Politechnika Poznańska

Wydział Automatyki, Robotyki i Elektrotechniki

Wizualizator ruchu IMU

Systemy mikroprocesorowe

Dokumentacja projektu

Kacper Krasiński, 151234

kacper.krasinski@student.put.poznan.pl

Kacper Grobelny, 151097

kacper.grobelny@student.put.poznan.pl

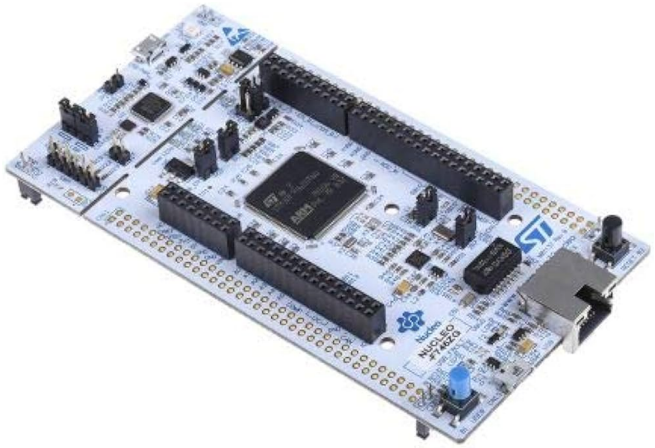
28 maja 2024

Spis treści

Opis Projektu

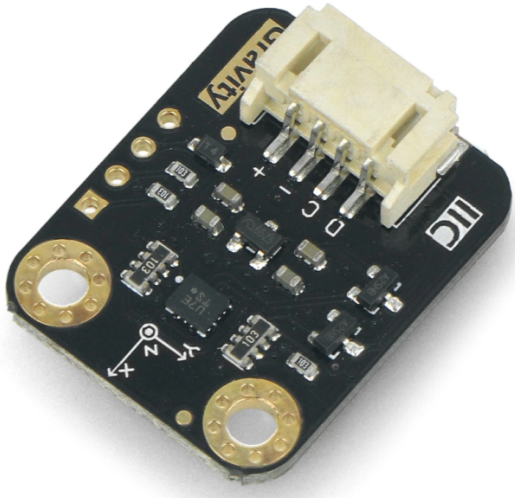
Projektem zaliczeniowym jest rejestrator ruchu bazujący na czujniku IMU. Projekt jest wzorowany na badaniach doktoranckich *Sebastian O.H. Madgwick* [1]. Różni się to jednak używanym IMU co za tym też idzie sposobem filtracji i obliczeń które wykonujemy w całości w systemie mikroprocesorowym. Dokumentacja do używanego tam *Inertial Measurement Unit* znajduje się na platformie GitHub [2]
Ruch rejestrowany jest przy pomocy poniższych komponentów, a dane przefiltrowane i przetworzone są zapisywane na karcie SD. Dane pobierane są z czujnika z częstotliwością 250 próbek na sekundę, a tablice przechowujące dane ograniczone są do 4500 próbek ze względu na pojemność mikrokontrolera która wynosi 1MB. Ograniczamy się do takiego zakresu ze względu na konieczne do obliczeń w późniejszych etapach generowane tablice o takiej samej długości. Taki zestaw danych daje nam możliwość zapisania i przetworzenia maksymalnie 18 sekund ruchu.
Do wykonania projektu użyto poniższych komponentów:

1. Płytka rozwojowa NUCLEO-F746ZG



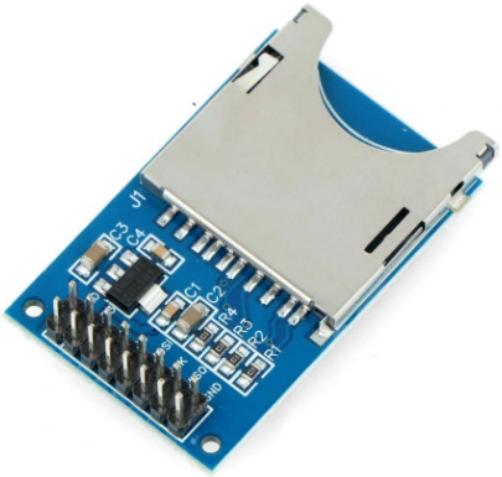
Rys. 1. Płytka rozwojowa NUCLEO oparta o STM32F746

