Drone Autopilot System For Vehicular Network Using VLC

- Capstone II Project Report -

Nazarbayev University
Department of Electrical and Electronics Engineering
School of Engineering

Copyright © Nazabayev University

Electrical and Electronics Engineering
Nazarbayev University
http://www.nuece.info

Title:
Drone Autopilot System For Vehicular Network Using VLC

Theme:
Scientific Theme

Project Period:
Fall Semester 2018 - Spring Semester 2019

Project Group:
XXX

Participant(s):
Sabit Shaiholla
Asset Bekov

Supervisor(s):
Professor Refik C. Kizilirmak

Copies: 1

Page Numbers: 56

Date of Completion:
April 30, 2019

Abstract:

Nowadays the usage of drones in everyday life applications have increased significantly and includes aspects such as aerial photography, shipping and delivery, 3D mapping, safety surveillance, and wireless internet access. This project delegates the autopilot system for drones that uses computer vision as navigation for vehicular network operating with visible light communication (VLC). The main aim of this project is to develop the autopilot system for drone and simulate the model where the object is to be recognized and followed up by commands generated from the designed autopilot system. The underlying purpose is to develop integrated model system (synergy of developed drone autopilot system and object recognition software framework) to recognize and follow the vehicles based on their locations.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

Preface **v**

1 Introduction **1**
1.1 Basic Concept 3
1.2 Visible Light Communication - VLC 3
1.3 Navigation and Guidance 4
1.4 Computer Vision - OpenCV 5
1.5 Autopilot System: Software and Hardware 5
1.6 Drone Flight Control 6

2 System Design **7**
2.1 Block Diagram of the System 7
2.2 Drone 8
2.3 Flight Controller: Pixhawk 9
2.4 Companion Computer: Raspberry Pi 12
2.5 On-board Camera: Raspberry Pi Camera Module 12
2.6 Telemetry and USB Modem 13

3 Software Design **14**
3.1 Communication Setup 14
3.2 MAVLink Protocol 17
3.3 MAVProxy 18
3.4 Basics of drone control 19
3.5 Basics of OpenCV 21
3.6 SITL Simulation 23

4 Autopilot Script **25**
4.1 Establishment of connection 25
4.2 Computation of optimal UAV position 27
4.3 Initialization of the mission 28
4.4 Computation of optimal UAV position 31
4.5 Frame Information 32

4.6 Marker Detection: Results 32

5 Conclusion **34**
5.1 Results 34
5.2 Problems Faced 35
5.3 Future Works 37

Bibliography **38**

A Main Code **42**

B Real-time Object Detection **50**

Preface

Nazarbayev University, April 30, 2019

Sabit Shaiholla
sabit.shaiholla@nu.edu.kz

Asset Bekov
asset.bekov@nu.edu.kz

Chapter 1

Introduction

In the recent times, there is a vast increase of interest in UAV (Unmanned Aerial Vehicle) development due to majority of applications in civilian and military aims. There are different application approaches in using UAVs in industry: starting from surveillance of objects and ground to generate point clouds, volume calculations, 3D model, industrial inspection of sites on the matter of assessment and finishing by UAV-based network of systems. As [1] state, the reasons of that might be the emergence of compact high power density batteries, long range and low-power radio control devices, low cost frames (mostly made by 3D printing plastics), and evolution of system-on-a-chip and single-board computers and motors. By drawing on the concept of state-of-art of UAVs, [2] implies that the rising importance of using small UAVs (whose weights are around 1 kg) in civilian services can be observed in numerous areas. For instance, flying drones with cameras and access to cloud-based data analytics will allow to continuously monitor the quality of agricultural fields in real-time [3]. Drones will allow mining companies to monitor and control operations related with excavations. Energy and infrastructure companies will be able to exhaustively survey pipelines, roads and cables without risking the safety of employees. In the same vein, [4] [5] in their research, underline the significance of usage of drones in transportation and delivery of goods due to their potential to reduce the cost and time of the operation substantially.

In this proposed project, the main objective addresses the importance of importing drone autopilot system into the vehicular network. In order to picture the current state of vehicular networks, [6] state the importance of security advancement of vehicular networks and investigate the design of secure protocols for vehicular network [7]. In addition, the increase of safety and convenience of road-users is directly related to reliability and authenticity of data transferred through vehicular network. In all that matter, the introduction of drone to this system will enhance the accuracy and security of vehicular network. To answer the question of providing such kind of service, one must ensure the stability and failure-free

features of the drone system. A qualitative study by [8] showed that incorporating UAV in vehicular network will enhance the availability of a connectivity path and will reduce the end-to-end data package delivery delay, thus proving that the hybrid architecture of UAV-based vehicular networks reliable, avangard, and progressive technology to be used. Another prominent research provide by [9] showed that the communication using ad-hoc between a drone and a car can reach up to 3 kilometer range, and achieves at least 50% delivery rate at 2.7 kilometers. This work was done taking into account the altitude of a drone, the antenna orientation, and the relative distance between a drone and a car.

The basic concept of drone-vehicles system is that the control features of drone are carried on the base (ground) station. This approach will import the limitations and restrictions to the system such as short range of connection and high possibility of appearance of obstacles which certainly will affect the data-transfer. One of the main components of UAV/drone is the autopilot system. Hence, the heuristic approach on this issue will be to import the autopilot system on drone so that it can analyze the data obtained without contacting to the base (ground) station. The critical issue of this approach is to design a reliable and failure-safe autopilot system. The analysis of data acquired from sensors such as accelerometer, gyroscope, magnetometer and GPS and to act according to this inquiry will be done on on-board computer of the drone. Furthermore, regarding the question of impact in terms of national priority, the implementation of this project on industrial level will potentially increase the security of vehicular networks and will service the inquiries of data on road traffic. Besides that, the proposed autopilot system will be designed in such a way so that it can be implemented with on other applications as well by changing the components based on your need. The potential impact of this project lies on the periphery of integrated system design and also the potential increase of interest in UAVs for various applications underlines the objectives of the design of drone autopilot system. In economic terms, increase in demand of usage of UAVs create a niche for expanding the abilities and capacity of them. Hence, the importance of design of drone autopilot system is emphasized.

The importance of introducing the UAV-based framework system for data transmission in vehicular network is highlighted. Nevertheless, the data capture and transmission itself is not fully covered in this project as it accentuates on the software and hardware development of autopilot system based on the locations of vehicles (objects). Hence, the data rate demand information is pre-defined in the function calculating the optimal and sub-optimal position of the UAV. The description of this function is provided in Section 4.2: Computation of optimal UAV position. Also, it should be noted that this function was developed as the main part of another research project, hence mathematical relations behind that function is out of scope of this project.

1.1 Basic Concept

Starting from very basics, in order to fly in accurate manner UAV uses different sensors, consisting from GPS, accelerometer, gyroscope, and magnetometer. The information obtained from these sensors is integrated into telemetry parameters and is processed on Pixhawk flight controller to determine the position of the drone on 3-axial space. For instance, the traveling from point A to point B will require to determine the initial and final coordinates of this path and the guidance system will execute the commands to accurately reach that final destination point. Each of sensors continually provide the data obtained to flight controller. By using a feedback loop the old data is compared with newly arrived ones and based on that difference the guidance system will execute commands which will allow to determine the optimal fly route. The determination of location is carried out by numerous data obtained from sensors' readings, and the correlation of accuracy of determination of the UAV location to the volume of data readings from sensors is apparent. To put it another way, the more data readings are obtained the more accurately the current location of UAV will be determined. Having said that, there arises another issue with the processing of all data obtained from sensors, that is the time needed to properly translate and process them. Minimization of the ratio of number of sensors which will be used to the processing speed of data readings from that sensors is the trade-off between accuracy rate and computational complexity. According to [2], the conventional drones use the data from the Inertial Measurement Unit (IMU) containing the three-axis accelerometers, gyroscopes, and global positioning system (GPS) for navigation and guidance. The degrees of freedom refers to the number of independent ways by which the dynamic of system of drone can move, and in general the IMU consists of 6 degrees of freedom (3-axis gyroscope and 3-axis accelerometer). In a study investigating the usage of radar/electro-optical data fusion for non-cooperative unmanned aircraft systems (UAS) sense and avoid, [10] state that non-cooperative sensors are able to detect targets autonomously so they are used for separation assurance and collision avoidance in UAV systems. However, there are limitations in using these set of sensors for navigation such as computational complexity of sensors' readings, unavailability of signals in restricted areas (in buildings) and high interference of signals. Hence, there arises a need for alternative navigation concept.

1.2 Visible Light Communication - VLC

Visible light is one type of optical communication, which can be used as an alternative to RF, where visible light spectrum is used for data transmission. Ordinary lamps can be used as an equipment in VLC, however different LEDs can also be used in order to increase the bit rate and different photodetection sensors can be

used as receivers. VLC has some sensitivity for objects, that can occur on the path between light source and a receiver, and block the signal. Another disadvantage is coverage distance, visible light communication works only for short distances, as data rate dramatically decreases with increase of distance. VLC bit rate can also be affected by noise coming from illumination of photodiodes for example by sunlight, which means that in outdoor use VLC bit rate can be lower that indoor. Thus, VLC can not compete with RF in case of long-range and very high speeds, however it has some advantages compered to RF communication. [11] First, security of connection and isolation of the signal. VLC can not penetrate walls or other big objects, blocking the path of light, therefore in order to be connected to the signal users must be close to the light source and the path between transmitter and receiver should be unobstructed. Additionally, two signals in different locations that are separated by the walls do not interfere with each other. Second, LEDs that can be used as a light source in VLC offer high life span, low-cost, and low power consumption.[12] Another advantage of VLC is that even if the distance covered is short, the area covered by the light source can be big depending on the intensity of the light. Additionally on of the advantages of VLC that it does not severely affect the environment. As researchs on dangerous effect of RF state that it can severely affect human and environment. [13]

1.3 Navigation and Guidance

The underlying aim of this project is to import computer vision-based system navigation and guidance to modify and correct the flying path. The implementation of computer vision and image processing in the proposed system will allow to capture the vehicles need to be followed and calculating the coordinates in 3-axial space which increases the bit-rate transmission of data. For these purposes the on-board camera connected to on-board computer will be used. The basic and core concept is when the camera will detect the distinguished object (in our case the vehicle with special marker) and streaming video frames will be processed on on-board computer (in our case Raspberry Pi) and in its turn computer will send a command via Micro Air Vehicle Link (MAVLink) protocol to Pixhawk to navigate the drone [14]. The overview of the general VLC system, with UAV as an intermediate step, can be seen on following figure:

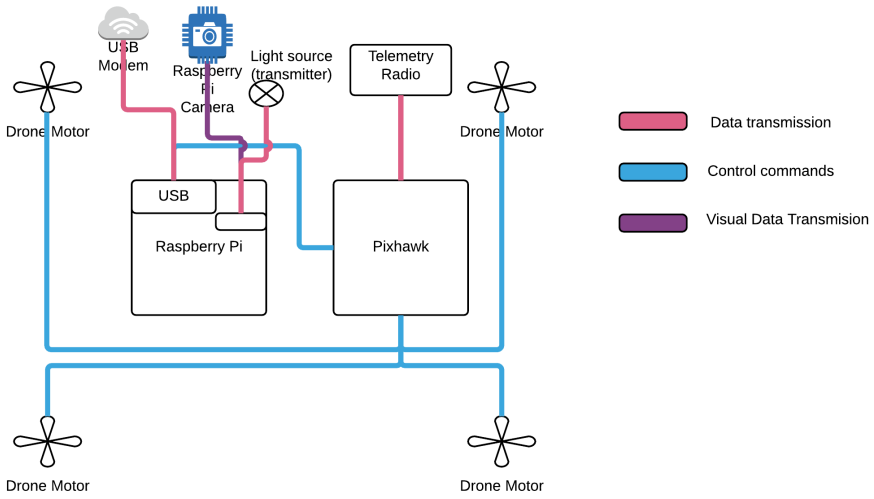


Figure 1.1: Block diagram of proposed system

1.4 Computer Vision - OpenCV

The process of object recognition based on the data reading of visual sensor is implemented by OpenCV library in Python programming language for real-time computer vision. According to [15], OpenCV is an open source computer vision and machine learning software library. The Python script for drone's control will be based on this library framework. In this project OpenCV is implemented into the drone commands function in order for the drone to follow the recognized objects. OpenCV is configured to detect objects, in this case it is moving car, and calculate the distance between the camera, which is located on the drone, and the object itself.

