

Spis Treści

1.	Wstęp.....	3
1.1	Opis problemu.....	3
1.2	Cel projektu.....	3
1.3	Schemat blokowy z dokumentacji wstępnej.....	3
2.	Realizacja.....	4
2.1	Hardware	4
	Wnioski	6
2.2	Oprogramowanie	7
	Wnioski	7
2.3	Aplikacja - GUI	8
	Wnioski	9
2.4	Filtracja.....	10
	Wnioski	15
2.5	Podsystem Bluetooth i Mechanizm Powiadamiania Opiekuna	16
	Wnioski	18

1. Wstęp

1.1 Opis problemu

Bezdech senny polega na powtarzającym się ograniczeniu przepływu powietrza podczas snu. Objawia się przerwami w oddychaniu trwającymi 10 sekund lub dłużej. Brak oddechu podczas snu może przyczynić się do zwiększenia ryzyka wystąpienia udaru mózgu, choroby Alzheimera oraz jaskry.

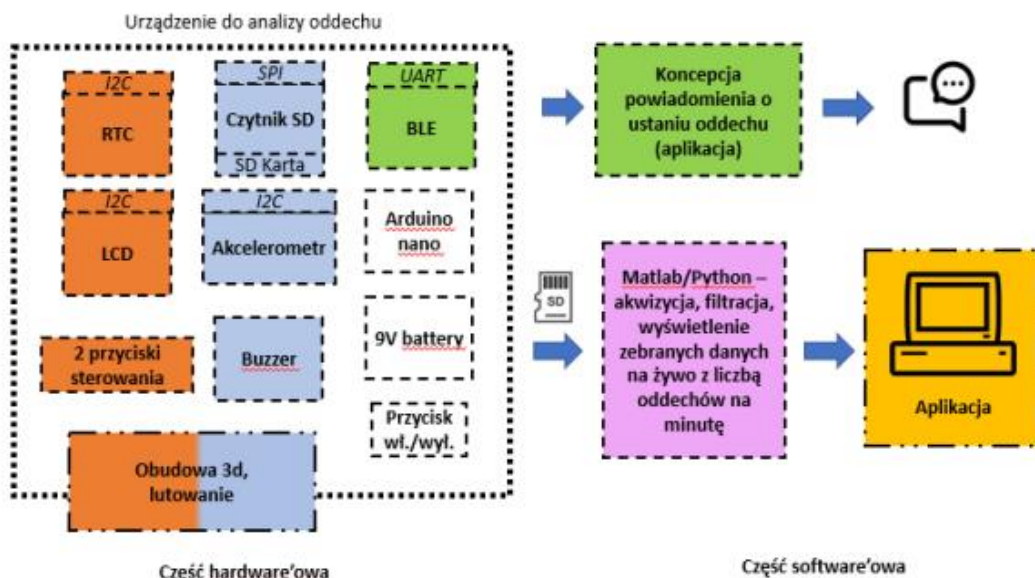
Leczenie jest bardzo różnorodne i zależy od przyczyny. Te z kolei dzielą się na ośrodkowe (neurologiczne) i zaporowe (zaburzenia przepływu powietrza podczas wdechu przez górne drogi oddechowe).

Zalecane jest przyjmowanie odpowiedniej pozycji ciała podczas snu, odstawienie wszelkich używek i substancji wpływających na szybkość oddechu i bicia serca oraz utrzymywanie prawidłowej wagi.

1.2 Cel projektu

Celem pracy jest zbudowanie urządzenia do analizy oddechu, które pozwoliłoby monitorować oddech przez całą noc. Jest to ważny element leczenia, ponieważ pozwala na wyciągnięcie wniosków na temat zdrowia pacjenta. Daje nam to informację o tym, czy wdrożone leczenie jest poprawne i wystarczające.

1.3 Schemat blokowy z dokumentacji wstępnej



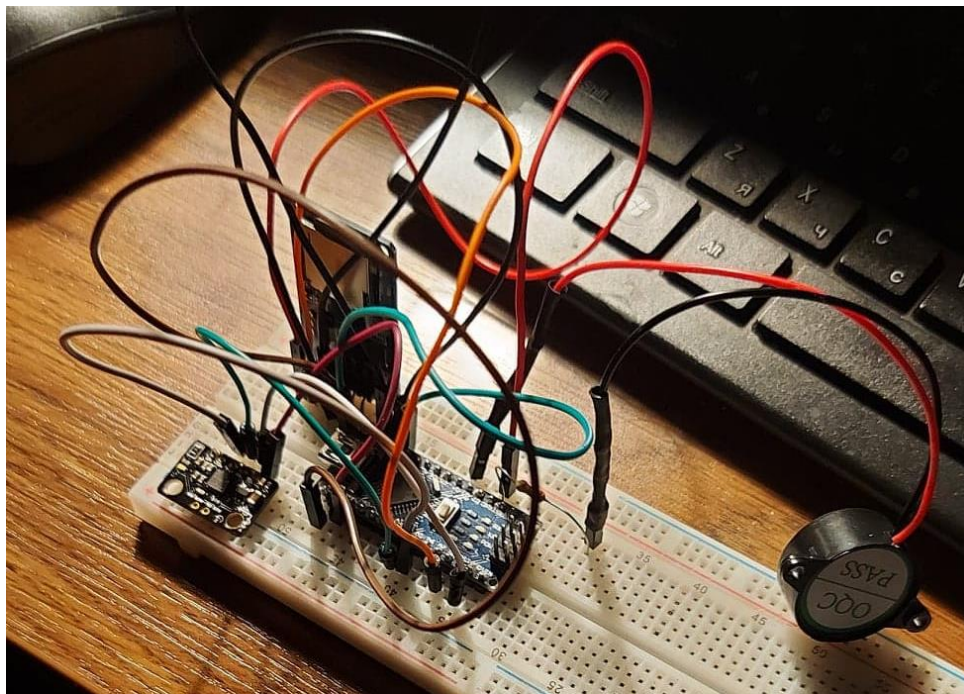
2. Realizacja

2.1 Hardware

Do zrealizowania projektu użyliśmy:

