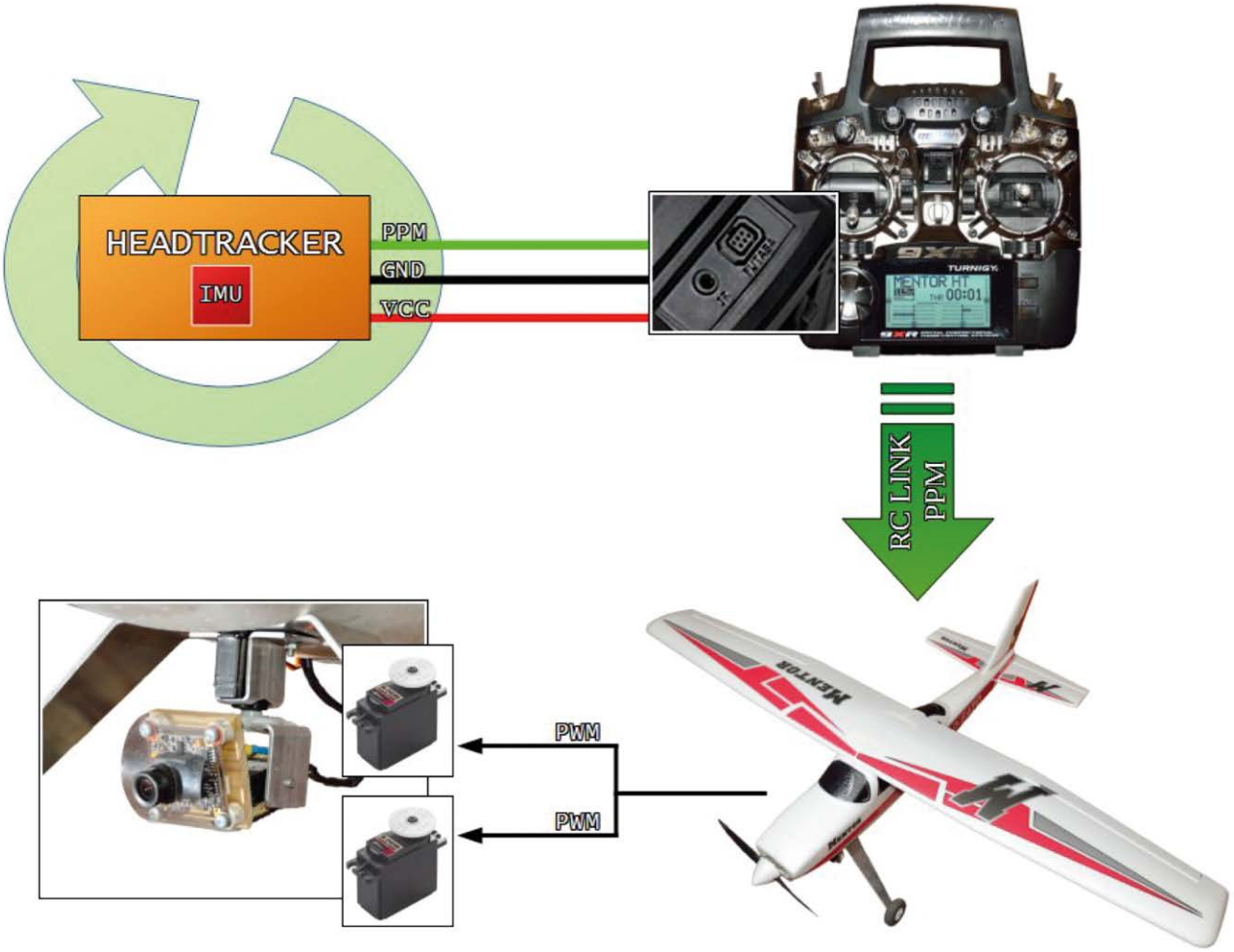
FPV Headtracker inercyjny sterownik ruchu (1)

Postęp technologiczny sprawił, iż wynalazki, które kilkadziesiąt lat temu budziły podziw i zachwyt, dziś są w normalnym użytkowaniu. Na przykład, w napięciu oglądało się filmy, w których pokazywane były zaawansowane „zachodnie technologie". Przykładem może być system Helmet-Mounted Display stosowany w lotnictwie wojskowym już w latach siedemdziesiątych. Systemy HMD szybko stały się standardem i nadal umożliwiają pilotowi podglądanie informacji na ekranach wbudowanych w hełm lotniczy, a oprócz tego dają też możliwość sterowania wybranymi funkcjami maszyny. Znamy systemy sterowania karabinem podczepionym do śmigłowca Apache za pomocą ruchów głowy pilota, gdzie cel jest namierzany bez konieczności odrywania uwagi od pilotażu maszyny. Kolejnym etapem rozwoju tej technologii jest świat „Artificial Reality" (AR), który możemy obserwować poprzez specjalne okulary, jak np. Oculus Rift, lub na ekranach telefonów komórkowych. Użytkownicy tych urządzeń wiedzą, iż efekt jest uzyskiwany dzięki wykorzystaniu pozycjonowania i żyroskopu, który mierzy kąty pochylenia w danej pozycji. Niewielu z nich jednak wie jak to w istocie działa. Dzięki miniaturyzacji i stosunkowo niskiej cenie technologii MEMS możemy zbudować podobny system we własnym warsztacie.

Rekomendacje: *urządzenie przyda się nie tylko do modelu, ale również do innych zastosowań, takich jak: monitoring obiektu, robotyka, urządzenia dla osób niepełnosprawnych i innych.*

Prezentowane przeze mnie rozwiązania mają charakter hobbystyczny, stąd wiele osób mądrzejszych ode mnie odbierze je jako niekompletne lub nadmiernie uproszczone. Sądzę jednak, iż taka forma prezentacji będzie lepiej przyswajalna dla osób, które podobnie jak ja, na co dzień nie zajmują się elektroniką lub modelami matematycznymi. Mam nadzieję, że uda mi się w prosty sposób pokazać krok po kroku zasadę działania systemu sterowania typu Head Tracking (podążanie za ruchem głowy). Istotne jest także to, że przedstawione techniki programowania oparte na timerach programowych oraz algorytmy takie jak filtr Kalmana będą miały zastosowanie w wielu innych projektach.

Artykuł ten jest przeznaczony dla osób mających choćby minimalną wiedzę na temat programowania w języku C systemów wbudowanych. Tak więc zaczynamy budować własny headtracker do zastosowań w lotnictwie, jednak tym razem nie będziemy sterowali karabinem maszynowym, ale kamerą podczepioną do samolotu RC.



Rysunek 1. Schemat działania Headtrackera

Zasada działania

Schemat działania systemu pokazano na **rysunku 1**. Kamera podczepiona do samolotu będzie sterowana w 2 płaszczyznach: oś pionowa Z (rozglądanie się w prawo i lewo) oraz oś poprzeczna X (spoglądanie do dołu i do góry). Pomijamy wzdłużną oś Y, ponieważ nie będziemy pochylali kamery na boki. Mimo tego finalne rozwiązanie umożliwi w prosty sposób dodanie tej funkcjonalności. Jest ona użyteczna w przypadku multikopterów i śmigłowców, które w odróżnieniu od samolotów poruszają się wzdłuż osi poprzecznej. Zatem zbudujemy urządzenie pomiarowe, które przyczepione do gogli z wbudowanym wyświetlaczem z dużą dokładnością wskaże nam kierunek patrzenia oraz pochylenie głowy. Informacje te przekażemy poprzez sterownik do urządzeń wykonawczych, które ustawią kamerę w żądanym kierunku.

Do osiągnięcia celu wykorzystamy cyfrowy układ IMU (Inertial Measurement Unit) zawierający żyroskop, akcelerometr oraz magnetometr. Jak się przekonamy w dalszej części artykułu, wszystkie 3 elementy muszą ze sobą współgrać, aby dokładne określić zwrot i kierunek w przestrzeni. W sprzedaży dostępnych jest wiele układów IMU, jednak do naszych celów musimy zaopatrzyć się w układ zawierający 3-osiowy żyroskop, 3-osiowy akcelerometr oraz 3-osiowy magnetometr. Krótko przypomnę, iż żyroskop mierzy prędkość kątową (nie mylić z pomiarem kątów), akcelerometr przyspieszenie, a magnetometr wielkość pola magnetycznego. Przypuszczam, że użycie aż 3 pomiarów do uzyskania informacji o kątach, które możemy odczytać z samego żyroskopu może budzić wątpliwości. Niestety, rozwiązanie oparte jedynie na żyroskopie jest nieefektywne i nie jest zależne od jakości (i ceny) żyroskopu. W naszym wypadku możemy zaopatrzyć się nawet w tańszy układ, mający rozdzielczość niższą niż 16 bitów, co będzie bez znaczącego wpływu na efekt finalny. Stracimy przy tym na płynności poruszania kamerą, lecz błędy pomiarów zostaną zniwelowane dzięki wykorzystaniu odpowiedniego filtrowania. Układy o rozdzielczości 16-bitowej mogą okazać się konieczne przy budowie różnego rodzaju stabilizatorów.

Nasze urządzenie zbudujemy na bazie układu LSM9DS0, który jest dostępny jako moduł z zamontowanym układem. Pojawia się kolejne pytanie: w jakim celu i w jaki sposób połączyć odczyty żyroskopu, akcelerometru i kompasu? Wykorzystamy do tego celu Filtr Komplementarny oraz Filtr Kalmana. Definicja filtru Kalmana dostępna w Wikipedii jest następująca: „algorytm rekurencyjnego wyznaczania

Podstawowe informacje:

- Demonstracja działania: *http://goo.gl/Wqhefq*
- Bazuje na akcelerometrze LM9S oraz na mikrokontrolerze 8-bitowym (ATmega8 lub ATmega168).
- Współpracuje z dowolną aparaturą do zdalnego sterowania.
- Wykorzystanie dwóch kanałów do sterowania ruchem dwóch serwomechanizmów.
- Wariant podstawowy jest przeznaczony do modeli samolotów.

