

WYDZIAŁ ELEKTRONIKI I TELEKOMUNIKACJI
POLITECHNIKA POZNAŃSKA

MAGISTERSKA PRACA DYPLOMOWA

**UKŁADY STERUJĄCE I OPROGRAMOWANIE DLA
MODELU CZTEROWIRNIKOWEGO ŚMIGŁOWCA**

inż. Dariusz Fertyk

Promotor
dr inż. Krzysztof Arnold

Poznań 2015

Spis treści

1 Wstęp	7
2 Charakterystyka kwadrokopterów	9
2.1 Zasada działania	9
2.2 Podstawowe podzespoły	14
2.2.1 Kontroler lotu	15
2.2.2 Moduł czujników	15
2.2.3 Moduł komunikacyjny	15
2.2.4 Sterowniki silników	16
2.2.5 Silniki	17
2.3 Konstrukcje nośne	17
2.4 Kierunki rozwoju	19
3 Podstawowe algorytmy stabilizacji lotu kwadrokopterów	21
3.1 Podstawy matematyczne	21
3.2 Algorytm kontroli prędkości obrotowych	23
3.3 Algorytm kontroli położenia kątowego	25
4 Czujniki stosowane w kwadrokopterach	31
4.1 Rodzaje i kryteria doboru czujników	31
4.2 Technologia MEMS	32
4.3 Akcelerometr	34
4.4 Żydroskop	37
4.5 Inercyjny zespół pomiarowy - IMU	41
5 Projekt kwadrokoptera	43
5.1 Wymagania techniczne	43
5.2 Koncepcja rozwiązania	44
5.2.1 Kontroler lotu	46
5.2.2 Moduł komunikacji radiowej	47
5.2.3 Moduł czujników	48
5.2.4 Kontrolery silników	49
5.3 Opis układowy	50
5.3.1 Kontroler lotu	50
5.3.2 Sterownik silnika	53
5.4 Konstrukcja mechaniczna	56

6 Oprogramowanie kwadrokoptera	59
6.1 Struktura oprogramowania	59
6.2 Kontroler lotu	60
6.2.1 Funkcje programowe	60
6.2.2 Komunikacja radiowa	60
6.2.3 Obsługa czujników	67
6.2.4 Algorytm stabilizacji lotu	70
6.2.5 Generowanie sygnałów sterujących	74
6.3 Sterownik silnika	76
6.3.1 Funkcje programowe	76
6.3.2 Odbiór sygnałów sterujących	77
6.3.3 Uśrednianie wartości sygnałów sterujących	79
6.3.4 Generacja sygnałów PWM	80
6.4 Interface użytkownika do strojenia parametrów lotu	83
6.5 Interfejs użytkownika do kontroli lotu	87
7 Uruchomienie	91
7.1 Zakres testów	91
7.2 Testy kontrolera silnika	91
7.2.1 Generator sygnałów testowych	91
7.2.2 Testy pojedynczego kontrolera silnika	94
7.2.3 Testy zespołu czterech kontrolerów silników	95
7.3 Testy kontrolera lotu kwadrokoptera	97
7.4 Testy aplikacji użytkownika	98
7.5 Strojenie regulatorów PID kwadrokoptera	98
8 Wnioski	103
Bibliografia	105

Rozdział 1

Wstęp

Rozwój techniki cyfrowej w trakcie ostatniej dekady doprowadził do powstania mikrokontrolerów, będących bardzo silną gałęzią rynku układów cyfrowych. Do ich niezaprzeczalnych zalet należą takie cechy jak:

- Niski pobór mocy.
- Obecność podstawowych sprzętowych interfejsów komunikacyjnych, do których zaliczają się:
 - UART,
 - I²C,
 - SPI.
- Obecność sprzętowych modułów liczników, zdolnych do generowania wielu sygnałów PWM jednocześnie.
- Obecność przetworników analogowo-cyfrowych oraz coraz częściej cyfrowo-analogowych.
- Dostępność układów w obudowach o niewielkich wymiarach.
- Niska cena jednostkowa.

Zalety te doprowadziły do konsekwentnego wzrostu popularności mikrokontrolerów w ciągu ostatnich lat, oraz do stosowania ich w rozmaitych aplikacjach, do których zaliczają się między innymi aplikacje mobilne takie jak autonomiczne i zdalnie sterowane roboty. Robotami, do konstrukcji których konstruktorzy bardzo chętnie stosują mikrokontrolery, jest przede wszystkim ostatnio rodzina czterowirnikowych śmigłowców, zwanych również kwadrokopterami.

Cechy takie jak prostota oraz wytrzymałość konstrukcji mechanicznej, możliwość pionowego startu i lądowania, a także umiejętność wykonywania rozmaitych manewrów powietrznych sprawiły, że od

chwili pojawienia się kwadrokopterów na rynku grono ich użytkowników stale się poszerza. Należą do niego już nie tylko piloci startujący w zawodach zdalnie sterowanych modeli latających, lecz coraz częściej firmy używające kwadrokopterów do realizacji takich zadań jak:

- Nagrywanie filmów kręconych z powietrza
- Patrolowanie niewielkich terenów
- Transport niewielkich ładunków

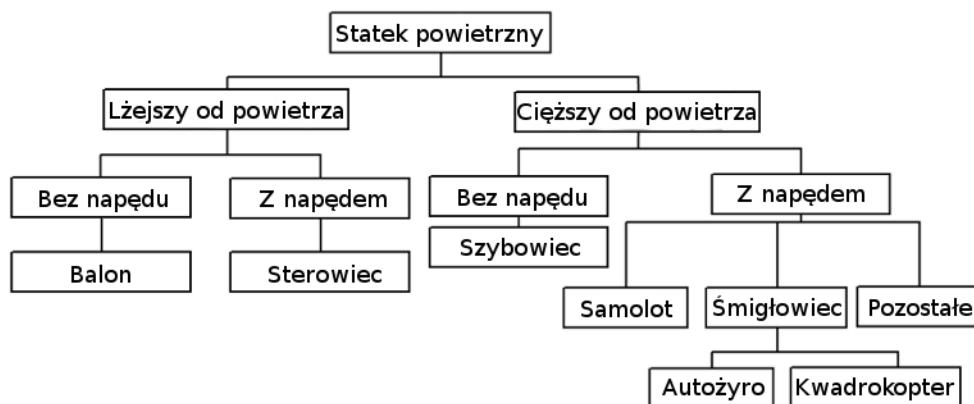
Widać zatem, że kwadrokoptery mają duży potencjał komercyjny, a co za tym idzie ich rozwijający się rynek będzie stale potrzebował nowych rozwiązań mechanicznych, elektronicznych oraz programistycznych. Stało się to, w połączeniu z faktem, iż kwadrokoptery same w sobie stanowią interesujący projekt z dziedziny elektroniki oraz programowania, główną motywacją do opracowania autorskiej konstrukcji czterowirnikowego śmigłowca.

Rozdział 2

Charakterystyka kwadrokopterów

2.1 Zasada działania

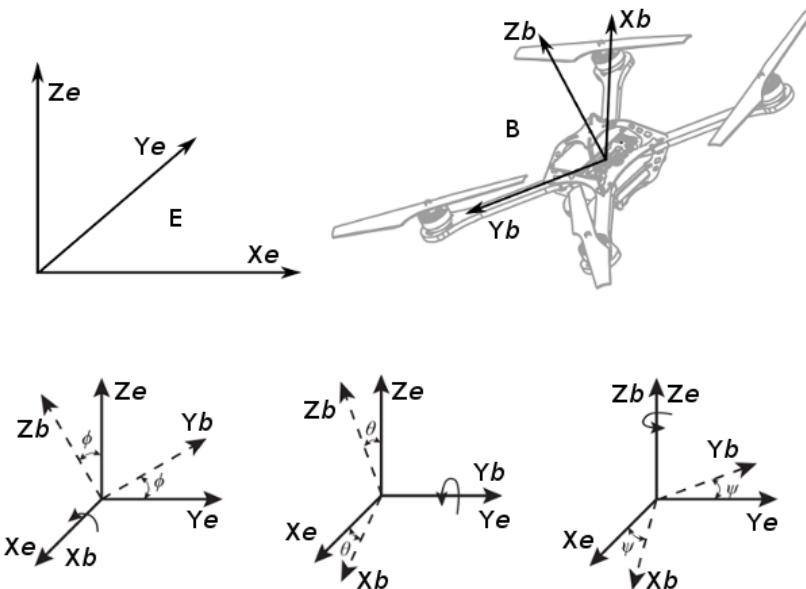
Kwadrokopter zaliczany jest do rodziny śmigłowców, czyli statków powietrznych wytwarzających siłę nośną dzięki ruchowi obrotowemu wirników (rys. 2.1). Kwadrokoptery różnią się nieco od konstrukcji klasycznego helikoptera - posiadają cztery wirniki umieszczone poziomo, które wirują parami w przeciwnie strony (dwa wirniki zgodnie z ruchem wskazówek zegara, dwa przeciwnie do ruchu wskazówek zegara). Dzięki obrotom śmigiel w przeciwnych kierunkach momenty bezwładności, próbujące obrócić kwadrokopter wokół własnej osi znoszą się, a co za tym idzie w tego typu konstrukcjach nie trzeba stosować wirnika ogonowego. Pociąga to za sobą w konsekwencji bardzo prostą konstrukcję mechaniczną oraz lepszą w porównaniu do helikopterów manewrowość [1].



RYS. 2.1: Klasyfikacje statków powietrznych [2, 3]

Do opisu ruchu i położenia kwadrokoptera wprowadza się dwa układy odniesienia: inercjalny układ odniesienia (np. układ współrzędnych pilota kwadrokoptera, zwany często układem współrzędnych Ziemi) oraz układ odniesienia kwadrokoptera [4, 5]. Przy tak zdefinowanych układach odniesienia możemy określić trzy kąty (rys. 2.2):

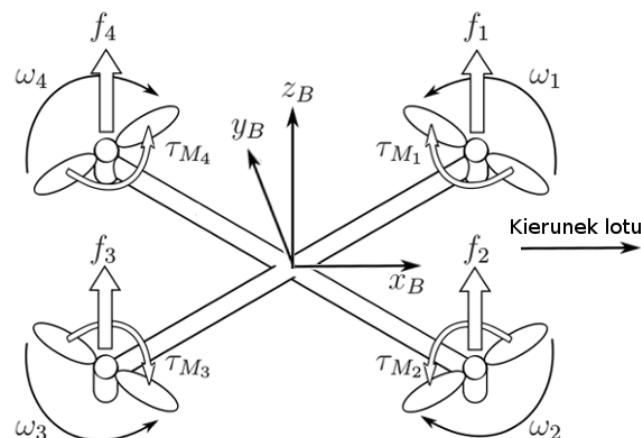
- Kąt przechylenia (ang. Roll) ϕ : jest to kąt obrotu wokół osi $X_b \parallel X_e$
- Kąt pochylenia (ang. Pitch) θ : jest to kąt obrotu wokół osi $Y_b \parallel Y_e$
- Kąt odchylenia (ang. Yaw) ψ : jest to kąt obrotu wokół osi $Z_b \parallel Z_e$



RYS. 2.2: Układ odniesienia zewnętrznego obserwatora E (inercjalny układ odniesienia) oraz układ odniesienia kwadrokoptera B, wraz z zaznaczonymi kątami przechylenia (ϕ), pochylenia (θ) oraz odchylenia (ψ) (konfiguracja „+”) [6]

Kąty (ϕ, θ, ψ) zwane są kątami Euler i służą do opisu przejścia między układem odniesienia Ziemi a układem odniesienia kwadrokoptera.

Chcąc zrozumieć w jaki sposób można osiągnąć ruch liniowy oraz obrotowy kwadrokoptera, należy spojrzeć na siły i momenty sił, które na niego działają (rys. 2.3).



RYS. 2.3: Siły i momenty sił działające na kwadrokopter (konfiguracja „X”) [7]

Na kwadrokopter (pomijając siłę grawitacji) działającą cztery siły ciągów silników f_1, f_2, f_3, f_4 , oraz cztery momenty sił $\tau_{M_1}, \tau_{M_2}, \tau_{M_3}, \tau_{M_4}$, gdzie τ_{M_i} jest momentem reakcji silnika, spowodowanym oporem aerodynamicznym śmigieł [8, 9]. Siła unosząca kwadrokopter w powietrzu jest sumą sił ciągu wszystkich silników. Dla przykładu konfiguracji kwadrokoptera, przedstawionej na rysunku 2.3 moment siły, obracający kwadrokopter wokół osi X_b (powodujący przechylenie) jest wynikiem różnicy $(f_1 + f_4) - (f_2 + f_3)$, natomiast moment siły obracający kwadrokopter wokół osi Y_b (powodujący pochylenie) jest wynikiem różnicy $(f_3 + f_4) - (f_1 + f_2)$. Moment siły obracający kwadrokopter wokół osi Z_b (powodujący odchylenie) jest wynikiem sumy $\tau_{M_1} + \tau_{M_2} + \tau_{M_3} + \tau_{M_4}$. Zmieniając wartości tych sił i momentów można kontrolować ruch liniowy kwadrokoptera we wszystkich kierunkach jak również ruch obrotowy wokół każdej z jego osi. Warto zauważyć, że siły ciągu silników oraz momenty sił działające na quadrocopter wynikają (przy założeniu, że skok łopat śmigieł się nie zmienia) bezpośrednio z prędkości kątowej wirników $\omega_1, \omega_2, \omega_3, \omega_4$. Dzięki temu zmieniając jedynie prędkości obrotowe wirników, co jest bardzo proste do osiągnięcia za pomocą elektronicznych sterowników, zyskujemy pełną kontrolę nad wszystkimi sześcioma stopniami swobody kwadrokoptera [8].

Mając świadomość sił działających na kwadrokopter możemy napisać warunki równowagi oraz ruchu dla przykładu z rysunku 2.3 [8, 9]:

- **Warunek zawisu w powietrzu**

Równowaga sił: $f_1 = f_2 = f_3 = f_4$ oraz $\sum_{i=1}^4 f_i = mg$

Zgodność kierunków: $f_{1,2,3,4} \parallel mg$

Równowaga momentów: $(\tau_{M_1} + \tau_{M_3}) + (\tau_{M_4} + \tau_{M_2}) = 0$

- **Warunek ruchu w pionie**

Brak równowagi sił: $f_1 = f_2 = f_3 = f_4$ ale $\sum_{i=1}^4 f_i \neq mg$

Zgodność kierunków: $f_{1,2,3,4} \parallel mg$

Równowaga momentów: $(\tau_{M_1} + \tau_{M_3}) + (\tau_{M_4} + \tau_{M_2}) = 0$

- **Warunek obrotu wokół osi X_b**

Brak równowagi sił: $f_1 + f_4 \neq f_2 + f_3$

Brak zgodność kierunków: $f_{1,2,3,4} \not\parallel mg$

Równowaga momentów: $(\tau_{M_1} + \tau_{M_3}) + (\tau_{M_4} + \tau_{M_2}) = 0$

- **Warunek obrotu wokół osi Y_b**

Brak równowagi sił: $f_3 + f_4 \neq f_1 + f_2$

Brak zgodność kierunków: $f_{1,2,3,4} \not\parallel mg$

Równowaga momentów: $(\tau_{M_1} + \tau_{M_3}) + (\tau_{M_4} + \tau_{M_2}) = 0$

- **Warunek obrotu wokół osi Z_b**

Równowaga sił: $\sum_{i=1}^4 f_i = mg$

Zgodność kierunków: $f_{1,2,3,4} \parallel mg$

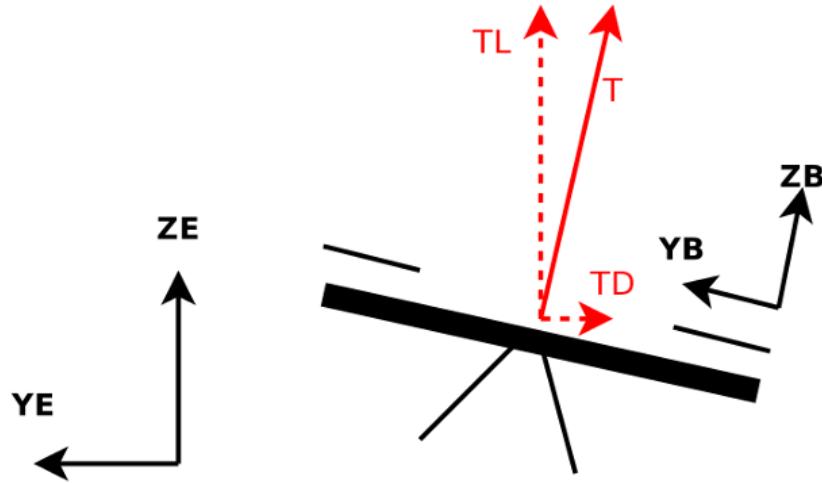
Brak równowagi momentów: $(\tau_{M_1} + \tau_{M_3}) + (\tau_{M_4} + \tau_{M_2}) \neq 0$

Warunek swobodnego zawisu w powietrzu jest najprostszy do rozpatrzenia: kwadrokopter będzie wisiał nieruchomo, gdy siły ciągu wszystkich czterech wirników będą działać pionowo w góre równoważąc jednocześnie siłę ciężkości, z jaką Ziemia przyciąga kwadrokopter. Dzięki temu że, sumy par prędkości obrotowych śmigieł kręcących się w przeciwnie strony są sobie równe, momenty sił, próbujące obrócić kwadrokopter wokół własnej osi znoszą się.

Warunek ruchu w pionie różni się od warunku swobodnego zawisu w powietrzu jedynie tym, że siły ciągu silników nadal są równoległe do siły ciężkości, ale jej nie równoważą. Uzyskuje się to przez równomierne zwiększenie lub zmniejszenie prędkości obrotowych wszystkich wirników. Możemy zatem rozróżnić warunek ruchu w góre: $\sum_{i=1}^4 f_i > mg$, oraz warunek ruchu w dół: $\sum_{i=1}^4 f_i < mg$.

Warunek obrotu wokół własnej osi uzyskuje się przez zwiększenie prędkości kątowych pary śmigieł (np. śmigieł obracających się zgodnie z ruchem wskazówek zegara) i jednoczesne zmniejszenie prędkości pary śmigieł obracających się w przeciwnym kierunku. Dzięki temu momenty sił przestają się równoważyć i kwadrokopter zaczyna obracać się wokół własnej osi.

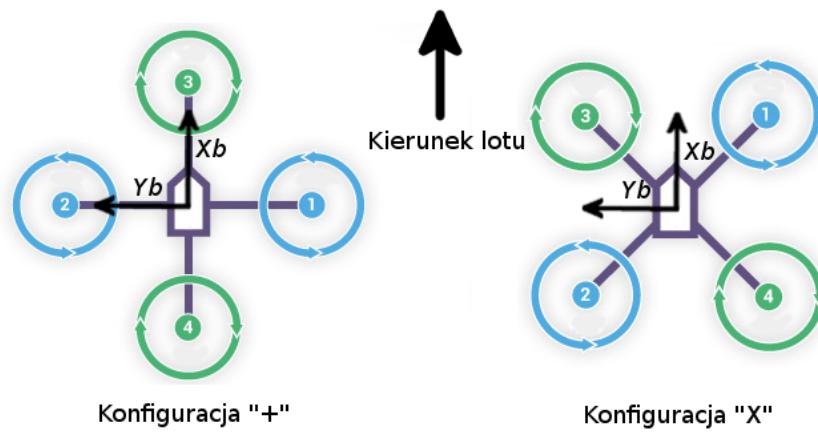
Najciekawszym przypadkiem do rozpatrzenia jest warunek ruchu poziomego. Aby kwadrokopter zaczął poruszać się wzdłuż osi X_e lub Y_e należy zmienić prędkości kątowe wirników tak, aby nastąpił obrót kwadrokoptera wokół osi odpowiednio Y_b lub X_b . Chcąc przykładowo uzyskać obrót wokół osi X_b (rys. 2.3), należy doprowadzić do braku równowagi między siłami ciągu generowanymi przez parę wirników 1 oraz 4 a siłami ciągu generowanymi przez parę wirników 2 oraz 3. Dzięki temu powstaje wypadkowy moment siły wokół osi X_b , powodujący przechylenie kwadrokoptera. Wypadkowa siła ciągu wszystkich silników nie działa już pionowo ku górze, przez co kwadrokopter zaczyna poruszać się w poziomie.



RYS. 2.4: Siły działające na kwadrokopter w trakcie ruchu postępowego

Jak widać na rysunku 2.4, w trakcie ruchu postępowego siła ciągu każdego wirnika dzieli się na dwie składowe - pionową i poziomą. Składowa pionowa ma na celu zrównoważenie siły grawitacji, natomiast składowa pozioma zapewnia ruch w poziomie. Warto zwrócić tu uwagę, że przy ruchu poziomym kwadrokoptera należy zwiększyć ciąg silników tak, aby składowa pionowa ciągu silnika nadal równoważyła siłę grawitacji.

W zależności od ustawienia wirników względem układu współrzędnych kwadrokoptera rozróżniamy jego dwie podstawowe konfiguracje - konfiguracja "+" oraz konfiguracja "X" [10, 11]



RYS. 2.5: Konfiguracje ustawień wirników względem osi kwadrokoptera

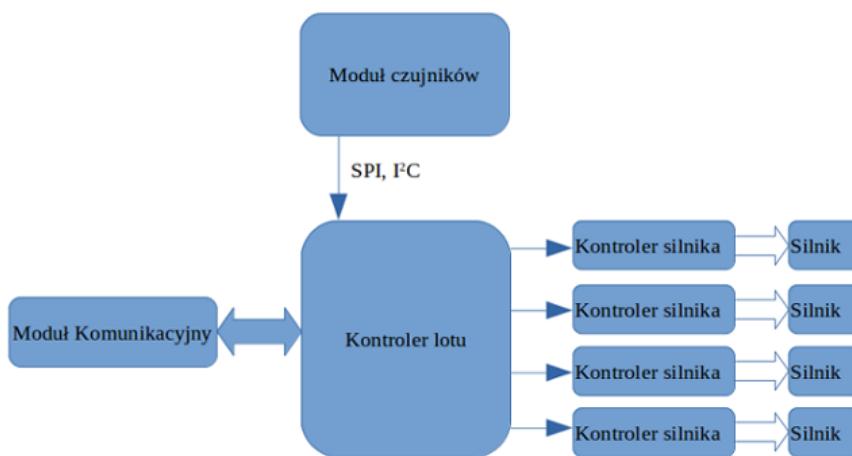
W przypadku konfiguracji "+" (Rys. 2.5) wirniki 1 oraz 2 umieszczone są na osi Y_b , natomiast wirniki 3 i 4 na osi X_b . Chcąc uzyskać ruch wzdłuż osi X_e lub Y_e należy zmienić prędkości obrotowe jedynie dwóch silników, podczas gdy prędkości obrotowe pozostałych dwóch pozostają bez zmian.

Sprawia to, że algorytm kontrolujący ruch kwadrokoptera jest prostszy niż dla konfiguracji "X". W konfiguracji "X" każdy z wirników leży pomiędzy osiami X_b i Y_b kwadrokoptera. W związku z tym, ruch wzduż osi X_e lub Y_e odbywa się dzięki zmianie prędkości obrotowych wszystkich wirników jednocześnie. Sprawia to, że algorytm sterujący będzie trudniejszy w implementacji, lecz niesie ze sobą jedną zaletę - chcąc uzyskać obrót kwadrokoptera np. wokół osi X_b przy konfiguracji "X" potrzebna będzie dwa razy mniejsza zmiana prędkości obrotowej każdego z czterech wirników niż dla konfiguracji "+" gdzie przy obrocie będą brały udział jedynie silniki 1 i 2. Dzięki temu konfiguracja "X" dużo lepiej sprawdza się w sytuacjach, gdzie silniki pracują blisko granicy maksymalnych obrotów (granicy nasycenia) [11].

2.2 Podstawowe podzespoły

W podstawowej konfiguracji system sterowania kwadrokopterem można podzielić na następujące moduły:

- kontroler lotu
- moduł czujników
- moduł komunikacyjny
- sterowniki silników
- silniki



RYS. 2.6: Podstawowe elementy składowe systemu sterowania kwadrokopterem

2.2.1 Kontroler lotu

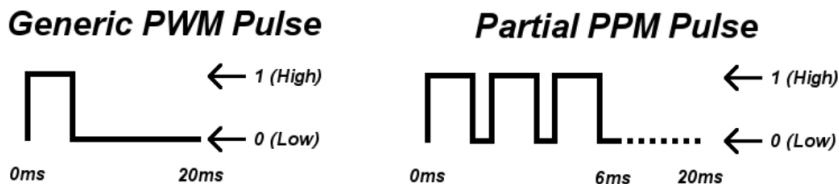
Kontroler lotu jest podstawowym modułem kwadrokoptera, odpowiedzialnym za stabilizację lotu oraz wykonywanie wszelkich manewrów. Składa się on zazwyczaj z mikrokontrolera wyposażonego w szeregowe lub równoległe interfejsy komunikacyjne, które umożliwiają odbieranie danych z modułu czujników oraz modułu komunikacyjnego, jak również przesyłanie danych do kontrolerów silników. Duża różnorodność obecnie dostępnych mikrokontrolerów sprawia, że można zaobserwować najrozmaitsze konstrukcje kontrolerów oparte o takie architektury jak AVR lub ARM, od własnoręcznie robionych płyt drukowanych po gotowe komputery jednopłytkowe (SBC), w tym ostatnio raspberry PI [12, 13]. W dziedzinie dostępnego oprogramowania również widoczna jest duża różnorodność. Wśród najbardziej popularnych rozwiązań wyróżnić można programy dedykowane do kontroli lotu, takie jak ArduPilot [14], programy napisane w oparciu o proste systemy operacyjne czasu rzeczywistego (RTOS), a także programy oparte o specjalne dystrybucje systemu LINUX [15].

2.2.2 Moduł czujników

Moduł czujników służy do zbierania informacji niezbędnych kontrolerowi lotu do utrzymania stabilizacji kwadrokoptera, jak również wykonywania odpowiednich manewrów. W ostatnich latach dzięki rozwojowi technologii MEMS [16] dostępne stały się układy scalone ze zintegrowanymi czujnikami, posiadające cyfrowe lub analogowe interfejsy komunikacyjne, zwane również IMU (ang. Inertial Measurement Unit) [17]. W najprostszej konfiguracji układy scalone wyposażone są w trzyosiowy akcelerometr i trzyosiowy żyroskop, dzięki czemu zapewniają komplet informacji potrzebnych dla kontrolera lotu. Niektóre moduły czujników oferują również sprzętową akcelerację obliczeń [18], zwalniając tym samym kontroler lotu z konieczności przeliczania położenia kwadrokoptera w przestrzeni.

2.2.3 Moduł komunikacyjny

Moduł komunikacyjny realizuje komunikację między pilotem a quadrokopterem. Obecnie najbardziej popularna droga komunikacji to droga radiowa lecz w tańszych konstrukcjach można spotkać jednostronną komunikację wykorzystującą promieniowanie podczerwone. Wśród prywatnych konstrukcji kwadrokopterów często można spotkać moduły radiowe pracujące w paśmie 2.4GHz lub 434MHz, wyposażone w interfejs SPI, a także znacznie prostsze w obsłudze moduły Bluetooth, których wadą jest znacznie mniejszy zasięg transmisji danych. Wśród gotowych modułów komunikacyjnych dostępnych na rynku najbardziej popularne są moduły pracujące w paśmie 2.4GHz, oferujące kilka analogowych kanałów wyjściowych z modulacją PWM lub PPM [19–22].



Rys. 2.7: Przykłady modulacji PWM i PPM używanych najczęściej w modelarstwie

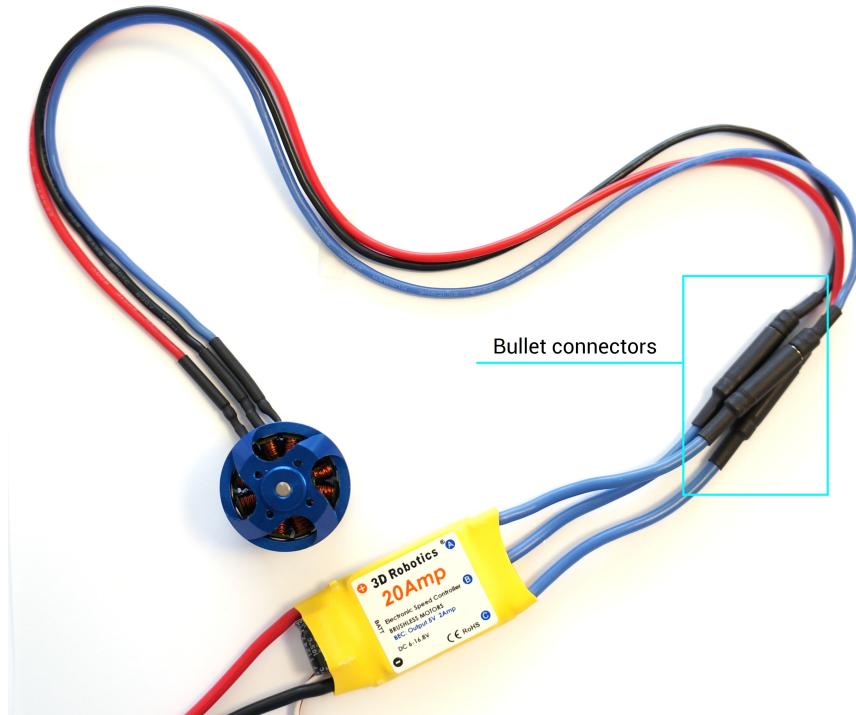
Rysunek 2.7 przedstawia dwie modulacje sygnałów najczęściej używane w modelarstwie.

Dla modulacji PWM, okres sygnału wynosi około 20ms, podczas gdy stan wysoki utrzymuje się w granicach od 1ms do 2ms. Taka modulacja sygnałów sterujących jest najczęściej stosowana do kontroli serwomechanizmów oraz prędkości obrotowych silników, gdzie szerokość impulsu 1ms oznacza brak obrotów a szerokość impulsu 2ms oznacza maksymalne obroty silnika.

Modulacja PPM polega na umieszczeniu kilku impulsów PWM w mniejszych odstępach czasowych, dzięki czemu zyskuje się możliwość sterowania serwomechanizmem lub prędkością silnika z większą rozdzielczością w czasie.

2.2.4 Sterowniki silników

Sterowniki silników odbierają dane z kontrolera lotu i na ich podstawie utrzymują zadaną prędkość obrotową silników. Stosowane są głównie w kwadrokopterach wykorzystujących silniki bezszczotkowe prądu stałego. Konieczność ich stosowania wynika z faktu pobierania znacznych prądów przez silniki, a także z dość skomplikowanej natury ich sterowania. Stworzenie oddzielnego modułu, który na wejściu przyjmuje sygnał sterujący o modulacji PWM lub PPM z kontrolera lotu oraz odpowiada za prawidłowe wysterowanie i utrzymanie stałej prędkości obrotowej silnika, w znacznym stopniu odciąża kontroler lotu. Dzięki zamontowaniu kontrolera silnika na oddzielnym PCB omija się też konieczność prowadzenia szerokich ścieżek zasilania silników oraz montowania tranzystorów mocy sterujących silnikami, co znacznie upraszcza projektowanie płytki drukowanej kontrolera lotu. Największą zaletą stosowania sterowników silników w postaci osobnych modułów jest możliwość przenoszenia kontrolera lotu między różnymi konstrukcjami nośnymi wyposażonymi w różne silniki i różne sterowniki silników. Jedyne o co trzeba zadbać, to zgodność interfejsów komunikacyjnych między wszystkimi sterownikami silników.