2. Akcelerometr i żyroskop DFRobot SEN0250, BMI160 6DoF IMU [3]



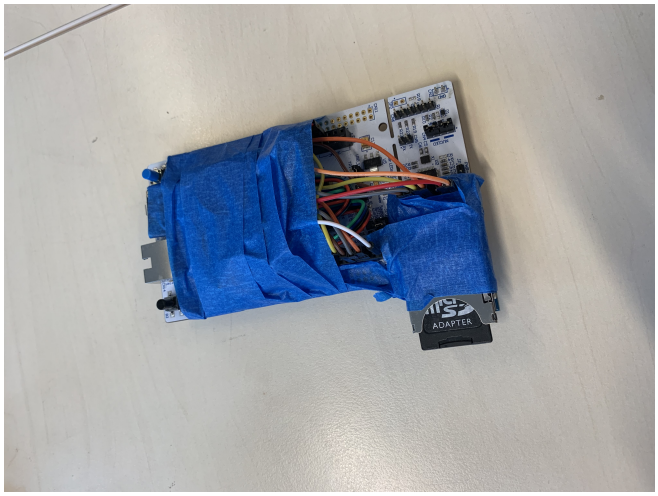
Rys. 2. Inertial Measurement Unit

3. Moduł czytnika kart SD [4]

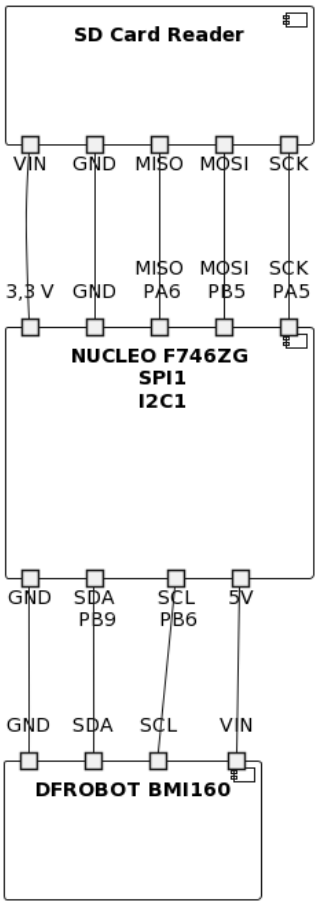


Rys. 3. Moduł czytnika kart SD

Zdjęcie połączonego układu przedstawiono poniżej:



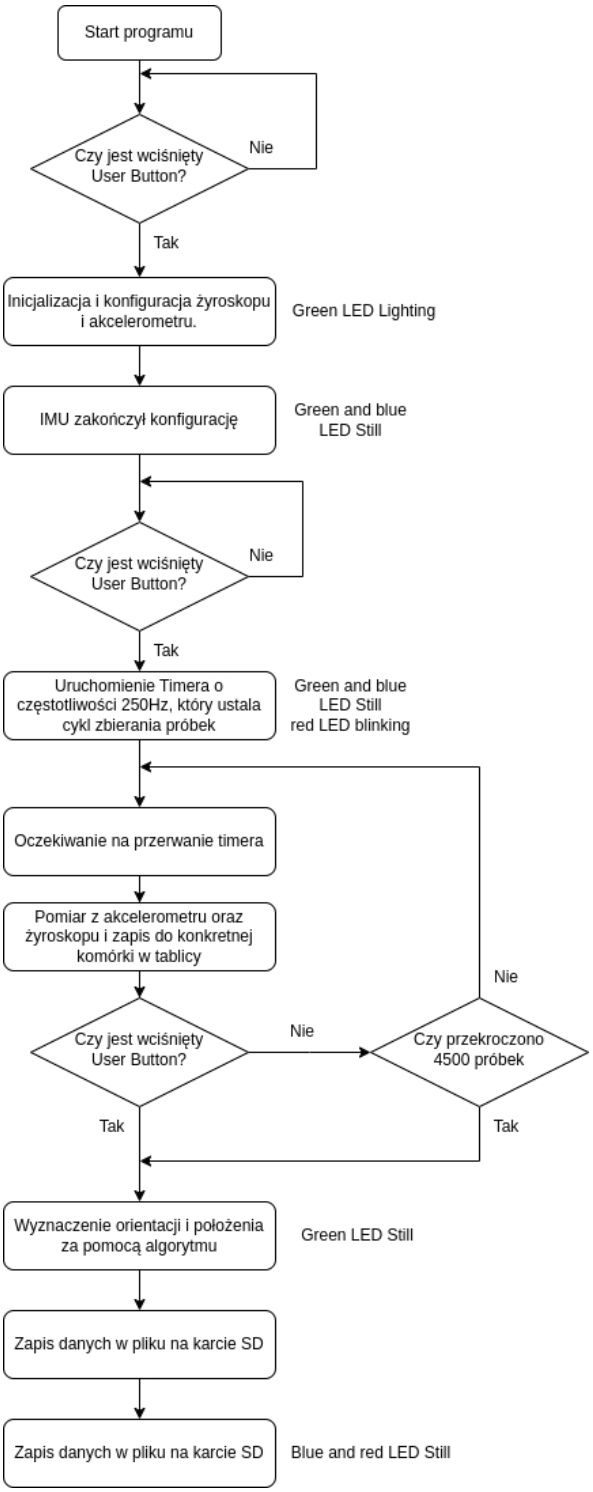
Rys. 4. Połączony układ



Rys. 5. Połączony układ

Opis kodu na płytce - rejestrator ruchu

System blokowy działania programu zaimplementowanego na mikrokontrolerze przedstawiony został poniżej:



Rys. 6. Graf sekwencji kodu

Program w przerwaniu timera z częstotliwością 250Hz odczytuje dane z akcelerometru oraz żyroskopu i zapisuje do tablicy z danymi, które następnie są przetwarzane w celu wyznaczenia trajektorii ruchu. Korzystamy z gotowej biblioteki dostępnej pod [5] przygotowanej przez `Bosch Sensortec GmbH`.

```
01. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
02. {
03.           if (htim->Instance == TIM1)
04.           {
05.                   bmi160ReadAccelGyro(&imu_t);
06.
07.                   if(sample_number < max_sample_number)
08.                   {
09.                           gyro_data[0][sample_number] = imu_t.BMI160_Gx_f32;
10.                           gyro_data[1][sample_number] = imu_t.BMI160_Gy_f32;
11.                           gyro_data[2][sample_number] = imu_t.BMI160_Gz_f32;
12.
13.                           acc_data[0][sample_number] = imu_t.BMI160_Ax_f32;
14.                           acc_data[1][sample_number] = imu_t.BMI160_Ay_f32;
15.                           acc_data[2][sample_number] = imu_t.BMI160_Az_f32;
16.
17.                           sample_number++;
18.                   }
19.           }
20. }
```

Po zebraniu próbek zostaje wywołana główna funkcja calculate, w której następuje przetworzenie zebranych danych w celu wyznaczenia położenia i rotacji urządzenia.

