

# **QUADROCOPTER**

(sterowanie, stabilizacja, autopilot)

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

Politechnika Wrocławska 2014

## Autorzy

*Marcin Ciopcia*

*Michał Drwięga*

*Daniel Gut*

*Alicja Jurasik*

*Agata Leś*

*Kacper Nowosad*

*Piotr Semberecki*

*Hanna Sienkiewicz*

*Mateusz Stachowski*

*Paweł Urbanik*

*Krzysztof Zawada*

Skład raportu wykonano w systemie L<sup>A</sup>T<sub>E</sub>X



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 dugo, 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

<b>1.</b>	<b>Wstęp, Piotrek</b>	5
1.1.	Quadrocopter, Krzysiek	5
1.1.1.	Własności obiektu, Daniel	5
1.2.	Dostępne rozwiązania, Kacper	5
1.3.	Cel pracy, Hania	5

## II. Zadania

<b>2.</b>	<b>Stabilizacja w punkcie - przygotowania, Daniel</b>	8
2.1.	Interfejs do istniejącego oprogramowania i sprzętu, Kacper	8
2.1.1.	Rozpoznanie protokołu quadrocoptera, Daniel	8
2.1.2.	Rozpoznanie możliwości platformy, Daniel	8
2.2.	Zdalne debagowanie, Krzysiek	9
2.2.1.	Dostępne rozwiązania, Mateusz	9
2.2.2.	Link radiowy, Mateusz	9
2.2.3.	Przegląd rozwiązań FPV, Krzysiek	9
2.3.	Czujniki odległości, Alicja	11
2.3.1.	Dobór czujników odległości, Hania	12
2.3.2.	Rozmieszczenie czujników odległości, Paweł	16
2.4.	Algorytmy rozpoznawania przesunięcia, Kacper	17
2.4.1.	Algorytm Lucas—Kanade, Piotrek	17
2.4.2.	Zmodyfikowany PTAM, Kacper	18
2.4.3.	Algorytm SVO, Hania	19
<b>3.</b>	<b>Stabilizacja w punkcie - realizacja, Krzysiek</b>	22
3.1.	Ahsoka, Marcin	22
3.1.1.	Wybór środowiska pracy	22
3.1.2.	Środowisko maszyny wirtualnej	22
3.1.3.	Środowisko uruchomieniowe - Nano 6060	22
3.1.4.	Pixhawk	24
3.1.5.	Administracja maszyną	25
3.1.6.	Paczka mavros	25
3.2.	Uruchomienie platformy latającej, Marcin	26
3.2.1.	Uzbrajanie silników, Marcin	26
3.3.	Uruchomienie kamery 3D, Michał	26
3.4.	Uruchomienie komunikacji, Michał	28
3.4.1.	Komunikacja ROS—kamera, Mateusz	29
3.4.2.	Komunikacja ROS—MAVLINK, Mateusz	29
3.4.3.	Komunikacja ROS—GroundStation, Krzysiek	29
3.4.4.	Komunikacja ROS—czujniki, Daniel	29
3.4.5.	Komunikacja MAVLINK—PixHawk, Mateusz	29
3.5.	Czujniki odległości	29
3.5.1.	Płytki PCB, Alicja	29

3.5.2. Oprogramowanie interfejsu, Michał . . . . .	30
3.5.3. Dekodowanie sygnału PPM, Michał . . . . .	32
3.5.4. Montaż czujników na platformie docelowej, Mateusz . . . . .	33
3.6. Implementacja algorytmów przesunięcia, Kacper . . . . .	33
3.6.1. Algorytm Lucas—Kanade, Piotrek . . . . .	33
3.6.2. Zmodyfikowany PTAM, Kacper . . . . .	33
3.6.3. Algorytm SVO, Hania . . . . .	34
3.7. Stabilizacja w poziomie, Alicja . . . . .	35
3.7.1. Fuzja sensoryczna, Alicja . . . . .	35
3.8. Stabilizacja w pionie, Paweł . . . . .	36
3.8.1. Fuzja sensoryczna, Michał . . . . .	36
3.9. Algorytm stabilizacji w punkcie . . . . .	37
<b>4. Stabilizacja w pomieszczeniu</b> . . . . .	42
4.1. Sterownik nadzędny, Marcin . . . . .	42
4.1.1. Zdarzeniowy sterownik antykolizyjny, Paweł . . . . .	42
4.1.2. Lot wzdłuż ściany, Krzysiek . . . . .	42
<b>III. Testy</b>	
<b>5. Testy stabilizacji</b> . . . . .	44
5.1. Testy integracyjne na platformie docelowej . . . . .	44
5.1.1. Testy modułu ROS . . . . .	44
5.1.2. Testy modułu czujników . . . . .	44
<b>IV. Zakończenie</b>	
<b>6. Podsumowanie</b> . . . . .	46
<b>7. Dodatki</b> . . . . .	47

Część I

Wstęp

# 1. Wstęp, Piotrek

## 1.1. Quadrocopter, Krzysiek

**Quadrocopter** (zwany także quadrotorem) jest to rodzaj wielosilnikowego helikoptera, napędzany czterema wirnikami.

W przeciwieństwie do większości helikopterów, quadrocoptery używają dwóch zestawów śmigieł: dwa o ruch wskazówek zegara (CW) i dwa o ruchu przeciwnym do ruchu wskaźówek zegara (CCW). Różnice momentu obrotowego pozwalają na stabilizację pojazdu podczas lotu jak i zmianę kierunku jego lotu.

W historii lotnictwa projekt quadrocopterów postrzegany był jako możliwość rozwiązania niektórych uporczywych problemów w locie pionowym: problemów sterowania momentem obrotowym wirnika głównego i wydajności wirnika ogonowego. Udane załogowe prototypy powstały w latach 20-tych i 30-tych XX-wieku. Jednak wczesne prototypy jak i późniejsze projekty cierpiały na niską wydajność i wymagały ogromnego doświadczenia i nakładu pracy od pilotów.

W ostatnim czasie bardzo dużą popularność zyskały jednak projekty quadrocopterów bezzałogowych, dzięki wykorzystaniu elektronicznych układów sterowania oraz elektronicznych czujników do stabilizacji pojazdu. Ich przewaga nad innymi pojazdami latającymi to przedewszystkim: małe wymiary, brak mechanicznych połączeń potrzebnych do zmiany kąta łopat, uproszczona konstrukcja, duża stabilizacja w powietrzu. Dodatkowo zastosowanie czterech śmigieł powoduje zmniejszenie ich średnicy, przez co wykazują mniejszą energię kinetyczną podczas lotu. Sprawia to że pojazdy te są bezpieczniejsze dla ścisłej interakcji gdyż szkody wyrządzone przez wirniki powinny być mniejsze.



Rysunek 1.1: Quadrocopter SKN JEDI udostępniony do projektu

### 1.1.1. Własności obiektu, Daniel

## 1.2. Dostępne rozwiązania, Kacper

## 1.3. Cel pracy, Hania

Głównym celem pracy jest wystartowanie w zawodach IMAV 2015 (ang. International Micro Air Vehicles). Aby to umożliwić należało zrealizować zadnia programowe, sprzętowe jak i wdrożyć się w tematykę robotów mobilnych - quadrocopterów. Konkurs IMAV dostarcza wiele wyzwań. Stabilizacja w punkcie, stabilizacja w pomieszczeniu, lot wzdłuż ściany oraz autopilot to główne zadania, których zrealizowanie stwarza w pełni autonomiczny quadrocopter. Ich osiągnięcie zależy od wykonania mniejszych zadań. Ważnym aspektem jest samo sterowanie, które wymaga informacji o aktualnym położeniu i o przesunięciu. Cele postawione do zrealizowania stabilizacji w punkcie to Do celów należą również testy wszystkiego co zostało wytworzzone. ....

**Część II**

**Zadania**

## **2. Stabilizacja w punkcie - przygotowania, Daniel**

### **2.1. Interfejs do istniejącego oprogramowania i sprzętu, Kacper**

Tu będzie opis ...

#### **2.1.1. Rozpoznanie protokołu quadrocoptera, Daniel**

Przygotowanie specyfikacji interfejsu, Piotr

Wybór interfejsu, Piotr

#### **2.1.2. Rozpoznanie możliwości platformy, Daniel**

## 2.2. Zdalne debagowanie, Krzysiek

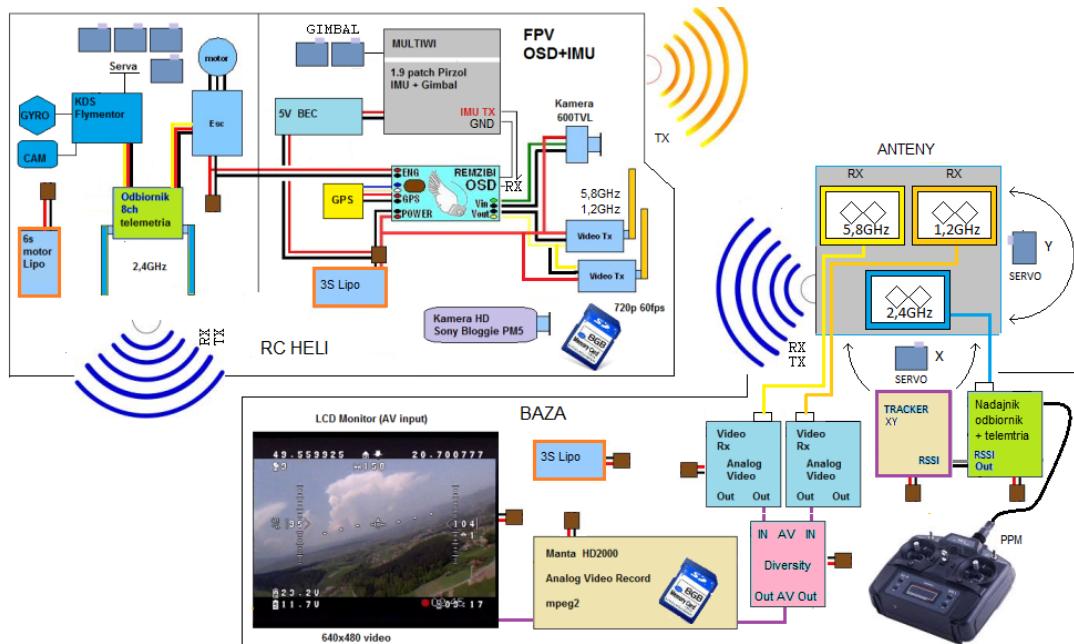
### 2.2.1. Dostępne rozwiązania, Mateusz

Ground Station, Mateusz

### 2.2.2. Link radiowy, Mateusz

### 2.2.3. Przegląd rozwiązań FPV, Krzysiek

Termin FPV (First Person View) można przetłumaczyć jako "Widok z pierwszego planu" (lub też "Widok pierwszoplanowy"). Jest to technika wykorzystywana w modelarstwie, dająca podgląd obrazu z kabiny zdalnie sterowanego pojazdu. Pozwala to na lepszą orientację w terenie, oraz lepsze wykrywanie przeszkód i problemów z pojazdem.