RYS. 2.8: Przykładowy sterownik silnika bezszczotkowego wraz z podłączonym silnikiem [23]

2.2.5 Silniki

Silniki wraz z przymocowanymi śmigłami zapewniają kwadrokopterowi siłę nośną. Można je podzielić na dwie klasy: silniki bezszczotkowe i silniki szczotkowe. Silniki bezszczotkowe charakteryzują się dużą mocą i sprawnością, lecz bardzo duży pobór prądu jest ich wadą. Silniki szczotkowe pobierają zdecydowanie mniej prądu niż silniki bezszczotkowe, lecz ich mała moc ogranicza maksymalną masę kwadrokoptera. Obecnie najczęściej stosowane są silniki bezszczotkowe, które dzięki dużej mocy umożliwiają dołączanie dodatkowych podzespołów do kwadrokoptera, takich jak moduł kamery ze stabilizacją, manipulatory, itp.

2.3 Konstrukcje nośne

Konstrukcja nośna, czyli rama kwadrokoptera, stanowi platformę do mocowania wszystkich podzespołów.



RYS. 2.9: Gotowa rama do kwadrokoptera [24]



RYS. 2.10: Rama kwadrokoptera zintegrowana z PCB [25]

Ramy kwadrokopterów można podzielić na trzy główne kategorie:

- Wykonana własnoręcznie
- Rama w postaci modułów do samodzielnego złożenia
- Rama zintegrowana z PCB

W przypadku wielu konstrukcji ich autorzy sami konstruują ramę, wykorzystując najczęściej takie materiały, jak aluminium lub włókno węglowe. Pozwala to dopasować projekt do swoich specyficznych potrzeb. W przypadku gdy potrzeby te nie są aż tak wyrafinowane, bardzo dobrym rozwiązaniem jest kupno gotowej ramy w formie modułów do samodzielnego złożenia (rys. 2.9). Dzięki takiemu rozwiązaniu omija się konieczność tworzenia całej mechaniki projektu od zera. Dla bardzo małych konstrukcji, przy których rama kwadrokoptera nie musi znosić dużych obciążzeń, a jednocześnie musi być jak najlżejsza, stosuje się ramę zintegrowaną z płytą drukowaną kontrolera lotu kwadrokoptera (rys. 2.10). Dzięki takiemu rozwiązaniu można uzyskać bardzo niski koszt oraz masę konstrukcji.

2.4 Kierunki rozwoju

Obecnie kwadrokoptery znajdują coraz szersze zastosowania, począwszy od kręcenia filmów oraz robienia zdjęć lotniczych, poprzez automatyczne patrolowanie niewielkich obszarów i przenoszenie niewielkich ładunków aż do zastosowań czysto rozrywkowych takich jak możliwość urządzenia wyścigów kwadrokopterów. Ich prostota konstrukcji oraz stale malejąca cena podzespołów sprawia, że grono użytkowników będzie w coraz większej mierze składać się z firm, znajdujących dla kwadrokopterów rozmaite zastosowania komercyjne.

Rozdział 3

Podstawowe algorytmy stabilizacji lotu kwadrokopterów

3.1 Podstawy matematyczne

Fizyczne podstawy lotu opisane w rozdziale 2 tłumaczą, że ruch postępowy kwadrokoptera w pionie jest wynikiem braku równowagi między wypadkową siłą ciągu wszystkich silników a siłą grawitacji działającą na urządzenie, natomiast ruch w płaszczyźnie poziomej dla konfiguracji przedstawionej na rysunku 2.3 jest wynikiem obrotu kwadrokoptera wokół osi X_b lub Y_b [8, 9]. Wiadomo również, że wypadkowa siła działająca na kwadrokopter może być zmieniana poprzez proporcjonalną zmianę prędkości obrotowych wszystkich czterech wirników jednocześnie, oraz że obrót kwadrokoptera wokół dowolnej jego osi jest wynikiem braku równowagi między prędkościami kątowymi odpowiednich par wirników. Informacje te prowadzą do konkluzji, że wszystkie sześć stopni swobody kwadrokoptera może być kontrolowanych jedynie za pomocą odpowiedniej zmiany prędkości obrotowych wirników. Powstaje zatem pytanie, w jaki sposób kontrolować prędkości obrotowe wirników, chcąc uzyskać zamierzoną zmianę położenia kwadrokoptera w przestrzeni.

Rozważmy model kwadrokoptera przedstawiony na rysunku 2.3. Dla tak oznaczonych osi oraz numeracji silników możemy zapisać następujący komplet równań [9]:

$$\begin{aligned}\dot{\phi} &= k_r((\omega_1 + \omega_4) - (\omega_2 + \omega_3)) \\ \dot{\theta} &= k_p((\omega_1 + \omega_2) - (\omega_3 + \omega_4)) \\ \dot{\psi} &= k_y((\omega_1 + \omega_3) - (\omega_2 + \omega_4)) \\ f &= k_t(\omega_1 + \omega_2 + \omega_3 + \omega_4)\end{aligned}\tag{3.1}$$

gdzie $\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$ są prędkościami kątowymi odpowiednio wokół osi X_b , Y_b , Z_b , wypadkowa siła ciągu silników została oznaczona jako f , a k_r , k_p , k_y , k_t są współczynnikami proporcjonalności zależnymi od własności fizycznych systemu. Po opuszczeniu nawiasów i uszeregowaniu zmennych, powyższy układ równań nabiera następującej postaci:

$$\begin{aligned}\dot{\phi} &= k_r\omega_1 - k_r\omega_2 - k_r\omega_3 + k_r\omega_4 \\ \dot{\theta} &= k_p\omega_1 + k_p\omega_2 - k_p\omega_3 - k_p\omega_4 \\ \dot{\psi} &= k_y\omega_1 - k_y\omega_2 + k_y\omega_3 - k_y\omega_4 \\ f &= k_t\omega_1 + k_t\omega_2 + k_t\omega_3 + k_t\omega_4\end{aligned}\tag{3.2}$$

Przyjmując w uproszczeniu, że $k_r = k_p = k_y = k_t = k$, możemy zapisać powyższy komplet równań w postaci macierzowej:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ f \end{pmatrix} = \begin{pmatrix} k & -k & -k & k \\ k & k & -k & -k \\ k & -k & k & -k \\ k & k & k & k \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix}\tag{3.3}$$

Jak widać, powyższy zestaw równań odpowiada na pytanie, w jaki sposób prędkości obrotowe kwadrokoptera wokół osi układu współrzędnych, oraz wypadkowa siła zależą od prędkości obrotowych wirników. Chcąc kontrolować położenie i orientację kwadrokoptera w przestrzeni musimy wyrazić prędkości obrotowe śmigieł jako funkcję zadanych prędkości obrotowych kwadrokoptera wokół osi układu odniesienia. Można tego dokonać przez nazwanie macierzy:

$$\begin{pmatrix} k & -k & -k & k \\ k & k & -k & -k \\ k & -k & k & -k \\ k & k & k & k \end{pmatrix} = K\tag{3.4}$$

a następnie obustronne pomnożenie równania 3.3 przez macierz K^{-1} .

Po tych operacjach, otrzymujemy równanie

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} = K^{-1} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ F \end{pmatrix}\tag{3.5}$$

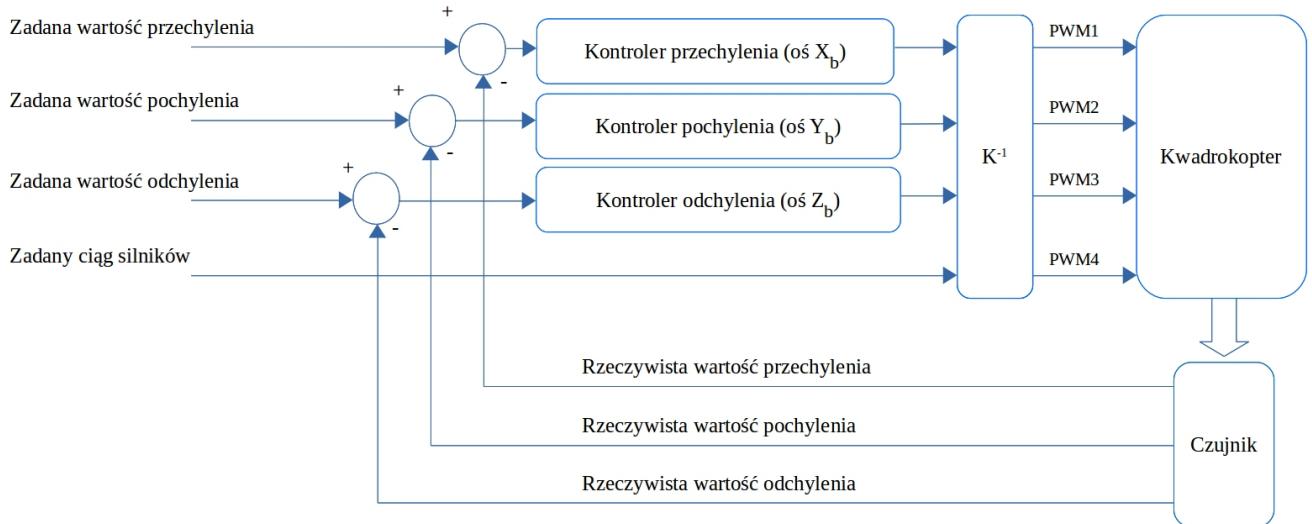
3.2 Algorytm kontroli prędkości obrotowych

Równanie 3.5 jest podstawą do stworzenia najprostszego algorytmu stabilizacji i kontroli lotu. Algorytm ten będzie liczył wartości sygnałów sterujących, przekładających się następnie na prędkości obrotowe śmigieł, na podstawie zadanych prędkości kątowych kwadrokoptera wokół osi układu współrzędnych, a także zadanego wypadkowego ciągu silników. Chcąc wyznaczać prędkości wirników w czysto matematyczny sposób, podczas implementacji algorytmu należałoby wyznaczyć wspomniane wcześniej współczynniki k_r , k_p , k_y , k_t , co byłoby stosunkowo trudne i czasochłonne. Podejście to ma również znaczącą wadę: wyliczone współczynniki zależałyby od fizycznych właściwości systemu (długość ramienia kwadrokoptera, masa śmigieł, kąt natarcia śmigieł itp.) i przy zmianie któregokolwiek z fizycznych parametrów modelu, określone wcześniej współczynniki należałoby przeliczyć na nowo, co w praktyce dyskwalifikuje taką formę algorytmu.

Alternatywą dla takiego podejścia jest zastosowanie algorytmu opartego o regulatory działające w pętli sprzężenia zwrotnego. Wykorzystuje on informację o rzeczywistych prędkościach obrotowych urządzenia porównując je z wartościami zadanyymi otrzymanymi na wejściu i na tej podstawie liczy odpowiednią poprawkę sygnałów sterujących silnikami [9].

Algorytm kontrolujący prędkość obrotową kwadrokoptera wokół jednej osi będzie składał się z następujących kroków:

1. Zmierz prędkość kątową wokół osi.
2. Porównaj zmierzoną prędkość kątową z wartością zadaną (oblicz błąd).
3. Wprowadź poprawkę w prędkości obrotowej wirników na podstawie wyliczonego błędu.
4. Idź do punktu 1.



Rys. 3.1: Najprostszy algorytm kontroli lotu kwadrokoptera

Rysunek 3.1 przedstawia strukturę omawianego algorytmu, zdolnego do kontroli prędkości obrotowej kwadrokoptera wokół każdej z trzech osi. Wartości zadane prędkości obrotowych wokół trzech osi układu współrzędnych transmitowane są drogą radiową, po czym trafiają na wejścia trzech regulatorów, odpowiedzialnych za utrzymanie stałych prędkości kątowych. Wartości wyjściowe z regulatorów wraz z wypadkową wartością ciągu wszystkich silników (również transmitowaną radiowo) konwertowane są następnie na wartości czterech sygnałów PWM, sterujących silnikami. Efektem sterowania silnikami jest pewien określony obrót kwadrokoptera w trójwymiarowej przestrzeni, który jest z kolei mierzony za pomocą czujnika (na przykład trzyosiowego żyroskopu). Rzeczywiste wartości prędkości kątowych kwadrokoptera są następnie wykorzystywane przez regulatorów w pętli sprzężenia zwrotnego, w celu wprowadzenia ewentualnej poprawki w wartościach sygnałów PWM. Algorytm nie precyzuje jaki rodzaj kontrolera ma być użyty, zatem najczęściej używany jest regulator PID, głównie ze względu na prostotę implementacji oraz szeroki zasób materiałów dostępnych na jego temat [9].

Omawiany algorytm jest bardzo prosty w implementacji, dzięki czemu można go zastosować nawet w mikrokontrolerach, wyposażonych w niewielkie zasoby sprzętowe. Daje on bardzo dużą kontrolę nad kwadrokopterem - przy ustaleniu zadanej prędkości obrotowej na przykład wokół osi X_b na wartość 30° s^{-1} kwadrokopter będzie się obracał z taką prędkością niezależnie od jego położenia względem ziemi. Pozwala to na wykonywanie rozmawitych akrobacji, jednakże wymaga od pilota dużej wprawy w sterowaniu maszyną, ze względu na konieczność ręcznej stabilizacji położenia kwadrokoptera względem ziemi. Dla przykładu, jeśli wychylimy drążek przechylenia, kwadrokopter w konfiguracji pokazanej na rysunku 2.3 otrzyma informację odnośnie prędkości obrotowej wokół osi Y_b i zacznie poruszać się wzduż osi X_e . W momencie gdy użytkownik cofnie drążek wychylenia do pozycji spoczynkowej kwadrokopter przestanie obracać się wokół osi Y_b , jednak ruch wzduż osi

X_e nie ustanie. Chcąc powrócić do stanu zawisu w powietrzu, użytkownik sam musi przesunąć drążek wychylenia w przeciwną stronę, tak aby kwadrokopter powrócił do pozycji poziomej.

3.3 Algorytm kontroli położenia kątowego

Dla wielu użytkowników brak automatycznej stabilizacji ruchu w płaszczyźnie poziomej może być uciążliwy, dlatego też warto zastanowić się co należy zrobić, aby zamiast kontroli nad prędkościami kątowymi wokół osi X_b oraz Y_b móc kontrolować kąt przechylenia (ϕ) lub kąt wychylenia kwadrokoptera (θ). Najprostszym rozwiązaniem będzie rozszerzenie algorytmu przedstawionego na rysunku 3.1 tak, aby na wejściu zamiast prędkości kątowych wokół osi X_b i Y_b przyjmował docelowe wartości kątów przechylenia i odchylenia. Od razu w oczy rzuca się podstawowy problem: w poprzedniej wersji algorytmu, chcąc kontrolować prędkość kątową, wykorzystywano rzeczywistą wartość prędkości kątowej kwadrokoptera w pętli sprzężenia zwrotnego. Przy chęci kontrolowania kąta przechylenia lub wychylenia, rozszerzona wersja algorytmu musi wykorzystać w pętli sprzężenia zwrotnego informację o rzeczywistym kącie wychylenia lub przechylenia o jaki obrócił się kwadrokopter. Dlatego też aby omawiany algorytm miał prawo działać, należy zastanowić się, w jaki sposób można uzyskać wspomniane kąty.

Obecnie dostępne czujniki, które można zamontować na prostym modelu kwadrokoptera, nie umożliwiają bezpośredniego pomiaru kątów przechylenia i wychylenia, dlatego też trzeba będzie sięgnąć do bardziej wyrafinowanych metod, bazujących na operacjach matematycznych dokonywanych na wynikach pomiarów z czujników takich jak:

- Żydroskop
- Akcelerometr

Wykorzystanie żyroskopu polega na całkowaniu jego pomiarów (prędkości kątowych) w czasie, tak aby uzyskać wartości kątów obrotu wokół odpowiednich osi. Rozwiązanie to ma jednak jedną wadę - sygnały wyjściowe obecnie produkowanych żyroskopów zawsze obarczone są niewielkim błędem (zależnym od temperatury i zmiennym w czasie), który będąc całkowany razem z rzeczywistą wartością prędkości obrotowej doprowadziłby do znacznej różnicy między rzeczywistą wartością kąta obrotu a wartością wynikającą z policzonej całki [9, 26]. Sprawia to, że pomiar kąta obrotu dokonany za pomocą takiej metody nie jest miarodajny.

Wykorzystanie akcelerometru polega na odczytywaniu przyspieszeń liniowych wzdłuż każdej z osi kwadrokoptera, a następnie na przekształceniu ich za pomocą funkcji trygonometrycznych na wartości kątów przechylenia i wychylenia zgodnie z poniższymi wzorami [27]:

$$\phi = \arctan\left(\frac{a_y}{\sqrt{a_z^2 + a_x^2}}\right) \quad (3.6)$$

$$\theta = \arctan\left(\frac{a_x}{\sqrt{a_z^2 + a_y^2}}\right) \quad (3.7)$$

gdzie wartości a_x, a_y, a_z są wartościami przyspieszeń liniowych mierzonych wzdłuż osi odpowiednio X_b, Y_b, Z_b .

Podane rozwiązań nie jest jednak idealne. Zadziałanie na kwadrokopter zewnętrznej siły (np. pudmuch wiatru) nadającej mu dodatkową wartość przyspieszenia może powodować błędą interpretację odbieranych pomiarów przez algorytm kontroli lotu. Co więcej, akcelerometr jest czujnikiem bardzo wrażliwym na wibracje, które będą powodować błąd między rzeczywistym kątem pochylenia lub przechylenia a kątem obliczonym na podstawie pomiarów.

Reasumując mamy dwie metody estymacji kąta pochylenia i przechylenia, obarczone następującymi niedoskonałościami:

- Całkowanie prędkości kątowej w czasie:
 - wartość wyjściowa obarczona błędem, który całkowany w czasie powoduje coraz większą odchyłkę wartości rzeczywistej kąta od wartości obliczonej.
- Wykorzystanie wskazań z akcelerometru i funkcji trygonometrycznych:
 - zewnętrzne siły mogą powodować błędą interpretację pomiarów przez algorytm kontroli lotu,
 - duża czułość akcelerometru na wibracje powoduje niedokładność estymacji kąta.

W świetle niedoskonałości każdej z dwóch wspomnianych metod, rozwiązaniem problemu będzie połączenie wartości pomiarów obu czujników i wykorzystanie odpowiedniej funkcji fuzji danych z sensorów (ang. sensor fusion function), umożliwiającej estymację położenia kwadrokoptera w przestrzeni [26].

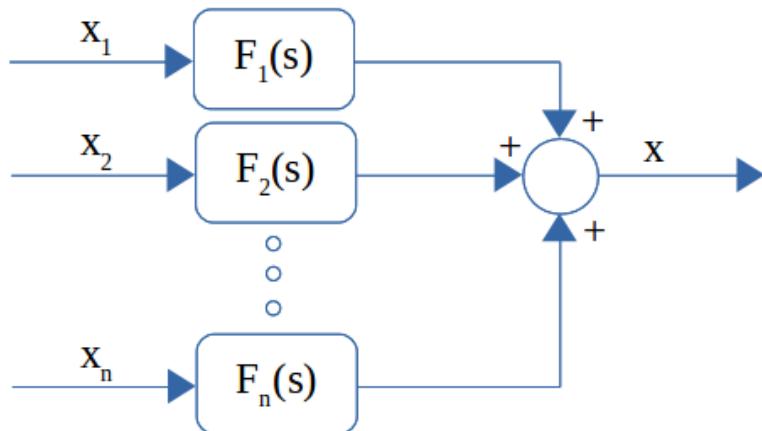
Wśród wielu znanych funkcji, które mogą zostać użyte do obliczania położenia kwadrokoptera w przestrzeni, wyróżnić można dwie najbardziej popularne [26, 28, 29]:

- Filtr Kalmana
- Filtry komplementarne

Filtr Kalmana, jest to algorytm rekurencyjny, który służy do określenia stanu badanego układu dynamicznego na podstawie jego modelu matematycznego (np. równań ruchu) oraz serii pomiarów wejścia i wyjścia tego układu. Algorytm ten, składa się z dwóch kroków:

- Faza predykcji - służy do określenia przewidywanego następnego stanu procesu, w oparciu o znajomość poprzedniego stanu procesu.
- Faza korekcji - na podstawie rzeczywistych wartości pomiarów dokonuje się aktualizacji estymaty stanu procesu, a także określa się jak bardzo wartość estymaty z kroku poprzedniego odbiegała od wartości rzeczywistej

Filtr Kalmana jest bardzo popularny w zastosowaniach łączenia danych z sensorów w celu określenia położenia statków powietrznych, jednakże jego złożoność obliczeniowa dyskwalifikuje go do zastosowań w małych kontrolerach lotu (np. opartych o architekturę AVR).



RYS. 3.2: Struktura filtru komplementarnego [28]

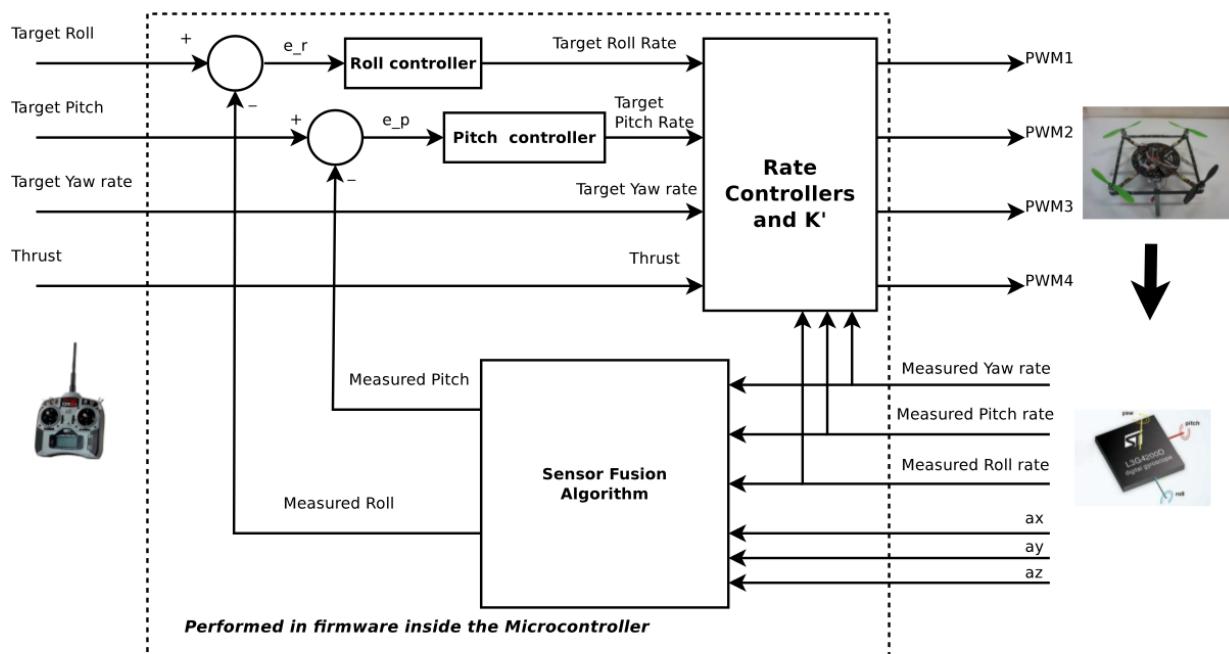
Rysunek 3.2 przedstawia strukturę filtru komplementarnego. Idea jego działania polega na łączeniu danych z różnych czujników, z których każdy narażony jest na zakłócenia o różnych częstotliwościach, usuwanie zakłóceń za pomocą odpowiednich filtrów (dolnoprzepustowych, górnoprzepustowych lub pasmowoprzepustowych) a następnie łączeniu przefiltrowanych sygnałów w wartość ostateczną. Filtr komplementarny powinien spełniać następujący warunek [28]:

$$\sum_{i=1}^n F_i(s) = 1 \quad (3.8)$$

Rzutując tą ideę na model kwadrokoptera, filtr taki może zostać użyty do łączenia danych z żyroskopu oraz akcelerometru. Wartości z żyroskopu dryfują w czasie, a zatem obarczone są błędem o niskiej częstotliwości, dlatego też do ich filtrowania użyty zostanie filtr górnoprzepustowy.

Wartości z akcelerometrów zakłócione są drganiami generowanymi przez silniki oraz ruchami kwadrokoptera, a zatem są to zakłócenia o wysokich częstotliwościach i do ich zniwelowania użyty zostanie filtr dolnoprzepustowy. Następnie przefiltrowane wartości po dokonaniu odpowiednich operacji matematycznych, mogą zostać wykorzystane do obliczenia kąta przechylenia i pochylenia kwadrokoptera.

Mając wiedzę na temat najbardziej popularnych funkcji fuzji danych z czujników, możemy powrócić do omawiania algorytmu kontroli położenia kwadrokoptera w przestrzeni, którego struktura została przedstawiona na rysunku 3.3.



RYS. 3.3: Algorytm kontroli położenia kwadrokoptera [9]

Jak widać w tej wersji algorytmu drogą radiową przesyłane są (oprócz wartości wypadkowego ciągu oraz zadanej prędkości kątowej wokół osi Z_b) docelowe wartości kątów przechylenia i pochylenia, jakie ma osiągnąć kwadrokopter. Wartości te trafiają razem z rzeczywistymi kątami przechylenia i pochylenia, o jakie obecnie obrócił się kwadrokopter na wejścia dwóch regulatorów, odpowiedzialnych za utrzymanie zadanego położenia kątowego w płaszczyźnie poziomej. Sygnały wyjściowe z wspomnianych dwóch regulatorów są prędkościami kątowymi wokół osi X_b i Y_b , jakie kwadrokopter powinien uzyskać w danej chwili, aby dążyć do docelowej orientacji w przestrzeni. Trafiają one wraz z wartością wypadkowego ciągu silników oraz zadana prędkością kątową wokół osi Z_b do bloku kontrolującego prędkości kątowe wszystkich osi układu odniesienia, czyli do algorytmu omawianego w poprzednim podrozdziale.

Algorytm w takiej postaci będzie zdolny do automatycznego wypoziomowania kwadrokoptera po puszczeniu przez pilota drążka przechylenia/wychylenia. Jest on bardziej złożony obliczeniowo oraz

daje mniejszą kontrolę nad kwadrokopterem w porównaniu do algorytmu omawianego w poprzednim podrozdziale, jednakże ze względu na wygodę użytkowania jest częściej stosowany w amatorskich konstrukcjach.

Rozdział 4

Czujniki stosowane w kwadrokopterach

4.1 Rodzaje i kryteria doboru czujników

W rozdziale 3 opisano podstawowe algorytmy kontroli i stabilizacji lotu kwadrokopterów. Algorytmy te łączyła jedna podstawowa cecha wspólna - wykorzystanie sprzężenia zwrotnego, w którym wiadomość o rzeczywistym stanie systemu (prędkości obrotowe lub położenie kątowe) była przekazywana jako jedna z informacji na wejście regulatorów wykorzystanych w algorytmie. Informacja ta pochodzi z odpowiednich czujników: akcelerometrów i żyroskopów. Dzięki niej kontroler lotu może dokonywać odpowiednich poprawek w sygnałach sterujących silnikami w celu utrzymania zadanych parametrów lotu. Widać zatem, że czujniki odgrywają jedną z kluczowych ról w całym procesie stabilizacji i kontroli lotu kwadrokoptera.

Obecnie na rynku dostępne są rozmaite czujniki zdolne do pomiarów różnych wielkości fizycznych. Do najbardziej popularnych czujników, z których korzysta się przy konstruowaniu modeli latających należą:

- **Akcelerometry** - czujniki służące do pomiaru wartości przyspieszeń liniowych
- **Żyroskopy** - czujniki służące do pomiaru wartości prędkości kątowych
- **Magnetometry** - czujniki służące do pomiaru natężenia pól magnetycznych
- **Barometry** - czujniki służące do pomiaru ciśnienia atmosferycznego

Wszystkie te czujniki mogą zostać użyte do określania położenia kwadrokptera w przestrzeni, jednakże dwa z nich (akcelerometry i żyroskopy) stanowią niezbędne minimum i tylko one będą omawiane szerzej w dalszej części tego rozdziału.

Po zawężeniu obszaru poszukiwań czujników jedynie do żyroskopów i akcelerometrów, użytkownik nadal pozostaje z bardzo szerokim wachlarzem dostępnych sensorów. Należy zatem dokonać wyboru elementu najbardziej pasującego do wymagań aplikacji na podstawie parametrów takich jak:

- Rodzaj wyjścia
 - Analogowe
 - Cyfrowe (np. I²C lub SPI)
- Maksymalny zakres pomiarowy
- Dokładność pomiarowa
- Maksymalna częstotliwość próbkowania
- Zakres napięć zasilania

Z punktu widzenia potencjalnego użytkownika kluczowymi parametrami dobieranych czujników są interfejsy komunikacyjne (w przypadku interfejsów cyfrowych również rodzaj użytego protokołu) a także maksymalny zakres pomiarowy.

4.2 Technologia MEMS

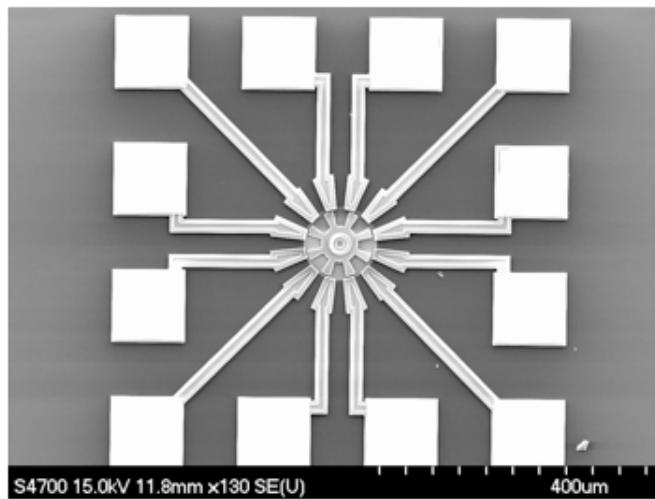
Rozmaitość typów czujników dostępnych na rynku wiąże się z jeszcze większą rozmaitością dostępnych technologii ich wykonania. Wśród popularnych technologii można wyróżnić:

- Mechaniczne
- Optyczne
- **Elektromechaniczne - technologia MEMS**

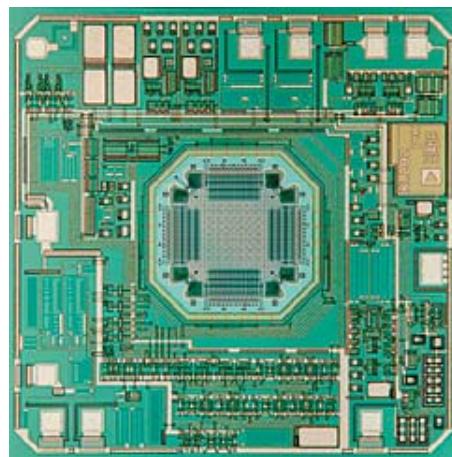
Wiele obecnie wykorzystywanych technologii umożliwia stworzenie bardzo czułych i dokładnych czujników (np. żyroskopy laserowe należą do jednych z najczulszych i najdokładniejszych żyroskopów). Posiadają one jednak poważne wady z punktu widzenia montażu na modelu kwadrokoptera, takie jak zbyt duże rozmiary i zbyt duża masa.