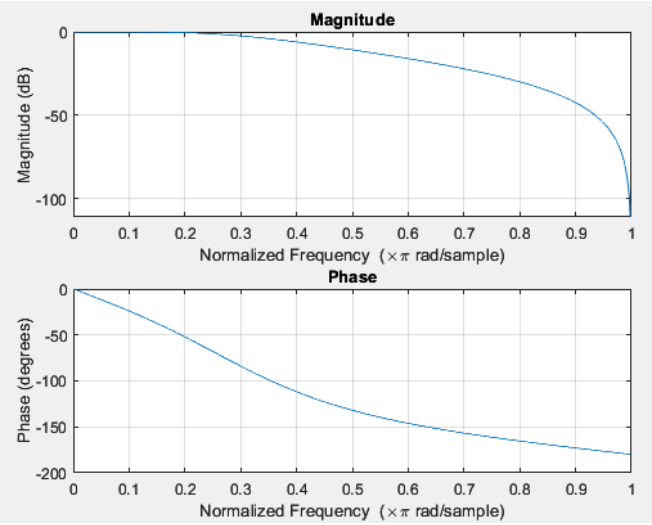
```
01. void calculate()
02. {
03.         float acc_total[max_sample_number];
04.         bool no_move[max_sample_number];
05.
06.         float velocity[3][max_sample_number];
07.
08.         float position[3][max_sample_number];
09.         float orientation[3][max_sample_number];
10.
11.
12.         for(int i=0; i<max_sample_number; i++)
13.         {
14.                 acc_total[i] = 0.0;
15.                 no_move[i] = 0;
16.                 velocity[0][i] = 0.0;
17.                 velocity[1][i] = 0.0;
18.                 velocity[2][i] = 0.0;
19.                 position[0][i] = 0.0;
20.                 position[1][i] = 0.0;
21.                 position[2][i] = 0.0;
22.                 orientation[0][i] = 0.0;
23.                 orientation[1][i] = 0.0;
24.                 orientation[2][i] = 0.0;
25.         }
26.
27.         filtr_raw_data();
28.
29.         calc_total_acc(acc_total);
30.
31.         filtr_total_acc(acc_total);
32.
33.         find_no_move(acc_total, no_move);
34.
35.         calc_orientation(orientation, no_move);
36.
37.         filtr_orientation_z(orientation);
38.
39.         rotation_of_axis(orientation);
40.
```

```
41.         calc_velocity(velocity, no_move);
42.
43.         velocity_compensation(velocity, no_move);
44.
45.         calc_position(position, velocity);
46.         corection_in_flat_move(position, velocity, no_move);
47.
48.         print(orientation, position, acc_total, velocity);
49.
50. }
```

Funkcja 'filtr_raw_data()' służy do filtrowania surowych danych zebranych z czujnika. Dane są przefiltrowane dolnoprzepustowo, w celu usunięcia szumów pomiarowych. Do filtracji została wykorzystana bilbioteka Arm CMSIS. W implementacji zastosowano filtr IIR drugiego rzędu o częstotiwości odcięcia 40Hz. Poniżej znajduje się transmitancja filtru. Filtr został przetransformowany do formy dyskretnej:

$$H(z) = \frac{0.4347 + 0.8695z^{-1} + 0.4347z^{-2}}{1 + 0.5193z^{-1} + 0.2197z^{-2}}$$

Oraz jego charakterystyka:



Rys. 7. Charakterystyka filtru 1-szego

```
01. void filtr_raw_data()
02. {
03.         for(int i=0; i<sample_number; i++)
04.         {
05.                 float actual = gyro_data[0][i];
06.                 arm_biquad_cascade_df1_f32(&iir_filter_gyro_x_low, &actual, &
                        gyro_data[0][i], 1);
07.                 actual = gyro_data[1][i];
08.                 arm_biquad_cascade_df1_f32(&iir_filter_gyro_y_low, &actual, &
                        gyro_data[1][i], 1);
09.                 actual = gyro_data[2][i];
10.                 arm_biquad_cascade_df1_f32(&iir_filter_gyro_z_low, &actual, &
                        gyro_data[2][i], 1);
11.                 actual = acc_data[0][i];
12.                 arm_biquad_cascade_df1_f32(&iir_filter_acc_x_low, &actual, &
                        acc_data[0][i], 1);
13.                 actual = acc_data[1][i];
14.                 arm_biquad_cascade_df1_f32(&iir_filter_acc_y_low, &actual, &
                        acc_data[1][i], 1);
15.                 actual = acc_data[2][i];
```

```
16.                 arm_biquad_cascade_df1_f32(&iir_filter_acc_z_low, &actual, &
                        acc_data[2][i], 1);
17.         }
18. }
```