Dodatkowe materiały na FTP:

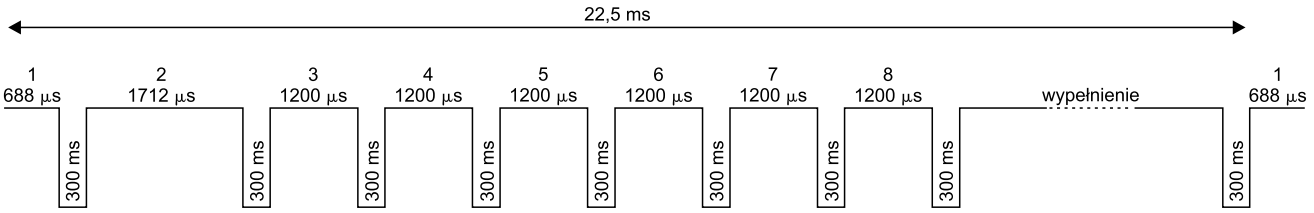
ftp://ep.com.pl, user: 66465, pass: td79fgh6

- wzory płytek PCB

Projekty pokrewne na FTP:
(wymienione artykuły są w całości dostępne na FTP)

AVT-5491	Czujnik inercyjny LSM9DS0 oraz jego zastosowanie praktyczne (EP 2/2015)
AVT-1806	Detektor drgań (EP 7/2014)
AVT-5393	ASO – Automatyczny system ostrzegania (EP 4/2013)
AVT-5387	gLogger – 3-osiowy rejestrator przyśpieszenia (EP 3/2013)
AVT-5223	Kieszonkowy akcelerometr (EP 2/2010)
Projekt 132	Miernik przyśpieszenia (EP 8/2005)

* Uwaga:
Zestawy AVT mogą występować w następujących wersjach:
AVT xxxx UK to zaprogramowany układ. Tylko i wyłącznie. Bez elementów dodatkowych.
AVT xxxx A płytka drukowana PCB (lub płytki drukowane, jeśli w opisie wyraźnie zaznaczono), bez elementów dodatkowych.
AVT xxxx A+ płytka drukowana i zaprogramowany układ (czyli połączenie wersji A i wersji UK) bez elementów dodatkowych.
AVT xxxx B płytka drukowana (lub płytki) oraz komplet elementów wymienionych w załączniku pdf
AVT xxxx C to nic innego jak zmontowany zestaw B, czyli elementy wlutowane w PCB. Należy mieć na uwadze, że o ile nie zaznaczono wyraźnie w opisie, zestaw ten nie ma obudowy ani elementów dodatkowych, które nie zostały wymienione w załączniku pdf
AVT xxxx CD oprogramowanie (nieczęsto spotykana wersja, lecz jeśli występuje, to niezbędne oprogramowanie można ściągnąć, klikając w link umieszczony w opisie kitu)
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! (UK, A, A+, B lub C). *http://sklep.avt.pl*



Rysunek 2. Standardowa ramka PPM

minimalno-wariancyjnej estymaty wektora stanu modelu liniowego dyskretnego układu dynamicznego na podstawie pomiarów wyjścia oraz wejścia tego układu". Mogę obiecać, że to trudne do wypowiedzenia jednym tchem zdanie będzie zrozumiałe po przeczytaniu tego artykułu.

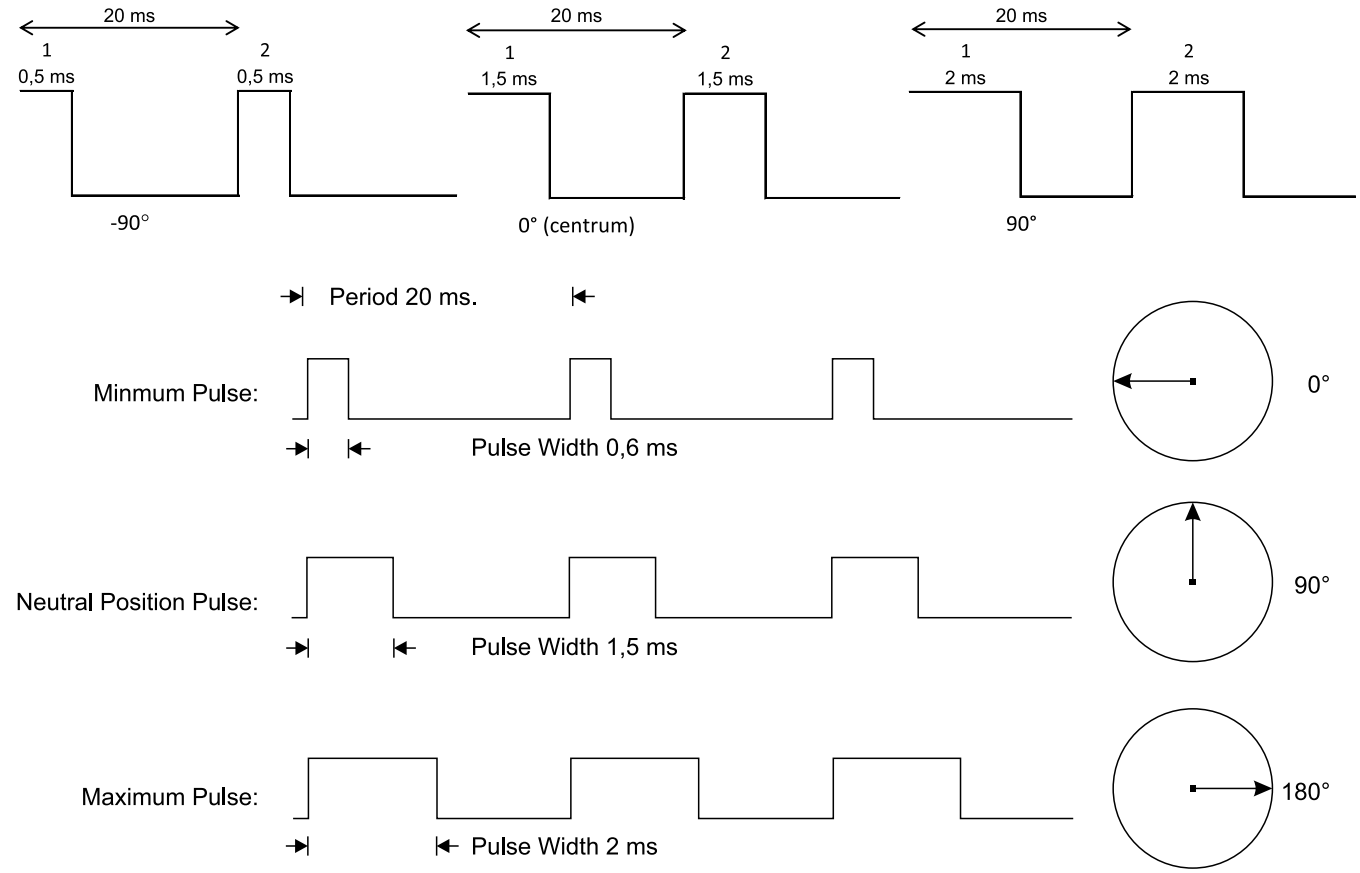
Kolejny element układanki to przekaźnik, który będzie wysyłał sygnały sterujące do urządzeń wykonawczych. W samolocie zdalnie sterowanym jest to nadajnik radiowy, poprzez który będziemy sterować serwami na pokładzie samolotu. Nadajnik będzie odbierał cyfrowy sygnał w modulacji PPM (Pulse Position Modulation) generowany przez nasz headtracker i przekazywał go drogą radiową do odbiornika znajdującego się w samolocie. Zatem niezbędne jest zapoznanie się z charakterystyką sygnału PPM wykorzystywanego w kontrolerze RC. Istotne dla nas informacje to:

- całkowity czas ramki PPM: 22500 µs,
- czas pauzy pomiędzy kanałami: 300 µs,
- czas dla pozycji centralnej: 1200 µs,
- czasy skrajnych wychyleń: ±512 µs (czyli od 668 µs do 1712 µs).

