AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI
INSTYTUT ELEKTRONIKI

Metodyki projektowania i modelowania systemów 1

Projekt krokomierza

Autor: **Michał Miś**
Kierunek studiów: Elektronika i Telekomunikacja

Kraków, 2023

Spis treści

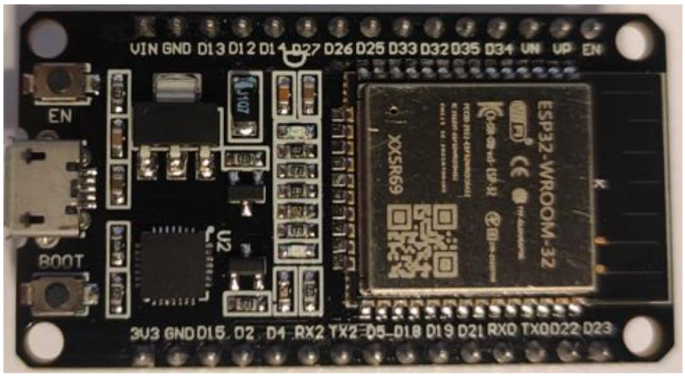
1. Opis projektu

Celem projektu było wykonanie urządzenia zliczającego ilość wykonanych kroków. Działanie urządzenia zostało oparte na mikrokontrolerze ESP32. Aby lepiej wykorzystać możliwości wybranego układu w aplikacji zostały również zaimplementowane dodatkowe funkcjonalności, takie jak pomiar pulsu, pomiar saturacji krwi czy nawet zbliżeniowy pomiar temperatury ciała. Dzięki wszystkim dodatkowym funkcjom urządzenie może spełniać rolę nie tylko krokomierza, lecz również opaski typu SmartBand. Kod źródłowy wykonanego projektu, wraz z wszystkimi potrzebnym bibliotekami znajduje się w repozytorium na platformie GitHub. Link do repozytorium: https://github.com/MichalMis/MPiMS_Project. Aby uruchomić projekt konieczne jest zainstalowanie oprogramowania ArduinoIDE. Wykorzystywana w projekcie wersja oprogramowania to Arduino IDE 2.2.1. Po zainstalowaniu oprogramowania załączone w repozytorium pliki bibliotek powinny zostać umieszczone w folderze: C:\Users\USER\Documents\Arduino\libraries.

2. Wykorzystany sprzęt

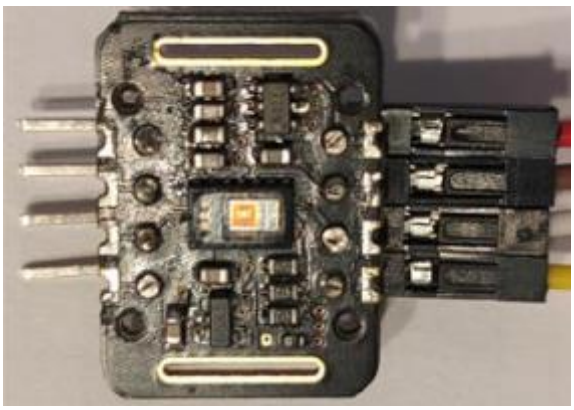
Do wykonania projektu zostały wykorzystane następujące elementy:

1. Mikrokontroler ESP32:



Rysunek 1. Mikrokontroler ESP32. Opracowanie własne.

2. Czujnik pulsu i saturacji krwi MAX30102:



Rysunek 2.Czujnik pulsu i saturacji krwi MAX30102. Opracowanie własne.

3. Pirometr MLX90614:

Rysunek 3. Pirometr MLX90614. Opracowanie własne.

4. Akcelerometr ADXL345

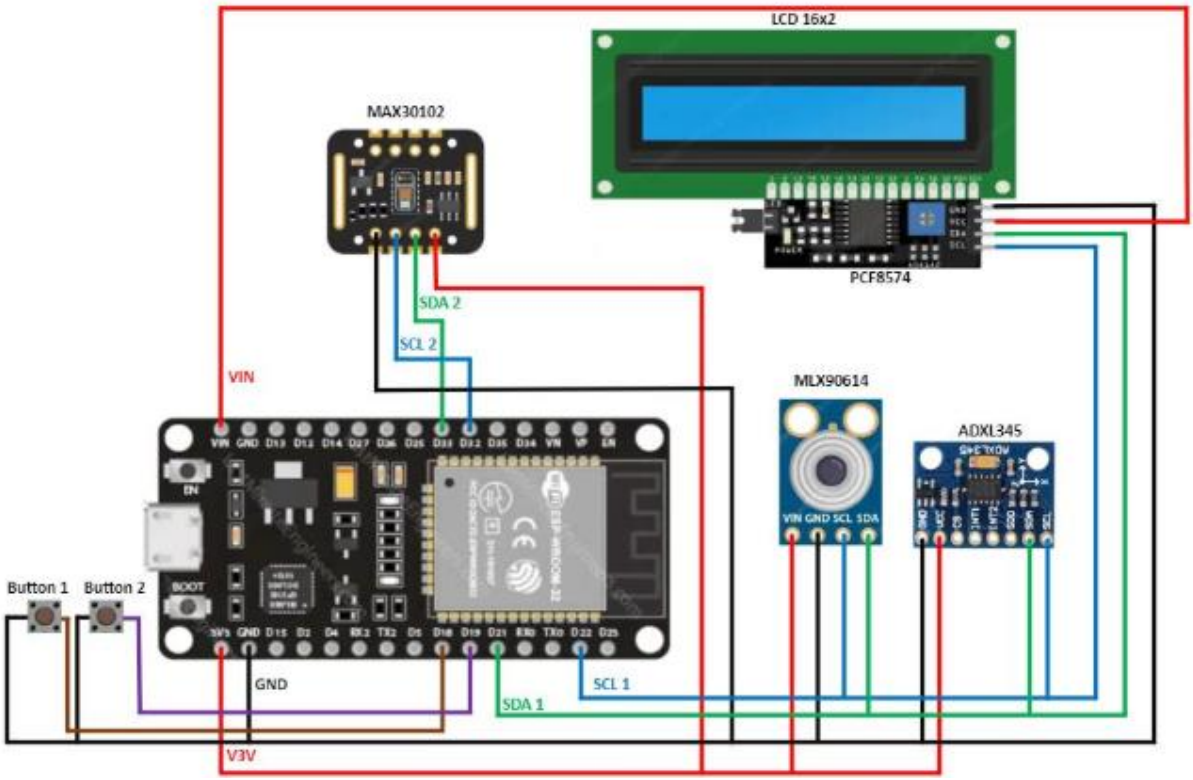
Rysunek 4. AkcelerometrADXL345. Opracowanie własne.

5. Wyświetlacz LCD16x2

Rysunek 5. Wyświetlacz LCD16x2. Opracowanie własne.

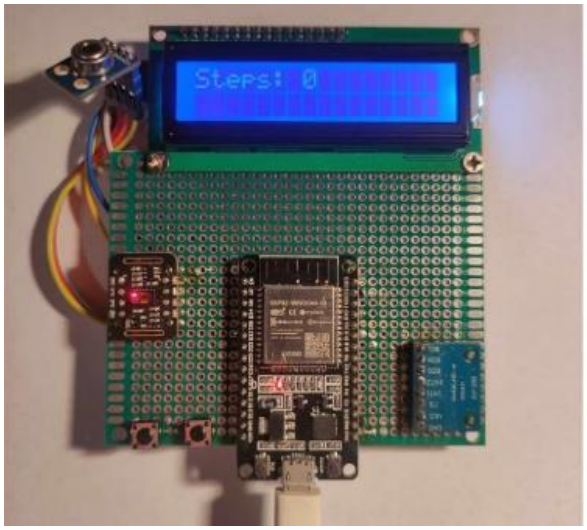
6. 2 x Przyciski

Wszystkie wyżej wymienione elementy powinny zostać połączone zgodnie ze schematem blokowym zamieszczonym poniżej, aby zapewnić prawidłowe działanie układu:



Rysunek 6. Schemat połączeniowy układu. Opracowanie własne.