1.5 Autopilot System: Software and Hardware

The guidance system of the UAV will be Pixhawk flight controller based on ArduPilot open source autopilot software. This device will allow to control the UAV based on predefined command tools and it introduces the autonomy of drone's flight. Moreover, the physical characteristics of Pixhawk will allow to attach several sensors on-board and has serial ports to connect to external computers. The reason of choice of Pixhawk as guidance system of our proposed project is pre-default autopilot system which is going to be a backbone of our autonomous autopilot system using computer vision-based navigation for vehicular network using VLC. Additionally, the reason of introducing the Raspberry Pi as an on-board computer

is the inability of Pixhawk flight controller to process the streamed video to navigate the UAV's flight. Hence, the processing of streaming video will be done on Raspberry Pi using OpenCV library. As a candidate for visual sensor the Raspberry Pi's Camera was chosen due to its compliance with Raspberry Pi and its lightness which is crucial in consideration of UAV's limitation of weight lifting.

1.6 Drone Flight Control

Drone can rotate around three axes (x, y, z) from drone's center of gravity. A number of studies [1] [16] have postulated a convergence between three-axial coordinates and angular coordinates, which include roll, pitch and yaw. Roll represents the movement of drone to the right or left sides (longitudinal axis), pitch - movement forward or backward (lateral or transverse axis), yaw - rotation of drone left or right. Additionally, throttle is not a directional but rather it controls the altitude of the drone and speed of the motors. As it appears to be, [17] [18] implies that drones are highly dynamical and nonlinear systems due to variations in input variables and resultant forces with high level of uncertainty introduced by factors such as speed, altitude, weight and turbulence. In general, there are 3 operating modes of drones [19] [20]:

1. Manual mode: roll, pitch, yaw and throttle control commands are sent directly from the user (via Internet or RC) to set the control in open-loop manner

2. Stabilized mode: four main control commands are sent as a setpoints in that to set the control in closed-loop manner

3. Autonomous mode: predefined set of GPS setpoints are governing the navigation of the drone

In this project, we are going to elaborate the third-type of operating mode but rather based on computer vision data for navigation and guidance.

Chapter 2

System Design

2.1 Block Diagram of the System

The proposed system will use an on-board camera to detect the marked object to follow it. In this case, the marked object will be the vehicle with a special marker on it. The aim of using the camera is to collect the information of local environment of the drone. In other words, the detection of the location of the marker with respect to the location of drone. Moreover, the telemetry of the drone will be transmitted via telemetry radio connected to Pixhawk flight controller and it can be sent to ground control station, in our case laptope, which can be used in order to monitor the direction and location of the drone. The Pi camera is connected via camera ports on Raspberry Pi [21]. After the processing the stream video and detecting the marked object, the control command will be send from Raspberry Pi to Pixhawk via MAVLink protocol. The Pixhawk in its turn will control the motors of the drone and the direct of flight, additionally altitude can be set to some fixed position in order to ease the code processing.

In block diagram of electrical connections, the drone frame acts as GND and positive PCB for all the connections. Four motors, ESC motor controllers are connected to the PCB itself, the battery is connected to the PCB via 3DR power module and the Pixhawk flight controller is connected to the battery via 3DR power module, as it requires maximum of 5V supply, whereas the battery is 11.1V. Main outputs of the Pixhawk are connected to the ESC motor controllers in order to control the speed and direction of the UAV. On top of the system raspberry pi is connected to Pixhawk and is powered by external power supply, in this case small powerbank. Rapsberry Pi camera is connected to camera pin on the Raspberry Pi and transmits the video stream to the Raspberry Pi itself.

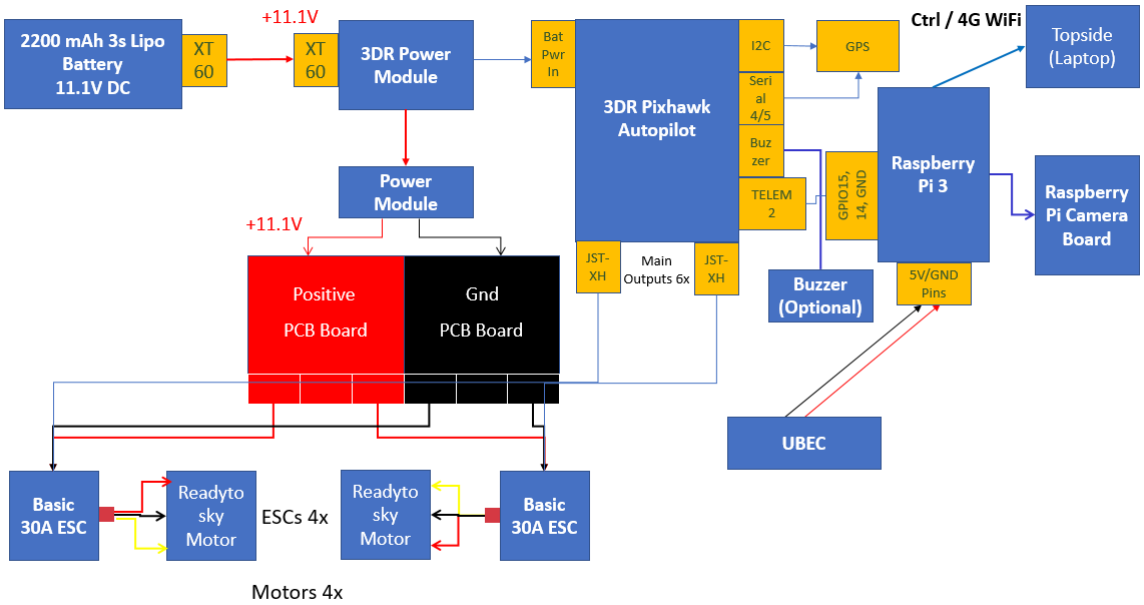


Figure 2.1: Electrical block diagram of proposed system

2.2 Drone

During the system design stage, the numerous types of flight drones were analyzed in order to choose the model which meets the requirements and budget of the project. The most specific criteria are that the drone has to be configurable and easily controlled, has the space to mount and competence to communicate with the on-board computer. Most of the drones were meant for aerial photography and mostly they were not easily configurable. Those types of drones that meet the requirements have the cost which exceeds the the project budget limitation. Therefore we have decided to buy the drone set which is meant for self-assembly. This drone set includes the 500 mm frame, Pixhawk flight controller, motors and blades for them, connectors, motor speed controllers, Pixhawk flight controller, GPS module. The load capacity of this drone set will allow the the mounting of on-board computer and camera. Pixhawk is an independent open-hardware project and it uses a multi-threaded, publish-subscribe design pattern and provides a Unix-based software interface for micro controller applications [22]. Pixhawk's ports will allow to connect several sensors such as GPS, magnetometer, accelerometer, radio and it has serial ports for communication with on-board computers. Moreover, [23] provided research on the topic of using Pixhawk as a basis for computer vision framework, the results of which stated that finding position of the drone on space using computer vision framework could outperform the GPS when there is

proper signal of the last one.

2.3 Flight Controller: Pixhawk

From the technological point of view, the control of small unmanned aerial vehicles (UAVs) is based on autonomous control system (ACS) and ground control systems (QCS), [24] states. These systems include the on-board sensors (magnetometer, accelerometer), monitoring framework (firmware), telemetry system (radio-based), power supply unit. The whole concept of autopilot system incorporates the radio navigational system and inertial measurement unit (IMU) system in small-sized autonomous control unit or flight controller of UAV. According to [25], the avionics system of the drone is the crucial and imperative control module of the system, as it is responsible for failure-free operation of the drone based on measured data by on-board devices. Moreover, [25] elaborate the idea that safety functionality of drone system is provoked by faults based on which the pre-defined set of "code red" commands are activated. For instance, when the power supply unit of the drone (battery) is sufficiently drained and reaches some threshold value (predefined) the flight controller activates the landing command. It can be seen that the precise and accurate implementation of this functionality is elementary and at the same time is principal one, as it is responsible for proper functioning of the drone system. Elaborating on the topic of avionics of the drone, [24] list some of key functions of flight controller, which includes the measurement of kinematic parameters of the drone (velocity, speed of rotation of motors, etc), evaluation of spatial parameters (altitude, latitude, longitude), computation of trajectory parameters (direction of the flight, yaw-pitch-roll angles), acceptance and implementation of control commands received from ground control station (GCS), implementation of pre-defined control functions during appearance of faulty situations (loss of GPS signals, loss of communication link from QCS, etc), monitoring status of on-board devices (battery charge level, safety switch, etc).

The flight controller of the drone processes the input data (digital control commands, radio control commands, sensor-based measurements) and generates output analog control signals (transforms into pulse-width modulated signals), [24] [26]. The main principle behind generated output signals is to minimize the variation and discrepancy of current status of the drone from the desired one defined by flight mission. Besides generating output control signals, [26] put forward the claim that one of the important capability of flight controller should be the potential to cope with other signals' interference whose source might be surrounding radio signals and inner wireless communication devices. Another important quality, according to [26], is the reliability of flight controller's hardware, as the failure of last directly affects the performance of implementation of flight mission.

Modern flight controllers and autopilot systems support multi-mode condition,

with three distinct modes, as [27] state:

1. Manual Flight Mode: all parameters (degrees of freedom) are controlled via remote controller (RC) or by ground control station (GCS). In this mode, no automation parameters are involved. The system status (battery charge level, safety switch, threshold altitude, etc) is observed from GCS.

2. Semi-automated or Assisted Flight Mode: this mode incorporates only some parameters control via RC or GSC such as velocity values. Additionally, the autopilot system is in stabilized mode which reduces the complexity of control parameters to be taken into account and by the pilot and assists the control of the drone.

3. Autonomous Flight Mode: the flight mission is implemented without RC or GCS interaction during the flight via pre-set spatial points. Additionally, the system parameters can be observed from GCS.

Application of modern control theory methodologies in autopilot systems is quite active research area with high potential of implementation in real-life situations. This view is supported by [25], which put forward claim that model-based real-time analysis of autopilot system is a crucial practice in defining the exactitude and preciseness in execution of low-level control commands. The software implementation of autopilot system is a sophisticated and complex issue, taking into account numerous factors such as modern demand in small-scale low-budget hardware applicable basis. In other words, the software implementation is resource-constrained due to modern economic view on size, cost, and energy effectiveness of hardware components of drone systems. Additionally, [25] state that the implementation of control tasks is time-constrained in real-time response systems, and to be efficient and effective it should be accomplished within bounded time limits. Encroachment of these boundary limits might cause direct harm to surrounding people and objects, and hence this parameter is vital in defining the reliability of an autopilot system. In most cases, the analysis of these factors are analyzed on software simulation (Section 3.4: SITL Simulation). As the particular solution to this issue, [28] proposes the technique to assess the closed-loop performance of the drone based on model predictive control (MPC). However, the real-time responses to provisional input cannot be analyzed in its full and in-depth nature from software simulation. That's why the proper analysis of control theory methodologies applied in designing a particular framework of the autopilot system is an imperative component in choosing the proper system.

To start with, the primary reason for choice of Pixhawk platform as the autopilot system for the basis of our drone is that it is widely used and supported platform which is also popular in the USA. The second reason is that it is an open-source platform, meaning that project's source code is publicly available. This is

an important point, as the core idea of our project is to implement computer vision characteristics to define the highest bit-rate spatial position of the drone in vehicular cluster operating via visible light communication. These configurations and modifications are easier to realize as it is an open-source platform, which can be adjusted according to user requirements. However, it should be noted that by being an open-source platform it also brings some compatibility issues related to software performance as this project is being supported by numerous distinctive developers. This to be said, should different part of source code being developed by different developers it would definitely bring some instability in software performance. The issues that appeared during the design stage will be discussed in further sections.

The physical layout of Pixhawk 1 flight controller can be observed on Figure 2.2. According to [29], it is the 1st generation of PX4 product series with 32-bit ARM processor with capability to integrate input and output interfaces (PX4IO). Moreover, [26] reports the following additional characteristics of this autopilot system:

1. 32-bit processor with core frequency of 168 MHz, 256 KB RAM, 2 MB flash memory and support of SD card for parameter and mission loading, and saving the logs of flights
2. 32-bit fail-safe co-processor for back-up flight safety system and command tasks override.
3. 16-bit main accelerometer and gyroscope.
4. 16-bit additional gyroscope, operates as a back-up if the main one fails
5. 14-bit acceloremeter and magnetometer
6. barometer

Besides this inner characteristics, [26] states that it supports following parameters: Spectrum DSM, DSM-2, DSM-X Satellite (SPKT/DSM port) for RC transmitter and receiver systems; 5 UART, SPI, I2C, USB, 2 CAN and 2 ADC inputs; 14 PWM and servo outputs; input power port.

Figure 2.2: Pixhawk 1 Flight Controller

2.4 Companion Computer: Raspberry Pi

Raspberry Pi 3 Model B+ has been chosen as an on-board computer, as it is one of the most widely used single board computers. Raspberry Pi is used in order to process all the codes, additionally, it supports Python which is very convenient for the system as all the codes has been written in Python. Raspberry is controlled through the terminal in Raspbian Operating System. To start with, it should be taken into account that the SD-card (on which will be the image of Raspbian OS) should be at least 8 GB memory size, as it will guarantee that all the necessary packages and libraries will be included in order to our code operate properly. Moreover, the bigger memory space will allow to save all the flight logs and video streams on that memory stick, and can be analyzed afterwards. The Raspbian OS can be downloaded from official web-site of Raspberry Pi.