Rysunek 2.1: Schemat podłączenia i działania FPV

System FPV zbudowany jest z:

Kamery analogowej umieszczonej na modelu RC,

Zestawu transmitującego zdalnie obraz z modelu ( zestaw Nadajnik-odbiornik)

Ekrantu pokazującego obraz z kamery na ziemi,

Dodatków pokazujących parametry lotu (OSD)

Każdy z elementów infrastruktury FPV musi być przystosowany do pracy z modelami latającymi. W przypadku budowanego modelu należało postawić przez wybór kilka założeń, jakie powinien spełniać FPV:

Zasięg ok 500m - zasięg i moc nadajnika ma tutaj duże znaczenie. Projektowany quadrocopter poruszać się będzie po otwartej przestrzeni, jak i we wnętrzu budynku. Moc nadajnika powinna być na tyle duża by móc rekompensować straty sygnału wynikające ze betonowych ścian.

mała waga - cały moduł musi być stosunkowo lekki. Na budowanym dronie znajdzie się wiele dodatkowych elementów podnoszących jego wagę, dlatego każdy z nich powinien być możliwie jak najlżejszy.

kamera wykorzystana w module powinna mieć stosunkowo jak najwyższą rozdzielczość.

Jednak w związku z małą przepustowością nadajnika należy unikać wysokich rozdzielczości HD. Wartość, która jest wykorzystywana w większości rozwiązań to rozdzielczość pomiędzy 520 TVL a 650 TVL.

Wykorzystanie gotowych zestawów ? pozwoli to uniknąć problemów przy podłączaniu, konfigurowaniu i obsłudze całego zestawu.

Przegląd rozwiązań FPV rozpoczął się od przejrzenia forów poświęconych tej dziedzinie modelarstwa. Zdobycie tam informacji i porad pozwoliły na wybór jednego gotowego już zestawu **Nadajnik-odbiornik-kamera**.

Główne cechy zestawu:



Rysunek 2.2: Wybrany gotowy zestaw FPV

- Częstotliwość pracy 5,8 GHz
- Moc znamieniowa nadajnika 500mW
- Niewielkie rozmiary nadajnika
- Zastosowanie innej częstotliwości pracy niż stosowane w sterowaniu modelami
- Niewielkie zapotrzebowanie na energię
- Waga wraz z anteną 65,5g
- Kamera CCD 1/3 SONY
- Rozdzielczość 420TVL
- Obiektyw 3,6mm

Wybrany zestaw to jeden z najlepszych zestawów do transmisji video FPV, stosowany również przez profesjonalistów do filmowania z powietrza. Pomimo dużego zasięgu cechuje się odpornością na zakłócenia.

Przesyłany obraz jest w formacie analogowym, dlatego aby móc go oglądać na ziemi potrzebny jest ekran.

W tym celu można używać zwykłego telewizora, ekranu w stacji naziemnej lub laptopa z zainstalowaną kartą telewizyjną. Powyższe sposoby sprawdzają się jednak kiedy korzystamy z FPV jako sprzętu nagrywającego, a dronem latamy utrzymując go w zasięgu wzroku. Kiedy jednak chcemy latać pojazdem po za zasięgiem wzroku, lub podczas lotu



Rysunek 2.3: Przykład obrazu widzianego w goglachFPV FatShark Attitude V2

chcemy obserwować otoczenie z perspektywy quadrocoptera rozwiązań te powodują ciągłą konieczność zerkania na ekran. Pole widzenia zostaje ciągle zakłócone przez otoczenie, a operator musi stać przy ekranie.

W przypadku tego projektu, quadrocopter ma wlecieć do wnętrza budynku. Spowoduje to utratę kontaktu wzrokowego pilota z maszyną. Aby mógł on widzieć obraz z wnętrza budynku a jednocześnie sprawnie pilotować zaproponowano użycie gogli FPV.

Jest to specjalne urządzenie, składające się z ekranu LCD oraz specjalnej soczewki, zamknięte w formie zakładanych na głowę gogli. Używając ich, pilot ma wrażenie jakby stał przed bardzo dużym ekranem, jednocześnie mogąc objąć wzrokiem całą jego powierzchnię.

Wybrany zestaw FPV nie został zakupiony, gdyż okazało się że na potrzeby tego projektu zamontowany zostanie zestaw udostępniony przez Koło Naukowe JEDI.

## 2.3. Czujniki odległości, Alicja

W robotyce mobilnej istnieje konieczność pomiaru odległości. Robot mobilny powinien znać odległość od przeszkody, aby móc ją ominąć i odległość od ściany, aby wzdułż niej podążać. W celu określenia dystansu konstruktorzy używają czujników odległości, które bazują na metodzie pomiaru czasu przelotu od nadajnika do obiektu oraz od obiektu do odbiornika. Nośnikiem energii mogą być: fale radiowe, podczerwień, ultradźwięki lub skupiona wiązka światła (laser).

### 2.3.1. Dobór czujników odległości, Hania

Dobierając czujniki należy uwzględnić środowisko w jakim będą pracować. Mogą być one narażone na takie czynniki jak turbulencje powietrza, zakłócenia akustyczne ze śmiegiel, szумy elektryczne czy wibracje. Każde z tych zjawisk generuje różne problemy, które mogą wpływać na niepożądane działanie czujników. Po specyfikacji potrzeb należy rozoznać się w dostępnych rozwiązaniach na rynku, następnie wybrać najlepsze rozwiązania.



Rysunek 2.4: Gogle FPV Fatshark Dominator

### Specyfikacja potrzeb, Hania

Specyfikacja potrzeb powinna uwzględnić zjawiska fizyczne, zakłócające pracę czujników jak i zadanie wykonywane przez quadrocopter.

#### Turbulencje powietrza

Turbulencje powietrza, wywoływanie przez śmigła, wpływają na zmniejszenie energii otrzymanego sygnału przez czujnika, mogą zmieniać kierunek oraz intensywność fal akustycznych. Można temu zapobiec umieszczając sensory jak najdalej od śmigieł. Przy pomiarze odległości do ziemi należy umieścić czujnik pod ramą jak najbliżej centrum.

#### Szum akustyczny wytwarzany przez śmigła

To zjawisko jest bardzo podobne do opisanego powyżej. Zmienia energię sygnału, który ma odebrać czujnik, szумy akustyczne ze śmigieł są również odbierane jako sygnały powracające od czujników. Zakłócenia te są kumulowane na zakończeniach śmigieł. W takim przypadku najlepiej unikać montażu czujnika w miejscach, w których czujnik jest skierowany bezpośrednio na śmigło. Dla najlepszego gąbkę montażową może być używany do blokowania takiego ścieżki, użytkownik może podłączyć czujnika pod elektroniki locie lub kombinacja tych dwóch może być używany.

#### Masa i napięcie zasilania

Najlepszym rozwiązaniem połączania czujników z masą i napięciem zasilającym jest połączenie typu gwiazda. Każdy czujnik ma osobne przewody zasilające. Zapobiega to zakłóceniom powodowanym przez podłączenie urządzeń o różnych woltarzach.

#### Szumy elektryczne generowane przez fale radiowe

Szumy elektryczne generowane przez fale radiowe mogą powodować niepoprawne odczyty przez czujniki ultradźwiękowe. Gdy poprawnie zostanie użyty filtr zasilacza i skorzysta się z takich interfejsów jak np. I2C, odczyty odległości nie powinien zostać uszkodzony przez zewnętrzne zakłócenia elektryczne. Należy użyć ekranowanych przewodów. Ekran



Rysunek 2.5: skaner



Rysunek 2.6: SHARP

ten musi być prawidłowo uziemiony do masy. Gdy ekran nie zostanie podłączony z masą, szумy mogą być jeszcze większe od tych, które występują przy braku ekranowanych kabli.

### Wibracje ramy

Drgania ramy mogą również powodować zakłócenia. Energia z ramki może być transmitowana do czujnika. W celu wyeliminowania tego hałasu można zastosować podkładki gumowe, taśmy z pianki lub inne materiały tłumiące wibracje.

### Dobranie czujników ze względu na zasięg

Zakresy czujników skierowanych pionowo w dół oraz w góre nie powinny przekraczać w sumie  $3m$ , ponieważ zakładamy lot w pomieszczeniu na wysokość oczu człowieka. Strefa martwa czujników umieszczonych w poziomie może wynosić  $40cm$ , dlatego, że lot wzdłuż ściany, przez tworzącą się poduszkę powietrzną będzie odbywał się w odległości około  $40cm$ .

### Dostępne rozwiązania, Alicja

Na podstawie specyfikacji dokonano rozeznania dostępnych na rynku czujników odległości. Odrzucono sensory cyfrowe ze względu na to, że nie zwracają informacji o odległości do obiektu. Wybierając czujniki analogowe kierowano się prędkością z jaką dokonują pomiaru. Dodatkowo podczas selekcji czujników istotny był zakres prawidłowego działania, cena oraz waga. Wyniki rozeznania dostępnych sensorów przedstawiono w tabeli 2.1. (((((obrazki miały być ułożone ładnie w jednym wierszu, ale w raporcie zbiorczym nie chce się to kompilować))))))

### Dobór czujników, Alicja

Na podstawie specyfikacji i rozeznania rynkowego dobrano czujniki odległości wykorzystane w quadrocopterze. Zakupiono 7 czujników SHARP GP2Y0A02YK0F oraz 2 sonary ultradźwiękowe MAXSONAR-EZL0. W tabeli 2.2 przedstawiono dane dotyczące wykorzystanych czujników.



