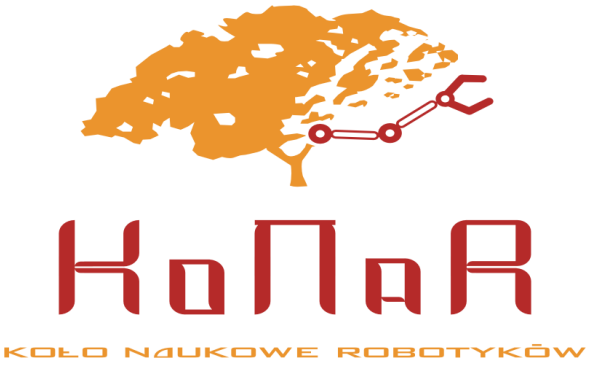


Proste metody obsługi akcelerometrów – zastosowanie w robotach mobilnych na przykładzie ADXL202 oraz MMA7260.

Jan Kędzierski

Edgar Ostrowski

Wrocław, 30.05.2007

Spis treści

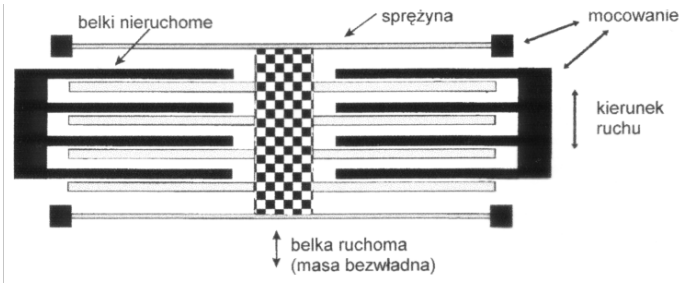
1 Wstęp

Jednym z podstawowych zagadnień pojawiających się przy sterowaniu robotem mobilnym jest problem lokalizacji, czyli określenia jego położenia w wyróżnionym, globalnym układzie współrzędnych. Częstokroć w tym celu wykorzystywane są metody odometrii. Ich stosowanie wymaga przyjęcia założenia o braku poślizgów elementów uczestniczących w procesie ruch. W przypadku kołowych robotów mobilnych, do najpopularniejszych metod odometrii zaliczyć można metodę, polegającą na estymacji przemieszczenia robota na podstawie obrotu jego kół. Założenie o braku poślizgu jest trudne do spełnienia w rzeczywistości. Dodatkowo, na platformę mogą działać nieznane siły zewnętrzne, pochodzące np. od zderzeń z innymi robotami poruszającymi się w pobliżu. Z tego względu metody odometrii często zawodzą przy określeniu położenia robota.

Jednym ze sposobów radzenia sobie z tym problemem jest zainstalowanie na robocie sensorów, dostarczających dodatkowych informacji o jego prędkościach, pozycji i orientacji. Mogą to być między innymi czujniki mierzące przyspieszenie – akcelerometry. Dane pochodzące z tego typu czujników, poddane prostym obliczeniom matematycznym mogą być źródłem bardzo przydatnych informacji. Przedstawione w sprawozdaniu metody są bardzo uproszczone i mogą być wykorzystywane w aplikacjach, w których precyzja pomiarów nie odgrywa znaczącej roli. Zawarto tu opis dwóch typów czujników ADXL202 firmy Analog Devices[1] oraz MMA7260 firmy Freescale[2]. Elementy te są dostępne na rynku za niezbyt wygórowaną cenę.

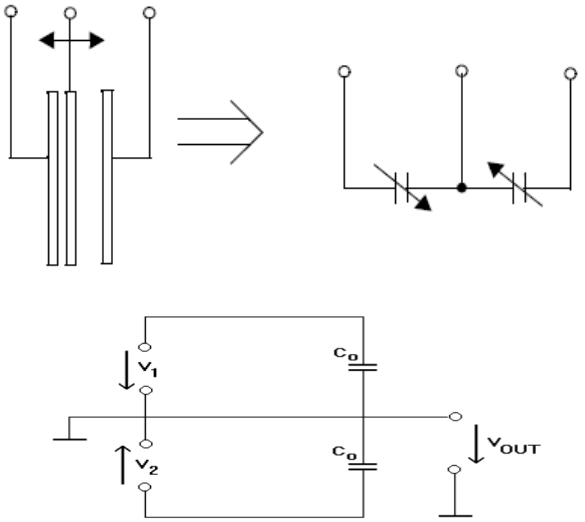
2 Zasada działania czujników

Sensory w swojej miniaturowej strukturze zawierają kondensator różnicowy [Rys. 1]. Centralna ruchoma belka, tworząc wraz z nieruchomymi belkami strukturę grzebieniową, jest wychylana z położenia równowagi przez siły bezwładności. Ruchoma belka jest jednocześnie ruchomą okładką w strukturze kondensatora różnicowego, który utworzony jest przez odpowiednie połączenia elektryczne belek struktury grzebieniowej.



Rys. 1 Widok uproszczony struktury mikromechanicznej czujnika.

Napięcia przyłożone do dolnej i górnej elektrody mają kształt prostokątny i są przesunięte w fazie o 180 stopni [Rys. 2]. Gdy elektroda środkowa jest w położeniu równowagi (obie pojemności równe) sygnał wyjściowy V_{OUT} = 0. Wychylenie się elektrody spowoduje „rozrównoważenie” układu i pojawienie się na wyjściu napięcia. Napięcie to, w dalszych blokach jest odpowiednio filtrowane, wzmacniane i demodulowane.



Rys. 2 Uproszczony schemat połączeń wewnętrznych czujnika.

3 Przegląd akcelerometrów

Obecnie na rynku można zakupić kilka tanich modeli czujników przyspieszenia. Produkują je głównie firmy Analog Devices, Freescale, a także od niedawna STMicroelectronics. Czujniki ostatniej firmy mają w sobie wbudowaną kompensację wpływu temperatury. W ramach projektu przebadano modele dwóch pierwszych producentów ADXL202 [1] oraz MMA7260 [2].

Przetwarzanie pomiarów i wysyłanie ich do komputera PC powierzono 32-bitowemu mikrokontrolerowi firmy Freescale MC68332 [3]. Jego zaletą jest niewątpliwie niezależna, programowalna jednostka czasowa TPU [4], pozwalająca realizować skomplikowane funkcje czasowe bez zaangażowania jednostki centralnej CPU. Układ ten ma również wbudowany kolejkowany interfejs QSPI [5] niezwykle przydatny przy cyklicznym wykonywaniu pomiarów np. przetwornikiem AC.

3.1 Model ADXL202 [1]

Cechy układu

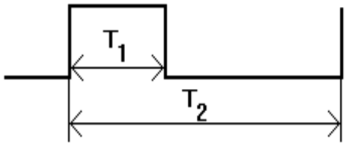
Czujnik ten posiada następujące własności:

- pomiar przyspieszeń w dwóch osiach
- wyjścia cyfrowe i analogowe
- pasmo 0.01-5kHz (5kHz – tylko na wyj. analogowym)
- zakres pomiaru ±2g (ADXL 202) ±10g (ADXL 210)
- możliwość podłączenia bezpośrednio do mikrokontrolera bez udziału przetwornika AC
- małe wymiary 5x5x2mm

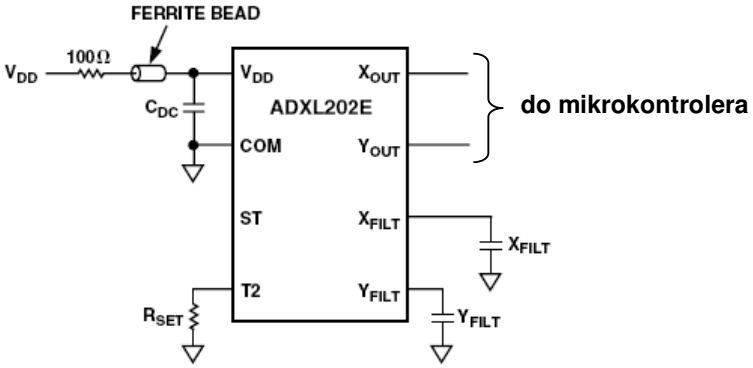
Fot. 1 Widok modułu z czujnikiem.

Podłączenie układu do mikrokontrolera

Dane z czujnika były odczytywane z wyjść cyfrowych. Na wyjściach cyfrowych układu generowany jest przez modulator DCM sygnał prostokątny o częstotliwości 1kHz [Tabela 1]. Wypełnienie tego sygnału jest wprost proporcjonalne do działającego na czujnik przyspieszenia [Rys 3]. Wynosi ono 50% dla ADXL202 zwiększa się o 12,5% / 1g.



Rys. 3 Czas wypełnienia T_1 – zmienia się wraz z działającym przyspieszeniem.



Schemat 1 Podłączenie układu.

Rezystor R_{set} [Schemat 1] służy do ustawienia czasu T_2, czyli częstotliwości sygnału generowanego na wyjściach cyfrowych. Poniżej przedstawiono wartości rezystora dla wybranych okresów. W projekcie użyto rezystora 125 kΩ.