2.5 On-board Camera: Raspberry Pi Camera Module

The most important criterion for choosing the on-board camera was its lightness and competence to transmit visual data to on-board computer. Hence we decided to use the Raspberry Pi camera because its easily compatible with Raspberry Pi and its lightweight. Moreover, as [30] states, it has enough resolution to detect the marked object at high speeds and different heights. In the same vein, [31] has built the vehicle collision avoidance system in which the Pi camera detects the objects at high speed.

2.6 Telemetry and USB Modem

In the case of the drone system used, there is antenna connected to the drone which sends telemetry of the drone. Its function is to send drone's position, longitude, altitude to the ground control station for monitoring purposes. USB modem in the system is used to establish the connection between on-board computer and ground control station.

Chapter 3

Software Design

3.1 Communication Setup

The main idea behind this developing autopilot system is to identify the target group via camera and calculate the distance to the objects, and process the maximum throughput spatial position according to the pre-defined function. The establishment of communication between the companion computer and flight controller is necessary as the main mission control commands are being sent from 1st one to the 2nd. Hence, the maintenance of this feature is archival.

To start with, the Raspbian OS should be installed on SD card according to the official web-site's manual. After that, by connecting external monitor and interfaces (keyboard and mouse), the setup of companion computer can begin. Firstly, proper Wi-Fi connection should be established as new packages and libraries would be downloaded and installed. After that, by opening the terminal window ("CTRL-SHIFT-T"), the pre-installed packages should be updated first, and then upgraded via the following commands:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

The commands should be entered without "$" sign, as it just represents the beginning of the line. After that, the required packages can be installed:

```
$ sudo apt-get install screen python-wxgtk2.8 python-matplotlib python-
    opencv python-pip python-numpy python-dev libxml2-dev libxslt-dev
    python-lxml
$ sudo pip install future
$ sudo pip install pymavlink
$ sudo pip install mavproxy
$ sudo pip install dronekit droneapi
```

First line installs the Python-related packages which will be used in our script. Second line installs the library package to port the code from Python-2 version to

Python-3 version and vice-versa. This is useful package as our script is written on Python-3 version, however in order to esatblish proper communication between companion computer and flight controller the Python-2 version will be used instead. Subsequent two lines installs the protocol and proxy server packages respectively. Last line will install the DroneKit-Python library which contains the commands to control the drone parameters with Python script.

During the flight mode, it will be problematic to retain the external interfaces of companion computer, and thus another proper method to communicate with it should be used. This is done by communicating via secure shell (SSH) network protocol. In order to operate with it, the SSH mode should be enabled on companion computer first. This can be done by entering into system configuration window of Raspberry Pi via the following command:

```
$ sudo raspi-config
```

The configuration menu will look like this:

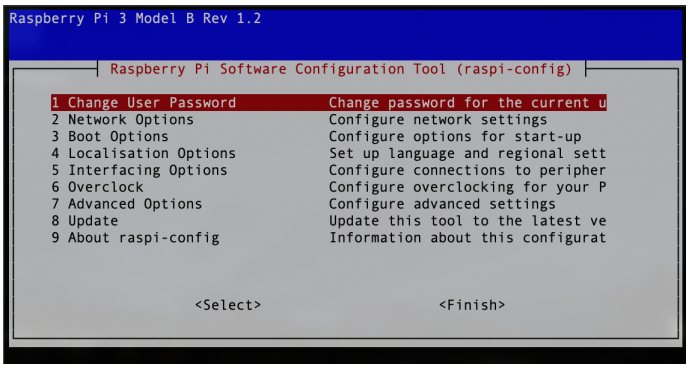


Figure 3.1: Software Configuration Window

The SSH channel can be enabled in "Interfacing Options". After each configuration setup, the computer should be rebooted so that the changes made will take action. Lastly, by again entering into configuration menu, the serial port control should be disabled from "Interfacing Options". This is done in order to make it available from hardware side, as the flight controller will connected to it. Another reboot should be done after this action. Before finally shutting it down, the IP-address of the Raspberry Pi should be wrote down as it will be used in SSH channel establishment. This can be easily done via *"ifconfig"* command.

The SSH channel communication between desktop computer (with ground control station) and companion computer now can be established. It should be noted that both of these devices should operate in the same network to be communicating between each other. In Linux and macOS based computers in the terminal window the following command should be entered:

```
$ ssh pi@<IP-address>
```

Instead of <IP-address> the real IP-address of Raspberry Pi should be entered. Then, the terminal window will prompt to enter the password to connect which, by default, is *"raspberry"*. As the result, the terminal window of Raspberry Pi is accessed via SSH channel. Moreover, as in this project we are using Raspberry Pi 3 Model B, it has some compatibility issue with bluetooth port enabled instead of serial one. In order to overcome this issue, the following command should be entered:

```
$ sudo nano /boot/config.txt
```

This will open booting configuration parameter list in command line text editor. Down to the bottom, the following two lines should be inserted:

```
enable_uart=1
dtoverlay=pi-disable-bt
```

This will enable the UART serial port and disable the bluetooth port instead. The same as before, after making changes in configuration parameters, Raspberry Pi should be rebooted.

Next part will be preparing the flight controller to be connected to the companion computer. First of all, it should be noted that the connection between two devices can be performed in two distinct ways:

1. By directly connecting USB-cable between two devices (USB 3.0 side - Raspberry Pi, Micro-USB side - Pixhawk)

2. By connecting via TX and RX pins (on Raspberry Pi - GPIO 14 and 15, on Pixhawk - TELEMETRY 2 port's 2nd and 3rd pins).

It has been found that both ways operates as it is expected, nevertheless the USB-cable connection was chosen for our project. Before connecting two devices, firmware installation on Pixhawk should be done firstly. In order to do that, Pixhawk controller is connected to desktop computer with ***QGroundControl*** software on it. In the software, the latest stable version of firmware can be uploaded to Pixhawk. After the uploading the firmware, several parameters should be set in order to enable proper communication. In "Parameters" list, following steps should be done:

1. SERIAL2_PROTOCOL=1 (this will enable MAVLink protocol on serial port)

2. SERIAL2_BAUD=921 (this will set the baud rate of serial port to 921600)

3. LOG_BACKEND_TYPE=3 (this will set APSync to to stream the dataflash log files to Raspberry Pi)

Conseqeuntly, the Pixhawk controller should be rebooted and connected to Raspberry Pi. From the terminal window connected via SSH channel, the following command should be entered:

```
mavproxy.py --master=/dev/ttyS0 --baudrate 921600 --aircraft MyCopter
```

This line of command will run the proxy-server between Raspberry Pi as the master (/dev/ttyS0 is the port on which USB cable is connected) and Pixhawk on predefined baud rate of 921600 bps.

3.2 MAVLink Protocol

MAVLink (Micro Air Vehicle Communication Protocol) is a lightweight messaging protocol for communicating with drones and between on-board drone components. MAVLink follows a modern hybrid publish-subscribe and point-to-point design pattern [32]. Structurally MAVLink message ('msg') has 17 bytes which is 6 bytes of header, 9 bytes of payload and 2 bytes of checksum. This protocol will be used to sent commands from Raspberry Pi to Pixhawk flight controller based on visual imagery processing data. It should be noted that this protocol supports several programming languages, including Python-2 version which is used in our main script.

According to [33], one of the important crucial thing to take into account while maintaining the connection using MAVLink protocol is to reduce the baud rate in order to minimize the occurance of errors during the transmission. This is true while the commands to flight controller are being sent via RC or GCS, where the main role of choice in this situation is given to signal-to-noise ration (SNR). In other words, the bigger should be the baud rate, in the for the proportional amount the SNR will be deprecated, hence the choice for smaller baud rate value is logically supported. However, in our case all the control commands are being sent from the companion computer connected directly to flight controller, hence the value of baud rate is not important unless the proper connection is maintained beforehand. The default value of baud rate for MAVLink protocol is 57600 bps. As we are using the USB-cable for connection between two devices, the baud rate was chosen to be 921600 bps, which is governed by Raspberry Pi's CPU's core frequency. The default value of baud rate over USB is 115200 bps.

Diving into how this protocol works, it should be noted that there are 4 distinct parameters in each MAVLink message sent:

1. System ID: the ID indicates the source of the message send, i.e. the companion computer in our situation
2. Component ID: the ID of the subsystem, which is not used in our case, and hence can be ignored
3. Message ID: represents what the payload is about
4. Payload: the main component of the message. In other words, it contains the desired command user want to send to flight controller.

According to [34], there are 18 distinct message types of this protocol, and 91 defined messages in the documentation. Moreover, [34] claim that the link between drone and GCS must be bidirectional, as with the command send from GCS to the UAV there is a response from last one containing the heartbeat, status of sensor devices, telemetry. The data is sent in packets. According to official documentation of MAVLink protocol [35], there are about 100 message types. There are some message types which can be used:

1. MAVLINK_MSG_ID_HEARTBEAT: this message is used by GCS every 1 second whether the flight controller is connected to it or not. If a number of the heartbeats is missed, then fail-safe command will be activated, for example landing or returning to launch (RTL) position. This parameter can be modified from GCS in advance.

2. MAVLINK_MSG_ID_PARAM_REQUEST_LIST: shows full parameter list

3. SET_MODE: sets the type of mode which will be requested by GCS

In order to be able to communicate with the flight controller with this protocol, the proxy media is required.

3.3 MAVProxy

The software package for communication between GCS and flight controller which uses MAVLink messaging protocol is called MAVProxy [36]. As it suggested by its name, it is a proxy-server for communication, designed for UAV-based technology. According to [37], MAVProxy is written on Python language, and hence will ease the process of connection establishment. The MAVProxy communication can be seen on the figure below:

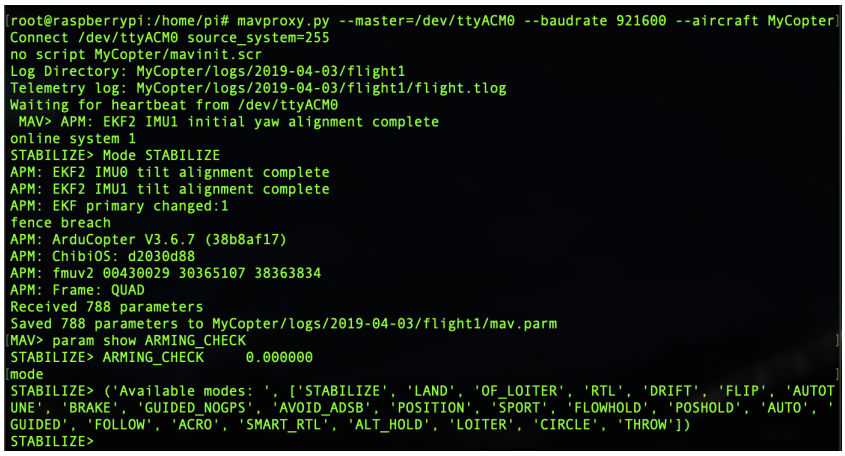


Figure 3.2: MAVProxy communication

The command for MAVProxy initialisation is following:

```
sudo -s mavproxy.py --master=/dev/ttyACM0 --baudrate 921600 --aircraft
    MyCopter
```

In the command above, the "master" parameter represents in which serial connections the flight controller is inserted (in this case, as it is connected via USB, the port is ACM0 which default for Raspbian OS). This can be checked via ***"ls /dev/tty*"*** command. The baudrate is set to 921600 as this is the core frequency of CPU in 3rd-generation Pi computer. "Aircraft" parameter can be changed as this is used as an example.