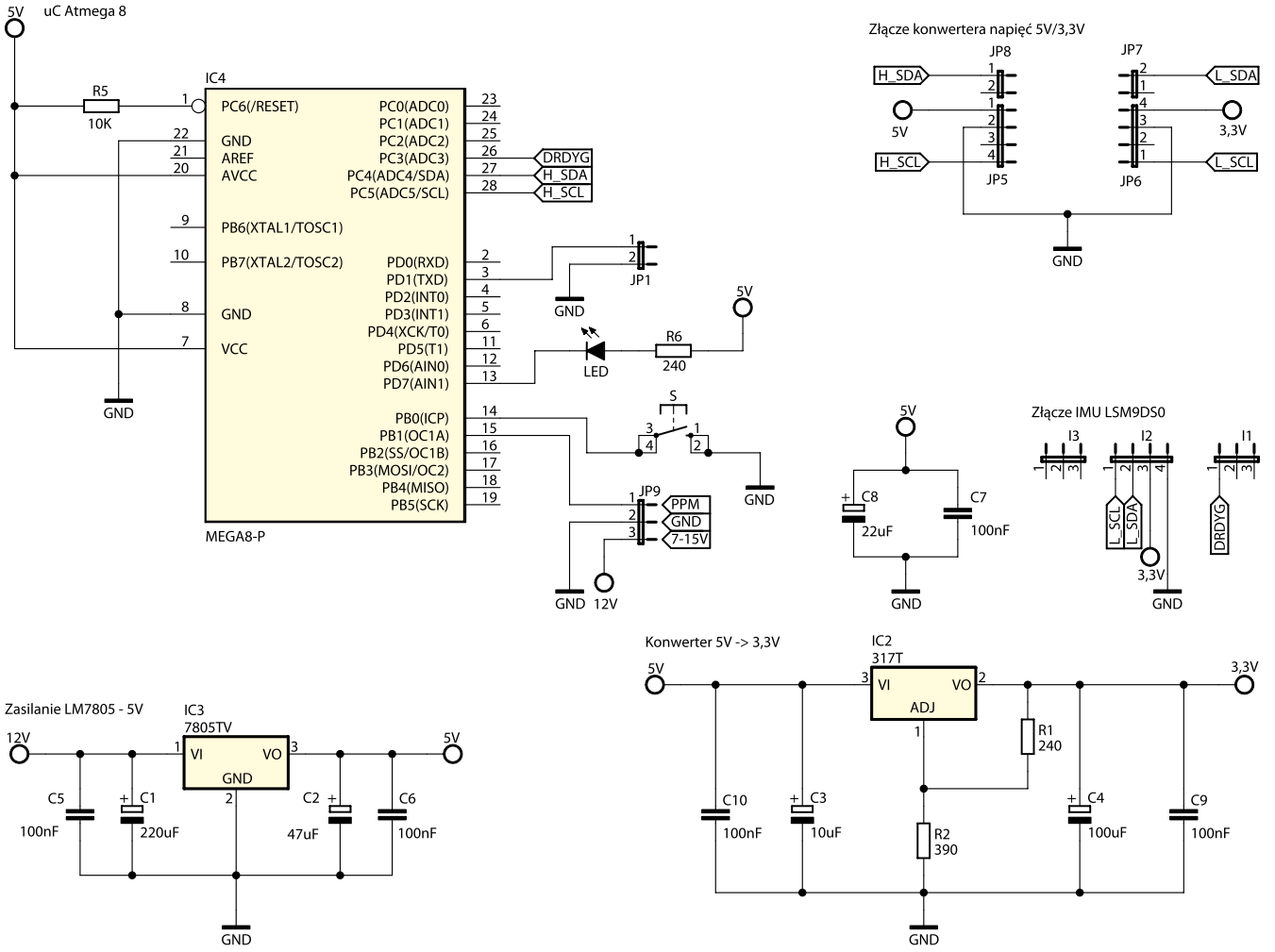
Czas pojedynczego kanału należy rozumieć jako sumę czasów poziomu wysokiego i pauzy, czyli przykładowo: dla pozycji centralnej czas wynosi 1500 µ (1200 µs + 300 µs). Na **rysunku 2** pokazano przykładową, 8-kanałową ramkę PPM, w której kanały 1 i 2 zostały ustawione w pozycjach skrajnych, natomiast pozostałe pozostawione zostały w pozycjach centralnych. Niezależnie od czasów poszczególnych kanałów, całkowita długość ramki pozostaje stała poprzez odpowiednie dobranie czasu wypełnienia (sygnał synchronizacji). Serwomechanizm przyłączony do kanału 1 zostanie wychylone w jedną skrajną pozycję, na do kanału 2 w przeciwną skrajną pozycję. Pozostałe serwomechanizmy zostałyby ustawione w pozycji centralnej. My będziemy sterowali tylko dwoma serwomechanizmami poprzez dowolnie wybrane dwa spośród ośmiu kanałów. Do pojedynczego serwomechanizmu nie dociera pełna ramka PPM, a jedynie sygnał PWM (Pulse Width Modulation). Konwersją ramki PPM do PWM i rozesłaniem sygnałów do poszczególnych serwomechanizmów zajmuje się odbiornik zdalnego sterowania. Przykładowe sygnały PWM zostały pokazane na **rysunku 3**. Dalsze objaśnianie sygnałów PPM oraz PWM wykracza poza zakres tego artykułu.

Upraszczając, Headtracker będzie zamieniał dane odczytane z IMU na sygnał PPM. Do tego celu wystarczy 8-bitowy mikrokontroler, na przykład ATmega8. Zawiera on wystarczająco dużo pamięci Flash, by pomieścić kod napisany w języku C, a przy taktowaniu 8 MHz jest w stanie wykonywać ponad 200 pomiarów na sekundę, co daje duży zapas mocy obliczeniowej.

Nie obejdzie się bez dokumentacji pdf wykorzystywanych przez nas układów, więc konieczne jest pobranie odpowiednich dokumentów z Internetu. W kolejnej części artykułu będę odwoływał się do noty LSM9DS0 oraz ATmega8, które są dostępne na stronach *www.sparkfun.com* oraz *www.atmel.com*. Umiejętność korzystania z dokumentacji dostarczanej przez producentów układów jest niezbędna do wykonania projektu. By w pełni zrozumieć ten tekst,



Rysunek 3. Przykładowe sygnały PWM sterujące serwomechanizmami



Rysunek 4. Schemat połączeń IMU, CPU i konwertera napięć logicznych

zalecam na bieżąco weryfikowanie dalszych rozważań z dokumentacją. Jest to często praca żmudna i trudno zrozumiała bez praktycznych przykładów, ale mam nadzieję, że wszystko będzie jasno opisane. Podkreślam, że wartości rejestrów (układów scalonych) użytych w programie wynikają wprost z dokumentacji i tam należy poszukiwać szczegółowych wyjaśnień.

Zacznijmy od schematu połączeń pokazanego na **rysunku 4**. Projekt zakłada użycie mikrokontrolera, diody sygnalizacyjnej, przycisku służącego do zerowania pozycji i kalibracji. Wykorzystamy także wspomniany układ LSM9DS0, który będzie przyłączany poprzez gniazdo goldpin. Z uwagi na fakt, iż IMU pracuje pod napięciem 3,3 V a procesor 5 V, konieczne będzie użycie konwertera napięć logicznych dla magistrali I²C. Użyjemy gotowego konwertera dostępnego w sklepie *botland.com.pl*, który także podłączymy do Headtrackera poprzez gniazdo goldpin. Taka konstrukcja pozwoli nam na wykorzystanie tych układów w innych projektach. Podpinamy zasilanie IMU oraz linie SDA i SCL poprzez konwerter do pinów PC4 i PC5. Opcjonalnie dołączamy wyjście DRDYG (Gyroscope data ready) bezpośrednio do pinu PC3. Pin DRDYG może zostać skonfigurowany w IMU jako wyjście przerwania wyzwalanego, gdy dostępne są nowe dane do odczytu z żyroskopu. Przerwanie to może zostać użyte przy włączonym trybie FIFO. Sygnał wyjściowy PPM będzie generowany na pinie PB1, który jest jednocześnie wyjściem OC1A. Skorzystamy przy tym ze sprzętowego generowania sygnału FastPWM, by w procedurze przerwania utworzyć PPM. Konieczne będzie także wyprowadzenie wyjścia TXD, które posłuży nam do debugowania kodu poprzez UART.

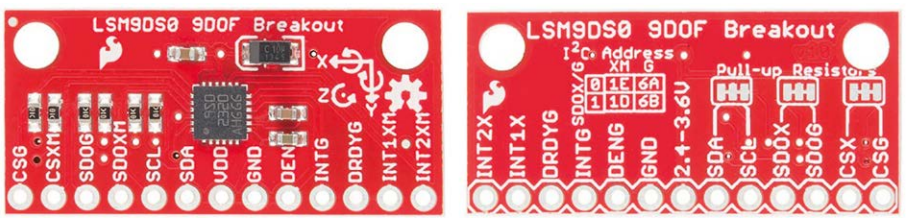
Kluczową informacją jest rozkład osi w układzie IMU, przedstawiony na **rysunku 5**. Należy także ustalić położenie układu w finalnym urządzeniu. W naszym przypadku będzie on odwrócony, co zostanie uwzględnione przy ustalaniu kierunku obrotów.

Zgodnie ze schematemz rys. 4, przystępujemy do inicjalizacji CPU i IMU. Poprzez Fusebity włączamy wewnętrzny oscylator 8 MHz. Uruchamiamy 2 timery sprzętowe. Pierwszy z nich (8-bit Timer/Counter2) będzie odpowiedzialny za odliczanie czasu trwania pętli głównej, dzięki czemu zdobędziemy informację o częstotliwości odczytu danych z IMU. Użyjemy go również do obsługi operacji asynchronicznych, takich jak obsługa przycisku i miganie diodą oraz do operacji zerowania pozycji. Inicjalizujemy przerwanie TIMER2_COMP_vect, które będzie się wykonywało co 1 ms:

```
//Konfiguracja Timer2 8-bit
//CTC; 64 prescaler
TCCR2 = (1<<WGM21) | (1<<CS22);
//Przerwanie co 1ms
OCR2 = 125;
```

Konfigurujemy 16-bit Timer/Counter1, który będzie odpowiedzialny za generowanie PPM w przerwaniu TIMER1_COMPA_vect:

```
//Konfiguracja Timer1 16-bit
```



Rysunek 5. Oznaczenie osi X, Y, Z oraz adresacja magistrali I²C

```
//Clear or Set OC1A on Compare Match,
WGM10,11,12,13-Fast PWM
TCCR1A = (1<<COM1A1) | (1<<WGM11) | (1<<WGM10);
//8 prescaler
TCCR1B = (1<<WGM12) | (1<<CS11) | (1<<WGM13);
```

Timer1 wyzwala przerwanie przy przepełnieniu rejestru OCR1A. Przy pomocy tego rejestru możliwe będzie sterowanie sygnałem FastPWM w taki sposób, by uzyskać ramkę PPM. Włączamy przerwanie Compare Match dla Timer1 i Timer2:

```
TIMSK = (1<<OCIE2) | (1<<OCIE1A);
```

Za pomocą kilku podstawowych makr definiujemy pozostałe porty:

```
#define PIN_LOW(PORT,PIN) PORT &= ~(1<<PIN)
#define PIN_HIGH(PORT,PIN) PORT |= (1<<PIN)
#define LED_OFF PIN_HIGH(LED_PORT,LED_PIN)
//Konfiguracja OUT PPM
PIN_HIGH(PPM_DDR,PPM_PIN);
PIN_HIGH(PPM_PORT,PPM_PIN);
//Konfiguracja IN DRDYG
PIN_LOW(DRDYG_DDR,DRDYG_PIN);
PIN_LOW(DRDYG_PORT,DRDYG_PIN);
//LED OUT
PIN_HIGH(LED_DDR,LED_PIN);
LED_OFF;
//Button IN
PIN_LOW(BTN_DDR,BTN_PIN);
PIN_HIGH(BTN_PORT,BTN_PIN);
```

Następnie inicjujemy komunikację I²C z IMU:

```
TWBR = 32; //32:100KHz @ F_CPU=8MHz
TWSR = (0<<TWPS1) | (0<<TWPS0); //Prescaler: 1
```

Częstotliwość magistrali I²C ustalamy na 100 kHz. Zgodnie z dokumentacją budujemy funkcje do komunikacji z CPU i przystępujemy do inicjalizacji IMU.