R_{set}	T_2
125 kΩ	1 ms
250 kΩ	2 ms
625 kΩ	5 ms
1,25 MΩ	10 ms

Tabela 1. Wartości rezystora R_{set}.

Kondensatory X_{filt} i Y_{filt} ograniczają pasmo czujnika. Jeżeli korzystamy z wyjść cyfrowych należy pamiętać, że próbkowanie sygnału nie może odbywać się częściej niż 1kHz. A zatem, stosując twierdzenie Kotielnikowa-Shannona częstotliwość zmian przyśpieszenia nie powinna być większa niż 500Hz. Należy zatem zastosować filtr dolno-przepustowy na wyjściach analogowych, które są wyprowadzone przed modulatorem przebiegów prostokątnych na wyjściu cyfrowym. Poniższa tabela przedstawia sposób doboru tych elementów. W projekcie użyto kondensatorów C_x , C_y =0,47 µF.

C_x , C_y	f
0,47 µF	10 Hz
0,10 µF	50 Hz
0,05 µF	100 Hz
0,027 µF	200 Hz
0,01 µF	500 Hz

Tabela 2. Wartości kondensatorów C_x i C_y.

Wyjścia cyfrowe podłączono do linii TPU mikrokontrolera.

Obsługa programowa czujnika.

Do pomiaru szerokości impulsu użyto funkcji PPWA (Period/Pulse-Width Accumulator) [6]. Pozwala ona zliczyć zarówno szerokość impulsu, podłączonego pod linie kanału, jak i ich ilość. W projekcie wykorzystano pierwsza opcję. Istnieje także możliwość zliczenia szerokości

kilku impulsów, co można wykorzystać do uśrednienia pomiarów. Należy jednak pamiętać, że rejestr, w którym jest przechowywana zliczona suma może być max 24 bitowy.

Kod programu konfigurujący wybrany kanał (13) do pracy z funkcją PPWA (dla jednego wyjścia czujnika):

```
#include "sim.h"            /* definicje rejestrow SIM dla 68332 */
#include "qsm.h"            /* definicje rejestrow QSM dla 68332 */
#include "tpu.h"            /* definicje rejestrow TPU dla 68332*/

#define PPWAFUN 15         /* kod funkcji PPWA w masce MW */
#define PPWA1   13         /* kanal dla PPWA dla przyspieszen X */

/* ustawienie wektora przerwan*/
*((void (**) ())(4*((TI_VECT<<4)+PPWA1))) = int_PPWA1;

TPURAM(PPWA1,0) =0x0007;       /* pomiar szerokosci impulsu z TCR1  */
TPURAM(PPWA1,1) =0x0800;       /* liczba impulsow do zmierzenia = 8 */
TPURAM(PPWA1,4) =0xff00 ;      /* wyzerowanie licznika UPPER */
TPURAM(PPWA1,5) =0x0000 ;      /* wyzerowanie licznika LOWER */


SetFun(PPWA1,PPWAFUN);         /* kod funkcji:PPWAFUN */
SetSeq(PPWA1,2);              /* 24-bitowy pomiar szerokosci impulsu */
SetSer(PPWA1,2);              /* zadanie: inicjalizacja */
SetCpr(PPWA1,3);              /* priorytet: wysoki */
EnInt(PPWA1);                 /* zezwolenie na przerwanie od kanlu z funkcja
PPWA1*/
```

Kod programu obsługujący przerwanie po zliczeniu 8 impulsów:

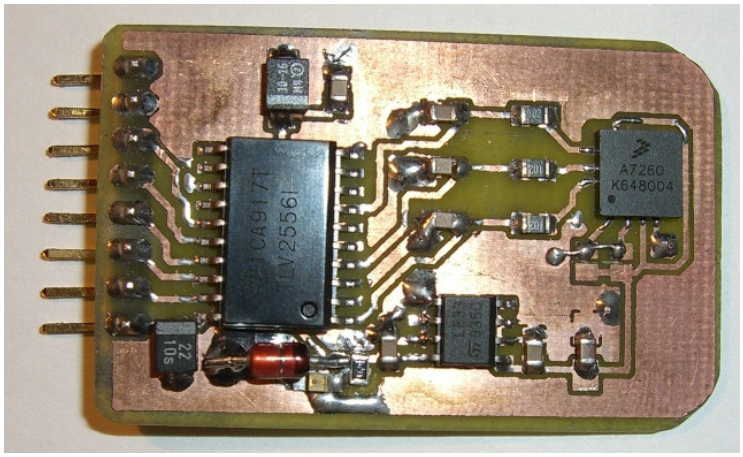
```
interrupt void int_PPWA1()
{
signed int pomiar=0;

pomiar=(signed int)TPURAM(PPWA1,5);  /* czytamy 16 bitow - tyle wstrarczy */
pomiar=(pomiar>>3);                  /* dzielimy pomiar przez 8 */
ClrInt(PPWA1);                       /* gasimy flagę przerwania */
}
```

3.2 Model MMA7260 [2]

Cechy układu

Czujnik ten posiada następujące własności:

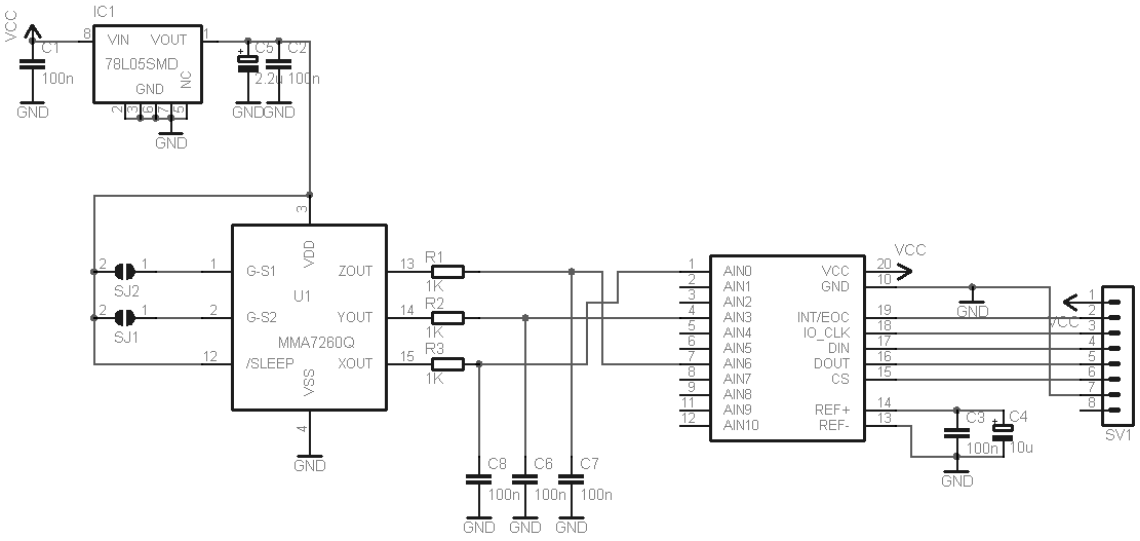
- pomiar przyspieszeń w trzech osiach
- wyjścia analogowe
- pasmo 11kHz
- wybór zakresów (1,5/2/4/6g)
- duża wytrzymałość na uderzenia
- wymaga przetwornika AC
- małe wymiary 6x6x1,45mm



Fot. 2 Widok modułu z czujnikiem.

Podłączenie układu do mikrokontrolera

Czujnik posiada tylko wyjścia analogowe, a zatem niezbędne jest użycie przetwornika AC. Do realizacji projektu wybrano przetwornik firmy Texas Instruments TLV2556 [7]. Układ ten posiada następujące cechy: 12- bitowa rozdzielczość, 11 kanałów, szybkość 200kS/s, wewnętrzny układ napięcia odniesienia 4,096V (1bit/1mV). Komunikacja z przetwornikiem odbywa się po interfejsie SPI. Dodatkowo konieczne było użycie stabilizatora redukującego napięcie z 5V na 3V3, gdyż takim właśnie zasilany jest akcelerometr. Filtr dolno-przepustowy wykonano z elementów RC (na każdym wyjściu).



Schemat. 2 Podłączenie czujnika.

Obsługa programowa czujnika.

Za pomiary wyjść z akcelerometru odpowiada wyżej opisany przetwornik. Użyty w projekcie mikrokontroler komunikuje się z przetwornikiem dzięki wbudowanemu interfejsowi QSPI. Dodatkowo może on pracować cyklicznie bez udziału jednostki CPU.

