metody Hilbert
  - TODO JAKIE PARAMETRY

Użytkownik może też skorzystać z automatycznej detekcji powyższych parametrów. Automatyczna detekcja jest włączona jako standardowe parametry. Opis implementacji metod wykrywania załamków:

```
bool panTompkinsRPeaksDetection(ECGSignalChannel *signal);
```

Funkcja otrzymuje na wejściu przefiltrowany sygnał `ECGSignalChannel *signal` z modułu `ECGBaseLine`. Z założenia sygnał jest pozbawiony składowej stałej, szumów pochodzących z mięśni orz znormalizowany w zakresie od -1 do 1. Detekcja pomija więc wstępną filtracje która została wykonana w poprzednim module. Zadania wykonywane wewnątrz można podzielić na następujące etapy:

- różniczkowanie sygnału
- potęgowanie sygnału
- obliczanie szerokości okna całkowania
- całkowanie sygnału ruchomym oknem
- obliczanie progu detekcji
- wykrywanie zespołów QRS
- wykrywanie załamków R

Jeśli podczas wykrywania nie wystąpi żaden problem zostanie stworzony wektor z punktami w jakich zostały wykryte załamki `R`. Natomiast jeśli z jakiegoś powodu wykrycie załamków nie powiedzie się zostanie rzucony wyjątek `RPeaksDetectionExveption`.

```
bool hilbertRPeaksDetection(ECGSignalChannel *signal);;
```

Funkcja również na wejściu otrzymuje sygnał z modułu `ECGBaseLine`. Zadania wykonywane podczas detekcji można podzielić następująco:

- TODO
- TODO
- TODO
- TODO
- TODO

Jeśli podczas wykrywania nie wystąpi żaden problem zostanie stworzony wektor z punktami w jakich zostały wykryte załamki `R`. Natomiast jeśli z jakiegoś powodu wykrycie załamków nie powiedzie się zostanie rzucony wyjątek `RPeaksDetectionExveption`

3.2.4 Warunki testowania

Początkowo moduł był testowany przy użyciu własnego przefiltrowanego sygnału testowego utworzonego w matlabie. Testowy sygnał został pozbawiony składowej stałej, zostały usunięte zakłócenia filtrami: dolnoprzepustowym oraz górnoprzepustowym. Cały sygnał został znormalizowany do zakresu od -1 do 1.

Sygnał można wczytać przy pomocy funkcji jaka znajduje się w klasie `RPeaksDetector` o nazwie `getMockedSignal`. W pliku `RPeaksDetector.h` znajdują się makrodefinicje które w prosty sposób pozwalają włączyć debugowanie procesu wykrywania sygnału oraz użycie testowego sygnału.

Makrodefinicja `USE_MOCKED_SIGNAL` włącza użycie testowego sygnału. Sygnał jaki dostarcza moduł `ECGBaseLine` jest ignorowany. Dodatkowo makrodefinicja `DEBUG` pozwala na wypisanie podstawowych informacji z przebiegu wykrywania załamków `R`. Poniżej przedstawiamy przykładowy log:

```
Thersold size not found, use automatic calculated value
Input parameters for R peaks module:
Detection method: PanTompkins
Moving window size: 0
Thersold size: 0
R peaks module started
Use mocked signal for R-peaks module.
Running module with custom parameters
Convolution [-0.125 -0.25 0.25 0.125]
Orginal signal size: 600000
Exponentiation ĝ2
Signal size after convolution: 599996
Moving window integration
Calculating moving window size
Moving window size: 24
Signal size after exponentiation: 599996
Calculating detection thersold
After moving window integration signal size: 599972
Final max value: 0.0438604
Final mean value: 0.00221016
Current thresold value: 0.0109651
Looking for points over thersold
Detect begin and end of QRS complex
Number of left points: 2093
Number of right points: 2093
Final R peaks detection
Number of detected R-peaks: 2093
Done
```

Kiedy moduł `ECGBaseLine` dostarczył nam prawidłowo przygotowany sygnał, rozpoczęliśmy testowanie na plikach pochodzących z bazy `MIT_BIH`. Wyniki jakie otrzymaliśmy można znaleźć w kolejnym rozdziale.