3.4 Basics of drone control

The drone software programming is mainly based on DroneKit-Python API and OpenCV library, as it was stated previously. To start with, the connection to the drone should be established. This can be done as following:

```
# Importing DroneKit-Python
import dronekit
# Connect to the Vehicle using "connection string"
# In this case IP address of designated drone
vehicle = connect('127.0.0.1:14550', wait_ready=True)
```

In this code, the default IP address is used (it may differ) and port number is 14550 because MAVlink usually uses this port for IP-based packet forwarding. The drone attributes can be simulated in SITL (Software in the Loop) library as following:

```
import dronekit_sitl
sitl = dronekit_sitl.start_default()
connection_string = sitl.connection_string()

# Importing DroneKit-Python
import dronekit
# Connecting to the drone simulation
vehicle = connect(connection_string, wait_ready=True)
# Obtain some drone attributes (states)
print (" GPS: %s" % vehicle.gps_0)
print (" Battery: %s" % vehicle.battery)
print (" Last Heartbeat: %s" % vehicle.last_heartbeat)
print (" Is Armable?: %s" % vehicle.is_armable)
print (" System status: %s" % vehicle.system_status.state)
print (" Mode: %s" % vehicle.mode.name)    # settable

# Close vehicle object before exiting script
vehicle.close()
# Shut down simulator
sitl.stop()
```

In the example above, the attributes of drone can be obtained: GPS position information (horizontal and vertical dilution of positions , fix type, number of satellites visible), battery level (including voltage and current values), time since last MAVLink heartbeat (message) was received (in seconds), defining whether the drone is armable or not, showing the system status, get and set the current flight mode. The output of the script can be seen below:

```
Start simulator (SITL)
Starting copter simulator (SITL)
SITL already Downloaded and Extracted.
Ready to boot.
Connecting to vehicle on: tcp:127.0.0.1:5760
>>> APM:Copter V3.3 (d6053245)
>>> Frame: QUAD
>>> Calibrating barometer
>>> Initialising APM...
>>> barometer calibration complete
>>> GROUND START
Get some vehicle attribute values:
 GPS: GPSInfo:fix=3,num_sat=10
 Battery: Battery:voltage=12.587,current=0.0,level=100
 Last Heartbeat: 0.858172230015
 Is Armable?: False
 System status: STANDBY
 Mode: STABILIZE
```

Figure 3.3: Obtaining the drone attributes in SITL

Arming the drone is defined as the process when the on-board computer causes flight controller to apply power to drone's motors, which will cause them to start spinning. The drone attributes can be set by **class dronekit.Attitude(pitch, yaw, roll)**. Moreover, the drone can be sent to a specific location using **simple_goto()** predefined function:

```
# Set mode to guided
vehicle.mode = VehicleMode("GUIDED")
# Set the LocationGlobal to head towards
# Parameters - latitude, longitude, altitude (in meters)
new_location = LocationGlobal(51.090482,71.400178,10)
vehicle.simple_goto(new_location)
```

In order to send control commands (target landing or target altitude holding), the appropriate MAVLink message should be sent using **message_factory()** and **send_mavlink(message)** functions to flight controller from on-board computer:

```
def target_land_msg(x, y):
    msg = vehicle.message_factory.landing_target_encode(
        0, # time_boot_ms (not used)
        0, # target num
        0, # frame
        x, # (x-coordinate)
        y, # (y-coordinate)
        0, # altitude (not needed as it is landing)
```

```
        0,0) # size of target in radians
    vehicle.send_mavlink(msg)
    vehicle.flush()
```

The main script of drone control movements are available on Appendix A.

3.5 Basics of OpenCV

OpenCV (Open Computer Vision) is an open source computer vision and machine learning software library [15]. Starting from very basics, the hardware that will be used is Raspberry Pi 3 Model B+ and Raspberry Pi Camera Module V2. The actual setup will be available at Capstone 2 part (when all hardware components for project will be available). Here is the code part for capturing the image:

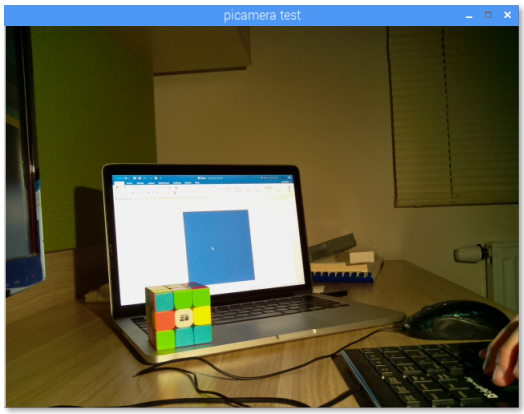
```
# Import modules
import picamera
import picamera.array
import time
import cv2

# Initialize camera
camera = picamera.PiCamera()
camera.resolution = (640,480)
rawCapture = picamera.array.PiRGBArray(camera)
# Warm up time
time.sleep(0.1)

# Capture image
camera.capture(rawCapture, format="bgr")
img = rawCapture.array

cv2.imshow("picamera test", img)
cv2.waitKey(0)
```

In the above code, **picamera.array** is a sub-module of picamera that allows to get NumPy arrays from the Pi camera, where NumPy array is used for image representation. Consequently, the camera object is initialized, camera resolution is set to 640x480 and **rawCapture** variable is initialized for storing the captured image. The resolution is set to low parameters as video stream and image processing are intensive for computing, despite the fact that the full resolution of Pi camera can be set to 3280x2464. The image is captured by **camera.capture(rawCapture, format="bgr")** funtion, where first argument is variable for storing the image and second variable for raw BGR (blue, green, red) format. Finally, the **cv2.imshow()** function displays the image output on the screen and waits for key interruption after which the window will be closed and program stops to run. Following that, the simple object detection can be done:

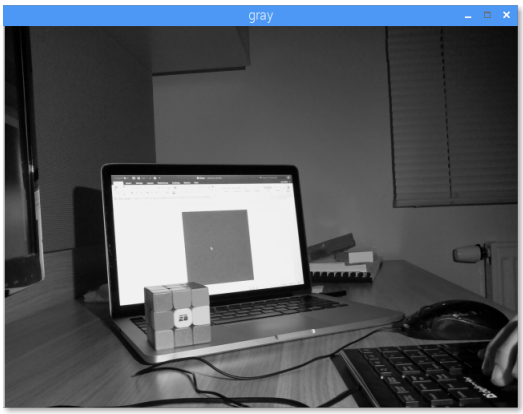
```
# Converting image color to Grayscale format
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Blurring the image to reduce external noise
blurred = cv2.GaussianBlur(gray, (9, 9), 0)

# Performing canny edge-detection
edged = cv2.Canny(blurred, 50, 150)

# Performing hough lines probalistic transform
lines = cv2.HoughLinesP(edged,1,np.pi/180,10,50,1)

# Drawing lines on input image
if(lines != None):
    for x1,y1,x2,y2 in lines[0]:
        cv2.line(resized,(x1,y1),(x2,y2),(0,255,0),2)

cv2.imshow("Line detection test", img)
cv2.waitKey(0)
```

In this part, the image color is converted to Grayscale format using **cv2.cvtColor()** function with first argument as the input file and second argument as convertion from BGR to Grayscale format using **cv2.COLOR_BGR2GRAY** predefined constant. Next, it is a known fact that during the photography there may be a noise (in case of using Pi camera the probability of that is high) which can cause the inaccuracy during the designated object detection. In order to alleviate the consequences of presence of noise the Gaussian based filter will be used to blur the image by **cv2.GaussianBlur()** function. Shortly saying, it leaves the contours of objects in the image. The first argument of this function is the input image, second argument is kernel size (width and height), third one is specification of standard deviation in X and Y axes (when 0 is provided this parameter will be calculated based on the kernel size). The next thing will to apply canny edge detection which will convert the image into binary image (containing only black and white pixels), and this is done by **cv2.Canny()** function. The first parameter of this function will be the input image (in this case the blurred image), and second and third parameters are minimum and maximum threshold values for canny edge detection, respectively. The last step would the designated object detection and can be done by applying some feature extraction function. The widely used one is Hough transform function [38] [39]. The simple case of using will be in detecting the lines using **cv2.HoughLinesP()** probabilistic transform function. HoughLinesP returns an array containing the start and end points of the lines detected on the image, where the first argument of the function is input binary-pixeled image, second and third arguments are distance and angle resolutions for the Hough transform, fourth argument is an accumulator threshold, and the fifth parameter is the minimum length of the line in pixels, and sixth argument is for defining the maximum gap granted between the lines. Finally, all lines will detected on original image

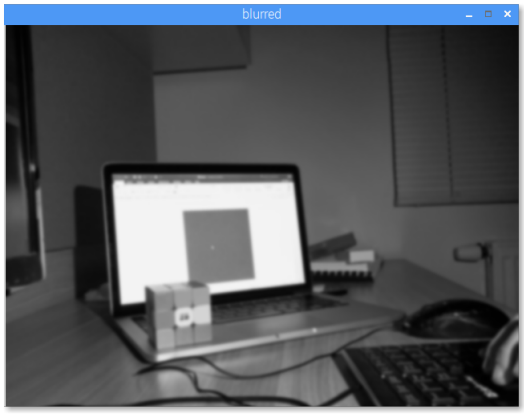


(a) Original photo taken by Pi camera

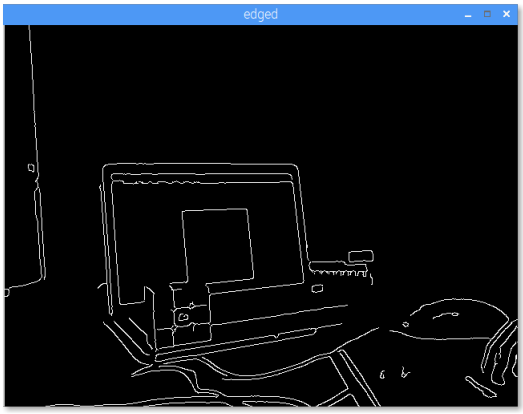


(b) Changing from RGB to Gray format

Figure 3.4: Preprocessing the photo taken by Pi camera



(a) Applying Gaussian blurring function



(b) Computing the contour map of the image

Figure 3.5: Processing the contour map

through the loop and the image will be shown on the screen.

On the Figure 3.4 and Figure 3.5 the processing of the image taken with Pi camera can be observed.

These are the foundation methods which will be used in the primary code of the project. The script for real-time object detection using OpenCV is available on Appendix B.

3.6 SITL Simulation

The SITL (software in the loop) simulator allows to simulate the drone behavior without any hardware [40]. It is a build using an ordinary C++ compiler, provid-

ing a native executable that allows to test the behaviour of the code without any hardware implementation. The software was installed according to official recommendations of ArduPilot open source platform. Here is represented the overview of SITL:

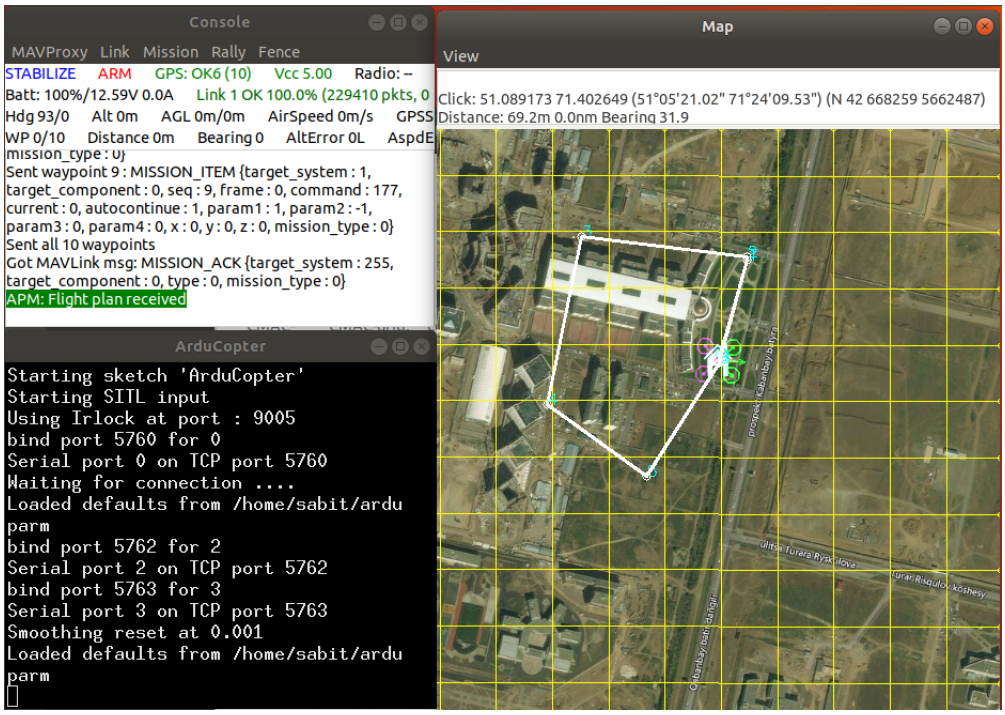


Figure 3.6: Overview of SITL

On this figure, the predefined mission was set on simulated flight controller of the drone consisting from 5 waypoints on university campus side. It is possible to load your own missions considering the parameters for drone such as altitude, roll, pitch and yaw. Moreover, there are several testing parameters that can be assessed for particular scenario. For instance, the behavior of the drone when it loses GPS lock can be tested in SITL. Additionally, drone's reaction to vibration can be tested on this software. Most importantly for us, as we are located in Astana, in the central part of Eurasia, where speed of wind can go up to 25 m/s, testing the behavior of drone's dynamics during windy conditions is essential. This can be done by setting the wind direction, speed and turbulence to test their effect on flight behaviour.