Kod programu inicjujący kolejkę QSPI:

```
#include "sim.h"            /* definicje rejestrow SIM dla 68332 */
#include "qsm.h"            /* definicje rejestrow QSM dla 68332 */
#include "tpu.h"            /* definicje rejestrow TPU dla 68332*/

/*Konfiguracja QSPI*/
#define ADCSEL 0xc           /* wybor ADC stanem niskim na PCS1 */
#define DESELECT 0xe         /* zaden SLAVE nie jest wybrany */


QPAR = 0x7B;            /* wszystkie dla QSPI*/
QPDR = DESELECT<<3;;    /* stany na porcie Q wylaczenie wszystkiego*/
QDDR = 0xFE;
SPCR0 = 0xB004;         /* MASTER, trnsfer - 12bitow CPOL=0, CPHA=0,*/
                        /* zegar = 2MHz */
SPCR1 = 0x1717;         /*  DSCKL, DTL = opoznienie przed i po*/
SPCR2 = 0x020F;         /* kolejka 0-2, zacznij od F*/

QTRAN[15] = 0xF00;      /* rejestr konfigurujący przetwornik AC */
QTRAN[0]  = 0x300;      /* pomiar z kanalu 3 */
QTRAN[1]  = 0x600;      /* pomiar z kanalu 6 */
QTRAN[2]  = 0x000;      /* pomiar z kanalu 0 */


QCOMD[15] = QsmDT | QsmBITSE | QsmDSCK | ADCSEL; /*czekaj przed i po, 12bit
QCOMD[0]  = QsmDT | QsmBITSE | QsmDSCK | ADCSEL; /*czekaj przed i po, 12bit
QCOMD[1]  = QsmDT | QsmBITSE | QsmDSCK | ADCSEL; /*czekaj przed i po, 12bit
QCOMD[2]  = QsmDT | QsmBITSE | QsmDSCK | ADCSEL; /*czekaj przed i po, 12bit
```

Kod uruchamiający kolejkę:

```
int RunQueue()
{
    SPCR1 |= QsmSPE;                /* wlacz QSPI */
    while(!(SPSR & QsmSPIF));       /* czekaj na koniec operacji */
    SPSR &= (~QsmSPIF);             /* zgas flage zakonczenia */
    return (1);
}
```

Kod obsługujący cykliczne przerwanie inicjujące pomiar. Należy pamiętać, że przerwanie to nie może być zbyt często zgłaszane. Częstość ta jest ograniczona przez czas, który przetwornik potrzebuje do wykonania pomiaru w trzech kanałach:

```
interrupt void int_PIT()
{
signed int accelX =0;
signed int accelY =0;
signed int accelZ =0;
```

```
RunQueue();

accelX =((signed int) QREC[0]);
accelY =((signed int) QREC[1]);
accelZ =((signed int) QREC[2]);
}
```

4 Prosty algorytm pozyskiwania prędkości i pozycji na podstawie przyspieszeń [8].

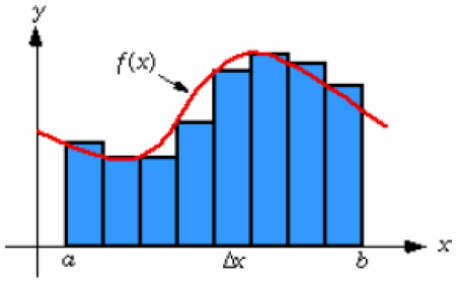
Aby pozyskać z przyspieszeń prędkość i przemieszczenie należy zastosować algorytm podwójnego całkowania. Z teoretycznego punktu widzenia to powinno wystarczyć. W praktyce jednak tak nie jest i należy zastosować pewne modyfikacje programowe jak i sprzętowe.

4.1 Teoria

Teoria mówi dokładnie tyle, że zmierzone przyspieszenia należy dwukrotnie zcałkować, aby uzyskać przemieszczenie.

$$V = \int (a)dt \qquad S = \int (V)dt \quad \therefore \quad \int \left(\int (a)dt \right) dt$$

W przypadku układów ciągłych np. analogowych jest to możliwe do uzyskania. W układach cyfrowych czas nie jest ciągły, a pomiary dokonywane są w odstępach. Poniższy rysunek przedstawia metodę prostokątów [Rys. 4]. Suma pól poszczególnych prostokątów reprezentuje zcałkowaną wartość pod krzywą. Im węższe są pola prostokątów, tym dokładniejsza staje się metoda.



Rys. 4 Metoda prostokątów

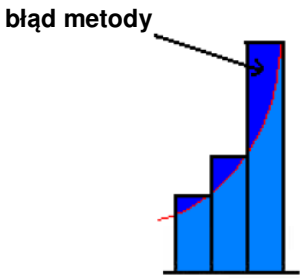
$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i)\,\Delta x$$

gdzie:

$$\Delta x = \frac{b-a}{n}$$

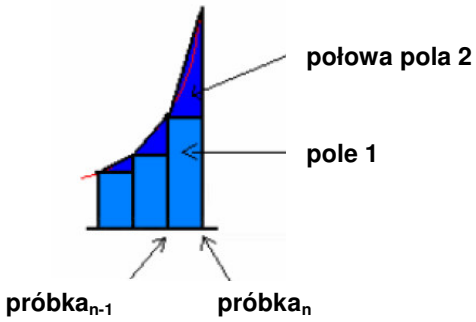
Naturalnie nie można uzyskać zbyt dużych częstotliwości pomiarów (wąskich prostokątów), gdyż ograniczeniem są przetworniki oraz układy przetwarzające np. mikrokontrolery.

Gdy częstotliwości sygnału będą zbliżone do częstotliwości pomiarów, błędy metody prostokątów zaczną mieć duże znaczenie [Rys 5].



Rys. 5 Błąd metody prostokątów

Błąd ten można dość skutecznie wyeliminować stosując inną metodę całkowania, np. metodę trapezów. Różni się tym od poprzedniej, że dzieli prostokąt na dwa pola. Pierwsze to prostokąt, który w całości znajduje się pod krzywą, a drugie to połowa pola, przez które przebiega krzywa [rys. 6].



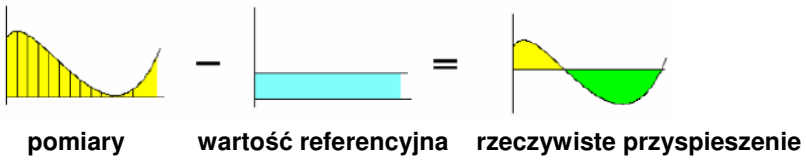
Rys. 6 Metoda trapezowa.

Pole z dwóch próbek jest obliczane w następujący sposób:

$$pole = \left(próbka_{n-1} + \left(\frac{próbka_n - próbka_{n-1}}{2} \right) \right) \times T$$

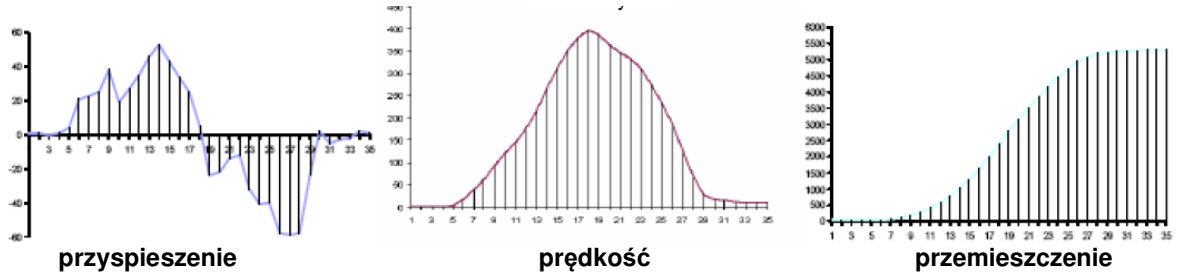
gdy odstępy czasowe są zawsze jednakowe można za T przyjąć 1.

Pozostaje jeszcze omówienie procesu kalibracji czujnika. Gdy nie działają na czujnik żadne przyspieszenia, to na wyjściu układu pojawi się sygnał prostokątny o wypełnieniu równym 50% (dla modelu ADXL202) lub napięcie równe połowie napięcia zasilania (dla modelu MMA7260). Należy zatem ustawić czujnik w pozycji, w której będzie pracował, zmierzyć sygnały na wyjściu i zapamiętać je jako wartości referencyjne. Po przeprowadzeniu kalibracji, od każdego następnego pomiaru należy odjąć wartość referencyjną uzyskując przyspieszenie bez stałej składowej [Rys 7].



Rys. 7 Usunięcie stałej składowej.

Ostatecznie stosując powyższy algorytm powinno się uzyskać przebiegi jak na rysunku 8.



Rys. 8 Uzyskanie prędkości i przemieszczenia.

