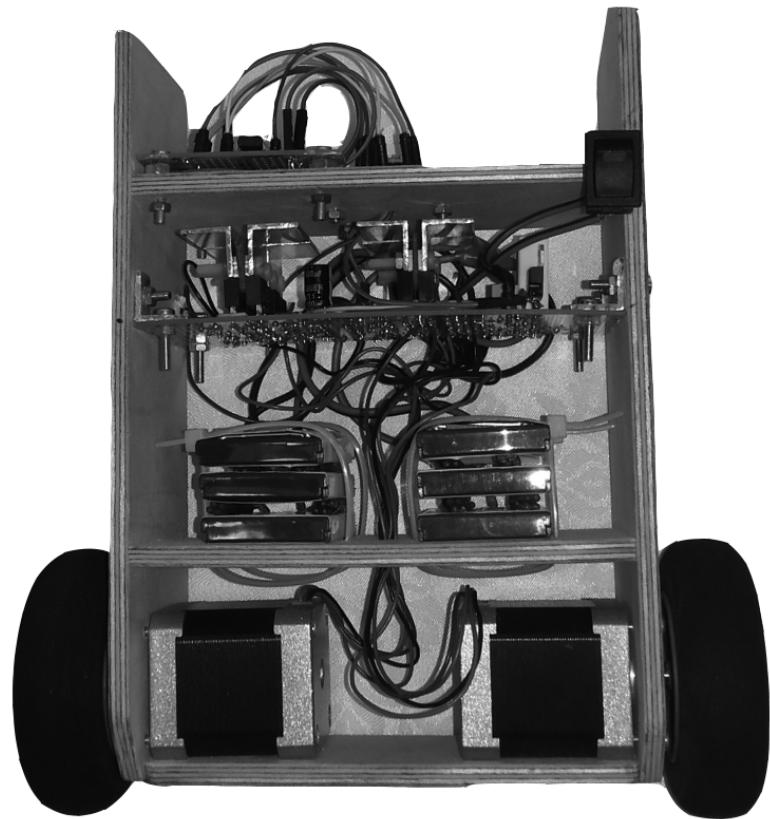


Dokumentacja techniczna dwukołowego robota balansującego

Automatyka i robotyka, elektryczny
Mateusz Staszków, 261006
mateuszstasz@gmail.com

Politechnika Warszawska, 2016



Rysunek 1: Zdjęcie z 18.02.2016

Spis treści

1 Wstęp	4
2 Mechanika	5
2.1 Parametry ogólne	5
2.2 Model w AutoCADzie	5
2.3 Obudowa	6
2.4 Napęd	7
2.5 Silniki	7
2.6 Koła	7
2.7 Matematyczny model układu	8
2.7.1 Obliczenie środka masy robota	8
2.7.2 Obliczenie momentu bezwładności	9
2.7.3 Wyznaczenie równań liniowych układu:	9
2.7.4 Opis układu w przestrzeni zmiennych stanu:	10
3 Elektronika	11
3.1 Zasilanie	11
3.1.1 Akumulatory litowo-jonowe	11
3.1.2 Stabilizacja napięcia	11
3.1.3 Bezpieczeństwo układu	12
3.2 Sterownik silników krokowych	13
3.2.1 Mikrokontroler AVR ATmega8	13
3.2.2 Płytnica uniwersalna	14
3.2.3 Własny projekt mostków H	15
3.2.4 Schemat elektroniczny	17
3.3 Moduł czujników MPU-6050 - żyroskop i akcelerometr	17
3.4 Wyświetlacz LCD 16x2 zgodny ze standardem HD44780	18
3.5 Układ regulacji położenia	20
3.5.1 Płytnica uniwersalna	20
3.5.2 Schemat elektroniczny	21
4 Teoria sterowania	22
4.1 Wyznaczenie sterowalności i obserwowalności układu otwartego	22
4.2 Wyznaczenie stabilności i odpowiedź skokowa układu otwartego	22
4.3 Projekt obserwatora Luenbergera pełnego rzędu	23
4.4 Projekt regulatora liniowo-kwadratowego (LQR)	25
4.5 Projekt regulatora liniowo-kwadratowego-Gaussa (LQG)	26
4.6 Kod Matlaba	28
5 Oprogramowanie	31
5.1 Sterownik silników krokowych	31
5.2 Obsługa czujników MPU-6050	36
5.3 Obsługa wyświetlacza HD44780	51
5.4 Regulacja położenia	53
6 Podsumowanie	63
7 Niezrealizowane pomysły	64
7.1 Ładowarka do akumulatorów	64
7.2 Pomiar prądu i napięcia baterii przez ADC	64
7.3 Zdalne sterowanie przez telefon z Androidem	64

8 Kosztorys	65
9 Bibliografia	66

1 Wstęp

Dwukołowy robot balansujący działa na zasadzie wahadła odwróconego. Jego zadaniem jest ciągłe utrzymywanie pionu, z możliwie jak najmniejszymi odchyłami. Jest to układ niestabilny i wymaga automatycznej regulacji - zadanie to spełnia regulator predykcyjny LQG - liniowo-kwadratowy-Gaussa, a za sprzężenia zwrotne odpowiadają czujniki położenia, a dokładniej przyspieszenia - akcelerometr i prędkości kątowej - żyroskop. Do sterowania silnikami nie zostało użyte sprzężenie zwrotne w postaci miaru prędkości i położenia kół - ponieważ zamontowane silniki są krokowe i mają dość duży moment (w sumie ponad 1Nm), co pozwala z powodzeniem zastosować sterowanie w pętli otwartej. Sterownik silników, jak i regulator, cała logika robota wraz z konstrukcją mechaniczną zostały wykonane wyłącznie przez autora. Stosunkowo krótki czas na wykonanie projektu i trudności w postaci zaliczeń na końcu semestru nie pozwoliły na naukę wytrawiania i samodzielne wykonanie własnych płytka PCB, dlatego cała elektronika jest umieszczona na płytach uniwersalnych i połączona przewodami do płytka stykowej (wybór wynikał z tego, że aktualnie byłem w posiadaniu jedynie takich kabli). Większość elementów była wybierana ze względu na cenę - stąd 6 ogniw litowo-jonowych z Nokii 3310, czy tani moduł żyroskop+akcelerometr. Robot jest zaprojektowany tak, aby każdy element był jak najlżejszy i można było go łatwo wymienić - na przykład: baterie zamocowane na trytki, silniki krokowe jedynie przykręcane na śruby do obudowy, elektronika ze sterowaniem silników i regulacji automatycznej także na śruby, czy zastosowanie lekkich aluminiowych radiatorów do chłodzenia tranzystorów mocy. Przy każdym mikrokontrolerze (AVR do sterowania silnikami i drugi AVR do rozwiązywania równań różniczkowych Riccatiego z LQG w czasie rzeczywistym, a także do obsługi czujników i wyświetlacza) jest zbudowane złącze programatora, co umożliwia bezproblemowe aktualizowanie oprogramowania.

Problem stabilizacji położenia w pionie dotyczy wielu gałęzi przemysłu, na przykład takich jak: jednoosobowe transportowe pojazdy typu Segway na lotniskach i w centrach handlowych, układy balansujące na nierównych powierzchniach, czy roboty humanoidalne.



Rysunek 2: Segway



Rysunek 3: Robot humanoidalny firmy Honda



Rysunek 4: Robot balansujący na kuli

2 Mechanika

2.1 Parametry ogólne

Masa: 1.4 kg

Wymiary: 8cm x 15cm x 22cm

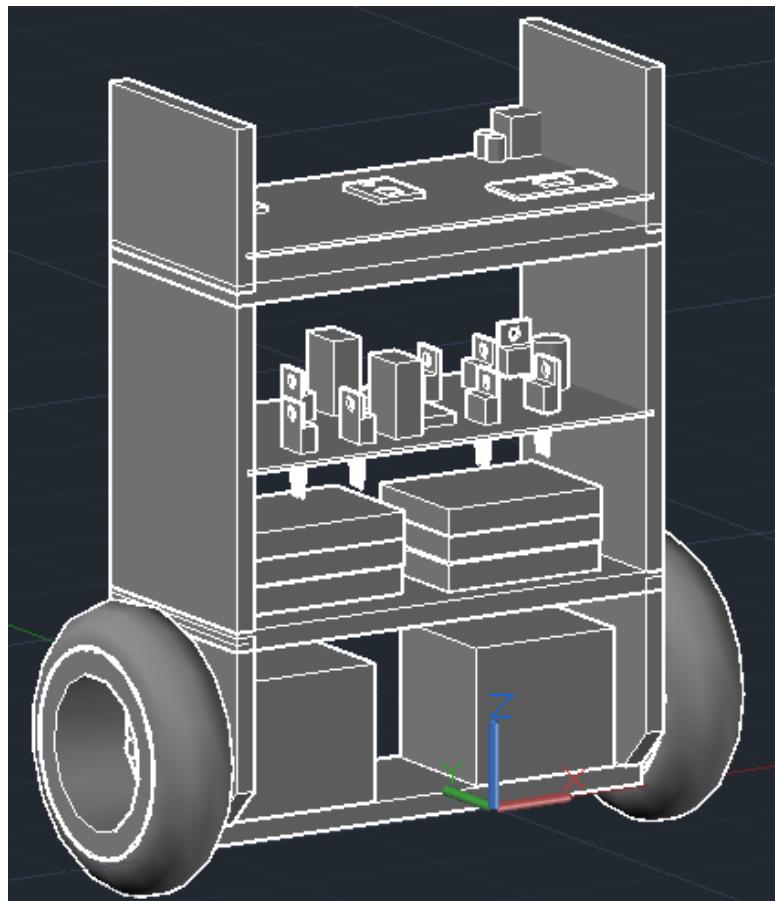
Moment bezwładności: 0.003 kgm^2

Wysokość środka masy: 6.5 cm od spodu dolnej półki

Środek masy robota znajduje się nieco ponad silnikami, (które stanowią ponad 50% wagi robota i znajdują się w dolnej części obudowy) w geometrycznym środku półki na akumulatory.

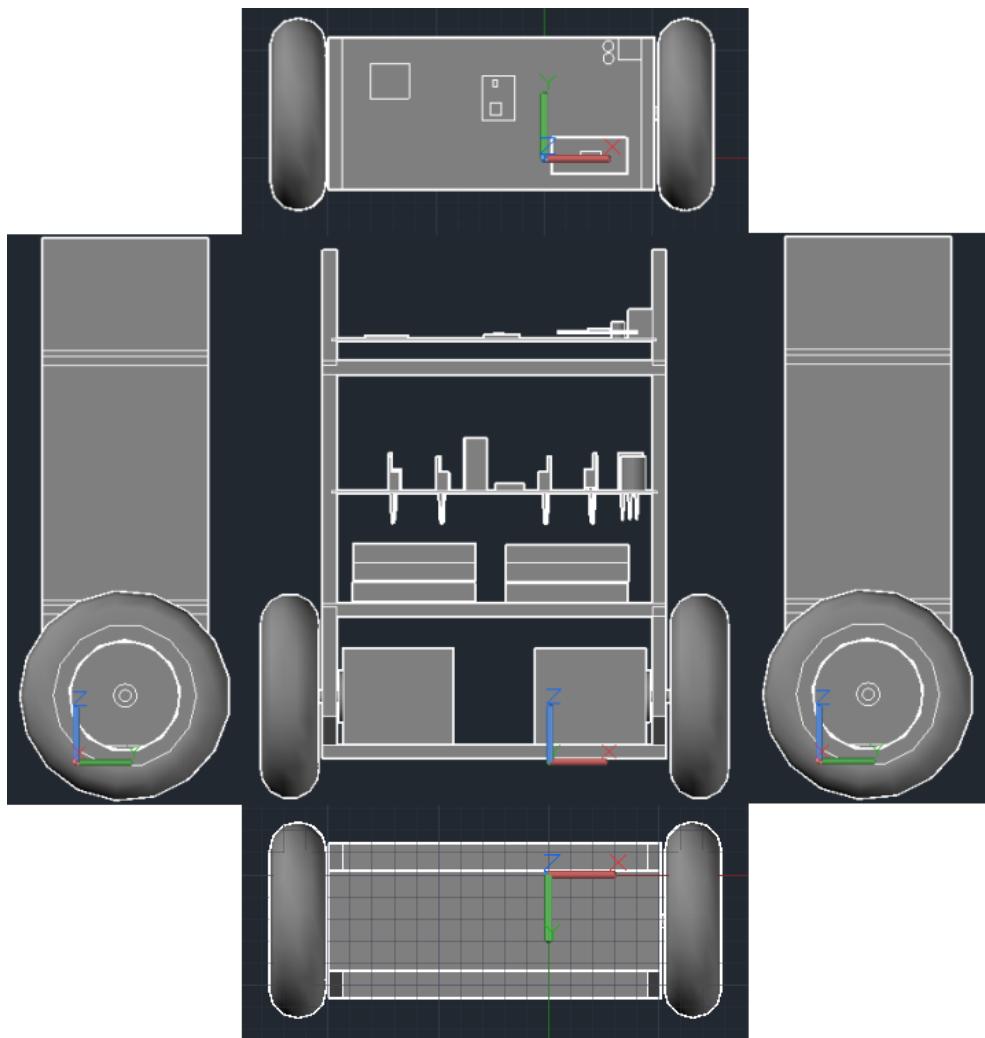
2.2 Model w AutoCADzie

Wstępny model robota został stworzony w programie Autodesk AutoCAD 2016. Model służył do początkowego ustalenia listy elementów potrzebnych do zakupu, a także do ustalenia wielkości kół czy szerokości obudowy.



Rysunek 5: Wstępny model robota

Rzuty modelu:



Rysunek 6: Rzuty: przedni, górny, dolny i boczne

2.3 Obudowa

Masa: ok. 0.2 kg

Wymiary: 7cm x 15cm x 22cm

Objętość drewna: ok. 360 cm^3

Gęstość użytego drewna: ok. $900 \frac{\text{kg}}{\text{m}^3}$

Obudowa jest w całości wykonana z drewna, a poszczególne elementy są połączone wkrętami. Składa się z czterech segmentów - kolejno od dołu:

- segment silników - odpowiadający za napęd robota i obniżający środek masy konstrukcji, silniki są przykręcane do ścian bocznych za pomocą śrub, co ułatwia ich wymianę. W ścianach bocznych umieszczono łożyska,
- część baterijna - odpowiadająca za zasilanie, akumulatory przymocowane są za pomocą wywiercenia dziur w półce i przełożenia plastikowych opasek zaciskowych - trytek, co umożliwia łatwą wymianę,
- część sterująca wyższymi prądami do 7 A, z zamontowanymi aluminiowymi radiatorami, całość znajdująca się na płytce uniwersalnej jest przykręcana do ścian bocznych śrubami,

- segment odpowiadający za regulację pozycji robota wyłącznie napięcia 0-5 V, z zamontowanym modułem z czujnikami i wyświetlaczem LCD, zamocowanie na śruby do półki.

2.4 Napęd

Z powodu dużego momentu obrotowego silników krokowych nie było potrzeby używania przekładni - koła są napędzane bezpośrednio z wałów silników, które opierają się na łożyskach (kulowych jednorzędowych) umieszczonych w ścianach bocznych. średnica całego łożyska to 10 mm, a średnica otworu to 5 mm.

2.5 Silniki

Nazwa: JK42HS48-1684

Masa: 0.34 kg

Wymiary: front 42 x 42 mm, długość korpusu 48 mm

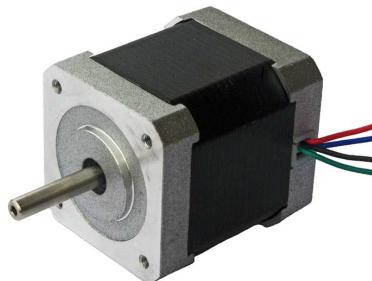
Moment obrotowy: 4.4 kg.cm

Krok: 1.8 stopnia

Prąd nominalny każdego uzwojenia: 1.68A

Napięcie: 2.8 V

Rezystancja uzwojenia: 1.65 ohm



Rysunek 7: Silnik krokowy

2.6 Koła

Masa: 20 g

Wymiary: Średnica: 80 mm, szerokość: 25 mm, otwór na wał silnika poszerzany wiertarką do 4.9 mm

Są to koła piankowe używane do modeli samolotów, charakteryzują się bardzo małą wagą.



Rysunek 8: Koło piankowe

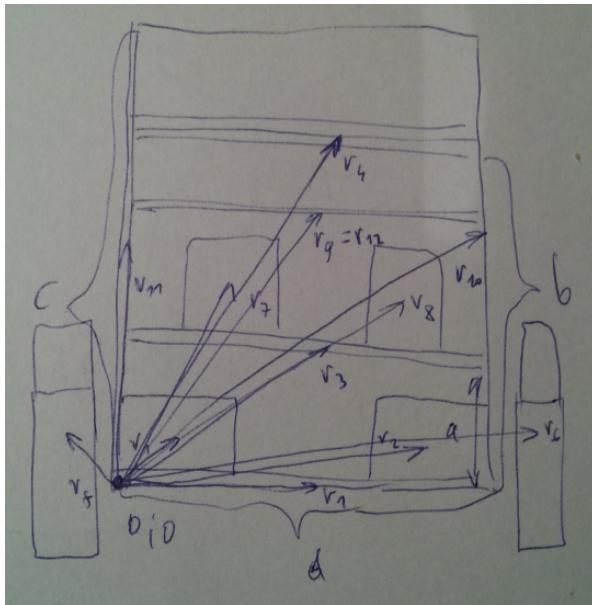
2.7 Matematyczny model układu

2.7.1 Obliczenie środka masy robota

Jako, że robot jest zbudowany symetrycznie w odniesieniu do szerokości i długości, obliczam środek masy jedynie ze względu na wysokość. Reszta wartości będzie się pokrywała ze środkiem geometrycznym bryły.

Przy obliczaniu jedynie przybliżyłem położenie i masę przewodów.

Rysunek poglądowy do zrozumienia oznaczeń wektorów:



Rysunek 9: wektory r środków geometrycznych poszczególnych obiektów

Środek masy został wyliczony w Matlabie i wynosi:

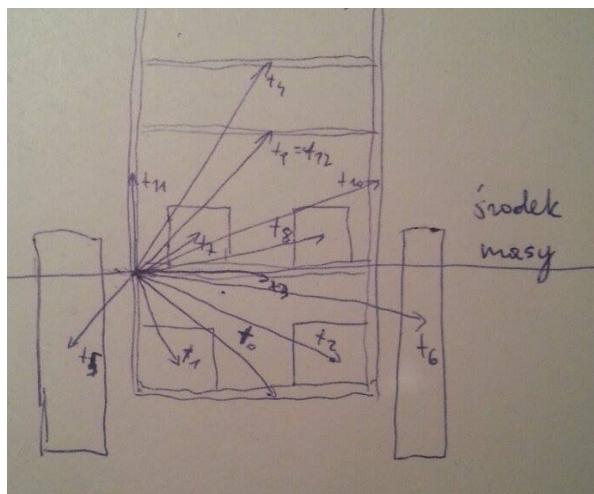
$$r_w = [6.5cm \quad 8cm]$$

Wektor: wysokość, szerokość

Środek masy w środku geometrycznym półki na akumulatory.