Zdjęcie przykładowego urządzenia z prawidłowo podłączonymi elementami zostało przedstawione poniżej:

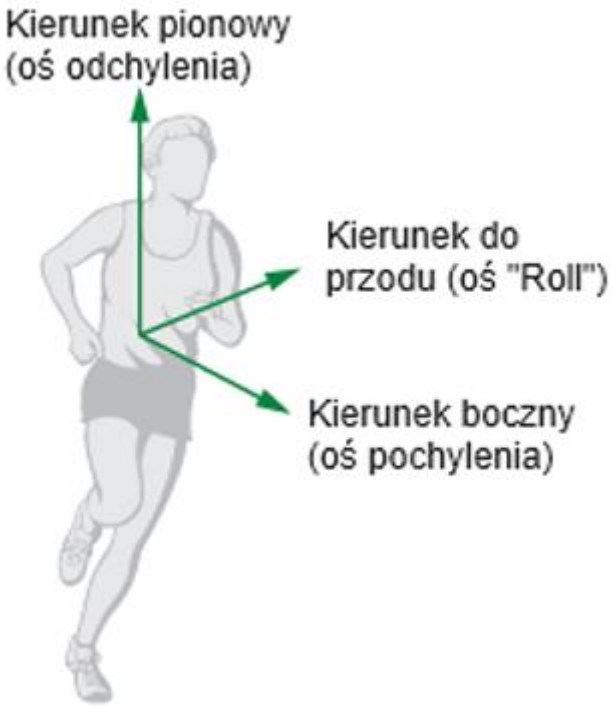


Rysunek 7. Zdjęcie gotowego układu. Opracowanie własne.

3. Opis działania kodu

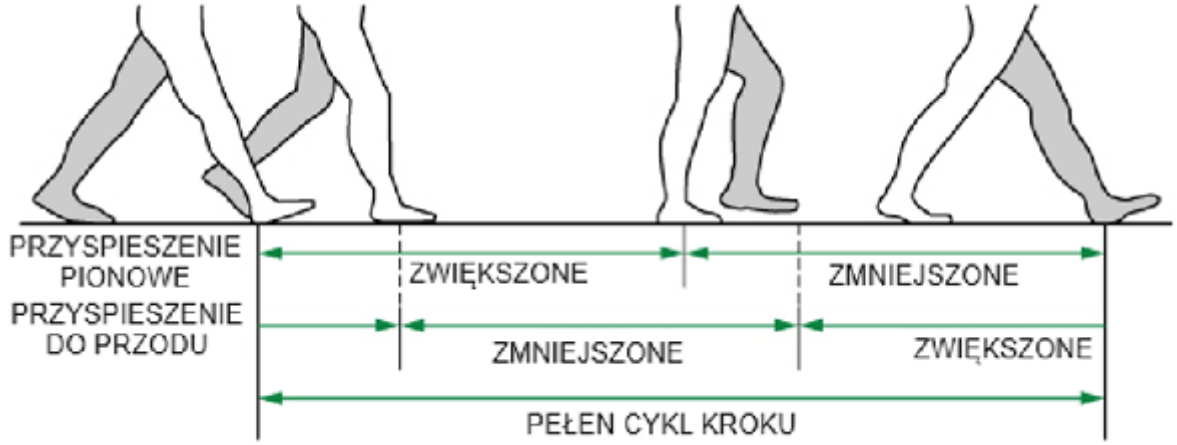
3.1 Opis funkcji void StepCounter()

Opisywana funkcja ma za zadanie rozpocząć komunikację z akcelerometrem ADXL345 za pomocą magistrali I2C, następnie konwertując dane we właściwy sposób, funkcja powinna inkrementować ilość wykonanych kroków przy przekroczeniu właściwego progu, który ustawiany jest w sposób dynamiczny na podstawie poprzednio zebranych próbek. Akcelerometr zbiera dane z poszczególnych trzech osi ruchu odchylenia, „Roll" a więc obrotu do przodu oraz pochylenia. Wszystkie one zostały przedstawione na rysunku poniżej:



Rysunek 8. Rysunek przedstawiający osie wychylenia podczas chodu. Źródło: Opracowanie własne na podstawie: https://www.analog.com/en/resources/analog-dialogue/articles/pedometer-design-3-axis-digital-acceler.html.

Przyglądając się dokładnie poszczególnym fazom chodu, zaobserwować można, jak zmienia się przyspieszenie w poszczególnych osiach ruchu, co przedstawione jest na rysunku 10. Wywnioskować jednocześnie można, że podczas ruchu co najmniej jedna z osi osiąga większe przyspieszenie:



Rysunek 9 Rysunek przedstawiający przyspieszenie podczas poszczególnych faz chodu. Źródło: https://www.analog.com/en/resources/analog-dialogue/articles/pedometer-design-3-axis-digital-acceler.html.

Korzystając z tej własności możliwe jest wykonanie odpowiedniego algorytmu liczącego wykonane kroki.

Kod odpowiedzialny za inicjalizację pliku :

```
   Wire.beginTransmission(ADXL345);
   Wire.write(0x2D);
```

Funkcja `Wire.beginTransmission()`oraz `Wire.write()` zaimplementowane są jako metody klasy TwoWire pochodzącej z pliku nagłówkowego Wire.h. Do funkcji beginTranssmision podawana jest zmienna globalna ADXL345, która jest adresem wykorzystanego akcelerometru w postaci kodu hexadecymalnego.

Następnie funkcja inicjalizuje pętle służącą do sczytywania danych z kolejnych rejestrów:

```
for (int i = 0; i < 100; i++) {
    Wire.beginTransmission(ADXL345);
    Wire.write(0x32);
    Wire.endTransmission(false);
    Wire.requestFrom((uint8_t)ADXL345, (size_t)6, (bool)true);
    X_out[i] = (int16_t)(Wire.read() | (Wire.read() << 8));
    X_out[i] = X_out[i] / 256.0;
    Y_out[i] = (int16_t)(Wire.read() | (Wire.read() << 8));
    Y_out[i] = Y_out[i] / 256.0;
    Z_out[i] = (int16_t)(Wire.read() | (Wire.read() << 8));
    Z_out[i] = Z_out[i] / 256.0;
```