Rysunek 2.7: SHARP



Rysunek 2.8: sonar



Rysunek 2.9: sonar

Tabela 2.1: Zestawienie dostępnych na rynku czujników spełniających założenia

nośnik energii	czujnik	cena [zł]	zasięg [m]	waga [g]	zdjęcie
laser	skaner laserowy RPLidar	1870	6	-	2.5
podczerwień	Sharp GP2Y0A02YK0F	59	0,2 - 1,5	4,8	2.6
podczerwień	Sharp GP2Y0A710K0F	139	1 - 5,5	10	2.7
ultradźwięki	US-020	16	0,02 - 7	-	2.8
ultradźwięki	XL-MaxSonar-EZL0	169	10	8	2.9

Tabela 2.2: Zestawienie danych wybranych czujników

	SHARP GP2Y0A02YK0F	Ultradźwiękowy MAXSONAR-EZL0 [MB1260]
Zakres pomiarowy	20 – 150cm	0 – 10m
Napięcie zasilania	4,5 – 5,5V	3,3 – 5,5V
Średni pobór prądu	33mA	3.4mA
Czas odpowiedzi	38ms	30ms
Wymiary	29,5 x 13,0 x 21,5 mm	22,1 x 19,9 x 25,11

### 2.3.2. Rozmieszczenie czujników odległości, Paweł

W trakcie planowania rozmieszczenia czujników odległości zostały wzięte pod uwagę następujące aspekty:

- minimalizacja ilości wykorzystanych czujników przy maksymalizacji dostarczanych przez nie informacji,
- zróżnicowanie rodzajów czujników (różne zakresy działania),
- możliwość obsługi wykorzystanych sensorów (wbudowane interfejsy).

#### Przegląd rozwiązań, Paweł

Poniżej wymienione zostały popularne projekty opisujące budowę quadrocoptera, posiadające dobrą dokumentację opartą na licencji opensource:

- <http://www.armokopter.at/>  
Analog MEMS gyroscopes, Analog MEMS accelerometers, digital motion sensor (MPU6050) + baro sensors + compass sensors
- <http://www.openpilot.org/>  
3-axis high-performance MEMS gyroscope, 3-axis high-performance MEMS accelerometer
- <http://code.google.com/p/arducopter/>  
Triple Axis magnetometer
- <http://aeroquad.com/>
- <http://ng.uavp.ch/>

W tych pracach nie występują czujniki odległości ułożone w płaszczyźnie wertykalnej, które byłyby wykorzystywane do stabilizacji w poziomej płaszczyźnie. Funkcję awaryjnego STOP pełnią zazwyczaj obręcze śmigieł, lub piankowe korpusy quadrocopterów. Jest to jednak tylko zabezpieczenie przed uszkodzeniem sprzętu, które w żaden sposób nie spełnia wymagań projektu.

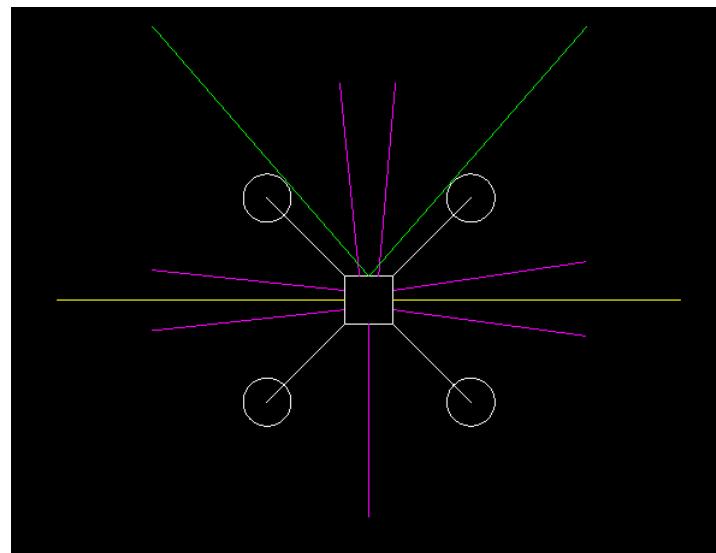
Taki rodzaj czujników (odległościowe czujniki odbiciowe) spotyka się w układach wspomagających manewry lądowania (skierowane w dół, pomagają wyliczać wektory aktualnej prędkości opadania).

#### Propozycja rozmieszczenia, Paweł

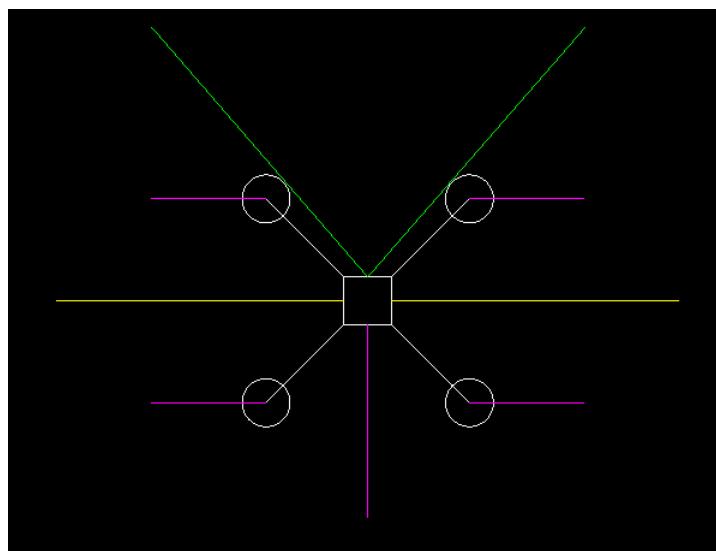
Zostały zaproponowane dwie konfiguracje przedstawione na rys. 2.10 oraz 2.11. Kolorem fioletowym zostały oznaczone czujniki bliskiego zasięgu — SHARP GP2Y0A02YK0F, żółtym sonary ultradźwiękowe — XL-MaxSonar-EZL0 [MB1260], natomiast zielony kolor to kąt i zasięg widzenia kamery 3D.

Ze względu na optymalizację kosztów oraz wagi układów sensorycznych ostatecznie zdecydowano się na rozwiązanie II. Dodatkowym aspektem wynikającym z przyjętej konfiguracji jest trywialny sposób kontroli równoległego lotu wzduż ściany, który polega na bezpośrednim porównywaniu odczytów z czujników znajdujących się po tej samej stronie quadrocoptera. Czujniki dalekiego zasięgu zostały zainstalowane ze względu na potrzebę podejmowania decyzji wzduż której ściany powinien poruszać się quadrocopter zaraz po wlecieniu do nieznanego pomieszczenia.

Do stabilizacji w pionie zostaną wykorzystane dwa czujniki typu SHARP skierowane prostopadle w dół (do ziemi) i w górę (w stronę sufitu). Zbierane pomiary zostaną uśrednione i wybrany do sterowania ten, który zmienia się wolniej (jego pochodna jest mniej-



Rysunek 2.10: Konfiguracja I



Rysunek 2.11: Konfiguracja II

sza). Dzięki zainstalowanym na platformie czujnikom inercyjnym wyliczanie prawdziwej wysokości, na której znajduje się quadrocopter, odbywa się będzie z wykorzystaniem bezpośrednich odczytów SHARP'a oraz aktualnymi wychyleniami platformy.

## 2.4. Algorytmy rozpoznawania przesunięcia, Kacper

Jednym z ważnych zadań niniejszego projektu, była detekcja przesunięcia, ruchu quadrocoptera, która powinna być dokonywana automatycznie przy wykorzystaniu sekwencji obrazów z kamery wizyjnej. Ponieważ projekt opiera się na platformie ROS założono, że istnieją już gotowe rozwiązania problemu detekcji ruchu kamery i dokonano przeglądu dostępnych algorytmów. Do implementacji i badań wybrano algorytm Lucas–Kanade oraz dwa algorytmy monoSLAM PTAM i SVO

### 2.4.1. Algorytm Lucas—Kanade, Piotrek

Jednym z podejść do wykrywania ruchu na obrazie z kamery jest wykorzystanie tzw. „Optical Flow” („przepływ optyczny”) [?], czyli wzorca ruchu, który dla obrazów 2D jest względną prędkością wyznaczoną dla wybranych pikseli na dwóch zdjęciach. W przypadku tworzonej aplikacji, dla dwóch sąsiednich klatek nagrania. Wydaje się to najbardziej naturalny sposób podejścia do problemu. Optical Flow jest określony poprzez dwa podstawowe założenia:

1. Jasności pikseli nie zmienia się pomiędzy dwoma klatkami,
2. Sąsiednie piksele są przesunięte o tą samą odległość ( $\Delta x, \Delta y$ ):

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \quad (2.1)$$

Zakładając, że to przesunięcie jest małe, prawą stronę równania rozwija się w szereg Taylora i otrzymuje:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t + H.O.T., \quad (2.2)$$

gdzie H.O.T. (ang. *higher-order terms*) to czynniki coraz wyższych rzędów, które mogą zostać pominięte. Z tego otrzymuje się następujące równania:

$$\frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t = 0, \quad (2.3)$$

lub

$$\frac{\partial I}{\partial x} \frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y} \frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} \frac{\Delta t}{\Delta t} = 0, \quad (2.4)$$

co w rezultacie daje:

$$\frac{\partial I}{\partial x} V_x + \frac{\partial I}{\partial y} V_y + \frac{\partial I}{\partial t} = 0, \quad (2.5)$$

gdzie  $V_x, V_y$  to składowe  $x$  i  $y$  prędkości albo inaczej przepływ optyczny  $I(x, y, t)$ , natomiast  $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$  i  $\frac{\partial I}{\partial t}$  to pochodne cząstkowe obrazu w  $(x, y, t)$ . Inaczej można to przedstawić jako:

$$I_x V_x + I_y V_y = -I_t. \quad (2.6)$$

W równaniu tym są jednak dwie niewiadome ( $V_x$  i  $V_y$ ), co uniemożliwia jego rozwiązańe w sposób dokładny i jednoznaczny. Jedną z metod, która przybliża jego rozwiązanie, to algorytm Lucas'a-Kanade [?]. Metoda ta opiera się na założeniu, że przesunięcie zawartości obrazu między dwoma ramkami jest minimalne i zakłada, że sąsiedztwo piksela (9 znajdujących się dookoła pikseli) ma podobnych ruch. Wobec tego jest rozwiązywanie 9. równań z dwiema zmiennymi, czyli układ jest nadokreślony:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1} I_x(q_i)^2 & \sum_{i=1} I_x(q_i)I_y(q_i) \\ \sum_{i=1} I_y(q_i)I_x(q_i) & \sum_{i=1} I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1} I_x(q_i)I_t(q_i) \\ \sum_{i=1} I_y(q_i)I_t(q_i) \end{bmatrix},$$

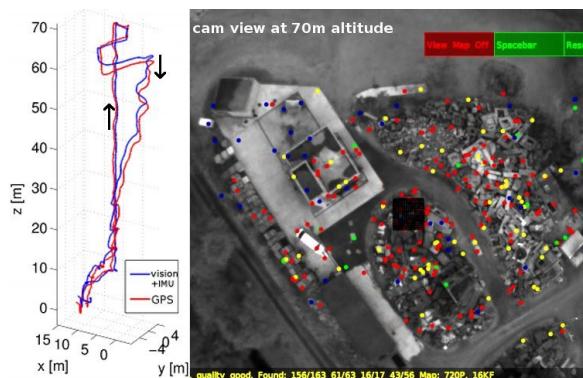
gdzie  $i=1$  do 9. Algorytm estymuje rozwiązanie poprzez wykorzystanie metody najmniejszych kwadratów [?].