4.2 Praktyczna implementacja metody.

Poniżej znajdują się fragmenty kodu realizującego powyższy algorytm:

Struktura zmiennych dla jednej osi:

```
struct typ_akcelerometr {
      signed int A_X;              /* aktualna wartość przyspieszenia */
      signed int V_X;              /* aktualna wartość prędkości */
      signed long I_A_X;           /* suma z kilku próbek */
      signed long I_V_X;           /* suma przyspieszeń chwilowych */
      signed int  A_X_old;         /* poprzednia wartość przyspieszenia */
      signed long A_X_zero;        /* wartość referencyjna  */
      signed int  V_window         /* próg */
      signed int  A_window;        /* próg */
      signed int  A_statex;        /* stan pomiaru */
      };

struct typ_akcelerometr accel;
```

Kod realizujący pojedyncze całkowanie – uzyskanie prędkości (MMA7260):

```
interrupt void int_PIT()
{
accel.A_X =((signed int) QREC[1]); /* odczytuje wartosc przyspieszenia */
/* odjecie odpomiaru wartosci referencyjnej */
accel.A_X =(signed int)(accel.A_X - accel.A_X_zero);
/* obliczenie pól */
accel.I_V_X+=((accel.A_X+((accel.A_X-settings.A_X_old)>>1)));
/* przechowanie aktualnej wartości pomiaru */
```

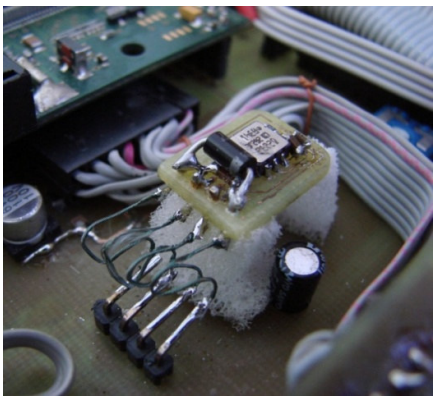
```
accel.A_X_old= accel.A_X;
}
```

Przedstawiona powyżej prosta metoda pozyskiwania prędkości i przemieszczenia, w praktyce nie działa efektywnie. Użyte czujniki charakteryzują się sporymi szumami na wyjściu, a także dużym wpływem temperatury na pomiar. Dodatkowo, jeżeli sensory umieszczone zostaną na platformie mobilnej, to w trakcie ruchu pojawią się spore zakłócenia. Drgania pochodzące od napędów (np. regulatory PID) oraz powierzchnia, po której porusza się robot, są źródłami sporych oscylacji na wyjściu akcelerometru. Jak wiadomo akcelerometry wykrywają także statyczne przyspieszenia. A zatem kolejnym problemem pojawiającym się w trakcie ruchu platformy jest wpływ grawitacji i sił odśrodkowych na pomiar. Wystarczy, że robot wjedzie na pochyłą powierzchnię, to zmieni się kąt działania siły grawitacji, a czujnik wskaże stałą niezerową wartość przyspieszenia. Z kolei jazda po łuku jest przyczyną powstawania sił odśrodkowych. I podobnie jak w przypadku wpływu grawitacji pomimo, że robot będzie poruszał się jednostajnie akcelerometr również wskaże stałą niezerową wartość przyspieszenia.

Jak widać, nie wystarczy stosować jedynie całkowanie sygnału, aby prawidłowo uzyskać wartości prędkości czy przemieszczenia robota. Niezbędne okazuje się stosowanie odpowiednich filtrów oraz innych zabiegów sprzętowych oraz programowych.

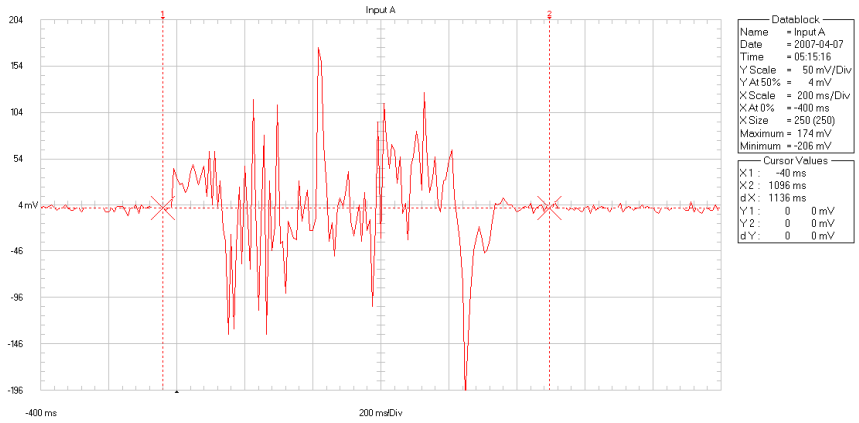
Mechaniczny filtr dolnoprzepustowy.

Aby stłumić drgania mechaniczne platformy, płytkę akcelerometru umieszczono na małej gąbce [Fot. 3]. Połączenia z płytą główną wykonano z elastycznego, nie łamiącego się przewodu. Dodatkowo moduł dociążono metalową nakrętką. Gąbka ta odgrywa jeszcze jedną ważną rolę. W przypadku, gdy robot uderzy w przeszkodę, pojawia się bardzo krótko trwały impuls na wyjściu akcelerometru. Jest on trudny do zmierzenia i powoduje powstawanie błędów podczas całkowania sygnału. Dzięki zainstalowaniu czujnika na gąbce, zwiększa się jego bezwładność, a zmiany przyspieszeń stają się mniej gwałtowne.

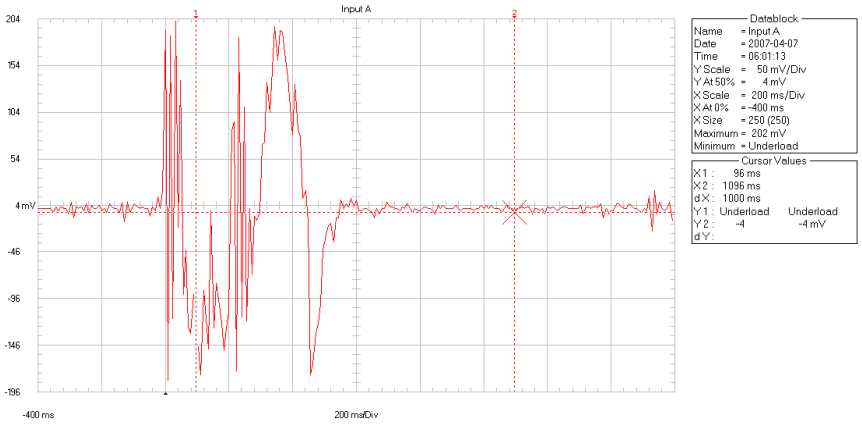


Fot. 3 Montaż modułu na gąbce.

Poniżej przedstawiono przebiegi napięć na wyjściu akcelerometru. Można zaobserwować, że sygnał na drugim wykresie jest mniej zakłócony.



Rys. 9 Przebieg napięcia na wyjściu czujnika podczas ruchu modułem po płycie biurka (MMA7260).



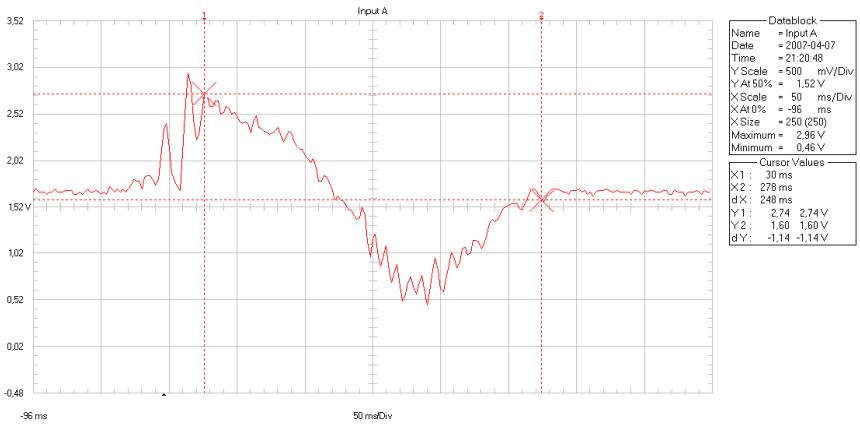
Rys. 10 Przebieg napięcia na wyjściu czujnika zainstalowanego na gąbce podczas ruchu modułem po płycie biurka (MMA7260).

Sprzętowy filtr dolnoprzepustowy.

Oprócz mechanicznego filtru, w postaci gąbki, zastosowano także dolnoprzepustowy filtr RC. Częstotliwość graniczną dla filtrów RC obliczono ze wzoru:

$$f_g = \frac{1}{2\pi RC}$$

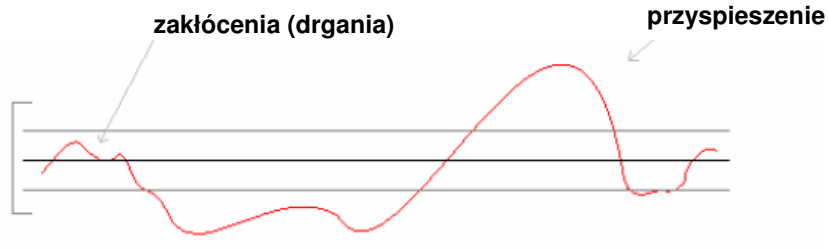
W projekcie częstotliwość graniczną dobierano eksperymentalnie. Niestety, im była ona mniejsza, tym słabszy był sygnał uzyskiwany na wyjściu czujnika. Robot porusza się z różnymi prędkościami - zależy to od sterowania. Ciężko więc dobrać wartości elementów filtru optymalnie. W projekcie przyjęto, że robot nie zmienia prędkości z przyspieszeniem, którego wartość jest poniżej pewnej granicy, przy której zmiany te nie są mierzalne.



