CENTRO DE INVESTIGACIONES
EN ÓPTICA, A.C.

Versión Definitiva. Incluye cambios sugeridos por revisores

Stabilization and Object Tracking based on Computer Vision: Application in a Quad-rotor

Masters in Optomechatronics

Advisor : Gerardo Flores Colunga, Ph. D.

Student: Luis Francisco Arreola Corzo, B. Eng.

October 2018
León, Guanajuato, México

Agradecimientos

Agradezco a mi asesor, Dr. Gerardo Flores, por acompañarme en este camino, por su confianza y paciencia. Al Dr. David Monzón, por ser el responsable de empezar esta etapa de mi vida en el CIO, por sus lecciones de vida y amistad.

A mi familia, porque aún desde la distancia su apoyo fue incondicional.

A todos mis compañeros y amigos del laboratorio de percepción y robótica y del CIO en general, por todos los momentos que compartimos, buenos y malos. A los doctores e ingenieros de los laboratorios de Optomecatrónica del CIO, por todo el apoyo brindado.

A mis sinodales, Dr. Apolinar Muñoz y Dr. Luis Valentín, por el tiempo dedicado al revisar este trabajo y por sus valiosos consejos.

Finalmente, agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo recibido para lograr terminar la maestría, asi como a la Dirección de Formación Académica de esta institución y a FORDECYT-CONACYT bajo el acuerdo 000000000292399 por el apoyo financiero para asistir a los congresos.

Abstract

This work presents a *Research and Development* thesis of a functional Unmanned Aerial Vehicle (UAV), integrating optical flow algorithms and navigation based on object detection using Computer-Vision Features Classifiers and Robotics.

This work presents two main problems to solve. In one of them, a position estimation system for Unmanned Aerial Vehicles consisting of hardware and software is presented. It is based on low-cost devices(GPS, commercial autopilot sensors and optical flow algorithm) implemented in an on-board microcomputer. Comparative tests were conducted using the proposed approach and the conventional one, where only fusion of GPS and inertial sensors are used. Experiments were conducted using a Quad-rotor in two flying modes: hovering and trajectory tracking in outdoor environments. Results demonstrate the effectiveness of the proposed approach in comparison with the conventional approaches presented in the vast majority of commercial drones.

Furthermore, the second problem addresses object detection and tracking based on Computer Vision techniques, in essence, a training technique known as Haar Feature-based cascade classifier, image processing is made on-board with a high processor single-board computer. Based on the detected object and its position, the UAV must track it to ensure to be positioned in a center position with respect to the object and in a safe distance to it. The object in question is a human face and the experiments were conducted in a two-steps detection, searching first for the upper-body and then for the face in that specific zone to be consequently followed by the UAV.

Justification

Nowadays, unmanned aerial vehicles (UAVs aka drones) are widely required for multiple applications with the aim of finding solutions to the market problems. This is due to the fact of the high capacity of navigation and access to remote places where land vehicles or even human beings can not access. In addition, the different components that can be added to UAVs to solve different types of problems makes them increasingly essential for several problem's solutions in areas such as security, agriculture, research, industry among others. The use of sensors and cameras together with artificial intelligence nowadays is fundamental for UAV perception of environment. In order to accomplish some of the aforementioned tasks, UAVs must detect and usually track objects presented in the environment.

In this thesis it is investigated the problem of object detection and tracking with an UAV using vision systems. Particularly human beings are detected by the drone. This project is the first in the are of robotic vision developed at the Perception and Robotics LAB at CIO. Terrestrial vision is going to be implemented instead of aerial vision, mainly because the firsts tests on the object recognition and tracking will be performed, and depending of the performance, the aerial tests would be implemented.

On the other hand, the position estimation is crucial for the majority of outdoor applications for an UAV. The addition of a camera-based system endowed with optical flow algorithm is an easy-to-implement solution to improve the position estimation taking into account that the UAV already has a conventional low-cost GPS and inertial sensors, making this project remain low-cost.

Publications

Conference papers

- **L. Arreola**, A. Montes de Oca, A. Flores, J. Sanchez and G. Flores. *"Improvement in the UAV position estimation with low-cost GPS and vision-based system: Application to a quad-rotor UAV."* International Conference on Unmanned Aircraft Systems 2018 (PUBLISHED).

- A. Montes de Oca, **L. Arreola**, A. Flores, J. Sanchez and G. Flores.*"Low-cost multi-spectral imagery system for crop monitoring."* International Conference on Unmanned Aircraft Systems 2018 (PUBLISHED).

- **L. Arreola**, G. Gudiño, A. Montes de Oca, and G. Flores.*"Object recognition and tracking using Haar-like Features Cascade Classifiers: Application to a quad-rotor UAV."* Winter Conference on Applications in Computer Vision 2019 (UNDER REVIEW).

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

One of the problems to solve in this work is the intention to improve the position estimation of an UAV using just low-cost devices and a standard GPS, this will be accomplished by adding an optical flow sensor to fusion with the INS sensors and GPS. The first section of this chapter focus on the background information about this topic.

Moreover, a Haar-like features classifier is implemented to detect and subsequently track an object with the UAV. The second section of this chapter summarizes the background and importance of the classifiers and other techniques based on vision-based systems to object detection.

In essence, this chapter focus on giving the background information about these two main topics. This information relies in the importance of these topics and the existing methods, in order to approach the objectives of this work of a proper way.

1.1 Background

At the Center for Research in Optics (CIO) is developed basic and applied research in the optics and photonics fields. Computer vision is one of the most important research topics investigated at CIO. Current research areas of vision systems, such as digital image processing, optical metrology, 3D recondtruction, among others are actually investigated at CIO. On the other hand, the field of robotics has had an increasing demand in research and industry last years. For that, in the last two years CIo's research on that topic is growing do to the great demand.

This project is based on these two fields: vision and robotics. It consists of the development of a functional prototype of an UAV with a vision system, capable of detecting an object of interest, to subsequently generate navigation commands, in such a way that the UAV is able to follow that object.

1.1.1 State of the art at CIO

Researchers from CIO have investigated several subjects related with the present project. Next I mention a few of them.

In 2007, H. Sossa-Azuela, F. Cuevas, et al. designed an application for classifying objects using morphological applications using 3-D Cartesian Geometric Moment Computation. Three-dimensional Cartesian moments are considered as an important feature in applications for recognition and shape description of objects. The authors present a novel proposal where, instead of using triple summations, given that the object is taken into account as a voxelized image, they first decompose the region into a set of cubes, and then add the moments of each cube of the partition. Once the partition has been obtained, the calculation of the moments is faster than that provided by previous methods [1].

In 2009, J. Apolinar Muñoz presented an automatic technique for the 3-D vision of the foot sole. The foot sole is scanned by a laser trough a glass window. The behaviour of the laser line gives the information about the contouring of the foot. The 3-D modelling is reached by performing an approximation network. The line shift is generated by the surface variation and the camera position, defining the structure of the network. The approach of this vision system is to fit the shoe sold mold to the foot sole via contour curves. The results are finally evaluated by a root mean square of error by the use of references from a contact method [2].

In 2013, Shuting Zou, G. Flores, et al. proposed a vision-based estimation for micro aerial vehicle (MAV) in unknown urban environments. The MAV used required to navigate from an initial outdoor position to a final position inside the building. The authors proposed two vision-based approaches using the information provided by the on-board vision system. An stereo camera system is implemented to identify in a effective way the target and estimate the distance to it. Besides, a RGB-D camera is used to capture point cloud [3]. The same group of authors present a continuation of this project in 2015. To identify the window, a feature base cascade classifier is implemented, due that it provides a fast and robust method to identify the window. The information provided by the GPS and the MAV's inertial sensors is used to develop a safe path-planning. The stereo-vision system estimates the relative position with respect to the MAV [4].

In May 2016, J. Martinez-Carranza, L. Valentin, et al. presented an obstacle detection during autonomous flight of an UAV using monocular SLAM. This monocular SLAM system is implemented to estimate the position of the UAV and to build a 3D map that represents the environment observed trough the camera on-board. Non other depth sensor or inertial sensor was used in this work. One of the main advantages of this proposal with respect to others, is that the localization and mapping does not depend of the GPS signal or of an external localization system, depending just of the image processing from the camera [5].

In September 2016, Israel Becerra, L. Valentin, et al. investigated the problem of confirming the identity of a candidate object with a mobile robot that was equipped with a vision system. This is a novel method into mix localization of the robot relative to the candidate object and confirm that is the desired object, drastically reducing false positives [6].

1.2 Stabilization improvement

In many outdoor Unmanned Aerial Vehicles (UAV aka drones) applications such as: inspection & monitoring, mapping, precision agriculture and civil engineering, just to mention a few, the precision in the drone position estimation is crucial. Hence commercial drones use to use high-precision Global Positioning System (GPS) or even Real-Time Kinematic (RTK) devices [7], [8]. The first half of this work presents an easy-to-implement application proposed as a low-cost option to the commercial technology available in the market. Such implementation is based on the combination of two popular technologies: conventional low-cost GPS and a vision-based system endowed with optical flow algorithm. On one hand, GPS provides to the drone the capability to follow an specific trajectory in global coordinates, however low-cost GPS have intrinsic sources that induce errors in the UAV position, this results in an insufficient accuracy for several of the aforementioned applications. On the other hand, optical flow devices provide information about UAV relative position w.r.t. a fixed reference frame. However, optical flow devices by itself are unable to provide global coordinates, which are necessary in the vast majority of outdoor applications, for instance in 3D reconstruction, mapping and object tracking.

The vast majority of the works available in the literature shows recent developments in the position estimation in denied GPS environments [9], [10], [11], [12]. A comparative study of optical flow and a combination of measurements from GPS-INS sensors through UAV flight tests is presented in [13], where inaccuracies in the optical flow are evident compared with the position provided by the GPS. The group of the authors of the last paper continues their research in [14], where they present an Unscented Information Filter for estimating the vehicle speed of GPS-denied navigation. In that work the authors use GPS measurements as reference values for the fusion algorithm using INS and optical flow information. In [15] a nonlinear observer for UAV velocity based on INS and optical flow data were implemented. The results do not show visible improvements in the accuracy of UAV position estimation w.r.t classical filter approaches. One of the most recent works presented at [16], proposes a fusion of the PX4Flow sensor [17] and INS system; the authors show some results with considerable error in position estimation (around 5 meters) from the desired trajectory, which is insufficient for various applications. Nowadays the use of RTK or PPK (Post-Processing Kinematic) systems for global navigation satellite systems (GNSS) are the most accurate for outdoor applications, this is due to the accuracy they own, which ranges between 20mm and 50mm [18], [19]. Although these technologies are one of the best options in the market for UAV position estimation in outdoor environments, they can be very expensive due to the amount of equipment needed compared to low-cost commercial drones. In the first half of this work, several outdoor experiments were conducted in the task of trajectory tracking and hovering at a given outdoor environment. Such experiments are compared with the classical approach where only GPS and inertial navigation sensors (INS) information are used to estimate the current UAV position. Such experiments show that the accuracy of the proposed system improves considerably UAV position estimation, and hence the accuracy in tracking a given trajectory. This works continues a research presented in [20], where a GPS/INS/Optical flow data fusion for position and velocity estimation is presented. In such a work the algorithm presents a considerable error in the estimation, also it has not been compared with any well-known approach.

The contribution of this work w.r.t to the existing literature is: a) the implementation of our approach in a Quad-rotor UAV; b) improvement in the optical flow algorithm; and c) enhancement of the data fusion algorithm. The presented results are compared with the data obtained by the classical Kalman Filter, which uses only GPS and INS information. This is provided by the PX4 firmware for Quad-rotors, a firmware very popular nowadays. Also, experiments were conducted in hovering and in trajectory tracking, in both cases the results are highly satisfactory in comparison with classical approaches presented in the literature. The obtained results demonstrate a considerable efficiency w.r.t. standard GPS and INS systems; such approaches usually have position errors around 2.5m [21], which results in an insufficient accuracy for certain applications. Therefore this approach results in a viable alternative for limited budget.

1.3 Object tracking

In the recent demands of the UAVs market, the military and industry stand out from the rest, among the most requested applications are the aerial photography, environment monitoring, farming, structure inspection and so forth. [22], [23], [24]. A vision-based system is one of the favorites solutions to these demands, due to its passive and low-cost characteristics. The second half of this work presents an easy-to-implement application to this topic, implementing an on-board computer and a monocular camera. The entire system consists in detecting an object by capture a frame from the camera and then the on-board computer processes the image trying to detect the object using Haar-like feature-based classifier. Due to the fact that the Haar classifiers are considered as *weak* classifiers [25], a cascade training is implemented to perform a more robust detection. Once the object is detected, the on-board computer determines the position of the object w.r.t the UAV, so it sends the corresponding information to the flight controller to ensure the correct tracking. For this problem, tracking refers to the capability that the systems has to detect a desired object and follow it with the information given by the frame, and refresh this information when a new frame is being analyzed. It is assumed that the object may be present different characteristics but, in general, all the possible objects has the same features, this is way a simple image processing algorithm is not useful for this work, therefore a Features Classifier is implemented.

There are several works that have explored object detection and tracking using UAVs. In [26], the object detection and tracking by the UAV is performed with a video previously recorded. This was one of the first stages of object tracking related to a UAV. Since then, researchers all around the world have explored this problem, this time during the navigation of the UAV [27], [28]. Multiple navigation strategies have been developed based on vision, for instance road following, power lines inspection, and navigation in orchards [29], [30], [31]. For this work the general strategy consists in tracking the object with the information of the continuous frames that are acquired by the camera. Each and every one of the frames is compared with the cascade training of the features classifiers to detect the desired object. The track consists of navigating the UAV in such a way that it is centered with respect to the object, and at a safe distance from it. This principle of functioning is the same that, for example, those who vision-based systems use to try to accomplish an autonomous landing based in a pre-established landmark that indicates the position where the UAV must land [32], [33].

The experiments that were conducted in this section of the work consists in the detection and tracking of a human face. At the beginning, slow velocities for the UAV movement were implemented to observe the time of reaction of the UAV once the object was detected and the on-board computer sent the navigation commands. The results from these experiments are highly satisfactory. The UAV had a reaction time of approximately 0.2 seconds after the centroid of the object and the subsequent commands of navigation were sent. The detection process consist in searching an upper body(from the shoulders to the head), and inside the sub-image of a ”detected upper body” the human face is now searched. This seemed to be a right decision mainly because no matter how well the training process is performed, there is always the possibility of detecting false objects, therefore searching an object inside of another makes the system more robust.

1.4 Outline of dissertation

Two main problems are addressed in this work. First, this work presents the behavior of an UAV Quad-rotor when an optical flow sensor is implemented, measuring and comparing its performance with the behavior of the UAV when only GPS + INS sensors are used. And second, object tracking based on computer vision is presented. In the following chapters, the work will be explained in detail.

The second chapter describes the theory fundamentals settled for the mathematical model of the Quad-rotor, the Extended Kalman filter algorithm, the Optical flow algorithms, and the

Haar Feature-based Cascade Classifier.

The methods used for this work are explained in Chapter 3. This chapter explains in detail both the hardware and software description.

In Chapter 4, the results obtained given the setup explained in Chapter 3 are shown. The chapter is mainly divided in the results of the optical flow implementation and the object detection and tracking performance.

Finally, Chapter 5 discusses the Conclusions and the Future Work related to this work.

CHAPTER 2

Theory Fundamentals

The Unmanned Aerial Vehicles (UAV), specifically Quad-rotors, are controlled using electronics control systems and sensors. They consist in several components. A frame, Electronic Speed Controllers (ESCs), motors, propellers, battery, control board and Inertial Measurement Unit (IMU) are considered essentials. IMU is used to get the actual information of attitude of the Quad-rotor consisting of a 3-axis accelerometer and gyroscope. The combination of IMU, a magnetometer and an on-board processing system is known as Attitude Heading Reference System (AHRS) [34]. The GPS module get the absolute position of the Quad-rotor. Other components as pressure, ultrasonic and infrared sensors are used to determine the absolute altitude. The information of all of these sensors are normally combined in a Kalman Filter.

2.1 Quad-rotor Model

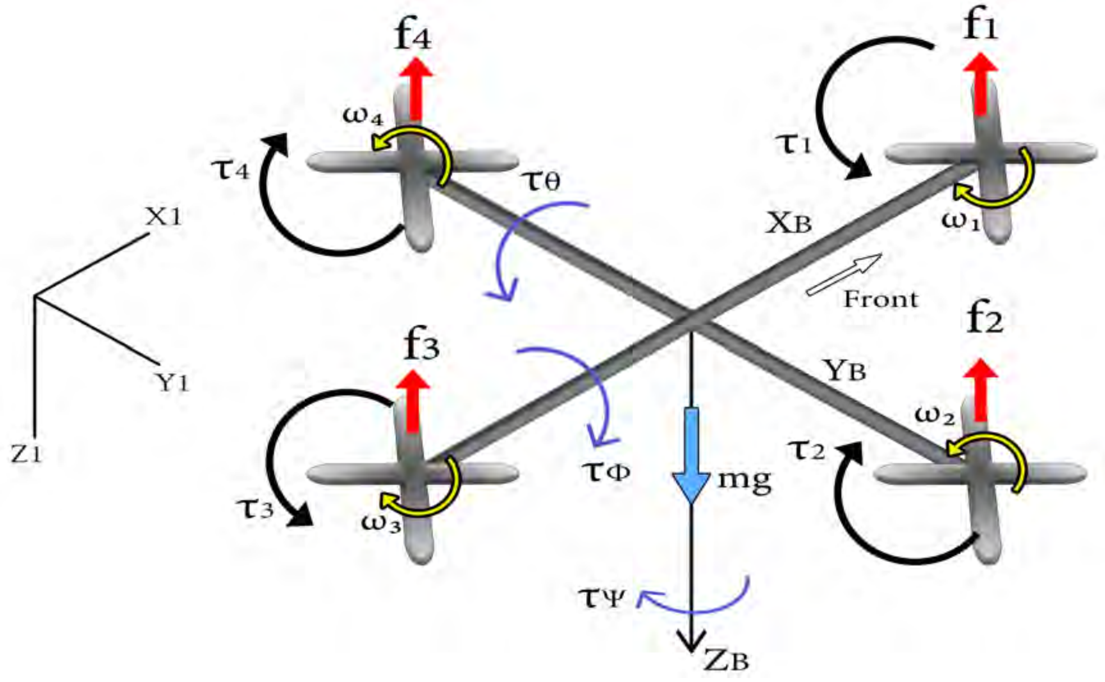


Figure 2.1: Quad-rotor Inertial and Body frame.

