AKADEMIA NAUK STOSOWANYCH W TARNOWIE

Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka Stosowana

2022/2023

Maciej Szostak

PRACA INŻYNIERSKA

Sterownik robota balansującego

Promotor pracy:

dr inż. Daniel Król

Tarnów 2022

Spis treści

Wprowadzenie

Dzięki szybkiemu rozwojowi technologii produkcji komponentów elektronicznych oraz rozwojowi gałęzi programowania mikroprocesorów, z roku na rok, coraz popularniejsze stały się różnego rodzaju urządzenia odpowiedzialne za szereg różnorakich czynności, jednak wyróżniających się niestandardową budową. Na rynku coraz częściej pojawiają się rozwiązania takie jak drony czy roboty jedno/dwukołowe, które dzięki swojej unikalnej budowie często mogę wykonywać zadania, jakie dla innych urządzeń mogą być niedostępne. Roboty takie znajdują coraz szersze zastosowanie w różnych dziedzinach naszego życia. Wraz ze wzrostem zapotrzebowania na takie produkty oraz z szybkim rozwojem technologii z nimi związanymi, można już mówić o kształtującej się nowej gałęzi przemysłu związanej z niestandardowymi układami sterowania. W praktyce jednak urządzenia te wykorzystują najczęściej proste i popularne metody sterowania, które zapewniają podstawowe funkcje robota, ale nie daję możliwości wykorzystania bardziej skomplikowanych rozwiązań. Ograniczenie to jest czysto teoretyczne i najczęściej wynika z braku czasu i ograniczeń kosztowych w trakcie tworzenia sterownika dla takiego robota, co niestety przekłada się na wykorzystanie najpopularniejszych rozwiązań kosztem wydajności.

Niniejsza praca przedstawia projekt, budowę oraz oprogramowanie przykładowego sterownika, który dzięki uniwersalnemu podejściu będzie mógł być wykorzystany w różnych urządzeniach, zarówno hobbystycznych jak i w rozwiązaniach przemysłowych. Przedstawione i omówione zostaną rozwiązania sprzętowe, ich ograniczenia, problematyka związana z działaniem, oraz możliwość i sposoby ich rozbudowy. W kwestii oprogramowania zastosowane zostało nowe podejście, dające więcej możliwości w zakresie sterowania i kontroli nad urządzeniem.

W pierwszym rozdziale przedstawiono założenia teoretyczne związane z układami sterowania dla przykładowego robota dwukołowego, a także opisano rolę, jaką będzie pełnił sterownik oraz szereg zadań, za które będzie odpowiedzialny.

W drugim rozdziale omówiona została część sprzętowa oraz oprogramowanie wykorzystane do stworzenia regulatora. W tym rozdziale przedstawione są kryteria wyboru

płytki prototypowej oraz narzędzi programistycznych, a także wykorzystane technologie i rozwiązania.

W trzecim rozdziale omówiona została budowa konkretnego modelu pojazdu dwukołowego. W osobnych rozdziałach szczegółowo omówione zostały elementy budowy robota, a także samego sterownika oraz wykorzystane przy ich projektowaniu rozwiązania projektowe. W każdym podrozdziale kolejno dokładnie opisany jest wybrany element, jego sposób działania, możliwości, a także rola w całym systemie oraz sposób rozbudowy.

W rozdziale czwartym szczegółowo omówiona została cała część programowa tworzonego systemu, z dokładniejszym wyróżnieniem poszczególnych ważniejszych aspektów oprogramowania.

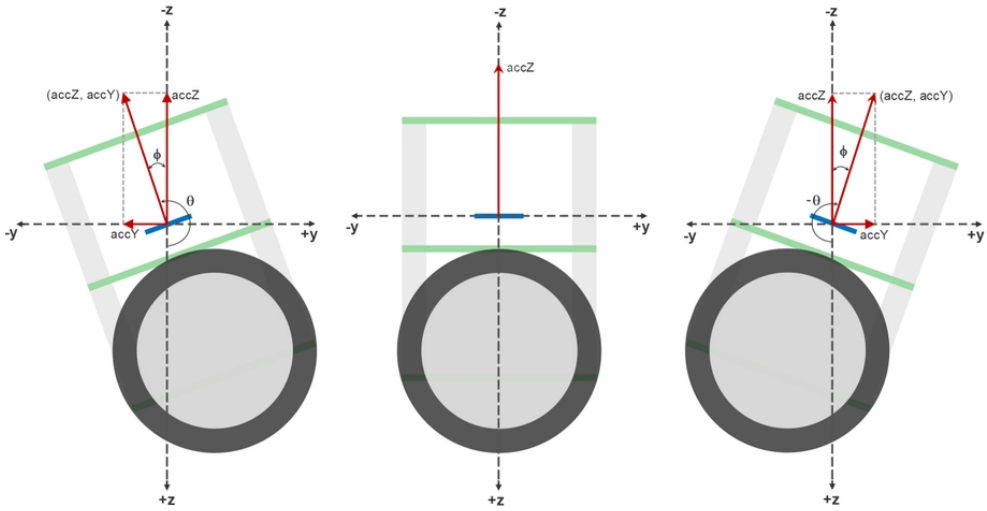
W piątym rozdziale zostały omówione i przedstawione na wykresie uzyskane efekty pracy regulatora w praktyce. Pokrótce opisane zostały podstawowe warunki, jakie powinien spełnić regulator oraz uzyskane rzeczywiste wyniki wraz z omówieniem.

1. Przedstawienie problemu

Głównym założeniem realizowanego projektu było stworzenie układu regulatora opartego o mikrokontroler wraz z odpowiednimi modułami, którego zadanie polegało na dążeniu i utrzymaniu stabilnej pozycji pionowej modelu robota dwukołowego. System ma być odporny na zakłócenia otoczenia oraz samodzielnie regulować swoje położenie.

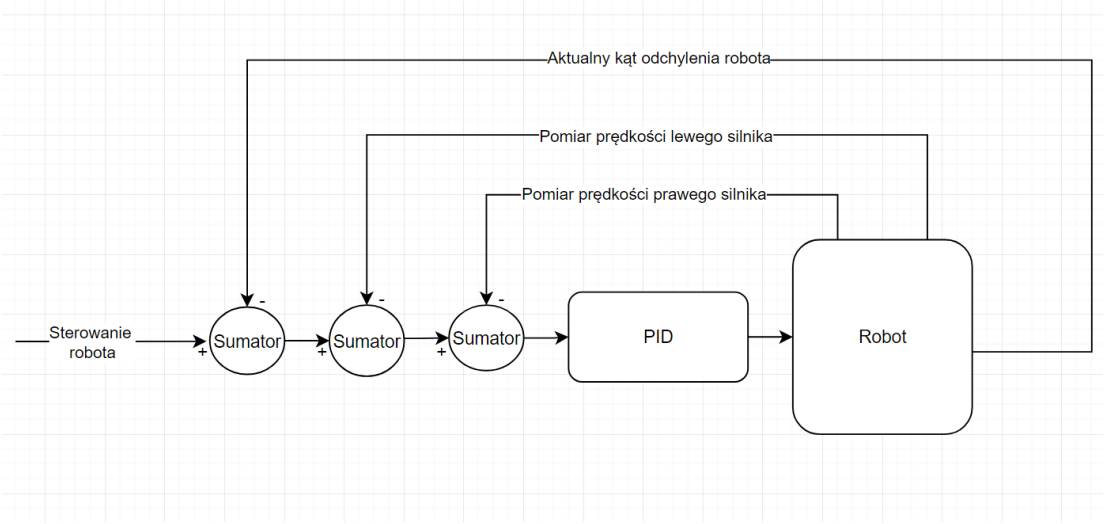
1.1 Metoda stabilizacji robota dwukołowego w pozycji pionowej

Podstawowym problemem sterowania robota dwukołowego jest stabilizacja pozycji pionowej, którą uzyskuje się dzięki poruszaniu robotem odpowiednio do tyłu i do przodu z właściwą prędkością. Ruch ten, oparty jest o aktualną wartość kąta odchylenia względem pozycji zerowej, której nazwa pochodzi od wartości kąta zero, jaką model osiąga w idealnej pozycji pionowej. W rzeczywistości rzadko kiedy udaje się osiągnąć i utrzymać na dłużej pozycję kąta zerowego, najczęściej robot stale porusza się w niewielkim zakresie w obu kierunkach, dążąc do utrzymania pozycji pionowej z zachowaniem niewielkiego marginesu błędu.



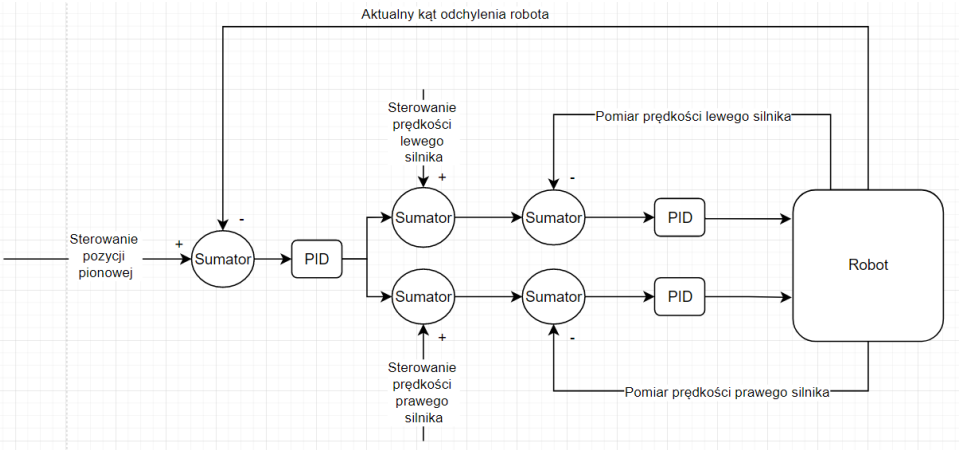
Rysunek 1. Pomiar wartości kąta zerowego względem dwóch osi środka modelu.

Istnieją różne metody sterowania, które mogłyby zostać wykorzystane: korekcja błędów, regulatory *PID* (*ang. Proportional–Integral–Derivative controller*), regulatory rozmyte i inne. W podobnych projektach opisanych w literaturze najczęściej spotykanym rozwiązaniem jest zastosowanie jednego globalnego regulatora *PID* lub nieco rzadziej, dwóch regulatorów *PID* — jednego odpowiedzialnego za regulowanie prędkości oraz drugiego nadrzędnego za utrzymanie pozycji pionowej. Metoda ta jest poprawna, natomiast jest także mocno ograniczana i podatna na błędy. W celu zmiany jednego z elementów systemu sterowania opartego tylko o jeden globalny regulator, konieczna byłaby za każdym razem zmian wszystkich parametrów takiego układu dla nowego systemu.



Rysunek 2. Przykład sterowania z wykorzystanie jednego regulatora PID.

W opracowywanym, w ramach niniejszej pracy regulatorze zdecydowano się na mało popularne jeszcze wykorzystanie 3 różnych regulatorów *PID*. Dwa z nich pracować będą równolegle, po jednym do sterowania każdego silnika, i odpowiadać będą za utrzymanie odpowiedniej prędkości. Natomiast trzeci, nadrzędny nad pozostałymi dwoma, odpowiadać będzie za dążenie i utrzymanie właściwej pozycji.



