Dokumentacja Techniczna Drona QUAD 4X

***Temat: Konstrukcja półautonomicznego drona z zastosowaniem algorytmu PID***

Opracował: Krzychu Jura

Spis treści

[**Krótki wstęp teoretyczny** 3](#_Toc161514606)

[Podłączenie Modułów elektronicznych 9](#_Toc161514607)

[Ustawienie Śmigieł i pinnout motorów oraz warunkowy kierunek obrotu oraz dokrecenie CW oraz CCW 12](#_Toc161514608)

[Co jeszcze jest niezbędne? 13](#_Toc161514609)

[Schemat Programu 14](#_Toc161514610)

[Działanie kodu, PID, Complementary filter, euler angles 15](#_Toc161514611)

[FUNKCJA IMPCALIBTST() 19](#_Toc161514612)

[FUNKCJA resetPID() oraz resetMotors() 19](#_Toc161514613)

[FUNKCJA anglesRead() i Kalibracja MPU6050 20](#_Toc161514614)

[FUNKCJA pidEquation() 22](#_Toc161514615)

[FUNKCJe update() i fly() 23](#_Toc161514616)

[FUNKCJA setup() i Loop() 25](#_Toc161514617)

[Podsumowanie 44](#_Toc161514618)

# Wprowadzenie do projektu

**Konstrukcja półautonomicznego drona z zastosowaniem algorytmu PID**

W ramach niniejszego projektu inżynierskiego zajmiemy się konstrukcją półautonomicznego drona, który będzie w stanie samodzielnie utrzymywać równowagę dzięki zastosowaniu zaawansowanego algorytmu PID (Proportional-Integral-Derivative). Celem projektu jest stworzenie systemu, który z jednej strony umożliwi dronowi autonomiczne stabilizowanie lotu, a z drugiej pozwoli operatorowi na pełną kontrolę kierunku i wysokości lotu za pomocą zdalnego sterowania. Konstrukcja półautonomicznego drona z zastosowaniem algorytmu PID i zdalnego sterowania stanowi ambitne wyzwanie inżynierskie, które ma na celu połączenie nowoczesnych technologii stabilizacji i precyzyjnego sterowania. Realizacja tego projektu nie tylko przyczyni się do rozwoju wiedzy i umiejętności w dziedzinie systemów bezzałogowych, ale także stworzy podstawy dla dalszych badań i innowacji w tej dynamicznie rozwijającej się branży.

**Opis działania**

Dron będzie wyposażony w system sterowania, który realizuje dwa główne zadania:

Autonomiczna stabilizacja lotu: Za utrzymanie równowagi drona w powietrzu odpowiadać będzie algorytm PID. Algorytm ten, poprzez ciągłą analizę i korektę parametrów lotu, takich jak pitch (pochylenie), roll (przechył) i yaw (obrót wokół osi pionowej), zapewni stabilność drona w każdych warunkach.

Zdalne sterowanie przez operatora: Operator drona będzie miał możliwość pełnego sterowania jego ruchem za pomocą kontrolera zdalnego sterowania. Przekazywanie poleceń do drona odbywać się będzie poprzez zadawanie odpowiednich wartości PWM (Pulse Width Modulation). Kontrola ta obejmuje następujące parametry:

* Throttle (przepustnica): kontroluje wysokość lotu drona,
* Pitch (pochylenie): kontroluje ruch drona do przodu i do tyłu,
* Roll (przechył): kontroluje ruch drona w lewo i w prawo,
* Yaw (obrót): kontroluje obrót drona wokół własnej osi pionowej.

Pozostałe aspekty stabilizacji lotu będą regulowane automatycznie przez system sterowania drona, co pozwoli operatorowi skupić się na precyzyjnym kierowaniu dronem w pożądanym kierunku.

**Cele projektu**

Głównym celem projektu jest stworzenie drona, który będzie łączył zalety autonomicznych systemów stabilizacji z elastycznością i precyzją zdalnego sterowania. Dzięki zastosowaniu algorytmu PID dron będzie w stanie szybko i efektywnie reagować na zmiany warunków lotu, minimalizując ryzyko utraty kontroli i zwiększając bezpieczeństwo operacji. Jednocześnie, dzięki zdalnemu sterowaniu, operator będzie mógł z łatwością nawigować dronem, wykorzystując jego pełen potencjał w różnych zastosowaniach praktycznych.

**Zakres prac**

Projekt obejmuje:

* Projektowanie i implementację algorytmu PID dla stabilizacji lotu drona,
* Integrację systemu zdalnego sterowania z dronem,
* Testowanie i optymalizację działania drona w różnych warunkach operacyjnych,
* Dokumentację procesu konstrukcji i wyników testów.

# Przegląd literatury

Drony, czyli bezzałogowe statki powietrzne (UAV - Unmanned Aerial Vehicles), zyskują na popularności dzięki ich szerokiemu zastosowaniu w różnych dziedzinach, takich jak fotografia, nadzór, dostawy, badania naukowe, czy rolnictwo. Jednym z kluczowych wyzwań w projektowaniu dronów jest utrzymanie ich stabilności w przestrzeni. W tym celu powszechnie stosuje się algorytmy PID (Proportional-Integral-Derivative), które pozwalają na precyzyjne sterowanie i stabilizację dronów. Ponadto, kontrola dronów może być realizowana za pomocą systemów transmitter & receiver, umożliwiających operatorowi zdalne sterowanie dronem.

Algorytm PID w stabilizacji dronów literatura

Algorytm PID jest jednym z najczęściej stosowanych regulatorów w inżynierii sterowania, w tym także w stabilizacji dronów. Składa się z trzech komponentów: proporcjonalnego (P), całkującego (I) i różniczkującego (D). Każdy z tych komponentów odpowiada za inną część odpowiedzi systemu, co pozwala na uzyskanie stabilnego i precyzyjnego sterowania.

"Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment" - Randal W. Beard, Timothy W. McLain

Beard i McLain omawiają teoretyczne i eksperymentalne aspekty dynamicznego sterowania quadkopterami, w tym zastosowanie algorytmów PID do stabilizacji lotu.

"Modern PID Control" - Katsuhiko Ogata

Ogata dostarcza szerokiego omówienia nowoczesnych technik sterowania PID, w tym ich zastosowań w różnych systemach inżynieryjnych, takich jak drony.

"PID Control in the Third Millennium: Lessons Learned and New Approaches" - Ramon Vilanova, Antonio Visioli

Książka zawiera przegląd najnowszych osiągnięć i metod w dziedzinie sterowania PID, z naciskiem na aplikacje w nowoczesnych systemach, takich jak drony.

"Systemy sterowania dronami" - Janusz Szczepański

Szczepański przedstawia kompleksowe omówienie systemów sterowania dronami, w tym zastosowanie algorytmów PID do stabilizacji lotu.

"Podstawy sterowania dronami: teoria i praktyka" - Piotr Kowalski

Kowalski opisuje podstawowe zasady sterowania dronami, w tym wykorzystanie algorytmów PID w różnych scenariuszach lotu.

Sterowanie dronami za pomocą transmitter & receiver literatura

Sterowanie dronami przez operatora za pomocą systemów transmitter & receiver jest kluczowym elementem w wielu aplikacjach. Transmittery umożliwiają wysyłanie sygnałów sterujących zdalnie, podczas gdy receivery odbierają te sygnały i przekształcają je w odpowiednie komendy dla drona.

"Remote Control Systems: Principles and Applications" - Frank L. Lewis

Lewis omawia zasady działania i zastosowania systemów zdalnego sterowania, w tym systemów transmitter & receiver stosowanych w dronach.

"Radio Control for Dummies: A Comprehensive Guide" - John D. Grainger

Grainger przedstawia przewodnik po systemach radiowego sterowania, wyjaśniając techniczne aspekty transmitterów i receiverów.

"Technologie zdalnego sterowania dronami" - Marek Wiśniewski

Wiśniewski analizuje technologie wykorzystywane w zdalnym sterowaniu dronami, w tym szczegółowy opis systemów transmitter & receiver.

"Zdalne sterowanie w robotyce i dronach" - Adam Nowak

Nowak dostarcza przeglądu technologii zdalnego sterowania, ze szczególnym uwzględnieniem ich zastosowania w dronach.

Publikacje naukowe

"A Survey of Quadrotor Drones Control Methods and Related Challenges" - Journal of Intelligent & Robotic Systems

Artykuł przedstawia przegląd metod sterowania quadkopterami, w tym wykorzystanie algorytmów PID do stabilizacji lotu.

"Enhanced PID Control for Quadrotor UAV" - IEEE Transactions on Control Systems Technology

Publikacja omawia zaawansowane techniki PID w kontekście sterowania quadkopterami, przedstawiając wyniki eksperymentalne.

"Stabilizacja lotu dronów za pomocą algorytmu PID" - Pomiary Automatyka Kontrola

Artykuł opisuje zastosowanie algorytmu PID do stabilizacji lotu dronów, prezentując zarówno teoretyczne, jak i praktyczne aspekty.

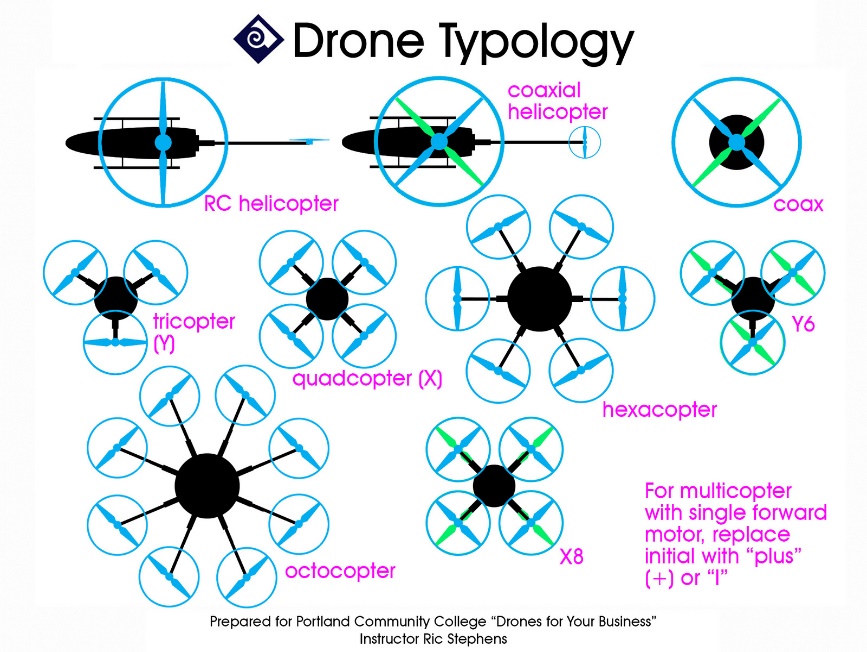
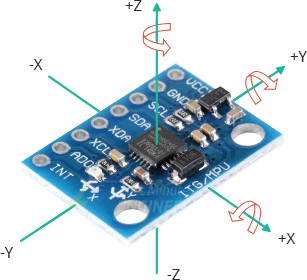
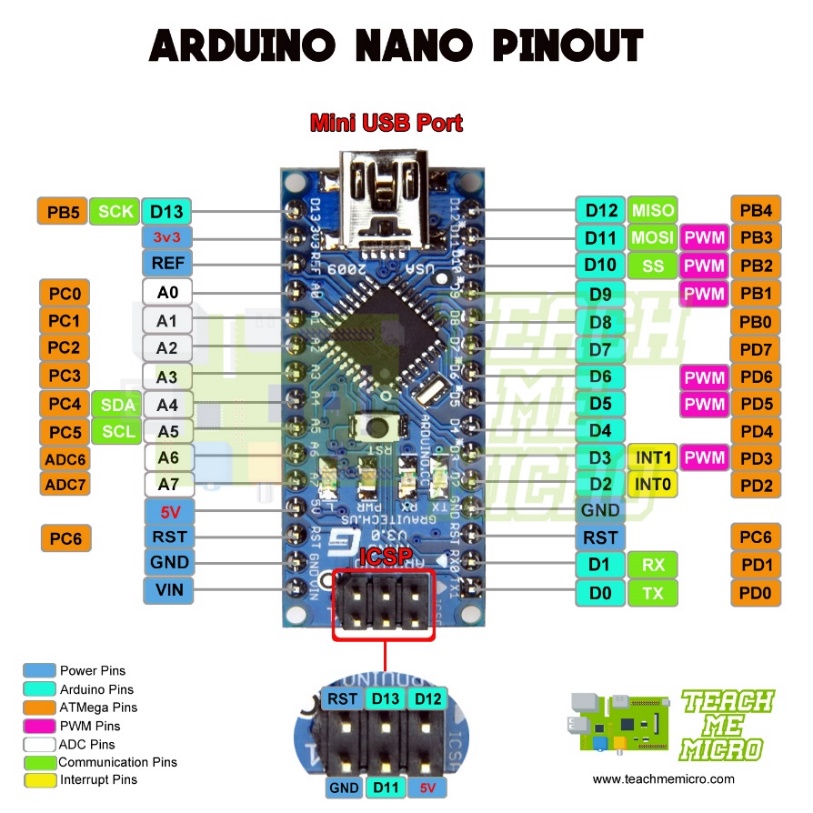
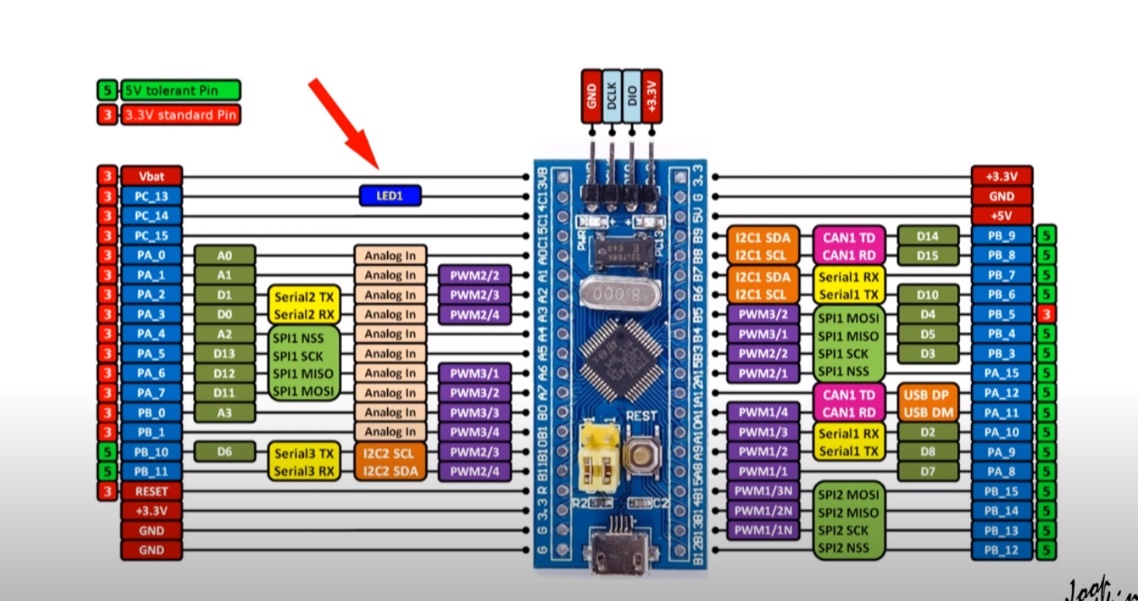
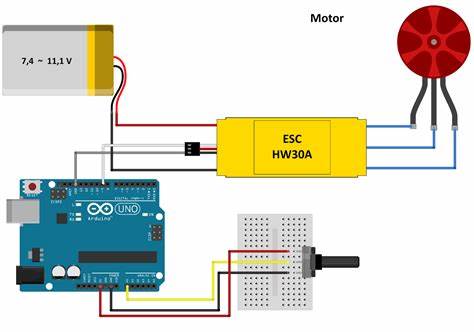
"Systemy sterowania dronami: badania i rozwój" - Przegląd Elektrotechniczny

Publikacja zawiera przegląd najnowszych badań w dziedzinie systemów sterowania dronami, ze szczególnym uwzględnieniem algorytmów PID i systemów transmitter & receiver.