Taking in consideration that the Quad-rotor is a rigid body, their dynamics can be explained using Newton-Euler equations. Each propeller rotates at an angular velocity ω_i producing a corresponding force f_i directed upwards and a counteracting torque τ_i directing at the opposite direction to the direction of rotation. The propellers with the angular speed ω_2 and ω_4 spin clockwise and the other two propellers spin counter-clockwise. This can be appreciated in Fig. 2.1, [35]. The four rotors generate their corresponding angular velocity, torque and force. The position and orientation is reached by variating the thrust of an specific rotor. For example, decreasing the thrust of the propellers 3 and 4 with respect to the propellers 1 and 2, will result in a roll to the left movement, and decreasing the thrust of the propellers 1 and four with respect to the propellers 2 and 3, will result in a pitch-forward movement. For a clockwise yaw movement, the propellers 1 and 3 must reduce its thrust with respect to the propellers 2 and 4.

2.1.1 Euler equations

Euler angles represent a sequence of three elemental rotations as shown in Fig. 2.2. Since any orientation can be achieved by composing three elemental rotations. These rotations start

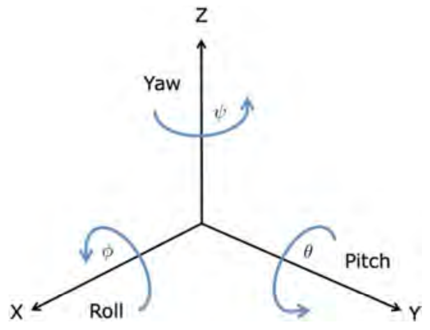


Figure 2.2: Euler angles for angular position in the inertial frame.

from a known standard orientation. This configuration is described by the following rotation matrices as in [36]. Therefore, the inertial position coordinates and the body reference coordinates are related by the rotation matrix obtained from computing

$$R(\phi,\theta,\psi) = R(\phi)R(\theta)R(\psi) \tag{2.1}$$

where R_ψ, R_θ and R_ϕ are defined as the three rotational matrices of each fixed axis

$$R_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi & C_\phi \end{bmatrix} R_{y,\theta} = \begin{bmatrix} C_\theta & 0 & S_\theta \\ 0 & 1 & 0 \\ -S_\theta & 0 & C_\theta \end{bmatrix} R_{z,\psi} = \begin{bmatrix} C_\psi & -S_\psi & 0 \\ S_\psi & C_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

The rotation matrix from the body frame E_B to the fixed frame E_I is an orthogonal 3x3 matrix given by Eq. (2.3), where C_{angle} and S_{angle} corresponds to $\cos(angle)$ and $\sin(angle)$ respectively [35], [37], [38]. The rotation matrix is now defined as

$$R = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\phi S_\theta C_\psi - C_\phi S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi S_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi & C_\phi C_\theta \end{bmatrix}. \tag{2.3}$$

The rotation matrix **R** is an orthogonal matrix, therefore R^{-1}=R^T, which is the rotational matrix that represents the inertial frame to the body frame. The first group of kinematic equations need to express translational velocity in terms of velocity components in the body frame. The UAV position vector **p** has its components p_x, p_y, and p_z located in the frame E_I. The vector η is defined as

$$\eta = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

and describes the angular velocities of the Quad-rotor in the body frame. To relate these two groups of variables, a time differentiation and a rotational transformation need to be applied as shown below

$$\frac{d}{dt}\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = R(\phi,\theta,\psi)\begin{bmatrix} u \\ v \\ w \end{bmatrix} = R(\phi,\theta,\psi)^{-1}\begin{bmatrix} u \\ v \\ w \end{bmatrix}. \tag{2.4}$$

Due to the time derivative of position in the inertial frame E_I, a rotation R needs to be used in Eq. (2.4) in order to obtain velocity components with respect to the inertial frame E_I, therefore the inverse of the Eq. (2.3) must be replaced. Substituting these matrix, the time derivate of the position vector is expressed as in [39] as

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \end{bmatrix} = \begin{bmatrix} C\theta C\psi & S\phi S\theta C\psi - C\phi S\psi & C\phi S\theta C\psi + S\phi S\psi \\ C\theta S\psi & S\phi S\theta S\psi + C\phi C\psi & C\phi S\theta S\psi - S\phi C\psi \\ -S\theta & S\phi C\theta & C\phi C\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \tag{2.5}$$

Another relationship that need to be defined is the one between the angular positions: ψ, θ, and ϕ; and the angular rates: p, q, and r. The time derivatives of angular positions are not equal to the angular rates because these two groups of variables are expressed in different coordinate frames. The Euler angles are defined subsequently along three different coordinate frames while the angular rates are defined in the body frame [39]. To obtain the equation that relates Euler angles and angular rates, angular rates need to be expressed as

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R(\phi)R(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}. \tag{2.6}$$

Roll angle is already defined in body frame so it does not need a rotation. Pitch and Yaw angles need an appropriate rotation so they can be defined in the body frame, therefore $R(\phi)$, and $R(\theta)$ need to be substituted to obtain a simplified expression as follows

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{2.7}$$

Multiplying both sides in Eq. (2.7) by the inverse matrix and rearranging the equation, expressions for the derivatives of the angular positions can be found in terms of angular positions and angular rates in the next equation

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.8}$$

From Eqs. (2.8) and (2.5), the kinematic equations of the Quadrotor are [36].

$$\begin{aligned}
\dot{\phi} &= p + q(S_\phi T_\theta) + r(C_\phi T_\theta) \\
\dot{\theta} &= q(C_\phi) - r(S_\phi) \\
\dot{\psi} &= q(S_\phi / C_\theta) + r(C_\phi / C_\theta) \\
\dot{p}_x &= u(C_\psi C_\theta) + v(C_\psi S_\theta S_\phi - C_\phi S_\psi) + w(C_\psi C_\phi S_\theta + S_\psi S_\phi) \\
\dot{p}_y &= u(C_\theta S_\psi) + v(C_\psi C_\phi + S_\psi S_\theta S_\phi) + w(C_\phi S_\psi S_\theta - C_\psi S_\phi) \\
\dot{p}_z &= -u(S_\theta) + v(C_\theta S_\phi) + w(C_\theta C_\phi)
\end{aligned} \tag{2.9}$$

The kinematic part of the modelling describes the motion of the Quad-rotor without considering the forces affecting it. In the next subsection, the dynamics equations are presented.

2.1.2 Newton equations

The rotational motion equation of the UAV can be obtained using Newton's second law. It begins by obtaining the time derivative of angular momentum as expressed next

$$\dot{\vec{H}} = \vec{M} \tag{2.10}$$

where $\vec{M}$ is the total external moment acting on the center of mass and $\vec{H}$ is the angular momentum. Typically, the angular momentum vector $\vec{H}$ is defined as the cross-product of the position vector $\vec{r}$ and the momentum vector $\vec{p}$ as

$$\vec{H} = \vec{r} \times \vec{p}. \tag{2.11}$$

Moment $\vec{H}$ can be expressed as its vector form as

$$\vec{H} = \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = \begin{bmatrix} yp_z - zp_y \\ yp_x - zp_z \\ yp_y - zp_x \end{bmatrix}. \tag{2.12}$$

The expression for angular momentum that describes the rate of change with respect of time of a rigid body about its center of mass is given by

$$\dot{\vec{H}} = \sum_i \vec{r}_i \times m_i \dot{\vec{r}}_i \tag{2.13}$$

where r_i is the position vector of a particle m_i. In this case, the position of all the particles are fixed in the body, therefore the velocity of each particle is defined by

$$\dot{\vec{r}}_i = \vec{\omega} \times \vec{r}_i. \tag{2.14}$$

As it is already known, $\vec{\omega}$ is the body angular velocity. In this case, the body is the quadrotor. Hence it is assumed that it has a continuous mass distribution, the particle mass can be represented by its density times an elemental volume ρdV. Now, the expressions of velocity and elemental volumes can be substituted, thus the summation expression in (2.13) can be replaced by an integral as follows

$$\vec{H} = \int_v \rho \vec{r} \times (\vec{\omega} \times \vec{r}) dV \tag{2.15}$$

defining $\vec{r}$ as its vector form as

$$\vec{r} = x\hat{i} + y\hat{j} + z\hat{k}. \tag{2.16}$$

On the other hand, the vector cross product can be defined as

$$\vec{r} \times (\vec{\omega} \times \vec{r}) = [(y^2+z^2)p - xyq - xzr]\hat{i} + [(x^2+z^2)q - yxp - yzr]\hat{j} + [(x^2+y^2)r - zxp - zyq]\hat{k}. \tag{2.17}$$

Using the previous equations, we can define the moments of inertia as

$$\begin{aligned}
I_{xx} &= \int_v \rho\,(y^2+z^2)\,dv \\
I_{yy} &= \int_v \rho\,(x^2+z^2)\,dv \\
I_{zz} &= \int_v \rho\,(x^2+y^2)\,dv \\
I_{xy} &= I_{yx} = \int_v \rho\,(xy)\,dv \\
I_{xz} &= I_{zx} = \int_v \rho\,(xz)\,dv \\
I_{yz} &= I_{zy} = \int_v \rho\,(yz)\,dv
\end{aligned}\;. \tag{2.18}$$

It is assumed that the four arms of the quad-rotor are symmetric and they are aligned with the x and y axes. This results in a diagonal inertia matrix, which is a diagonal matrix I where I_{xx} and I_{yy} are equal.

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{2.19}$$

This leads to determine the equations of the particles m_i of the system, which are defined by [36] as follows

$$\begin{aligned} m_x &= \dot{p}I_x - qrI_y + qrI_z \\ m_y &= \dot{q}I_y + prI_x - prI_z \\ m_z &= \dot{r}I_z - pqI_x + pqI_y \end{aligned} \quad . \tag{2.20}$$

From Newton's second law, we have

$$\vec{F} = m\dot{\vec{v}} \tag{2.21}$$

where $\vec{F}$ is defined as the total external force acting on the quad-rotor, $\vec{v}$ is the absolute velocity of the center of mass and m is the mass of the quad-rotor [40]. The vector $\vec{F}$can be expressed in terms of the x, y and z components as

$$\vec{F} = F_x\hat{\imath} + F_y\hat{\jmath} + F_z\hat{z}. \tag{2.22a}$$

Velocity vector is defined as

$$\vec{v} = u\hat{\imath} + v\hat{\jmath} + w\hat{k}. \tag{2.22b}$$

The quad-rotor is rotating with angular velocity, therefore the absolute acceleration can be expressed as

$$\dot{\vec{v}} = \dot{\vec{v}}_b + \vec{\omega} \times \vec{v} \tag{2.23}$$

where $\dot{\vec{v}}_r$ is the acceleration seen from the body axis. This term and the angular velocity can also be described in terms of a vector form as

$$\begin{aligned} \dot{\vec{v}}_b &= \dot{u}\hat{\imath} + \dot{v}\hat{\jmath} + \dot{w}\hat{k} \\ \vec{\omega} &= p\hat{\imath} + q\hat{\jmath} + r\hat{k} \end{aligned} \quad . \tag{2.24}$$

Expanding the cross product in Eq. (2.23), the equation results in

$$\vec{\omega} \times \vec{v} = \hat{i}(wq - vr) - \hat{j}(wp - ur) + k(vp - uq). \tag{2.25}$$

Considering force components in Eq. (2.22a) with gravitational force terms, the three force components can be written as follows

$$F = \begin{bmatrix} F_{x-applied} - mgsin(\theta) \\ F_{y-applied} + mgcos(\phi) \\ F_{z-applied} + mgcos(\theta)cos(\phi) \end{bmatrix} . \tag{2.26}$$

Also we can express each term of the Eq. (2.22a) using the Eq. (2.25) in the Eq. (2.21) as follows

$$\begin{aligned} F_x &= m(\dot{u} + wq - vr) \\ F_y &= m(\dot{v} + wp - ur) \\ F_z &= m(\dot{w} + vp - uq) \end{aligned} . \tag{2.27}$$

Eqs. (2.27) and (2.9) are the equations that define the mathematical model of the Quad-rotor in the body frame, these equations are put together next:

$$\begin{aligned}
\dot{\phi} &= p + q(S_\phi T_\theta) + r(C_\phi T_\theta) \\
\dot{\theta} &= q(C_\phi) - r(S_\phi) \\
\dot{\psi} &= q(S_\phi / C_\theta) + r(C_\phi / C_\theta) \\
\dot{P}_x &= u(C_\psi C_\theta) + v(C_\psi S_\theta S_\phi - C_\phi S_\psi) + w(C_\psi C_\phi S_\theta + S_\psi S_\phi) \\
\dot{P}_y &= u(C_\theta S_\psi) + v(C_\psi C_\phi + S_\psi S_\theta S_\phi) + w(C_\phi S_\psi S_\theta - C_\psi S_\phi) \\
\dot{P}_z &= -u(S_\theta) + v(C_\theta S_\phi) + w(C_\theta C_\phi)
\end{aligned} \tag{2.28a}$$

$$\begin{aligned}
F_x &= m(\dot{u} + wq - vr) \\
F_y &= m(\dot{v} + wp - ur) \\
F_z &= m(\dot{w} + vp - uq)
\end{aligned} \tag{2.28b}$$

where 2.28a represents the kinematic equations and 2.28b represent the dynamic equations of the Quad-rotor.

2.2 Kalman Filter

The Kalman Filter is the most famous method for prediction in systems with random noise. Is an algorithm that uses a series of measurements observed over time, containing noise and other inaccuracies. The resulting of the Kalman Filter produces more accurate estimations than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each time-step. The key idea behind Kalman Filter is presented in two steps: *Prediction* and *Update*. The first step consists in the prediction of the state variables of the system based on the inputs and previous values of system states. In the second step, the prediction is improved using the data of external measurements [41]. Because it is not always possible to measure each control variable, the Kalman Filter provides an estimate of the lost information of measurements and indirect noises. That leads to the most common applications of the Kalman Filter, such as the control of complex dynamic systems, like spacecrafts, manufacturing processes, ships and aircrafts. The Kalman Filter can be applied as a tool for almost exclusively for two purposes: *Estimation* and *performance analysis* of estimators.

1. **Estimation:** The Kalman Filter allow us to estimate the state of the dynamic systems that tends to have sort of random behaviors, using statistical information.

2. **Performance analysis:** The objective of the performance analysis of the estimation systems is to know how to determine the best use of the sensors types given as a set of the design criteria. These criteria are based on estimation accuracy and the cost of the system.

The Kalman Filter is based on the probability distribution of its estimation errors. It uses a parametric characterization in determining the most optimal filtering gains. This probability distribution is used in seeing the performance evaluation as a function of the *design parameters* of a system, these parameters could be:

- The type of sensors that will be used.
- All the locations and orientations of the sensors with respect to the estimated system.
- The known noise of the sensors.
- The methods used to pre-filtering the sensor's noise by smoothing.
- The data sampling rates for all the sensors that will be used.
- How improved is the model simplification to reduce the implementation requirements.

The Kalman filter is basically a set of mathematical equations with the purpose of getting an optimal estimator of predictor-corrector that minimizes the estimated errors covariance when the conditions are made. The standard Kalman filter process start by fitting the UAV dynamics on the Kalman Filter equations

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_k + \omega_k \\ y_k &= Cx_k + \upsilon_k \end{aligned} \tag{2.29}$$

where x_k is the state vector containing the terms of interest for the system at time k. u_k is the vector containing any control inputs. **A** is the state transition matrix which applies the effect of each system state parameter at time $k-1$ on the system state at time k. **B** is the control input matrix which applies the effect of each control input parameter in the vector u_k on the state vector. y_k is the vector of measurements and C_k is the transformation matrix that maps the state vector parameters into the measurement domain. ω_k and υ_k are process and measurement noises respectively. Due that the measurement is noisy, and therefore it does not reflect the true output, the input can be run through the knowing UAV model to make another estimation [42].

$$\begin{aligned} \hat{x}_k &= A\hat{x}_{k-1} + Bu_k \\ \hat{y}_k &= C\hat{x}_k \end{aligned} . \tag{2.30}$$

This estimation will also not be perfect, because a mathematical model is just an approximation of the real system. This is where the Kalman Filter acts, combining the two pieces of information (the measurement and the prediction) to find the optimal estimate output in the presence of process and measurement noises [43]. Computationally, this combination relates to the next equation

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k + K_k(y_k - C(A\hat{x}_{k-1} + Bu_k)). \tag{2.31}$$

The first part of Eq. (2.31) predicts the current state by using the previous time-step state estimation and the current input. This is called the *predicted state estimate* or *a priori estimate*, since it is calculated before the current measurement is taken, and can be called as $\hat{x}_k^-$. Rewriting Eq. (2.31), the equations results like

$$\hat{x} = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-). \tag{2.32}$$

The second part of Eq. (2.32) uses the measurement incorporated into the prediction to update the *a priori estimate*. This result is called *A posteriori estimate*. Therefore, the equations needed for the Kalman Filter are divided in two, the prediction equations:

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_k \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned} \tag{2.33}$$

and the update equations:

$$\begin{aligned} K_k^- &= \frac{P_k^- C^T}{CP_k^- C^T + R} \\ \hat{x}_k^- &= \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-) \\ P_k &= (I - K_k C)P_k^- \end{aligned} \quad . \tag{2.34}$$

These two group of equations define the two-steps process of the Kalman Filter. In Eq. (2.33), the system model is required to calculate the *a priori state estimate* and the error covariance P(the variance of the *a priori state estimate*). The second step of the algorithm, shown in Eq. (2.34), uses the *a priori state estimates* calculated in Eq. (2.33) and updates them to find the *a posteriori estimates* and the error covariance. The Kalman gain K_k is calculated such that minimizes the *a posteriori* error covariance P_k. In other words, the Kalman gain determines how important are the measurement ant the *a priori estimate* contributes to the calculation of

$\hat{x}_k$. If the measurement noise is small, then this is trusted more than the *a priori estimations* to the calculation and vice versa. Once the update equations are calculated, the *a posteriori estimates* are used to predict the new *a priori estimates* in the next time-step, repeating itself as a loop. The algorithm does not need all the past information, just the information of the estimated state and error covariance of the previous time-step and the measurement of the current time-step [43], [42], [44].

