Target tracking using visual servoing: the Parrot AR Drone 2.0 case study

Alexandra Domingos Reis Pereira

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors: Prof. Alexandra Bento Moutinho
Prof. José Raul Carreira Azinheira

January 2021

Acknowledgments

Throughout a very turbulent and unimaginable year, where I had to reconcile my first job with writing this thesis it seems like a dream that I am finally delivering it. After many hours of hard work, I am proud to have finished this thesis.

However, everyone knows it is impossible to perform research and experimental simulations without any difficulties. The help of my supervisors, family and friends was essential when facing these obstacles. I would first like to thank both my supervisors, Professor Alexandra Moutinho and Professor José Raul Azinheira, for their guidance, encouragement and advice during the process of writing this thesis. I am grateful for their very valuable comments on this present work and their openness to any question that came up.

Secondly, I want to express the immense gratitude I owe to my parents and my sister. Thank you for your unconditional support and patience.

I also would like to thank my college friends for all the good moments we passed together, their friendship and their constant support throughout my university path.

I want to specially thank my friend Luis for the time we both dedicated helping each other in the lab.

Last but not least, a special word to my friend and colleague Pedro Santos, who worked with me on all projects during my master's program and gave me a lot of assistance in this thesis.

Resumo

Esta investigação propõe a utilização de controlo por visão para controlar a posição de um veículo áereo não tripulado (VANT) em relação a um ponto de interesse no espaço. De forma a poder testar o controlo por visão experimentalmente e em laboratório, o quadrirotor Parrot AR Drone 2.0 é utilizado. As duas abordagens principais de controlo por visão são implementadas, controlo por visão baseado na imagem (IBVS) e controlo por visão baseado em posição (PBVS), a fim de que um ponto genérico no espaço, que esteja ou não a mover-se, seja seguido pelo veículo. Controladores proporcional integral e derivativo (PID) são usados para controlar a posição e a orientação do quadrirotor na execução desta tarefa. O atraso na aquisição da imagem do alvo a seguir apresenta-se como uma das principais desvantagens em utilizar a visão no controlo do movimento do quadrirotor. Tendo em conta que isso potencia a saída do alvo do campo de visão da câmara, foi desenvolvido um algoritmo que garante que o alvo continua a ser seguido e que o controlo por visão continua a funcionar. Este algoritmo previne também cenários em que a imagem está corrompida e o alvo não é detetado na imagem. Todas as medições dos sensores embutidos no veículo e as estimações da posição do alvo são validadas.

Este trabalho foi desenvolvido no âmbito do projeto Eye in the Sky - Utilização de Balões de Alta-Altitude para Apoio à Decisão em Operações de Combate a Incêndios Rurais (https://adai.pt/eyeinthesky/), financiado pela Fundação para a Ciência e a Tecnologia (PCIF/SSI/0103/2018).

Palavras-chave: controlo por visão, controlo por visão baseado na imagem, controlo por visão baseado na posição, seguimento de alvo, Parrot AR Drone 2.0

Abstract

This thesis proposes the use of visual servoing for controlling the unmanned aerial vehicles (UAV) position when monitoring wildfires. In order to experimentally test the control strategy in laboratory conditions, the quadrotor Parrot AR Drone 2.0 is used. The image-based visual servo (IBVS) and the position-based visual servo (PBVS) approaches are implemented for target tracking purposes. Proportional-Integral-Derivative (PID) controllers are implemented for position and heading control of the quadrotor, so that a generic interest point in space is tracked in both static and moving conditions. One of the most important drawbacks of real vision sensing discovered is the presence of a constant time delay on the image acquisition process. This causes the target point to leave several times the field of view, thus, an algorithm is developed to ensure that the target continues to be tracked and the visual servo control keeps on working. Plus, the algorithm prevents situations in which the image is corrupted and the target is not detected. All drone sensors measurements and target position estimations are validated through a motion capture system that tracks both the drone and the target during the experiments.

This work was done in the scope of project Eye In The Sky – Using High-Altitude Balloons for Decision Support in Wildfire Operations (https://adai.pt/eyeinthesky/), funded by Fundação para a Ciência e a Tecnologia (PCIF/SSI/0103/2018).

Keywords: visual servoing, image-based visual servo, position-based visual servo, target tracking, Parrot AR Drone 2.0

Contents

List of Tables

List of Figures

Nomenclature

Variables

e	Error
$\mathbf{K_{ext}}$	Extrinsic camera matrix
$\mathbf{K_{int}}$	Intrinsic camera matrix
$\mathbf{L_e}$	Interaction matrix
$\mathbf{P}$	Projection matrix
$\mathbf{R}$	Rotation matrix
$\mathbf{s}$	Vector of image features
$\mathbf{t}$	Translation vector
ω	Angular velocity
ϕ, θ, ψ	Euler angles
F	Focal length in pixels
f	Focal length
O	Frame origin
o_x, o_y	Image principal
r	Rotational velocity Cartesian component
s_θ	Skew parameter
s_x, s_y	Scale factors
u, v, w	Velocity Cartesian components
x, y	Image coordinates
X, Y, Z	Cartesian components
λ	Visual servo control law gain

D	Derivative gain
e_{ss}	Steady state error
g	Gravity acceleration
G_c	Controller transfer function
G_p	Plant transfer function
I	Integral gain
K	DC gain
M_p	Overshoot
P	Proportional gain
t_s	Settling time
Ts	Sample time
$\mathbf{I}$	Identity matrix

Subscripts

c	Camera frame
d	Drone frame
f	Fixed frame
i, e	Euclidean image coordinates
i, p	Image coordinates in pixels
r	Rotational component
t	Translational component
x, y, z	Cartesian components

Superscripts

*	Desired
T	Transpose

Chapter 1

Introduction

1.1 Motivation

Investment on Unmanned Aerial Vehicles (UAV) technologies is increasing alongside demand. This emerging technology has become very attractive to different market segments with a growing number of applications.

Besides the more traditional use cases, commonly related to military operations, like surveillance and monitoring [1], there is a growing interest on infrastructure inspection and goods delivery applications, [2]. On those cases, the efficiency is one of the motivators to replace human resources.

The UAV can also play a big role on local and global scale emergencies that endangers human lives, for instance the current pandemic. They actively support the government authorities in combat of coronavirus by disinfecting highly populated places, enforcing social distancing and delivering health goods [3].

Moreover, for operations in extreme weather conditions the UAV could be very helpful not only on search and rescue missions, but also on monitoring disasters impact to support decision making, [4].

The *Eye in the Sky* project is a good example of this UAV application. This project, in which this thesis is integrated, aims to develop an unmanned aerial platform for supporting firefighting operations on monitoring the perimeter of the fires and detecting hotspots in burning areas.

For this project, the images captured by the camera onboard the UAV are an essential source of data. Furthermore, the camera can be used as a sensor to control the relative pose of UAV, a technique known as visual servoing. Vision sensors are a very interesting option for when there is a limited payload to be carry out, since most of UAV includes a camera, and they can operate in a GPS-denied environment. Additionally, visual servoing techniques have become very popular for target tracking applications and many researches show its high effectiveness, [5],[6] and [7].

1.2 Topic Overview

An eye-in-hand camera configuration is used in a visual servo control loop when the robot has a camera mounted on it, like most of UAV do. Alternatively, the camera could be fixed in the workspace instead, creating an eye-to-hand configuration, [8]. For target tracking tasks eye-to-hand configuration requires the camera to be fixed on the target. This could be a successful approach for UAV landing purposes, [9]. However, in case of tracking a non-physical target and/or high altitude flight is required, for instance on wildfire perimeter monitoring or road following [7], this is not applicable. That is why the eye-in-hand configuration is commonly adopted on those target tracking applications.

There are two main types of visual servoing: image-based visual servo (IBVS) and position-based visual servo (PBVS). The IBVS uses image plane measurements for feedback control, which results in a strongly nonlinear and coupled system. On the other hand, the PBVS uses the image features to reconstruct the 3D pose of the target object and, so, an error in the Cartesian space is given as input to the controller. This approach became advantageous as it allows tasks being defined on the Cartesian space. However, the object's pose highly depends on the estimation of the camera intrinsic parameters given by camera calibration, which can introduce errors in the tracking law.

In order to determine the 3D parameters of the object required for PBVS scheme, the epipolar geometry can be used, assuming there is a stereo vision, [10]. Even if a monocular camera is available, there are always two subsequent images. Thus, the essential matrix can give the rotation matrix and translation vector up to a scalar factor between views. However, when the camera views are almost coincident the parameters estimations get poor. That is why the homography is usually preferred when the object image features belongs to the same plane.

Besides image data, additional information about the object model is needed, such as its real size or depth from the camera, so its position in 3D space is determined, [11]. To measure depth, sensors based on laser or sound ranging technologies could be integrated on the UAV. A stereo vision system can also be used for depth measurement through the triangulation method, [11]. Other researches propose alternative visual servo approaches to address the lack of depth information for monocular vision. Some of them are the so called hybrid visual servo schemes, [10] [12], first introduced by [13]. These schemes decouple the translation and rotation components through homography without needing depth data. Both image and Cartesian space data regulate the controller inputs in the hybrid visual servo approaches. Other researchers suggest adaptive control strategies based on *a priori* available image data to compensate for the depth lack, [14], [15]. Recent studies show very satisfying results of employing deep neural networks on depth estimation, [16], [17].

If visual servoing is applied to a quadrotor, a type of UAV, other challenges have to be addressed, due to its underactuated properties. Since the horizontal velocities are coupled with the roll and pitch variations, these rotations are misinterpreted as image errors. This could lead the object that is being tracked to be outside the field of view, [6]. In order to prevent the errors, [18] and [6] created a virtual camera independent from the quadrotor rotation to compensate for the roll and pitch movements. Even though this method provides a smoother quadrotor motion in image and Cartesian space than the classical IBVS, it does not prevent the target from leaving the field of view. The implementation of a Kalman filter is proposed by [19] to solve the temporary loss of the target.

1.3 Objectives

The present thesis aims to understand the effectiveness of visual servoing on controlling the UAV position when facing the Eye in the Sky project scenarios mentioned before. In order to do this, a laboratory experimental setup will be created. Here, the two basic visual servoing approaches, IBVS and PBVS, will be implemented in the quadrotor Parrot AR Drone 2.0. They will be first tested in a simulator environment and afterwards in the experimental setup. In both environments, the Parrot will be following a generic point in space, which ultimately could be a flame. In order to do this, a control strategy will be created so the position and heading of the quadrotor are controlled. Two different scenarios are tested for each visual servoing approach, one in which the target is motionless and the other where it is moving.

As previously mentioned, there is a high possibility of the target being outside the camera field of view when is tracked by a quadrotor. Thus this situation will also be addressed in this thesis.

At the end, this work aims to conclude about the drawbacks of using vision as a sensor and the advantages of choosing one of the visual servo approaches over the other in a experimental context.

1.4 Thesis Outline

This thesis is divided into six chapters. Chapter 1 introduces the present work by explaining the motivation behind it, the main objectives and the related work concerning visual servoing and its application on quadrotor control. In Chapter 2 the perspective projection relations of the pinhole camera model are derived and IBVS and PBVS control laws are presented. The model of the quadrotor dynamics, as well as the controllers designed for it, are the topics addressed in Chapter 3. Subsequently, the target tracking problem formulation is tackled in Chapter 4. In addition, all the algorithms and tools required to perform the target tracking task on both simulator and experimental environments are explained. Chapter 5 covers the results obtained for both visual servo approaches in the different environments, considering if the target is moving or not. Finally, Chapter 6 provides a summary of the research and briefly discusses future work directions.

Chapter 2

Visual Servoing

2.1 Camera Model

In order to describe the camera optics, the so-called pinhole camera model is used [20]. This is a purely geometric model that through the perspective projection maps the relation between 3D world coordinates and 2D image coordinates.

Consider a point W represented by its cartesian coordinates (X_f, Y_f, Z_f) in a fixed frame $\{O_f, \mathbf{X_f}, \mathbf{Y_f}, \mathbf{Z_f}\}$. The same point is defined with respect to another reference frame, in this case the camera frame $\{O_c, \mathbf{X_c}, \mathbf{Y_c}, \mathbf{Z_c}\}$, as (X_c, Y_c, Z_c), whose coordinates are obtained through a transformation between frames. This transformation is described in an homogeneous representation by

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R_{cf}} & \mathbf{t_{cf}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_f \\ Y_f \\ Z_f \\ 1 \end{bmatrix} \tag{2.1}$$

where $\mathbf{R_{cf}}$ and $\mathbf{t_{cf}}$ are the rotation matrix and the translation vector, respectively, which converts the fixed frame coordinates into the camera frame coordinates. Following the pinhole camera model illustrated in Figure 2.1, the Z-axis of the camera frame corresponds to the optical axis and (X_c, Y_c, Z_c) is projected on the image plane through a ray that connects this point and the optical center, which is the origin of the camera frame, O_c. The intersection of the ray with the image plane gives the image point w, which can be defined in a 2D euclidean frame parallel to the camera frame and centered at the principal point p as

$$\begin{bmatrix} x_{i,e} \\ y_{i,e} \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \tag{2.2}$$

where f denotes the focal length (world distance from the optical center to the image plane) and $(x_{i,e}, y_{i,e})$ are the euclidean image coordinates. With homogeneous coordinates (2.2) can be rewrit-

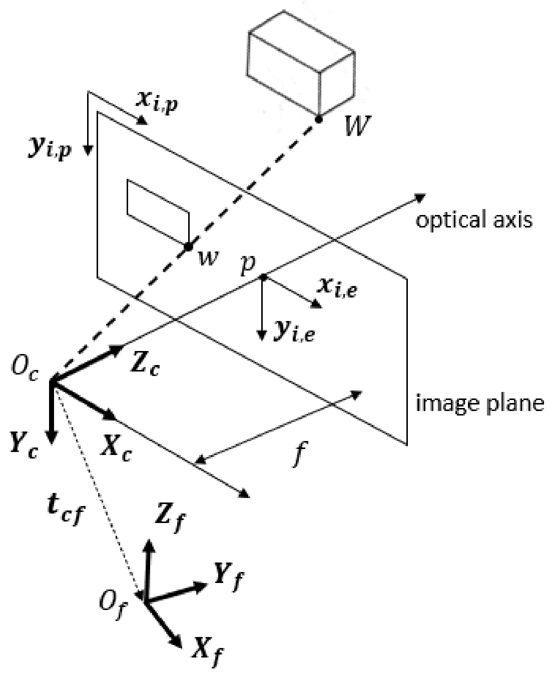


Figure 2.1: Pinhole Camera Model

ten as

$$Z_c \begin{bmatrix} x_{i,e} \\ y_{i,e} \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.3}$$

Since the image points in digital cameras are usually specified relative to the top-left corner of the image in pixel units, the coordinates of the image point w should be described by

$$\begin{bmatrix} x_{i,p} \\ y_{i,p} \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i,e} \\ y_{i,e} \\ 1 \end{bmatrix} \tag{2.4}$$

where $(x_{i,p}, y_{i,p}, 1)$ are the homogeneous coordinates of the point in pixels, (o_x, o_y) are the principal point coordinates in pixels, s_x and s_y are scale factors that count for the number of pixels per unit length in x and y direction, respectively, and s_θ is the skew parameter.

Finally, the overall relationship between the 3D world coordinates of the point, (X_f, Y_f, Z_f), and its corresponding image coordinates, $(x_{i,p}, y_{i,p})$, is given by (2.5). This one is obtained by replacing (2.1) in (2.3) and, subsequently its result in (2.4).

$$Z_c \begin{bmatrix} x_{i,p} \\ y_{i,p} \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & s_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R_{cf}} & \mathbf{t_{cf}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_f \\ Y_f \\ Z_f \\ 1 \end{bmatrix} = \mathbf{K_{int}}\,\mathbf{P}\,\mathbf{K_{ext}} \begin{bmatrix} X_f \\ Y_f \\ Z_f \\ 1 \end{bmatrix} \tag{2.5}$$

In (2.5) $\mathbf{K_{int}}$, represents the intrinsic camera matrix, $\mathbf{P}$ is a standard projection matrix and $\mathbf{K_{ext}}$ is named extrinsic camera matrix.

2.1.1 Intrinsic Camera Matrix Parameters

The intrinsic camera matrix parameters can be determined using the Camera Calibrator app from Matlab. The app requires images of a checkerboard with known square size captured by the camera being calibrated. To get better estimations, the images uploaded should cover different perspectives of the checkerboard with respect to the camera and the pattern should be all visible.

On the other hand, the focal length in world units, f, can be calculated through the camera perspective projection illustrated in Figure 2.2.

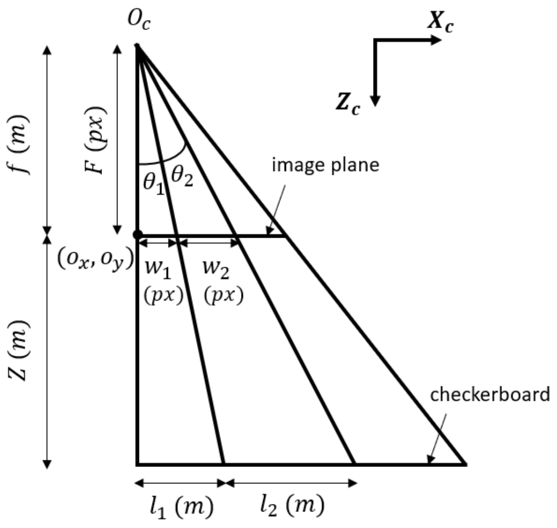


Figure 2.2: Perspective projection of the pinhole camera model

Considering that w_2 and l_2 are the square size of the checkerboard in pixel and metric units, respectively. There could be the case where the principal point is not aligned with the vertex of a square, as shown in Figure 2.2. Thus, the offset between the principal point and one of the square's vertex, represented in pixels by w_1 and in meters by l_1, should be taking into account when calculating f.

From similarity of triangles, the relationship between pixel and metric distances was established and given by (2.6). Through it, l_1 is obtained.

$$\begin{cases} \tan(\theta_1+\theta_2)(l_1+l_2) = \tan(\theta_1)l_1 \\ \tan(\theta_1+\theta_2)(w_1+w_2) = \tan(\theta_1)w_1 \end{cases} \Leftrightarrow l_1 = \frac{w_1 l_2}{w_2} \tag{2.6}$$

Once again, through trigonometric relations the focal length in world units, f, can be obtained by

$$\begin{cases} \tan(\theta_1) = \frac{w_1}{F} \\ \\ \tan(\theta_1) = \frac{l_1}{f+Z} \end{cases} \Leftrightarrow f = \frac{Fl_1}{w_1} - Z, \tag{2.7}$$

where F is the focal length in pixel units determined by the camera calibration process. This value depends on the axis of the camera frame considered. It is equal to fs_x in the X-axis direction and to fs_y in the Y-axis direction. The Z denotes the metric distance between the image and the checkerboard plane.

2.2 Visual Servo Control Schemes

According to [21], the goal of visual servo control schemes is to minimize the error

$$\mathbf{e}(t) = \mathbf{s}(t) - \mathbf{s}^*, \tag{2.8}$$

where $\mathbf{s}(t)$ is a vector of k visual features and $\mathbf{s}^*$ is the vector with the desired features values. Depending on the approach of the visual servo control, $\mathbf{s}$ has a different meaning, as it will be explained in Sections 2.2.1 and 2.2.2.

In order to establish a tracking control law, the relationship between the time variation of the visual features, $\dot{\mathbf{s}}$, and the camera velocity, $\mathbf{v_c} = (\mathbf{v}_{tc}, \mathbf{v}_{rc})$, which includes the linear, $\mathbf{v}_{tc} = (v_x, v_y, v_z)$, and angular, $\mathbf{v}_{rc} = (\omega_x, \omega_y, \omega_z)$, components, is defined by [21] as (2.9). Once again, the form of the interaction matrix, $\mathbf{L_e} \in \mathbb{R}^{k\times 6}$, is related to the visual servoing scheme adopted.

$$\dot{\mathbf{s}} = \mathbf{L_e}\mathbf{v_c} \tag{2.9}$$

Assuming the $\mathbf{s}^*$ values are constant, the time derivative of the error, $\dot{\mathbf{e}}$, is determined by (2.10), .

$$\dot{\mathbf{e}} = \mathbf{L_e}\mathbf{v_c} \tag{2.10}$$

To ensure the error converges to zero sufficiently fast, an exponential decrease of the error, described by (2.11), is applied, where λ is a scalar in s^{-1} dictating the error convergence rate. Therefore, the equations (2.10) and (2.11) combined gives the expression (2.12), which describes the relation between the velocity of the camera and the error.

$$\dot{\mathbf{e}} = -\lambda\mathbf{e} \tag{2.11}$$

$$\mathbf{L_e}\mathbf{v}_c = -\lambda\mathbf{e} \tag{2.12}$$

In the end, the control law of visual servoing is given by

$$\mathbf{v_c} = -\lambda\widehat{\mathbf{L}_\mathbf{e}^+}\mathbf{e}, \tag{2.13}$$

where $\mathbf{L}_\mathbf{e}^+ = (\mathbf{L_e}^T\mathbf{L_e})^{-1}\mathbf{L_e}^T$ is the Moore-Penrose pseudo-inverse of $\mathbf{L_e}$. But since the interaction matrix depends on parameters that are estimated, an approximation of $\mathbf{L}_\mathbf{e}^+$, $\widehat{\mathbf{L}_\mathbf{e}^+}$, is used instead in (2.13).

2.2.1 Image Based Control Law

According to [21], in IBVS $\mathbf{s}(t)$ is traditionally defined as the image coordinates of interest points, $(x_{i,e}, y_{i,e})$, which from now on will be mentioned as (x, y). Using (2.3) and (2.4), (x, y) is determined based on intrinsic camera matrix parameters as shown in (2.14).

$$\begin{cases} x = X_c/Z_c = (x_{i,p} - o_x)/fs_x \\ y = Y_c/Z_c = (y_{i,p} - o_y)/fs_y \end{cases} \tag{2.14}$$

To obtain the interaction matrix for IBVS, first the time derivative of $\mathbf{s}$ is taken:

$$\begin{cases} \dot{x} = \dot{X}_c/Z_c - X_c\dot{Z}_c/Z_c^2 = (\dot{X}_c - x\dot{Z}_c)/Z_c \\ \dot{y} = \dot{Y}_c/Z_c - Y_c\dot{Z}_c/Z_c^2 = (\dot{Y}_c - y\dot{Z}_c)/Z_c \end{cases} \tag{2.15}$$

Replacing the expressions for the spacial velocities $\dot{X}_c$, $\dot{Y}_c$ and $\dot{Z}_c$ given by (2.16) into (2.15), a relation between the time derivative of the image features $(\dot{x}, \dot{y})$ and the camera velocity components is established by (2.17).

$$(\dot{X}_c, \dot{Y}_c, \dot{Z}_c)^T = -\mathbf{v_{tc}} - \mathbf{v_{rc}} \times (X_c, Y_C, Z_c)^T \Leftrightarrow \begin{cases} \dot{X}_c = -v_x - \omega_y Z_c + \omega_z Y_c \\ \dot{Y}_c = -v_y - \omega_z X_c + \omega_x Z_c \\ \dot{Z}_c = -v_z - \omega_x Y_c + \omega_y X_c \end{cases} \tag{2.16}$$

$$\begin{cases} \dot{x} = -v_x/Z_c - xv_z/Z_c + xy\omega_x - (1 + x^2)\omega_y + y\omega_z) \\ \dot{y} = -v_y/Z_c - yv_z/Z_c + (1 + y^2)\omega_x - xy\omega_y - x\omega_z \end{cases} \tag{2.17}$$

Therefore, if (2.17) is rewritten in the form of (2.9), the following equation is obtained

$$(\dot{x}, \dot{y})^T = \mathbf{L_e}\mathbf{v}_c \tag{2.18}$$

where $\mathbf{L_e}$ is the interaction matrix given by:

$$\mathbf{L_e} = \begin{bmatrix} \frac{-1}{Z_c} & 0 & \frac{x}{Z_c} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z_c} & \frac{y}{Z_c} & (1+y^2) & -xy & -x \end{bmatrix} \tag{2.19}$$

In order to apply (2.13), $\widehat{\mathbf{L_e^+}}$ must be determined. In [21] there are three different approaches purposed for calculating $\widehat{\mathbf{L_e^+}}$.