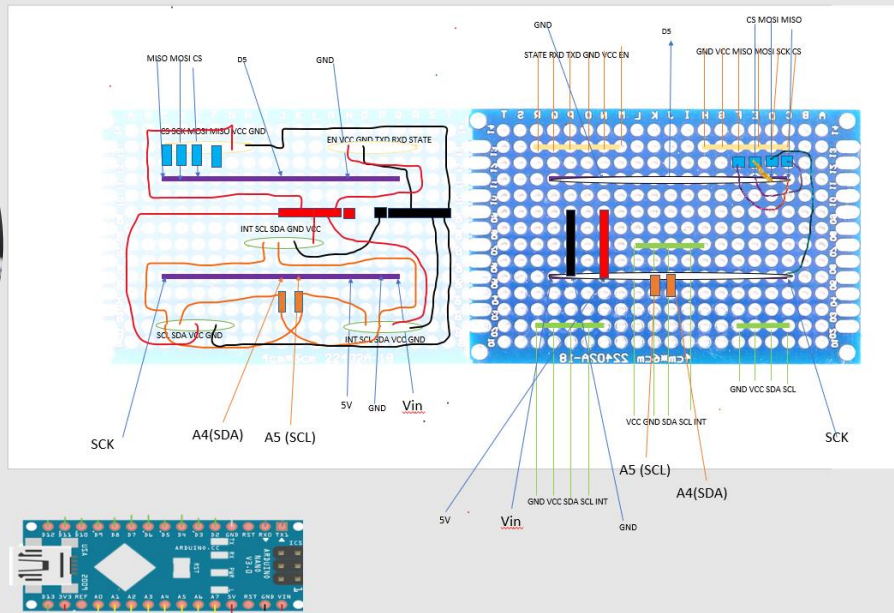
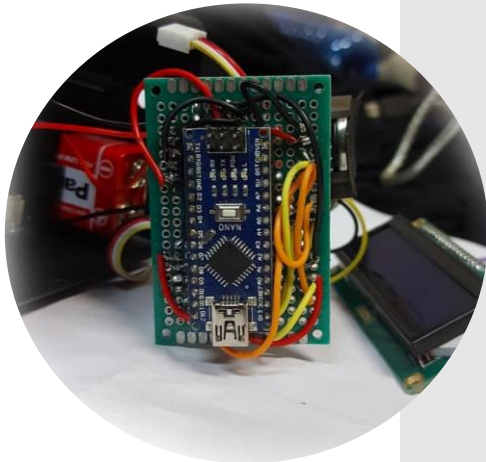
- MPU-6050 3-osiowy akcelerometr i żyroskop I2C moduł
- Moduł Bluetooth 4.0 BLE - HM-10
- Arduino Nano
- Moduł czytnika kart microSD
- Karta pamięci microSD
- Wyświetlacz LCD 2x16 znaków + konwerter I2C
- RTC DS1307 EEPROM 24C32 I2C – zegar czasu rzeczywistego + bateria 3.3V
- Buzzer
- Bateria 9V
- Inne: stabilizator, rezystory, kondensatory

Wszystkie wyżej wymienione elementy zostały połączone. Czytnik karty microSD, akcelerometr, moduł Bluetooth, wyświetlacz, zegar czasu rzeczywistego oraz buzzer powiadamiający o zatrzymaniu oddechu został podłączony do Arduino Nano.



Przykład zrealizowanego prototypu czytnika, akcelerometru i buzzera

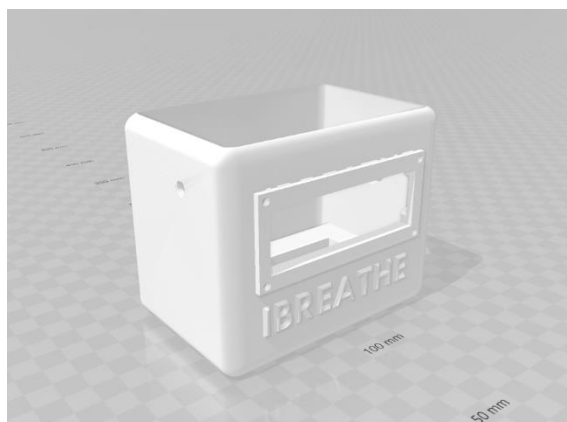
Została sporządzona płytki prototypowa. Koncepcja polega na tym, że da się z łatwością wymienić elementy modułów urządzenia. Dodatkowo taka płytki zaoszczędza miejsca, więc mogliśmy stworzyć wygodną do noszenia obudowę.



Ułożenie elementów na płytce stykowej

Po przetestowaniu systemu na płytce prototypowej elementy zostały zlutowane w taki sposób, aby mieściły się w zaprojektowanej i wydrukowanej obudowie.