3.2.5 Wyniki

Uzyskane wyniki wykrywania załamków `R` dla sygnałów pochodzących z bazy `MIT_BIH`.

sygnał	ilośc R	wykryte PanTompkins	wykryte Hilbert	skuteczność PanTompkins	skuteczność Hilbert
100	2273	TODO	TODO	TODO	TODO
101	1865	TODO	TODO	TODO	TODO
102	2187	TODO	TODO	TODO	TODO
103	2084	TODO	TODO	TODO	TODO
104	2229	TODO	TODO	TODO	TODO
105	2572	TODO	TODO	TODO	TODO
106	2027	TODO	TODO	TODO	TODO
107	2137	TODO	TODO	TODO	TODO
108	1774	TODO	TODO	TODO	TODO
109	2532	TODO	TODO	TODO	TODO
200	2601	TODO	TODO	TODO	TODO
201	2000	TODO	TODO	TODO	TODO
202	2136	TODO	TODO	TODO	TODO
203	2980	TODO	TODO	TODO	TODO
205	2656	TODO	TODO	TODO	TODO
207	2332	TODO	TODO	TODO	TODO
208	2955	TODO	TODO	TODO	TODO

3.3 Waves

3.4 HRV1

3.5 HRV2

Autor: Krzysztof Farganus

3.5.1 Opis zadania

Celem projektu jest opracowanie i implementacja metod geometrycznych analizy HRV.
Dane przyjmowane przez moduł:

- ciąg próbek załamków R z modułu R_PEAKS.

Dane zwracane przez moduł:

- wykres histogramu
- wskaźnik TINN
- Indeks Trójkątny
- wykres Poincare wraz z parametrami SD1 i SD2

3.5.2 Badania literaturowe

Techniki geometryczne służą do przedstawienia długookresowej zmienności rytmu serca [**?**]. Są łatwe do uzyskania, ponieważ bazują na aproksymacji histogramu trójkątem. Szerokość przedziałów klasowych histogramu ma tutaj kluczowe znaczenie, gdyż jej wartość wpływa na rezultaty metod geometrycznych. Głównie stosowany jest zakres wynoszący 7.8125 ms (1/128 s). Powodem jest częstotliwość próbkowania sygnału o najczęściej występującej wartości 128 Hz.

Cechy metod geometrycznych:

- odporność na zakłócenia ze względu na zastosowanie technik aproksymacyjnych
- eliminacja artefaktów zlokalizowanych poza trójkątem
- wyniki niezależne od jakości zapisu sygnału
- rezultaty zależne od czasu rejestracji
- wymagana duża liczba odstępów RR dla poprawnej analizy (minimalny czas zapisu – 20 minut)

Do najbardziej rozpowszechnionych metod należą:

- Wykres histogramu przedstawiający rozkład interwałów RR
- Indeks trójkątny (HRV triangular index) - całkowita liczba wszystkich odstępów RR podzielona przez liczbę odstępów RR o najczęściej spotykanym czasie trwania.
- Trójkątna interpolacja odstępów RR (TINN) – długość podstawy trójkąta aproksymującego histogram kolejnych odstępów interwałów RR rytmu zatokowego wyrażana w milisekundach.
- Wykres Poincare - graficzna reprezentacja korelacji pomiędzy kolejnymi interwałami , gdzie każdy odstęp RR jest opisany funkcją RR+1. Do analizy rozproszenia punktów na wykresie stosuje się dwa deskryptory: SD1 oraz SD2, odpowiadające odchyleniom standardowym. Pierwszy charakteryzuje rozkład punktów w poprzek linii identyczności, natomiast drugi wzdłuż tej linii.

Algorytm obliczania wskaźnika TINN [?]

