



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 PEKALA
Krzysztof PIEKUTOWSKI
Grzegorz PIETRZYK
Łukasz PODOLSKI
Mikołaj RZEPKA
Agata SITNIK
Leszek SOSNOWSKI
Aleksander STELIGA
Mateusz ŚLAŻYŃSKI
Łukasz ZIEŃKOWSKI

Opiekun:

mgr inż. Tomasz PIĘCIAK

23 stycznia 2013

Spis treści

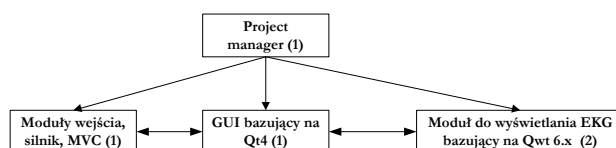
1	Specyfikacja zadania	3
2	Specyfikacja techniczna rozwiązania	4
2.1	Wykorzystane narzędzia	4
2.2	Projekt systemu	4
2.3	Główne elementy kodu systemu	6
3	Opisy modułów	7
3.1	Usuwanie linii bazowej	7
3.1.1	Opis zadania	8
3.1.2	Badania literaturowe	8
3.1.3	Opis procedur i metod	8
3.1.4	Warunki testowania	8
3.1.5	Wyniki	8
3.2	Wykrywanie załamków R	8
3.3	Waves	8
3.4	HRV1	8
3.5	HRV2	8
3.5.1	Opis zadania	9
3.5.2	Badania literaturowe	9
3.5.3	Opis procedur i metod	10
3.5.4	Warunki Testowania	14
3.5.5	Wyniki	14
3.6	HRV DFA	14
3.7	Klasyfikacja zespołów QRS	14
3.7.1	Opis zadania	14
3.7.2	Badania literaturowe	16
3.7.3	Opis procedur i metod	16
3.7.4	Warunki testowania	16
3.7.5	Wyniki	16
3.8	ST interval	16
3.8.1	Opis zadania	16
3.8.2	Badania literaturowe	17
3.8.3	Opis procedur i metod	17
3.8.4	Warunki testowania	19
3.8.5	Wyniki	19
3.9	T wave alt	19
3.10	HRT	19

1 Specyfikacja zadania

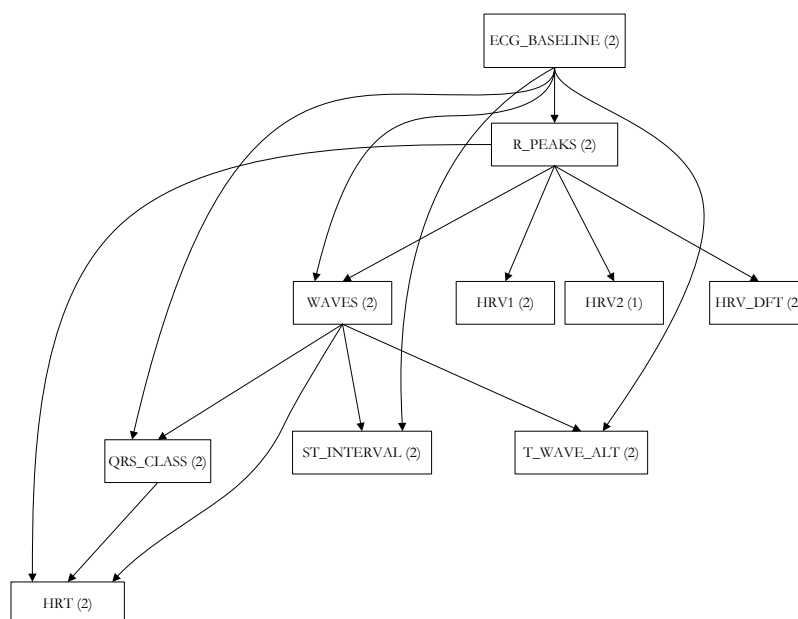
Celem projektu jest stworzenie zintegrowanego systemu pozwalającego na przeglądanie i automatyczną analizę sygnału EKG. Sygnał dostarczany jest w formie cyfrowej w standardzie wykorzystywanym w MIT-BIH Arrhythmia Database. Różne etapy przetwarzania, takie jak usuwanie linii bazowej, detekcja załamków R czy klasyfikacja zespołów QRS wykonywana jest przez różne zespoły (szczegółowe opisy specyfikacji modułów znajduje się w sekcji 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. W nawiasach obok nazwy zespołu podana jest liczba osób wchodzących w jego skład.