#### 2.4.2. Zmodyfikowany PTAM, Kacper

##### Algorytm PTAM - Parallel Tracking and Mapping

PTAM jest algorytmem rozwiązującym to samo zadanie, co algorytm monoSLAM, zaproponowanym w 2007 roku przez George Klein. Podejście to wykorzystuje jedną kamerę 6-DOF do śledzenia ruchu kamery i mapowania otoczenia tworząc chmurę punktów **2.12**. Algorytm ten wyróżnia się spośród innych algorytmów monoSLAM, tym że zadanie śledzenia ruchu i mapowania jest realizowane w czasie rzeczywistym. Różnicę tej metody można podsumować w kilku punktach:

- śledzenie i mapowanie odbywa się w dwóch równoległych wątkach,
- mapowanie jest oparte na klatkach kluczowych, które są przetwarzane przy użyciu metod wsadowych (Bundle Adjustment),
- mapa jest gęsto zainicjowana przez pary punktów stereo, gdzie wykorzystano algorytm pięciu punktów,
- nowe punkty są inicjowane przez wyszukiwanie epipolarne.



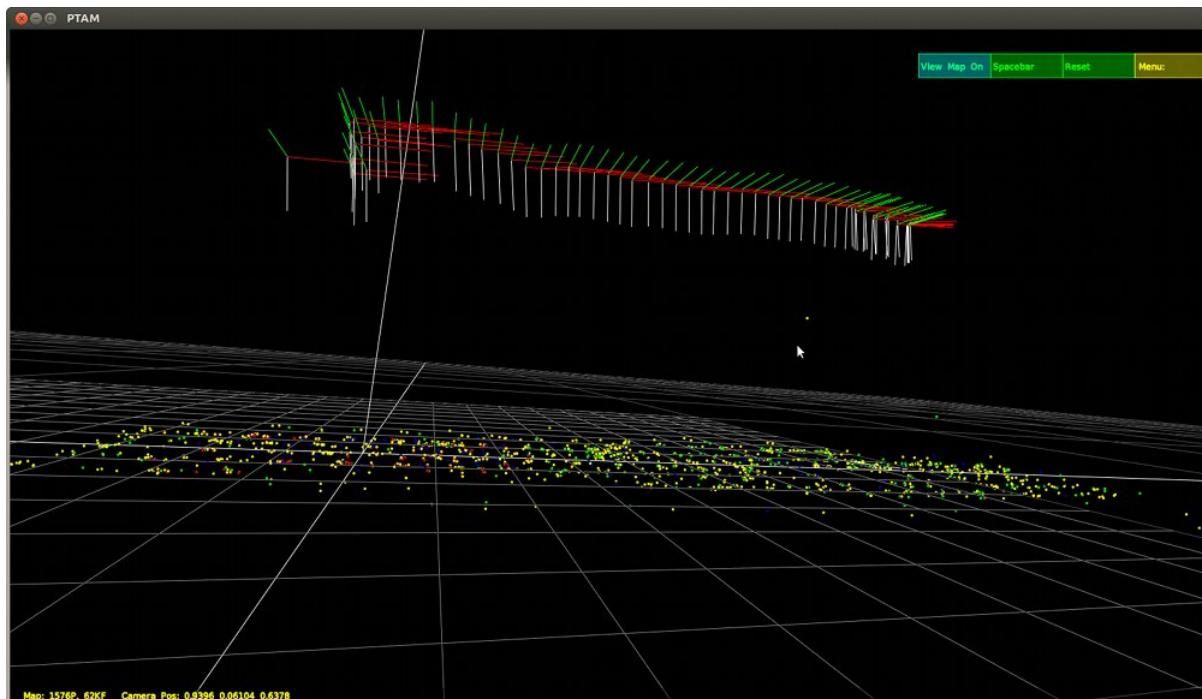
Rysunek 2.12: PTAM - Parallel Tracking and Mapping

Algorytm PTAM wyszczególnia dwa procesy: śledzenia i mapowania **2.13**. Proces śledzenia przemieszczenia kamery, przy założeniu, że posiadamy mapę punktów 3D, odbywa w następujących krokach:

- pobierana jest nowa ramka z aparatu z wyestymowaną wcześniejszą pozycją kamery z modelu przemieszczenia,
- mapa punktów jest przewidywana przez podążanie za wcześniej wyestymowaną pozycją ramki,
- mała liczba (ok. 20-50) punktów jest wyszukiwana przy dużym promieniu na zdjęciu (do tych punktów odnosić się będzie przesunięcie kamery)

- następnie aktualizowana jest pozycja kamery, która dopasowana jest według tych punktów,
- punkty są odwzorowywane na zdjęciu i następuje wyszukiwanie większej ilości punktów (ok. 1000),
- finalnie wyestymowana pozycja kamery jest obliczana z wszystkich zebranych punktów.

Zadanie mapowania odbywa się w dwóch etapach. Po pierwsze jest budowana wstępna mapa przy wykorzystaniu technik stereo. Następnie mapa ta jest ciągle wzbogacana i udoskonalana przez otrzymywanie chmur punktów z nowych klatek kluczowych oddawanych przez system śledzenia.



Rysunek 2.13: PTAM - śledzenie i mapowanie

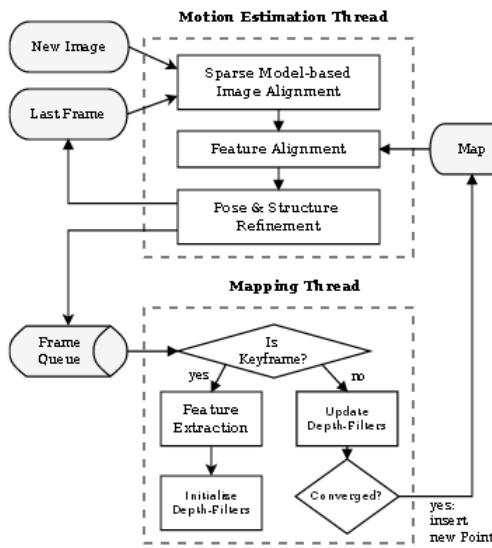
#### 2.4.3. Algorytm SVO, Hania

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, quadrocopter 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.



Rysunek 2.14: Schemat algorytmu SVO [?]

### Ogólny schemat algorytmu SVO

Program SVO (Semi-direct Monocular Visual Odometry) [?] jest zaimplementowany w ROSie. Wykorzystuje on obrazy dostarczone z pokładowej kamery. Quadrocopter zbiera informacje ze środowiska poprzez zdjęcie 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

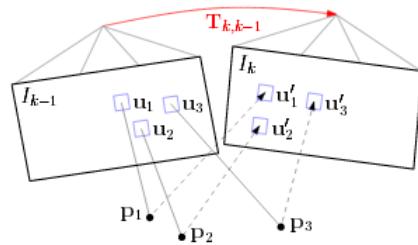
$$C_k = C_{k-1} \cdot A_k,$$

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 2.14 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.

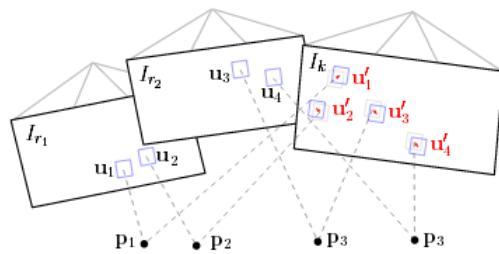
### Etapy algorytmu SVO

Estymację ruchu/położenia kamery można podzielić na trzy etapy

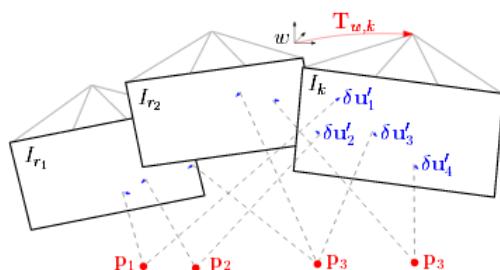
- etap pierwszy Rysunek 2.15 przedstawia zmianę pozycji między wcześniejszą i obecną ramką danych pośrednio przenosi pozycję  $p$  na odwzorowanie punktów w nowym obrazie. Algorytm minimalizuje fotometryczny błąd poprawki dotyczącej tych samych 3D punktów  $p$ .
- drugi etap Ze względu na niedokładności 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 2.16.



Rysunek 2.15: pierwszy etap [?]



Rysunek 2.16: drugi etap [?]



Rysunek 2.17: trzeci etap [?]

- trzeci etap Rysunek 2.17 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.

### **3. Stabilizacja w punkcie - realizacja, Krzysiek**

#### **3.1. Ahsoka, Marcin**

##### **3.1.1. Wybór środowiska pracy**

Do realizacji sterowania wielowirnikowcem wymagane jest środowisko, które spełnia szereg wymagań, m. in.:

1. Jest niezawodne.
2. Nie wymaga zbyt dużych zasobów sprzętowych.
3. Jest łatwe w konfiguracji.
4. Zapewnia wsparcie narzędziom wykorzystywanym w projekcie.

Dlatego też zdecydowano, aby projekt był realizowany przy użyciu środowiska Linux. Ponieważ najpopularniejszą dystrybucją jest w tej chwili Ubuntu Server, co implikuje dużą ilość dobrze napisanej dokumentacji/poradników, dobre wsparcie techniczne, bogactwo uruchomionych i gruntownie przetestowanych narzędzi - wybór ten okazał się najlepszą z rozpatrywanych opcji.

