

QUADROCOPTER

(sterowanie, stabilizacja, autopilot)

*Projekt przejściowy 2014/15
specjalności Robotyka
na Wydziale Elektroniki*

Politechnika Wrocławska 2014

Autorzy

Marcin Ciopcia

Michał Drwiega

Daniel Gut

Alicja Jurasik

Agata Leś

Kacper Nowosad

Piotr Semberecki

Hanna Sienkiewicz

Mateusz Stachowski

Paweł Urbaniak

Krzysztof Zawada

Skład raportu wykonano w systemie L^AT_EX



Praca udostępniana na licencji Creative Commons: *Uznanie autorstwa-Użycie niekomercyjne-Na tych samych warunkach 3.0*, Wrocław 2014. Pewne prawa zastrzeżone na rzecz Autorów. Zezwala się na niekomercyjne wykorzystanie treści pod warunkiem wskazania Autorów jako właścicieli praw do tekstu oraz zachowania niniejszej informacji licencyjnej tak długo, jak tylko na utwory zależne będzie udzielana taka sama licencja. Tekst licencji dostępny na stronie: <http://creativecommons.org/licenses/by-nc-sa/3.0/pl/>

Spis treści

I. Wstęp

1. Wstęp, Piotrek	6
1.1. Quadrocopter, Krzysiek	6
1.1.1. Własności obiektu, Daniel	6
1.2. Dostępne rozwiązania, Kacper	6
1.3. Cel pracy, Hania	6

II. Zadania

2. Stabilizacja w punkcie - przygotowania, Daniel	8
2.1. Interfejs do istniejącego oprogramowania i sprzętu, Kacper	8
2.1.1. Flying Machine Arena, Michał	8
2.1.2. Rozpoznanie protokołu quadrocoptera, Daniel	8
2.1.3. Rozpoznanie możliwości platformy, Daniel	8
2.2. Czujniki odległości, Alicja	9
2.2.1. Dobór czujników odległości, Hania	9
2.2.2. Rozmieszczenie czujników odległości, Paweł	9
2.3. Algorytmy rozpoznawania przesunięcia, Kacper	9
2.3.1. Algorytm Lucas—Kanade, Piotrek	9
2.3.2. Zmodyfikowany PTAM, Kacper	9
2.3.3. Algorytm SVO, Hania	9
3. Stabilizacja w punkcie - realizacja, Krzysiek	12
3.1. Ahsokam Marcin	12
3.2. Uruchomienie platformy latającej, Marcin	12
3.2.1. Uzbrajanie silników, Marcin	12
3.3. Uruchomienie kamery 3D, Michał	12
3.4. Uruchomienie komunikacji, Michał	14
3.4.1. Komunikacja ROS—kamera, Mateusz	15
3.4.2. Komunikacja ROS—MAVLINK, Mateusz	15
3.4.3. Komunikacja ROS—GroundStation, Krzysiek	15
3.4.4. Komunikacja ROS—czujniki, Daniel	15
3.4.5. Komunikacja MAVLINK—PixHawk, Mateusz	15
3.5. Czujniki odległości	15
3.5.1. Płytki PCB, Alicja	15
3.5.2. Oprogramowanie interfejsu, Michał	16
3.5.3. Dekodowanie sygnału PPM, Michał	19
3.5.4. Montaż czujników na platformie docelowej, Mateusz	20
3.6. Implementacja algorytmów przesunięcia, Kacper	20
3.6.1. Algorytm Lucas—Kanade, Piotrek	20
3.6.2. Zmodyfikowany PTAM, Kacper	20
3.6.3. Algorytm SVO, Hania	20
3.7. Stabilizacja w poziomie, Alicja	22
3.7.1. Fuzja sensoryczna, Alicja	22

3.8.	Stabilizacja w pionie, Paweł	23
3.8.1.	Fuzja sensoryczna, Michał	23
3.9.	Algorytm stabilizacji w punkcie	23
4.	Stabilizacja w pomieszczeniu	24
4.1.	Sterownik nadrzędny, Marcin	24
4.1.1.	Zdarzeniowy sterownik antykolizyjny, Paweł	24
4.1.2.	Lot wzdłuż ściany, Krzysiek	24
III. Testy		
5.	Testy stabilizacji	26
5.1.	Testy integracyjne na platformie docelowej	26
5.1.1.	Testy modułu ROS	26
5.1.2.	Testy modułu czujników	26
IV. Zakończenie		
6.	Podsumowanie	28
7.	Dodatki	29
	Bibliografia	30

Część I

Wstęp

1. Wstęp, Piotrek

1.1. Quadrocopter, Krzysiek

1.1.1. Własności obiektu, Daniel

1.2. Dostępne rozwiązania, Kacper

1.3. Cel pracy, Hania

Część II

Zadania

2. Stabilizacja w punkcie - przygotowania, Daniel

2.1. Interfejs do istniejącego oprogramowania i sprzętu, Kacper

2.1.1. Flying Machine Arena, Michał

2.1.2. Rozpoznanie protokołu quadrocoptera, Daniel

Przygotowanie specyfikacji interfejsu, Piotr

Wybór interfejsu, Piotr

2.1.3. Rozpoznanie możliwości platformy, Daniel

2.2. Czujniki odległości, Alicja

2.2.1. Dobór czujników odległości, Hania

Specyfikacja potrzeb, Hania

Dostępne rozwiązania, Alicja

Dobór czujników, Hania

2.2.2. Rozmieszczenie czujników odległości, Paweł

Przegląd rozwiązań, Paweł

Propozycja rozmieszczenia, Paweł

2.3. Algorytmy rozpoznawania przesunięcia, Kacper

2.3.1. Algorytm Lucas—Kanade, Piotrek

2.3.2. Zmodyfikowany PTAM, Kacper

2.3.3. Algorytm SVO, Hania

VO (ang. Visual Odometry) jest to proces estymacji ruchu pojazdu poprzez badanie zmian ruchu dzięki zdjęciom otrzymanym z pokładowej kamery. Warunki dobrej estymacji to

- odpowiednie oświetlenie sceny,
- dostosowanie szybkości zmienności otoczenia do użytego algorytmu, preferuje się raczej scenę statyczną,
- wybór odpowiednich zdjęć do przetwarzania.