Moduły zarządzania oprogramowaniem:



Moduły obliczeniowe:



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
 - KissFFT (zawarta w źródłach projektu)
- System budowania:
 - system budowy Microsoft Visual Studio 2010 (Windows)
 - CMake (Linux, MacOS X), generuje plik Makefile
 - pakiet L^AT_EX (dokumentacja)
 - program Dia (większość diagramów)

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.

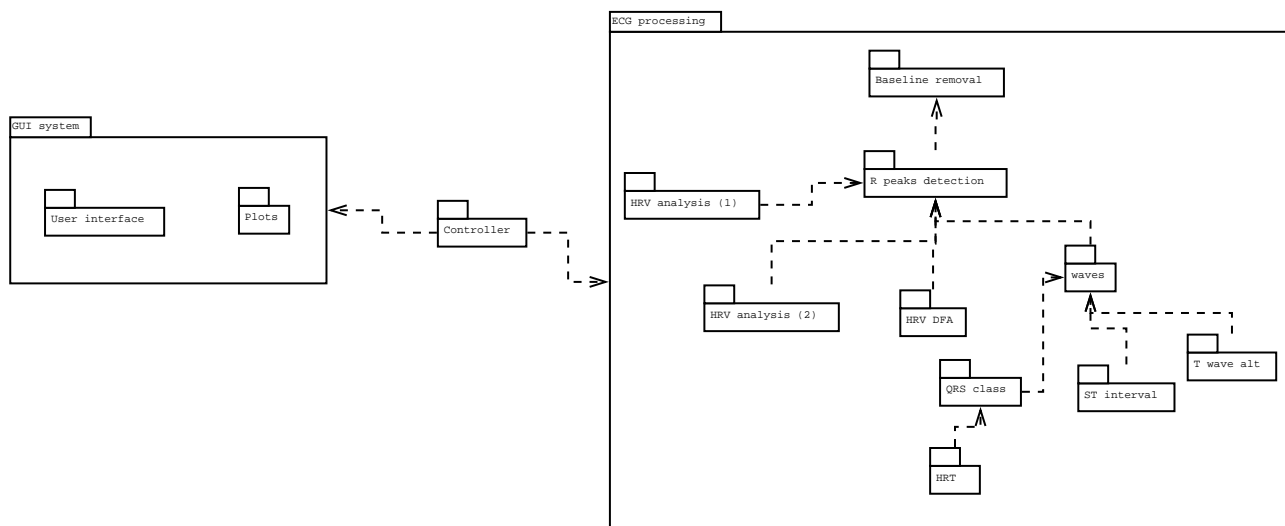
2.2 Projekt systemu

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. Zastosowano obiektowe podejście przy projektowaniu hierarchii klas realizujących przetwarzanie (rys. 5), umożliwiające bezproblemową wymianę implementacji dowolnego modułu na inną.

