

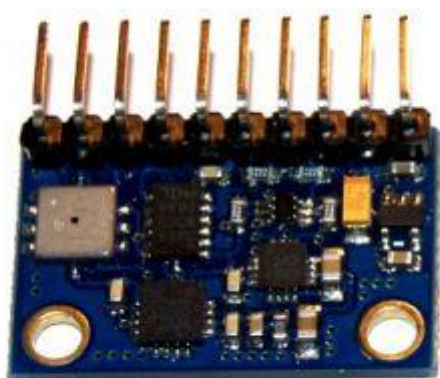
Korneliusz Jarzębski

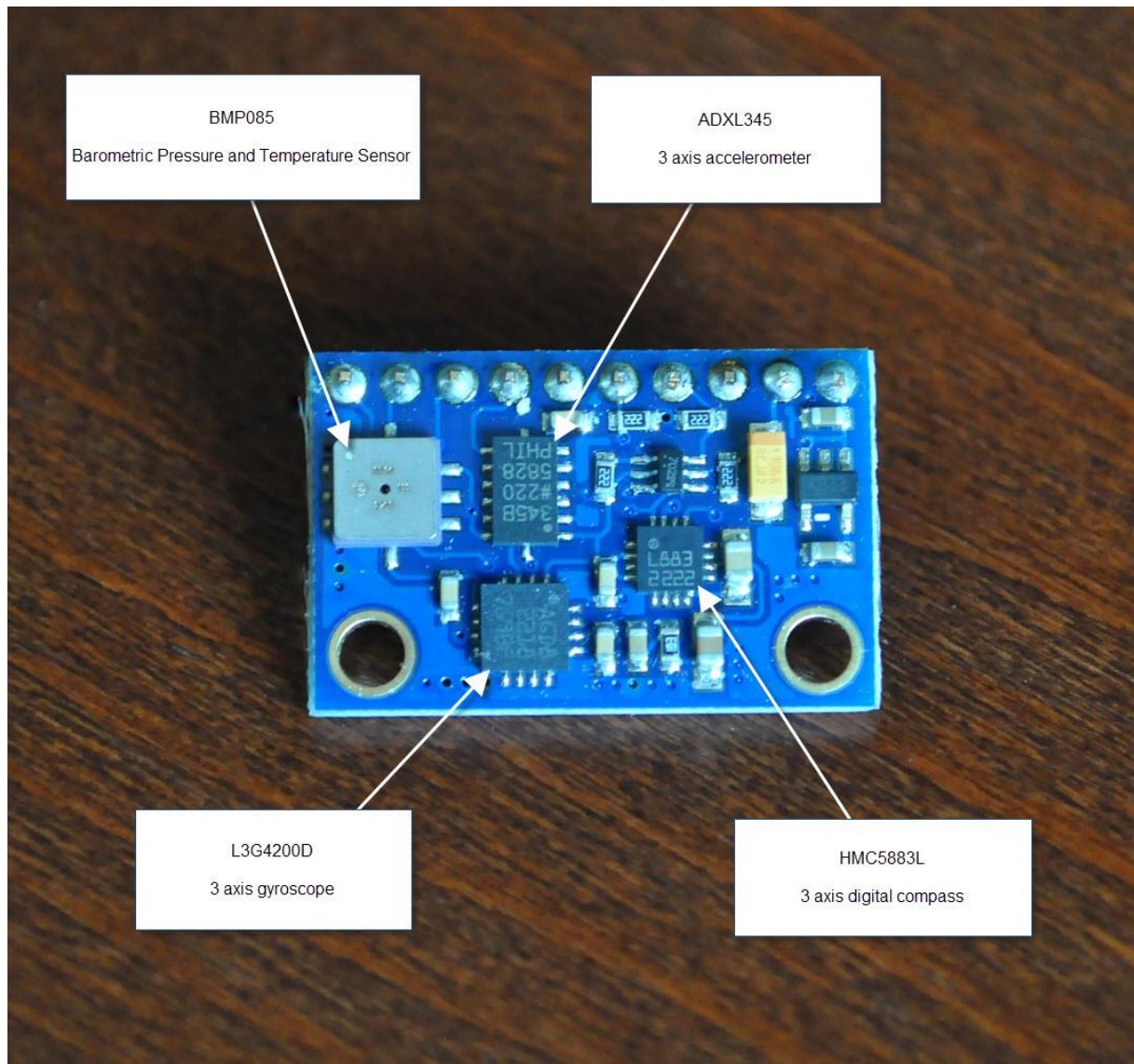
# GY-80: akcelerometr, żyroskop, magnetometr, barometr

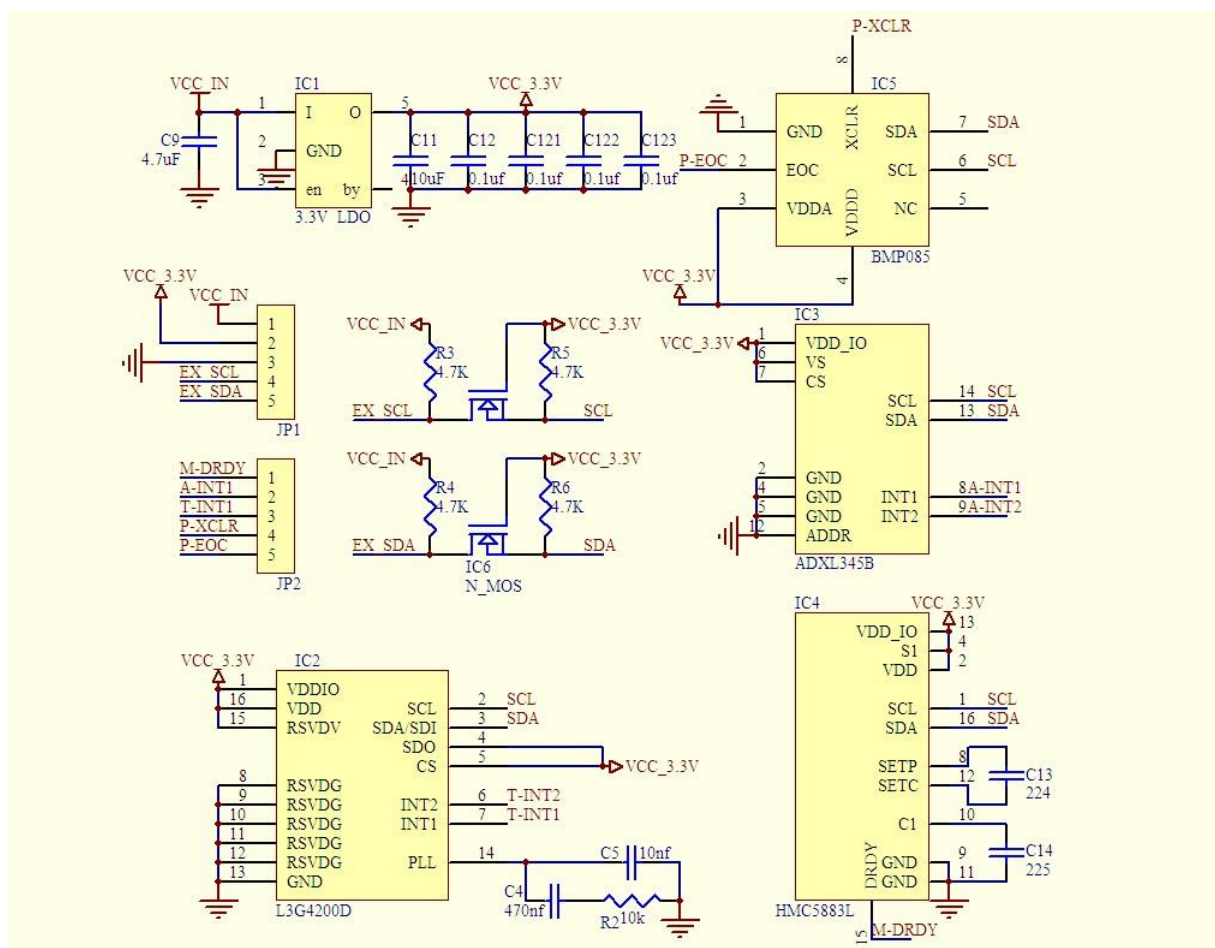
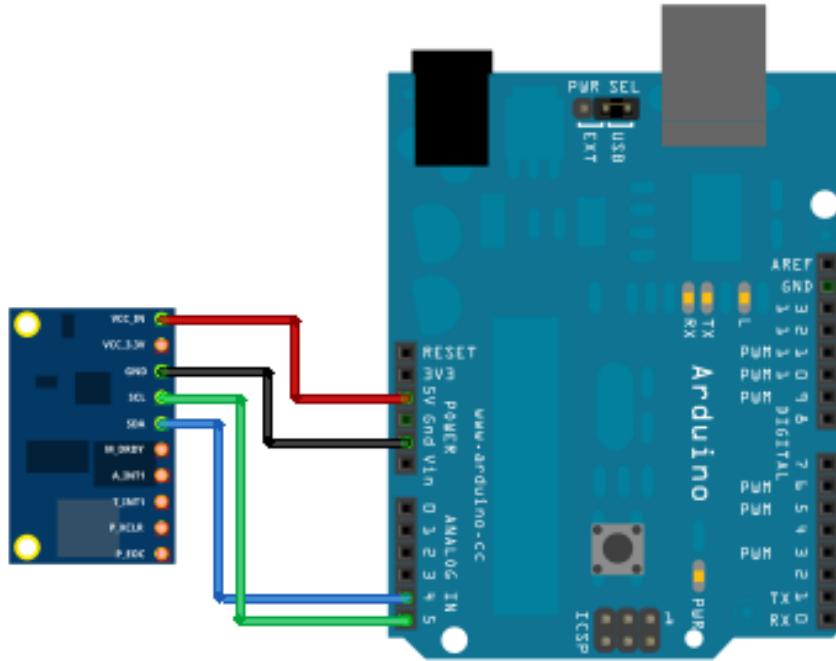
## Spis treści

GY-80: akcelerometr, żyroskop, magnetometr, barometr.....	1
3-osiowy żyroskop L3G4200D .....	4
3-osiowy akcelerometr ADXL345 .....	10
3-osiowy magnetometr HMC5883L .....	23
Czujniki ciśnienia i temperatury BMP085 / BMP180.....	31

**GY-80** to jeden z popularniejszych modułów **IMU** (*inertial measurement unit* - *inercyjny zespół pomiarowy*) wykorzystywany w budowie samo balansujących robotów lub multicopterów. Jego zaletą jest nie tylko cena, ale również udostępnienie pomiarów aż 10 stopni swobody do których zaliczamy **3-osiowy akcelerometr, 3-osiowy żyroskop, 3-osiowy magnetometr oraz barometr**.







## 3-osiowy żyroskop L3G4200D

**Żyroskop** to urządzenie służące do pomiaru lub utrzymania położenia kąтового, który działa w oparciu o zasadę zachowania momentu pędu. Układ **L3G4200D** stanowi osobną grupę żyroskopów prędkościowych, które nie utrzymują stałego kierunku, ale wskazują prędkość kątową obiektu, na którym się znajduje. **Żyroskopy** są głównie wykorzystywane do budowy **żyrokompasów**, stosowanych przy budowie samobalansujących **robotów** oraz **multicopterów**.

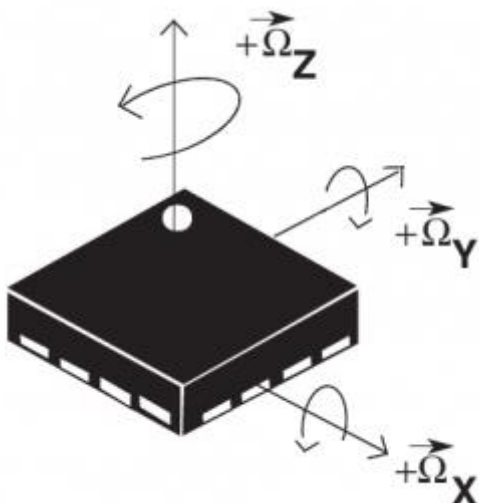
**Żyroskop** sam w sobie nie znajdzie zbyt dużego zastosowania, jednak idealnie nadaje się do wspomagania pomiarów z innych czujników np.: **akcelerometrów**, które oprócz wskazania przyspieszenia obiektu względem wybranej osi, dają możliwość określenia jego konfiguracji trzech osi: nachylenia (*Pitch*), przechylenia (*Roll*) oraz obrotu (*Yaw*). Praktycznie wyznaczenie tych osi jest możliwe za pomocą samego żyroskopu, jednak może okazać się niewystarczające.



Innym problemem jest wyznaczenie parametrów **Pitch**, **Roll** i **Yaw** za pomocą samego **akcelerometru**, który w miarę dokładnie poradzi sobie z wyznaczeniem tylko pierwszych dwóch (*Pitch* i *Roll*) - o ile nasz obiekt nie będzie się przemieszczał. Kompletnie nie nada się natomiast do wyznaczenia parametru **Yaw**. Tak jak wspomniałem wcześniej - idealnym rozwiązaniem jest uwzględnienie pomiarów zarówno z **akcelerometru** i **żyroskopu**, który skompensuje błędy wynikające z ruchu. Pomimo wykorzystania **akcelerometru** i **żyroskopu**, kłopotliwy okaże się wciąż **Yaw**, który docelowo powinno się określać za pomocą **magnetometru** (*kompasu*), ale również z uwzględnieniem pomiarów z **akcelerometru**. Koniec, końców - najlepsze rezultaty dadzą wszystkie trzy czujniki. Do tego tematu powrócę jeszcze w artykułach o **akcelerometrach** i **magnetometrach** oraz testu *inercyjnych zespołów pomiarowych IMU*.

Wróćmy jednak do naszego **żyroskopu**. Zobaczmy jak przedstawiają się wybrane osie obrotów względem samego układu.



