POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA
SPECJALNOŚĆ: SYSTEMY INFORMATYKI W MEDYCYNIE

PRACA DYPLOMOWA INŻYNIERSKA

System inspekcji obszarów z wykorzystaniem autonomicznych dronów

Autonomous drone-based scouting system

AUTOR:

Mateusz Bączek

PROWADZĄCY PRACĘ:

Dr inż. Michał Kucharzak, Katedra Systemów i Sieci Komputerowych

OCENA PRACY:

WROCŁAW, 2020

Spis treści

Spis rysunków

Spis listingów

Spis tabel

Skróty

GCS (ang. *Ground control station*)
JSON (ang. *JavaScript Object Notation*)
Protobuf (ang. *Protocol Buffers*)

Rozdział 1

Wstęp

1.1. Geneza pracy

Lotnictwo autonomiczne to prężnie rozwijający się sektor branży lotniczej. Technologie pozwalające na wykorzystanie autonomicznych dronów i samolotów w nowych projektach biznesowych są dostępne na wyciągnięcie ręki - istnieją zarówno systemy zamknięte, w pełni komercyjne, jak i projekty zupełnie otwarte, pozwalające na zapoznanie się z kodem źródłowym oprogramowania sterującego statkami powietrznymi i interakcję z aktywną społecznością pasjonatów, wspólnie rozwijającą projekt.

W świecie biznesu powstają coraz to nowe rozwiązania, wykorzystujące autonomiczne maszyny do świadczenia usług - od razu nasuwającym się rozwiązaniem jest autonomiczne dostarczanie paczek [1], ale istnieją też znacznie bardziej ambitne projekty: dostarczanie defibrylatora wprost do domu osoby, u której wystąpiło zatrzymanie akcji serca [2] lub generowanie trójwymiarowych map terenu, wykorzystując zdjęcia wykonanych z dronów [3]. Biznes jest stosunkowo młody, więc branża jest otwarta na innowatorów – firmy takie jak Boeing i Lockheed Martin sponsorują międzynarodowe konkursy przeznaczone dla młodych konstruktorów [4].

Zainteresowani autonomicznym lotnictwem inwestorzy nie ograniczają się do prywatnych firm. Rząd australijskiego stanu Queensland współorganizuje *UAV Challenge* – zawody skupione wokół rozwijania systemów wspierających służby medyczne [5].

Wykorzystanie otwartych technologii, skupionych wokół awiacji autonomicznej i połączenie ich z nowoczesnymi praktykami wdrażania oprogramowania to temat atrakcyjny zarówno z perspektywy inżynierii oprogramowania, jak i z perspektywy biznesowej. Koło naukowe Politechniki Wrocławskiej – *Akademicki Klub Lotniczy* (AKL) zajmuje się budowaniem i rozwijaniem autonomicznych dronów i samolotów [6]. Omawiany w pracy system powstał w ramach realizacji projektu z Wrocławskiego Funduszu Aktywności Studenckiej[7].

1.2. Cel pracy

Celem pracy jest stworzenie prototypu systemu inspekcji terenów, wykorzystującego autonomiczne drony. System ma integrować się z już istniejącym oprogramowaniem sterującym autonomicznymi maszynami, oraz wykorzystywać napisaną na potrzeby pracy infrastrukturę informatyczną, pozwalającą na planowanie tras lotów, obsługę telemetrii i rozpoznawanie obiektów na zdjęciach wykonanych w czasie lotu, za pomocą algorytmów sztucznej inteligencji. Finalnie, system ma generować raport podsumowujący każdy lot, w którym zawarta będzie trasa pokonana przez drona i zdjęcia wykonane w trakcie lotu, wraz z rozpoznanymi na nich obiektami.

Architektura systemu musi pozwalać na zautomatyzowanie procesu wdrażania systemu, oraz zautomatyzowanie wdrażania nowych funkcjonalności - każde z wdrożeń musi być poprzedzone testami integracyjnymi na poziomie całego systemu.

Prototyp ma być w pełni testowalny, zarówno na poziomie pojedynczych elementów systemu jak i na poziomie integracji całego projektu - testy muszą angażować wszystkie komponenty systemu, uruchomione wewnątrz w pełni zautomatyzowanego środowiska testowego.

1.3. Zakres pracy

Zakres pracy obejmuje elementy projektu związane z inżynierią i architekturą oprogramowania - proces projektowania struktury systemu, wybór technologii, zaprojektowanie punktów stykowych w systemie, automatyzacja procesu wdrażania systemu i nowych funkcjonalności.

