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 Urbaniak
Krzysztof Zawada

Skład raportu wykonano w systemie LaTeX

Spis treści

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

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

Specyfikacja potrzeb, Hania

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

Turbulencje powietrza

Turbulencje powietrza, wywoływane przez śmigła, wpływają na zmniejszenie energii otrzymanego sygnału przez czujnika, mogą zmieniać kierunek oraz intensywność fali 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, szumy 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 montażowy może być używany do blokowania takiego ścieżkę, 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 ten musi być prawidłowo uziemiony do masy. Gdy ekran nie zostanie podłączony z masą, szumy 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 górę nie powinny przekraczać w sumie $3m$, ponieważ zakładamy lot w pomieszczeniu na wysokości 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

Dobór czujników, Hania

2.2.2. 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/

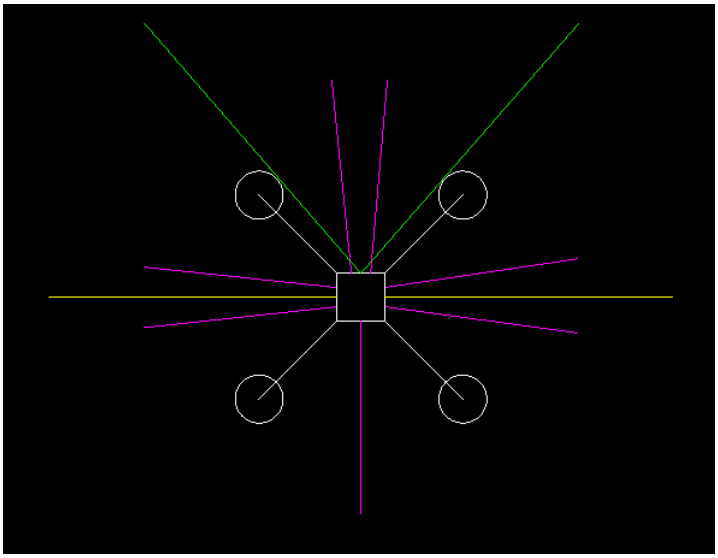
W tych pracach nie występują czujniki odległości ułożone w płaszczyźnie wertykalnej, które byłyby wykorzystywane do stabilizacji w poziomej płaszczyźnie. Funkcję awaryjnego STOP pełnią zazwyczaj obręcze śmigieł, lub piankowe korpusy quadrocopterów. Jest to jednak tylko zabezpieczenie przed uszkodzeniem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

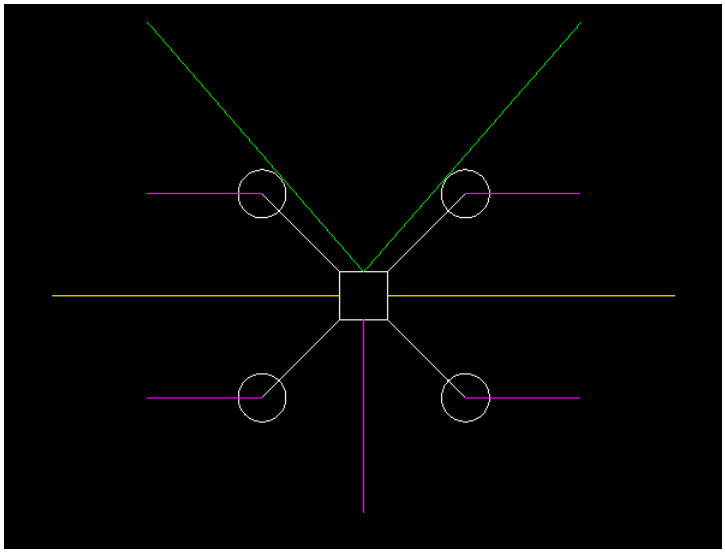
Propozycja rozmieszczenia, Paweł

Zostały zaproponowane dwie konfiguracje przedstawione na rys. 2.1 oraz 2.2. 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 wzdłuż ściany, który polega na