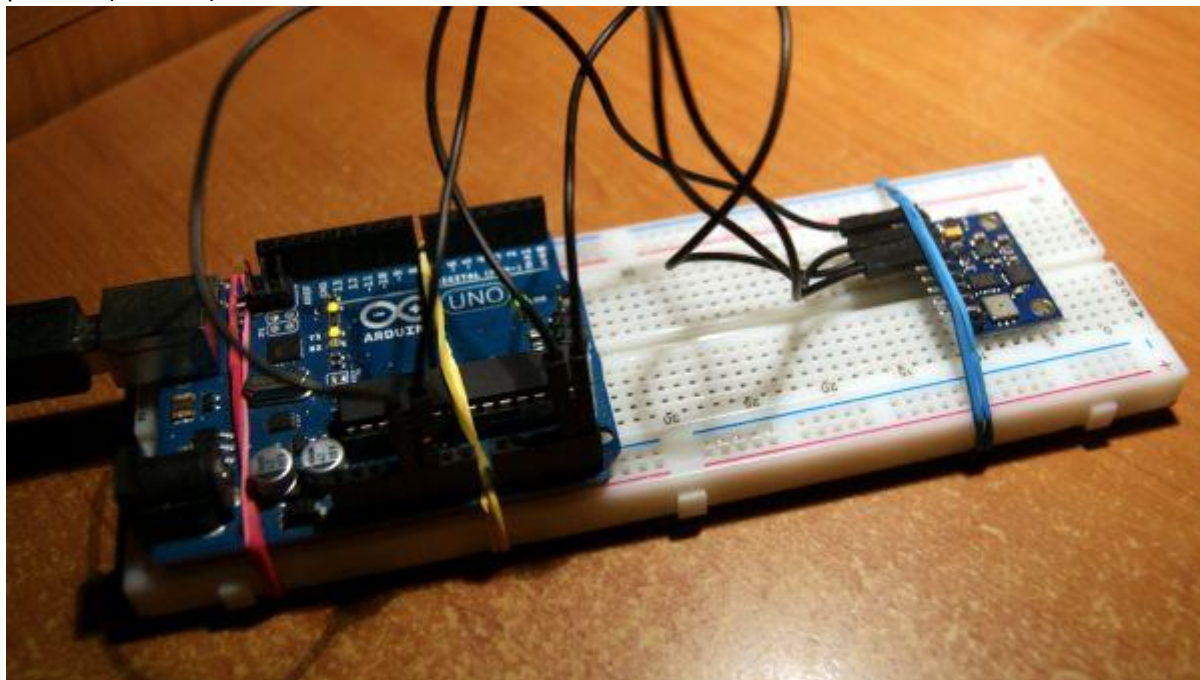


Jeśli chodzi o parametry **L3G4200D** to pozwala on na wybór jednego z trzech zakresów pomiarowych:  $\pm 250^\circ/\text{s}$ ,  $\pm 500^\circ/\text{s}$  lub  $\pm 2000^\circ/\text{s}$ . Czujnik obsługuje zarówno komunikację **I<sup>2</sup>C** jak i **SPI**, jednak najczęściej spotkamy się modułami przystosowanymi jedynie do magistrali **I<sup>2</sup>C**.

**L3G4200D** posiada również wyjścia cyfrowe (**INT1** i **INT2**) mogące sygnalizować (w zależności od konfiguracji rejestrów) pojawienie się nowego pomiaru, przepełnienie **FIFO** lub zbyt niskiej/wysokiej prędkości obrotu. Układ może być zasilany napięciem z zakresu **2,4 - 3,6 V**, natomiast logika we/wy do **1,71 V**. Typowy pobór prądu podczas pomiaru to zaledwie **6mA**. Należy zwrócić więc szczególną uwagę czy nasz moduł będzie tolerował zasilanie 5V, gdyż nieodpowiedni jego poziom może uszkodzić układ.

Pełna dokumentacja techniczna: <http://www.jarzebski.pl/datasheets/L3G4200D.pdf>

Osobiście posiadam moduł **IMU GY-80**, który pozwala na zasilanie napięciem **5V**. Pin oznaczony **SCL** (*adapter*) podłączamy do pinu **A5** (*Arduino*), natomiast pin **SDA** (*adapter*) do pinu **A4** (*Arduino*).



#### Biblioteka i program testowy

Niestety - przeszukując sieć, nie natrafiłem na w żadną dobrą bibliotekę dla **Arduino**, dlatego zbierając szczątkowe implementacje, postanowiłem napisać własną. W odróżnieniu od dostępnych

bibliotek, moja biblioteka pozwala na kalibrację układu w spoczynku (*do kompensacji późniejszych pomiarów*), wybór zakresu pomiarowego i normalizowanie danych, a także ustawienie współczynnika progu czułości. Oczywiście jest to pierwsza wersja, więc na pewno będzie jeszcze rozwijana. Biblioteka posiada również przykłady wykorzystania oraz program do wizualizacji pomiarów. Bibliotekę można pobrać stąd: <https://github.com/jarzebski/Arduino-L3G4200D>

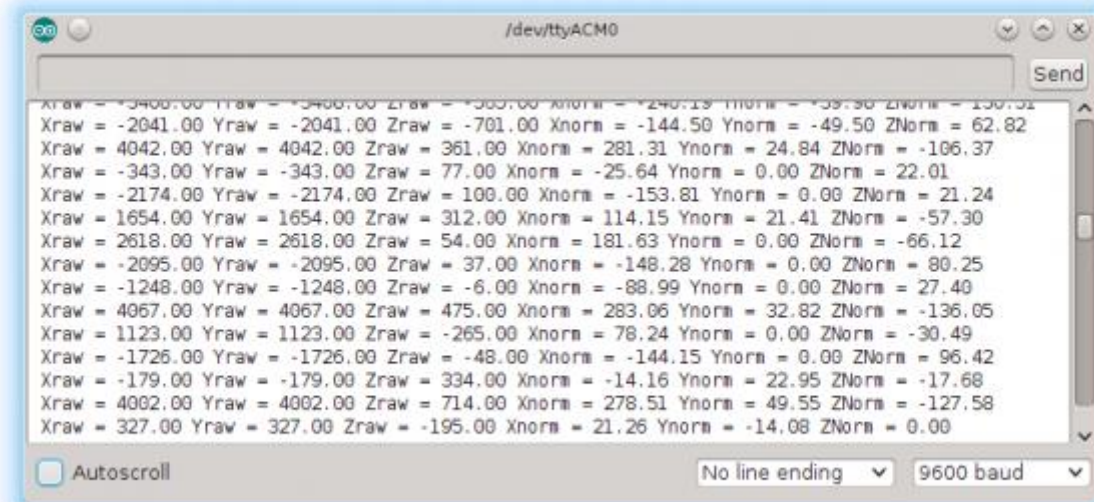
```
1. #include <Wire.h>
2. #include <L3G4200D.h>
3.
4. L3G4200D gyroscope;
5.
6. void setup()
7. {
8.   Serial.begin(9600);
9.
10.  // Inicjalizacja L3G4200D
11.  // 250 dps: L3G4200D_250DPS
12.  // 500 dps: L3G4200D_500DPS
13.  // 2000 dps: L3G4200D_2000DPS
14.  while(!gyroscope.begin(L3G4200D_2000DPS))
15.  {
16.    Serial.println("Nie odnaleziono L3G4200D. Sprawdź połączenie.");
17.    delay(500);
18.  }
19.
20.  // Sprawdzamy skalę
21.  Serial.print("Wybrana skala: ");
22.
23.  switch(gyroscope.getScale())
24.  {
25.    case L3G4200D_SCALE_250DPS:
26.      Serial.println ("250 dps");
27.      break;
28.    case L3G4200D_SCALE_500DPS:
29.      Serial.println ("500 dps");
30.      break;
31.    case L3G4200D_SCALE_2000DPS:
32.      Serial.println ("2000 dps");
33.      break;
34.  }
35.
36.  // Kalibracja żyroskopu. Powinna odbywać się w spoczynku zerowym
37.  gyroscope.calibrate();
38.
39.  // Ustawiamy próg czułości na 3.
40.  gyroscope.setThreshold(3);
41. }
42.
43. void loop()
```

```

44. {
45.   // Odczytujemy surowe dane z zyrokopu
46.   Vector raw = gyroscope.readRaw();
47.
48.   // Odczytujemy znormalizowane wyniki w °/s
49.   Vector norm = gyroscope.readNormalize();
50.
51.   // Wyświetlamy wyniki
52.   Serial.print(" Xraw = ");
53.   Serial.print(raw.XAxis);
54.   Serial.print(" Yraw = ");
55.   Serial.print(raw.YAxis);
56.   Serial.print(" Zraw = ");
57.   Serial.print(raw.ZAxis);
58.   Serial.print(" Xnorm = ");
59.   Serial.print(norm.XAxis);
60.   Serial.print(" Ynorm = ");
61.   Serial.print(norm.YAxis);
62.   Serial.print(" ZNorm = ");
63.   Serial.print(norm.ZAxis);
64.
65.   Serial.println();
66. }

```

Wynik działania



### Wizualizacja w Processing.org

Lekko zmodyfikowany program przedstawia się następująco:

1. `#include <Wire.h>`
2. `#include <L3G4200D.h>`
- 3.
4. `L3G4200D gyroscope;`
- 5.
6. `int LED = 13;`

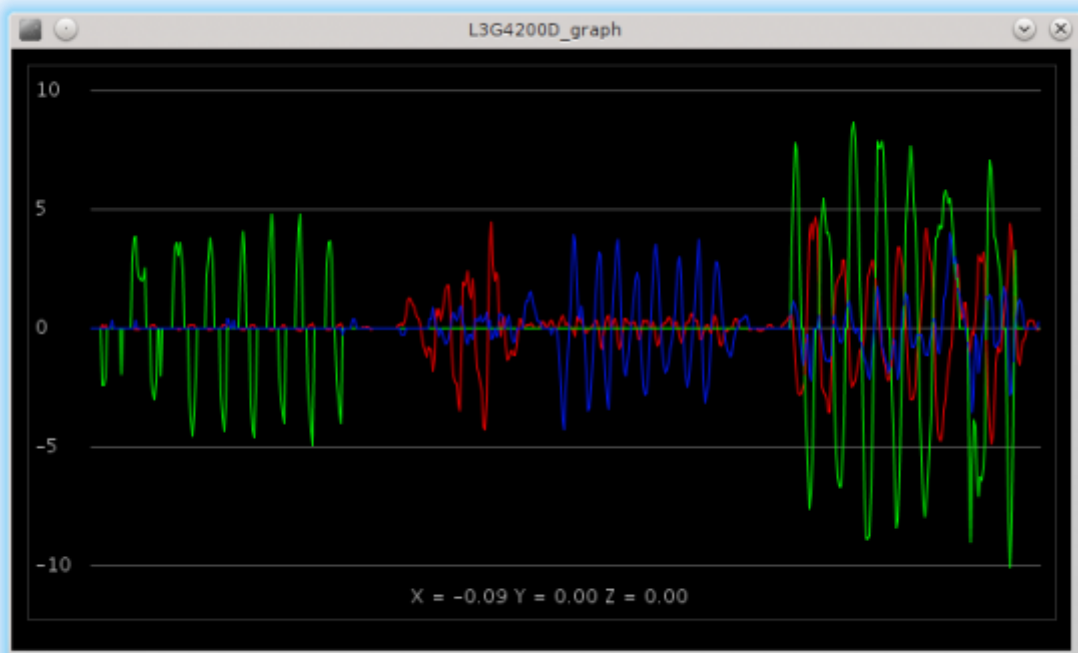
```

7.  boolean Blink = false
8.
9.  void setup()
10. {
11.  Serial.begin(9600);
12.  pinMode(LED, OUTPUT);
13.
14.  // Inicjalizacja L3G4200D
15.  while (!gyroscope.begin(L3G4200D_SCALE_2000DPS))
16.  {
17.    if (Blink)
18.    {
19.      digitalWrite(LED, HIGH);
20.    } else
21.    {
22.      digitalWrite(LED, LOW);
23.    }
24.
25.    Blink = !Blink;
26.
27.    delay(500);
28.  }
29.
30.  digitalWrite(LED, HIGH);
31.
32.  // Kalibracja żyroskopu. Powinna odbywać się w spoczynku zerowym
33.  gyroscope.calibrate();
34.
35.  // Ustawiamy próg czułości na 3.
36.  gyroscope.setThreshold(3);
37.  digitalWrite(LED, LOW);
38. }
39.
40. void loop()
41. {
42.  // Odczytujemy znormalizowane wyniki w °/s
43.  Vector norm = gyroscope.readNormalize();
44.
45.  // Output
46.  Serial.print(norm.XAxis);
47.  Serial.print(":");
48.  Serial.print(norm.YAxis);
49.  Serial.print(":");
50.  Serial.print(norm.ZAxis);
51.  Serial.println();
52. }

```

Program dla **Processing** znajdziecie w archwium biblioteki, dzięki któremu możemy obserwować wyniki pomiarów. Dla ułatwienia stopnie zostały zamienione na radiany:





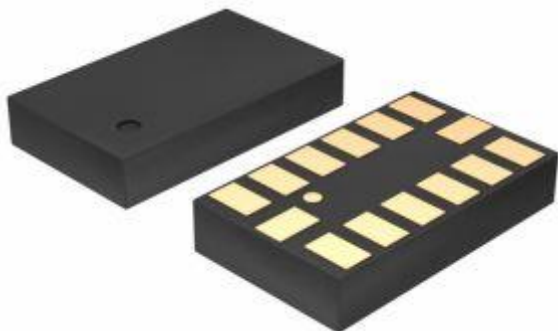
#### **Materiały dodatkowe**

Biblioteka **L3G4200D**: <https://github.com/jarzebski/Arduino-L3G4200D>

Dokumentacja techniczna: <http://www.jarzebski.pl/datasheets/L3G4200D.pdf>

## 3-osiowy akcelerometr ADXL345

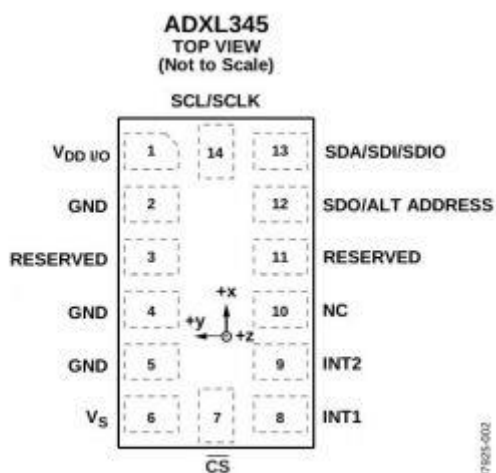
Zadaniem **akcelerometrów** jest pomiar przyspieszeń liniowych podczas własnego ruchu. Najczęściej stosuje się je do badania części ruchomych urządzeń oraz określenia ich przeciążeń. **Akcelerometry** możemy obecnie znaleźć w urządzeniach takich jak: **telefony komórkowe, tablety, aparaty fotograficzne**, a także w modelach zdalnego sterowania RC(np.: *multi-coptery*).