Praca opisuje także sposób testowania systemu - od weryfikacji poprawności działania poszczególnych komponentów, po pełne automatyczne testy integracyjne, wykorzystujące wszystkie komponenty systemu wraz ze zintegrowanym symulatorem drona.

Rozdział 2

Wymagania funkcjonalne systemu

2.1. Oprogramowanie na dronie

Wielowirnikowce podłączone do systemu muszą być zdolne do autonomicznego lotu - w kontekście pracy oznacza to zdolność do automatycznego startu, lądowania, stabilizacji oraz samodzielnego lotu do koordynatów GPS. W trakcie lotu, maszyna musi zbierać i wysyłać dwa rodzaje danych:

- dane telemetryczne,
- zdjęcia wykonane w czasie lotu.

Dane telemetryczne muszą zawierać informacje o pozycji drona oraz identyfikować maszynę za pomocą unikatowego identyfikatora oraz numeru lotu. Przesyłany jest też poziom naładowania baterii oraz informacja, czy w obecnym czasie prowadzone jest nagrywanie.

2.2. Protokoły wymiany danych

W przypadku systemu działającego autonomicznie, wymiana danych jest kluczowym elementem pozwalającym na sprawdzanie poprawności działania i diagnozowania błędów w systemie. Podczas lotów testowych często nie ma możliwości bezpośredniej obserwacji systemu lub ingerencji w jego sposób działania. Odpowiednia architektura zbierająca i archiwizująca dane z lotów pozwala znacznie szybciej wykryć potencjalne problemy i zapobiec krytycznym błędom.

2.2.1. Dane telemetryczne

Protokół do wymiany danych telemetrycznych powinien wysyłać dane w postaci binarnej, gdyż jest to bardziej efektywne niż kodowanie ich w postaci tekstowej (na przykład w formacie `JSON`, typowym dla języków wysokopoziomowych - wykorzystywanym w technologiach webowych).

Protokół powinien być w łatwy sposób rozszerzalny, pozwalając w przyszłości zredefiniować część wysyłanych pakietów lub dodać nowe dane, bez tracenia kompatybilności wstecznej bądź konieczności przebudowania całego systemu. Poszczególne komponenty systemu będą pisane

w różnych językach programowania - biorąc to pod uwagę, pożądaną cechą protokołu jest też możliwość szybkiego przeportowania go na inny język programowania.

2.2.2. Zdjęcia wykonane w trakcie lotu

Efektywny przesył zdjęć oraz filmów to temat zbyt obszerny i wymagający, żeby poruszać go w treści pracy - system powinien wykorzystywać dowolny prosty w implementacji protokół wysyłania zdjęć. Architektura systemu powinna zapewnić możliwość prostej wymiany tego komponentu, dzięki czemu w przyszłości będzie możliwe zastąpienie go przez bardziej zoptymalizowane rozwiązanie.

2.3. Oprogramowanie serwerowe

Serwer webowy jest komponentem, który odbiera, archiwizuje i przekazuje dane nadchodzące z dronów do aplikacji klienckiej. Jest punktem centralnym systemu, wykorzystywanym bezpośrednio przez wszystkie pozostałe elementy.

2.3.1. Odbiór i multipleksowanie telemetrii

Aby umożliwić diagnozowanie stanu systemu w czaie rzeczywistym - szczególnie stanu wykonujących lot wielowirnikowców, telemetria nadchodząca z maszyn nie może być jedynie archiwizowana na dysku serwera centralnego. Konieczną funkcjonalnością jest przesyłanie jej w czasie rzeczywistym do wielu jednocześnie podłączonych klientów.

Umożliwi to podjęcie akcji w przypadku wykrycia krytycznego błędu, który mógłby zakończyć się uszkodzeniem bądź rozbiciem drona, ułatwi też wykonywanie testów - zarówno na rzeczywistych maszynach, jak i wykorzystujących symulatory lotu.

2.4. Oprogramowanie klienckie

Aplikacja kliencka skupiona jest wokół trzech funkcjonalności:

1. planowanie tras i harmonogramu lotów,
2. odbiór telemetrii i zdjęć z dronów w czasie rzeczywistym,
3. przegląd i analiza telemetrii oraz zdjęć zarchiwizowanych z poprzednich lotów.