W ostatnich latach rozwój technologii MEMS umożliwił tworzenie czujników, które dzięki swoim niewielkim rozmiarom oraz małej masie stanowią idealne rozwiązanie dla bezzałogowych konstrukcji latających.

MEMS (ang. Micro Electro Mechanical Systems) jest terminem określającym technologię wytwarzania miniaturowych systemów elektromechanicznych, których wymiary fizyczne mieszczą się w przedziale od milimetrów do pojedynczych mikrometrów [30–32]. Dzięki omawianej technologii można wykonywać rozmaite rodzaje urządzeń, od nieskomplikowanych systemów bez żadnych ruchomych elementów po bardzo złożone struktury zawierające dziesiątki ruchomych elementów sterowanych za pomocą zintegrowanych układów elektronicznych.



Rys. 4.1: Miniaturowy silnik wykonany w technologii MEMS [33]



Rys. 4.2: Struktura trzyosiowego akcelerometru wraz ze zintegrowanymi układami elektronicznymi [34]

Rysunki 4.1 oraz 4.2 przedstawiają przykładowe struktury wykonane przy użyciu technologii MEMS.

Do produkcji struktur MEMS, najczęściej wykorzystuje się krzem, głównie ze względu na jego właściwości fizyczne oraz z uwagi na bardzo dużą dostępność płyt monokryształów tego pierwiastka, stanowiących podłoże produkcyjne. Dzięki temu struktury MEMS mogą być integrowane

z układami elektronicznymi w obrębie pojedynczego układu scalonego. Widać to na przykładzie rysunku 4.2, prezentującego jedną z możliwych konstrukcji trzyosiowego akcelerometru. Specjalizowane układy elektroniczne mierzą przyspieszenia liniowe działające na centralnie umieszczoną strukturę MEMS. Dzięki takim rozwiązaniom możliwe jest tworzenie miniaturowych elektromechanicznych czujników, które przy zachowaniu przyzwoitych parametrów cechują się bardzo niską ceną.

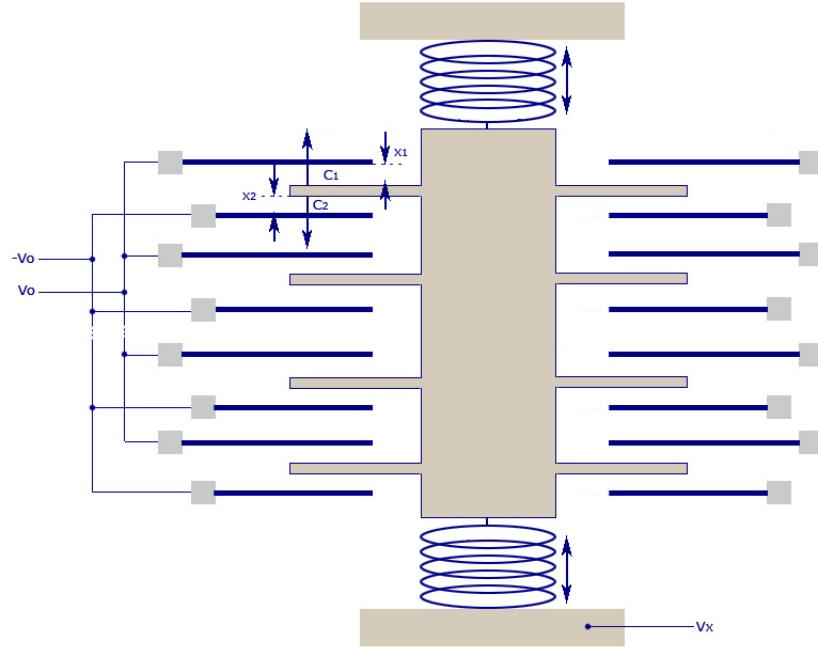
4.3 Akcelerometr

Istnieje wiele sposobów na stworzenie akcelerometru. Jedną z nich jest wykorzystanie materiałów piezoelektrycznych i mierzenie wytwarzanych przez nie napięć, proporcjonalnych do wartości przyspieszenia. Czujniki wykonane w tej technologii są jednak zazwyczaj stosunkowo duże i nie nadawałyby się do wykorzystania w konstrukcjach latających. Akcelerometry MEMS wykorzystują zmianę pojemności kondensatorów płaskich, będących elementem ich elektromechanicznej struktury [31].

Metoda pomiaru pojemności kondensatorów niesie ze sobą wiele zalet. Proces produkcji takich struktur jest bardzo prosty i nie wymaga dużych nakładów pracy. Dodatkowo pojemność kondensatorów złożonych z równoległych powierzchni wyrażana wzorem:

$$C = \epsilon_0 \epsilon_r \frac{S}{d} \quad (4.1)$$

gdzie ϵ_0 oznacza względna przenikalność elektryczną próżni, ϵ_r względna przenikalność elektryczną izolatora między okładkami kondensatora, S powierzchnię okładek, a d odległość między okładkami, nie zależy od temperatury, dzięki czemu czujniki zbudowane w oparciu o pomiar pojemności cechują się dużą dokładnością. Z powyższego wzoru widać, że kondensator płaski może również pełnić rolę na przykład czujnika wilgotności - wartość ϵ_r będzie zmieniać się wraz ze zmianą wartości pary wodnej w powietrzu. Jednak w przypadku akcelerometrów interesować nas będzie zmiana pojemności, wynikająca ze zmiany odległości między okładkami, która jest proporcjonalna do przyspieszeń linowych.



RYS. 4.3: Zasada działania akcelerometru MEMS

Rysunek 4.3 przedstawia budowę i zasadę działania akcelerometru MEMS [31, 35]. Składa się on z ruchomej masy odniesienia, zamocowanej za pomocą sprężyn do ramy pokrywającej się z układem odniesienia urządzenia. Masa odniesienia posiada okładki, które wraz z nieruchomymi okładkami zewnętrznej ramy tworzą kondensatory płaskie. Przesunięcie liniowe masy odniesienia, proporcjonalne do przyspieszenia działającego na układ może być mierzone za pomocą różnicy pojemności C_1 i C_2 , co opisują poniższe wzory [31]:

$$\begin{aligned} C_1 &= \epsilon_0 \epsilon_r \frac{S}{d + x} = C_0 - \Delta C_1 \\ C_2 &= \epsilon_0 \epsilon_r \frac{S}{d - x} = C_0 + \Delta C_2 \end{aligned} \quad (4.2)$$

W przypadku braku przyspieszenia działającego na układ, różnica pojemności wynosi 0 ponieważ $x_1 = x_2$. W przypadku działania przyspieszenia, różnicę pojemności wynikającą z faktu że $x_1 \neq x_2$ można wyrazić wzorem:

$$C_2 - C_1 = \Delta C_2 + \Delta C_1 = 2\epsilon_0 \epsilon_r S \frac{x}{d^2 - x^2} \quad (4.3)$$

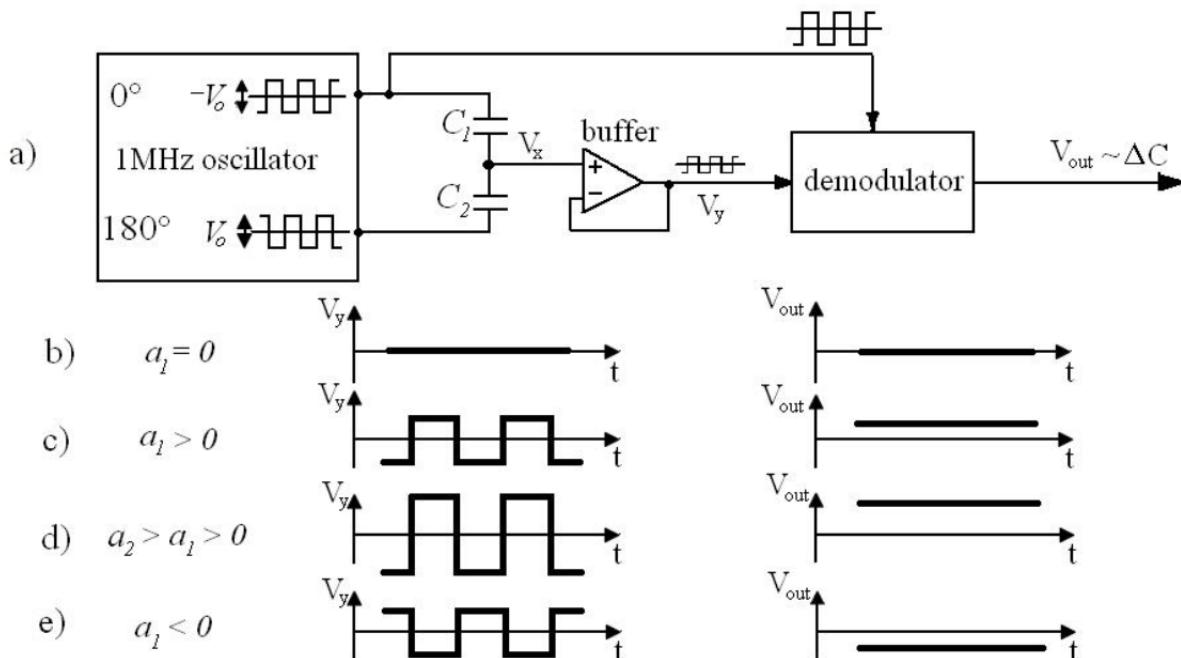
Chcąc wyznaczyć przesunięcie x, należy rozwiązać równanie kwadratowe:

$$\Delta C x^2 + \epsilon_0 \epsilon_r S x - \Delta C d^2 = 0 \quad (4.4)$$

Równanie to można jednak uprościć, jako że dla niewielkich przesunięć człon ΔCx^2 można pominać. Zatem równanie kwadratowe upraszcza się do postaci:

$$x \approx \frac{d^2}{\epsilon_0 \epsilon_r S} \Delta C = d \frac{\Delta C}{C_0} \quad (4.5)$$

Jak widać na rysunku 4.3 pojemności C_1 oraz C_2 składają się z wielu (z reguły około kilkudziesięciu) kondensatorów płaskich połączonych równolegle. Gdyby nie zastosować takiej konstrukcji, pojemność wytwarzana przez pojedynczą parę okładek byłaby tak mała, że powodowałoby to znaczne trudności podczas jej pomiaru.



RYS. 4.4: a) Układ elektryczny mierzący przyspieszenia na podstawie zmiany pojemności kondensatorów b,c,d,e) przebiegi sygnału na wyjściu wtórnika napięciowego oraz wartości napięć wyjściowych układu, dla różnych wartości przyspieszeń [31, 35]

Na rysunku 4.4 przedstawiono najprostszą postać układu służącego do pomiaru przyspieszeń liniiowych na podstawie zmiany pojemności kondensatorów. Składa się on z oscylatora, generującego dwa przebiegi prostokątne o częstotliwości 1MHz oraz amplitudzie V_0 , przesunięte względem siebie w fazie o 180°. Oba te przebiegi podłączone do pary kondensatorów C_1 i C_2 generują sygnał V_x , który jest napięciem masy odniesienia. Można to zapisać za pomocą następujących wzorów:

$$(V_x + V_0)C_1 + (V_x - V_0)C_2 = 0 \quad (4.6)$$

Korzystając z równań 4.2 oraz 4.5, powyższą zależność można przedstawić w następującej postaci:

$$V_x = V_0 \frac{C_2 - C_1}{C_2 + C_1} = \frac{x}{d} V_0 \quad (4.7)$$

zatem V_x jest również przebiegiem prostokątnym, o amplitudzie proporcjonalnej do liniowego przesunięcia masy odniesienia. Sygnał ten jest jednak bardzo słaby, dlatego też trafia dalej do układu, który go wzmacnia (w przypadku układu z rysunku 4.4 - jest to zwykły wtórnik napięciowy) a następnie trafia do demodulatora wraz z jednym z sygnałów oscylatora. Na wyjściu demodulatora można już zaobserwować sygnał, którego wartość jest proporcjonalna do przyspieszenia. Ostatnim z czynników, który musi być brany pod uwagę podczas wyznaczania przyspieszenia za pomocą tego typu czujnika są sprężyny, którymi masa odniesienia zamocowana jest do zewnętrznej ramy. Na podstawie prawa Hooke'a wiadomo, że zmiana długości sprężyny x jest proporcjonalna do zewnętrznej siły F_s która na nią działa. Łącząc prawo Hooke'a z drugą zasadą dynamiki Newtona, można zapisać następujące równanie, opisujące zależność między przemieszczeniem x masy odniesienia akcelerometru a przyspieszeniem, które na nią działa [31]:

$$a = \frac{k_s}{m} x \quad (4.8)$$

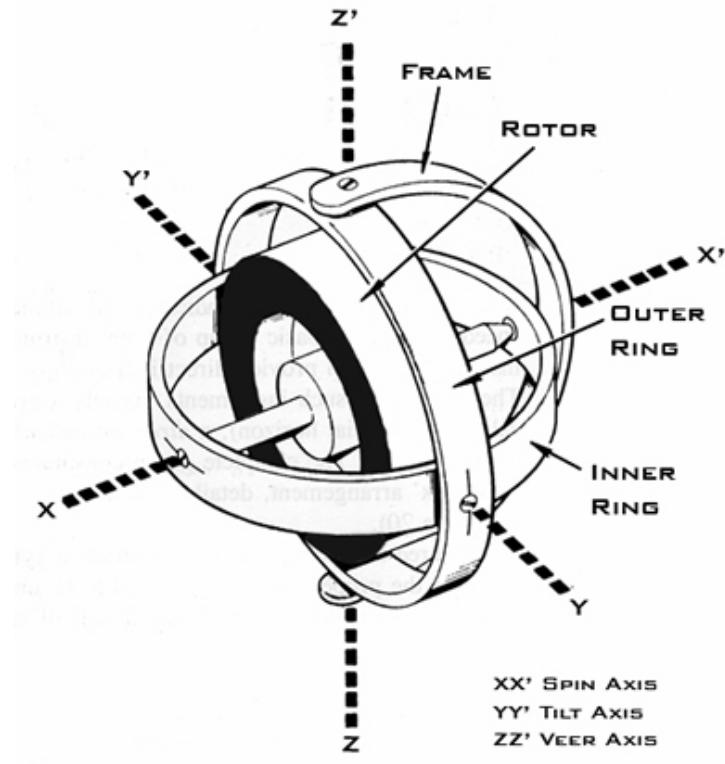
gdzie a to wartość przyspieszenia, k_s to współczynnik sprężystości sprężyn mocujących masę odniesienia do zewnętrznej ramy, m to masa ruchomego elementu (elektrycznej masy odniesienia), a x to wartość przemieszczenia. Wykorzystując równanie 4.7 można zapisać ostateczną postać wzoru, pozwalającą wyznaczyć wartość przyspieszenia w funkcji napięcia wyjściowego układu przedstawionego na rysunku 4.4

$$a = \frac{k_s d}{m V_0} V_x \quad (4.9)$$

4.4 Żyroskop

Żyroskop służy pomiaru prędkości obrotowej. Ponieważ stabilizacja lotu kwadrokoptera w najprostszej postaci polega na utrzymaniu stałej prędkości kątowej wokół każdej z trzech osi, żyroskop jest kluczowym czujnikiem, w jaki wyposaża się kwadrokopter [9, 19].

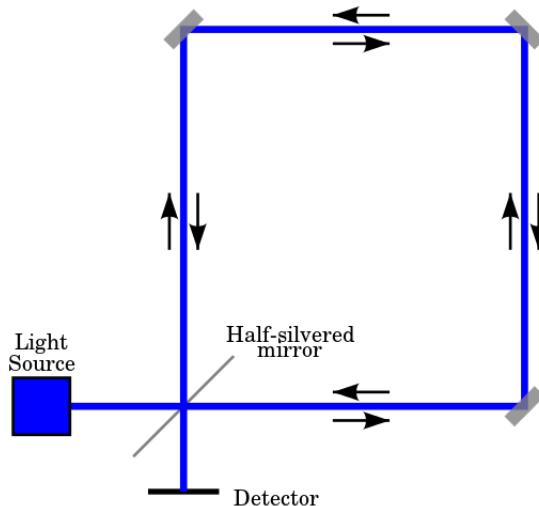
Żyroskopy MEMS należą do grupy żyroskopów z wibrującą strukturą, mierzących prędkość kątową dzięki sile Coriolisa. W wyniku jej działania ciało, (w żyroskopach jest to z reguły odpowiednio przygotowana miniaturowa płytka krzemiu) wprawione w wibracje w określonej płaszczyźnie będzie dążyć do utrzymania wibracji w tej płaszczyźnie mimo obrotu ramy, do której ciało to jest zamocowane. Chcąc uświadomić sobie, jak innowacyjna jest ta konstrukcja, warto najpierw omówić dwa najbardziej popularne typy żyroskopów - żyroskop mechaniczny oraz żyroskop laserowy.



RYS. 4.5: Przykład żyroskopu mechanicznego [36]

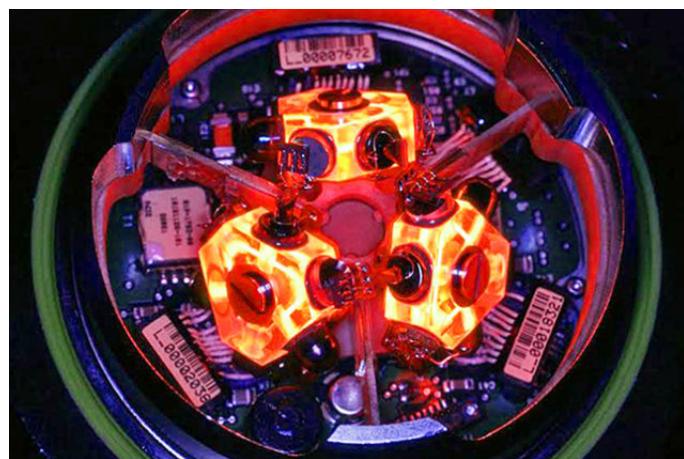
Rysunek 4.5 przedstawia przykładową budowę żyroskopu mechanicznego. Głównym elementem jest dysk wirujący z dużą prędkością, który utrzymuje stałe położenie kątowe. Osadzony jest w ramie, zazwyczaj w formie trzech pierścieni, umożliwiającej jego swobodny obrót wokół trzech osi. Dzięki temu wirująca masa może zachować swoje stałe położenie kątowe względem Ziemi niezależnie od obrotu urządzenia, w którym zamontowano żyroskop. Gdy w takiej konstrukcji mechanicznej, na stykach pierścieni ramy zamontowane zostaną enkodery absolutne [37] będzie możliwe odczytanie położenia kątowego żyroskopu w dowolnej chwili czasu, na podstawie którego będzie można wprost określić położenie kątowe urządzenia, w którym zastosowano ten czujnik.

Zdecydowaną wadą żyroskopów mechanicznych jest ich duża masa, oraz występowanie w nich wnętrzu wirujących elementów. W celu eliminacji tych wad, opracowano nowe rodzaje żyroskopów, takie jak żyroskopy laserowe, działające w oparciu o efekt Sagnaca.



RYS. 4.6: Zasada działania żyroskopów laserowych [38]

Rysunek 4.6 przedstawia zasadę działania żyroskopu laserowego, zwanego również interferometrem Sagnaca. Promień światła rostaje rodzielony na dwie wiązki, które następnie przebywają tę samą drogę w przeciwnych kierunkach. W wyniku ruchu obrotowego urządzenia, powstaje subtelna różnica w fazie między dwoma wiązkami, którą następnie można zauważać w postaci prążków interferyencyjnych. Żyroskopy wykorzystujące efekt Sagnaca należą do grupy bardzo precyzyjnych czujników, częściowo ze względu na swoją niewrażliwość na przyspieszenia liniowe oraz wibracje. Z tego względu stosuje się je głównie z aplikacjach lotniczych (samoloty pasażerskie), militarnych (zdalnie naprowadzane rakiety), oraz astronautycznych (międzynarodowa stacja kosmiczna) [39, 40].

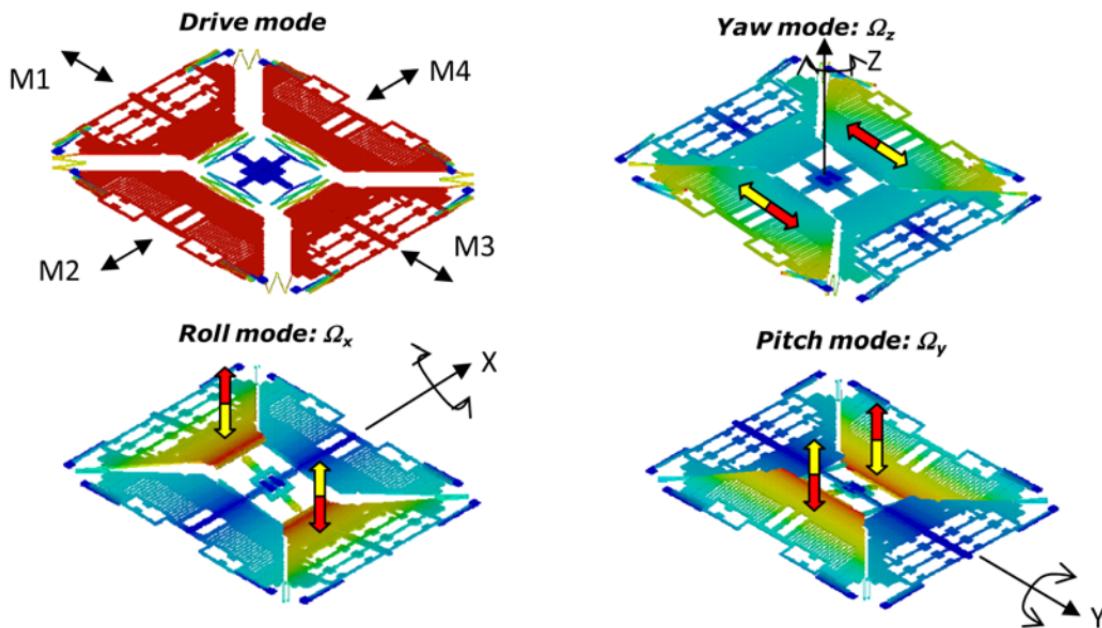


RYS. 4.7: Trzyosiowy żyroskop laserowy [39]

Rysunek 4.7 przedstawia przykładową konstrukcję trzyosiowego żyroskopu laserowego, używanego w samolotach wojskowych oraz zdalnie naprowadzanych rakietach.

Jak już wcześniej wspomniano, żyroskopy MEMS korzystają z zupełnie innych zasad fizycznych niż żyroskopy mechaniczne i laserowe. Zasada ich działania zostanie wyjaśniona na przykładzie

trzyosiowego żyroskopu, wykorzystującego odpowiednio ukształtowaną płytę krzemową do pomiaru prędkości kątowych, z jakimi czujnik obraca się wokół osi układu współrzędnych.



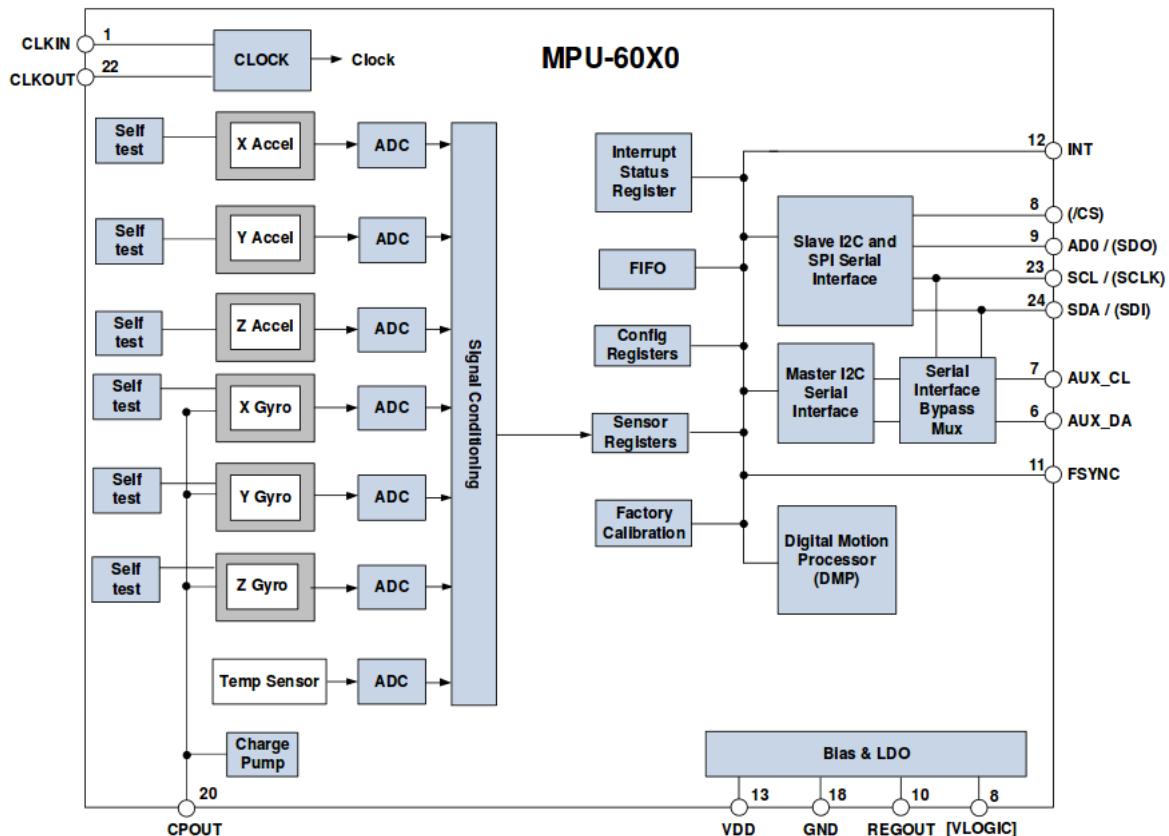
RYS. 4.8: Trzyosiowy żyroskop MEMS [41]

W żyroskopie, którego struktura pokazana jest na rysunku 4.8, mamy do czynienia z czterema masami M_1, M_2, M_3, M_4 przyjmowanymi za pomocą sprężyn do ramy stanowiącej punkt odniesienia i wibrującymi jednocześnie w płaszczyźnie poziomej. Dzięki takiej strukturze można wykryć prędkości kątowe wokół wszystkich trzech osi. Przy obrocie struktury wokół osi X masy M_1 oraz M_3 w wyniku działania siły Coriolisa odchylają się w przeciwnie strony wzdłuż osi prostopadłej do płaszczyzny drgań. Odchylenie to może zostać zmierzane za pomocą zmiany pojemności kondensatorów płaskich, zawartych w strukturze żyroskopu (kondensatory te będą działały na tej samej zasadzie co kondensatory używane w akcelerometrach MEMS). Dzięki temu można wyznaczyć wartość prędkości obrotowej wokół osi X . Dla obrotu wokół osi Y zachowanie struktury będzie wyglądać analogicznie, z tą różnicą że wychyleniu ulegną masy M_2 i M_4 . Dla obrotu wokół osi Z , dla struktury pokazanej na rysunku masy M_2 i M_4 nadal będą drgać w dotychczasowej płaszczyźnie, jednak w wyniku działania siły Coriolisa zaczną drgać w przeciwnie strony, co zostało pokazane za pomocą żółtych i czerwonych strzałek.

4.5 Inercyjny zespół pomiarowy - IMU

Inercyjny zespół pomiarowy (ang. IMU - Inertial Measurement Unit) jest to zbiór czujników, umożliwiający określenie orientacji statku powietrznego w przestrzeni. Wykorzystuje on najczęściej połączone pomiary z akcelerometrów i żyroskopów, czasem również magnetometrów. Obecnie

rozwój technologii MEMS umożliwił tworzenie inercyjnych zespołów pomiarowych zawartych we wnętrzu jednego układu scalonego. Przykładem takiego zespołu może być układ MPU-6050, którego struktura przedstawiona została na rysunku 4.9.



Rys. 4.9: Struktura inercjalnego zespołu pomiarowego MPU-6050 [18]

Układ składa się z trzyosiowego akcelerometru, trzyosiowego żyroskopu oraz wszelkich niezbędnych układów elektronicznych, dzięki którym możliwe jest odczytywanie wartości przyspieszeń liniowych oraz prędkości kątowych. Układ MPU-6050 został wybrany jako przykład nie bez powodu - zawiera on blok o nazwie DMP (ang. Digital Motion Procesor), który jest zdolny do automatycznego określania orientacji czujnika w przestrzeni na podstawie pomiarów z żyroskopu i akcelerometru. Jest to unikatowa cecha, dzięki której kontroler lotu wykorzystujący ten czujnik jest zwolniony z konieczności samodzielnie określania położenia statku powietrznego w przestrzeni.

Rozdział 5

Projekt kwadrokoptera

5.1 Wymagania techniczne

Zaprojektowanie modelu kwadrokoptera nie należy do prostych zadań. Należy uwzględnić wiele czynników z zakresu mechaniki, elektroniki oraz informatyki mogących mieć wpływ na powodzenie projektu. Jako że autor pracy nie posiada żadnego doświadczenia związanego z konstrukcjami mechanicznymi oraz z budowaniem robotów, przy tworzeniu modelu kwadrokoptera największy nacisk położono na prostotę konstrukcji i zastosowanych rozwiązań. Docelowo starano się osiągnąć tanią uniwersalną konstrukcję, którą w przyszłości będzie można poszerzać o nowe moduły elektroniczne lub mechaniczne. Chcąc zdobyć jak największe doświadczenie w projektowaniu oraz programowaniu tego typu konstrukcji, w projekcie duży nacisk położono na własnoręczne zaprojektowanie układów elektronicznych oraz samodzielne opracowanie algorytmów sterujących.

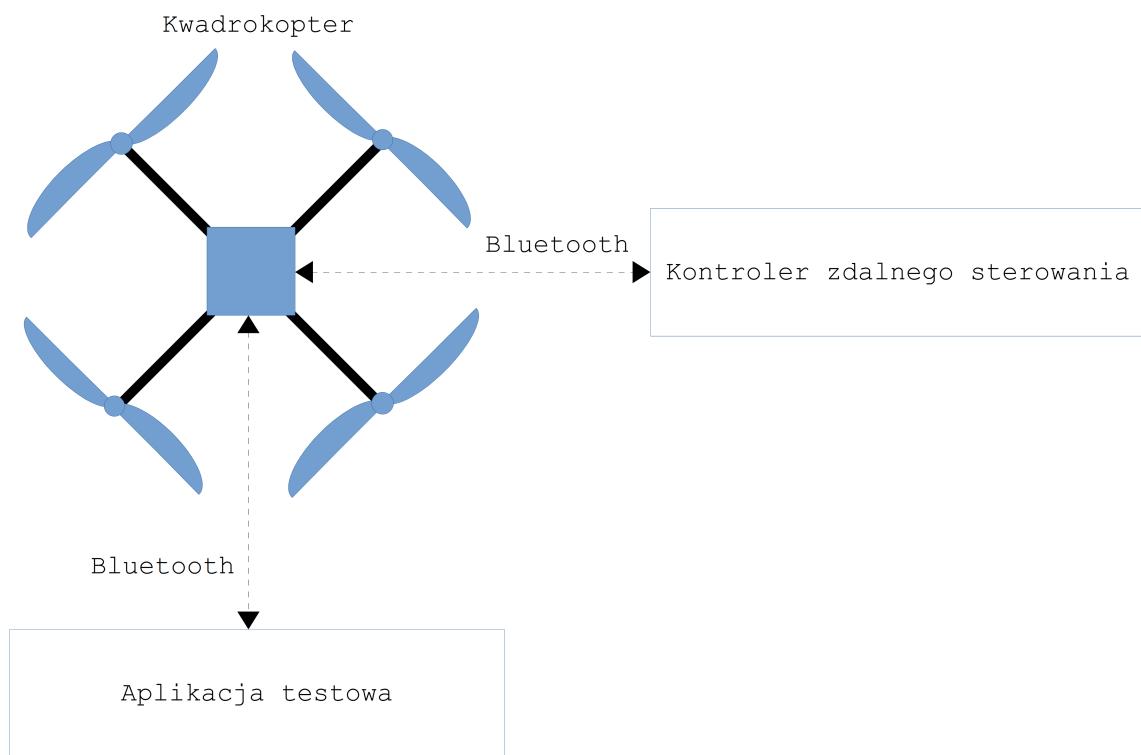
Podstawowa lista wymagań technicznych wygląda następująco:

- Niski koszt oraz uniwersalność konstrukcji.
- Ograniczona moc silników ze względów bezpieczeństwa.
- Czas lotu minimum 5 minut.
- Komunikacja radiowa o zasięgu minimum 5 metrów.
- Prosta, najlepiej gotowa konstrukcja mechaniczna.
- Prosta architektura elektroniki, oparta o mikrokontroler AVR.
- Oryginalny projekt PCB.
- Implementacja algorytmów sterowania w assemblerze lub w języku C.