Zalety

Ze względu na formę zawodów, quadcopter będzie poruszał się w zamkniętym pomieszczeniu, gdzie jak wiadomo sygnał GPS zawodzi. Dlatego zdecydowano się na użycie kamery pokładowej. Wizualna odometria może współpracować z innymi rozwiązaniami, dlatego finalnie do stabilizacji w punkcie system wizyjny zostanie połączony z czujnikami laserowymi oraz ultradźwiękowymi. Zminimalizuje to błąd estymacji ruchu, a co za tym idzie, będzie można przeciwdziałać dryfowi.

Ogólny schemat algorytmu SVO

Program SVO (Semi-direct Monocular Visual Odometry) [1] jest zaimplementowany w ROSie. Wykorzystuje on obrazy dostarczone z pokładowej kamery. Quadcopter zbiera informacje ze środowiska poprzez zdjęcia w dyskretnych chwilach czasu

$$I_1, I_2, \dots, I_n.$$

Odpowiednie macierze transformacji opisują relację pomiędzy dwiema pozycjami kamery

$$A_k = \begin{bmatrix} R_{k,k-1} & T_{k,k-1} \\ 0 & 1 \end{bmatrix}.$$

Pozycję kamery można obliczyć następująco

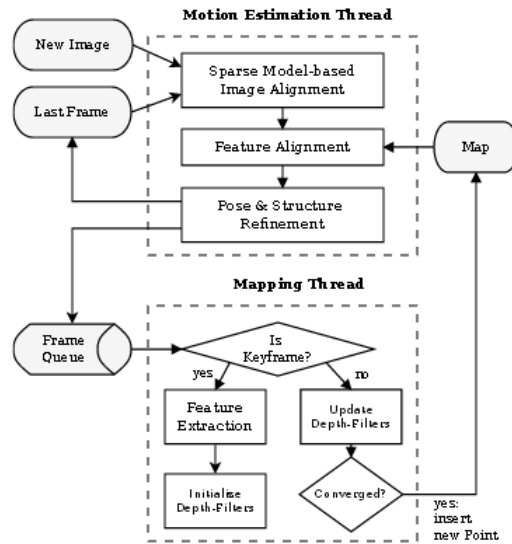
$$C_k = C_{k-1} \cdot A_k,$$

począwszy od znanej pozycji C_0 , ustawianej jako parametr. Głównym zadaniem algorytmu jest obliczenie transformacji A_k , po to żeby obliczyć pozycję kamery C_k , a co za tym idzie jej trajektorii, wykorzystując do tego obrazy I_k . Rysunek 2.1 przedstawia ogólny schemat algorytmu SVO. Jest on podzielony na dwa główne wątki, z czego jawnie korzystamy tylko z wątku estymującego ruch kamery. Można za pomocą tego rozwiązania zaimplementować algorytm SLAM.

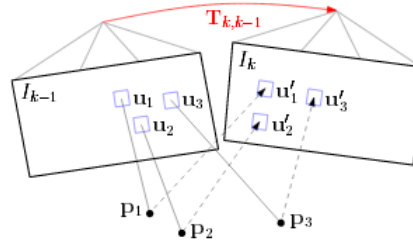
Etapy algorytmu SVO

Estymację ruchu/położenia kamery można podzielić na trzy etapy

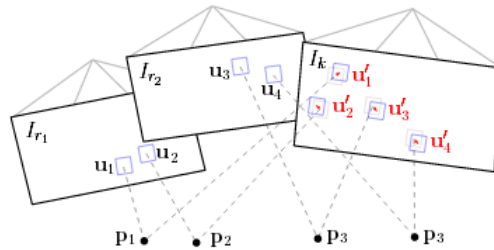
- etap pierwszy Rysunek 2.2 przedstawia zmianę pozycji między wcześniejszą i obecną ramką danych pośrednio przenosi pozycję p na odwzorowanie punktów w nowym obrazie. Algorytm minimalizuje fotometryczny błąd poprawki dotyczącej tych samych 3D punktów p .
- drugi etap Ze względu na niedokładności w punktach 3D oraz w estymacji pozycji kamery, fotometryczny błąd pomiędzy odpowiadającymi poprawkami (niebieskie kwadraty) w obecnej ramce i poprzedniej dodatkowo minimalizuje się przez optymalizację pozycji 2D każdego fragmentu indywidualnie, co jest pokazane na rysunku 2.3.
- trzeci etap Rysunek 2.4 wizualizuje ostatni etap, w którym to estymacja ruchu, pozycja kamery i struktura (punkty 3D) są optymalizowane, aby zminimalizować błąd odwzorowania, który został obliczony w poprzednim kroku.