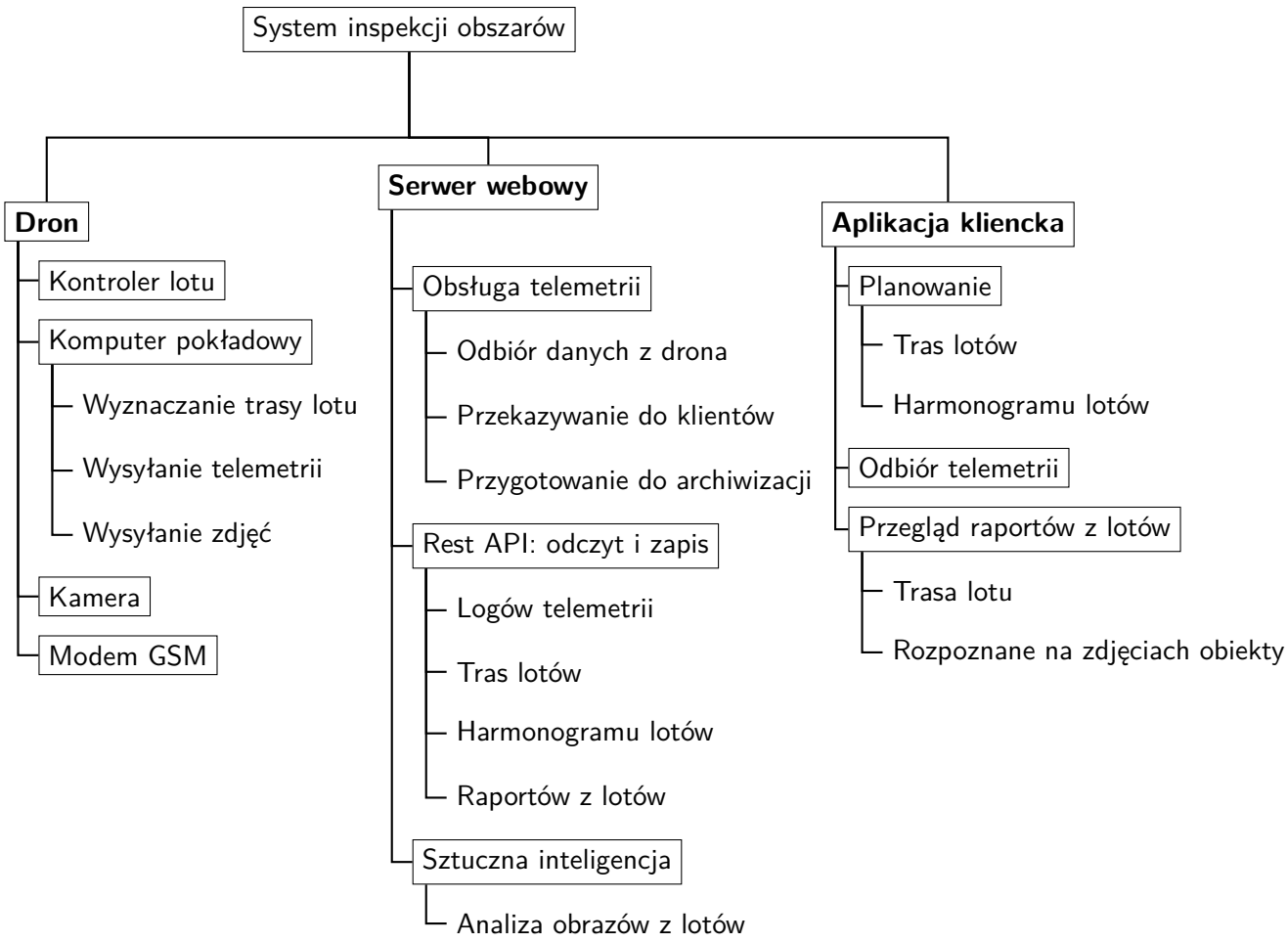
Kluczowym elementem aplikacji klienckiej jest obsługa mapy - wszystkie wymienione funkcjonalności wymagają wizualizacji nadchodzących danych geograficznych, rozszerzonych o dodatkowe informacje (na przykład godzina przelotu przez dany punkt lub zarejestrowane w danym miejscu zdjęcia i wykryte na nich obiekty).