Rys. 11 Przebieg napięcia na wyjściu czujnika zainstalowanego na gąbce i zwiększoną pojemnością kondensatora C z 100nF na 470nF (MMA7260).

Programowa eliminacja szumów i drgań mechanicznych.

Trudno jest całkowicie wyeliminować drgania mechaniczne przy pomocy samej gąbki. Konieczne jest jeszcze zastosowanie progowania sygnału. Małe przyspieszenia, których wartość znajduje się poniżej pewnej granicy, nie są użyte w procesie całkowania [Rys. 12].



Rys. 12 Progowanie sygnału.

Kod programu realizujący progowanie:

```
/* pole window przechowuje wartość progu */
if ((accel.A_X<=accel.A_window)&&(accel.A_X>=-accel.A_window)) accel.A_X=0;
```

Kalibracja czujnika.

Aby poprawnie wykalibrować czujnik, nie wystarczy wykonać pojedynczy pomiar. Dla uzyskania jak najlepszej wartości referencyjnej dokonuje się wielu pomiarów, a następnie je uśrednia.

Kod programu realizujący kalibrację (MMA7260):

```
void accel_calibrate() {
int i=0;
```

```
/* 4096 razy pobiera wyniki sumuje z poprzednim */
for (i=0;i<4096;i++)
      {
      SPCR1 |= 0x8000;
      while(!(SPSR & QsmSPIF));           /* czeka na koniec kolejki */
      accel.A_X_zero+=((signed long) QREC[1]);  /* dodaje */
      }
accel.A_X_zero = accel.A_X_zero>>12; /* dzieli przez liczbę pomiarów */
}
```

Programowy filtr dolnoprzepustowy.

Pomimo zastosowania gąbki oraz sprzętowego filtru RC pomiary wciąż bardzo są zaszumione. Należy zaimplementować uśrednianie pojedynczych próbek. Przy częstotliwości próbkowania 1kHz najlepszy efekt udało się uzyskać uśredniając od 32-64 próbek.

Kod realizujący uśrednianie pomiarów (MMA7260):

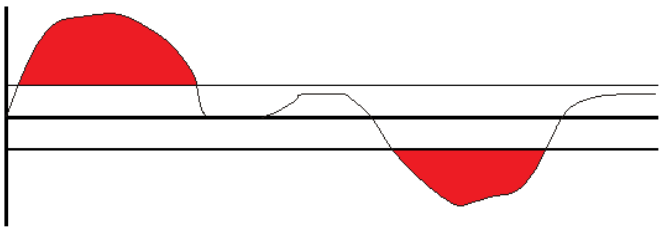
```
if (accel.A_statex<64)
{
      accel.I_A_X+=((signed long) QREC[1]);
      SPCR1 |= 0x8000;
      accel.A_statex++;
}
else
{
      accel.I_A_X=( accel.I_A_X>>6);
      accel.A_X =(signed int)( accel.A_X_zero - accel.I_A_X);
      accel.I_A_X=0;
}
```

Programowa realizacja korekcji zera .

Często zdarza się, że pomimo stosowania wszystkich powyższych zabiegów i tak dochodzi do powstawania błędów podczas całkowania. Przyczyn tych niepożądanych działań jest wiele. Robot startuje z miejsca, w którym został wykalibrowany a zatrzymuje się nieco dalej, gdzie powierzchnia już jest pod delikatnym kątem. Wpływ niewielkich zmian kąta działania siły grawitacji, w trakcie jazdy został zniwelowany np. przez progowanie. Jednakże podczas hamowania na powierzchni znajdującej się pod kątem na robota działały już inne siły [Rys. 13]

Przykład:

Platforma w miejscu A została wykalibrowana, a zatem na robota działa przyspieszenie równe 0. Robot rusza z przyspieszeniem równym 10, które trwa np. jedną sekundę. W trakcie jazdy robot wjechał na delikatne wzniesienie. Czujnik wskazał wartość przyspieszenia równą 2. Próg, przy którym pomiary zaczną być brane pod uwagę ustawiony jest na 3. Załóżmy, że robot hamuje w tym samym czasie, z takim samym przyspieszeniem tylko, że z przeciwnym znakiem. Wartość siły jaką wskaże czujnik będzie równa 10-2=8. Prędkość nie zostanie wtedy scałkowana do zera.



Rys. 13 Dryft zera – pole nad osia jest większe niż pole pod osią.

Zdecydowano zaimplementować także progowanie prędkości. Założymy, że robot nie porusza się z prędkościami mniejszymi niż pewna wartość graniczna. Wskazania poniżej tej wartości można wtedy traktować jako błędy i interpretować jako zerowe. Można tak uczynić jedynie w chwilach, kiedy sterowanie jest równe zero i przyspieszenia są równe zero. Jeżeli w trakcie jazdy i tak wystąpią błędy metody a wartość prędkości będzie nie prawdziwa, należy zatrzymać robota, tak aby nie działały na niego żadne siły. Po upływie czasu algorytm korygujący sam wyzeruje wskazanie.

Kod realizujący korekcję zera:

```
/* progowanie wartości prędkości */
if ((accel.V_A_X<=accel.V_window)&&
(accel.V_A_X>=-accel.V_window)&&(accel.A_X==0)){ accel.V_A_X=0;}

/* zerowanie prędkości jeżeli sterowanie jest =0 i nie działają na robota
żadne siły (z przyspieszeniem) */
if ((accel.A_X==0)&&( sterowanie==0)) end_movex++; else end_movex=0;
if (end_movex>=100) { accel.V_A_X=0; end_movex=0;}
```

Program

Kod realizujący pojedyncze całkowanie z uwzględnieniem wszystkich powyższych modyfikacji (MMA7260):

```
interrupt void int_PIT()
{
static int end_movex;

if (accel.A_statex<64)
{
      accel.I_A_X+=((signed long) QREC[1]);
      SPCR1 |= 0x8000;
      accel.A_statex++;
}
else
{
      accel.I_A_X=( accel.I_A_X>>6);
      accel.A_X =(signed int)( accel.A_X_zero - accel.I_A_X);
      accel.I_A_X=0;
      accel.A_statex=0;
}

if ((accel.A_X<=accel.A_window)&&(accel.A_X>=-accel.A_window)) accel.A_X=0;
accel.I_V_X+=((signed long)(accel.A_X+((accel.A_X-settings.A_X_old)>>1)));
```

```
accel.V_X=((signed int)( accel.I_V_X>>5)); /* przeskalowanie */

if ((accel.V_A_X<=accel.V_window)&&
   (accel.V_A_X>=-accel.V_window)&&(accel.A_X==0)){ accel.V_A_X=0;}
if ((accel.A_X==0)&&( sterowanie==0)) end_movex++; else end_movex=0;
if (end_movex>=100) { accel.V_A_X=0; end_movex=0;}

}
```

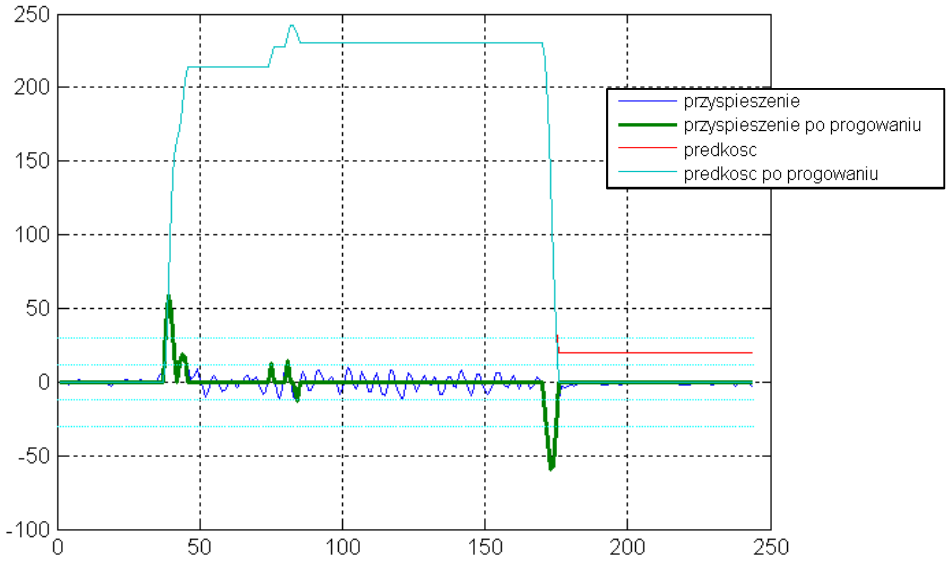
5. Wyniki badań (ADXL202)

Poniżej umieszczono kilka wykresów ilustrujących powyższe modyfikacje.