Obudowa została stworzona przy wykorzystaniu programu 3D Builder. Realizacja polegała na znalezieniu open-source'owych elementów naszego urządzenia i umieszczeniu/wycięciu ich z elementu sześcianu.



Zrzut ekranu obudowy w programie 3D Builder



Gotowe Urządzenie



Korektor postury został wykorzystany jako element do przenoszenia naszego urządzenia. Pacjent może wygodnie się poruszać, ponieważ urządzenie jest wygodnie przymocowane to korektora i pozwala nam to też monitorować oddech, ponieważ urządzenie ciągle znajduje się w okolicach brzucha.

Wnioski

Udało nam się zrealizować większość z naszych założeń hardware'owych. Porównując gotowy produkt ze schematem blokowym układu akwizycji z dokumentacji wstępnej, który jest widoczny w punkcie 1.3, tym czego nie zaimplementowaliśmy są dwa przyciski sterowania, które okazały się zbędne w trakcie realizacji.

2.2 Oprogramowanie

Oprogramowanie większości z elementów składających się na urządzenie do analizy oddechu polegało na znalezieniu i wykorzystaniu odpowiedniej biblioteki Arduino.

Czytnik kart SD	→	biblioteka SD
Akcelerometr	→	biblioteka MPU 9250
RTC	→	biblioteka DS1307RTC
Wyświetlacz LCD	→	biblioteka LiquidCrystal_I2C
Bluetooth	→	biblioteka SoftwareSerial

Do obsługi buzzera użyty został system przerwań licznika Arduino Nano.

Co zliczenie w obsłudze sprawdzana jest wartość zmiennej counter, która zlicza czas bezruchu pacjenta. Jeżeli bezruch trwa wystarczająco długo uruchamiany jest dźwięk informujący o braku oddechu. W przypadku wykrycia ruchu pacjenta counter się resetuje.

```
cli(); //stop interrupts for till we make the settings
/*1. First we reset the control register to make sure we start with everything disabled.*/
TCCR1A = 0; // Reset entire TCCR1A to 0
TCCR1B = 0; // Reset entire TCCR1B to 0

/*2. We set the prescaler to the desired value by changing the CS10 CS12 and CS12 bits. */
TCCR1B |= 00000001; // no prescaler, just clk_I0

/*3. We enable compare match mode on register A*/
TIMSK1 |= 00000010; //Set OCIE1A to 1 so we enable compare match A

/*4. Set the value of register A to 31250*/
OCR1A = 31250; //Finally we set compare register A to this value
sei(); //Enable back the interrupts

ISR(TIMER1_COMPA_vect){
  TCNT1 = 0; //First, set the timer back to 0 so it resets for next interrupt
  if(buzz_detect > 900 && buzz_detect <950){ digitalWrite(BUZZER, HIGH); died_signal =1000;} // Send 10KHz sound signal to 2 pin buzzer
}
```

Przykład uruchomienia licznika i obsługa przerwania

Wnioski


W trakcie realizacji, okazało się, że dwa urządzenia, akcelerometr i zegar RTC, komunikujące się z mikrokontrolerem I2C posiadają takie same adresy. Udało nam się rozwiązać ten problem przez wymianę akcelerometru na taki o innym adresie.

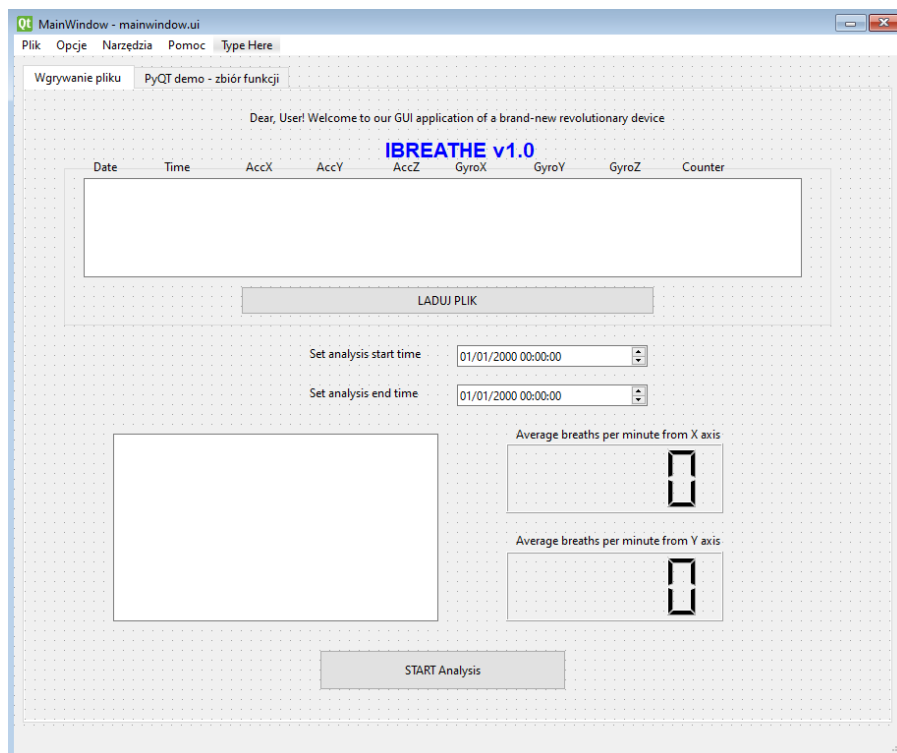
2.3 Aplikacja - GUI

Użytkownik ładuje plik txt pobrany z czytnika kart. Następnie może regulować czas, od którego zamierza analizować dane. Po naciśnięciu przycisku *Start Analysis* generowane są wykresy i statystyki.