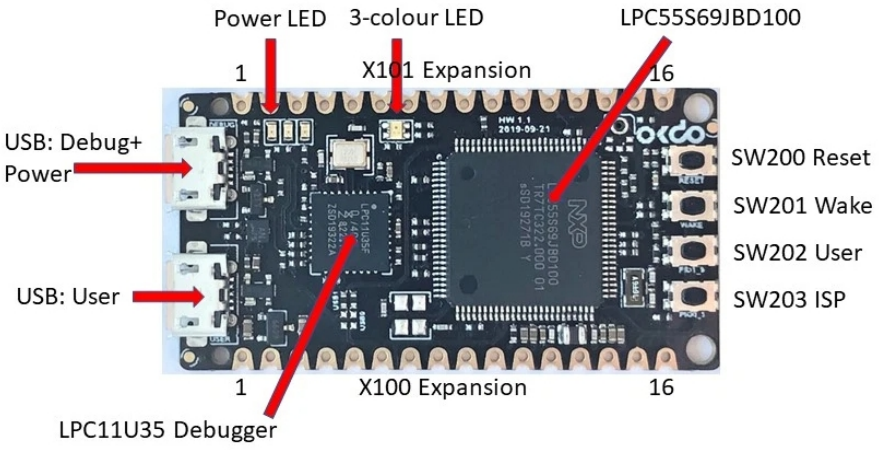
Rysunek 3. Układ sterowania z wykorzystaniem trzech regulatorów PID.

2. Wykorzystane elementy i technologie

Opisywany w niniejszej pracy system sterowania składa się z dwóch części — sprzętowej i programowej. Pierwszy etap projektu polegał na dobraniu odpowiednich modułów oraz płytki rozwojowej, na podstawie której zbudowany został sterownik. Następnie napisany został program, który w trakcie testów został dopracowany, aby gotowy sterownik mógł spełniać wszystkie założenia projektowe.

2.1 Dobór komponentów elektronicznych

Najważniejszym zadaniem robota, w trakcie działania, jest utrzymanie pozycji pionowej. W tym celu układ sterowania musiał na podstawie zebranych danych z enkoderów oraz modułu akcelerometru i żyroskopu, wykonać serię obliczeń dla trzech różnych regulatorów *PID*, a następnie na podstawie otrzymanych danych regulować pracę robota. Ponadto w przyszłości układ będzie mógł w czasie rzeczywistym, komunikować się z innym systemem sterowania na przykład aplikacją w telefonie użytkownika. Konstrukcja wyposażona została także w wyświetlacz graficzny pełniący rolą konsoli, na której wyświetlone zostały wszystkie dane w celu weryfikacji i kontroli. Biorąc pod uwagę wymagania, jakie musi spełniać opisywany sterownik oraz mając na uwadze dodatkowo precyzję tych obliczeń, dzięki czemu możliwa będzie stabilna praca robota, zdecydowano się wykorzystać płytkę rozwojową *OKDO E1*. Płytka oparta jest o mikrokontroler *LPC55S69* firmy *NXP*, zawierający 2 rdzenie *ARM Cortex-M33* oraz kilka koprocesorów – w tym *DSP* (*ang. Digital Signal Processor*).



Rysunek 4. Płytka ewaluacyjna OKDO E1.

2.1 Wykorzystane narzędzia

Całe oprogramowanie wykorzystywane w sterowniku zostało zaimplementowane w języku C i powstawało stopniowo. Ze względu na duże ułatwianie w kwestii konfiguracji środowiska dla systemów wbudowanych oraz łatwość integracji warstwy sprzętowej z warstwą programową, zdecydowano się wykorzystać natywne oprogramowanie *MCUXpresso* firmy *NXP*. Narzędzie, poza standardowymi zaletami zintegrowanego środowiska programistycznego, wyposażone zostało w graficzny edytor do konfiguracji funkcji wyprowadzeń, ustawień częstotliwości zegarów czy wykorzystywanych interfejsów. Ponadto możliwa jest komunikacja za pomocą konsoli z wykorzystaniem interfejsu *UART* (*ang. Universal Asynchronous Receiver-Transmitter*), a także narzędzia debugowania w czasie rzeczywistym z wykorzystaniem interfejsu *SWD* (*ang. Serial Wire Debug*), co zostało wykorzystane w trakcie testów. Oprogramowanie firmy *NXP* jest w pełni darmowe a ponadto stale rozwijane, na przykład pod względem *SDK* dla różnych układów, dzięki czemu możliwe było uzyskanie pomocy ze strony samego producenta w kwestii rozwoju oprogramowania budowanego sterownika.

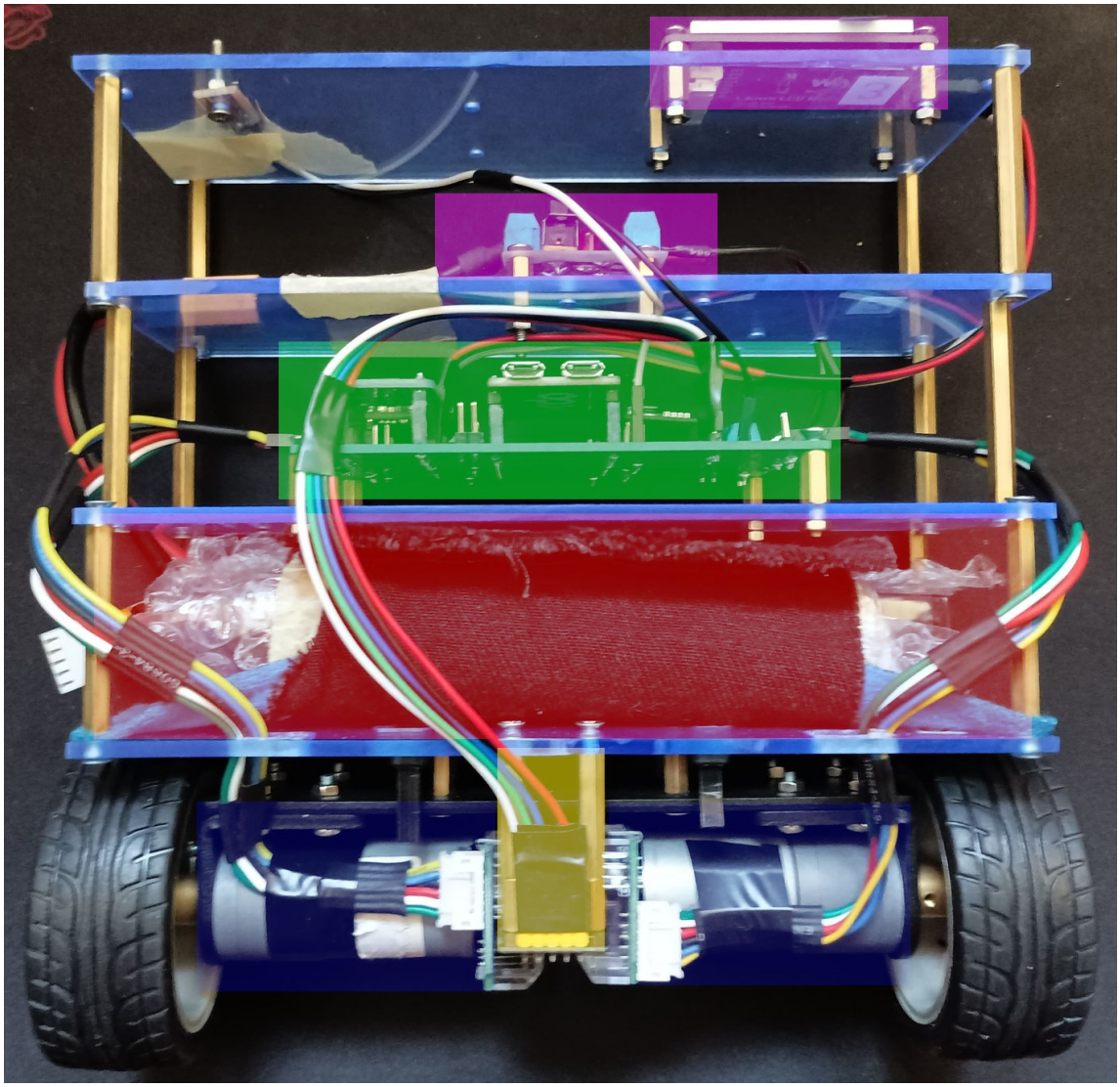
Rysunek 5. Logo środowiska MCUXpresso firmy NXP.

3. Realizacja sprzętowa

Omawiana w tym rozdziale realizacja sprzętowa projektu została podzielona na dwie główne części. W pierwszej, przedstawiona została ogólna budowa przykładowego robota dwukołowego. W drugiej zaś, szczegółowo opisano budowę samego sterownika. Obydwa fragmenty zostały dodatkowo podzielone na podrozdziały w celu uporządkowania wszystkich informacji.

3.1 Budowa przykładowego robota dwukołowego

Cała konstrukcja jest wzorowana na modelu odwróconego wahadła, szerzej opisywanego na przykład w publikacjach matematycznych. W celu podzielenia projektu robota na poszczególne mniejsze warstwy został on zbudowany w formie kilkupoziomowej drabinki. Wykorzystany w niniejszej pracy model robota dwukołowego oparty został o dwa silniki prądu stałego, zasilane z akumulatora i zamontowane do czteroczęściowej konstrukcji wykonanej ze szkła akrylowego. Bateria *Li-Pol* (*ang. Lithium Polymer Battery*), została umieszczona na poziomie zerowym, to znaczy jak najbliżej silników w celu obniżenia i wyrównania środka ciężkości całej konstrukcji. Na kolejny poziomie (pierwszym) został zamocowany sterownik robota, to znaczy specjalnie zaprojektowana płytka *PCB* (*ang. Printed Circuit Board*), na której znajduje się sekcja zasilania, płytka ewaluacyjna oraz złącza łączące: silnik, baterię, moduł żyroskopu, ekran *LCD* (*ang. Liquid-Crystal Display*) i inne. Na kolejnym poziomie (drugim) w celu łatwego dostępu i oddzielnie od modułu sterownika, został umieszczony przełącznik do wyłączenia lub włączenia robota. Na samej górze, ostatnim poziomie (trzecim) umieszczony został ekran *LCD*, na którym wyświetlane są różne parametry w trakcie działa robota.



Rysunek 6. Gotowy robot z wyróżnionymi elementami.

Na rysunku szóstym przedstawiony został przykładowy model robota dwukołowego z wyróżnionymi kolorami elementami budowy:

- czerwonym zaznaczona została bateria *Li-Pol*

- niebieskim silnik DC, wraz z enkoderami optycznymi

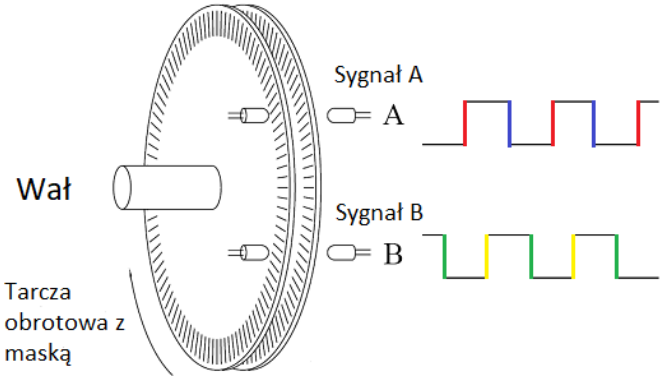
- żółtym moduł *MPU6050* (akcelerometru i żyroskopu)

- zielonym opracowywana w ramach tej pracy układ sterowania

- fioletowym pozostałe elementy (ekran *LCD*, przełącznik on/off)

3.1.2 Silniki DC

Do budowy robota użyte zostały dwa silniki z metalowymi przekładniami w stosunku (1:34), optycznymi enkoderami kwadraturowymi zamontowanym na podwójnym wale o rozdzielczości 360 impulsów na obrót. Napięcie zasilania wynosi (12 V), pobór prądu bez obciążenia (ok. 130 mA) a maksymalny (ok. 3,5A). Średnia prędkość bez obciążenia wynosi 320 obr/min a moment obrotowy 8kg/cm (0,79 N/m).