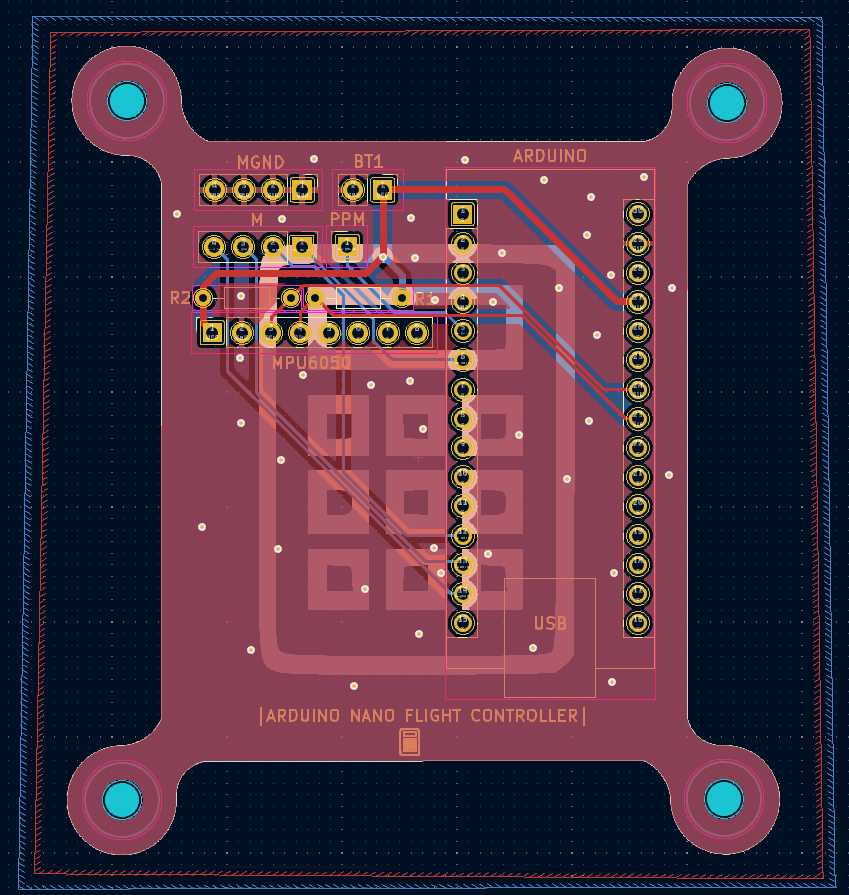
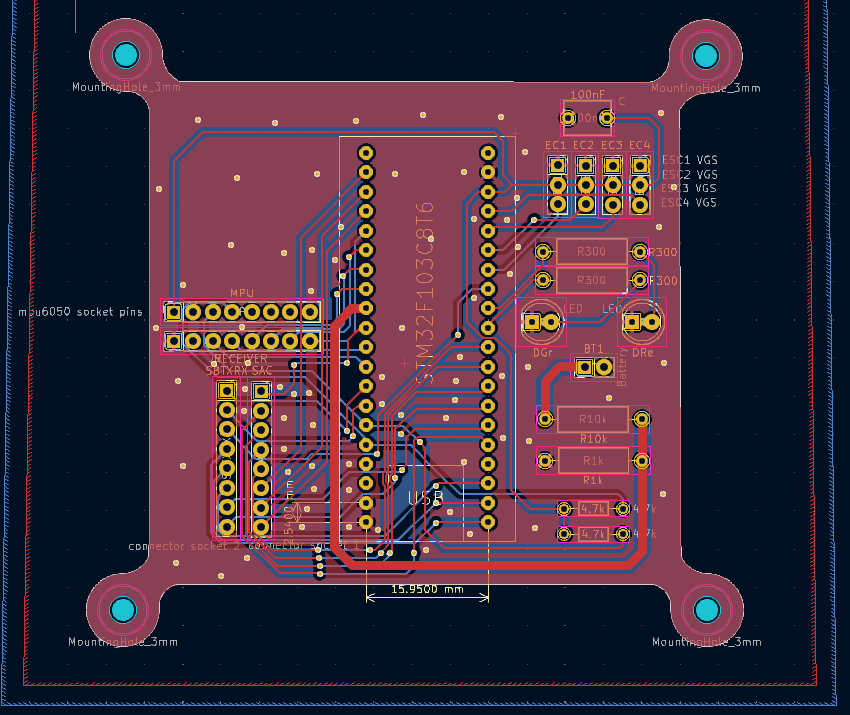
Literatura dotycząca stabilizacji dronów za pomocą algorytmu PID oraz zdalnego sterowania za pomocą systemów transmitter & receiver jest obszerna i obejmuje zarówno teoretyczne podstawy, jak i praktyczne zastosowania. Algorytm PID pozostaje kluczowym narzędziem w zapewnianiu stabilności i precyzji lotu, podczas gdy systemy zdalnego sterowania umożliwiają operatorom skuteczne zarządzanie dronami. Dostępne publikacje w języku angielskim i polskim dostarczają niezbędnej wiedzy i narzędzi do dalszych badań i rozwoju w tej dziedzinie.

# Krótki wstęp teoretyczny (metodologia cz.1)

Nie będzie tu dużo (ogólnej) teorii (bez praktyki bezwartościowa) [w sensie podam Ci teorie która odnosi się do tego co ci się przyda TYLKO] więc przejdę do tego jak bym chciał żeby na początku mi ktoś to wyjaśnił (kiedy zaczynałem ten projekt):

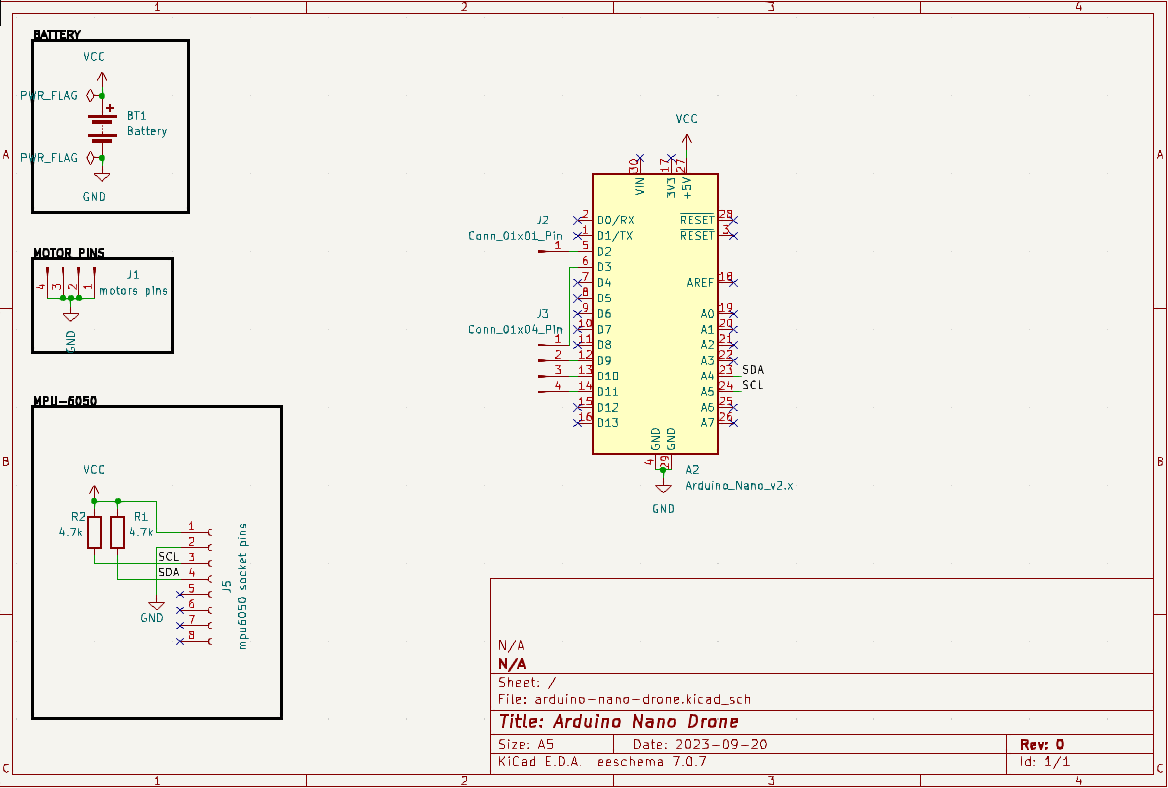
1. Typów dronów masz parę (masz rysunek poniżej), dlaczego o tym piszę? Bo każdym steruje się inaczej. To nie jest tak, że dasz sobie pwm i 7.4V (generalnie od 7.4 w naszym projekcie do 11.1V = 7.4 + 3.7 -> jak znasz lipo to wiesz o czym mowie jak nie to koniecznie się zapoznaj [Akumulatory litowo–polimerowe, Li–po – kompendium cz.1 • FORBOT](https://forbot.pl/blog/akumulatory-litowo-polimerowe-li-po-kompendium-cz-1-id291)) i lecisz. Jest masakrycznie dużo kruczków które w rzeczywistości niestety istnieją (fizyka pozdrawia).
2. Musisz wiedzieć czym jest prędkość obrotowa (pulsacja) -> pochodzi z żyroskopu, przyspieszenie grawitacyjne -> pochodzi z akcelerometru. Akcelerometr + żyroskop = IMU. (celowo tłumaczę jak dla przedszkolaka żebyś mnie dobrze zrozumiał nie będę operował zbyt technicznym słownictwem, chyba że będzie to zupełnie konieczne). Nasze IMU = MPU6050. Bardzo proszę - zapoznaj się ze zdjęciem to jaką masz orientację mpu6050 zależy od wektorów przyspieszenia oraz od prędkości kątowej, masz zdjęcie poniżej (mega często pojawia się w kodzie, wzdłuż osi Z mierzysz acceleration Z oraz angular velocity Z i analogicznie z pozostałymi osiami). Co także bardzo istotne w kontekście tego konkretnego projektu to to żeby mpu6050 było skierowane pinoutem do tyłu drona tak żeby kropka na IC mpu6050 wskazywała left back motor naszego drona. Sterowanie drona w kodzie zostało napisane właśnie pod takie ułożenie (ma to znaczenie przy obliczaniu wartości regulacyjnych pwm podawanych do ESC przy PID)
3. Ponieważ pierwsza płytka (rewizje: 1, 2) do drona była także napisana pod AVR, powinieneś wiedzieć jak obsługiwać arduino nano (najlepiej sprawnie umieć cos napisać na tej platformie). Mowa głownie o umiejętności generowania pwm, obsługi wejść wyjść oraz posługiwania się zaawansowanymi strukturami oraz funkcjami programistycznymi ([Kurs podstaw Arduino – spis treści, wstęp • FORBOT](https://forbot.pl/blog/kurs-arduino-podstawy-programowania-spis-tresci-kursu-id5290)). Co nas także będzie interesowało to, to że w gruncie rzeczy ta platforma nie nadaje się do tak zaawansowanego projektu jakim jest dron Quad 4X, maksymalna częstotliwość taktowania dla mikrokontrolera Atmega 328p który siedzi na pokładzie arduino nano to 20MHz. Dla porównania dużo tańszy stm32f103 (w oparciu o który zostały opracowane rewizje: 3, 4 pcb do drona) ma ok 4 razy większe taktowani. Co także ważne to, to żebyś umiał się połapać w wyprowadzeniach pinów. Programowanie w AVR będzie bardzo przydatne także ze względu na fakt, że będziemy używać specjalnej nakładki na arduino ide która pozwoli nam programować STM w sposób analogiczny do arduino. (Możemy także w tym celu użyć stmcubeide, kwestia gustu).
4. Nie będę się tu rozwodził nad działaniem silników BLDC ani ESC. Najważniejsze jest żebyś wiedział ze nie możesz sobie zrobić tak jak w DC ze dajesz sobie mostek H i sterujesz podając odpowiedni woltaż, tzn. żeby silniki BLDC zadziałały musisz je kontrolować przez ESC - electronic speed controler (nazwa mówi sama za siebie 😉). Upraszczając zasadę działania jak tylko się da: podajesz pwm na ESC -> im większy pwm (większe wypełnienie) tym szybciej silnik będzie się kręcił w 1 lub 2 stronę (bo TAK: ZALEZY TO NIESTETY OD SPECYFIKI ESC oraz od sposobu podłączenia przewodów ESC <-> motor)
5. PID, katy Eulera, filtr komplementarny, filtr Kalmana, różne typy FC – flight controller oraz algorytmów regulacji (np. PID controller) nie będę omawiał dokładnie w tej części tekstu. Wyjaśnię je na przykładach praktycznych, kiedy przejdziemy do kodu (MUSIAŁBYM CI PODAWAC OGÓLNĄ zaawansowana TEORIE KTÓREJ PODAWAĆ NIE CHCE. Co istotne to do obliczenia kątów użyto prostych zależności geometrycznych (dla acceleration angles) oraz gyro angles z uwzględnieniem zjawiska gyro drift (spowodowanego własnościami całki, którą bierzemy z angular velocity), do tak obliczonych kątów stosujemy filtr komplementarny w celu otrzymania przybliżonych wartości kątów (obyliśmy się bez kątów Eulera). Do regulacji wartości PWM podawanych na ESC (obliczone jako błędy w odchyleniu od ustalonego setpointu, (0, 0, 0) użyto algorytmu PID.

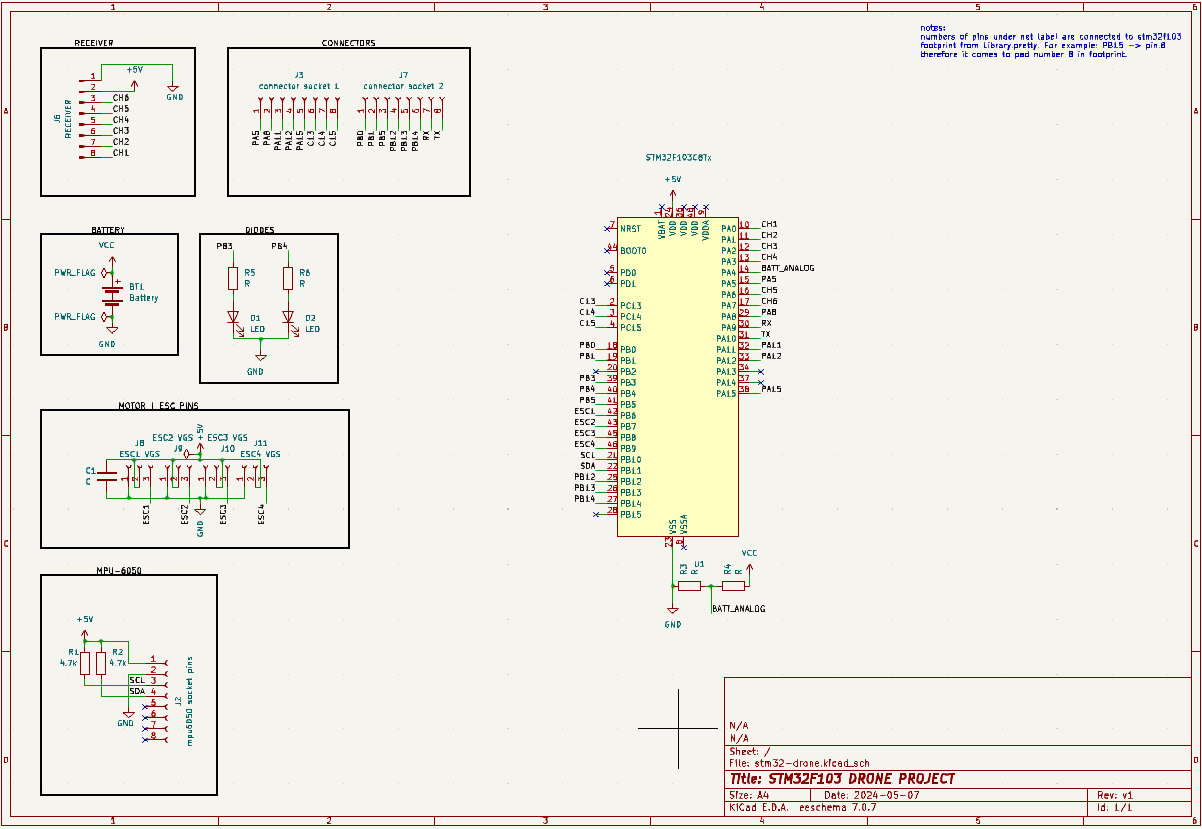
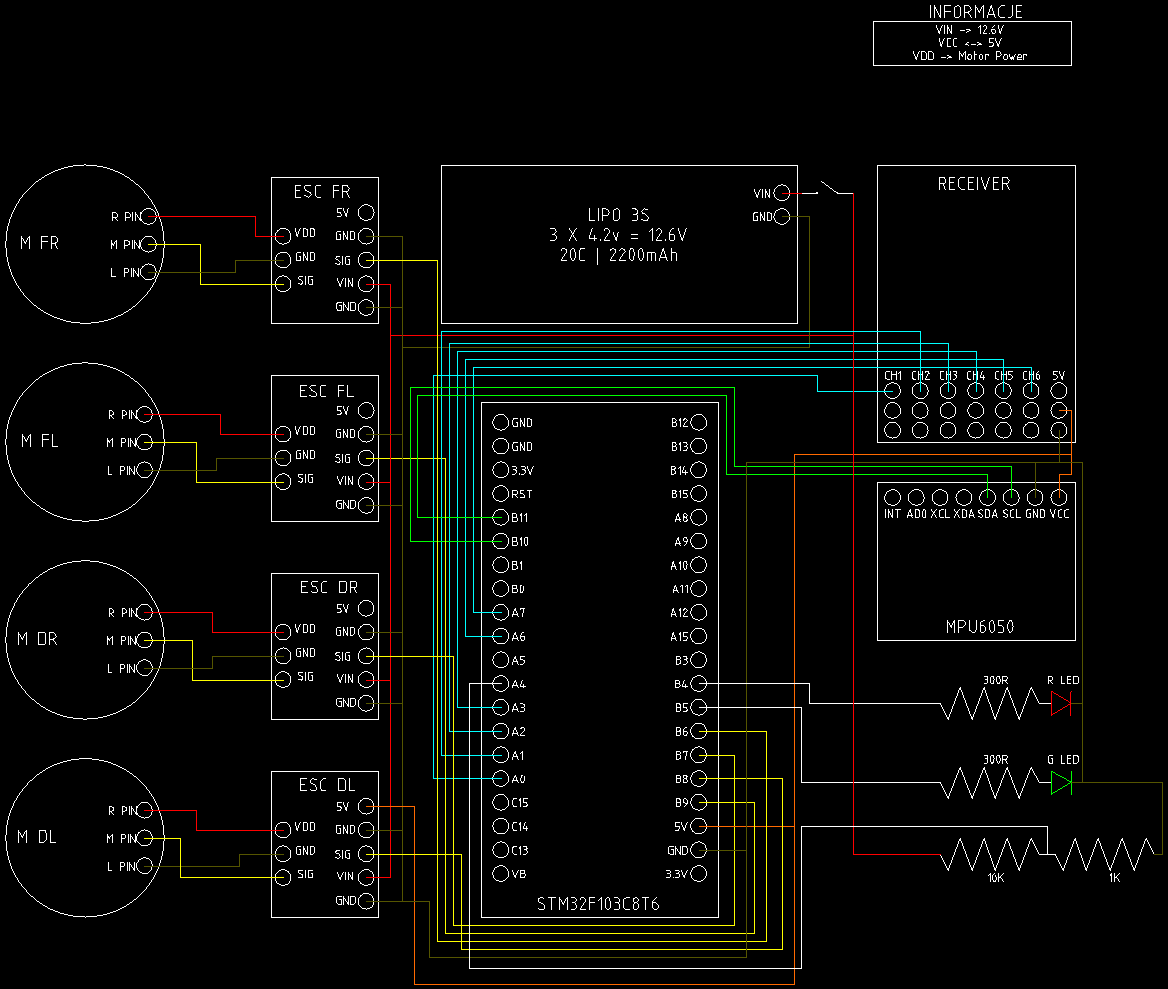
# Podłączenie Modułów elektronicznych (Metodologia cz.2)

Przedstawiam podłączenie modułów ESC -> motorów, mpu6050, baterii LIPO 3S, switch’a do AVR’ki (Jeżeli nie kapujesz mniej więcej polaczeń, które zrobiłem sięgnij proszę do fotki pinoutu dla arduino nano, masz tam piny sda, scl -> mpu, 5V, VIN, GND, PWM pins, to wszystko ma znaczenie, jeżeli czegokolwiek nie rozumiesz zgłoś się do Tomka, ew. do mnie na spokojnie wyjaśnimy o co chodzi.