2.2.1 Extended Kalman Filter

The Extended Kalman Filter (EKF) is the nonlinear version of the Kalman Filter, since the most realistic problems in robotics involves nonlinear functions. This work has implemented EKF for position estimation; taking into account the data obtained from an optical flow algorithm to improve UAV position accuracy. In order to implement the EKF, a mathematical model that represents the quadrotor nonlinear dynamics is needed [45]. Remembering the classic Kalman Filter linear model presented in Eq. (2.29), the nonlinear representation of the Kalman Filter aka EKF is

$$\begin{aligned} x_k &= f(x_{k-1}, u_k) + w_k \\ y_k &= g(x_k) + v_k \end{aligned} \tag{2.35}$$

where f and g are nonlinear states and output functions respectively. EKF is conformed by subsequent linearizion of model (2.35) around the equilibrium point of the system, resulting in a sequence of models similar to (2.29). EKF linearizes the non-linear function around the mean of the current state estimate [46]. At each time step, the linearizion is performed locally and the resulting Jacobian matrices are then used in the predictions and updates states of the EKF algorithm. Therefore, given the system presented in Eq. (2.35), the Jacobian are defined as follows:

$$\begin{array}{l} F = \dfrac{\partial f}{\partial x} \quad |\hat{x}_{k-1}, u_k \\ G = \dfrac{\partial g}{\partial x} \quad |\hat{x}_k \end{array} . \tag{2.36}$$

Finally, the linearized system resulting from this process becomes [47]:

$$\begin{array}{lcl} \Delta x_k & \approx & F\Delta x_{k-1} + \omega_k \\ \Delta y_k & \approx & G\Delta x_k + \upsilon_k \end{array} . \tag{2.37}$$

2.3 Optical Flow Algorithm

The Optical Flow (OF) is the pattern of apparent motion of an object between two consecutive frames. This motion can be caused by the movement of the object or the camera trough time. Mathematically, the optical flow consists in a 2D vector field. This 2D motion results from the projection of moving 3D objects in the image plane. Each vector contains the data that shows the movement of the detected objects from one frame to the next one, as shown in Fig. 3.3. The Optical flow algorithm is based on two main assumptions given next:

Assumption 1. *The pixel intensity of a detected object does not present significative changes between consecutive frames.*

Assumption 2. *Pixels near to the detected object have similar motions.*

The 2D motion equations establishes that $I(x,y,t)$ is the center voxel (a pixel in three dimension) on an $m \times n$ neighborhood, then it moves by Δx and Δy in a determined time Δt

to $(x+\Delta x, y+\Delta y, t+\Delta t)$, therefore the next equation is assumed [48].

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t). \tag{2.38}$$

Since the displacements are differential, and therefore they are very small, the Taylor approach is used around $I(x,y,t)$ to simplify the Eq. (2.38) as follows

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

where $H.O.T.$ means higher-order terms, thus they can be despised. Considering Assumptions 1 and 2, from Eq. (2.39) is obtained

$$\frac{\Delta I}{\Delta x}v_x + \frac{\Delta I}{\Delta y}v_y + \frac{\Delta I}{\Delta t} = 0 \tag{2.39}$$

where $v_x = \frac{\Delta x}{\Delta t}$ and $v_y = \frac{\Delta y}{\Delta t}$ are the optical flow, or in other words, the (v_x, v_y) are the components of the image velocity, and the $\frac{\Delta I}{\Delta x}, \frac{\Delta I}{\Delta y}, \frac{\Delta I}{\Delta t}$ are the image intensity derivatives of (x,y,t) aka I_x, I_y and I_z [49]. Clearly (v_x, v_y) are unknown and the Eq. (2.39) cannot be solved with two unknown variables, for that it is used the so-called *Gunnar-Farneback* algorithm.

2.3.1 Gunnar-Farneback

The Gunnar-Farneback algorithm [50] is designed to produce optical flow results. It consists in two-frame motion estimation algorithm. This method approximate each neighborhood of both frames by quadratic polynomials. From observing how an exact polynomial transforms under translation, a method to estimate displacement fields from the polynomial expansion coefficients is derived to lead an algorithm. The polynomial expansion idea is to approximate the neighborhood of some pixel with a polynomial. The method works only with quadratic polynomials, giving the local signal model, expressed in a local coordinate system

$$f_1(x) \approx x^T A_1 x + b_1^T x + c_1 \tag{2.40}$$

where A is a symmetric matrix, b a vector and c a scalar. The coefficients are estimated from a weighted least squares fit to the signal values in the neighborhood. This can be implemented of a more efficient way than common computational process by a hierarchical scheme of separable convolutions [51].

From Eq. (2.40) can be constructed a new signal f_2 by a global displacement d,

$$\begin{aligned} f_2(x) &= f_1(x-d) = (x-d)^T A_1 (x-d) + b_1^T (x-d) + c_1 \\ &= x^T A_1 x + (b_1 - 2A_1 d)^T x + d^T A_1 d - b_1^T d + c_1 \\ &= x^T A_2 x + b_2^T x + c_2. \end{aligned} \tag{2.41}$$

Equating the coefficients in the quadratic polynomial yields

$$A_2 = A_1, \tag{2.42a}$$

$$b_2 = b_1 - 2A_1 d, \tag{2.42b}$$

$$c_2 = d^T A_1 d - b_1^T d + c_1, \tag{2.42c}$$

the translation d can be solved from Eq. (2.42b)

$$d = -\frac{1}{2} A_1^{-1} (b2 - b1). \tag{2.43}$$

The assumptions that an entire signal being a simple polynomial and a global translation that relates two signals is quite unrealistic. To solve this, Eq. (2.40) needs to be replaced with local polynomial approximations. The first step is doing a polynomial expansion of both images, therefore the expansion coefficients are $A_1(x)$, $b_1(x)$ and $c_1(x)$ for the first image and $A_2(x)$, $b_2(x)$ and $c_2(x)$ for the second image. According to Eq. (2.42a), $A_1 = A_2$ should be true ideally, but in practice is neccesary do the approximation

$$A(x) = \frac{A_1(x) + A_2(x)}{2} \tag{2.44}$$

and

$$\Delta b(x) = -\frac{1}{2}(b_2(x) - b_1(x)) \tag{2.45}$$

to obtain the primary constraint

$$A(x)d(x) = \Delta b(x) \tag{2.46}$$

where d(x) indicates that the global displacement in Eq. (2.41) is also replaced with a spatially varying displacement field [50].

2.4 Haar Feature-based Cascade Classifiers

Haar feature-based cascade classifiers is an effective object detection method. It is a machine learning-like based approach where a cascade function is trained from positive and negative images. Then, it is used to detect objects in other images. Initially, the algorithm needs a big number of ”positive” images (images with the object to detect) and ”negative” images (images without the object to detect). Due that in recent years face detection has been widely studied for applications like face recognition or human-computer interaction, it can be found several techniques especially made for this task, the decision in this work to detect faces is because of this [52], [53]. Then, the algorithm extracts features from the images to use it later to detect the object in other images. For this, Haar features is implemented, these features is shown in Fig. 2.3 [54]. Each Haar feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle. The Haar features are computed considering the region that represents the pattern, and computes the difference between the white pixels and the black pixels in that region. As shown in Fig. 2.3, there are different patterns to build Haar features, although only a few examples of all the patterns are shown. The process is made at multiple scales to make the system scale invariant. The *multiple scales* term refers to just scale the image down to compute the same features again, and in this way make the features robust against size variations of a given object [55], [56].

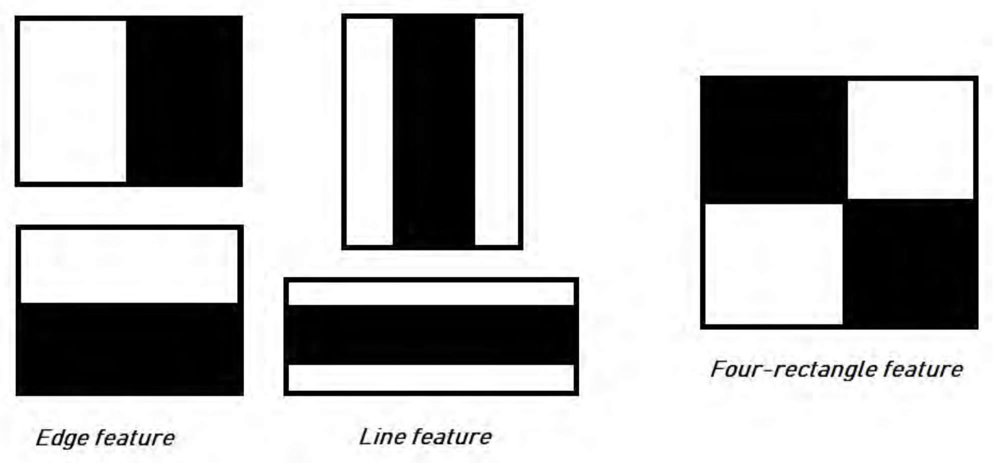


Figure 2.3: Different patterns considered as Haar features.

2.4.1 Integral image

The object detection procedure classifies images based on the value of features instead of working directly with pixels. This is mainly because a feature-based system operates faster than a pixel-based system. The system uses three kind of features shown in Fig. 2.3. The value of a *two-rectangle feature* is the difference between the sum of the pixels within two regular regions (Edge features). The regions must have the same size and shape and be adjacent whether horizontally or vertically. A *three-rectangle feature* calculates the sum within two rectangles subtracted from the sum in a rectangle in the center (Line feature). And finally a *Four-rectangle feature* computes the difference between two pairs of rectangles diagonally positioned. Given a base resolution of the detector as 24x24, this results in a large number of rectangle features, over 180,000. In order to compute this features rapidly, it is used a technique called *integral image*. The integral image at a location ***x*,*y*** contains the sum of the pixels left and above of ***x*,*y***, given by:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y')$$

where $ii(x,y)$ is the integral image at location x,y and $i(x',y')$ is the original image at location x,y. The integral image can be computed following the next pair of recurrences:

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \end{aligned} \tag{2.47}$$

where s(x,y) is the cumulative sum of a row, therefore s(x,-1) = 0 and *ii*(-1,y) = 0. Using the integral image allows to sum any rectangle and compute it in four array references. Following the same logic, and taking in consideration that the rectangles are adjacent, a two-rectangle feature(edge feature) can be computed in six array references, eight array references are needed for a three-rectangle features(Line feature) and nine for a four-rectangle feature [57].

2.4.2 Classifier training

To learn a classifier, the majority of machine learning approaches need a feature set and a training set of positive and negative images. The Haar Feature-based is based on adaptive boosting, both to select the features and train the classifier [58]. Due to the fact that Adaptive booster by itself is used to boost the classification of a *weak* learning algorithm, and considering that the feature results in over 180,000 rectangles, computing the complete set is expensive in terms of computation, therefore it is necessary to combine the exact features to form an effective classifier. To achieve this goal, the weak learning algorithm is designed to select the single rectangle feature which best separates the positive samples from the negative samples. For all features, the weak learner determines the optimal threshold classification function, in a way that the minimum number of examples are misclassified. Therefore, the weak classifier $h_j(x)$ consists of a feature f_j, a threshold θ_j and a parity p_j that determines the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1, & \text{if} \quad p_j f_j(x) < p_j \theta_j \\ 0, & \text{otherwise} \end{cases}$$

where x is the 24x24 pixel sub-window of the image. The summary of the boosting process can be seen at [59].

2.4.3 Cascade Classifier

The cascade classifier is an algorithm that achieves an improve detection performance while reducing the computation time. This implementation is necessary to perform a more robust detection, taking in consideration that all the classifiers are considered as *weak classifiers*. The overall form of a cascade classifier is that of a degeneration tree. A positive result from one first classifier leads to a evaluation of a second classifier that has been adjusted to achieve high detection rates. If the second classifier gives a positive result, it triggers a third classifier, and so on. In any stage, if a negative result outcomes, this result in an immediately rejection of the sub-window. This is shown in a graphic way in Fig. 2.4. Each stage of the cascade is

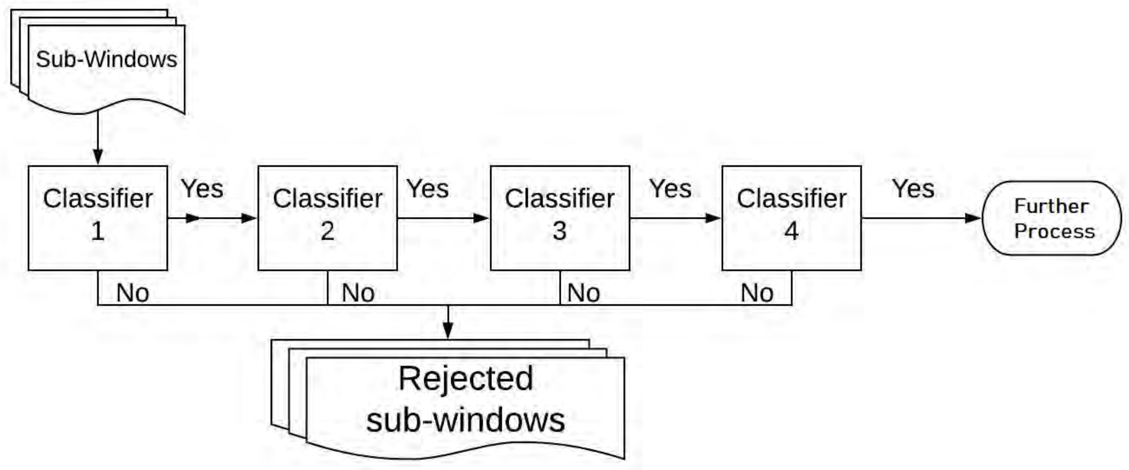


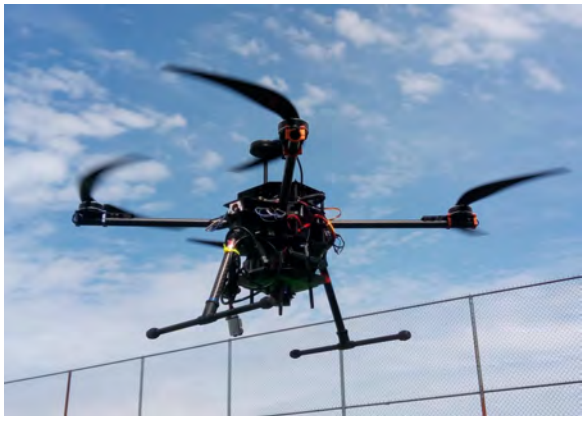
Figure 2.4: Schematic description of a detection cascade.

constructed by training classifiers using Adaptive boost, adjusting the threshold to minimize false negatives. The structure of the cascade is such as in any single image a majority of sub-windows are negative. This implies that the cascade attempts to reject as many negatives as possible in the earliest stages. This can be seen as a decision tree, where subsequent classifiers are trained using the ”experience” gained in the previous stages, so the further stages face a more difficult task in comparison to the first stages. In most cases, classifiers with more features will achieve higher detection rates and lower false positive rates, but this would require more time to compute. Instead, each stage of the cascade reduce the false positives rate and decreases the detection rate. A target is selected for the minimum reduction in false positives and the maximum decrease in detection. Each stage increase in complexity adding features until the target detection and false positives rates are met. The stages are added until the overall target for false positive and detection rate is met [59].

CHAPTER 3

Methods

This chapter describes the hardware and software of the systems that made up the two configurations used in the experiments. The first configuration is used to solve the Stability problem and the second configuration is used to execute the object tracking experiments. The same UAV is used for both configurations, just adding some other components that are necessary to the respectively experiments. Having said this, the first section of this chapter describes the UAV specifications and the subsequents sections explaining the software and hardware for the two configurations.


(a) UAV configuration for the first set of experiments.

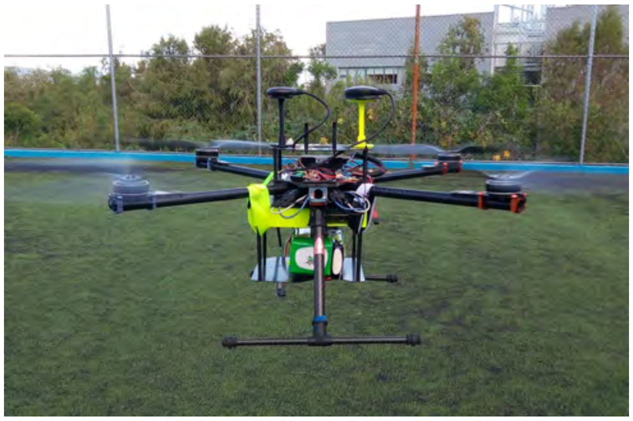

(b) UAV configuration for the second set of experiments.

Figure 3.1: The quad-rotor unmanned aerial vehicles used in this project. 3.1a shows the UAV with a camera at the bottom connected to a microcomputer to compute optical flow algorithm on-board. 3.1b shows the UAV with an On-Board computer and a monocular camera to perform the image processing.

3.1 UAV Configurations

The quad-rotors used in this project is shown at Fig. 3.1. For both configurations, this drone is equipped with a flight controller, a GPS, a camera rig, four ESCs, a set of four motors and their corresponding propellers, telemetry kit and a voltage regulator and for the first configuration, a Li-ion battery. The quad-rotor specifications are listed in Table 3.1. For the second configuration(Fig.3.1b), the battery is replaced by a Li-Po battery 6s 10A to compensate the extra weight that the computer and its protection add to the total weight of the UAV.