Oprócz określenia wartości przyspieszeń liniowych, możliwe jest wyznaczenie za ich pomocą ułożenia przestrzennego obiektu oraz wykonania określonych interakcji podczas jego poruszania się. Jednym z popularniejszych i niedrogich układów stosowanych w takich modułach jest **ADXL345** od **Analog Devices**. Układ ten może komunikować się mikrokontrolerem za pomocą magistrali **I<sup>2</sup>C** lub **SPI**, mierząc przyspieszenia we wszystkich trzech osiach w zakresie nawet do **±16 g** z rozdzielczością **13 bitów**.

Podczas pomiaru, maksymalny pobór prądu to zaledwie **23μA**, przy napięciu zasilania od **2.0** do **3.6V**. Ponieważ układ nie toleruje zasilania wyższego (5V), należy zwrócić szczególną uwagę na to, czy nasz moduł posiada możliwość zasilania takim napięciem. Na szczęście zdecydowana większość dostępnych modułów z tym układem posiada wbudowany regulator napięcia.

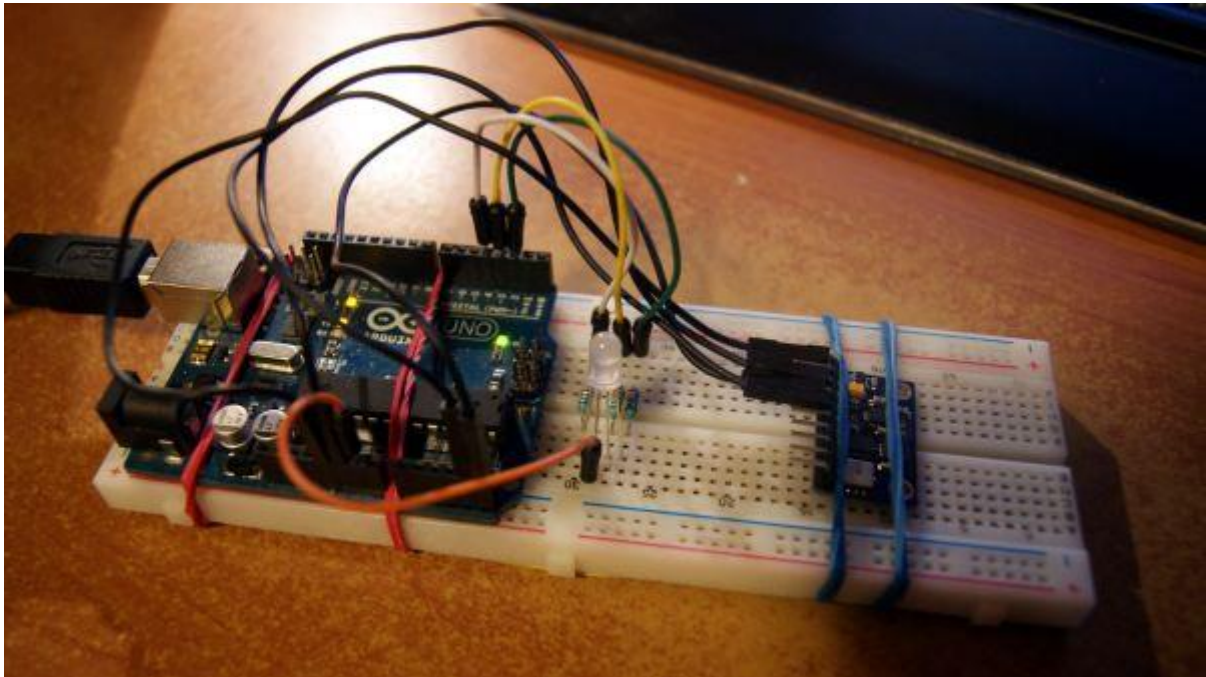
Układ jest zamknięty w obudowie **LGA** o wymiarach 5mm x 3mm i wysokości 1mm - niech Was nie zmyli ta "pchelka", ponieważ ma on (oprócz pomiaru przyspieszeń) jeszcze kilka niespodzianek. **ADXL345** posiada również wyjścia cyfrowe (**INT1** i **INT2**) mogące sygnalizować detekcję **stuknięcia**, **podwójnego stuknięcia** (*tap, double tap*), **aktywności** lub **nieaktywności** oraz stanu **swobodnego spadania**.



Pełna dokumentacja techniczna: <http://www.jarzebski.pl/datasheets/ADXL345.pdf>

**Podłączenie ADXL345 do Arduino**

W przypadku modułu **IMU GY-80**, możemy skorzystać z **5V** zasilania. Pin oznaczony **SCL** (*adapter*) podłączamy do pinu **A5** (*Arduino*), natomiast pin **SDA** (*adapter*) do pinu **A4** (*Arduino*). W moim układzie wykorzystałem również diodę **RGB** ze wspólną katodą oraz trzema rezystorami **220Ω**, sterowaną wyjściami cyfrowymi **Arduino** (2,3,4) do sygnalizacji przerwań.



Do obsługi modułów z układami **ADXL345** przygotowałem również odpowiednią do nich bibliotekę dla **Arduino**, którą można pobrać z repozytorium **Git**: <https://github.com/jarzebski/Arduino-ADXL345>

#### Prosty przykład

Pierwszym przykładem będzie odczyt surowych wartości oraz znormalizowanych ( $m/s^2$ ):

```
1. #include <Wire.h>
2. #include <ADXL345.h>
3.
4. ADXL345 accelerometer;
5.
6. void showRange(void)
7. {
8.   Serial.print("Wybrany zakres pomiarowy: ");
9.
10.  switch(accelerometer.getRange())
11.  {
12.    case ADXL345_RANGE_16G: Serial.println("+/- 16 g"); break;
13.    case ADXL345_RANGE_8G: Serial.println("+/- 8 g"); break;
14.    case ADXL345_RANGE_4G: Serial.println("+/- 4 g"); break;
15.    case ADXL345_RANGE_2G: Serial.println("+/- 2 g"); break;
16.    default: Serial.println("Bledny zakres"); break;
17.  }
18. }
19.
20. void showDataRate(void)
21. {
22.   Serial.print("Wybrana szybkość transmisji: ");
```

```

23.
24. switch(accelerometer.getDataRate())
25. {
26.   case ADXL345_DATARATE_3200HZ: Serial.println("3200 Hz"); break;
27.   case ADXL345_DATARATE_1600HZ: Serial.println("1600 Hz"); break;
28.   case ADXL345_DATARATE_800HZ: Serial.println("800 Hz"); break;
29.   case ADXL345_DATARATE_400HZ: Serial.println("400 Hz"); break;
30.   case ADXL345_DATARATE_200HZ: Serial.println("200 Hz"); break;
31.   case ADXL345_DATARATE_100HZ: Serial.println("100 Hz"); break;
32.   case ADXL345_DATARATE_50HZ: Serial.println("50 Hz"); break;
33.   case ADXL345_DATARATE_25HZ: Serial.println("25 Hz"); break;
34.   case ADXL345_DATARATE_12_5HZ: Serial.println("12.5 Hz"); break;
35.   case ADXL345_DATARATE_6_25HZ: Serial.println("6.25 Hz"); break;
36.   case ADXL345_DATARATE_3_13HZ: Serial.println("3.13 Hz"); break;
37.   case ADXL345_DATARATE_1_56HZ: Serial.println("1.56 Hz"); break;
38.   case ADXL345_DATARATE_0_78HZ: Serial.println("0.78 Hz"); break;
39.   case ADXL345_DATARATE_0_39HZ: Serial.println("0.39 Hz"); break;
40.   case ADXL345_DATARATE_0_20HZ: Serial.println("0.20 Hz"); break;
41.   case ADXL345_DATARATE_0_10HZ: Serial.println("0.10 Hz"); break;
42.   default: Serial.println("Bleda szybkość transmisji"); break;
43. }
44. }
45.
46. void setup(void)
47. {
48.   Serial.begin(9600);
49.
50.   // Inicjalizacja ADXL345
51.   Serial.println("Inicjalizacja ADXL345");
52.   if (!accelerometer.begin())
53.   {
54.     Serial.println("Nie odnaleziono ADXL345, sprawdź podłączenie!");
55.     delay(500);
56.   }
57.
58.   // Wybór zakresu pomiarowego
59.   // +/- 2G: ADXL345_RANGE_2G
60.   // +/- 4G: ADXL345_RANGE_4G
61.   // +/- 8G: ADXL345_RANGE_8G
62.   // +/- 16G: ADXL345_RANGE_16G
63.   accelerometer.setRange(ADXL345_RANGE_16G);
64.
65.   // Wyświetlenie aktualnych parametrów
66.   showRange();
67.   showDataRate();
68. }
69.
70. void loop(void)

```

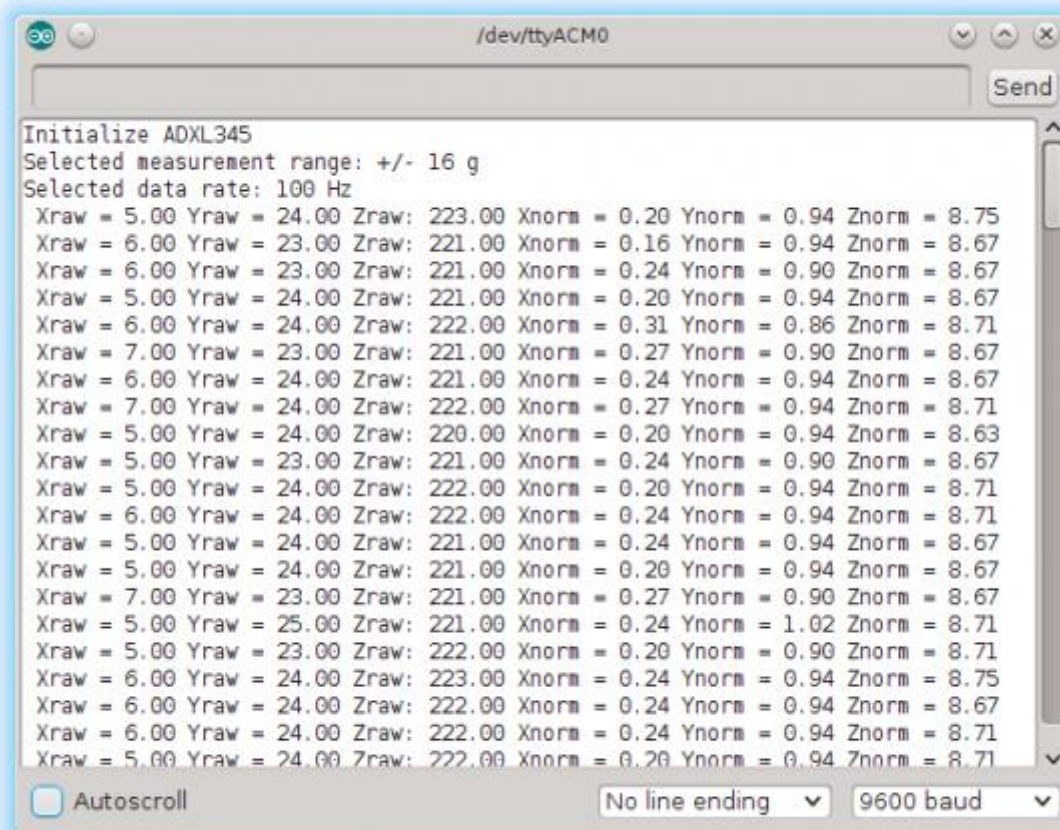
```

71. {
72.  // Odczyt wartosci surowych
73.  Vector raw = accelerometer.readRaw();
74.
75.  // Odczyt wartosci znormalizowanych
76.  Vector norm = accelerometer.readNormalize();
77.
78.  // Wyświetlenie danych surowych
79.  Serial.print(" Xraw = ");
80.  Serial.print(raw.XAxis);
81.  Serial.print(" Yraw = ");
82.  Serial.print(raw.YAxis);
83.  Serial.print(" Zraw: ");
84.  Serial.print(raw.ZAxis);
85.
86.  // Wyświetlenie danych znormalizowanych m/s^2
87.  Serial.print(" Xnorm = ");
88.  Serial.print(norm.XAxis);
89.  Serial.print(" Ynorm = ");
90.  Serial.print(norm.YAxis);
91.  Serial.print(" Znorm = ");
92.  Serial.print(norm.ZAxis);
93.
94.  Serial.println();
95.
96.  delay(200);
97. }

```



Wynik po uruchomieniu programu:



The screenshot shows a terminal window titled "/dev/ttyACM0" with a "Send" button in the top right. The output text is as follows:

```
Initialize ADXL345
Selected measurement range: +/- 16 g
Selected data rate: 100 Hz
Xraw = 5.00 Yraw = 24.00 Zraw: 223.00 Xnorm = 0.20 Ynorm = 0.94 Znorm = 8.75
Xraw = 6.00 Yraw = 23.00 Zraw: 221.00 Xnorm = 0.16 Ynorm = 0.94 Znorm = 8.67
Xraw = 6.00 Yraw = 23.00 Zraw: 221.00 Xnorm = 0.24 Ynorm = 0.90 Znorm = 8.67
Xraw = 5.00 Yraw = 24.00 Zraw: 221.00 Xnorm = 0.20 Ynorm = 0.94 Znorm = 8.67
Xraw = 6.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.31 Ynorm = 0.86 Znorm = 8.71
Xraw = 7.00 Yraw = 23.00 Zraw: 221.00 Xnorm = 0.27 Ynorm = 0.90 Znorm = 8.67
Xraw = 6.00 Yraw = 24.00 Zraw: 221.00 Xnorm = 0.24 Ynorm = 0.94 Znorm = 8.67
Xraw = 7.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.27 Ynorm = 0.94 Znorm = 8.71
Xraw = 5.00 Yraw = 24.00 Zraw: 220.00 Xnorm = 0.20 Ynorm = 0.94 Znorm = 8.63
Xraw = 5.00 Yraw = 23.00 Zraw: 221.00 Xnorm = 0.24 Ynorm = 0.90 Znorm = 8.67
Xraw = 5.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.20 Ynorm = 0.94 Znorm = 8.71
Xraw = 6.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.24 Ynorm = 0.94 Znorm = 8.71
Xraw = 5.00 Yraw = 24.00 Zraw: 221.00 Xnorm = 0.24 Ynorm = 0.94 Znorm = 8.67
Xraw = 5.00 Yraw = 24.00 Zraw: 221.00 Xnorm = 0.20 Ynorm = 0.94 Znorm = 8.67
Xraw = 7.00 Yraw = 23.00 Zraw: 221.00 Xnorm = 0.27 Ynorm = 0.90 Znorm = 8.67
Xraw = 5.00 Yraw = 25.00 Zraw: 221.00 Xnorm = 0.24 Ynorm = 1.02 Znorm = 8.71
Xraw = 5.00 Yraw = 23.00 Zraw: 222.00 Xnorm = 0.20 Ynorm = 0.90 Znorm = 8.71
Xraw = 6.00 Yraw = 24.00 Zraw: 223.00 Xnorm = 0.24 Ynorm = 0.94 Znorm = 8.75
Xraw = 6.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.24 Ynorm = 0.94 Znorm = 8.67
Xraw = 6.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.24 Ynorm = 0.94 Znorm = 8.71
Xraw = 5.00 Yraw = 24.00 Zraw: 222.00 Xnorm = 0.20 Ynorm = 0.94 Znorm = 8.71
```

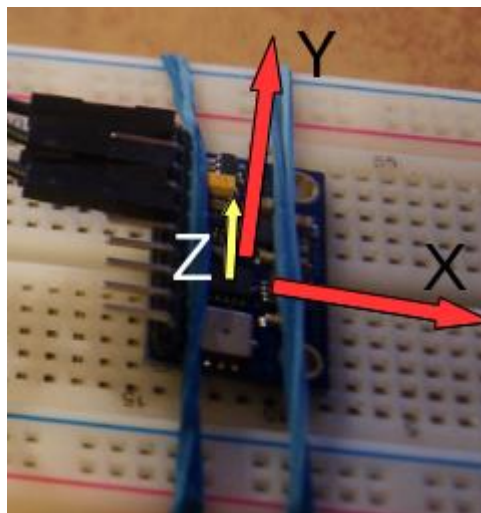
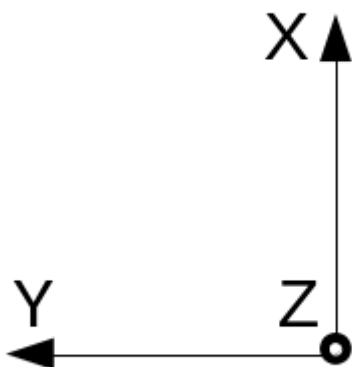
At the bottom of the window, there is an "Autoscroll" checkbox (unchecked), a "No line ending" dropdown menu, and a "9600 baud" dropdown menu.

### Interpretacja wyników

W wynikach znormalizowanych otrzymujemy wartość obecnego przyspieszenia względem wszystkich trzech osi układu. Jak widzimy, kiedy układ jest w spoczynku, na oś "Z" działa przyspieszenie o wartości około  $8.75 \text{ m/s}^2$ . Dlaczego takie? Pamiętajmy o przyspieszeniu grawitacyjnym (*normalnym*), które wynosi około  $9.80665 \text{ m/s}^2$ . Nie jest to oczywiście stała wartość, ponieważ jest zmienna w zależności od miejsca na ziemi - inna jest na biegunie ( $9.83 \text{ m/s}^2$ ), inna też na równiku ( $9.78 \text{ m/s}^2$ ). Wartość przyspieszenia  $9.80665 \text{ m/s}^2$  określamy również mianem przeciążenia o wartości **1 g**. Łatwo więc obliczyć, że **ADXL345** poradzi sobie z przyspieszeniami sięgającymi do  $157 \text{ m/s}^2$  ( $16 \text{ g}$ ). Błąd jak wskazuje układ to  $\Delta\delta \approx 1 \text{ m/s}^2$  czyli około **0.1 g**. Możemy taki układ skalibrować, jednak do naszych potrzeb nie będzie to konieczne.

### Wyznaczenie ułożenia przestrzennego obiektu (Pitch, Roll)

Zanim przejdziemy do obliczeń, przypomnijmy sobie jak wygląda konfiguracja osi **X,Y,Z** układu **ADXL345**:



Jak widać, przechylenia odbywają się na boki wokół **osi X**, natomiast nachylenia do góry i w dół wokół **osi Y**. Zmiana kierunku odbywa się wokół **osi Z**. Odpowiadają temu odpowiednie kąty:

- $\Phi$  - **Roll** - obrót wokół osi X
- $\theta$  - **Pitch** - obrót wokół osi Y
- $\psi$  - **Yaw** - obrót wokół osi Z

Na podstawie bezpośrednio odczytanych wartości przyspieszeń dla poszczególnych osi, możemy obliczyć kąt  $\Phi$  - **Roll** oraz  $\theta$  - **Pitch**. Nie jestem wybitnym matematykiem, aby przedstawić Wam szczegółowe metody obliczenia kątów na podstawie wartości wektorów przyspieszeń, dlatego odsyłam zainteresowanych do noty aplikacyjnej [AN3461](#), przygotowanej przez **Freescall Semiconductor**. Nas interesują jedynie końcowe wzory ze strony 10 powyższej noty:

$$\tan \phi_{xyz} = \left( \frac{G_{py}}{G_{pz}} \right)$$

$$\tan \theta_{xyz} = \left( \frac{-G_{px}}{G_{py} \sin \phi + G_{pz} \cos \phi} \right) = \frac{-G_{px}}{\sqrt{G_{py}^2 + G_{pz}^2}}$$

Skorzystamy z funkcji **atan2()** zamiast **tan2()**, aby ułatwić sobie kontrolę mianownika, który nie może być zerem. Dodatkowo wyeliminujemy dwuznaczności kąta w zależności od ćwiartki układu. Dla ułatwienia odczytu, otrzymany wynik w radianach przekształcimy sobie na stopnie.

W przykładzie wykorzystano również filtr dolnoprzepustowy za pomocą funkcji **lowPassFilter()**, która pozwoli nam na drobne wytłumienie zbyt wielkich skoków odczytu. Pełne wytłumaczenie i wyprowadzenie wzoru znajdziecie również na [Wikipedii](#).

```

1. #include <Wire.h>
2. #include <ADXL345.h>
3.
4. ADXL345 accelerometer;
5.
6. void setup(void)
7. {
8.   Serial.begin(9600);
9.
10.  // Inicjalizacja ADXL345
11.  Serial.println("Inicjalizacja ADXL345");
12.  if (!accelerometer.begin())
13.  {

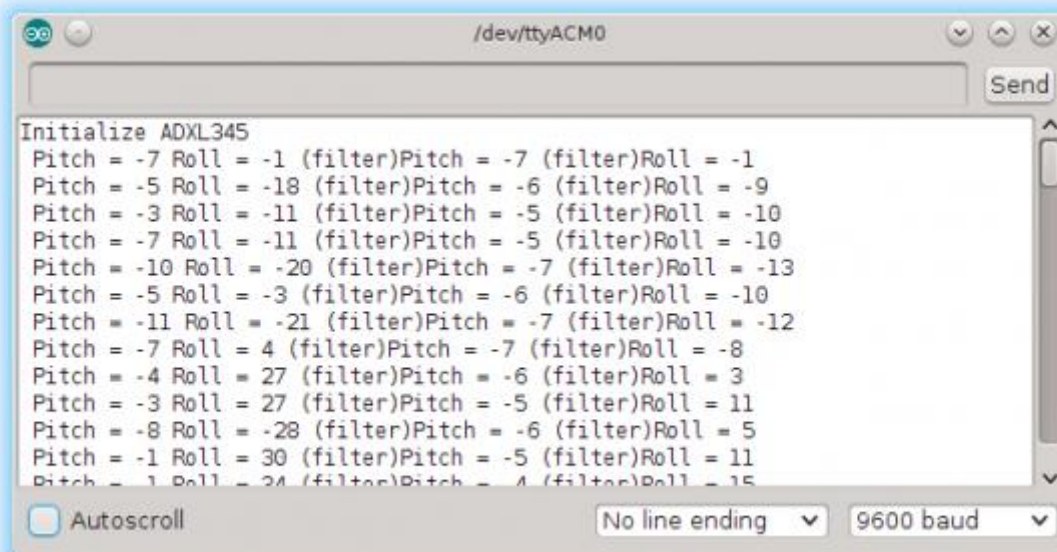
```

```

14. Serial.println("Nie odnaleziono ADXL345, sprawdz podlaczenie!");
15. delay(500);
16. }
17.
18. // Wybor zakresu pomiarowego
19. accelerometer.setRange(ADXL345_RANGE_16G);
20. }
21.
22. void loop(void)
23. {
24. // Odczyt znormalizowanych wartosci
25. Vector norm = accelerometer.readNormalize();
26.
27. // Filtr dolnoprzepustowy (0.1 - 0.9)
28. Vector filtered = accelerometer.lowPassFilter(norm, 0.15);
29.
30. // Obliczenie Pitch & Roll z danych znormalizowanych
31. int pitch = -
    (atan2(norm.XAxis, sqrt(norm.YAxis*norm.YAxis + norm.ZAxis*norm.ZAxis))*180.0)/M_PI;
32. int roll = (atan2(norm.YAxis, norm.ZAxis)*180.0)/M_PI;
33.
34. // Obliczenie Pitch & Roll z danych filtra dolnoprzepustowego
35. int fpitch = -
    (atan2(filtered.XAxis, sqrt(filtered.YAxis*filtered.YAxis +filtered.ZAxis*filtered.ZAxis))*180.0)/
    M_PI;
36. int froll = (atan2(filtered.YAxis, filtered.ZAxis)*180.0)/M_PI;
37.
38. // Wyświetlenie wartosci
39. Serial.print(" Pitch = ");
40. Serial.print(pitch);
41. Serial.print(" Roll = ");
42. Serial.print(roll);
43.
44. // Wyświetlenie wartosci (z filtra)
45. Serial.print(" (filter)Pitch = ");
46. Serial.print(fpitch);
47. Serial.print(" (filter)Roll = ");
48. Serial.print(froll);
49. Serial.println();
50.
51. delay(200);

```

Wynik po uruchomieniu programu:

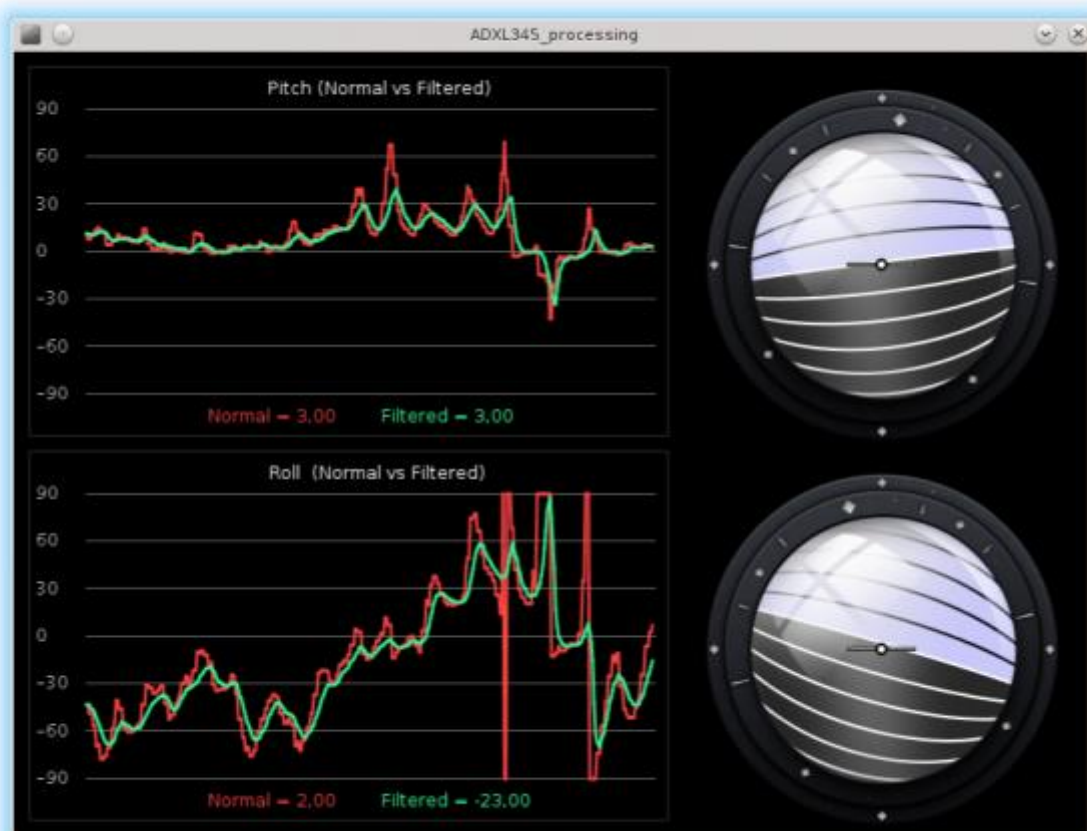


The screenshot shows a terminal window titled "/dev/ttyACM0" with a "Send" button. The output displays the initialization of the ADXL345 sensor followed by a series of Pitch and Roll readings. Each reading is shown in two forms: "Normal" (raw data) and "Filtered" (data after low-pass filtering). The data is as follows:

Pitch	Roll	Pitch (filter)	Roll (filter)
-7	-1	-7	-1
-5	-18	-6	-9
-3	-11	-5	-10
-7	-11	-5	-10
-10	-20	-7	-13
-5	-3	-6	-10
-11	-21	-7	-12
-7	4	-7	-8
-4	27	-6	3
-3	27	-5	11
-8	-28	-6	5
-1	30	-5	11
1	24	-4	15

At the bottom of the terminal, there is an "Autoscroll" checkbox and two dropdown menus: "No line ending" and "9600 baud".