2.7.2 Obliczenie momentu bezwładności

Do obliczenia momentu bezwładności wykorzystałem program Matlab.
Rysunek poglądowy do zrozumienia oznaczeń wektorów:



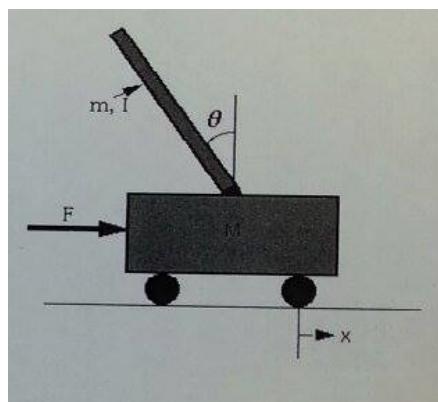
Rysunek 10: wektory t środków geometrycznych poszczególnych obiektów

Moment bezwładności jest równy:

$$I = 0.003 \text{ kgm}^2$$

2.7.3 Wyznaczenie równań liniowych układu:

Jest to układ odwróconego wahadła, który przypomina patyk znajdujący się na wózku, na który działa siła z zewnątrz. Układ taki wygląda następująco:



Rysunek 11: Układ wahadła odwróconego

Równania opisujące model:

$$\begin{aligned} F &= (M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta \\ -ml\ddot{x}\cos\theta &= (I+ml^2)\ddot{\theta} + mglsin\theta \end{aligned}$$

Dane układu:

- M - masa wózka: 0 kg
- m - masa odwróconego wahadła: 1.4 kg
- I - moment bezwładności odwróconego wahadła: 0.003 kgm^2 ,
- F - siła działająca na wózeczkę: nie większa niż 3 N ,
- x - położenie wózka - zmienna stanu mierzona w metrach,
- θ - wychylenie wahadła od pionu - zmienna stanu mierzona w stopniach,
- b - współczynnik tarcia wózka o podłoże: ok. 0.1 N/m/s,
- l - odległość od środka masy: 0.12 m,
- g - przyspieszenie ziemskie w Warszawie: 9.8123 m/s^2 ,

W przypadku robota balansującego na dwóch kołach równanie upraszcza się do postaci:

$$F = m\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta$$

$$-ml\ddot{x}\cos\theta = (I + ml^2)\ddot{\theta} + mg\sin\theta$$

Powyższe równania są nieliniowe, więc po linearyzacji w otoczeniu punktu $\theta = \pi$ otrzymałem następujące równania ($\theta = \pi + \phi$, gdzie ϕ reprezentuje małe odchylenia wahadła, a u to wejście układu):

$$u = m\ddot{x} + b\dot{x} - ml\ddot{\phi}$$

$$ml\ddot{x} = (I + ml^2)\ddot{\phi} - mg\phi$$

2.7.4 Opis układu w przestrzeni zmiennych stanu:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{Im} & \frac{mgl^2}{I} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-lb}{I} & \frac{mgl}{I} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{Im} \\ 0 \\ \frac{l}{I} \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u$$

Po wstawieniu liczb:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.1192 & 6.7359 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -1.2764 & 177.1575 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 1.1923 \\ 0 \\ 12.7640 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u$$

3 Elektronika

3.1 Zasilanie

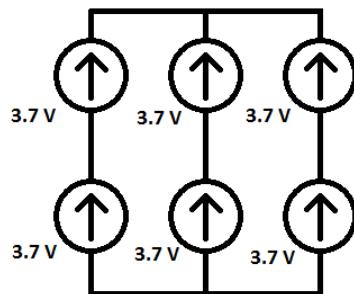
3.1.1 Akumulatory litowo-jonowe

Ogniwka użyte w robocie to niefirmowe akumulatory do telefonu Nokia 3310 o napięciu znamionowym 3.7 V i pojemności 1650 mAh. Jednakże w trakcie pracy napięcie na ogniwie zależy od stanu jego naładowania i może się wachać od 3.6 V do 4.2 V



Rysunek 12: Akumulator do telefonu Nokia 3310

Do stworzenia baterii posłużyły 6 ogniw. Po 2 ogniwka połączone szeregowo i 3 takie pakiety połączone równolegle:



Rysunek 13: Wizualizacja połączenia ogniw

Parametry baterii:

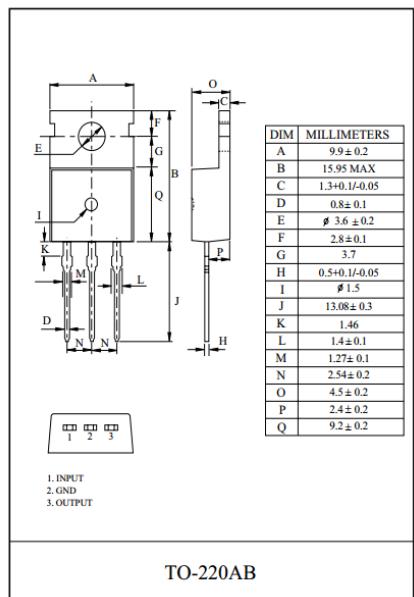
- Napięcie znamionowe: 7.4 V (zakres: 7.2V-8.4V)
- Pojemność: 5 Ah
- Masa: 0.48 kg

Bateria jest w stanie dostarczyć duży prąd ze względu na swoją pojemność, co było konieczne przy poborze prądu przez jedno uzwojenie silnika krokowego na poziomie 1.68 A.

3.1.2 Stabilizacja napięcia

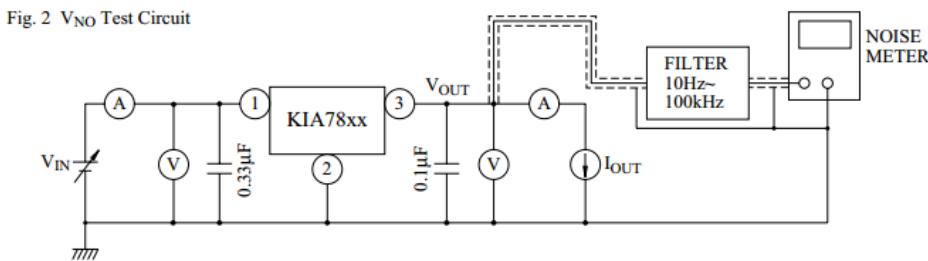
Napięcie z baterii jest stabilizowane przez stabilizator 7805 w obudowie TO-220 (co ułatwia chłodzenie układu scalonego) razem z kondensatorami MKT 330 nF na wejściu i ceramicznym 100 nF na wyjściu. Stabilizator dostarcza napięcie 5 V do układów mikroprocesorowych.

Fragment noty katalogowej producenta:



Rysunek 14: 7805, obudowa TO-220

Sposób połączenia:



Rysunek 15: 7805, schemat połączeń

3.1.3 Bezpieczeństwo układu

Elektronika jest zabezpieczona na kilka sposobów przed uszkodzeniem:

- Bateria jest podłączona do przełącznika, który służy jako główny włącznik robota - zabezpiecza przed włączeniem zasilania w trakcie ładowania ogniw, ponieważ mogłoby to spowodować przepięcia,
- Dodatni sygnał z baterii przechodzi przez szeregowo wpięty rezystor 0.1 ohm 5W, który ogranicza przepływ prądu do 7 A, ze względu na swoją moc.
- Do blaszki tranzystorów zostały przyjmocowane aluminiowe radiatory, przykręcone nylonowymi śrubami i izolowane termicznymi podkładkami.

3.2 Sterownik silników krokowych

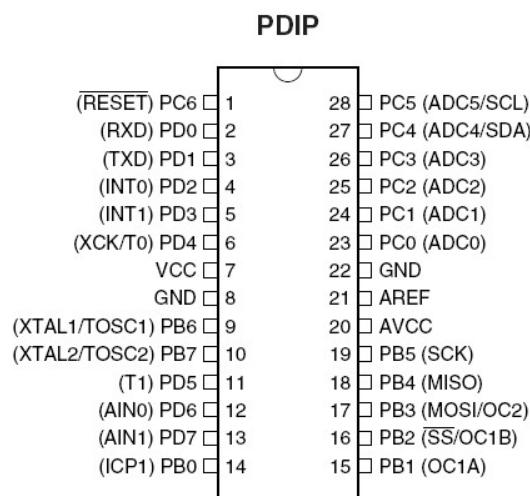
3.2.1 Mikrokontroler AVR ATmega8

Do sterowania silnikiem krokowym został wybrany ten mikrokontroler ze względu na cenę i łatwość programowania.

Sterowanie odbywa się w sposób półkrokowy i bez enkodera - program sam liczy kroki - globalnie i lokalnie (w trakcie przełączania cewek podczas kroku). Mostki zostały zbudowane w oparciu o technologię CMOS.

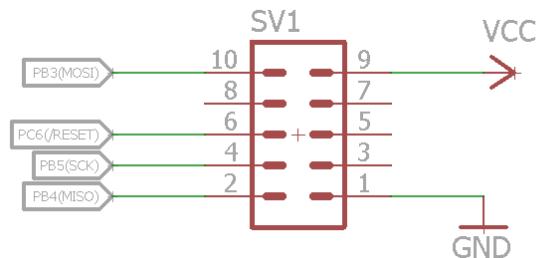
Do łatwego przeprogramowywania mikrokontrolera na płytce znajduje się złącze programatora zgodnego z USBASP o 10 pinowym standardzie KANDA.

Mikrokontroler ATmega8:



Rysunek 16: ATmega8

Schemat złącza KANDA:

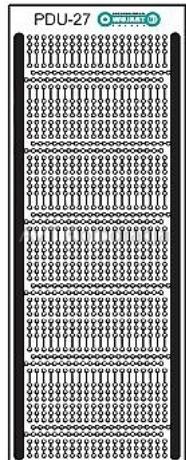


Rysunek 17: Złącze KANDA

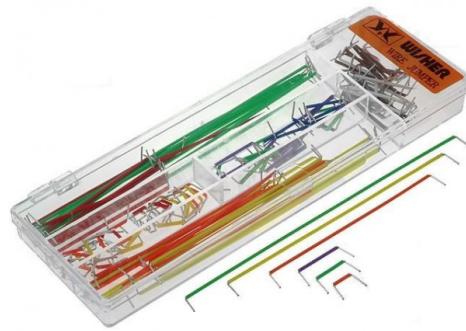
Przy podłączaniu mikrokontrolera zastosowałem kilka kondensatorów, dławik i rezystor. 100 nF między AREF a GND, VCC a GND i AVCC a GND, 100 uF między VCC a GND i 10 uH między AVCC i VCC oraz 10 kΩ podciągający RESET do VCC.

3.2.2 Płytki uniwersalna

Do umieszczenia elektroniki sterownika silników użyłem jednostronnej płytki uniwersalnej PDU 27. A do połączeń pomiędzy ścieżkami użyłem przewodów do płytki stykowej.

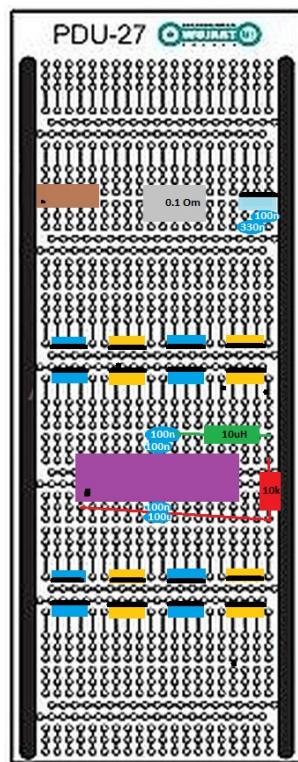


Rysunek 18: Schemat połączeń PDU-27



Rysunek 19: Przewody do płytki stykowej

Projekt umieszczenia elementów na płytce:



Rysunek 20: PDU27, elementy

Na rysunku elementy prostokątne niebieskie i pomarańczowe to odpowiednio MOSFETY z kanałem typu n i p, jasnoniebieski to stabilizator, brązowy to złącze KANDA (do programatora mikrokontrolera AVR), czerwony i szary to rezystory, a zielony dławik. Niebieskie elipsy to kondensatory. W trakcie fizycznej budowy układu niektóre elementy musiały zostać przesunięte, z powodu braku dostępu do kabla czy zredukowania ryzyka zwarcia, więc ostateczny wygląd płytka może nieco odbiegać od projektu. Płytnica została dodatkowo przycięta do 15 cm długości.

3.2.3 Własny projekt mostków H

Użyte tranzystory:

Tranzystor MOSFET z kanałem typu n: STP36NF06

Tranzystor MOSFET z kanałem typu p: IRF9530N

Sterownik składa się z czterech mostków H - jeden mostek na jedną cewkę silnika. Sygnały sterujące z mikrokontrolera wpadają jednocześnie na dwie bramki tranzystorów, co pozwala zaoszczędzić ilość zużywanych pinów i niemalże uniemożliwia zwarcie VCC i GND poprzez jednoczesne włączenie tranzystorów (jednak bardzo krótkie szpilki są możliwe ze względu na ładunki gromadzące się na bramce tranzystorów - budowa mostków uniemożliwia rozładowanie ładunku na okładkach tranzystorów, ponieważ zwiększyłyby to znacznie skomplikowanie układu).

Technologia CMOS została wybrana ze względu na niskie straty napięcia na złączu źródło-dren (niska rezystancja kanału MOSFET-ów rzędu setnych części Ω).

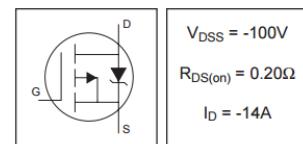
Wybrane tranzystory są dużej mocy - ok. 30W, więc zaoszczędziłem skomplikowania układu nie tworząc pomiaru prądu płynącego przez elementy półprzewodnikowe. Dodatkowo do tranzystorów nie zastosowałem diod zaporowych, ponieważ MOSFET-y zawierają taką diodę fabrycznie.

Użyte tranzystory są w obudowach TO-220 z widoczną blaszką w postaci radiatorka.

Wyciągi z not katalogowych:

Type	V_{DSS}	$R_{DS(on)}$	I_D
STP36NF06	60V	<0.040Ω	30A
STP36NF06FP	60V	<0.040Ω	18A ⁽¹⁾

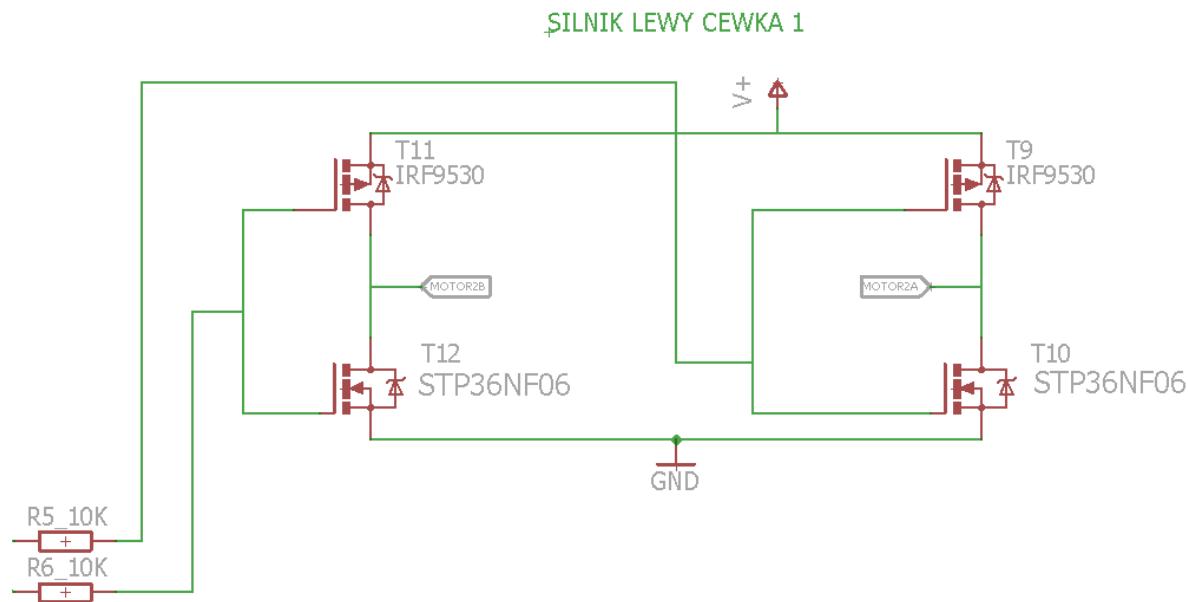
Rysunek 21: Właściwości STP36NF06



Rysunek 22: Właściwości IRF9530N

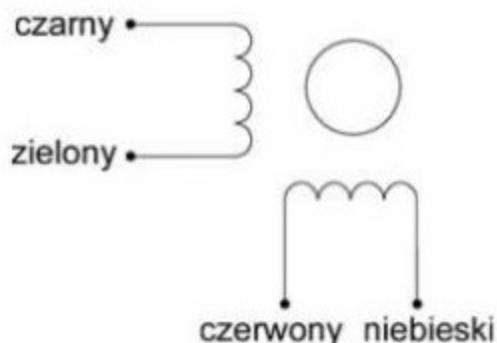
Jak widać tranzystory są do siebie bardzo podobne - co skutkuje poprawną pracą mostków.

Schemat budowy jednego mostka H:



Rysunek 23: Mostek H zaprojektowany w programie CADSOFT EAGLE

Wyprowadzenia cewek silnika:



Rysunek 24: Kolory przewodów z wyprowadzeniami cewek

Na schematach elektronicznych:

- niebieski - MOTORXA
- czerwony - MOTORXB
- zielony - MOTORXC
- czarny - MOTORXD

gdzie X to 1 (prawy) lub 2 (lewý).

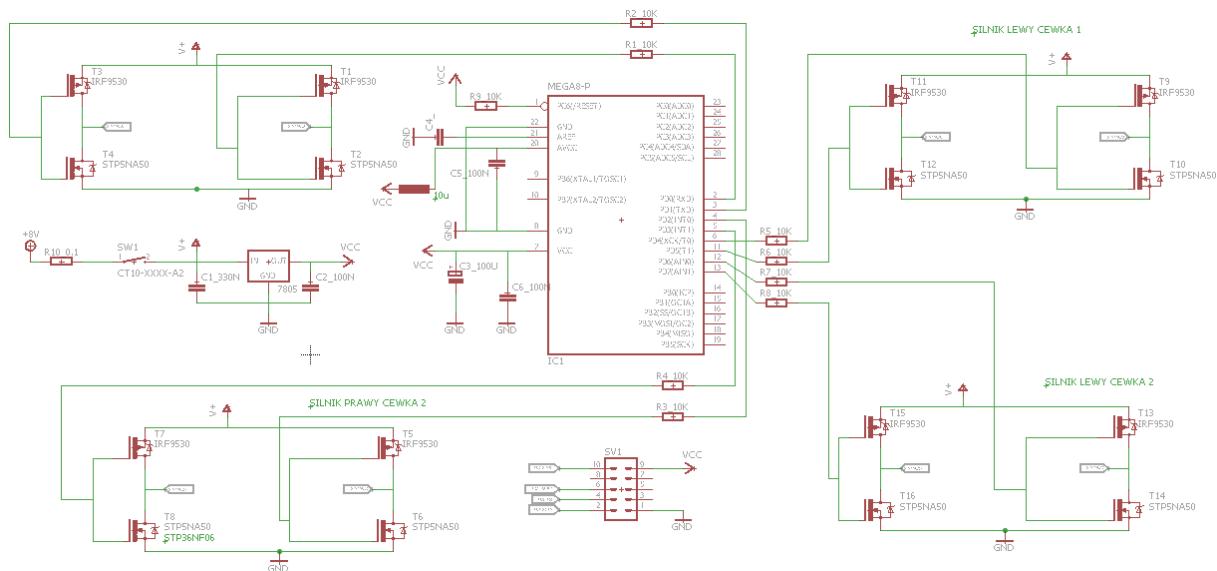
Kolejność załączania cewek przy sterowaniu półkrokowym:

nr cewki \ nr sekwencji	1	2	3	4	5	6	7	8
A	0	0		1	1	1		0
B	1	1		0	0	0		1
C		1	1	1		0	0	0
D		0	0	0		1	1	1

3.2.4 Schemat elektryczny

Schemat elektroniczny został zaprojektowany w programie CADSOFT EAGLE.