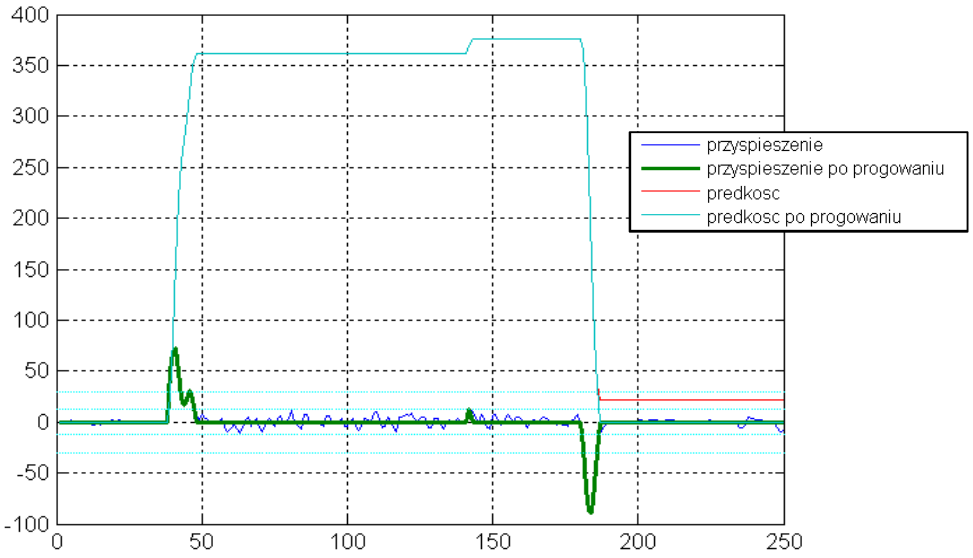
uśrednienie 8 próbek,

próg dla przyspieszenia = +/-12 ,

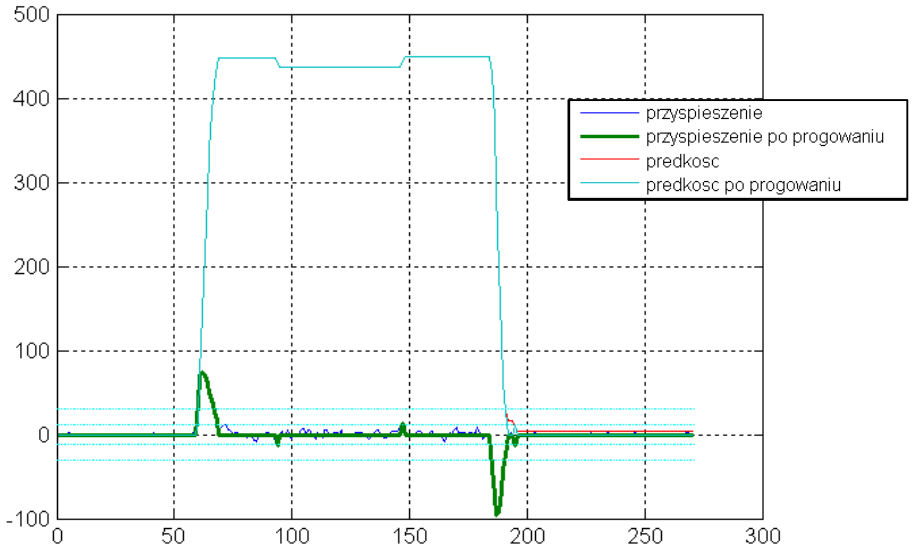
próg dla prędkości = +/-30



Rys. 14 Przebiegi przyspieszeń i prędkości. Zadana prędkość = 5.



Rys. 15 Przebiegi przyspieszeń i prędkości. Zadana prędkość =8.

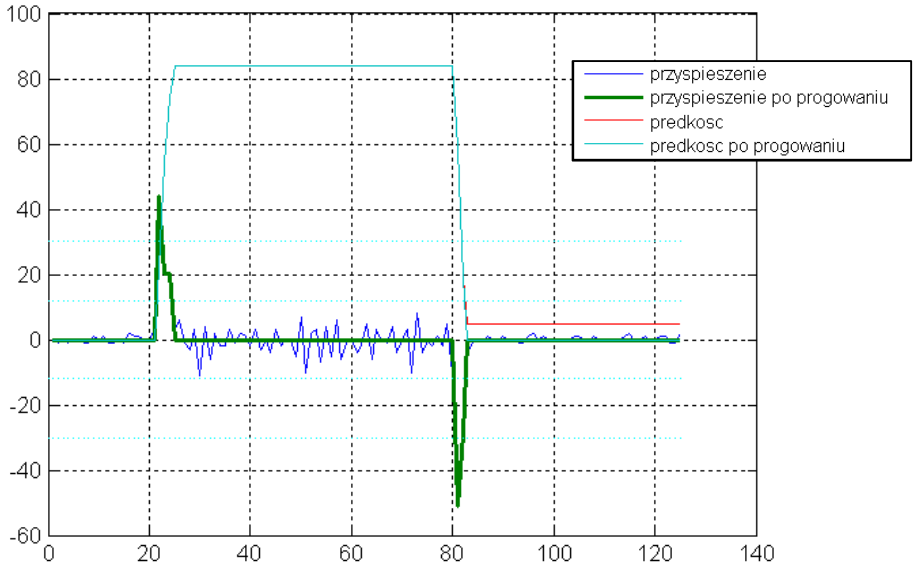


Rys. 16 Przebiegi przyspieszeń i prędkości. Zadana prędkość =10.

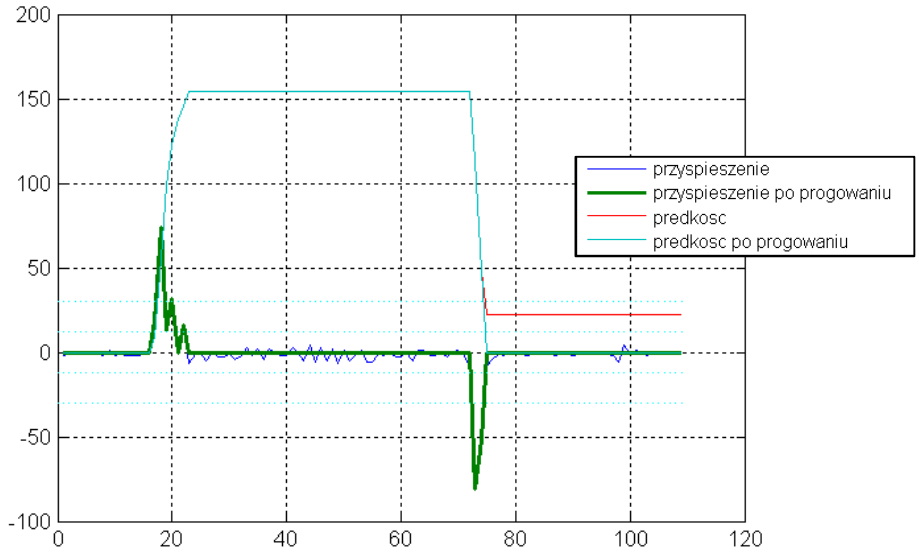
uśrednienie 32 próbek,

próg dla przyspieszenia = +/-12 ,

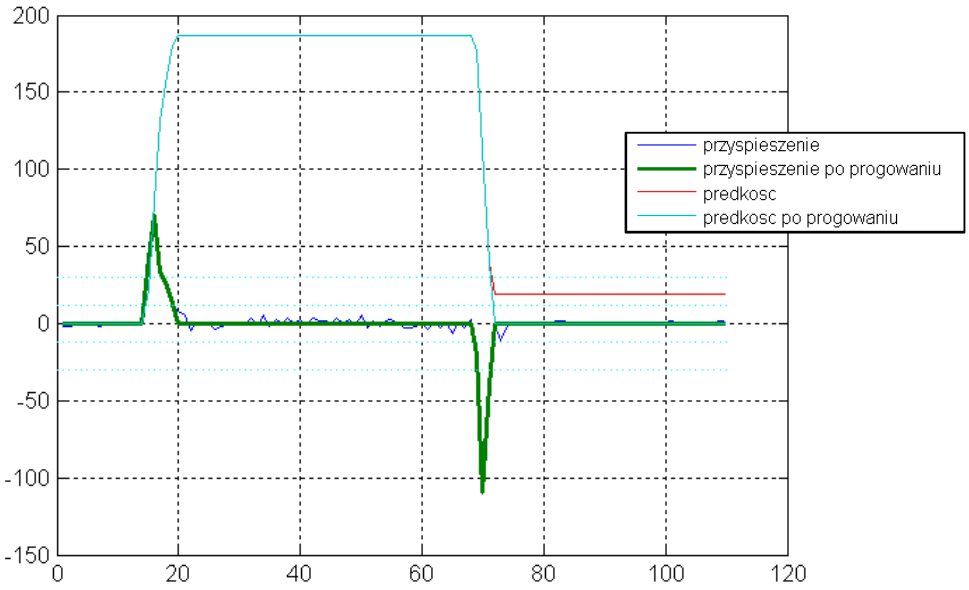
próg dla prędkości = +/-30



Rys. 17 Przebiegi przyspieszeń i prędkości. Zadana prędkość =5.



Rys. 18 Przebiegi przyspieszeń i prędkości. Zadana prędkość =8.