The first one assumes that the depth of each feature, Z_c, is available in every time step and, thus, $\mathbf{L_e}$ is always known. Hence, the estimated pseudo-inverse is given by

$$\widehat{\mathbf{L_e^+}} = \mathbf{L_e^+}. \tag{2.20}$$

The second approach takes $\mathbf{L_e}$ as $\mathbf{L_{e^*}}$, which corresponds to (2.19) when $\mathbf{e} = 0$, and, so,

$$\widehat{\mathbf{L_e^+}} = \mathbf{L_{e^*}^+}. \tag{2.21}$$

Finally, the last one combines the previous approaches presented and takes the form of

$$\widehat{\mathbf{L}_{\mathbf{e}}^{+}} = \frac{1}{2}(\mathbf{L}_{\mathbf{e}} + \mathbf{L}_{\mathbf{e}^*})^{+}. \tag{2.22}$$

2.2.2 Position Based Control Law

In the case of PBVS, according to [21] the object pose with respect to the camera frame is used to define $\mathbf{s}$. This is usually represented as $(\mathbf{t}, \alpha\mathbf{u})$, where $\mathbf{t}$ is a translation vector and $\alpha\mathbf{u}$ is the angle/axis parameterization for the rotation.

Let $\mathbf{s}$ parameters be specified with respect to the camera frame mentioned in Section 2.1 and the fixed frame be attached to the object, then the translation vector will be $\mathbf{t}_{cf}$. Thus, (2.8) will be given by

$$\mathbf{e} = (\mathbf{t}_{cf} - \mathbf{t}^*_{cf}, \alpha\mathbf{u}), \tag{2.23}$$

where $\mathbf{t}^*_{cf}$ is the desired translation vector.

For this error definition the interaction matrix takes the form of

$$\mathbf{L}_{\mathbf{e}} = \begin{bmatrix} -\mathbf{I} & \mathbf{t}_{cf} \\ \mathbf{0} & \mathbf{L}_{\alpha\mathbf{u}} \end{bmatrix}, \;\; \mathbf{L}_{\alpha\mathbf{u}} = \mathbf{I} - \frac{\alpha}{2}\mathbf{u} + \left(1 - \frac{sinc(\alpha)}{sinc^2(\alpha/2)}\right)\mathbf{u}^2, \tag{2.24}$$

in which $\mathbf{I}$ is a 3x3 identity matrix and $sinc(x)$ is the sinus cardinal function.

Subsequently, the pseudo-inverse matrix, $\mathbf{L}_{\mathbf{e}}^{+}$, which characterizes the control law of the PBVS scheme will be given by

$$\widehat{\mathbf{L}_{\mathbf{e}}^{+}} = \begin{bmatrix} -\mathbf{I} & \mathbf{t}_{cf}\,\mathbf{L}_{\alpha\mathbf{u}}{}^{-1} \\ \mathbf{0} & \mathbf{L}_{\alpha\mathbf{u}}{}^{-1} \end{bmatrix}. \tag{2.25}$$

So, for PBVS, the (2.13) can be simplified to

$$\begin{cases} v_c = -\lambda((\mathbf{t}_{cf} - \mathbf{t}^*_{cf}) + (\mathbf{t}^*_{cf})^T\alpha\mathbf{u}) \\ \omega_c = -\lambda\alpha\mathbf{u} \end{cases}, \tag{2.26}$$

since $\mathbf{L}_{\alpha\mathbf{u}}^{-1}\alpha\mathbf{u} = \alpha\mathbf{u}$.

Chapter 3

Quadrotor Model and Control

3.1 Parrot AR Drone 2.0

The Parrot AR Drone 2.0 (Parrot) is a quadrotor, i.e. it has four rotors, in a cross configuration, as shown in Figure 3.1. The black hull is attached in order to avoid collision and damage when flying indoor.

Figure 3.1: Parrot AR Drone 2.0 in www.drones.nl/drones/parrot-ar-drone-2-0-power-edition

Although the quadrotor has 6 degree of freedom (DOF), three translational and three rotational, its motion is controlled by the rotational speed of the four rotors coupled in pairs. As seen in Figure 3.2, the angular velocity of rotors 1 and 3, w_1 and w_3, and rotors 2 and 4, w_2 and w_4, actuates in opposite direction to guarantee that the drone does not rotate around itself. The rotational speed of the motors generates a force pointing upwards and a momentum acting with opposite direction from the propellers to make the quadrotor lift.

By varying the rotational speed of each rotor, four basic movements allow the quadrotor to reach a desired attitude and height: throttle, roll, pitch and yaw.

From the sensory equipment onboard of the quadrotor is possible to control the maneuvers described before. The altitude of the drone is measured by the ultrasonic altimeter. The inertial measurement unit uses the accelerometer, gyroscope and magnetometer to determine quadrotor's attitude angles and their rate of change. Moreover, the images captured from the bottom facing camera are used to estimate the horizontal velocity of the quadrotor. According to [22], the linear velocities are determined through two

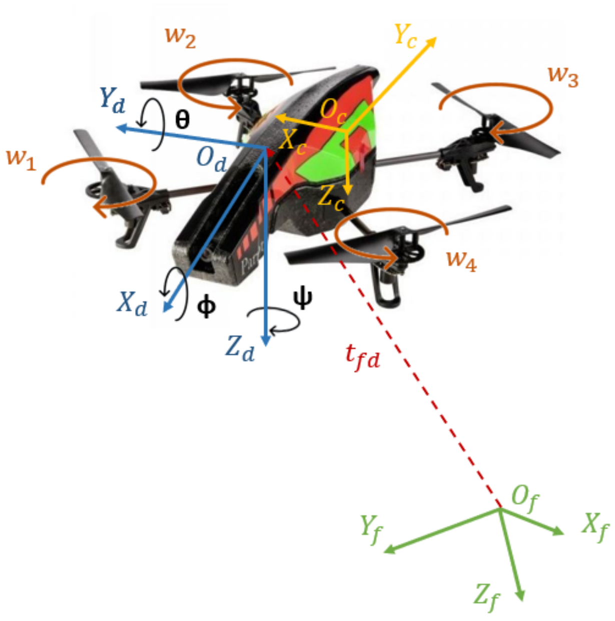


Figure 3.2: Referential frames representation: camera frame, drone frame and fixed frame

complementary algorithms: one computes optical flow over the whole image range and the other uses image features displacement over time.

All these state variables are measured relatively to the body-fixed frame of the quadrotor represented in Figure 3.2 by the unit vectors ($\mathbf{X_d}$,$\mathbf{Y_d}$,$\mathbf{Z_d}$), whose origin, O_d, is the drone's center of mass. However, if the drone's position is to be controlled, it must be defined in a frame that does not depend on drone motion and, so, a fixed frame $\{O_f, \mathbf{X_f}, \mathbf{Y_f}, \mathbf{Z_f}\}$ is used. Its origin's position will be defined on Section 4.1.

Accordingly with Figure 3.2, the translation from the drone frame to the fixed frame is represented by the vector $\mathbf{t_{fd}}$. The relative orientation between those frames is given by the rotation matrix

$$\mathbf{R_{fd}} = \mathbf{R}_z(\psi) \cdot \mathbf{R}_y(\theta) \cdot \mathbf{R}_x(\phi) =$$

$$= \begin{bmatrix} \cos(\theta)\cos(\psi) & \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) \\ \cos(\theta)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}, \quad (3.1)$$

where ϕ, θ and ψ are the Euler angles that describe the rotation around X, Y and Z-axis of drone frame, respectively.

3.2 Quadrotor Model

For the purpose of Parrot dynamics simulation and real-time control, the development kit models from [23] were used. Both developed in Simulink, the simulator model blocks were derived via system identification, whereas the real-time model blocks can send commands to and read states from the drone via

Wi-Fi connection.

Indeed, the simulator model is an approximation of the real model have the same control inputs and state variables available, thus the simulator model can be used for designing the controllers. The control inputs of the system are: roll and pitch angles, yaw rate and vertical velocity. The state variables available, that in the case of real-time are estimated through the sensors described in Section 3.1, are roll, pitch and yaw angles, altitude and longitudinal, lateral and vertical velocities.

3.2.1 Simulator Model

The quadrotor dynamics of the simulator model illustrated in Figure 3.3 is defined by six linear time-invariant state-space equations organized into four subsystems: lateral movement, longitudinal movement, heading and vertical movement. The first one addresses the roll and lateral velocity dynamics, represented by the 'ssRoll' and 'ssRoll2V' blocks, respectively. The 'ssPitch' and 'ssPitch2U' compose the second subsystem and describe the pitch and the longitudinal velocity dynamics, respectively. Finally, the third one includes the yaw dynamics represented by the 'ssYaw' block and the fourth one comprises the altitude dynamics depicted by the 'ssAltitude' block.

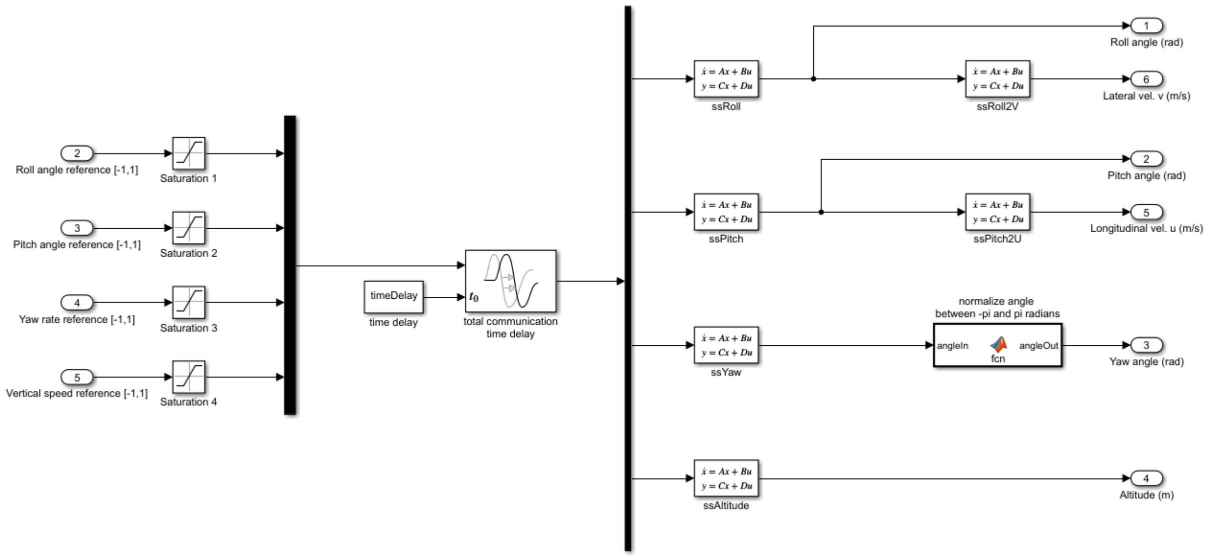


Figure 3.3: Simulator simulink model of quadrotor dynamics, named by [23] as ARDrone Simulator Block

Additionally, each subsystem in Figure 3.3 has a saturation block that limits the control references values to ensure the validity of the linear equations that describe quadrotor dynamics.

To execute the control law, the same sample time of the real-time model is applied, Ts = 0.065s, to better model the real system. Furthermore, a time delay of 0.26s (timeDelay = 4*Ts), represented in Figure 3.3 by the 'time delay' constant block, was added to the reference input values of quadrotor system to consider the effect of the communication delay between the host computer and the drone.

Lateral Movement

In (3.2) the lateral velocity dynamics is represented by tfRoll and tfRoll2V transfer functions.

$$tfV = tfRoll \times tfRoll2V = \frac{1.483s + 3.524}{s^2 + 4.268s + 12.69} \times \frac{4.774}{s + 0.4596} = \frac{7.081s + 16.82}{s^3 + 4.728s^2 + 14.65s + 5.83}. \tag{3.2}$$

The tfRoll transfer function, which describes the internal dynamics of roll movement as a second order system, is assumed to has embedded feedback control, because there is no controllable input variable available, and, as it can be seen in Figure 3.4, is already stabilized.

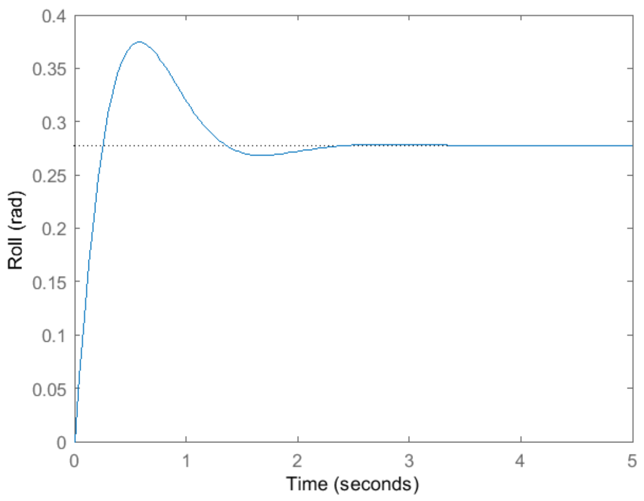


Figure 3.4: Step response of tfRoll transfer function

The tfRoll2V transfer function is an approximation of the kinematic relation between the roll angle, ϕ, and the lateral velocity, v. It is defined by (3.3) on time domain, where g stands for the acceleration of gravity. The equivalent Laplace representation of (3.3) is given by (3.4) and is a stable first-order system, like tfRoll2V. Since the real pole location of both transfer functions are close to each other (the pole from tfRoll2V is at -0.4596 and (3.4) pole is at zero), they have similar transient responses for small variations of roll.

$$\dot{v} = g * \phi \tag{3.3}$$

$$\frac{V(s)}{\Phi(s)} = \frac{g}{s} \tag{3.4}$$

Longitudinal Movement

The longitudinal dynamics of the quadrotor is mathematically expressed by tfPitch and tfPitch2U transfer functions in

$$tfU = tfPitch \times tfPitch2U = \frac{2.514s + 4.866}{s^2 + 3.978s + 11.92} \times \frac{-6.154}{s + 0.665} = \frac{-15.47s - 29.95}{s^3 + 4.643s^2 + 14.56s + 7,926}. \tag{3.5}$$

On one hand, the tfPitch characterizes the internal dynamics of pitch movement. Similarly to tfRoll transfer function, tfPitch is a second order system with embedded feedback control and, as shown in

Figure 3.5, is stable.

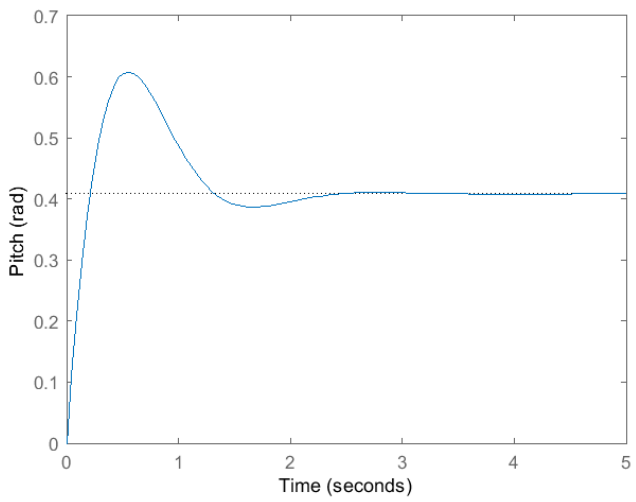


Figure 3.5: Step response of tfPitch transfer function

On the other hand, the tfPitch2U transfer function is an approximation of the kinematic relation expressed in time domain by (3.6), where θ is the pitch angle and u is the longitudinal velocity. Both tfPitch2U and (3.7), which is the Laplace representation of (3.6), are stable first-order systems and its real poles location in the s-plane are close to each other (the pole from tfPitch2U is at -0.665 and (3.7) pole is at zero), which means the transient responses are similar for small variations of pitch.

$$\dot{u} = -g * \theta \tag{3.6}$$

$$\frac{U(s)}{\Theta(s)} = -\frac{g}{s} \tag{3.7}$$

Heading

The dynamics of heading subsystem is described by the transfer function tfYaw, whose expression is given by

$$tfYaw = \frac{1.265}{s + 0.0059}. \tag{3.8}$$

This transfer function is consider an approximation of the kinematics expression (3.9) that relates the yaw angle, ψ, and the yaw rate, r, and whose Laplace representation is given by (3.10), since it is a first-order system and the transient responses are similar for small variations.

$$\dot{\psi} = r \tag{3.9}$$

$$\frac{\Psi(s)}{R(s)} = \frac{1}{s} \tag{3.10}$$

Vertical Movement

The vertical movement is characterized by the transfer function tfAltitude defined by

$$tf Altitude = \frac{0.1526s + 5.153}{s^2 + 5.82s}. \tag{3.11}$$

3.2.2 Real-time Model

The ARDrone Wi-Fi Block, illustrated in Figure 3.6, replaces the ARDrone Simulator Block in real-time experiments.

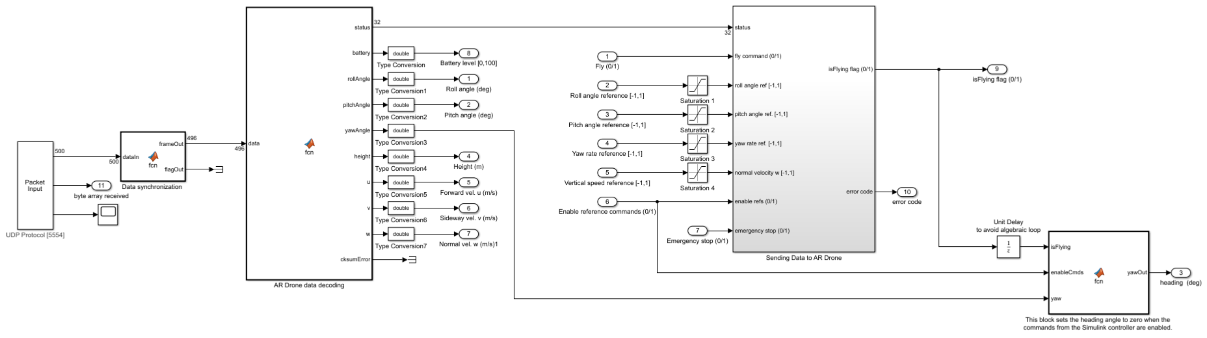


Figure 3.6: Real-time simulink model of quadrotor dynamics, named by [23] as ARDrone Wi-Fi Block

As stated before, this Simulink block receives the values of state variables from the drone sensors and sends AT-commands to control and configure the drone. Both information is transmitted through an UDP protocol defined inside Packet Input/Output Simulink blocks. The Packet Input block illustrated in Figure 3.6 receives information from the drone through the UDP port 5554, while the Packet Output block defined inside the *Sending Data to AR Drone* subsystem uses the UDP port 5556 to send information to the drone.

Furthermore, before the input control references commands are send to the drone, they are saturated to guarantee a linear behaviour of the system.

In this scenario, the sample time is Ts = 0.065s, to guarantee that the commands and states are updated every time step. This is true assuming that this value is higher than the communication delay due to Wi-Fi connection.

3.3 Quadrotor Control

The main goal of this project is to apply the classic visual servo control approaches, IBVS and PBVS, to the drone, so it can track a target, which can be a generic point in space. Thus, the position and heading of the drone must be controlled.

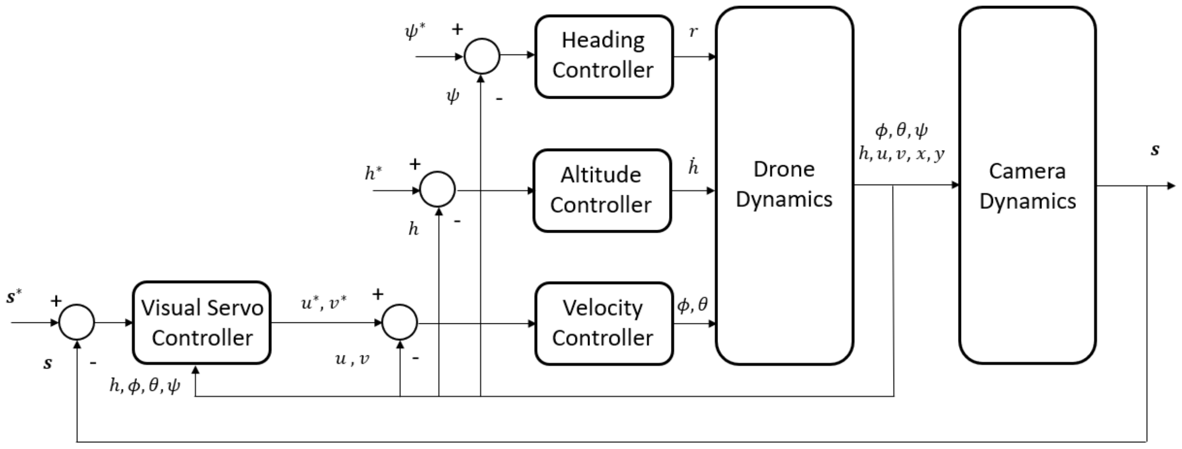


Figure 3.7: Quadrotor control architecture

The desired altitude and heading for the quadrotor are set as references to the vertical movement and heading subsystems in a closed loop. A cascade control strategy is proposed to control the drone's horizontal position through visual servoing. In this way, the outer loop of the cascade follows the visual control law, which gives the reference inputs (u^* and v^*) to the velocity closed loop systems (the inner loop of the cascade). To get the desired system response to position and heading inputs, classical Proportional-Integrative-Derivative (PID) controllers (3.12) are applied to the tracking errors, $e(t)$. In (3.12) the P is the proportional gain, D is the derivative gain and I is the integrative gain.

$$Pe(t) + D\frac{\partial e(t)}{\partial t} + I\int_0^t e(t)dt, \tag{3.12}$$

In order to obtain the PID controller gains, first the system transfer functions are computed taking in consideration the time delay mentioned in Section 3.2.1. The time delay is modelled as a first order Padé approximation (3.13), where θ is the time delay in seconds. Then, the transfer functions are discretized using the zero order hold method, which assumes the control inputs are piece wise constant over the sample time Ts=0.065s.

$$e^{-\theta s} \approx \frac{1 - \frac{\theta}{2}s}{1 + \frac{\theta}{2}s} \tag{3.13}$$

Furthermore, after the desired specifications of the system response are defined in terms of overshoot (M_p), settling time, t_s, (2% criteria) and steady state error (e_{ss}), the desired system dominant poles location (s_d) in the s-plane is determined. Knowing that, the different PID controller architectures are tested and the respective gains are set so the root locus fits the desired dominant poles location and, thus, respects the magnitude and angle conditions:

$$|KG_c(s_d)G_p(s_d)| = 1 \tag{3.14}$$

$$\arg[KG_c(s_d)G_p(s_d)] = -180°, \tag{3.15}$$

where K is the DC gain of the system, G_c is the controller transfer function and G_p is the transfer function of the system to be controlled.