Pomimo początkowej chęci pracy na aktualnym wydaniu LTS, tzn 14.04.1, o wiele lepszą alternatywą okazała się wersja 12.04.5 LTS - ze względu na fakt, iż ROS Hydro, dedykowany na tę platformę zawiera więcej przydatnych modułów, które w wypaku użycia nowszej wersji, wymagałyby portowania.

Do realizacji projektu wykorzystano Ubuntu 12.04.5 wraz z ROS Hydro. Ze względu na problem z cyfrowym podpisem paczek dla polskiego repozytorium, użyto wersji angielskiej.

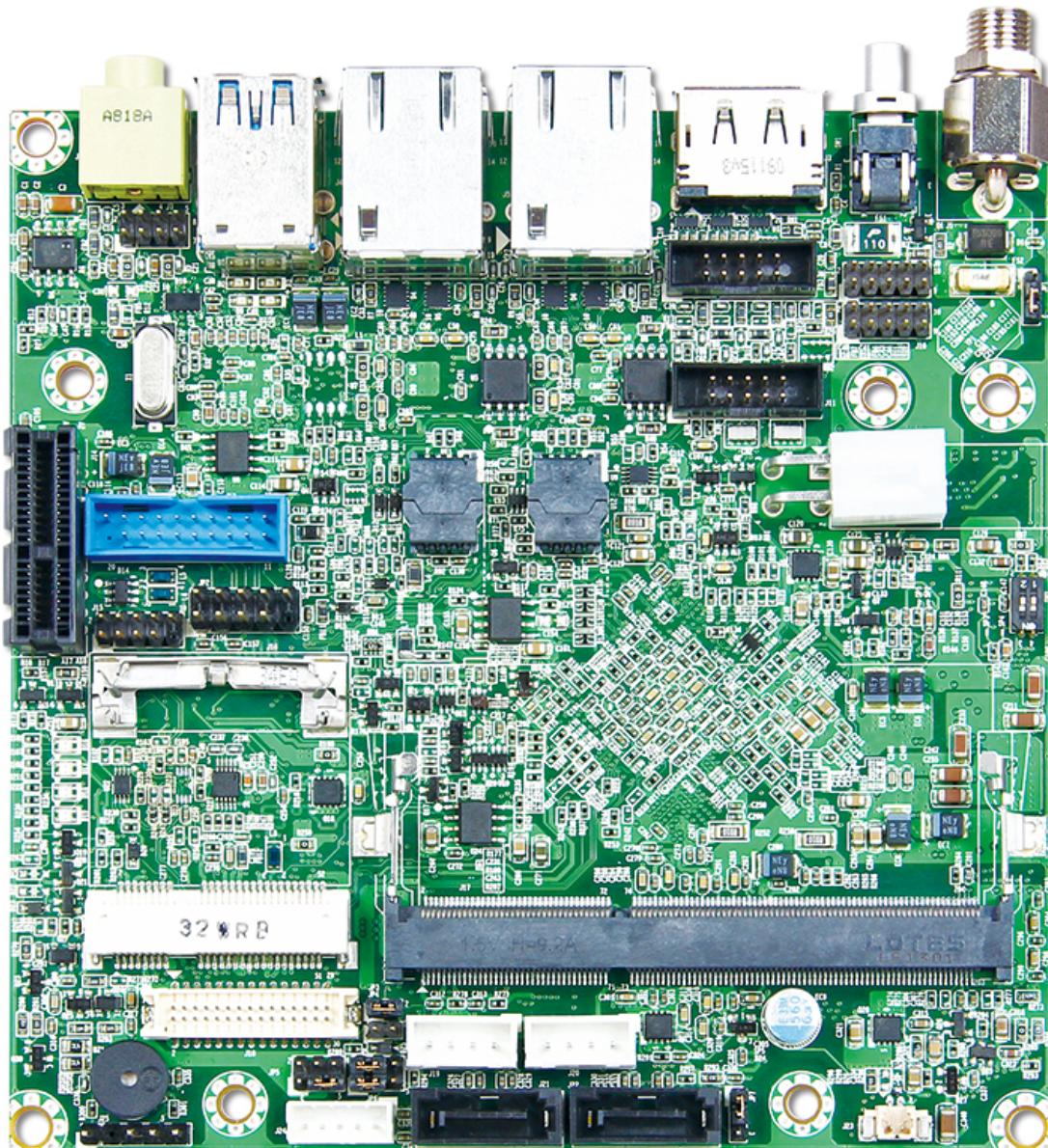
##### **3.1.2. Środowisko maszyny wirtualnej**

Do celów zapoznania się z środowiskiem ROS stworzono, przy użyciu porogramowania Virtualbox, środowisko typu sandbox. We wczesnych etapach projektów służyło ono do bezpiecznego eksperymentowania ze środowiskiem. Dzięki faktowi, iż była to maszyna wirtualna - przeinstalowanie całego systemu było banalną operacją. Pnadto na maszynie wirtualnej każdy z użytkowników miał prawa administratora, przez co mógł testować oprogramowanie przed docelową instalacją na platformie latającej.

##### **3.1.3. Środowisko uruchomieniowe - Nano 6060**

Jako platformę docelową wybrano komputer x86 typu Small-Form-Factor. Ze względu na mierną, w dniu dzisiejszym, kompatybilność systemu ROS z platformą ARM - po rozpoznaniu możliwości, zarzucono ten pomysł.

Wybór padł na produkt firmy Portwell, komputer Nano 6060 [3.1]. Jest to zwarta, kompaktowa konstrukcja (format mini-ITX), przez co nadaje się do montażu na obiektach latających.



Rysunek 3.1: Fotografia użytego komputera pokładowego - Nano 6060



Rysunek 3.2: Fotografia użytego kontrolera lotu - Pixhawk

Kluczowe parametry zastosowanego komputera:

1. Procesor Intel Atom E3845 - 4x 1.91 GHz
  2. 4 Gb pamięci RAM DDR3 .
  3. Dysk SSD.
  4. Złącze mini-PCIe - posłużyło do montażu karty sieciowej Atheros AR9285
  5. Zasilanie 12V
- Komputer jest zasilany z baterii robota za pomocą przetwornicy step-down.

### 3.1.4. Pixhawk

Jako kontroler lotu użyto komputera pokładowego Pixhawk [3.2]. Dzięki obsłudze protokołu mavlink oraz zaimplementowanym procedurom do sterownika nadzawanego stanowił on rozsądny wybór.

Kontroler jest podpięty do komputera pokładowego za pomocą złącza USB - co pozwala na administrowanie systemem Pixhawka, oraz poprzez przejścia UART - USB - służącą do czytania danych telemetrycznych.

Do kontrolera popięty jest także GPS oraz system kontroli stanu zasilania.

### 3.1.5. Administracja maszyną

#### Wymagane oprogramowanie

Aby uruchomić projekt, oprócz zainstalowania systemu, należy zainstalować następujące paczki:

- build-essential
- dselect
- geany
- geany-common
- geany-plugins
- geany-plugins-common
- git
- guvcview
- h264enc
- libcanberra-gtk-module
- libgtk2.0-dev
- liblapack-dev
- live-config-systemd
- openssh-blacklist
- openssh-blacklist-extra
- openssl-blacklist
- openssl-blacklist-extra
- python-dev
- python-pip
- python-rosdep
- python-rosinstall
- python-software-properties
- ros-hydro-camera-calibration
- ros-hydro-cmake-modules
- ros-hydro-desktop-full
- ros-hydro-gscam
- ros-hydro-mavlink
- ros-hydro-mavros
- ros-hydro-mavros-extras
- ros-hydro-usb-cam
- screen
- wireless-tools
- wpasupplicant

#### Uprawnienia użytkownika

Aby usprawnić pracę nad projektem zwykli użytkownicy współdzielili konto użytkownika *start*. Ma on uprawnienia jedynie do modyfikowania workspace'a programu catkin oraz pełni dostęp do wymaganych peryferów, gdyż należy do grup: *dialog* oraz *video*. Oprócz tego na maszynie istnieje konto administratora - *tano*.

### 3.1.6. Paczka mavros

Do obsługi komunikacji pomiędzy Pihawkiem a systemem ROS wykorzystano moduł *mavros*. Uruchamia ona topic */mavros/* oraz zapewnia łącze dla programu QGroundCon-

trol za pomocą protokołu UDP. Dzięki tej ostatniej funkcjonalności możliwe jest zarządzanie robotem za pomocą łącza WiFi, bądź ethernet.

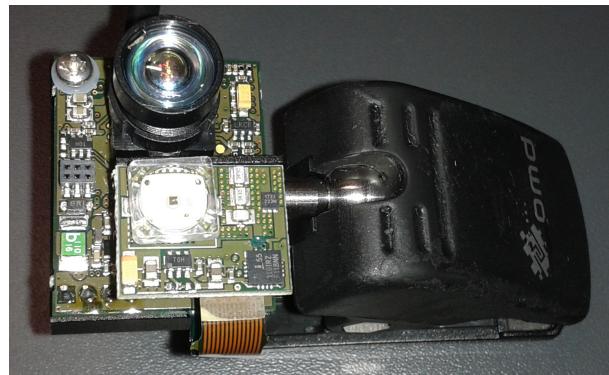
## 3.2. Uruchomienie platformy latającej, Marcin

### 3.2.1. Uzbrajanie 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 ???. Kamera ta wykorzystuje sensor *PMD PhotonICs 19k-S3*, a jej podstawowe parametry są następujące:

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

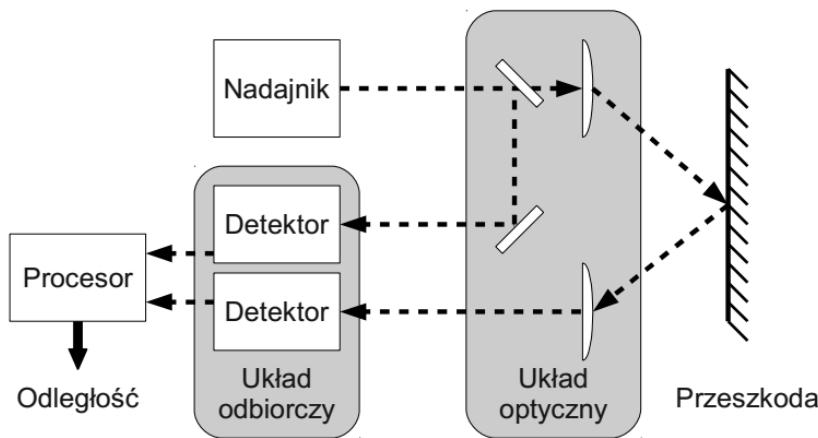


Rysunek 3.3: Wykorzystywana kamera 3D PMD CamBoard nano

Przedstawiona kamera wykorzystuje metodę pomiaru określana mianem *TOF (Time Of Flight)*, czyli mierzenia czasu przelotu impulsu. Budowa takiej kamery w postaci blokowej została przedstawiona na rysunku 3.4. Odnośnie 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.