Rysunek 7. Schemat działania enkodera optycznego.

3.1.3 Moduł MPU6050

Moduł żyroskopu i akcelerometru umieszczony został jak najbliżej silników w celu uzyskania jak najlepszej dokładności kąta odchylenia robota. Wykorzystany moduł jest obsługiwany za pomocą magistrali *I2C* (*ang. Inter-Integrated Circuit*), zasilany napięciem 3.3 V oraz posiada dodatkowe wyprowadzenie, którego funkcje można ustawić zależnie od potrzeb. Układ posiada po cztery zestawy ustawień precyzji czujników. Dane z modułu mają 16-bitową dokładność na każdą z trzech osi, osobno dla akcelerometru i żyroskopu oraz dla dodatkowego czujnika temperatury.

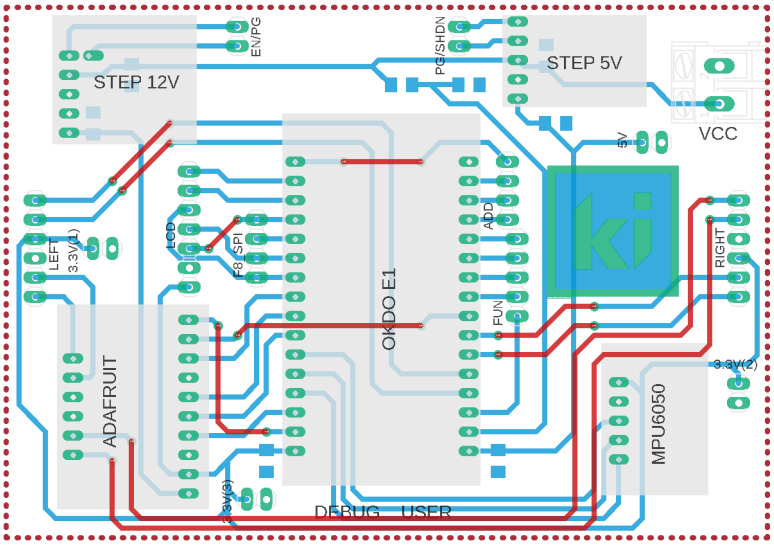
Opcje rozdzielczości akcelerometru i żyroskopu

Full Scale Range	LSB Sensitivity
± 250 °/s	131 LSB/°/s
± 500 °/s	65.5 LSB/°/s
± 1000 °/s	32.8 LSB/°/s
± 2000 °/s	16.4 LSB/°/s

Full Scale Range	LSB Sensitivity
±2*g*	16384 LSB/*g*
±4*g*	8192 LSB/*g*
±8*g*	4096 LSB/*g*
±16*g*	2048 LSB/*g*

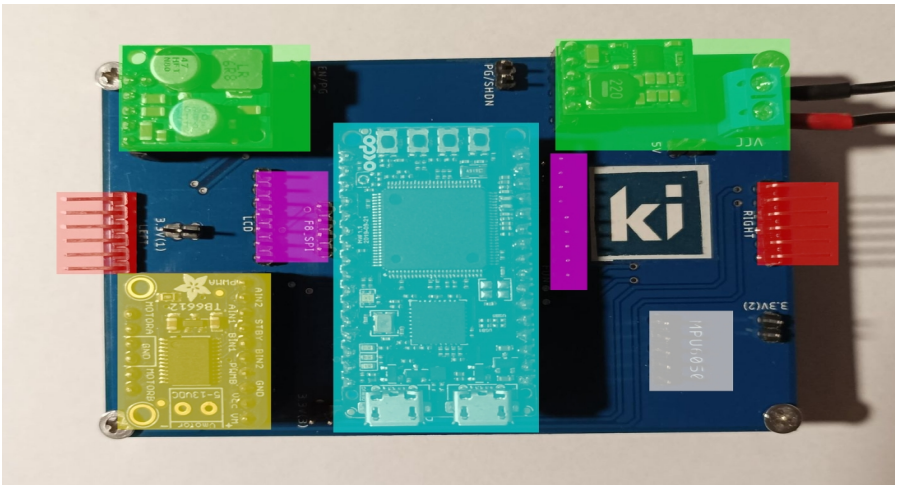
Rysunek 8. Wycinek z dokumentacji modułu MPU6050.

3.2 Projekt płytki PCB



Rysunek 9. Schemat płyty głównej regulatora

W tym podrozdziale szczegółowo opisana została warstwa sprzętowa omawianego w tej pracy sterownika samobalansującego. Głównym elementem systemu regulacji jest zaprojektowana płytka *PCB*, stanowiąca rodzaj płyty głównej, to znaczy łącząca płytkę ewaluacyjną ze wszystkimi modułami oraz posiadająca dodatkowe złącza ułatwiające podłączenie silników, baterii, ekranu *LCD*, i innych elementów. Projekt płytki został stworzony na podstawie opisanego wcześniej przykładowego robota dwukołowego, jednak sam sterownik jest uniwersalny i nadaję się do wykorzystania w innych projektach.

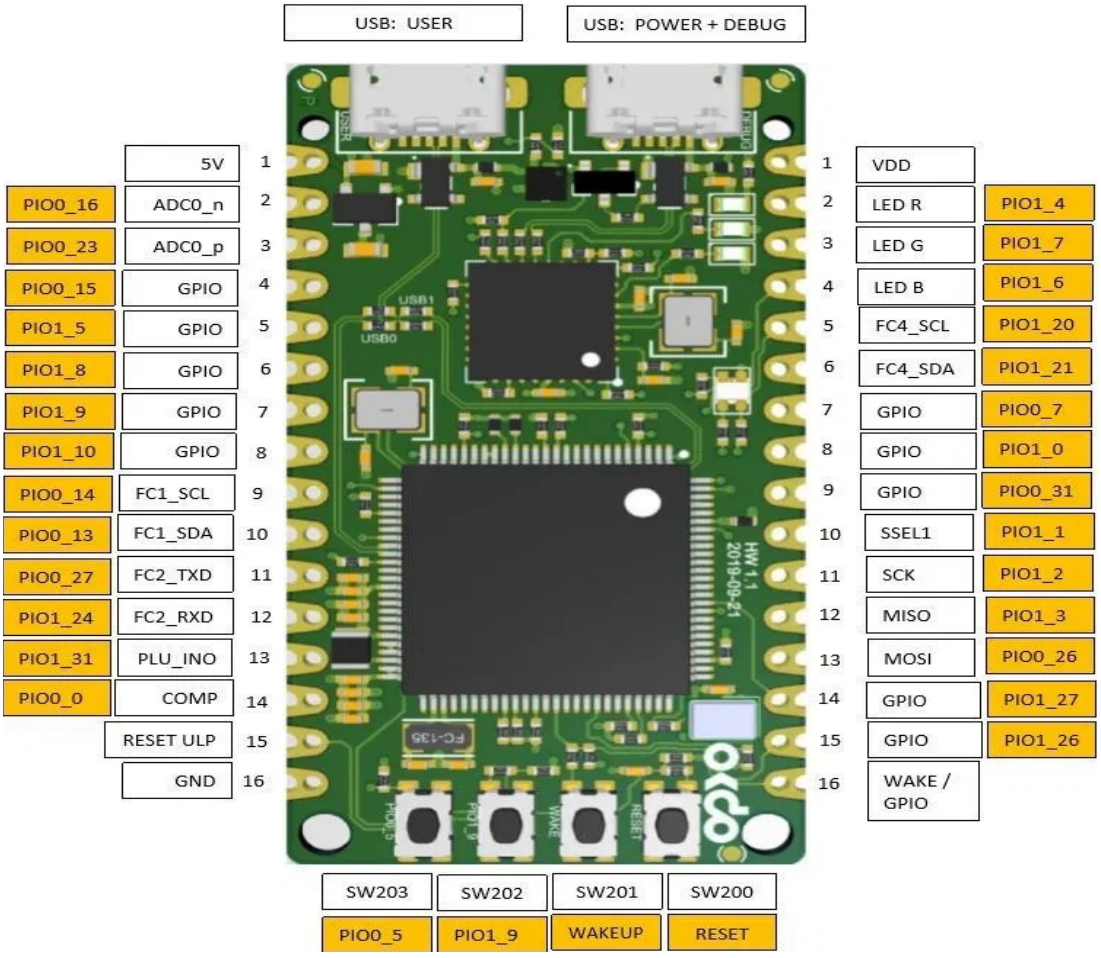


Rysunek 10. Płytka pcb z wyróżnionymi elementami.

Na zdjęciu kolorami zostały odznaczone ważniejsze elementy:

- niebieskim, płytka rozszerzeniowa *OKDO E1*
- zielonym, sekcja zasilania
- żółty, mostek H
- biały, złącze dla modułu *MPU6050*
- czerwony, złącza dla silników (prawe i lewe)
- fioletowym złącze dla ekranu *LCD* i inne

3.2.1 Płytka rozwojowa OKDO E1



Rysunek 11. Poglądowy schemat płytki ewaluacyjnej OKDO E1.

Głównym elementem regulatora jest płytka ewaluacyjna *OKDO E1* oparta na mikrokontrolerze *LPC55S69* zawierającym 2 rdzenie *ARM Cortex-M33* firmy *NXP*. Wybrane cechy specyfikacji tej płytki rozszerzeniowej zostały wcześniej omówione w rozdziale dotyczącym kryteriów doboru komponentów elektronicznych. W tej części opisane zostaną kolejno konkretne funkcje wszystkich pinów, wykorzystane w regulatorze samobalansującym oraz piny, które pozostają wolne do wykorzystania przy rozbudowie projektu.