3.3.1 Altitude and Heading Control

The altitude and heading PID controllers are developed for the linear subsystems introduced in Section 3.2.1 which are the vertical movement and heading, respectively.

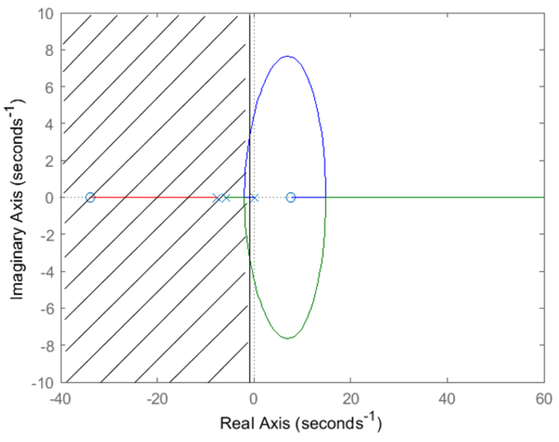
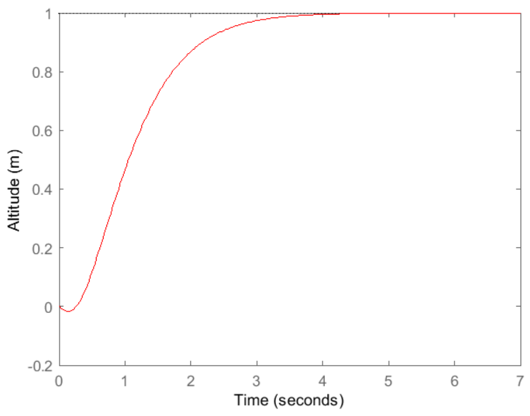


Figure 3.8: Root locus of vertical movement subsystem

In what concerns the vertical movement, it is a type 1 system and thus the steady state error is zero for a step input reference. A fast transient response of the system is desired so the drone lifts 1m in less than 5s. Hence, the dominant poles of the system should be located on the crossed region of the root locus illustrated in Figure 3.8. Since a low overshoot is preferred, the poles could be placed at the convergence point of blue and green lines inside the crossed region, where the overshoot is zero.



a)

	Specs Required	Specs Obtained
$M_p(\%)$	0	0
$t_s(s)$	<5	3.15
$e_{ss}(\%)$	0	0

b)

Figure 3.9: Step response of vertical movement closed loop system a) and the respective response characteristics b)

Having said that, with only a proportional gain the dominant poles are placed at the convergence point. The gain corresponds to P=0.93 and gives the step response characteristics desired, as shown in Figure 3.9.

In the case of the heading subsystem, it is a type zero system, which means there is a constant steady state error of the response with respect to a step input. However, this is not considered a problem since the yaw angle is kept fixed during the simulations to avoid the target being outside the field of view. Thus, a maximum steady state of 20$\%$ is established for the step input response.

Moreover, a fast and low overshoot transient response is desired so the image plane position remains fixed as fast as possible in a smooth way. Therefore, once again it is convenient to locate the dominant poles at the convergence point of the root locus of the system depicted in Figure 3.10, where the settling time is below 5s (crossed region) and the overshoot equals zero.

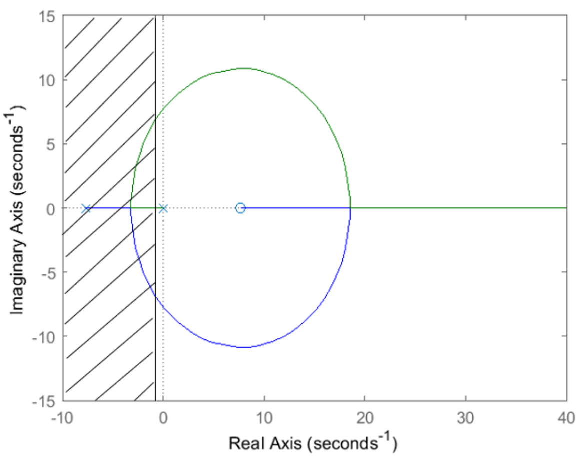
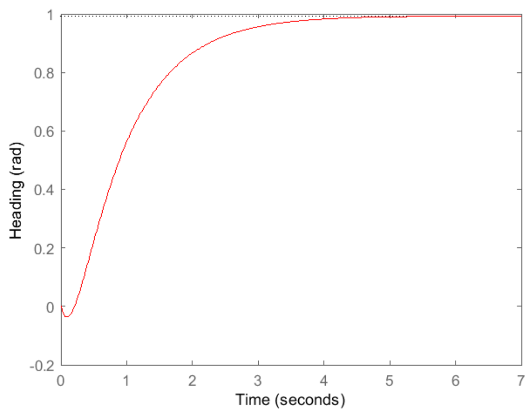


Figure 3.10: Root locus of heading subsystem

To place the poles at the desired location, a change of the system's gain is required. So, a proportional controller with P = 0.7 is applied to the system to give the desired step response illustrated in Figure 3.11.



a)

	Specs Required	Specs Obtained
$M_p(\%)$	0	0
$t_s(s)$	<5	3.48
$e_{ss}(\%)$	<20	0.81

b)

Figure 3.11: Step response of heading closed loop system a) and the respective response characteristics b)

3.3.2 Horizontal Velocity Control

Similarly to altitude and heading, the PID controllers of the cascade control's inner loop system were designed for the linear subsystems presented before on Section 3.2.1, named as longitudinal and lateral movement.

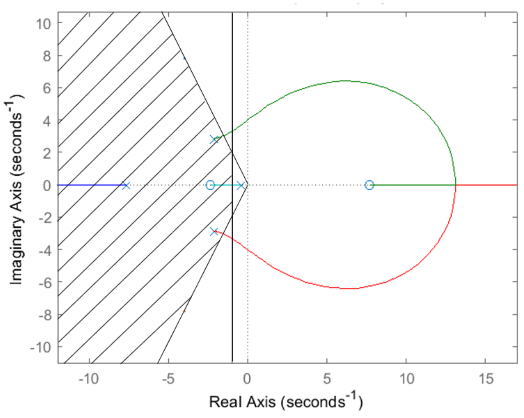


Figure 3.12: Root locus of longitudinal movement subsystem

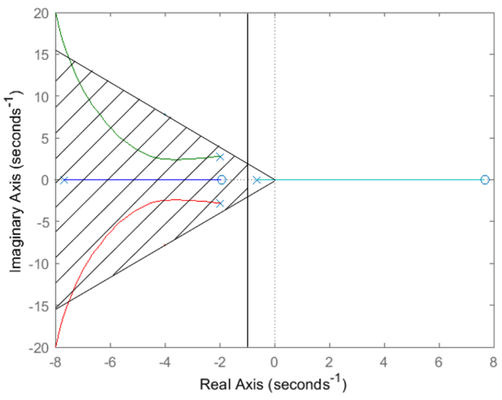
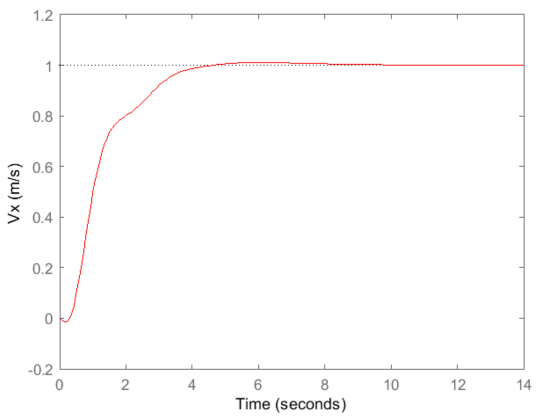


Figure 3.13: Root locus of lateral movement subsystem

Generally, the inner loop system responses should be 3/4 times faster than the outer loop ones and, so, there is high priority on having a low settling time, but low priority on having a minimum overshoot. Hence, the maximum value established for the settling time and overshoot percentage were 4s and 5$\%$, respectively, which corresponds to the crossed region of the systems root locus shown in Figures 3.12 and 3.13. Plus, a zero steady state error is desired for a step input, which is achieved with an integrator, since the system is type zero.



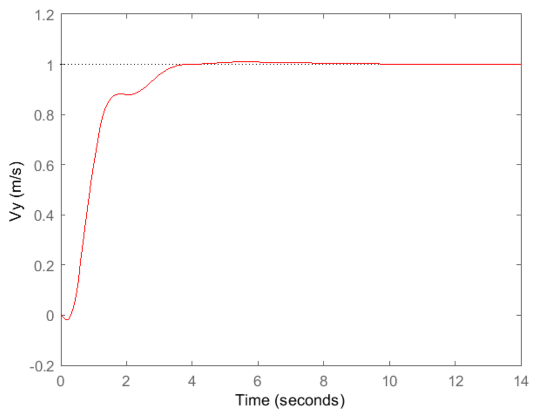
a)

	Specs Required	Specs Obtained
$M_p(\%)$	<5	1.1
$t_s(s)$	<4	3.79
$e_{ss}(\%)$	0	0

b)

Figure 3.14: Step response of longitudinal movement closed loop system a) and the respective response characteristics b)

This time a PI controller fits the required velocity systems' response characteristics to a step input, as shown in Figures 3.14 and 3.15. The gains of the controller are P = -0.222 and I = -0.204 , for the longitudinal subsystem and P = 0.61 and I = 0.327, for the lateral one.



a)

	Specs Required	Specs Obtained
$M_p(\%)$	<5	0.98
$t_s(s)$	<4	3.23
$e_{ss}(\%)$	0	0

b)

Figure 3.15: Step response of lateral movement closed loop system a) and the respective response characteristics b)

3.3.3 Visual Servo Control

In order to design the PID controllers to control the relative position of the drone's camera, first the linear equations that describes the relation between the cameras position and the target's relative horizontal position, both in the image space and Cartesian space must be obtained.

Having in mind that the velocity of the drone is the same of the camera since it is fixed to it, the camera's position can be determined as the integral of the velocity, after the drone's frame is transformed into the fixed frame. Due to the integral term the open loop systems of the outer loop are type 1.

In what concerns the relative position response, either defined on the image plane or on the Cartesian one, the steady state error must be close to zero so the drone reaches target. Also the position response should be approximately 3 times slower than the velocity one, as explained before, so the maximum settling time is set to 10s and the overshot should be the minimum to prevent the target being outside the field of view.

IBVS approach

Starting with the IBVS control approach, the measured output of the outer loop are the image coordinates of the target. Thus, the relation between it and the camera's position could be described by (2.5), which is highly nonlinear. To make it linear, this equation is approximated by a first order Taylor polynomial function:

$$f'(\delta x) = f(x_0) + \frac{\partial f}{\partial x}\Big|_{x_0} \delta x, \tag{3.16}$$

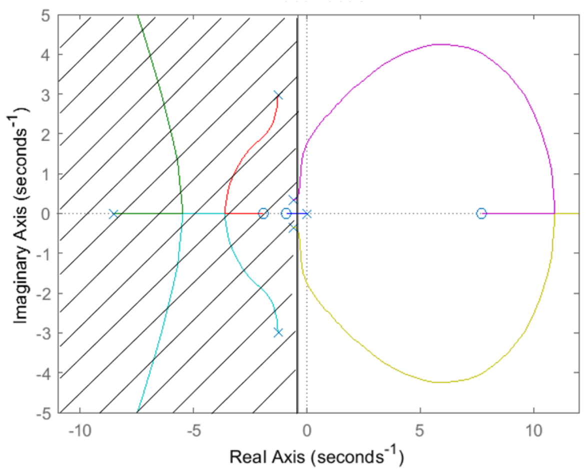
where $f(\mathbf{x}) = f(\theta, \phi, \psi, t_{cf_x}, t_{cf_y}, t_{cf_z})$ is the function that represents (2.5), $x_0 = (0, 0, \bar{\psi}, 0, 0, \bar{z})$ is the nominal point and $\delta x = x - x_0$ is the deviation from the nominal point.

The remaining variables from (2.5) are fixed parameters, since the intrinsic camera matrix parameters mentioned in Section 4.1.2 are specific for each camera characteristics and the target is considered static and coincident with the fixed frame, so $(X_f, Y_f, Z_f) = (0, 0, 0)$.

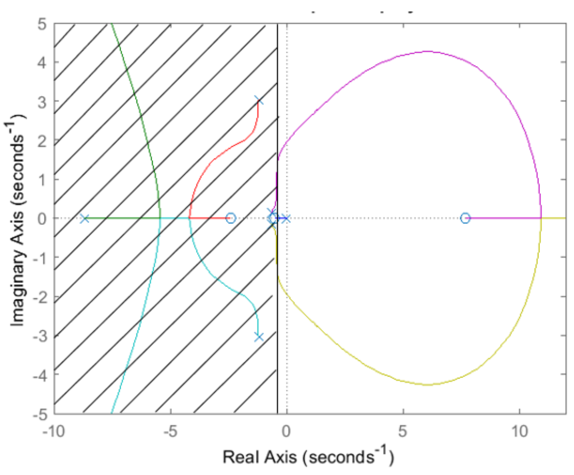
Taking all these assumptions in consideration, plus the small angle approximation for the trigonometric functions, the linear system which describes the image coordinates of the target is found to be

$$\begin{bmatrix} x_{i,p} \\ y_{i,p} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 147.66 & 0 & 0 \\ 0 & 0 & 0 & 0 & 147.77 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \phi \\ \psi \\ t_{cf_x} \\ t_{cf_y} \\ t_{cf_z} \end{bmatrix}. \tag{3.17}$$

Consequently, the open loop system of the outer loop is obtained and the PID controller can be designed. Having in mind the requirements for the systems response stated previously, the available area for the dominant poles location was identified by a cross region in the system's root locus depicted in Figure 3.16.



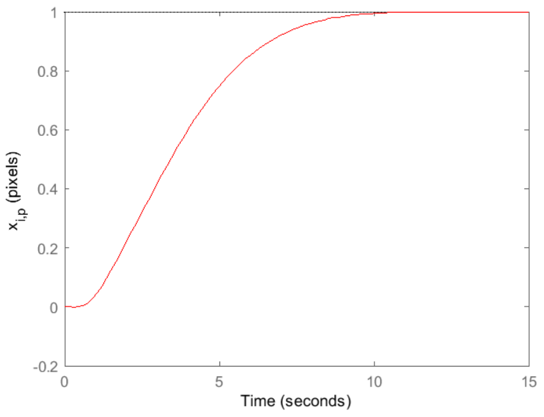
a) $x_{i,p}$ coordinate



b) $y_{i,p}$ coordinate

Figure 3.16: Root locus of target image coordinates linear system

As seen in Figures 3.17 and 3.18, the x and y coordinates of the image plane could respect the response specifications of a step input, with only a proportional gain of -0.00180 and -0.00197, respectively. These values replace λ in the visual servo control law.

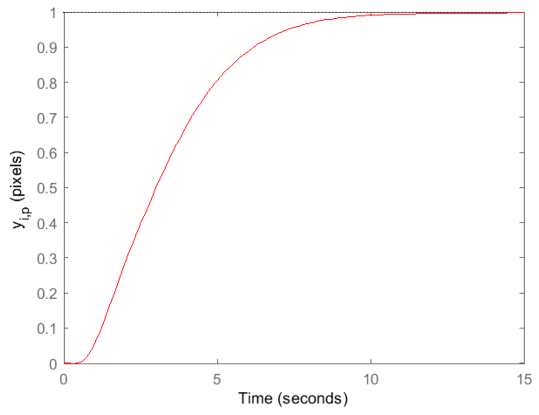


a)

	Specs Required	Specs Obtained
$M_p(\%)$	0	0
$t_s(s)$	10	8.74
$e_{ss}(\%)$	0	0

b)

Figure 3.17: Step response of target image coordinate $x_{i,p}$ closed loop system a) and the correspondent response characteristics b)



a)

	Specs Required	Specs Obtained
$M_p(\%)$	0	0
$t_s(s)$	10	8.72
$e_{ss}(\%)$	0	0

b)

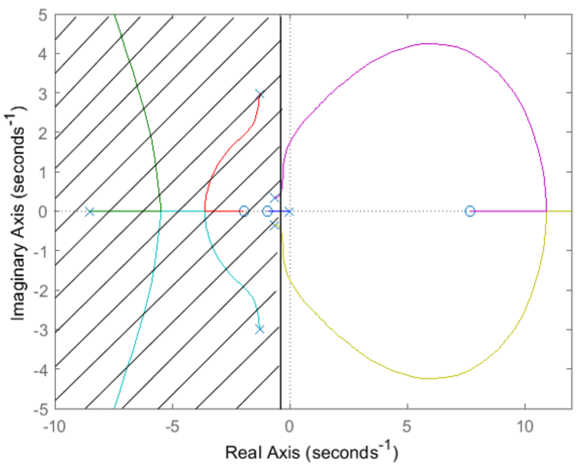
Figure 3.18: Step response of target image coordinate $y_{i,p}$ closed loop system a) and the correspondent response characteristics b)

PBVS approach

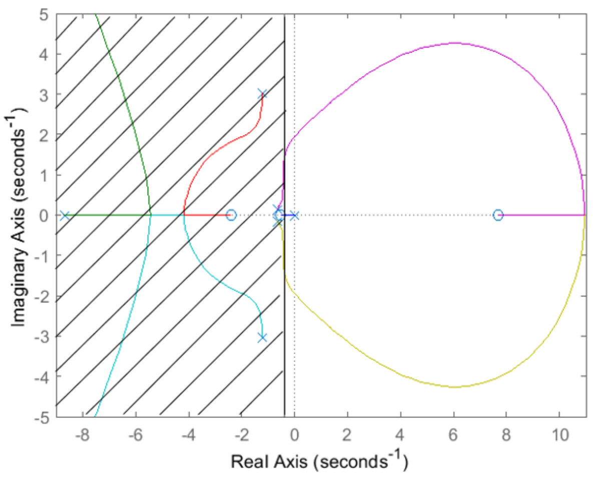
On the other hand, the measured output for the outer loop in PBVS control approach is the position of the target with respect the camera frame, which corresponds to $\mathbf{t_{cf}}$. Considering that the fixed frame origin is coincident with the object and this one is static, the camera's position in the fixed frame and the position of the target in the camera frame varies by the same amount. Plus, if the fixed frame axis direction is opposite to the camera frame axis, the linear relation between those variables is described as

$$\begin{bmatrix} t_{cf_x} \\ t_{cf_y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \phi \\ \psi \\ t_{fc_x} \\ t_{fc_y} \end{bmatrix} \tag{3.18}$$

Having 3.18, the open loop system of the outer loop is obtained and it is represented on the s-plane by Figure 3.19. In this figure the desired area for the system's dominant poles location is illustrated by a crossed region.



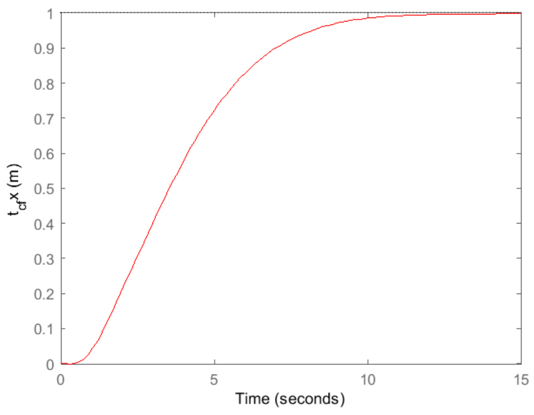
a) t_{cf_x} coordinate



b) t_{cf_y} coordinate

Figure 3.19: Root locus of target 3D coordinates linear system

Finally, the required specifications for the open loop system described to a step input were achieved by applying a proportional gain for both x and y directions of $\mathbf{t_{cf}}$, as shown in Figures 3.20 and 3.21, respectively. The values of the gains are 0.25331 and 0.27856 and they replace λ in the visual servo control law.

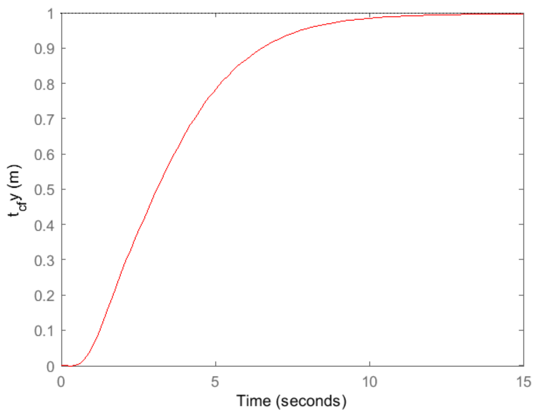


a)

	Specs Required	Specs Obtained
$M_p(\%)$	0	0
$t_s(s)$	<10	9.57
$e_{ss}(\%)$	0	0

b)

Figure 3.20: Step response of target 3D coordinate t_{cf_x} closed loop system a) and the correspondent response characteristics b)



a)

	Specs Required	Specs Obtained
$M_p(\%)$	0	0
$t_s(s)$	<10	9.47
$e_{ss}(\%)$	0	0

b)

Figure 3.21: Step response of target 3D coordinate t_{cf_y} closed loop system a) and the correspondent response characteristics b)

Chapter 4

Setup and Implementation

4.1 Problem Formulation

The image data is acquired by the bottom facing camera of the Parrot, which is fixed to it. This makes the visual servo control a eye-in-hand configuration problem.

