
# Noszony zestaw modułów inercyjnych DIY

*Digitalizacja ruchów ludzkiego ciała jest niezwykle ważna m.in. w systemach rzeczywistości wirtualnej czy rozszerzonej. Przeniesienie informacji o naturalnych ruchach np. dłoni do komputera, w czasie rzeczywistym, pozwala na zastosowanie tych informacji na przykład do sterowania oprogramowaniem. Pomysł ten nie jest nowy, a systemy pozwalające na śledzenie ruchów są już dostępne na rynku, jednakże daleko im do doskonałości. Większość z nich wykorzystuje ruchome, mechaniczne przetworniki do digitalizacji ruchów (np. czujniki potencjometryczne umieszczone w stawach) – stawiają one pewien mechaniczny opór podczas ruchu i działają na ogół tylko w jednej osi, a znakomita większość ludzkich stawów skręca się, przynajmniej w pewnym stopniu, w dwóch lub więcej osiach. Problem ten rozwiązują moduły inercyjne (IMU). Pozwalają one na śledzenie ruchu obiektu w trójwymiarowej przestrzeni i nie mają wad sensorów mechanicznych.*

W poniższym artykule przedstawimy ciekawy projekt modułu do śledzenia ruchów ciała z wykorzystaniem zestawu noszonych jednostek inercyjnych. Tego rodzaju system pozwala w łatwy sposób śledzić ruch całej postaci i przekazywać go do komputera. System taki w żaden sposób nie wpływa na sam ruch – przyczepione do kończyn sensory nie utrudniają ruchu ani nie ograniczają go do, na przykład, tylko jednego kierunku skrętu. Na **fotografii 1** widzimy zestaw dwóch modułów IMU umieszczonych w nadgarstkach.

Prezentowany w artykule zestaw stworzony został przez Michaela Stengla, doktoranta na politechnice w niemieckim Brunszwiku. Jest on od dawna zafascynowany rzeczywistością wirtualną i rozszerzoną, do której systemy digitalizacji ruchu – *motion capture* – są krytyczne. Inspiracją do stworzenia opisywanego poniżej systemu były analogiczne urządzenia, stosowane m. in. w kinematografii – są one dużo droższe niż przedstawiony poniżej system, jednakże, jak zaznacza sam autor konstrukcji, również o wiele precyzyjniejsze. Wynika to głównie z ogromnego nakładu sił i środków na zaawansowaną kalibrację tego systemu, co pozwala uzyskiwać o wiele więcej z porównywalnych sensorów IMU. Gorsze parametry opisywanego poniżej systemu nie sprawiają jednak, że nie jest on również użyteczny. Mimo mniejszej precyzji i np. braku aktywnej eliminacji dryfu modułów IMU, prezentowany projekt ma wiele ciekawych zastosowań.

## Aplikacje

**Motion Capture** Prezentowane noszone czujniki zapewniają kompletne dane o swojej orientacji, które mogą być rejestrowane i następnie wykorzystywane np. do animowania modelu ciała. Jednakże do pełnego przechwycenia ruchów ciała w przestrzeni 3D konieczne są także dane pozycyjne. Można je zapewnić za pomocą dodatkowego, skalibrowanego czujnika zewnętrznego, np. Microsoft Kinect lub korzystać

z zewnętrznych kamer i znaczników umieszczonych na postaci.

**Animacja awatara dla VR w czasie rzeczywistym** W przypadku animacji awatara w czasie rzeczywistym w systemie rzeczywistości wirtualnej (VR) zestaw noszonych czujników zapewnia informację na temat orientacji ciała. W pozycji siedzącej nie są wymagane żadne inne dane, ponieważ w systemach VR zmiany pozycyjne nie są potrzebne. Jeżeli w danym systemie wymagane byłyby, z jakiegoś powodu, także dane pozycyjne, to wystarczy system noszonych IMU uzupełnić pojedynczym punktem, śledzącym pozycję ciała – na przykład kamerą, śledzącą ruch otoczenia.

**Urządzenie interakcji 3D: mapowanie orientacji i pozycji** Czujnik może być używany jako urządzenie do sterowania interakcją z komputerem. Orientację czujnika można zmapować z dłoni do na przykład menu w komputerze. Względna zmiana pozycji dłoni (uzyskiwana z dwukrotnego całkowania przyspieszenia) mogłaby pozwolić na intuicyjną kontrolę prostych i złożonych interfejsów. Dzięki swojej wielowymiarowości (IMU działa w trzech wymiarach) może pozwolić na zmianę wielu parametrów w aplikacjach takich jak systemy modelowania 3D czy stacje robocze audio.