Całość jest napisana w języku Python. Używana jest nakładka PyQt6, umożliwia ona tworzenie interfejsu graficznego. Formatkę graficzną tworzy się w QT Designer.

Obsługa interfejsu

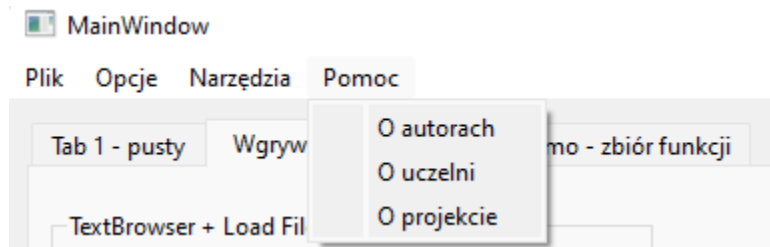
Po uruchomieniu aplikacji  `game_gui.py` powinno wyskoczyć poniższe okienko:



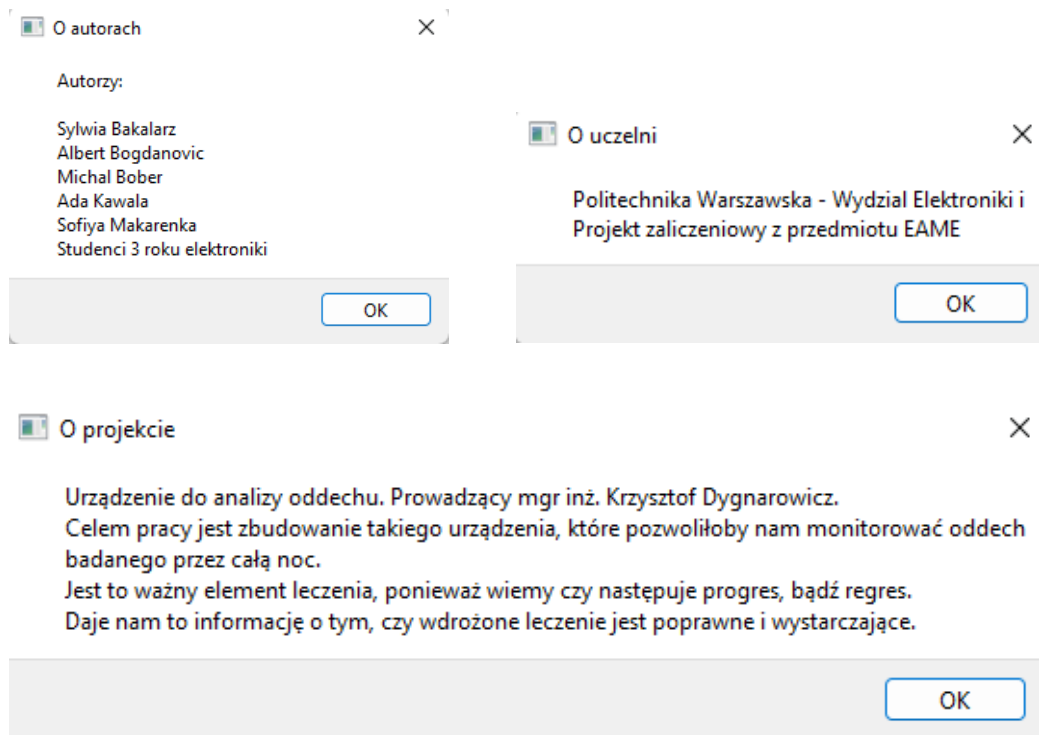
Przykładowy zrzut ekranu z GUI

Dodatkowa Funkcjonalność

W zakładce Pomoc mamy 3 podzakładki: *O autorach*, *O uczelni* oraz *O projekcie*.



Po kliknięciu każdej z nich dostajemy krótkie informacje o wybranej przez nas kategorii.



Zrzuty ekranu z zakładki Pomoc

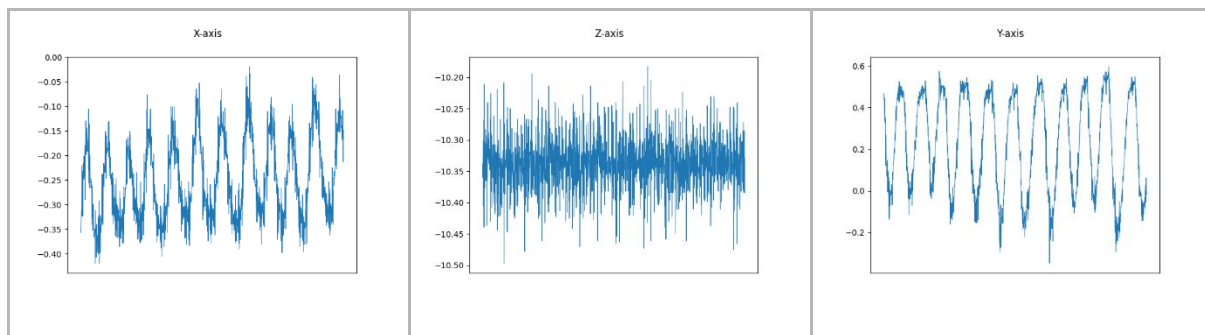
Wnioski

W zakładce Opcje oraz Narzędzia obecnie nie zostało wstawione nic. W dalszej części rozwijania projektu możliwe, że pojawią się tam ewentualne opcje związane z wyświetlaniem itp.