Chapter 4

Autopilot Script

4.1 Establishment of connection

As we have seen in previous chapters, the connection between companion computer and the flight controller can be established via several ways, including usage of proxy-server. However, for our purpose, it is required to authorize this connection from Python-written script. To start with, all the necessary packages and libraries should be imported:

```
# import the necessary packages
import argparse
import cv2
import numpy as np
import os
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from dronekit import *
import droneapi
import gps
import socket
import sys
```

Then, the following function for connection can be analyzed:

```
# connecting to the drone
def connect_to_drone():
    parser = argparse.ArgumentParser(description='Connecting to the vehicle')
    parser.add_argument('--connect', default='921600', help="vehicle connection target. Default '921600'")
    args = parser.parse_args()
    vehicle = connect('/dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v2.x_0-if00', baud = 921600, rate=6)
    return vehicle
```

In this function, first two lines show how command line arguments are parsed via "argparse" argument. Then, as we have stated before, the baudrate connection between companion computer and the flight controller is 921600 via serial port. Additionally, the name of the serial port is pre-defined for Pixhawk flight controllers. Nevertheless, the exact serial port can be tested via **"ls /dev/tty*"**. In the last line, the function returns the **"vehicle"** argument, which is the flight controller on the drone.

Next thing will be to change the mode of the flight controller. When the flight controller is powered up, it will be **"STABILIZE"** mode, in which the drone will remain in its current position. The mode should be changed to **"GUIDED"** mode, in which all commands received will be activated. Therefore, function that will change the mode of the flight controller will be as following:

```
# wait until the vehicle is in guided mode
def mode_change(vehicle):
    while (vehicle.mode.name != 'GUIDED'):
        time.sleep(2)
    return
```

Following that, if the prior fail-safe tests have been checked properly, then the drone can be armed. By arming it is meant the process of spinning the motors. If the drone will not be in armed position, then it would not follow any other command unless all prior fail-safe tests have been checked. Therefore, it is the uttermost importance to follow pre-flight test parameters in advance. Firstly, it will be checked whether or not the drone is armable. Then, if it is armable, the function will arm the drone motors. This can be done via the following function:

```
# checking whether the vehicle is armable or not
def checkArm(vehicle):
    while not vehicle.is_armable:
        print "waiting for vehicle to become armable"
        time.sleep(1)
    print "Armed"
    vehicle.armed = True
    while not vehicle.armed:
        print " Waiting for arming..."
        time.sleep(1)
    return
```

After that, drone should hover at some altitude after which it can start his main mission. In order to dot that, the following function can be written:

```
# takeoff to some altitude
def altitude_throttle(vehicle, alt):
    print "Throttling is in action"
    vehicle.simple_takeoff(alt)
    while not (vehicle.location.global_frame.alt>=alt*0.95):
        print " Altitude: ", vehicle.location.global_frame.alt
        time.sleep(1)
```

```
    return
```

The first argument is **"vehicle"** attribute obtained before during the connection. The second argument is **"alt"** which is the altitude (in meters) at which the drone should hover. Consequently, this attribute will be passed as an argument to predefined function **"simple_takeoff()"**. Last lines of **"while"** loop represents the current position of the drone. In other words, the current altitude of the drone will be printed on terminal until it reaches the desired altitude in GPS.

Lastly, the process of initializing the connection between companion computer and the flight controller can be finished by initializing the camera on companion computer. This can be done via the following function:

```
# initialize the camera
def initcam():
    camera = PiCamera()
    camera.resolution=(640,480)
    camera.framerate=50
    rawCapture=PiRGBArray(camera, size=(640,480))
    time.sleep(0.1)
    return camera, rawCapture
```

The first line defines the object **"camera"** as a state of Pi Camera. Second line represents resolution of the camera which is set to low-state 640x480 in order to increase the frame rate. With high resolution the frame rate cannot be increased due to computational restraints. The frame rate is set 50, which is quite close to not notice the fast movements on frames with naked human eye. Consequent line represents how Pi camera should capture the input data (in RGB form) with defined window size of 640x480. Lastly, this function will return two arguments: first one is the instance of Pi camera, and the second one is argument with raw input data.

These are the functions that are defined outside of the main class. They will be "called" in the main class. That would be the main functions which will initialize the establishment of connection between companion computer and the flight controller and prepare the drone to accomplish the main mission.

4.2 Computation of optimal UAV position

There will be used a function to define a position of UAV on space on which the bit-rate of data transmission of vehicular network will be maximized based on the distances of objects with respect to the drone's current spatial position. This function looks as following:

```
def opt_position(X_copter,Y_copter,D_position):
    SOW = 0 #initialize sum of weights to 0
    UNPu = [0, 0] #initizlize unnormalized UAV position to 0
    Ncar = 2 # number of cars
```

```
    w = [0]*Ncar #initialize weight i for each car
    p = [[0, 0], [0, 0]] #initialize position i for each car
    B = 20 * 10 ** 6#bandwidth (MHz)
    N = 10 **-18 #noise power (W/Hz)
    beta = 11 #correction factor (taken from Rahman-UAV paper)
    h = 20 #fixed UAV altitude

    #ASSIGN RATES AND POSITIONS FOR EACH CAR:
    Rate=[5*10**6, 19*10**6] # input the rate demand for each car
    p = D_position;    # input the position of each car

    #Calculating optimal UAV position
    w[0]=(N*B*(2**(beta*Rate[0]/B)-1)/(1.28*10**-9))**(1/4) #weight for car 1 (equation 2 in UAV paper)
    w[1]=(N*B*(2**(beta*Rate[1]/B)-1)/(1.28*10**-9))**(1/4) #weight for car 2 (equation 2 in UAV paper)
    p1_new = []
    for i in p[0]:
        p1_new.append(i*w[0])
    p2_new = []
    for i in p[1]:
        p2_new.append(i*w[1])
    UNPu=list(map(sum, zip(UNPu,p1_new,p2_new)))
    SOW=SOW+w[0]+w[1]
    pu = []
    for i in UNPu:
        pu.append(int(i/SOW))

    #UAV position
    return pu
```

This function was developed as part of another research project, hence our group did not thoroughly examined this topic. The output of the function will be the array of optimal position of UAV in 2D space (x and y axes) with pre-defined constant altitude.

4.3 Initialization of the mission

The main mission of the drone is to determine the right objects (vehicles) and to constantly update the optimal position of itself in space with respect to vehicles by maximizing the bit-rate data transmission. To start with, the cycling through raw input data frame by frame in order to determine the right objects will be done as following:

```
# start of the mission
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    image_array = frame.array
    status = "No Markers"
```

```
    center = "N/A"
    Deltas = "N/A"
    flight_update = "N/A"
    # make the image gray
    gray_image = cv2.cvtColor(image_array, cv2.COLOR_BGR2GRAY)

    # blurring the image
    blurred_image = cv2.GaussianBlur(gray_image, (7, 7), 0)

    # edges
    edged_image = cv2.Canny(blurred_image, 50, 150)

    # find contours in the edge map
    (contours, hierarchy) = cv2.findContours(edged_image.copy(), cv2.
    RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
```

The cycling is done by using **"for"** loop in continuous capture attribute of raw input data by frames. Frame itself is an NumPy array size of AxB which will be processed to find the right objects. The right objects (vehicles) will referred as markers further. The explanation for that will be provided in further sections. The consequent line represents the initialization of string **"status"** which will update the current status of drone, illustrating whether or not the markers are found in the current frame. Other subsequent strings **"center, Deltas, flight_update"** represents whether the center of camera is centralized with marker or not, the position of the center of camera, and whether or not there will be an update for the current path of the drone's flight, respectively. The subsequent three lines are discussed in Section 3.5 (Basics of OpenCV), which pre-processing functions. The last line represents the array of contours contained in edged map. As this array contains contours of all objects in the current frame, we will cycle through that array in order to find whether this frame contains the marker or not. This is done as following:

```
# cycle over the contours
for c in contours:
    # approximating the contour
    perimeter = cv2.arcLength(c, True)
    approximated = cv2.approxPolyDP(c, 0.01 * perimeter, True)
    # ensuring that the approximated contour is rectangular shape
    if len(approximated) >= 4 and len(approximated) <= 6:
        # compute the bounding box of the approximated contour and
        # use the bounding box to compute the aspect ratio
        (x, y, w, h) = cv2.boundingRect(approximated)
        aspectRatio = w / float(h)
        # compute the solidity of the original contour
        area = cv2.contourArea(c)
        hull = cv2.convexHull(c)
        hullArea = cv2.contourArea(hull)
        solidity = area / float(hullArea)
        # compute whether or not the width and height, solidity, and
        # aspect ratio of the approximated contour falls within the
```

```
    requirements
19      Dimensions = w > 25 and h > 25
20      Solidity_parameter = solidity > 0.9
21      AspectRatio_parameter = aspectRatio >= 0.8 and aspectRatio <= 1.2
```

Firstly, as we cycle through the array of contours, the Polygonal Approximation of digital curves using Ramer-Douglas-Peucker [41]. The overall hypothesis of algorithm is based on approximation of digital curve containing line segments to a similar curve with fewer number of points to construct it. As we commented above, the marker in our context is the geometric object of square shape. It is important to note that the usage of contour properties of simple geometric object is elementary compared to the method of training neural network to recognize the actual shapes of different vehicles, which also requires enormous computational power. Therefore, our group decided to use the basic properties of simple geometric objects such as square. To start with, the basic properties of square are following:

1. Square has four vertices
2. These vertices are approximately the same size

As the vertices are approximately of equal size, then the aspect ratio (ratio of square's width to the height) is close to 1. The **"perimeter"** parameter on the 4th line is the perimeter of the contour (or arc length). The subsequent line represents how this contour is approximated. The first argument is the contour itself, second one is called epsilon which is the maximum distance from actual contour to the approximated one, which is also an accuracy parameter: the smaller the value the more legitimate the approximation will be. As it can be seen, the value of approximation was chosen to be 1%. The last argument represents whether the curve is closed or not [42]. Then, we will check whether or not this approximated contour is closely object of rectangular shape. By this it is meant this approximated contour should contain from 4 to 6 points. To be exact, it should only contain 4 points, however there might be some noise introduced by fast movement of the drone in space. therefore, by implying that the approximated contour will contain only 4 points the function might discard the actual marker from the contour array. If the approximated contour will have from 4 to 6 points, then it will be proceeded further. The consequent lines compute the aspect ratio of the approximated contour. As it was stated earlier, aspect ratio is the ratio of width to height of rectangular object, according to the OpenCV Documentation [42]. Consequently, the area of approximated contour will be computed via **"cv2.contourArea()"** function. Then, the approximated contour is processed via convexity Hull, which will define the defects of contour and corrects it. After that the area of convexity Hull will be computed via same function, and as an argument there is inserted the convexity Hull defined as **"cv2.convexHull()"**. The last three lines of code indicates whether the parameters of the approximated contour falls with-in our desired criteria: firstly,

the contour (width and height) of the object should have at least 25 pixels, which which will guarantee that small rectangular-shaped objects will be discarded; secondly, solidity of the contour should be larger than 0.9; thirdly, the aspect ratio of the object's contour should lie between 0.8 and 1.2, which will guarantee that the recognized object is close to rectangular shape.

4.4 Computation of optimal UAV position

If the recognized object possess the required parameters, it will be processed further. The further processing can be analyzed in the following passage:

```
if Dimensions and Solidity_parameter and AspectRatio_parameter:
    # drawing boundaries around the marker and updating the status
    cv2.drawContours(image_array, [approximated], -1, (0, 0, 255), 4)
    status = "Marker(s) Acquired"
    # computing the center of the approximated contour region and drawing the crosshairs
    M = cv2.moments(approximated)
    (X_center, Y_center) = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
    (X1, X2) = (int(X_center - (w * 0.15)), int(X_center + (w * 0.15)))
    (Y1, Y2) = (int(Y_center - (h * 0.15)), int(Y_center + (h * 0.15)))
    cv2.line(image_array, (X1, Y_center), (X2, Y_center), (0, 0, 255), 3)
    cv2.line(image_array, (X_center, Y1), (X_center, Y2), (0, 0, 255), 3)
    (DX,DY) = (X_center-320,Y_center-240)
    D_position = [DX, DY]
    (X_corrected,Y_corrected) = (DX*(.0015*alt+.00005), DY*(.0015*alt+.00005))
```

After ensuring that the object possesses the required parameters, the **"crosshairs"** (bounding lines in the center of the rectangular contour) will be drawn. Consequently, the position parameters of the object can be computed. The parameters **"X_center, Y_center"** represents the central position parameters of the recognized marker. In the same manner, **"DX, DY"** variables represent spatial position of the object on the whole frame, which 640x480. Lastly, **"X_corrected,Y_corrected"** variables represent the difference in the position of the drone and the position of the objects, which should be converted into pixels.