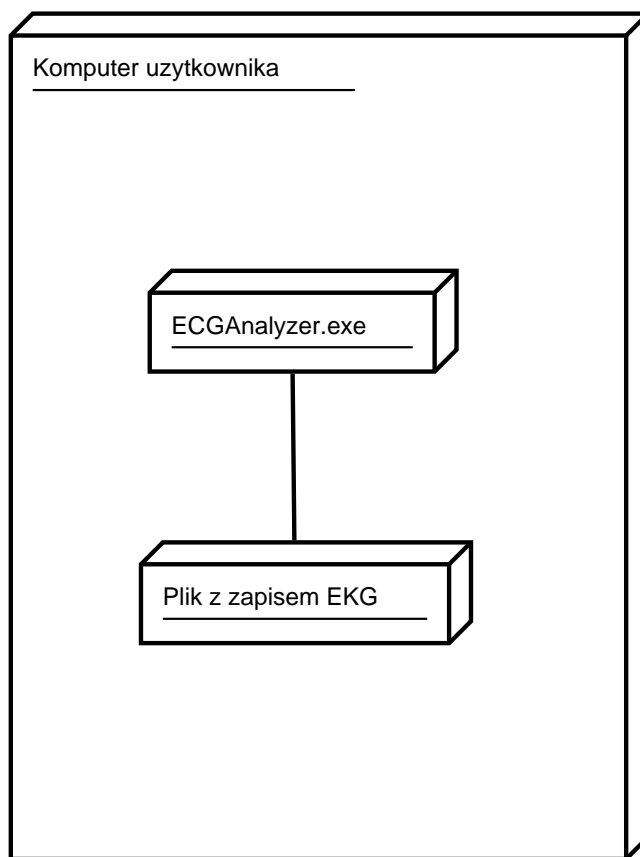
Samo przetwarzanie realizowane jest wielofazowo – moduły przeliczane są sekwencyjnie na osobnym wątku niż wątek zdarzeń GUI, co pozwala na zatrzymanie zbyt długo trwającej operacji. Zaimplementowano także mechanizm buforowania wyników, dzięki czemu przy zmianie parametrów pewnego modułu nie ma konieczności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

Cała aplikacja napisana jest jako samodzielny, wieloplatformowy program (rys. 3) – 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.

Diagram sekwencji przedstawiający sposób uruchamiania modułów jest przedstawiony na rys. 4. Każdy moduł otrzymuje najpierw zestaw parametrów (o ile posiada jakieś parametry), następnie zaś jest uruchamiany funkcją runModule. Kolejne moduły uruchamiane są sekwencyjnie według identycznego schematu. Wskutek akcji użytkownika możliwe jest przerwanie analizy pomiędzy poszczególnymi modułami.



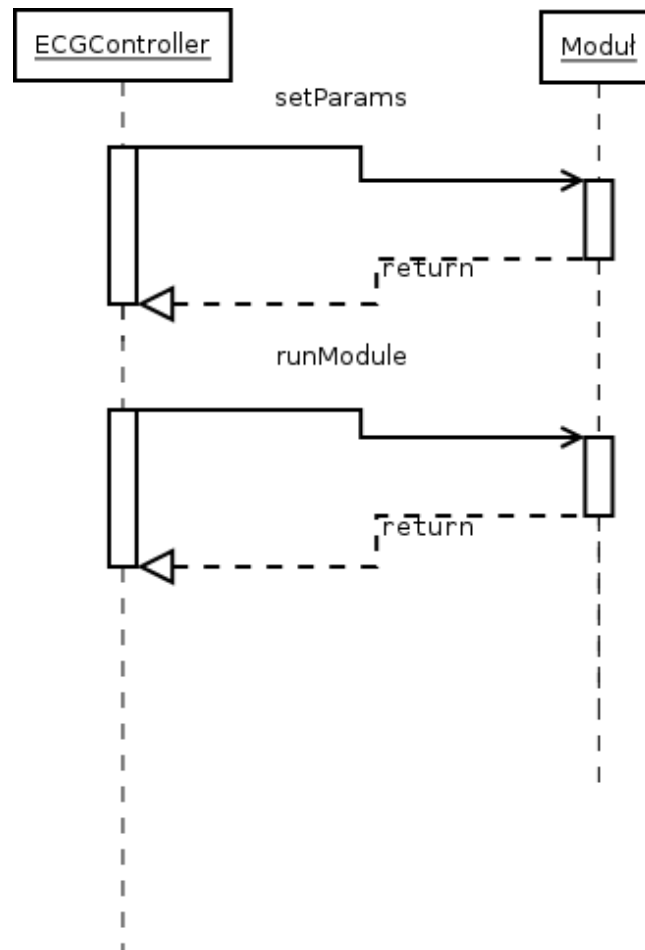
Rysunek 2: Diagram pakietów.



Rysunek 3: Diagram wdrożenia.

Diagram klas został przedstawiony na rys. 5. Obrazuje on zależności pomiędzy poszczególnymi klasami zrealizowanymi w ramach modułów oraz klasami pomocniczymi przechowującymi informacje przekazywane pomiędzy modułami, kontrolerem i widokiem.