Nieomówione dotychczas elementy indukcyjne i pojemnościowe pełnią rolę filtracji zasilania - eliminacji szpilek.



Rysunek 25: Sterownik silników zaprojektowany w programie CADSOFT EAGLE

3.3 Moduł czujników MPU-6050 - żyroskop i akcelerometr

Użyty moduł firmy Invensense zawiera wbudowany cyfrowy żyroskop i akcelerometr.

Napięcie zasilania: 3,3V - 5V

Zakresy pracy żyroskopu: 250 dps, 500 dps, 1000 dps, 2500 dps (dps - degrees per second)

Zakresy pracy akcelerometru: 2g, 4g, 8g, 16g

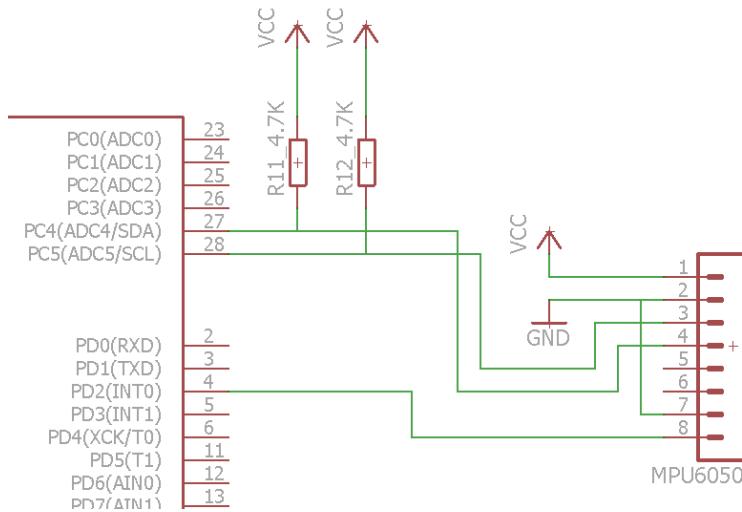
Wymiary: 20mm x 16mm

Waga: 0.9 g



Rysunek 26: MPU6050

Komunikacja z czujnikiem odbywa się za pośrednictwem magistrali szeregowej I2C.
Schemat połączenia modułu z mikrokontrolerem ATmega8:



Rysunek 27: Komunikacja szeregowa I2C

Czujnik został zamontowany idealnie 12 cm nad środkiem masy robota na podkładkach amortyzacyjnych, przykręconych nylonowymi śrubami w celu redukcji drgań powstających podczas pracy silnika.

3.4 Wyświetlacz LCD 16x2 zgodny ze standardem HD44780

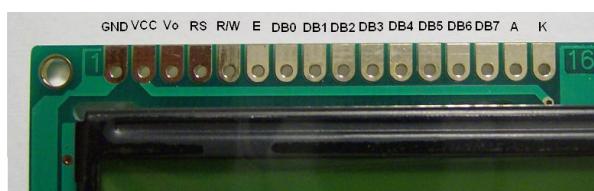
Wyświetlacz został umieszczony na robocie w celu identyfikacji bieżących odczytów z MPU6050.



Rysunek 28: HD44780

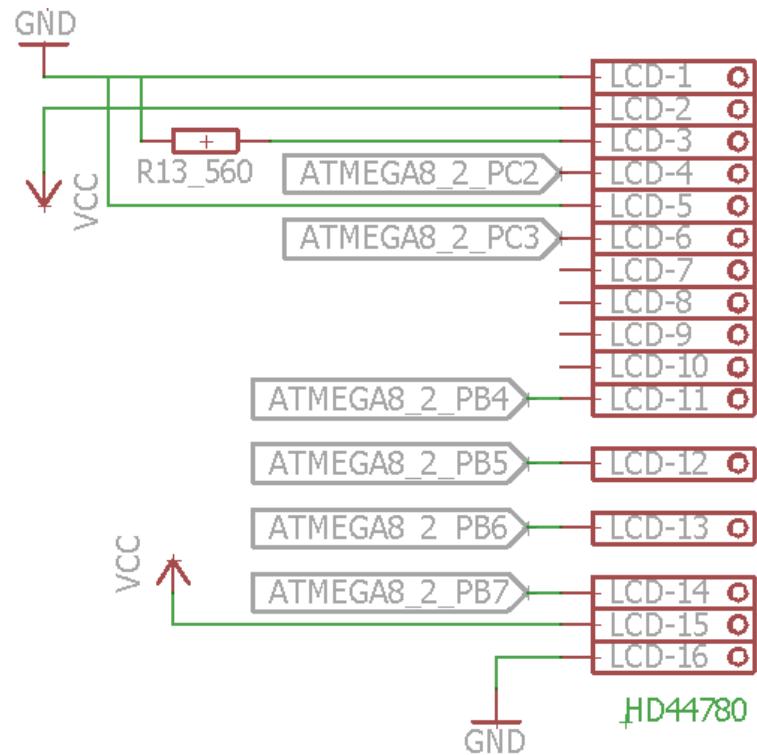
Komunikacja z wyświetlaczem odbywa się za pomocą linii RS i E (RW zwarte na stałe do masy) oraz 4 linii danych DB4, DB5, DB6 i DB7 (wysyłanie połówek bajtu).

Rozkład pinów wyświetlacza:



Rysunek 29: Rozkład pinów w standardzie HD44780

Schemat połączenia wyświetlacza z mikrokontrolerem ATmega8:



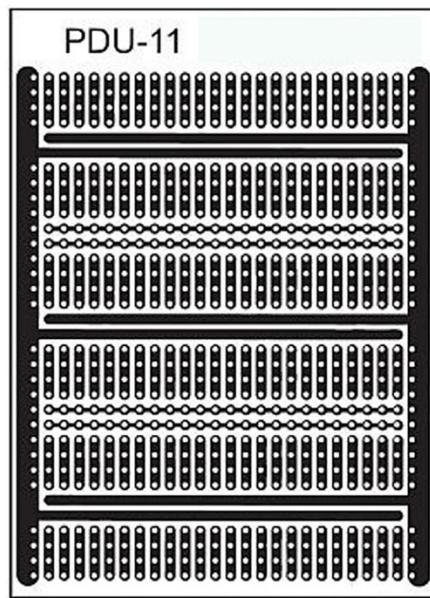
Rysunek 30: Komunikacja hd44780-ATmega8

3.5 Układ regulacji położenia

3.5.1 Płytki uniwersalna

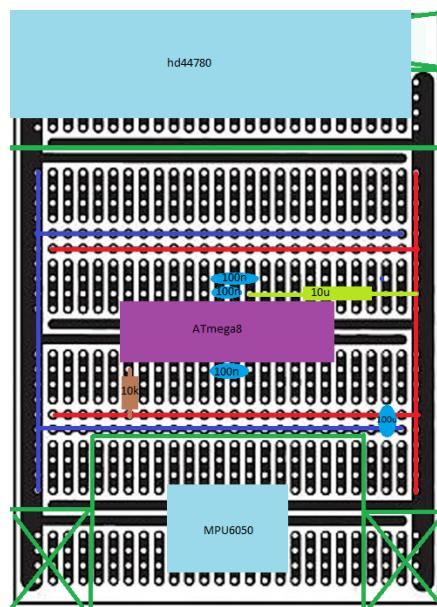
Do montażu elektroniki sterującej została wykorzystana płytka uniwersalna PDU11.

Fabryczne połączenia na płytce PDU11:



Rysunek 31: PDU11

Rozmieszczenie elementów na płytce:

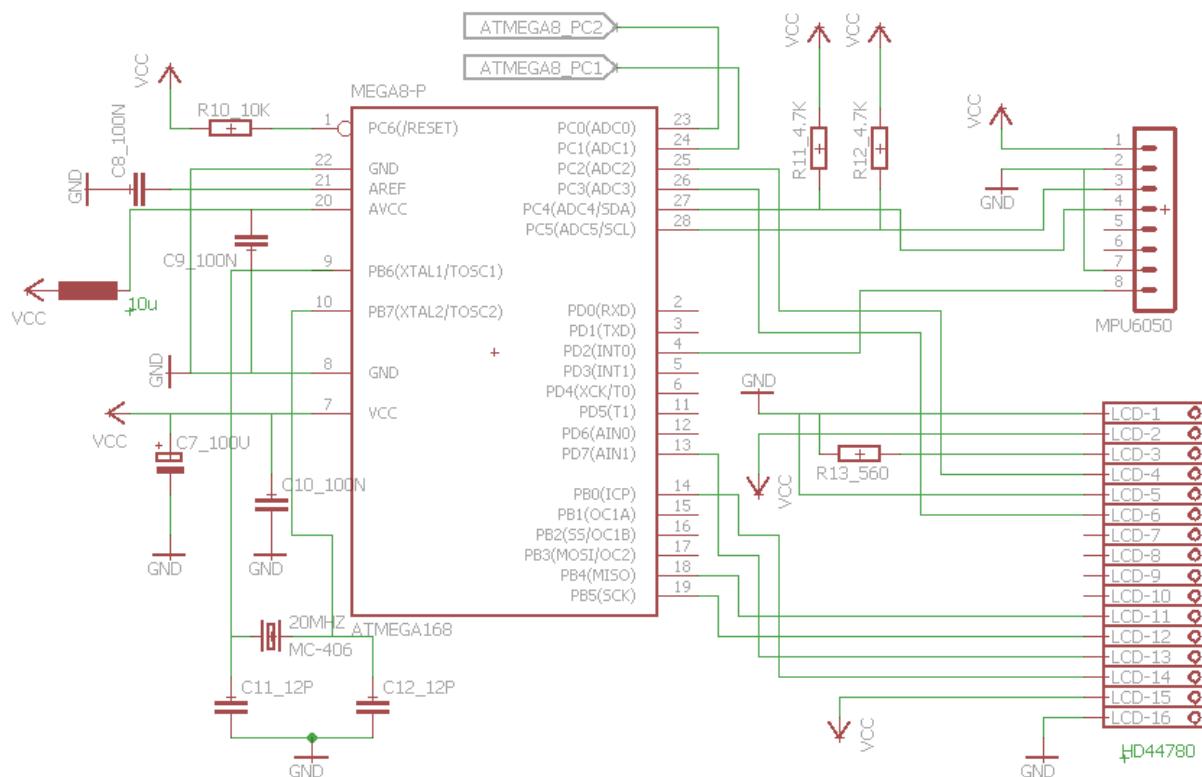


Rysunek 32: Rozkład elementów na PDU11

Zielone linie oznaczają miejsca przecięć płytki - konieczne do prawidłowego umiejscowienia i zorientowania modułu MPU6050.

3.5.2 Schemat elektryczny

Do regulacji położenia został wykorzystany drugi mikrokontroler AVR - rozbudowana wersja ATmega168. ATmega168 posiadająca więcej przerwań zewnętrznych i 16 kb pamięci (2 razy więcej niż ATmega8). Drugi mikrokontroler musiał być użyty ze względu na małą pamięć, ilość pinów i niskie taktowanie wcześniej użytego układu scalonego, a także w celu zmniejszenia stopnia skomplikowania programu. Dwa mikrokontrolery komunikują się za pomocą dwóch pinów. Na poniższym schemacie są to piny PC0 i PC1. PC0 odpowiada za generowanie sygnału PWM. Informacja jest odczytywana w sterowniku silnika za pomocą przetwornika analogowo-cyfrowego i zamieniana na czas pomiędzy przełączaniami cewek - czyli prędkość. Natomiast PC1 przybiera tylko 2 stany, które oznaczają kierunek obracania się kół. Mikrokontroler został "podkręcony" do częstotliwości taktowania zegara 20 Mhz za pomocą rezonatora kwarcowego, który jest znacznie stabilniejszym źródłem drgań niż wewnętrzny oscylator RC - zwiększa to dokładność odczytów czujników i większą ilość operacji na sekundę.



Rysunek 33: Schemat elektroniki odpowiadającej za stabilizację położenia (CADSOFT EAGLE)

4 Teoria sterowania

4.1 Wyznaczenie sterowalności i obserwowałości układu otwartego

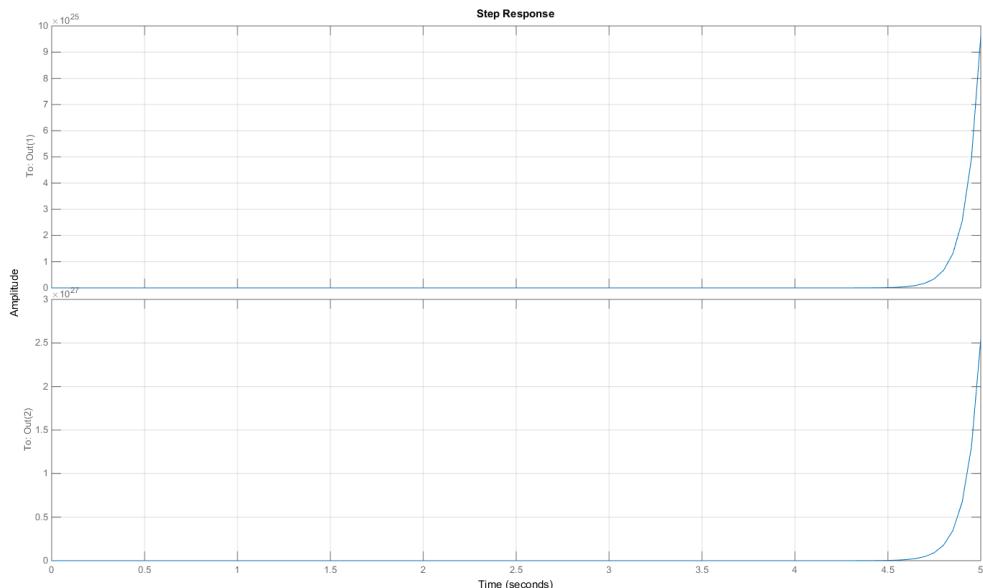
Używając gotowych funkcji w Matlabie obliczyłem rzędy macierzy obserwowałości i sterowalności - wynoszą 4. Oznacza to, że skonstruowany układ jest sterowalny i obserwowywalny.

4.2 Wyznaczenie stabilności i odpowiedź skokowa układu otwartego

Układ bez pętli sterowania jest niestabilny - posiada dodatni biegum:

$$\begin{bmatrix} 0 \\ -0.0707 \\ -13.3345 \\ 13.2859 \end{bmatrix}$$

Niestabilność układu można zilustrować zasymulowaną odpowiedzią skokową:

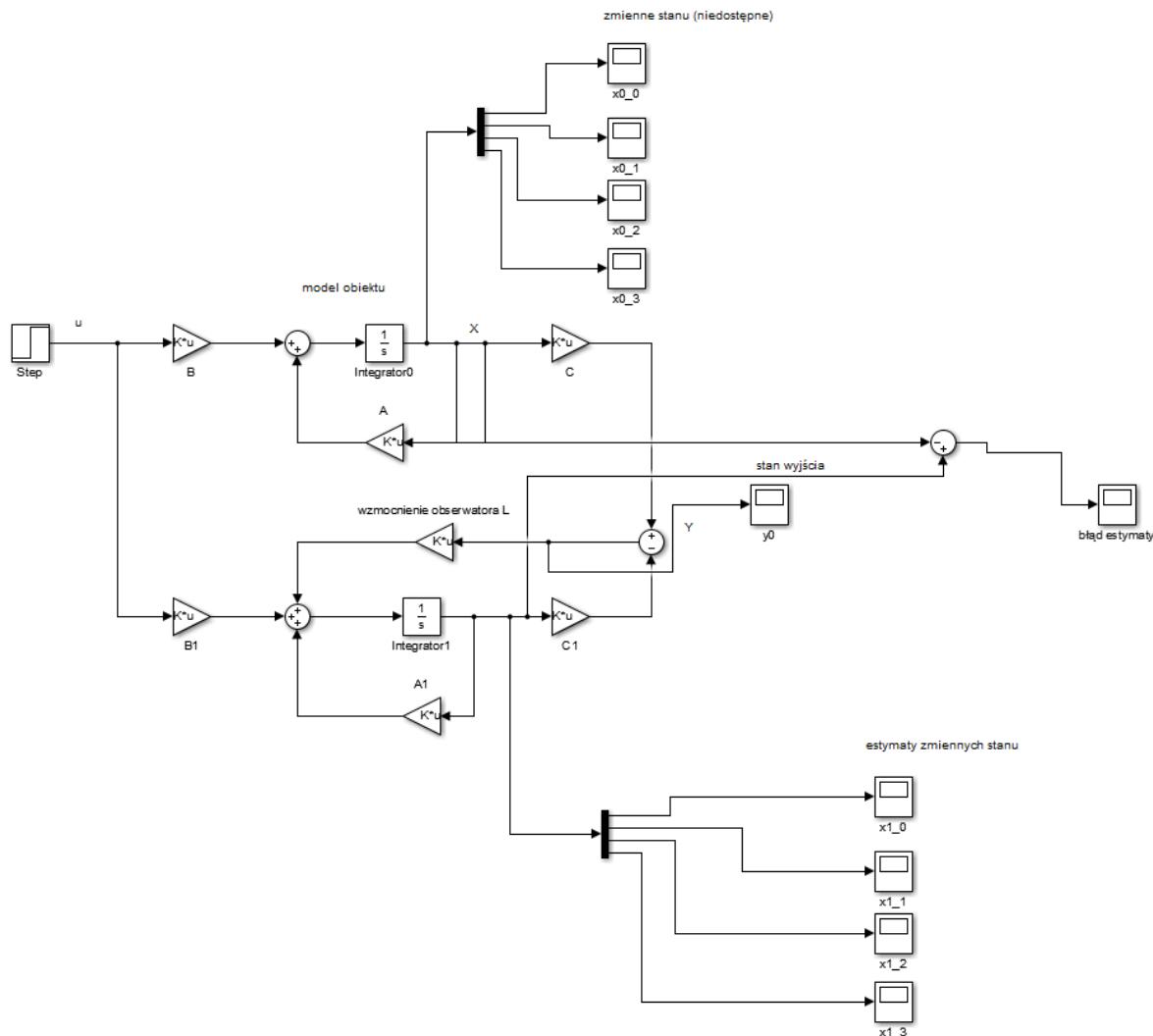


Rysunek 34: Odpowiedź skokowa układu otwartego

Układ po zadaniu impulsu dąży do nieskończoności.