Rysunek 2.1. Konfiguracja I



Rysunek 2.2. Konfiguracja II

bezpośrednim porównywaniu odczytów z czujników znajdujących się po tej samej stronie quadrocoptera. Czujniki dalekiego zasięgu zostały zainstalowane ze względu na potrzebę podejmowania decyzji wzdłuż której ściany powinien poruszać się quadrocopter zaraz po wleceniu 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 mniejsza). Dzięki zainstalowanym na platformie czujnikom inercyjnym wyliczanie prawdziwej wysokości, na której znajduje się quadrocopter, odbywać się będzie z wykorzystaniem bezpośrednich odczytów SHARP'a oraz aktualnymi wychyleniami platformy.

2.3. Algorytmy rozpoznawania przesunięcia, Kacper

2.3.1. Algorytm Lucas—Kanade, Piotrek

Jednym z podejść do wykrywania ruchu na obrazie z kamery jest wykorzystanie tzw. „Optical Flow" (*„przepływ optyczny"*) [1], 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 (Δx,Δy):

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2.1}$$

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

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

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

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

lub

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

co w rezultacie daje:

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

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:

$$I_x V_x + I_y V_y = -I_t. \tag{2.6}$$

W równaniu tym są jednak dwie niewiadome (V_x i V_y), co uniemożliwia jego rozwiązanie w sposób dokładny i jednoznaczny. Jedną z metod, która przybliża jego rozwiązanie, to algorytm Lucas'a-Kanade [4]. 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 rozwiązywane 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].

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, 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.

Ogólny schemat algorytmu SVO

Program SVO (Semi-direct Monocular Visual Odometry) [3] jest zaimplementowany w ROSie. Wykorzystuje on obrazy dostarczone z pokładowej kamery. Quadrocopter zbiera informacje ze środowiska poprzez zdjęcia w dyskretnych chwilach czasu

$$I_1, I_2, \ldots, I_n.$$

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

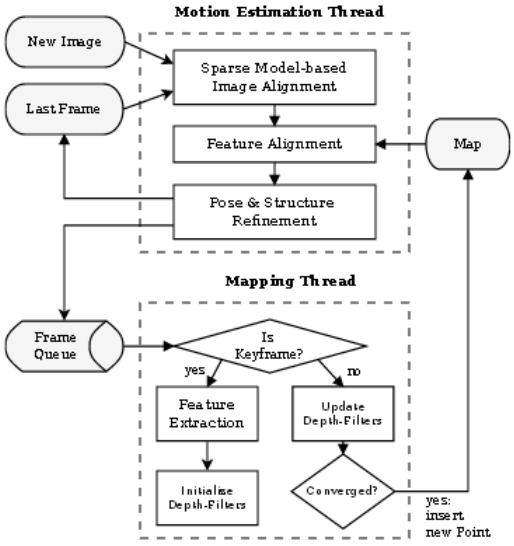
$$C_k = C_{k-1} \cdot A_k,$$

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 2.3 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.

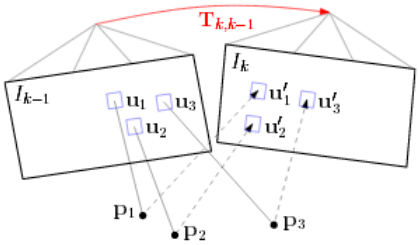
Etapy algorytmu SVO

Estymację ruchu/położenia kamery można podzielić na trzy etapy

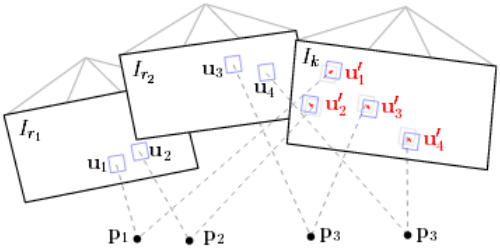
— etap pierwszy Rysunek 2.4 przedstawia zmiana pozycji między wcześniejszą i obecną ramką danych pośrednio przenosi pozycję p na odwzorowanie punktów w nowym obrazie. Algorytm minimalizuje fotometryczny błąd poprawki dotyczącej tych samych 3D punktów p.
— drugi etap Ze względu na niedokładności w punktach 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 2.5.
— trzeci etap Rysunek 2.6 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.