Diagram przypadków użycia, przedstawiony na rys. 6 pokazuje różne scenariusze, w których może być używany zrealizowany system. Każdemu przypadkowi odpowiada jeden lub więcej modułów. zachodzące pomiędzy nimi zależności są ukazywane relacjami zawierania.



Rysunek 4: Diagram sekwencji.

2.3 Główne elementy kodu systemu

Głównym elementem systemu jest klasa `ECGController` 5. Metoda `rerunAnalysis`, która się w niej znajduje odpowiada za kilka rzeczy:

1. Upewnianie się, że moduły mają aktualne parametry.
2. Rozpoczynanie analizy zapisu EKG.
3. Przerywanie analizy w przypadku, gdy już ona trwa.
4. Raportowanie do GUI o kolejnych etapach pracy systemu (który moduł aktualnie się wykonuje).

Ostatnie zadanie wykonywane jest za pomocą dwóch funkcyjnych argumentów wejściowych.

Parametry, ustawiane metodami `setParam*` mają postać mapy przypisującej nazwom poszczególnych parametrów odpowiadające im wartości liczbowe. Wykorzystywane są przede wszystkim w modułach usuwania linii bazowej oraz wykrywania załamków R.

Do klas pomocniczych można zaliczyć między innymi `IntSignal` czy `ECGSignalChannel`. Sygnały, które przechodzą pomiędzy systemami czynią to właśnie poprzez obiekty tych klas. Są one zrealizowane jako sprytnie wskaźniki do struktur trzymających wektory w stylu C biblioteki GSL, dzięki czemu możliwe jest pogodzenie szybkości działania z wygodną składnią i semantyką sprytnego wskaźnika.

Poszczególne moduły są tworzone w hierarchii obiektowej (rys. 5). Umożliwia to podmianę w przyszłości modułu na lepszą implementację bez poprawiania reszty silnika oraz przyczynia się do zmniejszenia współzależności.