For both configurations, the state vector required by the EKF are:

- Quaternions: (q_0, q_1, q_2, q_3).
- Velocity (North, East, Down).

UAV design	
Parameter	Value
Span	70 [cm]
Height	26 [cm]
Weight	≈ 2 [kg]
Propulsion	Brushless motor 330 [kv]
	Propeller 17x5.5 [in]
Max. Load	≈ 3.5 [kg]
Battery type 1	Li-Ion 6s
	Capacity 9500 [mAh]
Battery type 2	Li-Po 6s
	Capacity 10000 [mAh]
Flight controller	Pixhack v3
Firmware	ArduCopter 3.5.5
Estimated flight time	15 [min]

Table 3.1: Quadrotor UAV parameters.

- Position (North, East, Down).
- Delta Angle bias (x,y,z).
- Delta Velocity bias.
- Wind Vector (North, East).
- Earth Magnetic Field Vector (North, East, Down).
- Body Magnetic Field Vector (x,y,z).

And the output vector consists in:

- Roll angle ϕ.
- Pitch angle θ.
- Yaw angle ψ.

- Velocities in North, Down and East, (V_N, V_D, V_E).
- Positions IN North, Down and East relative to UAV takeoff position (P_N, P_D, P_E).
- (x, y, z) gyro biases, (G_x, G_y, G_z).

3.2 Stabilization Configuration

3.2.1 PX4Flow

PX4Flow is an optical flow smart camera. It has a native resolution of 752x480 pixels and computes optical flow on a 4x binned and cropped area at 400 Hz, giving it a very high light sensitivity. It works well in indoor and low outdoor light conditions applications without the need of an external LED for illumination. Being open-source, it can be easily modified or reprogrammed to adapt to another computer vision tasks [60]. The PX4Flow is shown in Fig. 3.2 and the camera specifications are listed in table 3.2 [61]. Some experiments were

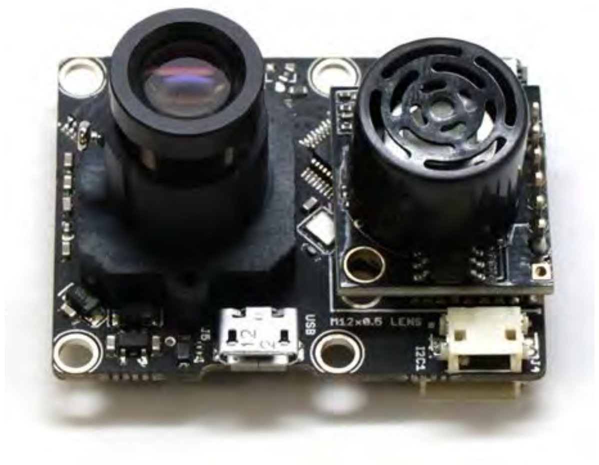


Figure 3.2: PX4Flow Camera.

conducted to show the effectiveness of this optical flow algorithm using a PX4Flow Optical

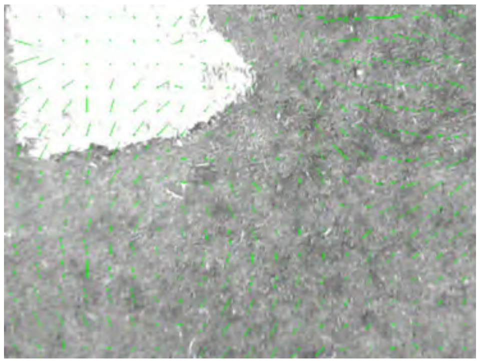
PX4Flow Specs	
Parameter	Value
Processor	168 MHz Cortex M4F CPU (128 + 64 KB RAM)
Image sensor	752x480 MT9V034
Lens	16 mm M12(IR block filter)
Size	45.5 mm x 35mm
Power consumption	115mA / 5V
Gyroscope	16-bit 2000deg/s 780Hz
Sonar	HRLV-EZ4

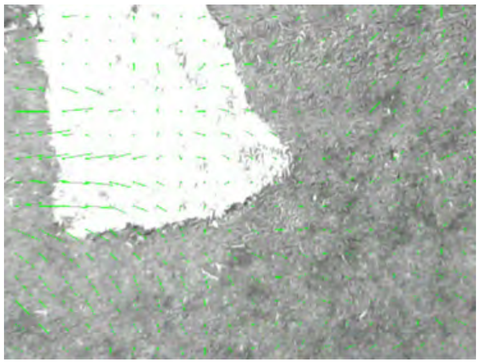
Table 3.2: PX4Flow Camera Specifications.

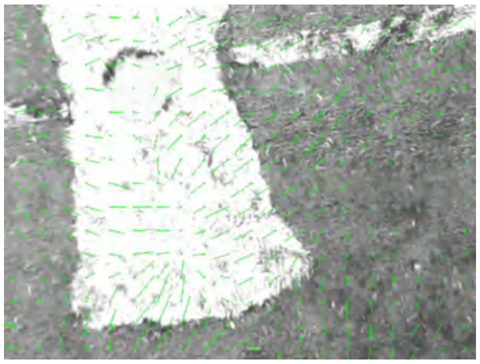
Flow Camera Board [17]. A sequence of four pictures are depicted in Fig. 3.3 showing the vectors indicating the OF direction. The complete video of this experiment can be seen at `https://youtu.be/_0Nj0PtnJsU`

3.2.1.1 Angular rate compensation and distance scaling

The On-board gyroscope values in the PX4Flow are stored into the main memory of the micro-controller and deliver the angular rotation rates of the camera. These data can be used for rotation compensation as the rotational parts of the motion field can be calculated using angular rates and focal length of the lens. In this way, the translational flow can be calculated even when the camera is rotating. When the ultrasonic sensor is attached, the optical flow values is scaled to its metric value assuming that the camera looks at a planar surface with the distance measured by the ultrasonic sensor. The details in this process is explained in [17].


(a)


(b)


(c)

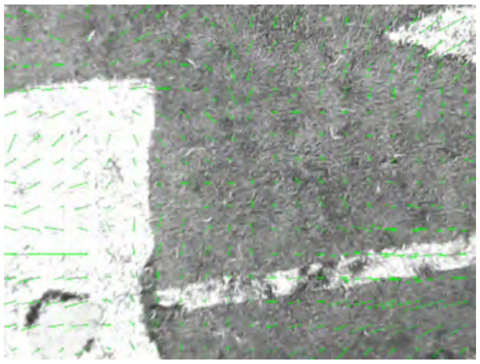

(d)

Figure 3.3: A sequence of ordered optical flow images taken from the drone. It can be observed in green the estimated velocity vectors.

3.3 Object Tracking Configuration

The Fig. 3.4 is a flow chart that describes the process of the object detection and tracking using the UAV. The program is designed in such a way that the object tracking ends when the battery of the UAV reaches a minimum established or if the user decides to finish it remotely. A frame is analyzed to try to detect the object that was previously trained by the Haar features cascade classifier. If the object is found, a bounding box surrounds the object and calculates the centroid. Then the navigation commands are performed to navigate the UAV in a certain way to achieve that the center of the image coincide with the centroid of the object. The width of the bounding box determines if besides of the roll and thrust movement are required or not, the UAV needs to make a pitch-forward to get closer to the object. If no object is detected, the UAV remains in its position in hovering mode. Once an action is performed, the program waits to the next frame and repeats the loop. Once the program ends, for any of the two reasons mentioned above, the UAV enters in ”Failsafe” mode, which causes it to position itself at five meters high w.r.t takeoff and once there, return to the point where it landed, to finally decrease the thrust and lands. All this process performed while Failsafe is known as *Return to launch* aka *RTL*. Appendix A shows the main python code used for this project, the functions used in the main code are explained in the subsequent content of this work.

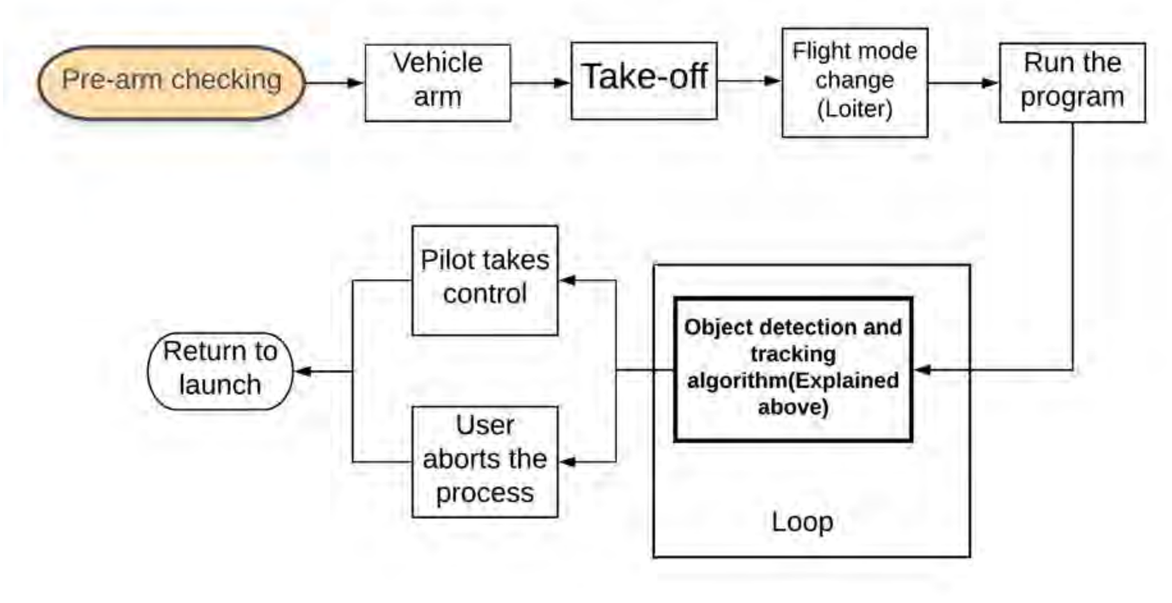


Figure 3.4: Test Flow chart describing the execution of the experiments.

3.3.1 NVIDIA Jetson TX2

For the object tracking experiment, few changes to the hardware are necessary. A Jetson TX2 developer kit computer is added (See Fig. 3.5). In this computer is stored the information about the training of the object to be detected. In addition, this computer perform all the process from obtaining the frames of the camera to detect the object according to the training and once located, to send the corresponding commands to the flight controller to navigate the UAV to track the object. The specifications of this On-board computer can be seen at [62]. The configuration of the UAV used for this experiments is shown in Fig. 3.1b.

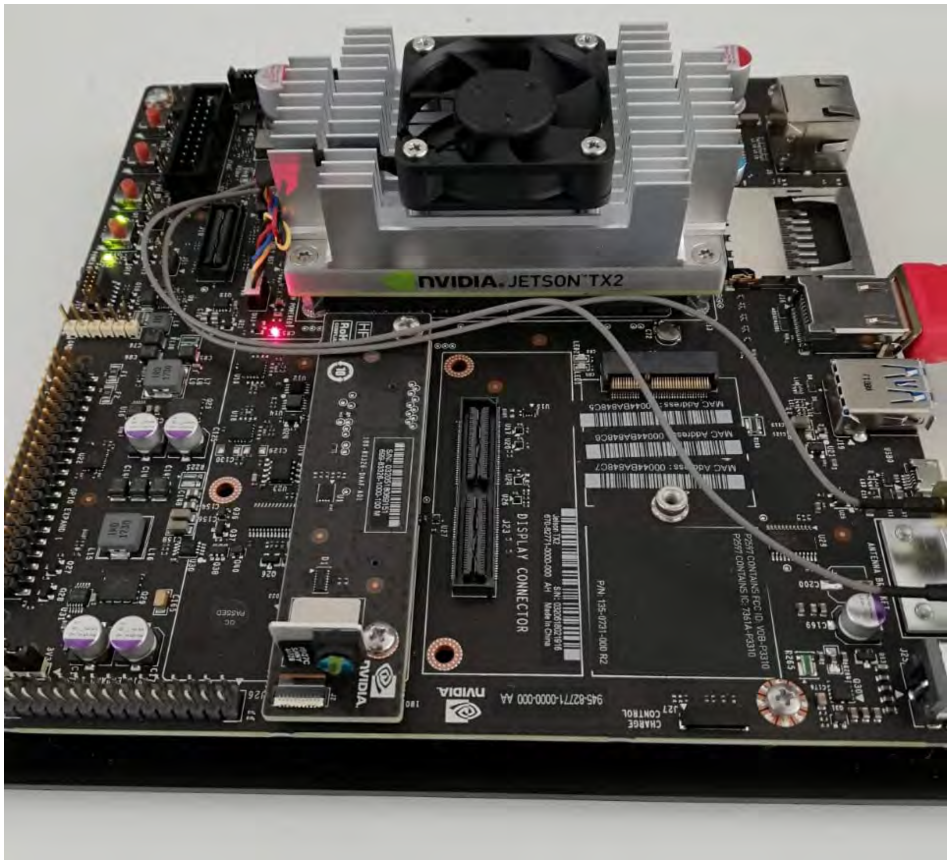
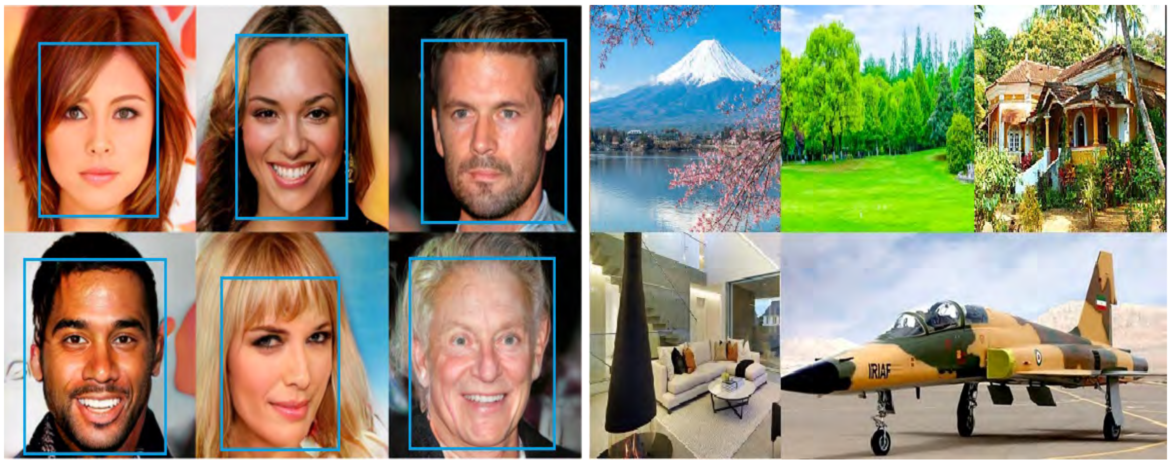


Figure 3.5: Jetson TX2 developer kit. On-board computer.



(a) Positive Images (b) Negative Images

Figure 3.6: Example of positive set of images and negative set of images. In the positive set of images is shown a bounding box surrounding the object in question within the image.

3.3.2 Haar cascades Training

For an own cascade classifier training, two types of images are needed: Negative and Positive. These sets must contain a big amount of images, to ensure the training to work in a better way. The recommended number of images is thousand images for the positive set and five hundred images for the negative set [63], [64]. An example of positives and negatives set of images is shown in Fig. 3.6

3.3.2.1 Negative samples

Negative samples are the set of images where the object can not be found, in other words, that contains everything the user do not want to detect. The set of negative images are taken from arbitrary images and must be prepared manually from the user and are enumerated in an special file. An example of negatives samples is shown at Fig. 3.6b. This is a text file where every line contains the image filename(including the directory if the images are not in the same location as the text file) of the negative sample images. The images in this set can be of different sizes, but each image should be at least equal or larger that the desired training windows size. The set of negative window samples will be used to tell the features classifier what not to look for, when trying to find the object of interest.

3.3.2.2 Positive samples

Exist two possible ways to generate the positive samples. One of them is making use of an opencv function call **$opencv_createssamples** [65]. This method boost the process to define what the model should look like generating ”artificial” images with the desired object. In this project the second method was performed tough. Like the negative samples, the user prepare manually the set of images. To ensure a robust model, the samples should cover a wide range of varieties that can occur within the object class. In the case of faces, the samples must consider different gender, emotions, races and even beard styles. An example of positive samples is shown at Fig. 3.6a For the upper body, the samples must consider different positions, haircuts styles and sizes. Inside the directory where the images are going to be located, a .dat file needs to be included. Each line of this file corresponds to an image. The first element of the line is the name of the image, followed by the number of objects within the image, followed by numbers indicating the coordinates of the object(s) bounding rectangle(s)(x, y, width, height) [66].

3.3.2.3 Training Haar cascade

Once the two sets of images are ready, the Haar cascade training is ready to be perform. Using OpenCV, the command to run is opencv_traincascade, that looks like:

```
$ opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1800 -numNeg 900 -numStages 10 -w 20 -h 20
```

Where **-data** refers to the direction where the trained classifier should be stored; **-vec** refers to the vec-file with positive samples; **-bg** refers to the background description file. This is the file direction containing the negative sample images; **-numPos** refers to the number of positive samples used in the training for every stage; **-numNeg** refers to the number of negative samples used in training for every stage; **-numStages** refers to the number of cascade stages to be trained. The more stages are selected, the training will be more robust but taking a longest time to compute; and finally **-w** and **-h** refers to the width and height of the training samples(in pixels) [67]. The output files is going to depend of the number of stages that the user selected. In the example, the directory **data** contains eleven .xml cascade training files, one per stage and the final.xml file that represents the whole cascade training.

3.3.3 Dronekit-Python

Dronekit-Python is a tool that runs on the On-board computer that allows users to create a communication between UAV's On-board computers and the ArduPilot flight controller using a low latency link. The On-board processing enhance the autopilot significantly, improving the behavior of the vehicle and performing tasks that are computationally intensive or time-sensitive, such as Computer Vision and path planning [68].