2.4 Filtracja

Ze względu na to, że w pliku tekstowym zapisują się dane surowe odczytane wprost z akcelerometru, wymagana jest filtracja w celu wygładzenia danych, usunięcia niepotrzebnych szumów i ułatwienia wyciągnięcia przydatnych w analizie oddechu parametrów.

Rysunek 1 przedstawia surowy sygnały wydobyty wprost z akcelerometru(różne osie), który zawiera dwie główne składowe - sygnał pochodzący z bicia serca i sygnał poruszania się klatki piersiowej podczas oddechu.



Surowy sygnał zebrany z akcelerometrów

Oprócz tego, postanowiliśmy, że skoro nie wiemy na jakim sprzęcie zostaną wyświetlone dane w warunkach klinicznych, potrzebna jest aplikacja UI, pozwalająca na korzystanie z oprogramowania nawet w konsoli.

W trakcie projektowania urządzenia i tworzenia oprogramowania, przyjęliśmy założenie, że badania będą prowadzone w warunkach, kiedy pacjent się nie porusza (czyli leży, stoi milcząc, śpi). Jak i w ECG, poruszanie się pacjenta wprowadza zbyt duże szумы i mocno zakłóca główny sygnał.

Opracowana filtracja pozwala na wyciągnięcie takich parametrów jak liczba oddechów na minutę, wyświetlenie dokładnej godziny śmierci badanego człowieka, wyświetlanie graficzne przebiegów oddechowych. Oprócz tego umożliwia ona na wybranie przedziału czasu w którym chcemy przeprowadzić analizę.

Oprogramowanie zostało zaimplementowane w języku Python ze względu na kompatybilność z graficzną wersją naszej aplikacji.

Opis metody filtracji

Proces filtracji zaczyna się już na poziomie oprogramowania Arduino. Wprowadziliśmy licznik, który zlicza w górę tylko w przypadku, jeśli nie są obserwowane zmiany wartości w chociażby jednej z osi akcelerometru. Buzzer zostaje włączony, gdy licznik osiągnie wartość równą 950. Jeśli licznik zliczy do 1000 stwierdzamy zgon. Dane są nadal zapisywane, ale licznik przestaje zliczać i pozostaje na stałej wartości 1000.

W procesie analizy przebiegów każdego z wymiarów, doszliśmy do wniosku, że oś Z niezbyt nadaje się do analizy oddechowej. W przeciwieństwie do danych z osi Z, dane z osi Y i X nie były tak bardzo zakłócone biciem serca. Oś Z mierzyła przyspieszenia spowodowane akustyką zastawek serca lepiej niż inne osie.

Jednym z celów naszego projektu było znalezienie średniej liczby oddechów na minutę w danym przedziale czasowym. W tym celu wymyślona została metoda obliczania liczby maksimów lokalnych otrzymanego przebiegu, a później skalowania otrzymanej liczby na liczbę oddechów na minutę za pomocą zwykłej proporcji.

Surowy sygnał, jak widać jest bardzo zaszumiony. Utrudnia to znacznie prawidłowe wyznaczenie ekstremów. Konieczne jest wygładzenie przebiegu. Oprócz tego, Python pozwala na znalezienie maksimów lokalnych względem pewnego poziomu. Nie wiemy dokładnie jaki to jest poziom w każdym z przypadków, bo za każdym razem może on się różnić, więc otrzymane dane też powinny zostać znormalizowane.

W celu wygładzenia osi X i Y został użyty wbudowany w Python filtr Savitzky'ego-Golaya. W celu normalizacji została użyta metoda z-score. Warto zauważyć, że nawet po wygładzeniu przebiegu nadal są widoczne małe zakłócenia, ale jest to już bardzo blisko poszukiwanego maksimum lokalnego. Z tego powodu wbudowana metoda `find_peaks()` wyznacza więcej szczytów niż tego potrzebujemy, ale później ręcznie pozbawiamy się tych zbędnych we własnej metodzie `find_peaks_custom()`, jeśli piki znajdują się zbyt blisko siebie.

Zaimplementowana analiza wyświetla dokładną godzinę śmierci pacjenta, szukając wartości 1000 w liście z zapisanymi wartościami zliczeń licznika. Później na podstawie indeksu pierwszej znalezionej wartości równej 1000, szukany jest dokładny czas w którym to zliczenie wystąpiło.

Sprawdzenie Funkcjonalności - Testy

Implementacja analizy nie mogła odbyć się bez przeprowadzenia testów. Warto podkreślić, że przeprowadzone testy nie są idealne i czasami zawierają w sobie odcinki wynikające z próby odłączenia od sieci lub przestawienia na powierzchnię, aby urządzenie się nie poruszało i Arduino mogło wykryć zgon.

Lista przeprowadzonych testów:

- *test1* - pacjent w ciągu 5 minut leży w łóżku i normalnie oddycha.
- *test2* - pacjent w ciągu 2 minut leży na boku w łóżku i normalnie oddycha.
- *test3* - pacjent w ciągu minuty stoi i rozmawia.
- *test4* - pacjent w ciągu 2 minut oddycha, a później udaje, że umarł.

Zgodnie z wcześniejszymi przypuszczeniami, analiza nie nadaje się do analizy w przypadku wystąpienia zbyt dużych zakłóceń, które powstały przy przeprowadzeniu testu trzeciego. Najbardziej wiarygodne wyniki uzyskujemy w teście pierwszym, bo warunki jego przeprowadzenia były najlepsze.