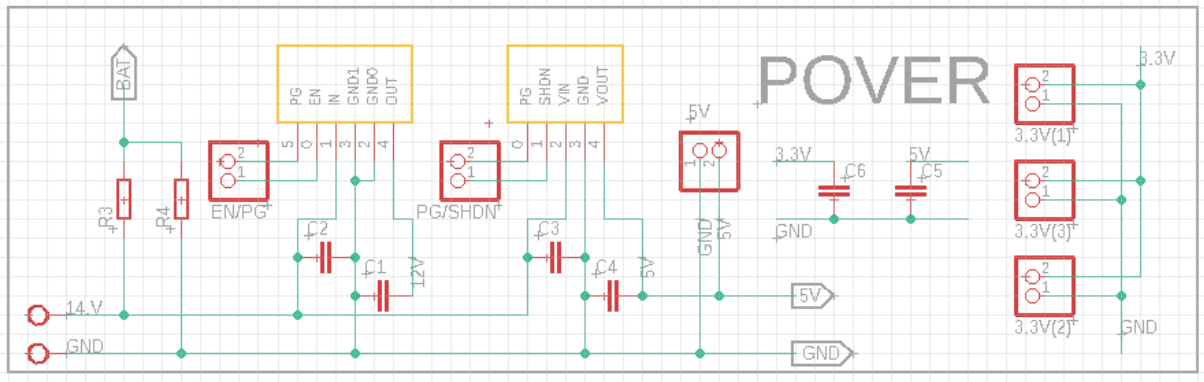
Tabela 1. Zestawienie pinów płytki OKDO E1 i ich funkcja dla regulatora samo balansującego

Oznaczenie pinów	Funkcje
5V, VDD (3.3V)	Zasilanie płytki i regulatora przez stabilizator liniowy do 3.3V
GND	Masa (Wspólna dla całego regulatora)
Reset	Wyprowadzony pin od resetu
WAKE	Wyprowadzony pin wybudzenia
PIO0_16 (ADC0_N)	Pomiar napięcia baterii przez wbudowany przetwornik ADC
PIO1_4 (LED_R), PIO1_7 (LED_G), PIO1_10, PIO0_7, PIO 1_0, PIO0_31	Podłączenie mostka-H
PIO1_6 (LED_B), PIO1_20 (FC4_SCL), PIO1_21 (FC4_SDA)	Podłączenie modułu MPU6050
PIO0_15, PIO1_5, PIO1_8, PIO1_9	Podłączenie enkoderów
PIO1_1 (SSEL1), PIO1_2 (SCK), PIO1_3 (MISO), PIO0_26 (MOSI), PIO1_27, PIO1_26	Podłączenie ekranu LCD
PIO0_23 (ADC0_P), PIO0_14 (FC1_SCL), PIO0_13 (FC1_SDA), PIO0_27 (FC2_TXD), PIO1_24 (FC2_RXD), PIO1_31, PIO0_0	Piny wolny

W tabeli skrótowo przedstawione zostały funkcje wszystkich pinów, które szczegółowo zostaną jeszcze omówione w następnym rozdziale dotyczącym części programowej. Dokładniejszy opis poszczególnych funkcji pinów wraz z wyjaśnieniem można znaleźć w dokumentacji mikrokontolera *LPC55s69*.

3.2.2 Sekcja zasilania

W tym podrozdziale pokrótce omówiona została cała sekcja zasilania, która została umieszczona na płytce *PCB* oraz elementy, które wchodzą w jej skład. Głównym zadaniem tej sekcji, dla tworzonego regulatora samobalansującego, jest obniżanie napięcia z akumulatora *Li-Po* do odpowiedniego poziomu dla każdego z komponentów.



Rysunek 12. Schemat połączeń sekcji zasilania regulatora.

Na rysunku 12, przedstawiony został schemat połączeń sekcji zasilania na płytce *PCB*. Akumulator *Li-Po* podłączony jest do regulatora złączem *XT30* poprzez włącznik on/off umieszczony poza płytką drukowaną. W trakcie pracy regulatora bateria dostarcza 14.8 V lub niższym w zależności od stopnia naładowania pakietu. Stopień naładowania lub rozładowanie baterii jest obniżony za pomocą dzielnika napięć złożonego z rezystorów (po lewej stronie na schemacie), aby mógł być zmierzony za pomocą 16-bitowego przetwornika analogowo-cyfrowego i wyświetlony na wyświetlaczu.

$$ADC\ Input\,(Max\,3.3\,V) = Input\ Power\,(Max\,14.8\,V) * \frac{R2(220\,\Omega)}{R1(1\,K\,\Omega) + R2(220\,\Omega)} \approx 2{,}66$$

Kolejnymi elementami układu zasilania są dwie przetwornice step-down firmy *Pololu*. Pierwszy moduł *D36V28F12* obniżający napięcie do 12 V o wydajności do 2.4 A, dostosowuje zasilanie dla silników, drugi *D24V10F5* obniża napięcie do 5 V o wydajności 1 A. Napięcie 5 V jest wykorzystywane po obniżeniu do 3.3 V przez stabilizator liniowy do zasilania płytki rozszerzeniowej *OKDO E1* oraz pozostałych komponentów np. mostka-H, wyświetlacza *LCD* itd. Całość sekcji zasilnia została także rozbudowana o kondensatory filtrujące na wejściu i wyjściu poszczególnych modułów o jednakowej uproszczonej wartość 100 nF.

4. Realizacja programowa

W tej części pracy został omówiony ogólnie cały sposób działania programu oraz dokładniej jego wybrane ważniejsze elementy. Całość programu została napisana w języku C z wykorzystanie narzędzi będących rozszerzeniem dla środowiska *MCUXpresso* do konfiguracji elementów takich jak interfejsy, zegary czy przerwania dla płytek rozszerzeniowych opartych na mikrokontrolerze *LPC55sS69*.

4.1 Struktura modułów

Listing 1. Szablony struktur do organizacji zmiennych.

```
typedef struct{
            int32_t position;
            float32_t speed;
            bool status;
            float32_t setSpeed;
            float32_t calcPWM;
            bool front;
            bool back;
            float32_t pwmDuty;
            void (*set_ptr)(bool, bool, uint8_t);
            float32_t error;
            arm_pid_instance_f32 coef;
}Engine_t;

typedef struct{
            MPU6050_t config;
            float32_t angleX, angleY, angleZ, temp, angle_error,
calcSpeed;
            arm_pid_instance_f32 coef;
}mpu6050_t;
```

Ważnym elementem tworzonego oprogramowania są przedstawione powyżej struktury dla silników oraz modułu akcelerometru i żyroskopu. Podstawowym ich zadaniem jest organizacja zmiennych, wskaźników, flag stanu, i innych pomniejszych struktur, które następnie będą wykorzystywane w całym programie.

Tabela 1. Zestawienie nazw zmiennych i ich funkcji w programie dla struktury Engine_t.

`int32_t position;`	Zmienna zliczająca liczbę impulsów z enkodera
`float32_t speed;`	Zmienna przechowująca obliczoną prędkość
`bool status;`	Flaga on/off dla silnika
`float32_t setSpeed;`	Zmienna przechowująca zadaną prędkość dla silnika
`float32_t calcPWM;`	Obliczona wartość sygnału PWM z regulatora PIDa
`bool front; bool back;`	Flagi określające kierunek jazdy
`float32_t pwmDuty;`	Wartość sygnału PWM dla mostka-H
`void (*set_ptr)(bool, bool, uint8_t);`	Wskaźnika na funkcję sterowania pinami mostka -H.
`float32_t error;`	Wartość błędu obliczoną dla sterownika PID
`arm_pid_instance_f32 coef;`	Wskaźnika na strukturę parametrów Kp, Ki, Kd dla regulatora PID

Tabela 2. Zestawienie nazw zmiennych i ich funkcji w programie dla struktury mpu6050_t.

`MPU6050_t config;`	Struktura dla interfejsu I2C modułu MPU6050
`float32_t  angleX , angleY , angleZ`	Obliczona wartość kąta X, Y, Z
`float32_t temp`	Obliczona wartość temperatury
`float32_t angle_error`	Wartość błędu kąta X dla regulatora PID
`float32_t calcSpeed`	Obliczona przez regulator wartość prędkości
`arm_pid_instance_f32 coef;`	Wskaźnika na strukturę parametrów Kp, Ki, Kd dla regulatora PID

W tabeli pierwszej przedstawiono szczegóły funkcji, jakie pełnią elementy szablonu struktury, który następnie został wykorzystany do stworzenia obiektu struktury organizujący zmienne i inne elementy dla każdego z silników. W podobny sposób funkcjonuje opisany w drugiej tabeli drugi szablon, struktury tym razem wykorzystanej do organizacji zmiennych i pozostałych składowych dla modułu akcelerometru i żyroskopu. Wszystkie pola każdego z obiektów są zainicjowane zerami w momencie deklarowania.

4.2 Przerwanie główne

Listing 2. Przerwanie główne programu.

```
void CB_PWM(uint32_t flags){
	if (plotDataIndex < dataSize){
		dataBuffer[plotDataIndex] = mpu6050.angleX;
		plotDataIndex++;
	}
	engineAngleUpdate();

	engineCalculationUpdate(&engineLeft);
	engineCalculationUpdate(&engineRight);
	enginePowerUpdate(&engineLeft);
	enginePowerUpdate(&engineRight);
}
```

Działanie programu sterownika zostało oparte o funkcje przerwań, z wykorzystaniem priorytetowego wektora przerwań *NVIC* (*ang. Nested Vectored Interrupt Controller*). Główne przerwanie o najwyższym priorytecie (zerowym) jest generowane za pomocą zegara *CTIME2* działającego z częstotliwością 100 KHz przy podzieleniu do 1 KHz dla funkcji *PWM* (*ang. Pulse-Width Modulation*). W skład funkcji głównego przerwania wchodzą w sumie cztery elementy odpowiedzialne za główne zadanie regulatora, jakim jest utrzymanie robota w pozycji pionowej — każde z nich kolejno zostaną omówione w podrozdziałach do tego rozdziału.

4.2.1 Obliczanie prędkość silników

Listing 3. Funkcja obliczająca RPM silnika.

```
#define ImpulsePerRotation (90.f * 34.f)
#define IntervalToSecond (1000.f * 60.f)

void engineCalculationUpdate(Engine_t * engine){
	engine->speed = (engine->position / ImpulsePerRotation) *
IntervalToSecond;
	engine->position = 0.0;
}
```

Funkcja *void engineCalculationUpdate(Engine_t * engine)* oblicza aktualną prędkość danego silnika w *RPM* (*rotation per minute*). W przypadku enkodera optycznego kwadraturowego o rozdzielczości 360 impulsów na obrót wykorzystujemy tylko 1/4 impulsów, ponadto w silnikach zamontowane są przekładnie w stosunku 34:1, a więc w trakcie jednego obrotu otrzymujmy 34 * 90 impulsów. Przerwanie główne wywoływane jest z częstotliwością 1 KHz, czyli co 1 ms. Na podstawie tych informacji jesteśmy w stanie po kilku przekształceniach, wyprowadzić gotowy wzór do obliczenia *RPM* naszego silnika.

ΔP = zmiana pozycji wału silnika, wyrażona w ilość impulsów z enkodera w trackie 1ms

$$Ilość\ obrotów\ w\ czasie\ 1\ ms = \frac{\Delta P}{(90 \cdot 34)}$$

$$Ilość\ obrotów\ w\ czasie\ 1000\ ms[1\ s] = 1000 \cdot \frac{(\Delta P)}{(90 \cdot 34)}$$

$$Ilość\ obrotów\ w\ czasie\ 60\ s[1\ min] = 60 \cdot 1000 \cdot \frac{\Delta P}{(90 \cdot 34)}$$

$$RPM = \frac{\Delta P}{(impulsePerRotation)} \cdot (intervalToSecond) = \frac{\Delta P}{(90 \cdot 34)} \cdot (60 \cdot 1000)$$

4.2.2 Regulator PID stabilizujący

Listing 4. Funkcja regulator PID dla kąta .

```
#define Zero_angle 0.0
void engineAngleUpdate()
{
            mpu6050.angle_error = Zero_angle - mpu6050.angleX;
            mpu6050.calcSpeed = arm_pid_f32(&mpu6050.coef,
mpu6050.angle_error);
            engineLeft.setSpeed =   mpu6050.calcSpeed;
            engineRight.setSpeed = mpu6050.calcSpeed;
}
```

Funkcja oblicza wartość błędu kąta, czyli różnicy aktualnego kąta obliczonego za pomocą modułu akcelerometru i żyroskopu a kąta zerowego (pozycji pionowej). Następnie za pomocą regulatora PID określany jest kierunek oraz szybkość, z jaką powinien się poruszać robot w celu przeciwdziałania odchyleniu i utrzymaniu idealnej pozycji pionowej. Przybliżone wartości Kp = 35, Ki = 1, Kd = 0.03 zostały dobrane ręcznie na podstawie wykresu danych z regulatora i kąta z modułu *MPU6050*.