The main use of Dronekit-Python in this project consists in arming the data package that contains the information about the velocity and direction that the UAV needs to perform given the information of the object tracking. These data packages connect with the flight controller using MAVLink, a protocol for communicating for small unmanned vehicles [69], [70].

The function *Send_NED_Velocity* is the function that uses Dronekit libraries to connect with the ArduPilot flight controller. The function is defined as:

```python
def Send_NED_Velocity(velocity_x, velocity_y, velocity_z, duration, vehicle):
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, #
        0b0000111111000111, # type_mask
        0, 0, 0, # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z, # m/s
        0, 0, 0, # x, y, z acceleration
        0, 0)
    vehicle.send_mavlink(msg)
    #time.sleep(0.25)
```

The function asks for five parameters, being the fourth parameter useless for this application. The necessary parameters are *velocity_x, velocity_y, velocity_z* and *vehicle*. The first three parameters represent the velocities in NED(North-East-Down) directions, being in this case the Pitch-forward, Roll to the right and a negative thrust. The reasons for these directions is that the directions are with respect to the body frame. It is necessary to specify the frame on which the NED positions are going to be calculated. **MAV_FRAME_BODY_OFFSET_NED** is the line that allow the NED positions being relative to the current body frame heading, like is shown in Fig. 3.7. Thanks to this parameter it can be specified the position in x meters north, y meters east and (-) z meters of the current UAV position. Velocity directions are in the North, East, Down (NED) frame [71].

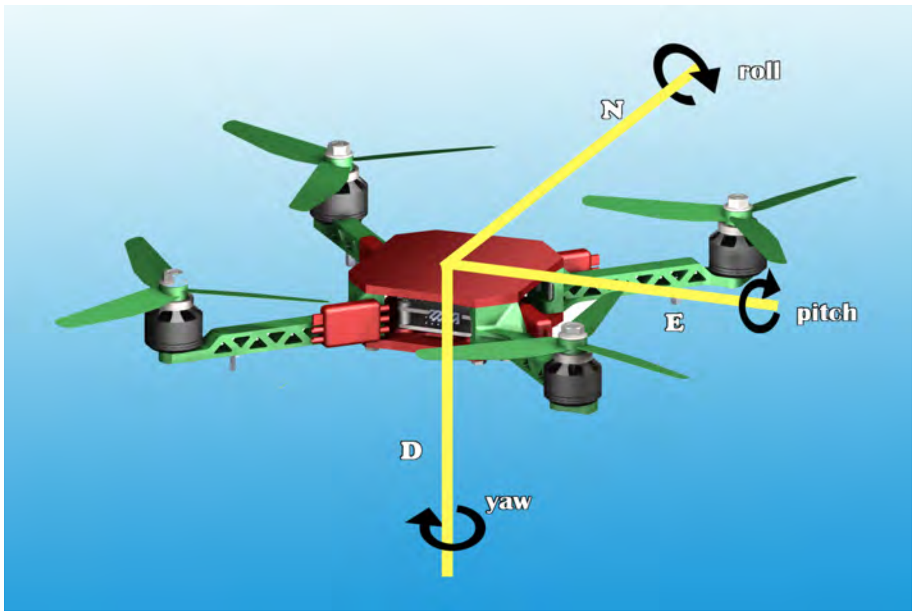


Figure 3.7: NED axes with respect to the body current frame.

CHAPTER 4

Results

This chapter shows the performed experiments and the results obtained from the methods shown in Chapter 3. One section of this chapter is about the results and discussion of the Stabilization Improvement, emphasizing the comparisons between the results data of the experiments using Optical flow with the results without using Optical flow. The other section of the chapter is about the results and discussion of the Object Tracking experiment, emphasizing the efficiency in the object detection and the movements of the UAV to track the object.

4.1 Stabilization Improvement

This section of the experiments are presented in two steps:

1. *Hovering flight*. The first part of the experiment consists of making flight tests in hov-

ering mode.

2. *Trajectory tracking flight.* In this experiment the UAV must follow a predefined path given as way-points.

In the next subsections we explain the obtained results of both aforementioned cases.

4.1.1 Hovering

The first experiment consists on flying the UAV in the flight mode called hovering, aka *loiter*. When this flight mode is activated, the UAV should keep flying in the actual position in which it is initially located. For comparison purposes, this mode is tested in the quadrotor platform shown at Fig. 3.1a, considering only GPS and INS data fusion by using the well-known *Arducopter* firmware, which uses a Kalman Filter to estimate the quadrotor states; then other test considering the addition of the PX4Flow. A video of the obtained results can be seen at `https://youtu.be/FTjoCMh64NQ`
The video consists in two clips with the UAV flying in hover mode. The first clip does not use the OF implementation, while the second clip does use it. Fig. 4.1 shows the performance of the flying test. In the first experiment, the flight was performed without the use of the OF algorithm, in order to visualize the normal behavior of this flight mode. Regarding the second experiment, it is performed with the optical flow implementation. It can be appreciate smoother movements without considerable variations, therefore the position error is minimized due the optical flow measurements.

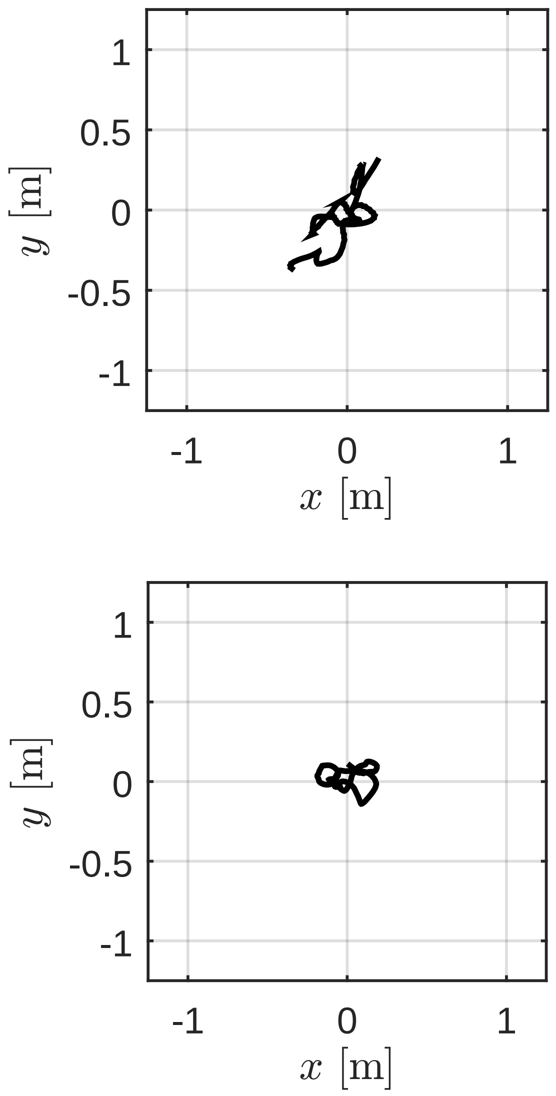


Figure 4.1: Loiter stability comparison. **Above**: Loiter mode **without using DOF algorithm**, only GPS and INS information is fused in the Kalman Filter. **Below**: Loiter mode **using DOF algorithm** and GPS-INS information. Both of them have the same axis limits (1.25m) to appreciate the differences.

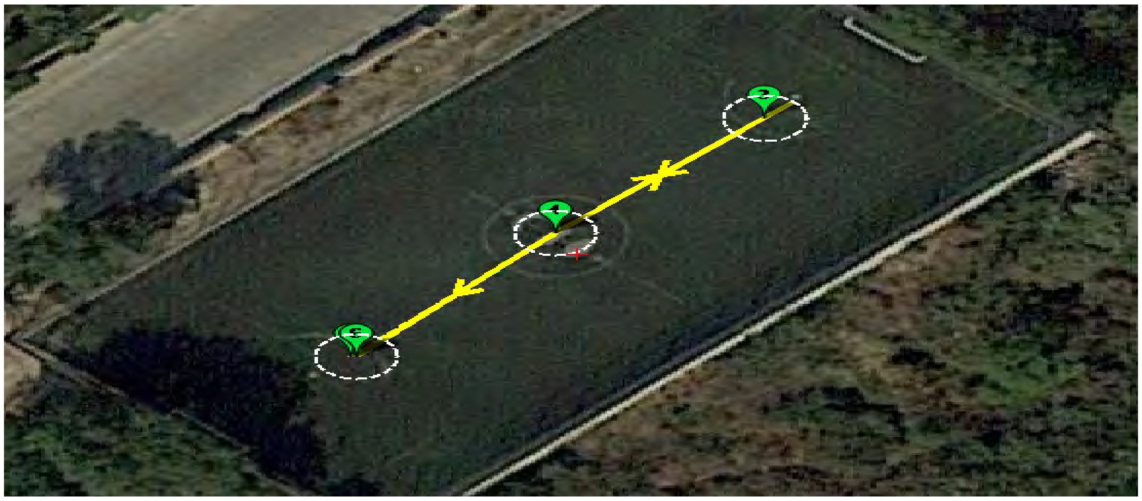


Figure 4.2: Google Earth view of the desired path. Such path consists of four waypoints located in a soccer field. The quadrotor UAV must take-off automatically from a given waypoint (home position). The first waypoint is in the center of the field and the UAV must go to that waypoint after takeoff. Second waypoint is in the bottom of the image. After reach this waypoint, the UAV turns 180 degrees and return again to the home position for landing.

4.1.2 Trajectory tracking

Now, a given trajectory must be followed by the Quad-rotor, in order to observe the performance of the fusion of OF algorithm together with GPS and INS devices. The trajectory is conformed by a line. Two waypoints are chosen to form the line; also two waypoints indicate take-off and landing position as is shown in Fig. 4.2. To replicate the common way to execute this flight path, first, such flight path was performed by the quadrotor without using OF algorithm. Some variations in height occurred during the flight path execution, this is due that only GPS and barometer information are fused to estimate drone's altitude. The flight path without optical flow was expected to have en error around 2.5 meters, this value is very close to the edge of the GPS error indicated by the manufacturer. The roll (ϕ) and pitch (θ) were in constant variations, showing inability to maintain a fixed course. Then, the same trajectory was performed but this time with the influence of optical flow data into the EKF.

In this test it can be observed that the movements are smoother w.r.t. the common approach. Also, the UAV position is smooth with minimal variations, demonstrating an improvement in the trajectory tracking. This performances are shown in Figs. 4.3 and 4.4, respectively. Further, a pair of videos showing the execution of the path can be seen at:

`https://youtu.be/jiyyPWVV3nE` (**Experiment with no OF implementation**.)

`https://youtu.be/rzISpDJs4t8` (**Experiment with OF implementation**.)

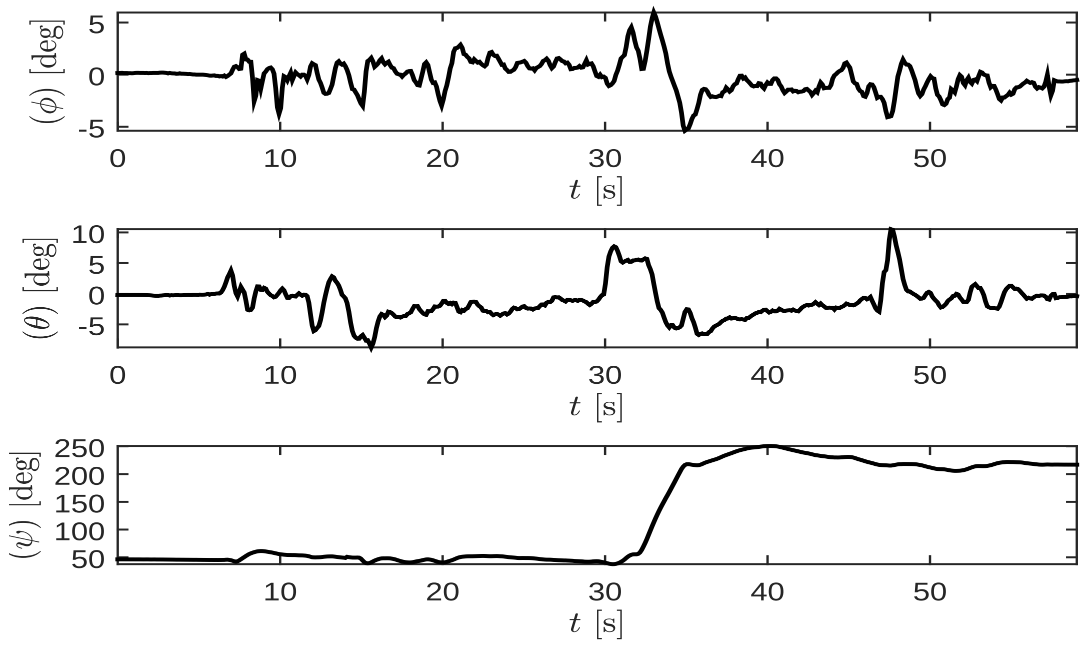


Figure 4.3: Angle displacement in roll (ϕ), pitch (θ) and yaw (ψ) during the trajectory tracking **without OF implementation**.

More interesting are the results in position estimation using dense optical flow algorithm. A graph in 2D is depicted at Fig. 4.5 which shows the performance of the proposed approach. In the performance without OF algorithm it can be appreciate a couple of details:

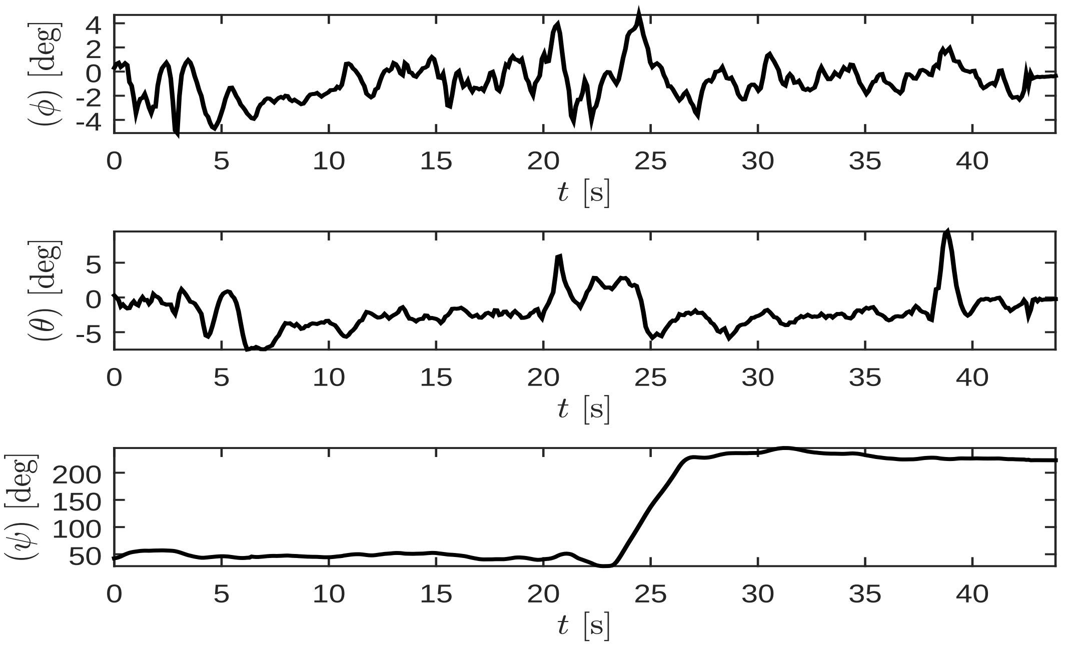


Figure 4.4: Angle displacement in roll (ϕ), pitch (θ) and yaw (ψ) during the trajectory tracking **with the OF implementation**.

- The drone take the turn of 180 degrees beyond the reference trajectory with an approximate difference of 1m in the *x* axis.
- In the second part of the trajectory, there is a maximum error (difference between reference and actual trajectory) of approximate 2.5m in the *y* axis.

In the performance with OF algorithm it can be observed that

- The UAV position in the turn of 180 degrees is in the same position as the reference trajectory.
- The maximum error noted is in the *y* axis during the first part of the path, that error is approximately equal to 1.5m.

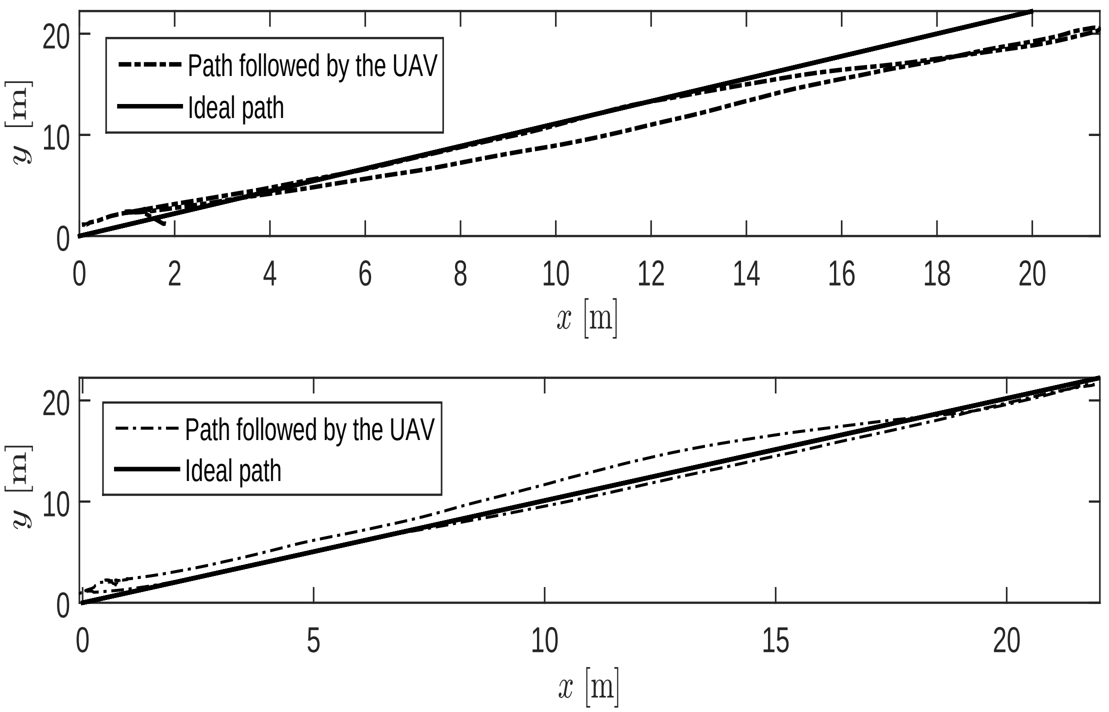


Figure 4.5: Graphs in the $x-y$ plane showing the UAV position performance during the trajectory tracking. Above, only GPS-INS information used in EKF; below, EKF with optical flow implementation.