**Proste gesty do obsługi komputera** Czujnik IMU może być również urządzeniem wskazującym (jak wskaźnik laserowy), można go użyć zamiast na przykład myszy lub klawiatury. Proste gesty wykonywane np. dłonią mogą wywoływać zaprogramowane funkcje, co pozwoli na intuicyjną obsługę komputera.

**Monitorowanie aktywności sportowej** Za pomocą prezentowanego czujnika można również monitorować ruchy ciała, aby poprawiać jakość treningów sportowych, wykonywać poprawnie ćwiczenia etc. Monitorowanie sposobu poruszania się ciała podczas wykonywania ćwiczeń pozwoli poprawić ich wydajność i uniknąć kontuzji. Dane z czujników mogą być rejestrowane, monitorowane i analizowane w czasie rzeczywistym. Autor samodzielnie zaimplementował na przykład symulator wioślarstwa, który wykorzystuje dane z IMU do animowania wioseł w łodzi. Prezentowany tutaj moduł nie jest wodoodporny, więc nie do końca nadaje się do monitorowania aktywności w ramach tej dziedziny sportu, jednakże doskonale oddaje możliwości tego systemu.

**Stabilizacja wideo i robienie zdjęć panoramicznych** Algorytmy stabilizacji filmów wideo bardzo często wykorzystują analizę obrazu do estymacji położenia kamery. W przypadku obrazów niewyraźnych lub z małą ilością szczegółów ocena ułożenia aparatu może się nie powieść. Dane na temat przyspieszenia i orientacji aparatu z sensora IMU mogą zapewnić wsparcie dla tego rodzaju algorytmów, zwiększając ich niezawodność i prędkość działania.

**Rysunek 1. Przykładowa aplikacja modułów IMU – zamontowane na nadgarstkach, pozwalają śledzić ruchy rąk**

Inną aplikacją, która wymaga estymacji położenia aparatu, jest robienie zdjęć panoramicznych. Istnieją już aplikacje korzystające z danych IMU ze smartfona. Większość nowoczesnych telefonów wyposażonych jest w funkcję zdjęć panoramicznych. Czujnik IMU i dowolny aparat mogą pozwolić na podobne działanie.

**Sztuka i performance** Rysowanie linii w przestrzeni 3D za pomocą własnych ruchów ciała jest bardzo ekspresyjną metodą malowania, która nie jest możliwa w przypadku tradycyjnych form sztuki. Czujniki IMU stanowią skuteczne, nowatorskie narzędzie do tworzenia fascynującej sztuki w przestrzeni i czasie. To zupełnie nowe wykorzystanie tego rodzaju technologii, które daje niemalże nieograniczone możliwości.

## Budowa sensorów

Aby zmontować samodzielnie taki sensor, najpierw zgromadzić trzeba potrzebne komponenty – moduły, które po połączeniu i oprogramowaniu przesyłać będą do komputera dane z czujnika IMU. Potrzebne są:

- moduł MPU-6050 z akcelerometrem i żyroskopem,
- moduł HC-05 z transceiverem Bluetooth,
- Arduino Pro Mini,
- ogniwo litowo-jonowe 3,7 V,
- przełącznik, gniazdko do podłączania ładowarki do ogniwa i (najlepiej kolorowe) kable do połączenia modułów ze sobą,
- gumka z rzepem do mocowania modułu,
- klej na ciepło i śrubki do montażu wszystkich elementów.

Montaż i oprogramowanie urządzenia podzielić można na kilka etapów:

1. Konfiguracja płytki z transceiverem Bluetooth
2. Programowanie Arduino Pro Mini
3. Przygotowanie obudowy – druk 3D i montaż pasków
4. Montaż elementów w obudowie i lutowanie połączeń między nimi.