After computing the location of the object (marker) with respect to the drone, the optimal position can be computed with **"opt_position()"** function. However, if there will be multiple objects (markers), then there should be introduced new algorithm for computing the optimal position. To start with, the **"Action mode"** was developed, which ensures that whenever the multiple objects are being recognized the optimal location will be calculated taking into account the current positions of the objects. In the case there won't be any objects to be recognized, the drone will activate **"Waiting mode"**, where it will drift on top of the last recognized object. Depending whether on frame the object will be recognized (with respect to the

centre of the frame), it will activate the **"Action mode"** with respective update in the flight's path.

4.5 Frame Information

Lastly, the frame's information will be updated on continuous manner. This can be in the following manner:

```
cv2.putText(image_array, status, (20, 30), cv2.FONT_HERSHEY_SIMPLEX,
    0.5,(0, 0, 255), 2)
cv2.putText(image_array, center, (20, 50), cv2.FONT_HERSHEY_SIMPLEX,
    0.5,(0, 0, 255), 2)
cv2.putText(image_array, Deltas, (20, 70), cv2.FONT_HERSHEY_SIMPLEX,
    0.5,(0, 0, 255), 2)
cv2.putText(image_array, flight_update, (20, 90), cv2.FONT_HERSHEY_SIMPLEX
    , 0.5,(0, 0, 255), 2)
cv2.putText(image_array, mode, (20, 110), cv2.FONT_HERSHEY_SIMPLEX,
    0.5,(0, 0, 255), 2)
cv2.rectangle(image_array,(290,200),(350,260),(0,0,225),2)
cv2.imshow("Frame", image_array)
key = cv2.waitKey(1) & 0xFF
rawCapture.truncate(0)
if key == ord('q'):
    break
```

If the **"q"** character will be pressed, the process will quit from the main loop and all windows (including video stream) will shutdown. Moreover, when this will be the case, the companion computer will stop sending commands to the flight controller, and the last one will execute the pre-defined mode **"Return To Launch"**. In this mode, the drone will return to its home position when there is no currently running mission.

4.6 Marker Detection: Results

The results of running the script from the Appendix B can be analyzed on the figures below. As it can be seen, the "Action mode" is activated when the recognized object (marker) is captured on the frame, and respective correction to the flight's path are being displayed on the frame and taken into action. From the second figure, it can be seen that the marker is acquired, however it was previously recognized, hence it is entering into "Waiting mode". On the last figure the view of the marker can be seen, which is distinguished rectangle with sides of approximately 10 cm.

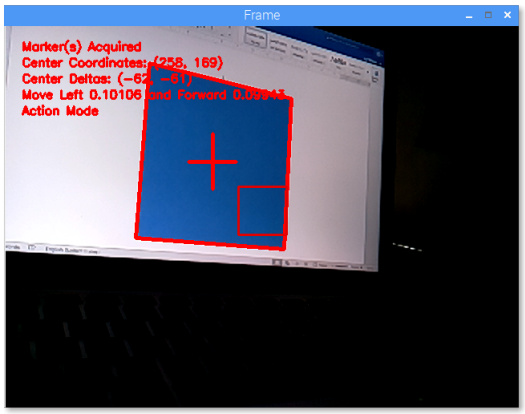


Figure 4.1: Action Mode of the Autopilot script

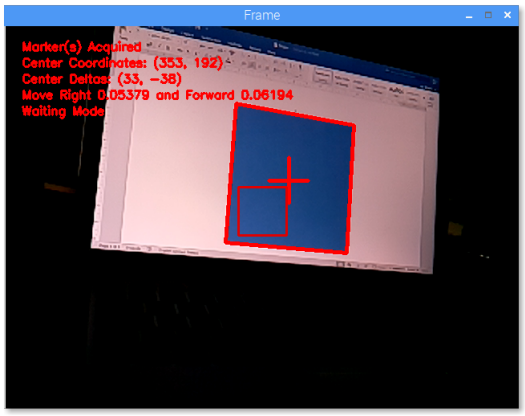


Figure 4.2: Waiting Mode of the Autopilot Script

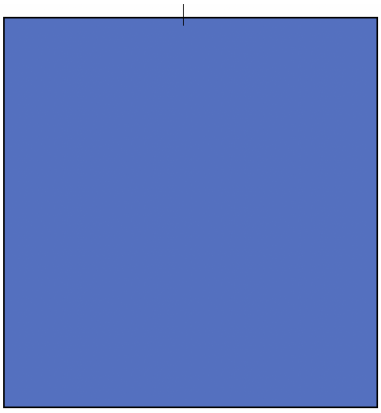


Figure 4.3: The view of the marker

Chapter 5

Conclusion

5.1 Results

The project proposes design of the drone autopilot system using computer based vision for object tracking. Rasbperry Pi is used as an on-board computer for the system and it executes and sends all the commands to Pixhawk flight controller. Pixhawk flight controller receives the commands from Raspberry Pi and triggers motors of the drone and gives drone designated direction. Drone is navigated using OpenCV, which is open source computer vision, by recognizing and tracking certain objects. The direction and current location of the drone are sent and monitored on the ground control station. Simple Pi camera is used for object recognition.

Usage of the drone in VLC based vehicular network might increase the reliability of the system, as VLC is short-ranged communication type with sensitivity to objects appearing on the light ray, and blocking it. In the VLC based network drones is used as an intermediate step between vehicles to make sure that the connection between moving cars on the road is stable and secure, as close visual contact with light source in the network is needed for some third party members to enter the connection. Additionally, usage of the drone as an intermediate step between vehicle communication might prevent the problem of the signal blocking in high dense traffic, where cars are actively moving between each other and might block the communication. Also the connection between vehicles on the road might increase the safety on the road as it assures data transfer between vehicles.

Basic take off commands has been tested without Rasbperry Pi with the new latest Pixhawk firmware - CHIBIOS v. 3.6.7

Figure 5.1: Basic take off to 1m command with latest firmware

Drone has been held for safety measurements.

As a result autopilot system with object recognition has been designed and the block diagram of it has been shown. Software simulation of the designed autopilot system has been done, analyzed and merged with OpenCV computer vision code. At the time object is recognized, system determines its location relative to UAV and sends command to Pixhawk in which directions to move in order to follow recognized object. Additionally, code for optimum location above the recognized object, depending on the link and bitrate demand, developed by our supervisor's master students group has been added to our system, which can be used in real VLC networks with UAV as the base station. System should send commands generated from code on Raspberry Pi to Pixhawk flight controller which would control the directions of the drone. At the same time system has not been tested fully on hardware due to issues related with GPS configuration, which cannot be solved due to poor GPS coverage in the region and cheap and poor GPS module purchased from China. More precise GPS module could solve the problem, however it is relatively expensive and would exceed the capstone budget. On the other hand, the system has been successfully designed and tested on the software and interface connection between Raspberry Pi and Pixhawk flight controller has been fully established.

5.2 Problems Faced

Main problems faced were during hardware testing of the autopilot system. Pixhawk flight controller requires specific arming checks before the drone to fly, which can not be skipped, avoided or turned off. Firstly there was an interfacing problem between Raspberry Pi and Pixhawk flight controller where proper firmware should have been chosen. There were two firmware options: CHIBIOS v. 3.6.7 (lat-

est stable release) and NUTTX v. 3.6.7 (older one), in the latest CHIBIOS version GPS checking before arming still has been mandatory, however RC checking could have been skipped, but the problem was in interfacing between latest CHIBIOS and Raspberry Pi as they has been configured at different baud rate which couldn't have been changed in that firmware, and, therefore Pixhawk did not understand commands sent from Raspberry Pi. Thus, it has been decided to choose NUTTX firmware for hardware simulation, however it had its own features and problems. Before the actual flight, the flight controller perform pre-arm checks which are mandatory. This include RC calibration, sensors configuration, obtaining proper GPS lock (with sufficient number of satellites). In order to get armed, before starting to receive commands, Pixhawk at NUTTX firmware required RC calibration with RC remote controller which we did not have. Thus, we have contacted the FabLab at Nazarbayev University Research and Innovation System(NURIS) and used RC remote controller in the lab in order to calibrate the UAV. Another problem was with GPS module, as before getting armed Pixhawk had to obtain its coordinates for UAV to start flying. GPS signal has not been receiving from the satellites in the dormitory or any other buildings, due to poor and cheap GPS module, therefore the work on hardware testing has been continued outside in some open areas away from buildings. Even when the GPS signal has been received, UAV could not properly indicate its exact position, and therefore would not start to fly. The problem with imprecise GPS location obtained comes from bad GPS signal in the region and poor GPS module. The problem with GPS configuration can be seen on the figure below, where UAV determines its current altitude relative to ground in negative position.

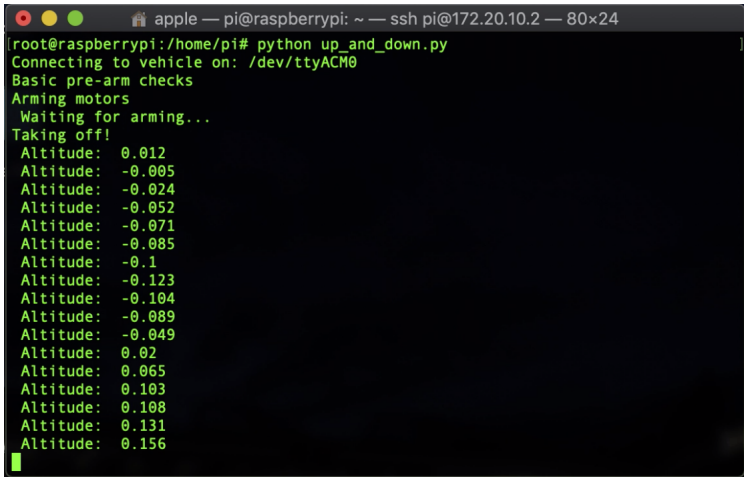


Figure 5.2: Basic takeoff command representing bad GPS configuration

5.3 Future Works

As a future work system can be fully implemented on hardware with more expensive components, especially GPS module. Additionally, current battery can maintain stable flight of the drone only for 15-20 minutes and for longer flights more expensive batteries can be purchased. More expensive battery with higher capacitor may increase the weight of the system and in case of overweight more powerful motors should also be purchased.

Furthermore, the applicable approach with usage of the VLC-based data transmission in vehicular network can be introduced with proper hardware development.

Bibliography

[1] H. Chao, Y. Cao, and Y. Chen, "Autopilots for small unmanned aerial vehicles: A survey", *International Journal of Control, Automation and Systems*, vol. 8, no. 1, pp. 36–44, 2010.

[2] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones", *Nature*, vol. 521, no. 7553, p. 460, 2015.

[3] B. H. Y. Alsalam, K. Morton, D. Campbell, and F. Gonzalez, "Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture", in *Aerospace Conference, 2017 IEEE*, IEEE, 2017, pp. 1–12.

[4] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, 2017.

[5] S. Ward, J. Hensler, B. Alsalam, and L. F. Gonzalez, "Autonomous uavs wildlife detection using thermal imaging, predictive navigation and computer vision", in *Proceedings of the 2016 IEEE Aerospace Conference*, IEEE, 2016, pp. 1–8.

[6] B. Parno and A. Perrig, "Challenges in securing vehicular networks", in *Workshop on hot topics in networks (HotNets-IV)*, Maryland, USA, 2005, pp. 1–6.

[7] A. Ashok, "Position: Dronevlc: Visible light communication for aerial vehicular networking", in *Proceedings of the 4th ACM Workshop on Visible Light Communication Systems*, ACM, 2017, pp. 29–30.

[8] W. Fawaz, R. Atallah, C. Assi, and M. Khabbaz, "Unmanned aerial vehicles as store-carry-forward nodes for vehicular networks", *IEEE Access*, vol. 5, pp. 23 710–23 718, 2017.

[9] S. A. Hadiwardoyo, E. Hernández-Orallo, C. T. Calafate, J. C. Cano, and P. Manzoni, "Experimental characterization of uav-to-car communications", *Computer Networks*, vol. 136, pp. 105–118, 2018.

[10] G. Fasano, D. Accardo, A. E. Tirri, A. Moccia, and E. De Lellis, "Radar/electro-optical data fusion for non-cooperative uas sense and avoid", *Aerospace Science and Technology*, vol. 46, pp. 436–450, 2015.

[11] T. Komine and M. Nakagawa, "Fundamental analysis for visible-light communication system using led lights", *IEEE Transactions on Consumer Electronics*, vol. 50, no. 1, pp. 100–107, 2004, ISSN: 0098-3063. DOI: 10.1109/TCE.2004.1277847.

[12] A. Jovicic, J. Li, and T. Richardson, "Visible light communication: Opportunities, challenges and the path to market", *IEEE Communications Magazine*, vol. 51, no. 12, pp. 26–32, 2013, ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6685754.