Rys. 1. projekt starej płytki pod AVR (Rev.1)

Rys. 2. Projekt nowej płytki pod STM32F103 (Rev.4)

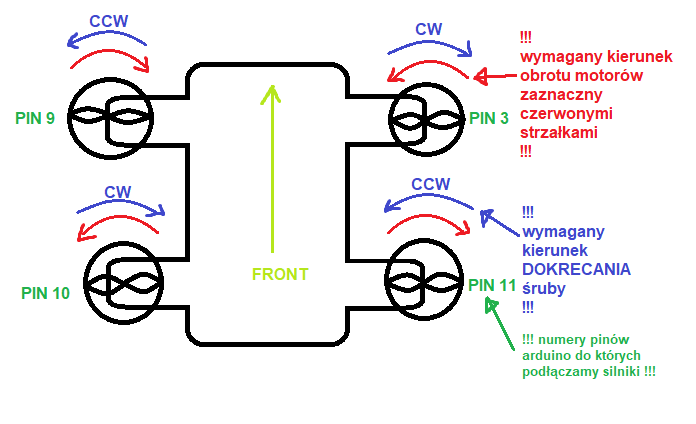
Różnice między dwoma pokazanymi płytkami są bardzo duże. Na przełomie 6 miesięcy całkowicie wyeliminowano arduino nano z projektu zastępując je STM32F103C8T6, dodano wejście analogowe które poprzez dzielnik napięcia oraz przetwornik ADC analizuje stan napięcia na akumulatorku LIPO 3S zasilającym cały układ. Dodano diody sygnalizujące błędy w poprawnym działaniu drona, dodano także dodatkowe elementy poprawiające pracę całego układu (dodatkowe kondensatorki). Szerokość ścieżek także została dokładnie obliczona z użyciem wzoru z normy IPC-2221 (narzędzie <https://www.digikey.pl/>). Wyprowadzono także pozostałe piny STM32F103 w celu możliwości przyszłej rozbudowy układu. Pokazałem schemat płytki flight controllera pora na schemat połączeń zewnętrznych (zasadniczo się nie zmienił po dodaniu nowej płytki)

Rys. 5. Dokładny schemat połączeń na płytce (Rev. 1)

Rys. 3 Schemat połączeń poza płytką

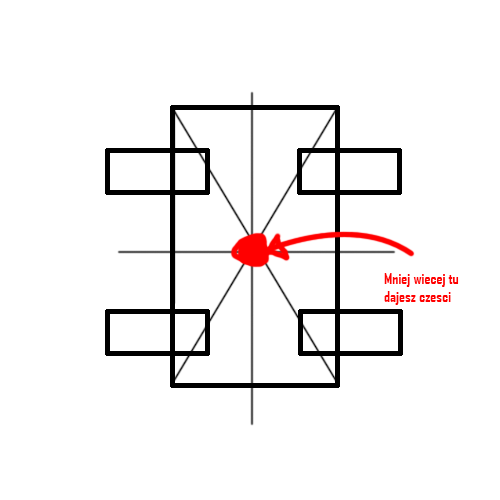
Rys. 4 Dokładny schemat połączeń na płytce (Rev. 4)

# Ustawienie Śmigieł i pinnout motorów oraz warunkowy kierunek obrotu oraz dokrecenie CW, oraz CCW (Metodologia cz.3)

Krótki wstęp: każdy motor w zależności od swojego położenia (lewy górny, lewy dolny, …) ma swój własny niepowtarzalny sygnał pwm regulujący jego szybkość niepowtarzalność nadaje mu właśnie PID. Dlatego MA ZNACZENIE do jakiego pinu AVR/STM podłączysz wyprowadzenie SIG od ESC, NIESTETY… Dodatkowo każdy z silników musi się obracać także w odpowiednim kierunku (żeby odpowiednie warunki aerodynamiczne były spełnione). Dodatkowo na każdym z silników śruba powinna dokręcać się w zależności w kierunku clockwise (CW) lub counter clockwise (CCW) tak by działająca siła odśrodkowa dokręcała ją w trakcie obrotu łopatek. Sporo tych warunków prawda? Musze Cię jeszcze zasmucić to nie koniec warunków, ale uprzedzałem jak coś. Pokazuję zdjęcie jak powinny się silniki kręcić i kierunek dokręcania śruby:

Podobnie sprawa ma się ze śmigłami, żeby dron mógł latać odpowiednie wymagania fizyczne muszą być spełnione i zapewniona odpowiednia siła ciągu, zarówno od motorów jak i od charakteru współdziałania łopatek z powietrzem WIĘC TAK, TO W JAKIM MIEJSCU DASZ JAKĄ ŁOPATKE TAKŻE MA ZNACZENIE (jeżeli łopatkę z wychylenie dolnym dasz w miejsce łopatki z wychyleniem górnym dron zamiast ładnie polecieć będzie wirował w miejscu. Sposób ułożenia łopatek:

## Co jeszcze jest niezbędne? (Metodologia cz.4)

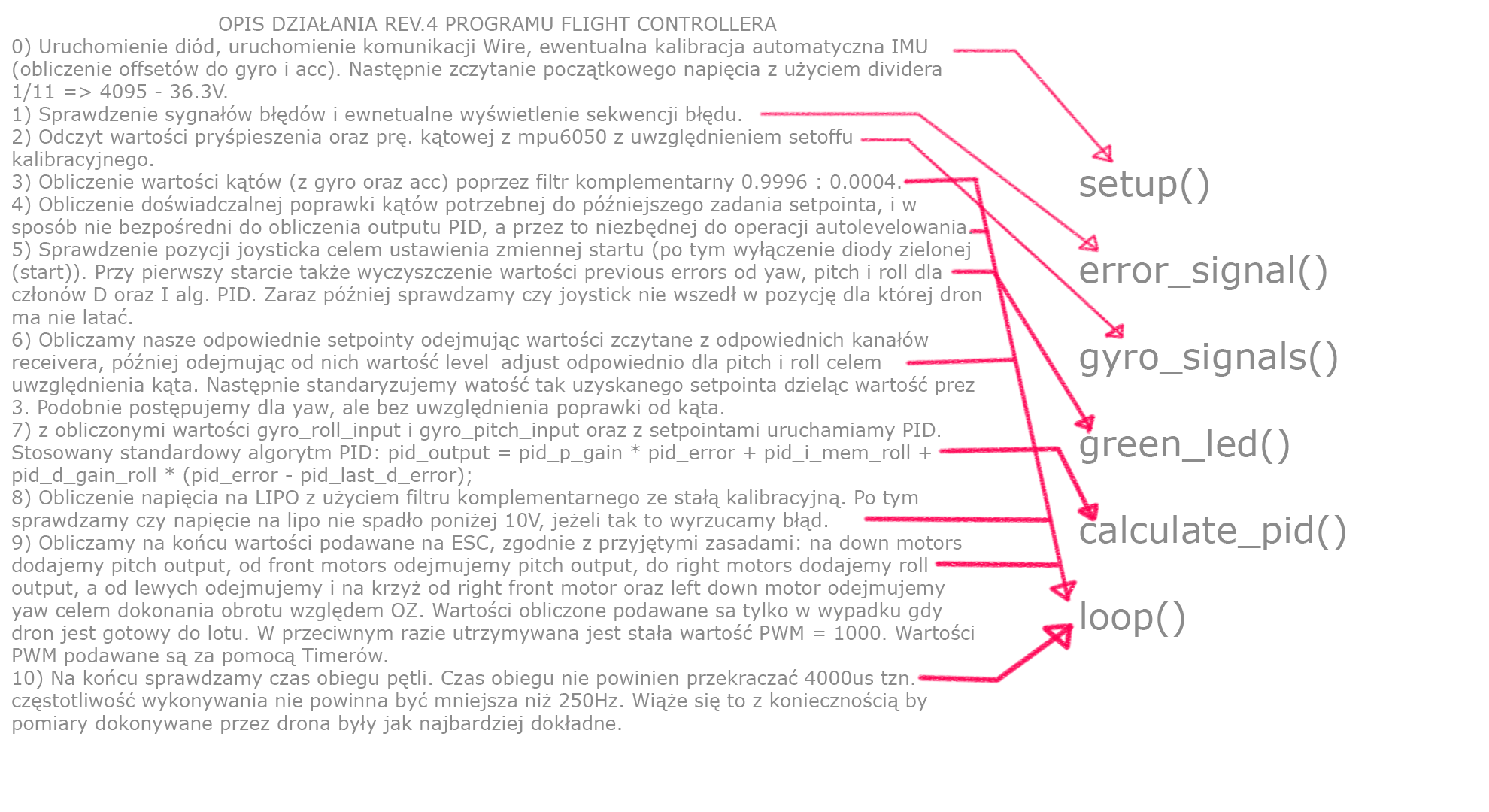
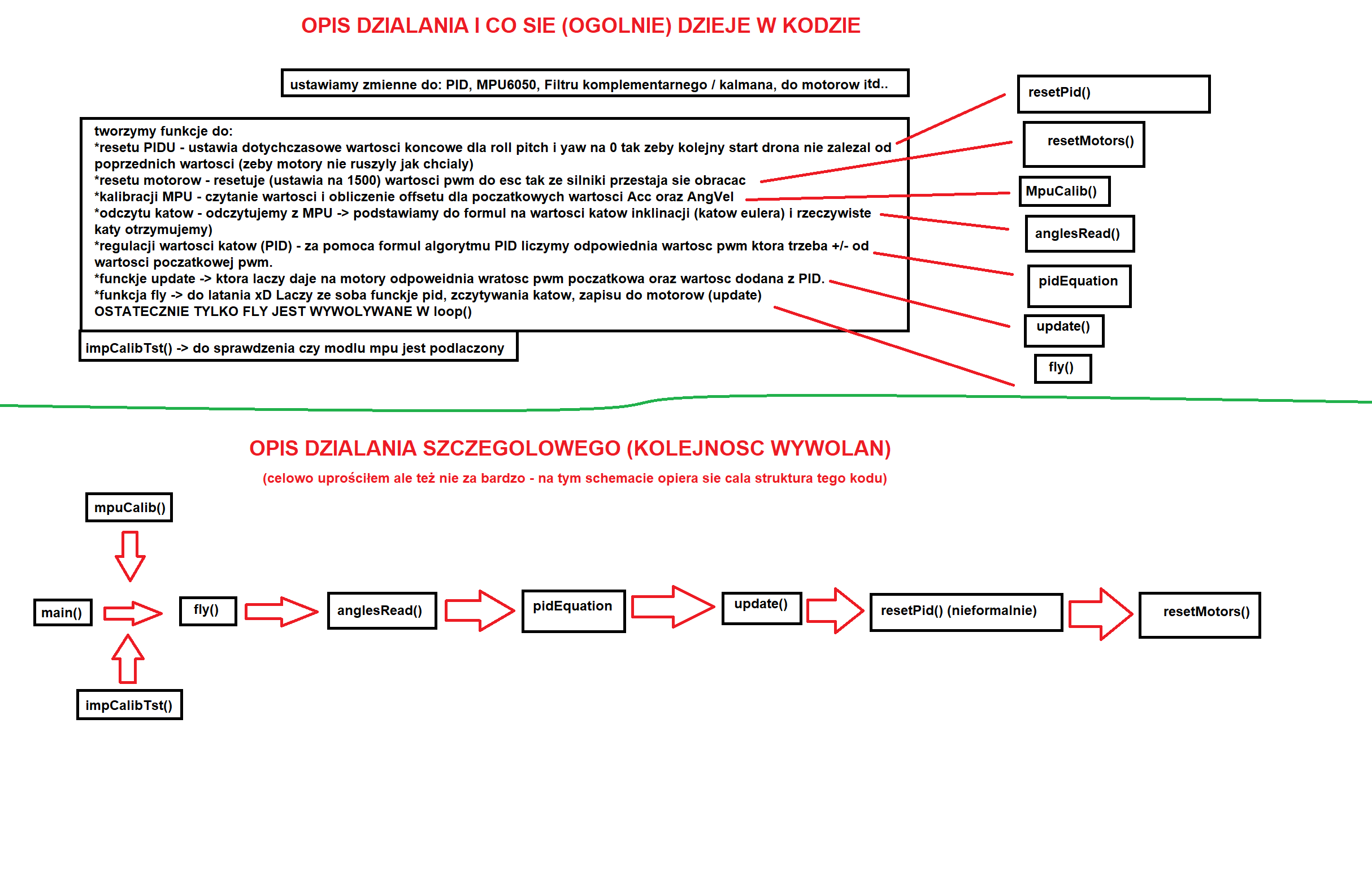
Musisz jeszcze ustawić odpowiednio środek ciężkości drona. Jeżeli nawalisz mu jakichś ciężkich gratów po bokach podczas startu będzie szorował łopatkami o podłogę - źle dla niego i źle dla Ciebie. Dlatego musisz zrobić tak by środek ciężkości drona znajdował się w przybliżeniu (dość dobrym) w dokładnie połowie konstrukcji samego drona, tzn:

Zasadniczo warunek ten osiągnąć jest dość prosto wystarczy, że upper base drona będzie w przybliżeniu kwadratowy i arms drona będą ułożone w stosunku do siebie pod kątem 90 stopni.

Mniej ważnym lecz nadal istotnym z punktu widzenia całości jest odpowiednie zbalansowanie śmigieł drona. W celu odpowiedniego zbalansowania śmigieł możemy użyć propeller balancer lub też zwyczajnie przykleić igłę do idealnie poziomej nawierzchni i położyć na niej śmigło tak by jedna część śmigła była po lewej, a druga po prawej (jest to bardzo ciężkie do osiągnięcia bez posiadania odpowiednich narzędzi) dlatego zaleca się propeller balancer.

# Schemat Programu (metodologia cz.5)

1. Jak działa program (schemat blokowy)

Myślę, że sam schemat działania już na podstawie tego co przeczytałeś powinien być w miarę jasny (jakby nie był to teraz wyjaśnię to jeszcze raz bardzo dokładnie (wszystko od początku), pamiętaj, że wtedy, kiedy to czytasz kod mógł ulec drobnym modyfikacjom które nie zostały jeszcze wprowadzone do tej pracy). Oczywiście funkcje (odpowiedzialne za określone, wypisane działanie) też wymagają wyjaśnienia, zrobimy to przy omawianiu kodu flight controllera.

Rys. 6. Opis działania kodu (Rev.1)

Rys. 7. Opis działania kodu (Rev.4)

# kod Flight controllera (AVR, rev.0)

Dobra pogadajmy teraz o czymś najważniejszym i zarazem najtrudniejszym, dotychczas były przedstawione kruczki mechaniczne, teraz będą przedstawione kruczki programistyczne.