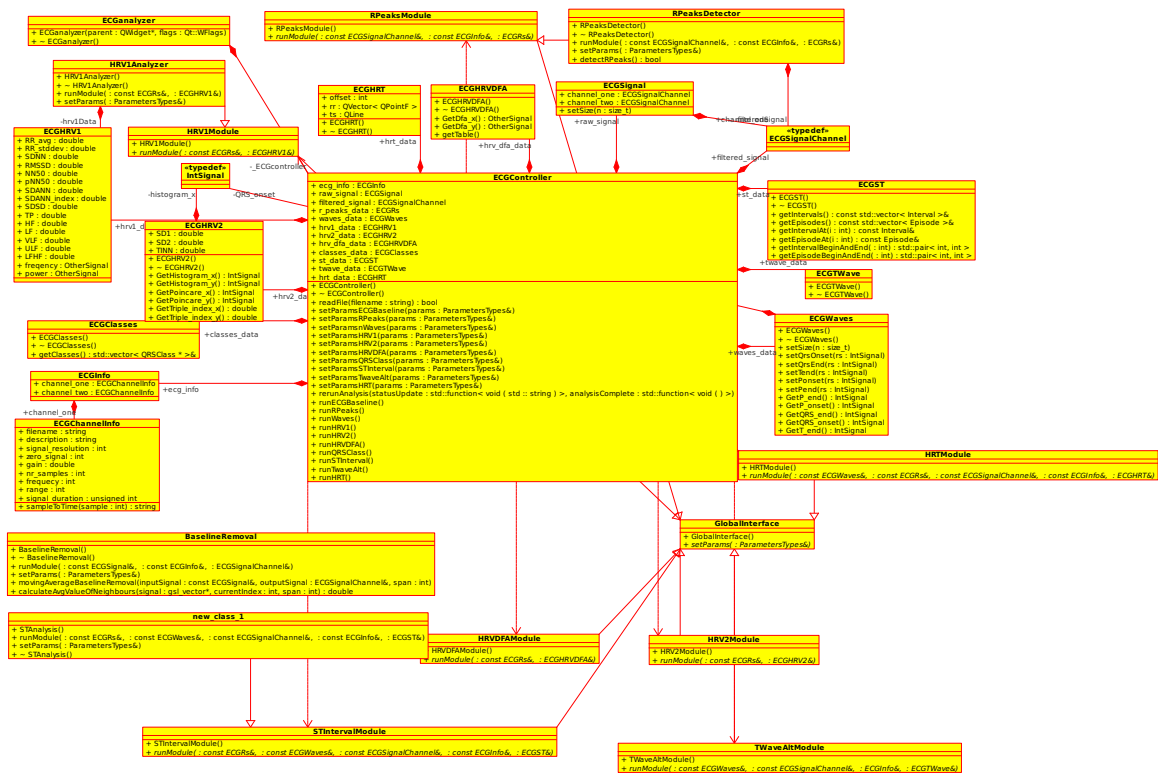


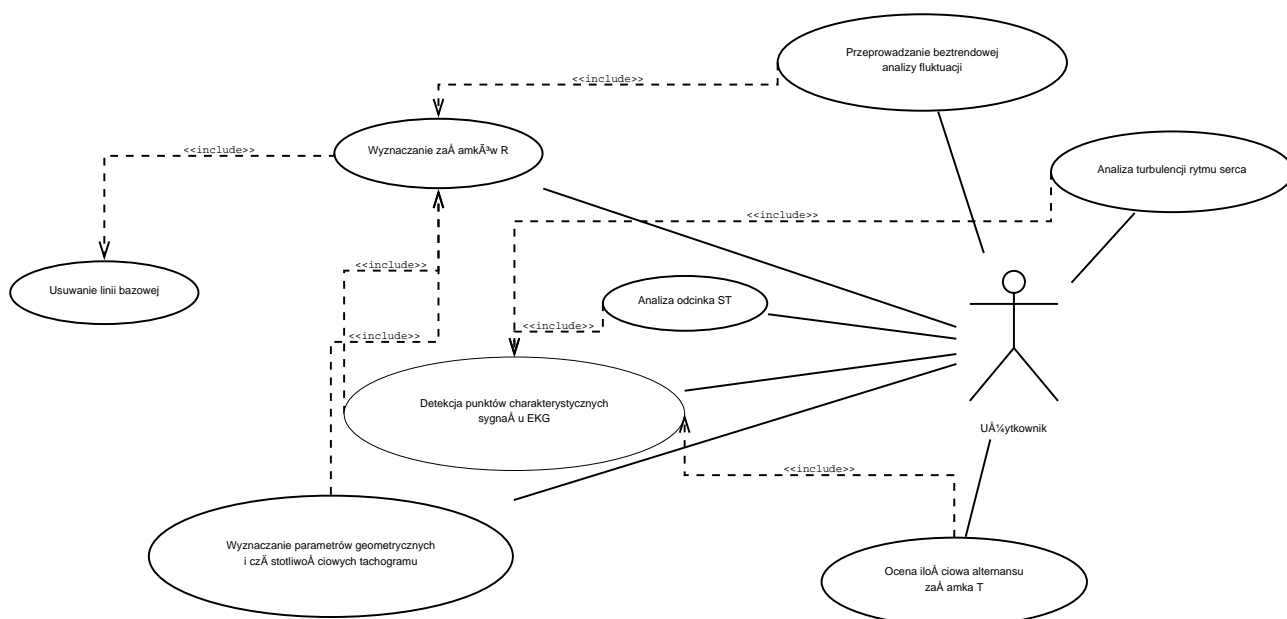
Diagram: class diagram Page 1

Rysunek 5: Diagram klas.

3 Opisy modułów

3.1 Usuwanie linii bazowej

Autorzy: Weronika Łabaj i Piotr Matuszkiewicz.



Rysunek 6: Diagram klas.

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 adaptacyjne 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

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 [2]. 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 [3]

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

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

a następnie znaleźć minimum z całki o wzorze:

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

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

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

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

$$D(t)^2 \quad (3)$$

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

$$(D(t) - q(t))^2 \quad (4)$$

Algorytm obliczania parametrów SD1 i SD2

Parametry SD1 i SD2 są wyznaczane według poniższych wzorów:

$$SD1 = \sqrt{\frac{1}{2} \cdot SDSD^2} \quad (5)$$

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

gdzie:

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

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 załawkami R
- `double SamplingInterval` – atrybut klasy `GeometricAnalysis` przechowujący częstotliwość wczytanego sygnału