Wykorzystana w programie funkcja regulatora *arm_pid_f32 (arm_pid_instance_f32 *S, float32_t in)* pochodzi z biblioteki *arm_math.h* która jest częścią pakietu *CMSIS-DSP* zgodnego z płytką rozwojową *OKDO E1* i dostarcza wszystkie potrzebne elementy do pełnego zaimplementowania funkcjonującego regulatora *PID* z dokładnością *float32_t*.

4.2.3 Regulator PID sterujący prędkości

Listing 5. Funkcja regulator PID dla silników

```
void enginePowerUpdate(Engine_t * engine)
{
              engine->error = (engine->setSpeed  - engine->speed);
              engine->calcPWM = arm_pid_f32(&engine->coef, engine->error);
              engine->pwmDuty = (engine->status ? engine->calcPWM : 0.0);


              engine->pwmDuty =((engine->pwmDuty > 100) ? (100) : (engine-
>pwmDuty ));
              engine->pwmDuty =((engine->pwmDuty < -100) ? (-100) :
(engine->pwmDuty ));
              engine->front = (engine->pwmDuty > 0);
              engine->back = (engine->pwmDuty < 0);


              engine->set_ptr(engine->front, engine->back, abs(engine-
>pwmDuty));
}
```

W kolejny etapie przerwania głównego wykonywana jest funkcja odpowiedzialna za wysterowanie silnikami. Przede wszystkim jest obliczana różnica w aktualnej prędkości a prędkości koniecznej do utrzymania pozycji pionowej wyliczonej przez regulator *PID* stabilizujący. Błąd prędkości wykorzystany jest następnie przez regulator *PID* sterujacy prędkością silników, do wyliczenia wartości sygnału *PWM* dla mostka-H. W tym momencie sprawdzana jest takaż flaga stanu silnika, określając czy silnik w ogóle powinien być włączony lub wyłączony. W kolejnej części funkcji obliczony wcześniej przez regulator *PID* sygnał *PWM* jest najpierw organiczny do zakresu minimum -100 i maksimum 100, a następnie na jego podstawie ustawiane są (opisane wcześniej) flagi określające kierunek, w którym ma poruszać się robot. Wartość absolutna sygnału *PWM* z zakresu <0,100> na koniec jest podawana za pomocą odpowiedniej funkcji do mostka- H. Wskaźnik do właściwej funkcji znajduje się w odpowiednim obiekcie struktury wykorzystanym do reprezentacji każdego z silników.

Tabela 3. Zestawianie wartości PID regulatorów silników .

Silnik lewy	Kp: 0.8, Ki: 0.08, Kd: 0.001
Silnik prawy	Kp: 0.8, Ki: 0.75, Kd: 0.002

W tabeli powyżej przedstawione są parametry regulatorów *PID* wykorzystanych do regulowania prędkości dla obydwu silników. Wartości te są do siebie zbliżone, ponieważ do budowy robota wykorzystano dwa identyczne silniki a delikatne różnice w ich działaniu pokrywa margines błędu producenta.

4.3 Obsługa modułu MPU6050

Ważnym elementem projektu jest moduł akcelerometru i żyroskopu wykorzystywany do pomiaru aktualnego kąta robota. Czujnik działa z częstotliwością 1 KHz oraz z dokładnością ± 16G dla akcelerometru i ± 2000º/sec dla żyroskopu, co daje w przeliczeniu rozdzielczość:

2 Bajty (65,536) / 16 <-16G, +16G> = 2048

danych pomiarowych czułości na przeciążenie 1g ,

oraz

2 bajty (65,536) / 4000 <-2000º/sec, +2000º/sec> = 16.384 ≈ 16.4

danych pomiarowych na jeden stopień kąta,

podane wartości rozdzielczość dotyczą wszystkich z osi x, y, z .

Ponadto dane filtrowane są przez *DLPF* (*ang. Digital Low Pass Filter*), wbudowany wewnętrzny filtr dolnoprzepustowy z najdłuższym czasem filtrowania, a następnie generowany jest sygnał przerwania na pinie *INT* i gotowe dane mogą zostać odczytane w funkcji przerwania.

Listing 6. Funkcja obsługująca przerwanie modułu MPU6050.

```
void CB_ENPU6050(pint_pin_int_t pintr ,uint32_t pmatch_status){
      MPU6050_readRawAll(&mpu6050.config, mpuBuffer);
      MPU6050_correctAll(&mpu6050.config, mpuBuffer);
      MPU6050_complementaryFilter(&mpuBuffer[0], &mpuBuffer[4],
&mpu6050.angleX);  }
```

Przerwanie obsługiwane jest przez osobną funkcją z priorytetem jeden (niższym od przerwania głównego), gdzie po odczytaniu zachodzi ich korekcja zgodnie z dokumentacją modułu. Następnie wartości osi filtrowane są z wykorzystaniem filtra uzupełniającego, gdzie przeliczane są na odpowiednie wartości kątów. Jako argumenty wejściowe dla funkcji podajemy, osobno bufory danych dla osi x, y, z akcelerometru i żyroskopu oraz bufor na gotowe przefiltrowane dane.

W funkcji filtrującej, w pierwsze kolejności obliczmy wartości kątów x, y, z wykorzystując gotowe wzory do obliczenia *roll, pitch, yaw* w radianach, a następnie zamieniamy je na stopnie.

$$axel_x = \arctan\left(\frac{x}{\sqrt{(y^2+z^2)}}\right),\ axel_y = \arctan\left(\frac{y}{\sqrt{(x^2+z^2)}}\right),\ axel_z = \arctan\left(\frac{\sqrt{(x^2+y^2)}}{(z)}\right)$$

W dalszej części funkcji obliczmy także wartości prędkości kątowej ze wzoru:

$$[gyro_x, gyro_y, gyro_z] = angleValue[0..2] + gyro[0..2]\cdot\Delta t$$

gdzie:

angleValue[0...2] - *wartości poprzedniego kąta,*

gyro[0...2] *- nowe wartości zmierzone żyroskopem,*

Δt *- delta czasu (czas między kolejnym pomiarami w ms).*

Na koniec za pomocą równania filtra aktualne dane kąta i prędkości kątowej są łączone i zapisane do bufora danych wyjściowych oraz bufora roboczego wykorzystanego przy obliczeniu prędkości kątowej.

$$angleValue[0] = \alpha\cdot gyro_x + \beta\cdot axel_x$$
$$angleValue[1] = \alpha\cdot gyro_y + \beta\cdot axel_y$$
$$angleValue[2] = \alpha\cdot gyro_z + \beta\cdot axel_z$$

$$\alpha = \frac{t}{(t+dt)} = \frac{1}{(1+0.1)} = \frac{10}{11} \approx 0.(90),\ \beta = (1-alfa) = 1-\frac{10}{11} = \frac{1}{11} \approx 0.(09)$$

gdzie:

Δt - *czas próbkowania 1 KHz = 1000 cykli/s = 1 ms = 0.001 s*

t *- średni czas wystąpienia zakłóceń danych akcelerometru*

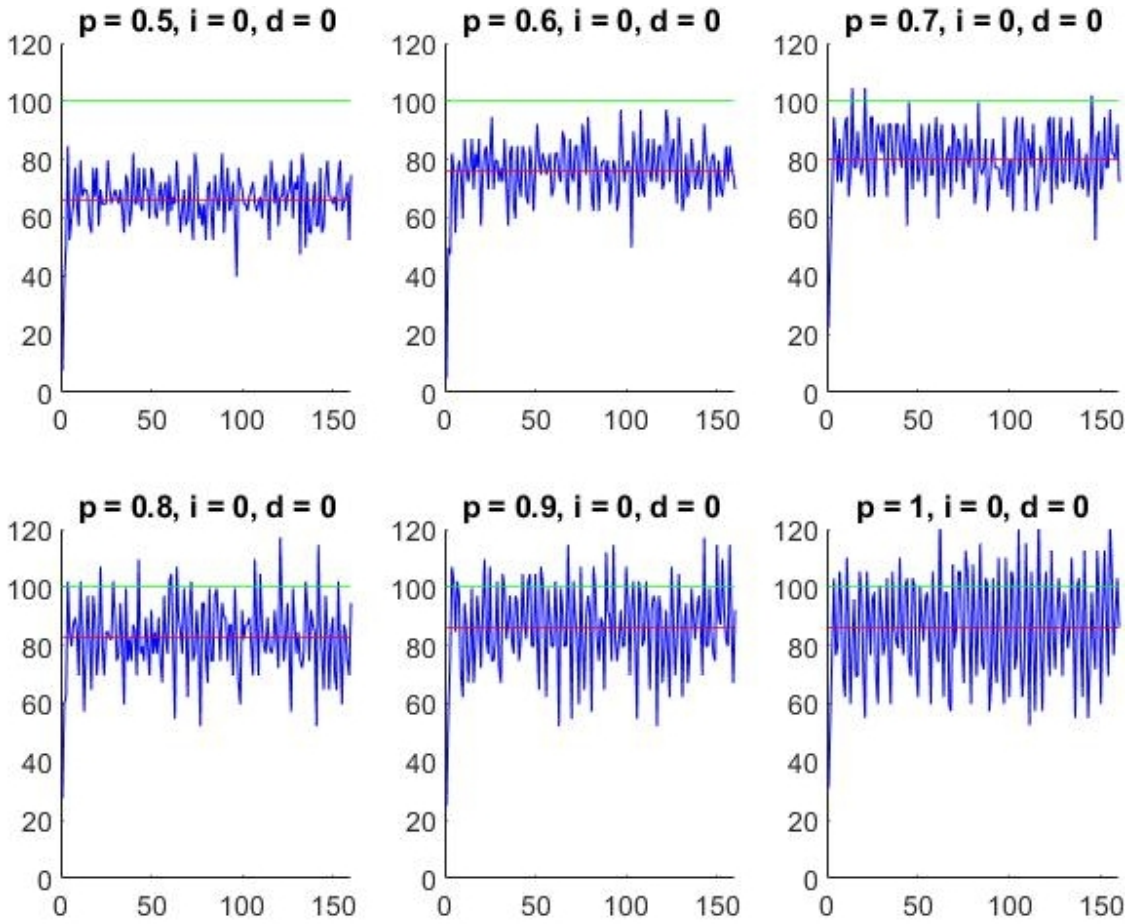
4.4. Strojenie regulatorów PID

W tym podrozdziale omówione zostały metody eksperymentalnego strojenia regulatorów *PID*, na podstawie wykresów odpowiedzi regulatora na zadany sygnał. W celu uporządkowania informacji został on podzielony na dwa podrozdziały. Pierwszy dotyczył będzie strojenia regulatora *PID* sterującego prędkościami silników. W całym systemie wykorzystane są dwa takie regulatory, osobno dla prawego i lewego silnika. Ze względu na wykorzystanie do budowy robota dwóch identycznych silników, w pierwszej kolejności dobrane zostaną ogólne wartości parametrów — *Kp, Ki, Kd*, a następnie indywidualnie skorygowane. Drugi podrozdział natomiast dotyczył będzie doboru parametrów dla regulatora sterującego utrzymaniem pozycji pionowej robota.

