Dokumentacja Techniczna Drona QUAD 4X

Opracował: Krzychu Jura

Spis treści

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

[Podłączenie Modułów elektronicznych 5](#_Toc161514607)

[Ustawienie Śmigieł i pinnout motorów oraz warunkowy kierunek obrotu oraz dokrecenie CW oraz CCW 6](#_Toc161514608)

[Co jeszcze jest niezbędne? 7](#_Toc161514609)

[Schemat Programu 7](#_Toc161514610)

[Działanie kodu, PID, Complementary filter, euler angles 8](#_Toc161514611)

[FUNKCJA IMPCALIBTST() 13](#_Toc161514612)

[FUNKCJA resetPID() oraz resetMotors() 13](#_Toc161514613)

[FUNKCJA anglesRead() i Kalibracja MPU6050 13](#_Toc161514614)

[FUNKCJA pidEquation() 15](#_Toc161514615)

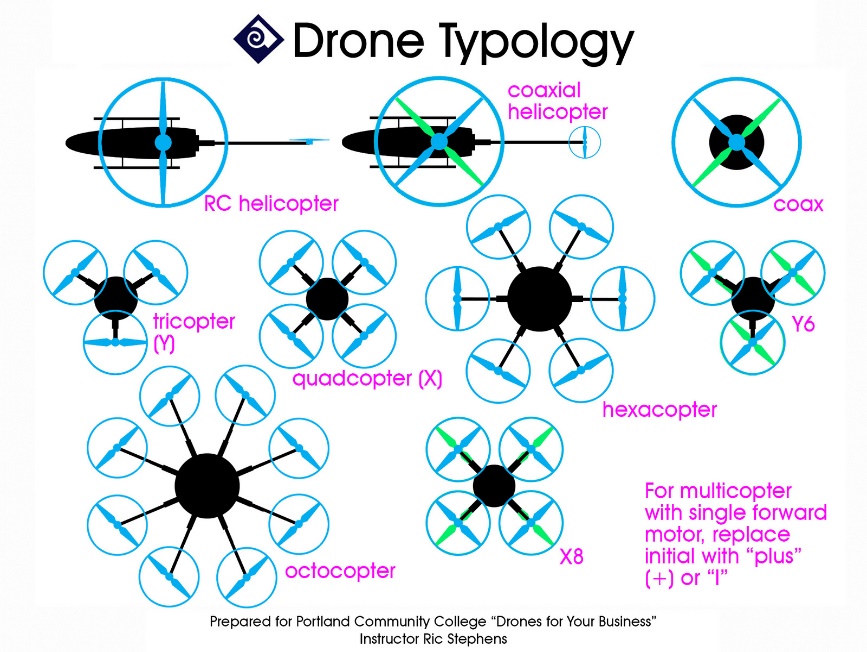
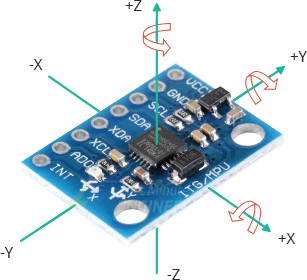
[FUNKCJe update() i fly() 17](#_Toc161514616)