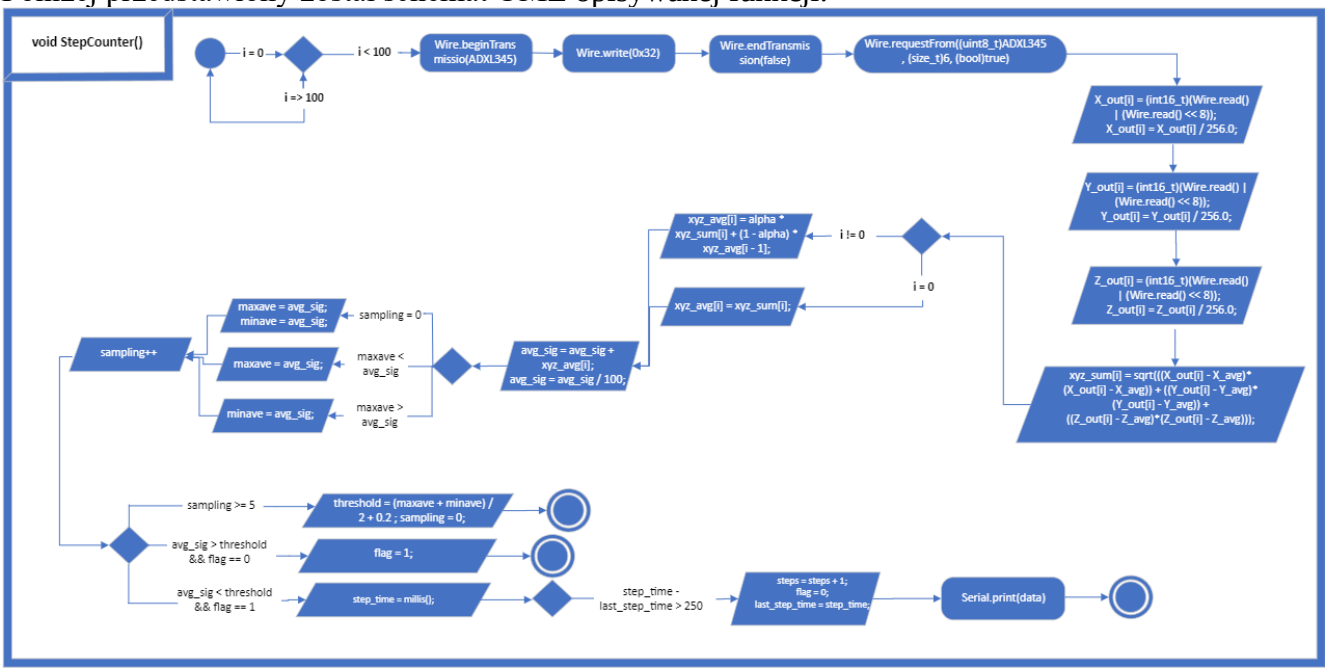
Dane z każdej osi zajmują po dwa rejestry, dlatego odczyt wykonywany jest 6-krotnie za pomocą funkcji Wire.read(). Funkcja Wire.requestFrom() służy do inicjowania żądania odczytu danych od urządzenia podrzędnego (slave) w komunikacji I2C. Jest to funkcja używana przez urządzenie nadrzędne (master), aby poprosić o dane od urządzenia podrzędnego. Zgodnie z dokumentacją wykorzystywanego czujnika, dane zapisywane są w 16 bitowym buforze. Następnie dane dla każdej osi konwertowane są odpowiednio skalowane i przeliczane na wartość typu float .

```
xyz_sum[i] = sqrt(((X_out[i] - X_avg) * (X_out[i] - X_avg)) + ((Y_out[i] -
Y_avg) * (Y_out[i] - Y_avg)) + ((Z_out[i] - Z_avg) * (Z_out[i] - Z_avg)));
if (i == 0) {
    xyz_avg[i] = xyz_sum[i];
}
else {
     xyz_avg[i] = alpha * xyz_sum[i] + (1 - alpha) * xyz_avg[i - 1];
     avg_sig = avg_sig + xyz_avg[i];
}
avg_sig = avg_sig / 100;
```

Jedną z metod na osiągnięcie odpowiedniego przebiegu wyjściowego (Przebieg pokazujący zmiany przyspieszenia we wszystkich osiach), jest obliczenie średniej kwadratowej różnic wartości zmiennej i wartości oczekiwanej, gdzie wartość zmienna to aktualnie zebrana dana, a wartość oczekiwana, to zebrana na początku średnia ze stu pierwszych zebranych danych pomiarowych. Następnie na zebranej wartości średniej możliwe jest zastosowanie filtru, który dodatkowo wyeliminuje wpływy szumów i drgań szukanego przebiegu. Po wykonaniu tych operacji program wyznacza średnią z zebranych stu próbek danych, co realizowane jest przez powyższy fragment kodu.

```
if (sampling == 0)
{
    maxave = avg_sig;
    minave = avg_sig;
}
else if (maxave < avg_sig)
{
    maxave = avg_sig;
}
else if (minave > avg_sig)
{
    minave = avg_sig;
}

sampling++;

if (sampling >= 5)
{
    threshold = (maxave + minave) / 2 + 0.2;
    sampling = 0;
}
```

Następnym krokiem jest wskazanie w programie momentu, w którym powinien on zliczyć kolejny krok. Zgodnie z artykułem „Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer", którego autorem jest Neil Zhao, krok jest zdefiniowany jako zdarzenie, podczas którego zbocze opadające przebiegu oznaczającego przyspieszenie, przecina przebieg wyznaczony przez kolejne wartości dynamicznego progu. Program ma więc za zadanie dodawać do licznika kolejne kroki w punktach odpowiadających powyższemu opisowi. Warto podkreślić, że metoda ta ma swoje ograniczenia i nie jest odporna na ruchy, które wywołują podobny do wykonania kroku przebieg przyspieszenia.
Poniżej przedstawiony został schemat UML opisywanej funkcji:



3.2 Opis funkcji void HeartBeatRate(long IR_Value)

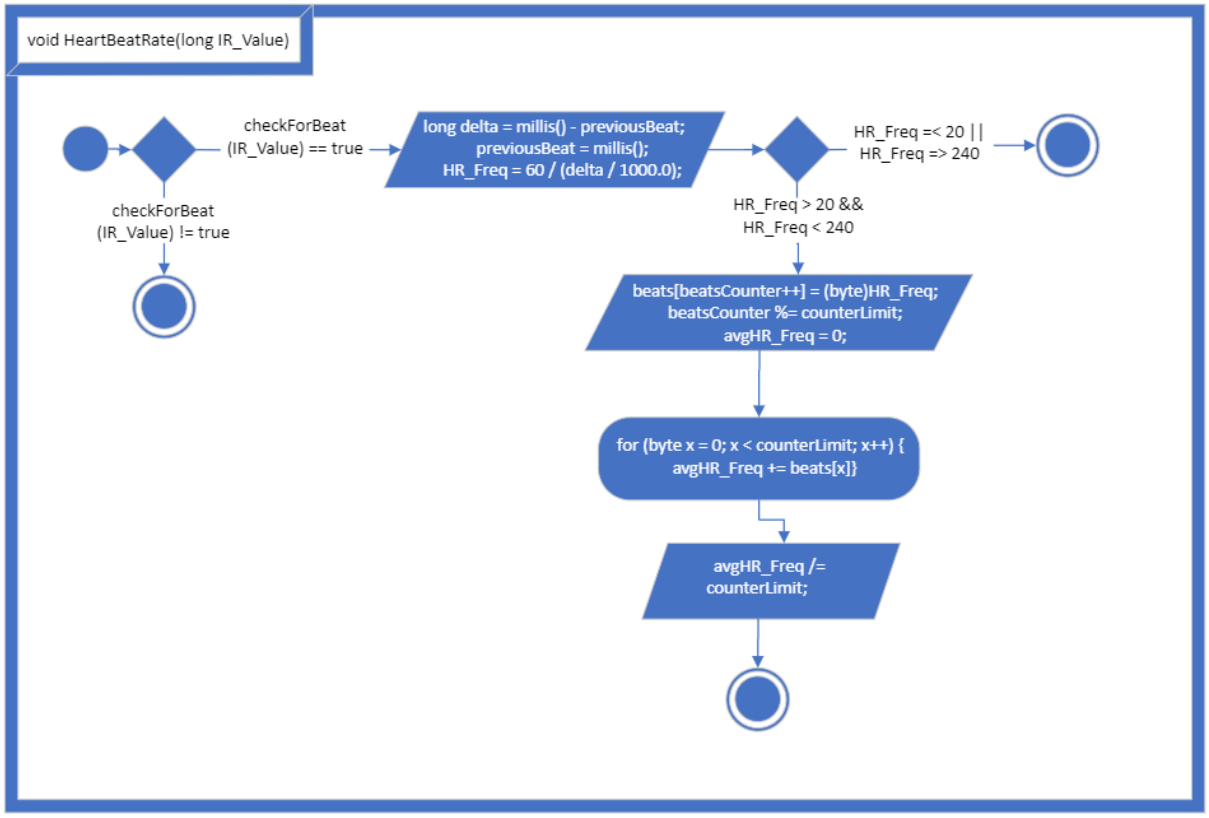
W celu wykonania prawidłowego pomiaru pulsu, konieczne jest wyemitowanie światła podczerwonego IR z zamontowanej na czujniku diody. Światło emitowane przez wymienioną diodę charakteryzuje się długością fali równą około 890nm. Kolejno korzystając ze zdolności krwinek czerwonych do absorbcji światła i używając fotodetektora, możliwe jest monitorowanie ilość odbitego światła. Podczas gdy ludzkie serce bije, przez tkanki przepływa większa ilość krwinek czerwonych, tym samym więcej światła jest przez nie pochłaniane, a do fotodetektora trafia mniejsza ilość odbitego światła. Fotodetektor wysyła do przetwornika ADC dane mówiące o ilości odbitego światła podczerwonego. Kolejno po przekształceniu danych z postaci analogowej na cyfrową możliwe jest określenie momentu, w którym wystąpiło bicie serca. Monitorując częstotliwość zmian odczytanej przez fotodetektor wartości światła podczerwonego, możliwe jest określenie częstotliwości bicia serca. W celu wykonania takiego pomiaru przy pomocy czujnika MAX30102, należy skorzystać z odpowiedniej funkcji biblioteki heartRate.h, dostarczonej przez producenta „SparkFun", a więc:

```
bool checkForBeat(int32_t sample);
```