4.4.1 Strojenie regulatora PID sterującego prędkością

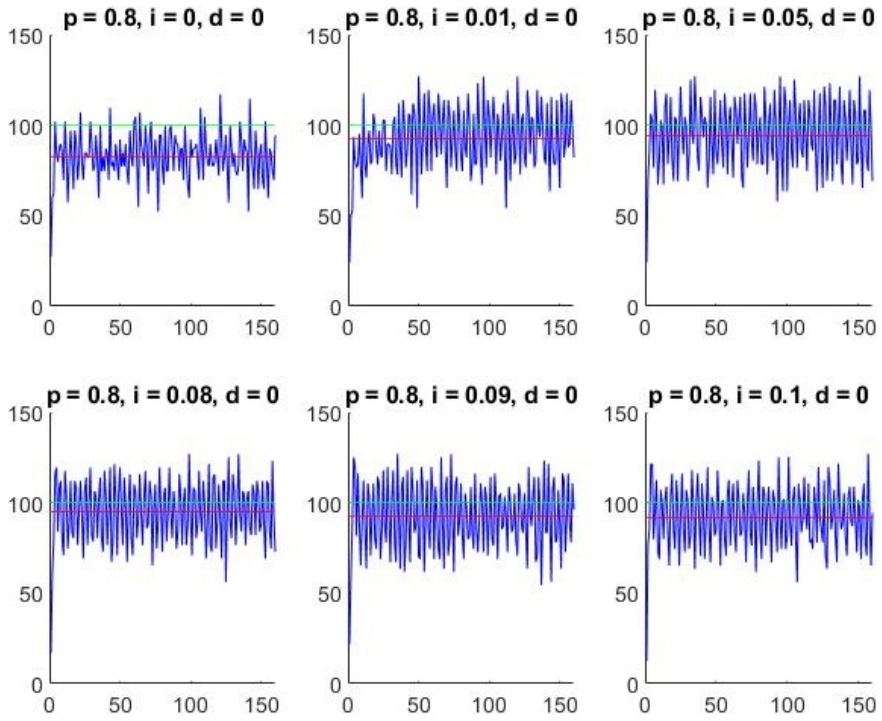
Obrana bezpieczna metoda strojenia parametrów *PID* oparta została, o dane chwilowej prędkości obliczanej z enkoderów oraz wykres przebiegu zmiany tych wartości w czasie wyświetlanym na ekranie *LCD*. Następnie wprowadzane były zmiany parametrów w kolejności *Kp, Ki, Kd* do momentu uzyskania jak najlepszej odpowiedzi skokowej. Wykorzystana funkcja kroku polegała na ustawieniu jako prędkość docelowej 1/4 prędkości maksymalnej silników na chwilę przed ich uruchomieniem. Cały proces oraz uzyskane w nim efekty zostały przedstawione i omówione na poniższych obrazkach, zawierających wykresy odpowiedzi skokowej prędkości silnika lewego dla różnych parametrów regulatora. Analogicznie wygląda sytuacja dla silnika prawego, o czym zostało już wspomniane na wstępie do tego rozdziału — ewentualne różnice w działaniu obu silników, które pokrywa margines błędu producenta, zostaną skorygowane pod koniec tego podrozdziału. Dodatkowo na wykresach przedstawione zostały kolorem-czerwonym linia średniej prędkość obliczonej z wartości odpowiedzi skokowej oraz kolorem zielonym wartość docelowa prędkości, jaką powinien uzyskać silnik.

Zgodnie z przyjętą strategią strojenia parametrów w pierwszej kolejności wybrany został parametr Kp, którego wartość początkowa wynosiła Kp = 0.5, natomiast pozostałe parametry ustawione zostały na wartość równą zero. Wpływ zmiany wartości tego parametru na przebieg odpowiedzi skokowej prędkości lewego silnika przedstawiony został na rysunku ósmym.



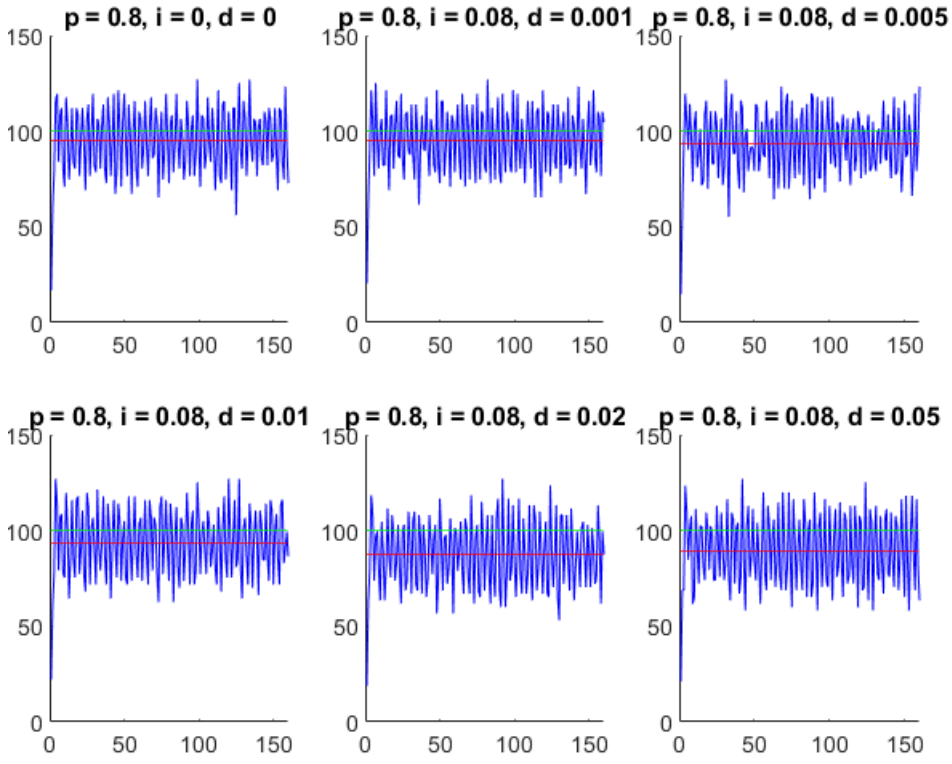
Rysunek 8. Wykresy odpowiedzi skokowej dla różnych wartości parametru P .

Na podstawie tych wykresów jako odpowiednia wartość zmiennej Kp dla regulatora wybrana została wartość Kp = 0.8. W kolejnym etapie strojenia regulatora *PID* rozbudowywany został parametr Ki członu całkującego i analogicznie jak dla poprzedniego parametru, rysunek zawierający wykresy odpowiedzi skokowej przedstawiony został poniżej.



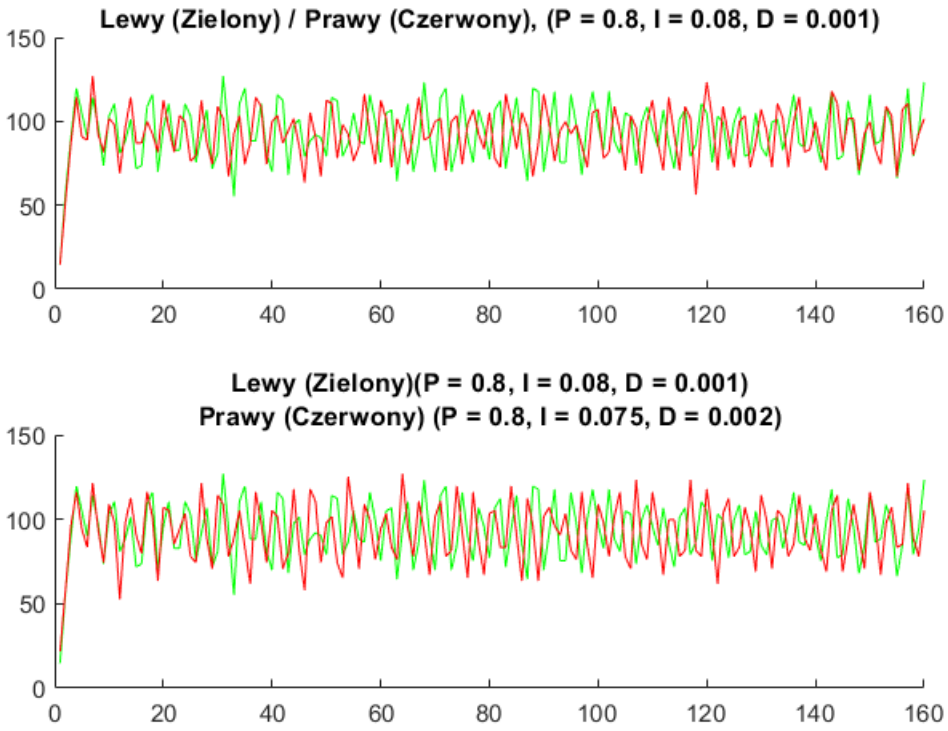
Rysunek 9, Wykresy odpowiedzi skokowej dla różnych wartości parametru I.

Najlepszy wpływ na wykres odpowiedzi skokowej (praca silnika), uzyskujemy dla wartości parametru Ki = 0.08 dla regulatora. W ostatnim etapie strojenia dostosowany został parametr Kd członu różniczkującego.



Rysunek 10. Wykresy odpowiedzi skokowej dla różnych wartości parametru D.

W wyniku opisanych w tym podrozdziale testów wstępne wartości parametrów dla regulatorów *PID* sterujących prędkościami silników zostały ustalone jako: Kp = 0.8 Ki = 0.08 Kd = 0.001. Ostatnim etapem dobierania wartości parametrów, było porównanie wykresu lewego silnika z wykresem prawego silnika i na ich podstawie wprowadzenie zmian wagi parametrów w niewielkim zakresie w celu uzyskania jak najbardziej zbliżonego wyniku dla obu silników. W tym wypadku wystarczyło ręcznie pozmieniać parametry w niewielkim zakresie w celu uzyskania jak najlepszego wyniku, uzyskany efekt został przedawniony na wykresie poniżej.



Rysunek 11. Porównanie wykresów odpowiedzi skokowej dla prawego i lewego silnika.