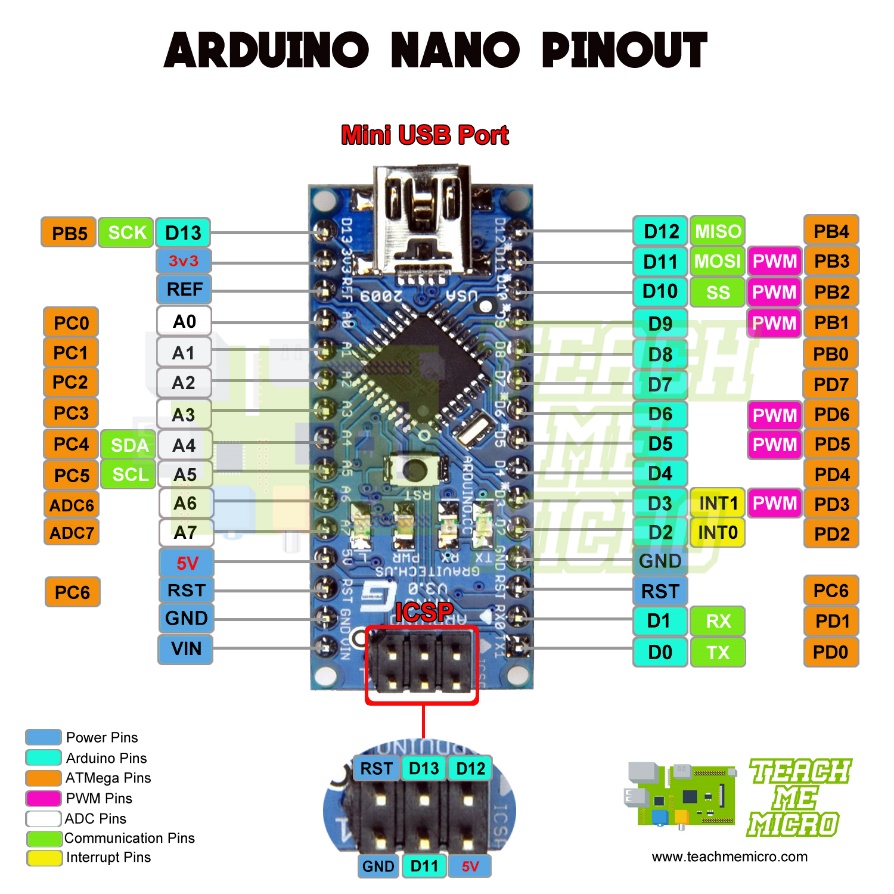
[FUNKCJA setup() i Loop() 18](#_Toc161514617)

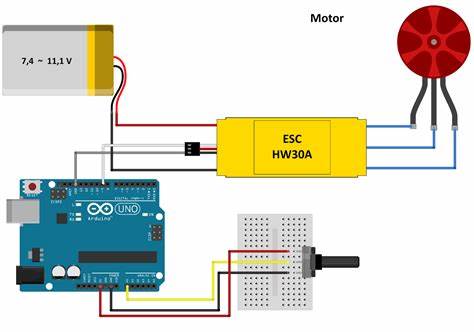
[Podsumowanie 19](#_Toc161514618)