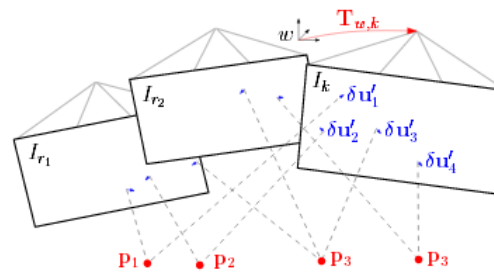
Rysunek 2.1. Schemat algorytmu SVO [1]



Rysunek 2.2. pierwszy etap [1]



Rysunek 2.3. drugi etap [1]



Rysunek 2.4. trzeci etap [1]

3. Stabilizacja w punkcie - realizacja, Krzysiek

3.1. Ahsokam Marcin

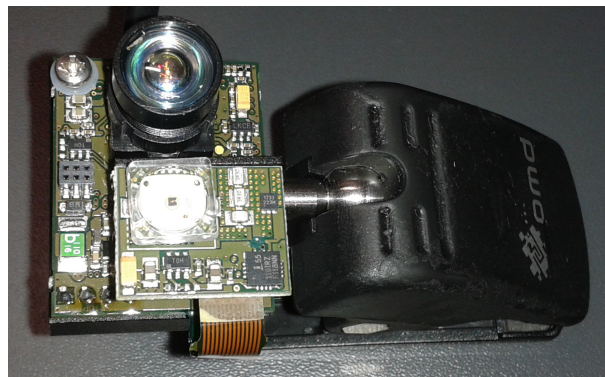
3.2. Uruchomienie platformy latającej, Marcin

3.2.1. Uzbieranie silników, Marcin

3.3. Uruchomienie kamery 3D, Michał

Quadrocopter wyposażony jest między innymi w kamerę 3D TOF **PMD CamBoard nano**, która została przedstawiona na rysunku 3.3. Kamera ta wykorzystuje sensor *PMD PhotonICs 19k-S3*, a jej podstawowe parametry są następujące:

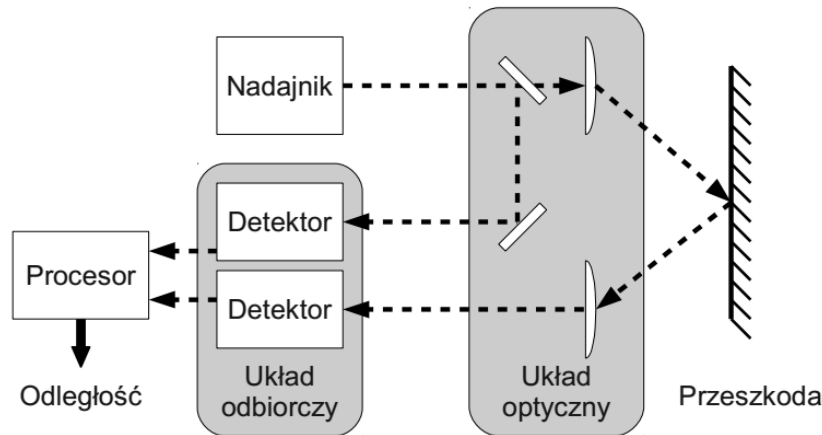
rozdzielczość obrazu 160x120 pikseli,
częstotliwość odświeżania obrazu do 90 Hz,
kąty widzenia 90° na 68°,
wymiały 37 x 30 x 25 mm,
długość fali światła 850nm,
pobór prądu około 500mA.



Rysunek 3.1. Wykorzystywana kamera 3D PMD CamBoard nano

Przedstawiona kamera wykorzystuje metodę pomiaru określaną mianem *TOF* (*Time Of Flight*), czyli mierzenia czasu przelotu impulsu. Budowa takiej kamery w postaci blokowej została przedstawiona na rysunku 3.2. Odnosnie zasady działania, układ nadajnika generuje odpowiednie impulsy światła, które po przejściu przez układ optyczny kierowane są w stronę przeszkody. Impulsy po odbiciu się od przeszkody wracają do układu odbiorczego, który z użyciem specjalizowanego procesora mierzy czas w jakim

impuls powrócił. Na tej podstawie, znając prędkość przelotu, obliczana jest odległość do przeszkody. Układy mierzące czas są zwykle bardzo dokładne, z tego względu, że w celu uzyskania rozdzielczości rzędu milimetrów pomiar czasu należy wykonywać z dokładnością do pojedynczych pikosekund. Inną metodą pomiaru jest metoda pośrednia polegająca na mierzeniu przesunięcia fazowego fali odbieranej względem nadawanej.



Rysunek 3.2. Budowa i zasada działania kamery TOF

Portowanie pakietu *pmd_camboard_nano*

W celu zapewnienia dostępu do obrazów kamery 3D z poziomu środowiska *ROS*, które umożliwi integrację tworzonego w ramach projektu oprogramowania zdecydowano się na przeniesienie pakietu *pmd_camboard_nano* obsługującego kamerę *PMD Camboard nano*, autorstwa Sergey'a Alexandrov'a do nowszej wersji *ROS Hydro* z wersji *Fuerte*.

Przeniesienie pakietu do *ROS'a* w wersji *Hydro* wymagało między innymi zmiany rozwiązania odnośnie budowania paczek. W poprzedniej wersji pakietu używano *roscbuild*, natomiast w nowszej wersji wykorzystano środowisko *catkin* oparte na wieloplatformowym systemie budowania pakietów *cmake*.

Dodatkowo, zdecydowano się na usunięcie z zależności pakietu *dynamic_reconfigure* odpowiadającego za dynamiczną zmianę parametrów pakietu, ze względu na problemy z jego uruchomieniem oraz w celu ujednolichenia sposobu konfiguracji całości oprogramowania, za pomocą plików *launch*. W obecnej wersji pełną konfigurację można przeprowadzić z użyciem odpowiedniego pliku *launch*, w którym opisane zostały parametry.