#include <Wire.h>

#include <Servo.h>

#include <MPU6050\_light.h>

#define MOTORDR 11   //Down Right motor

#define MOTORDL 10   //Down Left motor

#define MOTORUL 9    //Upper Left motor

#define MOTORUR 3    //Upper Right motor

#define MAXMICROSECONDS 2000

#define MINMICROSECONDS 1000

//motor vars

Servo upperLeftMotor;

Servo downLeftMotor;

Servo upperRightMotor;

Servo downRightMotor;

int throttleCutOff = 1500;

int maxFlyable = 1800;  //+140 from pid, = 1800

int minFlyable = 1650;  //+140 from pid, = 1650

float upperLeftPwm = 0.0f, downLeftPwm = 0.0f, upperRightPwm = 0.0f, downRightPwm = 0.0f;

float upperLeftPwmFinal = 0.0f, downLeftPwmFinal = 0.0f, upperRightPwmFinal = 0.0f, downRightPwmFinal = 0.0f;

float gainConst = 1.0f;

bool wasThereFlight = 0;

//pid vars

float rollPid = 0.0f, rollError = 0.0f, rollPrevError = 0.0f;    //vars for roll pid

float rollPTerm = 0.0f;    //initial values

float rollITerm = 0.0f;

float rollDTerm = 0.0f;

float rollPConstant = 0.7f; //3.55

float rollIConstant = 0.006f;   //0.003

float rollDConstant = 1.2f; //2.05

float desiredXAngle = 0.0f;    //desired roll angle 0 to stabilise

float pitchPid = 0.0f, pitchError = 0.0f, pitchPrevError = 0.0f; //vars for pitch pid

float pitchPTerm = 0.0f;   //initial values

float pitchITerm = 0.0f;

float pitchDTerm = 0.0f;

float pitchPConstant = 0.72f;   //3.55

float pitchIConstant = 0.006f;  //0.003

float pitchDConstant = 1.22f;   //2.05

float desiredYAngle = 0.0f;    //desired pitch angle

float yawPid = 0.0f, yawError = 0.0f, yawPrevError = 0.0f; //vars for pitch pid

float yawPTerm = 0.0f;   //initial values

float yawITerm = 0.0f;

float yawDTerm = 0.0f;

float yawPConstant = 0.72f;   //3.55

float yawIConstant = 0.006f;  //0.003

float yawDConstant = 1.22f;   //2.05

float desiredZAngle = 0.0f;    //desired yaw angle

//time vars

float time = 0.0f;

float timePrev = 0.0f;

float elapsedTime = 0.0f;

unsigned long flyTime = 0UL;

//GY521 vars

MPU6050 mpu(Wire);

double angleX = 0.0;

double angleY = 0.0;

double angleZ = 0.0;

void impCalibTst(void) {

    Wire.begin();

    byte status = mpu.begin();

    pinMode(LED\_BUILTIN, OUTPUT);

    while(status != 0) {

        digitalWrite(LED\_BUILTIN, HIGH);

        delay(500);

        digitalWrite(LED\_BUILTIN, LOW);

        delay(500);

    }

}

void resetPid(void) {   //used for resettin pid after completed fly

    rollPrevError = 0; pitchPrevError = 0;

    rollITerm = 0; rollDTerm = 0; rollPTerm = 0;

    pitchITerm = 0; pitchDTerm = 0; pitchPTerm = 0;

    rollPid = 0; pitchPid = 0;

}

void resetMotors(void) {  //used to reset motors after flight

    upperLeftMotor.writeMicroseconds(throttleCutOff);

    upperRightMotor.writeMicroseconds(throttleCutOff);

    downLeftMotor.writeMicroseconds(throttleCutOff);

    downRightMotor.writeMicroseconds(throttleCutOff);

}

void anglesRead(void) {

    mpu.update();

    angleX = mpu.getAngleY();  //calculatin inclination angles usin complementary filter

    angleY = mpu.getAngleX(); //changing order cause of mpu placement

    angleZ = mpu.getAngleZ();

}

void pidEquation(void) {

    desiredXAngle = 0;   //pid desired angles of inclination we would normally inputh here values from communication modules

    desiredYAngle = 0;

    desiredZAngle = 0;

    rollError = angleX - desiredXAngle;  //difference between current angle and desired one

    pitchError = angleY - desiredYAngle;

    yawError = angleZ - desiredZAngle;

    rollPTerm = rollPConstant \* rollError; //proportional terms of roll and pitch

    pitchPTerm = pitchPConstant \* pitchError;

    yawPTerm = yawPConstant \* yawError;

    rollITerm = rollITerm + (rollIConstant \* (rollError + rollPrevError) \* (elapsedTime / 2));    //integral term to smooth little error bet. -3deg:3deg

    pitchITerm = pitchITerm + (pitchIConstant \* (pitchError + pitchPrevError) \* (elapsedTime / 2));

    yawITerm = yawITerm + (yawIConstant \* (yawError + yawPrevError) \* (elapsedTime / 2));

    if(rollITerm > 400)   rollITerm = 400;

    else if(rollITerm < -400)   rollITerm = -400;

    if(pitchITerm > 400)   pitchITerm = 400;

    else if(pitchITerm < -400)   pitchITerm = -400;

    if(yawITerm > 400)   yawITerm = 400;

    else if(yawITerm < -400)   yawITerm = -400;

    rollDTerm = rollDConstant \* ((rollError - rollPrevError) / elapsedTime);   //derivative term = DConst\*(dE/dt)

    pitchDTerm = pitchDConstant \* ((pitchError - pitchPrevError) / elapsedTime);

    yawDTerm = yawDConstant \* ((yawError - yawPrevError) / elapsedTime);

    rollPid = rollPTerm + rollITerm + rollDTerm;   //final "calib" values from pid

    pitchPid = pitchPTerm + pitchITerm + pitchDTerm;

    yawPid = yawPTerm + yawITerm + yawDTerm;

    if(rollPid < -400) {

        rollPid = -400;   //security check to not exceed min/max pwm

    }

    if(rollPid > 400) {

        rollPid = 400;

    }

    if(pitchPid < -400) {

        pitchPid = -400;

    }

    if(pitchPid > 400) {

        pitchPid = 400;

    }

    if(yawPid < -400) {

        yawPid = -400;

    }

    if(yawPid > 400) {

        yawPid = 400;

    }

}

void update(int throttleValue) {

    timePrev = time;

    time = millis();    //one loop iteration time used for integration

    elapsedTime = (time - timePrev) / 1000;   //ms -> s : 1 / 1000

    upperRightPwm = gainConst \* (-rollPid - pitchPid - yawPid);   //motors pwm calculation

    downRightPwm = gainConst \* (-rollPid + pitchPid + yawPid);

    downLeftPwm = gainConst \* (rollPid + pitchPid - yawPid);

    upperLeftPwm = gainConst \* (rollPid - pitchPid + yawPid);

    upperLeftPwmFinal = upperLeftPwm + throttleValue;

    downLeftPwmFinal = downLeftPwm + throttleValue;

    upperRightPwmFinal = upperRightPwm + throttleValue;

    downRightPwmFinal = downRightPwm + throttleValue;

    if(upperRightPwmFinal < 1100) {  //security check to not exceed min/max pwm

        upperRightPwmFinal = 1100;

    }

    if(upperRightPwmFinal > 2000) {

        upperRightPwmFinal = 2000;

    }

    if(upperLeftPwmFinal < 1100) {

        upperLeftPwmFinal = 1100;

    }

    if(upperLeftPwmFinal > 2000) {

        upperLeftPwmFinal = 2000;

    }

    if(downRightPwmFinal < 1100)

    {

        downRightPwmFinal = 1100;

    }

    if(downRightPwmFinal > 2000) {

        downRightPwmFinal = 2000;

    }

    if(downLeftPwmFinal < 1100) {

        downLeftPwmFinal = 1100;

    }

    if(downLeftPwmFinal > 2000) {

        downLeftPwmFinal = 2000;

    }

    rollPrevError = rollError; //storin prev errors for roll and pitch

    pitchPrevError = pitchError;

    yawPrevError = yawError;

    upperLeftMotor.writeMicroseconds(upperLeftPwmFinal);

    downLeftMotor.writeMicroseconds(downLeftPwmFinal);

    upperRightMotor.writeMicroseconds(upperRightPwmFinal);

    downRightMotor.writeMicroseconds(downRightPwmFinal);

    Serial.println("UR, DR: " + String(upperRightPwmFinal) + " " + String(downRightPwmFinal) + " UL, DL: " + String(upperLeftPwmFinal) + " " + String(downLeftPwmFinal));

}

void fly(void) {

    if (wasThereFlight == 0) {

        for (int i = throttleCutOff; i < maxFlyable; i++) { //increasing throttle from 1500

            update(i);

        }

        flyTime = millis();

        while (millis() - flyTime < 5000UL) { //fly up until less than 5s

            anglesRead(); //readin angles

            pidEquation();  //readin final pid for roll and pitch angles

            update(maxFlyable);

        }

        rollPid = 0; pitchPid = 0;

        for (int i = maxFlyable; i > minFlyable; i--) { //decreasing throttle

            update(i);

        }

        rollPid = 0; pitchPid = 0;

        flyTime = millis();

        while (millis() - flyTime < 20000UL) { //falling

            anglesRead(); //readin angles

            pidEquation();  //readin final pid for roll and pitch angles

            update(minFlyable);

        }

        rollPid = 0; pitchPid = 0;

        for (int i = minFlyable; i > throttleCutOff; i--) { //slowin motors to no rotation pwm value

            update(i);

        }

        wasThereFlight = 1; //security check

        resetPid();

        resetMotors();

    }

}

void setup() {

    impCalibTst();

    upperLeftMotor.attach(MOTORUL, MINMICROSECONDS, MAXMICROSECONDS); //left front motor

    downLeftMotor.attach(MOTORDL, MINMICROSECONDS, MAXMICROSECONDS); //left back motor

    upperRightMotor.attach(MOTORUR, MINMICROSECONDS, MAXMICROSECONDS); //right front motor

    downRightMotor.attach(MOTORDR, MINMICROSECONDS, MAXMICROSECONDS); //right back motor

    delay(250); //for esc conf

    mpu.calcOffsets();  //gy521 calibration

    time = millis();

    delay(250); //for 1st dt

    Serial.begin(9600);

}

void loop() {

    fly();

}

Tak dobrze widzisz wkleiłem Ci cały kod…

## FUNKCJA IMPCALIBTST()

Zacznijmy go omawiać: na początku definiujemy zmienne do wszystkich funkcji, które określiliśmy sobie w części dotyczącej schematu programu. Pierwsza funkcja:  
void impCalibTst(void) {

Wire.begin();

byte status = mpu.begin();

pinMode(LED\_BUILTIN, OUTPUT);

while(status != 0) {

digitalWrite(LED\_BUILTIN, HIGH);

delay(500);

digitalWrite(LED\_BUILTIN, LOW);

delay(500);

}

}

Rozpoczynamy komunikację przez Wire i patrzymy czy funkcja begin() zwróci 0, jeżeli tak to bardzo dobrze komunikacja się powiodła, jeżeli nie to zamigaj dioda wbudowana (użyto makra LED\_BUILTIN). Przejdźmy dalej…

## FUNKCJA resetPID() oraz resetMotors()

Funkcje resetPid oraz resetMotors omówimy razem ponieważ maja taka sama zasadę działania -> resetują 😉 funkcja resetPid ustawia NAJWAŻNIEJSZE rollPid oraz pitchPid, yawPid na 0. Poza tym daje tez wszystkie inne parametry wykorzystywane podczas algorytmu PID na 0 tak żeby dron NIE MIAŁ PRAWA JUŻ POLECIEC.

void resetPid(void) { //used for resettin pid after completed fly

rollPrevError = 0; pitchPrevError = 0;

rollITerm = 0; rollDTerm = 0; rollPTerm = 0;

pitchITerm = 0; pitchDTerm = 0; pitchPTerm = 0;

rollPid = 0; pitchPid = 0; yawPid = 0;

}

Funkcja resetMotors analogicznie… dajemy wykrywany przez ESC sygnał PWM 1500 odpowiadający informacji: CHEJ WEŹ TE SILNIKI WYŁĄCZ! (czyli 0mm/s). Realizujemy za pomocą funkcji writeMicroseconds(). Da się tez od razu powiem za pomocą analog write ale jest to trudne ponieważ jak wiesz mamy tam do wyboru tylko zakres 0-255 (ciężko jest w tak małym zakresie regulować tak potężne silniki (mała zmiana PWM a ogromne zmiany szybkości)

void resetMotors(void) { //used to reset motors after flight

upperLeftMotor.writeMicroseconds(throttleCutOff);

upperRightMotor.writeMicroseconds(throttleCutOff);

downLeftMotor.writeMicroseconds(throttleCutOff);

downRightMotor.writeMicroseconds(throttleCutOff);

}

## FUNKCJA anglesRead() i Kalibracja MPU6050

Dobra teraz zaczyna się zabawa, spojrzmy na niepozorna funckje aglesRead():  
void anglesRead(void) {

mpu.update();

angleX = mpu.getAngleY(); //calculatin inclination angles usin complementary filter

angleY = mpu.getAngleX(); //changing order cause of mpu placement

angleZ = mpu.getAngleZ();

}

W pierwszym kroku aktualizujemy wskazania z MPU6050 (odczytujemy wartość z rejestru za pomocą funkcji update() następnie za pomocą funkcji katów eulera odczytujemy wartości odpowiednich katów: ROLL (kat obrotu wokół osi X), PITCH (kat obrotu­ wokół osi Y) i YAW (kat obrotu wokół osi Z). W tym przykładzie celowo użyłem biblioteki żeby nie utrudniać zbytnio tej funkcji. Po odczytaniu katów biblioteka już za nas odpowiednio łączy ze sobą katy odczytane za pomocą akcelerometru oraz żyroskopu za pomocą filtru komplementarnego. Jak działa filtr komplementarny? Uproszczę tutaj sprawę jest to kombinacja tkz. LPF i HPF (low pass filter i high pass filter). Tzn. ponieważ wiemy, że na skutek całkowania pojawia się zjawisko gyro drift a na skutek zbyt dużych szumów dla acceleration angles nie możemy brać tylko jednej z tych wartości musimy je jakość sprytnie zsumować tak żeby w rezultacie wynik był jak najbliższy rzeczywistości. Dlatego w praktyce przepuszczamy w większości kąt obliczony od gyro, a w mniejszości przepuszczamy kod od acceleration, tak tylko żeby korygować wartości kątów z gyro (tzn. dajemy HPF do gyro i LPF do accelerometer), w rezultacie mamy: 0.99 \* gyroAngle + 0.01 \* accAngle. Wkleję tutaj jeszcze kod starej funkcji do odczytu katów (używający wlasnie tego algorytmu, tyle że w postaci jawnej:

void anglesRead(void) {

Wire.beginTransmission(0x68); //startin communication for gyro

Wire.write(0x43); //gyro register

Wire.endTransmission(false);

Wire.requestFrom(0x68, 4, true);

rateY = Wire.read() << 8 | Wire.read(); //readin y first beacause of mpu placement on drone

rateX = Wire.read() << 8 | Wire.read();

rateX = (rateX / 32.8) - rateXError; //32.8 corresponding 1000dps value

rateY = (rateY / 32.8) - rateYError; //convertin to deg/s

angleXRate = rateX \* elapsedTime; //takin discrete for of integral for angle

angleYRate = rateY \* elapsedTime;

Wire.beginTransmission(0x68); //startin communication for acc

Wire.write(0x3B); //acc register

Wire.endTransmission(false);

Wire.requestFrom(0x68, 6, true);

accY = (Wire.read() << 8 | Wire.read()) / 4096.0; //readin y first beacause of mpu placement on drone

accX = (Wire.read() << 8 | Wire.read()) / 4096.0; //converting to m/s^2

accZ = (Wire.read() << 8 | Wire.read()) / 4096.0;

angleXAcc = (atan((accY) / sqrt(pow((accX), 2) + pow((accZ), 2))) \* radToDeg) - accAngleXError;

angleYAcc = (atan(-1 \* (accX) / sqrt(pow((accY), 2) + pow((accZ), 2))) \* radToDeg) - accAngleYError;

angleX = 0.99 \* (angleX + angleXRate) + 0.01 \* angleXAcc; //calculatin inclination angles usin complementary filter

angleY = 0.99 \* (angleY + angleYRate) + 0.01 \* angleYAcc;

}

Na temat katów eulera tez mógłbym się rozpisywać bardzo długo, ale nie jest to konieczne, my po prostu używamy wzorów tylko i aż tylko.