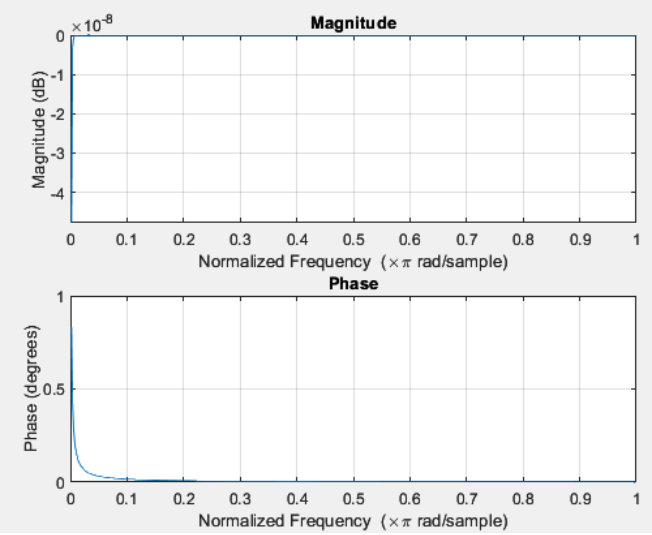
Następnie zostaje wyznaczona wypadkowa wartość przyspieszenia z akcelerometru.

```
01. void calc_total_acc(float acc_total[])
02. {
03.         for(int i=0; i<sample_number; i++)
04.         {
05.                 acc_total[i] = sqrt(acc_data[0][i]*acc_data[0][i]+acc_data[1][i]*
                        acc_data[1][i]+acc_data[2][i]*acc_data[2][i]);
06.         }
07. }
```

Obliczone wypadkowe przyspieszenie również zostaje przefiltrowane przy pomocy biblioteki Arm CMSIS. W tym przypadku jest to filtr dolnozaporowy, którego zadaniem jest usunięcie składowej stalej z wypadkowego przyspieszenia. Składowa stała jest spowodowana przyspieszeniem ziemskim równym $9.81 m/s^2$. Częstotliwość odcięcia filtru jest równa 0.0025Hz. Jest to filtr IIR drugiego rzędu zaprojektowany metodą Butterwortha. Poniżej znajduje się transmitancja filtru po przetransformowaniu do formy dyskretnej.

$$H_{\text{high}}(z) = \frac{z^2 - 2.0z + 1.0}{z^2 - 2.0z + 2.0}$$

Oraz jego charakterystyka:



Rys. 8. Charakterystyka filtru 2-go

```
01. void filtr_total_acc(float acc_total[])
02. {
03.         iir_state_acc_high[0] = calc_mean(acc_total, 100);      //set initial
                values x[n-1]
04.         iir_state_acc_high[1] = iir_state_acc_high[0];  //set initial values x[n
                -2]
05.
06.         for(int i=0; i<sample_number; i++)
07.         {
08.                 float actual = acc_total[i];
09.                 arm_biquad_cascade_df1_f32(&iir_filter_acc_high, &actual, &
                        acc_total[i], 1);
10.         }
11. }
```

W kolejnym kroku algorytm wyznacza, kiedy występował ruch, a kiedy czujnik znajdował się w stałym położeniu. Informacja ta jest potrzebna w kojenych funkcjach do niwelowania błędów podwójnego całkowania z przyspieszenia do prędkości i z prędkości do położenia. W przypadku braku ruchu, prędkości w osiach X, Y i Z nadpisywane są zerami. Informacja o braku ruchu jest również wykorzystywana w komensowaniu błędów wyznaczania położenia kątowego.

```
01. void find_no_move(float acc_total[], bool no_move[])
02. {
03.         int counter_of_no_move = 0;
04.
05.         for(int i=0; i<sample_number; i++)
06.         {
07.                 counter_of_no_move = 0;
08.                 while(fabs(acc_total[i]) < 0.07 && i<sample_number)
09.                 {
10.                         counter_of_no_move++;
11.                         i++;
12.                 }
13.
14.                 if(counter_of_no_move >= 60)
15.                 {
16.                         for(int j=0; j<counter_of_no_move; j++)
17.                         {
18.                                 no_move[i-1-j] = 1;
19.                         }
20.                 }
21.         }
22.
23.         for(int i=0; i<250; i++)
24.         {
25.                 no_move[i] = 1;
26.         }
27. }
```

Kolejna funkcja służy do wyznaczania orientacji. Odbywa się to poprzez numeryczne całkowanie prędkości kątowej z żyroskopu. W przypadku gdy nie występuje ruch, wyznaczona orientacja jest kompensowana. Jest ona wówczas wyznaczana na podstawie wektora przyspieszenia, który w przypadku braku ruchu jest skierowany zgodnie z wektorem przyspieszenia ziemskiego, czyli pionowo w dół. Korzystając z tej informacji, algorytm jest w stanie skorygować błąd.

```
01. void calc_orientation(float orientation[3][max_sample_number], bool no_move[])
02. {
03.         orientation[0][0] = 0.0;
04.         orientation[1][0] = 0.0;
05.         orientation[2][0] = 0.0;
06.         for(int i=1; i<sample_number; i++)
07.         {
08.                 if(!no_move[i])
09.                 {
10.                         orientation[0][i] = orientation[0][i-1] + gyro_data[0][i
                                ]*T;
11.                         orientation[1][i] = orientation[1][i-1] + gyro_data[1][i
                                ]*T;
12.                         orientation[2][i] = orientation[2][i-1] + gyro_data[2][i
                                ]*T;
13.                 }
14.                 else
15.                 {
16.                         orientation[0][i] = atan2(acc_data[1][i], acc_data[2][i])
                                *180/M_PI;
17.                         orientation[1][i] = -atan2(acc_data[0][i], acc_data[2][i
                                ])*180/M_PI;
18.                         orientation[2][i] = orientation[2][i-1];
```

```
                }
        }
}
```