Kolejnym rozwiązaniem, które uległo zmianie jest sposób dostarczania sterowników kamery. W bieżącej wersji są one umieszczane bezpośrednio w katalogu *pmd_driver* pakietu *pmd_camboard_nano*. Wykorzystane zostały sterowniki w wersji 1.3.2.

Wybrane topiki publikowane przez pakiet *pmd_camboard_nano*

- obraz w odcieniach szarości (*/camera/amplitude/image*),
- informacje o kamerze i generowanym obrazie (*/camera/amplitude/camera_info*),
- odległości od środka optycznego sensora (*~distance/image*),
- obraz głębokości w mm (*/camera/depth/image*),
- informacje o kamerze i mapie głębokości (*/camera/depth/camera_info*),
- chmura punktów (*~points_unrectified*),

Uruchomienie pakietu i weryfikacja poprawności działania

- `source devel/setup.bash` - załadowanie zmiennych środowiskowych workspace'a,
- `roslaunch pmd_camboard_nano pmd_camboard_nano.launch` - uruchomienie pakietu,
- `roslaunch image_view image_view image:=/camera/depth/image`
 - wyświetlenie obrazu głębi,
- `roslaunch image_view image_view image:=/camera/amplitude/image`
 - wyświetlenie obrazu w odcieniach szarości.

3.4. Uruchomienie komunikacji, Michał

Komunikacja ze sterownikiem quadcoptera odbywa się z wykorzystaniem protokołu **MAVLink**, który został zaprojektowany do wymiany informacji między podsystemami robota oraz do komunikacji ze stacją kontroli (**GCS**). Specyfikację protokołu można znaleźć pod następującym odnośnikiem: <https://pixhawk.ethz.ch/mavlink>.

Dla dwóch wersji ROS'a: hydro i indigo został zaimplementowany protokół MAVLink, w postaci pakietu **mavros**. Pakiet ten umożliwia z poziomu ROS'a dostęp do parametrów sterownika, oraz pozwala wysyłać do niego komendy sterujące. Ponadto, pakiet ten umożliwia dodawanie własnych pluginów. Dokładny opis jest dostępny pod następującym odnośnikiem: <http://wiki.ros.org/mavros>.

Poniżej wymienione zostały wybrane funkcjonalności pakietu **mavros** w przypadku załadowania domyślnego zestawu pluginów.

- Publikowanie topic'ów zawierających dane z IMU (plugin **imu_pub**), między innymi:
 - orientacja obliczona przez sterownik (`~imu/data`),
 - wartości zmierzone przez IMU (`~imu/data.raw`),
 - odczyty magnetometru (`~imu/mag`),
 - temperatura (`~imu/temperature`),
 - ciśnienie powietrza (`~imu/FluidPressure`).
- Subskrybowanie topic'ów z komendami lotu:
 - zadane prędkości (`~setpoint/cmd_vel`),
 - zadane przyspieszenia (`~setpoint/accel`),
 - zadane prędkości kątowe (`~setpoint/att_vel`),
 - zadana poza (`~setpoint/attitude`).

Warto również wspomnieć o dwóch podobnych pakietach: **roscopter** i **mavlink_ros**. Pierwszy z nich występuje dla ROS'a w wersjach: groovy i hydro. Jego funkcjonalności są jednak ograniczone w porównaniu do pakietu **mavros**. Natomiast drugi pakiet, **mavlink_ros** jest przestarzały, a twórcy zalecają zastąpienie go pakietem **mavros**.