Jeżeli chcesz poczytać więcej dobrze jest opisane to zagadnienie na stronie anglosaskiej wiki: [Euler angles - Wikipedia](https://en.wikipedia.org/wiki/Euler_angles)

Nasze IMU (MPU6050) musimy jeszcze jakoś skalibrować chcemy w końcu, żeby przy starcie kąt roll, pitch, yaw były równe zero by mieć prawidłowy punkt odniesienia. Jak to zrobić? SKALIBROWAĆ 😊 Służy do tego funkcja także z libki MPU6050\_light.h o nazwie:  
mpu.calcOffsets(). Zczytujemy od 1000 – 4000 pomiarów wartości kątowej oraz wartości przyspieszenia i limy z nich średnia. Po to by na końcu odejmować ta wartość początkowego odchyłu od pozostałych wartości tak by w rezultacie mieć 0 na początku, starzy kod, który robił to w sposób jawny (oczywiście odczytów z rejestrów nie będę tłumaczył nie ma to najmniejszego sensu, jest to po prostu sposób komunikacji z większości układów elektronicznych):

void calibMpu(void) {

Wire.begin(); //start wire comunication

Wire.beginTransmission(0x68); //start communication with gy521

Wire.write(0x6B); //reset current config

Wire.write(0x00);

Wire.endTransmission(true); //end transmission

Wire.beginTransmission(0x68); //start communication for gyro config

Wire.write(0x1B); //gyro register

Wire.write(0x10); //settin gyro register (1000dps scale)

Wire.endTransmission(true);

Wire.beginTransmission(0x68); //start communication for acc config

Wire.write(0x1C); //acc register

Wire.write(0x10); //settin acc register (8g scale)

Wire.endTransmission(true);

if(accCalib == 0) { //calibration for accelerometer

for(int a = 0; a < 2000; a++) {

Wire.beginTransmission(0x68);

Wire.write(0x3B); //acc register address

Wire.endTransmission(false);

Wire.requestFrom(0x68, 6, true); //16b for 1 info about acc

accY = (Wire.read() << 8 | Wire.read()) / 4096.0;

accX = (Wire.read() << 8 | Wire.read()) / 4096.0; //1 reister address = 8b = 2B

accZ = (Wire.read() << 8 | Wire.read()) / 4096.0;

accAngleXError = accAngleXError + ((atan((accY) / sqrt(pow((accX), 2) + pow((accZ), 2))) \* radToDeg)); //calculatin sum of acc angles

accAngleYError = accAngleYError + ((atan(-1 \* (accX) / sqrt(pow((accY), 2) + pow((accZ), 2))) \* radToDeg));

}

accAngleXError = accAngleXError / 2000;

accAngleYError = accAngleYError / 2000;

accCalib = 1;

}

if(rateCalib == 0) { //calibration for gyro

for(int i = 0; i < 2000; i++) {

Wire.beginTransmission(0x68);

Wire.write(0x43); //gyro registr address

Wire.endTransmission(false);

Wire.requestFrom(0x68, 4, true); //16b for 1 info about gyro, total of 4 \* 8 = 32b here

rateY = Wire.read() << 8 | Wire.read();

rateX = Wire.read() << 8 | Wire.read();

rateXError = rateXError + (rateX / 32.8); //calculatin sum of rates

rateYError = rateYError + (rateY / 32.8);

}

rateXError = rateXError / 2000;

rateYError = rateYError / 2000;

rateCalib = 1;

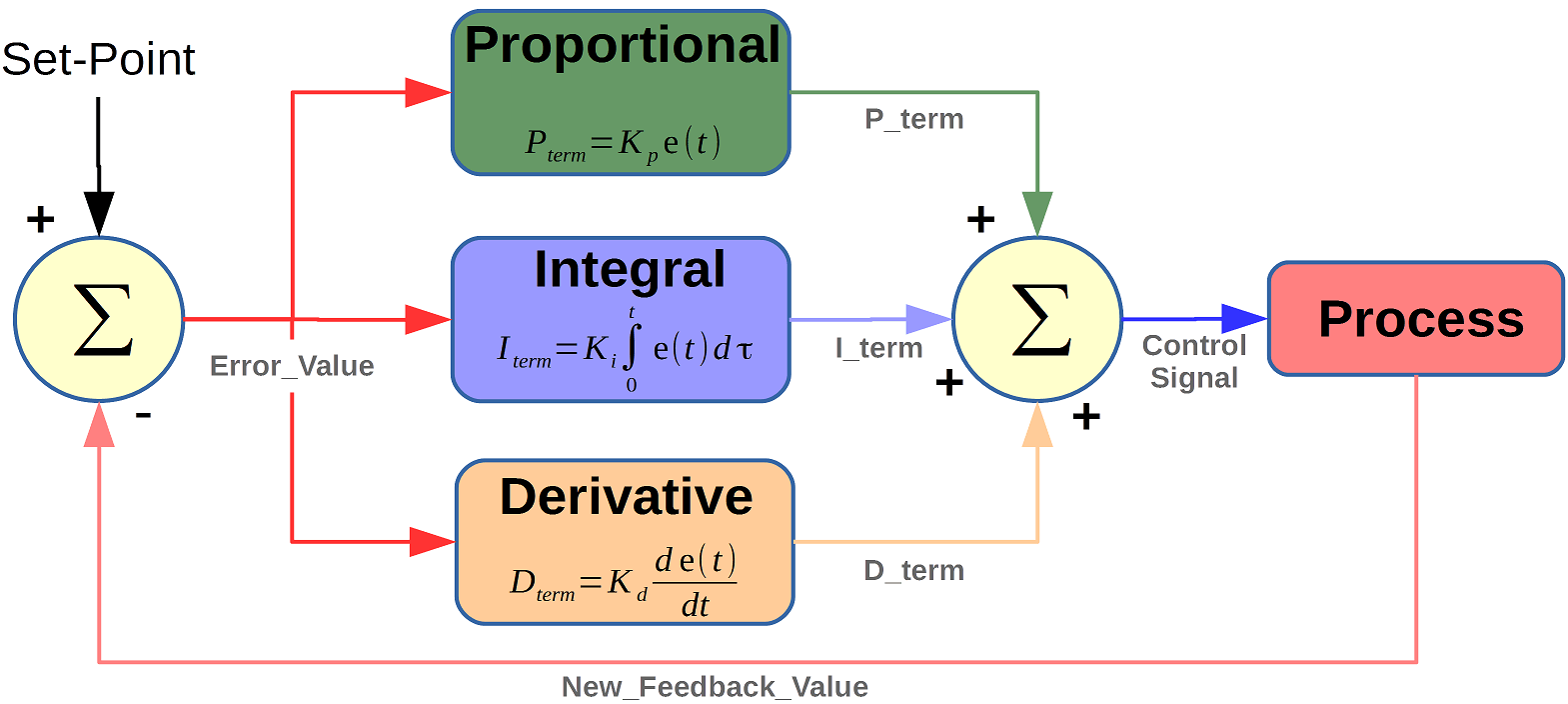
}

}

## FUNKCJA pidEquation()

Kolejna ciężka rzecz to PID… Teraz tak omówmy działanie tego algorytmu jedynie dla wartości ROLL angle, bo dla pitch i yaw jest to analogiczne. do prawidłowego działania pid potrzebuje wartość błędu miedzy wartością docelowa pewnej wartości która regulujemy a wartością aktualna. Żeby ta różnice uzyskać, dajemy rollError = angleX - desiredXAngle; W samym algorytmie mamy 3 części: cześć proporcjonalna, całkująca i różniczkująca które operują wlasnie na tym błędzie, przy czym za pomocą części proporcjonalnej liczmy wartość błędu aktualnego, różnicującej – części przyszłego i całkującej błędu przeszłego. Kombinacja tych 3 części daje nam bardzo dobre przybliżenie co dzieje się obecnie z regulowanym obiektem i jaki sygnał powinniśmy mu dać by odpowiedni wyregulować go do wartości oczekiwanej (setpoint). Potrzebujemy także odstępu czasu który liczmy odejmując od aktualnej wartości czasu zmierzonej za pomocą millis(), wartość poprzednia zmierzona w zmiennej time. Mamy więc wzory:

* rollPTerm = rollPConstant \* rollError
* rollITerm = rollITerm + (rollIConstant \* (rollError + rollPrevError) \* (elapsedTime / 2));
* rollDTerm = rollDConstant \* ((rollError - rollPrevError) / elapsedTime);

Cały schemat działania PID świetnie ilustruje tabela:

Kodzik:

void pidEquation(void) {

desiredXAngle = 0; //pid desired angles of inclination we would normally inputh here values from communication modules

desiredYAngle = 0;

desiredZAngle = 0;

rollError = angleX - desiredXAngle; //difference between current angle and desired one

pitchError = angleY - desiredYAngle;

yawError = angleZ - desiredZAngle;

rollPTerm = rollPConstant \* rollError; //proportional terms of roll and pitch

pitchPTerm = pitchPConstant \* pitchError;

yawPTerm = yawPConstant \* yawError;

rollITerm = rollITerm + (rollIConstant \* (rollError + rollPrevError) \* (elapsedTime / 2)); //integral term to smooth little error bet. -3deg:3deg

pitchITerm = pitchITerm + (pitchIConstant \* (pitchError + pitchPrevError) \* (elapsedTime / 2));

yawITerm = yawITerm + (yawIConstant \* (yawError + yawPrevError) \* (elapsedTime / 2));

if(rollITerm > 400) rollITerm = 400;

else if(rollITerm < -400) rollITerm = -400;

if(pitchITerm > 400) pitchITerm = 400;

else if(pitchITerm < -400) pitchITerm = -400;

if(yawITerm > 400) yawITerm = 400;

else if(yawITerm < -400) yawITerm = -400;

rollDTerm = rollDConstant \* ((rollError - rollPrevError) / elapsedTime); //derivative term = DConst\*(dE/dt)

pitchDTerm = pitchDConstant \* ((pitchError - pitchPrevError) / elapsedTime);

yawDTerm = yawDConstant \* ((yawError - yawPrevError) / elapsedTime);

rollPid = rollPTerm + rollITerm + rollDTerm; //final "calib" values from pid

pitchPid = pitchPTerm + pitchITerm + pitchDTerm;

yawPid = yawPTerm + yawITerm + yawDTerm;

if(rollPid < -400) {

rollPid = -400; //security check to not exceed min/max pwm

}

if(rollPid > 400) {

rollPid = 400;

}

if(pitchPid < -400) {

pitchPid = -400;

}

if(pitchPid > 400) {

pitchPid = 400;

}

if(yawPid < -400) {

yawPid = -400;

}

if(yawPid > 400) {

yawPid = 400;

}

}

Co jeszcze bardzo istotne to to żebyśmy zabezpieczyli drona przed tkz. zjawiskiem integral windup związanym z kumulowanie przeszłych błędów. Stąd dajemy ograniczenie: if(rollITerm > 400) rollITerm = 400; else if(rollITerm < -400) rollITerm = -400; No I też nie chcemy by wartości końcowe regulacji rollPid przekraczały zbyt duże wartości (to ustawiamy arbitralnie, poprzez doświadczenia). Natomiast stale P, I, D używane w procesie (u nas:

float rollPConstant = 0.7f; //3.55

float rollIConstant = 0.006f; //0.003

float rollDConstant = 1.2f; //2.05

) Wyznaczamy w sposób doświadczalny tak żeby dron nie wariował, u nas się to jeszcze nie udało :/ (służy do tego m.in. metoda Zieglera-Nicholsa, Cohena lub metoda lambda)

## FUNKCJe update() i fly()

Funkcje update() i fly() służą do kontroli lotu stricte dlatego omówimy je razem. Funkcja update oblicza odpowiednie wartości sygnałów podawanych na odpowiednie silniki te wartości obliczane są według ścisłych wzorów, tak żeby wszystko działało, np. dla prawego górnego silnika mamy: upperRightPwm = gainConst \* (-rollPid - pitchPid - yawPid); Gdzie gain const to dodatkowa stała wzmocnienia używana gdy wartość sygnału jest za słaba w stosunku do oczekiwań gdy dron zbyt słabo reaguje na wartości zmiany kata inklinacji. Od 1 do 4 doświadczalnie u nas przyjęta. Później sprawdzamy czy wypadkowa wartość, którą teraz mamy jako np.: upperRightPwmFinal = upperRightPwm + throttleValue; tzn. wartość, która chcemy uzyskać + wartość regulacji z Pid, mieści się w wymaganym zakresie od 1100 do 2000 tylko na takie wartości reaguje nasze ESC. Na końcu przypisujemy np. upperRightPwmFinal do motora upperRight za pomocą writeMicroseconds().

void update(int throttleValue) {

timePrev = time;

time = millis(); //one loop iteration time used for integration

elapsedTime = (time - timePrev) / 1000; //ms -> s : 1 / 1000

upperRightPwm = gainConst \* (-rollPid - pitchPid - yawPid); //motors pwm calculation

downRightPwm = gainConst \* (-rollPid + pitchPid + yawPid);

downLeftPwm = gainConst \* (rollPid + pitchPid - yawPid);

upperLeftPwm = gainConst \* (rollPid - pitchPid + yawPid);

upperLeftPwmFinal = upperLeftPwm + throttleValue;

downLeftPwmFinal = downLeftPwm + throttleValue;

upperRightPwmFinal = upperRightPwm + throttleValue;

downRightPwmFinal = downRightPwm + throttleValue;

if(upperRightPwmFinal < 1100) { //security check to not exceed min/max pwm

upperRightPwmFinal = 1100;

}

if(upperRightPwmFinal > 2000) {

upperRightPwmFinal = 2000;

}

if(upperLeftPwmFinal < 1100) {

upperLeftPwmFinal = 1100;

}

if(upperLeftPwmFinal > 2000) {

upperLeftPwmFinal = 2000;

}

if(downRightPwmFinal < 1100)

{

downRightPwmFinal = 1100;

}

if(downRightPwmFinal > 2000) {

downRightPwmFinal = 2000;

}

if(downLeftPwmFinal < 1100) {

downLeftPwmFinal = 1100;

}

if(downLeftPwmFinal > 2000) {

downLeftPwmFinal = 2000;

}

rollPrevError = rollError; //storin prev errors for roll and pitch

pitchPrevError = pitchError;

yawPrevError = yawError;

upperLeftMotor.writeMicroseconds(upperLeftPwmFinal);

downLeftMotor.writeMicroseconds(downLeftPwmFinal);

upperRightMotor.writeMicroseconds(upperRightPwmFinal);

downRightMotor.writeMicroseconds(downRightPwmFinal);

Serial.println("UR, DR: " + String(upperRightPwmFinal) + " " + String(downRightPwmFinal) + " UL, DL: " + String(upperLeftPwmFinal) + " " + String(downLeftPwmFinal));

}

Funkcja fly() służy już tylko i wyłącznie do obsługi lotu, tzn. chcemy zrealizować zadanie lec w górę na obrotach danych przez wartość 1800us = maxFlyable przez 5 sekund, zacznij zwalniać do 1650 (żeby zaczął opadać) i tak przez 20 sekund utrzymuj tą wartość żeby dron opadł na ziemie. W miedzy czasie reguluj wartość kata inklinacji drona za pomocą PID oraz anglesRead() który będzie mu dostarczał aktualne wartości katów. Na końcu ustaw zmienna kontrolna na 1 żeby dron już więcej nie poleciał, zresetuj pid i motory (żeby mieć dodatkowa pewność).

void fly(void) {

if (wasThereFlight == 0) {

for (int i = throttleCutOff; i < maxFlyable; i++) { //increasing throttle from 1500

update(i);

}

flyTime = millis();

while (millis() - flyTime < 5000UL) { //fly up until less than 5s

anglesRead(); //readin angles

pidEquation(); //readin final pid for roll and pitch angles

update(maxFlyable);

}

rollPid = 0; pitchPid = 0;

for (int i = maxFlyable; i > minFlyable; i--) { //decreasing throttle

update(i);

}

rollPid = 0; pitchPid = 0;

flyTime = millis();

while (millis() - flyTime < 20000UL) { //falling

anglesRead(); //readin angles

pidEquation(); //readin final pid for roll and pitch angles

update(minFlyable);

}

rollPid = 0; pitchPid = 0;

for (int i = minFlyable; i > throttleCutOff; i--) { //slowin motors to no rotation pwm value

update(i);

}

wasThereFlight = 1; //security check

resetPid();

resetMotors();

}

}

## FUNKCJA setup() i Loop()

I to by było na tyle jeszcze może komentarza wymagałoby co dzieje się funkcji setup()? Jeżeli tak to po prostu dołączamy tam silniki BLDC zgodnie ze standardem nadając wartości krańcowe dla ESC 1000 oraz 2000. (taki jest schemat po prostu). Oprócz tego kalibrujemy MPU6050 tak jak to opisaliśmy powyżej oraz sprawdzamy czy wgl. jest podłączone nasze MPU jak nie to migamy diodka. Jak wszystko jest git to przechodzimy do loop() która wywołuje fly(), która wywołuje anglesRead(), pidEquation(), update() i swoje działania realizuje i tak w kółeczko 😊

# kod Flight controllera (STM32F103C8T6, rev.4)

Kod jest odpowiedzialny za sterowanie dronem, w tym za konfigurację i kalibrację żyroskopu, obsługę sygnałów z odbiornika, implementację regulatorów PID oraz kontrolę diod LED sygnalizujących stan drona. Zaczniemy podobnie co poprzednio także wkleję Ci cały kod, a później przejdę do szczegółowego objaśniania go.