So, there is a constant relationship between the pose of the drone and the pose of the camera. This can be described by the transformation matrix that relates the drone frame coordinates and the camera frame coordinates given by

$$\begin{bmatrix} X_c & Y_c & Z_c & 1 \end{bmatrix}^T = \begin{bmatrix} \mathbf{R}_{cd} & \mathbf{t}_{cd} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_d & Y_d & Z_d & 1 \end{bmatrix}^T, \tag{4.1}$$

where

$$\mathbf{R}_{cd} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{t}_{cd} = \begin{bmatrix} 0 & -0.067 & -0.0435 + f \end{bmatrix}. \tag{4.2}$$

The rotation matrix $\mathbf{R}_{cd}$ can be easily determined by looking at the drone and camera frames orientation in Figure 3.2. On the other hand, the translation vector $\mathbf{t}_{cd}$ components were evaluated on SolidWorks, where the distance between the quadrotor center of mass and the center of camera lens was measured.

In order to test the application of the visual servo control algorithms presented before on the Parrot, a simple approach was studied. There is just one image feature to be tracked, which is the centroid of a rectangle. From now on the rectangular object is referred as target.

Moreover, the goal pose of the quadrotor is fixed and it is defined as the one where the target is placed at the center of the image. Plus, during the trajectory the altitude and the heading of the Parrot are kept constant.

In this way, the depth of the target relative to the camera frame, Z_c, will be always known if the ground where the target is placed corresponds to $Z_f = 0$ and the ultrasonic altimeter measurements from the drone are available.

Let is assume the fixed frame has the origin coincident with the centroid, $X_f = 0$ and $Y_f = 0$. Then, knowing the intrinsic camera matrix parameters (Section 2.1.1) and giving the image coordinates of the centroid in pixels $(x_{i,p}, y_{i,p})$ and the correspondent $\mathbf{R_{cf}} = \mathbf{R_{cd}}\mathbf{R_{fd}}^T$, (2.5) can be solved in order to determine $\mathbf{t}_{cf}$. Therefore, this translation vector, which represents the coordinates of the fixed frame origin in the camera frame, can be used to describe $\mathrm{s}(t)$ in PBVS control law.

Finally, since only the horizontal linear velocity is to be controlled, the PBVS control law is given by

$$\begin{cases} v_x = -\lambda(t_{cf_x} - t^*_{cf_x}) \\ v_y = -\lambda(t_{cf_y} - t^*_{cf_y}) \end{cases}, \tag{4.3}$$

where the pseudo-inverse matrix is 2x2 and takes the form of

$$\widehat{\mathbf{L_e^+}} = \begin{bmatrix} -\mathbf{I} \end{bmatrix}. \tag{4.4}$$

On the other hand, the $\mathbf{L_e}$ for the IBVS is represented by (4.6), which gives the control law in (4.5). In this case, the coordinates of the target's centroid, $x_{i,p}$ and $y_{i,p}$, constitute $\mathrm{s}(t)$. Since the altitude of the drone remains constant, Z_c is also constant and, thus, the three alternatives proposed for $\widehat{\mathbf{L_e^+}}$ in Section 2.2.1 are practically the same. But as Z_c is available every time step, the (2.20) is used.

$$\begin{cases} v_x = \frac{\lambda(x_{i,p} - x^*_{i,p})}{Z_c} \\ v_y = \frac{\lambda(y_{i,p} - y^*_{i,p})}{Z_c} \end{cases} \tag{4.5}$$

$$\mathbf{L_e} = \begin{bmatrix} \frac{-1}{Z_c} & 0 \\ 0 & \frac{-1}{Z_c} \end{bmatrix} \tag{4.6}$$

4.1.1 Simulink model

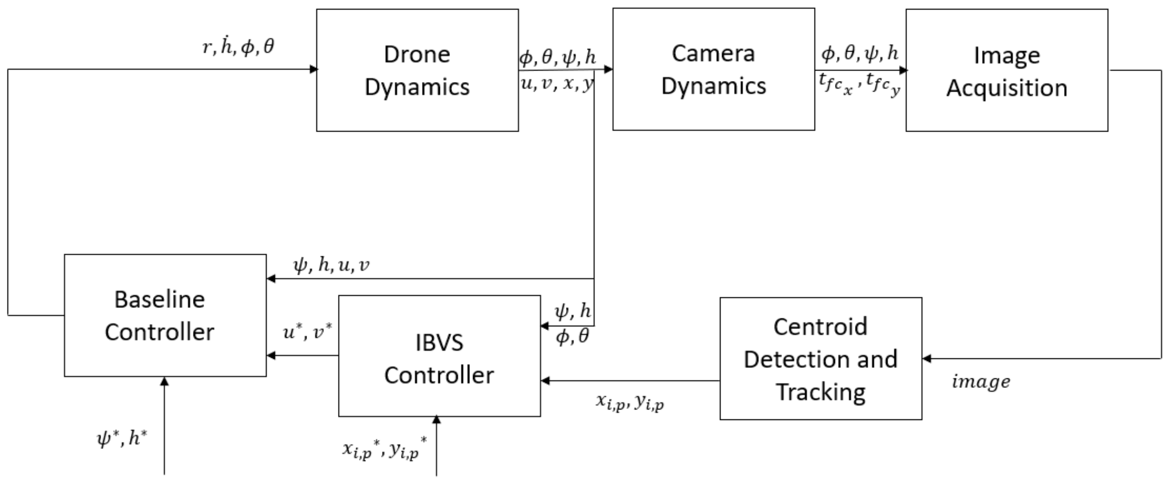


Figure 4.1: Block diagram of quadrotor control using IBVS

The IBVS and PBVS are implemented in Simulink according to the schemes represented in Figures 4.1 and 4.2, respectively, both in simulator and real environment.

These schemes are a detailed version of the control arquitecture illustrated in Figure 3.7, since they specify how the $\mathbf{s}$ is obtained for each visual servo control approach. In the case of IBVS, after the target's image is captured by the Image Acquisition module, the image is processed inside the Centroid Detection and Tracking module so the target's centroid location in the image plane is determined and given as input to the IBVS Controller. While in case of PBVS, besides going through the same steps described for the IBVS, the location of the target centroid in the image plane is converted into a position in space by the Target Position Estimation module, so the PBVS Controller can take it as input. This module implements the calculations explained before to determine $\mathbf{t_{cf}}$ on both simulation and real environment.

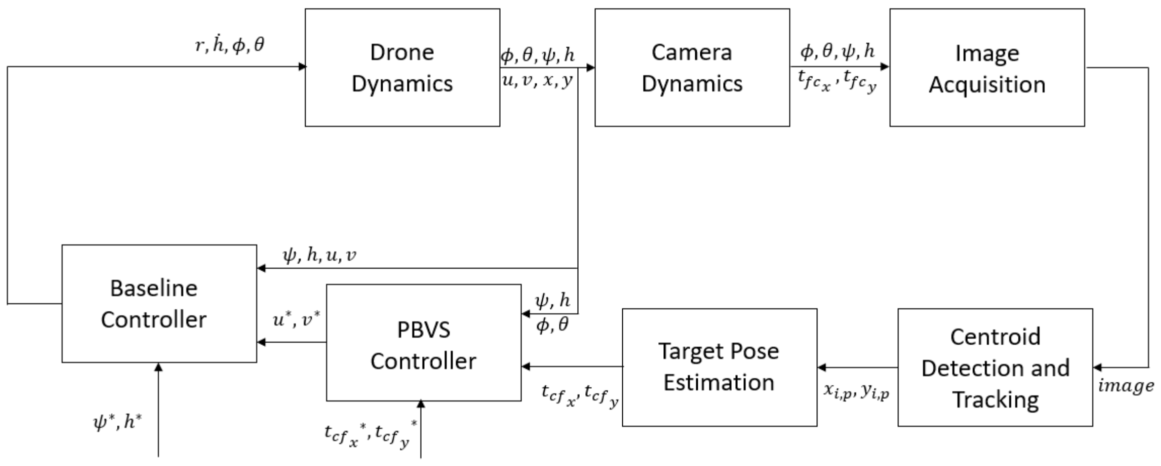


Figure 4.2: Block diagram of quadrotor control using PBVS

Even though the Drone and Camera Dynamics are the same since the bodies are attached to each other, they are represented on these schemes separately in order to resemble the frames defined in Section 3.1.

In Sections 4.2 and 4.3 the implementation of the modules introduced here are explained in detail for both simulator and experimental environment.

4.1.2 Camera Calibration

In order to follow the calibration process described in Section 2.1.1 and get the intrinsic camera parameters required for $\mathbf{t_{cf}}$ calculation, different checkerboard formats were used on the working environments. A printable version of a checkerboard like shown in Figure 4.3 was used in case of the real environment, while in the simulator environment a surface with a checkered pattern was created inside a virtual world, as shown in Figure 4.4.

Both the images have the same size, which is the one from the bottom facing camera of the drone (360x640). However, the intrinsic camera matrix for the real (4.7) and virtual (4.8) environments has

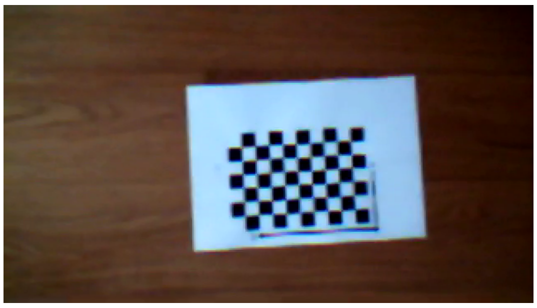


Figure 4.3: Checkerboard in Real World

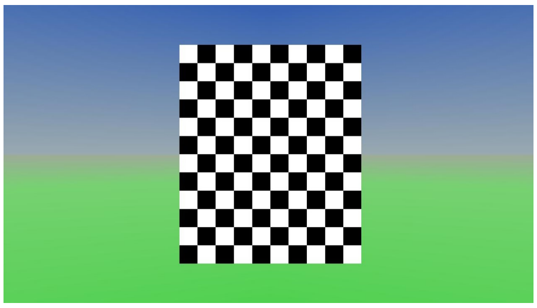


Figure 4.4: Checkerboard in Virtual World

some discrepancies in the fs_x and fs_y values. This is essentially explained by the fact that it is not possible to include the camera's diagonal field of view (64°) in the software where the virtual world was created. Plus, there are no options to introduce the pixel's size and lens specifications, like curvature and thickness, which directly influences the field of view and the focal length.

$$K_{int} = \begin{bmatrix} 823.12 & 0 & 320.57 \\ 0 & 821.31 & 180.89 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

$$K_{int} = \begin{bmatrix} 287.95 & 0 & 320 \\ 0 & 288.02 & 180 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

It is important to mention that the center of projection of the camera model, denoted in Figure 2.1 as O_c, neither correspond to the lenses plane in the real environment nor to the viewpoint defined in the virtual environment. The center of projection, which corresponds to the origin of the camera frame, is defined in the fixed frame as a point that distance f in z-direction from the ones previously mention.

4.2 Simulator environment

In the simulator environment the Drone Dynamics module consists on the transfer functions referred in Section 3.2.1.

The implementation of the Image Acquisition and Centroid Detection and Tracking modules are detailed in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 Virtual Environment and Image Acquisition

In order to test the visual servo control in the simulator model, the drone, the camera and the target were created in the virtual world editor from the Simulink 3D Animation application (VRML) to visualize and induce motion. Their geometries are associated to the respective frame already described. Apart from that, the camera also includes a viewpoint, which provides a target's view from the camera's perspective.

In order to get the views of the camera as images in the Simulink model, like the one in Figure 4.5, the VR to Video block is used. This allows to chose the desired viewpoint from the VRML generated file and the resolution of the video frames. Additionally, the position and orientation of the drone and the target can be defined as input variables of the block, which enables the control of these objects

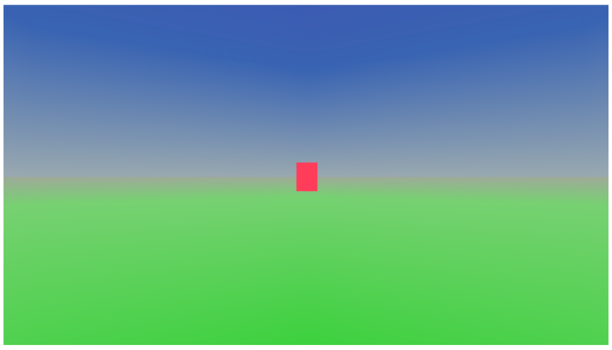


Figure 4.5: Camera view in VRML - the target is the red rectangle

movement. The camera motion is induced by the drone's one, since the camera frame is defined related to drone frame.

4.2.2 Target Centroid Detection

Having the image of the target, this one must be segmented from the rest of the environment, so it can be uniquely identified.

As seen in Figure 4.5, the image background contrasts with the target, which is the red rectangle. Therefore, the image turns into black and white based on the red channel of the RGB color space. The self-developed Matlab function *imbinarize* is used to make this image color space conversion, which also takes as input a threshold required for Otsu's method application. This threshold was established based on the histogram of the image's red channel illustrated in Figure 4.6, where the pixels from the target are the only ones detected upwards from the red line.

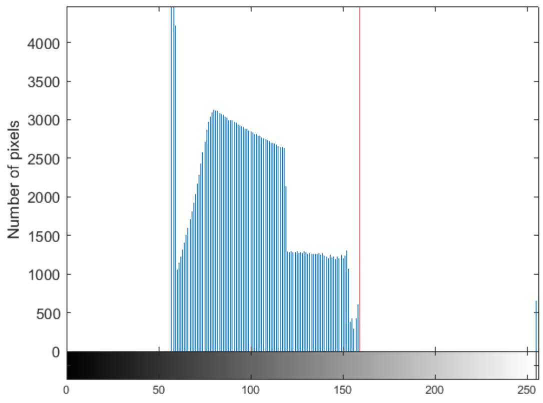


Figure 4.6: Histogram of the VRML image red channel

Then, the *regionprops* Matlab function is applied to the binary image to detect the regions segmented and its properties. In this case, since the target is the only region detected, there is only one centroid on the image, which is shown in Figure 4.7 with a red cross. In this environment, as opposed to the real one, is very unlikely that the target will be outside the field of view and, so, there is no strategy implemented to address this problem.

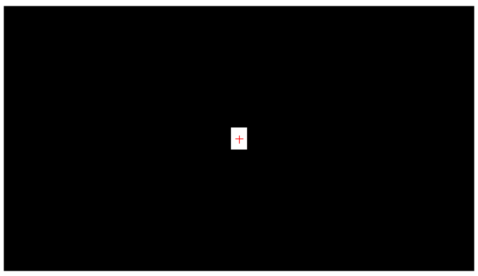


Figure 4.7: Target centroid identification after image being segmented

4.3 Experimental environment

In the experimental model there are two different modes for controlling the position of the drone: waypoint tracking and visual servoing, as shown in Figure 4.8. The first one, which was developed by [23], receives as reference inputs a user defined trajectory points that include information of the desired drone position, the yaw angle and the waiting time the drone should remain on that pose. As already mentioned, the second one has constant references inputs for both IBVS and PBVS case, aiming to drive the drone so the target is centered in the image of the bottom facing camera.

In a practical point of view, the waypoint tracking control mode allows the drone reaching a certain altitude (h^*) after it takes off to increase the so called working distance, i.e. the distance between the lens of the camera and the plane of the target. Having achieved the desired altitude, once the target appears on the image, the visual servo control mode is run. The algorithms of the modules required to determine s, namely Image Acquisition and Centroid Detection and Tracking, are described in detail on Sections 4.3.1 and 4.3.2, respectively.

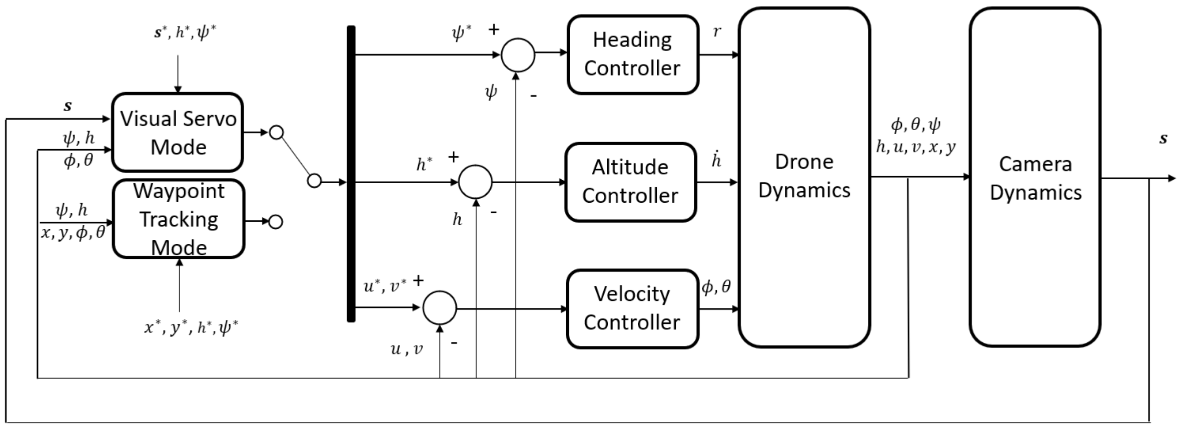


Figure 4.8: Quadrotor control architecture for the real environment

In order to change between modes, a switch block, illustrated in Figure 4.8, is implemented so unique values are set for the heading, altitude and velocity controller reference inputs (ψ^*, h^*, u^*, v^*).

The controllers parameters values, which are specified in Section 4.3.3, are the same for both modes, as the dynamics of the drone does not change. The result of the control action is then transmitted to the drone by AT commands through the ARDrone Wi-Fi Block at the same time it receives sensors data and status from the drone, as mentioned in Section 3.2.2.

4.3.1 Image Acquisition

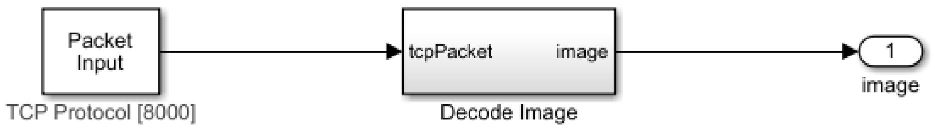


Figure 4.9: Image Acquisition Simulink blocks

The images from the bottom facing camera of the drone are transferred to the host computer in a packet format through a TCP protocol. This protocol is defined in a Packet Input block, as shown in Figure 4.9, and uses port 5555, as established in [24], for the server (drone) to stream video and port 8000 for the client (host computer) to receive it.

Considering there are two cameras fixed to the drone and only one video could be streamed, the video channel configurations must be changed so the bottom facing camera is set. To do so the AT command *AT*CONFIG=' strcmd "video:video_channel","1" char(13)* is defined inside *Drone State Request* Simulink block, which is part of *ARDrone Wi-Fi Block*. The meaning of the AT command strings can be found at [24].

After testing different sample times for the image acquisition process, it was found there is a trade off between image quality and speed of the process. Since the feedback control depends on the image data, its quality cannot be compromised, thus, the fundamental sample time presented in Section 3.2.2 was the one chosen, as it is the minimum value that guarantees image quality.

As mention before, the image is transmitted in a packet format. However, this type of format unit cannot be read by Simulink. So, a Simulink block, named in Figure 4.9 as *Decode Image*, developed by Rui Coelho in C language [25] was used to transform the packet format into uint8 format.

Asynchronous Data Acquisition

By receiving all data from the drone with the same sample time, it was possible to conclude about the existence of a time delay between image and the other sensors data acquisition.

To measure the time delay, two periodic signals were required from the two data sources, so cross-correlation could be applied between them, [26]. Therefore, a signal showing the variation of the area in pixels of the target was created based on the image, in order to be compared to the altitude of the drone measured with the ultrasonic altimeter. The target area was estimated using the Matlab function *regionprops*, after applying the image processing algorithm explained on Section 4.3.2.

In this way, by varying the altitude of the drone, at the highest point, the drone is further from the target and, thus, the area of the object on the image plane is minimum. The opposite happens when the Parrot is at the lowest altitude.

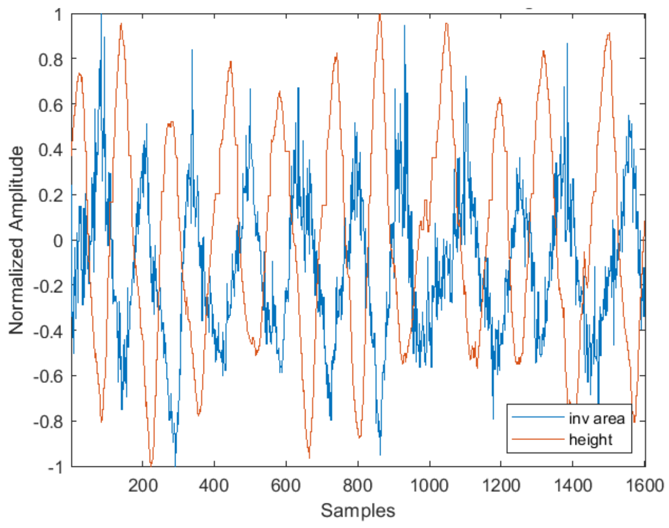


Figure 4.10: Time evolution of the normalized signals: inverse of the object's area and drone altitude

Hence, both signals were normalized between [-1,1] and the inverse of the area was used instead, so they could be compared, as shown in Figure 4.10.

Then, the cross correlation coefficients of the signals were determined using *xcorr* Matlab function, as well as the sample lags at which the coefficients were computed, as illustrated on the graph of Figure 4.11. In this way, the time delay can be determined as the $lag * Ts$ for the maximum cross-correlation coefficient.

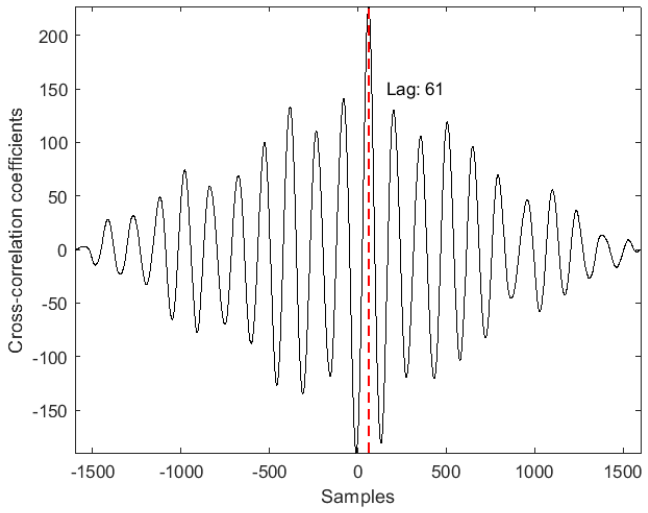


Figure 4.11: Cross-correlation of the normalized signals: inverse of the object's area and drone altitude

After repeating the acquisition process several times for different sample times and number of data samples, was found that the lag which aligns the signals is constant and it is equal to 61 samples. So, in the present case where Ts = 0.065s, the time delay corresponds to 3.956s, which means the image acquisition is delayed by 3.956s with respect to the other sensors information acquisition.