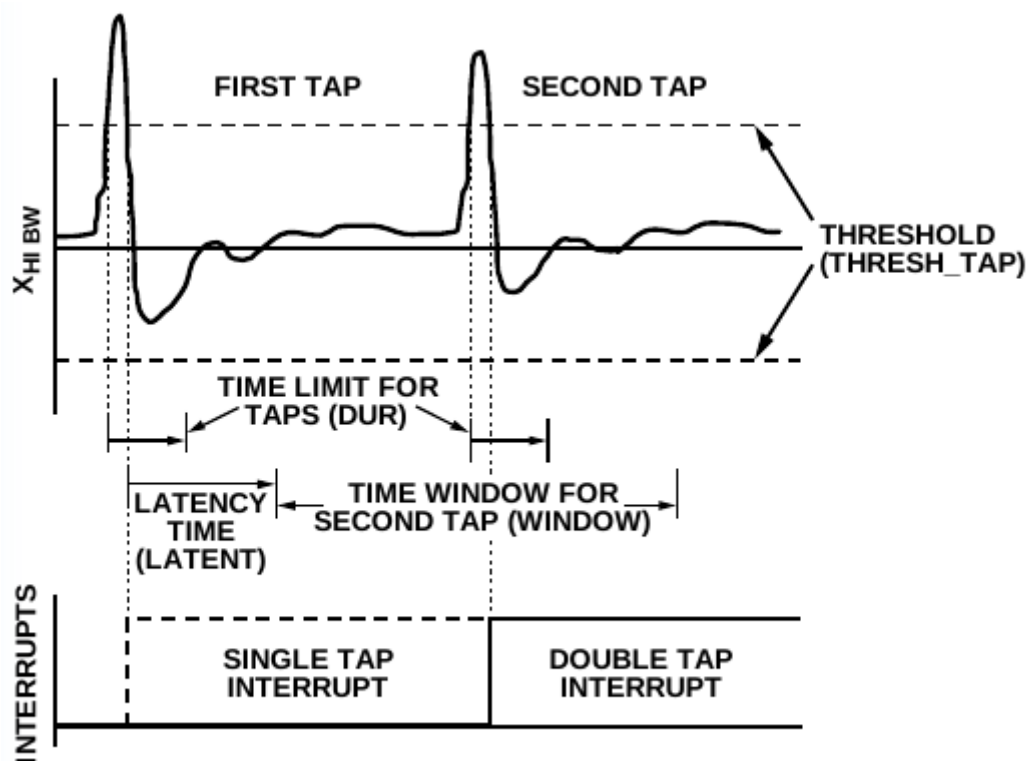
W repozytorium **Git** znajdziecie jeszcze wersję programu dla [Processingu](#) oraz [program wizualizacji danych](#). Można za jego pomocą porównać obliczenia bez i z wykorzystaniem **filtra dolnoprzepustowego**



**Detekcja nieaktywności, swobodnego spadania oraz stuknięcia**

Jak wspominałem wcześniej, **ADXL345** ma możliwość sygnalizowania o szeregu zdarzeń za pomocą przerw **INT1** lub **INT2**. Możemy zatem przechwycić moment wejścia w stan **aktywności**, **nieaktywności**, **swobodnego spadania** oraz **stuknięcia** - również podwójnego. Aby to uzyskać, należy ustawić szereg parametrów, które pozwolą układowi na odpowiednią interpretację wyników, czyli wartości granicznych i czasów trwania określonych fragmentów charakterystycznych. **Stuknięcie (tap)** i **podwójne stuknięcie (double tap)**.

Aby łatwiej zrozumieć ten proces, spójrzmy na poniższy wykres:



**Figure 46. Tap Interrupt Function with Valid Single and Double Taps**

Pierwszą wartością, która nas interesuje jest **THRESH\_TAP**. Jest to poziom wartości przyspieszenia, jakie musi wystąpić, aby zdarzenie zostało zakwalifikowane jako stuknięcie.

Parametr **DUR** określa czas trwania charakterystyki odpowiadającej przy stuknięciu. Jest to więc czas, w którym układ pomiarowy osiągnie wartość **THRESH\_TAP** oraz nastąpi reakcja działającej siły sprężystej odbicia (*dolna część*). Czas ten musi być odpowiednio krótki na tyle, aby zmieścić się w nim cały proces stuknięcia.

Parametr **LATENT** określa natomiast czas od momentu spadku wartości przyspieszenia poniżej wartości **THRESH\_TAP** do momentu całkowitej stabilizacji, czyli do momentu w którym nie działa na niego żadna siła powodująca przyspieszenie.

Ostatnim parametrem jest **WINDOW**, który określa czas, w którym może wystąpić kolejne stuknięcie po pełnym zakończeniu charakterystyki pierwszego. Należy pamiętać, że czas ten musi objąć pełny przebieg drugiego stuknięcia od momentu stabilizacji pierwszego.

Na podstawie tych wartości generowane jest przerwanie informujące o wykrytym zdarzeniu.

Parametry te określamy funkcjami: **setTapThreshold()**, **setTapDuration()**, **setDoubleTapLatency()** oraz **setDoubleTapWindow()**. Biblioteka pozwala również określić, względem której osi ma wykrywać wystąpienie stuknięcia.

#### **Przejęcie w stan aktywności i nieaktywności**

Powyższy opis stanowi przypadek stuknięć, jednak analogicznymi wartościami posługujemy się w przypadku detekcji aktywności i nieaktywności układu. Do określenia czy nastąpiła aktywność,



definiujemy jeden parametr **THRESH\_ACT** - jeśli wartość przyspieszenia przekroczy tą wartość - traktowane jest jako przejście w stan aktywności. Badanie nieaktywności odbywa się poprzez podanie dwóch parametrów: **THRESH\_INACT** oraz **TIME\_INACT**. Pierwszy określa poziom przyspieszenia poniżej którego musi znaleźć się odczytana wartość, natomiast drugi definiuje jak długo taki stan musi się utrzymywać, aby zdarzenie zostało zakwalifikowane jako przejście w stan nieaktywności. Jak łatwo wywnioskować, nie musi być to wcale przejście w stan całkowitego spoczynku, a może być określeniem granic braku czułości układu na działające przyspieszenia. Parametry te określamy za pomocą funkcji: **setActivityThreshold()**, **setInactivityThreshold()** oraz **setTimeInactivity()**. Tutaj także mamy również możliwość ustawienia, które osie układu będą badane.

### Swobodne spadanie

Bardzo ciekawą funkcją jest możliwość wykrycia procesu swobodnego spadania układu. Tutaj również posłużymy się dwoma parametrami: **THRESH\_FF** oraz **TIME\_FF**. Pierwszym z nich, jak się zapewne domyślicie, jest wartość przyspieszenia jaka musi występować podczas swobodnego spadania, drugi natomiast określa czas, jaki minimalny czas musi upłynąć od osiągnięcia poziomu **THRESH\_FF**, aby zakwalifikować to zdarzenie jako swobodne spadanie.

Parametry **THRESH\_FF** oraz **TIME\_FF**, przekazujemy za pomocą funkcji: **setFreeFallThreshold()** oraz **setFreeFallDuration()**.

### Jak odczytać zdarzenia pochodzące z przerwai?

Praktycznie wystarczy ustawić, które przerwanie będzie obsługiwało nasze zdarzenia (**INT1** lub **INT2**) oraz odczytać odpowiedni rejestr za pomocą funkcji **readActivites()**. Ważne jest, aby funkcja ta została wywołana po odczycie danych za pomocą funkcji **readNormalize()** lub **readRaw()**. Nie zapominajmy również o konfiguracji powyżej opisanych parametrów. W repozytorium **Git** znajdziecie [oddzielne przykłady](#) dla poszczególnych zdarzeń.

```
1. #include <Wire.h>
2. #include <ADXL345.h>
3.
4. ADXL345 accelerometer;
5.
6. int RedPin = 4;
7. int GreenPin = 3;
8. int BluePin = 2;
9.
10. long RedTime;
11. long GreenTime;
12. long BlueTime;
13. long DTime;
14.
15. void setup(void)
16. {
17.   Serial.begin(9600);
18.
19.   pinMode(RedPin, OUTPUT);
20.   pinMode(BluePin, OUTPUT);
21.   pinMode(GreenPin, OUTPUT);
22.   digitalWrite(RedPin, LOW);
23.   digitalWrite(BluePin, LOW);
24.   digitalWrite(GreenPin, LOW);
```

```

25.
26. // Inicjalizacja ADXL345
27. Serial.println("Inicjalizacja ADXL345");
28. if (!accelerometer.begin())
29. {
30.   Serial.println("Nie odnaleziono ADXL345, sprawdź podłączenie!");
31.   delay(500);
32. }
33.
34. // Wartości dla wykrycia swobodnego spadania
35. accelerometer.setFreeFallThreshold(0.35); // 0.35 g
36. accelerometer.setFreeFallDuration(0.1); // 0.10 s
37.
38. // Wartości dla wykrycia aktywności i jego braku
39. accelerometer.setActivityThreshold(1.2); // 1.20 g
40. accelerometer.setInactivityThreshold(1.2); // 1.20 g
41. accelerometer.setTimeInactivity(5); // 5.00 s
42.
43. // Badanie aktywności i jego braku we wszystkich osiach
44. accelerometer.setActivityXYZ(1);
45. accelerometer.setInactivityXYZ(1);
46.
47. // Badanie stuknieć tylko dla osi Z
48. accelerometer.setTapDetectionX(0); // Nie sprawdzamy osi X
49. accelerometer.setTapDetectionY(0); // Nie sprawdzamy osi Y
50. accelerometer.setTapDetectionZ(1); // Uwzględniamy jedynie os Z
51.
52. // Wartości dla wykrywania stuknieć
53. accelerometer.setTapThreshold(2.5); // 2.50 g
54. accelerometer.setTapDuration(0.02); // 0.02 s
55. accelerometer.setDoubleTapLatency(0.10); // 0.10 s
56. accelerometer.setDoubleTapWindow(0.30); // 0.30 s
57.
58. // Wybieramy przerwanie INT1
59. accelerometer.useInterrupt(ADXL345_INT1);
60. }
61.
62. void loop(void)
63. {
64.   long time = micros();
65.
66.   // Gaszenie zapalonych diod po upływie danego czasu od zapalenia
67.   if ((time - RedTime) > 300000) digitalWrite(RedPin, LOW);
68.   if ((time - BlueTime) > 300000) digitalWrite(BluePin, LOW);
69.
70.   // Opoznienie przed odczytem (poprawia wyniki) i odczytanie pomiaru
71.   delay(50);
72.

```

```

73. Vector norm = accelerometer.readNormalize();
74.
75. // Odczytanie aktywnosci
76. Activites activ = accelerometer.readActivites();
77.
78. // Jesli wykryto swobodne spdanie - mrugaj dioda
79. if (activ.isFreeFall)
80. {
81.   for (int i = 0; i <= 4; i++)
82.   {
83.     digitalWrite(RedPin, HIGH);
84.     digitalWrite(BluePin, HIGH);
85.     delay(100);
86.     digitalWrite(RedPin, LOW);
87.     digitalWrite(BluePin, LOW);
88.     delay(100);
89.   }
90.
91.   delay(200);
92.
93.   return;
94. }
95.
96. // Jesli wykryto podwojne stykniecie zapal czerwona
97. if (activ.isDoubleTap)
98. {
99.   digitalWrite(RedPin, HIGH);
100.     RedTime = micros();
101.   } else
102.   if (activ.isTap) // Jesli pojedyncze stuknienie zapal niebieska
103.   {
104.     digitalWrite(BluePin, HIGH);
105.     BlueTime = micros();
106.   }
107.
108.   // Jesli nieaktywny zapal zielona diode
109.   if (activ.isInactivity)
110.   {
111.     digitalWrite(GreenPin, HIGH);
112.     GreenTime = micros();
113.   }
114.
115.   // Jesli aktywny zgas zielona diode
116.   if (activ.isActivity)
117.   {
118.     digitalWrite(GreenPin, LOW);
119.   }
120. }

```

## Demo

### Materiały dodatkowe

Biblioteka **ADXL345**: <https://github.com/jarzebski/Arduino-ADXL345>

Dokumentacja techniczna: <http://www.jarzebski.pl/datasheets/ADXL345.pdf>

Nota aplikacyjna AN3461: <http://www.jarzebski.pl/datasheets/AN3461.pdf>

## 3-osiowy magnetometr HMC5883L

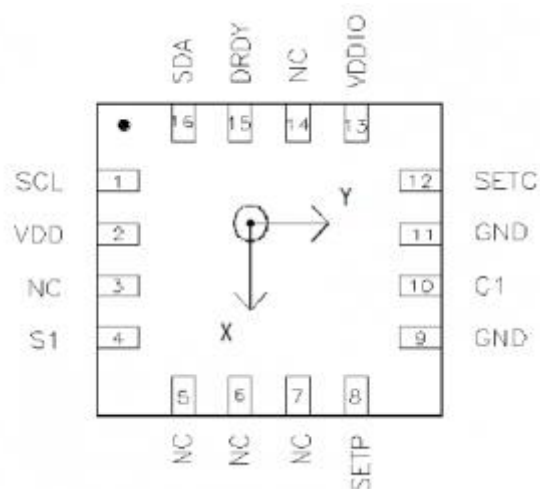
**HMC5883L** jest cyfrowym, 3-osiowym magnetometrem pozwalającym na pomiar szerokiego zakresu wielkości pola magnetycznego Ziemi wynoszącego  $\pm 8$  gaussa. Jego rozdzielczość 12 bitów umożliwia pomiar z dokładnością do 2 miligausów przy poborze prądu zaledwie 100µA. HMC5883L komunikuje się z mikrokontrolerem za pomocą szyny I<sup>2</sup>C z maksymalną częstotliwością pomiarów wynoszącą 75 Hz w trybie ciągłym.