#include <Wire.h>

TwoWire Wire2(2, I2C\_FAST\_MODE); // inicjalizacja I2C2 (piny B10 - SCL, B11 - SDA (domyslnie wire obsluguje I2C1)

// ustawienia pid dla roll

float pid\_p\_gain\_roll = 1.3;

float pid\_i\_gain\_roll = 0.04;

float pid\_d\_gain\_roll = 18.0;

int pid\_max\_roll = 400;

// ustawienia pid dla pitch

float pid\_p\_gain\_pitch = pid\_p\_gain\_roll;

float pid\_i\_gain\_pitch = pid\_i\_gain\_roll;

float pid\_d\_gain\_pitch = pid\_d\_gain\_roll;

int pid\_max\_pitch = pid\_max\_roll;

// ustawienia pid dla yaw

float pid\_p\_gain\_yaw = 4.0;

float pid\_i\_gain\_yaw = 0.02;

float pid\_d\_gain\_yaw = 0.0;

int pid\_max\_yaw = 400;

// odpowiada za wlaczenie kontroli poziomu do drona

boolean auto\_level = true;

float pid\_error\_temp;

float pid\_i\_mem\_roll, pid\_roll\_setpoint, gyro\_roll\_input, pid\_output\_roll, pid\_last\_roll\_d\_error;

float pid\_i\_mem\_pitch, pid\_pitch\_setpoint, gyro\_pitch\_input, pid\_output\_pitch, pid\_last\_pitch\_d\_error;

float pid\_i\_mem\_yaw, pid\_yaw\_setpoint, gyro\_yaw\_input, pid\_output\_yaw, pid\_last\_yaw\_d\_error;

//do standaryzacji setpointu dla pidu

float roll\_level\_adjust, pitch\_level\_adjust;

// ustawienie wartosci kalibracyjnych manualnych dla temp ~25\*C

uint8\_t use\_manual\_calibration = true;

int16\_t manual\_acc\_pitch\_cal\_value = 115;

int16\_t manual\_acc\_roll\_cal\_value = -78;

int16\_t manual\_gyro\_pitch\_cal\_value = -212;

int16\_t manual\_gyro\_roll\_cal\_value = 504;

int16\_t manual\_gyro\_yaw\_cal\_value = -45;

// adres gy521

uint8\_t gyro\_address = 0x68;

// zmienne  kalibracyjne oraz do acc i gyro

int16\_t temperature, count\_var;

int16\_t acc\_x, acc\_y, acc\_z;

int16\_t gyro\_pitch, gyro\_roll, gyro\_yaw;

int32\_t acc\_total\_vector;

int32\_t gyro\_roll\_cal, gyro\_pitch\_cal, gyro\_yaw\_cal;

float angle\_roll\_acc, angle\_pitch\_acc, angle\_pitch, angle\_roll;

// zmienne pomocnicze do sygnalizacji stanow drona

uint8\_t last\_channel\_1, last\_channel\_2, last\_channel\_3, last\_channel\_4;

uint8\_t highByte, lowByte, start;

uint8\_t error, error\_counter, error\_led;

float battery\_voltage;

// pwm do esc i zmienne do motors

int16\_t esc\_1, esc\_2, esc\_3, esc\_4;

int16\_t throttle, cal\_int;

// zmienne do receivera

int32\_t channel\_1\_start, channel\_1;

int32\_t channel\_2\_start, channel\_2;

int32\_t channel\_3\_start, channel\_3;

int32\_t channel\_4\_start, channel\_4;

int32\_t channel\_5\_start, channel\_5;

int32\_t channel\_6\_start, channel\_6;

// do kontrolowania czasu obiegu kazdej iteracji

uint32\_t loop\_timer, error\_timer;

// do ustawienia stanu na diodzie (sygnalizacja stanu drona) DO IMPL !!!NA PLYTCE!!!

void red\_led(int8\_t level) {

  digitalWrite(PB4, level);

}

void green\_led(int8\_t level) {

  digitalWrite(PB3, level);

}

// funkcja pid dzialajaca w przerwaniach

void calculate\_pid(void){

  // obliczenie dla czesci integrujacej, prop, roznicz roll

  pid\_error\_temp = gyro\_roll\_input - pid\_roll\_setpoint;

  pid\_i\_mem\_roll += pid\_i\_gain\_roll \* pid\_error\_temp;

  if(pid\_i\_mem\_roll > pid\_max\_roll)

    pid\_i\_mem\_roll = pid\_max\_roll;

  else if(pid\_i\_mem\_roll < pid\_max\_roll \* -1)

    pid\_i\_mem\_roll = pid\_max\_roll \* -1;

  // output dla roll KpEp + KiEi + KdEd

  pid\_output\_roll = pid\_p\_gain\_roll \* pid\_error\_temp + pid\_i\_mem\_roll + pid\_d\_gain\_roll \* (pid\_error\_temp - pid\_last\_roll\_d\_error);

  if(pid\_output\_roll > pid\_max\_roll)

    pid\_output\_roll = pid\_max\_roll;

  else if(pid\_output\_roll < pid\_max\_roll \* -1)

    pid\_output\_roll = pid\_max\_roll \* -1;

  pid\_last\_roll\_d\_error = pid\_error\_temp;

  // obliczenie dla czesci integrujacej, prop, roznicz pitch

  pid\_error\_temp = gyro\_pitch\_input - pid\_pitch\_setpoint;

  pid\_i\_mem\_pitch += pid\_i\_gain\_pitch \* pid\_error\_temp;

  if(pid\_i\_mem\_pitch > pid\_max\_pitch)

    pid\_i\_mem\_pitch = pid\_max\_pitch;

  else if(pid\_i\_mem\_pitch < pid\_max\_pitch \* -1)

    pid\_i\_mem\_pitch = pid\_max\_pitch \* -1;

  // output dla pitch KpEp + KiEi + KdEd

  pid\_output\_pitch = pid\_p\_gain\_pitch \* pid\_error\_temp + pid\_i\_mem\_pitch + pid\_d\_gain\_pitch \* (pid\_error\_temp - pid\_last\_pitch\_d\_error);

  if(pid\_output\_pitch > pid\_max\_pitch)

    pid\_output\_pitch = pid\_max\_pitch;

  else if(pid\_output\_pitch < pid\_max\_pitch \* -1)

    pid\_output\_pitch = pid\_max\_pitch \* -1;

  pid\_last\_pitch\_d\_error = pid\_error\_temp;

  // obliczenie dla czesci integrujacej, prop, roznicz yaw

  pid\_error\_temp = gyro\_yaw\_input - pid\_yaw\_setpoint;

  pid\_i\_mem\_yaw += pid\_i\_gain\_yaw \* pid\_error\_temp;

  if(pid\_i\_mem\_yaw > pid\_max\_yaw)

    pid\_i\_mem\_yaw = pid\_max\_yaw;

  else if(pid\_i\_mem\_yaw < pid\_max\_yaw \* -1)

    pid\_i\_mem\_yaw = pid\_max\_yaw \* -1;

  // output dla yaw KpEp + KiEi + KdEd

  pid\_output\_yaw = pid\_p\_gain\_yaw \* pid\_error\_temp + pid\_i\_mem\_yaw + pid\_d\_gain\_yaw \* (pid\_error\_temp - pid\_last\_yaw\_d\_error);

  if(pid\_output\_yaw > pid\_max\_yaw)pid\_output\_yaw = pid\_max\_yaw;

  else if(pid\_output\_yaw < pid\_max\_yaw \* -1)pid\_output\_yaw = pid\_max\_yaw \* -1;

  pid\_last\_yaw\_d\_error = pid\_error\_temp;

}

// funkcja pid dzialajaca w przerwaniach

void calibrate\_gyro(void) {

  // przeskakujemy kalibracje gdy mamy wartosci doswiadczalne ;)

  if (use\_manual\_calibration)

    cal\_int = 2000;

  else {

    cal\_int = 0;

    manual\_gyro\_pitch\_cal\_value = 0;

    manual\_gyro\_roll\_cal\_value = 0;

    manual\_gyro\_yaw\_cal\_value = 0;

  }

  //jezeli ustawilismy kalib automatyczna (podstawy drona musza byc !!!IDEALNIE ROWNE!!! -> do poprawy)

  if (cal\_int != 2000) {

    //probkujemy (2000d) dane z gyro

    for (cal\_int = 0; cal\_int < 2000 ; cal\_int ++) {

      if (cal\_int % 25 == 0)

        digitalWrite(PB4, !digitalRead(PB4));

      gyro\_signals();

      gyro\_roll\_cal += gyro\_roll;

      gyro\_pitch\_cal += gyro\_pitch;

      gyro\_yaw\_cal += gyro\_yaw;

      delay(4);

    }

    red\_led(HIGH);

    //bierzemy srednia artmetyczna z danych w celu uzyskania wart z kalibracji AVR be like

    gyro\_roll\_cal /= 2000;

    gyro\_pitch\_cal /= 2000;

    gyro\_yaw\_cal /= 2000;

    manual\_gyro\_pitch\_cal\_value = gyro\_pitch\_cal;

    manual\_gyro\_roll\_cal\_value = gyro\_roll\_cal;

    manual\_gyro\_yaw\_cal\_value = gyro\_yaw\_cal;

  }

}

// funkcja do pokazania sygnalu bledu

void error\_signal(void){

  if (error >= 100)

    red\_led(HIGH);

  //sprawdzenie czy minelo >= 250ms i stanu error'ow

  else if (error\_timer < millis()){

    error\_timer = millis() + 250;

    //do kodow bledow kazdy blad ma inny kodzik ;)

    if(error > 0 && error\_counter > error + 3)

      error\_counter = 0;

    if (error\_counter < error && error\_led == 0 && error > 0){

      red\_led(HIGH);

      error\_led = 1;

    }

    // wylaczamy diode i zwiekszamy counter bledow o 1

    else{

      red\_led(LOW);

      error\_counter++;

      error\_led = 0;

    }

  }

}

// funkcja do nawiazywania polaczenia z gy521

void gyro\_setup(void){

  // nawiazujemy polaczenie z gy521 i je aktywujemy

  Wire2.beginTransmission(gyro\_address);

  Wire2.write(0x6B);

  Wire2.write(0x00);

  Wire2.endTransmission();

  // ustawiamy skale 500dps

  Wire2.beginTransmission(gyro\_address);

  Wire2.write(0x1B);

  Wire2.write(0x08);

  Wire2.endTransmission();

  // ustawiamy skale (+/-)8g

  Wire2.beginTransmission(gyro\_address);

  Wire2.write(0x1C);

  Wire2.write(0x10);

  Wire2.endTransmission();

  // ustawiamy lpf w celu wyeliminowania zaklocen wysokiego pasma (tzn. ok. 42Hz)

  Wire2.beginTransmission(gyro\_address);

  Wire2.write(0x1A);

  Wire2.write(0x03);

  Wire2.endTransmission();

}

//funkcja do obslugi receivera (channel 1)

void handler\_channel\_1(void) {

  //jezeli stan na A0 jest HIGH -> zczytujemy czas RISE\_EDGE

  if (0b1 & GPIOA\_BASE->IDR  >> 0) {

    channel\_1\_start = TIMER2\_BASE->CCR1;

    TIMER2\_BASE->CCER |= TIMER\_CCER\_CC1P;

  }

  //obliczamy calkowita dlugosc trwania pulsu RISE\_EDGE-FALL\_EDGE

  else {

    channel\_1 = TIMER2\_BASE->CCR1 - channel\_1\_start;

    if (channel\_1 < 0)

      channel\_1 += 0xFFFF;

    TIMER2\_BASE->CCER &= ~TIMER\_CCER\_CC1P;

  }

}

//funkcja do obslugi receivera (channel 2)

void handler\_channel\_2(void) {

  //jezeli stan na A1 jest HIGH -> zczytujemy czas RISE\_EDGE

  if (0b1 & GPIOA\_BASE->IDR >> 1) {

    channel\_2\_start = TIMER2\_BASE->CCR2;

    TIMER2\_BASE->CCER |= TIMER\_CCER\_CC2P;

  }

  //obliczamy calkowita dlugosc trwania pulsu RISE\_EDGE-FALL\_EDGE

  else {

    channel\_2 = TIMER2\_BASE->CCR2 - channel\_2\_start;

    if (channel\_2 < 0)

      channel\_2 += 0xFFFF;

    TIMER2\_BASE->CCER &= ~TIMER\_CCER\_CC2P;

  }

}

//funkcja do obslugi receivera (channel 3)

void handler\_channel\_3(void) {

  //jezeli stan na A2 jest HIGH -> zczytujemy czas RISE\_EDGE

  if (0b1 & GPIOA\_BASE->IDR >> 2) {

    channel\_3\_start = TIMER2\_BASE->CCR3;

    TIMER2\_BASE->CCER |= TIMER\_CCER\_CC3P;

  }

  //obliczamy calkowita dlugosc trwania pulsu RISE\_EDGE-FALL\_EDGE

  else {

    channel\_3 = TIMER2\_BASE->CCR3 - channel\_3\_start;

    if (channel\_3 < 0)

      channel\_3 += 0xFFFF;

    TIMER2\_BASE->CCER &= ~TIMER\_CCER\_CC3P;

  }

}

//funkcja do obslugi receivera (channel 4)

void handler\_channel\_4(void) {

  //jezeli stan na A3 jest HIGH -> zczytujemy czas RISE\_EDGE

  if (0b1 & GPIOA\_BASE->IDR >> 3) {

    channel\_4\_start = TIMER2\_BASE->CCR4;

    TIMER2\_BASE->CCER |= TIMER\_CCER\_CC4P;

  }

  //obliczamy calkowita dlugosc trwania pulsu RISE\_EDGE-FALL\_EDGE

  else {

    channel\_4 = TIMER2\_BASE->CCR4 - channel\_4\_start;

    if (channel\_4 < 0)

      channel\_4 += 0xFFFF;

    TIMER2\_BASE->CCER &= ~TIMER\_CCER\_CC4P;

  }

}

//funkcja do obslugi receivera (channel 5)

void handler\_channel\_5(void) {

  //jezeli stan na A4 jest HIGH -> zczytujemy czas RISE\_EDGE

  if (0b1 & GPIOA\_BASE->IDR >> 6) {

    channel\_5\_start = TIMER3\_BASE->CCR1;

    TIMER3\_BASE->CCER |= TIMER\_CCER\_CC1P;

  }

  //obliczamy calkowita dlugosc trwania pulsu RISE\_EDGE-FALL\_EDGE

  else {

    channel\_5 = TIMER3\_BASE->CCR1 - channel\_5\_start;

    if (channel\_5 < 0)

      channel\_5 += 0xFFFF;

    TIMER3\_BASE->CCER &= ~TIMER\_CCER\_CC1P;

  }

}

//funkcja do obslugi receivera (channel 6)

void handler\_channel\_6(void) {

  //jezeli stan na A5 jest HIGH -> zczytujemy czas RISE\_EDGE

  if (0b1 & GPIOA\_BASE->IDR >> 7) {

    channel\_6\_start = TIMER3\_BASE->CCR2;

    TIMER3\_BASE->CCER |= TIMER\_CCER\_CC2P;

  }

  //obliczamy calkowita dlugosc trwania pulsu RISE\_EDGE-FALL\_EDGE

  else {

    channel\_6 = TIMER3\_BASE->CCR2 - channel\_6\_start;

    if (channel\_6 < 0)

      channel\_6 += 0xFFFF;

    TIMER3\_BASE->CCER &= ~TIMER\_CCER\_CC2P;

  }

}

// odczyt gyro i acc z mpu6050 (gy521)

void gyro\_signals(void) {

  //ropoczynamy komunikacje a pozniej zaczynamy czytac...

  Wire2.beginTransmission(gyro\_address);

  Wire2.write(0x3B);

  Wire2.endTransmission();

  Wire2.requestFrom(gyro\_address, 14);

  //shiftujemy w lewo bity zeby starczylo na kolejne dane

  acc\_y = Wire2.read() << 8 | Wire2.read();

  acc\_x = Wire2.read() << 8 | Wire2.read();

  acc\_z = Wire2.read() << 8 | Wire2.read();

  temperature = Wire2.read() << 8 | Wire2.read();

  gyro\_roll = Wire2.read() << 8 | Wire2.read();

  gyro\_pitch = Wire2.read() << 8 | Wire2.read();

  gyro\_yaw = Wire2.read() << 8 | Wire2.read();

  gyro\_pitch \*= -1;

  gyro\_yaw \*= -1;

  //odejmujemy od wartosci odczytaj wartosc manualnej (gdy mamy wlaczona) lub automatyczna (przypisana do manulanej ;))

  acc\_y -= manual\_acc\_pitch\_cal\_value;

  acc\_x -= manual\_acc\_roll\_cal\_value;

  gyro\_roll -= manual\_gyro\_roll\_cal\_value;

  gyro\_pitch -= manual\_gyro\_pitch\_cal\_value;

  gyro\_yaw -= manual\_gyro\_yaw\_cal\_value;

}

// timer do obslugi przerwan

void timer\_setup(void) {

  // przerwania do handlera 1, 2, 3, 4

  Timer2.attachCompare1Interrupt(handler\_channel\_1);

  Timer2.attachCompare2Interrupt(handler\_channel\_2);

  Timer2.attachCompare3Interrupt(handler\_channel\_3);

  Timer2.attachCompare4Interrupt(handler\_channel\_4);

  TIMER2\_BASE->CR1 = TIMER\_CR1\_CEN;

  TIMER2\_BASE->CR2 = 0;

  TIMER2\_BASE->SMCR = 0;

  TIMER2\_BASE->DIER = TIMER\_DIER\_CC1IE | TIMER\_DIER\_CC2IE | TIMER\_DIER\_CC3IE | TIMER\_DIER\_CC4IE;

  TIMER2\_BASE->EGR = 0;

  TIMER2\_BASE->CCMR1 = 0b100000001;

  TIMER2\_BASE->CCMR2 = 0b100000001;

  TIMER2\_BASE->CCER = TIMER\_CCER\_CC1E | TIMER\_CCER\_CC2E | TIMER\_CCER\_CC3E | TIMER\_CCER\_CC4E;

  TIMER2\_BASE->PSC = 71;

  TIMER2\_BASE->ARR = 0xFFFF;

  TIMER2\_BASE->DCR = 0;

  // przerwania do handlera 5, 6

  Timer3.attachCompare1Interrupt(handler\_channel\_5);

  Timer3.attachCompare2Interrupt(handler\_channel\_6);

  TIMER3\_BASE->CR1 = TIMER\_CR1\_CEN;

  TIMER3\_BASE->CR2 = 0;

  TIMER3\_BASE->SMCR = 0;

  TIMER3\_BASE->DIER = TIMER\_DIER\_CC1IE | TIMER\_DIER\_CC2IE;

  TIMER3\_BASE->EGR = 0;

  TIMER3\_BASE->CCMR1 = 0b100000001;

  TIMER3\_BASE->CCMR2 = 0;

  TIMER3\_BASE->CCER = TIMER\_CCER\_CC1E | TIMER\_CCER\_CC2E;

  TIMER3\_BASE->PSC = 71;

  TIMER3\_BASE->ARR = 0xFFFF;

  TIMER3\_BASE->DCR = 0;

  //wlaczenie timera 4 i dla niego przerwan

  TIMER4\_BASE->CR1 = TIMER\_CR1\_CEN | TIMER\_CR1\_ARPE;

  TIMER4\_BASE->CR2 = 0;