Na przedstawionym powyżej rysunku kolorem zielonym zaznaczony jest wykres odpowiedzi skokowej regulatora dla silnika lewego, natomiast kolorem czerwonym silnika prawego. Pierwsza część obrazka zawiera wykresy dla parametrów Kp = 0.8, Ki = 0.08, Kd = 0.001 identyczne dla obu regulatorów silników. Widoczna jest delikatna różnica między kolorem zielonym a czerwony (średnia wartość wykresu czerwonego jest znacząco mniejsza od wykresu zielonego). Aby zredukować różnice w odpowiedziach skokowych a co za tym idzie, zwiększyć stabilność pracy silników w projekcie regulatora dla robota — parametry dla silnika prawego zostały delikatnie zmodyfikowane i ustawione jako:

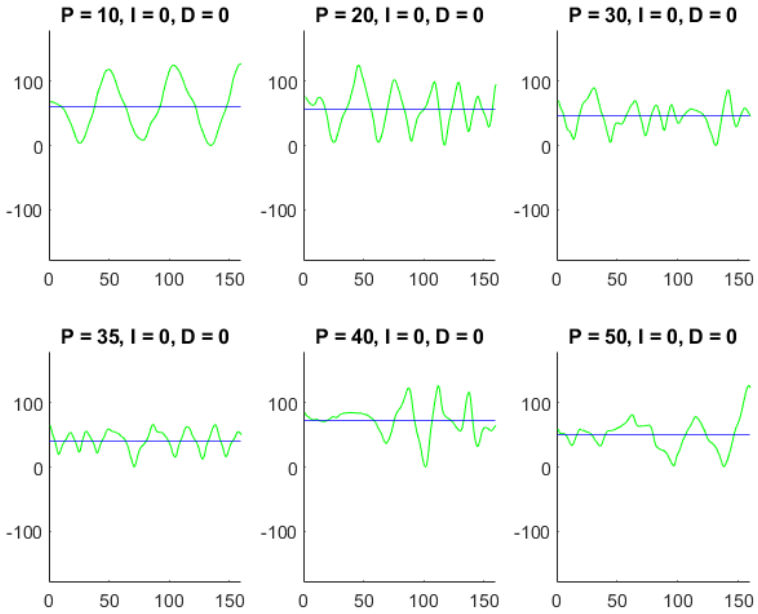
Kp = 0.8, Ki = 0.075, Kd = 0.002

Tabela 4. Zestawienie parametrów regulatorów PID.

Silnik: / Człon:	Lewy	Prawy
Proporcjonalny	0.8	0.8
Całkujący	0.08	0.075
Różniczkujący	0.001	0.002

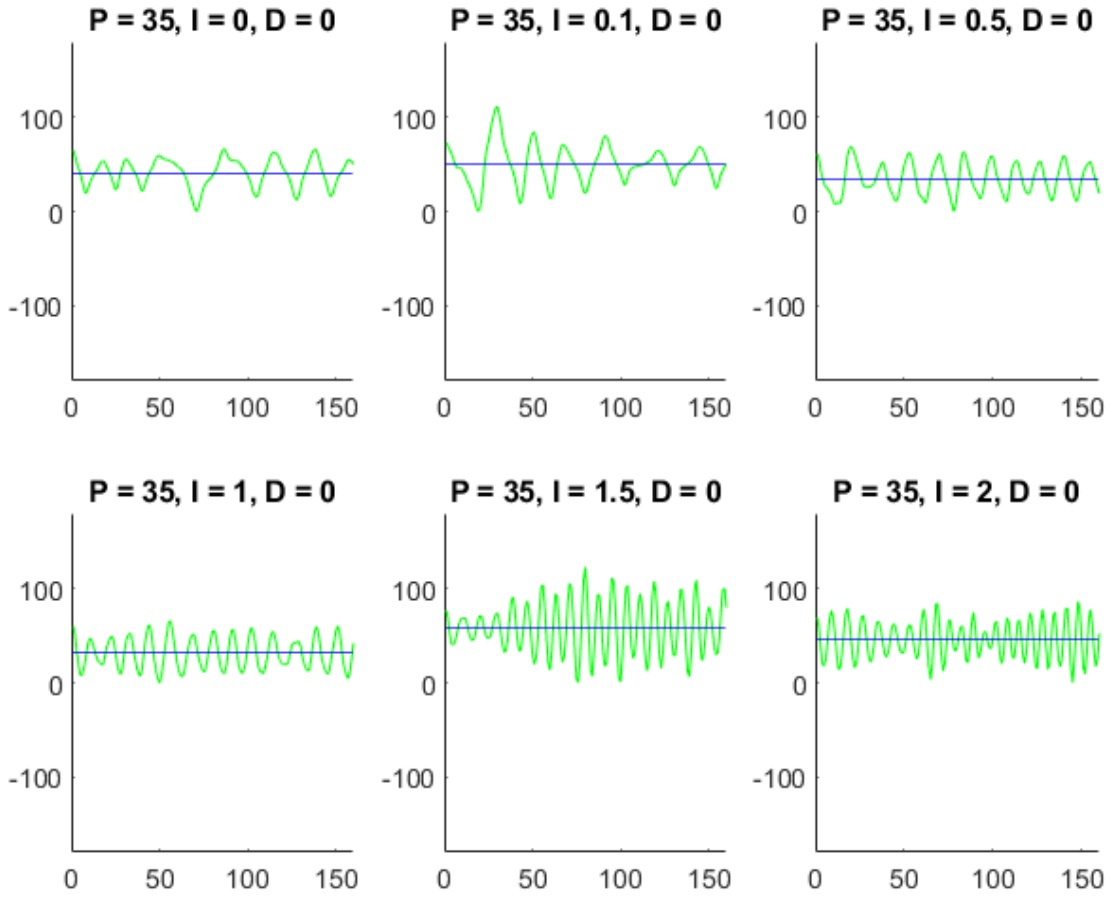
4.4.2 Strojenie regulatora PID sterującego utrzymaniem pozycji pionowej

W tym podrozdziale podobnie jak w poprzednim, dotyczącym strojenia regulatorów *PID* sterujących prędkościami, omówiona zostanie na podstawie wykresów i testów metoda ustawiania odpowiednich parametrów dla regulatora *PID* nadrzędnego odpowiedzialnego za główną funkcję robota, to znaczy utrzymanie pozycji pionowej. Zastosowana została identyczna metoda do tej opisanej i wykorzystanej poprzednio, natomiast zamiast odpowiedzi skokowej, zastosowany został wykres przedstawiający zmianę kąta odchylenia robota w czasie.



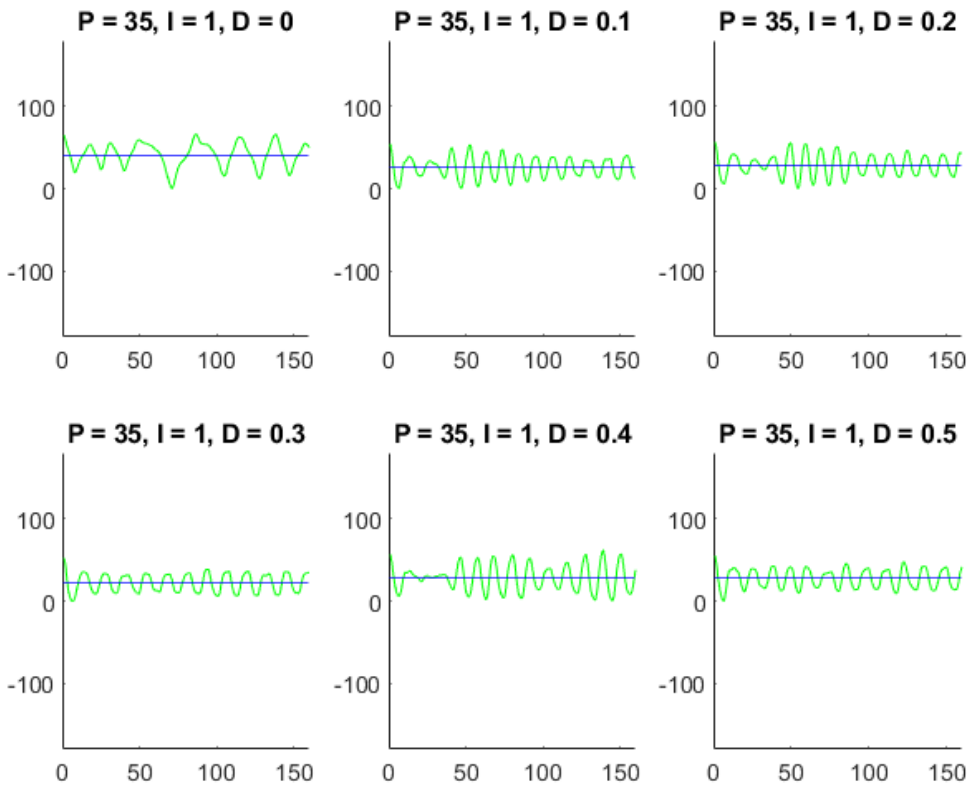
Rysunek 12. Wykresy odpowiedzi skokowej dla różnych wartości parametru P regulatora kątowego

Na powyższym obrazku przedstawione są wykresy aktualnego kąta odchylenia, względem kąta zerowego (niebieska linia) w trakcie pracy robota dla różnych wartości parametru Kp regulatora kątowego. Dla pierwszych trzech wykresów wartość członu proporcjonalnego jest za niska a odpowiedź regulatora wydłużona lub zbyt wolna. Na ostatnich dwóch wykresach widać natomiast pojawiającą się narastającą oscylację. Najlepszy rezultat uzyskujemy, poprzez ustawienie wartości *35* dla zmiennej Kp.



Rysunek 13. Wykresy odpowiedzi skokowej dla różnych wartości parametru I regulatora kątowego

W kolejnym etapie ustawiania parametrów dla regulatora kątowego najlepszy efekt uzyskamy przy wyborze wartości Ki = 1, co przedstawiono na wykresie (d). Ustawiając wartość parametru Ki poniżej 1, uzyskujemy za słabą odpowiedź regulatora, przez co robot nie może utrzymać pozycji stabilnej wystarczająco szybko (pojawiają się większe i mniejsze zakłócenia). Jeśli natomiast wykorzystamy wartość powyżej Ki > 1 w efekcie na zakłócenia robot zaczyna wpadać w narastające oscylacje.

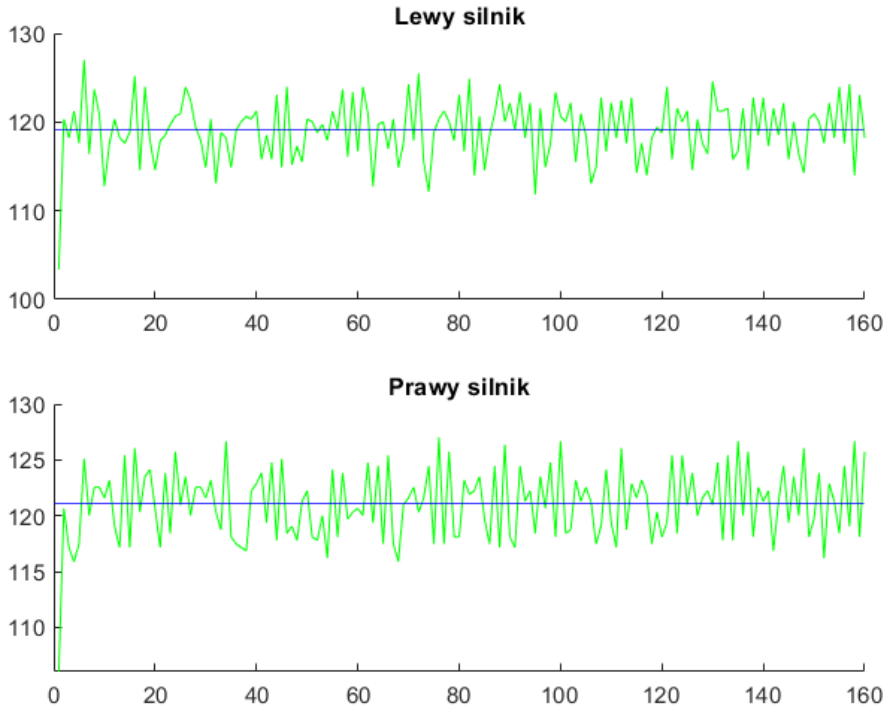


Rysunek 14. Wykresy odpowiedzi skokowej dla różnych wartości parametru I regulatora kątowego

Ostaniem etapem jest ustalenie wartości Kd regulatora, którego zmiana wpływa na wygładzenie wykresu. Najlepszy efekt udało się uzyskać dla wartości Kd = 0.3, co zostało przedstawione na wykresie (d), przedstawionym na rysunku 14. Podobnie jak przy pozostałych parametrach regulatora *PID* wybranie mniejszej wartości dawało za słaby efekt, natomiast wartość większa powodowała wzrost oscylacji, co skutkowało niestabilną pracą robota.