Rozdział 3

Architektura systemu: przegląd i wybór technologii

3.1. Zarys architektury

Rys. 3.1: Zarys architektury systemu



3.2. Dron

3.2.1. Kontroler lotu

Wielowirnikowce podłączone do systemu, muszą być wyposażone w kontroler lotu – umożliwiający autonomiczny lot, stabilizację oraz obsługę peryferiów takich jak czujniki oraz silniki.

Spośród aktywnie rozwijanych i popularnych projektów [8] tworzących oprogramowanie do kontrolerów lotu, można wyróżnić cztery najpopularniejsze inicjatywy:

Tab. 3.1: Najpopularniejsze otwarte projekty oprogramowania obsługującego kontrolery lotu

Nazwa projektu	**Rok założenia**	**Docelowy hardware**	**Licencja**
ArduPilot[9]	2009	otwarte mikrokontrolery ARM	GPLv3
AutoQuad[10]	2011	mikrokontrolery STM Cortex M4	GPLv3
LibrePilot[11]	2015	zamknięte źródłowo kontrolery lotu, bazujące na architekturze ARM	GPLv3
PX4 Autopilot[12]	2012	otwarte mikrokontrolery ARM	BSD

Spośród wymienionych projektów, ArduPilot posiada najbardziej rozbudowaną bazę dokumentacji i instrukcji. Architektura projektu umożliwia skompilowanie projektu na standardowy komputer typu PC uruchomienie go w wirtualnym środowisku[13], co ułatwia proces testowania oprogramowania, które steruje dronem – model maszyny jest symulowany, jednak warstwa komunikacji jest dokładnie taka sama, jak w przypadku pracy z prawdziwym dronem. Dodatkowo, projekt realizowany był w kole studenckim, w którym ArduPilot jest od lat wykorzystywany do sterowania dronami i samolotami, co przesądziło o zastosowaniu go jako oprogramowanie do kontrolera lotu.

3.2.2. Komputer pokładowy

Poza kontrolerem lotu, który zawiera jedynie oprogramowanie niezbędnie do sterowania dronem i udostępniania strumienia telemetriii, na maszynie musi znaleźć się też komputer pokładowy, do zastosowań bardziej ogólnych. Będzie on wykorzystywany do obsługi wysokopoziomowych peryferiów: kamery oraz modemu GSM. Do zadań komputera pokładowego będzie należała także realizacja logiki biznesowej systemu - odczytanie harmonogramu przelotów z serwera webowego i załadowanie do kontrolera lotu konkretnej trasy przelotu.

Wymagania sprzętowe

Aby wypełniać zadania wymienione powyżej, komputer pokładowy musi posiadać następujące interfejsy sprzętowe:

- `UART` – do komunikacji z kontrolerem lotu,
- `USB` – do komunikacji z modemem,
- `CSI` – do komunikacji z kamerą.

Większość dostępnych na rynku komputerów klasy SBC (*Single-board Computer*) posiada powyższe interfejsy, więc wymagania sprzętowe nie są tutaj ograniczeniem. W projekcie został wykorzystany najpopularniejszy do zastosowań amatorskich komputer *Raspberry Pi*.

Oprogramowanie

Na komputerze pokładowym zainstalowany jest system Linux – gwarantuje to bezproblemową obsługę peryferiów oraz dostępność stosu sieciowego, koniecznego do przesyłania danych telemetrycznych i zdjęć. Dodatkowo, wymagane jest oprogramowanie dekodujące i enkodujące wiadomości protokołu *MavLink*, który jest wykorzystywany przez ArduPilota do komunikacji z zewnętrznymi systemami [14].

Infrastruktura projektu ArduPilot dostarcza gotowe narzędziae do parsowania i tworzenia wiadomości w protokole *MavLink*. Jedną z nich jest `pymavlink` - implementacja protokołu *MavLink* w języku Python. `pymavlink` zawiera podstawowe funkcje i obiekty konieczne do komunikacji z kontrolerem lotu. Biblioteka jest niskopoziomowa i nie w całości napisana w sposób obiektowy – finalnie wykorzystujemy więc bibliotekę `dronekit-python`[15], która rozbudowuje `pymavlink` o w pełni obiektowy, wysokopoziomowy interfejs do komunikacji z kontrolerem lotu. Skrypty odpowiedzialne za logikę biznesową napisane są w Pythonie.