Układ składa się z 2 osobnych modułów typu slave I²C dostępnych na różnych adresach. Adresy te możemy w pewnym stopniu konfigurować poprzez sterowanie poziomem logicznym na wejściach SDOX i SDOG, które zostały ustawione przez producenta. Moduły oznaczone zostały literami **G** (żyroskop) oraz **XM** (akcelerometr i magnetometr). Zgodnie z rys. 5, domyślnie adresy I²C to 0x1D dla XM oraz 0x6B dla G. Inicjalizacja IMU odbywa się poprzez zapis określonych wartości do rejestrów układu. Poniższy kod przedstawia konfigurację poszczególnych bitów w rejestrach konfiguracyjnych. By go dobrze zrozumieć, konieczne jest zapoznanie się ze szczegółami opisanymi w dokumentacji, w której każdy bit został szczegółowo opisany.

```
/* GYRO */
//380Hz/100 Cutoff | XYZ Enable
i2c_write_byte(I2C_ADR_G,CTRL_REG1_G,0b10111111);
//High-pass disable
i2c_write_byte(I2C_ADR_G,CTRL_REG2_G,0b00100000);
//Date-ready on DRDY_G (dostępne w trybie FIFO)
i2c_write_byte(I2C_ADR_G,CTRL_REG3_G,0b00001000);
//2000 DPS
i2c_write_byte(I2C_ADR_G,CTRL_REG4_G,0b00100000);
//FIFO disable
i2c_write_byte(I2C_ADR_G,CTRL_REG5_G,0b00000000);
/* XM - Accel / Magnet / Temp */
//FIFO disable
i2c_write_byte(I2C_ADR_XM,CTRL_REG0_XM,0b00000000);
//Acceleration data rate 100Hz | Continuous Update
| XYZ Enable
i2c_write_byte(I2C_ADR_XM,CTRL_REG1_XM,0b01100111);
//Antialias 773Hz | Accel full-scale +/-4g
i2c_write_byte(I2C_ADR_XM,CTRL_REG2_XM,0b00001000);
//Accel data-ready on INT1_XM (DRDYA)
i2c_write_byte(I2C_ADR_XM,CTRL_REG3_XM,0b00000100);
//Magnet data-ready on INT2_XM (DRDYM) <- disable
i2c_write_byte(I2C_ADR_XM,CTRL_REG4_XM,0b00000000);
//Temp disable | Magnetic resolution Low | Magnetic
data rate 100Hz
i2c_write_byte(I2C_ADR_XM,CTRL_REG5_XM,0b00010100);
//Magnetic full scale +/-4 gauss
i2c_write_byte(I2C_ADR_XM,CTRL_REG6_XM,0b00100000);
//Sensor Continuous-conversion mode
i2c_write_byte(I2C_ADR_XM,CTRL_REG7_XM,0b10000000);
```

Przedstawioną konfigurację IMU należy traktować jako przykładową. Nic nie stoi na przeszkodzie, by użyć innej, dostosowanej do własnych potrzeb. Nie ma także potrzeby wyjaśniania w tym miejscu każdej z powyższych opcji, ponieważ skończyłoby się to skopiowaniem dokumentacji. Chciałbym jedynie zwrócić uwagę na kilka kluczowych aspektów:

1. Wyłączamy filtrowanie sprzętowe. Nie jest ono istotne, ponieważ tym zajmiemy się w sposób programowy.
2. Wyłączamy kolejkowanie FIFO. W danym momencie interesuje nas bieżący stan układu. Nie musimy się obawiać „zgubionych" czy nieodczytanych wartości z IMU, stąd ich kolejkowanie jest zbędne.
3. Ustalamy czułość żyroskopu na 2000 stopni na sekundę (DPS). Bardzo wątpliwe jest, aby ktoś potrafił kręcić głową z większą prędkością.
4. Ustalamy częstotliwość generowania danych z żyroskopu na 380 Hz. CPU będzie wykonywał pętlę główną w tempie około 200 Hz, dlatego jest zapewniona pewna nadwyżka danych od strony IMU. Należy pamiętać, iż w trybie FIFO układ informuje o dostępności nowych danych do odczytu poprzez ustawienie poziomu wysokiego na pinie DRDYG. W wypadku wyłączonego kolejkowania należy zapewnić odpowiednią pauzę pomiędzy kolejnymi odczytami w sposób programowy.
5. Ustalamy częstotliwość akcelerometru na 100 Hz. Większa ilość danych wydaje się zbędna i prowadzi do zwiększenia szumów spowodowanych drganiami układu. Bardziej istotne są dla nas odczyty z żyroskopu.
6. Ustalamy maksymalne mierzone przeciążenie na ±4g. Wyższa czułość na poziomie ±2g powoduje zwiększenie rozdzielczości odczytów, co wprowadza dodatkowe zakłócenia.
7. IMU posiada wbudowany termometr, który nam się nie przyda.
8. Ustalamy czułość magnetometru na ±4 Gaussy. Magnetometr jest niestety podatny na zakłócenia zewnętrzne. Można je zaobserwować poprzez przybliżenie ferromagnetyków do IMU na odległość 2…3 cm. Przy projektowaniu PCB należy mieć na uwadze to, aby układ IMU znajdował się w bezpiecznej odległości od elementów, które mogą zakłócać jego pracę.
9. Wyłączamy domyślny tryb stand-by oraz włączamy odczyty w osiach X, Y i Z.

Zainicjowany układ zaczyna udostępniać dane poprzez magistralę I²C, które możemy odczytać wprost z jego 8-bitowych rejestrów. Spróbujmy odczytać dane prędkości kątowych z żyroskopu. Według dokumentacji informacje zapisane są w sześciu komplementarnych rejestrach o adresach: *OUT_X_L_G (28h), OUT_X_H_G (29h), OUT_Y_L_G (2Ah), OUT_Y_H_G (2Bh), OUT_Z_L_G (2Ch), OUT_Z_H_G (2Dh)*. W celu odczytu wartości X musimy odczytać dwie 8-bitowe wartości spod adresów 28h i 29h oraz dokonać ich „złożenia" do liczby 16-bitowej. Z dokumentacji dowiemy się także, jaka jest kolejność bajtów oraz, że jest możliwy odczyt wielu bajtów w jednym transferze I²C. Dzięki temu możemy zbudować funkcję odczytującą 6 wartości jednocześnie, które następnie zamienimy na liczby 16-bitowe:

```
uint8_t i2c_read_sequence(const uint8_t sla, const
uint8_t adr, uint8_t dest[], const uint8_t cnt) {
  //Start
  if (i2c_start() != 0x08) { i2c_stop(); return 0; }
  //Wysłanie adresu urządzenia SLAVE + WRITE
```

```
  if (i2c_write_data((sla<<1) |
I2C_WRITE) != 0x18) { i2c_stop();
return 0; }
  //Wysłanie subadresu +
autoincrement
  if (i2c_write_data(adr |
(1<<7)) != 0x28) { i2c_stop();
return 0; }
  //Repeated Start
  if (i2c_start() != 0x10) { i2c_
stop(); return 0; }
  //Wysłanie adresu urządzenia
SLAVE + READ
  if (i2c_write_data((sla<<1) |
I2C_READ) != 0x40) { i2c_stop();
return 0; }
  //Odczyt danych
  for (uint8_t i=0; i<cnt; i++) {
    if (i < cnt-1) {
      //Odczyt z ACK
      if (i2c_read_data_ack(&dest[i]) != 0x50) {
i2c_stop(); return 0; }
    }
    else {
      //Odczyt ostatniego bajtu z NACK
      if (i2c_read_data_nack(&dest[i]) != 0x58) {
i2c_stop(); return 0; }
    }
  }
  i2c_stop();
  return 1;
}
```

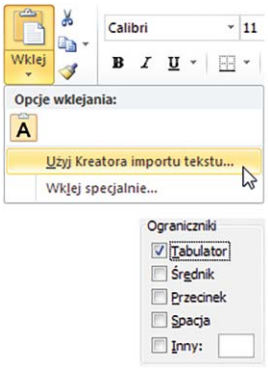
Po więcej informacji odsyłam do Elektroniki Praktycznej 02/2015, w której był zamieszczony opis komunikacji I²C na przykładzie LSM9DS0. Wszystkie kody błędów użyte w funkcji, jak i sama funkcja powstały na bazie dokumentacji LSM9DS0 i ATmega8. Jak widać błędy są weryfikowane na każdym etapie komunikacji, a w przypadku zaistnienia problemu odczyt jest dyskwalifikowany. Praktyczne wykorzystanie powyższej funkcji wygląda następująco:

```
uint8_t IMU_read(const uint8_t sla, const uint8_t
addr, double tab[]) {
  uint8_t data[6];
  if (i2c_read_sequence(sla, addr ,data, 6)) {
    tab[0] = (data[1]<<8) | data[0];  //X
    tab[1] = (data[3]<<8) | data[2];  //Y
    tab[2] = (data[5]<<8) | data[4];  //Z
    return 1;
  }
  else return 0;
}
```

Poprzez zmienną *sla* przekazujemy adres żyroskopu na magistrali I²C, a *addr* to adres pierwszego z szeregu rejestrów, od którego zaczynamy pobierać dane. Funkcja zwraca w tabeli wartości 6 rejestrów, które wystarczy połączyć poprzez wykonanie kilku przesunięć bitowych. W analogiczny sposób pobierane są dane z akcelerometru i magnetometru. Trzeba pamiętać, że IMU zwraca liczby całkowite ze znakiem, jednak są one konwertowane na typ zmiennoprzecinkowy, gdyż w dalszych obliczeniach wykorzystywany będzie ten typ danych.