5.Testy

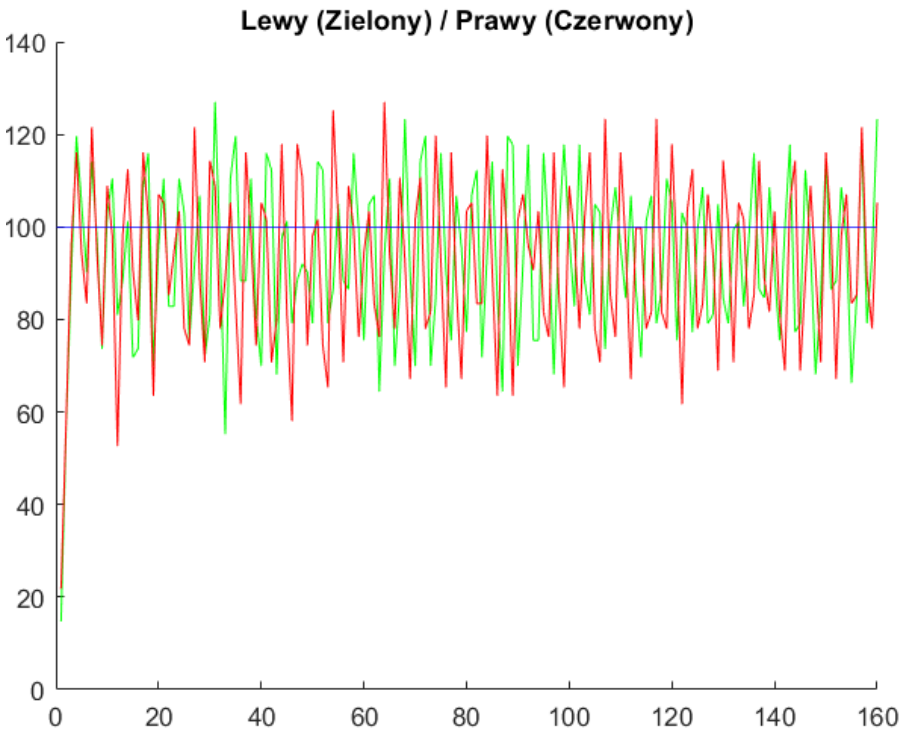
Skuteczność gotowego modułu regulator została sprawdzona na przykładowym robocie, którego budowa została opisana we wcześniejszych rozdziałach. Pokrótce w tym rozdziale przedstawione zostaną i opisane testy obejmujące trzy ważniejsza elementy regulatora samobalansującego.



Rysunek 15. Wykres prędkość silników.

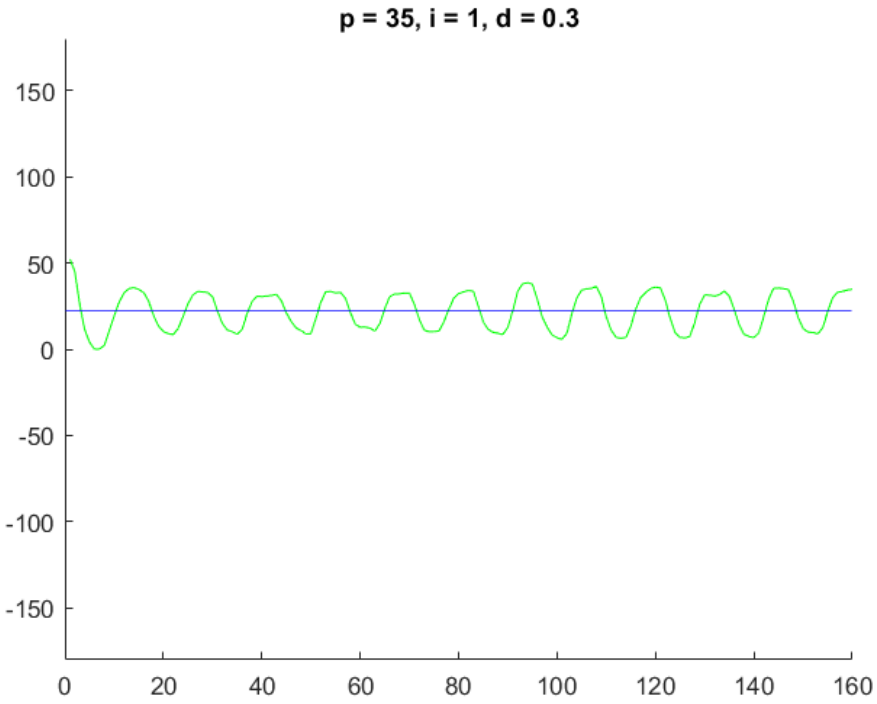
Na rysunku 15 przedstawione są wykresy prędkości obliczonej na podstawie danych odczytanych z enkoderów optycznych dla obydwu silników. Niebieskim kolorem zaznaczona została średnia wartość obliczona z zebranych danych dla każdego z wykresów, która wynosiła: *119.0777* dla silnika lewego oraz *121.0349* dla silnika prawego. Widoczne zakłócenia są efektem wykorzystania nieekranowanych kabli do podłączenia regulatora z silnika i enkoderami.

Na rysunku 16 natomiast przedstawione zostały wykresy prędkości silników lewego (zielony) i prawego (czerwony) z wykorzystaniem regulatorów *PID*. Obrazek ten został skopiowany z poprzedniego rozdziału dotyczącego strojenia regulatorów *PID*, gdzie także został dokładnie omówiony. W tym rozdziale dodatkowo narysowana została niebieska linia reprezentująca prędkość docelową, jaką powinien osiągnąć każdy z silników.



Rysunek 16. Porównanie wykresów odpowiedzi skokowej dla prawego i lewego silnika

Wartość zadana w tym teście wynosiła *100*, natomiast rzeczywista uzyskana średnia wartość dla każdego z silników wynosiła: *93.3634* dla silnika lewego oraz *92.1317* dla silnika prawego. Jak widać, wartości średnie prędkości silników są bardzo zbliżone do wartości zadanej, pomimo dużego zaszumienie sygnału oraz faktu, że w momencie startu prędkość początkowa robota wynosiła zero — co świadczy o tym, że parametry regulatorów *PID* zostały ustawione poprawnie i regulatory spełniają swoje zadanie.



Rysunek 17. Wykresy pracy sterownika

Na rysunku 17 kolorem zielonym przedstawiony został wykres kąta odchylenia odczytany z modułu *MPU6050* w danej chwili oraz kolorem niebieskim średnia wszystkich zebranych próbek wynosząca: *22.0036*. Obliczona wartość średnia nie jest równa zero, a jedynie oscyluje w pewnym stopniu, na co wpływ ma wiele cyników takich jak, waga, specyfika budowy robota testowego, parametry regulatora *PID*, a także inne zakłócenia powstałe w trakcie działania sterownika. Pomimo tego na wykresie widać, że praca sterownika jest stabilna, a powstałe oscylacje są regularne.

6. Wnioski i Podsumowania

W celu realizacji projektu wykorzystane zostały elementy z wielu dziedzin i zagadnień — automatyki, elektroniki i informatyki. Niektóre elementy wymagają jeszcze dopracowania, ale na tym etapie stworzone oprogramowanie wraz z zaprojektowanym sterownikiem spełnia podstawowe założenia projektu.

Dalszy etap rozwoju projektu zakłada elementy takie jak bezprzewodowe sterowanie, swobodne „wstawanie" robota z pozycji leżącej do pozycji pionowej, czy zastosowanie dodatkowych czujników odległości w celu omijanie przeszkód na trasie pojazdu.

Wartości dla regulatorów *PID* zostały dobrane metodą eksperymentalną. W celu strojenia optymalnego, dane zebrane z enkoderów można by wykorzystać do dostrojenia parametrów regulatora *PID* za pomocą narzędzi programu *MATLAB*.

W przedstawionym rozwiązaniu dla sterowników do robotów samobalansujących najważniejszą rolę odgrywają równomiernie współpracujące ze sobą regulatory *PID*. A zatem najważniejszym i zarazem najbardziej wrażliwym elementem układu są parametry tych regulatorów. Optymalny dobór i ustawienie współczynników *P, I, D* dla wszystkich regulatorów będzie miało znaczący wpływ na stabilność pracy sterownika i dopracowanie jego działania, oraz pozwoli na dalszy rozwój tego typu projektów.

Oprogramowanie zostało zaimplementowane w języku C, jednak jest na tyle uniwersalne, że bez problemu można go przełożyć na dowolny inne język czy technologię, jeśli zaistniałaby taka potrzeba. Cały projekt oparty jest o jeden układ na płytce *PCB*, dzięki czemu bardzo łatwo jest wykonać kilka kopii takiego sterownika i zastosować go w różnych konstrukcjach robotów.

Bibliografia

[1] Moduł MPU6050 - Dokumentacja [Online]

https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[2] Akcelerometr i Żyroskop - Informacje ogólne [Online]

https://pl.wikipedia.org/wiki/Żyroskop oraz https://pl.wikipedia.org/wiki/Przyspieszeniomierz

[3] Pitch, Yaw, Roll - Informacje ogólne [Online]

https://en.wikipedia.org/wiki/Aircraft_principal_axes

[4] Regulator PID i jego strojenie - Informacje ogólne [Online]

https://en.wikipedia.org/wiki/PID_controller

[5] Płytka rozwojowa OKDO E1 - Dokumentacja [Online]

https://www.okdo.com/p/okdo-e1-development-board/

[6] LPC 55S69 - Dokumentacja [Online]

https://www.nxp.com/docs/en/user-guide/UM11424.pdf

[7] MCUXpresso SDK — Dokumentacja [Online]

https://mcuxpresso.nxp.com/api_doc/dev/116/index.html

[8] Komunikacja I2C i SPI - Informacje ogólne [Online]

https://en.wikipedia.org/wiki/I²C oraz https://en.wikipedia.org/wiki/SPI

[9] Biblioteka CMSIS DSP - Dokumentacja [Online]

https://arm-software.github.io/CMSIS_5/DSP/html/index.html

[10] MATLAB - Dokumentacja [Online]

https://www.mathworks.com/help/matlab/

[11] Projektowanie PCB - Dokumentacja [Online]

http://eagle.autodesk.com/eagle/documentation