[13] P. P. Salian, S. Prabhu, P. Amin, S. K. Naik, and M. K. Parashuram, "Visible light communication", in *2013 Texas Instruments India Educators' Conference*, 2013, pp. 379–383. DOI: 10.1109/TIIEC.2013.74.

[14] S. A.S. O. Al-Kadhim and I. A. Collage, "Communicating with raspberry pi via mavlink", 2016.

[15] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with opencv", *Communications of the ACM*, vol. 55, no. 6, pp. 61–69, 2012.

[16] B. T. M. Leong, S. M. Low, and M. P.-L. Ooi, "Low-cost microcontroller-based hover control design of a quadcopter", *Procedia Engineering*, vol. 41, pp. 458–464, 2012.

[17] A. K. Yadav and P. Gaur, "Ai-based adaptive control and design of autopilot system for nonlinear uav", *Sadhana*, vol. 39, no. 4, pp. 765–783, 2014.

[18] M. Liu, G. Egan, and Y. Ge, "Identification of attitude flight dynamics for an unconventional uav", in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, IEEE, 2006, pp. 3243–3248.

[19] J. Liénard, A. Vogs, D. Gatziolis, and N. Strigul, "Embedded, real-time uav control for improved, image-based 3d scene reconstruction", *Measurement*, vol. 81, pp. 264–269, 2016.

[20] G. Crespo, G. Glez-de Rivera, J. Garrido, and R. Ponticelli, "Setup of a communication and control systems of a quadrotor type unmanned aerial vehicle", in *Design of Circuits and Integrated Circuits (DCIS), 2014 Conference on*, IEEE, 2014, pp. 1–6.

[21] R. P. Documentation. (2018). Camera module, [Online]. Available: https://www.raspberrypi.org/documentation/usage/camera/.

[22] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms", in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, IEEE, 2015, pp. 6235–6240.

[23] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "The pixhawk open-source computer vision framework for mavs", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. 1, p. C22, 2011.

[24] V. Kortunov, O. Mazurenko, A. Gorbenko, W. Mohammed, and A. Hussein, "Review and comparative analysis of mini-and micro-uav autopilots", in *2015 IEEE international conference actual problems of unmanned aerial vehicles developments (APUAVD)*, IEEE, 2015, pp. 284–289.

[25] A. Patelli and L. Mottola, "Model-based real-time testing of drone autopilots", in *Proceedings of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, ACM, 2016, pp. 11–16.

[26] Z. Yang, F. Lin, and B. M. Chen, "Survey of autopilot for multi-rotor unmanned aerial vehicles", in *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2016, pp. 6122–6127.

[27] H. Eisenbeiss and M. Sauerbier, "Investigation of uav systems and flight modes for photogrammetric applications", *The Photogrammetric Record*, vol. 26, no. 136, pp. 400–421, 2011.

[28] A. Hernandez, H. Murcia, C. Copot, and R. De Keyser, "Model predictive path-following control of an ar. drone quadrotor", in *Proceedings of the XVI Latin American Control Conference (CLCA'14), Cancun, Quintana Roo, Mexico*, 2014, pp. 14–17.

[29] L. Feng and Q. Fangchao, "Research on the hardware structure characteristics and ekf filtering algorithm of the autopilot pixhawk", in *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, IEEE, 2016, pp. 228–231.

[30] S. Brahmbhatt, "Embedded computer vision: Running opencv programs on the raspberry pi", in *Practical OpenCV*, Springer, 2013, pp. 201–218.

[31] L. Ujjainiya and M. K. Chakravarthi, "Raspberry—pi based cost effective vehicle collision avoidance system using image processing", *ARPN J. Eng. Appl. Sci*, vol. 10, no. 7, 2015.

[32] N. Butcher, A. Stewart, and S. Biaz, "Securing the mavlink communication protocol for unmanned aircraft systems", *Appalachian State University, Auburn University, USA*, 2013.

[33] J. A. Marty, "Vulnerability analysis of the mavlink protocol for command and control of unmanned aircraft", AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF ..., Tech. Rep., 2013.

[34] S. Atoev, K.-R. Kwon, S.-H. Lee, and K.-S. Moon, "Data analysis of the mavlink communication protocol", in *2017 International Conference on Information Science and Communications Technologies (ICISCT)*, IEEE, 2017, pp. 1–3.

[35] M. Documentation. (2019). Micro air vehicle communication protocol, [Online]. Available: `https://mavlink.io/en/`.

[36] H. Choi, M. Geeves, B. Alsalam, and F. Gonzalez, "Open source computer-vision based guidance system for uavs on-board decision making", in *2016 IEEE Aerospace Conference*, IEEE, 2016, pp. 1–5.

[37] M. O. Milhouse, "Framework for autonomous delivery drones", in *Proceedings of the 4th Annual ACM Conference on Research in Information Technology*, ACM, 2015, pp. 1–4.

[38] P. E. Hart, "How the hough transform was invented [dsp history]", *IEEE Signal Processing Magazine*, vol. 26, no. 6, 2009.

[39] O. Barinova, V. Lempitsky, and P. Kholi, "On detection of multiple object instances using hough transforms", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1773–1784, 2012.

[40] O. Meister, N. Frietsch, J. Seibold, and G. F. Trommer, "Software-in-the-loop simulation for small autonomous vtol uav with teaming capability", *Institute of Systems Optimization, Germany*, 2008.

[41] A. Salhi, B. Minaoui, and M. Fakir, "Robust automatic traffic signs detection using fast polygonal approximation of digital curves", in *2014 International Conference on Multimedia Computing and Systems (ICMCS)*, IEEE, 2014, pp. 433–437.

[42] O. Documentation. (2019). Opencv documentation, [Online]. Available: `https://docs.opencv.org/3.1.0/index.html`.

Appendix A

Main Code

```
# import the necessary packages
import argparse
import cv2
import numpy as np
import os
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from dronekit import *
import droneapi
import gps
import socket
import sys


# connecting to the drone
def connect_to_drone():
    parser = argparse.ArgumentParser(description='Connecting to the vehicle')
    parser.add_argument('--connect', default='921600', help="vehicle connection target. Default '921600'")
    args = parser.parse_args()
    vehicle = connect('/dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v2.x_0-if00', baud = 921600, rate=6)
    return vehicle

# wait until the vehicle is in guided mode
def mode_change(vehicle):
    while (vehicle.mode.name != 'GUIDED'):
        time.sleep(2)
    return

# checking whether the vehicle is armable or not
def checkArm(vehicle):
```

```
    while not vehicle.is_armable:
        print "waiting for vehicle to become armable"
        time.sleep(1)
    print "Armed"
    vehicle.armed = True
    while not vehicle.armed:
        print " Waiting for arming..."
        time.sleep(1)
    return

# takeoff to some altitude
def altitude_throttle(vehicle, alt):
    print "Throttling is in action"
    vehicle.simple_takeoff(alt)
    while not (vehicle.location.global_frame.alt>=alt*0.95):
        print " Altitude: ", vehicle.location.global_frame.alt
        time.sleep(1)
    return

# initialize the camera
def initcam():
    camera = PiCamera()
    camera.resolution=(640,480)
    camera.framerate=50
    rawCapture=PiRGBArray(camera, size=(640,480))
    time.sleep(0.1)
    return camera, rawCapture

def opt_position(X_copter,Y_copter,D_position):
    SOW = 0 #initialize sum of weights to 0
    UNPu = [0, 0] #initizlize unnormalized UAV position to 0
    Ncar = 2 # number of cars
    w = [0]*Ncar #initialize weight i for each car
    p = [[0, 0], [0, 0]] #initialize position i for each car
    B = 20 * 10 ** 6#bandwidth (MHz)
    N = 10 **-18 #noise power (W/Hz)
    beta = 11 #correction factor (taken from Rahman-UAV paper)
    h = 20 #fixed UAV altitude

    #ASSIGN RATES AND POSITIONS FOR EACH CAR:
    Rate=[5*10**6, 19*10**6] # input the rate demand for each car
    p = D_position;   # input the position of each car

    #Calculating optimal UAV position
    w[0]=(N*B*(2**(beta*Rate[0]/B)-1)/(1.28*10**-9))**(1/4) #weight for car 1 (equation 2 in UAV paper)
    w[1]=(N*B*(2**(beta*Rate[1]/B)-1)/(1.28*10**-9))**(1/4) #weight for car 2 (equation 2 in UAV paper)
    p1_new = []
    for i in p[0]:
        p1_new.append(i*w[0])
```

```
    p2_new = []
    for i in p[1]:
        p2_new.append(i*w[1])
    UNPu=list(map(sum, zip(UNPu,p1_new,p2_new)))
    SOW=SOW+w[0]+w[1]
    pu = []
    for i in UNPu:
        pu.append(int(i/SOW))

    #UAV position
    return pu

def main():
        #variable init
        X_copter = 0
        Y_copter = 0
        mode = "N/A"
        alt = 20 #altitude (meters)

        # connect to drone
        vehicle = connect_to_drone()

        # initializing  camera
        (camera, rawCapture) = initcam()

        # change mode to guided
        mode_change(vehicle)

        # checking arm parameter and arm the drone
        checkArm(vehicle)

        # takeoff to some altitude (1.5 meter)
        altitude_throttle(vehicle, alt)

        # start of the mission
        for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
                image_array = frame.array
                status = "No Markers"
                center = "N/A"
                Deltas = "N/A"
                flight_update = "N/A"
                # make the image gray
                gray = cv2.cvtColor(image_array, cv2.COLOR_BGR2GRAY)

                # blurring the image
                blurred = cv2.GaussianBlur(gray, (7, 7), 0)

                # edges
                edged = cv2.Canny(blurred, 50, 150)