3.4.1. Komunikacja ROS—kamera, Mateusz

3.4.2. Komunikacja ROS—MAVLINK, Mateusz

3.4.3. Komunikacja ROS—GroundStation, Krzysiek

3.4.4. Komunikacja ROS—czujniki, Daniel

3.4.5. Komunikacja MAVLINK—PixHawk, Mateusz

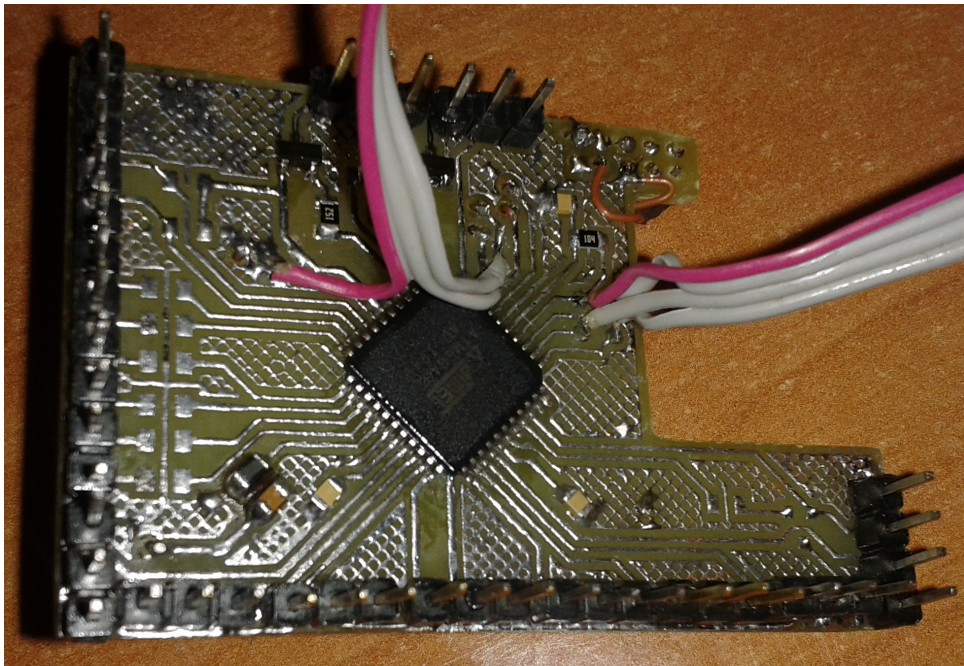
3.5. Czujniki odległości

3.5.1. Płytki PCB, Alicja

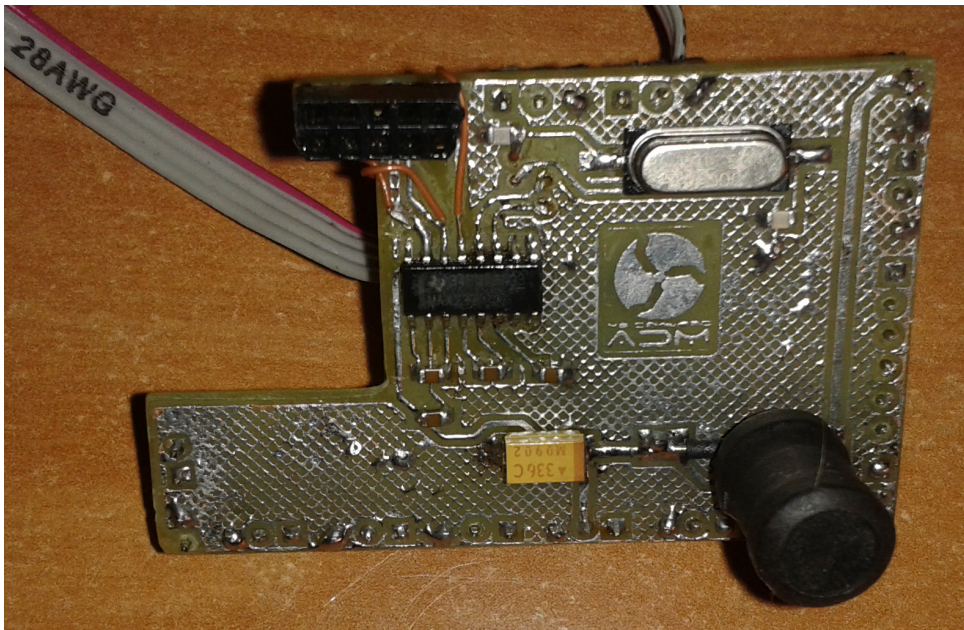
Specyfikacja potrzeb, Marcin

Wykonanie i uruchomienie płytki PCB, Michał

Na rysunku 3.3 przedstawiono wykonany interfejs czujników. Widoczne złącza umożliwiają podłączenie zasilania, czujników odległości Sharp, stopnia mocy oświetlenia, czy sygnału z aparatury RC.



(a) widok z góry



(b) widok z dołu

Rysunek 3.3. Przedstawienie wykonanego interfejsu

3.5.2. Oprogramowanie interfejsu, Michał

Urządzenie spełnia rolę interfejsu sterującego wybranymi podzespołami quadcoptera oraz jednostki pomiarowej, mierzącej odległości robota od przeszkód. Wyposażone zostało w mikrokontroler Atmel Atmega32. Aktualnie zaimplementowane funkcjonalności są następujące:

- cykliczne wykonywanie pomiarów napięć czujników odległości (porty ADC1-ADC7),
- pomiar odległości poprzez zliczanie czasu trwania impulsów (PW) generowanych przez sonary:

- przerwania zewnętrzne,
- timer zliczający czas.
- wysyłanie pomiarów do komputera nadrzędnego (UART),
- sterowanie oświetleniem quadcoptera,
- dekodowanie sygnału PPM z zadajnika RC,
- obsługa Pixhawk Failsafe.

Oprogramowanie zostało utworzone w języku C, z wykorzystaniem środowiska Atmel Studio 6, które jest nieodpłatnie udostępniane przez producenta mikrokontrolerów. Szczegółowa dokumentacja oprogramowania została utworzona z wykorzystaniem generatora Doxygen.

Protokół komunikacji z komputerem nadrzędnym

W celu umożliwienia komunikacji urządzenia z komputerem nadrzędnym konieczne było zaprojektowanie odpowiedniego protokołu komunikacyjnego. Warto zaznaczyć, że do omawianej komunikacji wykorzystuje się interfejs szeregowy UART.

Główną ideą zaprojektowanego protokołu komunikacji jednostki pomiarowej z komputerem jest wykorzystanie architektury klient-serwer. Rolę klienta pełni aplikacja uruchomiona na komputerze nadrzędnym, która wysyła odpowiednio zakodowane rozkazy. Natomiast interfejs sensorów dekoduje i realizuje otrzymywane rozkazy. W dalszej części przedstawiono szczegółowy opis protokołu.

Weryfikacja poprawności ramek

Suma kontrolna

W celu weryfikowania poprawności transmisji zaimplementowano obliczanie prostej sumy kontrolnej. Algorytm zlicza liczbę bitów o stanie wysokim w ciągu wysłanych bajtów. Metoda ta nie zapewnia dużej odporności na zakłócenia, ponieważ jest duże prawdopodobieństwo, że suma bitów o wysokim stanie niepoprawnego ciągu bajtów będzie równa sumie poprawnego. Jednak zaletą takiej sumy kontrolnej jest niewielka złożoność obliczeniowa.