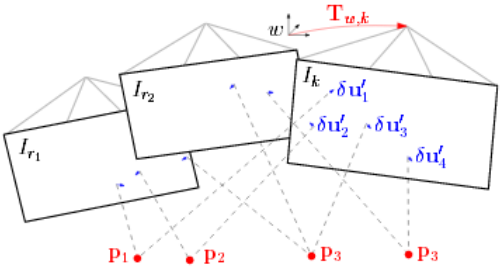
Rysunek 2.3. Schemat algorytmu SVO [3]



Rysunek 2.4. pierwszy etap [3]



Rysunek 2.5. drugi etap [3]



Rysunek 2.6. trzeci etap [3]

3. Stabilizacja w punkcie - realizacja, Krzysiek

3.1. Ahsokam Marcin

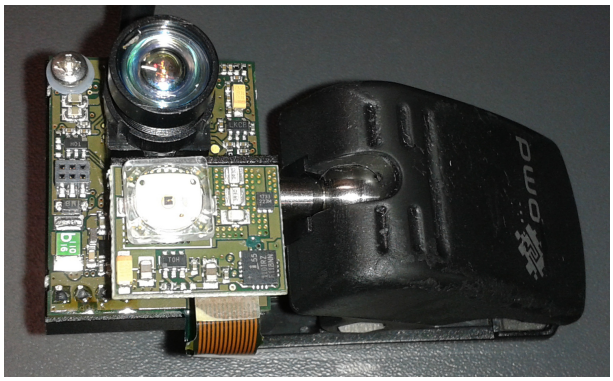
3.2. Uruchomienie platformy latającej, Marcin

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

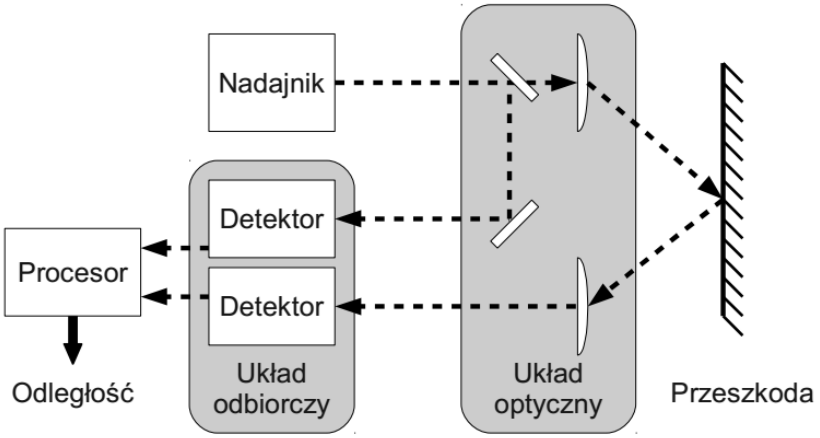
rozdzielczość obrazu 160x120 pikseli,
częstotliwość odświeżania obrazu do 90 Hz,
kąty widzenia 90^o na 68^o,
wymiary 37 x 30 x 25 mm,
długość fali światła 850nm,
pobór prądu około 500mA.



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. 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.