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

Argumenty funkcji:

- `rpeaks` – wektor numerów próbek z załawkami 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 wektora przez liczbę zmiennoprzecinkową

Używane zmienne:

- `unsigned int rpeaks_size` – długość wektora z numerami próbek załawków 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 ¶meterTypes` – 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 7, 8 oraz 9 prezentują wyniki pierwszego testu opisanego w poprzedniej podsekcji:

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 sposobów oceny morfologii pobudzenia służą:

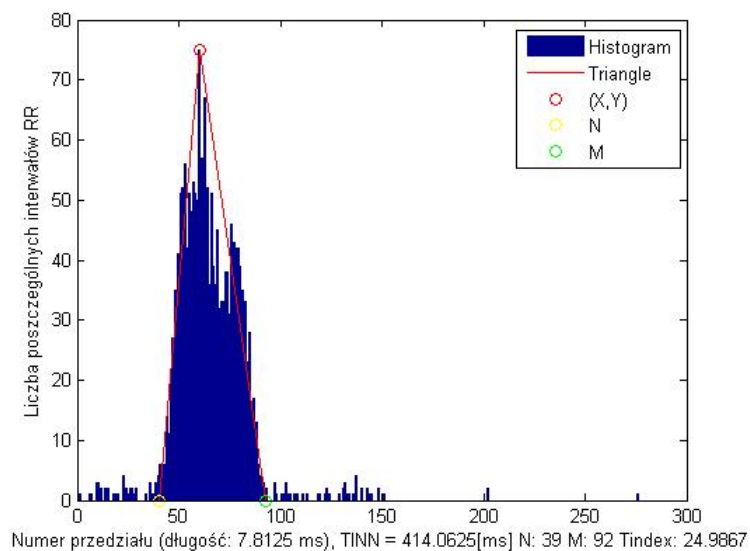
- współczynnik kształtu Malinowskiej,
- stosunek części ujemnej do dodatniej sygnału,

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

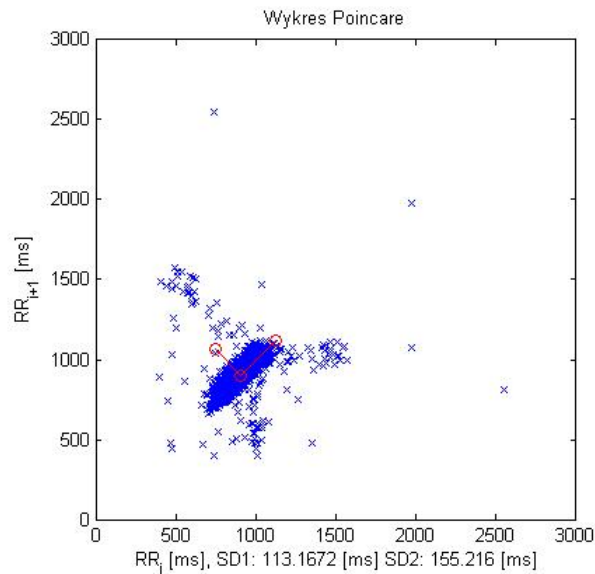
Szukane moduł programu klasyfikujący zespół QRS na podstawie ich morfologii