5.2 Koncepcja rozwiązania

Zanim zaczęto analizować wymagania techniczne dotyczące samego kwadrokoptera, określono minimalną architekturę systemu, który musiał powstać, aby kwadrokopter mógł wznieść się w powietrzu. Zdecydowano się na rozwiązanie, w skład którego wchodzą:

- Kwadrokopter.
- Kontroler zdalnego sterowania.
- Aplikacja służąca do testów i strojenia parametrów lotu, przeznaczona na komputer PC.



RYS. 5.1: Schemat funkcjonalny projektowanego systemu

Rysunek 5.1 przedstawia ogólną architekturę zaprojektowanego systemu.

Kontroler zdalnego sterowania jest niezbędnym elementem systemu, za pomocą którego użytkownik wysyła sygnały sterujące kwadrokopterem drogą radiową. W obecnych czasach kontrolery lotu mogą stanowić bardzo skomplikowane systemy, wysyłające sygnały radiowe z użyciem wielu kanałów radiowych, co z reguły znajduje odzwierciedlenie w kosztach takiego urządzenia. W związku z tym, chcąc minimalizować koszty, przy projektowaniu systemu obrano inne podejście. Zdecydowano się na wykorzystanie telefonu komórkowego z systemem Android, wyposażonego w ekran

dotykowy oraz wbudowany moduł Bluetooth. Dzięki napisaniu aplikacji w języku Java, która działa na wspomnianej platformie sprzętowej, można osiągnąć praktycznie zerowy (jako że autor projektu posiadał telefon przed projektowaniem systemu) koszt tworzenia kontrolera zdalnego sterowania.

Podejście to ma wyraźne zalety w porównaniu do kupionego lub własnoręcznie opracowanego sprzętowego kontrolera zdalnego sterowania:

- **W porównaniu do kupionego kontrolera lotu uzyskuje się:**

- brak konieczności dopasowywania tworzonego systemu do gotowego standardu radiowej transmisji sygnałów sterujących, możliwość opracowania własnego protokołu służącego do komunikacji z kwadrokopterem transmitowanego w formie cyfrowej przez interfejs Bluetooth,
- dużo niższy koszt kontrolera.

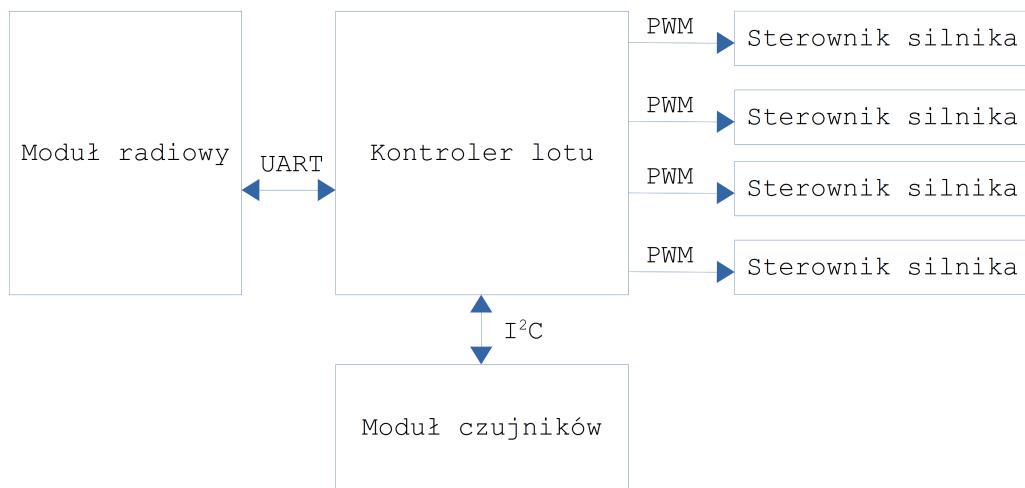
- **W porównaniu do własnoręcznie wykonanego sprzętowego kontrolera lotu uzyskuje się:**

- krótszy czas tworzenia kolejnych prototypów,
- niższy koszt kontrolera.

Zanim nastąpi moment uniesienia się kwadrokoptera w powietrze, należy najpierw nastroić regulatory zaimplementowane w algorytmie kontroli lotu. Podczas projektowania systemu zdecydowano się powierzyć tę rolę oddzielnej aplikacji przeznaczonej na komputer PC. Stało się tak między innymi za sprawą dużej liczby danych, które taka aplikacja potencjalnie musi wysyłać oraz przetwarzać. Chcąc mieć możliwość ustawiania licznych parametrów algorytmu, użytkownik powinien mieć możliwość ujrzenia tych parametrów jednocześnie, co dyskwalifikuje ekran telefonu do wykorzystania w takiej roli. Ponieważ w przypadku projektowania kontrolera zdalnego sterowania zdecydowano się na wykorzystanie interfejsu Bluetooth, mając na uwadze prostotę i niski koszt całego systemu, aplikacja testowa przeznaczona na komputer również wykorzystuje ten interfejs.

Kwadrokopter ma możliwość komunikacji z jednym z pozostałych elementów systemu w danym czasie, a zatem użytkownik może albo dokonywać kalibracji algorytmu kontroli lotu, albo wysyłać sygnałów sterujących lotem kwadrokoptera.

Po zaproponowaniu ogólnej architektury systemu, można przejść do projektowania systemu sterującego kwadrokoptera. Schemat blokowy systemu przedstawia rysunek 5.2.



RYS. 5.2: Schemat blokowy systemu sterującego kwadrokoptera

Przy projektowaniu ekipy kwadrokoptera główną ideą, jaka przyświecała autorowi projektu, była uniwersalność konstrukcji, tak aby można ją było w przyszłości rozbudowywać. Dlatego też zdecydowano się na modułową konstrukcję, składającą się z czterech głównych elementów:

- Kontroler lotu
- Moduł komunikacji radiowej
- Moduł czujników
- Kontrolery silników

5.2.1 Kontroler lotu

Kontroler lotu jest jednostką odpowiedzialną za utrzymanie kwadrokoptera w powietrzu. Jak widać na rysunku 5.2, komunikuje się on z modułem czujników za pomocą magistrali I²C oraz z modułem komunikacji radiowej za pomocą interfejsu UART. Ponadto musi on generować sygnały do sterowników silników zgodne z modelarskim standardem PWM opisany w 2 rozdziale. Wybór mikrokontrolera użytego w tym module padł na układ z rodziny AVR - ATmega328p. Poniżej przedstawiono podstawowe cechy użytego mikrokontrolera [42]:

- Maksymalna częstotliwość taktowania 20MHz.
- Sześć niezależnych kanałów PWM.
- Interfejs USART.

- Interfejs I²C.
- Napięcie zasilania 1.8V - 5.5V.
- 32kB pamięci programu.
- 2kB pamięci RAM.
- Dostępność w obudowie TQFP-32.

Jak zatem widać, układ ten spełnia wszystkie wymagania stawiane przed jednostką kontroli lotu, posiadając niezbędne interfejsy komunikacyjne, odpowiednią liczbę kanałów PWM, zdolnych do generowania sygnałów dla sterowników silników oraz pojemność pamięci programu i danych pozwalającą na implementację stosunkowo złożonych algorytmów. Niskie minimalne napięcie zasilania pozwala na zasilanie tego mikrokontrolera bezpośrednio z akumulatora litowo-polimerowego, dzięki czemu można uniknąć stosowania przetwornic podwyższających napięcie.

5.2.2 Moduł komunikacji radiowej

Moduł komunikacji radiowej jest odpowiedzialny za odbieranie sygnałów sterujących wysyłanych przez użytkownika urządzenia. W projekcie zdecydowano się na transmisję danych sterujących i danych używanych do strojenia algorytmu kwadrokoptera za pomocą jednego modułu, najprościej jest więc przesyłać dane w postaci cyfrowej zgodnie z protokołem opracowanym specjalnie do tego zadania. Wybór padł na moduł HC-05 ze względu na następujące parametry [43]:

- Napięcie zasilania 3.3V.
- Prąd pobierany podczas transmisji 8mA.
- Maksymalny zasięg 10m.
- Komunikacja poprzez interfejs UART.
- Niska cena.



Rys. 5.3: Moduł Bluetooth HC-05 [43]

Dzięki tym cechom możliwe jest zasilanie modułu bezpośrednio z akumulatora litowo-polimerowego zamontowanego w kwadrokopterze oraz przesyłanie danych sterujących i kalibracyjnych za pomocą prostego protokołu.

5.2.3 Moduł czujników

Moduł czujników odpowiedzialny jest za dostarczanie informacji o rzeczywistych parametrach fizycznych (położenie kątowe, prędkość kątowa) kwadrokoptera. Obecnie na rynku można spotkać rozmaite moduły czujników przeznaczone do konstrukcji latających, oferujące akcelerometry, żyroskopy, magnetometry i barometry. Wiele z tych konstrukcji ma jednak stosunkowo wysoką cenę, a ich popularność wśród użytkowników jest niewielka. Wybór padł więc na bardzo popularny a co za tym idzie tani moduł oparty o układ scalony MPU-6050, oferujący następujące parametry [18, 44]:

- 3-osiowy Akcelerometr i 3-osiowy żyroskop zintegrowane w jednej obudowie.
- Uniwersalny interfejs komunikacyjny I²C.
- Napięcie zasilania 2.4V - 3.5V.
- Prąd pobierany w trakcie pracy poniżej 5.5mA.
- Szeroki zakres konfiguracji skali pomiarowej czujników.
 - Akcelerometr $\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$
 - Żyroskop $\pm 250^\circ/s$, $\pm 500^\circ/s$, $\pm 1000^\circ/s$, $\pm 2000^\circ/s$
- Niewielkie wymiary fizyczne modułu.



Rys. 5.4: Moduł czujników MPU-6050

5.2.4 Kontrolery silników

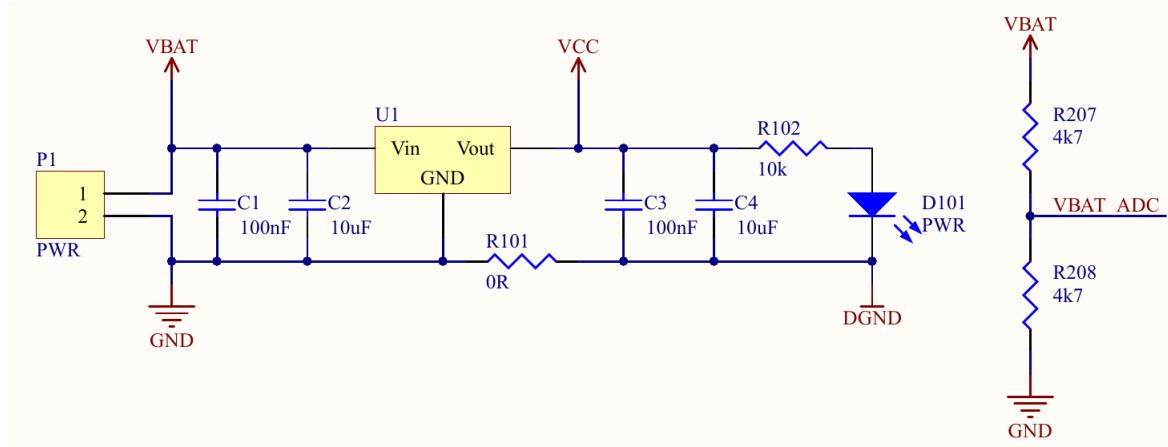
Kontrolery silników odpowiedzialne są za odbieranie sygnałów sterujących wysyłanych przez kontroler lotu i przetwarzanie ich na sygnał PWM, sterujący silnikiem. Przy opracowaniu koncepcji omawianych modułów, największy nacisk położono na prostotę i niewielkie wymiary konstrukcji. Procesor użyty w kontrolerze powinien posiadać minimalnie jedno wejście cyfrowe podłączone do kontrolera lotu, oraz jedno wyjście cyfrowe generujące sygnał PWM sterujący silnikiem. Wybór padł na mikrokontroler ATtiny85 ze względu na następujące cechy [45]:

- Dwa niezależne kanały PWM.
- Zewnętrzne źródła przerwania.
- Napięcie pracy 1.8V - 5.5V.
- Dostępność w obudowie SOIC-8.

Wybrany mikrokontroler spełnia wszystkie stawiane przed nim wymagania, oferując przy okazji ilość obszar znacznie przekraczający wymagania najprostszego algorytmu sterującego silnikami, co umożliwia potencjalny rozwój oprogramowania w przyszłości.

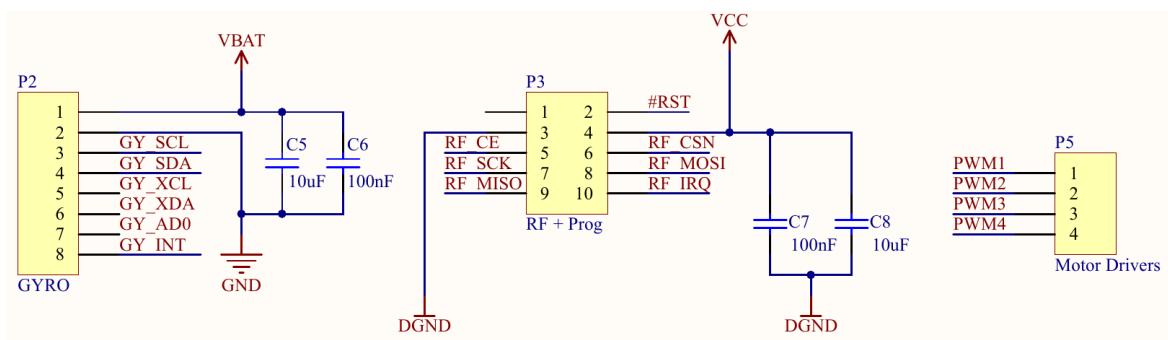
5.3 Opis układowy

5.3.1 Kontroler lotu



RYS. 5.5: Kontroler lotu - blok zasilania

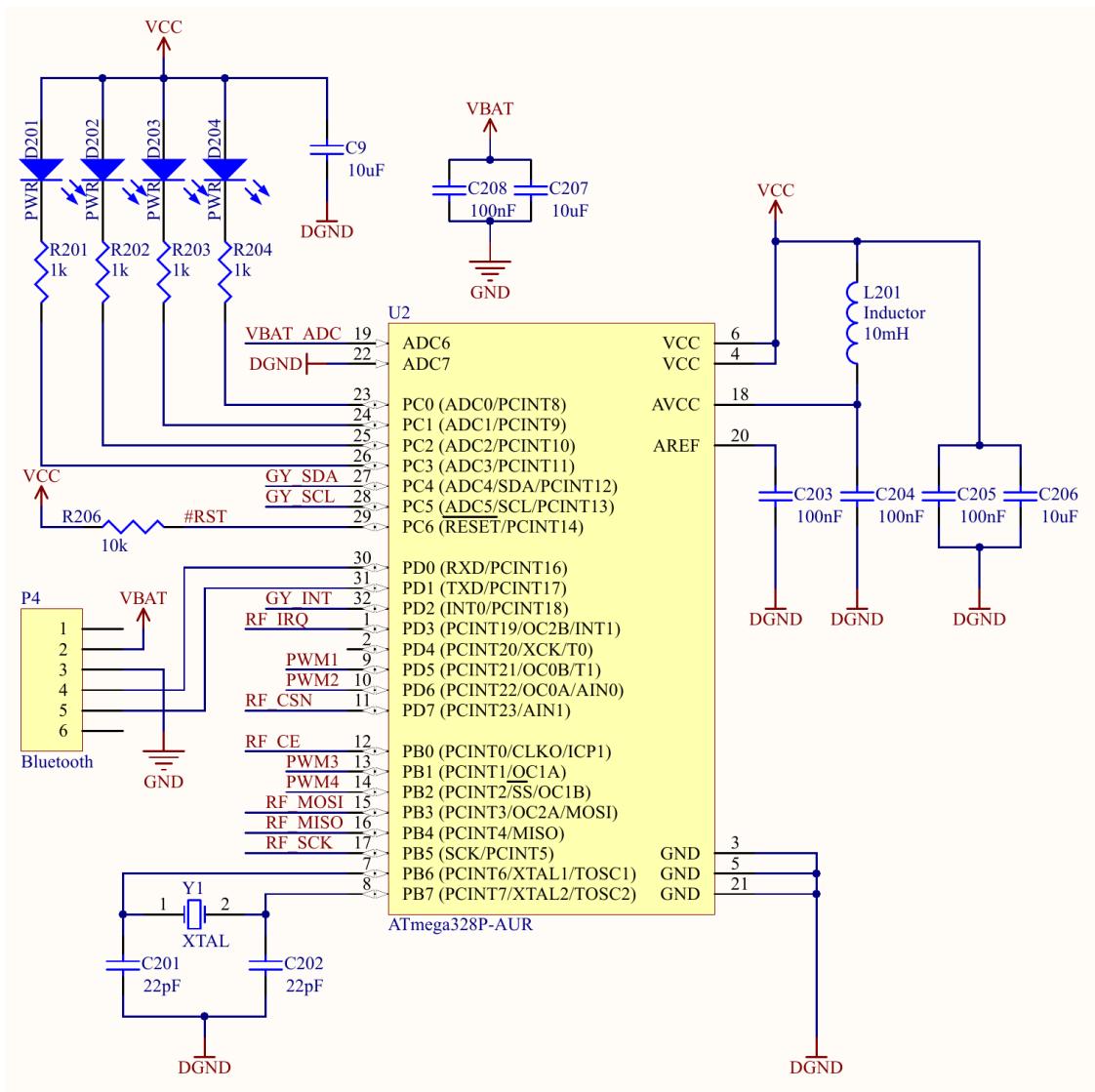
Rysunek 5.5 przedstawia blok zasilania kontrolera lotu kwadrokoptera. Składa się on ze stabilizatora LDO, o minimalnym napięciu wejściowym w okolicach 3.5V, dającego na wyjściu 3.3V. Tak niskie napięcie wejściowe niezbędne było w celu zasilania kontrolera bezpośrednio z akumulatora litowo-polimerowego. Poza stabilizatorem napięcia, blok zasilania składa się kondensatorów filtrujących napięcie, diody wskazującej obecność napięcia zasilania oraz dzielnika rezystancyjnego, wykorzystywanego przez mikrokontroler do pomiaru napięcia głównego akumulatora. Jako że ujemny biegum akumulatora będzie bezpośrednio podłączony do silników kwadrokoptera, mogą na nim występować różne zakłócenia związane z ich pracą. Dlatego też w obrębie płytki kontrolera lotu utworzono masę cyfrową, połączoną z masą analogową (ujemnym biegunem akumulatora) za pomocą rezystora R101.



RYS. 5.6: Kontroler lotu - złącza sygnałowe

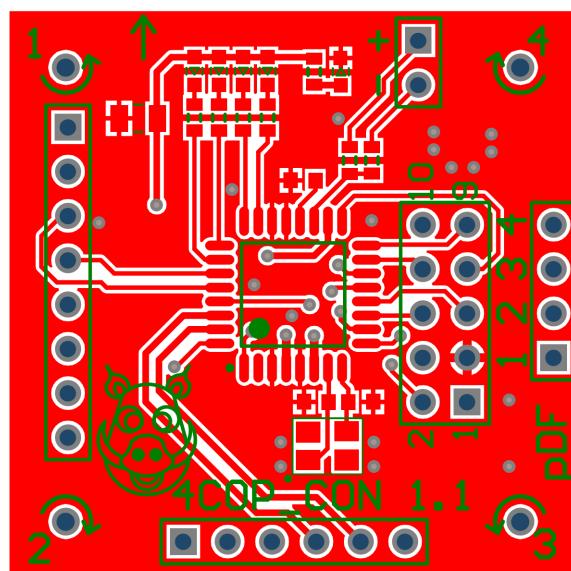
Rysunek 5.6 przedstawia złącza zamontowane na płytce kontrolera lotu.

- Złącze P2 przeznaczone jest dla modułu czujników MPU-6050. Ponieważ moduł ten posiada własny stabilizator napięcia zasilania do złącza doprowadzone jest bezpośrednio zasilanie z akumulatora wraz z niezbędnymi kondensatorami filtrującymi zasilanie. Z pozostałych wyprowadzeń modułu czujników, używane są jedynie linie interfejsu I²C oraz wyjście sygnału przerwania (GY_INT).
- Złącze P3 przeznaczone jest do programowania mikrokontrolera wykorzystanego w kontrolezie lotu oraz zostało przewidziane jako potencjalne złącze modułu radiowego, działającego w paśmie 2.4GHz i komunikującego się z mikrokontrolerem za pomocą interfejsu SPI.
- Złącze P5 służy do wyprowadzenia sygnałów PWM sterujących silnikami, skąd następnie trafiają do kontrolerów silników.

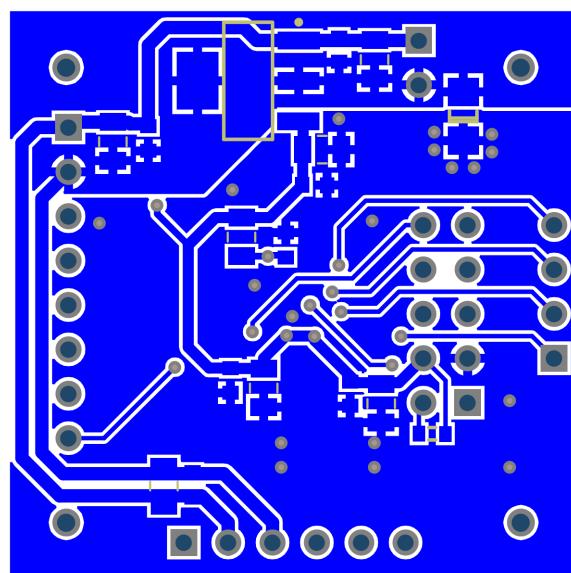


RYS. 5.7: Kontroler lotu - jednostka centralna

Rysunek 5.7 przedstawia główny schemat ideowy kontrolera lotu. Składa się on z mikrokontrolera ATmega328P taktowanego zewnętrznym rezonatorem kwarcowym o częstotliwości 16MHz. Do złącza P4 podłączono wyprowadzenia interfejsu UART, za pomocą którego realizowana jest komunikacja z modułem Bluetooth. Pozostałe wyprowadzenia procesora wykorzystane są jako interfejsy komunikacyjne (I^2C , SPI) a także jako wyjścia sygnałów trafiających do sterowników silników (PWM1 - PWM4). Piny PC0 - PC4 zostały użyte do sterowania diodami LED, służącymi jako wskaźnik działania programu. Wejście ADC6 zostało wykorzystane do pomiaru napięcia akumulatora litowo-polimerowego, tak aby kontroler lotu mógł wysłać do pilota ostrzeżenie o konieczności ładowania.



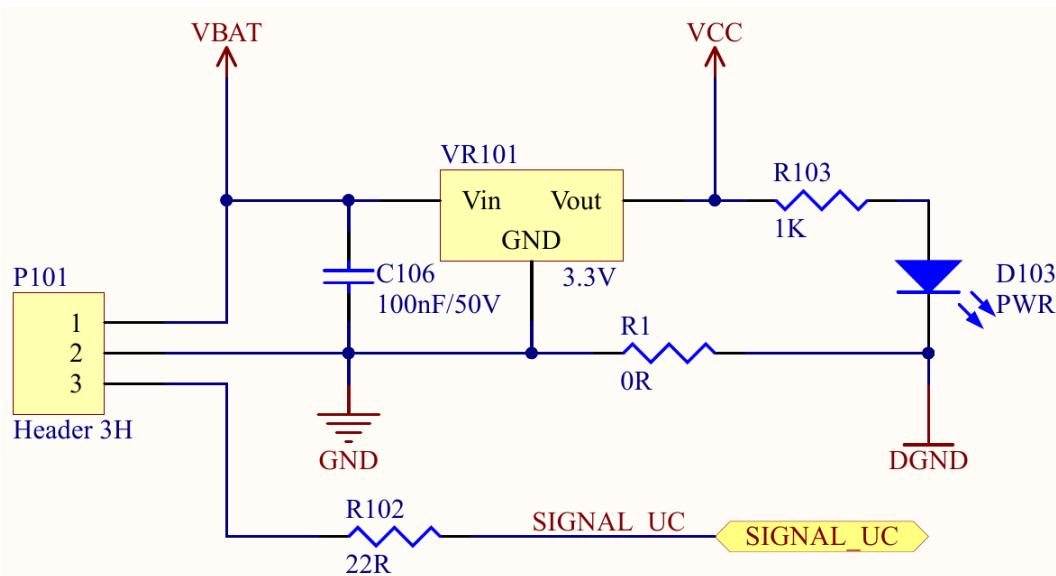
Rys. 5.8: Kontroler lotu - płytka PCB, warstwa górna



Rys. 5.9: Kontroler lotu - płytka PCB, warstwa dolna

Rysunki 5.8 oraz 5.9 przedstawiają górną i dolną warstwę płytka PCB kontrolera lotu. Na warstwie górnej zamontowany jest mokrokontroler wraz ze wszystkimi złączami, które są wykorzystywane przez zewnętrzne moduły. Jedynymi liniami, przy prowadzeniu których trzeba było zwrócić uwagę na ograniczenie długości, są linie od rezonatora kwarcowego, dlatego też został on umieszczony na tej samej warstwie co mikrokontroler, tuż pod nim, tak aby linie sygnału zegarowego miały minimalną długość oraz aby nie było konieczności prowadzenia ich przez przelotki. Pozostałe sygnały ze względu na niewielką maksymalną częstotliwość nie musiały spełniać reguł odnośnie wyrównania długości ścieżek lub dopasowania impedancji. Warstwa dolna płytka drukowanej została przeznaczona w głównej mierze na montaż elementów z bloku zasilania (stabilizator LDO, kondensatory filtrujące napięcie) oraz poprowadzenie linii zasilania. Na tej warstwie widać również podział masy zasilania na masę analogową oraz cyfrową połączone rezystorem R101. Masa analogowa została rozprowadzona w ten sposób, aby nie przechodziła pod żadnymi ważnymi liniami sygnałowymi (I^2C , SPI) oraz co ważniejsze pod liniami sygnału zegarowego pochodzącego od rezonatora kwarcowego. Rozmiar płytka drukowanej oraz rozmieszczenie otworów montażowych zostały dobrane tak, aby ułatwić montaż kontrolera lotu na ramie kwadrokoptera wybranej do tego projektu.

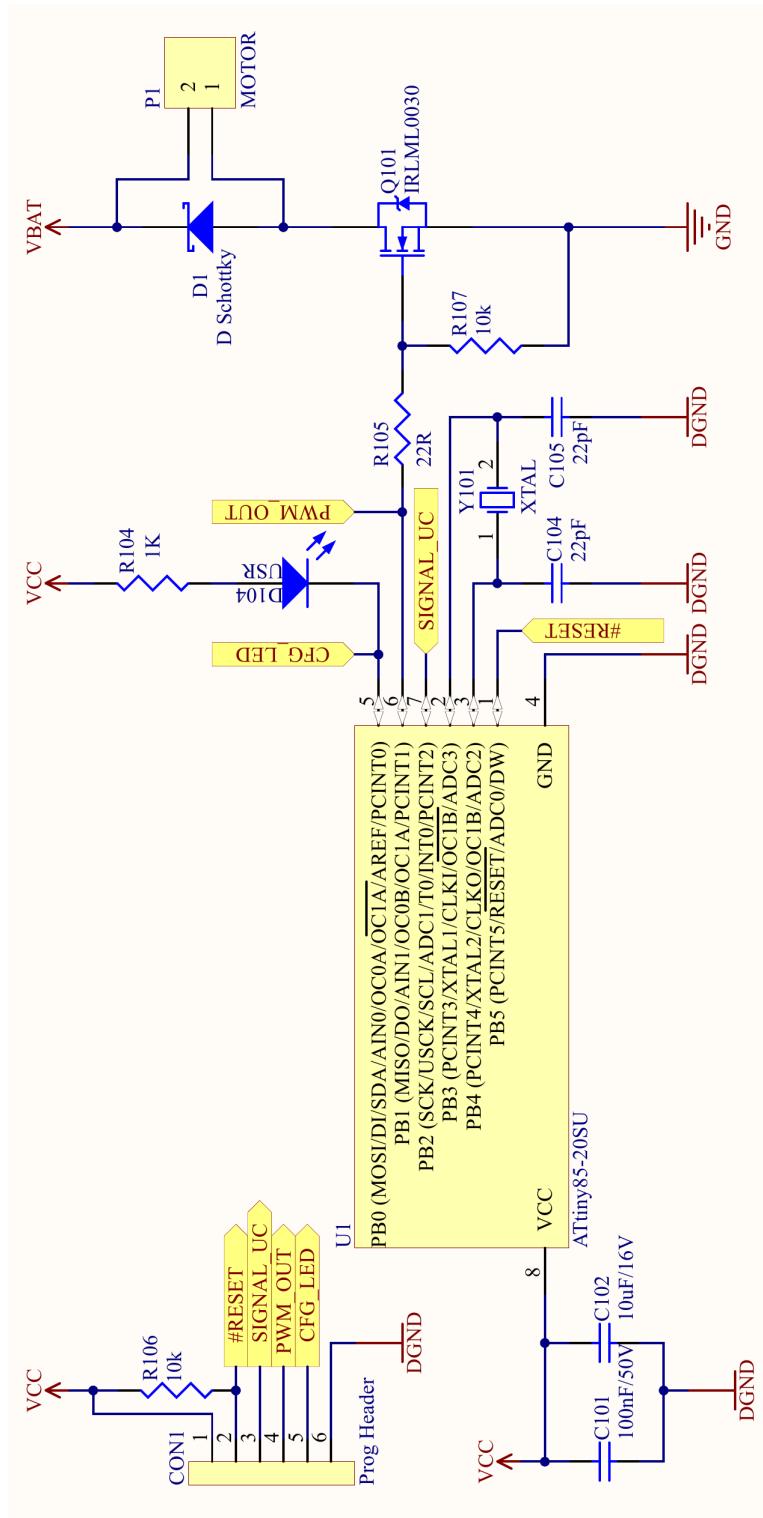
5.3.2 Sterownik silnika



Rys. 5.10: Sterownik silnika - blok zasilania

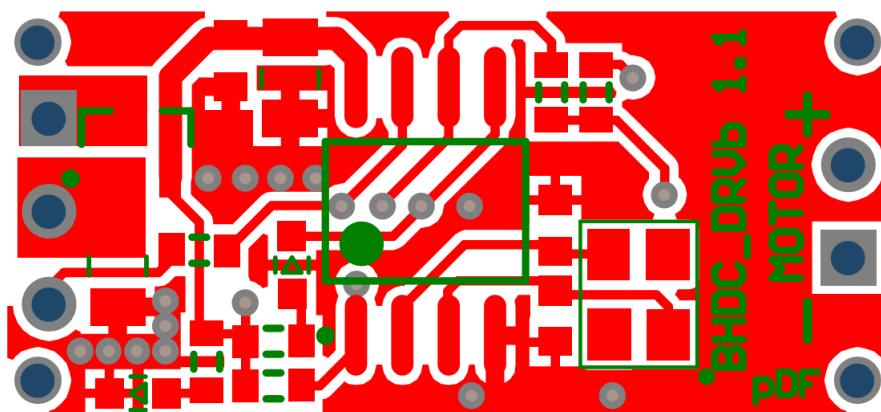
Rysunek 5.10 przedstawia blok zasilania sterownika silnika. Jest on bardzo podobny do zasilania kontrolera lotu i składa się ze stabilizatora LDO o minimalnym napięciu wejściowym 3.5V, rezystora R1 łączącego masę analogową i cyfrową układu, diody informującej o obecności napięcia zasilania oraz diody sygnalizacyjnej, używanej jako wskaźnik działania programu wgranego do

mikrokontrolera. Złącze P101 służy do podłączenia napięcia zasilania (piny 1 oraz 2) oraz do podłączenia sygnału sterującego SIGNAL_UC, informującego moduł o zadanej prędkości obrotowej silnika.