3.3. Protokoły wymiany danych

3.3.1. Dane telemetryczne - biblioteka `protobuf`

W trakcie lotu, drony regularnie wysyłają dane telemetryczne, identyfikując się, podając number lotu oraz informują o swoim obecnym położeniu a także podają stan baterii. Nie jest wykluczone, że w przyszłości może pojawić się potrzeba dołączenia do danych telemetrycznych nowych informacji, na przykład w przypadku, gdy do maszyny zostanie dodane nowe oprzyrządowanie, lub gdy system byłby adaptowany do obsługi innego rodzaju misji.

Dodatkowo, skala projektu wymusza utrzymywanie implementacji tego samego protokołu w wielu różnych językach programowania – oprogramowanie wysyłające telemetrię z drona napisane jest w Pythonie, interfejs webowy wyświetlający telemetrię będzie musiał być jednak napisany w języku JavaScript (ponieważ tylko ten język jest wspierany przez przeglądarki internetowe). W przyszłości może pojawić się konieczność dodania wsparcia dla innego języka programowania, niestety wraz z dodawaniem coraz to kolejnych danych telemetrycznych i zwiększaniem liczby wspieranych języków, rośnie szansa na popełnienie błędu w implementacji protokołu.

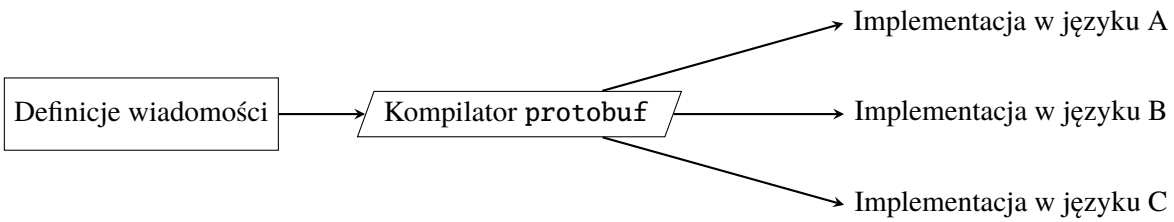
Jednym z możliwych rozwiązań tego problemu jest wykorzystanie narzędzi do generowania kodu. Narzędzia tego typu pozwalają na zdefiniowanie standardu danych telemetrycznych we własnym (specyficznym dla narzędzia) języku, opisującym strukturę danych przesyłanych przez protokół. Następnie, narzędzia takie generują implementację protokołu w wybranych językach

programowania. Dzięki automatycznemu generowaniu implementacji protokołu, programiści nie muszą dbać o spójność implementacji pomiędzy wieloma językami.

Wykorzystywanym w projekcie narzędziem do generowania implementacji protokołu jest `protobuf` (*Protocol Buffers*)[16] – biblioteka napisana w firmie Google, stworzona z myślą o zapewnianiu wydajnej komunikacji w czasie rzeczywistym pomiędzy wieloma systemami informatycznymi. W obecnej wersji (`v3.14.0`), biblioteka pozwala na generowanie kodu w językach:

- Python
- C++
- JavaScript
- Go
- Java
- C#

Rys. 3.2: Diagram przedstawiający system generowania implementacji protokołu, na podstawie narzędzia `protobuf`



Poza automatycznym generowaniem kodu protokołu, `protobuf` zapewnia także:

- wydajne wykorzystanie miejsca – biblioteka upakowuje dane w formie binarnej,
- kompatybilność wsteczną – w przypadku dodania nowego typu pakietu telemetrii, implementacja zachowuje spójność z poprzednimi wersjami. Dzięki temu możliwe jest stopniowe wprowadzanie zmian w wielokomponentowym systemie,
- automatycznie generowane mechanizmy, pozwalające na sprawdzanie, czy dane umieszczone w pakiecie telemetrii są poprawne (sprawdzanie typów, sprawdzanie czy zostały wypełnione wszystkie pola pakietu),
- interfejs programistyczny, umożliwiający samodzielne dopisanie wsparcia nowych języków do kompilatora `protobuf`.

Listing 3.1: Przykład definicji pakietu `protobuf`

```
syntax = "proto3";

package Telemetry;

/**
  Wiadomość zawierająca pozycję
  maszyny w trakcie lotu
*/
message Position {
  float lat = 1;
```

```
  float lng = 2;
  float alt = 3;
  float heading = 4;
}

message TelemFrameHeader {
  /* Flight metadata */
  fixed32 timestamp = 1;
  int32 machine_id = 2;
  fixed32 flight_id = 3;

  /**
        Kompozycja wcześniej
        zdefiniowanej wiadomości
  */
  Position position = 4;
}
```