In Figs. 4.6 and 4.7, the velocities in the north and east directions are shown. Such figures correspond to the cases with and without the use of the OF algorithm. Since the UAV desired trajectory is given as a complete straight line from home position to a predefined point, and then return to the home position, it is expected that in Fig. 4.6 and Fig. 4.7 presents a sinusoidal behavior. Fig. 4.6 tries to emulate this behavior, but not as good as Fig. 4.7 where OF is implemented.

Figs. 4.8 and 4.9 shows the three-dimensional view of the paths performed by the UAV. In both cases, using and not using OF algorithm, there was some height variations, this is

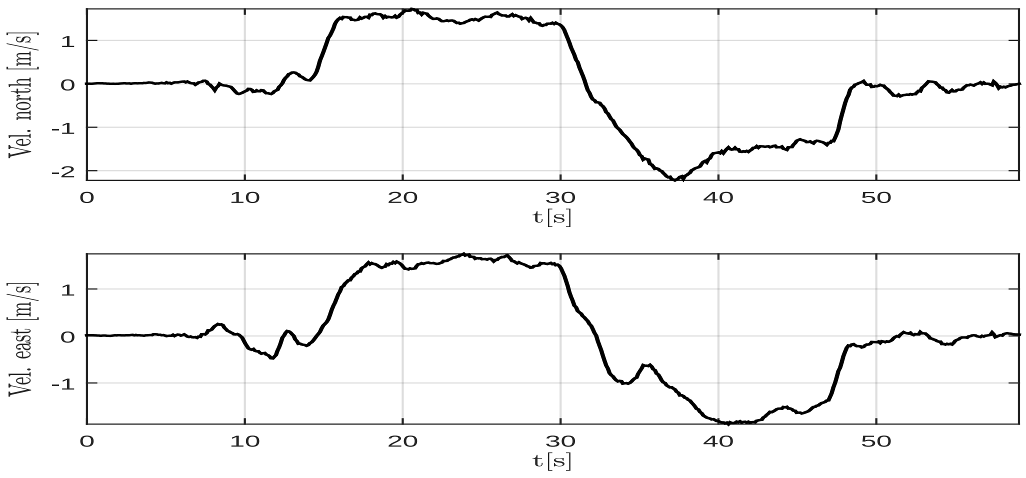


Figure 4.6: Velocities in the UAV north and east direction during trajectory tracking **without using OF**.

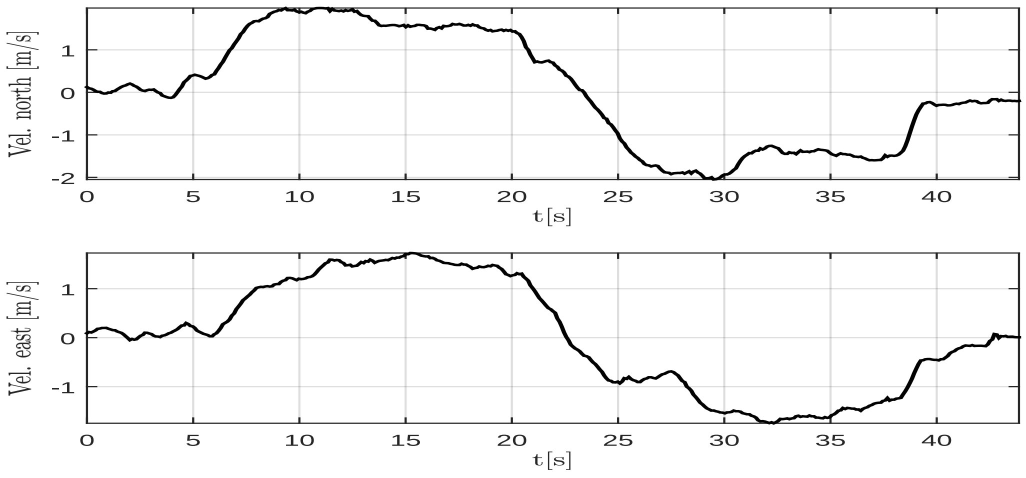


Figure 4.7: Velocities in the UAV north and east direction during the during the trajectory tracking **using optical flow**.

due to the fact that only information of GPS and barometer is used to estimate Quad-rotor altitude. As one can see from aforementioned figures, the improvement in the execution of the route when the optical flow is implemented is considerable smoother and more precise than the classical approach.

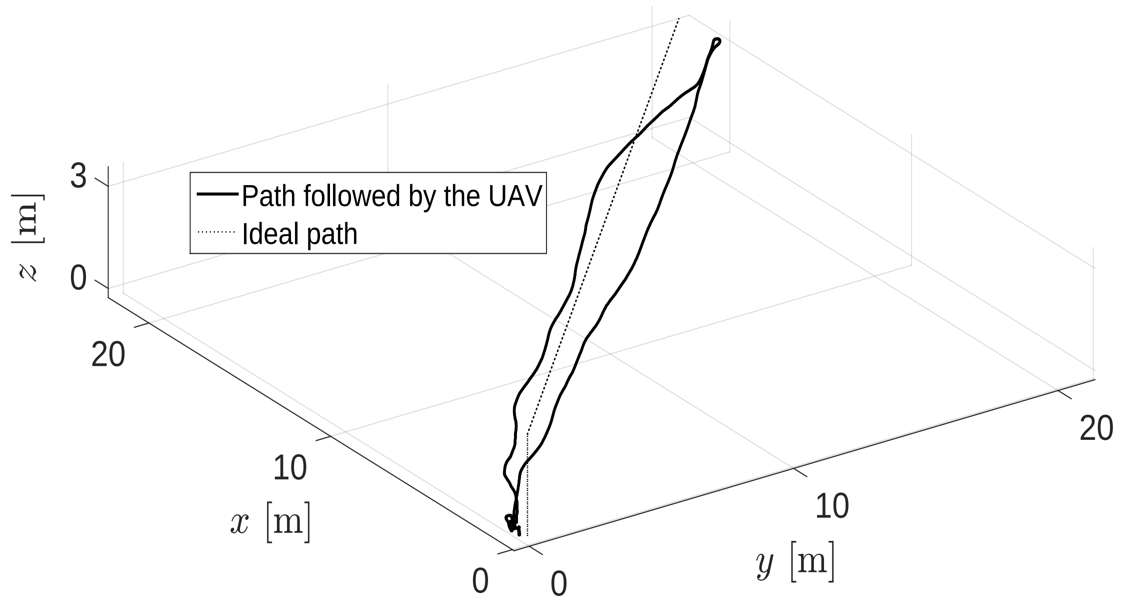


Figure 4.8: Three-dimensional view of the desired path and actual path performed by the UAV during the trajectory tracking **without OF algorithm**.

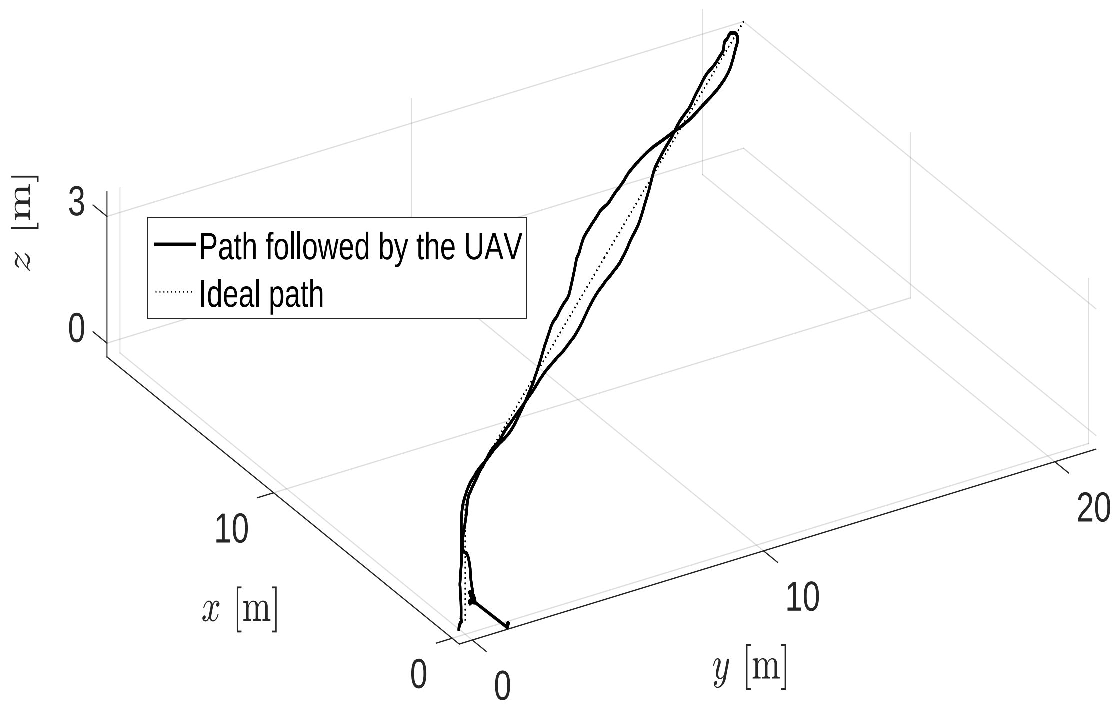


Figure 4.9: Three-dimensional view of the desired path and actual path performed by the UAV during the trajectory tracking **with OF algorithm**.

4.2 Object tracking

Once the stability problem is solved, the object tracking problem must be addressed. The flowchart that describes the steps for the experiments in the rest of this chapter is shown at Fig. 4.10.

4.2.1 Object detection

The object selected for the detection in this section is a human face, due to the number of specific parts of a face that can be chosen for the training in order to make the recognition more robust. In this case, the process to detect a face consists in two separate trainings.

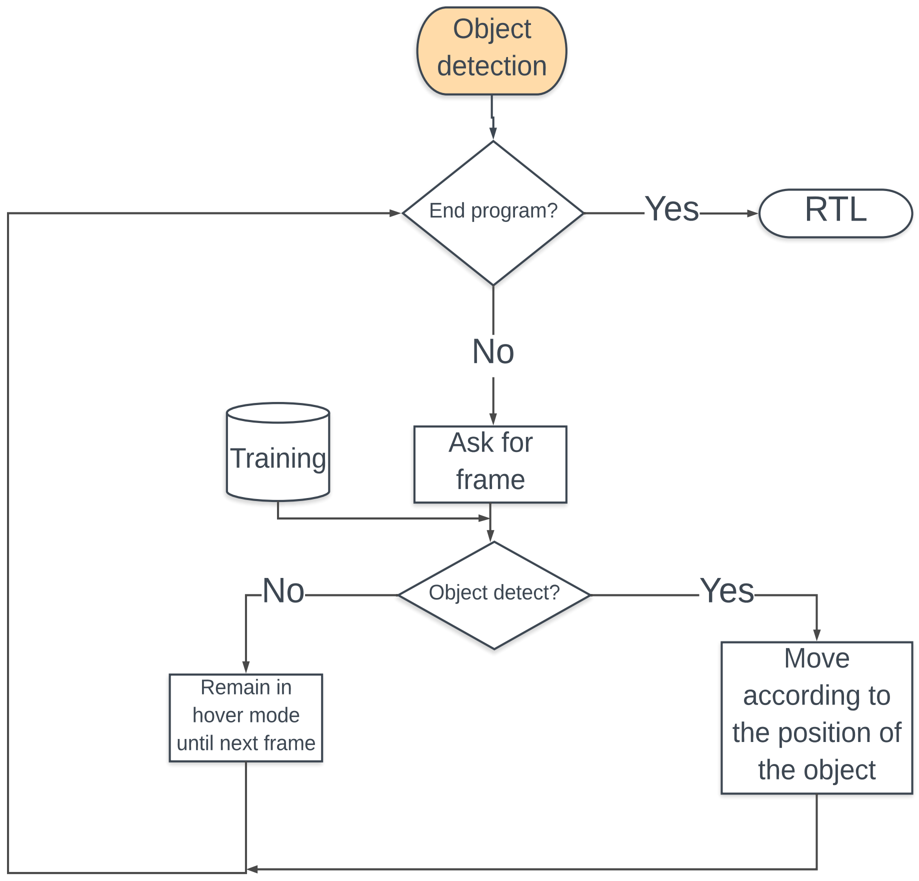

Object detection
End program?
Yes
RTL
No
Training
Ask for frame
No
Object detect?
Yes
Remain in hover mode until next frame
Move according to the position of the object

Figure 4.10: Chart describing the general flow of the object detection and tracking algorithm.

1. **Upper body detection:** The Haar Feature Classifier algorithm is trained to detect the upper body of a human being. A rectangle is drawn covering the full area of the image where the upper body is detected, as is shown in Fig. 4.11. This training is not enough, because no matter how well the process of training is performed, the object detection sometimes may lead to detect false objects.

2. **Complete face detection:** Now, the Haar Feature Classifier algorithm is trained to detect a complete face. In this way, the detection goes from something general to a

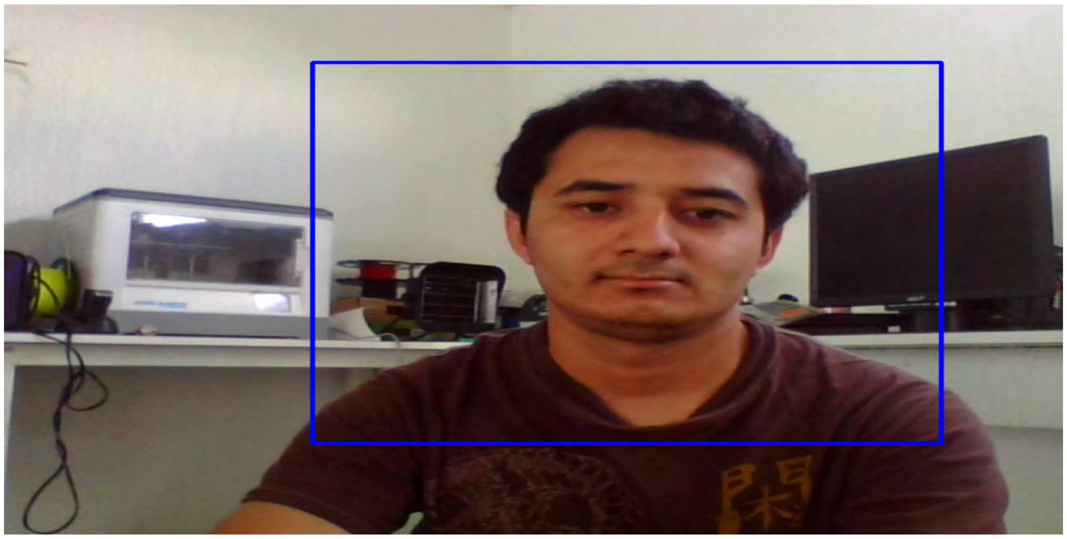


Figure 4.11: Upper body detection using Haar-cascade training.

more specific object. Like the previous case, rectangles are drawn around the detected object, as is shown in Fig. 4.12

3. **Combining both trainings:** Like it was previously stated, no matter how good the training is, it is always a percentage of probability that a false object would be detected. To minimize this probability, the face detection training is made only inside a loop where the upper body detection is performing, like is shown at Fig. 4.13. This is done to ensure no faces will be detected outside an upper body and also the fact that is less probably that in a false upper body detection also false faces would be detected.

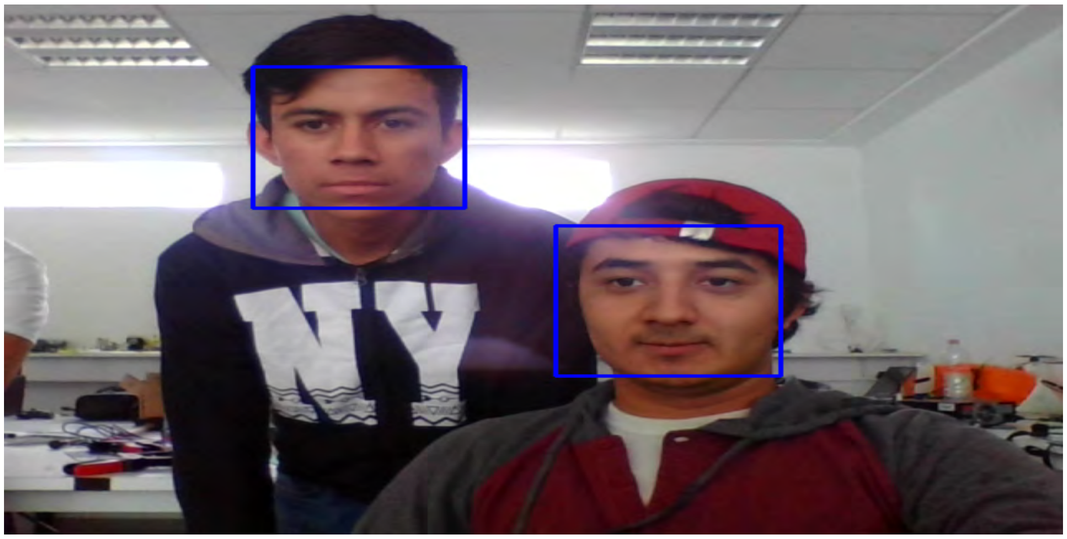


Figure 4.12: Face detection using Haar-Cascade training.