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 umożliwia 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 *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_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,
— 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, 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

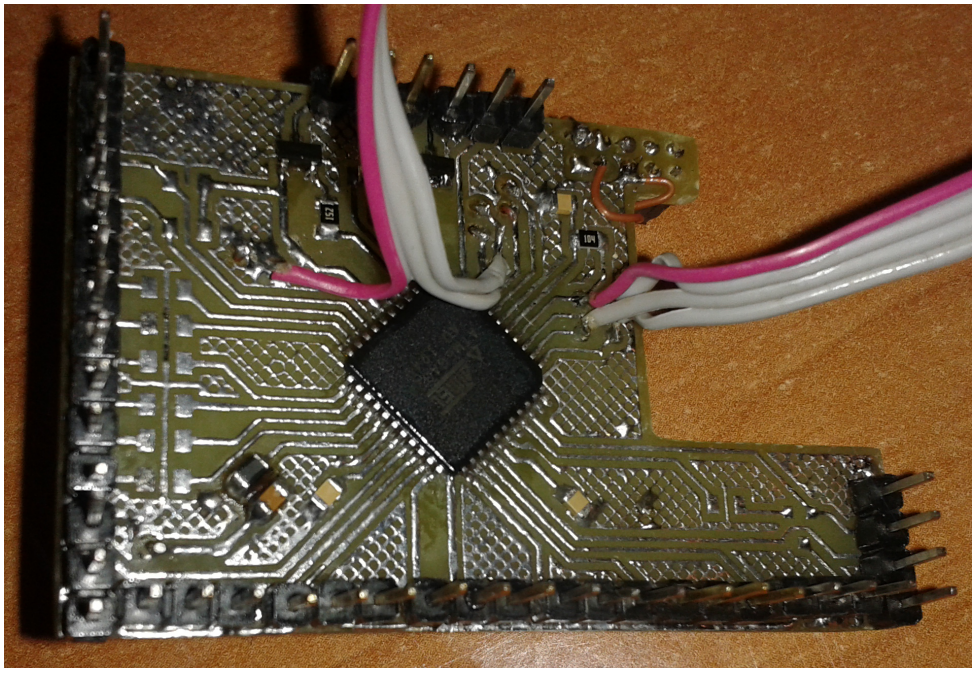
3.5. Czujniki odległości

3.5.1. Płytka PCB, Alicja

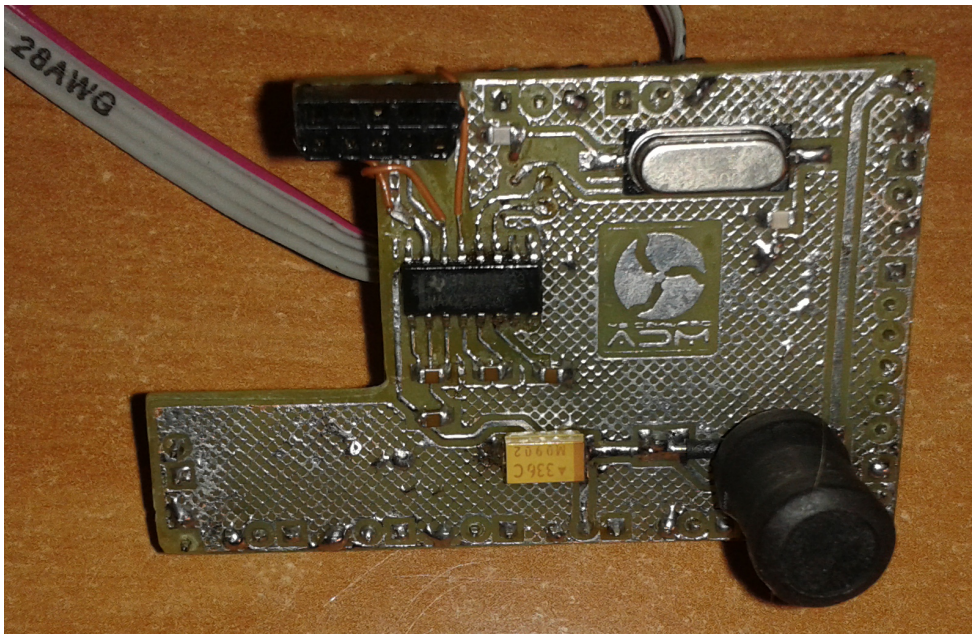
Specyfikacja potrzeb, Marcin

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 quadrocoptera 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 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 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 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 3.4.