# Krótki wstęp teoretyczny

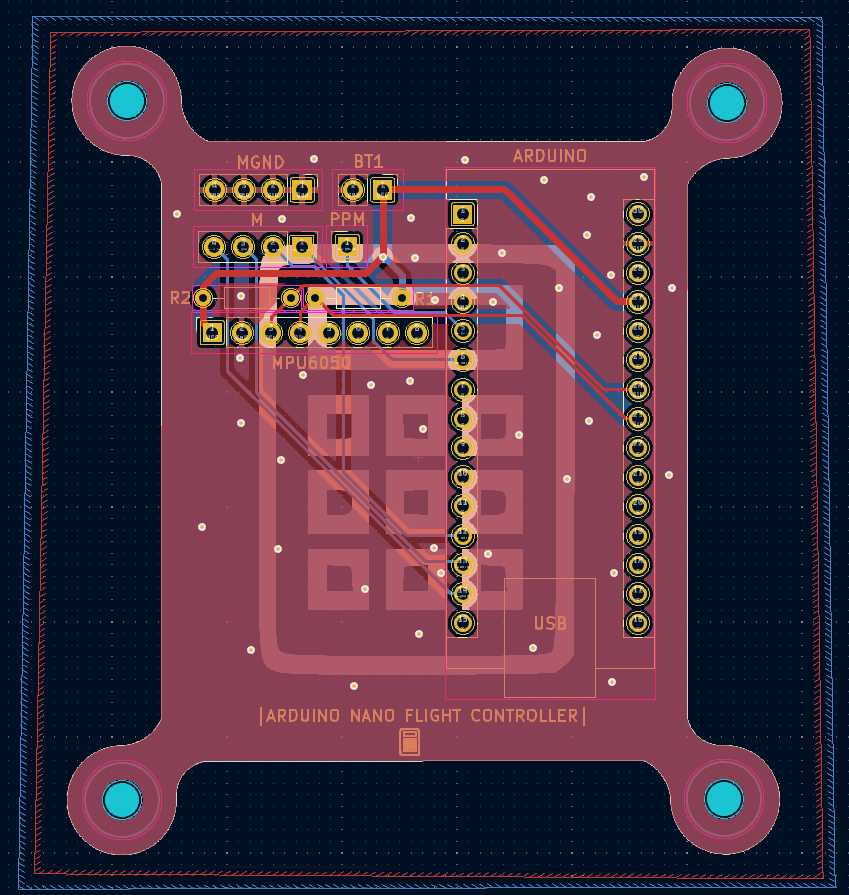
Nie będzie tu dużo (ogolnej) teorii (bez praktyki bezwartościowa) [w sensie podam Ci teorie która odnosi się do tego co ci się przyda TYLKO] więc przejdę do tego jak jak bym chciał żeby na początku mi ktoś to wyjaśnił (kiedy zaczynałem ten projekt):

1. Typów dronów masz parę (masz rysun poniżej), dlaczego o tym piszę wgl? 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 predkosc obrotowa (pulsacja) -> pochodzi z zyroskopu, przyspieszenie grawitacyjne -> pochodzi z akcelerometru. Akcelerometr + zyroskop = IMU. (celowo tlumacze jak dla przedszkolaka zebys mnie dobrze zrozumial nie będę operowal jakims słownictwem dla lamusow). Nasze IMU = MPU6050. Pls zapoznaj się ze zdjęciem jak masz orientacja mpu6050 zalezy od wektorow przyspieszenia oraz od predkosci katowej, masz fote (mega często pojawia się w kodzie, wzdłuż osi Z mierzysz accelerationZ oraz AngularVelocityZ i analogicznie z pozostałymi osiami):
3. Musisz wiedzieć jak obslugiwac arduino nano (najlepiej sprawnie umiec cos napisac na tej platformie). Mowa glownie o umiejetnosci generowania pwm, obsługi wejść wyjść oraz posługiwania się zaawansowanymi strukturami oraz funkcjami programisycznymi ([Kurs podstaw Arduino – spis treści, wstęp • FORBOT](https://forbot.pl/blog/kurs-arduino-podstawy-programowania-spis-tresci-kursu-id5290)). Co nas także będzie interesowalo ze w gruncie rzeczy ta platforma nie nadaje się do tak zaawanzsowanego projektu jakim jest dron Quad 4X, maksymalna czestotliwosc taktowania dla mikrokontrolera atmega 328p który siedzi na pokadzie tej avr to 20MHz to malo popularny stm ma ok 4 razy więcej ze sredniej polki. Still poki co pracujemy na tym co mamy. Co ważne zebys umial się polapac w wyprowadzeniach pinow dam ci fote w kodzie trzeba w końcu do jakiś wyjść podlaczyc te motorki cn (wyprowadzenia dk takie same jak w projekcie)?