Aby obliczyć parametr TINN, czyli wyznaczyć wartości punktów N i M należy opracować funkcję multiliniową q(t) o postaci:

- $q(t) = 0$ dla $t \leqslant N$
- $q(t) = 0$ dla $t \geqslant M$
- $q(X) = Y$ w pozostałych przypadkach

a następnie znaleźć minimum z całki o wzorze:

$$\int_0^{+\infty} (\ D(t)) - q(t)\ )^2 dt \tag{1}$$

spośród wszystkich kombinacji punktów (N,M) , gdzie D(t) to wartość histogramu. W układzie dyskretnym poprzedni wzór wygląda następująco:

$$\sum (\ D(t) - q(t)\ )^2 \rightarrow \text{minimum} \tag{2}$$

przy czym dla $t\,\epsilon(0, N)$ oraz $t\,\epsilon(M, \infty)$ ma postać:

$$D(t)^2 \tag{3}$$

natomiast dla $t\,\epsilon\,\langle 0, N\rangle$ wygląda następująco:

$$(D(t) - q(t))^2 \tag{4}$$

Algorytm obliczania parametrów SD1 i SD2

Parametry SD1 i SD2 są wyznaczane według poniższych wzorów:

$$SD1 = \sqrt{\frac{1}{2} \cdot SDSD^2} \tag{5}$$

$$SD2 = \sqrt{2 \cdot SDNN^2 - \frac{1}{2} \cdot SDSD^2} \tag{6}$$

gdzie:

- SDNN to odchylenie standardowe interwałów RR: $SDNN = \sqrt{\frac{1}{N-1} \cdot \sum_{i=1}^{N} \left(\bar{RR} - RR_i\right)^2}$
- SDSD to odchylenie standardowe różnic pomiędzy dwoma sąsiadującymi interwałami: $SDSD = \sqrt{E\left\{\triangle RR_j^2\right\} - E\left\{\triangle RR_j\right\}^2}$

3.5.3 Opis procedur i metod

`GeometricAnalysis::runModule` – wirtualna funkcja uruchamiająca moduł HRV2.

Argumenty funkcji:

- `ECGInfo info` – dane o wczytanym sygnale
- `ECGRs ecgRs` – wektor numerów próbek zawierający załamki R
- `ECGHRV2 ecgHRV2` – instancja klasy przechowującej wyniki analizy zmienności rytmu serca metodami geometrycznymi

Funkcja zwraca:
Funkcja przekazuje wyniki metod geometrycznych analizy HRV do instancji klasy `ECGHRV2`.

Używane funkcje:

- `PrepareRRSignal`
- `MakeHistogramAndGeometricalParams`
- `MakePoincareAndSDParams`
- `SetHRV2Params`

Używane zmienne:

- `ECGRs rpeaks` – atrybut klasy `GeometricAnalysis` przechowujący wektor numerów próbek z załamkami R
- `double SamplingInterval` – atrybut klasy `GeometricAnalysis` przechowujący częstotliwość wczytanego sygnału

`GeometricAnalysis::PrepareRRSignal` – funkcja przekształca wektor numerów próbek z załamkami R na wektor interwałów RR w milisekundach.

Argumenty funkcji:

- `rpeaks` – wektor numerów próbek z załamkami R

Funkcja zwraca:
Funkcja zapisuje wektor interwałów RR w atrybucie klasy `RR_intervals`.

Używane funkcje:

- `gsl_vector_int_get` – metoda GSL pobierająca całkowitą wartość danego elementu z wektora
- `gsl_vector_set` – metoda GSL zapisująca zmiennoprzecinkową wartość w danym elemencie wektora

- `gsl_vector_scale` – metoda GSL mnożąca każdy element wektoru przez liczbę zmiennoprzecinkową

Używane zmienne:

- `unsigned int rpeaks_size` – długość wektora z numerami próbek załamków R
- `double SamplingInterval` – częstotliwość wczytanego sygnału

`GeometricAnalysis::MakeHistogramAndGeometricalParams` – funkcja tworzy wektor wartości histogramu, oblicza wysokość i pozycję kolumny histogramu reprezentującą najczęściej powtarzający się interwał RR, wylicza długość oraz pozycję podstawy trójkąta aproksymującego, oszacowuje wartość indeksu trójkątnego.