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 quadrocoptera.

1101 0001 - 0xD1
Polecenie wyłączenia oświetlenia quadrocoptera.

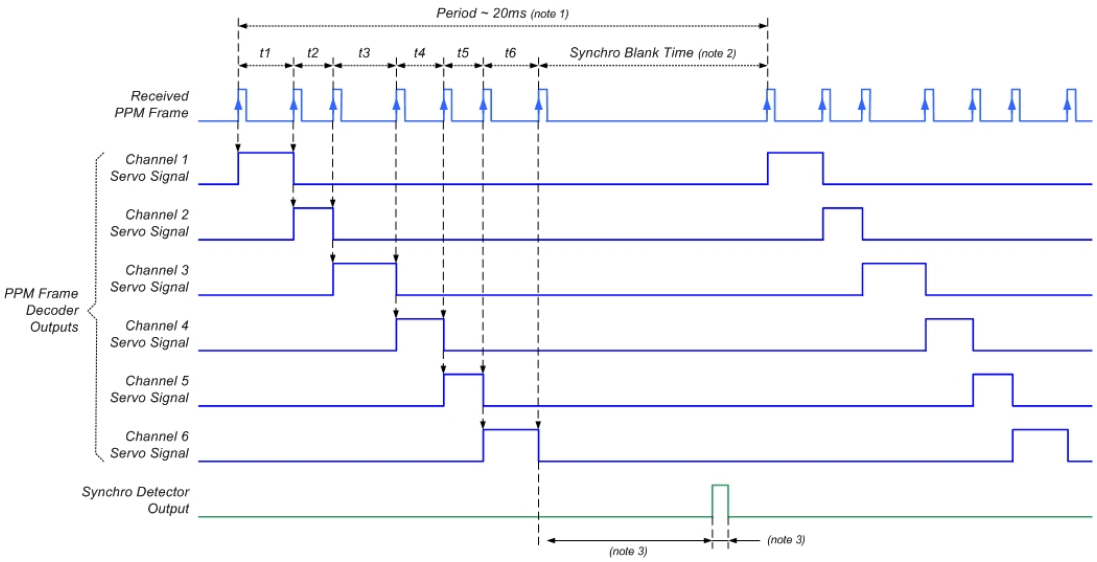
1101 0010 - 0xD2
Włączenie Pixhawk Failsafe (ustawienie stanu wysokiego).

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 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 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

```
ISR (INT2_vect)
{
  cli();
  if (MCUCR & _BV(ISC2)) // rising edge interrupt
  {
    TCNT2 = 0; // Timer2 reset
    timeCntPPM = 0;
    MCUCR &= ~_BV(ISC2); // change to falling edge detect
  }
  else // falling edge interrupt
  {
    if ((timeCntPPM + TCNT2) > MAX_PPM_WIDTH) // new frame
    {
      channelPPMcnt = 0;
    }
    else // measurement for PPM channel
    {
      channelPPM[channelPPMcnt] = timeCntPPM + TCNT2;
      if (channelPPMcnt < 7) // increase channel counter
      {
        channelPPMcnt++;
      }
    }
  }
  GIFR |= (1 << INTF1); // interrupt flag clear
  sei();
}
```

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-

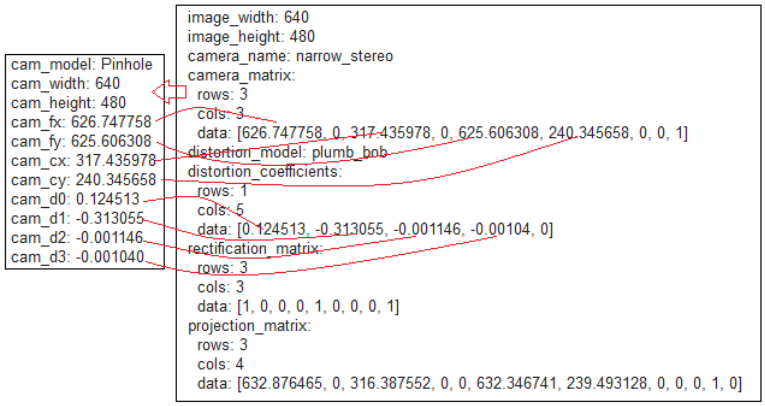
mu, należy uruchomić plik launchowy, dostarczający obraz z kamery. Poniższy wydruk przedstawia uruchomienie kamery.