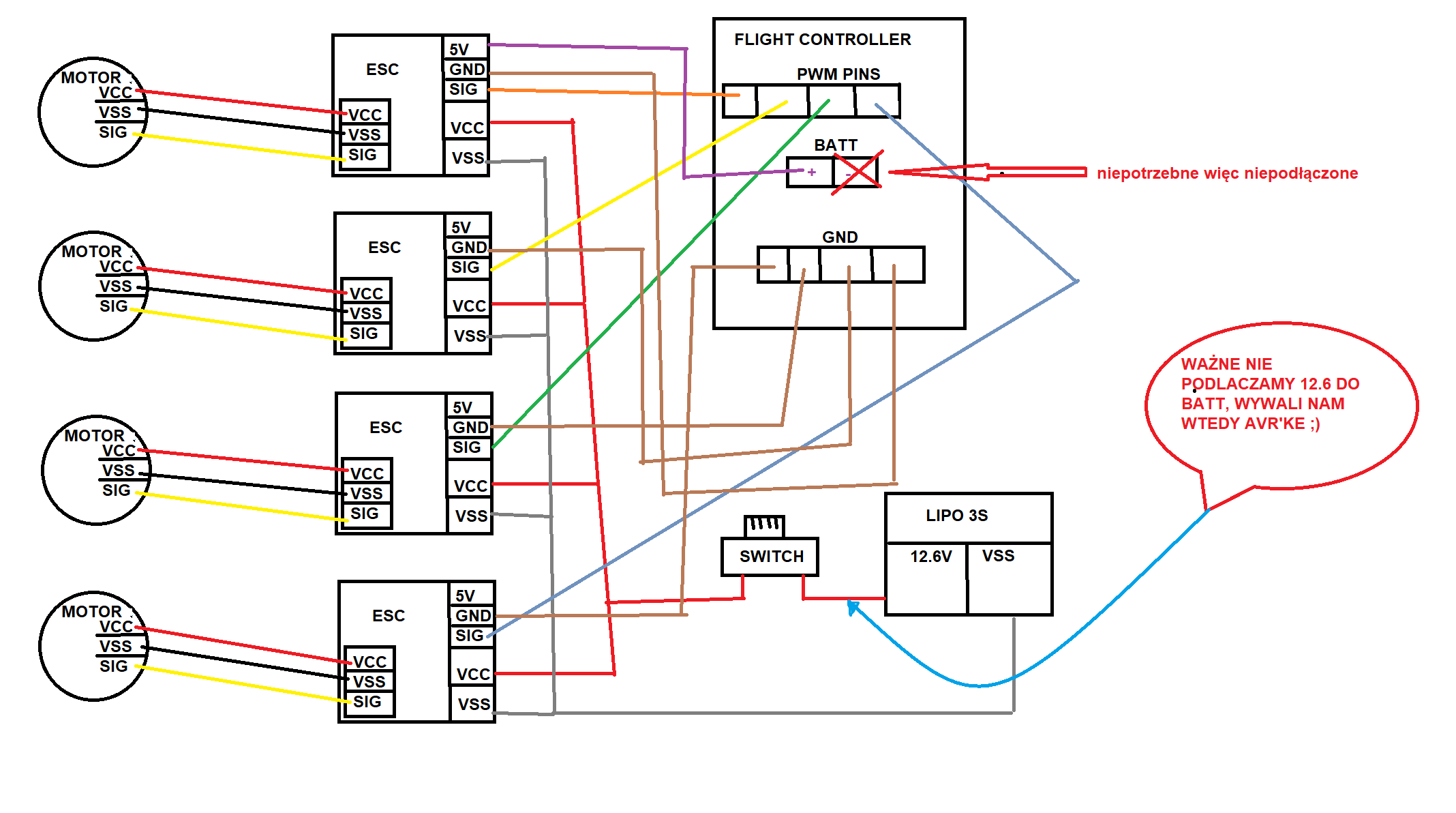


1. Nie będę się tu rozwodził nad działaniem silników bldc ani esc. Najważniejsze jest żebys wiedział ze nie możesz sobie zrobić tak jak w DC ze dajesz sobie mostek H i sterujesz sb podajac odpowiedni woltaż, CZYLI: żeby silniki BLDC zadziałały musisz je kontrolować przez ESC – electronic speed controler (nazwa mówi sama za siebie 😉). Upraszczajac zasade działania jak tylko się da: podajesz pwm na esc -> im większy pwm tym szybciej silnik będzie się kręcił w 1 lub 2 stronę (bo TAK: ZALEZY TO NIESTETY OD SPECYFIKI ESC) (tych kroczków będzie dużo więcej, ale uprzedzałem jak cos xD)
2. PID, katy eulera, filtr komplementarny, filtr kalmana, różne typu FC – flight controller oraz PC – PID controller nie będę omawiał wyjdzie w praniu (MUSIALBYM CI PODAWAC OGOLNA zaawansowana TEORIE KTOREJ PODAWAC NIE CHCE [wyjaśnimy na przykładach])