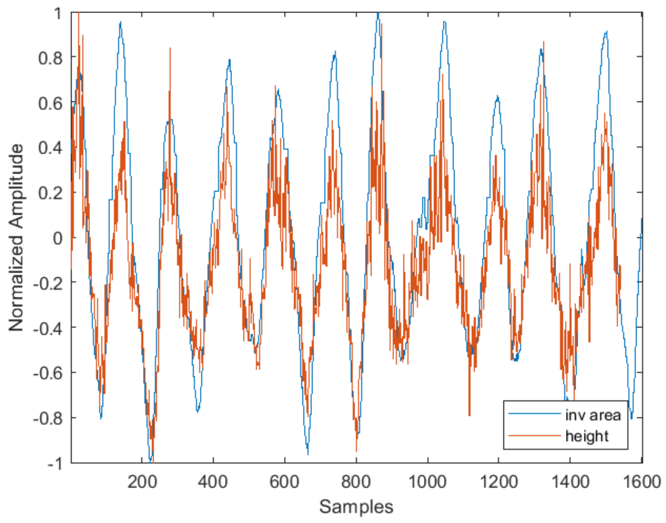


Figure 4.12: Alignment of the normalized signals: inverse of the object's area and drone altitude

Such time delay can be explained by the fact the image and the other sensors data are transmitted through different protocols between drone and host computer. A TCP protocol is used for the image data acquisition, while a UDP protocol is used to acquire the remaining data.

By using a TCP protocol all images captured are delivered by the host computer in the correct order, even if the image no longer corresponds to the current time. On the other hand, a UDP protocol, which is commonly used for streaming, does not require that all data is delivered. In this case, the priority is to transmit the most recent data available to the host computer.

Moreover, the image size is much bigger than the size of the other data, like height, which increases the latency in the communication process. So, for the purpose of this work, it would be preferred to have the drone's camera images transmitted by a UDP protocol, since only the most updated images are required.

This phenomenon makes the centroid location and, consequently, the target position estimations be incorrectly determined for the actual time, since these measurements depend on image data. Therefore, the control action will be compromised, as the system will respond to a velocity reference delayed in time. Additionally, increases the possibility of the target being out of the field of view.

The obvious solution to this problem would be synchronized both signals as shown in Figure 4.12. However, this implies delaying all sensors data acquisition, except image, and readjust the controllers parameters, which would make the system unstable.

Since the communication protocols are drone's predefined settings and the sample time cannot be reduced to not compromise the quality of the images, the experimental tests will proceed with a time delay of 3.955s between image and other sensors data acquisition.

4.3.2 Target Centroid Detection and Tracking

In this module the target's centroid coordinates are determined and it is ensured that even when the target is out of the image, the visual servo control keeps working so the target could be tracked.

The *Centroid Detection and Tracking* module can be divided into two parts, the image processing algorithm and the centroid tracking algorithm. Both of them are presented in detail in the following sub-sections.

Image Processing Algorithm

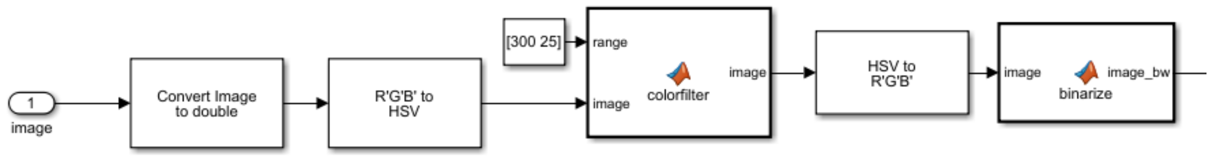


Figure 4.13: Image processing algorithm Simulink blocks

The main objective of the image processing algorithm is to segment the red target. To do so, a red color filter and a image binarization functions were created and they are represented in Figure 4.13 as a Matlab function blocks named *colorfilter* and *binarize*, respectively.

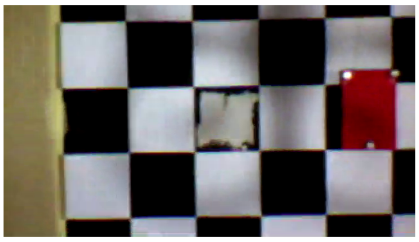


Figure 4.14: Camera original image

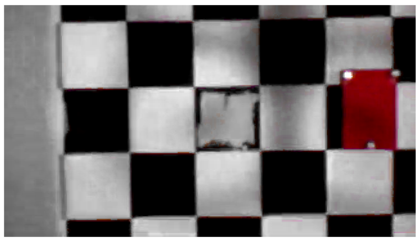


Figure 4.15: Image after red color filter is applied

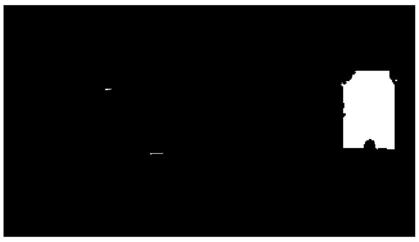


Figure 4.16: Image segmentation

So, after the *Centroid Detection and Tracking* module receives the image coming from the *Image Acquisition* module, the image format type is changed from *uint8* to *double* and then the color space is converted from *RGB* to *HSV*, since *colorfilter* is designed for it.

Basically, the *colorfilter* algorithm identifies the pixels of an image like the one in Figure 4.14 with hue values between the range [300 25], which corresponds to red color. The ones within the range stays with its HSV values and so its RGB values, but the remaining ones are turn into gray scale when the color space changes to RGB as illustrated in Figure 4.15, since the saturation values are set to zero.

Then, in order to segment the target, the image is converted into black and white, like shown in Figure 4.16 (the unfilled space of the rectangle corresponds to the reflective markers, which are grey). This is done by applying a threshold based on the difference between red and green channel values and the red and blue channel values, because there is a huge contrast between them as the red color will appear as black in green and blue channels.

As most of the image processing algorithms, their performance can be influenced by the brightness conditions. In this present work they could vary a lot because the drone structure is in between light focus and camera. However, the range of the color filter and the threshold chosen in binarization process guarantee that even if a shadow covers the target and different elements are identified like in Figure 4.16, the target will always be the largest element being segmented.

Centroid Tracking Algorithm

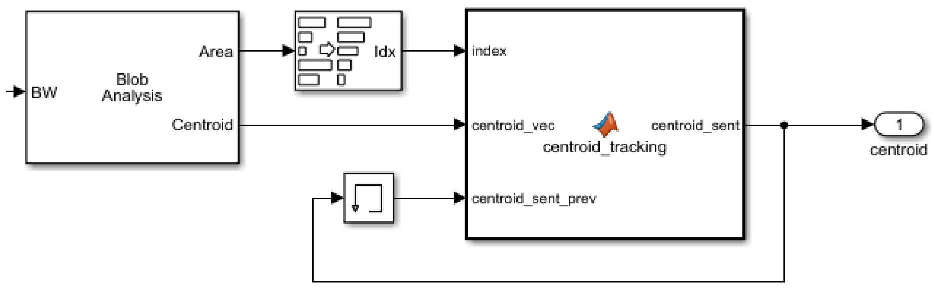


Figure 4.17: Centroid tracking algorithm Simulink blocks

Having the target being segmented on the image, the centroid tracking algorithm can detect its centroid and consider the cases where the target is not correctly identified or is out of the field of view, so the drone could keep tracking it.

First of all, the area and the centroid of all blob in the black and white image received from the image processing algorithm are detected by the Blob Analisys Simulink block shown in Figure 4.17. Even though the image processing algorithm is robust enough to always segment the target, it may identify other smaller blobs as exemplified in Figure 4.16. Thus, the centroid of the target will be selected as the one with the greatest area.

Assuming that on the first time the visual servo control mode is on the centroid tracking algorithm can identify the target centroid, there are three possible subsequent scenarios. The first one is the normal case where the target is inside the camera field of view. In the second one the image acquisition glitches and the image gets distorted. The last one addresses the case when the target is outside the field of view.

In order to understand how these scenarios and the target's centroid are determined, a brief description of the algorithm represented in Figure 4.17 as *centroid_tracking* is presented. In Appendix A the pseudo code of the algorithm can be found.

The algorithm has as inputs the actual centroid of the target determined by the Blob Analisys, **Centroid**, and the previous output of the algorithm, **CentroidSentPrev**, accessed through the Memory block shown in Figure 4.17. The output is the **CentroidSent**, which, depending on the scenario, has different definitions and is one of the inputs to the visual servo controllers.

After the variables being initialized, the algorithm first checks if the **Centroid** coordinates are equal to zero or not to determined whether the target is outside or inside the camera's field of view. Then, the scenarios previously described are analysed.

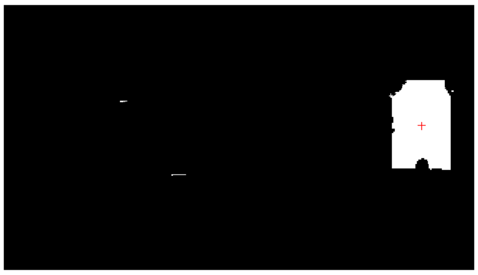
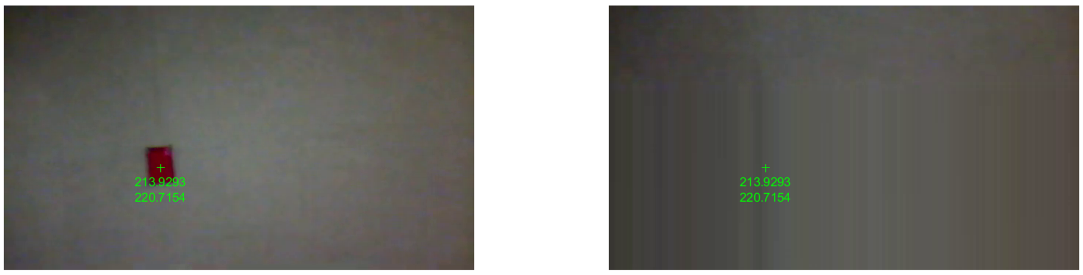


Figure 4.18: Target is inside the camera's field of view - default case for centroid tracking algorithm

The first scenario is defined as the default condition, where the **CentroidSent** returns the actual centroid (**Centroid**), as shown in Figure 4.18.

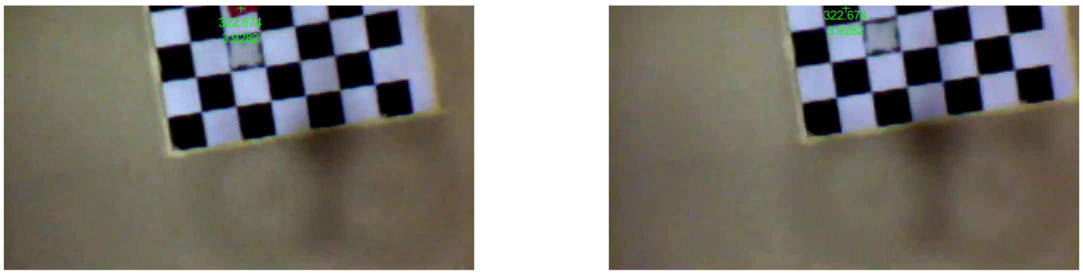


a) previous iteration

b) actual iteration

Figure 4.19: Image gets distorted while the target is inside the field of view and, so, the target centroid coordinates of the actual iteration b) will correspond to the one from previous iteration a)

In the second scenario, the image of the target does not correspond to the reality as most of the times there is no red region or the red region of the target spreads all over the image. Consequently, the image processing algorithm either does not have elements to segment or segments a larger area of red pixels which is deviated from the actual position of the target. Thus, if facing the case illustrated in Figure 4.19, where there is no red region, the centroid of the target is assumed to be the one from previous iteration (**CentroidPrev**). On the other hand, if the second case occurs, the norm of the distance between the actual centroid and the previous one will be bigger than a threshold, X, and if there is no occurrence of this scenario in previous iteration, the **CentroidSent** is also the **CentroidPrev**.



a) previous iteration b) actual iteration

Figure 4.20: Target leaves the image for the first time and, so, the target centroid coordinates of the actual iteration b) will correspond to the one from previous iteration a)

In what concerns the third scenario, there are three ways of identifying it. The first one is when the target is outside the image, after a normal case happens, like shown in Figure 4.20, and, thus, the centroid given as output is the **CentroidPrev**. On the second one two things could happen: the target is consecutively outside the field of view or the target is outside the image for the first time after an occurence of the second scenario. Since this previous way of determining the third scenario occurs after an exception, the **CentroidSent** is the **CentroidSentPrev**. The last one, corresponds to a case where, for the first time, the target is outside the field of view, but there are blobs being segmented. In this case, the centroid identified belongs to one of the blobs, but since it is further from the last target's centroid detected, the output of the algorithm is the **CentroidPrev**.

In case the target is moving, the predictions made for its centroid image coordinates on the previous scenarios are fair, since it is always moving in approximately the same direction.

Finally, it could be the case of consecutive occurrences of the second or third scenarios. In both situations the centroid given as output of the algorithm (**CentroidSent**) is the one previously outputted (**CentroidSentPrev**).

4.3.3 PID controllers' parameters

Based on the PID controllers designed for the simulator model in Section 3.3, the gains of the controllers were tuned for the experimental tests. In general, all gains' value were decreased in order to get a smooth response of the real system. All controllers parameters are shown in Tables 4.1 and 4.2.

	Altitude	Heading	Long. Velocity (Vx)	Lat. Velocity (Vy)
P	0.3	0.1	-0.14	0.23
I	0	0	-0.1	0.115
D	0	0	0	0

Table 4.1: PID controller gains for the control of altitude, heading and velocity of the drone

	IBVS		PBVS	
	$x_{i,p}$	$y_{i,p}$	t_{cf_x}	t_{cf_y}
λ	-0.0001	-0.0001	0.1	0.1

Table 4.2: IBVS and PBVS control law gains

4.3.4 Ground Truth of Sensors' Data Estimations

In order to evaluate the accuracy of the drone sensors measurements used for feedback control, the data was recorded by the Qualisys Track Manager (QTM) motion capture system installed in Robotics Arena of IST and is established as the ground truth.

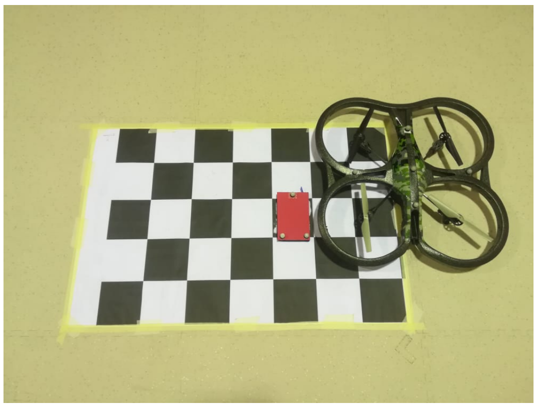


Figure 4.21: Experimental setup

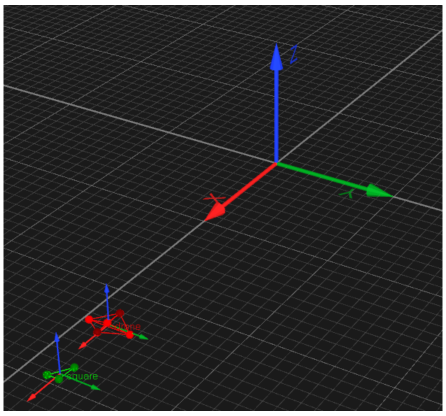


Figure 4.22: QTM virtual environment

The QTM is a software that allows 2D/3D motion capture of rigid bodies. During the capture the rigid bodies are automatically tracked by QTM, providing its 6 DOF data in real-time.

To set the Parrot as a rigid body in QTM, 5 retro-reflective markers were fixed to its indoor hull as shown in Figure 4.21, so the drone could be identified and tracked. As seen in both Figures 4.21 and 4.22, the same happens with the target, but only 3 markers are required to track its translational motion. The origin of the rigid body frames is assumed by the system to be the geometric center of the markers, which corresponds approximately to the center of the hull and the center of the target, respectively.

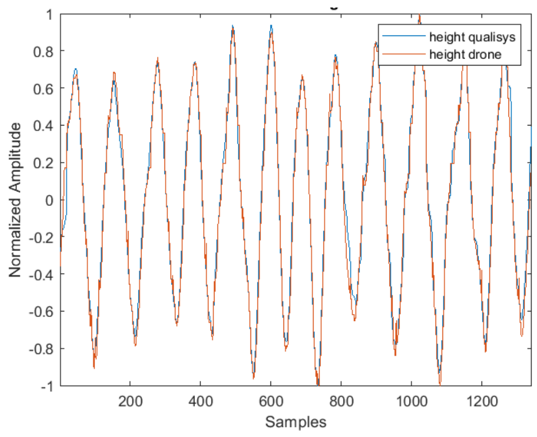


Figure 4.23: Normalized altitude signals given by QTM and ultrasonic altimeter

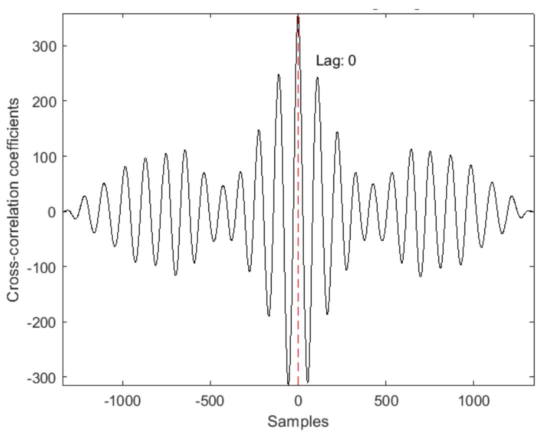
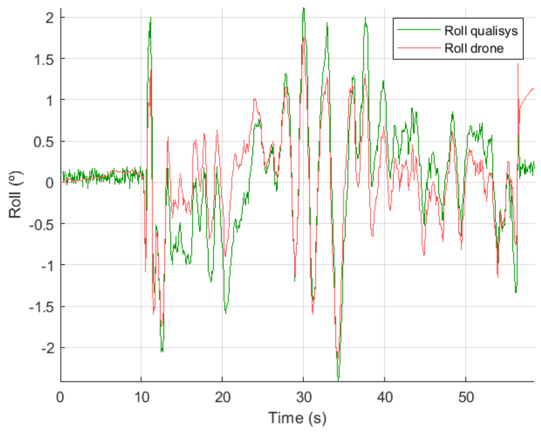
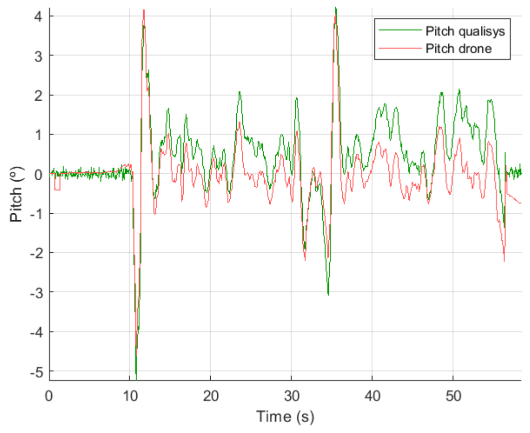


Figure 4.24: Cross-correlation of the normalized altitude signals

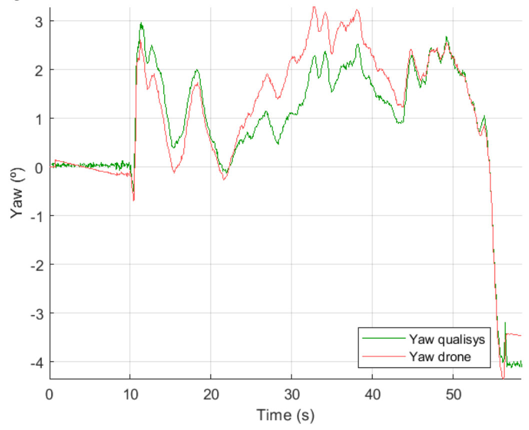
By receiving both data in the same Simulink model, all drone's sensors and QTM measurements were acquired with same sample time. As shown in Figure 4.23, the height data acquired from QTM and from the ultrasonic altimeter are synchronized, since the maximum correlation coefficient between signals (Figure 4.24) corresponds to zero lag. In this way, data from the drone's embedded sensors, except camera, and from the QTM can be directly compared.



a) Roll angle



b) Pitch angle



c) Yaw angle

Figure 4.25: Comparison of QTM and IMU measurements for drone rotational angles: roll (a)), pitch (b)) and yaw (c))

As seen in Figure 4.25, the drone's orientation measurements given by the QTM and the drone's embedded IMU follow approximately the same variation. Even though there are some discrepancies between the signals, they do not play a major concern since its maximum value is around 1^{o} for all of them.

In what concerns the altitude measurements represented in Figure 4.26, there is a small offset of 0.1m between the QTM and the ultrasonic altimeter data, which is clearly visible when the altitude is nearly constant. This happens since the measurements reference points are slightly different.

Finally, the horizontal translational velocities estimated by the image-based algorithms mentioned in Section 3.1 are compared to the derivative of the horizontal position data given by QTM, as shown in Figure 4.27. Although the derivative data is very noisy, it can be seen that both signals follow the same trend.

In order to test the variability of the image-based algorithms on estimating the translational velocities,

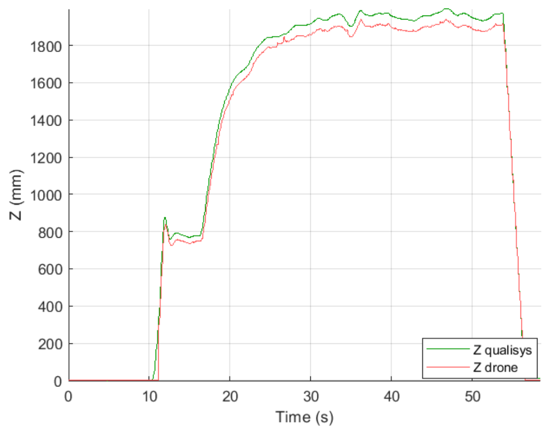
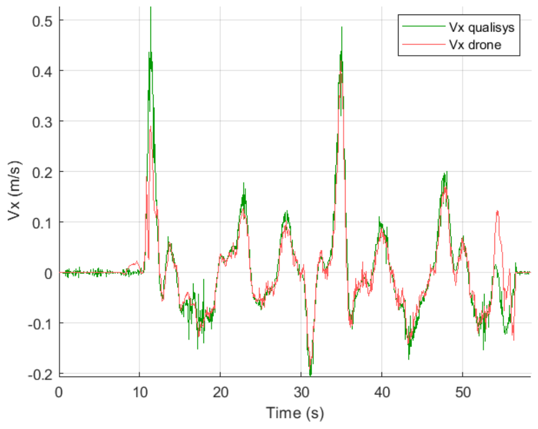
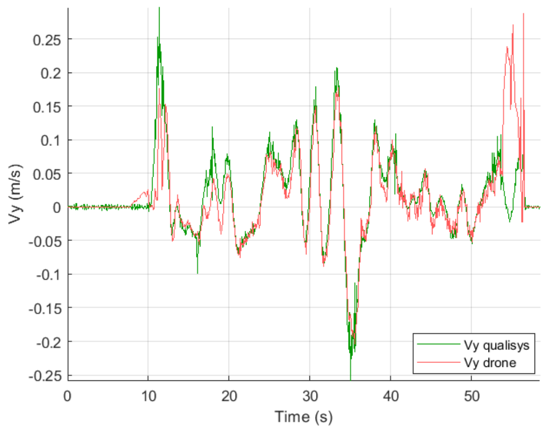


Figure 4.26: Comparison of QTM and ultrasonic altimeter measurements for drone altitude

several tests were perform where the drone is hovering at different altitudes.



a) Velocity in x direction



b) Velocity in y direction

Figure 4.27: Comparison of QTM position derivatives and image-based estimations for drone translational velocities in x (a)) and y (b)) directions of drone frame

Tables 4.3 and 4.4 show the maximum of the error's module and the rms of the error between velocities in x and y direction, respectively, for each altitude between 1 and 2.8m with a step size of 0.3m. By looking at the estimators, it is clear that the altitude and the error of the velocity measurements are independent variables, since there is no positive or negative correlation between them.