Powyższa funkcja zwraca wartość logiczną TRUE w momencie, w którym wykryty został moment bicia serca, a więc wystąpiła odpowiednia różnica pomiędzy kolejnym odczytanymi wartościami odbitego światła podczerwonego. Następnie poprzez odpowiednie obliczenia możliwe jest uzyskanie uśrednionej wartość pulsu. Kod służący do obliczenia wartości tętna umieszczony został poniżej:

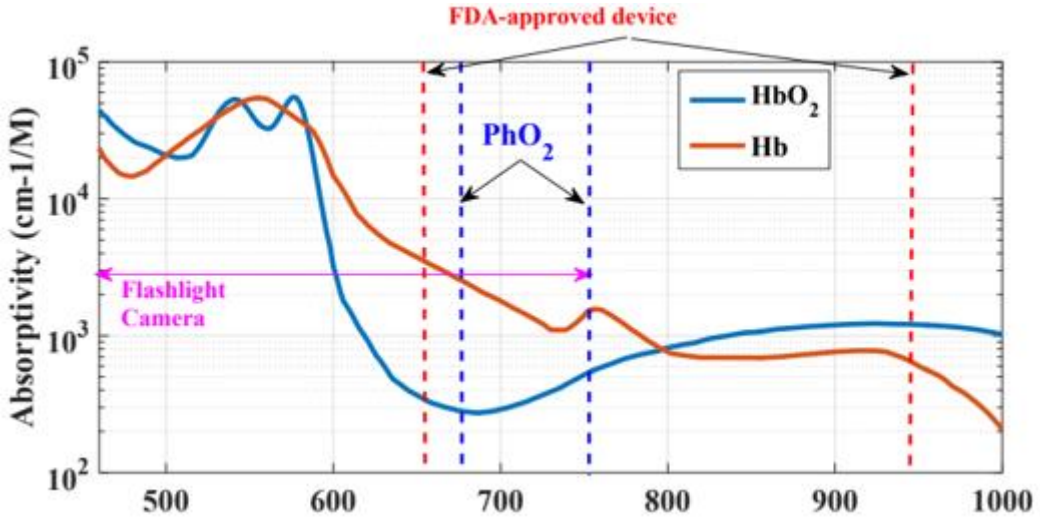
```
void HeartBeatRate(long IR_Value) {
    if (checkForBeat(IR_Value) == true) {
          long delta = millis() - previousBeat;
          previousBeat = millis();

          HR_Freq = 60 / (delta / 1000.0);

          if (HR_Freq > 20 && HR_Freq < 240) {
                beats[beatsCounter++] = (byte)HR_Freq;
                      beatsCounter %= counterLimit;
                avgHR_Freq = 0;
                for (byte x = 0; x < counterLimit; x++) {
                      avgHR_Freq += beats[x];
                }
                avgHR_Freq /= counterLimit;
          }
    }
}
```

Poniżej przedstawiony został schemat UML opisywanej funkcji:



3.3 Opis funkcji void SpO2_Calc(long IR_Value)

Po prawidłowym oszacowaniu wartości pulsu, kolejną wartością badaną przez układ MAX30102 jest saturacja krwi. Zasada działania pomiaru opiera się, podobnie jak w przypadku pulsu, na absorbcji światła przez krwinki czerwone. Tym razem jednak przepływającą krew podzielić można na odtlenioną i utlenioną. Ta różnica ujawnia się właśnie w różnicy absorbcji światła o różnych długościach fali. W komercyjnych pulsoksymetrach, do obliczenia współczynnika absorpcji wybiera się długości fal odpowiadające barwie czerwonej i podczerwieni. Wykres efektu absorbcji światła przez krew utlenioną i odtlenioną, dla różnych długości fal, przedstawiony w artykule „PhO2: Smartphone based Blood Oxygen Level Measurement Systems using Near-IR and RED Wave-guided Light” został zamieszczony poniżej:



Rysunek 10. Wykres absorbcji światła o różnych długościach fali przez krew utlenioną i odtleniona. Źródło:https://www.researchgate.net/publication/323786965_PhO2_Smartphone_based_Blood_Oxygen_Level_Measurement_Systems_using_Near-IR_and_RED_Wave-guided_Light.

Korzystając z wykresu przedstawionego na rysunku 3, można zaobserwować, że w zakresie długości fal o świetle czerwonym, krew utleniona absorbuje znacząco mniej światła niż krew odtleniona. Sytuacja ta kształtuję się odwrotnie w zakresie fal podczerwonych, gdzie to właśnie krew odtleniona gorzej absorbuje światło. Na podstawie stosunku wartości światła czerwonego oraz podczerwonego, odebranych przez fotodetektor oblicza się procent saturacji krwi. Dokładna metoda obliczania saturacji krwi została odzwierciedlona w kodzie programu poniżej:

```
if (Sensor_max.available()) {
    k++;
    IR_SpO2 = Sensor_max.getFIFOIR();
    Red_SpO2 = Sensor_max.getFIFORed();
    double_IR_SpO2 = (double)IR_SpO2;
    double_Red_SpO2 = (double)Red_SpO2;
```