4.3 Projekt obserwatora Luenbergera pełnego rzędu

Model obserwatora w Simulinku:



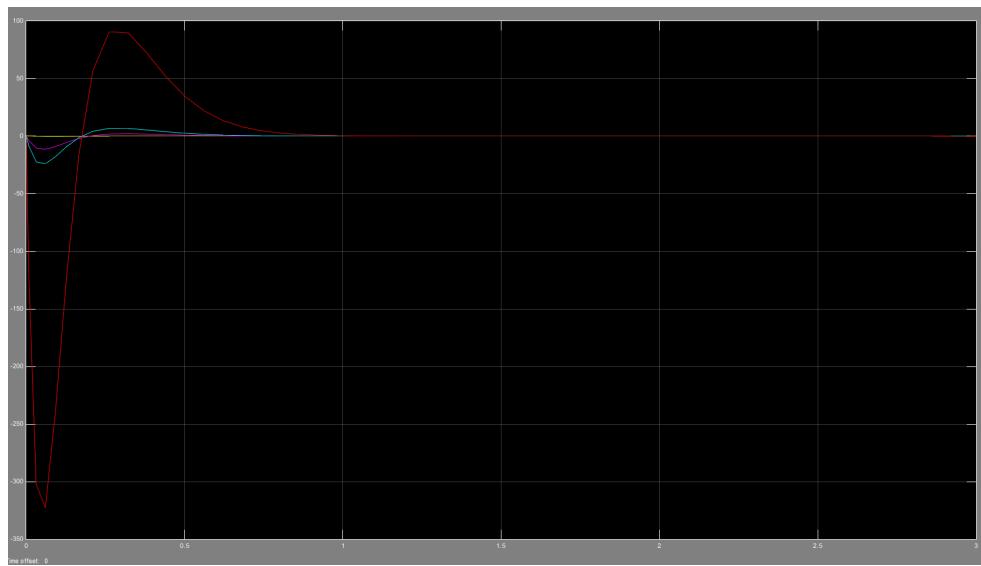
Rysunek 35: Obserwator Luenbergera

Wzmocnienie obserwatora zostało wyliczone według wektora żądanых pierwiastków

$$\begin{bmatrix} -11 \\ -12 \\ -13 \\ -14 \end{bmatrix}$$

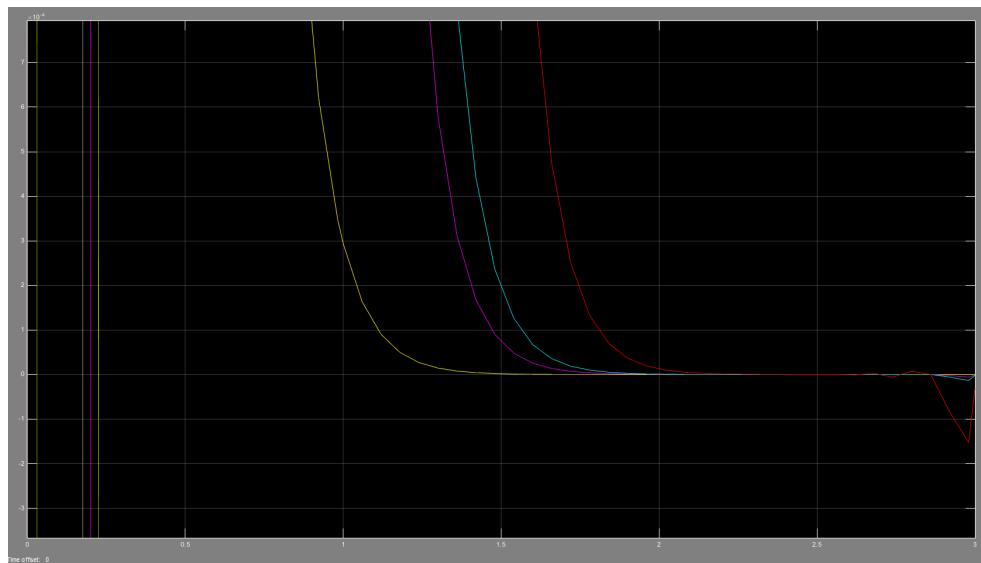
i wynosi: $\begin{bmatrix} 50 \\ 1106 \\ 2464 \\ 32753 \end{bmatrix}$

Błąd estymaty obserwatora przedstawia wykres:



Rysunek 36: Błąd estymacji obserwatora

W powiększeniu:



Rysunek 37: Błąd estymacji obserwatora

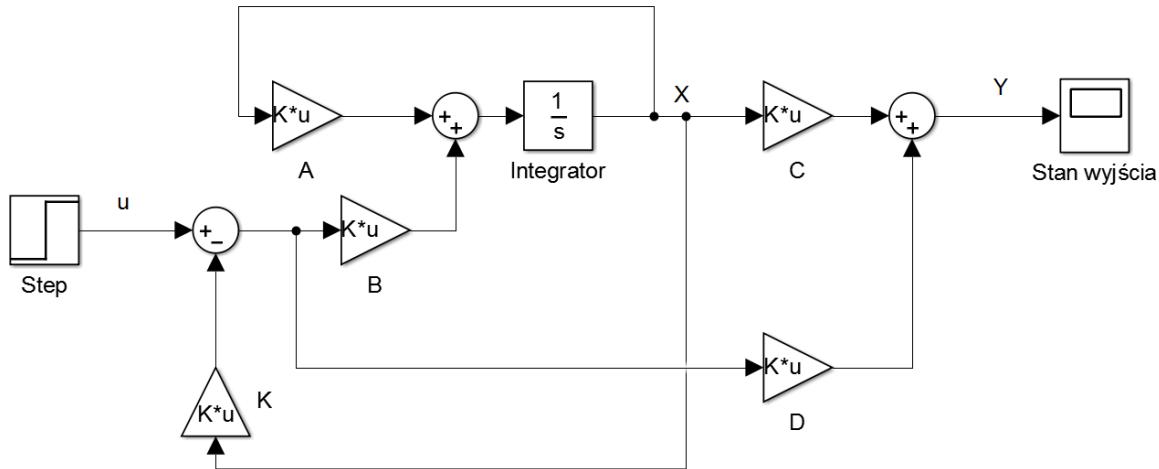
Błąd estymaty obserwatora jest bardzo mały – rzędu 10^{-4} , co świadczy o poprawnym wzmocnieniu obserwatora.

4.4 Projekt regulatora liniowo-kwadratowego (LQR)

Zastosowanie regulatora LQR zapewnia optymalne sterowanie, ale wymaga matematycznego modelu układu. Założenia projektowe to:

- Czas ustalenia odpowiedzi poniżej 3 sekund
- Przeregulowanie pozycji poniżej 10

LQR w Simulink'u:



Rysunek 38: Regulator liniowo-kwadratowy

Macierz wzmacnień K sprzężenia zwrotnego jest obliczana rozwiązyując równanie różniczkowe Riccatiego o postaci:

$$K = R^{-1}B^T P(t)$$

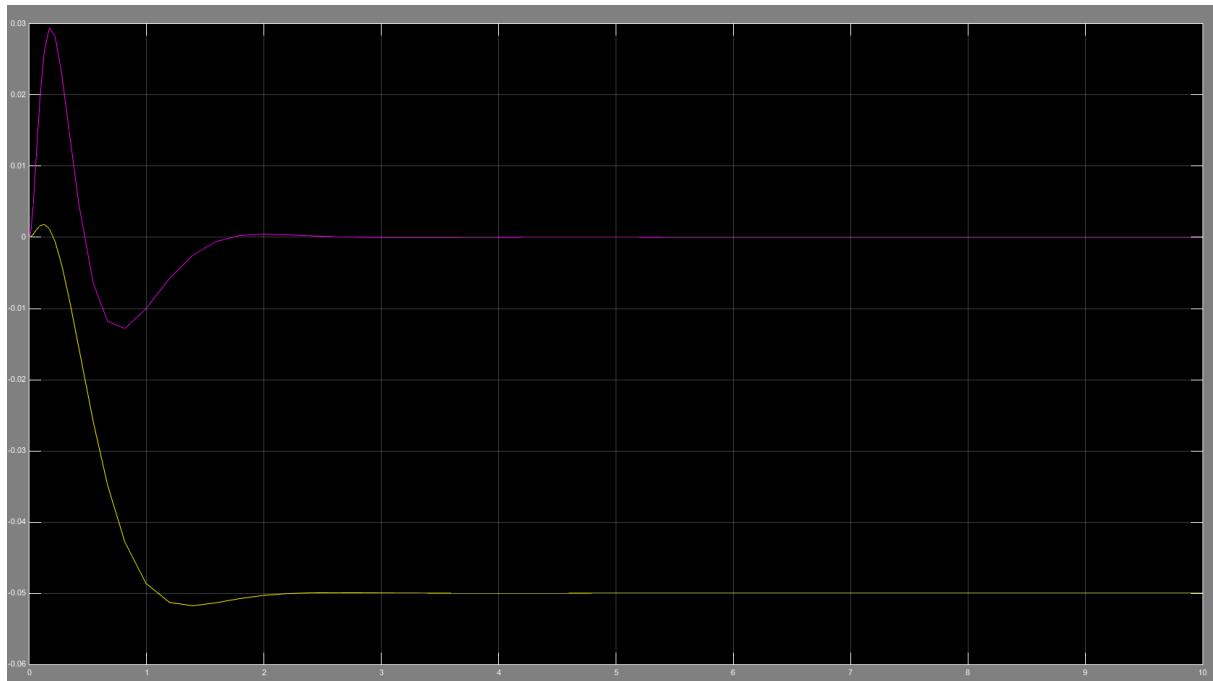
$$\dot{P}(t) = A^T P(t) + P(t)A - P(t)BR^{-1}B^T P(t) + Q$$

Gdzie Q i R to macierze wagowe wskaźnika J . Wartości macierzy Q i R dobrąłem tak, aby spełnić założenia projektowe:

- Przemieszczenie wózka w przybliżeniu nie większe niż 5 cm: $Q_{11} = 1/0.05^2$
- Prędkość robota nie większa niż 3 m/s: $Q_{22} = 1/3^2$
- Mały kąt wychylenia wahadła nie większy niż 7 stopni: $Q_{33} = 1/7^2$
- Prędkość kątowa wychylenia wahadła nie większa niż 30 stopni/s: $Q_{44} = 1/30^2$

Wartości pozostałych elementów macierzy Q przyjąłem jako zerowe – uzyskując w ten sposób macierz symetryczną. Podobnie dla macierzy R , która ma wymiary 1×1 , jest to kwadrat odwrotności największej siły zadziałającej na wahadło czyli ok. 4 N, więc $R = [1/4^2]$.

Wyjście układu:



Rysunek 39: Wyjście układu

Układ jest stabilny i spełnia założenia projektowe.

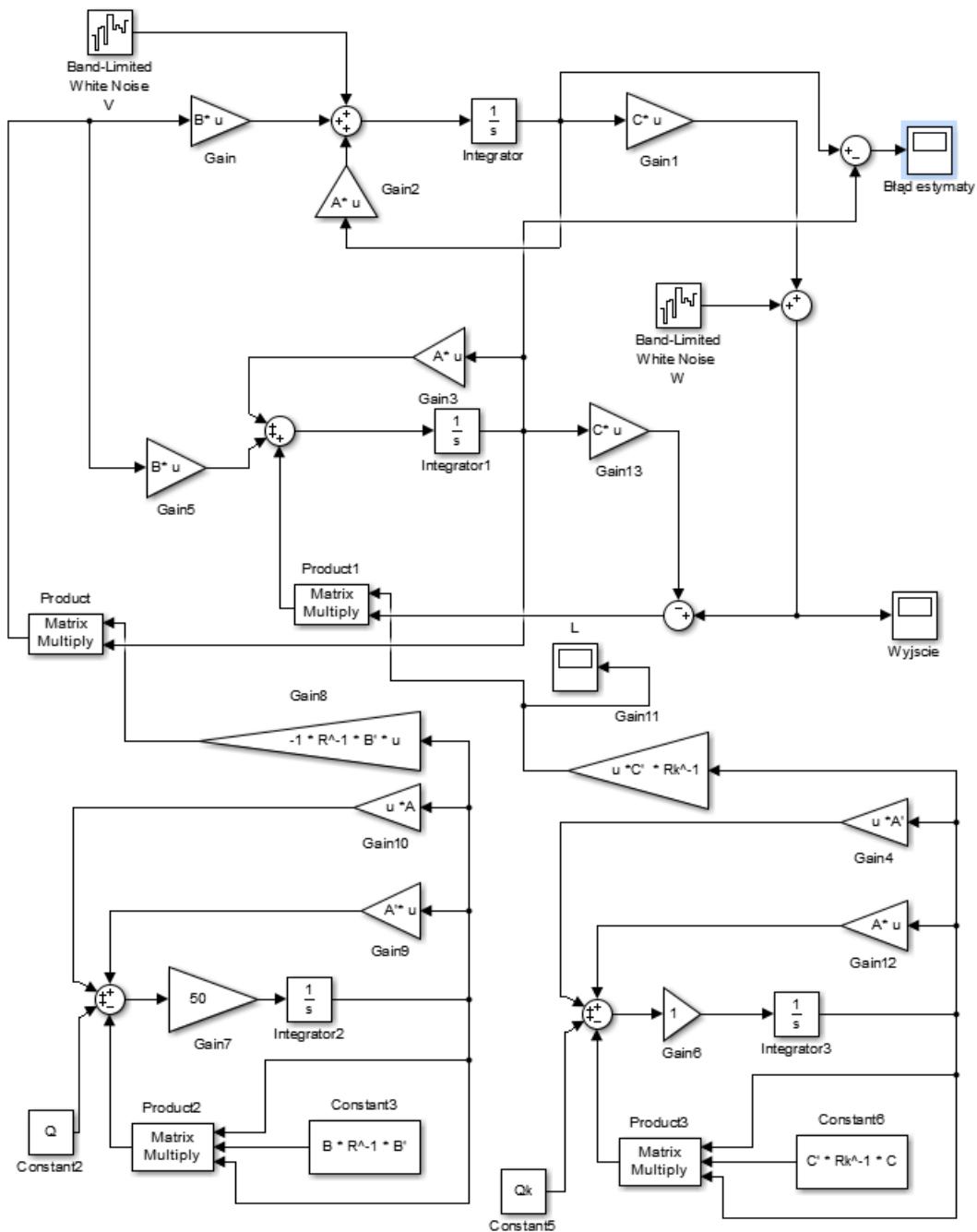
Macierze Q i R:

$$Q = \begin{bmatrix} 400.0000 & 0 & 0 & 0 \\ 0 & 0.1111 & 0 & 0 \\ 0 & 0 & 0.0204 & 0 \\ 0 & 0 & 0 & 0.0011 \end{bmatrix}, R = [0.0625]$$

4.5 Projekt regulatora liniowo-kwadratowego-Gaussa (LQG)

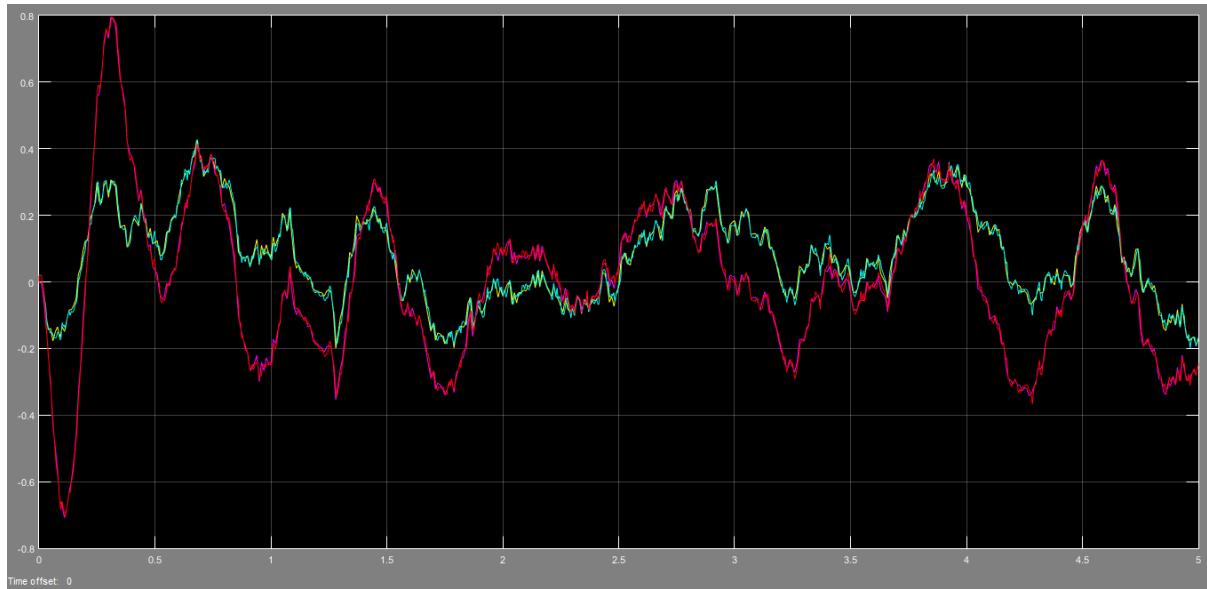
Do układu wahadła odwróconego konieczne jest zastosowanie sterowania predykcyjnego, aby układ działał poprawnie. W tym celu do regulatora LQR zaprojektowałem filtr Kalmana:

Model LQG w Simulinku:



Rysunek 40: Regulator liniowo-kwadratowy-Gaussa

Prezentacja predykcji filtru:



Rysunek 41: Wyjście układu i wyjście obserwatora ze wzmacnieniem Kalmana

Jak widać filtr Kalmana nadąża za zmianami wyjść układu i estymuje je z małym błędem.