### Portowanie pakietu *pmd\_camboard\_nano*

W celu zapewnienia dostępu do obrazów kamery 3D z poziomu środowiska *ROS*, które umożliwia integrację tworzonego w ramach projektu oprogramowania zdecydowano się na



Rysunek 3.4: Budowa i zasada działania kamery TOF

przeniesienie pakietu *pmd\_cambord\_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 rozwiązań odnośnie budowania paczek. W poprzedniej wersji pakietu używano *rosbuild*, 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 ujednolicenia 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\_cambord\_nano*. Wykorzystane zostały sterowniki w wersji 1.3.2.

### Wybrane topiki publikowane przez pakiet *pmd\_cambord\_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\_cambord\_nano pmd\_cambord\_nano.launch* - uruchomienie pakietu,
- rosrun *image\_view image\_view image:=/camera/depth/image*
  - wyświetlenie obrazu głębi,
- rosrun *image\_view image\_view image:=/camera/amplitude/image*
  - wyświetlenie obrazu w odcieniach szarości.

### 3.4. Uruchomienie komunikacji, Michał

Komunikacja ze sterownikiem quadrocoptera 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, przewidziane zostały w nim funkcjonalności pozwalające na dodawanie własnych wtyczek (pluginów). Dokładny opis jest dostępny pod następującym odnośnikiem: <http://wiki.ros.org/mavros>. Poniżej wymienione zostały wybrane funkcjonalności pakietu **mavros** w przypadku załączania domyślnego zestawu pluginów.

- Publikowanie topic'ów zawierających dane z IMU (plugin **imu\_pub**), między innymi:
  - orientacja obliczona przez sterownik ( $\sim imu/data$ ),
  - wartości zmierzane przez IMU ( $\sim imu/data_raw$ ),
  - odczyty magnetometru ( $\sim imu/mag$ ),
  - temperatura ( $\sim imu/temperature$ ),
  - ciśnienie powietrza ( $\sim imu/FluidPressure$ ).
- Subskrybowanie topic'ów z komendami lotu:
  - zadane prędkości ( $\sim setpoint/cmd\_vel$ ),
  - zadane przyspieszenia ( $\sim setpoint/accel$ ),
  - zadane prędkości kątowe ( $\sim setpoint/att\_vel$ ),
  - zadana poza ( $\sim setpoint/attitude$ ).

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

Tworzenie platformy, która będzie mogła poruszać się autonomicznie powoduje konieczność otrzymywania od niej pełnego wachlarza informacji o jej położeniu. Warto w tym celu wykorzystać GroundStation, czyli stację naziemną.

Jest to centrum sterowania bezzałogowych statków powietrznych, pozwalające na komunikację z platformą poprzez terminal. Pomimo że platforma porusza się autonomicznie to należy podać jej współrzędne po których może (lub ma) się poruszać.

Dane telemetryczne potrzebne do komunikacji ze stacją naziemną pochodzą bezpośrednio ze sterownika PixHawk. Posłuży do tego komunikacja z systemem operacyjnym

ROS zaimplementowanym na platformie Ahsoka.

Groundstation jest to GUI oparte na GTK+, co pozwala na wyświetlanie i aktualizację danych dostarczonych bezpośrednio przez topic?i w ROS?ie. Dane mogą być wyświetlane przez jeden z programów do zdalnej telemetrii takich jak: **CityFlyer** czy **QGroundControl**. Ze względu na lepszą integrację z ROSem do projektu wybrany został QGroundControl. Dodatkowo program ten umożliwia pełną symulację lotu bez uruchamiania pojazdu.

Komunikację z ROS-GroundStatnion uzyskano dzięki gotowym poradnikom i biblioteką MAVLink. Dzięki temu uzyskano połączenie platformy latającej z QGroundControl. Połączenie następuje poprzez sieć WI-FI.

#### **3.4.4. Komunikacja ROS—czujniki, Daniel**

#### **3.4.5. Komunikacja MAVLINK—PixHawk, Mateusz**

### **3.5. Czujniki odległości**

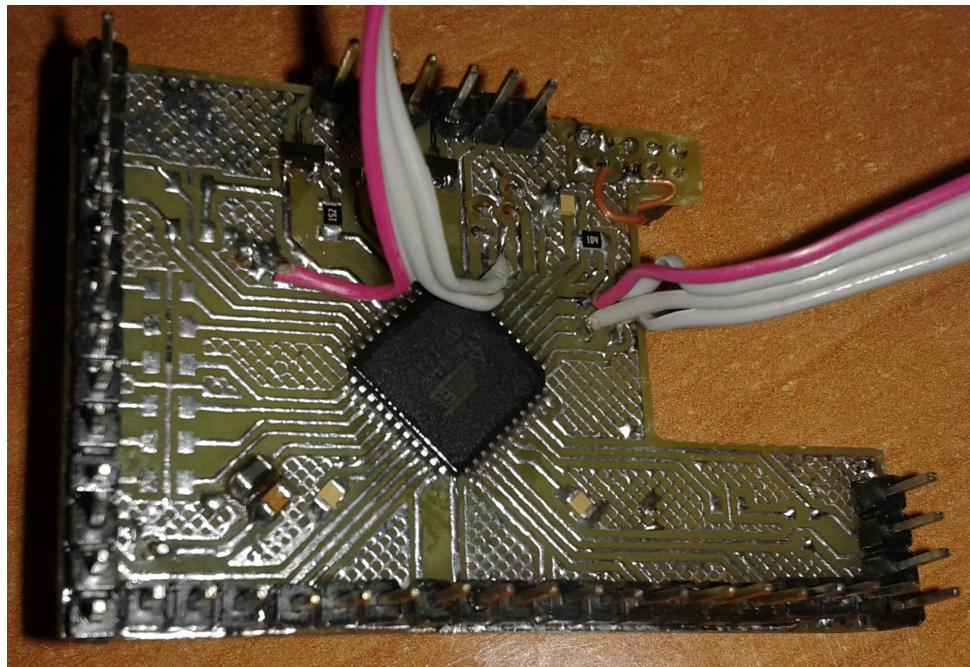
#### **3.5.1. Płytkę PCB, Alicja**

W celu poprawnego uruchomienia czujników odległości trzeba zaprojektować i wykonać płytke PCB. Płytkę powinna obsługiwać sygnały wejściowe z czujników, odpowiednio je przetworzać oraz publikować.

#### **Specyfikacja potrzeb, Marcin**

#### **Wykonanie i uruchomienie płytki PCB, Michał**

Na rysunku 3.5 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 3.5: Przedstawienie wykonanego interfejsu

### 3.5.2. Oprogramowanie interfejsu, Michał

Urządzenie spełnia rolę interfejsu sterującego wybranymi podzespołami quadrocoptera oraz jednostki pomiarowej, mierzącej odległość 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 nadziednego (UART),
- sterowanie oświetleniem quadrocoptera,
- 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 nadziednym

W celu umożliwienia komunikacji urządzenia z komputerem nadziednym 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 nadziednym, 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 odebranym 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 3.6.

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

Rysunek 3.6: 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 odczytyanych 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  $\mu s$ . 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  $\mu s$  w sygnale PPM dla kanału nr1,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr2,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr3,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr4,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr5,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr6,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr7,
- 2 bajty - szerokość impulsu w  $\mu s$  w sygnale PPM dla kanału nr8,

## Rozkazy konfiguracyjne - 0xD

### 1101 0000 - 0xD0

Polecenie uruchomienia oświetlenia quadrocoptera.

### 1101 0001 - 0xD1

Polecenie wyłączenia oświetlenia quadrocoptera.

### 1101 0010 - 0xD2

Włączenie Pixhawk Failsafe (ustawienie stanu wysokiego).

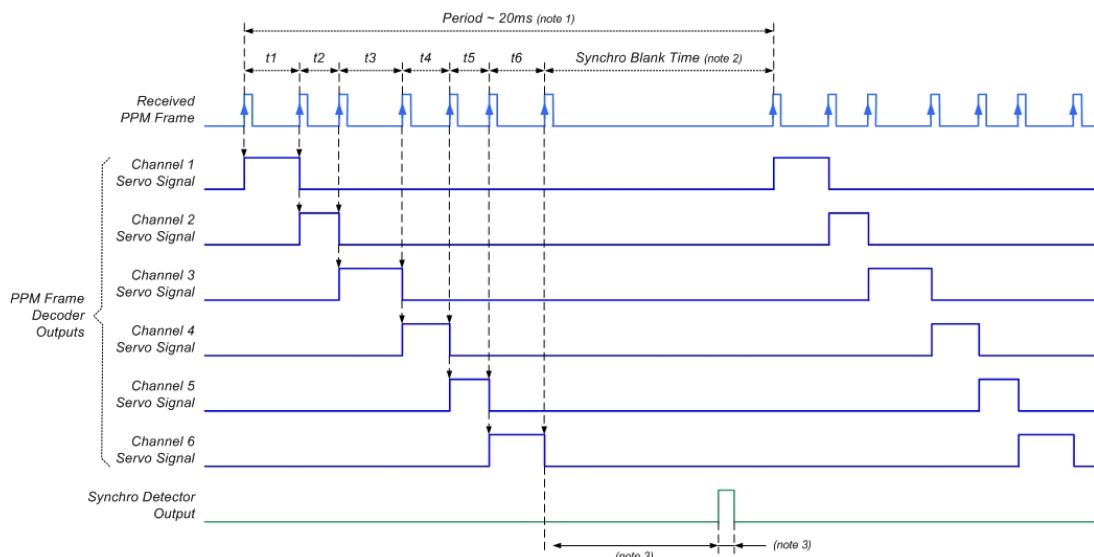
### 1101 0011 - 0xD3

Wyłączenie Pixhawk Failsafe (ustawienie stanu niskiego).

### 3.5.3. Dekodowanie sygnału PPM, Michał

W celu umożliwienia ręcznego sterowania quadrocopterem 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 ??.