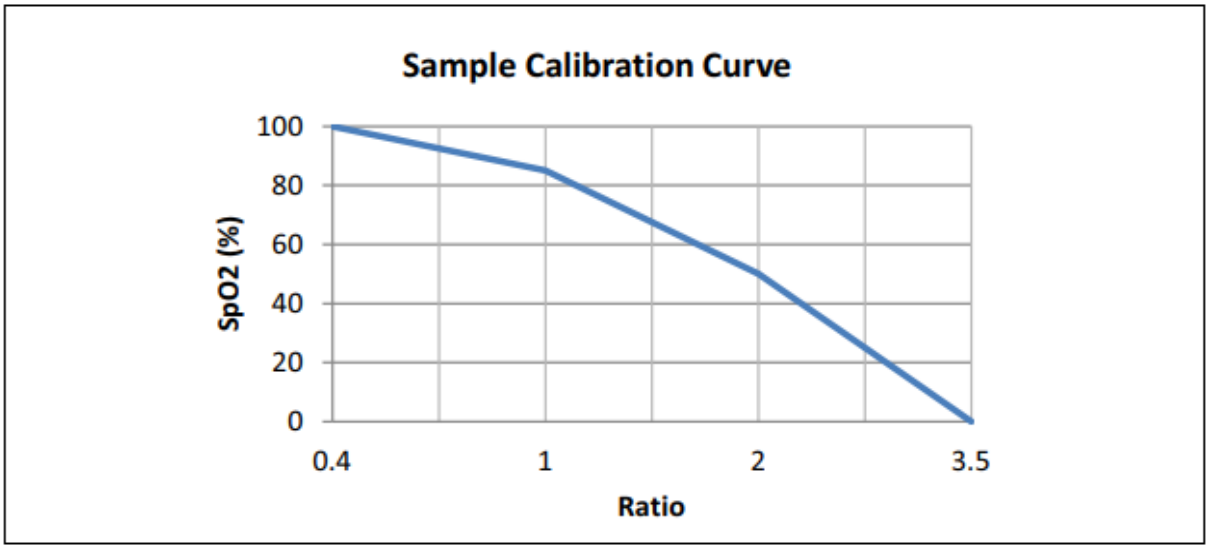
W powyższej części kodu wykorzystane są funkcję z biblioteki MAX30105.h. Gdy dostępne są dane z czujnika, co sprawdzane jest w pierwszej instrukcji warunkowej kodu, pobierane są informacje z czujnika o ostatnio odczytanej wartości odebranego przez fotodetektor światła IR oraz czerwonego (funkcje z biblioteki MAX30105.h). Kolejno dokonywane jest rzutowania powyższych danych do typu double ze względu na późniejsze ich użycie w funkcji:

```
avgIR_SpO2 = avgIR_SpO2 * filter_rate + (double)IR_SpO2 * (1.0 - filter_rate);
avgRed_SpO2 = avgRed_SpO2 * filter_rate + (double)Red_SpO2 * (1.0 - filter_rate);
IR_RMS += (double_IR_SpO2 - avgIR_SpO2) * (double_IR_SpO2 - avgIR_SpO2);
RED_RMS += (double_Red_SpO2 - avgRed_SpO2) * (double_Red_SpO2 - avgRed_SpO2);
```

Kolejnym krokiem jest odpowiednie obliczenie wartości średniej zebranych wartości światła IR oraz czerwonego, oraz następnie jej odfiltrowanie, w taki sposób, aby otrzymać wartość DC mierzonego sygnału. Następnie z danych o wartości światła IR oraz czerwonego obliczona zostaje ich wartość RMS, a więc wartość skuteczna badanego sygnału. Posiadając powyższe dane, możliwe staje się skorzystanie ze wzoru (1), na obliczenie parametru „Ratio”, zamieszczonego w nocie katalogowej producenta [11]. Opisywany wzór został umieszczony poniżej:

$$Ratio = \frac{Red\_AC\_Vrms \,/\, Red\_DC}{IR\_AC\_Vrms \,/\, IR\_DC}$$

Wyliczony parametr pozwala na późniejsze obliczenie odpowiedniej wartości saturacji krwi, do czego używana jest również przykładowa krzywa kalibracji dołączona do dokumentacji przez producenta i zamieszczona poniżej.



Rysunek 11. Przykładowa krzywa kalibracji dołączona do dokumentacji używanego czujnika. Źródło: www.microchip.com

Obliczenie prawidłowej wartości saturacji krwi zostało zrealizowane za pomocą poniższego kodu:

```
if ((k % Num) == 0) {
     double  Ratio  =  (sqrt(IR_RMS)  /  avgIR_SpO2)  /  (sqrt(RED_RMS)  /
     avgRed_SpO2);
    SpO2 = -23.3 * (Ratio - 0.4) + 100;
     Filtered_SpO2 = Alpha_SpO2 * Filtered_SpO2 + (1.0 - Alpha_SpO2) * SpO2;
```

Dzięki krzywej kalibracji umieszczonej powyżej, możliwe jest oszacowanie parametrów prostej umieszczonej na wykresie. Parametry prostej pozwolą zaś na uzależnienie szukanej wartości saturacji krwi od wartości parametru Ratio. Ponieważ sytuacja, w której saturacja krwi wynosi mniej niż 86%, jest sytuacją skrajnie niebezpieczną dla człowieka, a jednocześnie mało realną do osiągniecia podczas korzystania z czujnika, do obliczenia parametrów krzywej został przyjęty tylko fragment krzywej kalibracyjnej z zakresu Ratio większego od 0.4 i mniejszego od 1. Korzystając z prostych wzorów matematycznych, a więc:

$$\begin{cases} 0.4a * x + b = 100 \\ ax + b = 86 \end{cases}$$

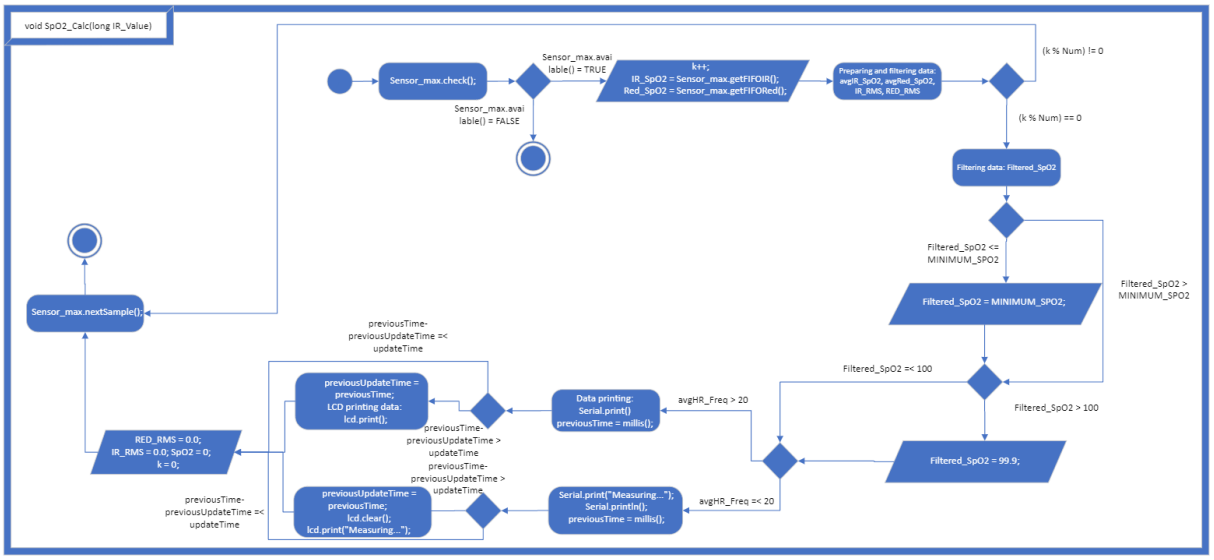
Obliczony zostaje parametr a=-23,3, następnie przesuwając krzywą względem osi X o wartość równą 0,4 otrzymane zostaje równanie w postaci:

$$SpO2 = -23{,}3 * (Ratio - 0{,}4) + 100$$

Za pomocą tak zapisanego równania, w wyżej zamieszczonym kodzie zostaje obliczona wartość saturacji krwi, która następnie zostaje odfiltrowana i przekazana jako zmienna do kolejnych funkcji programu. Tak obsłużony czujnik daje możliwość osiągnięcia wiarygodnych wartości zarówno pulsu jak i saturacji krwi. Trzeba jednak pamiętać o tym, że przez

ograniczania i brak odpowiedniej certyfikacji, czujnik ten nie może być wykorzystywany w celach medycznych i nadaje się tylko do użytku komercyjnego.