## Konfiguracja transceivera Bluetooth HC-05

W pierwszej kolejności, przed podłączeniem modułu do systemu, musimy go skonfigurować i nadać mu zrozumiałą dla systemu nazwę. Dokonujemy tego, podłączając się do modułu poprzez interfejs UART. Możemy podłączyć do HC-05 konwerter USB-UART TTL albo wykorzystać inny moduł Arduino. Autor skorzystał z drugiej opcji i podłączył do modułu Arduino UNO z odpowiednim szkicem (programem).

Przed skonfigurowaniem HC-05 musimy poznać adres sprzętowy naszego modułu.

**Tabela 1. Lista połączeń Arduino UNO i HC-05 do programowania modułu Bluetooth**

| HC-05 | Arduino UNO |
|---|---|
| TXD | RX |
| RXD | TX |
| VCC | 5.05.2019 |
| GND | GND |
| WakeUp | 3,3 V |

**Tabela 2. Lista połączeń Arduino UNO i Mini Pro do programowania**

| Arduino Mini Pro | Arduino UNO |
|---|---|
| GND | RESET |
| VCC | 3,3 V |
| GND | GND |
| TXS | TX |
| RXI | RX |

**Listing 1. Szkic (kod w Arduino IDE) Programu dla Arduino Pro Mini do obsługi IMU i modułu Bluetooth.**

```
// Biblioteka Arduino Wire jest wtnagana, jeżeli korzystamy z implementacji I2CDEV_ARDUINO_WIRE
// protkołu I2Cdev w I2Cdev.h
#include „Wire.h“

// Biblioteki I2Cdev oraz MPU6050 muszą być zainstalowane jak normalne biblioteki lub trzeba
// dostarczyć pliki .cpp/.h files dla obu klas w ścieżce path projektu
#include „I2Cdev.h“

#include „MPU6050_6Axis_MotionApps20.h“

// Domyślny adres I2C dla tej klasy to 0x68, inny adres podać można jako parametr
// AD0 niski  = 0x68 (domyślnie)
// AD0 wysoki = 0x69
MPU6050 mpu;

#define INT_PIN 2

// wartości do kontroli stanu MPU
bool dmpReady = false;        // Przyjmuje wartość true gdy inicjalizacja DMP zakończy się sukcesem
uint8_t mpuIntStatus;         // Przechowuje status przerwania z MPU
uint8_t devStatus;            // Status returna po każdej operacji (0 = sukces, !0 = błąd)
uint16_t packetSize;          // Spodziewana wielkość pakietu z DMP (domyślnie 42 bity)
uint16_t fifoCount;           // Liczba wszystkich bitów w FIFO
uint8_t fifoBuffer[64]; // Bufor FIFO

// Zmienne orientacji i ruchu
Quaternion q;

typedef union {
 float floatingPoint[4];
 byte binary[16];
} binaryFloat;

// Bufor dla danych wyjściowych
char dataOut[256];

// Funkcja detekcji przerwania
volatile bool mpuInterrupt = false;               // wskazuje gdy pin przerwania MPU ma wysoki stan
void dmpDataReady() {
          mpuInterrupt = true;
}

// Początkowa inicjalizacja
void setup() {
          // Uruchom interfejs I2C
          Wire.begin();

          // inicjalizacja komunikacji szeregowej
          // UWAGA: Zegar 8 MHz lub mniej nie pozwala na uzyskanie odpowiedniej prędkości
          // transmisji w sposób niezawodny, ponieważ taktowanie jej nie zgrywa się z zegarem.
          // W takim wypadku skorzystać trzeba z prędkości 38400 lub niższej, albo zastosować
          // zewnętrzny oscylator kwarcowy dla timera UARTa.
          Serial.begin(57600);
          while (!Serial);

          // Inicjalizacja urządzeń
          Serial.println(F(„Inicjalizacja urządzeń I2C...“));
          mpu.initialize();

          // Weryfikacja połączenia
          Serial.println(F(„Testowanie komunikacji z urządzeniem...“));
          Serial.println(mpu.testConnection() ? F(„Połączono z MPU6050“) : F(„Nie połączono z MPU6050“));

          // Załadowanie i konfiguracja DMP
          Serial.println(F(„Inicjalizacja DMP...“));
          devStatus = mpu.dmpInitialize();

          // upewnienie się, że zadziałało - jeśli działa poprawnie, to zwraca zero
          if (devStatus == 0) {

                    // Uruchom DMP gdy jest gotowe
                    Serial.println(F(„Uruchamianie DMP...“));
                    mpu.setDMPEnabled(true);

                    // Włącz przerwania w Arduino
                    Serial.println(F(„Uruchamianie wykrywania przerwań...“));
                    attachInterrupt(INT_PIN, dmpDataReady, RISING);
                    mpuIntStatus = mpu.getIntStatus();

                    // Ustaw flagę gotowości DMP aby główna pętla programu wiedziała, kiedy może go użyć
                    Serial.println(F(„DMP gotowe! Oczekiwanie na pierwsze przerwanie...“));
                    dmpReady = true;

                    //Ustaw przewidywaną wielkość pakietów DMP do porównywania w przyszłości
                    packetSize = mpu.dmpGetFIFOPacketSize();
          } else {
                    // Błąd:
                    // 1 = problem z odczytem pamięci
                    // 2 = niepowodzenie podczas aktualizacji konfiguracji DMP
                    Serial.print(F(„Inicjalizacja DMP zakończyła się niepowodzeniem (błąd „));
                    Serial.print(devStatus);
                    Serial.println(F(„)“));
          }
}

// Główna pętla programu
void loop() {
          // Jeżeli programowanie się nie powiodło, nic nie rób
          if (!dmpReady) return;

          if (mpuInterrupt || fifoCount >= packetSize)
          {
                    // Reset flagi przerwania i pobranie wartości bajtu INT_STATUS
                    mpuInterrupt = false;
                    mpuIntStatus = mpu.getIntStatus();

                    // Pobranie obecnej długościu kolejki FIFO
                    fifoCount = mpu.getFIFOCount();

                    // Sprawdzenie czy kolejka nie jest przepełniona (co nie powinno się zdarzyć)
```