Rysunek 3.7: Sposób kodowania informacji w sygnale PPM

Po zdekodowaniu sygnału PPM, wartości o poszczególnych kanałach wysyłane są do komputera nadziedznego *nanode*, gdzie przetwarzane są w środowisku *ROS* i następnie wysyłane do sterownika quadrocoptera 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)
{
3   cli();
5   if (MCUCR & _BV(ISC2)) // rising edge interrupt
7   {
9     TCNT2 = 0; // Timer2 reset
11    timeCntPPM = 0;
13    MCUCR &= ~_BV(ISC2); // change to falling edge detect
15  }
17  else // falling edge interrupt
19  {
21    if ((timeCntPPM + TCNT2) > MAX_PPM_WIDTH) // new frame
23    {
25      channelPPMcnt = 0;
27    }
29    else // measurement for PPM channel
31    {
33      channelPPM[channelPPMcnt] = timeCntPPM + TCNT2;
35      if (channelPPMcnt < 7) // increase channel counter
37      {
39        channelPPMcnt++;
41      }
43    }
45    GIFR |= (1 << INTF1); // interrupt flag clear
47    sei();
49  }
51}

```

## 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

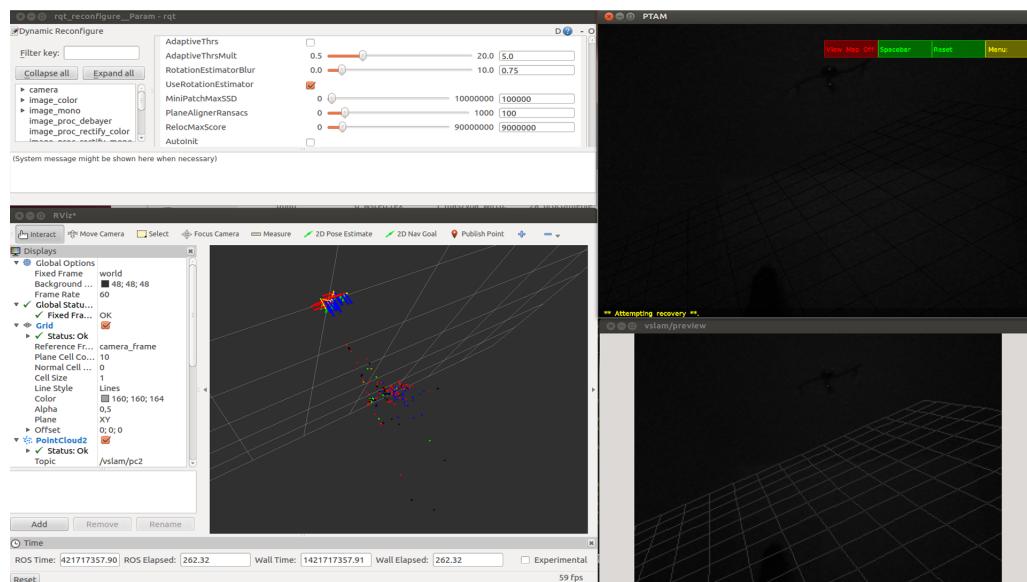
#### PTAM a ROS

Ponieważ algorytm wykrywania przemieszczenia miał być stosowany na platformie ROS, znaleziono zmodyfikowaną wersję algorytmu PTAM. Modyfikacje te odnoszą się głównie do kompatybilności kodu z platformą ROS oraz możliwości uruchomienia algorytmu na platformach z mocno ograniczoną możliwością obliczeniową. Zmodyfikowana wersja PTAM korzysta z komponentów platformy ROS dzięki temu uzyskano możliwości:

- obraz wejściowy uzyskiwany jest z węzła, a obraz zweryfikowany wraz ze swoimi komponentami jest publikowany. Umożliwia to użytkownikowi kontrole nad algorytmem bez interfejsu człowiek-maszyna.
- Wizualizacja trajektorii kamery i chmury punktów została przeniesiona do RVIZ dzięki czemu można dokonać wizualizacji w stacji naziemnej, w razie potrzeby.
- Parametry regulacyjne mogą być zmieniane dynamicznie w specjalnym GUI.

#### Uruchomienie kamery

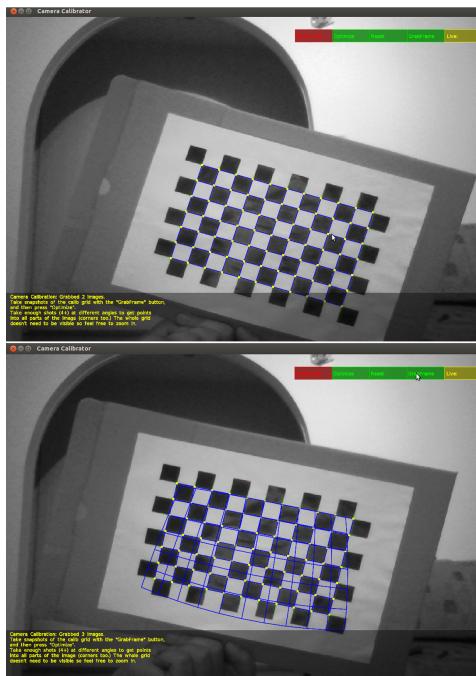
Do uruchomienia algorytmu PTAM potrzebny jest monochromatyczny obraz z kamery, która została uruchomiona przy użyciu sterowników z paczki *gscam* oraz *image\_proc*. Dla ułatwienia uruchamiania algorytmu został napisany plik launch, który uruchamia kamerę, algorytm i komponenty wspomagające jego działanie jak: *rviz*, *rqt\_reconfigurator* 3.8.



Rysunek 3.8: PTAMi i komponenty wspomagające

## Kalibracja kamery

Do prawidłowego działania algorytmu ptam należy skalibrować używaną kamerę. Do tej czynności posłużył wbudowany w paczkę algorytmu moduł do kalibracji **3.9**. Kalibracja kamery polegała na wykonaniu dziesięciu zdjęć „szachownicy”, a następnie pozwoleniu wykonania optymalizacji wbudowanemu modułowi. Wynikiem powyższej operacji był plik \*.yaml.



Rysunek 3.9: PTAM - kalibracja kamery

## Uruchomienie algorytmu

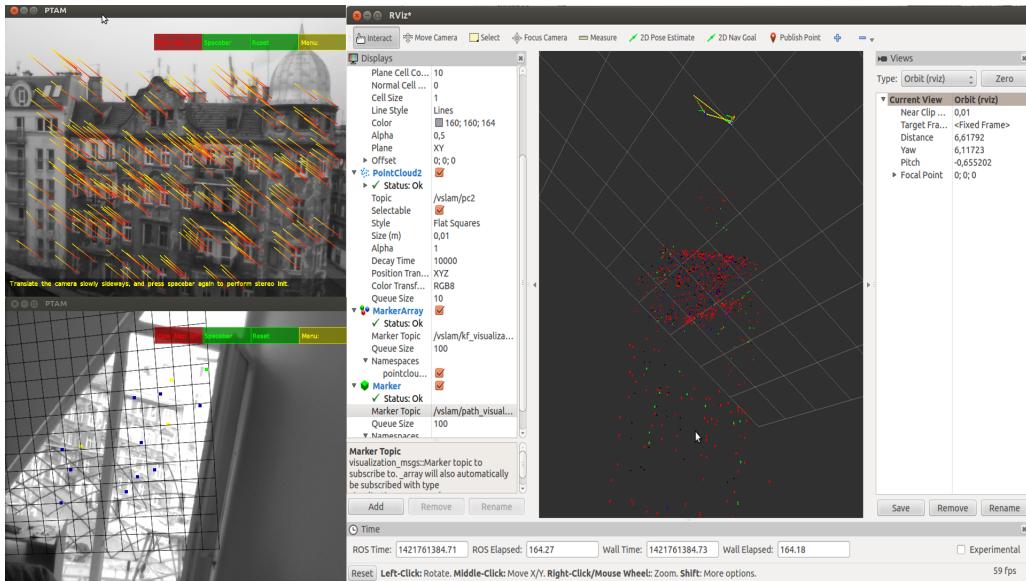
Po uzyskaniu obrazu monochromatycznego oraz skalibrowaniu kamery można przejść do uruchomienia algorytmu. Algorytm uruchamia wcześniej wspomniany plik launch, a dokonuje się tego komendą: `roslaunch ptam ptam.launch`. Po uruchomieniu mamy dostęp do podglądu z kamery, chmury punktów i ścieżki przesunięcia w *rvizie* oraz gui z dynamiczną konfiguracją umożliwiającą dostosowywanie działania algorytmu do naszych potrzeb. Kolejnym krokiem jest inicjalizacja pierwszego przesunięcia. Dokonać jej można przez naciśnięcie spacji, przesunięcie kamery i ponowne naciśnięcie spacji. Należy pamiętać, aby przesuwając kamerę rzeczywiście zmienić jej pozycję, a nie tylko kąt obrotu. Ilustruje to rysunek **3.10**. Po wykonaniu inicjalizacji przesunięcia, możemy obserwować w *rvizie* zbieraną chmurę punktów oraz przesunięcie kamery reprezentowane przez małe układy współrzędnych połączone żółtą linią.

## Pozycja kamery

Paczka algorytmu PTAM publikuje pozycje kamery w topicu `/vslam/pose`. Struktura topicu i uzyskanie dostępu do niego jest takie same jak w algorytmie SVO i zostało to opisane w kolejnym rozdziale.

## Modyfikacje

Na potrzeby projektu zautomatyzowano proces inicjalizacji pierwszego przesunięcia.



Rysunek 3.10: PTAM - działanie algorytmu

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

Wydruk 3.2: Uruchomienie kamery

```

1 <launch>
  2   <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
  3     / output="screen" >
    4       <param name="video_device" value="/dev/video0" />
    5       <param name="image_width" value="640" />
    6       <param name="image_height" value="480" />
    7       <param name="pixel_format" value="jpeg" />
    8       <param name="camera_frame_id" value="usb_cam" />
    9       <param name="io_method" value="mmap"/>
  10   </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 \*.yml na format \*.yaml.