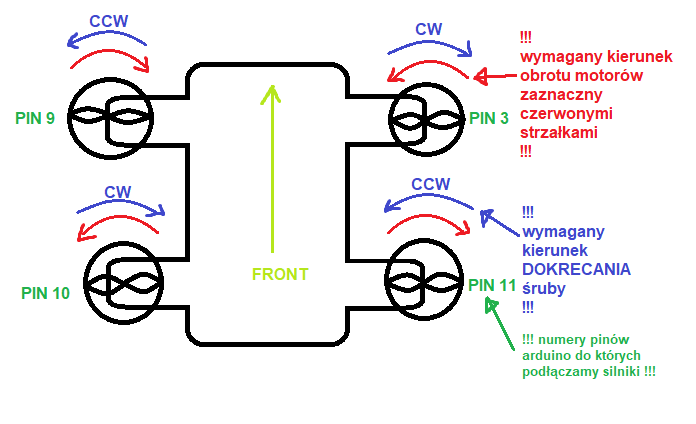
# Podłączenie Modułów elektronicznych

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 ocb.

Db. Pokazałem schemat płytki flight controllera pora na schemat połączeń zewnętrznych:

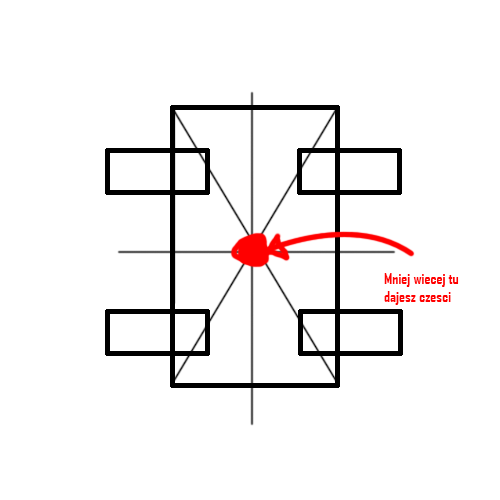


# Ustawienie Śmigieł i pinnout motorów oraz warunkowy kierunek obrotu oraz dokrecenie CW oraz CCW

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 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 sila odśrodkowa dokręcała ją w trakcie obrotu łopatek. Sporo tych warunków prawda? Musze się jeszcze zasmucić to nie koniec warunków, ale uprzedzałem jak coś…. Db pokazuję fotę jak powinny się silniki kręcić i kierunek dokręcania śruby:

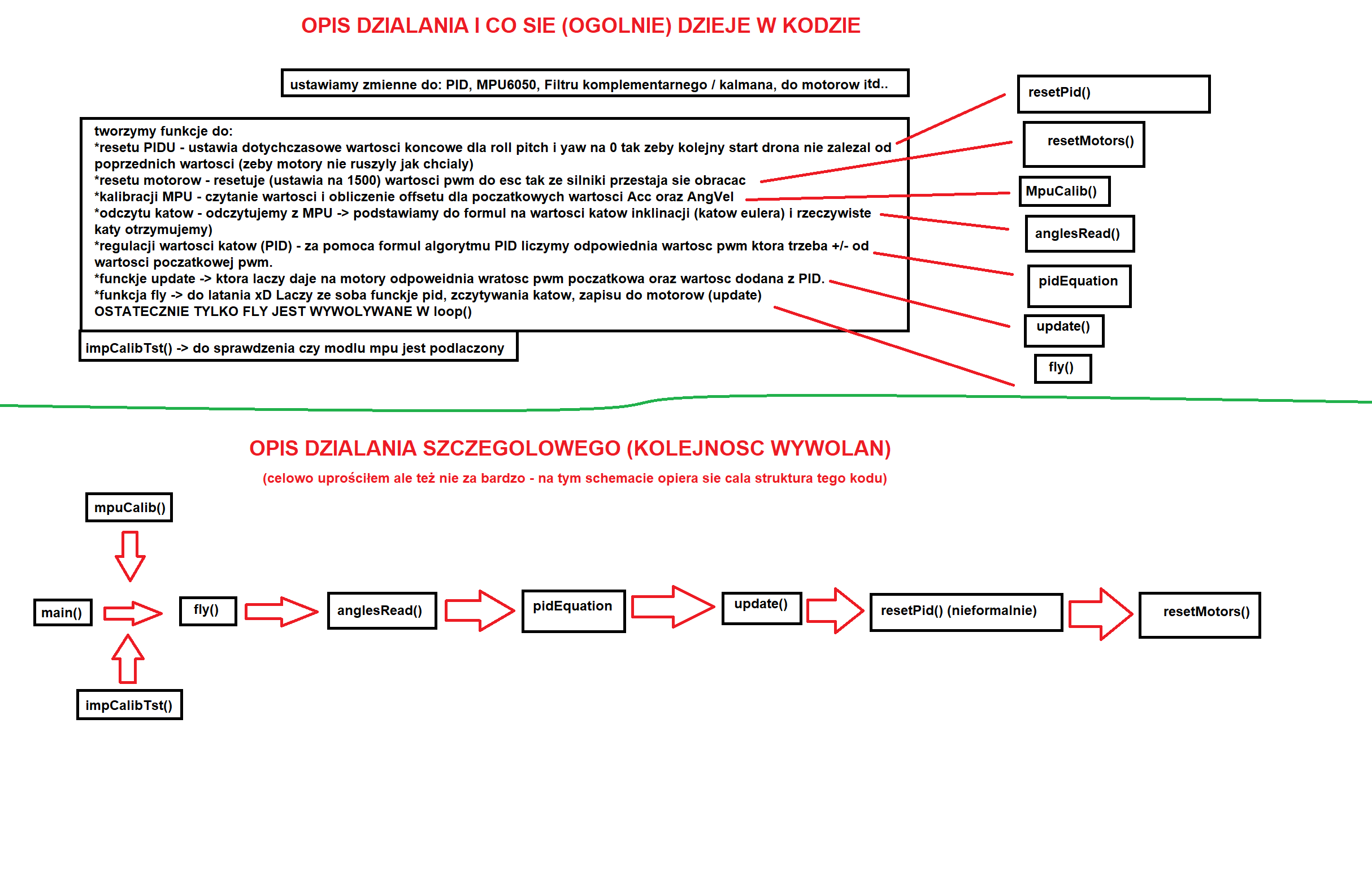
Podobnie sprawa ma się ze śmigłami, żeby dron mógł latać odpowiednie wymagania fizyczne muszą być spełnione i zapewniona odpowiednia siła ciągu, zarówno od motorów jak i od charakteru współdziałania łopatek z powietrzem WIEC TAK, TO W JAKIM MIEJSCU DASZ JAKA LOPATKE TAKŻE MA ZNACZENIE (jeżeli łopatkę 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?