Metodyka analizy danych

Skoro potrafimy już odczytywać pomiary z IMU, to nadszedł czas, by się im przyjrzeć i zrozumieć, co mamy do dyspozycji. Każda grupa liczb (GX, GY, GZ), (AX, AY, AZ) oraz (MX, MY, MZ) wymaga oddzielnego wyjaśnienia. Do wykonania kalibracji układów konieczne jest



	A	B	C	D	E	F	G	H	I	J	K	L
1	tsek	lp.	gfx	gfy	gfz	afx	afy	afz	mfx	mfy	mfz	millis
2	0,053	1	0	0	0	260	-301	8304	-21	1029	6050	53
3	0,107	2	0	0	0	168	-253	8274	-78	993	6066	54
4	0,159	3	0	0	0	196	-260	8271	-23	1002	6085	52
5	0,213	4	0	0	0	313	-279	8298	-14	1014	6082	54
6	0,267	5	0	0	0	226	-276	8303	-29	997	6073	54
7	0,319	6	0	0	0	242	-302	8308	-3	1050	6059	52
8	0,372	7	0	0	0	214	-262	8349	-16	1033	6056	53
9	0,425	8	0	0	0	421	-296	8444	41	1015	6073	53
10	0,478	9	0	0	0	348	-254	8486	28	1097	6063	53
11	0,531	10	0	0	0	297	-270	8363	-55	1034	6076	53
12	0,585	11	0	0	0	401	-294	8408	-32	1034	6078	54
13	0,640	12	0	0	0	332	-208	8475	-1	1044	6059	55
14	0,694	13	0	0	0	196	-232	8331	-33	1028	6074	54

Rysunek 6. Przykład analizy danych z UART w Excelu

uruchomienie połączenia przez UART, które posłuży do wysyłania danych na terminal PC. Do analizy odczytów z IMU na komputerze PC wystarczy Excel lub podobny arkusz kalkulacyjny. Dzięki opisanej niżej metodologii nie będzie konieczności programowania specjalnych narzędzi do komunikacji z portem COM. Wystarczy darmowy program putty (*www.chiark.greenend.org.uk*), który umożliwia odczyt danych przez port szeregowy PC. Wybór programu terminala jest oczywiście kwestią własnych preferencji. W ustawieniach zmieniamy maksymalną liczbę wyświetlanych linii z domyślnych 200 na 5000 (Putty→Window→Lines of scrollback). Następnie wysyłany na terminal wymagane dane, na przykład prędkości kątowe z żyroskopu. Każdą wartość oddzielamy znakiem tabulatora *\t*, a nową linię oznaczamy poprzez wysłanie do terminala kolejno znaków *\r* i *\n*. Wykonujemy kilka ruchów IMU, a gdy ekran zapełni się danymi z kilkuset (lub kilku tysięcy) próbek, klikamy prawym przyciskiem myszy na pasek okna programu putty, wybieramy *Copy All to Clipboard* i całość wklejamy do Excela poprzez kreatora importu. Warto zaznaczyć, że liczby zmiennoprzecinkowe najczęściej przesyłamy przez UART w postaci liczb całkowitych, przez co obniża się ich precyzja. W takim wypadku przed wysłaniem liczby na terminal należy pomnożyć ją przykładowo przez 1000 (metoda znana jako skalowanie liczb całkowitych). W ten sposób uzyskamy dokładność 3 miejsc po przecinku. Jak pokazano na **rysunku 6**, mamy 9 kolumn (C-K) z danymi z IMU. W Excelu dodana została także kolumna L *millis*, zawierająca czas wykonania pętli w milisekundach obliczony przez CPU w *Timer2*. Na tej podstawie dodana została kolumna A *tsek*, która zawiera upływający czas serii danych w sekundach. Teraz możemy wizualizować dane w postaci przejrzystych wykresów. Dodatkowo możemy wprowadzić 2 zmienne *min* i *max*, w których będziemy zapamiętywali odczyty ekstremalne:

```
if (RAW_X < min) min=RAW_X;
if (RAW_X > max) max=RAW_X;
```

Te zmienne także wysyłamy na ekran PC.

Kalibracja żyroskopu

Teoretycznie, nieruchomy układ IMU powinien wskazywać zerowe prędkości kątowe, jednak w praktyce żyroskop zwraca pewne wartości. Oznacza to konieczność kalibracji, a więc doprowadzenie do sytuacji, w której nieruchomy układ nie będzie wykazywał ruchu. Żyroskop jest bardzo czuły na wszelkie drgania, dlatego istotne jest, aby kalibracja wykonywana była na stabilnym podłożu. Zastosowana przeze mnie metoda kalibracji żyroskopu polega na odczycie i uśrednieniu kilku tysięcy próbek dla każdej z trzech osi. W wyniku tego otrzymamy liczbę prezentującą średnie odchylenie prędkości kątowej w pewnym czasie, jaką wskazuje żyroskop w każdym odczycie. Poniżej znajduje się przykładowy kod automatyzujący kalibrację żyroskopu.

```
void gyroAdjustment(const uint16_t cnt, double t[])
{
  uint16_t gcnt=0; //Licznik odczytów
  double gsum[3] = { 0.0, 0.0, 0.0 }; //Sumy
  double gyro[3];
```

```
  while (gcnt<cnt) {
    //Odczekanie na nowy pomiar z Gyro, minimum 3ms
    delay_ms(3);
    //Pobranie odczytów żyroskopu z rejestrów IMU
    if (IMU_read(I2C_ADR_G, OUT_X_L_G, gyro)) {
      for (uint8_t i=0; i<3; i++) gsum[i] +=
gyro[i]; //Sumowanie
      gcnt++;
    }
  }

  //Obliczenie korekt dla osi X, Y i Z
  for (uint8_t i=0; i<3; i++) t[i] = -gsum[i]/
(double)gcnt;
}
```

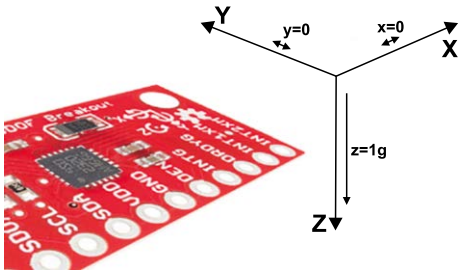
Funkcja oblicza i zwraca w tabeli *t[]* średni błąd pomiarów na podstawie próbek w liczbie *cnt*. Uruchamiana jest automatycznie tuż po włączeniu zasilania IMU, jednak aby czas inicjalizacji układu nie został wydłużony, liczba próbek jest ograniczona do 100. Dokładna kalibracja na podstawie 3000 próbek wykonywana jest po dłuższym przytrzymaniu przycisku. Korekta odbywa się poprzez dodanie wartości odchylenia do kolejnych odczytów RAW (ang. surowy, czyli odczytany bezpośrednio z IMU). Następnie konwertujemy dane RAW do DPS (stopnie na sekundę). Konwersja odbywa się przy uwzględnieniu wartości z rejestru *CTRL_REG4_G*, w którym zapisaliśmy wartość 2000 DPS. 16-bitowe liczby odczytywane z żyroskopu odnoszą się do tej wielkości w następujący sposób: 0 RAW = 0 DPS; 32767 RAW = 2000 DPS; -32767 RAW = -2000 DPS. Schemat funkcji przekształcającej dane RAW na DPS wygląda następująco:

```
double res = 2000.0 / 32767.0;
GX_DPS = (G_RAW_X + G_KOR_X) * res;
GY_DPS = (G_RAW_Y + G_KOR_Y) * res;
GZ_DPS = (G_RAW_Z + G_KOR_Z) * res;
```

Zmienne GX, GY oraz GZ zawierają skalibrowane prędkości kątowe wyrażone w stopniach na sekundę. W przypadku zmiany wartości w rejestrze CTRL_REG4_G konieczna jest także korekta dzielnej w zmiennej res.