Listing 1. cd.

```
                    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
                              // Reset FIFO, abyśmy mogli kontynuować na czysto
                              mpu.resetFIFO();
                              Serial.println(F(„Przepełnienie FIFO!“));

                    // Jeśli kolejka FIFO nie jest przepełniona to sprawdzamy czy dane z DMP są gotowe
                    } else if (mpuIntStatus & 0x02) {
                              // Oczekiwanie na gotowe dane o poprawnej długości
                              while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

                              // Odczytaj pakiet z kolejki FIFO
                              mpu.getFIFOBytes(fifoBuffer, packetSize);

                              // Śledzenie długości FIFO na wypadek gdyby dostępny był więcej niż jeden pakiet
                              // co pozwala czytać dane od razu, zamiast czekać na kolejne przerwanie.
                              fifoCount -= packetSize;

                              // Wyznaczenie kątów YPR z danych z bufora
                              mpu.dmpGetQuaternion(&q, fifoBuffer);

                        binaryFloat hi;

                        hi.floatingPoint[0] = q.x;
                        hi.floatingPoint[1] = q.y;
                        hi.floatingPoint[2] = q.z;
                        hi.floatingPoint[3] = q.w;
                        Serial.write(hi.binary,16);
                    }
          }
}
```

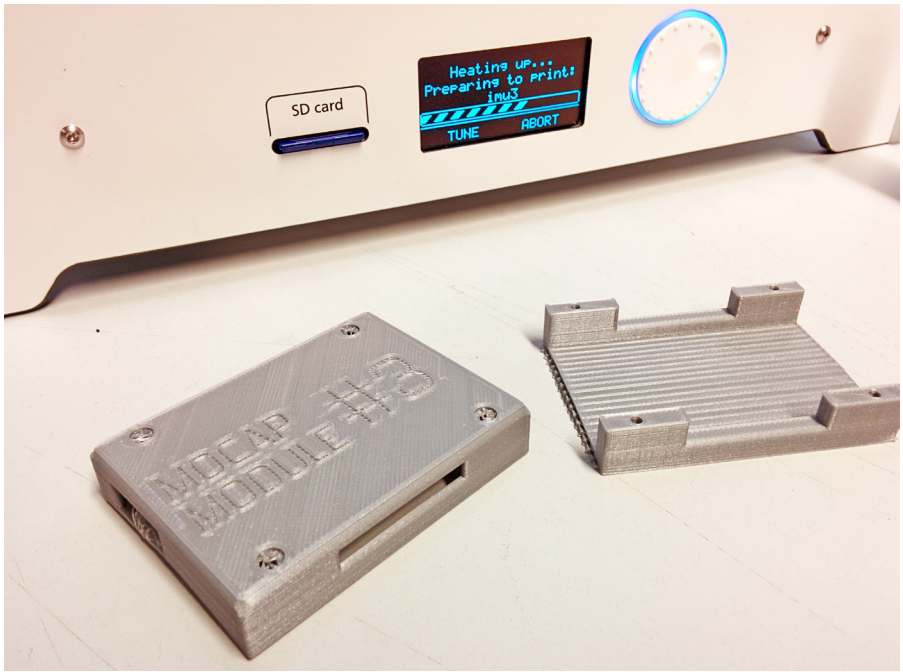
**Tabela 3. Lista połączeń modułu Arduino Mini Pro z HC-05 oraz MPU-6050**