Kolejna funkcja służy do przejścia z układu współrzędnych czujnika do układu współrzędnych pomieszczenia. Odbywa się to przy pomocy macierzy rotacji, która przelicza wektory wyrażone w osiach X, Y i Z z układu czujnika do wektorów w osiach X, Y i Z układu otoczenia.

```
void rotation_of_axis(float orientation[3][max_sample_number])
{
        for(int i=0; i<sample_number; i++)
        {
                float s1 = sin(-orientation[0][i]*M_PI/180);
                float c1 = cos(-orientation[0][i]*M_PI/180);
                float s2 = sin(-orientation[1][i]*M_PI/180);
                float c2 = cos(-orientation[1][i]*M_PI/180);
                float s3 = sin(-orientation[2][i]*M_PI/180);
                float c3 = cos(-orientation[2][i]*M_PI/180);

                float accX = acc_data[0][i];
                float accY = acc_data[1][i];
                float accZ = acc_data[2][i];

                //rotation matrix
                acc_data[0][i] = (accX*c2 - accZ*s2)*9.81;
                acc_data[1][i] = (accY*c1 + accX*s1*s2 + accZ*c2*s1)*9.81;
                acc_data[2][i] = (accX*c1*s2 - accY*s1 + accZ*c1*c2)*9.81 - 9.81;

                float total_ax_rotated = fabs(acc_data[0][i])*0.5;

                acc_data[0][i] = s3*total_ax_rotated;
                acc_data[1][i] = c3*total_ax_rotated;
        }
}
```

Funkcja 'velocity_compensation()' służy do kompensacji prędkości spowodowanej małą szybkością próbkowania. Podczas wykonywania kroku przyspieszenie z akcelerometru całkuje się dając prędkość w każdej z osi. Po wykonaniu kroku prędkość powinna się wyzerować. Funkcja ta koryguje prędkości dla próbek podczas wykonywania kroku, tak aby po zakończeniu ruchu prędkości były równe 0.

```
void velocity_compensation(float velocity[3][max_sample_number], bool no_move[])
{
        int i = 0;
        int i_first_sample_with_move = 0;
        int i_last_sample_with_move = 0;
        float velZ_end_step = 0.0;
        int time_of_step = 0;

        while(i < sample_number-1)
        {
                while(no_move[i] && i < sample_number-1)
                {
                        i++;
                }
                i_first_sample_with_move = i;
                while(!no_move[i] && i < sample_number-1)
                {
                        i++;
                }
                i_last_sample_with_move = i-1;
                velZ_end_step = velocity[2][i_last_sample_with_move]*0.5;
                time_of_step = i_last_sample_with_move-i_first_sample_with_move;

                if(time_of_step > 50)
```

```
25.                 {
26.                         for(int j=i_first_sample_with_move; j<=
                                i_last_sample_with_move; j++)
27.                         {
28.                                 velocity[2][j] = velocity[2][j] - (velZ_end_step/
                                       time_of_step)*(j-i_first_sample_with_move);
29.                         }
30.                 }
31.         }
32. }
```

Następnie jest obliczana pozycja, poprzez całkowanie prędkości.

```
01. void calc_position(float position[3][max_sample_number], float velocity[3][
        max_sample_number])
02. {
03.         position[0][0] = 0.0;
04.         position[1][0] = 0.0;
05.         position[2][0] = 0.0;
06.         for(int i=1; i<sample_number; i++)
07.         {
08.                 position[0][i] = position[0][i-1] + velocity[0][i]*T;
09.                 position[1][i] = position[1][i-1] + velocity[1][i]*T;
10.                 position[2][i] = position[2][i-1] + velocity[2][i]*T;
11.         }
12. }
```

KW kolejnym kroku wykonywana jest funkcja 'correction_in_flat_move()'. Podobnie jak poprzednie funkcje kompensujące służy ona do zrekompensowania małej częstotliwości próbkowania. Wykrywa czy ruch mastępuje na płaskiej powierzchni. Wówczas kompensuje wysokość, tak aby podczas ruchu na płaskiej powierzchni wysokość końcowa po wykonaniu kroku się nie zmieniała.

```
01. void corection_in_flat_move(float position[3][max_sample_number], float velocity
        [3][max_sample_number], bool no_move[])
02. {
03.         int samples_in_step = 0;
04.         int number_of_steps = 0;
05.         int i=0;
06.
07.         while(i < sample_number-1)
08.         {
09.                 i++;
10.                 while(!no_move[i] && i < sample_number-1)
11.                 {
12.                         i++;
13.                         samples_in_step++;
14.                 }
15.                 if(samples_in_step > f*0.3)
16.                 {
17.                         number_of_steps++;
18.                 }
19.                 samples_in_step = 0;
20.         }
21.
22.         int i_last_sample_without_move = 0;
23.         int i_next_sample_without_move = 0;
24.
25.         float height = position[2][sample_number-1];
26.         if(fabs(height/number_of_steps) < 0.7)
27.         {
28.                 i = 0;
29.                 while(i < sample_number-1)
30.                 {
31.                         while(no_move[i] && i < sample_number-1)
```

```
32.                             {
33.                                     i++;
34.                             }
35.                             i_last_sample_without_move = i-1;
36.                             while(!no_move[i] && i < sample_number-1)
37.                             {
38.                                     i++;
39.                             }
40.                             i_next_sample_without_move = i;
41.
42.                             float dH = position[2][i_next_sample_without_move]-
                                    position[2][i_last_sample_without_move];
43.                             float all_Z_move = 0.0;
44.                             for(int j=i_last_sample_without_move; j<
                                    i_next_sample_without_move+1;j++)
45.                             {
46.                                     all_Z_move += fabs(velocity[2][j]*T);
47.                             }
48.                             float Z_move_down = (all_Z_move-dH)/2.0;
49.                             float Z_move_up = all_Z_move - Z_move_down;
50.
51.                             if(fabs(Z_move_down) > 0.00001)
52.                             {
53.                                     float vel_correction_ratio = Z_move_up/
                                            Z_move_down;
54.
55.                                     for(int j=i_last_sample_without_move; j<
                                            i_next_sample_without_move+1;j++)
56.                                     {
57.                                             if(velocity[2][j] < 0.0)
58.                                             {
59.                                                     velocity[2][j] = velocity[2][j]*
                                                            vel_correction_ratio;
60.                                             }
61.
62.                                     }
63.                             }
64.                     }
65.
66.             }
67.
68.             calc_position(position, velocity);
69.     }
```