Kalibracja akcelerometru

Akcelerometr mierzy przyspieszenia w trzech osiach. Jeśli jest nieruchomy, działa na niego jedynie przyśpieszenie ziemskie o wartości 1g. Odczyty RAW z akcelerometru łatwo zrozumieć, jeśli położymy IMU na równej powierzchni, aby oś Z była skierowana pionowo w dół. W takim układzie przyspieszenie wzdłuż osi X i Y powinno wynosić 0, natomiast na osi Z powinniśmy widzieć wartość reprezentującą 1g. Analogicznie jak w przypadku żyroskopu liczby RAW zamieniamy na przyspieszenie g. Tym razem nasza dzielna wynosi 4 (rejestr CTRL_REG2_XM). Wyślijmy zatem na terminal dane RAW osi Z oraz jej przekształcenie wyrażone w g. Jeżeli oś będzie ukierunkowana pionowo (**rysunek 8**), zobaczymy odchylenia od normy wymagające kalibracji. Polega ona w tym przypadku na wycentrowaniu odczytów, aby rozkład amplitudy wokół zera był równomierny. Przykładowo, nieskalibrowane odczyty RAW na osi Z mogą

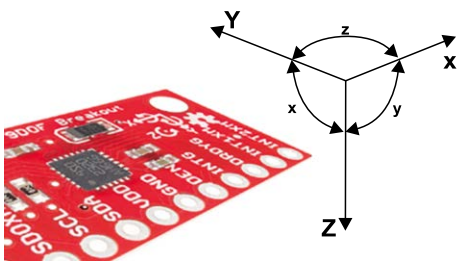


Rysunek 8. Kalibracja akcelerometru – oś Z

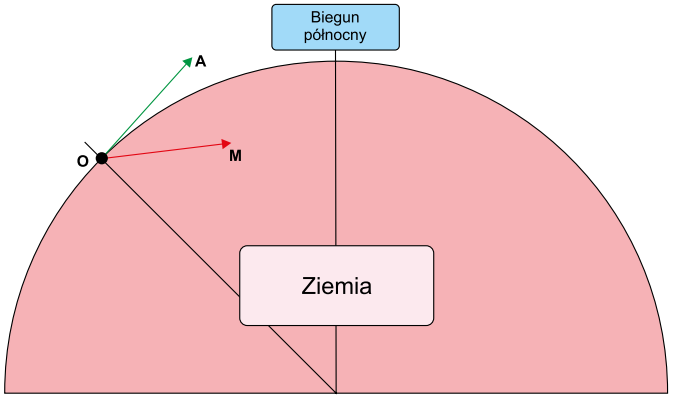
wykazywać wartości w przedziale od -8000 do +8400. Dążymy do sytuacji, w której zakres zmienia się od około -8200 (-1g) do +8200 (+1g) poprzez dodanie korekty wynoszącej -200. Układamy IMU tak, aby odczyty na osiach X i Y oscylowały w okolicy 0 i zapisujemy wartość na osi Z. Następnie odwracamy IMU tak, aby ponownie X i Y były równe 0 i ponownie sprawdzamy wartość na osi Z, która tym razem będzie ze znakiem przeciwnym. Wprowadzamy korektę odczytu. Powtarzamy powyższe czynności, układając IMU w każdej płaszczyźnie i w dwóch kierunkach (XY, XZ i YZ). Wprowadzany korekty do czasu aż będziemy zadowoleni z wyników. Kalibracja nie musi być idealna. W dalszych obliczeniach nie będzie konieczności obliczania wartości g.

Kalibracja magnetometru

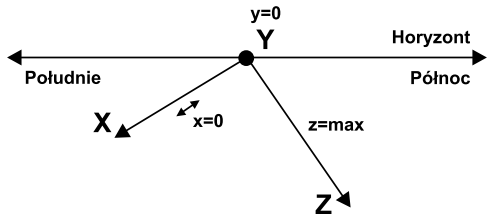
Pozostał nam do skalibrowania elektroniczny kompas. Jak wspomniałem, na dokładność odczytów mogą wpływać inne elementy znajdujące się w pobliżu układu IMU w tym także ścieżki PCB, przez które płynie prąd. Producent zaleca przesunięcie ścieżek prądowych o natężeniu powyżej 10 mA kilka milimetrów od sensorów. Wskazane jest, aby powtórzyć kalibrację w finalnym układzie, a nawet w otoczeniu, w którym będzie używany (np. w docelowym urządzeniu). Przed rozpoczęciem kalibracji należy upewnić się, że pobliskie przedmioty ferromagnetyczne będą znajdowały się w odpowiedniej odległości od IMU bu ograniczyć ich wpływ na odczyty. Kalibracja magnetometru jest dwuetapowa. Pierwszy etap jest podobny do kalibracji akcelerometru i polega na wycentrowaniu pomiarów na każdej z 3 osi. By tego dokonać, musimy umieć interpretować dane RAW. Na **rysunku 9** w uproszeniu pokazane zostały dwa wektory sił mierzonych przez IMU. Obserwator znajduje się w miejscu oznaczonym literą **O**. Wektor **A** reprezentuje ruch poziomy po powierzchni Ziemi. Magnetometr mierzy siły wzdłuż wektora **M**. Kierunek tego wektora jest równoległy do sił pola magnetycznego, zwrot zaś wskazuje biegun północny. Jak widać wektor **M** nie jest równoległy do **A** lecz jest pochylony pod pewnym kątem, który jest bliski 0º na równiku i równy 90º na biegunie. Do wykonania kalibracji należy znaleźć ten wektor przy pomocy magnetometru. Znajomość kierunku północnego w miejscu wykonywania kalibracji będzie ułatwieniem. Zaczynamy od kalibracji osi Z. Uruchamiamy terminal i rozpoczynamy wysyłanie danych RAW. Ustawiamy IMU



Rysunek 7. Pomiary żyroskopu



Rysunek 9. Wektory sił mierzone przez IMU



Rysunek 10. Kalibracja magnetometru – oś Z

w taki sposób, by oś Z skierowana była równolegle do osi północ-południe. Następnie pochylamy IMU, aby oś Z przecinała podłogę pod pewnym kątem. Odczyt ekstremalny uzyskamy w chwili, gdy wartości na osiach X i Y będą oscylowały w okolicy zera (**rysunek 10**). W tej pozycji obrót IMU wokół osi Z nie ma wpływu na odczyty. Odwracamy oś Z o 180º i w ten sposób otrzymujemy pomiar ze znakiem przeciwnym. Procedura wygląda analogicznie jak w przypadku akcelerometru. Po wyznaczeniu i uwzględnieniu korekt trzech osi zauważymy, że przedziały wyników na każdej z nich są różne. Przykładowo: skorygowany pomiar X zawiera się w przedziale od -3035 do 3035, Y od -2685 do 2678, Z od -2882 do 2883. Oś X została wycentrowana z najlepszym rezultatem. Osie Y i Z nadal mają pewne niewielkie odchylenia, które można pominąć. Kolejnym etapem kalibracji jest normalizacja. Z uwagi na fakt, iż moduły przedziałów 3035, 2685 oraz 2883 odzwierciedlają to samo natężenie pola magnetycznego, należy je przeskalować do wspólnej miary poprzez mnożenie przez odpowiedni współczynnik. Do jego obliczenia wybieramy jedną z wartości, która będzie punktem odniesienia (np. X 3035) i dzielimy ją przez odczyty z IMU:

```
wspMX = 3035 / 3035 = 1
wspMY = 3035 / 2685 = 1.133308439
wspMZ = 3035 / 2883 = 1.052722858
```

Finalnie kalibracja naszego magnetometru wyglądać będzie następująco:

```
kalMX = rawMX + korektaMX; // *1
kalMY = (rawMY + korektaMY) * wspMY;
kalMZ = (rawMZ + korektaMZ) * wspMZ;
```

gdzie:
*kal** – skalibrowany odczyt,
*raw** – odczyt bezpośrednio z IMU,
*korekta** – wartości centrujące odczyty,
*wsp** – wartości współczynników normalizacyjnych,

Przedstawione przeze mnie metody kalibracji zapewniają bardzo zadowalające efekty. Wymagane jest jedynie zachowanie należytej staranności przy ich wykonywaniu. W Internecie dostępne inne opracowania na temat kalibracji IMU. Zachęcam do ich poznania.

Obliczanie kątów obrotu IMU na podstawie pomiarów akcelerometru i magnetometru

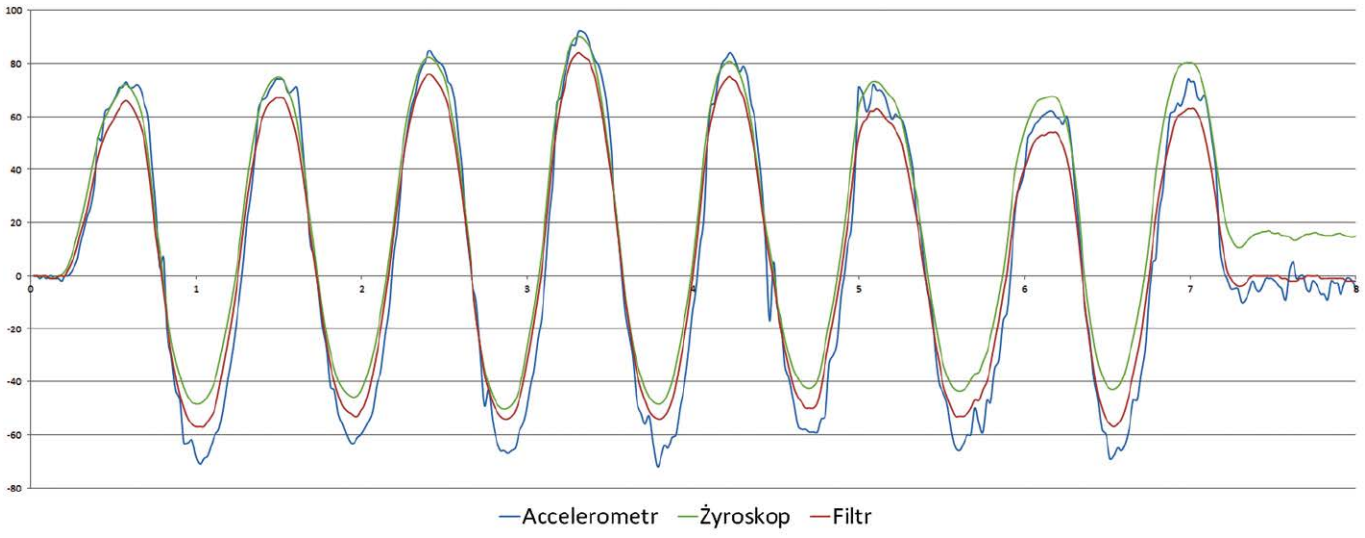
Na **rysunku 11** zamieszczono o wykres zmienności kąta wychylenia wokół osi X w czasie. Wykres powstał przez naprzemienny obrót IMU do góry i do dołu. Kolorem zielonym przedstawiono wskazania żyroskopu. Wyraźnie widać, że już po siedmiu sekundach wykres oddalił się od położenia centralnego prawie o 20 stopni. Jest to sytuacja naturalna i nazywana dryfem żyroskopu. Dla porównania linia niebieska przedstawia kąty mierzone na podstawie wskazań akcelerometru. Zauważymy, że akcelerometr nie wykazuje efektu dryfu, jednak w porównaniu do żyroskopu jego wskazania są obarczone dużo większym szumem. Jeśli użylibyśmy odczytów akcelerometru do sterowania Headtrackerem, nasze serwomechanizmy uległyby w krótkim czasie awarii z powodu częstych i szybkich zmian kierunku. Jeśli natomiast użylibyśmy jedynie żyroskopu, nasz Headtracker również nie nadawałby się użytku. By uzyskać zadowalający efekt musimy „połączyć" oba odczyty, by uzyskać przebieg zaprezentowany kolorem czerwonym.