4.6 Kod Matlaba

```
%Obliczenie srodka masy
%wektory zawierajace składowa pionowa srodkow masy poszczegolnych obiektow
%robotu

r0=0.3;
r1=2.7;
r2=2.7;
r3=6.6;
r4=17.7;
r5=2.7;
r6=2.7;
r7=8.65;
r8=8.65;
r9=13.3;
r10=11;
r11=11;
r12=13.3;

%masy poszczegolnych obiektow

m0=70*0.35;%
m1=340;
m2=340;
m3=35;%
m4=35+45;%
m5=20;
m6=20;
m7=180;
m8=180;
```

```

m9=90;
m10=150*0.35;%
m11=150*0.35;%
m12=10;

%masa obudowy
mob=m0+m3+m4+m10+m11-45;
%gestosc drewna
d=mob*1000/216;

%wartosc bezwzgledna odleglosci srodkow geometrycznych obiektow od srodka
%masy (srodek masy wyliczylem wczesniej i na sztywno wstawilem 6.5 cm)
t0=abs(0.3-6.5);
t1=abs(2.7-6.5);
t2=abs(2.7-6.5);
t3=abs(6.6-6.5);
t4=abs(17.7-6.5);
t5=abs(2.7-6.5);
t6=abs(2.7-6.5);
t7=abs(8.65-6.5);
t8=abs(8.65-6.5);
t9=abs(13.3-6.5);
t10=abs(11-6.5);
t11=abs(11-6.5);
t12=abs(13.3-6.5);

%masa
mm=m0+m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11;
mm2=mm/1000;

%srodek masy
rr=(m0*r0+m1*r1+m2*r2+m3*r3+m4*r4+m5*r5+m6*r6+m7*r7+m8*r8+m9*r9+m10*r10+m11*r11+m12*r12);
rr2=rr/100;

%moment bezwladnosci robota wzgledem srodka masy (os obrotu bryly)
%II=(mm2*(rr2-0.027)*(rr2-0.027))/3
%II2=(m0*(t0^2)+m1*(t1^2)+m2*(t2^2)+m3*(t3^2)+m4*(t4^2)+m5*(t5^2)+m6*(t6^2)+m7*(t7^2)+m8*(t8^2)+m9*(t9^2)+m10*(t10^2)+m11*(t11^2)+m12*(t12^2));
%Wartosci parametrow
M = 0; %masa wozka - brak wozka (rysunek obrazujacy wozek w rozdz. 2.7.3
       dokumentacji)
m = mm2; %masa wahadla
b = 0.1; %wsp. tarcia wozka o podloze (przyblizony)
l = 0.12; %odleglosc do srodka masy wahadla
I = II2; %moment bezwladnosci wahadla
g = 9.8123; %wartosc przyspieszenia ziemskiego w Warszawie

%Macierze ukladu A,B,C,D
A = [0 1 0 0
      0 (-b*(I+m*l*l))/(I*(M+m)+M*m*l*l) (m*m*g*l*l)/(I*(M+m)+M*m*l*l) 0
      0 0 0 1
      0 (-m*l*b)/(I*(M+m)+M*m*l*l) (m*g*l*(M+m))/(I*(M+m)+M*m*l*l) 0];
B = [0
      (I+m*l*l)/(I*(M+m)+M*m*l*l)
      0
      (m*l)/(I*(M+m)+M*m*l*l)];

```

```

C= [1 0 0 0
     0 0 1 0];
D = [0
      0];

%sprawdzenie obserwonalnosci i sterowalnosci
S=ctrb(A,B);
ster=rank(S);
O=obsv(A,C);
obser=rank(O);

sys=ss(A,B,C,D); %funkcja ladnie wyswietla macierze systemowe
bieguny=pole(sys); %obliczenie biegunow wielomianu char.

step(sys,5);%odpowiedz ukladu otwartego na skok jednostkowy T =5sek
grid;

%LQR

%wagi poszczegolnych zmiennych
Q11 = 1/(0.05^2);
Q22 =1/(3^2);
Q33 =1/(7^2);
Q44= 1/(30^2);

Q=[Q11 0 0 0
   0 Q22 0 0
   0 0 Q33 0
   0 0 0 Q44];

R=1/16;

K = lqr(sys,Q,R);
%K = [-80.0000 -28.6388 63.7184 5.6401]

%Obserwator Luenbergera

L= place(A',C'*[1;0],[-11,-12,-13,-14]');

%Filtr Kalmana

Qk = [1/10 0 0 0
      0 1/10 0 0
      0 0 1/10 0
      0 0 0 1/10];
Rk = 1/(1000^2);

```

5 Oprogramowanie

5.1 Sterownik silników krokowych

Doświadczalnie wyznaczyłem, że maksymalna prędkość silnika jest osiągana dla czasu pomiędzy przełączeniami cewek wynoszącego $500\text{ }\mu\text{s}$

Kod napisany w języku C, w środowisku Eclipse Mars z pluginem AVR:

```
/*
 * main.c
 * Sterownik silnika krokowego:
 * 0,5 Nm - bipolarny
 *
 * Created on: 3 sty 2016
 * Author: Mateusz Staszkow
 */

#include <avr/io.h>
#include <util/delay.h>

//silnik prawy
//zielony
#define c1 (1<<PD2)
//czarny
#define c2 (1<<PD3)
//czerwony
#define c3 (1<<PD1)
//niebieski
#define c4 (1<<PD0)

//silnik lewy
//zielony
#define l1 (1<<PD6)
//czarny
#define l2 (1<<PD7)
//czerwony
#define l3 (1<<PD5)
//niebieski
#define l4 (1<<PD4)

//0 do przodu, 1 do tyłu
#define PIN_KIERUNEK (1<<PC1)
#define KIERUNEK (PIN_C & PIN_KIERUNEK)

//dwie cewki, duży moment, sterowanie polkrokowe
//prawy
#define krok1p PORTD &= ~c1; PORTD |= c2;
#define krok2p PORTD &= ~c1; PORTD |= c2; PORTD |= c3; PORTD &= ~c4;
#define krok3p PORTD |= c3; PORTD &= ~c4;
#define krok4p PORTD &= ~c2; PORTD |= c1; PORTD |= c3; PORTD &= ~c4;
#define krok5p PORTD &= ~c2; PORTD |= c1;
#define krok6p PORTD &= ~c2; PORTD |= c1; PORTD |= c4; PORTD &= ~c3;
#define krok7p PORTD |= c4; PORTD &= ~c3;
#define krok8p PORTD &= ~c1; PORTD |= c2; PORTD |= c4; PORTD &= ~c3;
//lewy
#define krok1l PORTD &= ~l1; PORTD |= l2;
```

```
#define krok21 PORTD &= ~11; PORTD |= 12; PORTD |= 13; PORTD &= ~14;
#define krok31 PORTD |= 13; PORTD &= ~14;
#define krok41 PORTD &= ~12; PORTD |= 11; PORTD |= 13; PORTD &= ~14;
#define krok51 PORTD &= ~12; PORTD |= 11;
#define krok61 PORTD &= ~12; PORTD |= 11; PORTD |= 14; PORTD &= ~13;
#define krok71 PORTD |= 14; PORTD &= ~13;
#define krok81 PORTD &= ~11; PORTD |= 12; PORTD |= 14; PORTD &= ~13;

void polkrokowy_prawy_przod(int,int);
void polkrokowy_prawy_tyl(int,int);
void polkrokowy_prawy_hamuj(int,int);
void polkrokowy_lewy_przod(int,int);
void polkrokowy_lewy_tyl(int,int);
void polkrokowy_lewy_hamuj(int,int);

void przod(int,int,int);
void tyl(int,int,int);

//czas pomiedzy krokami ( w mikrosekundach)
volatile int czas_t;
volatile int czas_p;

volatile int czas_m;

//wybor sterowania
volatile int sterowanie;

//licznik krokow
volatile int cnt_p;
volatile int cnt_l;

//zmienne opoznienie
#define czekaj(czas) for(int i=0;i<(czas);i++) _delay_us(1);

uint16_t pomiar(uint8_t kanal);

int main(void) {

    //inicjalizacja ADC
    ADMUX |= (1<<REFS0);
    ADCSRA |= (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);

    int test=0;
    int i;

    //domyslnie maksymalna szybkosc
    czas_t = 500;
    czas_p = 500;

    //wyzerowanie licznika
    cnt_p = 0;
    cnt_l = 0;

    //ustawienie portow silnik prawy
    DDRD |= c1|c2|c3|c4;
    //ustawienie portow silnik lewy
    DDRD |= 11|12|13|14;
    DDRC &= ~(PIN_KIERUNEK);
```

```

PORTC |= (PIN_KIERUNEK);

while(1) {
    if(KIERUNEK) {
        tyl(cnt_p,cnt_l,(2*pomiar(5)+500));
    } else {
        przod(cnt_p,cnt_l,(2*pomiar(5)+500));
    }
}

uint16_t pomiar(uint8_t kanal) {
    ADMUX = (ADMUX & 0xF8) | kanal;
    ADCSRA |= (1<<ADSC);
    while(ADCSRA & (1<<ADSC));
    return ADCW;
}

void polkrokowy_prawy_przod(int pozycja,int czas_m) {
    switch(pozycja%8)
    {
        case 0:
            if(cnt_p == 400) cnt_p = 0;
            krok1p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 1:
            krok2p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 2:
            krok3p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 3:
            krok4p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 4:
            krok5p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 5:
            krok6p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 6:
            krok7p;
            cnt_p++;
            if(cnt_p == 400) cnt_p = 0;
            czekaj(czas_m);
        case 7:
    }
}

```

```
krok8p;
cnt_p++;
if(cnt_p == 400) cnt_p = 0;
czekaj(czas_m);
}
if(cnt_p == 400) cnt_p = 0;
}

void polkrokowy_lewy_przod(int pozycja, int czas_m) {
switch(pozycja%8)
{
case 0:
if(cnt_l == 400) cnt_l = 0;
krok1l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 1:
krok2l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 2:
krok3l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 3:
krok4l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 4:
krok5l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 5:
krok6l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 6:
krok7l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
case 7:
krok8l;
cnt_l++;
if(cnt_l == 400) cnt_l = 0;
czekaj(czas_m);
}
if(cnt_p == 400) cnt_p = 0;
}

void polkrokowy_prawy_tyl(int pozycja, int czas_m) {
switch(pozycja%8)
```

```

{
    case 0:
        if(cnt_p == 0) cnt_p = 400;
        krok1p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 7:
        krok8p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 6:
        krok7p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 5:
        krok6p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 4:
        krok5p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 3:
        krok4p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 2:
        krok3p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    case 1:
        krok2p;
        cnt_p--;
        if(cnt_p == 0) cnt_p = 400;
        czekaj(czas_m);
    }
    if(cnt_p == 0) cnt_p = 400;
}

void polkrokowy_lewy_tyl(int pozycja, int czas_m) {
    switch(pozycja%8)
    {
        case 0:
            if(cnt_l == 0) cnt_l = 400;
            krok1l;
            cnt_l--;
            if(cnt_l == 0) cnt_l = 400;
            czekaj(czas_m);
        case 7:
            krok8l;
            cnt_l--;

```

```

if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
case 6:
krok7l;
cnt_l--;
if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
case 5:
krok6l;
cnt_l--;
if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
case 4:
krok5l;
cnt_l--;
if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
case 3:
krok4l;
cnt_l--;
if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
case 2:
krok3l;
cnt_l--;
if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
case 1:
krok2l;
cnt_l--;
if(cnt_l == 0) cnt_l = 400;
czekaj(czas_m);
}
if(cnt_l == 0) cnt_l = 400;
}

void przod(int pozycja_p,int pozycja_l,int czas) {
polkrokowy_prawy_przod(pozycja_p,czas);
polkrokowy_lewy_tyl(pozycja_l,czas);
}
void tyl(int pozycja_p,int pozycja_l, int czas) {
polkrokowy_prawy_tyl(pozycja_p,czas);
polkrokowy_lewy_przod(pozycja_l,czas);
}

```

5.2 Obsługa czujników MPU-6050

Program składa się z kilku plików:

- Pakiet mpu6050:
mpu6050.c i mpu6050.h - odpowiadające za obsługę czujnika
mpu6050registers.h - pozyskany z internetu (w celu identyfikacji rejestrów czujnika), źródło:
<https://github.com/mkschreder/avr-ultimate-driver-pack/blob/master/mpu6050registers.h>
- Pakiet i2chw: i2cmaster.h i twimastertimeout.c - odpowiadające za komunikację I2C

Program zamienia odczytane wyniki z czujnika na g i dps.

Kod napisany w języku C, w środowisku Eclipse Mars z pluginem AVR:

i2chw/i2cmaster.h

```
//i2chw/i2cmaster.h

#ifndef _I2CMASTER_H
#define _I2CMASTER_H 1

#include <avr/io.h>

#define I2C_READ 1

#define I2C_WRITE 0

//inicjalizacja
extern void i2c_init(void);

//zatrzymanie transmisji
extern void i2c_stop(void);

//start transmisji
extern unsigned char i2c_start(unsigned char addr);

extern unsigned char i2c_rep_start(unsigned char addr);
extern void i2c_start_wait(unsigned char addr);
extern unsigned char i2c_write(unsigned char data);
extern unsigned char i2c_readAck(void);
extern unsigned char i2c_readNak(void);
extern unsigned char i2c_read(unsigned char ack);
#define i2c_read(ack) (ack) ? i2c_readAck() : i2c_readNak();

#endif
```

i2chw/twimastertimeout.c

```
//i2chw/twimastertimeout.c

#include <inttypes.h>
#include <compat/twi.h>

#include "i2cmaster.h"

//zegar i2c
#define SCL_CLOCK 100000L

//maksymalne opoznienie i2c
#define I2C_TIMER_DELAY 0xFF

//inicjalizacja i2c
void i2c_init(void)
{
    //100 kHz, prescaler = 1

    TWSR = 0;
    TWBR = ((F_CPU/SCL_CLOCK)-16)/2;
}
```

```
//0 - sukces
//argument - adres urzadzenia
unsigned char i2c_start(unsigned char address)
{
    uint32_t i2c_timer = 0;
    uint8_t twst;

    // warunek startu
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // czekanie na zakończenie transmisji
    i2c_timer = I2C_TIMER_DELAY;
    while (!(TWCR & (1<<TWINT)) && i2c_timer--);
    if(i2c_timer == 0)
        return 1;

    // sprawdzanie wartości statusu rejestru TWI
    twst = TW_STATUS & 0xF8;
    if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

    // wysyłanie adresu urządzenia
    TWDR = address;
    TWCR = (1<<TWINT) | (1<<TWEN);

    //czekanie na zakończenie transmisji - ACK/NACK
    i2c_timer = I2C_TIMER_DELAY;
    while (!(TWCR & (1<<TWINT)) && i2c_timer--);
    if(i2c_timer == 0)
        return 1;

    // sprawdzanie wartości statusu rejestru TWI
    twst = TW_STATUS & 0xF8;
    if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

    return 0;
}

//argument - adres urzadzenia
void i2c_start_wait(unsigned char address)
{
    uint32_t i2c_timer = 0;
    uint8_t twst;

    while ( 1 )
    {

        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        i2c_timer = I2C_TIMER_DELAY;
        while (!(TWCR & (1<<TWINT)) && i2c_timer--);

        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) continue;
```

```

TWDR = address;
TWCR = (1<<TWINT) | (1<<TWEN);

i2c_timer = I2C_TIMER_DELAY;
while(!(TWCR & (1<<TWINT)) && i2c_timer--);

twst = TW_STATUS & 0xF8;
if ( (twst == TW_MT_SLA_NACK ) || (twst ==TW_MR_DATA_NACK) )
{
    //urzadzenie zajete - wyslij STOP
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

    //czekaj na wykonanie STOP
    i2c_timer = I2C_TIMER_DELAY;
    while((TWCR & (1<<TWSTO)) && i2c_timer--);

    continue;
}
//if( twst != TW_MT_SLA_ACK) return 1;
break;
}

unsigned char i2c_rep_start(unsigned char address)
{
    return i2c_start( address );
}

void i2c_stop(void)
{
    uint32_t i2c_timer = 0;

    //warunek STOP
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

    //czekaj
    i2c_timer = I2C_TIMER_DELAY;
    while((TWCR & (1<<TWSTO)) && i2c_timer--);
}

//return: 0 - sukces
unsigned char i2c_write( unsigned char data )
{
    uint32_t i2c_timer = 0;
    uint8_t twst;

    // wyslij znak
    TWDR = data;
    TWCR = (1<<TWINT) | (1<<TWEN);

    //czekaj na zakonczenie transmisi
    i2c_timer = I2C_TIMER_DELAY;
    while(!(TWCR & (1<<TWINT)) && i2c_timer--);
}

```

```
if(i2c_timer == 0)
    return 1;

//sprawdz wartosc rejestrów TWI
twst = TW_STATUS & 0xF8;
if( twst != TW_MT_DATA_ACK) return 1;
return 0;
}

unsigned char i2c_readAck(void)
{
    uint32_t i2c_timer = 0;

    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    i2c_timer = I2C_TIMER_DELAY;
    while(!(TWCR & (1<<TWINT)) && i2c_timer--);
    if(i2c_timer == 0)
        return 0;

    return TWDR;
}

unsigned char i2c_readNak(void)
{
    uint32_t i2c_timer = 0;

    TWCR = (1<<TWINT) | (1<<TWEN);
    i2c_timer = I2C_TIMER_DELAY;
    while(!(TWCR & (1<<TWINT)) && i2c_timer--);
    if(i2c_timer == 0)
        return 0;

    return TWDR;
}
```