Potwierdzanie rozkazów

Zwiększenie niezawodności transmisji zrealizowano przez implementację potwierdzania rozkazów. Potwierdzanie dotyczy tylko rozkazów, dla których nie jest oczekiwana transmisja zwrotna, czyli między innymi rozkazów konfiguracyjnych. W przypadku tej grupy oczekuje się potwierdzenia w postaci krótkiej ramki składającej się z kodu rozkazu oraz sumy kontrolnej.

Opis protokołu

W pierwszym odebranych bajcie danych przez jednostkę pomiarową, dwa najstarsze bity są bitami kontrolnymi o wartościach wysokiego stanu logicznego (11). Kolejne 6 bitów określa rozkaz, który należy wykonać. W ten sposób można zakodować do 64 różnych rozkazów, co wydaje się być wartością zupełnie wystarczającą. W przypadku, gdy nie ma dodatkowych danych dla jednostki pomiarowej, drugi bajt zawiera sumę kontrolną. Ogólna postać ramki danych została przedstawiona na rysunku 3.4.

Nr bajtu	1	2....n-1	n
Funkcja	2 bity startowe i kod rozkazu	Dodatkowe dane	Suma kontrolna

Rysunek 3.4. Ogólna postać ramki danych

Zaimplementowany protokół zakłada przesyłanie bardziej znaczących bajtów przed mniej znaczącymi (Big endian). Każdy dodatkowo przesyłany ciąg danych jest weryfikowany sumą kontrolną, której algorytm został przedstawiony w poprzednim podrozdziale. Rozkazy podzielono na cztery części w zależności od wartości pierwszych czterech bitów:

0xC (1100) żądania przesłania informacji zwrotnej,

0xD (1101) rozkazy konfiguracyjne,

0xE (1110) pozostałe rozkazy,

0xF (1111) pozostałe rozkazy.

W kolejnych podrozdziałach przedstawiono opisy poszczególnych grup rozkazów. Rozkazy opisane zostały przez podanie odpowiadających im wartości bajtów i krótkich opisów działania.

Żądania przesłania informacji zwrotnej - 0xC

1100 0000 - 0xC0

Weryfikacja poprawności działania transmisji. Oczekuje się odpowiedzi w postaci 0101 0101 - 0x55.

1100 1001 - 0xC1

Polecenie wysłania zmierzonych wartości z sensorów odległości oraz odczytanych wartości z sygnału PPM. Jednostką pomiarów sensorów Sharp są *mV*. W przypadku sonarów odległość podawana jest w *cm*. Wartości sygnału PPM dla kolejnych kanałów podawane jest w μs . Kolejność pomiarów w ramce określona została następująco:

- 2 bajty - napięcie w mV odczytane z sensora Sharp 1,
- 2 bajty - napięcie w mV odczytane z sensora Sharp 2,
- 2 bajty - napięcie w mV odczytane z sensora Sharp 3,
- 2 bajty - napięcie w mV odczytane z sensora Sharp 4,
- 2 bajty - napięcie w mV odczytane z sensora Sharp 5,
- 2 bajty - napięcie w mV odczytane z sensora Sharp 6,
- 2 bajty - napięcie w mV odczytane z sensora Sharp 7.
- 2 bajty - odległość zmierzona sonarem nr 1,
- 2 bajty - odległość zmierzona sonarem nr 2,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr1,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr2,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr3,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr4,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr5,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr6,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr7,
- 2 bajty - szerokość impulsu w μs w sygnale PPM dla kanału nr8,

Rozkazy konfiguracyjne - 0xD

1101 0000 - 0xD0

Polecenie uruchomienia oświetlenia quadcoptera.

1101 0001 - 0xD1

Polecenie wyłączenia oświetlenia quadcoptera.

1101 0010 - 0xD2

Włączenie Pixhawk Failsafe (ustawienie stanu wysokiego).

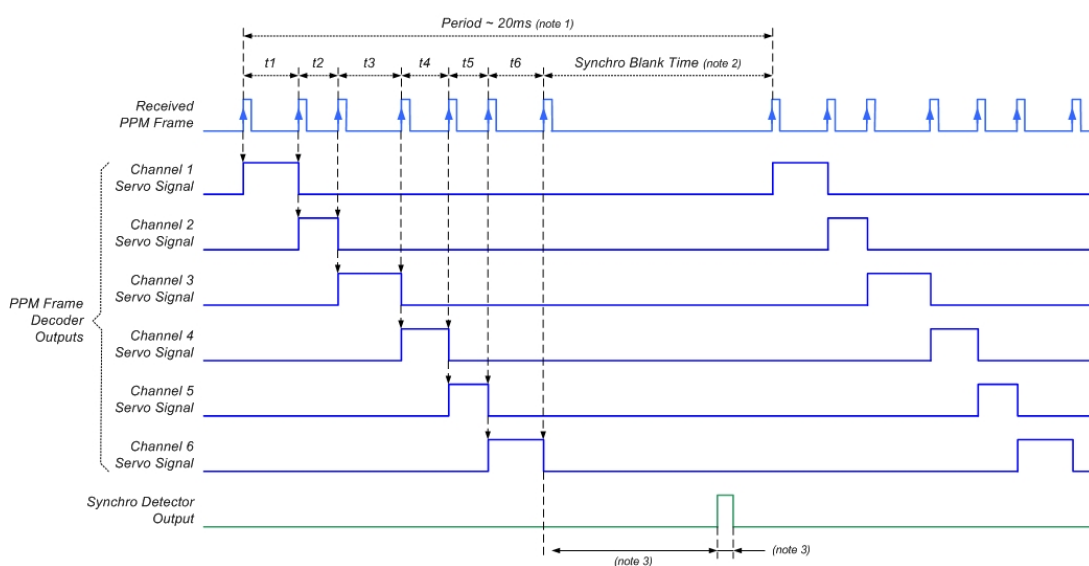
1101 0011 - 0xD3

Wyłączenie Pixhawk Failsafe (ustawienie stanu niskiego).