Rysunek 3.11 przedstawia ręczne przekonwertowanie pliku kalibracyjnego.

```

cam_model: Pinhole
cam_width: 640
cam_height: 480
cam_fx: 626.747758
cam_fy: 625.606308
cam_cx: 317.435978
cam_cy: 240.345658
cam_d0: 0.124513
cam_d1: -0.313055
cam_d2: -0.001146
cam_d3: -0.001040
image_width: 640
image_height: 480
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [626.747758, 0, 317.435978, 0, 625.606308, 240.345658, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.124513, -0.313055, -0.001146, -0.001040, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [632.876465, 0, 316.387552, 0, 0, 632.346741, 239.493128, 0, 0, 1, 0]

```

Rysunek 3.11: Konwersja pliku \*.yml na \*.yaml

```

header:
  seq: 5892
  stamp:
    secs: 1418673322
    nsecs: 170325262
  frame_id: /world
ns: trajectory
id: 8195
type: 1
action: 0
pose:
  position:
    x: -0.00851521046703
    y: -0.00349843886924
    z: 0.0106530645546
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0
scale:
  x: 0.006
  y: 0.006
  z: 0.006

```

std\_msgs/Header header  
uint32 seq  
time stamp  
string frame\_id  
string ns  
int32 id  
int32 type  
int32 action  
geometry\_msgs/Pose pose  
geometry\_msgs/Point position  
float64 x  
float64 y  
float64 z  
geometry\_msgs/Quaternion orientation  
float64 x  
float64 y  
float64 z  
float64 w  
geometry\_msgs/Vector3 scale  
float64 x  
float64 y  
float64 z

Rysunek 3.12: Wycinek informacji zawartych w topicu /svo/points

## Pozycja kamery

Do dalszego przetwarzania informacji dostarczonych z działającej paczki SVO można wykorzystać wiadomość publikowaną poprzez topic `/svo/points`. Rysunek 3.12 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.

## 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

Problem stabilizacji w poziomie sprowadza się do utrzymania zadanej orientacji. Do stabilizacji można wykorzystać czujniki inercyjne (akcelerometr, żyroskop), czujniki odległości czy kamery (przetwarzanie obrazów). Stabilizacja poziomu dla quadrocoptera jest

istotna, ponieważ minimalna odchyłka od poziomej orientacji powoduje zmianę kierunku siły ciągu i destabilizuje układ. Dodatkowo zachowanie orientacji horyzontalnej znacznie ułatwia sterowanie obiektami latającymi. W tej sekcji będzie mowa o stabilizowaniu w poziomie z wykorzystaniem wielu sensorów.

### 3.7.1. Fuzja sensoryczna, Alicja

Fuzja sensoryczna to proces polegający na łączeniu informacji wniesionych przez pojedyncze czujniki aby uzyskać informację o stanie dynamicznym obiektu. Sygnały pochodzące z fuzji są bardziej wartościowe gdyż proces ten eliminuje zaszumione lub błędne próbki. W praktyce fuzja sygnałów to zastosowanie matematycznego algorytmu filtracji oraz algorytmu decydującego, który pomiar jest wiarygodny.

#### Czujniki wykorzystywane w fuzji, Alicja

Do fuzji wykorzystano czujniki odległości zorientowane na quadrocopterze horyzontalnie oraz kamerę.

#### 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 quadrocoptera 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  $\delta$ , 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...* CDN

Wydruk 3.3: Pobranie wartości położenia kamery

```

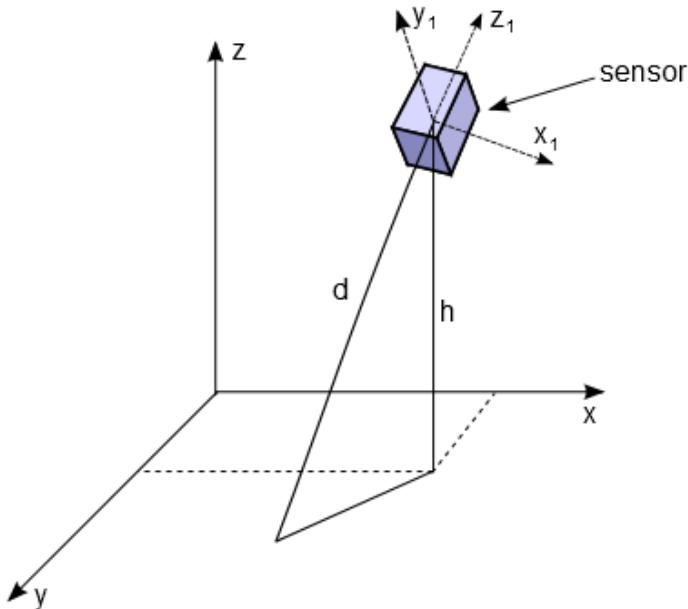
1 void chatterCallback(const visualization_msgs::
2 /Marker::ConstPtr& msg)
3 {
4     t [0] [0]=msg->pose.position.x;
5     t [0] [1]=msg->pose.position.y;
6     t [0] [2]=msg->pose.position.z;
7 }
```

## 3.8. Stabilizacja w pionie, Paweł

### 3.8.1. Fuzja sensoryczna, Michał

W celu zapewnienia stabilizacji wysokości quadrocoptera realizowany jest algorytm sterowania, wykorzystujący fuzję danych z sensorów odległości Sharp oraz orientacji quadrocoptera. Zadaniem fuzji danych pomiarowych jest obliczenie rzeczywistej wysokości quadrocoptera względem podłoża. Obliczenie tej wysokości posiadaając jedynie pomiary z

sensorów odległości nie jest możliwe, ze względu na znaczne zmiany orientacji robota w trakcie lotu, co eliminuje bezpośrednie wykorzystanie zmierzonej wartości jako wysokości.



Rysunek 3.13: Zależności geometryczne rzeczywistej wysokości od mierzonej odległości

Na rysunku 3.13 przedstawiono zewnętrzny układ odniesienia:  $x$ ,  $y$ ,  $z$  oraz układ lokalny:  $x_1$ ,  $y_1$ ,  $z_1$ , związany z sensorem odległości i jednocześnie robotem latającym. Posiadając orientację quadrocoptera odczytaną ze sterownika Pixhawk, wykorzystując zależności geometryczne, można wyznaczyć wysokość rzeczywistą  $h$  na podstawie zmierzonej odległości  $d$ .

### Algorytm stabilizacji wysokości

Dane wejściowe:

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

Dane wyjściowe:

- $F_z$  – siła w osi  $z$  quadrocoptera.

Kroki algorytmu:

1. Obliczenie rzeczywistych odległości od podłoża i sufitu na podstawie odczytów sensorów oraz orientacji quadrocoptera względem zewnętrznego układu odniesienia.
2. 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 nadzędny, Marcin**

#### **4.1.1. Zdarzeniowy sterownik antykolizyjny, Paweł**

Sterownik antykolizyjny pełni ważną rolę w autonomicznym locie quadrocoptera. Ma za zadanie wykrywać obiekty znajdujące się na kolizyjnej trajektorii lotu, odpowiednio je interpretować i reagować w postaci siłowego sprzężenia zwrotnego podawanego na regulator nadzędny quadrocoptera.

#### **Analiza wykonalności, Paweł**

Do wykonania sterownika niezbędne są elementy zaprezentowane poniżej.

- Dostęp do danych sensorycznych quadrocoptera.  
Realizacja odbywa się będzie z wykorzystaniem paczek danych publikowanych w odpowiednich Topicach systemu.
- Dostęp do obrazu kamery 3D.  
Dostęp do chmury punktów kamery 3D wraz z przypisanymi do nich odległościami.
- Zastosowanie odpowiednich filtrów danych wejściowych.  
Nałożenie filtrów uśredniających, wybierających odpowiednią część obrazu do analizy, progowanie wartości odległości.
- Algorytm wyliczający wektor sił odpychających.  
Algorytm wyliczający wektory sił odpychających na podstawie danych wejściowych odpowiednio je przy tym skalując i normalizując.
- Przesłanie danych do sterownika nadzędneg  
Mechanizm umożliwiający publikację wyliczonych danych w systemie przekazywania informacji zaimplementowanym na quadrocopterze.

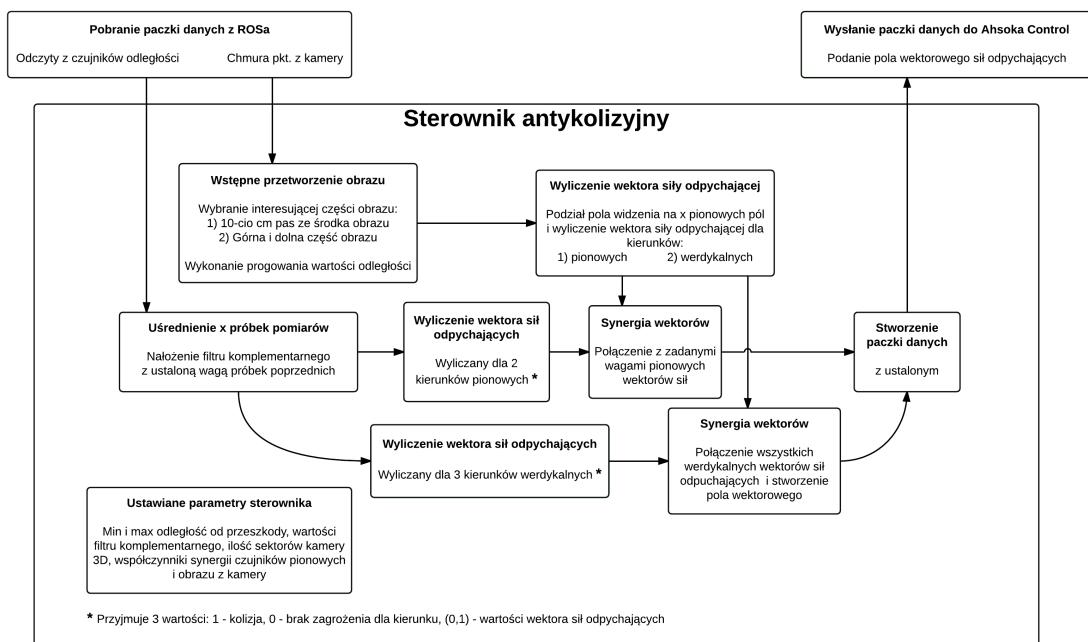
#### **Specyfikacja, Paweł**

Specyfikacja potrzeb została przedstawiona na rys. 4.1

#### **4.1.2. Lot wzdłuż ściany, Krzysiek**

#### **Analiza wykonalności, Mateusz**

#### **Specyfikacja, Mateusz**



Rysunek 4.1: Schemat blokowy działania sterownika antykolizyjnego

**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**