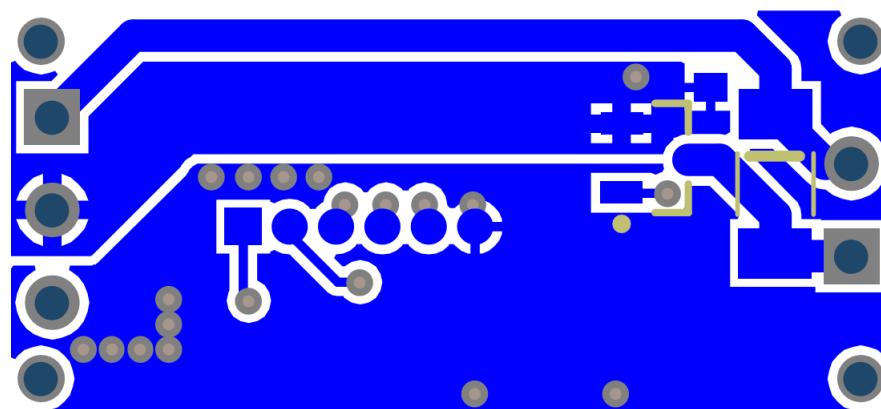


RYS. 5.11: Sterownik silnika - jednostka centralna

Rysunek 5.11 przedstawia główny schemat kontrolera silnika. Składa się on z mikrokontrolera ATtiny85 taktowanego zewnętrznym rezonatorem o częstotliwości 12MHz. Sygnał PWM pochodzący z kontrolera lotu przechodzi przez rezystor R102 tworzący z pojednorodnością wejścia mikrokontrolera filtr dolnoprzepustowy tłumiący część zakłóceń. Następnie trafia on na pin mikrokontrolera zdolny do generowania zewnętrznego przerwania INT0. Z mikrokontrolera wyprowadzony jest sygnał PWM trafiający na tranzystor Q101 sterujący silnikiem. Rezystor R105 zapobiega impulsom prądowym, które występują podczas ładowania i rozładowywania pasożytniczej pojednorodności bramki tranzystora, natomiast rezystor R107 wymusza niski stan na bramce tranzystora (co za tym idzie wyłączenie silnika) przy braku sygnału sterującego ze strony mikrokontrolera. Dioda Schottky'ego D1 ma za zadanie niwelowanie zakłóceń generowanych przez silnik szczotkowy podłączony do złącza P1. Sygnał CFG_LED podłączony jest do diody D104 i służy jako wskaźnik działania programu uruchomionego na mikrokontrolerze. Złącze CON1 służy do programowania mikrokontrolera i ze względu na oszczędność miejsca na płytce drukowanej zostało zrealizowane w formie padów testowych, do których lutuje się przewody z programatora. Ze względu na możliwe duże zakłócenia występujące w obrębie tego układu, przy mikrokontrolerze zastosowano duże kondensatory filtrujące napięcie zasilania.



Rys. 5.12: Sterownik silnika - płytka PCB, warstwa góra



Rys. 5.13: Sterownik silnika - płytka PCB warstwa dolna

Rysunek 5.12 oraz 5.13 przedstawiają górną i dolną warstwę płytka PCB sterownika silnika. Na górnej warstwie umieszczono zdecydomaną większość elementów oraz poprowadzono prawie wszystkie sygnały. Na warstwie dolnej umieszczono tranzystor sterujący silnikiem oraz diodę Schottky'ego niwelującą zakłuczenia. Przy projektowaniu tej płytka najważniejszą kwestią było prawidłowe poprowadzenie linii zasilania silnika oraz pola masy. Dlatego też na warstwie dolnej widać szeroką ścieżkę doprowadzającą zasilanie z akumulatora do złącza silnika oraz poligon masy analogowej, którym prąd płynący przez silnik będzie wracał do źródła zasilania. Kluczowym aspektem przy prowadzeniu tego poligona było zadbanie, aby nie przechodził on pod liniami sygnału zegarowego idącymi na warstwie górnej. Dlatego też na warstwie dolnej widać duże pole masy cyfrowej znajdującej się pod większością linii sygnałowych warstwy górnej. Koncentracja elementów na płytce jest stosunkowo duża i wynika z chęci zaprojektowania sterownika o minimalnych rozmiarach.

5.4 Konstrukcja mechaniczna

Ponieważ autor pracy nie ma doświadczenia w projektowaniu konstrukcji mechanicznych, zdecydowano się na użycie gotowej ramy do kwadrokoptera, opisanej w Internecie. W większości sklepów jedyne ramy dostępne do kupienia przewidziane są do współpracy z silnikami bezszczotkowymi dużej mocy, co kłoci się z wymaganiem technicznym mówiącym o małej mocy silników. W tej sytuacji najprostszym rozwiązaniem było kupienie gotowego kwadrokoptera i wykorzystanie jedynie mechaniki, do której zaprojektowano kontroler lotu i sterowniki silników.



RYS. 5.14: Model kwadrokoptera WL Toys V929



RYS. 5.15: Model kwadrokoptera WL Toys V929 - widok od spodu

Po zapoznaniu się z ofertą kwadrokopterów dostępnych w Internecie wybór padł na model WL Toys V929 przedstawiony na rysunku [5.14](#) oraz [5.15](#). Główne zalety modelu WL Toys V929 to:

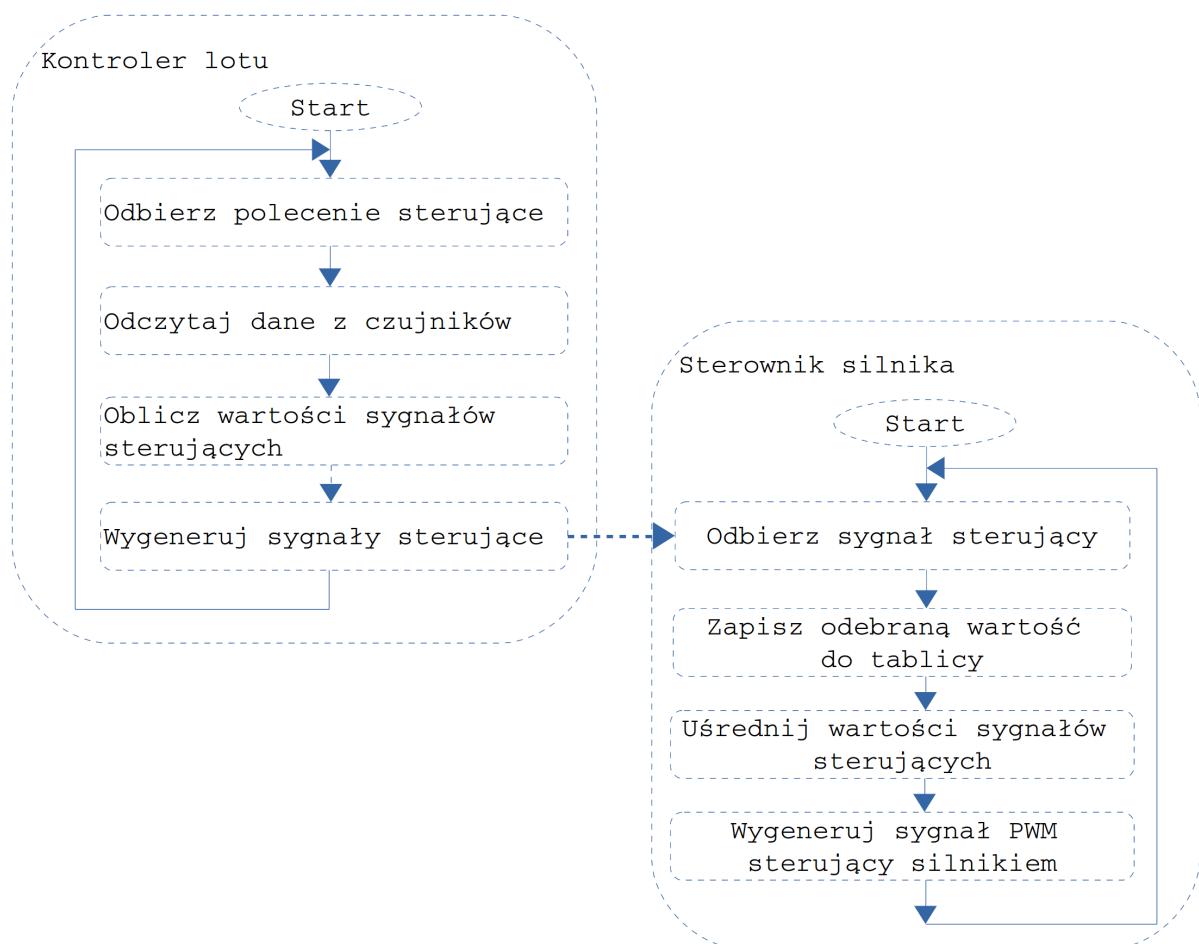
- Niska cena przy stosunkowo dużych rozmiarach.
- Bardzo duża popularność modelu w sklepach internetowych.
- Prosta i wytrzymała konstrukcja mechaniczna ze wspornikami silników wykonanymi z włókien węglowych.
- Duża dostępność części zamiennych.

Rysunek [5.15](#) przedstawia model od spodu. Widać na nim prostotę konstrukcji nośnej kwadrokoptera - cała rama wykonana jest w postaci centralnie umieszczonego pierścienia, do którego przymocowane są cztery wsporniki śmigieł. Na końcach wsporników znajdują się koszyki zawierające silniki szczotkowe stosunkowo małej mocy, wraz z zębatkami oraz śmigłami. Taka konstrukcja mechaniczna pozwala na proste dodawanie własnoręcznie wykonanych pozdespołów elektronicznych (na przykład sterowników silników) i tworzenie gotowej platformy zdolnej do lotu.

Rozdział 6

Oprogramowanie kwadrokoptera

6.1 Struktura oprogramowania



Rys. 6.1: Struktura oprogramowania kontrolera lotu oraz sterownika silnika

Rysunek 6.1 przedstawia strukturę oprogramowania kontrolera lotu oraz sterownika silnika (dla uproszczenia na schemacie przedstawiony jest tylko jeden sterownik silnika). Pogrubioną strzałką łączącą bloki kontrolera lotu oraz sterownika silnika zaznaczono przesyłanie sygnałów sterujących generowanych przez kontroler lotu w przyjętym formacie modelarskim PWM opisany w rozdziale 2.

6.2 Kontroler lotu

6.2.1 Funkcje programowe

Oprogramowanie kontrolera lotu realizuje cztery podstawowe zadania:

- Odbiór poleceń
- Odczyt pomiarów z czujników
- Generowanie sygnałów sterujących
- Wysyłanie sygnałów sterujących do kontrolerów silników

Pierwsza wersja oprogramowania powstała w języku assembler avr, jednakże zrezygnowano z tej koncepcji ze względu na dużą liczbę gotowych bibliotek służących do obsługi czujników pomiarowych, napisanych w języku C oraz ze względu na istotne trudności powstające przy programowaniu w assemblerze operacji arytmetycznych na liczbach 16-bitowych oraz 32-bitowych. Operacje te są wykonywane przez algorytm stabilizacji lotu.

Druga, finalna wersja oprogramowania została napisana w języku C, z wykorzystaniem gotowych bibliotek służących do obsługi czujnika MPU-6050 oraz bibliotek implementujących regulator PID, w których autor pracy wprowadził poprawki adaptujące je do projektu kwadrokoptera.

6.2.2 Komunikacja radiowa

Komunikacja radiowa realizowana jest za pomocą modułu Bluetooth HC-06 (lub kompatybilnego HC-05) udostępniającego interfejs UART do nadawania i odbierania danych z poziomu mikrokontrolera sterującego kwadrokopterem. Chcąc zrealizować płynną transmisję danych między kwadrokopterem a użytkownikiem obrano następujące założenia:

- Kwadrokopter jest widziany z punktu użytkownika jako zbiór 8-bitowych rejestrów.

- Wartości rejestrów są kwadrokoptera przechowywane w pamięci RAM jako tablica zmiennych 8-bitowych.
- Protokół binarny służący do wymiany danych między użytkownikiem a kwadrokopterem ma poniższe cechy:
 - minimum dwie komendy: zapis rejestrów i odczyt rejestrów,
 - użytkownik zawsze inicjuje transmisję.

Idea zastosowania protokołu binarnego znacząco upraszcza implementację oprogramowania po stronie mikrokontrolera, skracając tym samym czas potrzebny na przetwarzanie danych. Struktura wiadomości wygląda następująco:

<Kod komendy><Długość wiadomości w bajtach><Pole danych wiadomości>

Pola "Kod komendy" oraz "Długość wiadomości" mają rozmiar jednego bajtu, pole danych wiadomości ma długość o dwa bajty mniejszą niż wartość wpisana w polu długości wiadomości. Pole danych wiadomości może zawierać różną treść w zależności od kodu komendy.

- Komenda zapisu rejestrów - pole danych wiadomości ma długość co najmniej trzech bajtów
 - Adres pierwszego rejestru do zapisu
 - Liczba rejestrów do zapisu
 - Pole zawierające wartości kolejnych zapisywanych rejestrów
- Komenda odczytu rejestrów - pole danych wiadomości ma długość dwóch bajtów
 - Adres pierwszego rejestru do odczytu
 - Liczba rejestrów do odczytu

Poniższy listing przedstawia strukturę zdefiniowaną w programie, która jest używana do przechowywania danych odebranych przez kwadrokopter.

```
#define FRAME_DATA_BUFFER_SIZE 35
typedef struct
{
    union
    {
        struct
        {
            volatile uint8_t command;
            volatile uint8_t data_count;
            uint8_t data[FRAME_DATA_BUFFER_SIZE];
        } frame;
        uint8_t raw_data[2+FRAME_DATA_BUFFER_SIZE];
    };
}
```

```
//Frame control signals
volatile uint8_t data_ready;
volatile uint8_t data_index;
}QuadrocopterData;
```

Składa się ona z tablicy "raw_data" przechowującej dane bezpośrednio odebrane przez interfejs UART. Następnie za pomocą unii, dane z tablicy "raw_data" rzutowane są na kod komendy, długość wiadomości oraz pole danych wiadomości [46].

Dodatkowo w skład struktury wchodzą dwie flagi:

- data_ready - flaga ustawiana w przerwaniu odbioru danych przez interfejs UART, informująca program główny o odebraniu całej wiadomości.
- data_index - index wykorzystywany przez procedurę odbierającą dane przez interfejs UART do zapisywania danych w tablicy "raw_data".

Poniższy listing przedstawia procedurę inicjalizacji interfejsu UART. Składa się ona z wyzerowania pól struktury "quadrocopter_rx_data" przechowującej dane odebrane przez kwadroopter oraz z inicjalizacji sprzętowego modułu UART, w który wyposażony jest mikrokontroler ATmega328p, poprzez ustawienie rejestrów:

- UBRR0H, UBRR0L, UCSR0A - rejesty odpowiedzialne za konfigurację szybkości transmisji
- UCSR0C - rejestr odpowiedzialny za ustawianie formatu ramki interfejsu UART
- UCSR0B - rejestr odpowiedzialny za włączanie nadajnika oraz osobiornika sprzętowego modułu UART oraz włączanie przerwania odbioru danych

```
void UART_init(void)
{
    uint8_t index;
    for(index = 0; index < 18; index++)
    {
        quadrocopter_rx_data.raw_data[index] = index;
    }
    quadrocopter_rx_data.data_ready = 0;
    quadrocopter_rx_data.data_index = 0;
    quadrocopter_rx_data.frame.command = 0;
    quadrocopter_rx_data.frame.data_count = 1;

    //Set baud rate - 57600 baud rate + double speed mode
    UBRROH = (uint8_t)(34>>8);
    UBRROL = (uint8_t)(34);
    UCSR0A = (1<<U2X0);
    //Set frame format: 8data, No parity, 1stop bit
```

```
UCSROC = (3<<UCSZ00);
//Enable receiver and transmitter
UCSROB = ((1<<RXCIE0)|(1<<RXENO)|(1<<TXENO));

}
```

Przedstawiony poniżej listing przedstawia deklaracje rejestrów wewnętrznych kwadrokoptera, widocznych z poziomu użytkownika.

```
#define REG_COUNT 35
uint8_t reg_array[REG_COUNT];// = {0};
typedef void (*handler)(void);
handler reg_update_handler_array[REG_COUNT];// = {0};

#define REG_STATUS 0
#define REG_COMMAND 1

#define REG_THRUST 2
#define REG_PITCH 3
#define REG_ROLL 4
#define REG_YAW 5

#define REG_PID_X_PH 6//-
#define REG_PID_X_PL 7//-
#define REG_PID_X_IH 8//-- PID X
#define REG_PID_X_IL 9//-- PID X
#define REG_PID_X_DH 10//-
#define REG_PID_X_DL 11//-

#define REG_PID_Y_PH 12//-
#define REG_PID_Y_PL 13//-
#define REG_PID_Y_IH 14//-- PID Y
#define REG_PID_Y_IL 15//-- PID Y
#define REG_PID_Y_DH 16//-
#define REG_PID_Y_DL 17//-

#define REG_PID_Z_PH 18//-
#define REG_PID_Z_PL 19//-
#define REG_PID_Z_IH 20//-- PID Z
#define REG_PID_Z_IL 21//-- PID Z
#define REG_PID_Z_DH 22//-
#define REG_PID_Z_DL 23//-

#define REG_LED 24

#define REG_FL_PWM 25
#define REG_FR_PWM 26
#define REG_BL_PWM 27
#define REG_BR_PWM 28

#define REG_FR_BL_ENABLE 29
#define REG_FL_BR_ENABLE 30

#define REG_GX_OFFSET 31
```

```
#define REG_GY_OFFSET          32
#define REG_GZ_OFFSET          33
#define REG_LOG1_ENABLE         34
```

Jak widać jest to tablica 8-bitowych zmiennych "reg_array" o rozmiarze zdefiniowanym za pomocą makra "REG_COUNT". Następnie zdefiniowane są nazwy kolejnych rejestrów wraz z przypisanymi do nich adresami. Funkcje poszczególnych rejestrów omówiono poniżej:

- Adresy 0-1: Nieużywane rejesty, przewidziane na potrzeby rozszerzania funkcji protokołu w przyszłości.
- Adresy 2-5: Rejestry używane w trakcie lotu, przechowujące zadane wartości ciągu silników oraz prędkości kątowych wokół trzech osi.
- Adresy 6-11: Rejestry przechowujące 16-bitowe wartości współczynników P,I,D kontrolera PID, odpowiedzialnego za stabilizację prędkości kątowej wokół osi X.
- Adresy 12-17: Rejestry przechowujące 16-bitowe wartości współczynników P,I,D kontrolera PID, odpowiedzialnego za stabilizację prędkości kątowej wokół osi Y.
- Adresy 18-23: Rejestry przechowujące 16-bitowe wartości współczynników P,I,D kontrolera PID, odpowiedzialnego za stabilizację prędkości kątowej wokół osi Z.
- Adres 24: Rejestr używany do testów, włączający debugowe diody LED na płytce kontrolera lotu.
- Adresy 25-28: Rejestry używane do testów, służące do wpisania bezpośredniej wartości sygnałów PWM wysyłanych do odpowiedniego kontrolera silnika.
- Adresy 29-30: Rejestry używane do testów, służące do włączania testowania regulatora PID dla osi X lub Y.
- Adresy 31-33: Rejestry używane do programowego wprowadzania offsetu dla wartości odczytywanych z żyroskopu.
- Adres 34: Rejestr używany do testów, służący do włączania wysyłania komunikatów debugowych z programu.

Oprócz tablicy przechowującej wartości rejestrów wewnętrznych kwadrokoptera, w kodzie programu zdefiniowana jest również tablica wskaźników na funkcje, które są wołane przy modyfikacji wartości odpowiadającego im rejestru wewnętrznego kwadrokoptera. Jest to bardzo proste i pożyteczne narzędzie, dające możliwość automatycznego wywoływania odpowiednich funkcji przy modyfikacji wybranych rejestrów.

Poniższy listing przedstawia funkcję służącą do zapisu rejestrów kwadrokoptera. Jako parametry wejściowe przyjmuje ona rozmiar całej odebranej ramki danych wyrażony w bajtach, oraz wskaźnik na pole danych wiadomości.

```
#define INVALID_DATA_COUNT -2
#define INVALID_REG_ADDRESS -3
int8_t reg_write_command(uint8_t data_count, uint8_t *data)
{
    uint8_t reg_address;
    uint8_t reg_count;
    uint8_t *reg_values;
    reg_address = data[0];
    reg_count = data_count - 1;
    reg_values = data+1;

    if(0 == reg_count)
    {
        return INVALID_DATA_COUNT;
    }
    if(REG_COUNT <= reg_address)
    {
        return INVALID_REG_ADDRESS;
    }
    uint8_t index;
    uint8_t ret;
    ret = 0;
    for(index = 0; ((index < reg_count) && ((reg_address + index) < REG_COUNT)); index++)
    {
        uint8_t temp;
        temp = reg_address + index;
        UART_print_s("[Addr: ");
        UART_print_d(&temp,UINT_8);
        temp = reg_values[index];
        UART_print_s(" Value: ");
        UART_print_d(&temp,UINT_8);
        UART_print_s("]");
        UART_println();
        reg_array[reg_address+index] = reg_values[index];
        if(0 != reg_update_handler_array[reg_address+index])
        {

            UART_print_s("Running register update handler");
            UART_println();
            reg_update_handler_array[reg_address+index]();
        }
        ret++;
    }
    return ret;
}
```

Zapis rejestrów odbywa się w dwóch krokach:

Sprawdzanie prawidłowości odebranych danych: funkcja sprawdza czy pola odpowiadające liczbie rejestrów do zapisu oraz adresu pierwszego rejestru do zapisu mają prawidłowe wartości.

Zapis rejestrów: funkcja przechodzi po kolejnych indeksach tablicy przechowującej wartości rejestrów kwadrokoptera, i zastępuje je nowymi wartościami pobieranymi z pola danych wiadomości. Jednocześnie przy każdym zapisie rejestrów funkcja sprawdza, czy w tablicy reg_update_handler_array znajduje się wskaźnik na funkcję, która ma zostać wywołana podczas zmiany jego wartości. Jeśli pole w tablicy nie ma wartości 0, następuje wywołanie procedury przypisanej do modyfikowanego rejestrów.

Działanie funkcji odczytującej wartości rejestrów kwadrokoptera jest analogiczne, dlatego też zbędne jest omawianie kodu tej funkcji.

Wykorzystanie omówionych funkcji oraz struktur, z pominięciem kodu realizującego pozostałe zadania kontrolera lotu takie jak odczyt danych z czujników oraz generowanie i wysyłanie sygnałów sterujących, wygląda następująco:

```
int main(void)
{
    UART_init();
    reg_array_init();
    handlers_init();

    while(1)
    {
        if(quadrocopter_rx_data.data_ready)
        {
            switch((char)(quadrocopter_rx_data.frame.command))
            {
                case 'w':
                    reg_write_command(quadrocopter_rx_data.frame.data_count, quadrocopter_rx_data.frame.data);
                    break;

                case 'r':
                    reg_read_command(quadrocopter_rx_data.frame.data_count, quadrocopter_rx_data.frame.data);
                    break;

                default:
                    UART_print_s("Unknown command");
                    UART.println();
                    break;
            }
            QuadrocopterRxDataClear();
        }
    }
}
```

Funkcje "reg_array_init" oraz "handlers_init" służą odpowiednio do zerowania początkowych wartości rejestrów kwadrokoptera oraz do odpowiedniego przypisania wskaźników funkcji wołanych podczas zmiany wartości rejestrów kwadrokoptera. Funkcja "QuadrocopterRxDataClear" służy do zerowania odpowiednich pól struktury przechowującej odebrane dane, pozwalając tym samym na prawidłowy odbiór następnej wiadomości.

6.2.3 Obsługa czujników

Pomiar prędkości kątowych kwadrokoptera realizowany jest za pomocą czujnika MPU-6050 komunikującego się z mikrokontrolerem za pomocą magistrali I²C. Do jego obsługi w dużej mierze została wykorzystana gotowa biblioteka napisana na platformę Arduino, która jednak w oryginalnej wersji przygotowana została w języku C++, dlatego też zostały do niej wprowadzone odpowiednie poprawki, mające na celu przejście z programowania obiektowego wykorzystującego klasy na programowanie z wykorzystaniem struktur.

Aby możliwa była komunikacja z czujnikiem, należy najpierw dokonać inicjalizacji sprzętowego modułu TWI, w który wyposażony jest mikokontroler ATmega328p, a następnie należy prawidłowo zainicjalizować sam układ MPU-6050 tak, aby możliwy był odczyt danych z żyroskopu.

W projekcie zachowano oryginalną strukturę biblioteki do obsługi modułu MPU-6050, która wygląda następująco:

- I2C_dev - najwyższa warstwa abstrakcji, dająca użytkownikowi dostęp do urządzeń podłączonych do magistrali I²C.
- TwoWire - middleware biblioteki, realizujący odbiór i wysyłanie danych oraz zarządzanie transmisją danych.
- TWI - najniższa warstwa biblioteki, zapewniająca podstawową warstwę abstrakcji sprzętowej.

Poniższy listing przedstawia funkcję inicjalizującą urządzenie podłączone do magistrali I²C. Jak widać jest to prosta funkcja, która wywołuje jedynie funkcję inicjalizującą umieszczoną w niższej warstwie biblioteki.

```
void I2C_dev_init(void)
{
    TwoWire_begin();
}
```

Poniższy listing przedstawia funkcję inicjalizującą middleware biblioteki obsługującej interfejs TWI. Inicjalizuje ona indeksy służące do przesuwania się po buforach nadawczym oraz odbiorczym, wykorzystywanych w trakcie transmisji danych oraz zeruje zmienne informujące o liczbie bajtów znajdujących się w każdym z tych buforów. Na sam koniec funkcja ta odwołuje się do niższej warstwy biblioteki za pomocą funkcji "twi_init()"

```
uint8_t TwoWire_rxBuffer[BUFFER_LENGTH];
volatile uint8_t TwoWire_rxBufferIndex = 0;
volatile uint8_t TwoWire_rxBufferLength = 0;

volatile uint8_t TwoWire_txAddress = 0;
uint8_t TwoWire_txBuffer[BUFFER_LENGTH];
```

```
volatile uint8_t TwoWire_txBufferIndex = 0;
volatile uint8_t TwoWire_txBufferLength = 0;

uint8_t TwoWire_transmitting = 0;
void (*TwoWire_user_onRequest)(void);
void (*TwoWire_user_onReceive)(int);

void TwoWire_begin(void)
{
    TwoWire_rxBufferIndex = 0;
    TwoWire_rxBufferLength = 0;

    TwoWire_txBufferIndex = 0;
    TwoWire_txBufferLength = 0;

    twi_init();
}
```

Poniższy listing przedstawia funkcję inicjalizacyjną zlokalizowaną w najniższej warstwie biblioteki. Jak widać polega ona głównie na włączaniu wewnętrznych rezystorów podciągających na liniach danych oraz zegarowych interfejsu TWI, ustawianiu odpowiedniej szybkości transmisji danych oraz włączeniu modułu TWI wraz z przerwaniami odbioru danych.

```
void twi_init(void)
{
    // initialize state
    twi_state = TWI_READY;

#if defined(__AVR_ATmega168__)
    // activate internal pull-ups for twi
    // as per note from atmega8 manual pg167
    sbi(PORTC, 4);
    sbi(PORTC, 5);
#else
    // activate internal pull-ups for twi
    // as per note from atmega128 manual pg204
    sbi(PORTD, 0);
    sbi(PORTD, 1);
#endif

    // initialize twi prescaler and bit rate
    cbi(TWSR, TWPS0);
    cbi(TWSR, TWPS1);
    TWBR = ((CPU_FREQ / TWI_FREQ) - 16) / 2;

    /* twi bit rate formula from atmega128 manual pg 204
    SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
    note: TWBR should be 10 or higher for master mode
    It is 72 for a~16mhz Wiring board with 100kHz TWI */

    // enable twi module, acks, and twi interrupt
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);
}
```

Po zainicjalizowaniu interfejsu TWI wraz ze wszystkimi warstwami abstrakcji oferowanymi przez bibliotekę można przystąpić do inicjalizacji samego układu MPU-6050. Dokonuje się jej za pomocą funkcji `MPU6050_initialize`, której kod został przedstawiony na poniższym listingu.

```
void MPU6050_initialize(void)
{
    devAddr = MPU6050_DEFAULT_ADDRESS;
    MPU6050_setClockSource(MPU6050_CLOCK_PLL_XGYRO);
    MPU6050_setFullScaleGyroRange(MPU6050_GYRO_FS_250);
    MPU6050_setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
    MPU6050_setSleepEnabled(0); // thanks to Jack Elston for pointing this one out!
}
```

Zadaniem powyższej funkcji jest ustawienie odpowiedniego źródła sygnału taktującego układ MPU6050, zainicjalizowanie akcelerometru oraz żyroskopu, a na sam koniec wybudzenie czujnika ze stanu uśpienia, w którym znajduje się domyślnie po podaniu napięcia zasilania.

```
void MPU6050_getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy, int16_t* gz)
{
    I2Cdev_readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14, buffer);
    *ax = (((int16_t)buffer[0]) << 8) | buffer[1];
    *ay = (((int16_t)buffer[2]) << 8) | buffer[3];
    *az = (((int16_t)buffer[4]) << 8) | buffer[5];
    *gx = (((int16_t)buffer[8]) << 8) | buffer[9];
    *gy = (((int16_t)buffer[10]) << 8) | buffer[11];
    *gz = (((int16_t)buffer[12]) << 8) | buffer[13];
}
```

Po poprawnym zainicjalizowaniu czujnika MPU-6050 można przejść do odczytu wartości wyjściowych z akcelerometru i żyroskopu. Dokonuje się tego za pomocą funkcji `MPU6050_getMotion6` przedstawionej na powyższym listingu. Funkcja jako parametry przyjmuje sześć 16-bitowych wskaźników na zmienne zadeklarowane w obszarze kodu z którego jest ona wołana, wpisując do nich odpowiednio wartości przyspieszeń liniowych w osiach (zmienne `ax`, `ay`, `az`) oraz prędkości kątowych (zmienne `gx`, `gy`, `gz`). Do odczytu wartości pomiarów z czujników wykorzystana jest funkcja `I2Cdev_readBytes`, która jako parametry przyjmuje adres urządzenia na magistrali I²C, adres pierwszego rejestru wewnętrznego czujnika MPU-6050 do odczytu, liczbę rejestrów do odczytu, oraz wskaźnik na roboczy bufor, do którego zostaną skopiowane dane. W wywołaniu tej funkcji ostatni argument, którym jest "buffer", jest globalną tablicą zdefiniowaną w obrębie najwyższej warstwy biblioteki obsługującej czujnik MPU-6050.