Przedstawienie funkcjonalności aplikacji UI

Poniżej umieszczono zrzuty ekranu wraz z krótkimi opisami.

- Zaczynamy przygodę od możliwości wybrania jedną z dwóch opcji:

```
-----
IBREATHE v1.1
-----
Dear, User! Welcome to our UI application of a brand-new revolutionary device:

IBREATH

What do you want to do now?
Press 'y' to start analyses.
Press 'e' if you've just opened wrong application.
```

- Wybierając opcję e, opuszczamy naszą aplikację

```
-----
THANK YOU, SAYONARA!
-----
```

- Wybierając opcję y zaczynamy dialog:

```
-----  
                                START  
-----  
Here you can see list of files that contain data from device  
  
data/test1.txt  
data/test2.txt  
data/test3.txt  
data/test4.txt  
  
Please, enter name of file with extension:
```

Wyświetlona zostaje zawartość całego folderu – przeprowadzone wcześniej testy.

- Od razu sprawdzimy, czy wprowadzenie błędnej nazwy wyrzuci nas z programu:

```
Please, enter name of file with extension:  
test.txt  
This is not a valid file name. Do you want to try again?  
  
Press 'y' to enter a new name of file.  
Press 'e' if you want to have a break for a cup of tea and exit a program.
```

Widzimy że proszą nas o powtórne wprowadzenie danych.

- Teraz wprowadzono dobrą nazwę pliku. Wyświetlony został przedział czasu, w którym przeprowadzony był test, a dalej użytkownik jest proszony o wprowadzenie czasu startowego od którego chce wyświetlić analizę:

```
Please, enter name of file with extension:  
test1.txt  
Data was recorded in the time interval from 28.5.2022 at 18.26.5 to 28.5.2022 at 18.32.0  
  
Enter the hour from which you want to start analysis:  
18  
Enter minute from which you want to start analysis:  
26  
Enter seconds from which you want to start analysis:  
10  
  
START TIME: 18:26:10
```

- Dalej wprowadzamy godzinę zakończenia analizy:

```
Data was recorded in the time interval from 28.5.2022 at 18.26.5 to 28.5.2022 at 18.32.0  
  
Enter stop hour:
```

- Spróbujmy wprowadzić niepoprawne dane:

```
Enter stop hour:
dsf
Please input integer only...
```

Oraz przedział czasu, który nie mieści się w czasie przeprowadzenia badania:

```
Data you've provided is wrong. Check correct time interval. Please repeat again.

Data was recorded in the time interval from 28.5.2022 at 18.26.5 to 28.5.2022 at 18.32.0

Enter the hour from which you want to start analysis:
```

- Po wprowadzeniu poprawnych danych, wyświetla się cała analiza:

```
ANALYSIS WAS STARTED...
START TIME: 18:26:10
STOP TIME: 18:27:20

INFORMATION:

The patient is still alive.

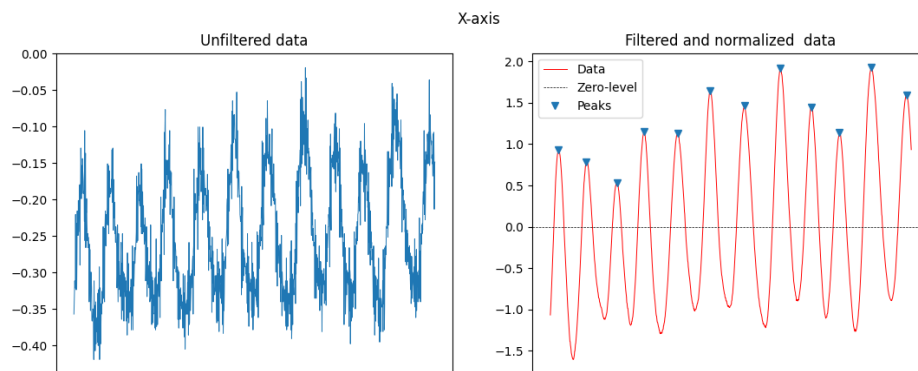
Czas, który zajmuje analiza: 70
Average breaths per minute from X axis: 11
Average breaths per minute from Y axis: 10
```