Poniżej przedstawiony został schemat UML opisywanej funkcji:

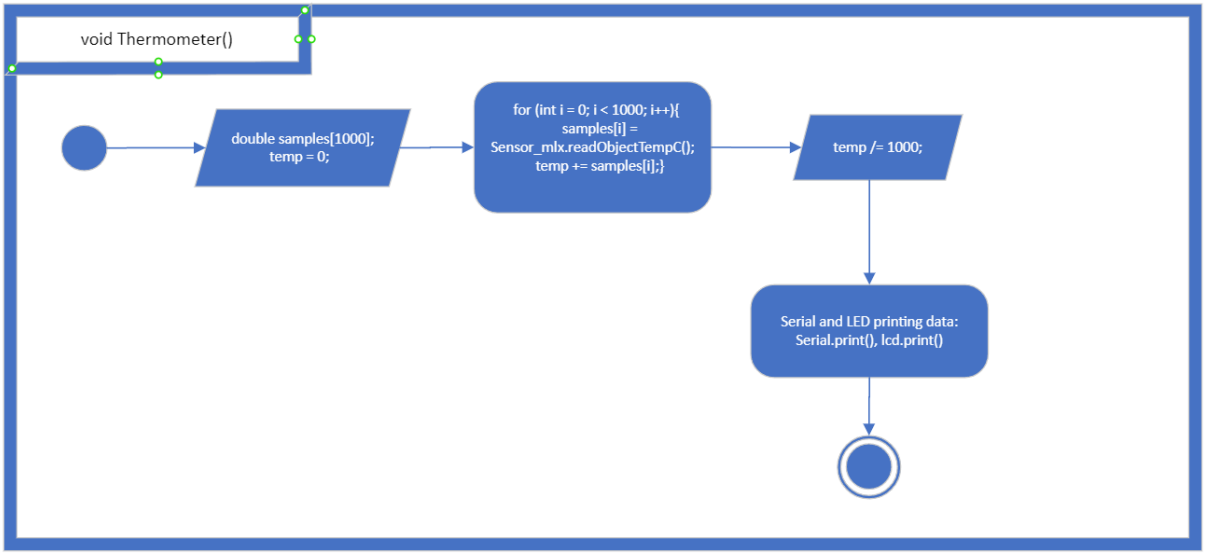


3.4 Opis funkcji void Thermometer()

Układ pirometru składa się z trzech głównych elementów. Jednym z nich jest soczewka, pozostałymi zaś są detektor światła podczerwonego, procesor DSP oraz 17-bitowy przetwornik ADC. Światło wyemitowane przez badany obiekt trafia przez specjalną soczewkę do detektora światła podczerwonego. Następnie dane przekazane przez detektor zostają odebrane przez 17-bitowy przetwornik analogowo-cyfrowy. Te z kolei już w formie cyfrowej trafiają do procesora DSP, gdzie obliczona zostaje temperatura, o której informacje są kolejno przekazywane, za pomocą magistrali I2C, do mikrokontrolera. Dzięki zastosowaniu tych elementów oraz dostarczonej przez producenta biblioteki, kod stworzonego do obsługi czujnika programu upraszcza się do kilku przedstawionych poniżej linii:

```
void Thermometer() {
    double samples[1000];
    temp = 0;
    for (int i = 0; i < 1000; i++) {
        samples[i] = Sensor_mlx.readObjectTempC();
        temp += samples[i];
    }
      temp /= 1000;
```

Poniżej przedstawiony został schemat UML opisywanej funkcji:



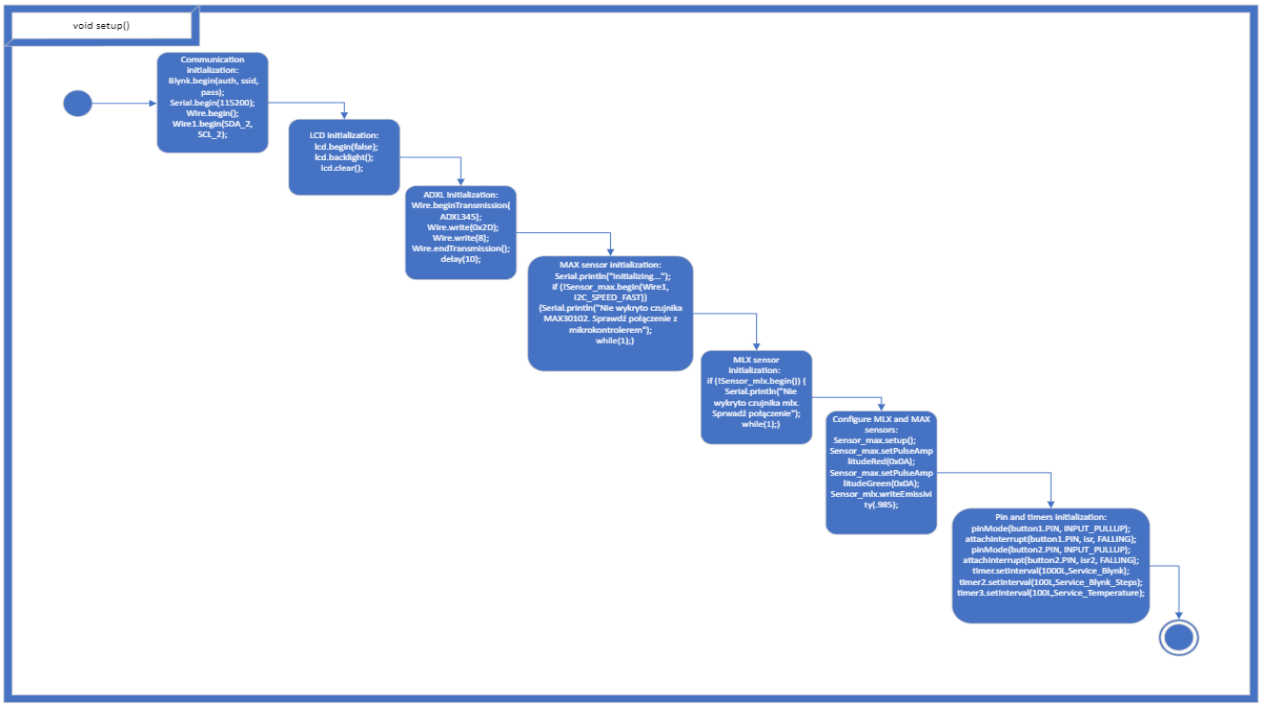
3.5 Opis funkcji void setup()

Podobnie do innych programów pisanych z myślą o mikrokontrolerach, kod do opisywanego projektu posiada również dwie główne funkcje, a więc: void Setup(), która odpowiedzialna jest za inicjalizację:

- Wszystkich czujników
- Transmisji po I2C
- Transmisji szeregowej
- Przerwań
- Timerów
- Funkcji wykorzystywanych do użycia aplikacji Blynk

Funkcja Setup jest wykonywana w programie tylko raz, w przeciwieństwie do funkcji void loop(), która, jak wskazuje nazwa, wykonywana jest nieustannie w pętli.