mpu6050/mpu6050registers.h

```
//mpu6050/mpu6050registers.h

#ifndef MPU6050REGISTERS_H_
#define MPU6050REGISTERS_H_

#define MPU6050_RA_XG_OFFSET_TC 0x00 // [7] PWR_MODE, [6:1] XG_OFFSET_TC, [0]
    _OTP_BNK_VLD
#define MPU6050_RA_YG_OFFSET_TC 0x01 // [7] PWR_MODE, [6:1] YG_OFFSET_TC, [0]
    _OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFSET_TC 0x02 // [7] PWR_MODE, [6:1] ZG_OFFSET_TC, [0]
    _OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN 0x03 // [7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN 0x04 // [7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN 0x05 // [7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFSET_H 0x06 // [15:0] XA_OFFSET
#define MPU6050_RA_XA_OFFSET_L_TC 0x07
#define MPU6050_RA_YA_OFFSET_H 0x08 // [15:0] YA_OFFSET
#define MPU6050_RA_YA_OFFSET_L_TC 0x09
#define MPU6050_RA_ZA_OFFSET_H 0x0A // [15:0] ZA_OFFSET
```

```
#define MPU6050_RA_ZA_OFFSET_L_TC 0x0B
#define MPU6050_RA_XG_OFFSET_USRH 0x13 // [15:0] XG_OFFSET_USR
#define MPU6050_RA_XG_OFFSET_USRL 0x14
#define MPU6050_RA_YG_OFFSET_USRH 0x15 // [15:0] YG_OFFSET_USR
#define MPU6050_RA_YG_OFFSET_USRL 0x16
#define MPU6050_RA_ZG_OFFSET_USRH 0x17 // [15:0] ZG_OFFSET_USR
#define MPU6050_RA_ZG_OFFSET_USRL 0x18
#define MPU6050_RA_SMPLRT_DIV 0x19
#define MPU6050_RA_CONFIG 0x1A
#define MPU6050_RA_GYRO_CONFIG 0x1B
#define MPU6050_RA_ACCEL_CONFIG 0x1C
#define MPU6050_RA_FF_THR 0x1D
#define MPU6050_RA_FF_DUR 0x1E
#define MPU6050_RA_MOT_THR 0x1F
#define MPU6050_RA_MOT_DUR 0x20
#define MPU6050_RA_ZRMOT_THR 0x21
#define MPU6050_RA_ZRMOT_DUR 0x22
#define MPU6050_RA_FIFO_EN 0x23
#define MPU6050_RA_I2C_MST_CTRL 0x24
#define MPU6050_RA_I2C_SLV0_ADDR 0x25
#define MPU6050_RA_I2C_SLV0_REG 0x26
#define MPU6050_RA_I2C_SLV0_CTRL 0x27
#define MPU6050_RA_I2C_SLV1_ADDR 0x28
#define MPU6050_RA_I2C_SLV1_REG 0x29
#define MPU6050_RA_I2C_SLV1_CTRL 0x2A
#define MPU6050_RA_I2C_SLV2_ADDR 0x2B
#define MPU6050_RA_I2C_SLV2_REG 0x2C
#define MPU6050_RA_I2C_SLV2_CTRL 0x2D
#define MPU6050_RA_I2C_SLV3_ADDR 0x2E
#define MPU6050_RA_I2C_SLV3_REG 0x2F
#define MPU6050_RA_I2C_SLV3_CTRL 0x30
#define MPU6050_RA_I2C_SLV4_ADDR 0x31
#define MPU6050_RA_I2C_SLV4_REG 0x32
#define MPU6050_RA_I2C_SLV4_DO 0x33
#define MPU6050_RA_I2C_SLV4_CTRL 0x34
#define MPU6050_RA_I2C_SLV4_DI 0x35
#define MPU6050_RA_I2C_MST_STATUS 0x36
#define MPU6050_RA_INT_PIN_CFG 0x37
#define MPU6050_RA_INT_ENABLE 0x38
#define MPU6050_RA_DMP_INT_STATUS 0x39
#define MPU6050_RA_INT_STATUS 0x3A
#define MPU6050_RA_ACCEL_XOUT_H 0x3B
#define MPU6050_RA_ACCEL_XOUT_L 0x3C
#define MPU6050_RA_ACCEL_YOUT_H 0x3D
#define MPU6050_RA_ACCEL_YOUT_L 0x3E
#define MPU6050_RA_ACCEL_ZOUT_H 0x3F
#define MPU6050_RA_ACCEL_ZOUT_L 0x40
#define MPU6050_RA_TEMP_OUT_H 0x41
#define MPU6050_RA_TEMP_OUT_L 0x42
#define MPU6050_RA_GYRO_XOUT_H 0x43
#define MPU6050_RA_GYRO_XOUT_L 0x44
#define MPU6050_RA_GYRO_YOUT_H 0x45
#define MPU6050_RA_GYRO_YOUT_L 0x46
#define MPU6050_RA_GYRO_ZOUT_H 0x47
#define MPU6050_RA_GYRO_ZOUT_L 0x48
#define MPU6050_RA_EXT_SENS_DATA_00 0x49
#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
```

```
#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
#define MPU6050_RA_EXT_SENS_DATA_07 0x50
#define MPU6050_RA_EXT_SENS_DATA_08 0x51
#define MPU6050_RA_EXT_SENS_DATA_09 0x52
#define MPU6050_RA_EXT_SENS_DATA_10 0x53
#define MPU6050_RA_EXT_SENS_DATA_11 0x54
#define MPU6050_RA_EXT_SENS_DATA_12 0x55
#define MPU6050_RA_EXT_SENS_DATA_13 0x56
#define MPU6050_RA_EXT_SENS_DATA_14 0x57
#define MPU6050_RA_EXT_SENS_DATA_15 0x58
#define MPU6050_RA_EXT_SENS_DATA_16 0x59
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS 0x61
#define MPU6050_RA_I2C_SLV0_DO 0x63
#define MPU6050_RA_I2C_SLV1_DO 0x64
#define MPU6050_RA_I2C_SLV2_DO 0x65
#define MPU6050_RA_I2C_SLV3_DO 0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67
#define MPU6050_RA_SIGNAL_PATH_RESET 0x68
#define MPU6050_RA_MOT_DETECT_CTRL 0x69
#define MPU6050_RA_USER_CTRL 0x6A
#define MPU6050_RA_PWR_MGMT_1 0x6B
#define MPU6050_RA_PWR_MGMT_2 0x6C
#define MPU6050_RA_BANK_SEL 0x6D
#define MPU6050_RA_MEM_START_ADDR 0x6E
#define MPU6050_RA_MEM_R_W 0x6F
#define MPU6050_RA_DMP_CFG_1 0x70
#define MPU6050_RA_DMP_CFG_2 0x71
#define MPU6050_RA_FIFO_COUNTH 0x72
#define MPU6050_RA_FIFO_COUNTL 0x73
#define MPU6050_RA_FIFO_R_W 0x74
#define MPU6050_RA_WHO_AM_I 0x75

#define MPU6050_TC_PWR_MODE_BIT 7
#define MPU6050_TC_OFFSET_BIT 6
#define MPU6050_TC_OFFSET_LENGTH 6
#define MPU6050_TC OTP_BNK_VLD_BIT 0

#define MPU6050_VDDIO_LEVEL_VLOGIC 0
#define MPU6050_VDDIO_LEVEL_VDD 1

#define MPU6050_CFG_EXT_SYNC_SET_BIT 5
#define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
#define MPU6050_CFG_DLPF_CFG_BIT 2
#define MPU6050_CFG_DLPF_CFG_LENGTH 3

#define MPU6050_EXT_SYNC_DISABLED 0x0
#define MPU6050_EXT_SYNC_TEMP_OUT_L 0x1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L 0x2
```

```
#define MPU6050_EXT_SYNC_GYRO_YOUT_L 0x3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L 0x4
#define MPU6050_EXT_SYNC_ACCEL_XOUT_L 0x5
#define MPU6050_EXT_SYNC_ACCEL_YOUT_L 0x6
#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L 0x7

#define MPU6050_DLPF_BW_256 0x00
#define MPU6050_DLPF_BW_188 0x01
#define MPU6050_DLPF_BW_98 0x02
#define MPU6050_DLPF_BW_42 0x03
#define MPU6050_DLPF_BW_20 0x04
#define MPU6050_DLPF_BW_10 0x05
#define MPU6050_DLPF_BW_5 0x06

#define MPU6050_GCONFIG_FS_SEL_BIT 4
#define MPU6050_GCONFIG_FS_SEL_LENGTH 2

#define MPU6050_GYRO_FS_250 0x00
#define MPU6050_GYRO_FS_500 0x01
#define MPU6050_GYRO_FS_1000 0x02
#define MPU6050_GYRO_FS_2000 0x03

#define MPU6050_ACONFIG_XA_ST_BIT 7
#define MPU6050_ACONFIG_YA_ST_BIT 6
#define MPU6050_ACONFIG_ZA_ST_BIT 5
#define MPU6050_ACONFIG_AFS_SEL_BIT 4
#define MPU6050_ACONFIG_AFS_SEL_LENGTH 2
#define MPU6050_ACONFIG_ACCEL_HPF_BIT 2
#define MPU6050_ACONFIG_ACCEL_HPF_LENGTH 3

#define MPU6050_ACCEL_FS_2 0x00
#define MPU6050_ACCEL_FS_4 0x01
#define MPU6050_ACCEL_FS_8 0x02
#define MPU6050_ACCEL_FS_16 0x03

#define MPU6050_DHPF_RESET 0x00
#define MPU6050_DHPF_5 0x01
#define MPU6050_DHPF_2P5 0x02
#define MPU6050_DHPF_1P25 0x03
#define MPU6050_DHPF_0P63 0x04
#define MPU6050_DHPF_HOLD 0x07

#define MPU6050_TEMP_FIFO_EN_BIT 7
#define MPU6050_XG_FIFO_EN_BIT 6
#define MPU6050_YG_FIFO_EN_BIT 5
#define MPU6050_ZG_FIFO_EN_BIT 4
#define MPU6050_ACCEL_FIFO_EN_BIT 3
#define MPU6050_SLV2_FIFO_EN_BIT 2
#define MPU6050_SLV1_FIFO_EN_BIT 1
#define MPU6050_SLV0_FIFO_EN_BIT 0

#define MPU6050_MULT_MST_EN_BIT 7
#define MPU6050_WAIT_FOR_ES_BIT 6
#define MPU6050_SLV_3_FIFO_EN_BIT 5
#define MPU6050_I2C_MST_P_NSR_BIT 4
#define MPU6050_I2C_MST_CLK_BIT 3
#define MPU6050_I2C_MST_CLK_LENGTH 4
```

```
#define MPU6050_CLOCK_DIV_348 0x0
#define MPU6050_CLOCK_DIV_333 0x1
#define MPU6050_CLOCK_DIV_320 0x2
#define MPU6050_CLOCK_DIV_308 0x3
#define MPU6050_CLOCK_DIV_296 0x4
#define MPU6050_CLOCK_DIV_286 0x5
#define MPU6050_CLOCK_DIV_276 0x6
#define MPU6050_CLOCK_DIV_267 0x7
#define MPU6050_CLOCK_DIV_258 0x8
#define MPU6050_CLOCK_DIV_500 0x9
#define MPU6050_CLOCK_DIV_471 0xA
#define MPU6050_CLOCK_DIV_444 0xB
#define MPU6050_CLOCK_DIV_421 0xC
#define MPU6050_CLOCK_DIV_400 0xD
#define MPU6050_CLOCK_DIV_381 0xE
#define MPU6050_CLOCK_DIV_364 0xF

#define MPU6050_I2C_SLV_RW_BIT 7
#define MPU6050_I2C_SLV_ADDR_BIT 6
#define MPU6050_I2C_SLV_ADDR_LENGTH 7
#define MPU6050_I2C_SLV_EN_BIT 7
#define MPU6050_I2C_SLV_BYTE_SW_BIT 6
#define MPU6050_I2C_SLV_REG_DIS_BIT 5
#define MPU6050_I2C_SLV_GRP_BIT 4
#define MPU6050_I2C_SLV_LEN_BIT 3
#define MPU6050_I2C_SLV_LEN_LENGTH 4

#define MPU6050_I2C_SLV4_RW_BIT 7
#define MPU6050_I2C_SLV4_ADDR_BIT 6
#define MPU6050_I2C_SLV4_ADDR_LENGTH 7
#define MPU6050_I2C_SLV4_EN_BIT 7
#define MPU6050_I2C_SLV4_INT_EN_BIT 6
#define MPU6050_I2C_SLV4_REG_DIS_BIT 5
#define MPU6050_I2C_SLV4_MST_DLY_BIT 4
#define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5

#define MPU6050_MST_PASS_THROUGH_BIT 7
#define MPU6050_MST_I2C_SLV4_DONE_BIT 6
#define MPU6050_MST_I2C_LOST_ARB_BIT 5
#define MPU6050_MST_I2C_SLV4_NACK_BIT 4
#define MPU6050_MST_I2C_SLV3_NACK_BIT 3
#define MPU6050_MST_I2C_SLV2_NACK_BIT 2
#define MPU6050_MST_I2C_SLV1_NACK_BIT 1
#define MPU6050_MST_I2C_SLV0_NACK_BIT 0

#define MPU6050_INTCFG_INT_LEVEL_BIT 7
#define MPU6050_INTCFG_INT_OPEN_BIT 6
#define MPU6050_INTCFG_LATCH_INT_EN_BIT 5
#define MPU6050_INTCFG_INT_RD_CLEAR_BIT 4
#define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT 3
#define MPU6050_INTCFG_FSYNC_INT_EN_BIT 2
#define MPU6050_INTCFG_I2C_BYPASS_EN_BIT 1
#define MPU6050_INTCFG_CLKOUT_EN_BIT 0

#define MPU6050_INTMODE_ACTIVEHIGH 0x00
#define MPU6050_INTMODE_ACTIVELOW 0x01

#define MPU6050_INTDRV_PUSH_PULL 0x00
```

```
#define MPU6050_INTDRV_OPENDRAIN 0x01

#define MPU6050_INTLATCH_50USPULSE 0x00
#define MPU6050_INTLATCH_WAITCLEAR 0x01

#define MPU6050_INTCLEAR_STATUSREAD 0x00
#define MPU6050_INTCLEAR_ANYREAD 0x01

#define MPU6050_INTERRUPT_FF_BIT 7
#define MPU6050_INTERRUPT_MOT_BIT 6
#define MPU6050_INTERRUPT_ZMOT_BIT 5
#define MPU6050_INTERRUPT_FIFO_OFLOW_BIT 4
#define MPU6050_INTERRUPT_I2C_MST_INT_BIT 3
#define MPU6050_INTERRUPT_PLL_RDY_INT_BIT 2
#define MPU6050_INTERRUPT_DMP_INT_BIT 1
#define MPU6050_INTERRUPT_DATA_RDY_BIT 0

#define MPU6050_DMPINT_5_BIT 5
#define MPU6050_DMPINT_4_BIT 4
#define MPU6050_DMPINT_3_BIT 3
#define MPU6050_DMPINT_2_BIT 2
#define MPU6050_DMPINT_1_BIT 1
#define MPU6050_DMPINT_0_BIT 0

#define MPU6050_MOTION_MOT_XNEG_BIT 7
#define MPU6050_MOTION_MOT_XPOS_BIT 6
#define MPU6050_MOTION_MOT_YNEG_BIT 5
#define MPU6050_MOTION_MOT_YPOS_BIT 4
#define MPU6050_MOTION_MOT_ZNEG_BIT 3
#define MPU6050_MOTION_MOT_ZPOS_BIT 2
#define MPU6050_MOTION_MOT_ZRMOT_BIT 0

#define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT 7
#define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT 4
#define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT 3
#define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT 2
#define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT 1
#define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT 0

#define MPU6050_PATHRESET_GYRO_RESET_BIT 2
#define MPU6050_PATHRESET_ACCEL_RESET_BIT 1
#define MPU6050_PATHRESET_TEMP_RESET_BIT 0

#define MPU6050_DETECT_ACCEL_ON_DELAY_BIT 5
#define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH 2
#define MPU6050_DETECT_FF_COUNT_BIT 3
#define MPU6050_DETECT_FF_COUNT_LENGTH 2
#define MPU6050_DETECT_MOT_COUNT_BIT 1
#define MPU6050_DETECT_MOT_COUNT_LENGTH 2

#define MPU6050_DETECT_DECREMENT_RESET 0x0
#define MPU6050_DETECT_DECREMENT_1 0x1
#define MPU6050_DETECT_DECREMENT_2 0x2
#define MPU6050_DETECT_DECREMENT_4 0x3

#define MPU6050_USERCTRL_DMP_EN_BIT 7
#define MPU6050_USERCTRL_FIFO_EN_BIT 6
#define MPU6050_USERCTRL_I2C_MST_EN_BIT 5
```

```
#define MPU6050_USERCTRL_I2C_IF_DIS_BIT 4
#define MPU6050_USERCTRL_DMP_RESET_BIT 3
#define MPU6050_USERCTRL_FIFO_RESET_BIT 2
#define MPU6050_USERCTRL_I2C_MST_RESET_BIT 1
#define MPU6050_USERCTRL_SIG_COND_RESET_BIT 0

#define MPU6050_PWR1_DEVICE_RESET_BIT 7
#define MPU6050_PWR1_SLEEP_BIT 6
#define MPU6050_PWR1_CYCLE_BIT 5
#define MPU6050_PWR1_TEMP_DIS_BIT 3
#define MPU6050_PWR1_CLKSEL_BIT 2
#define MPU6050_PWR1_CLKSEL_LENGTH 3

#define MPU6050_CLOCK_INTERNAL 0x00
#define MPU6050_CLOCK_PLL_XGYRO 0x01
#define MPU6050_CLOCK_PLL_YGYRO 0x02
#define MPU6050_CLOCK_PLL_ZGYRO 0x03
#define MPU6050_CLOCK_PLL_EXT32K 0x04
#define MPU6050_CLOCK_PLL_EXT19M 0x05
#define MPU6050_CLOCK_KEEP_RESET 0x07

#define MPU6050_PWR2_LP_WAKE_CTRL_BIT 7
#define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH 2
#define MPU6050_PWR2_STBY_XA_BIT 5
#define MPU6050_PWR2_STBY_YA_BIT 4
#define MPU6050_PWR2_STBY_ZA_BIT 3
#define MPU6050_PWR2_STBY_XG_BIT 2
#define MPU6050_PWR2_STBY_YG_BIT 1
#define MPU6050_PWR2_STBY_ZG_BIT 0

#define MPU6050_WAKE_FREQ_1P25 0x0
#define MPU6050_WAKE_FREQ_2P5 0x1
#define MPU6050_WAKE_FREQ_5 0x2
#define MPU6050_WAKE_FREQ_10 0x3

#define MPU6050_BANKSEL_PRFTCH_EN_BIT 6
#define MPU6050_BANKSEL_CFG_USER_BANK_BIT 5
#define MPU6050_BANKSEL_MEM_SEL_BIT 4
#define MPU6050_BANKSEL_MEM_SEL_LENGTH 5

#define MPU6050_WHO_AM_I_BIT 6
#define MPU6050_WHO_AM_I_LENGTH 6

#define MPU6050_DMP_MEMORY_BANKS 8
#define MPU6050_DMP_MEMORY_BANK_SIZE 256
#define MPU6050_DMP_MEMORY_CHUNK_SIZE 16

#endif
```

mpu6050/mpu6050.h

```
//mpu6050/mpu6050.h

#ifndef MPU6050_H_
#define MPU6050_H_

#include <avr/io.h>
#include "mpu6050registers.h"
```

```
//ustawienia i2c
#define MPU6050_I2CFLEURYPATH "../i2chw/i2cmaster.h"
#define MPU6050_I2CINIT 1

//definicje
#define MPU6050_ADDR (0x68 <<1) //AD0 --> GND: 0x68, AD0 --> VCC: 0x69

//definicje
#define MPU6050_GYRO_FS MPU6050_GYRO_FS_2000
#define MPU6050_ACCEL_FS MPU6050_ACCEL_FS_2

#define MPU6050_GYRO_LSB_250 131.0
#define MPU6050_GYRO_LSB_500 65.5
#define MPU6050_GYRO_LSB_1000 32.8
#define MPU6050_GYRO_LSB_2000 16.4

#define MPU6050_GGAIN MPU6050_GYRO_LSB_2000

#define MPU6050_ACCEL_LSB_2 16384.0
#define MPU6050_ACCEL_LSB_4 8192.0
#define MPU6050_ACCEL_LSB_8 4096.0
#define MPU6050_ACCEL_LSB_16 2048.0
#if MPU6050_ACCEL_FS == MPU6050_ACCEL_FS_2
#define MPU6050_AGAIN MPU6050_ACCEL_LSB_2

#define MPU6050_CALIBRATEDACCGYRO 1 //1 gdy skalibrowany
#if MPU6050_CALIBRATEDACCGYRO == 1
#define MPU6050_AXOFFSET 0
#define MPU6050_AYOFFSET 0
#define MPU6050_AZOFFSET 0
#define MPU6050_AXGAIN 16384.0
#define MPU6050_AYGAIN 16384.0
#define MPU6050_AZGAIN 16384.0
#define MPU6050_GXOFFSET -42
#define MPU6050_GYOFFSET 9
#define MPU6050_GZOFFSET -29
#define MPU6050_GXGAIN 16.4
#define MPU6050_GYGAIN 16.4
#define MPU6050_GZGAIN 16.4
#endif

//funkcje
extern void mpu6050_init();
extern uint8_t mpu6050_testConnection();

#if MPU6050_GETATTITUDE == 0
extern void mpu6050_getRawData(int16_t* ax, int16_t* ay, int16_t* az,
    int16_t* gx, int16_t* gy, int16_t* gz);
extern void mpu6050_getConvData(double* axg, double* ayg, double* azg,
    double* gxds, double* gyds, double* gzds);
#endif

extern void mpu6050_setSleepDisabled();
extern void mpu6050_setSleepEnabled();

extern int8_t mpu6050_readBytes(uint8_t regAddr, uint8_t length, uint8_t*
    *data);
```

```
extern int8_t mpu6050_readByte(uint8_t regAddr, uint8_t *data);
extern void mpu6050_writeBytes(uint8_t regAddr, uint8_t length, uint8_t *
    data);
extern void mpu6050_writeByte(uint8_t regAddr, uint8_t data);
extern int8_t mpu6050_readBits(uint8_t regAddr, uint8_t bitStart, uint8_t
    length, uint8_t *data);
extern int8_t mpu6050_readBit(uint8_t regAddr, uint8_t bitNum, uint8_t
    *data);
extern void mpu6050_writeBits(uint8_t regAddr, uint8_t bitStart, uint8_t
    length, uint8_t data);
extern void mpu6050_writeBit(uint8_t regAddr, uint8_t bitNum, uint8_t
    data);

#endif
```