Altitude (m)	|Error max| (m/s)	RMS error (m/s)
1	0.0672	0.0147
1.3	0.0723	0.0204
1.6	0.0947	0.0245
1.9	0.0658	0.0157
2.2	0.0660	0.0156
2.5	0.1872	0.0224
2.8	0.0453	0.0148

Table 4.3: Statistics of the error between QTM and image-based measurements for the x velocity component of the drone at different flight altitudes

Even though the image resolution decreases with increasing camera's altitude, it is proved that as long as the checkerboard pattern surface shown in Figure 4.22 is inside the camera's field of view, the

Altitude (m)	|Error max| (m)	RMS error (m)
1	0.0596	0.0194
1.3	0.2183	0.0384
1.6	0.2641	0.0367
1.9	0.1329	0.0221
2.2	0.1332	0.0220
2.5	0.1432	0.0239
2.8	0.0701	0.0210

Table 4.4: Statistics of the error between QTM and image-based measurements for the y velocity component of the drone at different flight altitudes

translational velocities estimations are fair and independent from the altitude. Probably the squares of the checkerboard are sufficiently large and there is a huge contrast between the its colors, so its features could be detected uniquely by the image-based algorithms used to estimate the velocities.

Chapter 5

Results

5.1 Simulator environment

In order to fairly compare the results between the visual servo approaches, the initial position of the drone is the same for all simulations. This is (0.5,1.5,-2)m and it is defined in a referential parallel to the drone frame, whose origin belongs to target's plane. Plus, in the cases where the target is moving, its velocity is set to 0.1m/s.

Moreover, the simulation results are validated with the findings described in [21].

5.1.1 IBVS

Static Target

In the case where the target is static, the error (2.8) converges to zero and the trajectory of the target's centroid in the image plane is almost a straight line, as illustrated in Figure 5.1.

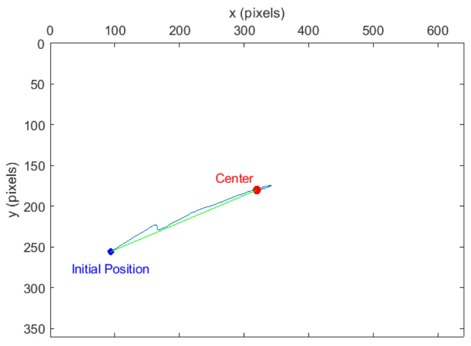


Figure 5.1: Target trajectory in image space - IBVS simulated for static target in simulator model

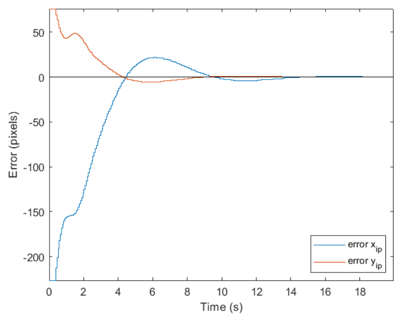
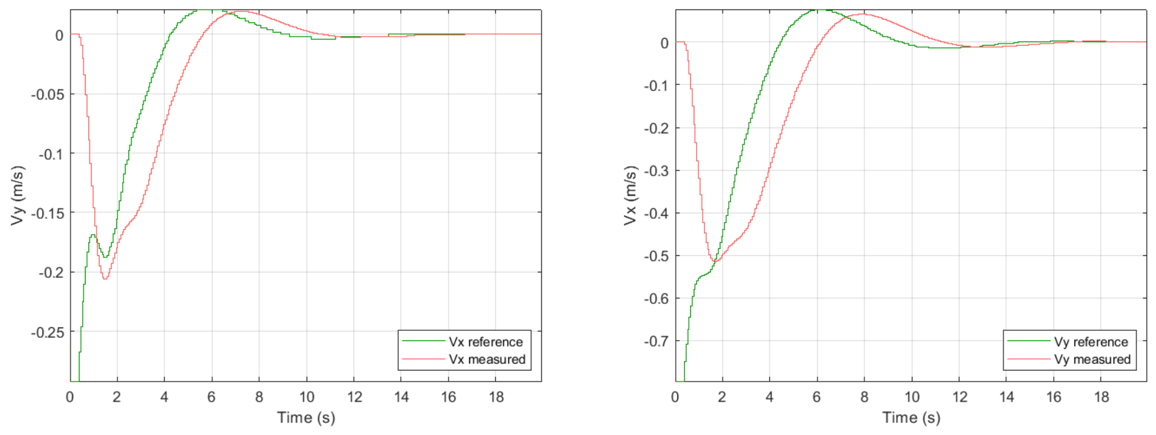


Figure 5.2: Error of x and y target image coordinates - IBVS simulated for static target in simulator model

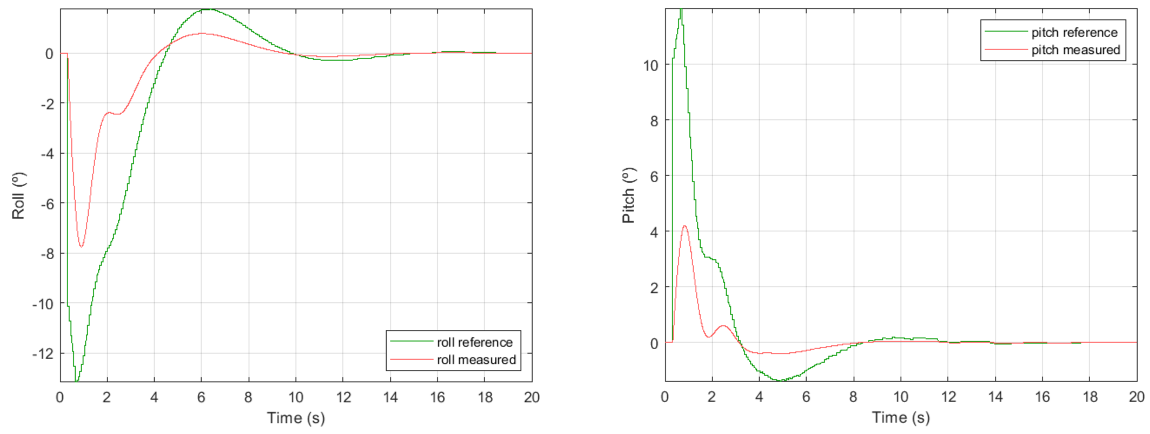
However, at the beginning of the simulation the trajectory is further from the green straight line shown in Figure 5.1, which according to [21] is explained by the high condition number of $\mathbf{L}_\mathbf{e}^+$ and it is translated in a high peak of the camera velocity as depicted in Figure 5.3.



a) Velocity in x direction b) Velocity in y direction

Figure 5.3: Reference and measured velocity a) x and b) y components relative to drone frame - IBVS simulated for static target in simulator model

As expected there is an exponential decrease of the image tracking error (Figure 5.2), but only after the first 6s of simulation, which means the convergence properties of the image behaviour are far from ideal. This effect is also shown in the velocity and Euler angles signals represented in Figures 5.3 and 5.4, respectively.



a) Roll b) Pitch

Figure 5.4: Reference and measured rotational angles a) Roll and b) Pitch relative to drone frame - IBVS simulated for static target in simulator model

In IBVS approach the orientation of the drone with respect to the target is interpreted as an error in the image plane. Thus, the actuation of the roll and pitch angles increases the error in the image plane and delays its convergence to zero.

Moving Target

When the target is moving with constant velocity, there will always be a constant steady state error of the target centroid coordinates in the image plane, since the linear systems that describe the behaviour of these coordinates are type 1, as referred in Section 3.3.3.

Therefore, if the target moves in the x direction of the drone frame, there will be a constant steady state error in the y direction of the image plane as illustrated in Figure 5.5 and in more detail in Figure

5.6. This error corresponds to 26 pixels, which is 7.22% of the image size on that direction.

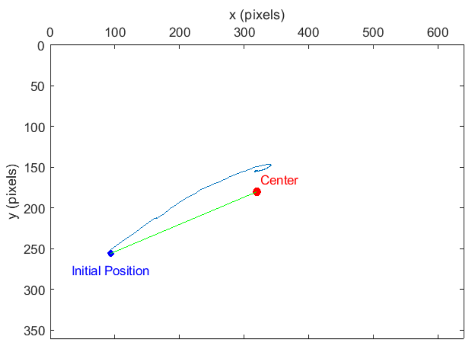


Figure 5.5: Target trajectory in image space - IBVS simulated for a target moving forward in simulator model

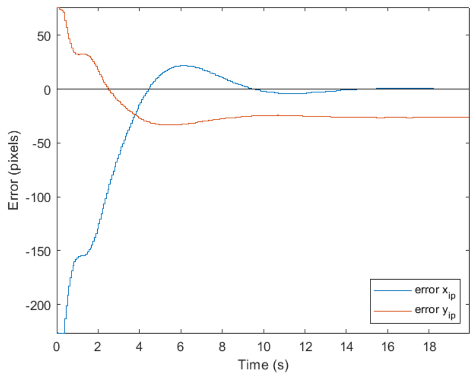


Figure 5.6: Error of x and y target image coordinates - IBVS simulated for a target moving forward in simulator model

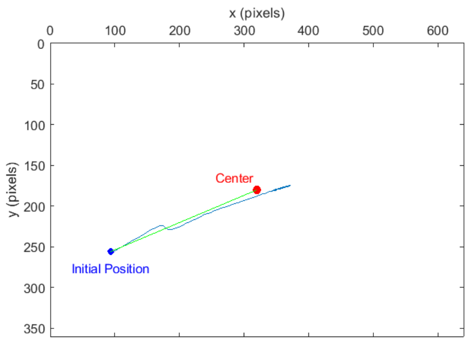


Figure 5.7: Target trajectory in image space - IBVS simulated for a target moving laterally in simulator model

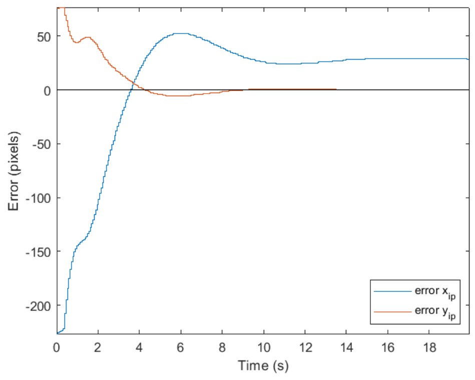


Figure 5.8: Error of x and y target image coordinates - IBVS simulated for a target moving laterally in simulator model

On the other hand if the target is moving in the y direction of the drone's frame, the steady state error occurs for the x coordinate of the image plane, as shown in Figures 5.7 and 5.8. In this case the error is 28 pixels, which corresponds to 4.37% of the length of the image.

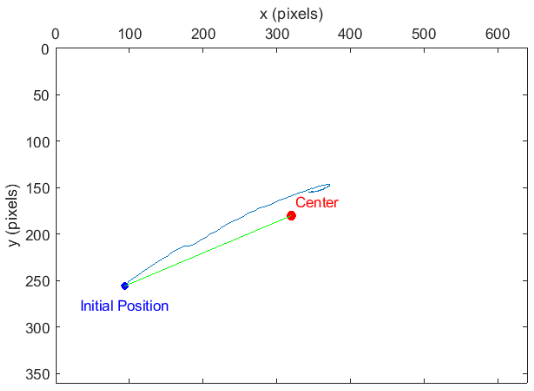


Figure 5.9: Target trajectory in image space - IBVS simulated for a target moving diagonally in simulator model

So, in the case the target is moving on both directions there will be both steady state errors mentioned above, as illustrated in Figure 5.9.

All the characteristics of IBVS approach described for the static target are applied to these cases.

5.1.2 PBVS

Static Target

According to [21], in the PBVS approach the trajectory of the target's centroid on the image plane should be a pure straight line, which does not happen for the target trajectory on the cartesian plane. As seen in Figures 5.10 and 5.11 both predictions occur when the target is motionless.

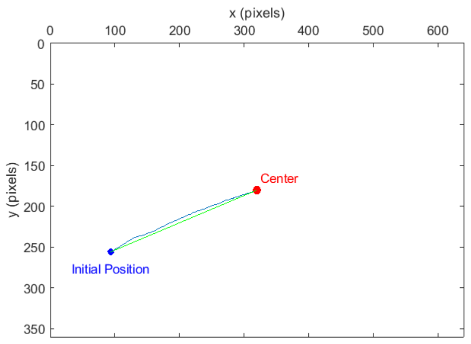


Figure 5.10: Target trajectory in image space - PBVS simulated for static target in simulator model

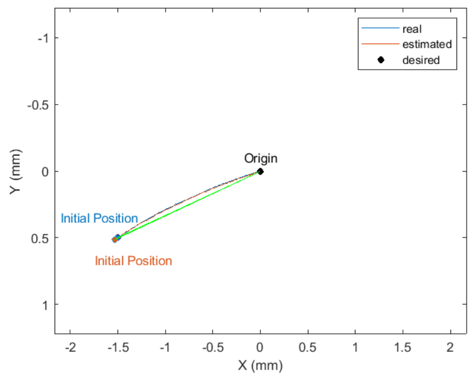


Figure 5.11: Target trajectory in Cartesian space - PBVS simulated for static target in simulator model

In contrast with the IBVS approach, there is an exponential decrease of the error (2.8) and velocity components almost from the beginning of the simulation (around 1s), as illustrated in Figure 5.12 and 5.13, respectively. Plus, those signals meet zero at the same time, which is approximately 11s.

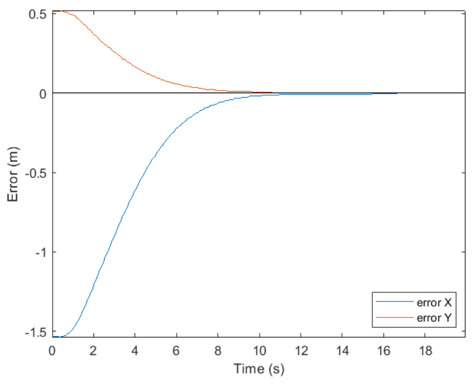


Figure 5.12: Error of x and y target 3D coordinates with respect to camera frame - PBVS simulated for static target in simulator model

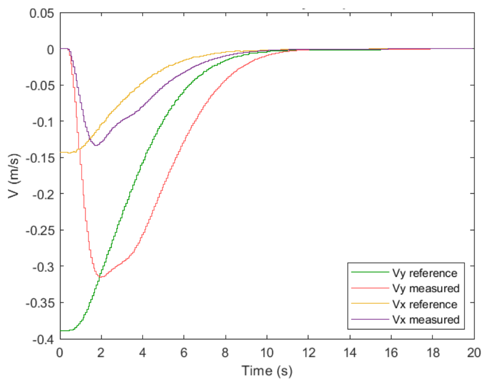


Figure 5.13: References and measured velocities of the drone relative to drone frame - PBVS simulated for static target in simulator model

Moreover, the target position estimations made using the camera model described in Section 2.1 are very close to the real values set in the virtual world environment, as shown in Figure 5.11. By looking into Figure 5.14 it is possible to conclude that the maximum errors of the estimations are 36mm and 17mm in the x and y directions of the camera frame. These values are consider low since they correspond to 0.83% and 0.7% of the image size on those directions, respectively.

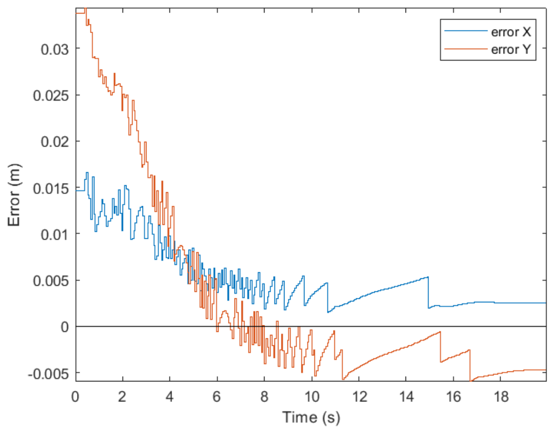


Figure 5.14: Error of target horizontal position estimations with respect to the camera frame - PBVS simulated for static target in simulator model

The position estimation errors might have been introduced by the estimated values for the intrinsic camera matrix parameters. This explains the higher error in y direction in comparison with the error in the x direction shown in Figure 5.14, since the target is further from the center of the image in the larger direction of the image plane.

Moving Target

Once again if the target is moving with constant velocity there will always be a steady state error of the target position in the cartesian plane, since the linear systems that represent the dynamics of those coordinates are type 1, as mentioned in Section 3.3.3.

Having said that, when the target is moving in the x direction of the drone's frame, there is a constant steady state error in y direction of the camera frame, as shown in Figure 5.15 and more in detail in 5.16. The error is equal to 0.36m, which in the y direction of the image plane is equivalent to 50 pixels (13.9$\%$ of the width of the image size).

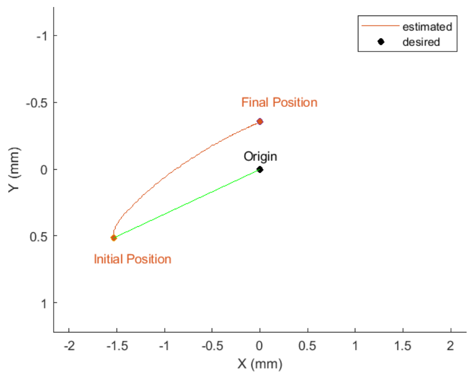


Figure 5.15: Target trajectory in Cartesian space - PBVS simulated for a target moving forward in simulator model

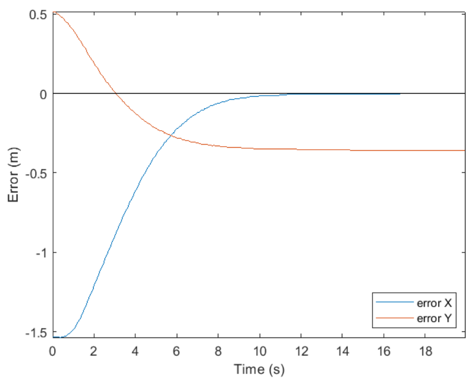


Figure 5.16: Error of x and y target 3D coordinates with respect to camera frame - PBVS simulated for a target moving forward in simulator model

On the other hand, if the target moves in the y direction of the drone's frame, there is a constant steady state error in the x direction of the camera frame, as seen in Figures 5.17 and 5.18. It corresponds to 0.39m, which is equivalent to 55 pixels error in the x direction of the image plane (8.6% of the length of the image size).

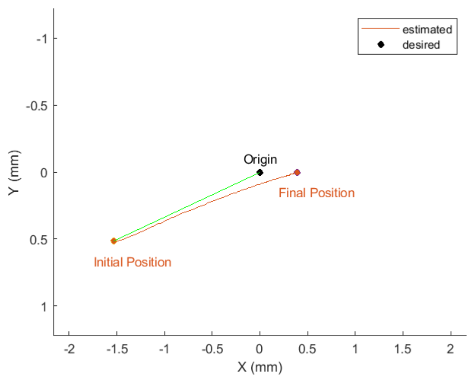


Figure 5.17: Target trajectory in Cartesian space - PBVS simulated for a target moving laterally in simulator model

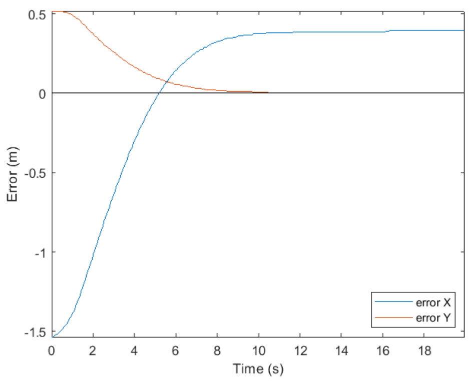


Figure 5.18: Error of x and y target 3D coordinates with respect to camera frame - PBVS simulated for a target moving laterally in simulator model

Finally, if the target is moving in x and y directions, there will be a steady state error on both directions with the same magnitudes presented for the previous cases, as shown in Figure 5.19.

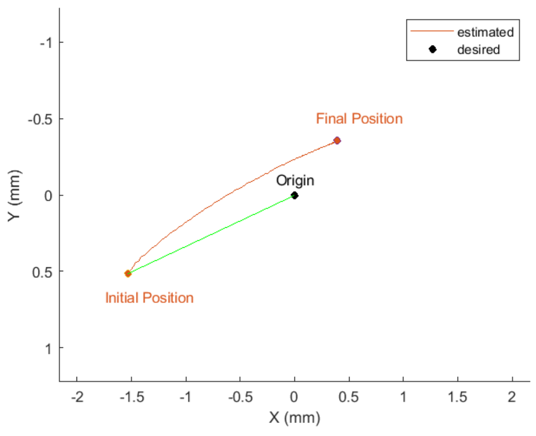


Figure 5.19: Target trajectory in Cartesian space - PBVS simulated for a target moving diagonally in simulator model

5.2 Experimental environment

Taking into account that the time delay between the image and other sensors data acquisition compromises the control action, a saturation is applied to the error of visual servo control laws. In this way, the velocity references, which are delayed in time, are less aggressive to the system when the error is large. This proposed solution is tested and compared to the classical approaches of IBVS and PBVS for the case the target is static.

5.2.1 IBVS

Static Target

First of all, the initial position of the target in the image plane is corrected for both error approaches. As expected, the target approaches the center of the image following a smooth trajectory, which is close to the straight line that connects the initial position point and the center one, as shown in Figures **??** and 5.21.