Zakres pomiarowy	Rozdzielczość	Wzmocnienie
± 0.88 Ga	0.73 mG	1370
± 1.3 Ga	0.92 mG	1090
± 1.9 Ga	1.22 mG	820
± 2.5 Ga	1.52 mG	660
± 4 Ga	2.27 mG	440
± 4.7 Ga	2.56 mG	390
± 5.6 Ga	3.03 mG	330
± 8.1 Ga	4.35 mG	230

**HMC5883L** pozwala na osiągnięcie częstotliwości pomiarów nawet do 160 Hz, jeśli skorzystamy z trybu pojedynczego pomiaru oraz monitorowania przerwania **DRDY**. Ciekawą możliwością jaką daje ten układ, to wybór ilości próbek (1, 2, 4 lub 8), które podlegają uśrednieniu końcowego wyniku. Napięcie zasilania mieści się w zakresie od 2.0 do 3.6V, dlatego układ również nie toleruje zasilania wyższego (5V) - należy zwrócić szczególną uwagę na to, czy nasz moduł posiada możliwość zasilania takim napięciem. Układ jest zamknięty w obudowie LCC o wymiarach 3mm x 3mm i wysokości 0.91mm.

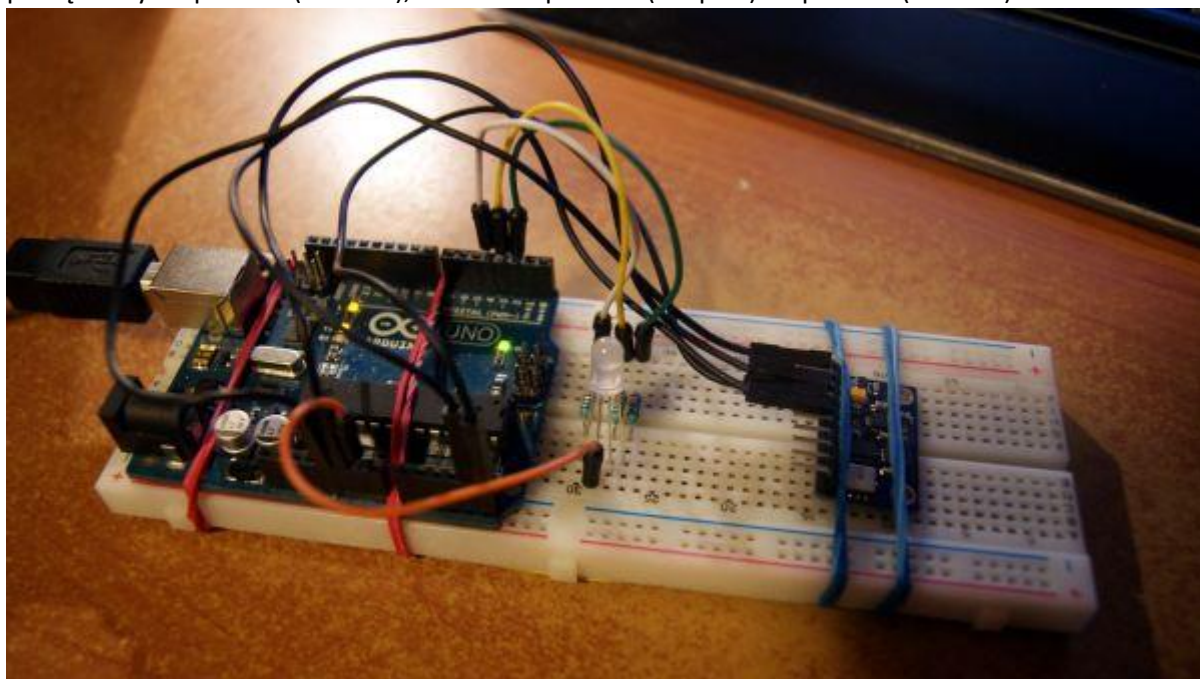




Pełna dokumentacja techniczna: <http://www.jarzebski.pl/datasheets/HMC5883L.pdf>

#### Podłączenie HMC5883L do Arduino

W przypadku modułu IMU GY-80, możemy skorzystać z 5V zasilania. Pin oznaczony **SCL** (*adapter*) podłączamy do pinu **A5** (*Arduino*), natomiast pin **SDA** (*adapter*) do pinu **A4** (*Arduino*).



Do obsługi modułów z układami **HMC5883L** skorzystamy z przygotowanej na tą okazję biblioteki dla **Arduino**, którą można pobrać z repozytorium **Git**: <https://github.com/jarzebski/Arduino-HMC5883L>

#### Prosty przykład

Pierwszym przykładem będzie odczyt surowych wartości oraz znormalizowanych (*mg*):

1. `#include <Wire.h>`
2. `#include <HMC5883L.h>`
- 3.
4. `HMC5883L compass;`
- 5.
6. `void setup()`
7. `{`
8. `Serial.begin(9600);`

```

9.
10. // Inicjalizacja HMC5883L
11. Serial.println("Initialize HMC5883L");
12. while (!compass.begin())
13. {
14.   Serial.println("Nie odnaleziono HMC5883L, sprawdz polaczenie!");
15.   delay(500);
16. }
17.
18. // Ustawienie zakresu pomiarowego
19. // +/- 0.88 Ga: HMC5883L_RANGE_0_88GA
20. // +/- 1.30 Ga: HMC5883L_RANGE_1_3GA (domyslny)
21. // +/- 1.90 Ga: HMC5883L_RANGE_1_9GA
22. // +/- 2.50 Ga: HMC5883L_RANGE_2_5GA
23. // +/- 4.00 Ga: HMC5883L_RANGE_4GA
24. // +/- 4.70 Ga: HMC5883L_RANGE_4_7GA
25. // +/- 5.60 Ga: HMC5883L_RANGE_5_6GA
26. // +/- 8.10 Ga: HMC5883L_RANGE_8_1GA
27. compass.setRange(HMC5883L_RANGE_1_3GA);
28.
29. // Ustawienie trybu pracy
30. // Uspienie: HMC5883L_IDLE
31. // Pojedynczy pomiar: HMC5883L_SINGLE
32. // Ciagly pomiar: HMC5883L_CONTINOUS (domyslny)
33. compass.setMeasurementMode(HMC5883L_CONTINOUS);
34.
35. // Ustawienie czestotliwosci pomiarow
36. // 0.75Hz: HMC5883L_DATARATE_0_75HZ
37. // 1.50Hz: HMC5883L_DATARATE_1_5HZ
38. // 3.00Hz: HMC5883L_DATARATE_3HZ
39. // 7.50Hz: HMC5883L_DATARATE_7_50HZ
40. // 15.00Hz: HMC5883L_DATARATE_15HZ (domyslny)
41. // 30.00Hz: HMC5883L_DATARATE_30HZ
42. // 75.00Hz: HMC5883L_DATARATE_75HZ
43. compass.setDataRate(HMC5883L_DATARATE_15HZ);
44.
45. // Liczba usrednionych probek
46. // 1 probka: HMC5883L_SAMPLES_1 (domyslny)
47. // 2 probki: HMC5883L_SAMPLES_2
48. // 4 probki: HMC5883L_SAMPLES_4
49. // 8 probki: HMC5883L_SAMPLES_8
50. compass.setSamples(HMC5883L_SAMPLES_1);
51. }
52.
53. void loop()
54. {
55.   // Pobranie pomiarow surowych
56.   Vector raw = compass.readRaw();

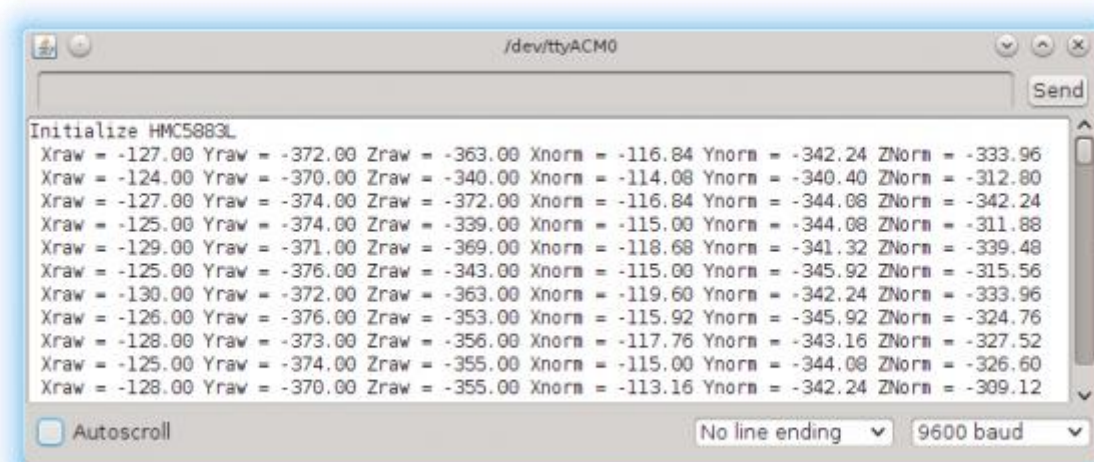
```

```

57.
58. // Pobranie pomiarow znormalizowanych
59. Vector norm = compass.readNormalize();
60.
61. // Wyświetlenie wyników
62. Serial.print(" Xraw = ");
63. Serial.print(raw.XAxis);
64. Serial.print(" Yraw = ");
65. Serial.print(raw.YAxis);
66. Serial.print(" Zraw = ");
67. Serial.print(raw.ZAxis);
68. Serial.print(" Xnorm = ");
69. Serial.print(norm.XAxis);
70. Serial.print(" Ynorm = ");
71. Serial.print(norm.YAxis);
72. Serial.print(" ZNorm = ");
73. Serial.print(norm.ZAxis);
74. Serial.println();
75.
76. delay(100);
77. }

```

Wynik działania



### HMC5883L jako kompas cyfrowy

Znajomość wartości pola magnetycznego Ziemi (*wektor X i wektor Y*) pozwala na określenie bieżącego kierunku południka magnetycznego, a tym samym uzyskanie cyfrowego kompasu.

Kierunek południka magnetycznego można obliczyć z prostej zależności:

**kierunek\_pomiaru (rad) = atan( wektor\_y, wektor\_x );**

Aby poprawnie wyznaczyć kierunek, konieczne jest również uwzględnienie czynnika błędu **-deklinacji magnetycznej**. Deklinacja magnetyczna spowodowana jest zarówno położeniem bieguna magnetycznego Ziemi w innym miejscu niż biegun geograficzny oraz zróżnicowanymi warunkami magnetycznymi w miejscu pomiaru (*np. poprzez występowaniem dużej ilości rud żelaza*). Warto zauważyć, że deklinacja magnetyczna jest parametrem zmiennym w czasie, bowiem biegun

magnetyczny Ziemi stale się przemieszcza. Wartość aktualnej deklinacji magnetycznej znajdziemy na specjalnych mapach magnetycznych, a także na mapach nawigacyjnych.

Na szczęście żyjemy w czasach **Internetu** i odpowiednią mapę znajdziemy pod adresem: <http://magnetic-declination.com/>

Deklinacja magnetyczna dla Bytomia



Jak widzimy, deklinacja magnetyczna dla mojej lokalizacji wynosi **plus 4 stopnie i 26 minut** (wschód). Wartość tą musimy przeliczyć na radiany:

**kąt\_deklinacji = (stopnie + (minuty / 60.0)) / (180 / Pi);**

**kąt\_deklinacji = (4.0 + (26.0 / 60.0)) / (180 / Pi);**

Obliczoną wartość kąta deklinacji dodajemy (wynik *POSITIVE*) lub odejmujemy (wynik *NEGATIVE*) od wartości zmierzonej z magnetometru:

**kierunek = kierunek\_pomiaru ± kąt\_deklinacji**

Jeśli nie znamy kąta deklinacji, możemy przyjąć wartość zero. W następnej kolejności musimy zadbać o to, aby otrzymany wynik kierunku bieguna magnetycznego mieścił się w **zakresie  $0\pi - 2\pi$**  (*chyba, że chcemy mieć kompas, który pokazuje na przykład  $370^\circ$  zamiast  $10^\circ$* ).

jeśli **kierunek < 0** to **dodajemy do niego  $2\pi$**

jeśli **kierunek >  $2\pi$**  to **odejmujemy od niego  $2\pi$**

Teraz możemy zamienić już radiany na stopnie:

**kierunek (deg) = kierunek (rad) \* (180 /  $\pi$ )**

Istotnym problemem (*brzydką cechą*) magnetometru **HMC5883L** jest nierównomierny pomiar pola magnetycznego w zakresie od  $1^\circ \div 180^\circ$  oraz od  $180^\circ \div 360^\circ$ . Dla pierwszego przedziału nasz magnetometr będzie generował przekłamane wyniki od  $1^\circ \div 240^\circ$ , natomiast dla drugiego od  $240^\circ \div 360^\circ$ . Można w łatwy sposób to skorygować funkcją **map()** (*patrz poniższy przykład programu*). Do pełni szczęścia możemy jeszcze wygładzić wskazania naszego kompasu, ustawiając jego reakcję na zmianę o  $3^\circ$ .