Argumenty funkcji:

- `RR_intervals` – wektor interwałów RR w milisekundach

Funkcja zwraca:

- histogram_x – pozycje kolumn histogramu
- histogram_y – wysokości kolumn histogramu
- X – pozycja najwyższej kolumny histogramu
- Y – wysokość najwyższej kolumny histogramu
- N – początek podstawy trójkąta aproksymującego
- M – koniec podstawy trójkąta aproksymującego
- HRVTriangularIndex – indeks trójkątny
- TINN – wskaźnik TINN

Używane funkcje:

- `gsl_vector_max` – metoda GSL zwracająca największy element danego wektora
- `gsl_vector_min` – metoda GSL zwracająca najmniejszy element danego wektora
- `gsl_vector_int_max_index` – metoda GSL zwracająca pozycję największego elementu danego wektora
- `gsl_vector_int_get` – metoda GSL pobierająca całkowitą wartość danego elementu z wektora
- `gsl_vector_get` – metoda GSL pobierająca zmiennoprzecinkową wartość danego elementu z wektora
- `gsl_vector_set` – metoda GSL zapisująca zmiennoprzecinkową wartość w danym elemencie wektora
- `gsl_vector_int_set` – metoda GSL zapisująca całkowitą wartość w danym elemencie wektora
- `gsl_interp_alloc` – metoda GSL tworząca wskaźnik do nowo utworzonego obiektu interpolacji
- `gsl_interp_accel_alloc` – metoda GSL tworząca wskaźnik na obiekt iteratora do wyszukiwania interpolacji
- `gsl_interp_init` – metoda GSL wyliczająca funkcję interpolującą

- `gsl_interp_eval` – metoda GSL zwracająca punkt funkcji interpolującej

Używane zmienne:

- `double RRmax` – najdłuższy interwał RR
- `double RRmin` – najkrótszy interwał RR
- `IntSignal Histogram` – tymczasowy wektor punktów histogramu
- `unsigned Histogram_size` – liczba wszystkich kolumn histogramu
- `unsigned int RR_intervals_size` – liczba wszystkich interwałów RR
- `double minimum` – zmienna przechowująca minimalną wartość całki z algorytmu wyznaczania TINN
- `double x[3], y[3]` – tablica współrzędnych trójkąta aproksymującego

`GeometricAnalysis::MakePoincareAndSDParams` – funkcja wylicza punkty wykresu Poincare oraz parametry: SD1 i SD2.

Argumenty funkcji:

- `RR_intervals` – wektor interwałów RR w milisekundach

Funkcja zwraca:

- poincare_x – współrzędne osi OX wykresu Poincare
- poincare_y – współrzędne osi OY wykresu Poincare
- parametr SD1
- parametr SD2

Używane funkcje:

- `gsl_vector_get` – metoda GSL pobierająca zmiennoprzecinkową wartość danego elementu z wektora
- `gsl_vector_int_set` – metoda GSL zapisująca całkowitą wartość w danym elemencie wektora
- `gsl_stats_sd` – metoda GSL wyliczająca odchylenie standardowe
- `gsl_vector_int_sub` – metoda GSL odejmująca dwa wektory

Używane zmienne:

- `unsigned int RR_intervals_size` – liczba wszystkich interwałów RR
- `IntSignal diff` – wektor różnic pomiędzy sąsiednimi interwałami RR

`GeometricAnalysis::SetHRV2Params` – funkcja przekazuje wyniki analizy HRV metodami geometrycznymi do instancji klasy ECGHRV2.

Argumenty funkcji:

- `ECGHRV2 &hrv2`

Funkcja zwraca:
Funkcja zwraca wyniki analizy geometrycznej do `ECGHRV2 &hrv2`.

Używane funkcje:

- wszystkie funkcje pozwalające na zapis wyników do atrybutów klasy ECGHRV2.

Używane zmienne:
Funkcja nie używa dodatkowych zmiennych.