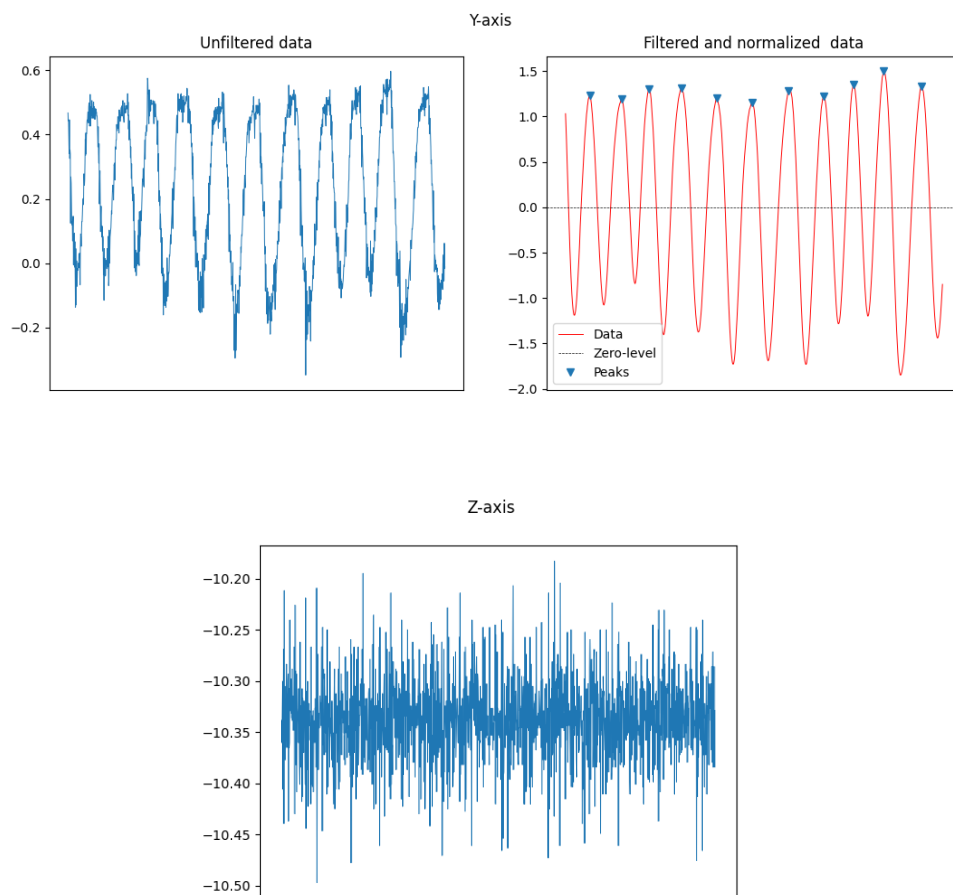
```
ANALYSIS WAS STARTED...
START TIME: 18:47:10
STOP TIME: 18:49:10

INFORMATION:

The patient died at 18.48.48

Czas, który zajmuje analiza: 120
Average breaths per minute from X axis: 8
Average breaths per minute from Y axis: 2
```





Dane przed i po filtracji dla każdej osi

Wnioski

Zaimplementowana analiza pokazuje najbardziej wiarygodne wyniki, kiedy badanie było zrobione w spokojnych warunkach a człowiek oddychał spokojnie.

2.5 Podsystem Bluetooth i Mechanizm Powiadamiania Opiekuna

Zgodnie z dokumentacją wstępną projektu, zaprojektowane urządzenie powinno informować opiekuna pacjenta o wystąpieniu braku oddechu u pacjenta, a w praktyce epizodu bezdechu sennego. Celem tej części projektu było zaprojektowanie systemu wykorzystującego moduł Bluetooth Low Energy. Moduł połączony jest z płytką Arduino Nano. Aby umożliwić komunikację ze smartfonem stworzona została aplikacja na system Android umożliwiającą przesyłanie powiadomień do wspomnianego opiekuna. Dodatkowo został zaprojektowany zestaw grafik wektorowych, które zostały użyte jako ikona i baner aplikacji.

Obsługa BLE HM-10 po stronie Arduino

Wysięk Arduino związany z całym podsystemem powiadamiania opiekuna jest minimalny – kod związany z BLE przesyła jedynie z pomocą wyjścia seryjnego pojedynczy znak („0”), gdy tylko pojawi się odpowiedni trigger – w poniższym kodzie jest to wpisanie „0” w Serial Monitorze, co miało na etapie projektowania aplikacji symulować sygnał informujący o wykryciu bezdechu pacjenta. Jest to celowy zabieg związany z koniecznością oszczędności zasobów układu.

Warto wspomnieć, że została tutaj wykorzystana biblioteka *SoftwareSerial*, która, jak sama nazwa mówi, pozwala na programowe emulowanie komunikacji seryjnej na innych cyfrowych pinach płytki Arduino. Użyty w projekcie moduł Bluetooth LE HM-10 domyślnie komunikuje się z układem poprzez interfejs szeregowy UART – wykorzystanie wspomnianej biblioteki pozwoliło uniknąć konieczności podłączenia modułu do portów szeregowych i umożliwiło podłączenie go do praktycznie dowolnego portu cyfrowego.

```
#include <SoftwareSerial.h>

SoftwareSerial BLE_serial(11,10); // (RX,TX) - zweryfikować przy podłączaniu
boolean isDead = false;

void setup()
{
  BLE_serial.begin(9600); // BLE
  Serial.begin(9600); // Serial Monitor - usunąć przy łączeniu kodu.
  delay(100);
}

void loop()
{
  while(!Serial.available()) {} // dopóki ktoś nie wklepie czegoś do konsoli

  // Trup czy nie trup? (Jak zwraca co innego niż 0 to ma się dobrze)
  // Przy łączeniu kodu zastąpić np. tym samym warunkiem co do buzzera
  if(Serial.read() == '0')
  {
    isDead = true;
  }

  // Wrzucamy do BLE komunikat zależnie od wartości zmiennej.
  if(isDead == true)
    BLE_serial.write("0"); // Aplikacja ma identyczną notację
                          // czyli "0" - trup, gdy przesłane coś innego, to jest ignorowane.
  isDead = false;
}
```