Ostatnim etapem jest zapis obliczonych danych na karcie SD. W 08. linijce kodu widzimy zapisywany na karcie SD plik, a zatem formatem jest plik .csv z separatorem w postaci średnika ';'
Do zapisu plików na kartę SD wykorzystano bibliotekę pochodzącą z repozytorium [6]. Dziękujemy autorom za pracę

```
01. void print(float orientation[3][max_sample_number], float position[3][
        max_sample_number], float acc_total[max_sample_number], float velocity[3][
        max_sample_number])
02. {
03.             fresult = f_mount(&fs,"",0);
04.             fresult = f_open(&fil, "dane.csv", FA_OPEN_ALWAYS | FA_READ | FA_WRITE);
05.             for(int i=0; i<sample_number; i++)
06.             {
07.                     char text[200];
08.                     int length = sprintf(text, "%.2f;%.2f;%.2f;%.4f;%.4f;%.4f;%.2f
                            ;%.2f;%.2f;%.2f;%.2f;%.2f;%.2f;%.2f;%.2f;\r\n",
                            orientation[0][i], orientation[1][i], orientation[2][i],
                            position[0][i], position[1][i], position[2][i], acc_total[i
                            ], velocity[0][i], velocity[1][i], velocity[2][i], gyro_data
```

```
                        [0][i], gyro_data[1][i], gyro_data[2][i], acc_data[0][i],
                        acc_data[1][i], acc_data[2][i]);
09.                       //int parametry_len = sprintf (parametry,"Text written
                              from STM32");
10.                       fresult = f_lseek(&fil,f_size(&fil));
11.                       fresult = f_write(&fil,text,length,&bw);
12.                   //HAL_UART_Transmit(&huart3, (uint8_t*)text, length, 1000);
13.         }
14.         f_close(&fil);
15.         fresult = f_mount(NULL,"",1);
16. }
```

Opis kodu w MATLABie - wizualizator ruchu

Po zebraniu danych przez rejestrator ruchu którym jest płytka rozwojowa NUCLEO połączona z IMU (ang. Inertial Measurement Unit) na kartę SD dzięki skryptowi `from_matrix.m` zaimplementowanemu w MATLABie [7] jesteśmy w stanie zwizualizować zebrane dane ruchu wraz z aktualną orientacją czujnika. Dzieje się to z pomocą załączonego niżej kodu. Kod powstał z pomocą plików pomocniczych oprogramowania bez użycia bibliotek/funkcji zewnętrznych.
Odczyt z pliku, który przykładowo wygląda jak poniżej:

...

```
7.34;-18.76;-4.54;1.7829;0.0522;0.6766;0.02;0.00;0.00;0.00;-1.76;0.54;-0.24;0.00;0.00;0.44;
7.36;-18.68;-4.54;1.7829;0.0522;0.6766;0.02;0.00;0.00;0.00;-0.94;0.80;-0.19;0.00;0.00;0.44;
7.66;-18.88;-4.55;1.7829;0.0522;0.6766;0.02;0.00;0.00;0.00;0.08;1.05;-0.23;0.00;0.00;0.41;
8.14;-19.10;-4.55;1.7829;0.0522;0.6766;0.01;0.00;0.00;0.00;0.86;1.21;-0.32;0.00;0.00;0.29;
8.43;-19.12;-4.55;1.7829;0.0522;0.6766;-0.00;0.00;0.00;0.00;1.15;1.23;-0.40;0.00;0.00;0.14;
8.41;-19.03;-4.56;1.7829;0.0522;0.6766;-0.01;0.00;0.00;0.00;1.10;0.98;-0.47;0.00;0.00;0.02;
8.42;-19.05;-4.56;1.7829;0.0522;0.6766;-0.02;0.00;0.00;0.00;0.81;0.53;-0.56;0.00;0.00;-0.00;
8.55;-19.22;-4.56;1.7829;0.0522;0.6766;-0.01;0.00;0.00;0.00;0.27;0.12;-0.76;0.00;0.00;0.11;
8.51;-19.25;-4.56;1.7829;0.0522;0.6766;0.01;0.00;0.00;0.00;-0.37;-0.03;-1.00;0.00;0.00;0.30;
8.32;-18.96;-4.57;1.7829;0.0522;0.6766;0.02;0.00;0.00;0.00;-0.78;0.14;-1.14;0.00;0.00;0.41;
8.20;-18.49;-4.57;1.7829;0.0522;0.6766;0.02;0.00;0.00;0.00;-0.61;0.45;-1.14;0.00;0.00;0.39;
8.23;-18.21;-4.57;1.7829;0.0522;0.6766;0.01;0.00;0.00;0.00;0.19;0.65;-0.93;0.00;0.00;0.29;
8.46;-18.42;-4.58;1.7829;0.0522;0.6766;-0.00;0.00;0.00;0.00;1.13;0.71;-0.49;0.00;0.00;0.16;
```

...