| Arduino Mini Pro | HC-05 | MPU-6050 |
|---|---|---|
| Pin 10 | TXD | |
| Pin 11 | RXD | |
| 3,3 V | VCC | |
| GND | GND | |
| Pin 2 | | INT (przerwanie) |
| 3,3 V | | VCC |
| GND | | GND |
| A5 | | SCL |
| A4 | | SDA |
| RAW | Zasilanie z baterii (3,7 V) | |
| GND | Zasilanie z baterii (masa) | |

Możemy w tym celu posłużyć się telefonem komórkowym lub laptopem z Bluetooth i we właściwościach widzianego przez urządzenie sprzętu odnaleźć moduł i sprawdzić adres sprzętowy. W przypadku modułu autora adres sprzętowy to 00:14:01:03:55:35. Musimy zapisać adres, a następnie skonfigurować Arduino.

Do modułu Arduino UNO ładujemy przykładowy szkic w Arduino IDE pod nazwą SoftwareSerialExample, ustawiamy w metodzie mySerial.begin() prędkość komunikacji na 38400 a następnie kompilujemy szkic w Arduino IDE i ładujemy go do pamięci Arduino UNO. Możemy teraz odłączyć Arduino od USB i podłączyć je do modułu Bluetooth zgodnie z połączeniami zapisanymi w **tabeli 1**.

Teraz podłączamy Arduino UNO do USB i w Arduino IDE otwieramy terminal monitora portu szeregowego (prędkość 9600 bodów), gdzie możemy wysyłać komendy poprzez port szeregowy do modułu Bluetooth. Aby przetestować komunikację, wysyłamy *„AT”*, na co odpowiedzią powinno być *„OK”*. Jeśli tak nie jest, upewnijmy się,




**Fotografia 2. Wydrukowana obudowa dla modułów**

że wszystko poprawnie skonfigurowaliśmy i połączyliśmy i spróbujmy ponownie. Następnie, musimy skonfigurować moduł – w pierwszej kolejności podać mu jego adres sprzętowy, uważając na format podawania adresu (uwaga na przecinki):

```
AT+BIND=0014,01,035535
```

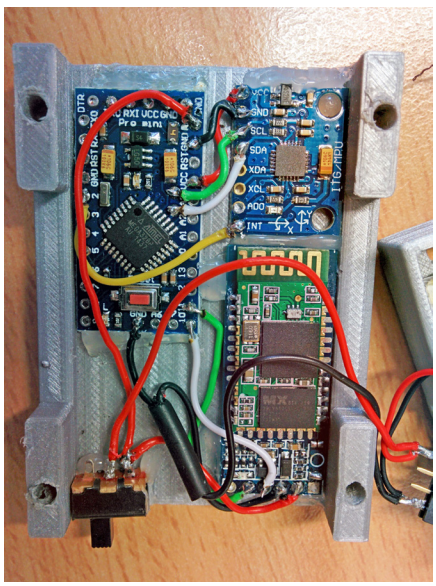
A potem podać nową nazwę urządzenia:

```
AT+NowaNazwaModułu
```

na co powinniśmy dostać odpowiedź *„OKsetname”*. Teraz możemy odłączyć Arduino od komputera, a następnie od modułu HC-05. Transceiver Bluetooth jest już gotowy.

## Konfiguracja modułu Arduino Mini Pro

Arduino Mini Pro nie jest wyposażone w konwerter USB-UART na płytce, więc aby je zaprogramować, potrzebny jest zewnętrzny moduł. W tym celu autor także wykorzystał Arduino UNO. Połączenia



**Fotografia 3. Elektronika modułu IMU zamontowana w obudowie**

pomiędzy modułami Arduino opisano w **tabeli 2**. Pamiętajmy, aby łączyć ze sobą moduły niepodłączone do zasilania/USB i dopiero po połączeniu modułów podłączyć je do komputera.