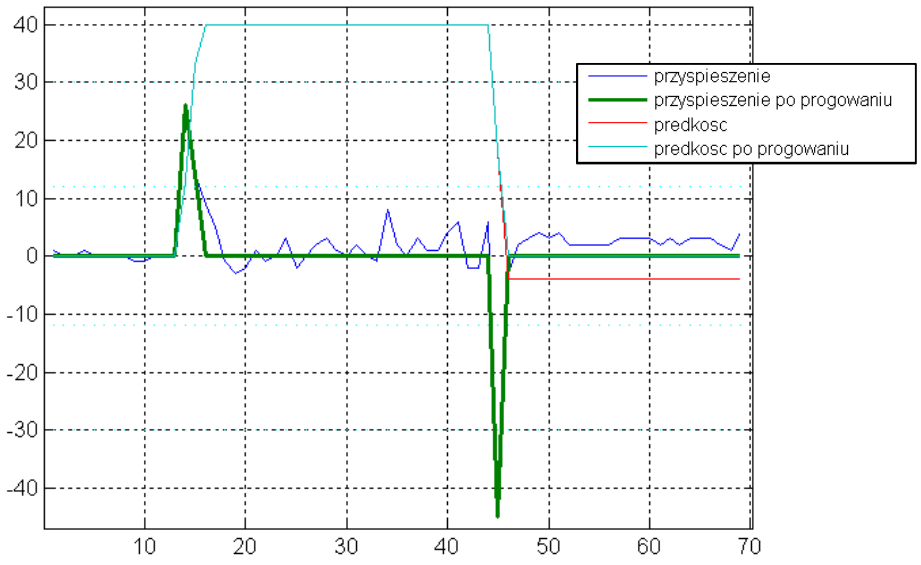


Rys. 19 Przebiegi przyspieszeń i prędkości. Zadana prędkość =10.

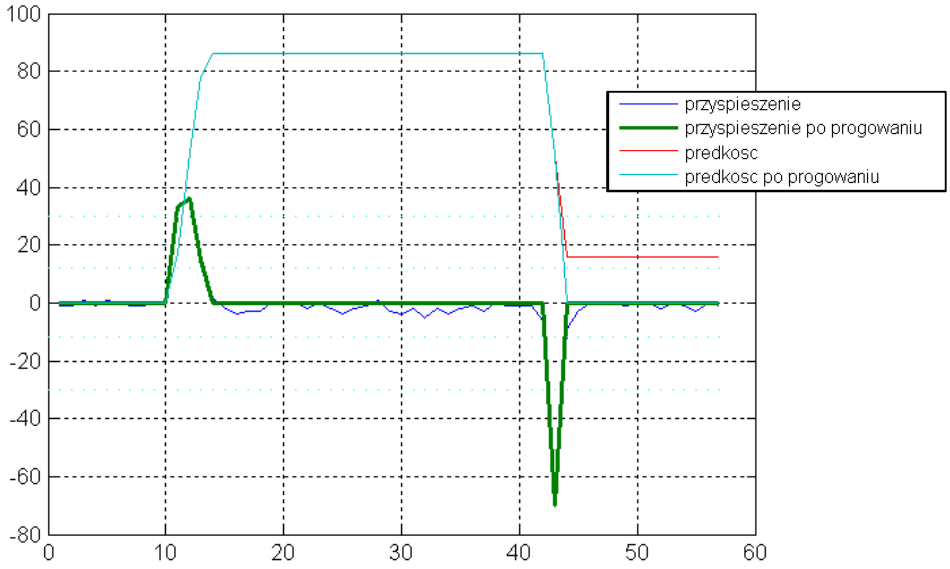
uśrednienie 64 próbek,

próg dla przyspieszenia = +/-12 ,

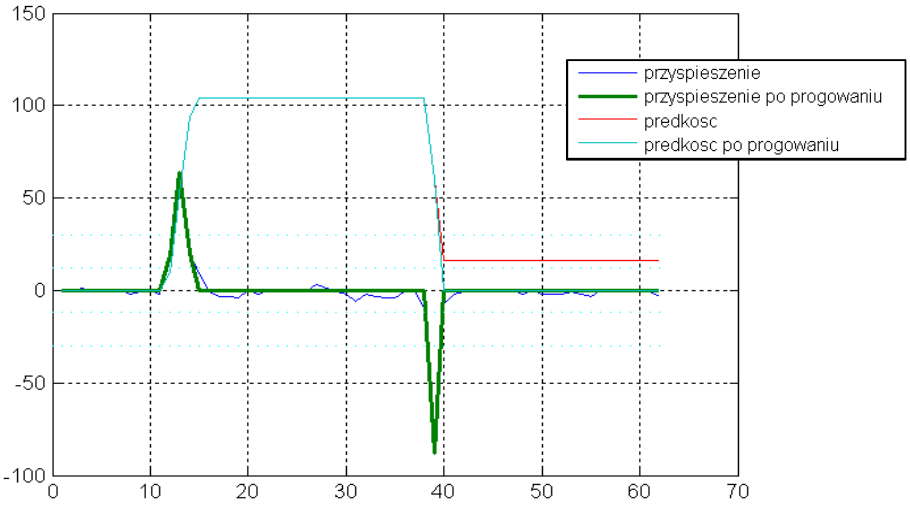
próg dla prędkości = +/-30



Rys. 20 Przebiegi przyspieszeń i prędkości. Zadana prędkość =5.



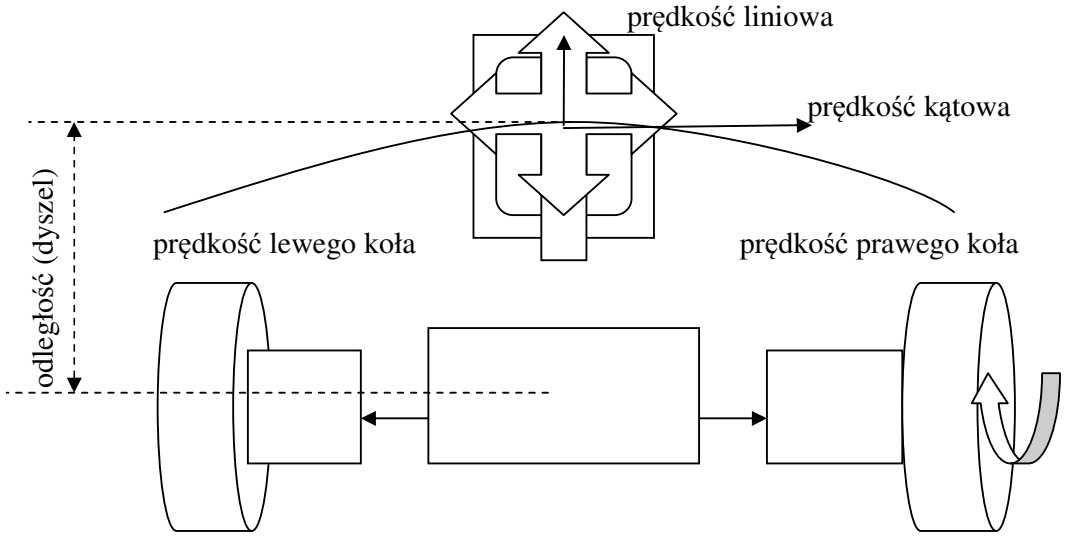
Rys. 21 Przebiegi przyspieszeń i prędkości. Zadana prędkość =8.



Rys. 22 Przebiegi przyspieszeń i prędkości. Zadana prędkość =10.

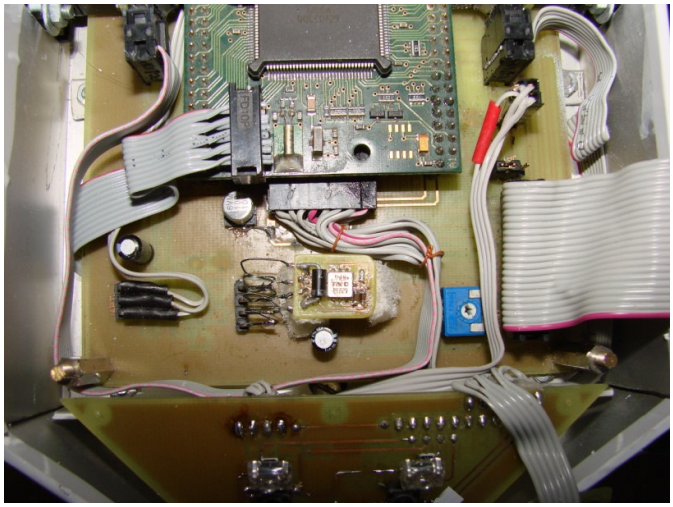
6. Przykład zastosowania

Czujnik umieszczono na robocie typu unicycle klasy (2,0) „BlueScreen” [9]. Zainstalowano go w niewielkiej odległości od osi kół robota. Dzięki temu możliwy był pomiar prędkości kątowej i liniowej robota. Odległość ta nie powinna być zbyt duża ponieważ podczas obracania się platformy w miejscu siła odśrodkowa będzie wprowadzać błędy pomiaru.



Rys 23 Schemat blokowy platformy.

Fot 4 Widok platformy.



Fot 5 Widok zainstalowanego czujnika.