Poniżej przedstawiony został schemat UML opisywanej funkcji:

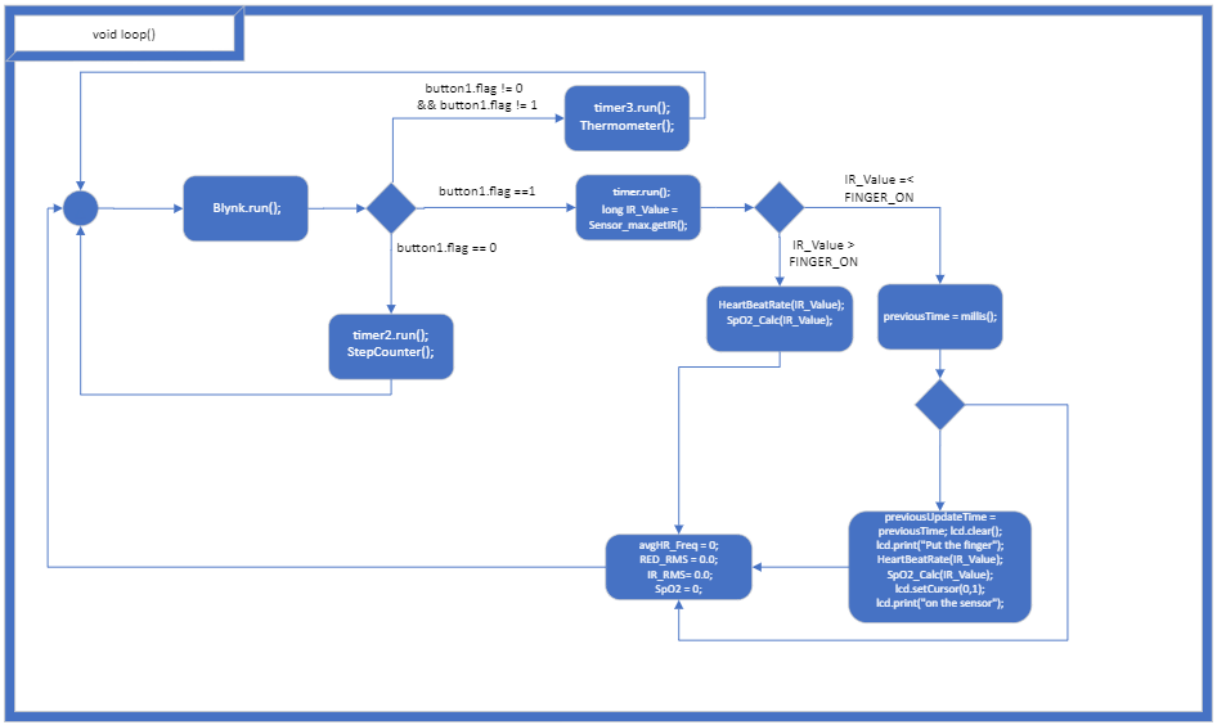


3.6 Opis funkcji void loop()

Funkcja void loop() to główna funkcja programu, która jest wykonywana nieustannie w pętli. Jedynie, gdy wywołane zostanie przerwanie, funkcja loop zostaje zatrzymana na czas wykonania instrukcji przerwania. Przerwania to dobre rozwiązania w programie, w którym ma zostać zrealizowana krótka czynność, będąca odpowiedzią na wystąpienie określonego czynnik zewnętrznego. Trzeba jednak pamiętać, o tym, że jeśli funkcja przerwania będzie wykonywana zbyt długo, to zadziała kolejny mechanizm nazwany „watchdog-iem", który monitoruje, czy pętla główna programu jest stale wykonywana. Jeśli funkcja przerwania, zatrzymuje główną funkcję programu na zbyt długo, to watchdog zresetuje układ myśląc, że ten się zawiesił. W projekcie przerwania zostały użyte w celu obsługi przycisków. Jeden z przycisków wykorzystywany w projekcie jest odpowiedzialny za ustawianie flagi, która z kolei wskazuje na tryb pracy mikrokontrolera, drugi przycisk służy zaś do zerowania ilości wykonanych kroków. Odpowiednio przygotowane funkcje do obsługi poszczególnych czujników, są natomiast wywoływane we wcześniej opisywanej funkcji loop. Kod głównej funkcji programu zamieszczony został poniżej:

```
void loop() {
    Blynk.run();
    if (button1.flag == 0) {
        timer2.run();
        StepCounter();
    }
    else if (button1.flag == 1) {
        timer.run();
        long IR_Value = Sensor_max.getIR();
        if (IR_Value > FINGER_ON) {
            HeartBeatRate(IR_Value);
            SpO2_Calc(IR_Value);
        }
        else {
            previousTime = millis();
            if (previousTime - previousUpdateTime > updateTime) {
                previousUpdateTime = previousTime;
                lcd.clear();
                lcd.print("Put the finger");
                lcd.setCursor(0, 1);
                lcd.print("on the sensor");
            }
            avgHR_Freq = 0;
            RED_RMS = 0.0; IR_RMS = 0.0; SpO2 = 0;
        }
    }
    else {
        timer3.run();
        Thermometer();
    }
  }
```

Poniżej przedstawiony został schemat UML opisywanej funkcji:

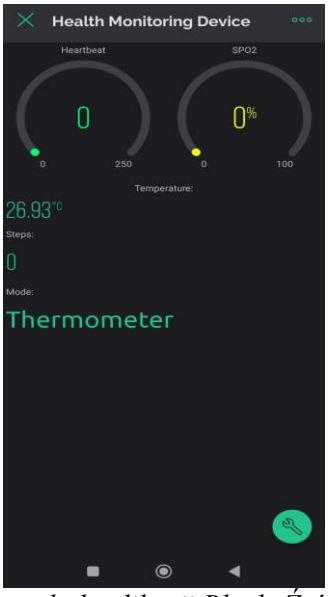


3.7 Opis funkcji do obsługi aplikacji Blynk

Mikrokontroler ESP32 został wyposażony w moduł do komunikacji bezprzewodowej. Jest to zarówno komunikacja za pomocą bluetooth jak i WiFi. W celu wykonania projektu wykorzystana została komunikacja za pomocą standardu WiFi. Komunikacja ta umożliwia przesłanie najważniejszych, zebranych przez mikrokontroler, danych na inne urządzenia. Zapewniony przez producenta moduł obsługuje WiFi w standardach 802.11 b/g/n. Działając w trybie 802.11n pasmo działania WiFi to 2,4GHz, a prędkość maksymalna transmisji to 150Mbps. Najważniejszą cechą wybranego mikrokontrolera jest możliwość pracy w dwóch trybach WiFi. Pierwszy z nich to tryb, w którym kontroler może pracować jako punkt dostępowy (ang. Access Point). W tym trybie wybrane ESP32 ma możliwość działania w sposób podobny do routera, a więc inne urządzenia mogą połączyć się siecią stworzoną przez mikrokontroler. Nie jest jednak wtedy możliwe połączenie z żadną inną siecią, a co za tym idzie, nie jest również możliwy dostęp do Internetu. Istnieje też drugi tryb pracy, a więc tryb stacji. Wtedy mikrokontroler musi połączyć się ze znajdującym się w okolicy access point-em, podobnie jak wszystkie inne urządzenia korzystające z tej sieci. Dzięki temu, mikrokontroler działa jak klient w sieci WiFi, a więc inne urządzenia mogą wysyłać do niego żądania lub na odwrót, mikrokontroler może wysyłać żądania do pozostałych urządzeń. Korzystając z tego trybu ESP32 ma również zapewnione połączenie z Internetem (Jeśli router, do którego podłączony jest mikrokontroler ma połączenie z Internetem). To właśnie drugi tryb, a więc tryb stacji, został wybrany do realizacji projektu. Dzięki temu możliwe jest uzyskanie komunikacji z innym urządzeniem, np. takim jak telefon, który również jest połączony z tym samym access point-em. W celu wyświetlenia wszystkich wartości, na innym niż ESP32, kliencie, została również użyta aplikacja Blynk. Tym samym w celu połączenia mikrokontrolera użyta została funkcja z biblioteki BlynkSimpleEsp32.h:

```
Blynk.begin(auth, ssid, pass);
```

W powyżej przedstawionym fragmencie kodu, auth oznacza token uwierzytelniający wygenerowany przez aplikację Blynk, ssid to nazwa sieci, z którą łączy się mikrokontroler oraz pass to hasło dostępu do tej sieci. Użycie aplikacji Blynk pozwala na wyświetlenie wszystkich wyników w przystępny wizualnie sposób, a przy tym jest bardzo prosta w obsłudze. Przykładowy wygląd aplikacji dla utworzonego programu wygląda następująco:



Rysunek 12. Przykładowy wygląd aplikacji Blynk. Źródło: Opracowanie własne.