Przykład kodu programu, służącego do odczytywania danych pomiarowych z czujnika MPU-6050 wygląda następująco:

```
int main(void)
{
    int16_t ax;
    int16_t ay;
    int16_t az;
```

```
int16_t gx;
int16_t gy;
int16_t gz;

timer2_init();
sei();

MPU6050_initialize();

MPU6050_setXGyroOffset(100);
MPU6050_setYGyroOffset(30);
MPU6050_setZGyroOffset(50);

while(1)
{
    if(timer2_tick)
    {
        timer2_tick = 0;
        MPU6050_getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    }
}
```

Jak widać wewnątrz głównej funkcji programu tworzone są zmienne przechowujące wartości odczytów z akcelerometru i żyroskopu czujnika MPU-6050. Timer2 mikrokontrolera ATmega328p jest zainicjalizowany za pomocą funkcji "timer2_init()" w taki sposób, że generuje przerwanie z częstotliwością około 100Hz. W procedurze obsługi tego przerwania ustawiana jest flaga timer2_tick, wykorzystywana przez program główny jako zezwolenie na odświeżenie danych z czujników. Funkcje MPU6050_setXGyroOffset, MPU6050_setYGyroOffset, MPU6050_setZGyroOffset, służą do kalibracji żyroskopu.

6.2.4 Algorytm stabilizacji lotu

Algorytm stabilizacji i kontroli lotu przyjmuje dane wyjściowe z żyroskopu informujące o rzeczywistych prędkościach kątowych kwadrokoptera wokół wszystkich osi układu współrzędnych i na podstawie ich wartości oraz wartości zadanych prędkości kątowych oraz ciągu silników oblicza wartości sygnałów sterujących wysyłanych do sterowników silników. Podczas przetwarzania tych danych pojawia się jednak problem różnicy skali wartości dla sygnałów z żyroskopu oraz sygnałów sterujących odbieranych drogą radiową. Jak to pokazano w podrozdziale 6.2.3, żyroskop wewnątrz czujnika MPU-6050 konfigurowany jest tak, aby maksymalna mierzona prędkość kątowa wynosiła 250 stopni na sekundę. Jako że wartości wyjściowe z żyroskopu są 16-bitową liczbą ze znakiem wartość -32768 będzie oznaczała prędkość kątową -250 stopni na sekundę a wartość 32767 oznaczać będzie 250 stopni na sekundę. Zadane wartości prędkości kątowych przesyłane są jako liczby 8-bitowe ze znakiem, dlatego też należy zrzutować je na przedział adekwatny do wartości

odczytywanych z żyroskopu. Gdyby wartości zadane zostały zrzutowane bezpośrednio na przedział (-32768,32767) oznaczałoby to, że dla maksymalnego wychylenia manetek pilota kwadrokoptera, zacząłby się on obracać z prędkością kątową 250 stopni na sekundę co z jednej strony dawałoby możliwość wykonywania bardzo agresywnych manewrów, lecz utrudniałoby pilotaż niedoświadczonej osobie. Dlatego też ze względów bezpieczeństwa jako maksymalną wartość prędkości obrotowej wokół każdej z osi przyjęto 30 stopni na sekundę. Osiągnięto to za pomocą odpowiednio przygotowanych definicji preprocesora, przedstawionych na poniższym listingu

Poniższe definicje PITCH_MAX_RANGE, ROLL_MAX_RANGE, YAW_MAX_RANGE określają maksymalne prędkości kątowe przechylenia, wychylenia oraz odchylenia wyrażone w stopniach na sekundę. Następnie wartości te są rzutowane na przedział (-32768,32767) oznaczający prędkości kątowe z zakresu (-250,250) stopni na sekundę. Powyższe definicje preprocesora używane są do określenia przedziału wartości, w których powinny zawierać się odczyty z żyroskopu.

```
#define GYRO_MAX_RANGE 250L
#define MAX_INT16 32767L
#define MIN_INT16 -32768L

#define PITCH_MAX_RANGE 30L //max pitch rotation speed in deg/sec
#define ROLL_MAX_RANGE 30L //max roll rotation speed in deg/sec
#define YAW_MAX_RANGE 30L //max yaw rotation speed in deg/sec

#define PITCH_MAX_VALUE ((PITCH_MAX_RANGE * MAX_INT16)/GYRO_MAX_RANGE)
#define PITCH_MIN_VALUE (-PITCH_MAX_VALUE)

#define ROLL_MAX_VALUE ((ROLL_MAX_RANGE * MAX_INT16)/GYRO_MAX_RANGE)
#define ROLL_MIN_VALUE (-ROLL_MAX_VALUE)

#define YAW_MAX_VALUE ((YAW_MAX_RANGE * MAX_INT16)/GYRO_MAX_RANGE)
#define YAW_MIN_VALUE (-YAW_MAX_VALUE)
```

Funkcja map służy do rzutowania liczby x zawierającej się w przedziale (in_min, in_max) na przedział (out_min, out_max). Jej definicję oraz przykład użycia przedstawiają dwa poniższe listingi.

```
int16_t map(int16_t x, int32_t in_min, int32_t in_max, int32_t out_min, int32_t out_max)
{
    return (int16_t)((int32_t)x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

temp_i16 = (int16_t)(reg_array[REG_THRUST]);
thrust_full_range = map(temp_i16,0,255,0,32767);

temp_i8 = (int8_t)(reg_array[REG_PITCH]);
temp_i16 = (int16_t)temp_i8;
pitch_full_range = map(temp_i16,-127,127,PITCH_MIN_VALUE,PITCH_MAX_VALUE);

temp_i8 = (int8_t)(reg_array[REG_ROLL]);
temp_i16 = (int16_t)temp_i8;
roll_full_range = map(temp_i16,-127,127,ROLL_MIN_VALUE,ROLL_MAX_VALUE);
```

```
temp_i8 = (int8_t)(reg_array[REG_YAW]);
temp_i16 = (int16_t)temp_i8;
yaw_full_range = map(temp_i16, -127, 127, YAW_MIN_VALUE, YAW_MAX_VALUE);
```

Powyższy fragment kodu odpowiedzialny jest za pobieranie zadanych wartości ciągu oraz prędkości kątowych kwadrokoptera, oraz rzutowanie ich na przedziały określone za pomocą definicji preprocesora, ograniczając tym samym maksymalne prędkości z jakimi kwadrokopter będzie mógł się obracać wokół swoich osi.

Po odpowiednim przygotowaniu zadanych wartości prędkości kątowych oraz ciągu silników, algorytm sterujący przechodzi do wygenerowania sygnałów sterujących dla kontrolerów silników. Służą mu do tego regulatory PID, których implementacja została w dużej mierze oparta na dokumentacji "AVR221: Discrete PID controller" [47]. Kod realizujący funkcje regulatora PID przedstawia poniższy listing:

```
int16_t pid_Controller(int16_t setPoint, int16_t processValue, struct PID_DATA *pid_st)
{
    int32_t error;
    int32_t p_term;
    int32_t d_term;
    int32_t i_term;
    int32_t ret;

    error = (int32_t)setPoint - (int32_t)processValue;

    pid_st->sumError = pid_st->sumError + error;
    if(pid_st->sumError > pid_st->maxSumError)
    {
        pid_st->sumError = pid_st->maxSumError;
    }
    else if(pid_st->sumError < -(pid_st->maxSumError))
    {
        pid_st->sumError = -(pid_st->maxSumError);
    }

    p_term = (int32_t)(pid_st->P_Factor) * error;
    i_term = (int32_t)(pid_st->I_Factor) * (pid_st->sumError);
    d_term = (int32_t)(pid_st->D_Factor) * (int32_t)(pid_st->lastProcessValue - processValue);

    pid_st->lastProcessValue = processValue;

    ret = (p_term + i_term + d_term) / SCALING_FACTOR;

    if(ret > 32767)
    {
        ret = 32767;
    }
    else if(ret < -32767)
    {
        ret = -32767;
    }
}
```

```
    return((int16_t)ret);
}
```

Jak widać jest to klasyczna implementacja regulatora PID, obliczającego na wstępie uchyb między wartością zadaną a zmierzoną. Następnie po obliczeniu całki z wartości uchybu i sprawdzeniu, czy nie wybiega ona poza wartości graniczne, obliczane są kolejno człony P, I oraz D regulatora. Po ich dodaniu regulator sprawdza czy wartość wyjściowa nie wykracza poza maksymalną lub minimalną dozwoloną wartość sygnału wyjściowego i w razie potrzeby dokonuje odpowiedniej korekcji. Wśród argumentów wejściowych do tej funkcji, jedynym który może nie być zrozumiałym w pierwszej chwili jest wskaźnik pid_st - jest to wskaźnik na strukturę PID_DATA, przechowującą wartości członów proporcjonalnego, całkującego oraz różniczkującego regulatora, wraz z całką sygnału błędu oraz ostatnią zmierzoną wartością sygnału, która niezbędna jest przy liczeniu jego pochodnej w następnej iteracji algorytmu.

```
pid_pitch = pid_Controller(pitch_full_range, gx, &pidData_Array[PITCH_PID]);
pid_roll = pid_Controller(roll_full_range, gy, &pidData_Array[ROLL_PID]);
pid_yaw = pid_Controller(yaw_full_range, gz, &pidData_Array[YAW_PID]);
```

Wykorzystanie omówionego wcześniej regulatora PID w kodzie kontrolera lotu przedstawia powyższy listing. Trzy wywołania funkcji "pid_Controller" używane są do obliczenia wartości sygnałów sterujących wychyleniem, przechyleniem oraz odchyleniem, które następnie użyte są do wygenerowania końcowych sygnałów PWM, wysyłanych do sterowników silników.

Kolejny listing przedstawia algorytm generowania sygnałów PWM, trafiających następnie do sterowników silników. Sygnały te są generowane tylko w przypadku, gdy zadana wartość ciągu silników przekracza 5% całkowitego zakresu. Jest to zabezpieczenie mające na celu uniemożliwienie pojawienia się ujemnych wartości sterujących silnikami - sygnały wyjściowe z regulatorów PID mogą przyjmować wartości dodatnie lub ujemne. Gdyby wartość zadanego ciągu silników wynosiła 0, zmienne pwm_fr, pwm_fl, pwm_br, pwm_bl mogłyby przyjąć wartości ujemne co doprowadziłoby do nieprawidłowego działania kwadrokoptera. W kodzie programu, w celu rozróżnienia który sygnał PWM odpowiada któremu silnikowi, użyto następujących oznaczeń: f-front (silnik przedni), b-back(silnik tylny), l-left(silnik lewy), r-right(silnik prawy), zatem sygnał pwm_fr jest odpowiedzialny za sterowanie przednim prawym silnikiem kwadrokoptera itd. Jak widać na powyższym listingu, sygnały pwm generowane są w trzech etapach:

```
if(thrust_full_range >= ((32767 * 5)/100))//5% of total range
{
    LED_PC1_ON;
    pwm_fr = (int32_t)thrust_full_range + (int32_t)pid_pitch -
        (int32_t)pid_roll + (int32_t)pid_yaw;

    pwm_fl = (int32_t)thrust_full_range + (int32_t)pid_pitch +
        (int32_t)pid_roll - (int32_t)pid_yaw;
```

```

pwm_br = (int32_t)thrust_full_range - (int32_t)pid_pitch -
          (int32_t)pid_roll - (int32_t)pid_yaw;

pwm_bl = (int32_t)thrust_full_range - (int32_t)pid_pitch +
          (int32_t)pid_roll + (int32_t)pid_yaw;

pwm_fr = (pwm_fr > 32767) ? 32767 : ((pwm_fr < 0) ? 0 : pwm_fr);
pwm_fl = (pwm_fl > 32767) ? 32767 : ((pwm_fl < 0) ? 0 : pwm_fl);
pwm_br = (pwm_br > 32767) ? 32767 : ((pwm_br < 0) ? 0 : pwm_br);
pwm_bl = (pwm_bl > 32767) ? 32767 : ((pwm_bl < 0) ? 0 : pwm_bl);

pwm_fr = (pwm_fr >> 8); // [0~32767] to [0~127] range conversion
pwm_fl = (pwm_fl >> 8); // [0~32767] to [0~127] range conversion
pwm_br = (pwm_br >> 8); // [0~32767] to [0~127] range conversion
pwm_bl = (pwm_bl >> 8); // [0~32767] to [0~127] range conversion

pwm_fr += 127;
pwm_fl += 127;
pwm_br += 127;
pwm_bl += 127;
}

else
{
    LED_PC1_OFF;
    pwm_fr = 10;
    pwm_fl = 10;
    pwm_br = 10;
    pwm_bl = 10;
}

```

- Obliczenie wstępnej wartości sygnału zgodnie ze wzorem przedstawionym w rozdziale 3
- Sprawdzenie czy sygnał pwm nie wykroczył poza dopuszczalny przedział (0,32767)
- Rzutowanie wartości sygnałów PWM z przedziału (0,32767) na przedział (0,127) tak aby mogły zostać użyte przez timery mikrokontrolera ATmega328p, generujące przebiegi wyjściowe na wyprowadzeniach układu.

6.2.5 Generowanie sygnałów sterujących

Generowanie sygnałów PWM zgodnych z modelarskim standardem omówionym w rozdziale 2 odbywa się za pomocą Timera0 oraz Timera1 mikrokontrolera ATmega328p. Bardzo ważnym aspektem przy opracowaniu koncepcji systemu był dobór częstotliwości sygnału zegarowego, którym taktowany jest główny mikrokontroler. Zadanie nie było trywialne ze względu na dużą liczbę typów rezonatorów kwarcowych, generujących różne częstotliwości sygnałów taktujących oraz możliwość użycia jednego z kilku możliwych trybów pracy każdego z użytych timerów. Ostatecznie zdecydowano się na rozwiązanie uwzględniające:

- Sygnał taktujący mikrokontroler o wartości 16MHZ
- Ustawienie obu timerów w tryb pracy PWM z kontrolą fazy.

Zgodnie z notą katalogową mikrokontrolera ATmega328p, częstotliwość generowanego przebiegu PWM dla wybranego trybu pracy Timera0 wyraża się wzorem [42]:

$$f_{OC_{nx}PCPWM} = \frac{f_{CLK_IO}}{N * 510} \quad (6.1)$$

gdzie N to wartość dzielnika częstotliwości zegarowej przyjmująca jedną z wartości: (1,8,64,256,1024)

Zatem przy taktowaniu mikrokontrolera zegarem o częstotliwości 16MHz i wybraniu dzielnika o wartości 64, generowany sygnał będzie miał częstotliwość w okolicach 500Hz, dzięki czemu będzie spełniał założenia przyjętego standardu.

Dla Timera1 wartość wzoru na częstotliwość sygnału PWM wraz z możliwymi wartościami dzielnika sygnału taktującego jest analogiczna.

```
void gpio_init(void)
{
    //PWM PINS
    DDRD |= ((1<<PD5)|(1<<PD6));

    DDRB |= ((1<<PB1)|(1<<PB2));
}
```

Samo włączenie Timera0 oraz Timera1 nie daje pewności czy sygnały PWM będą generowane na wyjściach układu, ponieważ wymagana jest jeszcze odpowiednia inicjalizacja pinów mikrokontrolera. Powyższy listing przedstawia funkcję, użytą do inicjalizacji wyprowadzeń PD5, PD6, PB1 oraz PB2 mikrokontrolera, wykorzystywanych następnie przez Timer0 oraz Timer1 jako wyjścia sygnałów PWM.

Poniższy listing przedstawia funkcje inicjalizujące Timer0 oraz Timer1, ustawiające dzielnik sygnału zegarowego na wartość 64, oraz włączające tryb pracy jako PWM z korekcją fazy.

```
void timer0_init(void)
{
    TCCR0A = ((1<<COM0A1)|(1<<COM0A0)|(1<<COM0B1)|(1<<COM0B0)|(1<<WGM00));
    //Phase Correct PWM, output A inverting, output B inverting

    OCR0A = 255;
    OCR0B = 255;

    TIMSK0 = (1<<TOIE0);

    TCCR0B=((1<<CS01)|(1<<CS00));//Timer Start, prescaler 64
}
```

```
void timer1_init(void)
{
    TCCR1A = ((1<<COM1A1)|(1<<COM1A0)|(1<<COM1B1)|(1<<COM1B0)|(1<<WGM10));
    //Phase Correct PWM, output A inverting, output B inverting

    OCR1A = 255;
    OCR1B = 255;

    TCCR1B = ((1<<CS11)|(1<<CS10)); //Timer Start, prescaler 64
}
```

Po prawidłowym zainicjalizowaniu wyjść mikrokontrolera oraz timerów używanych do generowania sygnałów sterujących, do zmiany ich współczynnika wypełnienia używa się funkcji `pwm_set` przedstawionej na poniższym listingu. Jako parametry wejściowe przyjmuje ona cztery 8-bitowe wartości sygnałów PWM, które następnie wpisuje do odpowiednich rejestrów Timera0 oraz Timera1.

```
void pwm_set(uint8_t fl, uint8_t fr, uint8_t bl, uint8_t br)
{
    OCROA = ~fl;
    OCROB = ~fr;
    OCR1A = ~bl;
    OCR1B = ~br;
}
```

6.3 Sterownik silnika

6.3.1 Funkcje programowe

Sterownik silnika został wykonany w oparciu o mikrokontroler ATiny85. Z zasobów tego mikrokontrolera wykorzystano dwa timery - Timer0 służący do generowania sygnału PWM sterującego silnikiem, oraz Timer1, który wraz z zewnętrznym żądaniem przerwania służy do pomiaru długości odebranego impulsu sterującego.

Kontroler silnika odpowiedzialny jest za trzy zadania:

- Odbiór sygnałów sterujących.
- Uśrednianie wartości sygnałów sterujących.
- Generowanie sygnałów PWM.

Algorytm działania sterownika jest bardzo prosty i opiera się na włączeniu zewnętrznego żądania przerwania aktywowanego smianą stanu logicznego odbieranego sygnału sterującego. W przypadku wywołania zewnętrznego przerwania, procedura przerwania sprawdza czy jest to początek czy

koniec impulsu sterującego, próbując poziom logiczny linii PB2. W zależności od zaistniałej sytuacji timer przeznaczony do mierzenia czasu trwania impulsu jest uruchamiany lub zatrzymywany oraz ustawiana jest odpowiednia flaga dla programu głównego. Na podstawie jej wartości program główny decyduje czy, można wygenerować nową wartość sygnału sterującego silnikiem.

6.3.2 Odbiór sygnałów sterujących

Zamieszczony poniżej listing przedstawia procedurę inicjalizującą pin PB2 mikrokontrolera, odpowiedzialny za odbiór sygnału sterującego jako pin wejściowy z wewnętrznym podciąganiem, a także inicjalizującą zewnętrzne przerwanie żądania aktywowane zmianą stanu logicznego pinu PB2.

```
INTO_INIT:  
    in r16,DDRB  
    cbr r16,(1<<PB2)  
    out DDRB,r16  
  
    in r16,PORTB  
    sbr r16,(1<<PB2)  
    out PORTB,r16  
  
    in r16,MCUCR  
    sbr r16,(1<<ISC00)  
    out MCUCR,r16  
  
    in r16,GIMSK  
    sbr r16,(1<<INT0)  
    out GIMSK,r16  
  
    in r16,GIFR  
    sbr r16,(1<<INTF0)  
    out GIFR,r16  
  
ret
```

Kolejny listing przedstawia procedurę inicjalizującą sprzętowy moduł Timera1, który jest używany do pomiaru czasu trwania impulsów sterujących. Wartość zegara taktującego mikrokontroler (12MHz) oraz wartość dzielnika tej częstotliwości używanego przez timer zostały tak dobrane, aby impuls o czasie trwania 2ms, czyli impuls o maksymalnej możliwej długości zgodny z objętym modelarskim standardem, spowodował zliczenie przez timer 500 cykli. Dzięki temu można uzyskać stosunkowo dużą dokładność pomiarów sygnału sterującego docierającego do kontrolera silnika.

```
TIMER1_INIT:  
    ldi r16,250  
    out OCR1C,r16  
  
    clr r16  
    out TCNT1,r16  
  
    in r16,TCCR1  
    sbr r16,(1<<PWM1A)
```

```

cbr r16,(1<<COM1A1)|(1<<COM1A0)
out TCCR1,r16

in r16,TIMSK
sbr r16,(1<<TOIE1)
out TIMSK,r16

in r16,TIFR
sbr r16,(1<<TOV1)
out TIFR,r16

ret

```

Poniższy listing przedstawia procedurę przerwania, uruchamianą przy każdej zmianie stanu logicznego pinu PB2 mikrokontrolera, odbierającego sygnał sterujący, pochodzący z kontrolera lotu. Procedura ta w zależności od zaistniałej sytuacji (początek impulsu sterującego, koniec impulsu sterującego) uruchamia lub zatrzymuje timer mierzący czas trwania impulsu sterującego oraz ustawia odpowiednie flagi dla programu głównego.

```

.org 0x0F
INTO_INTERRUPT:
    push r16
    in r16,SREG
    push r16
    //---
        sbis PINB,PB2
        rjmp PULSE_END
    PULSE_START:
        TIMER1_START

        clr r16
        sts INPUT_DISCONNECTED,r16
        rjmp INTO_INTERRUPT_END
    PULSE_END:
        TIMER1_STOP

        ldi r16,1
        sts PULSE_END_FLAG,r16
    //---
    INTO_INTERRUPT_END:
    pop r16
    out SREG,r16
    pop r16
reti

```

Poniższy listing przedstawia procedurę przepelenienia timera mierzącego czas trwania impulsów sterujących. Jak wspomniano wcześniej, impuls o maksymalnej długości równej 2ms spowoduje zliczenie przez timer 500 cykli. Dlatego też przerwanie to przy każdym wywołaniu zwiększa licznik przepelenień timera, dzięki czemu program główny będzie mógł określić czas trwania odebranego

impulsu. Ponieważ pin mikrokontrolera odpowiedzialny za odbiór sygnałów sterujących ma włączony rezystor podciągający, procedura jest w stanie wykryć sytuację odłączenia przewodu, którym podawany jest sygnał sterujący. W takim wypadku na linii PB2 zapanuje na stałe stan wysoki, co spowoduje włączenie Timera1 przez procedurę zewnętrznego przerwania i jego ciągłe zliczanie czasu trwania impulsu. Dzięki warunkowi, który sprawdza czy timer nie przepędził się 255 razy, sytuacja taka zostanie rozpoznana jako błąd systemu, i odpowiednia flaga zostanie ustawiona na potrzeby wykorzystania przez program główny.

```
TIMER1_OVERFLOW:
    push r16
    in r16,SREG
    push r16
    //---
        lds r16,TIMER1_OVERFLOW_COUNTER
        inc r16
        sts TIMER1_OVERFLOW_COUNTER,r16
        cpi r16,255
        breq PULSE_LENGTH_ERROR
        rjmp TIMER1_OVERFLOW_END
    //---
PULSE_LENGTH_ERROR:
    TIMER1_STOP
    TIMER1_OVERFLOW_COUNTER_CLEAR
    ldi r16,1
    sts INPUT_DISCONNECTED,r16
TIMER1_OVERFLOW_END:
    pop r16
    out SREG,r16
    pop r16
reti
```

6.3.3 Uśrednianie wartości sygnałów sterujących

Poniższy listing przedstawia procedurę zapisu odebranej wartości długości impulsu sterującego do tablicy, przechowującej osiem ostatnich wartości. Procedura operuje na wydzielonym fragmencie pamięci RAM, traktując go jak bufor cykliczny.

```
PULSE_LENGTH_ARRAY_WRITE:
    push r16
    lds r16,PULSE_LENGTHS_ARRAY_INDEX
    inc r16
    cpi r16,PULSE_LENGTHS_ARRAY_LENGTH
    brne NO_OVERLAP
    clr r16
NO_OVERLAP:
    sts PULSE_LENGTHS_ARRAY_INDEX,r16

    ldi XH,high(PULSE_LENGTHS_ARRAY)
    ldi XL,low(PULSE_LENGTHS_ARRAY)
    add XL,r16
```

```

    clr r16
    adc XH,r16
    pop r16
    st X,r16
ret

```

Poniższy listing przedstawia procedurę uśredniającą wartości ośmiu ostatnich wartości sygnału sterującego i zapisującą wynik w pamięci RAM dla dalszego wykorzystania przez program główny i procedury generujące sygnał PWM sterujący silnikiem.

```

AVERAGE_PULSE_LENGTHHS :
    rcall PULSE_LENGTH_ARRAY_WRITE

    ldi XH,high(PULSE_LENGTHS_ARRAY)
    ldi XL,low(PULSE_LENGTHS_ARRAY)
    clr r18//LOW BYTE
    clr r19//HIGH BYTE
    ldi r17,PULSE_LENGTHS_ARRAY_LENGTH
    AVERAGE_LOOP_01:
        ld r16,X+
        add r18,r16

        clr r16
        adc r19,r16

        dec r17
        brne AVERAGE_LOOP_01

        ldi r17,3
    AVERAGE_LOOP_02:
        ror r19
        ror r18
    dec r17
    brne AVERAGE_LOOP_02

    sts PULSE_LENGTH_AVERAGED,r18
ret

```

6.3.4 Generacja sygnałów PWM

Generowanie sygnału sterującego silnikiem odbywa się za pomocą Timera0, w który wyposażony jest mikrokontroler ATtiny85. Sterownik silnika wykorzystuje obydwa kanały wyjściowe Timera, przy czym jeden podłączony jest do tranzystora sterującego silnikiem, natomiast drugi do diody LED, służącej jako wskaźnik współczynnika wypełnienia sygnału sterującego silnikiem przydatny przy testach układu.

Poniższy listing przedstawia procedurę inicjalizującą Timer0, ustawiając go w tryb pracy PWM z korekcją fazy [45]. Preskaler ustawiony na wartość 8, sprawia że generowany sygnał PWM ma

częstotliwość około 2kHz - wartość ta została dobrana metodą prób i błędów tak, aby możliwe było najlepsze sterowanie silnikiem.

```

TIMERO_INIT:
    ldi r16,(1<<COM0A1) | (1<<COM0AO) | (1<<COM0B1) | (1<<WGM00)
    out TCCR0A,r16

    ldi r16,(1<<CS01)//PWM 1960 Hz
    out TCCR0B,r16

    clr r16
    out TCNT0,r16

ret

```

Poniższy listing przedstawia procedurę używaną do zmiany współczynnika wypełnienia sygnału PWM sterującego silnikiem oraz sygnału podłączonego do debugowej diody LED.

```

SETUP_PWM:
    lds r16,PULSE_LENGTH_AVERAGED
    out OCR0B,r16
    out OCR0A,r16
SETUP_PWM_END:
ret

```

Przedstawiony niżej listing przedstawia kod głównej pętli programu. Jak widać zasada jego działania opiera się na sprawdzaniu flag ustawianych przez odpowiednie przerwania, informujących o odebraniu impulsu sterującego lub o wystąpieniu błędu w jego odbiorze. W przypadku odebrania impulsu następuje zapis jego wartości do tablicy, uśrednienie ośmiu ostatnich pomiarów i modyfikacja sygnału PWM sterującego silnikiem. W przypadku wystąpienia błędu w odbiorze sygnału pętla główna stopniowo wygasza sygnał PWM sterujący silnikiem przez wpisywanie wartości zerowych do kolejnych pól bufora cyklicznego przechowującego osiem ostatnich wartości sygnału sterującego. Rozwiążanie to ma na celu uniknięcie gwałtownych zmian współczynnika wypełnienia sygnału sterującego silnikiem, które mogłyby powodować uszkodzenia silnika lub mikrokontrolera ATtiny85.

```

//=====MAIN=====
MAIN_INIT:
    ldi r16,high(RAMEND)
    ldi r17,low(RAMEND)
    out SPH,r16
    out SPL,r17

    rcall GPIO_INIT
    rcall RAM_VARIABLES_INIT
    rcall INTO_INIT
    rcall TIMERO_INIT
    rcall TIMER1_INIT
    sei

MAIN_LOOP:
    //---

```

```
lds r16,PULSE_END_FLAG
cpi r16,1
breq PULSE_RECEIVED
//---
lds r16,INPUT_DISCONNECTED
cpi r16,1
breq PULSE_NOT RECEIVED
//---
rjmp MAIN_LOOP_END
//=====
PULSE_NOT RECEIVED:
    lds r16,PULSE_LENGTH_AVERAGED
    cpi r16,0
    brne ATTENUATE_PWM
    sts INPUT_DISCONNECTED,r16
    rjmp MAIN_LOOP_END
ATTENUATE_PWM:
    dec r16
    sts PULSE_LENGTH_AVERAGED,r16
    rcall PULSE_LENGTH_ARRAY_WRITE
    rjmp PWM_UPDATE
//=====
PULSE_RECEIVED:
    clr r16
    sts PULSE_END_FLAG,r16

    lds r16,TIMER1_OVERFLOW_COUNTER
COND_01:
    cpi r16,0//Pulse shorter than 1ms
    brne COND_02
        rjmp PULSE_LENGTH_CONDITIONS_END
COND_02:
    cpi r16,1//Pulse length 1ms<=L<=2ms
    brne COND_03
        in r16,TCNT1
        rjmp PULSE_LENGTH_CONDITIONS_END
COND_03:
    ldi r16,250
    rjmp PULSE_LENGTH_CONDITIONS_END
//=====
PULSE_LENGTH_CONDITIONS_END:
    rcall AVERAGE_PULSE_LENGTHS
    TIMER1_CLEAR
    TIMER1_OVERFLOW_COUNTER_CLEAR
PWM_UPDATE:
    rcall SETUP_PWM

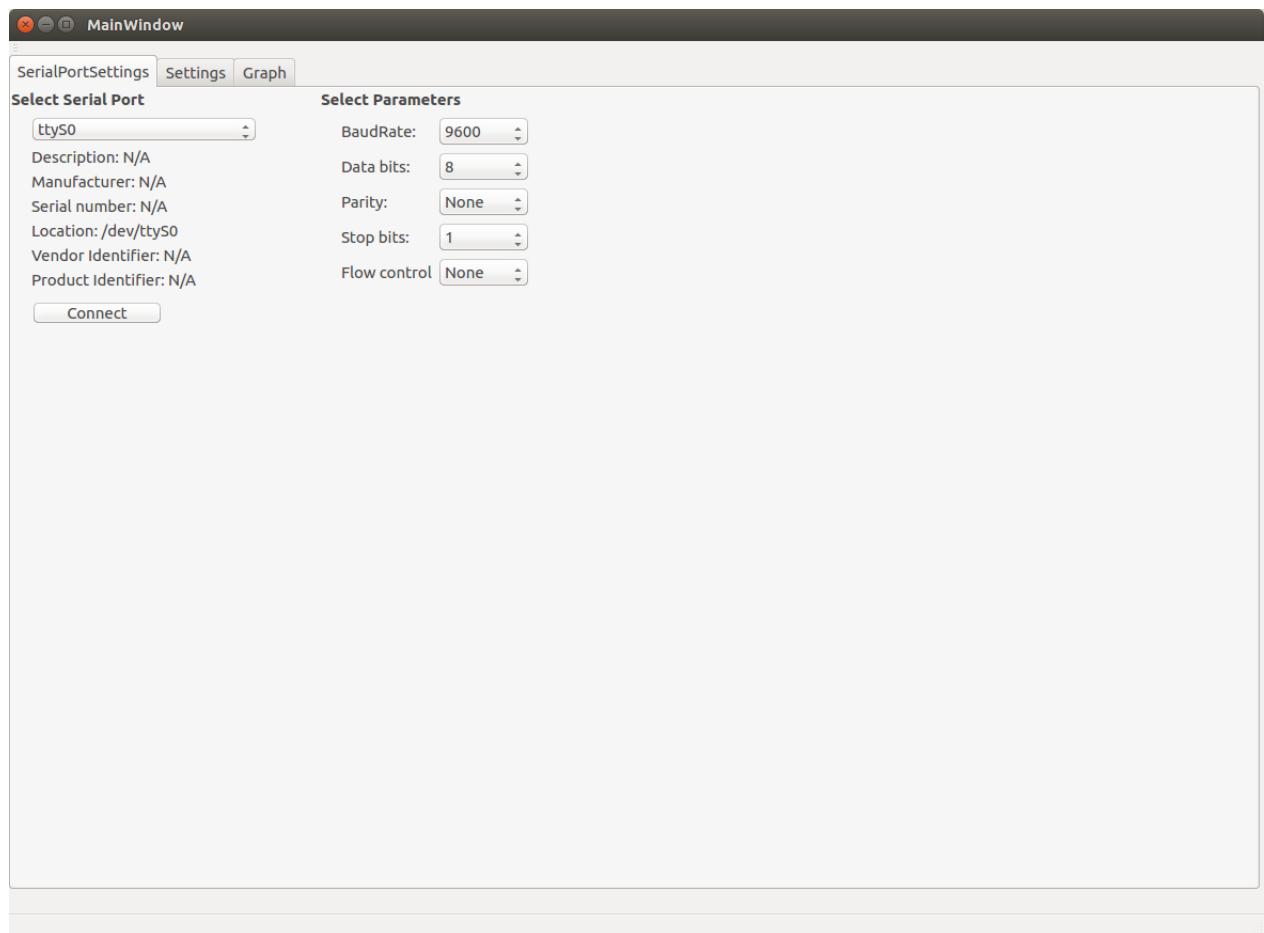
MAIN_LOOP_END:
rjmp MAIN_LOOP
//=====
```

6.4 Interface użytkownika do strojenia parametrów lotu

Aplikacja służąca do strojenia parametrów lotu kwadrokoptera nie jest przedmiotem projektu kwadrokoptera, podrozdział ten nie będzie więc skupiać się na samej implementacji, lecz na omówieniu funkcji, jaką spełnia ona w systemie oraz przedstawieniu możliwości udostępnianych użytkownikowi.