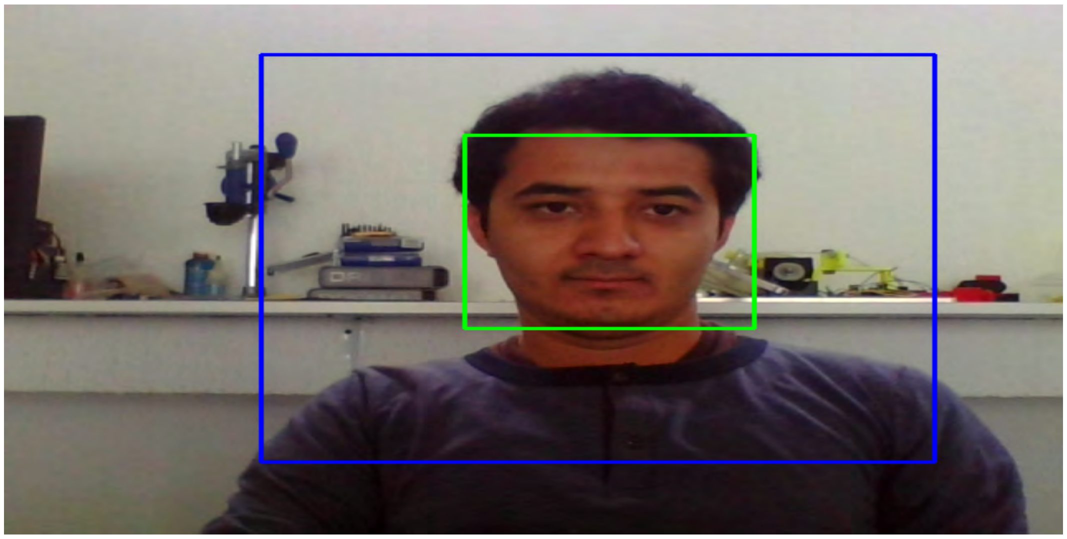


Figure 4.13: Combination of upper body and face detection using Haar-cascade training.

4.2.2 UAV Navigation

The next part of the experiment consists in identify the pixel where the centroid of the object is located from the image obtained. The way to calculate the centroid is very simple. Once the number of pixels of the width and height of the bounding box are obtained, the two values are divided by two. The resulting values are summed with each value of the coordinate from the top-left corner of the bounding box respectively. In this way, the two values obtained represent the coordinate of the centroid from the detected object. When the centroid is not in the center of the image, we can classify the position by three zones, and depending of the zone, will be the movement that the UAV needs to make, these can also be classified in three types, as is shown in Fig. 4.14. The velocity and direction of the UAV to move to the desired pixel is assigned depending of the position of the centroid, for example

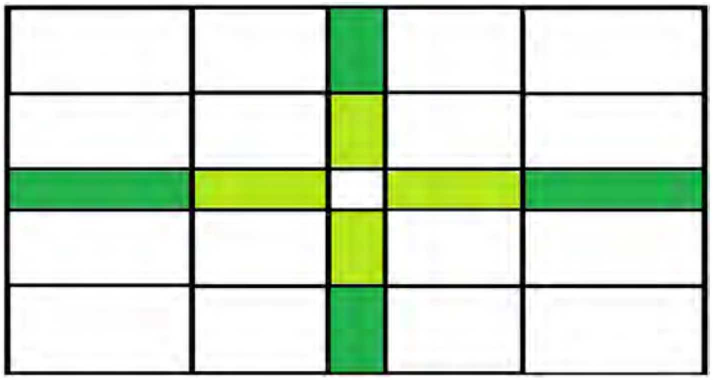
```
    elif x<290 and y<210:
        if x<145 and y<105:
            print ("Izquierda rapido Arriba rapido")
            Send_NED_Velocity(0,-roll_f,-th_f,1,vehicle)
        elif x<145 and y>105:
            print ("Izquierda rapido Arriba lento")
            Send_NED_Velocity(0,-roll_f,-th_s,1,vehicle)
        elif x>145 and y<105:
            print ("Izquierda lento Arriba rapido")
            Send_NED_Velocity(0,-roll_s,-th_f,1,vehicle)
        elif x>145 and y>105:
            print ("Izquierda lento Arriba lento ")
            Send_NED_Velocity(0,-roll_s,-th_s,1,vehicle)
```

If in this portion of the code, the first condition is satisfied, means that the centroid of the object detected is located at top-left of the image, then another condition needs to be satisfied to determine if the distance of the object to the center of the image is near or far. This leads to four possible cases represented in four *if-elif* conditions in where the object could be (The first and third conditions are represented in Fig. 4.14b and the second and fourth conditions are represented in Fig. 4.14c), this determines if the (in this case) negative thrust movement needs to be in a fast or slow velocity and if a fast or slow negative roll is required. The variable *roll_f* and *roll_s* defines a fast and slow velocity in roll angle respectively, while *th_f* and *th_s* are variables to define a fast and slow velocity in thrust respectively. A similar block of code is used for each and every one of the cases shown in Fig. 4.14.

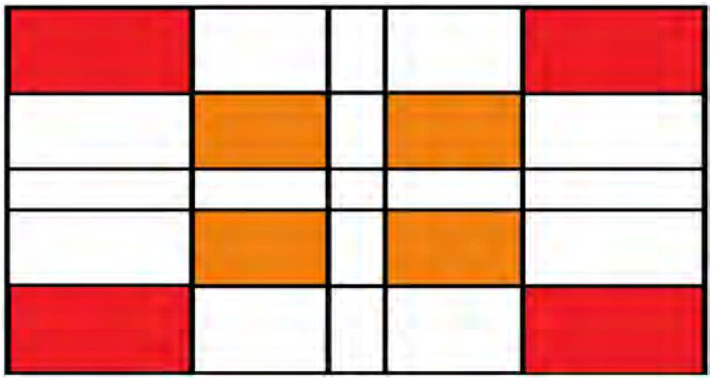
Figs. 4.15, 4.16 and 4.17 shows two sequences of images representing the object recognition and tracking performed by the UAV. The full video of the external view and the drone cam view, can be found following the next link:

`https://youtu.be/DKqalep2rMM`

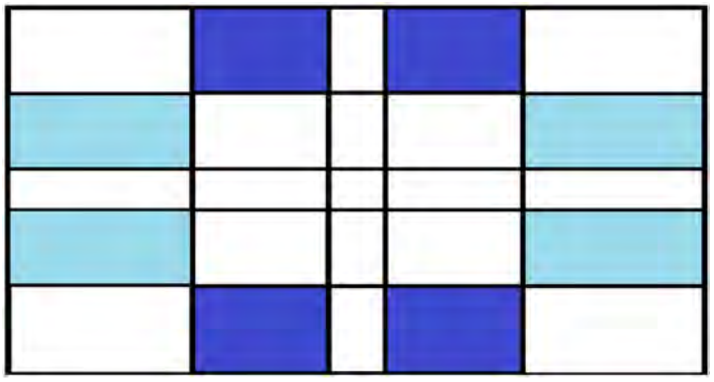
As it was stated in subsection 4.2.1, the video generated during the flight shows that the program analyzes about four frames per second, having a reaction time between the object detection and the UAV movement of about 0.2 seconds. It also can be observed that bounding boxes appear in places where the algorithm detects false upper bodies, but in none of these cases a face where found, only appearing inside the real upper-body detection.



(a) Movements in simple direction. Soft green means slow movement, while hard green means fast movement.

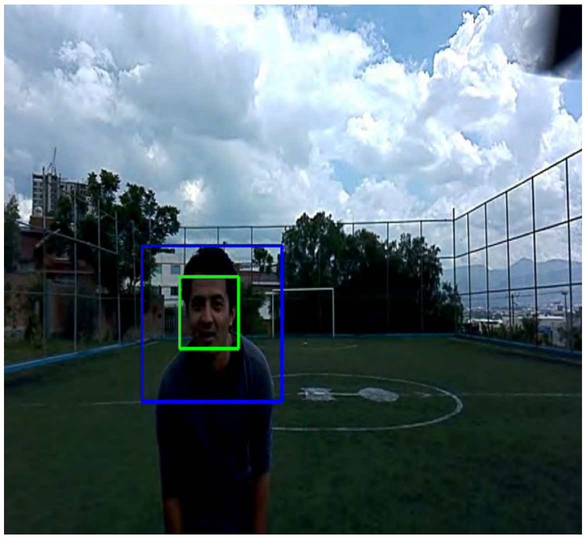


(b) Movements in two directions at same speed. Orange means slow movement, while red means fast movement.

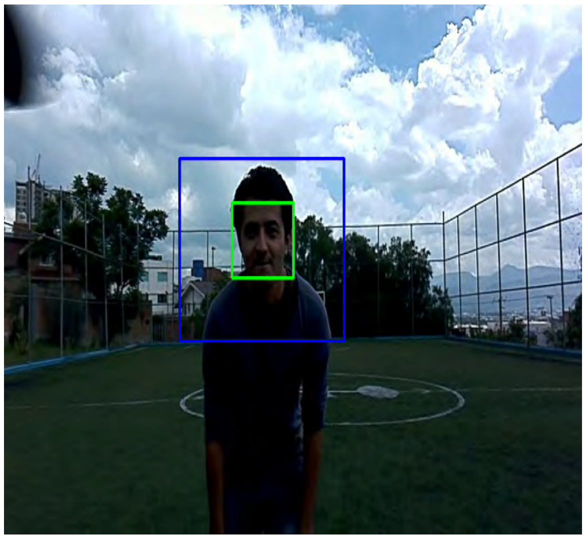


(c) Movements in two directions at different speed. Soft blue means fast roll movement and slow thrust movement. Hard blue means fast thrust movement and slow roll movement.

Figure 4.14: Roll and thrust movements depending of the zone of the image where the center of the object is detected.


(a) Drone cam view 1

(b) External view 1


(c) Drone cam view 2

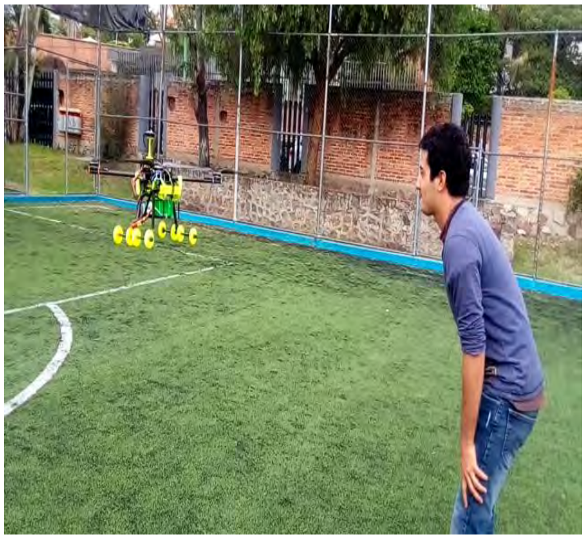
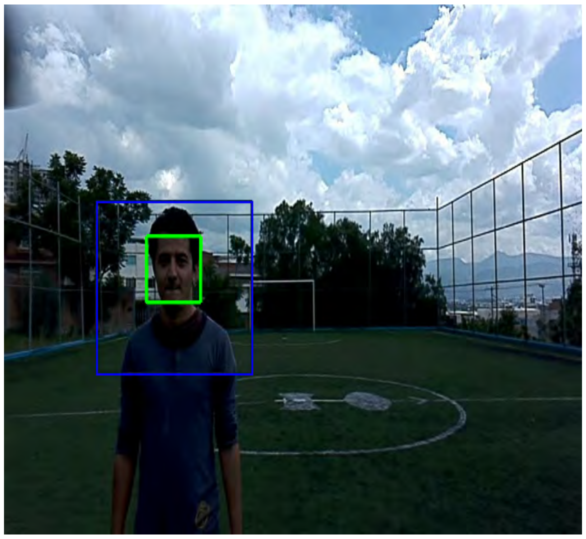
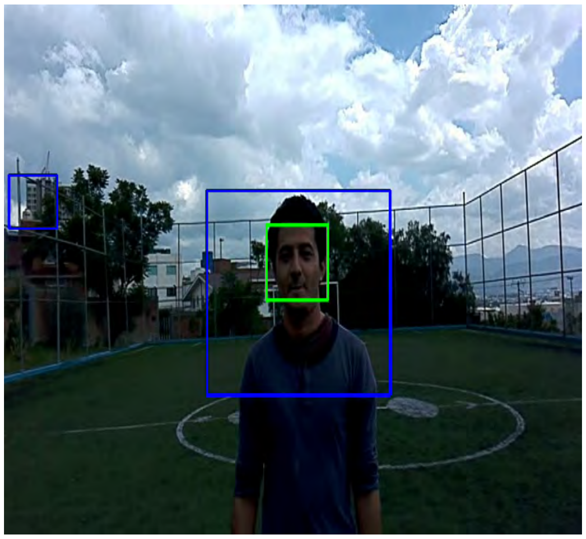

(d) External view 2

Figure 4.15: Part 1 of a sequence of ordered object tracking images taken from the drone perspective and from an external video recording the full process respectively. The images of each row corresponds to the same moment showing both views.


(a) Drone cam view 3

(b) External view 3


(c) Drone cam view 4

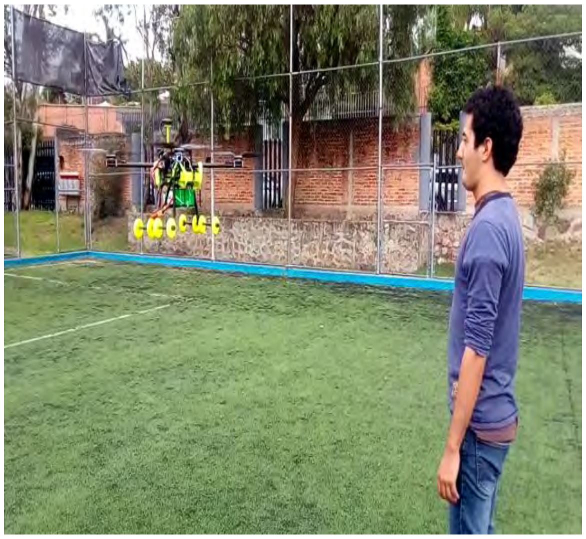
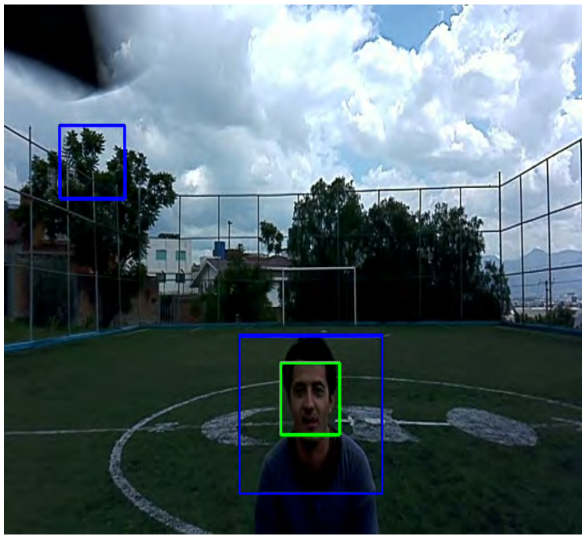
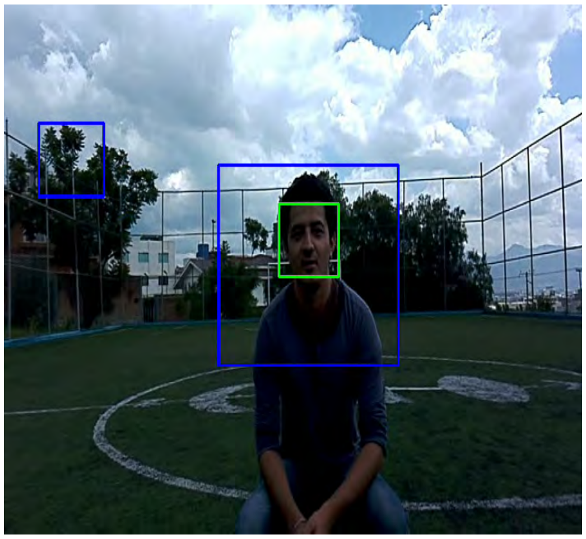

(d) External view 4

Figure 4.16: Part 2 of a sequence of ordered object tracking images taken from the drone perspective and from an external video recording the full process respectively. The images of each row corresponds to the same moment showing both views.


(a) Drone cam view 5

(b) External view 5


(c) Drone cam view 6

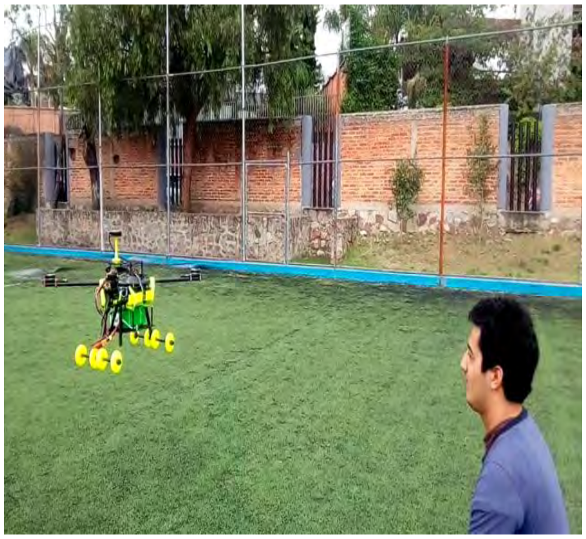

(d) External view 1

Figure 4.17: Part 3 of a sequence of ordered object tracking images taken from the drone perspective and from an external video recording the full process respectively. The images of each row corresponds to the same moment showing both views.