`GeometricAnalysis::MakeRRsignal` – funkcja tworząca testowy wektor interwałów RR w milisekundach.

Argumenty funkcji:
Nie przyjmuje argumentów.

Funkcja zwraca:
Funkcja zwraca testowy wektor interwałów RR jako instancję klasy ECGRs.

Używane funkcje:

- `gsl_vector_int_set` – metoda GSL zapisująca całkowitą wartość w danym elemencie wektora
- `setRs` – zapisuje wektor w instancji klasy ECGRs

Używane zmienne:

- `int RRsignal_length` – długość wektora interwałów RR
- `int RRsignal[]` – tablica wartości interwałów RR

`GeometricAnalysis::setParams` – funkcja modyfikująca szerokość kolumny histogramu.

Argumenty funkcji:

- `ParametersTypes &parameterTypes` – zawiera dane ustawień poszczególnych modułów

Funkcja zwraca:
Nie zwraca niczego.

Używane funkcje:

- `parameterTypes.find(histogram_bin_length)` - wyszukuje parametr modyfikujący szerokość kolumny histogramu

Używane zmienne:

- `double HistogramParameter` – zmienna przechowująca rezultat funkcji `parameterTypes.find`.
- `double HistogramBinLength` – atrybut klasy `GeometricAnalysis` przechowujący szerokość kolumny histogramu.

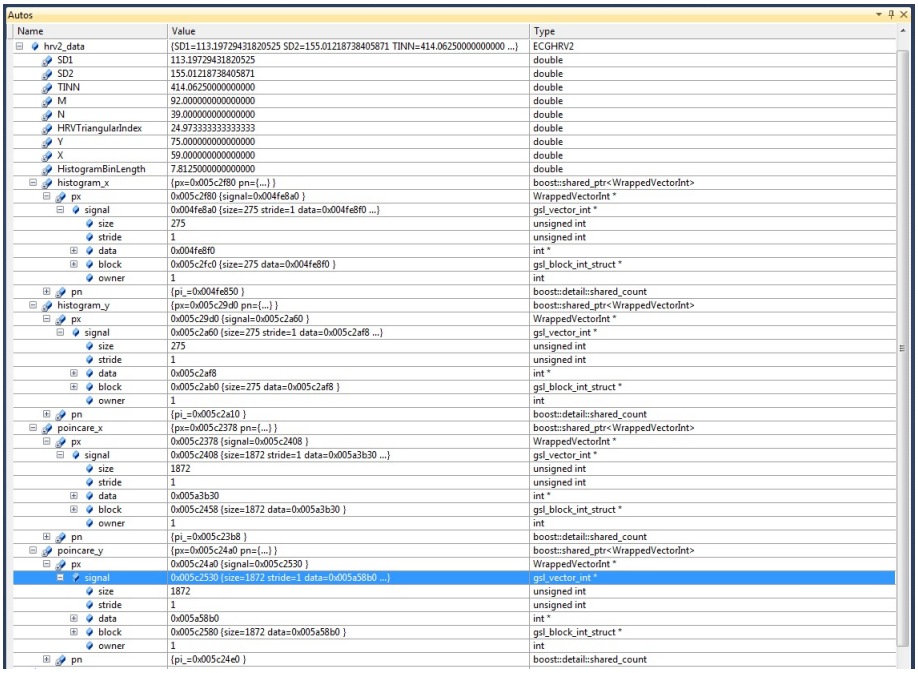
3.5.4 Warunki Testowania

Podczas procesu implementacji, moduł HRV2 poddawany był dwóm rodzajom testów:

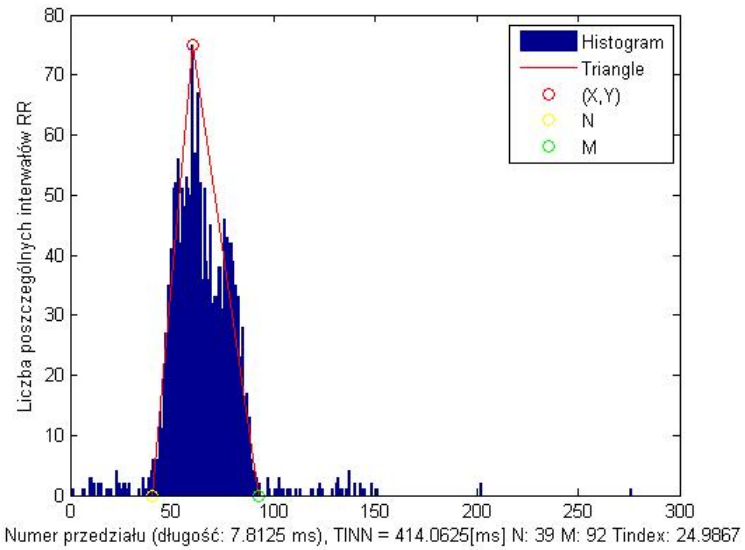
- Pierwszy test polegał na porównaniu wyników działania modułu zaimplementowanego w Matlabie oraz Visual Studio Ultimate 2010. W tym celu jako dane wejściowe wykorzystano wektor interwałów RR wczytywany z pliku tekstowego dla Matlaba z tablicy jednowymiarowej dla Visual Studio.
- Drugi test sprawdzał działanie programu, gdy ten korzystał z testowych wyników modułu R_PEAKS.

3.5.5 Wyniki

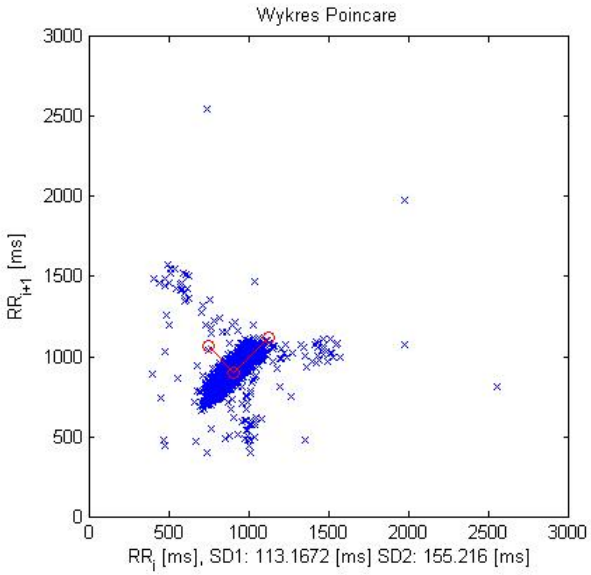
Rysunki 4, 5 oraz 6 prezentują wyniki pierwszego testu opisanego w poprzedniej podsekcji:



Rysunek 4: Wyniki modułu HRV2 – Visual Studio Ultimate 2010



Rysunek 5: Wykres histogramu – Matlab



Rysunek 6: Wykres Poincare – Matlab

3.6 HRV DFA

3.7 Klasyfikacja zespołów QRS

Autorzy: Krzysztof Bębenek

3.7.1 Opis zadania

Temat Metody detekcji morfologicznego pochodzenia zespołu QRS.

Opis Proces automatycznego klasyfikowania zespołów QRS należy do jednego z trudniejszych procesów podczas przetwarzania sygnałów EKG. Jest jednak on niezbędny w dalszych etapach analizy podczas których brane są pod uwagę tylko niepoprawne pobudzenia. Do prostych sposób oceny morfologi pobudzenia służą:

- współczynnik kształtu Malinowskiej,
- stosunek części ujemnej do dodatniej sygnału,

Dane ciąg próbek przefiltrowanego sygnału EKG, wektory QRS_onset oraz QRS_end określające początek oraz koniec zespołu QRS

Szukane moduł programu klasyfikujący zespołu QRS na podstawie ich morfologii

3.7.2 Badania literaturowe

3.7.3 Opis procedur i metod

3.7.4 Warunki testowania

3.7.5 Wyniki

3.8 ST interval

Autorzy: Bartłomiej Bułat i Krzysztof Piekutowski.