mpu6050/mpu6050.c

```
//mpu6050/mpu6050.c

#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "mpu6050.h"

//i2c fluery lib
#include MPU6050_I2CFLEURYPATH

volatile uint8_t buffer[14];

int8_t mpu6050_readBytes(uint8_t regAddr, uint8_t length, uint8_t *data) {
    uint8_t i = 0;
    int8_t count = 0;
    if(length > 0) {
        //request register
        i2c_start(MPU6050_ADDR | I2C_WRITE);
        i2c_write(regAddr);
        _delay_us(10);
        //read data
        i2c_start(MPU6050_ADDR | I2C_READ);
        for(i=0; i<length; i++) {
            count++;
            if(i==length-1)
                data[i] = i2c_readNak();
            else
                data[i] = i2c_readAck();
        }
        i2c_stop();
    }
    return count;
}

int8_t mpu6050_readByte(uint8_t regAddr, uint8_t *data) {
    return mpu6050_readBytes(regAddr, 1, data);
```

```
}

void mpu6050_writeBytes(uint8_t regAddr, uint8_t length, uint8_t* data) {
    if(length > 0) {
        //write data
        i2c_start(MPU6050_ADDR | I2C_WRITE);
        i2c_write(regAddr); //reg
        for (uint8_t i = 0; i < length; i++) {
            i2c_write((uint8_t) data[i]);
        }
        i2c_stop();
    }
}

void mpu6050_writeByte(uint8_t regAddr, uint8_t data) {
    return mpu6050_writeBytes(regAddr, 1, &data);
}

int8_t mpu6050_readBits(uint8_t regAddr, uint8_t bitStart, uint8_t length,
    uint8_t *data) {
    int8_t count = 0;
    if(length > 0) {
        uint8_t b;
        if ((count = mpu6050_readByte(regAddr, &b)) != 0) {
            uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
            b &= mask;
            b >>= (bitStart - length + 1);
            *data = b;
        }
    }
    return count;
}

int8_t mpu6050_readBit(uint8_t regAddr, uint8_t bitNum, uint8_t *data) {
    uint8_t b;
    uint8_t count = mpu6050_readByte(regAddr, &b);
    *data = b & (1 << bitNum);
    return count;
}

void mpu6050_writeBits(uint8_t regAddr, uint8_t bitStart, uint8_t length,
    uint8_t data) {
    if(length > 0) {
        uint8_t b = 0;
        if (mpu6050_readByte(regAddr, &b) != 0) {
            uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
            data <<= (bitStart - length + 1);
            data &= mask;
            b &= ~mask;
            b |= data;
            mpu6050_writeByte(regAddr, b);
        }
    }
}

void mpu6050_writeBit(uint8_t regAddr, uint8_t bitNum, uint8_t data) {
    uint8_t b;
```

```
mpu6050_readByte(regAddr, &b);
b = (data != 0) ? (b | (1 << bitNum)) : (b & ~(1 << bitNum));
mpu6050_writeByte(regAddr, b);
}

void mpu6050_setSleepDisabled() {
    mpu6050_writeBit(MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_SLEEP_BIT, 0);
}

void mpu6050_setSleepEnabled() {
    mpu6050_writeBit(MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_SLEEP_BIT, 1);
}

//test polaczenia
uint8_t mpu6050_testConnection() {
    mpu6050_readBits(MPU6050_RA_WHO_AM_I, MPU6050_WHO_AM_I_BIT,
                      MPU6050_WHO_AM_I_LENGTH, (uint8_t *)buffer);
    if(buffer[0] == 0x34)
        return 1;
    else
        return 0;
}

//inicjalizacja czujnika
void mpu6050_init() {
    #if MPU6050_I2CINIT == 1
    //init i2c
    i2c_init();
    _delay_us(10);
    #endif

    //odczekaj na start
    _delay_ms(100);

    mpu6050_setSleepDisabled();
    _delay_ms(10);

    //zrodlo zegara *****
    mpu6050_writeBits(MPU6050_RA_PWR_MGMT_1, MPU6050_PWR1_CLKSEL_BIT,
                      MPU6050_PWR1_CLKSEL_LENGTH, MPU6050_CLOCK_PLL_XGYRO);
    //set DLPF bandwidth to 42Hz
    mpu6050_writeBits(MPU6050_RA_CONFIG, MPU6050_CFG_DLPF_CFG_BIT,
                      MPU6050_CFG_DLPF_CFG_LENGTH, MPU6050_DLPF_BW_42);
    //sample rate
    mpu6050_writeByte(MPU6050_RA_SMPLRT_DIV, 4);
    //gyro range
    mpu6050_writeBits(MPU6050_RA_GYRO_CONFIG, MPU6050_GCONFIG_FS_SEL_BIT,
                      MPU6050_GCONFIG_FS_SEL_LENGTH, MPU6050_GYRO_FS);
    //accel range
    mpu6050_writeBits(MPU6050_RA_ACCEL_CONFIG, MPU6050_ACONFIG_AFS_SEL_BIT,
                      MPU6050_ACONFIG_AFS_SEL_LENGTH, MPU6050_ACCEL_FS);
}

//odczyt danych
void mpu6050_getRawData(int16_t* ax, int16_t* ay, int16_t* az, int16_t*
    gx, int16_t* gy, int16_t* gz) {
    mpu6050_readBytes(MPU6050_RA_ACCEL_XOUT_H, 14, (uint8_t *)buffer);
```

```
*ax = (((int16_t)buffer[0]) << 8) | buffer[1];
*ay = (((int16_t)buffer[2]) << 8) | buffer[3];
*az = (((int16_t)buffer[4]) << 8) | buffer[5];
*gx = (((int16_t)buffer[8]) << 8) | buffer[9];
*gy = (((int16_t)buffer[10]) << 8) | buffer[11];
*gz = (((int16_t)buffer[12]) << 8) | buffer[13];
}

//konwersja na g i dps
void mpu6050_getConvData(double* axg, double* ayg, double* azg, double*
    gxds, double* gyds, double* gzds) {
int16_t ax = 0;
int16_t ay = 0;
int16_t az = 0;
int16_t gx = 0;
int16_t gy = 0;
int16_t gz = 0;
mpu6050_getRawData(&ax, &ay, &az, &gx, &gy, &gz);

#if MPU6050_CALIBRATEDACCGYRO == 1
*axg = (double)(ax-MPU6050_AXOFFSET)/MPU6050_AXGAIN;
*ayg = (double)(ay-MPU6050_AYOFFSET)/MPU6050_AYGAIN;
*azg = (double)(az-MPU6050_AZOFFSET)/MPU6050_AZGAIN;
*gxds = (double)(gx-MPU6050_GXOFFSET)/MPU6050_GXGAIN;
*gyds = (double)(gy-MPU6050_GYOFFSET)/MPU6050_GYGAIN;
*gzds = (double)(gz-MPU6050_GZOFFSET)/MPU6050_GZGAIN;
}
```

5.3 Obsługa wyświetlacza HD44780

LCD/lcd44780.h

```
//LCD/lcd44780.h

#ifndef LCD_H_
#define LCD_H_

// rozdzielczosc wyświetlacza LCD (wiersze/kolumny)
#define LCD_ROWS 2 // ilosc wierszy wyświetlacza LCD
#define LCD_COLS 16 // ilosc kolumn wyświetlacza LCD

// 0 - pin RW podłączony na stale do GND
// 1 - pin RW podłączony do mikrokontrolera
#define USE_RW 0

#define LCD_D7PORT B
#define LCD_D7 0
#define LCD_D6PORT D
#define LCD_D6 7
#define LCD_D5PORT D
#define LCD_D5 5
#define LCD_D4PORT D
#define LCD_D4 4

#define LCD_RSPORT C
#define LCD_RS 2
```

```

#define LCD_RWPORT B
#define LCD_RW 1

#define LCD_EPORT C
#define LCD_E 3

#define USE_LCD_LOCATE 1 // ustawia kursor na wybranej pozycji Y,X
(Y=0-3, X=0-n)

#define USE_LCD_CHAR 1 // wysyla pojedynczy znak jako argument funkcji

#define USE_LCD_STR_P 1 // wysyla string umieszczony w pamieci FLASH
#define USE_LCD_STR_E 1 // wysyla string umieszczony w pamieci FLASH

#define USE_LCD_INT 1 // wyswietla liczbe dziesietna na LCD
#define USE_LCD_HEX 1 // wyswietla liczbe szesnastkowa na LCD

#define USE_LCD_DEFCHAR 1 // wysyla zdefiniowany znak z pamieci RAM
#define USE_LCD_DEFCHAR_P 1 // wysyla zdefiniowany znak z pamieci FLASH
#define USE_LCD_DEFCHAR_E 1 // wysyla zdefiniowany znak z pamieci EEPROM

#define USE_LCD_CURSOR_ON 0 // obsluga wlaczania/wylaczania kursora
#define USE_LCD_CURSOR_BLINK 0 // obsluga wlaczania/wylaczania migania
    kursora
#define USE_LCD_CURSOR_HOME 0 // ustawia kursor na pozycji poczatkowej

// definicje adresow w DDRAM dla roznych wyswietlaczy
// inne sa w wyswietlaczach 2wierszowych i w 4wierszowych
#if ( (LCD_ROWS == 4) && (LCD_COLS == 16) )
#define LCD_LINE1 0x00 // adres 1 znaku 1 wiersza
#define LCD_LINE2 0x28 // adres 1 znaku 2 wiersza
#define LCD_LINE3 0x14 // adres 1 znaku 3 wiersza
#define LCD_LINE4 0x54 // adres 1 znaku 4 wiersza
#else
#define LCD_LINE1 0x00 // adres 1 znaku 1 wiersza
#define LCD_LINE2 0x40 // adres 1 znaku 2 wiersza
#define LCD_LINE3 0x10 // adres 1 znaku 3 wiersza
#define LCD_LINE4 0x50 // adres 1 znaku 4 wiersza
#endif

// Makra upraszczajace dostep do portow
// *** PORT
#define PORT(x) SPORTE(x)
#define SPORTE(x) (PORT##x)
// *** PIN
#define PIN(x) SPIN(x)
#define SPIN(x) (PIN##x)
// *** DDR
#define DDR(x) SDDR(x)
#define SDDR(x) (DDR##x)

// Komendy sterujace
#define LCDC_CLS 0x01
#define LCDC_HOME 0x02
#define LCDC_ENTRY 0x04
#define LCDC_ENTRYR 0x02
#define LCDC_ENTRYL 0

```

```
#define LCDC_MOVE      0x01
#define LCDC_ONOFF     0x08
#define LCDC_DISPLAYON  0x04
#define LCDC_CURSORON   0x02
#define LCDC_CURSOROFF  0
#define LCDC_BLINKON    0x01
#define LCDC_SHIFT      0x10
#define LCDC_SHIFTDISP  0x08
#define LCDC_SHIFTR     0x04
#define LCDC_SHIFTL     0
#define LCDC_FUNC       0x20
#define LCDC_FUNC8B     0x10
#define LCDC_FUNC4B     0
#define LCDC_FUNC2L     0x08
#define LCDC_FUNC1L     0
#define LCDC_FUNC5x10   0x04
#define LCDC_FUNC5x7    0
#define LCDC_SET_CGRAM  0x40
#define LCDC_SET_DDRAM  0x80

// deklaracje funkcji na potrzeby innych modułów
void lcd_init(void);
void lcd_cls(void);
void lcd_str(char * str);

void lcd_locate(uint8_t y, uint8_t x);

void lcd_char(char c);
void lcd_str_P(const char * str);
void lcd_str_E(char * str);
void lcd_int(int val);
void lcd_hex(uint32_t val);
void lcd_defchar(uint8_t nr, uint8_t *def_znak);
void lcd_defchar_P(uint8_t nr, const uint8_t *def_znak);
void lcd_defchar_E(uint8_t nr, uint8_t *def_znak);

void lcd_home(void);
void lcd_cursor_on(void);
void lcd_cursor_off(void);
void lcd_blink_on(void);
void lcd_blink_off(void);

#endif /* LCD_H_ */
```

LCD/lcd44780.c

5.4 Regulacja położenia

Implementacja sterowania z rozdz. 4

Program zawiera pakiet do rozwiązywania macierzy "matrix":

matrix/matrix.h

```
//matrix/matrix.h
```

```
#include <stdlib.h>

struct _Matrix
{
    int row_size;
    int col_size;
    float **matrix_entry;
};

typedef struct _Matrix Matrix;

//alokacja pamięci dla macierzy
Matrix *matrix_alloc(int row_size, int col_size);

//kopiowanie macierzy
void matrix_copy(Matrix *matrix1, Matrix *matrix2);

//mnożenie
Matrix *matrix_multiply(const Matrix *matrix1, const Matrix *matrix2);

//zwolnienie pamięci
void matrix_free(Matrix *matrix);

//funkcje pomocnicze
void row_divide(Matrix *matrix, int pivot);
void row_operation(Matrix *multiplier_matrix, Matrix *matrix, int pivot,
    int row_index);
void matrix_row_reduce( Matrix *matrix, int zero_control);

//odejmowanie
void matrix_subtract(Matrix *result, Matrix *matrix1, Matrix *matrix2);

//dodawanie
void matrix_add(Matrix *result, Matrix *matrix1, Matrix *matrix2);

//macierz odwrotna
void matrix_invert(Matrix *inverse_matrix);

//transpozycja
Matrix * matrix_transpose(Matrix *m);

//sprawdzanie zer
void error_zeros( Matrix *matrix, int control_index);
```

matrix/matrix.c

```
//matrix/matrix.c

#include "matrix.h"

Matrix *matrix_alloc(int row_size, int col_size)
{
    int j;
    Matrix *new_matrix = malloc(sizeof(Matrix));

    new_matrix->row_size = row_size;
    new_matrix->col_size = col_size;
    new_matrix->matrix_entry = malloc( new_matrix->row_size *sizeof(float
```

```

        *);
    for(j = 0 ; j < new_matrix->row_size ; j++)
    {
        new_matrix->matrix_entry[j] = malloc(
            new_matrix->col_size*sizeof(float) );
    }

    return new_matrix;
}

void matrix_copy(Matrix *matrix1, Matrix *matrix2)
{
    int i, j;
    for (i = 0; i < matrix1->row_size; i += 1)
    {
        for (j = 0; j < matrix1->col_size; j += 1)
        {
            matrix2->matrix_entry[i][j] = matrix1->matrix_entry[i][j];
        }
    }
}

Matrix *matrix_multiply(const Matrix *matrix1, const Matrix *matrix2)
{
    int i, j,k, sum;
    Matrix *result = matrix_alloc( matrix1->row_size,matrix2->col_size);
    for (i = 0; i < matrix1->row_size; i += 1)
    {
        for (k = 0; k < matrix2->col_size; k += 1)
        {
            sum = 0;

            for (j = 0; j < matrix1->col_size; j += 1)
            {
                sum += matrix1->matrix_entry[i][j] * matrix2->matrix_entry[j][k];
            }

            result->matrix_entry[i][k] = sum;
        }
    }
    return result;
}

void matrix_free( Matrix *matrix)
{
    int j;
    for(j = 0 ; j < matrix->row_size ; j++)
    {
        free(matrix->matrix_entry[j]);
    }
    free(matrix->matrix_entry);
    free(matrix);
}

void row_divide(Matrix *matrix, int pivot)
{
    int j;
    float divisor = matrix->matrix_entry[pivot][pivot],

```

```

        result;

    for(j = pivot; j < matrix->col_size; j++)
    {
        result = (matrix->matrix_entry[pivot][j] / divisor);
        matrix->matrix_entry[pivot][j] = result;
    }

}

void row_operation(Matrix *multiplier_matrix,Matrix *matrix, int pivot,
    int row_index)
{
    int j;
    float multiplier = (matrix->matrix_entry[row_index][pivot] /
        matrix->matrix_entry[pivot][pivot]);
    if(multiplier_matrix != NULL)
    {
        multiplier_matrix ->matrix_entry[row_index][pivot] = multiplier;
    }

    for(j=0; j < matrix->col_size; j++)
    {
        matrix->matrix_entry[row_index][j] -= multiplier *
            matrix->matrix_entry[pivot][j];
    }
}

void matrix_row_reduce( Matrix *matrix, int zero_control )
{
    int pivot, row_index;
    float multiplier;
    for( pivot = 0; pivot < matrix->row_size ; pivot++)
    {

        error_zeros(matrix, zero_control);
        if( (matrix->matrix_entry[pivot][pivot] != 1) &&
            (matrix->matrix_entry[pivot][pivot] != 0) )
        {
            row_divide(matrix, pivot);
        }

        for (row_index = pivot+1; row_index < matrix->row_size; row_index++)
        {
            if (matrix->matrix_entry[pivot][pivot] != 0)
            {
                row_operation(NULL,matrix, pivot, row_index);
            }
        }

        for(row_index = pivot-1; row_index >=0; row_index --)
        {
            if (matrix->matrix_entry[pivot][pivot] != 0)
            {
                row_operation(NULL,matrix, pivot, row_index);
            }
        }
    }
}

```

```

        }

}

void matrix_subtract(Matrix *result, Matrix *matrix1, Matrix *matrix2)
{
    int i, j;

    for(i = 0; i < matrix1->row_size; i += 1)
    {
        for (j = 0; j < matrix1->col_size; j += 1)
        {
            result->matrix_entry[i][j] = matrix1->matrix_entry[i][j] -
                matrix2->matrix_entry[i][j];
        }
    }
}

void matrix_add(Matrix *result, Matrix *matrix1, Matrix *matrix2)
{
    int i, j;

    for (i = 0; i < matrix1->row_size; i += 1)
    {
        for (j = 0; j < matrix1->col_size; j += 1)
        {
            result->matrix_entry[i][j] = matrix1->matrix_entry[i][j] +
                matrix2->matrix_entry[i][j];
        }
    }
}

Matrix *matrix_transpose(Matrix *m) {
    Matrix *t = matrix_alloc(m->col_size, m->row_size);
    int i, j;
    for(i=0;i<m->row_size;i++) {
        for(j=0;j<m->col_size;j++) {
            t->matrix_entry[j][i]=m->matrix_entry[i][j];
        }
    }
    return t;
}

void matrix_invert(Matrix *inverse_matrix)
{
    int j, k;
    Matrix *temp_matrix = matrix_alloc(inverse_matrix->row_size,
        inverse_matrix->col_size * 2);

    matrix_copy(inverse_matrix, temp_matrix);

    for(j = 0; j < temp_matrix->row_size; j++)
    {
        for(k = 3; k < temp_matrix->col_size; k++)
        {
            if( j+3 == k)
            {
                temp_matrix->matrix_entry[j][k] = 1;
            }
        }
    }
}