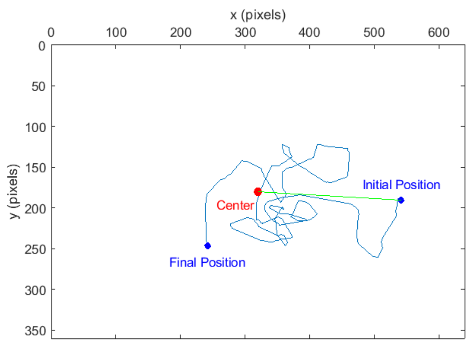


Figure 5.20: Target trajectory in image space - IBVS simulated for a static target in real-time model

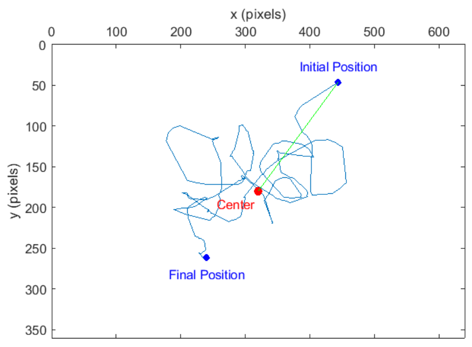


Figure 5.21: Target trajectory in image space - IBVS with error saturation simulated for a static target in real-time model

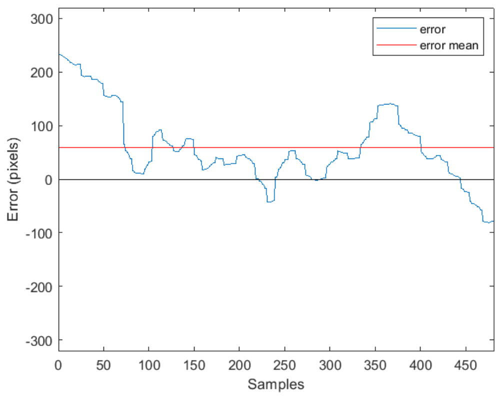
After correcting the initial position, both approaches do consecutive attempts to position the target in the center of the image. However, the case in which the error is saturated (Figure 5.21) shows more attempts.

This effect is also shown by the graphs in Figure 5.22, where the position error in both image directions, x and y, is depicted. Even though the error does not converge to zero, there are several times where it crosses zero or almost reaches that value. Once again the proposed approach has more of these occurrences.

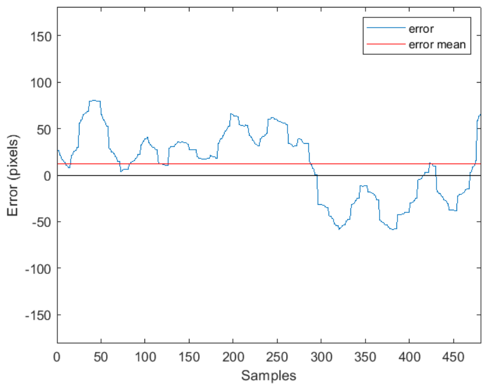
However, looking into the modulus of the mean values of the error, there is no clear relation between them and the type of error. The mean of the error in x direction is bigger for the non saturated case, while the one in y direction is bigger for the saturated case.

So, in order to take a better overview, a batch of 5 tests was done for the two different cases, where the drone takes off from the same position and whose statistics are depicted on Tables 5.1 and 5.2. It was found that the average of the error's modulus mean value in x and y direction are 69 and 42 for the simple case and 68 and 42 for the saturated one, respectively. On the other hand, the average values for the maximum of the error in x and y directions are 206 and 133 for the simple error and 178 and 120 for the saturated error, respectively. Even though the average of the error's modulus maximum value are bigger for the simple case than the saturated one, there is no clear evidence that the error's saturation improves the IBVS performance, as the average values of the mean error's modulus are practically the same.

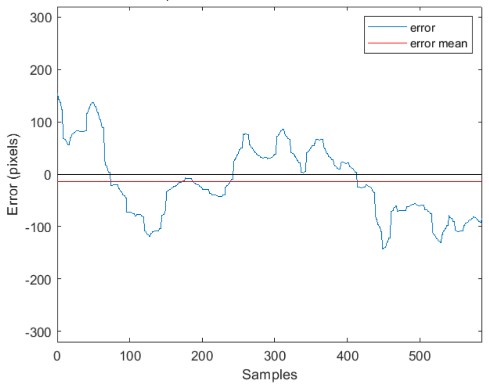
In fact the target covers a larger region on the image plane in case of using the simple error, which



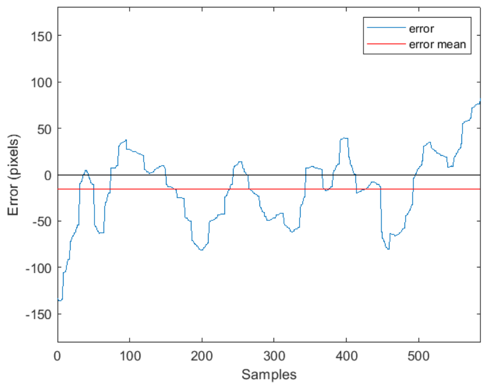
a) x coordinate



b) y coordinate



c) x coordinate - error saturation



d) y coordinate - error saturation

Figure 5.22: Error of x and y target image coordinates - IBVS simulated with and without saturation for static target in real-time model

means it is most probable of getting the target outside the camera's field of view than the saturated one. This is likely to happen because the maximum of the error's modulus corresponds to 32% and 37% of the image size for the x and y directions, respectively.

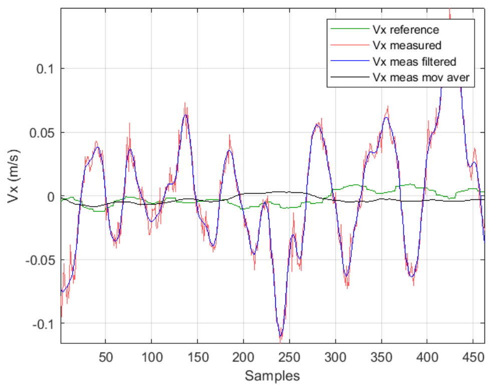
Test No	x error mean (pixels)	y error mean (pixels)	x error max (pixels)	y error max (pixels)
1	83	43	256	125
2	70	45	183	118
3	64	47	192	147
4	45	36	153	101
5	80	42	248	174

Table 5.1: Statistics of x and y target image coordinates error signals - IBVS simulated for a static target in real-time model

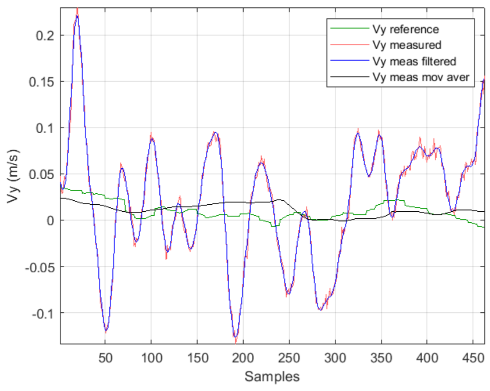
Test No	x error mean (pixels)	y error mean (pixels)	x error max (pixels)	y error max (pixels)
1	72	45	166	114
2	68	31	186	102
3	55	49	174	112
4	76	44	192	160
5	70	42	170	110

Table 5.2: Statistics of x and y target image coordinates error signals - IBVS with error saturation simulated for a static target in real-time model

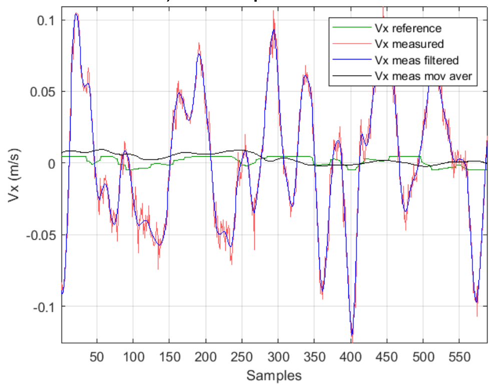
One of the big reasons for the permanent variation of the position signal on both cases presented before is the incapability of the drone to follow the horizontal velocity references. This comes clear when looking into the magnitudes of red and green signals in Figure 5.23, which corresponds to the reference and sensor's measured velocities, respectively. Such discrepancies might be explained by the image-based algorithms used to estimate the velocities not being able to distinguished between translational and angular motion, [27]. This coupling effect of the quadrotor dynamics cannot also be detected by the QTM measurements and for that reason the drone velocity estimations were validated in Section 4.3.4.



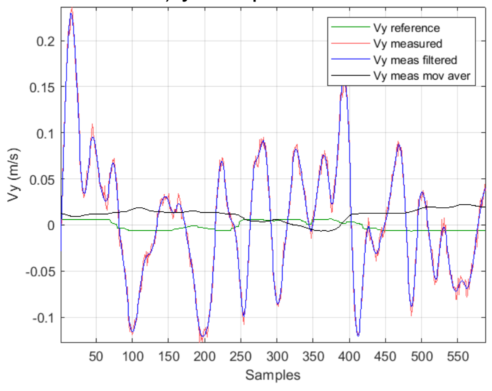
a) x component



b) y component



c) x component - error saturation



d) y component - error saturation

Figure 5.23: Reference and measured velocity x (a) and c)) and y (b) and d)) components relative to drone frame - IBVS simulated with and without saturation for static target in real-time model

Moreover there is no clear evidence that the moving average of the measured velocity represented in black (Figure 5.23) follows the trend of the reference one. So, it is difficult to evaluate the effectiveness of the IBVS control.

Moving Target

Also a moving target with constant velocity was tested to evaluate the drone capability on tracking it, even if it is momentarily outside the camera's field of view. The target is moved by hand using a rope fixed to it, therefore the velocity is not keep exactly constant.

Let is consider the case where the target is moving straight in the x direction of drone's frame. By looking at Figure 5.24 is clear that the drone cannot reach the target in order to get it in the center of the the image.

The simulator tests shown that the error in the y direction of the image plane should converge to a

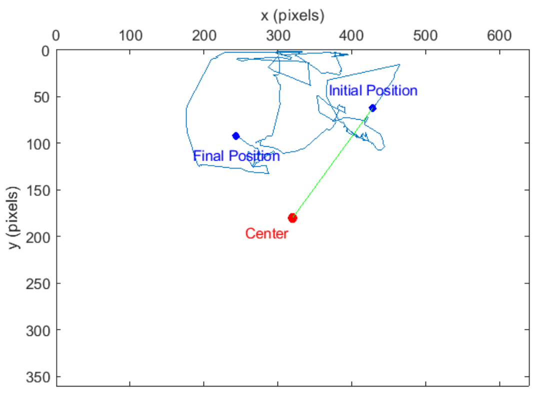
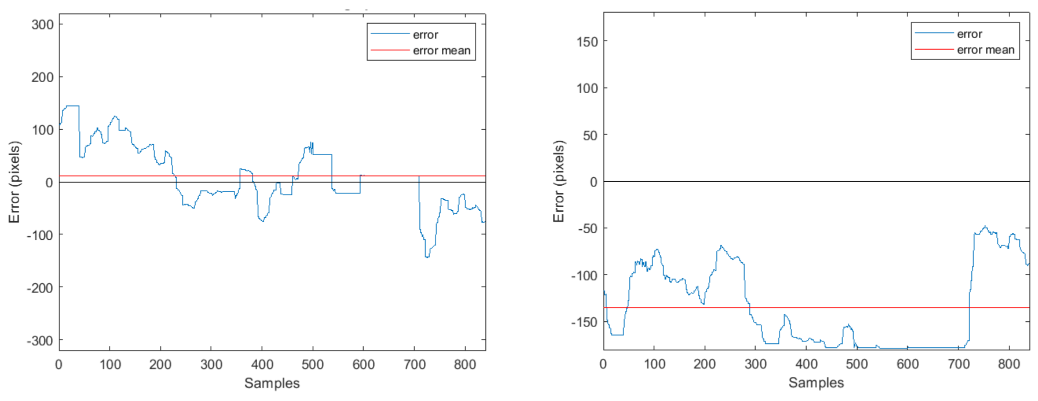


Figure 5.24: Target trajectory in image space - IBVS simulated for a target moving forward in real-time model

steady state close to zero. However, due to the time delay between image and the other sensors data acquisition and variations of the target's velocity, this does not happen in the real environment.



a) x coordinate

b) y coordinate

Figure 5.25: Error of x (a)) and y (b)) target image coordinates - IBVS simulated for a target moving forward in real-time model

In Figure 5.25 is illustrated the position error in x and y directions correspondent to the case presented in Figure 5.24, where the constant lines depicted on the graphs represent the moments when the target is out of the field of view. In this case, since the target is moving in the opposite direction of the camera frame y-axis, the target tends to reach the upper boarder of the image, which corresponds to 0 pixels in y direction of the image plane and to -180 pixels in terms of error in this direction. That is why the mean of the error in the y direction (-135 pixels) is bigger in modulus than the one for the x direction (11 pixels).

Even though the drone never reaches the target, the drone can track it, since there is always a point which identifies the target on the image plane. To illustrate it, the x and y position of the target and the drone given by the Qualisys were compared, as shown in Figure 5.26.

Despite the great variation of drone's position in y direction of Qualisys fixed frame, its mean value (217 mm) is close to the target's y position ($\approx$ 312 mm), as illustrated in Figure 5.27, and the drone's x position follows the target one.

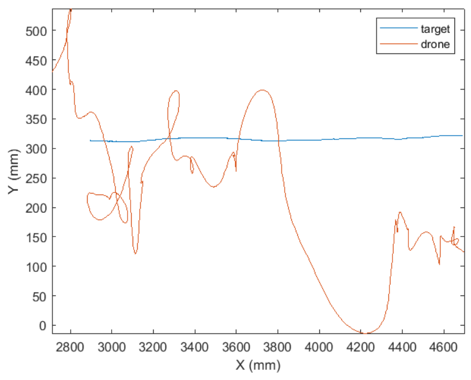


Figure 5.26: QTM data for target and drone trajectory in Cartesian space - IBVS simulated for a target moving forward in real-time model

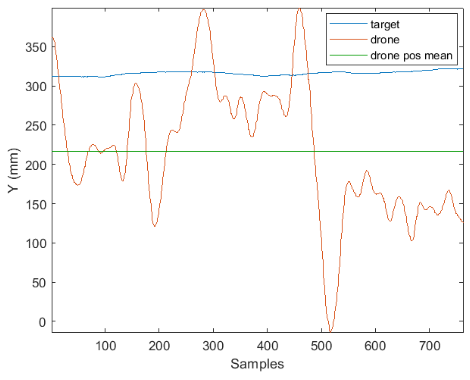


Figure 5.27: Error between QTM data for target and drone position in Cartesian space - IBVS simulated for a target moving forward in real-time model

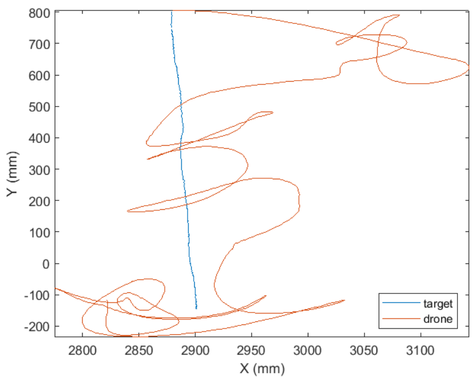


Figure 5.28: QTM data for target and drone trajectory in Cartesian space - IBVS simulated for a target moving laterally in real-time model

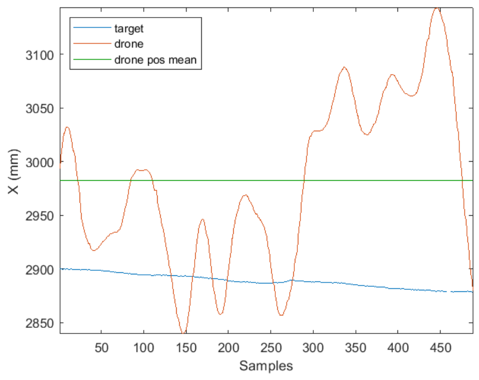


Figure 5.29: Error between QTM data for target and drone position in Cartesian space - IBVS simulated for a target moving laterally in real-time model

When the target is moving in y direction of the drone's frame, there is less variation of the drone's position in the perpendicular direction compared to the previous case, as shown in Figure 5.28 and more in detail in 5.29. Once again, the mean value of the drone's position in the opposite direction to the one the target is moving (2982 mm) is almost coincident with the correspondent target's position coordinate ($\approx$ 2890 mm).

Furthermore, the tracking performance of the drone when the target moving on both x and y direction was also evaluated. It was found that the drone keeps track of the target in both x and y direction with an almost constant offset, as seen in Figure 5.30.

Moreover, the visual servo control effect is visible for this previous scenario, since the velocity references are bigger. Looking into Figure 5.31, it is clear that the trend of the velocity measured, represented by the moving average signal, follows the reference velocity.

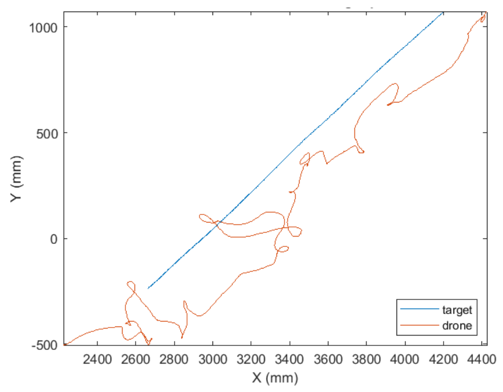
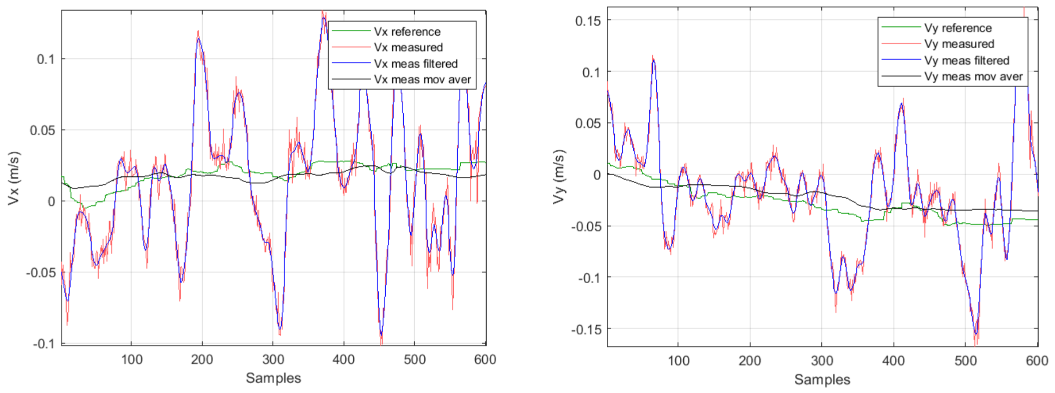


Figure 5.30: QTM data for target and drone trajectory in Cartesian space - IBVS simulated for a target moving diagonally in real-time model



a) x component

b) y component

Figure 5.31: Reference and measured velocity x (a)) and y (b)) components relative to drone frame - IBVS simulated for a target moving diagonally in real-time model

5.2.2 PBVS

Validation of camera position estimation

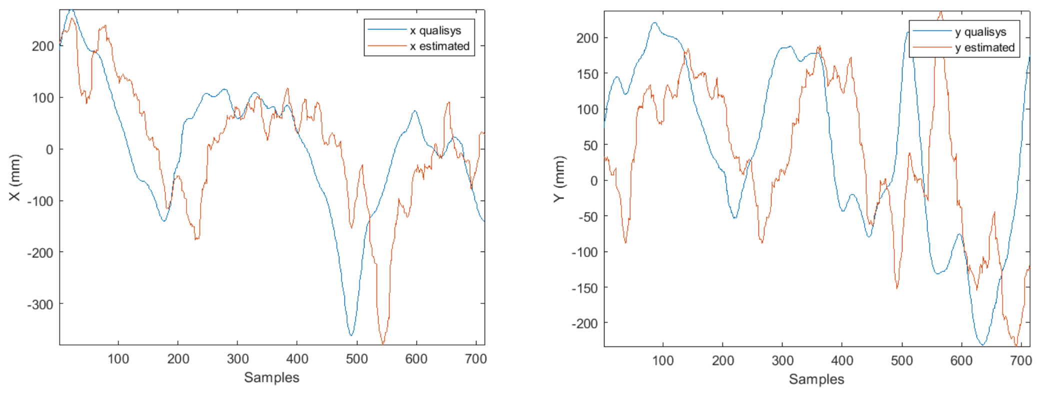
In order to validate the position estimation of the target expressed relative to the camera frame, $\mathbf{t}_{cf}$, the QTM position data for the drone and the target was used. For the sake of comparison, the camera position in Qualisys is considered to be the drone position, whose measurements are available.

As seen in Figure 5.32, there is a huge discrepancy between the estimations of the camera model and the QTM measurements.

Since the target position estimated by the camera model depends on the image data, it is delayed 61 samples relatively to the Qualisys data, as proven on Section 4.3.1. Thus, if the offset is corrected, the x and y signals become synchronized, as shown in Figure 5.33.

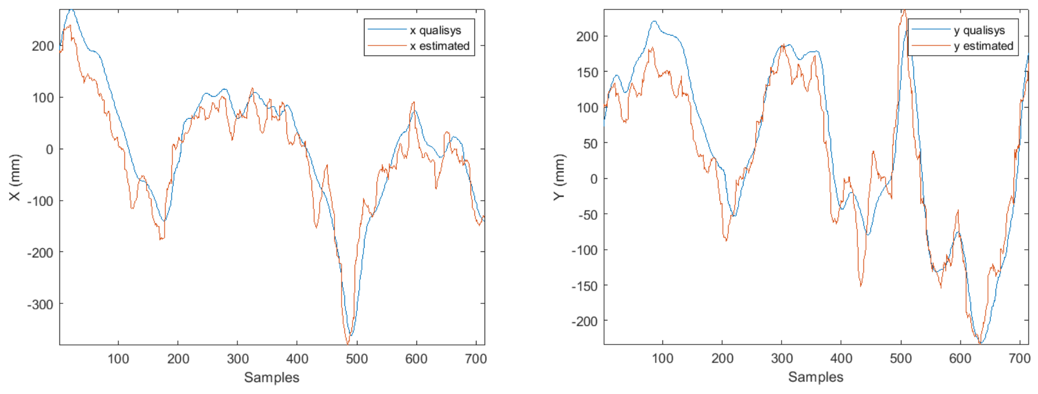
In this way, the data can be truly compared. As shown in Figure 5.34, the error between the estimations and Qualisys data varies in time. This is explained by the randomness associated to the image processing algorithm used to detect target centroid, whose coordinates in pixels influences the calculation of $\mathbf{t}_{cf}$. Despite that, the error of the position estimation is low.

Considering the case being analyzed, the rms of the error, which is 0.043m for the x position and 0.040m for the y position, represents 3.4$\%$ and 5.6$\%$ of the image size in x and y direction, respectively.