3.8.1 Opis zadania

Celem projektu jest wyznaczenie odcinków ST, pomiar poziomu odcinka ST względem linii izoelektrycznej i jego nachylenia, a także detekcja epizodów ST oraz parametry ilościowe i jakościowe

epizodów ST. Celowość analizy odcinka ST względem linii izoelektrycznej związana jest z predykcją choroby wieńcowej, miażdżycy oraz niedotlenieniem mięśnia serca.

Dane przyjmowane przez moduł:

- sygnał ECG_BASELINE;
- wektor numerów próbek załamków R z modułu R PEAKS;
- wektor numerów próbek punktów charakterystycznych z modułu WAVES:
  - QRS_{onset}
  - QRS_{end}
  - T_{end}

Dane zwracane przez moduł:

- wektor wykrytych odcinków ST zawierających informaje:
  - początek i koniec odcinka;
  - poziom izolinii;
  - pomiar poziomu względem izolinii;
  - pomiar nachylenia odcinka ST;
  - opis słowny interwału;
- wektor wykrytych epizodów wraz z parametrami:
  - numer próbki początku epizodu;
  - numer próbki końca epizodu.

3.8.2 Badania literaturowe

3.8.3 Opis procedur i metod

Główna funkcjonalność modułu znajduje się w klasie `STAnalysis`. Klasa ta, według przyjętego schematu rozszerza abstrakcyjny moduł analizy odcinka ST. W tej klasie znajdują się drzewo klas prywatnych reprezentujących poszczególne algorytmy analizy. W szczególności dwie klasy `SimpleAnalizer` oraz `ComplexAnalizer`. Pierwsza klasa reprezentuje najprostszy algorytm zaprezentowany w [**?**, p. 155], druga zaś implementuje lekko zmodyfikowany algorytm opisany w [**?**].

Lista i opis najważniejszych funkcji:

```
void STAnalysis::SimpleAnalizer::analyse(const int it, const ECGRs& rpeaks,
  const ECGWaves& waves, const ECGSignalChannel& signal,
  const  ECGInfo& info, ECGST& output);
```

Funkcja analizująca załamek ST w zespole QRS numer `it`. Wykorzystując punkty sygnału obliczone we wcześniejszych modułach (tj. QRS_{onset}, QRS_{end} oraz T_{end}) oblicza koniec odcinka ST i wylicza jego parametry:

- przesunięcie względem izolinii
- wartość nachylenia w stosunku do izolini
- długość odcinka ST

Uzywając wczesniej ustawionego parametru `simple_thresh`, odcinek ST określany jest jako „normalny”, „uniesiony” lub „obniżony”.

Parametry:

- `const int it` – numer badanego zespołu QRS, jako liczba porządkowa numerów próbek z tablicy załamków R.
- `const ECGR& rpeaks` – struktura zawierająca tablicę numerów próbek kolejnych załamków R
- `const ECGEaves& waves` – struktura zawierająca tablice przechowujące numery próbek punktów charakterystycznych kolejnych zespołów QRS: P_{onset}, QRS_{onset}, QRS_{end} oraz T_{end}.
- `const ECGSignalChanel& signal` – jeden kanał odfiltrowanego sygnału z usuniętym przesunięciem izolinii.
- `const ECGInfo& info` – struktura zawierająca informacje o badanym sygnale EKG, m.in. częstotliwość.
- `ECGST& output` – parametr wyjściowy, struktura zawierająca tablice wszystkich interwałów ST wraz z ich parametrami, oraz tablice zarejestrowanych epizodów ST wraz z ich parametrami.

```
void STAnalysis::ComplexAnalizer::analyse(const int it, const ECGRs& rpeaks,
  const ECGWaves& waves, const ECGSignalChannel& signal,
  const  ECGInfo& info, ECGST& output);
```

Funkcja analizująca załamek ST w zespole QRS numer `it`. Wykorzystując punkty sygnału obliczone we wcześniejszych modułach (tj. QRS_{onset}, QRS_{end} oraz T_{end}) oblicza koniec odcinka ST (wykorzystując bardziej zaawansowane algorytmy od poprzedniej funkcji), a następnie oblicza jego parametry:

- przesunięcie względem izolinii,
- wartość nachylenia w stosunku do izolinii,
- długość odcinka ST,
- klasyfikacja kształtu.