CHAPTER 5

Conclusions and Future Work

5.1 Conclusions

The implementation of a vision-based system to the INS-Sensors result in a plausible improvement in the UAV position estimation. The conducted experiments demonstrates its effectiveness. Since the quadrotor UAV is a underactuated system, the movements in roll and pitch were smoother when the optical flow is implemented. The yaw movements are very similar in both approaches, as one can see at Figs. 4.3 and 4.4, since it is only used to initiate the return trajectory. Two flight modes have been tested: hovering aka loiter and trajectory tracking. In both cases the drone presented minimal error with the use of the OF algorithm compared to the conventional EKF approaches where no optical flow algorithm is implemented. As a consequence the Quad-rotor flies in a more stable way in $x-y$ position. However, the experiments can be improved in a way that the altitude remain constant.

The object detection using Haar Feature-based Cascade Classifiers method seems to work admissible. To ensure that the training of the object to be detected is reliable, the process must be meticulous and follow all the steps correctly. For example, at least 500 images with object and 250 images without object are required to guarantee a good training. If possible, the number of positive images should as big as possible or identify in the best way their location of the object within the image. It is also advisable to include in the positive images objects that are similar to those of our interest, and again delimit exclusively where the object is located, to give our training a greater capacity to discern. Including objects that are similar to those of our interest in the negative images is also a good idea. The decision to perform a two-steps detection, one being the upper body and another the face, demonstrates to work better taking in consideration that just the upper body detection tends to detect in a wrong way, but the possibility to detect false faces in a false upper body decreases exponentially. In fact, during the experiments never was the case where a complete object(Upper body + face) was detected wrongly.

5.2 Future Work

Future works involving the OF implementation include: a) improving the UAV altitude estimation adding lidar or vision-based altitude systems; b) testing the proposed approach in fixed-wing aircraft; and c) compare the results in an specific application, for instance: orthomosaics generations and NDVI indexes for precision agriculture.

With respect to the object detection and tracking, future work include: a) Train the algorithms to detect the Helipad sign, in order to develop a project of safe landing once the camera detects the desired object.; b) Use different methods to object detection. Methods like neuronal networks or tensor-flow should be addressed to make a comparison about their performance. Particularly, tensor flow can be implemented taking advantage of GPU features, therefore the Jetson TX2 would be better exploited; c) The capability of the UAV to perform yaw movements in case the object makes a rotation on its own axis. The most effective method could be detect when the two points that form the horizontal top line of the bounding box are not in the same row of the image or in a certain hysteresis, indicating that the object is rotated with respect to the camera and therefore, with respect to the Quad-rotor heading; and d) Replicate the full process with the UAV looking for the object from an aerial view.

CHAPTER 6

Bibliography

[1] H. Sossa-Azuela, F. Cuevas, C. Aguilar-Ibanez, and H. Benite-Munoz, “3-D Cartesian Geometric Moment Computation using Morphological Operations and its Application to Object Classification,” *Ingenieria, investigacion tecnologica*, vol. 8, pp. 111 – 123, 06 2007. [Online]. Available: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-77432007000200005&nrm=iso

[2] J. A. M. Rodríguez, “Computer vision of the foot sole based on laser metrology and algorithms of artificial intelligence,” *Optical Engineering*, vol. 48, pp. 48 – 48 – 14, 2009. [Online]. Available: https://doi.org/10.1117/1.3275455

[3] S. Zhou, G. Flores, R. Lozano, and P. Castillo, “Vision-based window estimation for mav in unknown urban environments*,” *IFAC Proceedings Volumes*, vol. 46, no. 30, pp. 107 – 111, 2013, 2nd IFAC Workshop on Research, Education

and Development of Unmanned Aerial Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015402794

[4] S. Zhou, G. Flores, E. Bazan, R. Lozano, and A. Rodriguez, "Real-time object detection and pose estimation using stereo vision. an application for a quadrotor mav," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, Nov 2015, pp. 72–77.

[5] J. Martinez-Carranza, L. Valentin, F. Márquez-Aquino, J. C. González-Islas, and N. Loewen, "Obstacle detection during autonomous flight of drones using monocular slam," *Journal of Research in Computing Science*, vol. 114, p. 111–124, 2016. [Online]. Available: https://pdfs.semanticscholar.org/be0f/07f5ab53b672a96ccd5025ffad378236203c.pdf

[6] I. Becerra, L. M. Valentín-Coronado, R. Murrieta-Cid, and J.-C. Latombe, "Reliable confirmation of an object identity by a mobile robot: A mixed appearance/localization-driven motion approach," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1207–1233, 2016. [Online]. Available: https://doi.org/10.1177/0278364915620848

[7] P. Henkel and A. Sperl, "Real-time kinematic positioning for unmanned air vehicles," in *2016 IEEE Aerospace Conference*, March 2016, pp. 1–7.

[8] S. Takahashi, N. Kubo, N. Yamaguchi, and T. Yokoshima, "Real-time monitoring for structure deformations using hand-held rtk-gnss receivers on the wall," in *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Sept 2017, pp. 1–7.

[9] R. He, S. Prentice, and N. Roy, “Planning in information space for a quadrotor helicopter in a gps-denied environment,” in *2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 1814–1820.

[10] S. Ahrens, D. Levine, G. Andrews, and J. P. How, “Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments,” in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 2643–2648.

[11] S. E. Tsai and S. H. Zhuang, “Optical flow sensor integrated navigation system for quadrotor in gps-denied environment,” in *2016 International Conference on Robotics and Automation Engineering (ICRAE)*, Aug 2016, pp. 87–91.

[12] Y. Hong, X. Lin, Y. Zhuang, and Y. Zhao, “Real-time pose estimation and motion control for a quadrotor uav,” in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, June 2014, pp. 2370–2375.

[13] H. Chao, Y. Gu, J. Gross, G. Guo, M. L. Fravolini, and M. R. Napolitano, “A comparative study of optical flow and traditional sensors in uav navigation,” in *2013 American Control Conference*, June 2013, pp. 3858–3863.

[14] M. B. Rhudy, H. Chao, and Y. Gu, “Wide-field optical flow aided inertial navigation for unmanned aerial vehicles,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 674–679.

[15] L. Fusini, J. Hosen, H. H. Helgesen, T. A. Johansen, and T. I. Fossen, “Experimental validation of a uniformly semi-globally exponentially stable non-linear observer for gnss- and camera-aided inertial navigation for fixed-wing uavs,” in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2015, pp. 851–860.

[16] J. Qi, N. Yu, and X. Lu, “A uav positioning strategy based on optical flow sensor and inertial navigation,” in *2017 IEEE International Conference on Unmanned Systems (ICUS)*, Oct 2017, pp. 81–87.

[17] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys, “An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications,” in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 1736–1741.

[18] M. Perez-Ruiz, D. C. Slaughter, C. Gliever, and S. K. Upadhyaya, “Tractor-based real-time kinematic-global positioning system (rtk-gps) guidance system for geospatial mapping of row crop transplant,” *Biosystems Engineering*, vol. 111, no. 1, pp. 64 – 71, 2012.

[19] “Precise positioning system for uavs.” [Online]. Available: https://www.suasnews.com/2016/05/43820/

[20] D. A. Mercado, G. Flores, P. Castillo, J. Escareno, and R. Lozano, “Gps/ins/optic flow data fusion for position and velocity estimation,” in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2013, pp. 486–491.

[21] U-blox, “Neo-7 u-blox 7 gnss modules data sheet,” U-blox, USA, Tech. Rep. UBX-13003830-R07, Nov. 2014.

[22] A. M. de Oca, L. Arreola, A. Flores, J. Sanchez, and G. Flores, “Low-cost multispectral imaging system for crop monitoring,” in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 443–451.

[23] J. M. Vazquez-Nicolas, E. Zamora, I. Gonzalez-Hernandez, R. Lozano, and H. Sossa, “Towards automatic inspection: crack recognition based on quadrotor uav-taken im-

ages," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 654–659.

[24] X. Li and L. Yang, "Design and implementation of uav intelligent aerial photography system," in *2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 2, Aug 2012, pp. 200–203.

[25] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. International Conference on Image Processing*, vol. 1, Sept 2002, pp. I–I.

[26] S. Zhang, "Object tracking in unmanned aerial vehicle (uav) videos using a combined approach," in *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 2, March 2005, pp. ii/681–ii/684 Vol. 2.

[27] D. Deneault, D. Schinstock, and C. Lewis, "Tracking ground targets with measurements obtained from a single monocular camera mounted on an unmanned aerial vehicle," in *2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 65–72.

[28] T. Gaspar, P. Oliveira, and C. Silvestre, "Uav-based marine mammals positioning and tracking system," in *2011 IEEE International Conference on Mechatronics and Automation*, Aug 2011, pp. 1050–1055.

[29] O. Araar and N. Aouf, "Visual servoing of a quadrotor uav for autonomous power lines inspection," in *22nd Mediterranean Conference on Control and Automation*, June 2014, pp. 1418–1424.

[30] N. Stefas, H. Bayram, and V. Isler, "Vision-based uav navigation in orchards," *IFAC-PapersOnLine*, vol. 49, no. 16, pp. 10 – 15, 2016, 5th IFAC Conference on Sensing,

Control and Automation Technologies for Agriculture AGRICONTROL 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896316315646

[31] L. R. G. Carrillo, G. R. F. Colunga, G. Sanahuja, and R. Lozano, "Quad rotorcraft switching control: An application for the task of path following," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 4, pp. 1255–1267, July 2014.

[32] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward end-to-end control for uav autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 115–123.

[33] S. Lee, T. Shim, S. Kim, J. Park, K. Hong, and H. Bang, "Vision-based autonomous landing of a multi-copter unmanned aerial vehicle using reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 108–114.

[34] A. L. Chan, S. L. Tan, and C. L. Kwek, "Sensor data fusion for attitude stabilization in a low cost quadrotor system," in *2011 IEEE 15th International Symposium on Consumer Electronics (ISCE)*, June 2011, pp. 34–39.

[35] A. Chovancová, T. Fico, Ľuboš Chovanec, and P. Hubinsk, "Mathematical modelling and parameter identification of quadrotor (a survey)," *Procedia Engineering*, vol. 96, pp. 172 – 181, 2014, modelling of Mechanical and Mechatronic Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877705814031981

[36] F. Sabatino, "Quadrotor control: modeling, nonlinear control design, and simulation," Master's thesis, KTH, School of Electrical Engineering (EES), Sweden, 2015.

[37] A. Alaimo, V. Artale, C. Milazzo, A. Ricciardello, and L. Trefiletti, “Mathematical modeling and control of a hexacopter,” in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2013, pp. 1043–1050.

[38] T. Luukkonen, “Modelling and control of quadcopter,” in *Independent research project in applied mathematics*, August 2011.

[39] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton, NJ, USA: Princeton University Press, 2012.

[40] T. S Alderete, “Simulator aero model implementation,” 05 2018.

[41] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises*, 4th ed. Wiley-IEEE Press, 2012.

[42] R. Faragher, “Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes],” *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 128–132, Sept 2012.

[43] G. Welch and G. Bishop, “An introduction to the kalman filter,” Chapel Hill, NC, USA, Tech. Rep., 1995.

[44] M. Ulusoy. (1999) Controls-tech-talks. [Online]. Available: https://github.com/MeldaUlusoy/Controls-Tech-Talks

[45] M. Montemerlo and S. Thrun, *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[46] S. D. Levy. (2016) The extended kalman filter: An interactive tutorial for non-experts. [Online]. Available: https://home.wlu.edu/~levys/kalman_tutorial/

[47] C. Derivation, , and M. I. Ribeiro, "Kalman and extended kalman filters:," 2004.

[48] K. Noor, E. A. Siddiquee, D. Sarma, A. Nandi, S. Akhter, S. Hossain, K. Andersson, and M. S. Hossain, "Performance analysis of a surveillance system to detect and track vehicles using haar cascaded classifiers and optical flow method," in *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, June 2017, pp. 258–263.

[49] D. Patel and S. Upadhyay, "Optical flow measurement using lucas kanade method," *International Journal of Computer Applications*, vol. 61, no. 10, pp. 6 – 10, 2013.

[50] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, J. Bigun and T. Gustavsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370.

[51] G. Farnebäck, "Polynomial expansion for orientation and motion estimation," LINKÖPING UNIVERSITY, SWEDEN, Tech. Rep., 2002.

[52] A. Kasinski and A. Schmidt, "The architecture and performance of the face and eyes detection system based on the haar cascade classifiers," *Pattern Analysis and Applications*, vol. 13, no. 2, pp. 197–211, May 2010. [Online]. Available: https://doi.org/10.1007/s10044-009-0150-5

[53] H. Cevikalp, B. Triggs, and V. Franc, "Face and landmark detection by using cascade of classifiers," in *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, April 2013, pp. 1–7.

[54] OpenCV. (2018) Face detection using haar cascades. [Online]. Available: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

[55] V. G. J. Prateek and D. Millan, *OpenCV By Example*. 35 Livery Street, Birmingham B3 2PB, UK.: Packt Publishing Ltd., 2016.

[56] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan 1998, pp. 555–562.

[57] P. Viola and M. Jones, "Robust real-time object detection," in *International Journal of Computer Vision*, 2001.

[58] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002200009791504X

[59] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Dec 2001.

[60] P. D. Team. (2013) Px4flow smart camera. [Online]. Available: https://docs.px4.io/en/sensor/px4flow.html

[61] P. autopilot, *PX4FLOW – Smart Camera QUICK START*.

[62] N. Corporation. (2017) Nvidia jetson the embedded platform for autonomous everything. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[63] opencv dev team. (2011) Cascade classifier training. [Online]. Available: https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html

[64] R. Ajna and T. Hersan. Training haar cascades. [Online]. Available: https://memememememememe.me/post/training-haar-cascades/

[65] PythonProgramming.net. (2016) Creating your own haar cascade opencv python tutorial. [Online]. Available: https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/

[66] OpenCV. (2016) Cascade classifier training. [Online]. Available: https://docs.opencv.org/3.4.2/dc/d88/tutorial_traincascade.html

[67] sentdex. (2016) Training haar cascade object detection - opencv with python for image and video analysis 20. [Online]. Available: https://youtu.be/eay7CgPlCyo?list=PLQVvvaa0QuDdttJXlLtAJxJetJcqmqlQq

[68] D. Robotics. (2015) Guiding and controlling copter. [Online]. Available: http://python.dronekit.io/guide/copter/guided_mode.html

[69] I. Free Software Foundation. (2007) Mavlink developer guide. [Online]. Available: https://mavlink.io/en/

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

[71] A. D. Team. (2016) Copter commands in guided mode. [Online]. Available: http://ardupilot.org/dev/docs/copter-commands-in-guided-mode.html#set_position_target_local_ned

APPENDIX A

Main code - Object detection and tracking.

```
from dronekit import connect, VehicleMode
from GuideCommands import Take_Off, Yaw_Motion, movement, face_detection
import time
import cv2
import sys
sys.path.append('/usr/local/lib/python2.7/dist-packages')
upper_body_cascade = cv2.CascadeClassifier('HS.xml')
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml
    ')
#Start connection
print "Starting connection..."
connection_string ="/dev/ttyUSB0"
print("Connecting to vehicle on: %s" % (connection_string,))
vehicle = connect(connection_string, wait_ready=True)
```

```
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
vehicle.mode = VehicleMode("GUIDED")
print "Guided"
x,y,w = face_detection(frame,upper_body_cascade,face_cascade)
while True:
    movement(x,y,w,vehicle)
    ret, frame = cap.read()
    x,y,w = face_detection(frame,upper_body_cascade,face_cascade)
    time.sleep(0.2)
    if cv2.waitKey(1) == 27:
        break
cap.release()
cv2.destroyAllWindows
print "Closing connection..."
vehicle.close()
print("Done")
```

APPENDIX B

```
from pymavlink import mavutil
from dronekit import connect, VehicleMode, LocationGlobalRelative
import math
import time
import cv2
import numpy

#Function to send NED velocity vector to vehicle
def Send_NED_Velocity(velocity_x,velocity_y,velocity_z,duration,vehicle):
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,       # time_boot_ms (not used)
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, #
        0b0000111111000111, # type_mask
        0, 0, 0, # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z, # m/s
        0, 0, 0, # x, y, z acceleration
        0, 0)
    vehicle.send_mavlink(msg)
    #time.sleep(0.25)
```

```

def movement(x,y,w,vehicle):
    roll_f=0.3
    roll_s=0.2
    th_f=0.25
    th_s=0.175
    pitch=0.2
    if w>90:
        if x<290 and y>210 and y<270:
            if x<145:
                print ("Izquierda Rapido")
                Send_NED_Velocity(0,-roll_f,0,1,vehicle)
            else:
                print ("Izquierda lento")
                Send_NED_Velocity(0,-roll_s,0,1,vehicle)
        elif x>350 and y>210 and y<270:
            if x<495:
                print ("Derecha lento")
                Send_NED_Velocity(0,roll_s,0,1,vehicle)
            else:
                print ("Derecha rapido")
                Send_NED_Velocity(0,roll_f,0,1,vehicle)
        elif y<210 and x>290 and x<350:
            if y>105:
                print ("Arriba lento")
                Send_NED_Velocity(0,0,-th_s,1,vehicle)
            else:
                print ("Arriba rapido")
                Send_NED_Velocity(0,0,-th_f,1,vehicle)
        elif y>270 and x>290 and x<350:
            if y<375:
                print ("Abajo lento")
                Send_NED_Velocity(0,0,th_s,1,vehicle)
            else:
                print ("Abajo rapido")
                Send_NED_Velocity(0,0,th_f,1,vehicle)
```

```
def face_detection(frame,upper_body_cascade,face_cascade):
    posx = 320
    posy = 240
    ew = 91
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    body = upper_body_cascade.detectMultiScale(gray,1.3,2)
    for (x,y,w,h) in body:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_color = frame[y:y+h,x:x+w]
        face = face_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in face:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
            posx = round((ew/2)+ex+x)
            posy = round((eh/2)+ey+y)
    #cv2.imshow('img',frame)
    return posx, posy, ew
```