Name	Value	Type
hmv2_data	{SD1=113.19729431820525 SD2=155.01218738405871 TINN=414.0625000000000 ...}	ECGHRV2
SD1	113.19729431820525	double
SD2	155.01218738405871	double
TINN	414.06250000000000	double
M	92.00000000000000	double
N	39.00000000000000	double
HRVTriangularIndex	24.973333333333333	double
Y	75.00000000000000	double
X	59.00000000000000	double
HistogramBinLength	7.812500000000000	double
histogram_x	{px=0x005c2f80 pn={...}}	boost::shared_ptr<WrappedVectorInt>
px	0x005c2f80 (signal=0x004fe8a0)	WrappedVectorInt *
signal	0x004fe8a0 (size=275 stride=1 data=0x004fe8f0 ...)	gsl_vector_int *
size	275	unsigned int
stride	1	unsigned int
data	0x004fe8f0	int *
block	0x005c2f00 (size=275 data=0x004fe8f0)	gsl_block_int_struct *
owner	1	int
pn	{pi=0x004fe850}	boost::detail::shared_count
histogram_y	{py=0x005c29d0 pn={...}}	boost::shared_ptr<WrappedVectorInt>
py	0x005c29d0 (signal=0x005c2af8)	WrappedVectorInt *
signal	0x005c2af8 (size=275 stride=1 data=0x005c2af8 ...)	gsl_vector_int *
size	275	unsigned int
stride	1	unsigned int
data	0x005c2af8	int *
block	0x005c2ab0 (size=275 data=0x005c2af8)	gsl_block_int_struct *
owner	1	int
pn	{pi=0x005c2a10}	boost::detail::shared_count
poincare_x	{px=0x005c2378 pn={...}}	boost::shared_ptr<WrappedVectorInt>
px	0x005c2378 (signal=0x005c2408)	WrappedVectorInt *
signal	0x005c2408 (size=1872 stride=1 data=0x005a3b30 ...)	gsl_vector_int *
size	1872	unsigned int
stride	1	unsigned int
data	0x005a3b30	int *
block	0x005c2458 (size=1872 data=0x005a3b30)	gsl_block_int_struct *
owner	1	int
pn	{pi=0x005c23b8}	boost::detail::shared_count
poincare_y	{py=0x005c24a0 pn={...}}	boost::shared_ptr<WrappedVectorInt>
py	0x005c24a0 (signal=0x005c2530)	WrappedVectorInt *
signal	0x005c2530 (size=1872 stride=1 data=0x005a58b0 ...)	gsl_vector_int *
size	1872	unsigned int
stride	1	unsigned int
data	0x005a58b0	int *
block	0x005c2580 (size=1872 data=0x005a58b0)	gsl_block_int_struct *
owner	1	int
pn	{pi=0x005c24a0}	boost::detail::shared_count

Rysunek 7: Wyniki modułu HRV2 – Visual Studio Ultimate 2010



Rysunek 8: Wykres histogramu – Matlab



Rysunek 9: Wykres Poincare – Matlab

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 informacje:
 - 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 `SimpleAnalyzer` oraz `ComplexAnalyzer`. Pierwsza klasa reprezentuje najprostszy algorytm zaprezentowany w [1, p. 155], druga zaś implementuje lekko zmodyfikowany algorytm opisany w [4].

Lista i opis najważniejszych funkcji:

```
void STAnalysis::SimpleAnalyzer::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

Używając wcześniej 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 załamek R.
- `const ECGRs& rpeaks` – struktura zawierająca tablicę numerów próbek kolejnych załamek R
- `const ECGWaves& waves` – struktura zawierająca tablice przechowujące numery próbek punktów charakterystycznych kolejnych zespołów QRS: P_{onset} , QRS_{onset} , QRS_{end} oraz T_{end} .
- `const ECGSignalChannel& 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::ComplexAnalyzer::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 zakrzywionych 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 załamek R.
- `const ECGR& rpeaks` – struktura zawierająca tablicę numerów próbek kolejnych załamek 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.

Funkcje pomocnicze klasy `ComplexAnalyzer`:

```
int STAnalysis::ComplexAnalyzer::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 między dwoma punktami. Oprócz numeru próbki od początku sygnału, zwracana jest wartość największej różnicy między 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 między 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

Literatura

- [1] Piotr Augustyniak. *Przetwarzanie sygnałów elektrodiagnostycznych*. Uczelniane wydawnictwa naukowo-dydaktyczne AGH, 2001.
- [2] T. Krauze, P. Guzik, and H. Wysocki. Zmienność rytmu serca: aspekty techniczne. *Nowiny Lekarskie*, 9(ISSN 0860-7397):973 – 984, 2001.
- [3] Task Force of The European Society of Cardiology, The North American Society of Pacing, and Electrophysiology (Membership of the Task Force listed in the Appendix). Heart rate variability standards of measurement, physiological interpretation, and clinical use. *European Heart Journal*, 17:354–381, 1996.
- [4] Zhao Shen, Chao Hu, and Jingsheng Liao. An algorithm of st segment classification and detection. *Proceedings of the 2010 IEEE International Conference on Automation and Logistics*, 2010.