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 xD Dlatego musisz zrobić tak by środek ciężkości drona znajdował się w przybliżeniu (dość dobrym) w dokładnie połowie konstrukcji samego drona, tzn:

# Schemat Programu

1. Jak działa program (schemat blokowy)

Myślę, że jest to jasne dosyć (sam schemat działania) (jakby nie było to na chill’ku wyjaśnię jeszcze raz bardzo dokładnie wszystko od początku przy omawianiu kodu, przy czym miej na uwadze ze projekt wciąż jest w fazie rozwojowej co prawda już raczej testów, ale kod od czasu do czasu wciąż ulega zmianie). Oczywiście funkcje tez wymagają wyjaśnienia, zrobimy to przy omawianiu kodu.

# Działanie kodu, PID, Complementary filter, euler angles

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 wsytskich 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 zasade działania -> resetują 😉 funkcja resetPid ustawia NAJWAZNIEJSZE rollPid oraz pitchPid, yawPid na 0. Poa tym daje tez wszystkie inee 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 sygnal PWM 1500 odpowiadajacy informacji: CHEJ WEŹ TE SILNIKI WYLACZ! (czyli 0mm/s). Realizujemy za pomocą funkcji writeMicroseconds(). Da sie 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 malym zakresie regulować tak potężne silniki (mala zmana 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 (odczytujmey wartość z rejestru za pomocą funkcji update() następnie za pomocą funkcji katow eulera odczytujemy wartości odpowiednich katow: ROLL (kat obrotu wokół osi X), PITCH (kat obrotu wokół osi Y) i YAW (kat obrotu wokół osi Z). W tym przykładzie celowo uzylem biblioteki żeby nie utrudniać zbytnio tej funkcji. Po odczytaniu katow biblioteka już za nas odpowiednio laczy ze sobą katy odczytane za pomocą akcelerometru oraz żyroskopu za pomocą filtru komplementarnego. Jak działa filtr komplementarny? Uproszcze tutaj sprawę jest to kombinacja tkz. LPF i HPF (lowa pass filter i high pass filter). Tzn. Poniewaz wiemy ze na skutek calkowania pojawia sie zjawisko gyro drift a na skutek zbyt dużych szumow dla acceleration angles nie możemy brac tylko jednej z tych wartości musismy je jakość sprytnie zsumować tak żeby w rezultacie wynik był jak najbliższy rzeczywistości. Dlatego w praktyce przepuszczamy w wiekszsosci Kat obliczony od gyro, a w mniejszości przepuszcamy kod od acc, tak tylko żeby korygować warytosci katow z gyro (tzn dajemy HPF do gyro i LPF do accelerometer), w rezultacie mamy: 0.99 \* gyroAngle + 0.01 \* accAngle. W DUZYM SKROCIE… Jeżeli chcesz wiedzieć więcej proszę poczytaj bo temat jest naprawdę obszerny… Wkleje tutaj jeszcze kod starej funkcji do odczytu katow (uzywajacy wlasnie tego algorytmu tyle ze 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 katow eulera tez moglbym się rozpisywać bardzo długo, ale nie jest to konieczne, my po prostu używamy wzorow tylko i az 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 kat roll, pitch, yaw były rowne zero by mieć prawidłowy punkt odniesienia. Jak to zrobić? SKALIBROWAC 😊 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;