W Arduino IDE wprowadzić musimy szkic, który kontrolować będzie działanie modułu. Szkic zaprezentowany jest na **listingu 1**. Po skompilowaniu i wgraniu programu moduł jest gotowy do działania. Możemy więc przystąpić do finalnego montażu całego urządzenia.

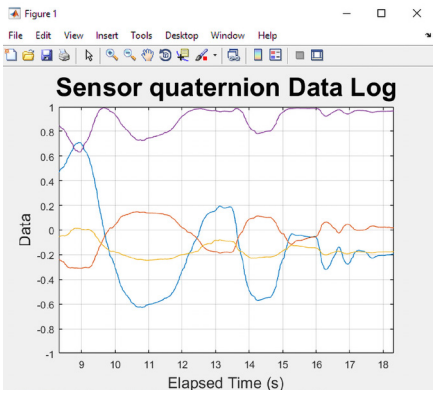
## Przygotowanie obudowy i montaż elementów

Pierwszym krokiem podczas montażu jest przygotowanie obudowy. Wykonana została ona techniką druku 3D. Pliki STL można pobrać na stronie autora lub z repozytorium na GitHubie. Gotowe wydruki pokazane są na **fotografii 2**. W górnej części obudowy wklejona zostaje elektronika (patrz **fotografia 3**), a przez dolną przewlekana jest gumka z doszytymi rzepami, która pozwoli na instalację urządzenia np. na ręce. Gumka jednocześnie mocuje baterie.

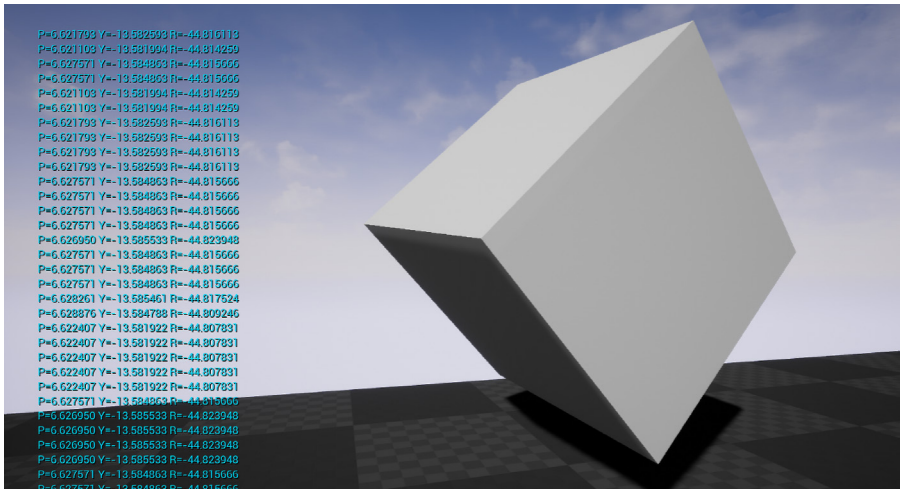
Po przygotowaniu obudowy w górną, pokazaną na rysunku 3 część wklejane