IMU możemy oczywiście obracać w dowolnych kierunkach, dlatego musimy ustalić umowną kolejność obrotów. Zakładamy więc, że obrót wykonywany jest w pierwszej kolejności względem osi X, następnie Y i na koniec Z. Ma to swoje uzasadnienie, ponieważ na podstawie wskazań akcelerometrów nie jest możliwe wyliczenie kąta względem osi pionowej Z. Do tego celu zostanie wykorzystany magnetometr. Obliczanie kątów rozpoczynamy od zapisania podstawowych macierzy obrotów, wykorzystywanych między innymi do przekształceń w przestrzeni 3D (np. w grafice komputerowej):

$$X(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix}, Y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix},$$

$$Z(\gamma) = \begin{pmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Trójwymiarową macierz obrotu uzyskujemy po przemnożeniu powyższych macierzy ustalonej kolejności: $O_{XYZ}=X(\alpha)\cdot Y(\beta)\cdot Z(\gamma)$. *Operacje mnożenia macierzy, jak i kolejne etapy obliczeń wykonywane były za pomocą programu Microsoft Mathematics*. Jeśli na IMU nie działa przyspieszenie liniowe, wówczas jedyną działającą siłą jest przyspieszenie ziemskie skierowane pionowo w dół wzdłuż osi Z, zatem jest możliwe ustalenie kątów obrotu X i Y względem wektora Z, który zawsze jest skierowany w tym samym kierunku. W ten sposób otrzymamy macierz obrotu akcelerometru, wyrażoną w postaci:



Rysunek 11. Dryf żyroskopowy i filtrowanie komplementarne

$$A = \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} = X(\alpha) \cdot Y(\beta) \cdot Z(\gamma) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} -\sin(\beta) \\ \sin(\alpha) \cdot \cos(\beta) \\ \cos(\alpha) \cdot \cos(\beta) \end{pmatrix} \Rightarrow \begin{cases} A_x = -\sin(\beta) \\ A_y = \sin(\alpha) \cdot \cos(\beta) \\ A_z = \cos(\alpha) \cdot \cos(\beta) \end{cases}$$

W otrzymanym układzie równań należy zauważyć brak zależności pomiędzy przyspieszeniami a kątem obrotu wokół osi Zγ, co jest faktem. Natomiast możliwe jest wyznaczenie kąta α:

$$\frac{A_y}{A_z} = \frac{\sin(\alpha) \cdot \cos(\beta)}{\cos(\alpha) \cdot \cos(\beta)} \Rightarrow \tan(\alpha) = \frac{A_y}{A_z}$$

Rozwiązanie dla kąta β jest podane wprost jako:

$$\sin(\beta) = -A_x$$

Obliczenie kąta obrotu wokół osi Y jest zależne jedynie od przyspieszenia mierzonego na osi X. Do wykonania powyższych obliczeń konieczne jest dokonanie normalizacji odczytów z akcelerometru:

$$A_{xnorm} = \frac{A_x}{|A|}, A_{ynorm} = \frac{A_y}{|A|}, A_{znorm} = \frac{A_z}{|A|}, |A| =$$

$$= \sqrt{A_x^2 + A_y^2 + A_z^2}$$

Możemy teraz przejść do wyznaczenia kierunku z magnetometru. Zgodnie z projektem PCB i **rysunkiem 12** zakładamy, że oś Y będzie wskazywać północ, a odchylenie od horyzontu odbywa się wokół osi X o kąt ϵ (kąt inklinacji). Wówczas składowe wektora horyzontalnego można zapisać w postaci:

$$\begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \begin{pmatrix} 0 \\ M_0 \cdot \cos(\varepsilon) \\ M_0 \cdot \sin(\varepsilon) \end{pmatrix}$$

gdzie M_0 jest pewną mierzoną wielkością pola magnetycznego, natomiast,,to odczyty z magnetometru. Przy pochyleniu IMU o 0° wektor wskazujący horyzont jest równoległy do osi Y, natomiast przy pochyleniu o 90° do osi Z. Kierunek – czyli obrót wokół osi Z – należy wyznaczyć z uwzględnieniem powyższego kąta inklinacji. A zatem:

$$Z(\gamma)_{tilt} = Z(\gamma) \cdot \begin{pmatrix} 0 \\ M_0 \cdot \cos(\varepsilon) \\ M_0 \cdot \sin(\varepsilon) \end{pmatrix} = \begin{pmatrix} M_0 \cdot \sin(\gamma) \cdot \cos(\varepsilon) \\ M_0 \cdot \cos(\gamma) \cdot \cos(\varepsilon) \\ M_0 \cdot \sin(\varepsilon) \end{pmatrix}$$

Powyższe przekształcenie jest niczym rzutowanie kąta inklinacji na płaszczyznę horyzontalną. Mając wyliczone kąty na podstawie wskazań akcelerometru (α i β) możemy wykonać obrót wektora pola magnetycznego wokół osi X i Y tym samym „rzutując" je na tę samą płaszczyznę poziomą:

$$X(\alpha) \cdot Y(\beta) \cdot \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} =$$

$$= \begin{pmatrix} M_x \cdot \cos(\beta) - M_z \cdot \sin(\beta) \\ M_x \cdot \sin(\alpha) \cdot \sin(\beta) + M_y \cdot \cos(\alpha) + M_z \cdot \sin(\alpha) \cdot \cos(\beta) \\ M_x \cdot \sin(\beta) \cdot \cos(\alpha) - M_y \cdot \sin(\alpha) + M_z \cdot \cos(\alpha) \cdot \sin(\beta) \end{pmatrix}$$

W efekcie uzyskujemy układ równań, z którego wyliczymy kąt γ:

$$\begin{pmatrix} M_0 \cdot \sin(\gamma) \cdot \cos(\varepsilon) \\ (M_0 \cdot \cos(\gamma) \cdot \cos(\varepsilon)) = \\ M_0 \cdot \sin(\varepsilon) \end{pmatrix}$$

$$= \begin{pmatrix} M_x \cdot \cos(\beta) - M_z \cdot \sin(\beta) \\ M_x \cdot \sin(\alpha) \cdot \sin(\beta) + M_y \cdot \cos(\alpha) + M_z \cdot \sin(\alpha) \cdot \cos(\beta) \\ M_x \cdot \sin(\beta) \cdot \cos(\alpha) - M_y \cdot \sin(\alpha) + M_z \cdot \cos(\alpha) \cdot \sin(\beta) \end{pmatrix}$$

$$M_0 \cdot \sin(\gamma) \cdot \cos(\varepsilon) = M_x \cdot \cos(\beta) - M_z \cdot \sin(\beta)$$

$$M_0 \cdot \cos(\gamma) \cdot \cos(\varepsilon) =$$

$$= M_x \cdot \sin(\alpha) \cdot \sin(\beta) + M_y \cdot \cos(\alpha) + M_z \cdot \sin(\alpha) \cdot \cos(\beta)$$

$$\tan(\gamma) = \frac{M_x \cdot \cos(\beta) - M_z \cdot \sin(\beta)}{M_x \cdot \sin(\alpha) \cdot \sin(\beta) + M_y \cdot \cos(\alpha) + M_z \cdot \sin(\alpha) \cdot \cos(\beta)}$$

Kąty α,β i γ obliczymy za pomocą funkcji języka C:

```
accAngleX = atan2(aft[1], aft[2]);
accAngleY = asin(-aft[0]);
magAngleZ = atan2(Mx*cos(p)-Mz*sin(p),
Mx*sin(r)*sin(p)+My*
cos(r)+Mz*sin(r)*cos(p));
```

gdzie:

magAngleZ – kąt względem kierunku północnego obliczony z uwzględnieniem pochylenia IMU,

*mag** – skalibrowany odczyt z magnetometru,

r – kąt accAngleX,

p – kąt accAngleY.