```

```
    else
    {
        temp_matrix->matrix_entry[j][k] = 0;
    }
}

matrix_row_reduce(temp_matrix, temp_matrix->row_size);

for(j = 0; j < temp_matrix->row_size; j++)
{
    for(k = 3; k < temp_matrix->col_size; k++)
    {
        inverse_matrix->matrix_entry[j][k-3] = temp_matrix->matrix_entry[j][k];
    }
}

matrix_free(temp_matrix);
}

void error_zeros( Matrix *matrix, int control_index)
{
    int i,j,count;
    for(i=0; i<matrix->row_size; i++)
    {
        count=0;
        for(j = 0; j < matrix->col_size; j++)
        {
            if( matrix->matrix_entry[i][j] == 0)
            {
                count++;
            }
            else
            {
                return;
            }
        }
    }
}
```

main.c

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <math.h> //include libm

#include "matrix/matrix.h"

#include "mpu6050/mpu6050.h"
// #define UART_BAUD_RATE 57600
// #include "uart uart.h"
#include "LCD/lcd44780.h"

volatile uint8_t pwml;
```

```

volatile int licznik;

int main(void) {

    //programowy pwm
    DDRC |= (1<<PC0) | (1<<PC1);
    PORTC |= (1<<PC0);
    PORTC |= (1<<PC1);

    TCCR2B |= (1<<WGM21); // tryb CTC
    TCCR2B |= (1<<CS20); // preskaler = 1
    OCR2B = 199; // dodatkowy podzial częstotliwości przez 200
    TIMSK2 |= (1<<OCIE2B);

    /*TCCR2 |= (1<<WGM21); // tryb CTC
    TCCR2 |= (1<<CS20); // preskaler = 1
    OCR2 = 199; // dodatkowy podzial częstotliwości przez 200
    TIMSK |= (1<<OCIE2); */

    int16_t ax = 0;
    int16_t ay = 0;
    int16_t az = 0;
    int16_t gx = 0;
    int16_t gy = 0;
    int16_t gz = 0;
    double axg = 0;
    double ayg = 0;
    double azg = 0;
    double gxds = 0;
    double gyds = 0;
    double gzds = 0;

    //uart_init(UART_BAUD_SELECT(UART_BAUD_RATE, F_CPU));
    sei();

    mpu6050_init();
    _delay_ms(50);

    int maxa=0,maxg=0,aax,pom;

    lcd_init();
    int licz=1,suma=0,_ax,fi,bak;
    int i;
    int ii,jj;

    //pid
    int wzmocnienieP=100; //Wzmocnienie
    float stalaI=0.4; //Stała czasowa całkowania
    float stalaD=0.1; //Stała czasowa różniczkowania
    float czas=0.15; //Czas zmian wielkości
    int predkosc=125; //Predkosc silników
    int sterowanie,uchyb,uchyb_pop=0;

    //KF
    Matrix *x_post = matrix_alloc(4,1);
    for(ii=0;ii<4;ii++) {
        for(jj=0;jj<1;jj++) {

```

```

        x_post->matrix_entry[ii][jj]=0;
    }
}
Matrix *P_post = matrix_alloc(4, 4);
for(ii=0;ii<4;ii++) {
    for(jj=0;jj<4;jj++) {
        P_post->matrix_entry[ii][jj]=1;
    }
}
Matrix *V = matrix_alloc(4, 4);
for(ii=0;ii<4;ii++) {
    for(jj=0;jj<4;jj++) {
        if(ii==jj) V->matrix_entry[ii][ii]=0.8;
        else V->matrix_entry[ii][jj]=0;
    }
}

Matrix *W = matrix_alloc(2, 2);
W->matrix_entry[0][0]=0.02;
W->matrix_entry[0][1]=0;
W->matrix_entry[1][0]=0;
W->matrix_entry[1][1]=4;

for(;;) {
    lcd_cls();
    mpu6050_getRawData(&ax, &ay, &az, &gx, &gy, &gz);
    mpu6050_getConvData(&axg, &ayg, &azg, &gxds, &gyds, &gzds);

    //Wypełnianie macierzy A
    Matrix *A;
    A = matrix_alloc(4, 4);

    for(ii=0;ii<4;ii++) {
        for(jj=0;jj<4;jj++) {
            A->matrix_entry[ii][jj]=0;
        }
    }
    A->matrix_entry[0][1]=1;    A->matrix_entry[2][3]=1;
    A->matrix_entry[1][1]=-0.1192;  A->matrix_entry[1][2]=6.7359;
    A->matrix_entry[3][1]=-1.2764;  A->matrix_entry[3][2]=177.1575;

    //Wypełnianie macierzy B
    Matrix *B;
    B = matrix_alloc(4, 1);

    for(ii=0;ii<4;ii++) {
        B->matrix_entry[ii][0]=0;
    }
    B->matrix_entry[1][0]=1.1923;    B->matrix_entry[3][0]=12.7640;

    //Wypełnianie macierzy C
    Matrix *C;
    C = matrix_alloc(2, 4);

    for(ii=0;ii<2;ii++) {
        for(jj=0;jj<4;jj++) {
            C->matrix_entry[ii][jj]=0;
        }
    }
}

```

```

        }

    }

C->matrix_entry[0][0]=1;    C->matrix_entry[1][2]=1;

//u=pwm1

Matrix *x;
x = matrix_alloc(4,1);
x->matrix_entry[0][0]=0.000628*licznik; //m
##########
x->matrix_entry[1][0]=6.28/(-0.0588*pwm1+20); //m/s
x->matrix_entry[2][0]=(ax*9)/1600; //16000 - 90 stopni
x->matrix_entry[3][0]=gy; //

Matrix *y = matrix_alloc(2,1);
y->matrix_entry[0][0]=x->matrix_entry[0][0];
y->matrix_entry[1][0]=x->matrix_entry[2][0];

lcd_cls();
lcd_int((int)x->matrix_entry[2][0]);
_delay_ms(500);

//FILTR KALMANA
Matrix *Ax = matrix_alloc(4,1);
Matrix *Bu = matrix_alloc(4,1);
Matrix *x_pri = matrix_alloc(4,1);
Matrix *AP = matrix_alloc(4,4);
Matrix *AT = matrix_alloc(4,4);
Matrix *APAT = matrix_alloc(4,4);
Matrix *P_pri = matrix_alloc(4,4);
Matrix *eps = matrix_alloc(2,1);
Matrix *CX = matrix_alloc(2,1);

// x(t+1|t) = Ax(t|t) + Bu(t)
Ax=matrix_multiply(A, x_post);
Bu->matrix_entry[0][0]=0;
Bu->matrix_entry[1][0]=pwm1*B->matrix_entry[1][0];
Bu->matrix_entry[2][0]=0;
Bu->matrix_entry[3][0]=pwm1*B->matrix_entry[3][0];
matrix_add(x_pri, Ax, Bu);

// P(t+1|t) = AP(t|t)A^T + V
AP=matrix_multiply(A, P_post);
AT=matrix_transpose(A);
APAT=matrix_multiply(AP, AT);
matrix_add(P_pri, APAT, V);

// eps(t) = y(t) - CX(t|t-1)
CX=matrix_multiply(C, x_pri);
matrix_subtract(eps,y,CX);

Matrix *CP = matrix_alloc(2,4);
Matrix *CPCT = matrix_alloc(2,2);
Matrix *CT = matrix_alloc(4,2);
Matrix *S = matrix_alloc(2,2);
Matrix *PCT = matrix_alloc(4,2);
//Matrix *S1 = matrix_alloc(2,2);
Matrix *K = matrix_alloc(4,2);

```

```

Matrix *Keps = matrix_alloc(4,1);
Matrix *KS = matrix_alloc(4,2);
Matrix *KT = matrix_alloc(2,4);
Matrix *KSCT = matrix_alloc(4,4);

// S(t) = CP(t|t-1)C^T + W
CP=matrix_multiply(C, P_pri);
CT=matrix_transpose(C);
CPCT=matrix_multiply(CP, CT);
matrix_add(S, CPCT, W);

// K(t) = P(t|t-1)C^TS(t)^-1
PCT=matrix_multiply(P_pri, CT);
matrix_invert(S);
K=matrix_multiply(PCT, S); //S^-1

// x(t|t) = x(t|t-1) + K(t)eps(t)
Keps=matrix_multiply(K, eps);
matrix_add(x_post, x_pri, Keps);

// P(t|t) = P(t|t-1) - K(t)S(t)K(t)^T
matrix_invert(S);
KS=matrix_multiply(K, S); //S
KT=matrix_transpose(K);
KSCT=matrix_multiply(KS, KT);
matrix_subtract(P_post, P_pri, KSCT);

//PID
fi=x_post->matrix_entry[2][0];
uchyb = -1*fi;
sterowanie = wzmocnienieP*uchyb + stalaD*((uchyb - uchyb_pop)/czas); //
+ stalaI*calka(uchyb);
pwml = predkosc - sterowanie;
uchyb_pop=uchyb;

}

}

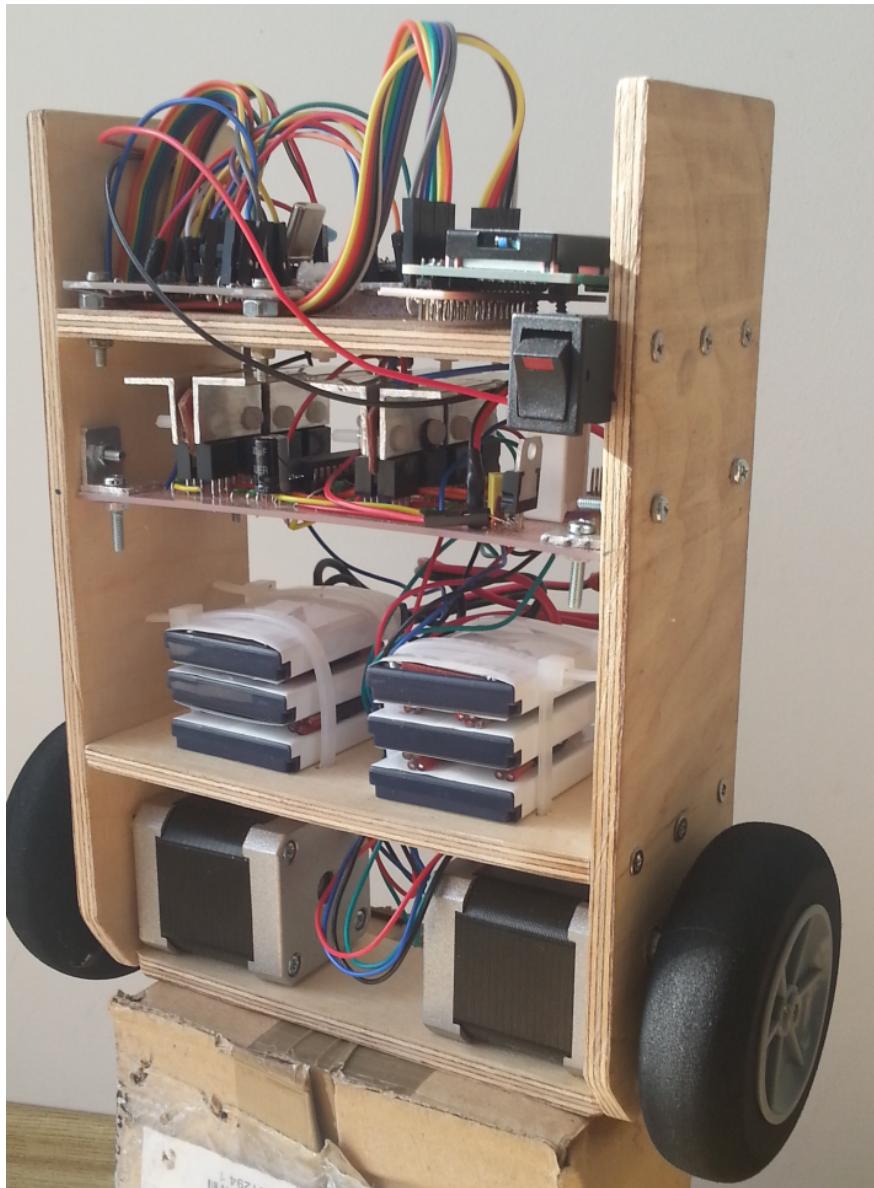
ISR( TIMER2_COMPB_vect )
{
    static uint8_t cnt; // definicja licznika PWM
    if(cnt>=pwml) PORTC |= (1<<PC0); else PORTC &= ~ (1<<PC0);
    cnt++;
}

```

6 Podsumowanie

Okres budowy robota to ok. od 18 grudnia do 19 lutego, czyli nieco ponad 2 miesiące. Robot zostanie z pewnością rozbudowany o szybszy procesor, ładowarkę, zdalne sterowanie, czy sygnalizację LED. Konstrukcja idealnie nadaje się do doskonalenia programowania i konstruowania sterowników silników krokowych i automatycznej regulacji.

Po ukończeniu budowy widać dużą słabość użytych płyt jednostronnych uniwersalnych na elektronikę, ponieważ zajmują dużo miejsca, ścieżki się odklejają pod wpływem kilku rozlutowań, a także nie dają możliwości ekranowania ścieżek sygnałowych, co przysparza wielu zakłóceń. Można również zauważyć dużą niedoskonałość własnych sterowników silników krokowych, ponieważ zapewniają sterowanie tylko na pół kroku, co wprawia całą konstrukcję w dość duże drgania. Środek masy robota powinien być nieco wyżej, gdyż ułatwia to sterowanie tego typu robotem.



Rysunek 42: Konstrukcja na dzień 21.02.2016

7 Niezrealizowane pomysły

Na niektóre pomysły nie wystarczyło czasu, ale planuję je wykonać w najbliższym czasie.

7.1 Ładowarka do akumulatorów

Ładowarka działająca w oparciu o mikrokontroler - najpewniej AVR ATmega8.

Ładowanie akumulatorów Li-ion składa się z trzech faz i wymaga sporej wiedzy na temat charakterystyk pojedynczego ogniska, aby je bezpiecznie i szybko ładować.

7.2 Pomiar prądu i napięcia baterii przez ADC

Implementacja programu non-stop sprawdzającego stan baterii i wyłączającego silniki, przy wykryciu zbyt niskiego napięcia lub zbyt wysokiego prądu. Prawdopodobne wykorzystanie ładowarki, gdyż byłaby ona podłączona do każdego ogniska z osobna (najprawdopodobniej poprzez zastosowanie balancerów).

7.3 Zdalne sterowanie przez telefon z Androidem

Dodanie modułu Wi-Fi, sprzężenie go z mikrokontrolerem odpowiadającym za regulację położenia i dodanie paru linii sterowania do sterownika silników, aby umożliwić skręcanie.

Napisanie programu w Javie na urządzenia Android.

8 Kosztorys

Przedmiot	Ilość sztuk	Cena (zł)	Łączna cena z wysyłką (zł)
Sklejka 6mm 1m ²	1	4.50	4.50
Koło piankowe φ80mm	2	20.00	40.00
Bipolarny silnik krokowy JK42HS48-1684	2	59.00	130.00
Łożysko kulowe otwór φ5mm	2	9.00	18.00
Akumulator Li-ion 1650 mAh Nokia 3310	6	12.00	72.00
Trytka (plastikowa opaska zaciskowa)	4	1.00	4.00
Śruby, wkręty do drewna, nakręty i podkładki	po 40	-	10.00
Śruby nylonowe M3	12	0.50	6.00
Nakrętki nylonowe M3	4	1.00	4.00
MOSFET kanał typu n STP36NF06	8	1.50	12.00
MOSFET kanał typu p IRF9530N	8	2.00	16.00
Zestaw przewodów do płytki stykowej	1	16.00	16.00
Grubsze przewody baterijne 1m	1	2.00	2.00
Kondensatory 12p, 100n, 100u, 330n	10	-	3.10
Rezonator kwarcowy 20 MHz	1	1.00	1.00
Dławik osiowy 10u	2	0.25	0.50
Rezystor 10k, 4.7k i 560 0.5W	7	0.10	0.70
Aluminiowa listwa ok. 1m	1	2.00	2.00
Podkładki termiczne 3mm	2x5	3.00	6.00
Mikrokontroler ATmega8	1	8.00	8.00
Mikrokontroler ATmega168	1	25.00	25.00
Podstawkę precyzyjną 28 pin	2	2.00	4.00
Jednorzędowa listwa goldpin	1	2.00	2.00
Dwurzędowa listwa goldpin	1	3.00	3.00
Rezystor bocznikowy 0R1 5W	1	0.30	0.30
Płytkę uniwersalną PDU27	1	11.30	11.30
Płytkę uniwersalną PDU11	1	7.90	7.90
Stabilizator napięcia 7805	1	0.80	0.80
Przełącznik 12V 13A	1	2.00	2.00
Żyroskop i akcelerometr MPU-6050	1	16.00	20.00
Wyświetlacz LCD HD44780 16x2	1	20.00	20.00
Łączna cena obudowy	-	-	78.50
Łączna cena sterownika silników i baterii	-	-	81.75
Łączna cena baterii	-	-	74.00
Łączna cena silników	-	-	130.00
Łączna cena układu regulacji położenia	-	-	79.70
Łączna cena dodatkowych narzędzi	-	-	10.00
Łączny koszt budowy robota	-	-	453.95

9 Bibliografia

- M. Kardaś - Mikrokontrolery AVR Język C - podstawy programowania, ATNEL, Szczecin 2013,
- T. Kaczorek - Teoria sterowania, PWN, Warszawa, 1996,
- H. Tunia, M.P.Kaźmierkowski, Podstawy automatyki napędu elektrycznego, PWN,
- Owczarek J., Pochanke A., Sochocki R. i inni: Elektryczne maszynowe elementy automatyki, WNT. Warszawa, 1983,
- Kulka Z., Nadachowski M. - Liniowe układy scalone i ich zastosowanie , WKiŁ, Warszawa 1975,
- Pałczyński B., Stefański W.: Półprzewodnikowe stabilizatory prądu i napięcia stałego. MON, Warszawa 1971,
- B. Kernighan, D. Ritchie - The C Programming Language
- C. Noviello - Mastering STM32,
- Wykłady dr J. Abotta na University of Utah, kierunek Telerobotics - State-Space Control Systems,
- B. Hejrati, K. L. Crandall, J. M. Hollerbach, and J. J. Abbott, "Kinesthetic Force Feedback and Belt Control for the Treadport Locomotion Interface," IEEE Trans. Haptics, 8(2):176-187, 2015,
- Dokumentacje techniczne użytych elementów elektronicznych z internetu