**Listing 2. Kod w MATLAB-ie do analizy danych z modułów IMU – obliczania kwaternionów i prezentacji ich w czasie**

```
% wyczyść ekran
clear
clc

% konfiguracja portu szeregowego (tutaj wpisz odpowiednie wartości)
serialPort = ‚COM4‘;
s = serial(serialPort);
s.BaudRate = 57600;
s.FlowControl = ‚hardware‘;
s.Timeout = 1000;
s.OutputBufferSize = 1000;

%%

% inicjalizacja danych wejściowych
qx = single(0);
qy = single(0);
qz = single(0);
qw = single(0);

% konfiguracja wykresu
plotTitle = ‚Sensor quaternion data log‘;          % tytuł wykresu
xLabel = ‚Elapsed Time (s)‘;                       % etykieta osi X
yLabel = ‚Data‘;                                   % etykieta osi Y
plotGrid = ‚on‘;                                   % ‚off‘ by wyłączyć siatkę
min = -1.0;                                       % ustawienie minimum w osi Y
max = 1.0;                                        % ustawienie minimum w osi X
scrollWidth = 10;                                 % ustawienie okresu na wykresie, poniżej zera wykreśla wszystkie dane
delay = .01;                                      % upewnij się, że aby próbkować szybciej niż rozdzielczość ekranu

time = 0;
data = zeros(100,4);
count = 0;

plotGraph = plot(rand(4));
size(plotGraph)

title(plotTitle,‘FontSize‘,25);
xlabel(xLabel,‘FontSize‘,15);
ylabel(yLabel,‘FontSize‘,15);
axis([0 10 min max]);
grid(plotGrid);


%%

fopen(s);
disp(‚Zamknij wykres i zakończ sesję‘);

tic

while ishandle(plotGraph)                         % pętla, która działa gdy wykres jest aktywny

        % odczytaj wartości z portu szeregowego (wysłane poprzez Bluetooth) i wyznacz wartości kwaternionów
        while(s.BytesAvailable > 16)
                X = fread(s,16,‘char‘);

                qx = typecast(uint8(X(1:4)), ‚single‘);
                qy = typecast(uint8(X(5:8)), ‚single‘);
                qz = typecast(uint8(X(9:12)), ‚single‘);
                qw = typecast(uint8(X(13:16)), ‚single‘);
        end

        % narysuj punkty na wykresie i wyświetl go
        if(~isempty(qx) && isfloat(qx))           % upewnij się, że dane mają odpowiedni typ
                count = count + 1;
                time(count) = toc;                % wyznacz czas, jaki minął

                data(count,1) = qx;
                data(count,2) = qy;
                data(count,3) = qz;
                data(count,4) = qw;

                for k = 1:numel(plotGraph)
                        if(scrollWidth > 0)
                                set(plotGraph(k), ‚XData‘, time(time > time(count)-scrollWidth), ‚YData‘, data(time >
                                time(count)-scrollWidth,k))
                                axis([time(count)-scrollWidth time(count) min max]);
                        end
                end

                % aktualiacja wykresów
                pause(delay);
        end
end

% Zamknięcie portu szeregowego i skasowanie niepotrzebnych już zmiennych
fclose(s);

clear count dat delay max min plotGraph plotGrid plotTitle s ...
                scrollWidth serialPort xLabel yLabel;

disp(‚Program zakończony...‘);
```




**Rysunek 4. Wykres kwaternionów w funkcji czasu, zebranych z modułów IMU zaprezentowany w MATLAB-ie**



**Rysunek 5. Zrzut ekranu programu do testowania modułu IMU z wykorzystaniem Unreal Engine**

są poszczególne moduły. Następnie trzeba je ze sobą połączyć zgodnie z opisem w **tabeli 3**. Połączenia te najlepiej wykonać, lutując pomiędzy poszczególnymi polami w modułach krótkie odcinki kolorowych kabli. Kolory pomogą z czytelnością połączeń, na wypadek gdybyśmy chcieli coś później zmieniać bądź też debugowali później gotowy układ.

Po podłączeniu całości (baterię podpinamy jako ostatnią) system jest gotowy do działania. Dobrze jest podłączyć baterię poprzez przełącznik. Dzięki temu będziemy w stanie wygodnie włączać i wyłączać urządzenie podczas korzystania z niego.

## Podsumowanie

Po uruchomieniu układu i sparowaniu naszego nowego urządzenia z np. komputerem możemy rozpocząć odbieranie danych z modułu. Autor wykorzystał dwa narzędzia do przetestowania – MATLAB oraz program oparty na Unreal Engine. Prostszy do prześledzenia jest skrypt MATLAB-a, który pokazano na **listingu 2**, wyświetla on wykres prezentujący kwaterniony pozycji w czasie. Przykładowy wykres z MATLAB-a zamieszczono na **rysunku 4**. Z kolei na **rysunku 5** zaprezentowano zrzut ekranu z aplikacji napisanej z wykorzystaniem Unreal Engine, gdzie sześcian na ekranie śledzi ruchy modułu IMU w przestrzeni. Oba przykłady są do pobrania z repozytorium autora na Githubie.

**Nikodem Czechowski**
*http://bit.ly/2FkkOxM*
*http://bit.ly/2RnFMAy*