3.5.3. Dekodowanie sygnału PPM, Michał

W celu umożliwienia ręcznego sterowania quadcopterem przez operatora należało zrealizować dekodowanie sygnału PPM odbieranego przez aparaturę. Sygnał ten podłączony jest do wykonanego interfejsu i dekodowany przez umieszczony w nim mikrokontroler ATmega32.

PPM (Pulse Position Modulation) jest sygnałem, w którym informacje kodowane są z wykorzystaniem modulacji położenia impulsów. Umożliwia przesyłanie wartości w kilku kanałach (w przypadku dostępnej aparatury jest to 8 kanałów). Postać takiego sygnału przedstawiona została na rysunku 3.5.3.



Po zdekodowaniu sygnału PPM, wartości o poszczególnych kanałach wysyłane są do komputera nadrzędnego *nano6060*, gdzie przetwarzane są w środowisku *ROS* i następnie wysyłane do sterownika quadcoptera Pixhawk'a. Takie podejście umożliwia ręczne sterowanie, ale także realizację korekcji sterowania z wykorzystaniem modułu antykolizyjnego.

Realizacja dekodowania sygnału PPM na mikrokontrolerze

Na potrzeby dekodowania sygnału PPM wykorzystano przerwanie zewnętrzne generowane w zależności od rodzaju zbocza sygnału. Rozpoznawanie początku ramki zrealizowane zostało przez badanie przekroczenia maksymalnego odstępu między impulsami, które

w przypadku zwracanych wartości kanałów wynosi 2ms. Poniżej przedstawiono funkcję realizującą obsługę przerwania generowanego w przypadku zmiany stanu pinu do którego podłączono sygnał PPM.

Wydruk 3.1. Obsługa przerwania realizującego dekodowanie PPM

```
1 ISR (INT2_vect)
2 {
3     cli();
4     if (MCUCR & _BV(ISC2)) // rising edge interrupt
5     {
6         TCNT2 = 0; // Timer2 reset
7         timeCntPPM = 0;
8         MCUCR &= ~_BV(ISC2); // change to falling edge detect
9     }
10    else // falling edge interrupt
11    {
12        if ((timeCntPPM + TCNT2) > MAX_PPM_WIDTH) // new frame
13        {
14            channelPPMcnt = 0;
15        }
16        else // measurement for PPM channel
17        {
18            channelPPM[channelPPMcnt] = timeCntPPM + TCNT2;
19            if (channelPPMcnt < 7) // increase channel counter
20            {
21                channelPPMcnt++;
22            }
23        }
24    }
25    GIFR |= (1 << INTF1); // interrupt flag clear
26    sei();
27 }
```

Komunikacja ROS-czujniki, Daniel

3.5.4. Montaż czujników na platformie docelowej, Mateusz

3.6. Implementacja algorytmów przesunięcia, Kacper

3.6.1. Algorytm Lucas—Kanade, Piotrek

3.6.2. Zmodyfikowany PTAM, Kacper

3.6.3. Algorytm SVO, Hania

Uruchomienie kamery

Do działania algorytmu SVO jest potrzebny obraz z kamery, która została uruchomiona dzięki sterownikom zaimplementowanym w paczce *usb_cam*. Po uruchomieniu algoryt-

mu, należy uruchomić plik launchowy, dostarczający obraz z kamery. Poniższy wydruk przedstawia uruchomienie kamery.

Wydruk 3.2. Uruchomienie kamery

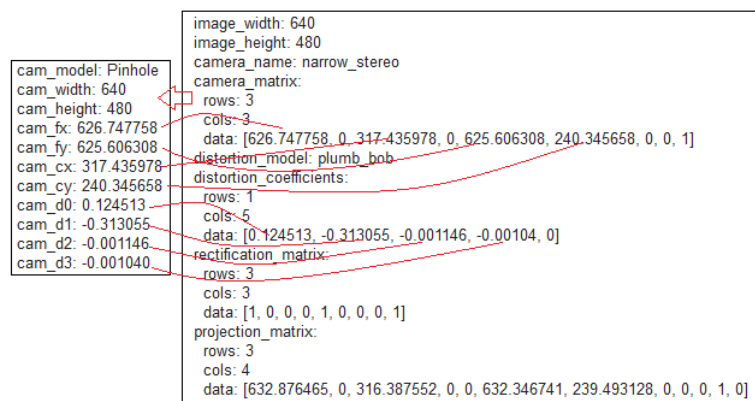
```

1 <launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
3  // output="screen" >
    <param name="video_device" value="/dev/video0" />
5    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
7    <param name="pixel_format" value="mjpeg" />
    <param name="camera_frame_id" value="usb_cam" />
9    <param name="io_method" value="mmap"/>
  </node>
11 </launch>

```

Kalibracja kamery

Do uruchomienia algorytmu potrzebny jest plik kalibracyjny. Kalibrację kamery wykonano standardowym narzędziem zaimplementowanym w ROSie w paczce `camera_calibration`. Wynikiem tej operacji jest plik `*.yaml`. Na potrzeby algorytmu przekonwertowano plik `*.yaml` na format `*.yaml`.