Listing 3.2: Przykład wykorzystania wygenerowanej implementacji pakietów w języku Python

```
# Wygenerowany moduł zawierający definicje
# pakietów, domyślnie nosi nazwę definitions_pb2
import definitions_pb2
import time

header = definitions_pb2.TelemFrameHeader()

header.timestamp = int(time.time())
header.machine_id = 1
header.flight_id = 5

header.position.lat = 5
header.position.lng = 10
header.position.alt = 15
header.position.heading = 20

# Postać binarna, gotowa do zapisania do pliku
# lub wysłania przez protokół UDP/WebSocket
serialised_header = header.SerializeToString()
print("Serialised header:")
print(serialised_header)
```

3.3.2. Zdjęcia wysyłane w trakcie lotu - biblioteka `imagezmq`

Wykonane przez drony zdjęcia wysyłane są na serwer webowyza pomocą biblioteki `imagezmq`. Jak zaznaczono we wstępie, wysyłanie strumienia wideo to temat zbyt skomplikowany, aby poruszać go w pracy – rozwiązanie wykorzystywane do przesyłu zdjęć zostało wybrane ze względu na prostotę instalacji i implementacji.

Biblioteka `imagezmq` służy do wysyłania obrazów za pomocą protokołu `zmq`.

3.4. Oprogramowanie serwerowe

3.5. Oprogramowanie klienckie

3.6. Struktura repozytoriów

3.7. Wspólne punkty stykowe - `git submodules`

3.8. Podsumowanie architektury

Rozdział 4

Wdrażanie systemu

4.1. Konteneryzacja

4.2. Automatyczne budowanie projektów

4.3. Automatyczne aktualizacje kontenerów

Rozdział 5

Testy systemu

5.1. Testy jednostkowe

5.2. Testy integracyjne

5.2.1. Symulacja i symulatory

5.3. Systemy ciągłej integracji

5.4. Testy w terenie

Rozdział 6

Podsumowanie

6.1. Wyniki testów

6.2. Osiągnięta sprawność

6.3. Pola do poprawy

6.4. Wnioski

Literatura

[1] Amazon Inc, "Amazon prime air," 2013. https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[2] A. Claesson, A. Bäckman, M. Ringh, L. Svensson, P. Nordberg, T. Djärv, and J. Hollenberg, "Time to Delivery of an Automated External Defibrillator Using a Drone for Simulated Out-of-Hospital Cardiac Arrests vs Emergency Medical Services," *JAMA*, vol. 317, pp. 2332–2334, 06 2017.

[3] F. Remondino, L. Barazzetti, F. Nex, M. Scaioni, and D. Sarazzi, "Uav photogrammetry for mapping and 3d modeling-current status and future perspectives," vol. XXXVIII-1/C22, 01 2011.

[4] R. Pogrzebny and K. Florencka, "Sukces polskich studentów na zawodach sae aero design w usa," 2018. https://naukawpolsce.pap.pl/aktualnosci/news%2C29012%2Csukces-polskich-studentow-na-zawodach-sae-aero-design-w-usa.html.

[5] UAV Challenge , "Sponsors and supporters 2019 & 2020," 2019. https://uavchallenge.org/about/sponsors-and-supporters/.

[6] "Akademicki klub lotniczy - strona internetowa," 2020. https://akl.pwr.edu.pl/.

[7] "Wrocławski fundusz aktywności studenckich - strona internetowa," 2020. https://wca.wroc.pl/fast-fundusz-aktywnosci-studenckiej.

[8] E. S. M. Ebeid, M. Skriver, and J. Jin, "A survey on open-source flight control platforms of unmanned aerial vehicle," 08 2017.

[9] "Ardupilot - strona domowa projektu," 2020. https://ardupilot.org/.

[10] "Autoquad - historia projektu," 2020. http://autoquad.org/home/autoquad-project-timeline/.

[11] "Librepilot - strona domowa projektu," 2020. https://www.librepilot.org/site/index.html.

[12] "Px4 autopilot - strona domowa projektu," 2020. https://px4.io/.

[13] "Ardupilot - dokumentacja funkcjonalności symulatora autopilota," 2020. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[14] "Ardupilot - dział dokumentacji poświęcony protokołowi mavlink," 2020. https://ardupilot.org/dev/docs/mavlink-commands.html.

[15] "Dronekit - strona domowa projektu," 2020. https://dronekit.io/.

[16] "Protocol buffers - strona domowa projektu," 2020. https://developers.google.com/protocol-buffers.