  TIMER4\_BASE->SMCR = 0;

  TIMER4\_BASE->DIER = 0;

  TIMER4\_BASE->EGR = 0;

  TIMER4\_BASE->CCMR1 = (0b110 << 4) | TIMER\_CCMR1\_OC1PE |(0b110 << 12) | TIMER\_CCMR1\_OC2PE;

  TIMER4\_BASE->CCMR2 = (0b110 << 4) | TIMER\_CCMR2\_OC3PE |(0b110 << 12) | TIMER\_CCMR2\_OC4PE;

  TIMER4\_BASE->CCER = TIMER\_CCER\_CC1E | TIMER\_CCER\_CC2E | TIMER\_CCER\_CC3E | TIMER\_CCER\_CC4E;

  TIMER4\_BASE->PSC = 71;

  TIMER4\_BASE->ARR = 5000;

  TIMER4\_BASE->DCR = 0;

  TIMER4\_BASE->CCR1 = 1000;

  TIMER4\_BASE->CCR1 = 1000;

  TIMER4\_BASE->CCR2 = 1000;

  TIMER4\_BASE->CCR3 = 1000;

  TIMER4\_BASE->CCR4 = 1000;

  pinMode(PB6, PWM);

  pinMode(PB7, PWM);

  pinMode(PB8, PWM);

  pinMode(PB9, PWM);

}

// bateria, gyro, kalibracja gyro

void setup() {

  //batt

  pinMode(4, INPUT\_ANALOG);

  // umozliwia uzycie pb3 i pb4 jako i/o pin

  afio\_cfg\_debug\_ports(AFIO\_DEBUG\_SW\_ONLY);

  pinMode(PB3, OUTPUT);

  pinMode(PB4, OUTPUT);

  //na pb3

  green\_led(LOW);

  // na pb4

  red\_led(HIGH);

  // do debugu

  //Serial.begin(115200);

  //delay(250);

  // setup timerow do przerwan

  timer\_setup();

  delay(50);

  // sprawdzenie poprawnosci komunikacji z mpu6050

  Wire2.begin();

  Wire2.beginTransmission(gyro\_address);

  error = Wire2.endTransmission();

  while (error != 0) {

    //kod bledu komunikacji z gy521 jest 2

    error = 2;

    error\_signal();

    delay(4);

  }

  // setup do zyroskopu (kalibracja zyroskopu)

  gyro\_setup();

  //jezeli kalibracja automatyczna -> wywolanie sekwencji sygnalizacyjnej

  if (!use\_manual\_calibration) {

    for (count\_var = 0; count\_var < 1250; count\_var++) {

      if (count\_var % 125 == 0) {

        digitalWrite(PB4, !digitalRead(PB4));

      }

      delay(4);

    }

    count\_var = 0;

  }

  //kalibracja zyroskpu

  calibrate\_gyro();

  //nieprawidlowe sygnaly transmittera sygnaly bledu

  while (channel\_1 < 990 || channel\_2 < 990 || channel\_3 < 990 || channel\_4 < 990)  {

    //kod bledu 3

    error = 3;

    error\_signal();

    delay(4);

  }

  error = 0;

  //koniecznosc ustawienia throttle w pozycji poczatkowej

  while (channel\_3 < 990 || channel\_3 > 1050) {

    //gdy nie mamy ustawienia poczatkowego -> blad

    error = 4;

    error\_signal();

    delay(4);

  }

  error = 0;

  //wylaczamy diode kiedy wszystkie procedury wykonane

  red\_led(LOW);

  // 0 - 0V ; 4095 - 36.3V (11\*3.3, poniewaz divider 1/11)

  battery\_voltage = (float)analogRead(4) / 112.81;

  //wlaczamy timer do kontroli czasu kazdej iteracji

  loop\_timer = micros();

  //sygnalizacja gotowosci do startu

  green\_led(HIGH);

}

void loop() {

  // pokazujmey sygnaly ew bledow

  error\_signal();

  // do zczytania acc, gyro, temp ustandaryzowanych za pomoca calib values

  gyro\_signals();

  // konwersja na deg/s (65.5 = 1 deg/sec) wartosci do pid

  gyro\_roll\_input = (gyro\_roll\_input \* 0.7) + (((float)gyro\_roll / 65.5) \* 0.3);

  gyro\_pitch\_input = (gyro\_pitch\_input \* 0.7) + (((float)gyro\_pitch / 65.5) \* 0.3);

  gyro\_yaw\_input = (gyro\_yaw\_input \* 0.7) + (((float)gyro\_yaw / 65.5) \* 0.3);

  // kat obliczany z gyro (kat dryfu) 0.0000611 = 1 / (250Hz / 65.5)

  angle\_pitch += (float)gyro\_pitch \* 0.0000611;

  angle\_roll += (float)gyro\_roll \* 0.0000611;

  // poprawka kata o wychylenie yaw

  angle\_pitch -= angle\_roll \* sin((float)gyro\_yaw \* 0.000001066);

  angle\_roll += angle\_pitch \* sin((float)gyro\_yaw \* 0.000001066);

  // obliczenie katow z przyspieszenia

  acc\_total\_vector = sqrt((acc\_x \* acc\_x) + (acc\_y \* acc\_y) + (acc\_z \* acc\_z));

  // katy obliczane z przyspieszen dla kolejno pitch i roll

  if (abs(acc\_y) < acc\_total\_vector) {

    angle\_pitch\_acc = asin((float)acc\_y / acc\_total\_vector) \* 57.296;

  }

  if (abs(acc\_x) < acc\_total\_vector) {

    angle\_roll\_acc = asin((float)acc\_x / acc\_total\_vector) \* 57.296;

  }

  //filterek komplementarny

  angle\_pitch = angle\_pitch \* 0.9996 + angle\_pitch\_acc \* 0.0004;

  angle\_roll = angle\_roll \* 0.9996 + angle\_roll\_acc \* 0.0004;

  // doswiadczalna poprawka katow

  pitch\_level\_adjust = angle\_pitch \* 15;

  roll\_level\_adjust = angle\_roll \* 15;

  // jezeli nie chcemy by poprawka byla stosowana do autolevel'owania

  if (!auto\_level) {

    pitch\_level\_adjust = 0;

    roll\_level\_adjust = 0;

  }

  // start silnikow: throttle LOW ; yaw LEFT

  if (channel\_3 < 1050 && channel\_4 < 1050)

    start = 1;

  // kiedy yaw > centrum (czyli kiedy joystick puścisz) startujemy silniki

  if (start == 1 && channel\_3 < 1050 && channel\_4 > 1450) {

    start = 2;

    // sygnal - wystartowalismy

    green\_led(LOW);

    // na starcie kat z acc = katowi z gyro (niwelujemy rozbieznosci)

    angle\_pitch = angle\_pitch\_acc;

    angle\_roll = angle\_roll\_acc;

    // resetujemy tez wartosci do integral i d z pid zeby nie zaburzac startu

    pid\_i\_mem\_roll = 0;

    pid\_last\_roll\_d\_error = 0;

    pid\_i\_mem\_pitch = 0;

    pid\_last\_pitch\_d\_error = 0;

    pid\_i\_mem\_yaw = 0;

    pid\_last\_yaw\_d\_error = 0;

  }

  // zatrzymanie silnikow: throttle LOW ; yaw RIGHT

  if (start == 2 && channel\_3 < 1050 && channel\_4 > 1950) {

    start = 0;

    // sygnal gotowosci do startu

    green\_led(HIGH);

  }

  // setpoint dla roll

  pid\_roll\_setpoint = 0;

  // setpoint do pida dla pitch, roll, yaw zalezy od wartosci poszczegolnych wartosci z receivera (pasmo 1508 - 1492 jest martwym pasmem dodajemy je do poprawy setpointa)

  if (channel\_1 > 1508)

    pid\_roll\_setpoint = channel\_1 - 1508;

  else if (channel\_1 < 1492)

    pid\_roll\_setpoint = channel\_1 - 1492;

  //obliczamy setpoint odejmujac wartosc korekcji kata

  pid\_roll\_setpoint -= roll\_level\_adjust;

  //zeby otrzymac wartosc ustandaryzowana setpoint'a w deg/s

  pid\_roll\_setpoint /= 3.0;

  // setpoint dla pitch

  pid\_pitch\_setpoint = 0;

  // setpoint do pida dla pitch, roll, yaw zalezy od wartosci poszczegolnych wartosci z receivera (pasmo 1508 - 1492 jest martwym pasmem dodajemy je do poprawy setpointa)

  if (channel\_2 > 1508)

    //odwrocenie pitch NOWE

    pid\_pitch\_setpoint = -channel\_2 + 1508;

  else if (channel\_2 < 1492)

    pid\_pitch\_setpoint = -channel\_2 + 1492;

  //obliczamy setpoint odejmujac wartosc korekcji kata

  pid\_pitch\_setpoint -= pitch\_level\_adjust;

  //zeby otrzymac wartosc ustandaryzowana setpoint'a w deg/s

  pid\_pitch\_setpoint /= 3.0;

  // setpoint dla yaw

  pid\_yaw\_setpoint = 0;

  // setpoint do pida dla pitch, roll, yaw zalezy od wartosci poszczegolnych wartosci z receivera (pasmo 1508 - 1492 jest martwym pasmem dodajemy je do poprawy setpointa)

  if (channel\_3 > 1050) {

    if (channel\_4 > 1508)

      //od razu ze standaryzacja /3

      pid\_yaw\_setpoint = (channel\_4 - 1508) / 3.0;

    else if (channel\_4 < 1492)

      pid\_yaw\_setpoint = (channel\_4 - 1492) / 3.0;

  }

  //obliczenie wartosci outputu z pid

  calculate\_pid();

  // obliczenie U na batt z pomoca filtra komplementarnego (doswiadczalne wyznaczenie)

  battery\_voltage = battery\_voltage \* 0.92 + ((float)analogRead(4) / 1410.1);

  //jezeli U < 10V daj kod bledu

  if (battery\_voltage < 10.0 && error == 0)

    error = 1;

  //throttle na kanale 3 odczytywane bezposrednio

  throttle = channel\_3;

  // gdy gotowy do startu

  if (start == 2) {

    // 200 miejsca na pid i usera

    if (throttle > 1800) throttle = 1800;

    // obliczenie pwm na esc 1, 2, 3, 4

    esc\_1 = throttle - pid\_output\_pitch + pid\_output\_roll - pid\_output\_yaw;

    esc\_2 = throttle + pid\_output\_pitch + pid\_output\_roll + pid\_output\_yaw;

    esc\_3 = throttle + pid\_output\_pitch - pid\_output\_roll - pid\_output\_yaw;

    esc\_4 = throttle - pid\_output\_pitch - pid\_output\_roll + pid\_output\_yaw;

    // nie przekraczamy wartosci granicznych

    if (esc\_1 < 1100)

      esc\_1 = 1100;

    if (esc\_2 < 1100)

      esc\_2 = 1100;

    if (esc\_3 < 1100)

      esc\_3 = 1100;

    if (esc\_4 < 1100)

      esc\_4 = 1100;

    // nie przekraczamy wartosci granicznych

    if (esc\_1 > 2000)

      esc\_1 = 2000;

    if (esc\_2 > 2000)

      esc\_2 = 2000;

    if (esc\_3 > 2000)

      esc\_3 = 2000;

    if (esc\_4 > 2000)

      esc\_4 = 2000;

  }

  // jezeli nie start utrzymuj staly pwm 1000 na silnikach

  else {

    esc\_1 = 1000;

    esc\_2 = 1000;

    esc\_3 = 1000;

    esc\_4 = 1000;

  }

  // podanie wartosci throttle z receivera na esc 1, 2, 3, 4

  TIMER4\_BASE->CCR1 = esc\_1;

  TIMER4\_BASE->CCR2 = esc\_2;

  TIMER4\_BASE->CCR3 = esc\_3;

  TIMER4\_BASE->CCR4 = esc\_4;

  //resetujemy timer 4

  TIMER4\_BASE->CNT = 5000;

  // obieg petli musi wynosic dokladnie 4000us jak nie to wystawiamy sygnal bledu

  if (micros() - loop\_timer > 4050)

    error = 5;

  while (micros() - loop\_timer < 4000);

  loop\_timer = micros();

}

## Inicjalizacja I2C

#include <Wire.h>

TwoWire Wire2(2, I2C\_FAST\_MODE); // inicjalizacja I2C2 (piny B10 - SCL, B11 - SDA (domyślnie wire obsługuje I2C1)

Kod inicjalizuje magistralę I2C2, która będzie używana do komunikacji z czujnikami drona.

## Ustawienia PID

// ustawienia pid dla roll

float pid\_p\_gain\_roll = 1.3;

float pid\_i\_gain\_roll = 0.04;

float pid\_d\_gain\_roll = 18.0;

int pid\_max\_roll = 400;

// ustawienia pid dla pitch

float pid\_p\_gain\_pitch = pid\_p\_gain\_roll;

float pid\_i\_gain\_pitch = pid\_i\_gain\_roll;

float pid\_d\_gain\_pitch = pid\_d\_gain\_roll;

int pid\_max\_pitch = pid\_max\_roll;

// ustawienia pid dla yaw

float pid\_p\_gain\_yaw = 4.0;

float pid\_i\_gain\_yaw = 0.02;

float pid\_d\_gain\_yaw = 0.0;

int pid\_max\_yaw = 400;

Te ustawienia określają wzmocnienia (proporcjonalne, całkowe i różniczkowe) dla regulatorów PID używanych do kontrolowania osi roll, pitch i yaw drona.

## Zmienne globalne

boolean auto\_level = true;

float pid\_error\_temp;

float pid\_i\_mem\_roll, pid\_roll\_setpoint, gyro\_roll\_input, pid\_output\_roll, pid\_last\_roll\_d\_error;

float pid\_i\_mem\_pitch, pid\_pitch\_setpoint, gyro\_pitch\_input, pid\_output\_pitch, pid\_last\_pitch\_d\_error;

float pid\_i\_mem\_yaw, pid\_yaw\_setpoint, gyro\_yaw\_input, pid\_output\_yaw, pid\_last\_yaw\_d\_error;

float roll\_level\_adjust, pitch\_level\_adjust;

Zmienne te są używane w funkcji regulatora PID oraz do przechowywania danych z czujników.

## Kalibracja ręczna

uint8\_t use\_manual\_calibration = true;

int16\_t manual\_acc\_pitch\_cal\_value = -218;

int16\_t manual\_acc\_roll\_cal\_value = -41;

int16\_t manual\_gyro\_pitch\_cal\_value = -197;

int16\_t manual\_gyro\_roll\_cal\_value = -403;

int16\_t manual\_gyro\_yaw\_cal\_value = -89;

Jeżeli use\_manual\_calibration jest ustawione na true, to wartości manual będą używane do kalibracji czujników.

## Adres i zmienne dla żyroskopu i akcelerometru

uint8\_t gyro\_address = 0x68;

int16\_t temperature, count\_var;

int16\_t acc\_x, acc\_y, acc\_z;

int16\_t gyro\_pitch, gyro\_roll, gyro\_yaw;

int32\_t acc\_total\_vector;

int32\_t gyro\_roll\_cal, gyro\_pitch\_cal, gyro\_yaw\_cal;

float angle\_roll\_acc, angle\_pitch\_acc, angle\_pitch, angle\_roll;

Te zmienne przechowują adres urządzenia I2C oraz dane odczytane z żyroskopu i akcelerometru.

## Zmienne pomocnicze do sygnalizacji stanów drona

uint8\_t last\_channel\_1, last\_channel\_2, last\_channel\_3, last\_channel\_4;

uint8\_t highByte, lowByte, flip32, start;

uint8\_t error, error\_counter, error\_led;

float battery\_voltage;

Zmienne te są używane do przechowywania stanów sygnalizacji błędów oraz napięcia baterii.

## PWM do ESC i zmienne do motors

int16\_t esc\_1, esc\_2, esc\_3, esc\_4;

int16\_t throttle, cal\_int;

Zmienne te są używane do kontrolowania silników drona.

## Zmienne do odbiornika

int32\_t channel\_1\_start, channel\_1;

int32\_t channel\_2\_start, channel\_2;

int32\_t channel\_3\_start, channel\_3;

int32\_t channel\_4\_start, channel\_4;

int32\_t channel\_5\_start, channel\_5;

int32\_t channel\_6\_start, channel\_6;

Zmienne te są używane do przechowywania danych z kanałów odbiornika.

## Kontrola czasu obiegu każdej iteracji

uint32\_t loop\_timer, error\_timer;

Zmienne te przechowują czas iteracji głównej pętli programu.

## Funkcje sygnalizacji stanu diodami LED

void red\_led(int8\_t level) {

digitalWrite(PB4, level);

}

void green\_led(int8\_t level) {

digitalWrite(PB3, level);

}

Funkcje te kontrolują diody LED sygnalizujące stan drona.