Kod Arduino

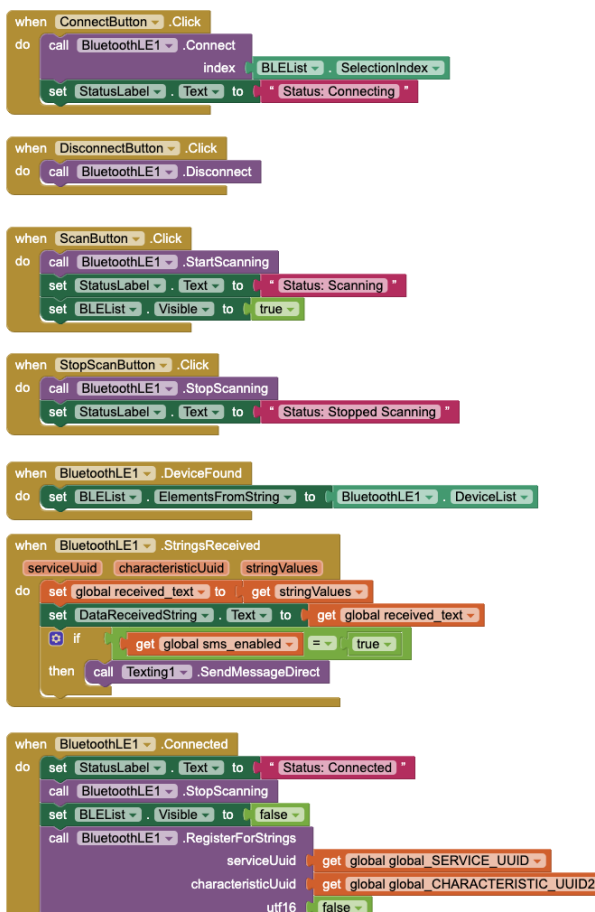
Aplikacja na smartfona

Sercem całego podsystemu powiadamiania jest aplikacja *iBreathe* na smartfona z systemem Android. Oprogramowanie to przeznaczone jest do kontroli systemu powiadomień oraz konfiguracji komunikacji telefonu pacjenta i urządzenia do analizy oddechu.

Aplikacja została zaprojektowana i zaimplementowana z pomocą kreatora *App Inventor* stworzonego przez *Massachusetts Institute of Technology*. Jest to przyjazne i intuicyjne, graficzne środowisko programistyczne (kod składa się z bloków), które pozwala na znacznie szybsze projektowanie aplikacji niż „tradycyjne” języki programowania. Z uwagi na popularność serwisu, powstaje wiele natywnych lub tworzonych przez użytkowników rozszerzeń pozwalających na tworzenie zaawansowanych aplikacji – na przykład wykorzystane w naszej aplikacji rozszerzenie BluetoothLE.

Aplikacja umożliwia:

- Włączenie/ Wyłączenie powiadamiania opiekuna przez SMS
- Ustawienie numeru telefonu opiekuna, na który będą wysyłane powiadomienia
- Obsługę połączeń BLE wraz z kontrolą przepływu danych
- Ustalenie hasła blokującego możliwość manipulowania połączeniem BLE



Przykładowy fragment kodu – obsługa BLE



Zrzut ekranu z aplikacji iBreathe

Instrukcja Obsługi:

1. Przejdź do zakładki Connection Manager
2. Naciśnij przycisk *Scan*, aby wyszukać dostępne urządzenia BLE.
3. Wybierz swoje urządzenie medyczne z listy poniżej (w naszym przypadku będzie to prawdopodobnie „HM10 (...)\”. Opcjonalnie zatrzymaj skanowanie przyciskiem *Stop Scan*.
4. Naciśnij przycisk *Connect*. O udanym połączeniu się z urządzeniem zostaniesz poinformowany w polu *Status*.
5. Opcjonalnie ustal hasło, które zablokuje możliwość manipulowania połączeniem BLE. Aby to zrobić wpisz wybrane przez siebie hasło w polu tekstowym i naciśnij przycisk *Set Password*. Aby odblokować możliwość manipulacji połączeniem BLE wpisz ustalone wcześniej hasło i naciśnij przycisk *Unlock Changes*.

Wnioski

Warto zauważyć, że podobnie jak w innych częściach projektu urządzenia, w podsystemie powiadamiania jest dużo miejsca na ulepszenia.

Po pierwsze, z uwagi na fakt wykorzystania *MIT App Inventor* nie udało się uzyskać działania aplikacji w tle/ przy zablokowanym ekranie – jest to główne ograniczenie kreatora. Istnieją rozszerzenia stworzone przez społeczność, pozwalające na jego ominięcie, jednak te, które udało się znaleźć były dość słabo udokumentowane. To znacząco ogranicza możliwość zastosowania urządzenia w warunkach rzeczywistych – pacjent musiałby pozostawiać swój telefon na całą noc z włączonym ekranem i aplikacją.

Kolejnym możliwym udoskonaleniem aplikacji byłaby możliwość zapamiętywania sparowanych urządzeń medycznych po zamknięciu aplikacji *iBreathe* w pamięci ROM. Dzięki takiemu rozwiązaniu możliwe byłoby automatyczne łączenie z urządzeniem przy ponownym otwarciu aplikacji, co wyeliminowałoby ewentualne problemy po stronie pacjenta (np. w przypadku crashu aplikacji, rozładowania telefonu itp.).

We wstępnych rozważaniach dotyczących projektu zespół brał także pod uwagę możliwość transmisji danych pomiarowych przez BLE. Dane miałyby być przekazywane z pomocą odpowiedniego protokołu na serwer i tam przechowywane. Zrezygnowaliśmy jednak z tego rozwiązania z uwagi na ograniczony czas implementacji systemu. Niemniej jednak, byłaby to ciekawa funkcjonalność.