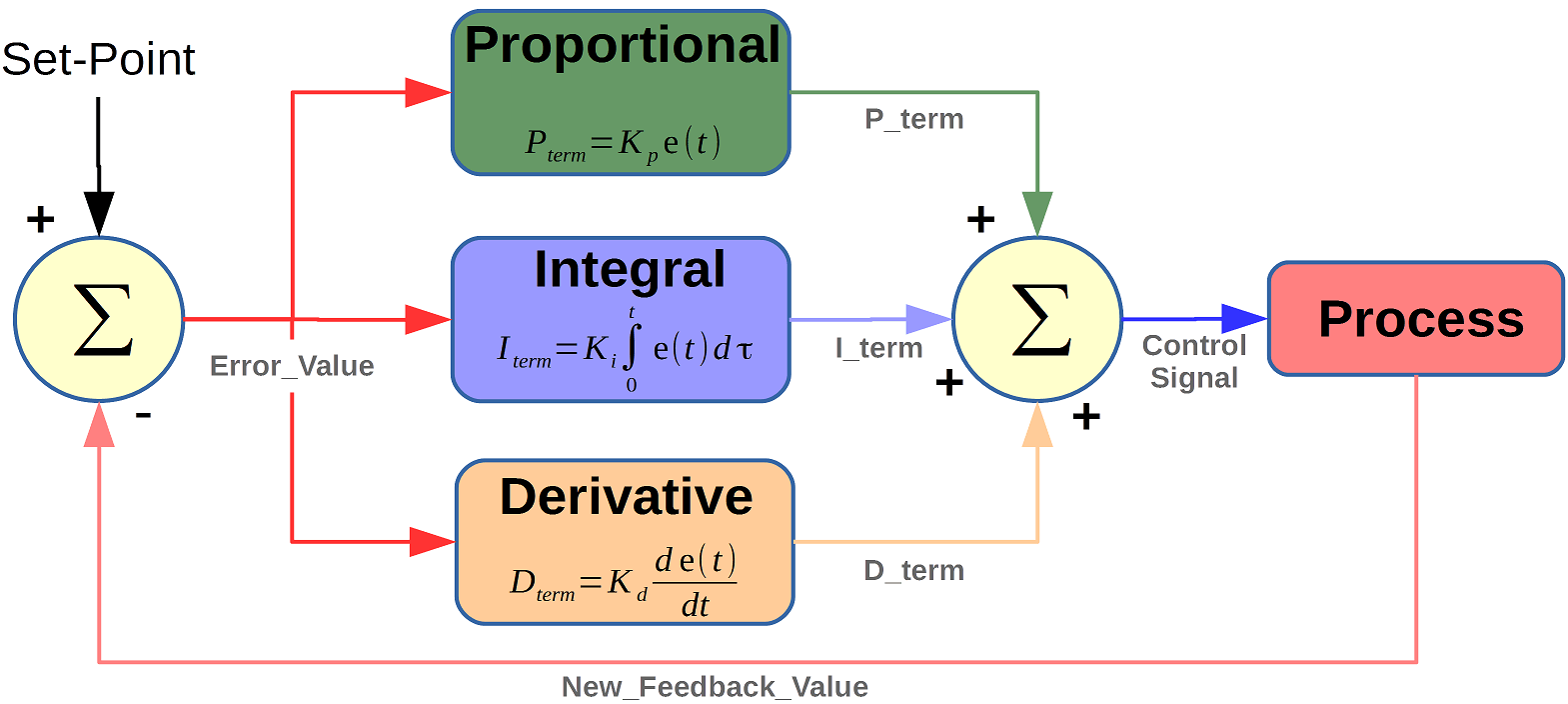
}

}

## 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 wiec wzory:

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

Caly schemat dzialania PID swietnie 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 :/ (sluzy 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 cosnt to dodatkowa stala wzocnienia uzywana 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 która 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 juz 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 ta 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 😊

# Podsumowanie

Jeżeli masz jakiekolwiek zastrzeżenia co do projektu, myślisz że coś można poprawić, proszę daj nam znać ( skontaktuj się z: Krzysztofem (Krzychu) lub Tomkiem (Tomek)). Jak na razie dron nie działa według zamierzeń. W komputerze na wartościach wszystko jest idealni, jednak, gdy odpalamy drona w realu wszystko się chrzani i PID szlag trafia, dron leci ale nie potrafi się w prawidłowy sposób regulować mimo ze podczas symulacji wszystko jest ok. Podejrzewamy iż jest to spowodowane dużymi ograniczeniami które daje platforma arduino Nano i jej moc obliczeniowa, powinniśmy zdecydowanie ten projekt zrobić na czymś mocniejszym typu RPI Pico lub STM32L lub F obojętne. Projekt nadal jest w realizacji.