Rysunek 3.5. Konwersja pliku `*.yaml` na `*.yaml`

Rysunek 3.5 przedstawia ręczne przekonwertowanie pliku kalibracyjnego.

Pozycja kamery

Do dalszego przetwarzania informacji dostarczonych z działającej paczki SVO można wykorzystać wiadomość publikowaną poprzez topic `/svo/points`. Rysunek 3.6 pokazuje strukturę kluczowego topicu SVO. Typ wiadomości to `visualization_msgs`. Wiadomość ta jest zbiorem informacji używanych przez takie pakiety jak, np. `rviz`. Główną wiadomością w tym topicu jest `visualization_msgs/Marker`. Do dalszego przetwarzania użyto komunikatu zawierającego informację o pozycji kamery. Można tą informację uzyskać poprzez subskrybowanie się do tego topicu.

header:	
seq: 5892	std_msgs/Header header
stamp:	uint32 seq
secs: 1418673322	time stamp
nsecs: 170325262	string frame_id
frame_id: /world	string ns
ns: trajectory	int32 id
id: 8195	int32 type
type: 1	int32 action
action: 0	geometry_msgs/Pose pose
pose:	geometry_msgs/Point position
position:	float64 x
x: -0.00851521046703	float64 y
y: -0.00349843886924	float64 z
z: 0.0106530645546	geometry_msgs/Quaternion orientation
orientation:	float64 x
x: 0.0	float64 y
y: 0.0	float64 z
z: 0.0	float64 w
w: 0.0	geometry_msgs/Vector3 scale
scale:	float64 x
x: 0.006	float64 y
y: 0.006	float64 z
z: 0.006	

Rysunek 3.6. Wycinek informacji zawartych w topicu /svo/points

Modyfikacja

Algorytm SVO posiada wątek użytkownika. Program zmodyfikowano tak, by działał automatycznie, bez czekania na akcję użytkownika, który decydował o starcie algorytmu, przy jego wcześniejszym resecie.

3.7. Stabilizacja w poziomie, Alicja

3.7.1. Fuzja sensoryczna, Alicja

Czujniki wykorzystywane w fuzji, Alicja

Dostępne rozwiązania, Alicja

Realizacja, Hania

Do realizacji fuzji z algorytmu SVO pozyskiwane są informacje o pozycji kamery. Dzięki znanej pozycji początkowej $[x_0, y_0, z_0]$ kamery można obliczyć przesunięcie. Na tej podstawie wysyłana jest wiadomość do sterownika o przesunięciu, uzyskanym z algorytmu wizualnego. Pozycję kamery należy przekonwertować na pozycję quadcoptera przez odpowiednią macierz rotacji (komentarz: jeszcze nie wiem jaką, bo nie wiem, z której kamery w końcu będę korzystać i gdzie ona będzie). Wiadomość wysyłana do sterownika jest umieszczana w topicu o typie *geometry-msg/Vector3*. Parametr δ , który trzeba dobrać podczas testu, będzie decydował czy wystąpiło przesunięcie, czy też nie. Poniższy wydruk pokazuje funkcję rosową *Callback*, która uzyskuje informację o pozycji kamery z topicu /svo/points.

Wydruk 3.3. Uruchomienie kamery

```

1 <launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
3  // output="screen" >
    <param name="video_device" value="/dev/video0" />
5    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
7    <param name="pixel_format" value="mjpeg" />

```

```
9     <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
11 </node>
</launch>
```

3.8. Stabilizacja w pionie, Paweł

3.8.1. Fuzja sensoryczna, Michał

Algorytm stabilizacji wysokości

Dane wejściowe:

- d_1, d_2 – pomiary z dwóch czujników odległości: górnego i dolnego,
- q – orientacja quadcoptera względem zewnętrznego układu odniesienia, w postaci kwaternionu (z Pixhawk),

Dane wyjściowe:

- F_z – siła w osi z quadcoptera.

Kroki algorytmu

1. Obliczenie rzeczywistych odległości od podłoża i sufitu na podstawie odczytów sensorów oraz orientacji quadcoptera względem zewnętrznego układu odniesienia.
2. Fuzja pomiarów – wybranie do stabilizacji odległości, która wolniej się zmienia (mniejsza pochodna). Celem jest uniezależnienie wysokości od niewielkich przeszkód.
3. Regulator PID:
 - wejście: wybrana wysokość h ,
 - wyjście: sterowanie w postaci siły w osi Z .

Czujniki wykorzystywane w fuzji, Paweł

Realizacja, Michał

3.9. Algorytm stabilizacji w punkcie

4. Stabilizacja w pomieszczeniu

4.1. Sterownik nadrzędny, Marcin

4.1.1. Zdarzeniowy sterownik antykolizyjny, Paweł

Analiza wykonalności, Paweł

Specyfikacja, Paweł

4.1.2. Lot wzdłuż ściany, Krzysiek

Analiza wykonalności, Mateusz

Specyfikacja, Mateusz

Część III

Testy

5. Testy stabilizacji

5.1. Testy integracyjne na platformie docelowej

5.1.1. Testy modułu ROS

5.1.2. Testy modułu czujników

Część IV

Zakończenie

6. Podsumowanie

7. Dodatki

Bibliografia

- [1] Ch. Forster, M. Pizzoli, D. Scaramuzza, *SVO: Fast Semi-Direct Monocular Visual Odometry*.
IEEE International Conference on Robotics and Automation (ICRA), 2014,