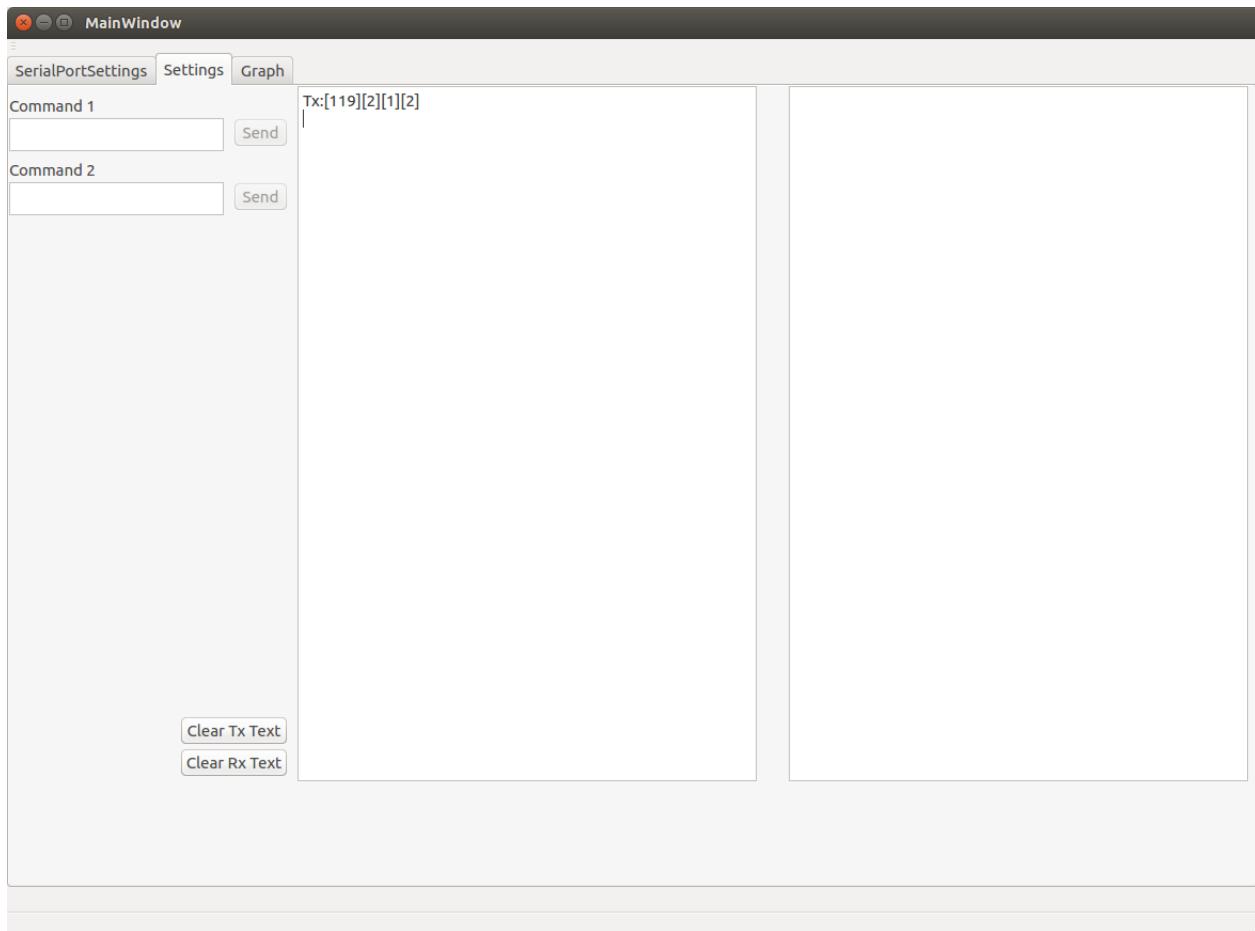
Podstawowym zadaniem omawianej aplikacji jest udostępnianie użytkownikowi systemu prostego sposobu konfiguracji dowolnych rejestrów wewnętrznych kwadrokoptera, odczytywanie komunikatów wysyłanych przez aplikację kontrolera lotu oraz przedstawianie w postaci graficznej wartości odczytywanych z żyroskopu i akcelerometru w celu prostego sprawdzenia poprawności działania algorytmu stabilizacji i kontroli lotu.

Aplikacja korzysta z modułu Bluetooth obecnego w komputerze użytkownika, widzianego jako port szeregowy. Po ustawieniu odpowiednich parametrów transmisji, użytkownik inicjuje połączenie z kwadrokopterem i w przypadku powodzenia operacji nawiązuje z nim łączność.



Rys. 6.2: Aplikacja użytkownika do strojenia parametrów lotu - ustawienia portu szeregowego

Rysunek 6.2 przedstawia ekran służący do konfiguracji parametrów portu szeregowego. Z rozwijanej listy widocznej pod napisem "Select serial port" użytkownik wybiera moduł HC-06 znajdujący się na płytce kontrolera lotu kwadrokoptera. Po wcisnięciu przycisku "Connect" aplikacja rozpoczyna próbę nawiązania połączenia z kwadrokopterem informując użytkownika o przebiegu operacji za pomocą paska informacyjnego u dołu ekranu (w przypadku braku komunikatów pasek ten pozostaje pusty).



Rys. 6.3: Aplikacja użytkownika do strojenia parametrów lotu - ustawienia rejestrów kwadrokoptera

Rysunek 6.3 przedstawia okno służące do zapisu rejestrów wewnętrznych kwadrokoptera. W lewej części okna widać dwa pola opisane jako "Command1" oraz "Command2", w których użytkownik wpisuje komendy jakie mają zostać wysłane. Komendy powinny być wpisywane zgodnie z omówionym wcześniej protokołem, używanym do komunikacji z kwadrokopterem, jednakże dla wygody użytkowania wszystkie pola ramki uzupełnia się w trybie tekstowym, a aplikacja przetwarza tę postać na wartości binarne. Dla przykładu, gdy użytkownik chce zapisać trzy rejestyre wewnętrzne kwadrokoptera począwszy od rejestru o adresie 1 wartościami 156, 10 i 250 musi on wpisać w pole "Command1" lub "Command2" następujący ciąg znaków:

w,1,3,156,10,250

gdzie:

- w - oznacza komendę zapisu rejestrów.
- 1 - oznacza adres pierwszego zapisywanej rejestrów.
- 3 - oznacza liczbę zapisywanych rejestrów.
- 156 - oznacza wartość pierwszego zapisywanej rejestrów.
- 10 - oznacza wartość drugiego zapisywanej rejestrów.
- 250 - oznacza wartość trzeciego zapisywanej rejestrów.

Poszczególne pola wiadomości oddzielone są przecinkiem.

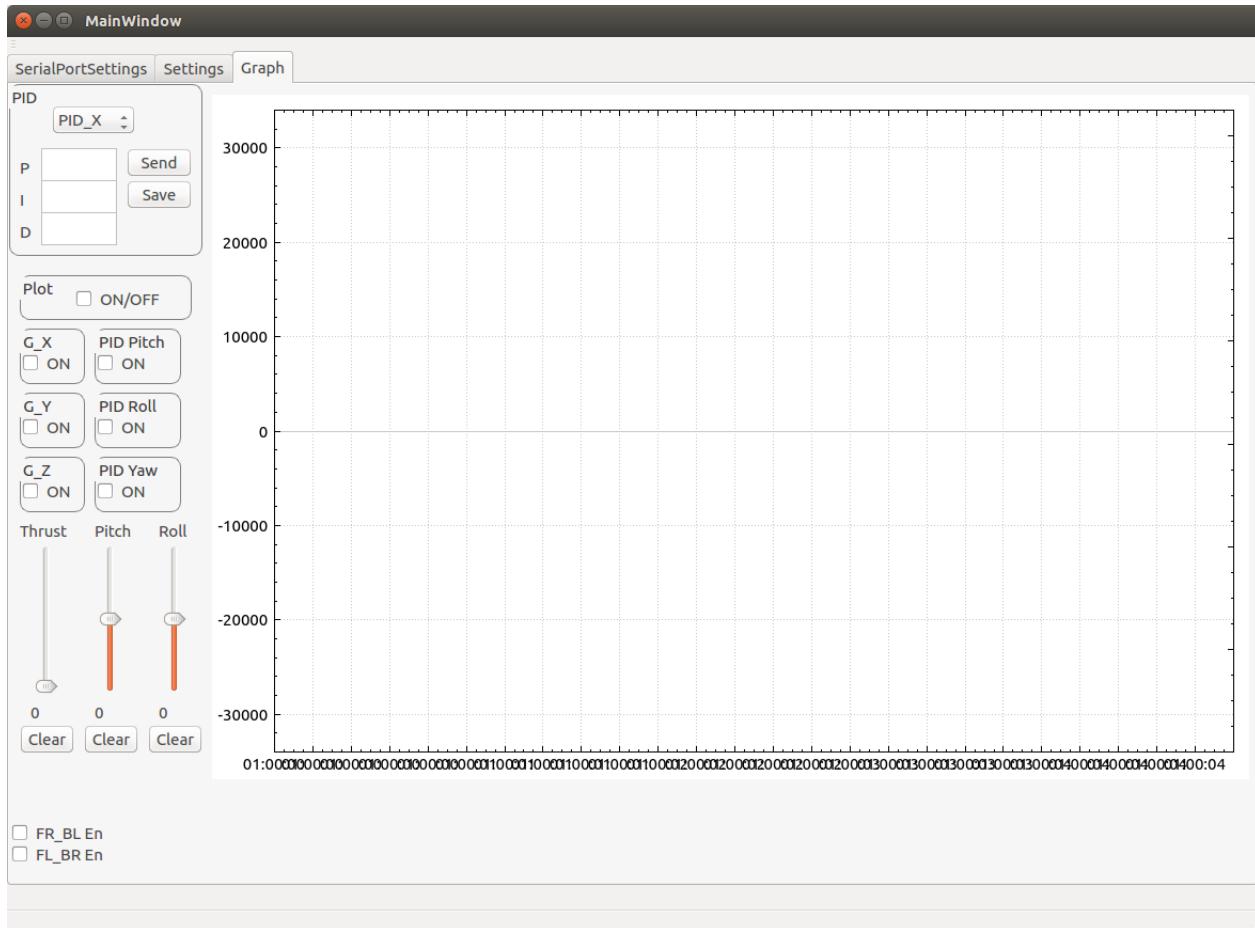
Analogicznie przy chęci odczytu 5 rejestrów począwszy od rejestrów o adresie 7 użytkownik powinien wpisać następującą komendę:

r,2,7,5

gdzie:

- r - oznacza komendę odczytu rejestrów.
- 2 - oznacza długość pola danych wiadomości w bajtach.
- 7 - oznacza adres pierwszego rejestrów do odczytu.
- 5 - oznacza liczbę rejestrów do odczytu.

Dane wysyłane przez użytkownika w stronę kwadrokoptera pokazują się w oknie po lewej stronie ekranu, natomiast dane wysłane przez kwadrokopter otrzymane przez aplikacje w oknie po prawej stronie ekranu. Przyciski "Clear Tx Text" oraz "Clear Rx Text" służą do czyszczenia tekstu odpowiednio w lewym i prawym oknie wiadomości.



Rys. 6.4: Aplikacja użytkownika do strojenia parametrów lotu - ustawienia regulatorów PID

Rysunek 6.4 przedstawia główne okno służące do strojenia regulatorów PID używanych przez algorytm stabilizacji lotu oraz do przedstawiania odczytów z żyroskopu umieszczonego w kwadrokopterze, jak również wartości wyjściowych z regulatorów PID w postaci graficznej.

W lewym górnym rogu okna widać pole wyboru nastawianego w danej chwili regulatora PID. Po wybraniu jednej z trzech możliwych wartości (PID_X, PID_Y, PID_Z) aplikacja odczytuje wartości członów P, I oraz D wybranego regulatora z pamięci kwadrokoptera. Następnie po modyfikacji parametrów regulatora następuje wysłanie nowych nastaw przez wcisnięcie przycisku "Send". Przycisk "Save" powoduje wysłanie do kwadrokoptera komendy zapisującej obecne nastawy wszystkich regulatorów PID w pamięci EEPROM kontrolera lotu. Poniżej części odpowiedzialnej za konfigurację regulatorów PID, znajdują się pola służące do włączania i wyłączania całego wykresu oraz do wybierania, które jego składniki (wartości pomiarów żyroskopu, wartości wyjściowe regulatorów PID) mają być widoczne dla użytkownika. Poniżej widać trzy suwaki, służące do płynnej zmiany zadanej wartości ciągu silników oraz prędkości przechylenia i wychylenia. Są one używane do testów i ustawiania parametrów regulatorów PID odpowiedzialnych za obrót kwadrokoptera wokół osi X i Y.

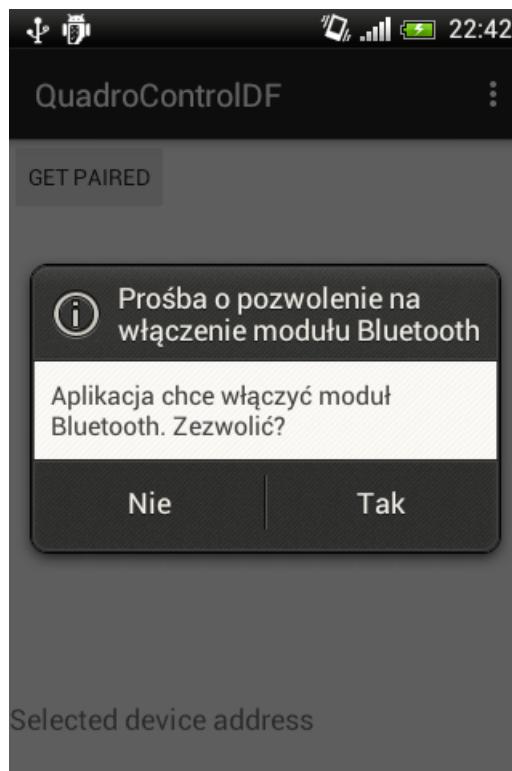
6.5 Interfejs użytkownika do kontroli lotu

Podobnie jak w przypadku aplikacji służącej do ustawiania parametrów lotu kwadrokoptera, aplikacja przeznaczona służąca do kontroli kwadrokoptera nie jest przedmiotem niniejszej pracy dyplomowej, dlatego też omówione zostanie jej działanie, a nie sama implementacja.

Omawiana aplikacja służy jedynie do kontroli kwadrokoptera w trakcie lotu. W związku z tym musi spełniać dwa podstawowe zadania:

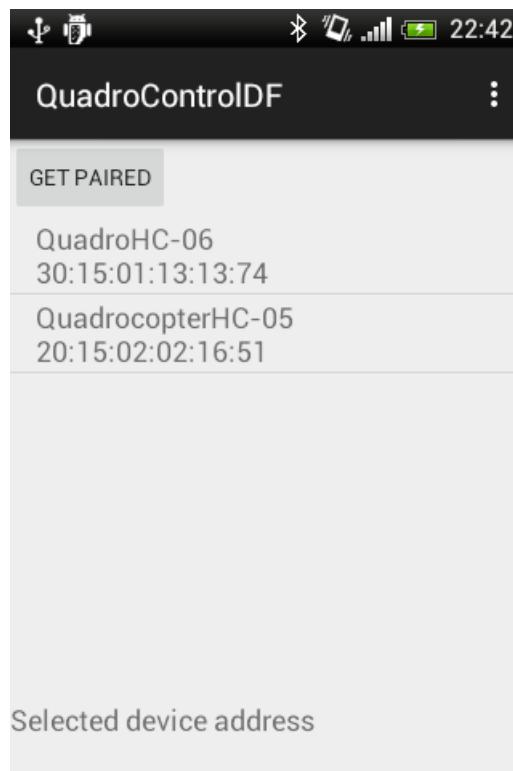
- Ustanowienie połączenia radiowego z kwadrokopterem.
- Przesyłanie do kwadrokoptera sygnałów sterujących jego lotem.

Obsługa aplikacji jest bardzo prosta. Użytkownik po jej włączeniu zostaje poproszony o możliwość włączenia modułu Bluetooth, w który wyposażony jest telefon. Następnie z listy urządzeń Bluetooth widzianych przez telefon, użytkownik wybiera moduł HC-06 umieszczony w płytce kontrolera lotu. Po wybraniu odpowiedniego urządzenia użytkownik uruchamia ekran odpowiedzialny za kontrolę kwadrokoptera, co inicjalizuje próbę nawiązania połączenia z kwadrokopterem. Gdy połączenie się powiedzie, użytkownik rozpoczyna sterowanie kwadrokopterem za pomocą dwóch joysticków wyświetlonych na ekranie telefonu.



RYS. 6.5: Aplikacja użytkownika - ekran startowy

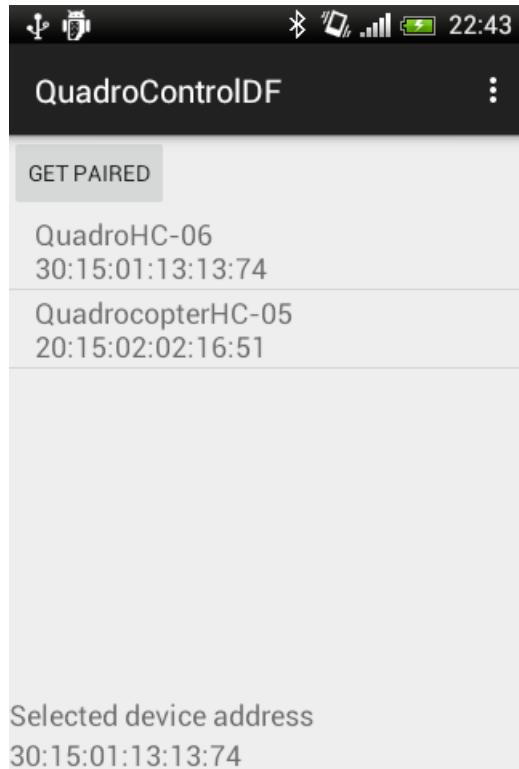
Rysunek 6.5 przedstawia wygląd komunikatu ukazującego się użytkownikowi zaraz po włączeniu aplikacji. W przypadku braku zgody na włączenie modułu Bluetooth aplikacja zostanie zamknięta.



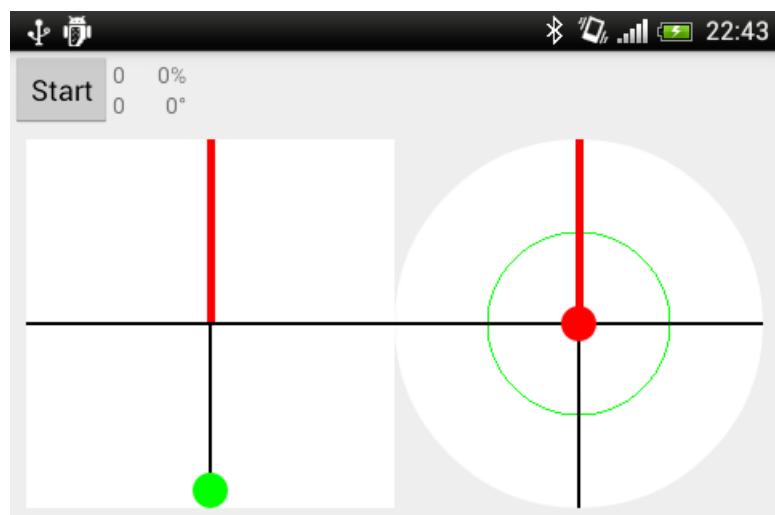
Rys. 6.6: Aplikacja użytkownika - parowanie urządzeń

Rysunek 6.6 przedstawia ekran widoczny po wyrażeniu zgody na włączenie modułu Bluetooth wewnętrz telefonu. Aplikacja pobiera listę urządzeń, które są widoczne dla telefonu i przedstawia ich nazwy wraz z unikalnymi identyfikatorami.

Rysunek 6.7 przedstawia wygląd głównego okna aplikacji po wyborze modułu Bluetooth umieszczonego w płytce kwadroopterze. Identyfikator wybranego urządzenia pojawił się w dolnej części ekranu. Z tego miejsca użytkownik może przejść do ekranu, za pomocą którego możliwa jest kontrola lotu kwadrooptera.



RYS. 6.7: Aplikacja użytkownika - wybór urządzenia



RYS. 6.8: Aplikacja użytkownika - ekran kontroli

Rysunek 6.8 przedstawia ekran służący do kontroli lotu kwadrokoptera. Widoczne są na nim dwa joysticki, z czego joystick widoczny po lewej stronie odpowiedzialny jest za kontrolę ciągu silników (ruch joysticka góra - dół) oraz za obrót kwadrokoptera wokół osi Z (ruch joysticka lewo-prawo), natomiast joystick widoczny po prawej stronie odpowiedzialny jest za przekształcenie (ruch joysticka lewo-prawo) oraz pochylenie (ruch joysticka góra-dół). W lewym górnym rogu ekranu widać przycisk "Start" służący do rozpoczęcia lotu kwadrokoptera oraz cztery pola wyświetlające

wartości liczbowe informujące o położeniu w pionie i poziomie każdego z joyskicków, służące do celów testowych.

Rozdział 7

Uruchomienie

7.1 Zakres testów

Po zaprojektowaniu wstępnej wersji płyt drukowanych kontrolera lotu oraz sterowników silników oraz po napisaniu wstępnych wersji aplikacji użytkownika do strojenia parametrów lotu oraz aplikacji do kontroli lotu kwadrokoptera można było przejść do ostatniego etapu tworzenia projektu czyli do uruchomienia, który został podzielony na następujące kroki:

- Testy kontrolera silnika.
- Testy kontrolera lotu kwadrokoptera.
- Testy aplikacji użytkownika.
- Strojenie regulatorów PID kwadrokoptera.

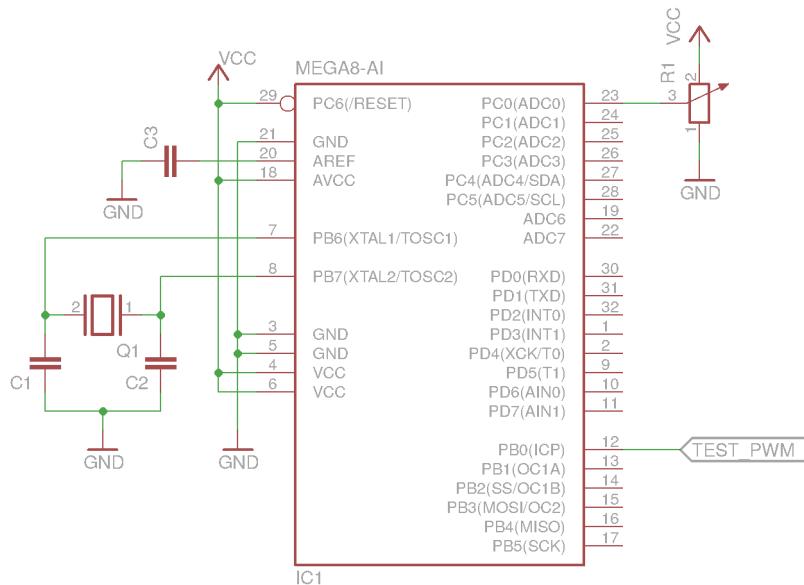
W kolejnych podrozdziałach opisane zostaną wszystkie wykonane kroki, wraz z opisem problemów, jakie pojawiły się w trakcie ich realizacji oraz ich rozwiązań przyjętych przez autora pracy.

7.2 Testy kontrolera silnika

7.2.1 Generator sygnałów testowych

Na potrzeby testów kontrolera silnika należało stworzyć prosty generator sygnału PWM zgodnego z przyjętym modelarskim standardem. Układ ten powstał w oparciu o mikrokontroler ATmega8, w którym za pomocą zewnętrznego potencjometru kontrolowano czas trwania impulsów sterujących

kontrolerem silnika w zakresie od 1ms do 2ms. Oprogramowanie zostało napisane w języku C, natomiast cały układ został zmontowany przy użyciu uniwersalnej płytki drukowanej.



Rys. 7.1: Generator testowego sygnału PWM w standardzie modelarskim

Rysunek 7.1 przedstawia uproszczony schemat omawianego układu. Na wyjściu oznaczonym jako "TEST_PWM" pojawiają się impulsy sterujące kontrolerem silnika, o szerokości wahającej się w przedziale od 1ms do 2ms, w zależności od położenia potencjometru.

Poniższy listing przedstawia kod programu, napisany na potrzeby generowania impulsów testowych. Jak widać wyjściowy sygnał PWM jest generowany programowo, ze względu na chęć uzyskania większej dokładności wytwarzanego przebiegu. Algorytm działania układu jest bardzo prosty - 8-bitowa wartość, pobierana z przetwornika analogowo-cyfrowego konwertowana jest następnie na 16-bitową wartość stanowiącą wartość współczynnika wypełnienia impulsu o szerokości 1ms. Następnie do tej wartości dodawana jest stała wartość będąca opóźnieniem pełnej 1ms i tak przygotowana wartość zostaje odliczona przez 16-bitowy Timer1 mikrokontrolera. Generowane impulsy powtarzają się co 4ms, dzięki wpisaniu do rejestru OCR1A timera przeliczonej wartości odpowiadającej wspomnianemu opóźnieniu.

```
#include <avr/io.h>
#include <avr/interrupt.h>

#ifndef F_CPU 14745600

#define MS_1 14745
#define MS_2 29491
#define MS_3 44236
#define MS_4 58982

void timer_init(void)
{
```

```
TCCR1A |= ((1<<COM1B1) | (1<<WGM11) | (1<<WGM10));
TCCR1B |= ((1<<WGM13) | (1<<WGM12));
OCR1A = MS_4;
OCR1B = MS_1;

}

void timer_start(void)
{
    TCCR1B |= (1<<CS10);
}

void adc_init(void)
{
    ADMUX |= ((1<<REFS0) | (1<<ADLAR));
    ADCSRA |= ((1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0));
}

void adc_start(void)
{
    ADCSRA |= (1<<ADSC);
}

void gpio_init(void)
{
    DDRB |= ((1<<PBO)|(1<<PB1)|(1<<PB2));
    DDRC &= ~(1<<PC0);
}

int main(void)
{
    gpio_init();
    timer_init();
    timer_start();
    adc_init();
    uint16_t OCR1B_value;
    while(1)
    {
        while(!(TIFR & (1<<OCF1B)))
        {
            //wait
        }
        TIFR |= (1<<OCF1B);
        PORTB ^= (1<<PBO);
        adc_start();
        while(ADCSRA & (1<<ADSC))
        {
            //wait
        }
        PORTB ^= (1<<PBO);

        OCR1B_value = (uint16_t)ADCH;
        OCR1B_value *= 57;
        OCR1B_value += MS_1;
    }
}
```

```

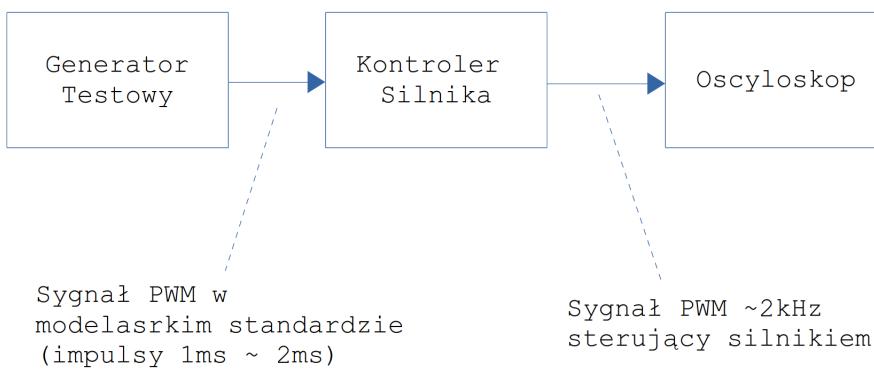
while(TCNT1 < MS_3)
{
    //wait
}
PORTB ^= (1<<PBO);
OCR1B = OCR1B_value;

while(!(TIFR & (1<<TOV1)))
{
    //wait
}
TIFR |= (1 << TOV1);
PORTB ^= (1<<PBO;

}
}

```

7.2.2 Testy pojedynczego kontrolera silnika



RYS. 7.2: Schemat blokowy układu pomiarowego pojedynczego kontrolera silnika

Rysunek 7.2 przedstawia system pomiarowy zestawiony do badania pojedynczego kontrolera silnika. Na jego wejście podawany był sygnał testowy zgodny z modelarskim standardem o wspólniku wypełnienia kontrolowanym przez użytkownika i zmiennym w czasie. W trakcie testów badano sygnał PWM wygenerowany przez sterownik oraz jego parametry przy podłączeniu docelowego silnika szczotkowego.