3.8 Opis przeprowadzonych testów

Pierwszym z testów przeprowadzonych na wykonanym projekcie, to próby mające na celu potwierdzenie poprawności wskazywanej przez urządzenie wartości pulsu. W tym celu za pomocą powstałego układu zostały zebrane dane z dziesięciu kolejnych pomiarów pulsu, które wykonane były w różnych odstępach czasu. Pomiary wykonywane były na jednej osobie. Równocześnie wykonywane były one w różnych sytuacjach, takich jak np. moment zaraz po wykonaniu ćwiczeń fizycznych, czy moment po dłuższym odpoczynku. Miało to na celu uzyskanie różnych wartości tętna osoby badanej. Jednocześnie zebrane zostały pomiary za pomocą innego urządzenia, które w tym przypadku było opaską sportową firmy Xiaomi. Wyniki zostały zebrane w tabeli:

	Wyniki zebrane za pomocą projektowanego urządzenia [BPM]	Wyniki zebrane za pomocą opaski sportowej Xiaomi [BPM]
Pomiar 1	75	73
Pomiar 2	82	80
Pomiar 3	94	104
Pomiar 4	63	70
Pomiar 5	91	87
Pomiar 6	71	70
Pomiar 7	84	78
Pomiar 8	86	83
Pomiar 9	66	81
Pomiar 10	81	78

Kolejnym badanym parametrem jest saturacja krwi. Podobnie jak w przypadku pulsu, odniesieniem do wyników, osiągniętych przez zaprojektowane urządzenie, będzie dostępna na rynku elektroniki użytkowej opaska sportowa firmy Xiaomi. Posiada ona również możliwość pomiaru saturacji krwi. Warto jednak podkreślić, że sama wykorzystywana do pomiarów opaska sportowa, również charakteryzuje się pewną niedokładnością pomiarów i nie jest urządzeniem medycznym. Opaska firmy Xiaomi wykorzystuje jednak podobną technologię pomiaru, do tej która użyta została do stworzenia urządzenia zaprojektowanego w ramach opisywanego projektu. Biorąc pod uwagę wymienioną cechę obu urządzeń, pomiary wykonane opaską sportową Xiaomi, będą stanowić wystarczająco dobre odniesienie do pomiarów, wykonanych przy pomocy zaprojektowanego urządzenia. W celu zbadania prawidłowości zbieranych przez opisywane urządzenie danych, wykonano 10 pomiarów saturacji krwi używając jednocześnie w tym celu, wymienionej wcześniej opaski sportowej. Zebrane wyniki zostały przedstawione w tabeli poniżej:

	Wyniki zebrane za pomocą projektowanego urządzenia [%]	Wyniki zebrane za pomocą opaski sportowej Xiaomi [%]
Pomiar 1	98.44	95
Pomiar 2	96.02	92
Pomiar 3	99.40	97
Pomiar 4	99.38	98
Pomiar 5	99.05	97
Pomiar 6	99.90	99

Pomiar 7	98.23	97
Pomiar 8	99.07	98
Pomiar 9	99.60	98
Pomiar 10	96.67	94

Jednym z badanych przez zaprojektowane urządzenie parametrów zdrowia jest temperatura ciała. Podobnie do poprzednich parametrów, po stworzeniu i zaprogramowaniu urządzenia, konieczne jest zweryfikowanie poprawności mierzonych danych. W tym celu na grupie dwóch zdrowych osób wykonano 5 kolejnych pomiarów przy pomocy zaprojektowanego urządzenia. Pomiary wykonany były poprzez umieszczenie odpowiedniego sensora blisko czoła osoby badanej. Wyniki były zapisywane gdy urządzenie wyświetlało powtarzalną wartość. Oczekiwane wyniki pomiarów powinny oscylować wokół 36,6°C. W celu potwierdzenia tego założenia obu osobom początkowo zmierzono temperaturę za pomocą klasycznego termometru elektronicznego, który potwierdził przewidywaną wcześniej wartość temperatury. Zgromadzone dane przedstawiono w tabeli poniżej:

	Temperatura zmierzona dla pierwszej badanej osoby [°C]	Temperatura zmierzona dla drugiej badanej osoby [°C]
Pomiar 1	36.04	35.20
Pomiar 2	35.51	36.34
Pomiar 3	36.17	36.27
Pomiar 4	35.98	36.13
Pomiar 5	36.68	35.96

W celu weryfikacji poprawności metody zliczania wykonanych kroków, przeprowadzone zostały testy zaprojektowanego urządzenia, które polegały na wykonaniu 5 prób przejścia określonego odcinka drogi, trzymając równocześnie opisywane urządzenie w ręce. Odcinek drogi mierzony był w krokach, a każdy z nich liczył dokładnie 50 kroków. Następnie porównane zostały wyniki uzyskane za pomocą urządzenia, z realną wartością wykonanych kroków. Wyniki przeprowadzonego w ten sposób testu zostały zebrane w tabeli poniżej:

	Liczba kroków zliczonych przez projektowane urządzenie	Liczba kroków wykonanych w próbie
Pomiar 1	53	50
Pomiar 2	54	50
Pomiar 3	46	50
Pomiar 4	41	50
Pomiar 5	44	50

Podsumowanie:

Zaprojektowane, w ramach realizowanego projektu, urządzenie potwierdziło zdolność do monitorowania podstawowych parametrów zdrowia takich jak puls, saturacja krwi czy temperatura ciała. Ponadto urządzenie, zgodnie z założeniami, zdolne jest do zliczania wykonanej przez użytkownika ilości kroków. Problemem napotkanym przy tworzeniu projektu stała się dokładność wykonywanych, przy pomocy projektowanego urządzenia, pomiarów. W zależności od badanego parametru, skala rozbieżności pomiędzy wartością uzyskaną, a

wartością oczekiwaną jest różna. Najgorzej pod tym względem wypadł pomiar temperatury, czego powody zostały opisane w rozdziale 4.2. Zaprojektowane urządzenie, pomimo potwierdzonej możliwości do zwracania przybliżonych wartości poszczególnych parametrów zdrowia, nie może być jednak uznane za urządzenie medyczne. Użycie prawidłowych komponentów spełniających odpowiednie standardy byłoby konieczne, aby możliwe stało się nazwanie podobnego urządzenia mianem medycznego. Organizacje takie jak IEEE tworząc różnego rodzaju standardy, w tym te medyczne, przywiązują również wagę do implementacji programu, który również powinien być napisany we właściwy dla określonego standardu sposób. Dodatkowo aplikacja Blynk jako darmowe oprogramowanie, w dużym stopniu ogranicza możliwości urządzenia do wyświetlania wszystkich informacji na urządzeniu podpiętym do tej samej sieci WiFi. Przedstawiony projekt potwierdził jednak możliwość wykonania urządzenia o założonych na początku pracy celach, a także dodatkowo daje możliwość do dalszego jego rozwoju.