```

```
131                 # find contours in the edge map
132                 (contours, hierarchy) = cv2.findContours(edged.copy(), cv2
    .RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
133
134                 # cycle over the contours
135                 for c in contours:
136                         # approximate the contour
137                         perimeter = cv2.arcLength(c, True)
138                         approximated = cv2.approxPolyDP(c, 0.01 *
    perimeter, True)
139                         # ensure that the approximated contour is
    rectangular shape
140                         if len(approximated) >= 4 and len(approximated) <=
     6:
141                         # compute the bounding box of the approximated
    contour and
142                         # use the bounding box to compute the aspect ratio
143                                 (x, y, w, h) = cv2.boundingRect(
    approximated)
144                                 aspectRatio = w / float(h)
145                                 # compute the solidity of the original
    contour
146                                 area = cv2.contourArea(c)
147                                 hullArea = cv2.contourArea(cv2.convexHull(
    c))
148                                 solidity = area / float(hullArea)
149                                 # compute whether or not the width and
    height, solidity, and
150                                 # aspect ratio of the contour falls within
     appropriate bounds
151                                 Dimensions = w > 25 and h > 25
152                                 Solidity_parameter = solidity > 0.9
153                                 AspectRatio_parameter = aspectRatio >= 0.8
     and aspectRatio <= 1.2
154                                 # ensure that the contour passes all our
    tests
155                                 if Dimensions and Solidity_parameter and
    AspectRatio_parameter:
156                                         # draw an outline around the
    marker and update the status text
157                                         cv2.drawContours(image_array, [
    approximated], -1, (0, 0, 255), 4)
158                                         status = "Marker(s) Acquired"
159
160                                         # compute the center of the
    contour region and draw the
161                                         # crosshairs
162                                         M = cv2.moments(approximated)
163                                         (X_center, Y_center) = (int(M["m10
    "] / M["m00"]), int(M["m01"] / M["m00"]))
164                                         (X1, X2) = (int(X_center - (w *
```

```
0.15)), int(X_center + (w * 0.15)))
                                        (Y1, Y2) = (int(Y_center - (h *
0.15)), int(Y_center + (h * 0.15)))
                                        cv2.line(image_array, (X1,
Y_center), (X2, Y_center), (0, 0, 255), 3)
                                        cv2.line(image_array, (X_center,
Y1), (X_center, Y2), (0, 0, 255), 3)
                                        (DX,DY) = (X_center-320,Y_center
-240)
                                        D_position = [DX, DY]
                                        (X_corrected,Y_corrected) = (DX
*(.0015*alt+.00005), DY*(.0015*alt+.00005))
                                        if not (Y_copter>-60 and (-60<
X_copter<60)):
                                                # Action mode
                                                if (DY<-30): # check if
whether it could be backwards
                                                        center = "Center
Coordinates: ({centerx}, {centery})".format(centerx=str(X_center),
centery=str(Y_center))
                                                        Deltas = "Center
Deltas: ({deltaX}, {deltaY})".format(deltaX=str(DX),deltaY=str(DY))
                                                        if DX>0:
                                                            if DY>0: #
bottom right of the frame
flight_update = "Move Right {magdelx} and Backward {magdelY}".format(
magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                            else: # top
right of screen
flight_update = "Move Right {magdelx} and Forward {magdelY}".format(
magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else:
                                                            if DY>0: #
bottom left of the frame
flight_update = "Move Left {magdelx} and Backward {magdelY}".format(
magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                            else: # top
left of the frame
flight_update = "Move Left {magdelx} and Forward {magdelY}".format(
magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        # calculating
optimal position of the drone
                                                        opt_position(
X_copter,Y_copter,DX,DY)
                                                        X_copter = pu[1]
                                                        Y_copter = pu[2]
                                                        mode = "Action
```

```
Mode"

                                        elif Y_copter>-60:
                                                if (DX<-30 and DY<-30) or (DX<-30 and DY>30) or (DX>30 and DY<-30) or (DX>30 and DY>30) or (DX<30 and DX>-30 and DY>30) or (DX<30 and DX>-30 and DY<-30) or (DY<30 and DY>-30 and DX>30) or (DY<30 and DY>-30 and DX<-30):
                                                        center = "Center Coordinates: ({centerx}, {centery})".format(centerx=str(X_center), centery=str(Y_center))
                                                        Deltas = "Center Deltas: ({deltaX}, {deltaY})".format(deltaX=str(DX),deltaY=str(DY))
                                                        # draw the status text on the frame
                                                        if DX>0:
                                                            if DY>0: # bottom right of the frame
                                                                flight_update = "Move Right {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                            else: # top right of the frame
                                                                flight_update = "Move Right {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else:
                                                            if DY>0: # bottom left of the frame
                                                                flight_update = "Move Left {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                            else: # top left of the frame
                                                                flight_update = "Move Left {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        # calculating optimal position of the drone
                                                        opt_position(X_copter,Y_copter,DX,DY)
                                                        X_copter = pu[1]
                                                        Y_copter = pu[2]
                                                        mode = "Waiting Mode"
                                        else:
                                                if (DX<-30 and DY<-30) or (DX<-30 and DY>30) or (DX>30 and DY<-30) or (DX>30 and DY>30) or (DX<30 and DX>-30 and DY>30) or (DX<30 and DX>-30 and DY<-30) or (DY<30 and DY>-30 and DX>30) or (DY<30 and DY>-30 and DX<-30):
                                                        center = "Center Coordinates: ({centerx}, {centery})".format(centerx=str(X_center),
```

```
centery=str(Y_center))
                                                    Deltas = "Center Deltas: ({deltaX}, {deltaY})".format(deltaX=str(DX),deltaY=str(DY))
                                                    if DX>0:
                                                        if DY>0: # bottom right of the frame
                                                            flight_update = "Move Right {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else: # top right of the frame
                                                            flight_update = "Move Right {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                    else:
                                                        if DY>0: # bottom left of the frame
                                                            flight_update = "Move Left {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else: # top left of the frame
                                                            flight_update = "Move Left {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                    # calculating optimal position of the drone
                                                    opt_position(X_copter,Y_copter,DX,DY)
                                                    X_copter = pu[1]
                                                    Y_copter = pu[2]
                                                    mode = "Waiting Mode"



                cv2.putText(image_array, status, (20, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                            (0, 0, 255), 2)
                cv2.putText(image_array, center, (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                            (0, 0, 255), 2)
                cv2.putText(image_array, Deltas, (20, 70), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                            (0, 0, 255), 2)
                cv2.putText(image_array, flight_update, (20, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                            (0, 0, 255), 2)
                cv2.putText(image_array, mode, (20, 110), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                            (0, 0, 255), 2)
```

```
                cv2.rectangle(image_array,(290,200),(350,260),(0,0,225),2)
                cv2.imshow("Frame", image_array)
                key = cv2.waitKey(1) & 0xFF
                rawCapture.truncate(0)
                if key == ord('q'):
                    break
        cv2.destroyAllWindows()

        return

if __name__ == "__main__":
        main()
```

Appendix B

Real-time Object Detection

```
# import the necessary packages
import argparse
import cv2
import numpy as np
import os
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from dronekit import *
import droneapi
import gps
import socket
import sys

# initialize the camera
def initcam():
    camera = PiCamera()
    camera.resolution=(640,480)
    camera.framerate=50
    rawCapture=PiRGBArray(camera, size=(640,480))
    time.sleep(0.1)
    return camera, rawCapture

def opt_position(X_copter,Y_copter,D_position):
    SOW = 0 #initialize sum of weights to 0
    UNPu = [0, 0] #initizlize unnormalized UAV position to 0
    Ncar = 2 # number of cars
    w = [0]*Ncar #initialize weight i for each car
    p = [[0, 0], [0, 0]] #initialize position i for each car
    B = 20 * 10 ** 6#bandwidth (MHz)
    N = 10 **-18 #noise power (W/Hz)
    beta = 11 #correction factor (taken from Rahman-UAV paper)
    h = 20 #fixed UAV altitude

```

```
    #ASSIGN RATES AND POSITIONS FOR EACH CAR:
    Rate=[5*10**6, 19*10**6] # input the rate demand for each car
    p = D_position;    # input the position of each car

    #Calculating optimal UAV position
    w[0]=(N*B*(2**(beta*Rate[0]/B)-1)/(1.28*10**-9))**(1/4) #weight for car 1 (equation 2 in UAV paper)
    w[1]=(N*B*(2**(beta*Rate[1]/B)-1)/(1.28*10**-9))**(1/4) #weight for car 2 (equation 2 in UAV paper)
    p1_new = []
    for i in p[0]:
        p1_new.append(i*w[0])
    p2_new = []
    for i in p[1]:
        p2_new.append(i*w[1])
    UNPu=list(map(sum, zip(UNPu,p1_new,p2_new)))
    SOW=SOW+w[0]+w[1]
    pu = []
    for i in UNPu:
        pu.append(int(i/SOW))

    #UAV position
    return pu

def main():
        #variable init
        X_copter = 0
        Y_copter = 0
        mode = "N/A"
        alt = 20 #altitude (meters)

  (camera, rawCapture) = initcam()

        # start of the mission
        for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
                image_array = frame.array
                status = "No Markers"
                center = "N/A"
                Deltas = "N/A"
                flight_update = "N/A"
                # make the image gray
                gray = cv2.cvtColor(image_array, cv2.COLOR_BGR2GRAY)

                # blurring the image
                blurred = cv2.GaussianBlur(gray, (7, 7), 0)

                # edges
                edged = cv2.Canny(blurred, 50, 150)

                # find contours in the edge map
```

```
            (contours, hierarchy) = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

            # cycle over the contours
            for c in contours:
                # approximate the contour
                perimeter = cv2.arcLength(c, True)
                approximated = cv2.approxPolyDP(c, 0.01 * perimeter, True)
                # ensure that the approximated contour is rectangular shape
                if len(approximated) >= 4 and len(approximated) <= 6:
                # compute the bounding box of the approximated contour and
                # use the bounding box to compute the aspect ratio
                    (x, y, w, h) = cv2.boundingRect(approximated)
                    aspectRatio = w / float(h)
                    # compute the solidity of the original contour
                    area = cv2.contourArea(c)
                    hullArea = cv2.contourArea(cv2.convexHull(c))
                    solidity = area / float(hullArea)
                    # compute whether or not the width and height, solidity, and
                    # aspect ratio of the contour falls within appropriate bounds
                    Dimensions = w > 25 and h > 25
                    Solidity_parameter = solidity > 0.9
                    AspectRatio_parameter = aspectRatio >= 0.8 and aspectRatio <= 1.2
                    # ensure that the contour passes all our tests
                    if Dimensions and Solidity_parameter and AspectRatio_parameter:
                        # draw an outline around the marker and update the status text
                        cv2.drawContours(image_array, [approximated], -1, (0, 0, 255), 4)
                        status = "Marker(s) Acquired"

                        # compute the center of the contour region and draw the
                        # crosshairs
                        M = cv2.moments(approximated)
                        (X_center, Y_center) = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
                        (X1, X2) = (int(X_center - (w * 0.15)), int(X_center + (w * 0.15)))
```

```
                                        (Y1, Y2) = (int(Y_center - (h * 0.15)), int(Y_center + (h * 0.15)))
                                        cv2.line(image_array, (X1, Y_center), (X2, Y_center), (0, 0, 255), 3)
                                        cv2.line(image_array, (X_center, Y1), (X_center, Y2), (0, 0, 255), 3)
                                        (DX,DY) = (X_center-320,Y_center-240)
                                        D_position = [DX, DY]
                                        (X_corrected,Y_corrected) = (DX*(.0015*alt+.00005), DY*(.0015*alt+.00005))
                                        if not (Y_copter>-60 and (-60<X_copter<60)):
                                                # Action mode
                                                if (DY<-30): # check if whether it could be backwards
                                                        center = "Center Coordinates: ({centerx}, {centery})".format(centerx=str(X_center), centery=str(Y_center))
                                                        Deltas = "Center Deltas: ({deltaX}, {deltaY})".format(deltaX=str(DX),deltaY=str(DY))
                                                        if DX>0:
                                                                if DY>0: # bottom right of the frame
                                                                        flight_update = "Move Right {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                                else: # top right of screen
                                                                        flight_update = "Move Right {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else:
                                                                if DY>0: # bottom left of the frame
                                                                        flight_update = "Move Left {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                                else: # top left of the frame
                                                                        flight_update = "Move Left {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        # calculating optimal position of the drone
                                                        opt_position(X_copter,Y_copter,DX,DY)
                                                        X_copter = pu[1]
                                                        Y_copter = pu[2]
                                                        mode = "Action Mode"
```

```
                                        elif Y_copter>-60:
                                                if (DX<-30 and DY<-30) or (DX<-30 and DY>30) or (DX>30 and DY<-30) or (DX>30 and DY>30) or (DX<30 and DX>-30 and DY>30) or (DX<30 and DX>-30 and DY<-30) or (DY<30 and DY>-30 and DX>30) or (DY<30 and DY>-30 and DX<-30):
                                                        center = "Center Coordinates: ({centerx}, {centery})".format(centerx=str(X_center), centery=str(Y_center))
                                                        Deltas = "Center Deltas: ({deltaX}, {deltaY})".format(deltaX=str(DX),deltaY=str(DY))
                                                        # draw the status text on the frame
                                                        if DX>0:
                                                            if DY>0: # bottom right of the frame
                                                                flight_update = "Move Right {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                            else: # top right of the frame
                                                                flight_update = "Move Right {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else:
                                                            if DY>0: # bottom left of the frame
                                                                flight_update = "Move Left {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                            else: # top left of the frame
                                                                flight_update = "Move Left {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        # calculating optimal position of the drone
                                                        opt_position(X_copter,Y_copter,DX,DY)
                                                        X_copter = pu[1]
                                                        Y_copter = pu[2]
                                                        mode = "Waiting Mode"
                                        else:
                                                if (DX<-30 and DY<-30) or (DX<-30 and DY>30) or (DX>30 and DY<-30) or (DX>30 and DY>30) or (DX<30 and DX>-30 and DY>30) or (DX<30 and DX>-30 and DY<-30) or (DY<30 and DY>-30 and DX>30) or (DY<30 and DY>-30 and DX<-30):
                                                        center = "Center Coordinates: ({centerx}, {centery})".format(centerx=str(X_center), centery=str(Y_center))
```

```
                                                    Deltas = "Center Deltas: ({deltaX}, {deltaY})".format(deltaX=str(DX),deltaY=str(DY))
                                                    if DX>0:
                                                        if DY>0: # bottom right of the frame
                                                            flight_update = "Move Right {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else: # top right of the frame
                                                            flight_update = "Move Right {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                    else:
                                                        if DY>0: # bottom left of the frame
                                                            flight_update = "Move Left {magdelx} and Backward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                        else: # top left of the frame
                                                            flight_update = "Move Left {magdelx} and Forward {magdelY}".format(magdelx=str(abs(X_corrected)),magdelY=str(abs(Y_corrected)))
                                                    # calculating optimal position of the drone
                                                    opt_position(X_copter,Y_copter,DX,DY)
                                                    X_copter = pu[1]
                                                    Y_copter = pu[2]
                                                    mode = "Waiting Mode"



                    cv2.putText(image_array, status, (20, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)
                    cv2.putText(image_array, center, (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)
                    cv2.putText(image_array, Deltas, (20, 70), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)
                    cv2.putText(image_array, flight_update, (20, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)
                    cv2.putText(image_array, mode, (20, 110), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                (0, 0, 255), 2)
                    cv2.rectangle(image_array,(290,200),(350,260),(0,0,225),2)
```

```
                cv2.imshow("Frame", image_array)
                key = cv2.waitKey(1) & 0xFF
                rawCapture.truncate(0)
                if key == ord('q'):
                    break
        cv2.destroyAllWindows()

        return

if __name__ == "__main__":
        main()
```