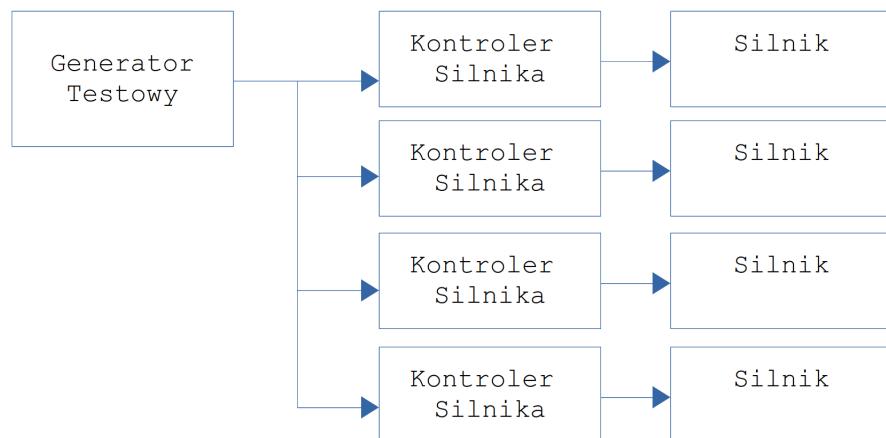
W trakcie testów pojedynczego kontrolera silników napotkano na następujące problemy:

- **Zakłócenia zasilania powodujące restart mikrokontrolera ATtiny85** - przyczyną problemu ukazały się zakłócenia generowane przez silnik szczotkowy w momencie przełączania jego uzwojeń. Przedostawały się one na linie zasilania mikrokontrolera i powodowały jego niedeterministyczne zachowania do których zaliczały się częste zawieszenia i restarty układu.

Rozwiązaniem problemu okazało się dodanie większych kondensatorów odsprzęgających przy samych wyprowadzeniach zasilania mikrokontrolera oraz dodanie kondensatora o bardzo małej wartości (22pF) tuz przy diodzie Schottky'ego zabezpieczającej silnik, które wspólnie służyły jako filtr zakłóceń mogących przedostać się na linie zasilania mikrokontrolera. Po tych zabiegach uzyskano znaczną poprawę stabilności zasilania mikrokontrolera oraz poprawę pracy silnika szczotkowego, dzięki czemu zawieszenia i ponowne uruchomienia układu zostały całkowicie zniwelowane.

- **Źle dobrana częstotliwość sygnału PWM, sterującego silnikiem** - błąd ten występował dla sygnału PWM sterującego silnikiem o częstotliwości 18kHz. Prawdopodobną przyczyną były zbyt duże skoki napięcia wytwarzane w trakcie przełączania uzwojeń silnika. Rozwiązaniem problemu okazało się zmniejszenie częstotliwości sygnału PWM do wartości około 2kHz, przy której zakłócenia nie były już tak dokuczliwe.
- **Zbyt gwałtowne zmiany wyjściowego sygnału PWM, przy ewentualnych błędach sygnału sterującego** - błąd ten występował przy nieprzewidzianych zmianach stanu sygnału wejściowego, takich jak zwarcie do plusa zasilania lub jego odłączenie. Powodowało to gwałtowne zmiany współczynnika wypełnienia sygnału PWM sterującego silnikiem, co w przypadku jednego prototypu doprowadziło do jego trwałego uszkodzenia. Rozwiązaniem tego problemu było dodanie procedury uśredniającej wartości ośmiu ostatnich czasów trwania impulsów sterujących, dzięki czemu nawet najbardziej gwałtowne zmiany sygnału wejściowego nie zagrażały już sterownikowi silnika.

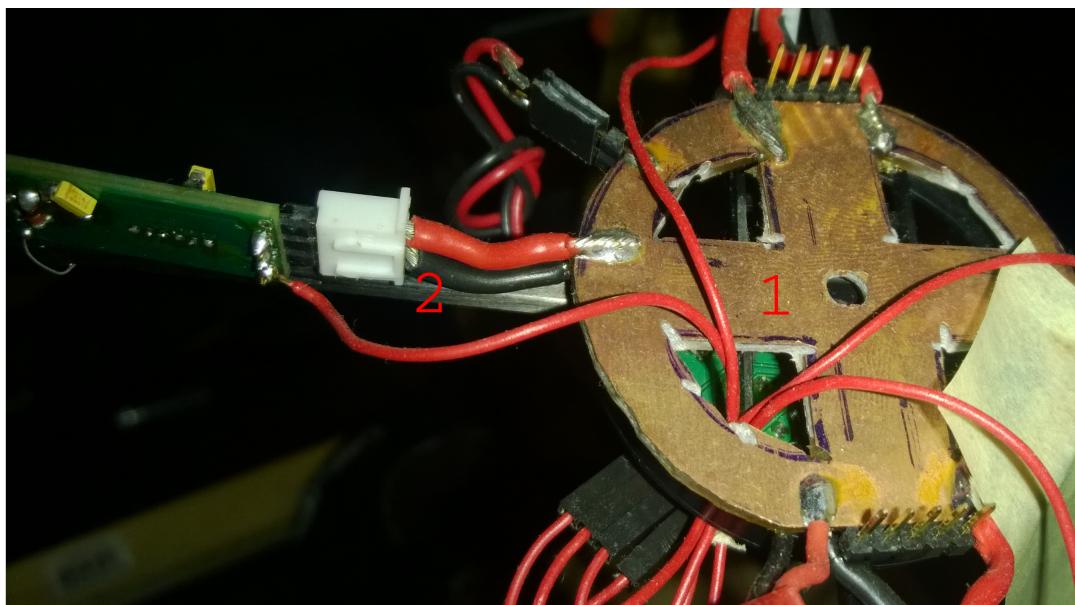
7.2.3 Testy zespołu czterech kontrolerów silników



Rys. 7.3: Schemat blokowy układu testowego czterech kontrolerów silników

Rysunek 7.3 przedstawia schemat układu zestawionego do przeprowadzenia testów czterech kontrolerów silników po zamontowaniu ich na ramie kwadrokoptera. Celem tego testu było sprawdzenie rówoczesnego działania wszystkich kontrolerów silników oraz upewnienie się, czy będą one w stanie unieść kwadrokopter. W trakcie testów napotkano na następujące problemy:

- **Nieprawidłowe doprowadzanie zasilania do kontrolerów silników** - problem ten występował w związku z zastosowaniem zbyt długich i ciejkich przewodów zasilających sterowniki silników. Przy niskich lub przy bardzo wysokich obrotach silników, spadki napięcia odkładające się na przewodach zasilających spowodowane impulsami prądowymi generowanymi przez silniki powodowały przypadkowe zachowanie się systemu. Najczęstszą reakcją było działanie jednego sterownika, podłączonego najbliżej akumulatora zasilającego, przy jednoczesnym ciągłym restartowaniu pozostałych sterowników. Rozwiązaniem tego problemu było zaprojektowanie specjalnej płytki rozprowadzającej zasilanie po kwadrokopterze za pomocą możliwie jak najszerzej miedzianych powierzchni, oraz skrócenie i pogrubienie przewodów zasilających sterowniki silników.



Rys. 7.4: Zdjęcie płytki rozprowadzającej zasilanie w kwadrokopterze (1) oraz skróconych i pogrubionych przewodów zasilających sterownik silnika (2)

Rysunek 7.4 przedstawia płytę rozprowadzającą zasilanie w kwadrokopterze opracowaną na podstawie wyników testów wszystkich czterech kontrolerów silników. Wykonano ją z dwustronnego laminatu, wykorzystując górną stronę płytki na rozprowadzenie masy akumulatora zasilającego kwadrokopter, a dolną stronę na podłączenie jego dodatniego bieguna zasilania. Po zamontowaniu płytki w kwadrokopterze problemy wynikające ze spadków napięcia na przewodach zasilających sterowniki silników przestały występować. W dalszym przebiegu testów regulacja pracy silników przebiegała bez problemów.

7.3 Testy kontrolera lotu kwadrokoptera



Rys. 7.5: Schemat blokowy zestawu testowego kontrolera lotu kwadrokoptera

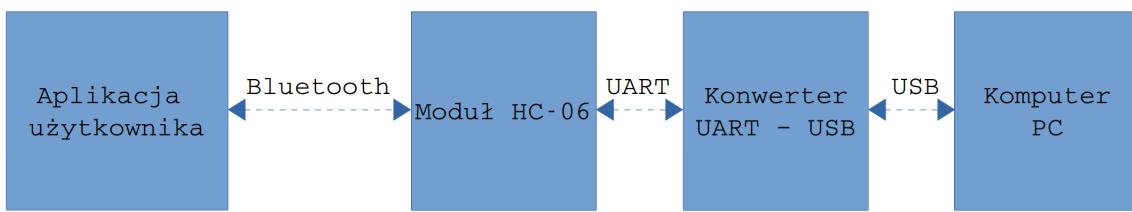
Rysunek 7.5 przedstawia schemat blokowy zestawu testowego użytego podczas testów kwadrokoptera. Komputer PC użytkownika wyposażony w moduł Bluetooth odpowiedzialny był za generowanie sygnałów testowych odbieranych i przetwarzanych po stronie testowanego układu. Testy kontrolera zostały podzielone na dwa etapy:

- **Testy generowania sygnałów PWM** - testy te polegały na wysyłaniu testowych danych, które bez udziału jakiegokolwiek algorytmu kontroli przekładały się na współczynniki wypełnienia impulsów sterujących kontrolerami silników. Dane były przesyłane w postaci binarnej zgodnie z protokołem omówionym w poprzednim rozdziale, a reakcja kontrolera obserwowana była za pomocą oscyloskopu. Testy te miały przede wszystkim sprawdzić działanie komunikacji radiowej, stabilność implementacji odbioru danych po stronie kontrolera lotu oraz prawidłowość generowanych sygnałów sterujących kontrolerami silników.
- **Testy komunikacji z czujnikiem MPU-6050** - testy te polegały na odczytywaniu wartości pomiarów z akcelerometru i żyroskopu oraz wysyłaniu ich w postaci tekstowej do komputera PC, gdzie następuała weryfikacja ich poprawności. W ich trakcie dokonywano zmiany położenia kątowego modułu czujników w przestrzeni weryfikując jednocześnie zmiany wartości odebranych pomiarów.
- **Wstępne testy algorytmu stabilizacji i kontroli lotu** - testy te polegały na wysyłaniu binarnych wartości zadanego ciągu silników oraz prędkości kątowych kwadrokoptera wokół trzech osi układu współrzędnych, a następnie na obserwacji generowanych sygnałów sterujących. Ich celem było sprawdzenie stabilności algorytmu kontroli lotu kwadrokoptera oraz przetestowanie implementacji regulatorów PID.

Wszystkie testy przebiegły pomyślnie, potwierdzając stabilność implementacji wszystkich modułów programowych kontrolera lotu.

7.4 Testy aplikacji użytkownika

Krokiem, który następował po testach prototypów kontrolerów silników oraz kontrolera lotu, były testy aplikacji użytkownika do strojenia parametrów lotu oraz do kontroli kwadrokoptera. Obie te aplikacje pomimo zapewniania użytkownikowi zupełnie innych funkcji przy komunikacji z kwadrokopterem w gruncie rzeczy realizują to samo zadanie, polegające na przetwarzaniu poleceń wydawanych za pomocą interfejsu graficznego na odpowiednie komendy wysyłane do kontrolera lotu.



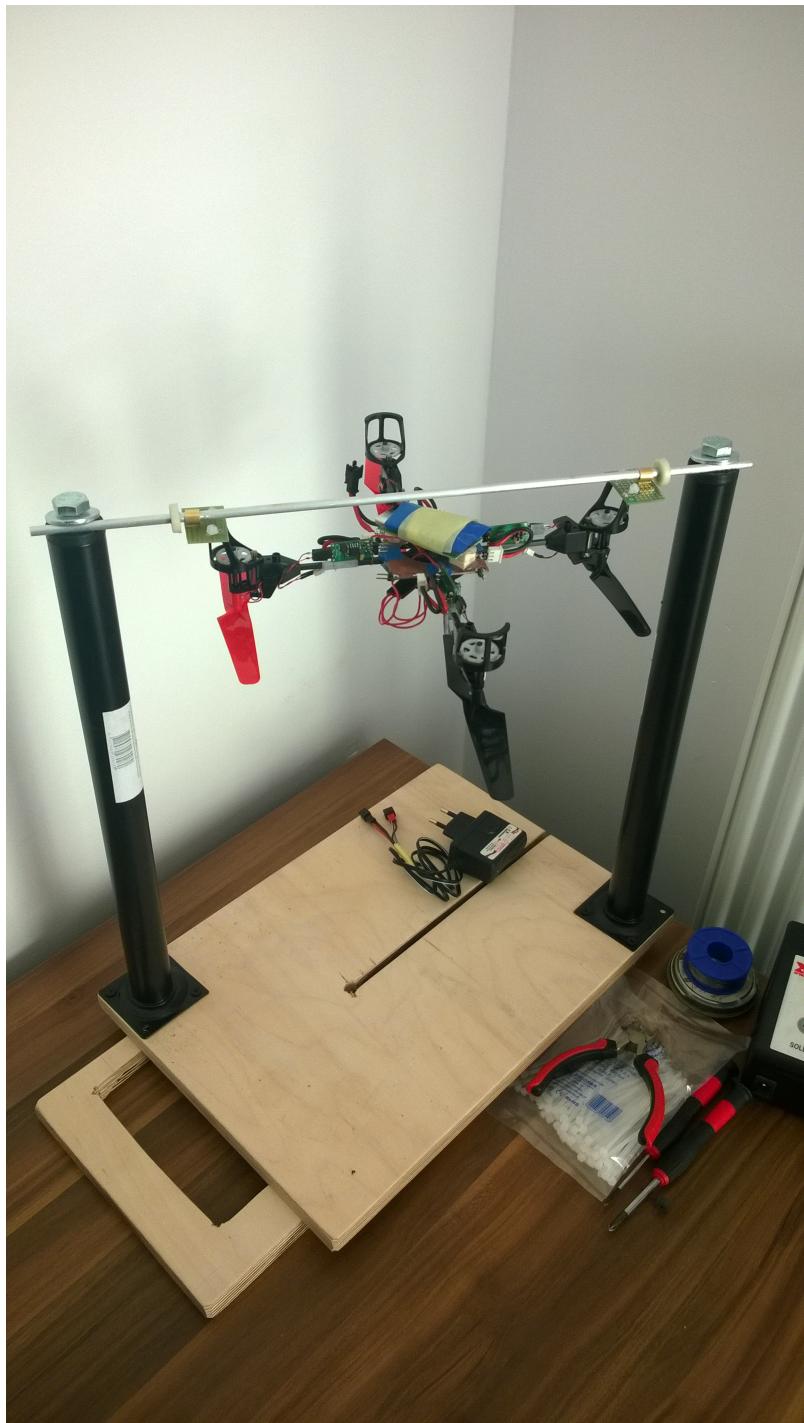
Rys. 7.6: Schemat blokowy zestawu testowego aplikacji użytkownika

Rysunek 7.6 przedstawia zestaw testowy, którego użyto do testów obu aplikacji użytkownika wspomnianych we wcześniejszej części tego podrozdziału. Aplikacja wysyłała odpowiednie komendy, w zależności od akcji podjętych przez użytkownika, przez interfejs Bluetooth. Dane te trafiały następnie do modułu HC-06 który wysyłał je przez interfejs UART do konwertera UART-USB widzianego przez komputer PC jako port szeregowy. W omawianym zestawie testowym nie użyto modułu Bluetooth wbudowanego w komputer PC, lecz zewnętrznego modułu identycznego do tego, który został zamontowany w kwadrokopterze, w celu odtworzenia docelowych warunków komunikacji radiowej.

Testy aplikacji użytkownika miały na celu zbadanie stabilności jej działania, oraz sprawdzenie poprawności generowanych komend. Dane odbierane po stronie komputera PC wyświetlane były w postaci binarnej i analizowane pod kątem zgodności z definicją protokołu komunikacyjnego oraz logicznej spójności danych.

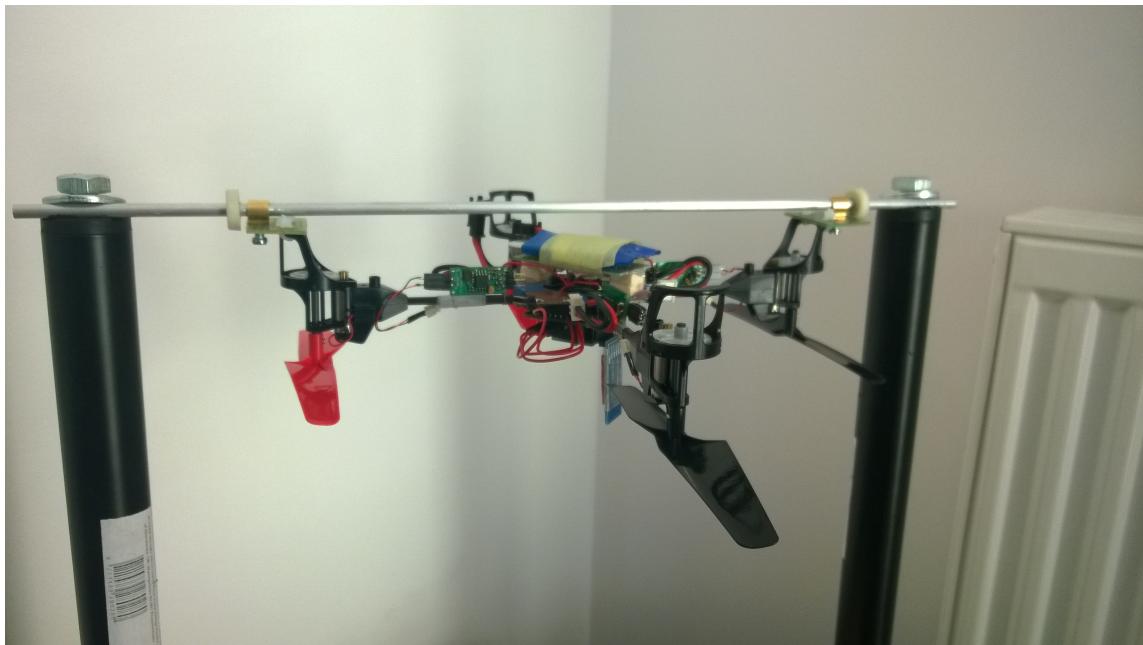
7.5 Strojenie regulatorów PID kwadrokoptera

Ostatnim etapem uruchomienia kwadrokoptera było nastrojenie jego regulatorów PID. Aby urządzenie miało zapewnioną stabilizację w płaszczyźnie poziomej, należy odpowiednio nastroić regulatorы odpowiedzialne za utrzymanie zadanej prędkości kątowej wokół osi X oraz Y. Dokonuje się tego przez unieruchomienie dowolnej pary przeciwnieństwanych silników oraz zawieszenie kwadrokoptera na odpowiadającej im pozostałe parze ramion tak, aby mógł obracać się wokół osi kalibrowanego regulatora.



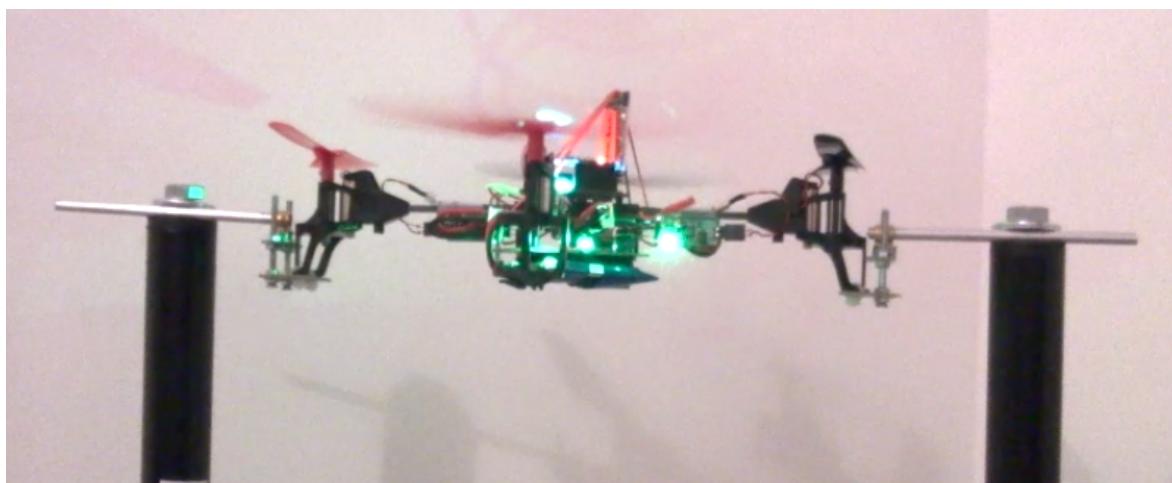
Rys. 7.7: Stanowisko testowe służące do strojenia regulatorów PID kwadrokoptera

Rysunek 7.7 przedstawia stanowisko użyte do strojenia regulatorów PID dla osi X oraz Y kwadrokoptera. Składa się ono z podstawy, do której przymocowano dwie pionowe metalowe rury. Na ich końcach przymocowano poziomy aluminiowy pręt, do którego za pomocą specjalnie zaprojektowanych łączników przymocowany został kwadrokopter. Stanowisko zostało zaprojektowane w taki sposób, aby mogło zostać wykonane możliwie niskim kosztem, z łatwo dostępnych materiałów.



Rys. 7.8: System mocowania kwadrokoptera do stanowiska testowego

Rysunek 7.8 przedstawia zbliżenie kwadrokoptera ukazujące dokładnie sposób jego mocowania do stanowiska testowego. Widoczna na zdjęciu wersja łączników musiała zostać poprawiona w związku z faktem, iż ich konstrukcja wymuszała znaczne odsunięcie środka ciężkości kwadrokoptera od jego osi obrotu, przez co pojawiały się błędy w trakcie testowania algorytmu kontroli lotu.



Rys. 7.9: System poprawionego mocowania kwadrokoptera do stanowiska testowego

Rysunek 7.9 przedstawia poprawioną wersję montażu kwadrokoptera do stanowiska testowego. Jak widać kształt łączników został tak zmieniony, aby oś obrotu pokrywała się w dużej mierze z położeniem środka ciężkości kwadrokoptera. Co więcej, nowe łączniki umożliwiają zmianę położenia osi obrotu względem ramy kwadrokoptera, dzięki czemu potencjalne zmiany jego środka ciężkości nie będą stwarzać problemów w trakcie testów. Drugą widoczną zmianą jest rozcięcie

aluminiowego pręta na dwie części, co było koniecznością, jako że przy zamontowaniu nowych łączników zahaczałby on o ramę kwadrokoptera.

Strojenie regulatorów PID przebiegło pomyślnie, dzięki czemu zyskano możliwość płynnej regulacji prędkości obrotowej kwadrokoptera wokół osi X_b oraz Y_b .

Rozdział 8

Wnioski

Celem niniejszej pracy było zapoznanie się z zasadą działania i stabilizacją lotu czterowirnikowych śmigłowców. Jako że autor pracy w chwili rozpoczęcia realizacji projektu nie posiadał żadnego doświadczenia w tworzeniu zdalnie sterowanych robotów, nacisk położono na prostotę projektowanych rozwiązań. Docelowo starano się osiągnąć prostą i uniwersalną konstrukcję, stanowiącą punkt wyjściowy dla przyszłych wersji kwadrooptera.

W ramach pracy wykonano następujące zadania:

- Opracowano projekt kwadrooptera:
 - Opracowano modułową koncepcję urządzenia z podziałem na kontroler lotu i sterowniki silników
 - Określono liczbę procesorów oraz określono zadania, za które są odpowiedzialne
 - Stworzono schematy elektryczne oraz projekty płyt drukowanych dla układów kontrolera lotu i sterowników silników
 - Dokonano montażu prototypu
 - Wykorzystując dostępne biblioteki programowe, napisano algorytm stabilizacji i kontroli lotu kwadrooptera
 - Opracowano binarny protokół używany do komunikacji z kwadroopterem
- Napisano aplikacje użytkownika:
 - Aplikacja przeznaczona na platformę Android, służąca do kontroli kwadrooptera
 - Aplikacja przeznaczona na platformę PC, służąca do testów oprogramowania kwadrooptera oraz strojenia parametrów algorytmu kontroli lotu
- Opracowano stanowisko testowe, przeznaczone do strojenia algorytmu kwadrooptera

- Dokonano testów wszystkich zaprojektowanych modułów elektronicznych oraz aplikacji
- Dokonano strojenia algorytmu kwadrokoptera

Efektem wykonania zadań wymienionych powyżej jest opracowanie kwadrokoptera będącego uniwersalnym modułowym urządzeniem, które w prosty sposób może być modyfikowane i rozszerzane o nowe moduły elektroniczne (np. nowe moduły transmisji radiowej, dodatkowe moduły czujników). Zaprojektowane urządzenie spełnia wszystkie założenia projektowe i wymagania techniczne, co dowodzi słuszności zastosowanych rozwiązań. W ramach projektu stworzono dwie wersje urządzenia. Błędy wykryte w pierwszej rewizji kwadrokoptera zostały poprawione w rewizji drugiej, która po przejściu testów układów elektronicznych oraz oprogramowania stała się finalną rewizją prezentowaną w pracy.

Bibliografia

- [1] Castillo P., Dzul A., Lozano R. Real-time stabilization and tracking of a four rotor mini-rotorcraft. *IEEE Transactions on Control Systems Technology*, 2003.
- [2] Bouabdallah S. *Design and control of quadrotors with application to autonomous flying*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2007.
- [3] Bouabdallah S., Murrieri P., Siegwart R. Design and control of an indoor micro quadrotor. *International Conference on Robotics and Automation*, 2004.
- [4] Premerlani W., Bizard P. Direction cosine matrix imu: Theory, 2009.
- [5] Hehn M., D'Andrea R. Quadrocopter trajectory generation and control. Institute for Dynamic Systems and Control.
- [6] Wei Dong, Guo-Ying Gu, Xiangyang Zhu, Han Ding. Modeling and control of a quadrotor uav with aerodynamic concepts. *World Academy of Science, Engineering and Technology*, 2013.
- [7] Luukkonen T. Modelling and control of quadcopter. Technical report, Aalto University, 2011.
- [8] Carrillo G, Lòpez D., Lozano R., Pégard C. *Quad Rotorcraft Control: Vision-Based Hovering and Navigation*. Springer, 2013.
- [9] Santoro C. How does a quadrotor fly? <http://www.slideshare.net/corradosantoro/quadcopter-31045379>.
- [10] Ali Reza Partovi. Development of a cross style quadrotor. Technical report, NATIONAL UNIVERSITY OF SINGAPORE, 2012.
- [11] Orsag M., Bogdan S. Influence of forward and descent flight on quadrotor dynamics. In *Recent Advances in Aircraft Technology*. InTech, 2012.
- [12] <https://code.google.com/p/owenquad/>, .
- [13] <http://www.stm32.eu/node/293>, .
- [14] <http://ardupilot.com>, .

- [15] <http://diydrones.com/profiles/blogs/raspberry-pi-quadrotor-running-pengupilot-on-linux>, .
- [16] <http://www.eng.utah.edu/~gale/mems/Lecture%2001%20Introduction%20to%20MEMS.pdf>, .
- [17] <http://www.ssl.umd.edu/projects/RangerNBV/thesis/2-4-1.htm>, .
- [18] <http://43zrtwysvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>, .
- [19] <http://owenson.me/build-your-own-quadcopter-autopilot>, .
- [20] Servo control interface in detail. <https://www.pololu.com/blog/17/servo-control-interface-in-detail>, .
- [21] Hobby servo fundamentals. <https://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf>, .
- [22] <http://www.endurance-rc.com/ppmtut.php>, .
- [23] <http://copter.ardupilot.com/wp-content/uploads/sites/2/2013/06/esc-motor-connect.jpg>, .
- [24] <http://fpvcentral.net/2013/03/high-performance-quadcopter-for-120-step-1-the-shopping-list>, .
- [25] <https://www.pinterest.com/pin/184858759674653805>, .
- [26] Urdhwareshe R., Bakshi M., Naiknavare P., Naik S. Design and Implementation of IMU Sensor Fusion and PID Control in Quadrotor. *International Journal of Electronics and Communication*, 2014.
- [27] Pedley M. *Tilt Sensing Using a Three-Axis Accelerometer*. Freescale Semiconductor, 2013.
- [28] Grygiel R., Bieda R., Wojciechowski K. Metody wyznaczania kątów z żyroskopów dla filtru komplementarnego na potrzeby określania orientacji imu. *Przegląd Elektrotechniczny*, 2014.
- [29] Wnuk M. Filtracja komplementarna w inercyjnych czujnikach orientacji. Technical report, Politechnika Wrocławskiego, 2014.
- [30] Marcin Karbowniczek. Układy MEMS. *Elektronika Praktyczna*, 2012.
- [31] Andrejasic M. Mems accelerometers. Master's thesis, University of Ljubljana, 2008.
- [32] Chollet F, Liu M. A (not so) short introduction to mems. <http://memscyclopedia.org/Document/IntroMEMS.pdf>.
- [33] https://www.mems-exchange.org/images/about_mems/microactuator.png, .

- [34] <http://electronicdesign.com/components/miniature-mems-accelerometer-adds-motion-sensing-consumer-products>, .
- [35] <http://www.instrumentationtoday.com/mems-accelerometer/2011/08/>, .
- [36] http://www.dutchops.com/Portfolio_Marcel/Articles/Instruments/Gyroscopic_Instruments/Theory_Gyroscopes.htm, .
- [37] https://en.wikipedia.org/wiki/Rotary_encoder, .
- [38] http://torahtimes.org/articles/starlight/Sagnac_interferometer.png, .
- [39] <http://foxtrotalpha.jalopnik.com>this-rare-photo-shows-the-most-sensitive-part-of-a-comb-1627339307>, .
- [40] <http://www.airspacemag.com/flight-today/how-things-work-ring-laser-gyros-32371541/?no-ist>, .
- [41] *Everything about STMicroelectronics 3-axis digital MEMS gyroscopes.* ST Microelectronics, 2011.
- [42] http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf, .
- [43] http://www.robotshop.com/media/files/pdf/rb-ite-12-bluetooth_hc05.pdf, .
- [44] <http://43zrtwsvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>, .
- [45] http://www.atmel.com/Images/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf, .
- [46] Grębosz J. *Symfonia C++*. Oficyna Kallimmach, 1999.
- [47] Avr221: Discrete pid controller. <http://www.atmel.com/images/doc2558.pdf>, .