Używając wcześniej ustawionego parametru `complex_thresh`, odcinek ST określany jest jako „normalny", „uniesiony" lub „obniżony". Algorytm ocenia również to, czy załamek jest prosty czy zakrzywiony w oparciu o wcześniej ustawiony parametr `type_thresh`. Dla prostych odcinków określa kierunek, czy narasta, opada czy jest poziomy. Do określenia tej cechy wykorzystywany jest parametr `slope_thresh`. Dla zakrzywiony odcinków ST oceniana jest wypukłość krzywej. Szczegółowy opis działania znajduje się w poprzednim rozdziale.

Parametry:

- `const int it` – numer badanego zespołu QRS, jako liczba porządkowa numerów próbek z tablicy załamków R.
- `const ECGR& rpeaks` – struktura zawierająca tablicę numerów próbek kolejnych załamków R
- `const ECGEaves& waves` – struktura zawierające tablice przechowujące numery próbek punktów charakterystycznych kolejnych zespołów QRS: P_{onset}, QRS_{onset}, QRS_{end} oraz T_{end}.
- `const ECGSignalChanel& signal` – jeden kanał odfiltrowanego sygnału z usuniętym przesunięciem izolinii.
- `const ECGInfo& info` – struktura zawierająca informacje o badanym sygnale EKG, m.in. częstotliwość.
- `ECGST& output` – parametr wyjściowy, struktura zawierająca tablice wszystkich interwałów ST wraz z ich parametrami, oraz tablice zarejestrowanych epizodów ST wraz z ich parametrami.

Funkcje pomocnicze klasy `ComplexAnalizer`:

```
int STAnalysis::ComplexAnalizer::getTPeak(const OtherSignal& sig,
  int from, int to);
```

Funkcja wyszukująca położenia punktu T_{peak} na zadanym odcinku sygnału. Początkiem wyszukiwania szczytu fali T jest zwykle punkt 20ms za punktem QRS_{end}, a punktem końcowym jest koniec fali T. Funkcja wykorzystuje dyskretną pochodną sygnału.

Parametry:

- `const OtherSignal& sig` - cały, odfiltrowany sygnał EKG
- `int from` - numer próbki w której należy zacząć poszukiwania
- `int to` - numer próbki w której należy skończyć poszukiwania

```
std::pair<int, double> maxDistanceSample(const OtherSignal& signal,
  int from, int to);
```

Funkcja szukająca numeru próbki najbardziej oddalonego od liniowej interpolacji fragmentu sygnału miedzy dwoma punktami. Oprócz numeru próbki od początku sygnału, zwracana jest wartość największej różnicy miedzy punktem sygnału, a prostą interpolacji.

Parametry:

- `const OtherSignal& signal` - badany sygnał
- `int from` - numer próbki będącej początkiem interesującego fragmentu sygnału, równocześnie, pierwszy punkt interpolacji liniowej.
- `int to` - numer próbki będącej końcem interesującego fragmentu sygnału, równocześnie, drugi punkt interpolacji

```
std::pair<int, int> overBelowSamples(const OtherSignal& signal,
  int from, int to);
```

Funkcja obliczająca ilość próbek ponad i poniżej prostej interpolującej sygnał miedzy dwoma punktami. Funkcja wykorzystywana jest do określenia wypukłości odcinka ST.

Parametry:

- `const OtherSignal& signal` - badany sygnał
- `int from` - numer próbki będącej początkiem interesującego fragmentu sygnału, równocześnie, pierwszy punkt interpolacji liniowej.
- `int to` - numer próbki będącej końcem interesującego fragmentu sygnału, równocześnie, drugi punkt interpolacji

3.8.4 Warunki testowania

3.8.5 Wyniki

3.9 T wave alt

3.10 HRT