7. Wyniki badań czujnika na poruszającym się robocie (ADXL202).

Poniżej przedstawiono wyniki badań ruchu robota. Porównano zadane prędkości z pomiarami po przetworzeniu. Pierwsze badanie polegało na wykalibrowaniu robota w miejscu startu, zadaniu prędkości i odczytaniu wskazów. Drugie to sekwencja trzech różnych sterowań.

Warunki początkowe:

Vlin=0 – prędkość liniowa

Vkąt=0 – prędkość kątowa

zadane prędkości Vlin=6, Vkąt=0	zadane prędkości Vlin=7, Vkąt=0	zadane prędkości Vlin=15 Vkąt=0	zadane prędkości Vlin=-15 Vkąt=0	zadane prędkości Vlin=8, Vkąt=5	zadane prędkości Vlin=9, Vkąt=4
Vlin=6, Vkąt=0	Vlin=8, Vkąt=0	Vlin=18, Vkąt=0	Vlin=-18, Vkąt=0	Vlin=10, Vkąt=6	Vlin=12, Vkąt=0
Vlin=0, Vkąt=0	Vlin=7, Vkąt=0	Vlin=18, Vkąt=0	Vlin=-16, Vkąt=0	Vlin=10, Vkąt=6	Vlin=12, Vkąt=6
Vlin=6, Vkąt=0	Vlin=6 Vkąt=0	Vlin=17, Vkąt=0	Vlin=-16, Vkąt=0	Vlin=10, Vkąt=0	Vlin=10, Vkąt=0
Vlin=7, Vkąt=0	Vlin=7, Vkąt=0	Vlin=17, Vkąt=0	Vlin=-17, Vkąt=0	Vlin=10, Vkąt=0	Vlin=10, Vkąt=0
Vlin=7, Vkąt=0	Vlin=8, Vkąt=0	Vlin=16, Vkąt=0	Vlin=-19, Vkąt=0	Vlin=10, Vkąt=7	Vlin=10, Vkąt=0
Vlin=7, Vkąt=0	Vlin=6, Vkąt=0	Vlin=17, Vkąt=0	Vlin=-18, Vkąt=0	Vlin=10, Vkąt=6	Vlin=10, Vkąt=0
Vlin=6, Vkąt=0	Vlin=7, Vkąt=0	Vlin=20, Vkąt=0	Vlin=-18, Vkąt=0	Vlin=7, Vkąt=6	Vlin=10, Vkąt=0
Vlin=5, Vkąt=0	Vlin=7, Vkąt=0	Vlin=17, Vkąt=0	Vlin=-16, Vkąt=0	Vlin=9, Vkąt=6	Vlin=10, Vkąt=0
Vlin=7, Vkąt=0	Vlin=8, Vkąt=0	Vlin=20, Vkąt=0	Vlin=-18, Vkąt=0	Vlin=8, Vkąt=6	Vlin=8, Vkąt=0

Tabela 3 Wyniki pomiarów ruchu robota na wprost i po łuku (zawsze po kalibracji, start z tego samego miejsca)

Warunki początkowe:

Vlin=0 – prędkość liniowa

Vkąt=0 – prędkość kątowa

Sterowania zadawano sekwencyjnie krok po kroku.

Sterowanie 1 zadane prędkości Vlin=9, Vkąt=0	Sterowanie 2 zadane prędkości Vlin=-9, Vkąt=0	Sterowanie 3 zadane prędkości Vlin=0 Vkąt=0
Vlin=11, Vkąt=0	Vlin=-10, Vkąt=0	Vlin=0, Vkąt=0
Vlin=12, Vkąt=0	Vlin=-10 Vkąt=0	Vlin=0, Vkąt=0
Vlin=11, Vkąt=0	Vlin=-13 Vkąt=0	Vlin=4, Vkąt=0
Vlin=11, Vkąt=0	Vlin=-11, Vkąt=0	Vlin=0, Vkąt=0
Vlin=11, Vkąt=0	Vlin=-11, Vkąt=0	Vlin=0, Vkąt=0
Vlin=11, Vkąt=0	Vlin=-11, Vkąt=0	Vlin=0, Vkąt=0
Vlin=12, Vkąt=0	Vlin=-11, Vkąt=0	Vlin=0, Vkąt=0
Vlin=10, Vkąt=0	Vlin=-12, Vkąt=0	Vlin=0, Vkąt=0
Vlin=11, Vkąt=0	Vlin=-12, Vkąt=0	Vlin=0, Vkąt=0

Tabela 4 Wyniki pomiarów ruchu robota na wprost, wstecz i zatrzymanie. (zawsze po kalibracji, start z tego samego miejsca)

Warunki początkowe:

Vlin=0 – prędkość liniowa

Vkąt=0 – prędkość kątowa

Sterowania zadawano sekwencyjnie krok po kroku.

Sterowanie 1 zadane prędkości Vlin=15, Vkąt=0	Sterowanie 2 zadane prędkości Vlin=-15, Vkąt=0	Sterowanie 3 zadane prędkości Vlin=0 Vkąt=0
Vlin=17, Vkąt=0	Vlin=-13, Vkąt=0	Vlin=0, Vkąt=0
Vlin=16, Vkąt=0	Vlin=-17 Vkąt=0	Vlin=0, Vkąt=0
Vlin=20, Vkąt=0	Vlin=-15 Vkąt=0	Vlin=0, Vkąt=0
Vlin=13, Vkąt=0	Vlin=-21, Vkąt=0	Vlin=-5, Vkąt=0
Vlin=15, Vkąt=0	Vlin=-19, Vkąt=0	Vlin=-5, Vkąt=0

Vlin=14, Vkąt=0	Vlin=-24, Vkąt=0	Vlin=8, Vkąt=0
Vlin=16, Vkąt=0	Vlin=-19, Vkąt=0	Vlin=-4, Vkąt=0
Vlin=17, Vkąt=0	Vlin=-19, Vkąt=0	Vlin=0, Vkąt=0
Vlin=16, Vkąt=0	Vlin=-19, Vkąt=0	Vlin=0, Vkąt=0

Tabela 5 Wyniki pomiarów ruchu robota na wprost, wstecz i zatrzymanie. (zawsze po kalibracji, start z tego samego miejsca)

8. Podsumowanie

Zakupione na potrzeby realizacji opisywanego projektu akcelerometry, charakteryzowały się słabymi parametrami. Spore szumy, wpływ temperatury oraz „dryft zera" obarczały uzyskiwane pomiary sporymi błędami. Pomimo wielu zabiegów, całkowanie sygnału z akcelerometrów wciąż nie dawało jednoznacznych informacji o aktualnej prędkości czy też położeniu. Nie mniej jednak, udawało się z pewnym przybliżeniem oszacować tendencje ruchu platformy oraz siły jakie na nią działają w danym momencie.

Niezwykle trudnym przypadkiem okazał się ruch platformy po łuku oraz po płaszczyznach pochyłych. Działające siły odśrodkowe oraz siła grawitacji zupełnie zmylały czujnik, który pomimo ruchu jednostajnego wskazywał wciąż niezerową wartość przyspieszenia. Przyjęto więc, że platforma porusza się po płaszczyznach prostopadłych do kierunku działania siły grawitacji.

Dobrym pomysłem mogłoby być zainstalowanie żyroskopów w systemie mierzących prędkość skrętu platformy oraz odchylenie od powierzchni. Pomogłoby to poprawnie mierzyć przyspieszenia podczas jazdy po łuku.

W ramach projektu podjęto próbę wyznaczenia nachylenia robota do wektora grawitacji na podstawie pomiarów osi Z (MMA7260). Niestety rozdzielczość czujnika nie jest na tyle wysoka, aby poprawnie móc dokonać obliczeń. Zwłaszcza, że zmiany na wyjściu czujnika mierzącego przyspieszenie ziemskie mają postać: $e = g\cos\alpha$, a pochodna cosinusa na początku jest bardzo mała. Zainstalowanie czujnika pod kątem 45^{o} być może polepszyłoby efekt.

9 Literatura

[1] *MMA7260Q: ±1.5g - 6g Three Axis Low-g Micromachined Accelerometer, Freescale.*

[2] *ADXL202E: Low-Cost 2 g Dual-Axis Accelerometer, Analog Devices.*

[3] Wnuk M. *Moduł z Mikrokontrolerem MC68332*, *Raport ICT serii 07/2004, Wrocław, 2004*

[4] *TPU Time Processor Unit Reference Manual, Freescale Semiconductors Inc.*

[5] *QSM Queued Serial Module Reference Manual, Freescale Semiconductors Inc.*

[6] *Period/Pulse-Width Accumulator TPU Function (PPWA), Freescale Semiconductors Inc.*

[7] *TLV2556: 12-BIT, 200-KSPS, 11 Channel, Low power, Serial ADC with internal reference, Texas Instruments.*

[8] *Implementing Positioning Algorithms Using Accelerometers. AN3397 Rev 0, 02/2007, Freescale Semiconductors Inc.*

[9] Kędzierski J. Ostrowski E. *Robot mobilny klasy (2,0) „Blue Screen”.* http://www.konar.ict.pwr.wroc.pl/infopage.php?id=13