za pomocą funkcji `dlmread('file', ';')` oraz zapis odpowiednio wybranych danych do struktur `Rot` i `Pos` w celu prostszej obsługi. Przy wizualizacji wykorzystujemy tlko 6 pierwszych kolumn, jednak reszta danych była wykorzystywana przy znajdowaniu i naprawianiu błędów. Do zapisu do struktur posłużyła funkcja `struct(field1, value1, ..., value3)` z odpowiednio zadeklarowanymi zmiennymi. W ten sam sposób zdefiniowano strukturę Pos. wartości `value1, value2, value3` są pobierane z odczytanego pliku csv. Resztę wartości zadeklarowano tak jak poniżej:

```
cosys = [[0 0.1 0 0 0 0];
         [0 0 0 0.1 0 0];
         [0 0 0 0 0 0.1]];
smuga = 10;
plthdl(length(Rot.x)) = line();
txhdl(length(Rot.x)) = text();
```

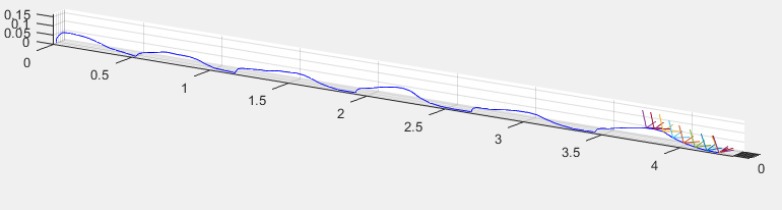
Poniższa pętla generuje wykres układu współrzędnych który jest obracany o rotację otrzymaną z mikrokontrolera. Nadpisywane są osie układu: `X, Y, Z` ustawiany kąt widzenia wykresu oraz podtrzymywany przebieg pozycji otrzymanej z rejestratora ruchu.

```
for i = 1:length(Orientation.X)
    plthdl(i) = plot3(cosys(1,:), cosys(2 ,:), cosys(3,:));  hold on;
    plthdl(i).Visible = "off";
    xlim([min(Pos.x)-1 max(Pos.x)+1]);
    ylim([min(Pos.y)-1 max(Pos.y)+1 ]);
```

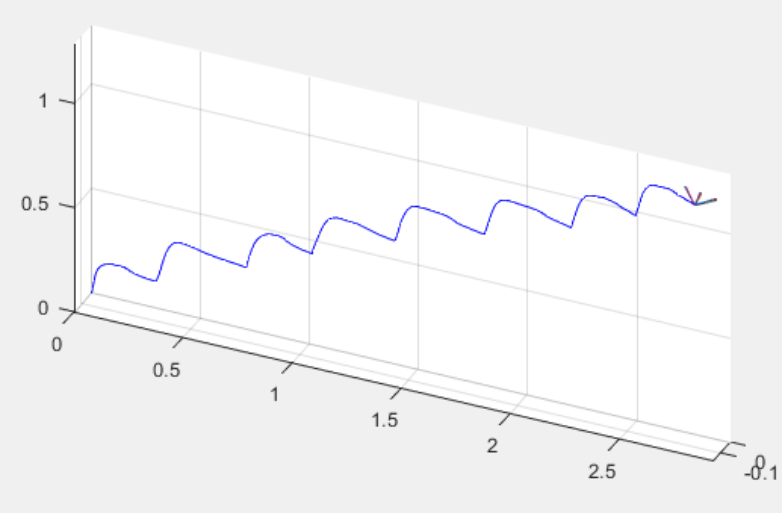
```
06.     zlim([min(Pos.z)-1 max(Pos.z)+1]); axis tight square equal
07.     rotate(plthdl(i),[1 0 0], Rot.x(i), [Pos.x,Pos.y,Pos.z]);
08.     rotate(plthdl(i),[0 1 0], Rot.y(i), [Pos.x,Pos.y,Pos.z]);
09.     rotate(plthdl(i),[0 0 1], Rot.z(i), [Pos.x,Pos.y,Pos.z]);
10.     plthdl(i) = plot3(plthdl(i).XData+Pos.x(i), plthdl(i).YData+Pos.y(i), plthdl(
            i).ZData+Pos.z(i))
11.     plthdl(i).Visible = "on";
12.     txhdl = text(plthdl(i).XData, plthdl(i).YData, plthdl(i).ZData,{'','X','','Y'
            ,'','Z'});% c
13.     plot3(Pos.x(1:i), Pos.y(1:i), Pos.z(1:i), 'Color', 'Blue');
14.     view(-75, 45);
15.     grid on;
16.
17.     if i > smuga
18.         plthdl(i-smuga).reset;
19.     end
20.     pause(1e-15);
21.     delete(txhdl);
22. end
```

Wyniki

Wyniki wizualizatora ruchu przedstawiono na filmiku zamieszczonym na platformie YouTube [8] zamieszczono tam kolejno ruch - chód po płaskiej powierzchni, wchodzenie po schodach prostych, oraz wchodzenie po schodach z klatką schodową. Natomiast w repozytorium [9] cały kod. Poniżej zrzuty ekranu z wykresami końcowymi:

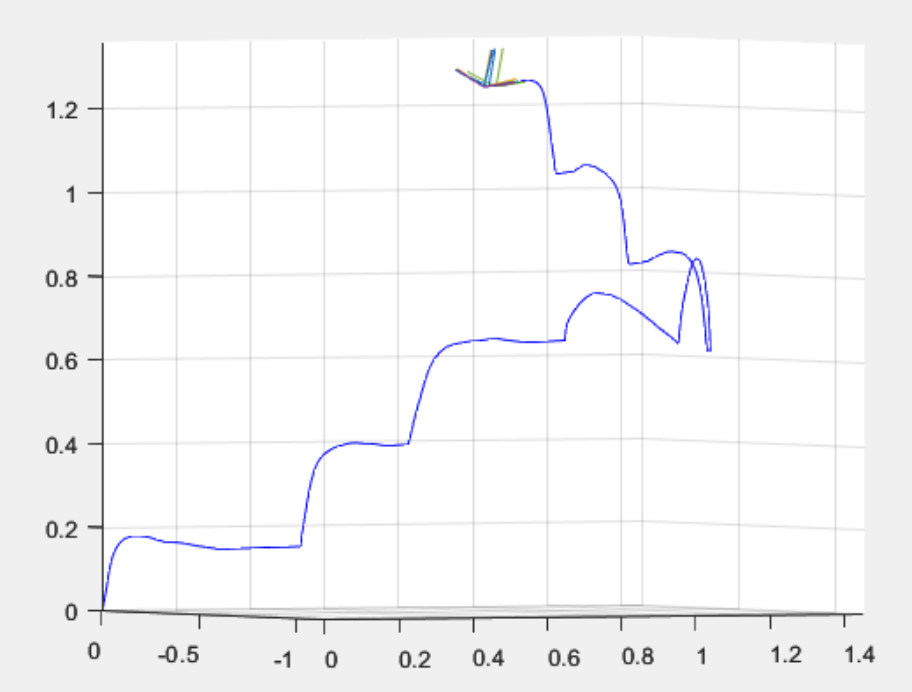


Rys. 9. Wartości pozycji w chodzie po płaskiej przestrzeni



Rys. 10. Wartości pozycji w chodzie po schodach na wprost

Wykresy powyżej przedstawiają idealną sytuację, gdy ruchy wykonywane są powoli, stopa nie jest przemieszczana po wykonaniu ruchu, nie ma gwałtownych zmian orientacji i wysokość się wyłącznie zwiększa. Do takiej wizualizacji jest również udostępnione nagranie z ruchem rzeczywistym [10]. Podczas głębszych testów dostrzeżono kilka elementów które możnaby usprawnić. Między innymi dobieranie funkcji kompensujących w zależności od wykonywanego typu ruchu - klasyfikacja typu ruchu na podstawie mniejszego zbioru zmiennych. Zapis danych na karcie SD mógłby być wydajniejszy, docelowo brak danych niefiltrowanych bądź filtrowanych częściowo. Dla bardziej dynamicznych ruchów należałoby sprawdzić czy przyjęta wartość szybkości próbkowania jest wystarczająca z powodu dużej dynamiki zmian. W załączonym materiale wideo [11] widzimy też nieudaną próbę zarejestrowania ruchu zejścia po schodach po wcześniejszym wejściu. Niektóre funkcje wymagają poprawy aby równie dobrze działały przy każdym typie ruchu. Również na wcześniej wspomnianej pracy [1] nie widzimy prób prezentujących ruch w innych płaszczyznach bądź z większą dynamiką niż taką którą udało się nam przetestować z powodzeniem.
Projekt mógłby być rozwijany w kierunku rejestrowania bardziej dynamicznych ruchów, przesyłania danych w chwili rzeczywistej na przykład poprzez bluetooth bądź access point, tym samym czyszczenia zapisanego buforu i pobierania danych bez limitu pojemności. Również testy innego rodzaju ruchów jak i ruchu nie tylko na stopie a także na przykład rzuconego/jeżdżącego/kroczącego obiektu.



Rys. 11. Wartości pozycji w chodzie po schodach z klatką schodową

Bibliografia

1. MADGWICK, Sebastian O.H. *3D Tracking with IMU*. 2011. Dostępne także z: https://youtu.be/6ijArKE8vKU?feature=shared.
2. *Github X-IMU* [Resources]. Dostępne także z: https://github.com/xioTechnologies/x-IMU3-Software.git.
3. *IMU sensor DFRobot* [Component]. Dostępne także z: https://botland.com.pl/gravity-akcelerometry-i-zyroskopy/19380-gravity-bmi160-6dof-imu-3-osiowy-akcelerometr-i-zyroskop-dfrobot-sen0250-6959420913381.html?fbclid=IwAR1hFrZxonPVF486MNzfM7oKAX1Up_kdZxvCpqis2YSZA1tZi6Hz3_9faCo.
4. *Moduł czytnika kart SD* [Component]. Dostępne także z: https://botland.com.pl/akcesoria-do-kart-pamieci/1507-modul-czytnika-kart-sd-5903351241342.html.
5. *Github z biblioteką do bmi160* [Resources]. Dostępne także z: https://github.com/boschsensortec/BMI160_driver.git.
6. *Github z biblioteką do czytnika kard SD* [Resources]. Dostępne także z: https://github.com/eziya/STM32_SPI_SDCARD.git.
7. INC., The MathWorks. *MATLAB version: 9.13.0 (R2022b)*. Natick, Massachusetts, United States: The MathWorks Inc., 2022. Dostępne także z: https://www.mathworks.com.
8. KACPER GROBELNY, Kacper Krasiński. *Filmik z wynikami wizualizatora ruchu* [Resources]. Dostępne także z: https://youtu.be/qUkBFtd2EQ8?si=fn2aTX197TLYUPL0.
9. KACPER KRASIŃSKI, Kacper Grobelny. *Github do projektu wizualizatora ruchu* [Resources]. Dostępne także z: https://github.com/Krasa35/IMU_MoveVis.git.
10. KACPER GROBELNY, Kacper Krasiński. *Wizualizator IMU - test wejścia po schodach* [Resources]. Dostępne także z: https://youtu.be/G9f1QtU-dxM?feature=shared.
11. KACPER GROBELNY, Kacper Krasiński. *Wizualizator IMU - test wejścia po schodach* [Resources]. Dostępne także z: https://youtu.be/p07kD3Nh7i4?feature=shared.