Wydruk 3.2. Uruchomienie kamery

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
  // output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="mjpeg" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
</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 *.yml. Na potrzeby algorytmu przkonwertowano plik *.yml na format *.yaml.



Rysunek 3.5. Konwersja pliku *.yml 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
  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 3.6. 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 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 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

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

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 quadrocoptera względem zewnętrznego układu odniesienia, w postaci kwaternionu (z Pixhawka),

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. 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

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 nadrzędny quadrocoptera.

Analiza wykonalności, Paweł

Do wykonania sterownika niezbędne są elementy zaprezentowane poniżej.

— Dostęp do danych sensorycznych quadrocoptera.
Realizacja odbywać 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 nadrzędnego.
Mechanizm umożliwiający publikację wyliczonych danych w systemie przekazywania informacji zaimplementowanym na quadrocopterze.

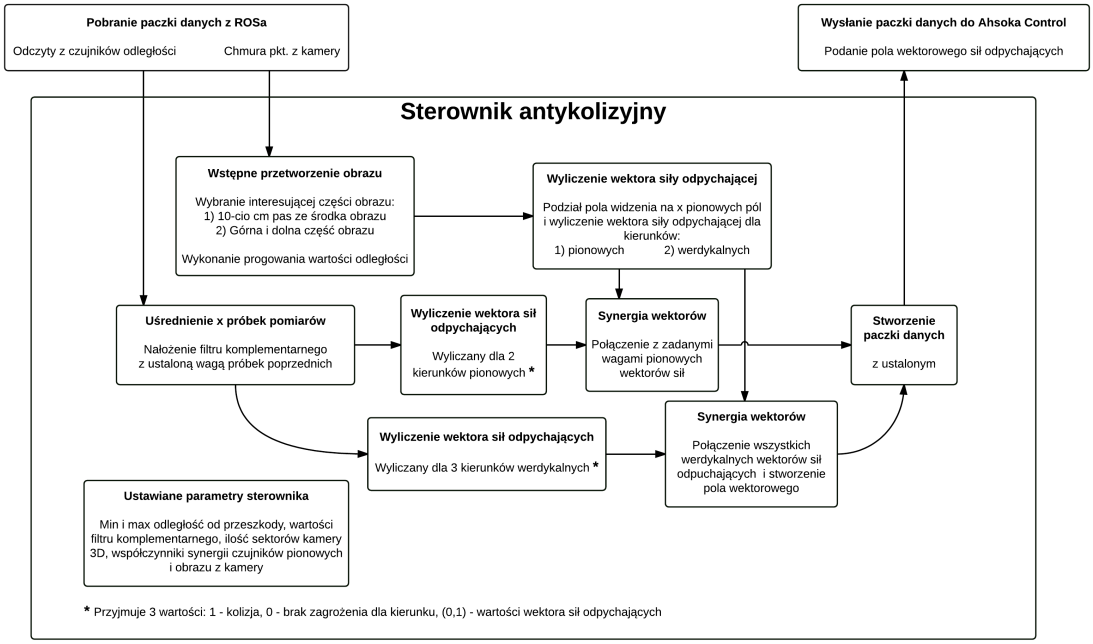
Specyfikacja, Paweł

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

Bibliografia

[1] S. S. Beauchemin, B. J. L. *The computation of optical flow.* ACM New York, USA, 1995.

[2] A. Björck. *Numerical Methods for Least Squares Problems.* SIAM, 1996.

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

[4] B. D. Lucas, T. Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging Understanding Workshop*, strony 121 – 130, 1981.