Nasz program przedstawia się więc następująco:

1. `#include <Wire.h>`
2. `#include <HMC5883L.h>`
- 3.
4. `HMC5883L compass;`
- 5.
6. `int previousDegree;`
- 7.
8. `void setup()`
9. `{`
10. `Serial.begin(9600);`

```

11.
12. // Inicjalizacja HMC5883L
13. Serial.println("Initialize HMC5883L");
14. while (!compass.begin())
15. {
16.   Serial.println("Nie odnaleziono HMC5883L, sprawdź połączenie!");
17.   delay(500);
18. }
19.
20. // Ustawienie zakresu pomiarowego
21. compass.setRange(HMC5883L_RANGE_1_3GA);
22.
23. // Ustawienie trybu pracy
24. compass.setMeasurementMode(HMC5883L_CONTINUOUS);
25.
26. // Ustawienie częstotliwości pomiarów
27. compass.setDataRate(HMC5883L_DATARATE_15HZ);
28.
29. // Liczba усrednionych probek
30. compass.setSamples(HMC5883L_SAMPLES_4);
31. }
32.
33. void loop()
34. {
35.   // Pobranie wektorów znormalizowanych
36.   Vector norm = compass.readNormalize();
37.
38.   // Obliczenie kierunku (rad)
39.   float heading = atan2(norm.YAxis, norm.XAxis);
40.
41.   // Ustawienie kąta deklinacji dla Bytomia 4'26E (positive)
42.   // Formula: (deg + (min / 60.0)) / (180 / M_PI);
43.   float declinationAngle = (4.0 + (26.0 / 60.0)) / (180 / M_PI);
44.   heading += declinationAngle;
45.
46.   // Korekta katów
47.   if (heading < 0)
48.   {
49.     heading += 2 * PI;
50.   }
51.
52.   if (heading > 2 * PI)
53.   {
54.     heading -= 2 * PI;
55.   }
56.
57.   // Zamiana radianów na stopnie
58.   float headingDegrees = heading * 180/M_PI;

```

```

59.
60. // Output
61. Serial.print(" Heading = ");
62. Serial.print(heading);
63. Serial.print(" Degrass = ");
64. Serial.print(headingDegrees);
65. Serial.println();
66.
67. delay(100);
68.
69. delay(100);
70. }

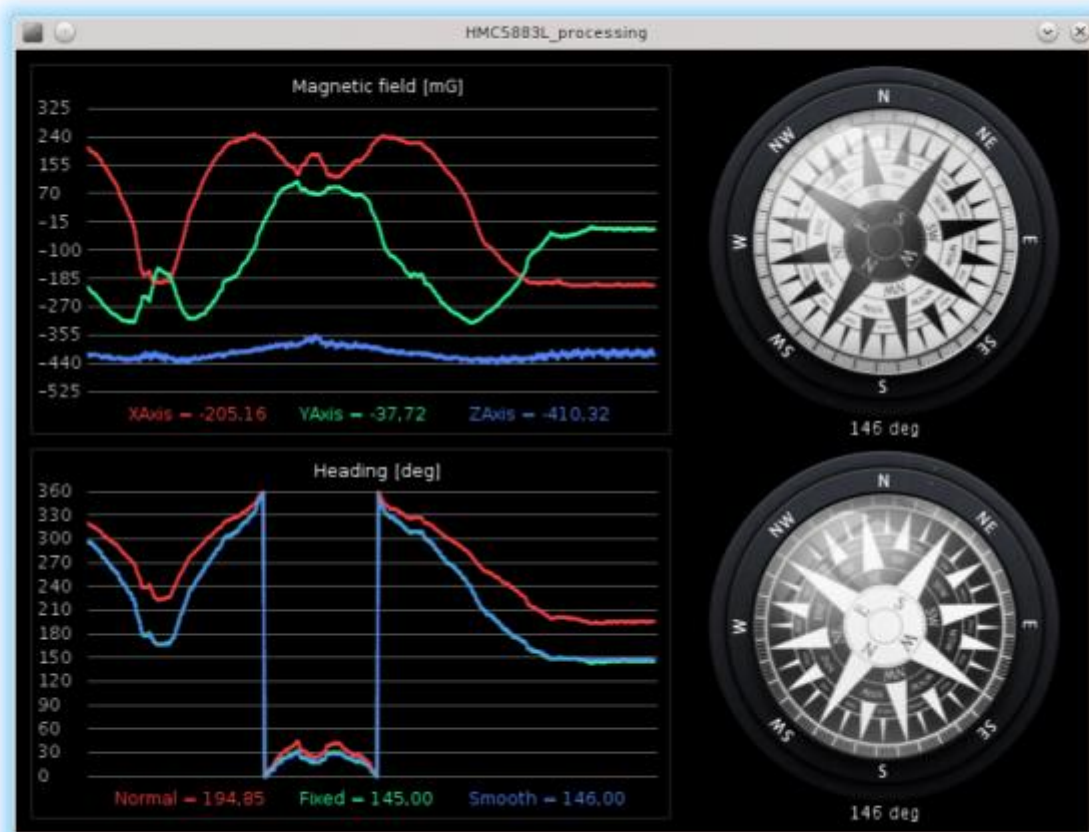
```

Wynik działania:



Program do **processingu** znajdziecie w archwium biblioteki.





### Uwaga na koniec

Niestety kompas jest wrażliwy na wszelkie przechylenia, fałszując kalkulację kierunku południka magnetycznego. Musi się więc znajdować na płaskiej powierzchni, aby wyniki były poprawne. Niepożądany efekt wpływu przechyleń można na szczęście wyeliminować za pomocą akcelerometru (na przykład [ADXL345](#) lub [MPU6050](#)). Jak tego dokonać? Dowiesz się z tego artykułu - [Korepcja przechyleń kompasów cyfrowych](#).

### Demo

#### Materiały dodatkowe

Biblioteka **HMC5883L**: <https://github.com/jarzebski/Arduino-HMC5883L>

Dokumentacja techniczna: <http://www.jarzebski.pl/datasheets/HMC5883L.pdf>



## Czujniki ciśnienia i temperatury BMP085 / BMP180

**BMP085** i **BMP180** to dwa zgodne elektrycznie czujniki ciśnienia atmosferycznego i temperatury firmy **Bosch**.

Charakteryzują się pomiarem ciśnienia w zakresie **od 300 do 1100 hPa**, co daje nam możliwość określenia wysokości **od +9000 do -500 metrów** względem poziomu morza. Głównymi różnicami jakie występują pomiędzy **BMP085** a **BMP180** to rozmiar oraz dokładność pomiaru w trybie wysokiej rozdzielczości pomiaru, w którym przewagę ma nowszy model oznaczony **BMP180**.



	BMP085	BMP180
Zakres mierzonego ciśnienia	300 - 1100 hPa	
Napięcie zasilania	1.8 - 3.6 V	
Pobór prądu	5 µA / pomiar	
Rozmiar	5.0 mm x 5.0 mm	3.6 mm x 3.8 mm
Wysokość	1.2 mm	0.93 mm
Dokładność (tryb niskiego poboru energii)	0.06 hPa (0.5m)	
Dokładność (tryb wysokiej rozdzielczości)	0.03 hPa (0.25m)	0.02 hPa (0.17m)
Temperatura pracy	-40 to +85°C	

Pełna karta katalogowa BMP085: <http://www.jarzebski.pl/datasheets/BMP085.pdf>

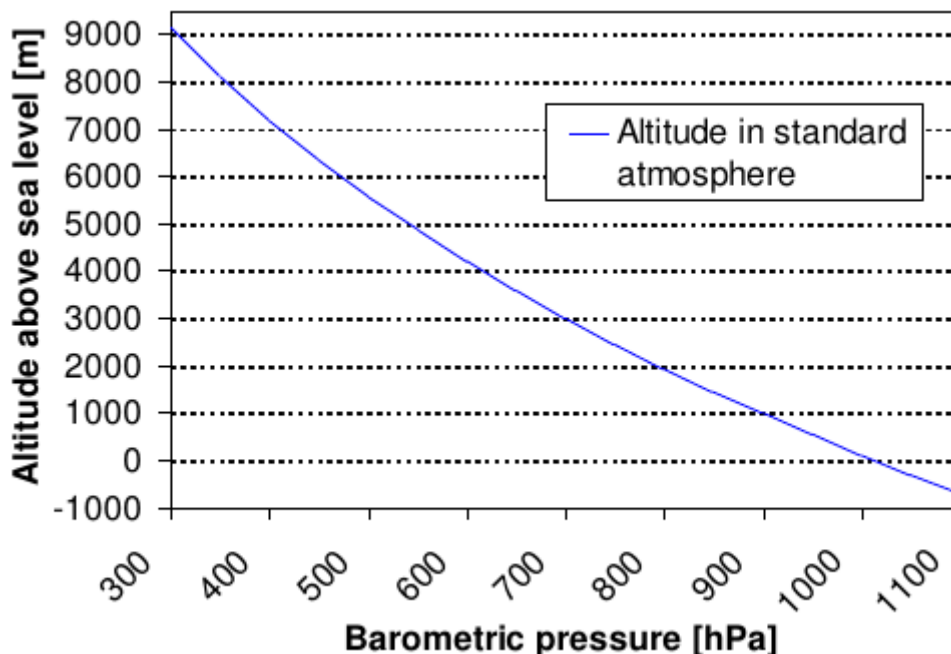
Pełna karta katalogowa BMP180: <http://www.jarzebski.pl/datasheets/BMP180.pdf>

### Obliczanie wysokości na podstawie pomiaru ciśnienia atmosferycznego

Znając ciśnienie jakie panuje na poziomie morza  $p_0$  (np.: 1013.25 hPa) oraz pomiar  $p$ , możemy określić aktualną wysokość, wyliczając ją z poniższego wzoru.

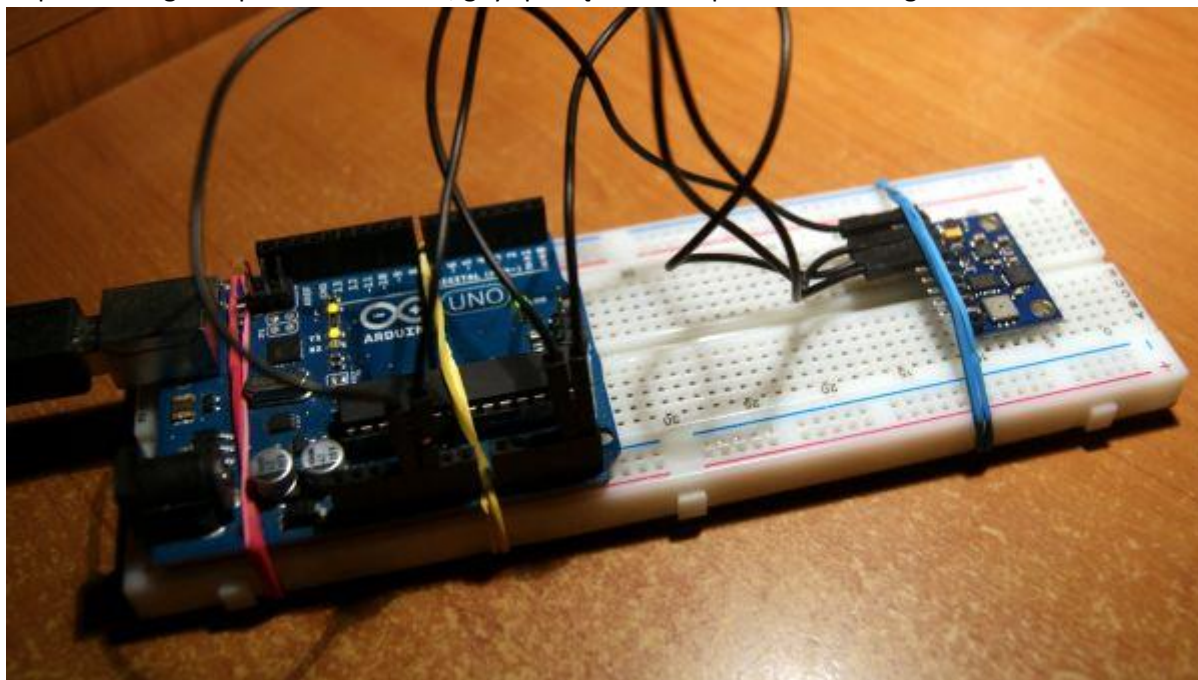
$$\text{altitude} = 44330 * \left( 1 - \left( \frac{p}{p_0} \right)^{\frac{1}{5.255}} \right)$$

Zależność pomiędzy wysokością a ciśnieniem przedstawia poniższa charakterystyka, z której wynika, że zmiana ciśnienia  $\Delta p = 1 \text{ hPa}$  odpowiada wysokości  $\Delta h = 8.43 \text{ m}$ . Natomiast zmiana wysokości  $\Delta h = 10 \text{ m}$  odpowiada zmianie ciśnienia  $\Delta p = 1.2 \text{ hPa}$



#### Połączenie z Arduino

Na rynku istnieje kilka gotowych modułów z czujnikami **BMP085** oraz **BMP180**. Różnią się one przede wszystkim poziomem napięcia zasilania, które typowo wynosi **3.3V**. W przypadku posiadanych przeze mnie modułów **IMU GY-80** oraz **IMU GY-87** zasilanie może być zarówno **5V** jak i **3.3V**. Jeśli zdecydujemy się na zasilanie **5V** należy zwrócić szczególną uwagę na podłączenie do odpowiedniego **5V** pinu modułu **IMU**, gdyż podłączenie do pinu **3.3V** może go uszkodzić.



Pin oznaczony **SCL** (*adapter*) podłączamy do pinu **A5** (*Arduino*), natomiast pin **SDA** (*adapter*) do pinu **A4** (*Arduino*).

Nie zapomnijmy również o masie **GND**.

### Przykładowy program

Na szczęście w przypadku czujników **BMP085** i **BMP180** możemy posłużyć się gotową biblioteką przygotowaną przez **Adafruit**: <https://github.com/adafruit/Adafruit-BMP085-Library> (lub mirror [BMP085.zip](#)). Bibliotekę należy rozpakować do katalogu **libraries**.