Funkcja *atan2* zwraca wartości kątów w przedziale od -π do π. Obliczony kąt odchylenia od kierunku północnego zmienia się w przedziałach -π do 0 (obrót w jedną stronę) i od 0 do π (obrót w stronę przeciwną). Chcemy jednak, aby kąt obliczany był nie względem północy a względem początkowego położenia headtrackera. W tym celu dokonujemy przekształcenia względem pozycji początkowej *start_angle*:

```
double calcAngleAbs(double angle, double start_
angle) {
  if (fabs(angle-start_angle) > M_PI) {
    if (start_angle>0) return
(angle+M_2PI-start_angle);
    else return (angle-M_2PI-start_angle);
  }
  else return (angle-start_angle);
}
```

gdzie:

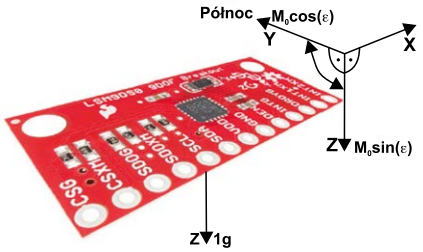
angle – aktualny kąt,

start_angle – zapamiętany kąt początkowy.

To samo przekształcenie wykorzystujemy zarówno przy obliczeniu kąta względem osi Z (magAngleZ), jak i X (accAngleX).

W następnej kolejności obliczamy czas pomiędzy odczytami z IMU. Zgodnie z tym, o czym była mowa przy okazji inicjalizacji CPU, wykorzystujemy przerwanie timera sprzętowego uruchamiane co 1 milisekundę:

```
ISR(TIMER2_COMP_vect) {
  //Zliczanie milisekund
  millis++;
  //Timery programowe
  //Timer1: asynchroniczne miganie diodą, reset
  if (timer1>0) timer1--;
  //Timer2: obsługa przycisku, kalibracja żyroskopu
  if (timer2>0) timer2--;
}
```



Rysunek 12. Składowe wektora horyzontalnego przy pochyleniu IMU

Obliczyliśmy kąty w DPS odczytane z żyroskopu oraz kąty względne mierzone na podstawie wskazań akcelerometru i magnetometru. Należy zwrócić uwagę na znaki, z jakimi wykonywane są obliczenia. Znaki są sprawą umowną, jednak muszą być spójne. Zatem jeśli obracamy IMU w lewo względem osi Z i żyroskop odczytuje ten ruch ze znakiem ujemnym, wówczas kąt obliczany z magnetometru również musi być ze znakiem ujemnym itd. W przypadku stwierdzenia niezgodności znaków należy dokonać korekt poprzez odpowiednią zmianę znaków wyliczonych kątów.

W celu zniwelowania gryfu żyroskopowego i uzyskania efektu jak na rys. 11 wykorzystamy filtr komplementarny ze zmiennym współczynnikiem, który wyrażony jest następującym wzorem:

$$angle = W \cdot (angle + gyro \cdot dt) + (1 - W) \cdot accAngle$$

gdzie:
angle – szukany kąt obrotu wokół danej osi,
W – współczynnik przyjmujący wartości od 0 do 1,
gyro – aktualny odczyt z żyroskopu w DPS,
dt – czas w sekundach mierzony od ostatniego odczytu z IMU,
*gyro*dt* – kąt, o jaki dokonano obrotu od poprzedniego pomiaru,
accAngle – kąt wyliczony ze wskazań akcelerometru lub magnetometru w stopniach.

Do poprzedniej wartości kąta dodawany jest DPS (analogicznie jak w przypadku wykresu z rys. 11) oraz dokonuje się „scalenie" tej wartości z kątem accAngle. Powyższa funkcja filtrująca będzie zastosowana do kompensacji dryfu żyroskopowego w obu płaszczyznach:

```
gyroAngleX = filtrKomplementarny(gyroAngleX, gyroX,
accAngleX*R2D, dt, calcWspC(gyroX));
gyroAngleZ = filtrKomplementarny(gyroAngleZ, gyroZ,
magAngleZ*R2D, dt, calcWspC(gyroZ));
```

gdzie:
gyroAngleX – finalny kąt obrotu względem osi X (pochylenie w górę i w dół),
gyroAngleZ – finalny kąt obrotu względem osi Z (obrót w lewo i prawo),
R2D – mnożnik konwertujący jednostki miary kątów z radianów na stopnie (R2D=180/p),
calcWspC – funkcja obliczająca współczynnik *W*, zależna od prędkości obrotu DPS.

Wadą powyższego rozwiązania jest to, iż na obliczenia kątów duży wpływ wywierają ruchy liniowe IMU, czyli mierzone przyspieszenia. Przyczyniają się one do złej stabilizacji kamery przy niewielkich prędkościach obrotowych. W celu polepszenia ergonomii pracy konieczne jest zastosowanie dodatkowego algorytmu filtrującego. W tym celu utworzona została funkcja *calcWspC*, której argumentem jest aktualna prędkość obrotu DPS. Jej zadaniem jest obliczenie współczynnika filtru komplementarnego. Przy „niskich" prędkościach DPS, współczynnik jest bliski 1, a jego wartość stopniowo maleje wraz ze wzrostem DPS, jak pokazano na **rysunku 13**. Wartość 1 oznacza, że obliczenia wykonywane są w 100% na podstawie pomiarów żyroskopu, co wzmaga efekt dryfu. Wartość 0 oznacza wprowadzenie do obliczeń 100% szumów z pomiarów akcelerometru. Zadowalające efekty uzyskuje się przy współczynniku wyższym od 0,95. Dodatkowo przy „bardzo niskich" prędkościach obrotowych wskazane jest, aby kamera pozostawała całkowicie nieruchoma. Pozostaje określić metodą doświadczalną, co rozumiemy przez wartości „niskie" i „bardzo niskie". Załóżmy, że prędkości obrotowe niższe od dps_{min} nie będą powodowały zmian kątów, w przedziale od dps_{min} do dps_{max} nastąpi stopniowe zmniejszanie współczynnika *W* zgodnie z funkcją wykładniczą o wykładniku *p* aż do wartości W_{min}. Współczynnik obliczymy na podstawie następującej funkcji:

$$F = \begin{cases} 1 & dla |dps| \le dps_{min} \\ 1 - M \cdot (|dps| - dps_{min})^p & dla dps_{min} < |dps| < dps_{max} \\ W_{min} & dla |dps| \ge dps_{max} \end{cases}$$

gdzie mnożnik *M* jest wartością stałą i wynosi:

$$M = (dps_{max} - dps_{min})^{-p}$$

Współczynnik obliczymy ze wzoru:

$$W = W_{min} + (1 - W_{min}) \cdot F$$

Wartości parametrów mogą zostać dobrane następująco: dps_{min}=3, dps_{max}=60, p=2, W_{min}=0,95, co zostało pokazane na rys. 13. Całkowite ograniczenie obrotu kamery przy prędkościach niższych od dps_{min} uzyskamy poprzez wyzerowanie odczytów z żyroskopu.

```
if (fabs(gyro)<=dps_min) gyro=0;
```

Powyższe rozwiązanie niweluje drgania przy powolnych ruchach headtrackera prawie do zera i poprawia tym samym stabilność kamery. Przy odpowiednim dobraniu parametrów krzywej możliwe jest uzyskanie efektu precyzyjnego sterowania kamerą. Jednakże przy szybszych obrotach nasz układ nadal jest podatny na szumy liniowe, które także należy zminimalizować. Do tego celu wykorzystamy filtrowanie, którego współpomysłodawcą był amerykański inżynier węgierskiego pochodzenia Rudolf Kalman. Szczegółowe omówienie teorii filtru Kalmana w krótkim artykule nie jest możliwe, jednak polecam zapoznanie się z nią w najprostszym wydaniu. Na pewno poniższy algorytm przyda się w przyszłości.

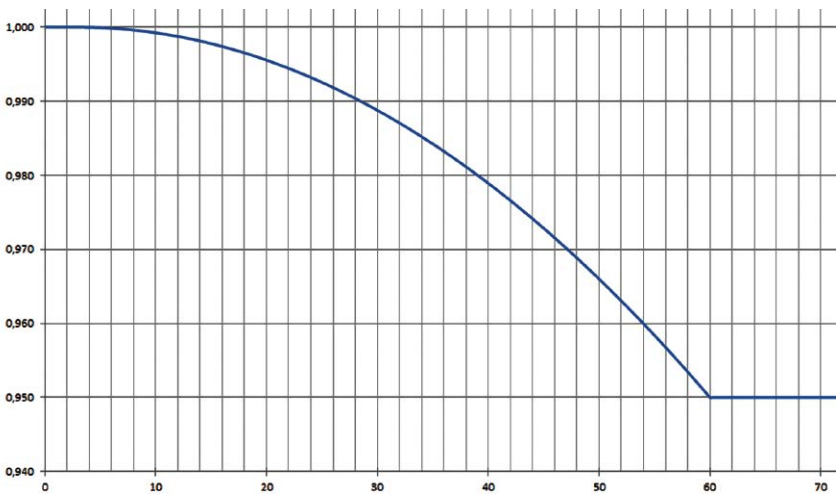
Podsumowanie

W kolejnej części artykułu omówimy dalsze szczegóły implementacji (w tym filtr Kalmana i generowanie ramki PPM) oraz zajmiemy się praktyczną realizacją układową.

Arkadiusz Witczak

Materiały źródłowe:

1. *https://goo.gl/uydmnr*
2. *https://goo.gl/qBOh2N*
3. *http://goo.gl/yyV91n*
4. *http://goo.gl/8leJ8w*
5. *http://goo.gl/ZTUZkk*
6. *http://goo.gl/S7fNv9*
7. *http://goo.gl/QH7alV*
8. *http://goo.gl/BZJ3uD*
9. *http://goo.gl/oEw1og*
10. *http://goo.gl/C0G93L*
11. *http://goo.gl/n3m7R4*
12. *http://goo.gl/VQroFr*



Rysunek 13. Współczynnik filtru komplementanego w zależności od DPS