## Funkcja do obsługi timerów (w tym przerwań)

void timer\_setup(void) {

// przerwania do handlera 1, 2, 3, 4

Timer2.attachCompare1Interrupt(handler\_channel\_1);

Timer2.attachCompare2Interrupt(handler\_channel\_2);

Timer2.attachCompare3Interrupt(handler\_channel\_3);

Timer2.attachCompare4Interrupt(handler\_channel\_4);

TIMER2\_BASE->CR1 = TIMER\_CR1\_CEN;

TIMER2\_BASE->CR2 = 0;

TIMER2\_BASE->SMCR = 0;

TIMER2\_BASE->DIER = TIMER\_DIER\_CC1IE | TIMER\_DIER\_CC2IE | TIMER\_DIER\_CC3IE | TIMER\_DIER\_CC4IE;

TIMER2\_BASE->EGR = 0;

TIMER2\_BASE->CCMR1 = 0b100000001;

TIMER2\_BASE->CCMR2 = 0b100000001;

TIMER2\_BASE->CCER = TIMER\_CCER\_CC1E | TIMER\_CCER\_CC2E | TIMER\_CCER\_CC3E | TIMER\_CCER\_CC4E;

TIMER2\_BASE->PSC = 71;

TIMER2\_BASE->ARR = 0xFFFF;

TIMER2\_BASE->DCR = 0;

// przerwania do handlera 5, 6

Timer3.attachCompare1Interrupt(handler\_channel\_5);

Timer3.attachCompare2Interrupt(handler\_channel\_6);

TIMER3\_BASE->CR1 = TIMER\_CR1\_CEN;

TIMER3\_BASE->CR2 = 0;

TIMER3\_BASE->SMCR = 0;

TIMER3\_BASE->DIER = TIMER\_DIER\_CC1IE | TIMER\_DIER\_CC2IE;

TIMER3\_BASE->EGR = 0;

TIMER3\_BASE->CCMR1 = 0b100000001;

TIMER3\_BASE->CCMR2 = 0;

TIMER3\_BASE->CCER = TIMER\_CCER\_CC1E | TIMER\_CCER\_CC2E;

TIMER3\_BASE->PSC = 71;

TIMER3\_BASE->ARR = 0xFFFF;

TIMER3\_BASE->DCR = 0;

//wlaczenie timera 4 i dla niego przerwan

TIMER4\_BASE->CR1 = TIMER\_CR1\_CEN | TIMER\_CR1\_ARPE;

TIMER4\_BASE->CR2 = 0;

TIMER4\_BASE->SMCR = 0;

TIMER4\_BASE->DIER = 0;

TIMER4\_BASE->EGR = 0;

TIMER4\_BASE->CCMR1 = (0b110 << 4) | TIMER\_CCMR1\_OC1PE |(0b110 << 12) | TIMER\_CCMR1\_OC2PE;

TIMER4\_BASE->CCMR2 = (0b110 << 4) | TIMER\_CCMR2\_OC3PE |(0b110 << 12) | TIMER\_CCMR2\_OC4PE;

TIMER4\_BASE->CCER = TIMER\_CCER\_CC1E | TIMER\_CCER\_CC2E | TIMER\_CCER\_CC3E | TIMER\_CCER\_CC4E;

TIMER4\_BASE->PSC = 71;

TIMER4\_BASE->ARR = 5000;

TIMER4\_BASE->DCR = 0;

TIMER4\_BASE->CCR1 = 1000;

TIMER4\_BASE->CCR1 = 1000;

TIMER4\_BASE->CCR2 = 1000;

TIMER4\_BASE->CCR3 = 1000;

TIMER4\_BASE->CCR4 = 1000;

pinMode(PB6, PWM);

pinMode(PB7, PWM);

pinMode(PB8, PWM);

pinMode(PB9, PWM);

}

* Timer2.attachCompare1Interrupt(handler\_channel\_1);
* Timer2.attachCompare2Interrupt(handler\_channel\_2);
* Timer2.attachCompare3Interrupt(handler\_channel\_3);
* Timer2.attachCompare4Interrupt(handler\_channel\_4);

Te linie kodu ustawiają przerwania porównawcze dla Timer 2, przypisując im odpowiednie funkcje obsługi dla kanałów 1-4. (timer wywołuje przerwania dla poszczególnych kanałów)

* TIMER2\_BASE->CR1 = TIMER\_CR1\_CEN; - Włącza Timer 2.
* TIMER2\_BASE->DIER … = ... - Umożliwia przerwania dla kanałów 1-4.
* TIMER2\_BASE->CCMR1 i TIMER2\_BASE->CCMR2 - Ustawia tryb porównawczy dla kanałów 1-4.
* TIMER2\_BASE->CCER … - Włącza wyjścia dla kanałów 1-4.
* TIMER2\_BASE->PSC = 71; - Ustawia preskaler na 71.
* TIMER2\_BASE->ARR = 0xFFFF; - Ustawia wartość auto-reload na maksymalną.

Te linie kodu ustawiają przerwania porównawcze dla Timer 3, przypisując im odpowiednie funkcje obsługi dla kanałów 5-6.

* Timer3.attachCompare1Interrupt(handler\_channel\_5);
* Timer3.attachCompare2Interrupt(handler\_channel\_6);

Timer 4 jest konfigurowany do generowania sygnałów PWM na wyjściach odpowiednich do kontrolowania silników drona.

* TIMER4\_BASE->CCR1, TIMER4\_BASE->CCR2, TIMER4\_BASE->CCR3, TIMER4\_BASE->CCR4

## Funkcja regulatora PID

void calculate\_pid(void) {

// obliczenia dla osi roll

pid\_error\_temp = gyro\_roll\_input - pid\_roll\_setpoint;

pid\_i\_mem\_roll += pid\_i\_gain\_roll \* pid\_error\_temp;

if(pid\_i\_mem\_roll > pid\_max\_roll)

pid\_i\_mem\_roll = pid\_max\_roll;

else if(pid\_i\_mem\_roll < pid\_max\_roll \* -1)

pid\_i\_mem\_roll = pid\_max\_roll \* -1;

pid\_output\_roll = pid\_p\_gain\_roll \* pid\_error\_temp + pid\_i\_mem\_roll + pid\_d\_gain\_roll \* (pid\_error\_temp - pid\_last\_roll\_d\_error);

if(pid\_output\_roll > pid\_max\_roll)

pid\_output\_roll = pid\_max\_roll;

else if(pid\_output\_roll < pid\_max\_roll \* -1)

pid\_output\_roll = pid\_max\_roll \* -1;

pid\_last\_roll\_d\_error = pid\_error\_temp;

// obliczenia dla osi pitch

pid\_error\_temp = gyro\_pitch\_input - pid\_pitch\_setpoint;

pid\_i\_mem\_pitch += pid\_i\_gain\_pitch \* pid\_error\_temp;

if(pid\_i\_mem\_pitch > pid\_max\_pitch)

pid\_i\_mem\_pitch = pid\_max\_pitch;

else if(pid\_i\_mem\_pitch < pid\_max\_pitch \* -1)

pid\_i\_mem\_pitch = pid\_max\_pitch \* -1;

pid\_output\_pitch = pid\_p\_gain\_pitch \* pid\_error\_temp + pid\_i\_mem\_pitch + pid\_d\_gain\_pitch \* (pid\_error\_temp - pid\_last\_pitch\_d\_error);

if(pid\_output\_pitch > pid\_max\_pitch)

pid\_output\_pitch = pid\_max\_pitch;

else if(pid\_output\_pitch < pid\_max\_pitch \* -1)

pid\_output\_pitch = pid\_max\_pitch \* -1;

pid\_last\_pitch\_d\_error = pid\_error\_temp;

// obliczenia dla osi yaw

pid\_error\_temp = gyro\_yaw\_input - pid\_yaw\_setpoint;

pid\_i\_mem\_yaw += pid\_i\_gain\_yaw \* pid\_error\_temp;

if(pid\_i\_mem\_yaw > pid\_max\_yaw)

pid\_i\_mem\_yaw = pid\_max\_yaw;

else if(pid\_i\_mem\_yaw < pid\_max\_yaw \* -1)

pid\_i\_mem\_yaw = pid\_max\_yaw \* -1;

pid\_output\_yaw = pid\_p\_gain\_yaw \* pid\_error\_temp + pid\_i\_mem\_yaw + pid\_d\_gain\_yaw \* (pid\_error\_temp - pid\_last\_yaw\_d\_error);

if(pid\_output\_yaw > pid\_max\_yaw)

pid\_output\_yaw = pid\_max\_yaw;

else if(pid\_output\_yaw < pid\_max\_yaw \* -1)

pid\_output\_yaw = pid\_max\_yaw \* -1;

pid\_last\_yaw\_d\_error = pid\_error\_temp;

}

Funkcja calculate\_pid jest odpowiedzialna za obliczanie wyjść regulatorów PID dla osi roll, pitch i yaw.

## Funkcja kalibracji żyroskopu

void calibrate\_gyro(void) {

if (use\_manual\_calibration)

cal\_int = 2000;

else {

cal\_int = 0;

manual\_gyro\_pitch\_cal\_value = 0;

manual\_gyro\_roll\_cal\_value = 0;

manual\_gyro\_yaw\_cal\_value = 0;

}

if (cal\_int != 2000) {

for (cal\_int = 0; cal\_int < 2000 ; cal\_int ++) {

if (cal\_int % 25 == 0)

digitalWrite(PB4, !digitalRead(PB4));

gyro\_signals();

gyro\_roll\_cal += gyro\_roll;

gyro\_pitch\_cal += gyro\_pitch;

gyro\_yaw\_cal += gyro\_yaw;

delay(4);

}

red\_led(HIGH);

gyro\_roll\_cal /= 2000;

gyro\_pitch\_cal /= 2000;

gyro\_yaw\_cal /= 2000;

}

else {

gyro\_pitch\_cal = manual\_gyro\_pitch\_cal\_value;

gyro\_roll\_cal = manual\_gyro\_roll\_cal\_value;

gyro\_yaw\_cal = manual\_gyro\_yaw\_cal\_value;

}

}

Funkcja calibrate\_gyro kalibruje żyroskop, używając wartości manualnych lub automatycznych obliczeń.

## Funkcja pętli głównej

void loop() {

loop\_timer = micros();

// odczytywanie sygnałów z żyroskopu i akcelerometru

gyro\_signals();

// obliczanie PID

calculate\_pid();

// kontrola silników na podstawie wyjść PID

motor\_control();

while (micros() - loop\_timer < 4000);

}

Pętla loop() jest główną pętlą programu, odpowiedzialną za odczytywanie danych z czujników, odpowiednie ich przekształcanie, obliczanie wartości PID na podstawie zadanych setpointów i aktualnych danych z czujników oraz kontrolę silników. Przy czym cały program pracuje z częstotliwością >= 250 Hz.

# Omówienie otrzymanych wyników i porównanie

Osiągnięcie stabilnego lotu drona, sterowanego za pomocą odbiornika i nadajnika, jest istotnym krokiem w rozwoju autonomicznych systemów lotniczych. Po wielu próbach udało się stworzyć system, który skutecznie stabilizuje się w powietrzu. Wdrożenie kontrolera PID oraz filtru komplementarnego w kodzie zapewniło precyzyjne i płynne sterowanie dronem, co jest kluczowe dla jego stabilności i bezpieczeństwa lotu. Takie rozwiązania mają szerokie zastosowanie, od dronów rekreacyjnych po profesjonalne systemy do zadań specjalistycznych, takich jak inspekcje infrastruktury, dostawy, czy misje ratunkowe.

W literaturze istnieje wiele metod sterowania dronami, od prostych systemów stabilizacji po zaawansowane algorytmy uczenia maszynowego. W porównaniu z bardziej skomplikowanymi podejściami, zastosowanie kontrolera PID i filtru komplementarnego jest stosunkowo proste, ale bardzo efektywne. Daje to ponadto wiele możliwości rozwoju tego systemu w przyszłości.

Kontroler PID (Proportional-Integral-Derivative) jest szeroko stosowany w systemach sterowania ze względu na swoją prostotę i skuteczność. W moim projekcie kontroler PID został zaimplementowany w celu utrzymania stabilności lotu poprzez korektę błędów w położeniu drona. Filtr komplementarny natomiast łączy dane z akcelerometru i żyroskopu, co pozwala na uzyskanie bardziej stabilnych i dokładnych pomiarów kątów nachylenia.

W porównaniu z bardziej zaawansowanymi metodami, takimi jak filtry Kalmana czy algorytmy oparte na sieciach neuronowych, moje podejście oferuje dobrą równowagę między złożonością implementacji a efektywnością działania. Choć bardziej zaawansowane metody mogą zapewnić lepszą precyzję i stabilność, są one również bardziej złożone i wymagają większych zasobów obliczeniowych, co może być niepraktyczne dla prostszych systemów.

Pomimo sukcesu w stabilizacji drona, istnieją pewne ograniczenia w moim rozwiązaniu: ograniczona precyzja PID - kontroler PID, choć skuteczny, może nie być wystarczający w bardziej dynamicznych i wymagających środowiskach. Reakcja na szybkie zmiany warunków lotu może być ograniczona przez prostą naturę algorytmu PID.

Wrażliwość na zakłócenia: Zastosowany filtr komplementarny, chociaż skuteczny w warunkach normalnych, może być wrażliwy na zakłócenia i szumy w danych z czujników.

Brak zaawansowanych funkcji autonomicznych: System sterowania w moim projekcie opiera się na manualnym sterowaniu za pomocą nadajnika i odbiornika, co ogranicza autonomiczne możliwości drona. Częściowa autonomiczność sprowadza się natomiast do samodzielnej regulacji poziomu równowagi przez drona.

Aby poprawić wyniki i rozszerzyć funkcjonalność drona, można podjąć kilka działań:

Implementacja filtru Kalmana: Zastąpienie filtru komplementarnego bardziej zaawansowanym filtrem Kalmana może poprawić dokładność i stabilność pomiarów, szczególnie w obecności zakłóceń. Zaawansowane algorytmy sterowania: Wprowadzenie algorytmów adaptacyjnych lub sterowania predykcyjnego może poprawić reakcję drona na dynamiczne zmiany warunków lotu. Autonomiczne funkcje: Dodanie funkcji autonomicznych, takich jak automatyczne unikanie przeszkód, planowanie trasy czy autonomiczne lądowanie, może znacząco zwiększyć użyteczność drona. W planach jest dodanie ów funkcji autonomicznego odlotu i przylotu za pomocą czujnika barometrycznego. Można także dodatkowo dołożyć integrację z systemami wizyjnymi, tu mam na myśli, że wykorzystanie kamer i algorytmów przetwarzania obrazu może pozwolić na bardziej zaawansowane funkcje nawigacyjne i autonomiczne. Dodatkowo można także zastanowić się nad lepszymi sposobami optymalizacji energetycznej można np. wprowadzić tkz. tryb uśpienia, celem ograniczenia zużycia energii po zadanym czasie nieaktywności.

Opracowany system sterowania dronem, oparty na kontrolerze PID i filtrze komplementarnym, osiągnął zamierzony cel stabilizacji lotu. Porównanie z innymi rozwiązaniami wykazało, że choć istnieją bardziej zaawansowane metody, moje podejście jest efektywne i stosunkowo proste do implementacji. Ograniczenia projektu wskazują na możliwość dalszego rozwoju, szczególnie w zakresie precyzji, odporności na zakłócenia oraz autonomicznych funkcji. Przyszłe badania mogą skupić się na implementacji bardziej zaawansowanych algorytmów i funkcji, co pozwoli na dalszą poprawę wydajności i użyteczności drona.

# Podsumowanie

Jeżeli masz jakiekolwiek zastrzeżenia co do projektu, to jest myślisz, że coś można poprawić, proszę daj nam znać (skontaktuj się z: Krzysztofem (Krzychu)). Jak na razie dron działa według zamierzeń (na AVR nie działał według zamierzeń, w komputerze na wartościach wszystko było idealnie, jednak, gdy odpalamy drona w realu wszystko się chrzani i PID nie działa poprawnie, dron wówczas wznosił się, ale nie potrafił w prawidłowy sposób regulować się, mimo że podczas symulacji wszystko jest ok, było to spowodowane dużymi ograniczeniami sprzętowymi avr. Łącznie na przestrzeni roku czasu, przeprowadzono ogromną ilość prób (w większości zakończonych porażkami), co ostatecznie doprowadziło po 11 rewizji kodu FC pod AVR do zastąpienia AVR na rzecz STM32F103C8T6. Tutaj już po 3 rewizji kodu udało się ostatecznie osiągnąć sukces, gdzie dron potrafił w poprawny sposób się autolevelować oraz poprawnie reagował na zadane wartości sygnałów z transmitera. Obecnie projekt jest w fazie zakończenia.

# Bibliografia

1. Beard, Randal W., McLain, Timothy W. Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. Wiley, 2012.
2. Ogata, Katsuhiko. Modern PID Control. Prentice Hall, 2010.
3. Vilanova, Ramon, Visioli, Antonio. PID Control in the Third Millennium: Lessons Learned and New Approaches. Springer, 2011.
4. Szczepański, Janusz. Systemy sterowania dronami. PWN, 2018.
5. Kowalski, Piotr. Podstawy sterowania dronami: teoria i praktyka. Wydawnictwo Naukowe PWN, 2015.
6. Lewis, Frank L. Remote Control Systems: Principles and Applications. Springer, 2009.
7. Grainger, John D. Radio Control for Dummies: A Comprehensive Guide. Wiley, 2016.
8. Wiśniewski, Marek. Technologie zdalnego sterowania dronami. Wydawnictwo Naukowe PWN, 2019.
9. Nowak, Adam. Zdalne sterowanie w robotyce i dronach. PWN, 2020.
10. A Survey of Quadrotor Drones Control Methods and Related Challenges. Journal of Intelligent & Robotic Systems, 2017.
11. Enhanced PID Control for Quadrotor UAV. IEEE Transactions on Control Systems Technology, 2019.
12. Stabilizacja lotu dronów za pomocą algorytmu PID. Pomiary Automatyka Kontrola, 2018.
13. Systemy sterowania dronami: badania i rozwój. Przegląd Elektrotechniczny, 2020.