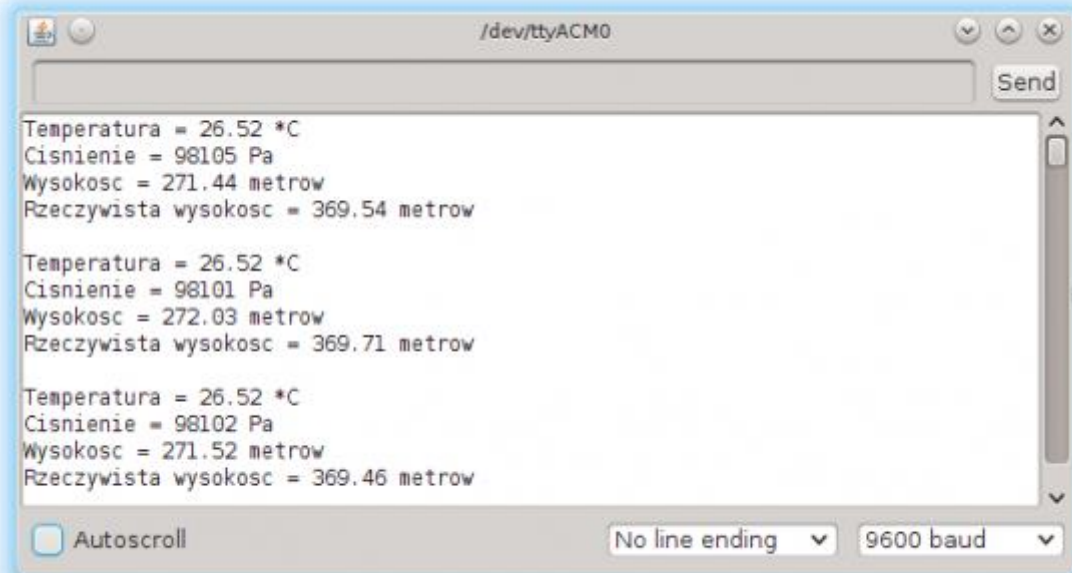
```
1. #include <Wire.h>
2. #include <Adafruit_BMP085.h>
3.
4. Adafruit_BMP085 bmp;
5.
6. void setup()
7. {
8.   Serial.begin(9600);
9.
10.  // Jako parametr mozemy podac dokladnosc - domyslnie 3
11.  // 0 - niski pobór energii - najszybszy pomiar
12.  // 1 - standardowy pomiar
13.  // 2 - wysoka precyzja
14.  // 3 - super wysoka precyzja - najwolniejszy pomiar
15.  if (!bmp.begin())
16.  {
17.    Serial.println("Nie odnaleziono czujnika BMP085 / BMP180");
18.    while (1) {}
19.  }
20. }
21.
22. void loop()
23. {
24.  // Odczytujemy temperaturę
25.  Serial.print("Temperatura = ");
26.  Serial.print(bmp.readTemperature());
27.  Serial.println(" *C");
28.
29.  // Odczytujemy ciśnienie
30.  Serial.print("Ciśnienie = ");
31.  Serial.print(bmp.readPressure());
32.  Serial.println(" Pa");
33.
34.
35.  // Obliczamy wysokosc dla domyslnego cisnienia przy poziomie morza
36.  // p0 = 1013.25 millibar = 101325 Pascal
37.  Serial.print("Wysokosc = ");
38.  Serial.print(bmp.readAltitude());
39.  Serial.println(" metrow");
40.
41.  // Jesli znamy aktualne cisnienie przy poziomie morza,
42.  // mozemy dokladniej wyliczyc wysokosc, padajac je jako parametr
43.  Serial.print("Rzeczywista wysokosc = ");
44.  Serial.print(bmp.readAltitude(102520));
```

```

45. Serial.println(" metrow");
46.
47. Serial.println();
48. delay(500);
49. }

```

Wynik działania



### Wizualizacja

Spróbujmy jeszcze pokazać na wykresie poszczególne parametry za pomocą processingu. W tym przypadku skorzystamy nieco z zmodyfikowanego programu dla **Arduino**. Wyliczymy również wysokość samodzielnie, ponieważ funkcja **readAltitude()** ponownie będzie odczytywała ciśnienie do wyliczeń, które już mamy.

```

1. #include <Wire.h>
2. #include <Adafruit_BMP085.h>
3.
4. Adafruit_BMP085 bmp;
5.
6. float sealevelPressure;
7.
8. void setup()
9. {
10. Serial.begin(9600);
11.
12. if (!bmp.begin())
13. {
14. Serial.println("Nie odnaleziono czujnika BMP085 / BMP180");
15. while (1) {}
16. }
17. }
18.
19. void loop()

```

```

20. {
21.   // Odczytujemy temperature i cisnienie
22.   float temp = bmp.readTemperature();
23.   float pressure = bmp.readPressure();
24.
25.   // Sami wyliczamy wysokosc
26.   sealevelPressure = 101325;
27.   float altitude = 44330 * (1.0 - pow(pressure / sealevelPressure, 0.1903));
28.
29.   // Wyczucamy dane na port szeregowy
30.   Serial.print("D:");
31.   Serial.print(temp);
32.   Serial.print(":");
33.   Serial.print(pressure);
34.   Serial.print(":");
35.   Serial.print(altitude);
36.   Serial.println();
37. }

```

Program wizualizacyjny:

```

1.  import processing.serial.*;
2.
3.  Serial myPort;
4.
5.  boolean hasData = false;
6.
7.  int actualSample = 0;
8.  int maxSamples = 300;
9.  int sampleStep = 2;
10.
11. float[] tempValues = new float[maxSamples+1];
12. int minTemp = 0;
13. int maxTemp = 0;
14.
15. float[] pressureValues = new float[maxSamples+1];
16. int minPressure = 0;
17. int maxPressure = 0;
18.
19. float[] altitudeValues = new float[maxSamples+1];
20. int minAltitude = 0;
21. int maxAltitude = 0;
22.
23. void setup ()
24. {
25.   size(670, 750);
26.   myPort = new Serial(this, Serial.list()[0], 9600);
27.   myPort.bufferUntil(10);
28.   background(0);
29. }

```

```

30.
31. void drawChart(String
    title, float[] data, int minValue, int maxValue, int x, int y, int h, int ls, int fs)
32. {
33.   strokeWeight(1);
34.   noFill();
35.   stroke(50,50,50);
36.   rect(x, y, (maxSamples*sampleStep)+50, h+50);
37.
38.   strokeWeight(1);
39.   stroke(90,90,90);
40.
41.   for (float t = minValue; t <= maxValue; t = t + ls)
42.   {
43.     float line = map(t, minValue, maxValue, 0, h);
44.     line(x+40, y+h-line+16, x+(maxSamples*sampleStep)+40, y+h-line+16);
45.     fill(200, 200, 200);
46.     textSize(fs);
47.     text(int(t), 5+x, h-line+20+y);
48.   }
49.
50.   textSize(14);
51.   String title2 = title + " " + nf(data[actualSample-1], 0, 2);
52.   text(title2, ((maxSamples*sampleStep)/2)-(textWidth(title2)/2)+40, h+40+y);
53.
54.   strokeWeight(2);
55.   stroke(204, 102, 0);
56.
57.   for (int i = 1; i < actualSample; i++)
58.   {
59.     float v0 = map(data[i-1], minValue, maxValue, 0, h);
60.     float v1 = map(data[i], minValue, maxValue, 0, h);
61.     line(((i-1)*sampleStep)+40+x, h-v0+16+y, (i*sampleStep)+40+x, h-v1+16+y);
62.   }
63. }
64.
65. void draw ()
66. {
67.   if (!hasData) return;
68.
69.   background(0);
70.
71.   drawChart("Temperatura", tempValues, minTemp, maxTemp, 10, 10, 100, 1, 14);
72.   drawChart("Cisnienie", pressureValues, minPressure, maxPressure, 10, 200, 200, 50, 10);
73.   drawChart("Wysokosc", altitudeValues, minAltitude, maxAltitude, 10, 490, 200, 1, 12);
74. }
75.
76. void nextSample()

```

```

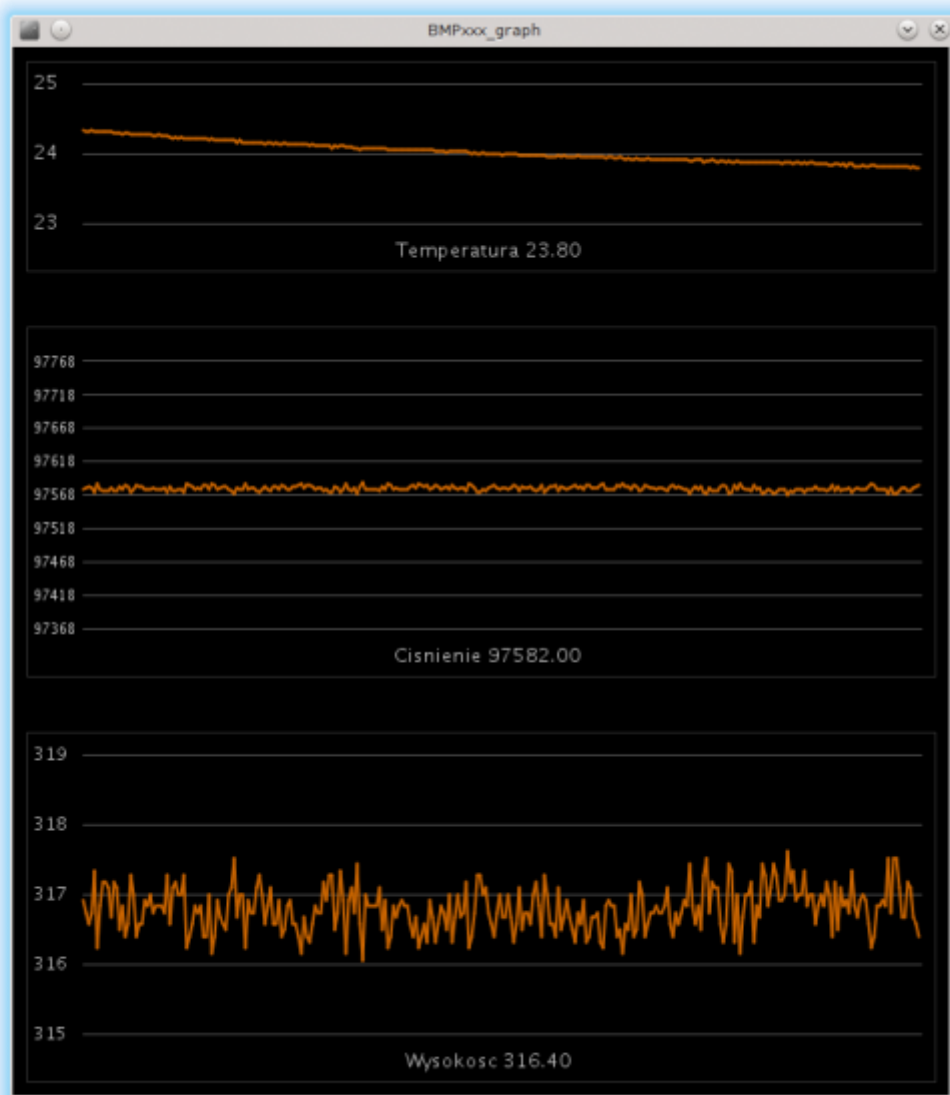
77. {
78.   if (actualSample == maxSamples)
79.   {
80.     float lastTemp = tempValues[maxSamples];
81.     float lastPressure = pressureValues[maxSamples];
82.     float lastAltitude = altitudeValues[maxSamples];
83.
84.     for (int i = 1; i <= (maxSamples-1); i++)
85.     {
86.       tempValues[i-1] = tempValues[i];
87.       pressureValues[i-1] = pressureValues[i];
88.       altitudeValues[i-1] = altitudeValues[i];
89.     }
90.
91.     tempValues[(maxSamples-1)] = lastTemp;
92.     pressureValues[(maxSamples-1)] = lastPressure;
93.     altitudeValues[(maxSamples-1)] = lastAltitude;
94.   } else
95.   {
96.     actualSample++;
97.   }
98. }
99.
100.    void serialEvent (Serial myPort)
101.    {
102.      String inString = myPort.readStringUntil(10);
103.
104.      if (inString != null)
105.      {
106.        inString = trim(inString);
107.        String[] list = split(inString, ':');
108.
109.        String testString = trim(list[0]);
110.
111.        if (list.length != 4) return;
112.
113.        float temp = float(list[1]);
114.        float pressure = float(list[2]);
115.        float altitude = float(list[3]);
116.
117.        if (actualSample == 0)
118.        {
119.          for (int i = 0; i <= maxSamples; i++)
120.          {
121.            tempValues[i] = temp;
122.            pressureValues[i] = pressure;
123.            altitudeValues[i] = altitude;
124.          }

```



```
125.     }
126.
127.     tempValues[actualSample] = temp;
128.     pressureValues[actualSample] = pressure;
129.     altitudeValues[actualSample] = altitude;
130.
131.     maxTemp = floor(max(tempValues))+1;
132.     minTemp = ceil(min(tempValues))-1;
133.
134.     maxPressure = floor(max(pressureValues))+200;
135.     minPressure = ceil(min(pressureValues))-200;
136.
137.     maxAltitude = floor(max(altitudeValues))+2;
138.     minAltitude = ceil(min(altitudeValues))-2;
139.
140.     if (actualSample > 1)
141.     {
142.         hasData = true;
143.     }
144.
145.     nextSample();
146. }
147. }
```

## Wynik działania



O ile pomiar ciśnienia jest w miarę dokładny na potrzeby meteorologiczne, to przeliczona za jego pomocą wysokość może wahać się w zakresie  $\pm 1$  m. Warto również podkreślić fakt, że większa dokładność pomiaru ciśnienia w czujniku **BMP180** rozwiązywana jest programowo, a nie sprzętowo - uśredniany jest 3-krotny pomiar.

### Materiały dodatkowe

Biblioteka dla Arduino: <http://www.jarzebski.pl/arduino/BMP085/BMP085.zip>

Przykłady z powyższego wpisu: [http://www.jarzebski.pl/arduino/BMP085/BMPxxx\\_examples.zip](http://www.jarzebski.pl/arduino/BMP085/BMPxxx_examples.zip)

Pełna karta katalogowa BMP085: <http://www.jarzebski.pl/datasheets/BMP085.pdf>

Pełna karta katalogowa BMP180: <http://www.jarzebski.pl/datasheets/BMP180.pdf>