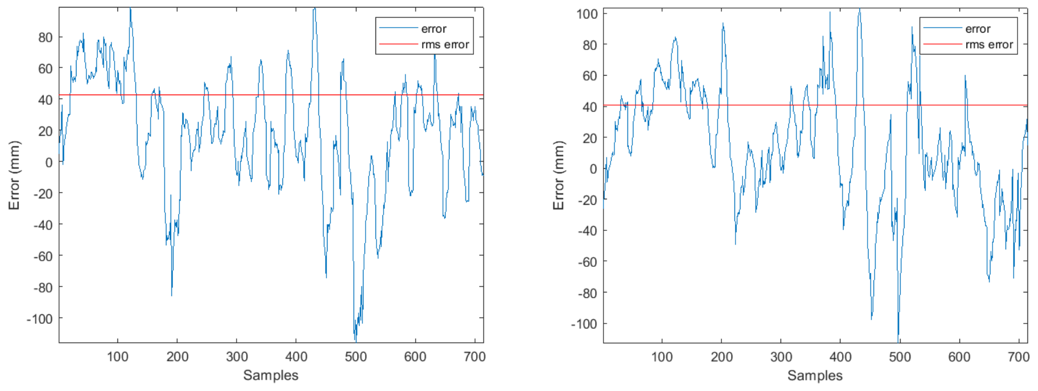
a) x component b) y component

Figure 5.32: Camera model estimations and QTM data for target x (a)) and y (b)) 3D coordinates with respect to camera frame - PBVS simulated for a static target in real-time model



a) x component b) y component

Figure 5.33: Synchronized camera model estimations and QTM data for target x (a)) and y (b)) 3D coordinates with respect to camera frame - PBVS simulated for a static target in real-time model



a) x coordinate b) y coordinate

Figure 5.34: Error of x (a)) and y (b)) target 3D coordinates estimations - PBVS simulated for static target in real-time model

Plus, the maximum of the error's module, which is 0.116m for the x position and 0.112m for the y position, corresponds to 9.2% and 15.7% of the image size in x and y direction, accordingly.

In a general overview, this tendency is illustrated on Table 5.3, where the errors are represented as a percentage relatively to the size of the field of view in millimeters. The mean of the rms error is 6.1% and 7.6% for x and y direction, respectively, while the mean of the module of the maximum error is 15.1% in

Test No	rmse x (%)	rmse y (%)	max x (%)	max y (%)
1	5.9	6.3	12.0	15.0
2	8.0	8.9	18.3	20.4
3	7.7	7.8	21.0	27.1
4	3.7	6.8	11.6	21.5
5	5.1	8.4	12.6	22.3

Table 5.3: Statistics of x and y 3D target coordinates estimated through the camera model - PBVS simulated for a static target in real-time model

x direction and 21.3% in y direction.

Errors could have be introduced by inaccurate estimations of intrinsic camera parameters and by considering different world points for the camera frame origin location.

Static Target

In Figures 5.35 and 5.36 is illustrated the target position evolution with respect to the camera frame for the PBVS and PBVS with error saturation, respectively. For both approaches, the target gets closer to the camera frame origin following a curved line, as expected. Then, the drone makes consecutive attempts to reach the origin every time its position is deviated from it.

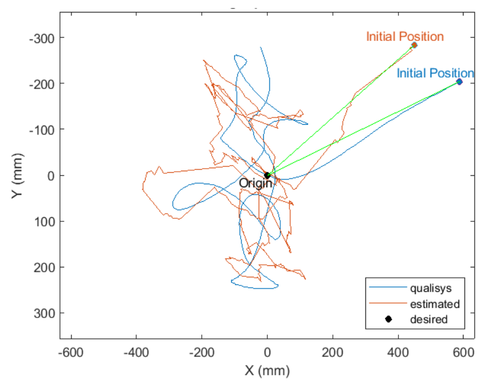


Figure 5.35: Target trajectory in Cartesian space - PBVS simulated for a static target in real-time model

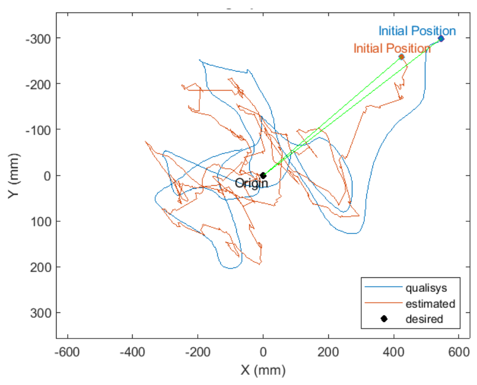
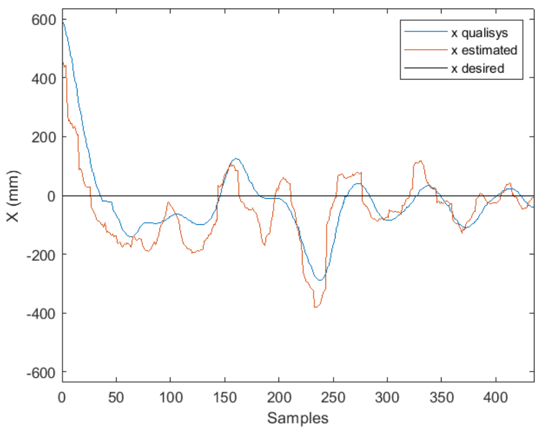


Figure 5.36: Target trajectory in Cartesian space - PBVS with error saturation simulated for a static target in real-time model

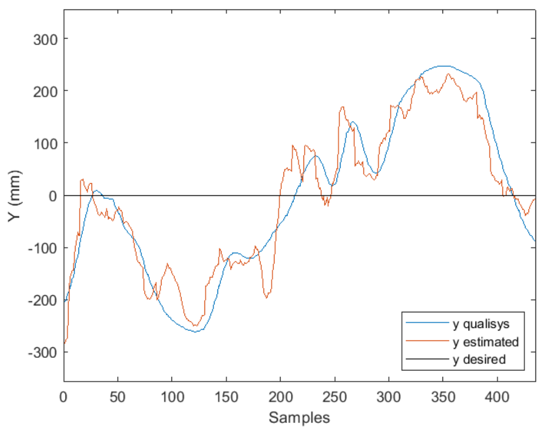
Also, by looking into the position error graphs in Figure 5.37, it is clear that there are several times where the error crosses zero or almost reach it for both cases.

Additionally, the mean of the errors' module in x and y direction, which are 89mm and 128mm for the simple case and 169mm and 97mm for the case where the error is saturated, show this trend since they are very close to zero. However, based on those values it is not clear if there is any advantage on applying an error saturation.

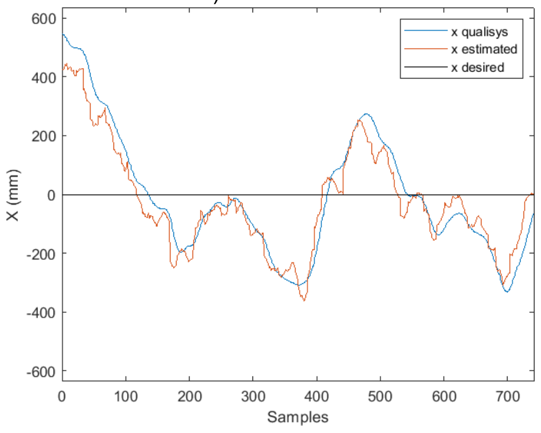
In order to understand it, several tests were performed in which the starting position of the drone is the same. From its statistics results on Tables 5.4 and 5.5, it was found that the average values of the mean error's modulus in x and y direction are 94mm and 115mm for the simple error and 152mm and 101mm for the saturated one. Plus, the average values of the maximum error's modulus in x and y direction are 297mm and 298mm for the first case mentioned before and 399mm and 243mm for the



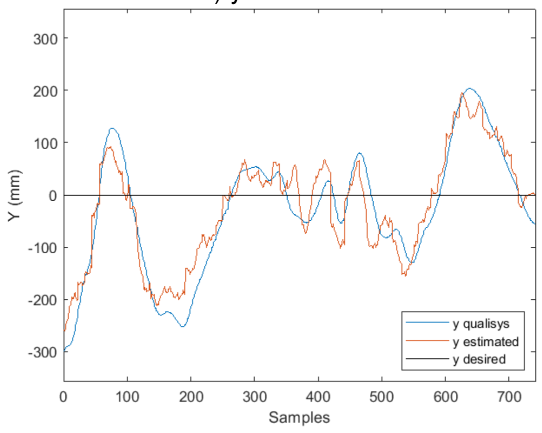
a) x coordinate



b) y coordinate



c) x coordinate - error saturation



d) y coordinate - error saturation

Figure 5.37: Error of x and y target 3D coordinates - PBVS simulated with and without saturation for static target in real-time model

second one. Although there is a small improvement on reducing the displacement of the drone with respect to the target in y direction using the saturation, it causes a higher increase of the error in the x direction.

Test No	x error mean (mm)	y error mean (mm)	x error max (mm)	y error max (mm)
1	57	79	141	220
2	70	114	275	353
3	143	141	394	335
4	102	122	312	309
5	99	121	361	272

Table 5.4: Statistics of x and y 3D target coordinates error signals - PBVS simulated for a static target in real-time model

Test No	x error mean (mm)	y error mean (mm)	x error max (mm)	y error max (mm)
1	164	145	431	228
2	204	91	543	285
3	123	72	310	160
4	143	102	319	275
5	126	97	394	266

Table 5.5: Statistics of x and y 3D target coordinates error signals - PBVS simulated with error saturation for a static target in real-time model

The effect of the saturation is also visible when comparing the magnitudes of the measured velocity signals in Figure 5.38. It could lower the maximum values in approximately 25% and 40% in x and y direction, respectively. Besides this, there are big discrepancies between the measured and the reference input velocities for both error approaches, which might be explained by the reasons mentioned in Section 5.2.1 for the IBVS case.

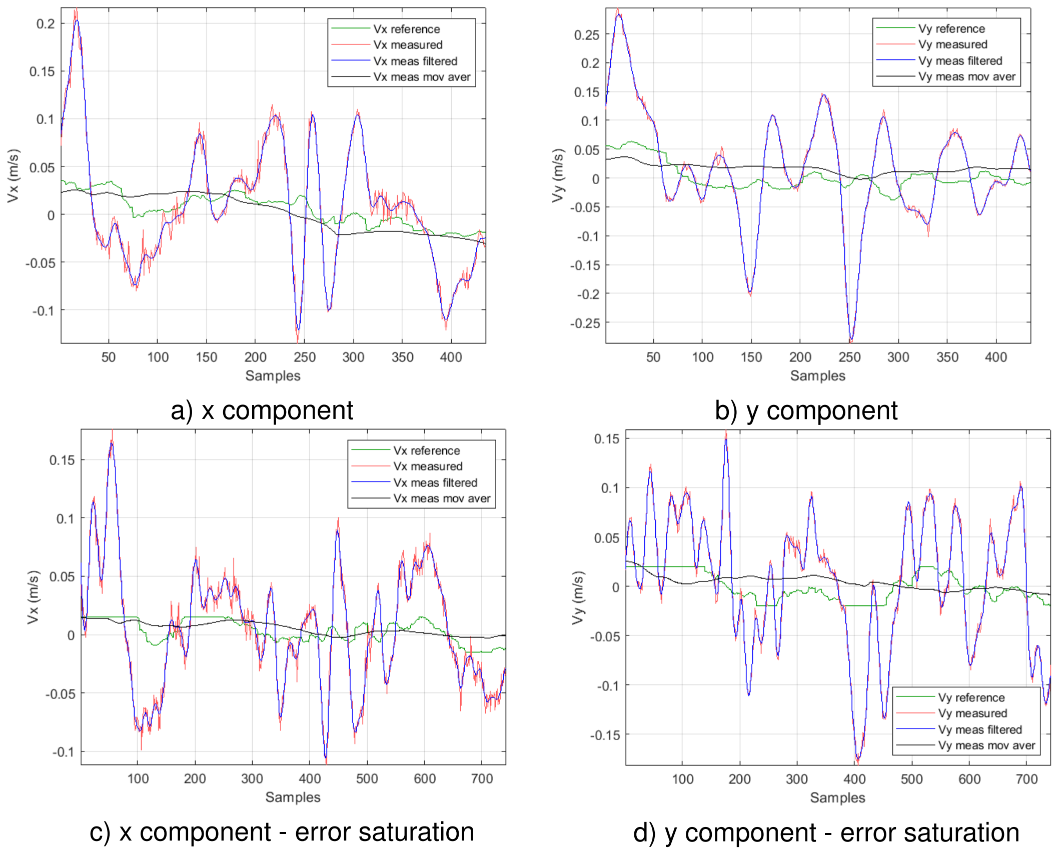


Figure 5.38: Reference and measured velocity x (a) and c)) and y (b) and d)) components relative to drone frame - PBVS simulated with and without saturation for static target in real-time model

Once again, the effect of the visual servo control is not clear based on the velocity graphs of Figure 5.38, since the reference inputs have a lower variation than the measured values and the trend of the measured velocities (black line) does not always follow the reference ones.

Moving Target

Simmilarly to the IBVS case, different target motion scenarios were tested: foward motion, i.e moving in the x direction of the drone's frame, lateral motion, i.e. moving in the y direction of the drone's frame, and diagonal motion, i.e. moving in both x and y directions of the drone's frame.

Considering an example of the first scenario, it is visible in Figure 5.39, where the position of the target with respect to camera frame is illustrated, that the drone cannot reach the target in order to align it with the camera's frame origin.

By looking at Figure 5.40 b), where the position error in the direction the target is moving is depicted, it is clear that the error fluctuation is deviated from zero, but not converges to a steady state, as expected.

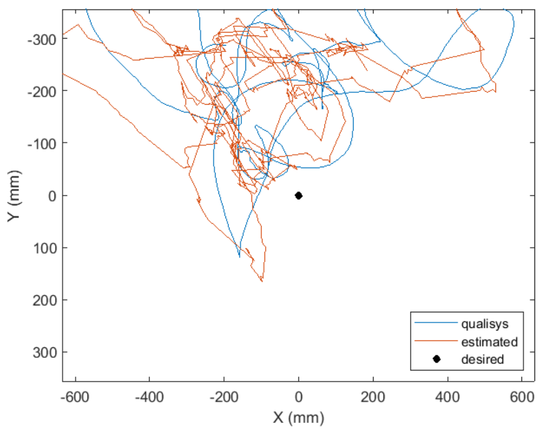


Figure 5.39: Target trajectory in Cartesian space - PBVS simulated for a target moving forward in real-time model

Plus, its values are almost all negative, which means the drone is most of the time behind the target. However, the mean value of the error, which is -239 mm for the estimated position and -194 mm for the real one, indicates that this deviation is small when compare to the length of the field of view in this direction, that is 713 mm. Hence, the drone could be able to track the target without this being out of the field of view.

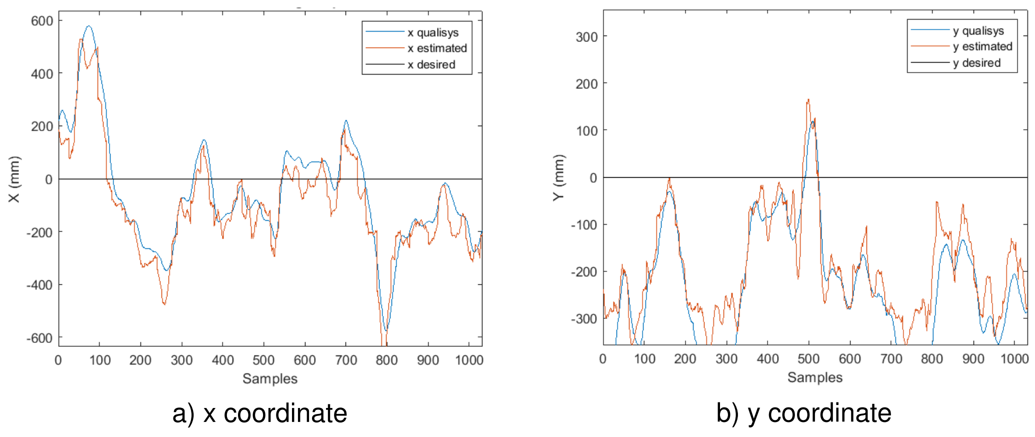
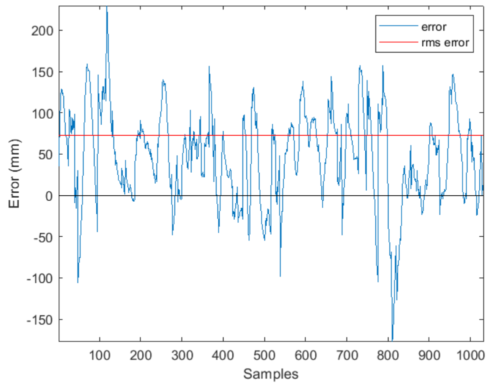


Figure 5.40: Error of x (a)) and y (b)) target 3D coordinates - PBVS simulated for a target moving forward in real-time model

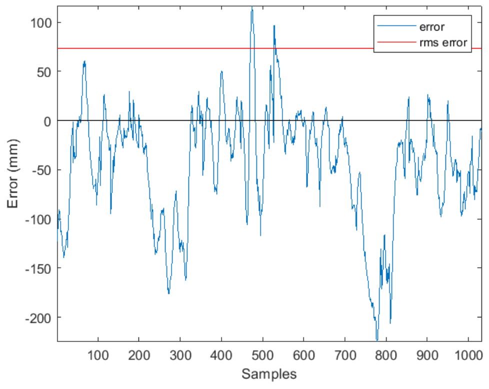
Even though the target is barely moving on the y direction of the drone's frame, the correspondent position error relative to the camera's frame, which is illustrated in the graph on the left side of Figure 5.40, shows large fluctuation peaks. Nevertheless, the mean value of the position error is -52mm for the estimated data, which is close to the one obtained for the real data, -97mm, and to zero.

By analysing the target position illustrated in Figure 5.39 it is not clear that the target gets out of the field of view, whose limits are [-633.2,633.2] mm in x direction and [-356.4,356.4] mm in y direction. But looking at Figure 5.40 is visible that the lowest value of the position error in x direction is lower than -633.2 mm and there are several times where the error in y position is lower than -356 mm.

Moreover, the error between the estimated position measurements and the Qualisys data depicted in Figure 5.41 is small, as its rms values are 0.073m and 0.074m for x and y direction, respectively. Hence, the estimations are considered a valid data set.



a) x coordinate



b) y coordinate

Figure 5.41: Error between estimations and QTM x (a)) and y (b)) target 3D coordinates - PBVS simulated for a target moving forward in real-time model

By plotting the Qualisys data for the drone and the target position in Figure 5.42, it is clear that even though the drone cannot reach the target, it can track it.

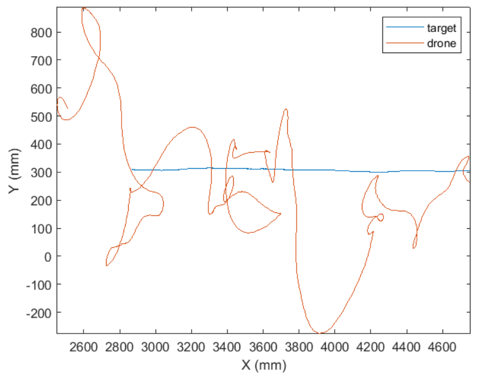


Figure 5.42: QTM data for target and drone trajectory in Cartesian space - IBVS simulated for a target moving forward in real-time model

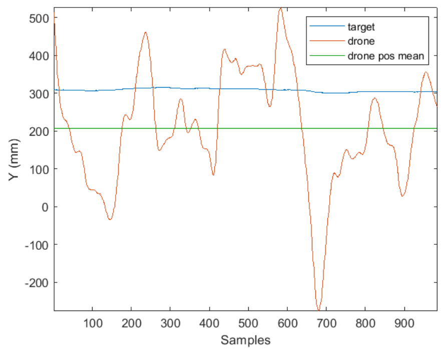


Figure 5.43: Error between QTM data for target and drone position in Cartesian space - PBVS simulated for a target moving forward in real-time model

Although there is a great variation of drone's position in y direction of Qualisys fixed frame (Figure 5.43), its mean value (207 mm) is close to the target's y position (≈300 mm) and the drone's x position follows the target one.

Take into consideration the second scenario illustrated in Figure 5.44, there is a smaller deviation of drone's position in the opposite direction to the one the target is taking compared to the previous case. As shown in Figure 5.45, the fluctuation is around 2959mm, which is near the target's x position (≈2930mm). Once again, the drone is still able to track the target in the y direction of drone's frame.

Finally, the third scenario shows a very good tracking performance. As illustrated in Figure 5.46, the drone keeps pace with the target.

Also for this scenario is visible the effect of the visual servo control. As seen in Figure 5.47, the moving average signal of the measured velocity follows the upward trend of the velocity reference in the x direction and the downward trend in the case of the y component of the velocity reference.

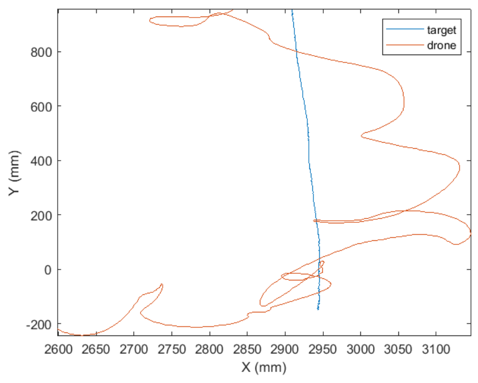


Figure 5.44: QTM data for target and drone trajectory in Cartesian space - PBVS simulated for a target moving laterally in real-time model

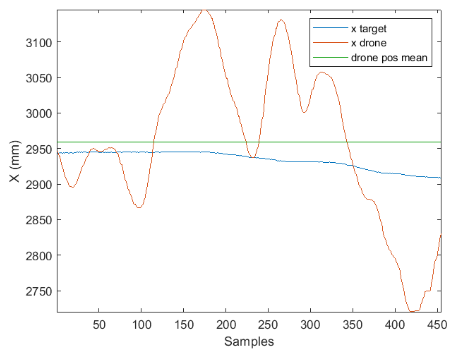


Figure 5.45: Error between QTM data for target and drone position in Cartesian space - PBVS simulated for a target moving laterally in real-time model

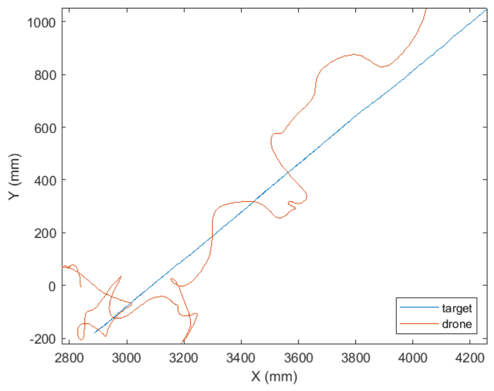
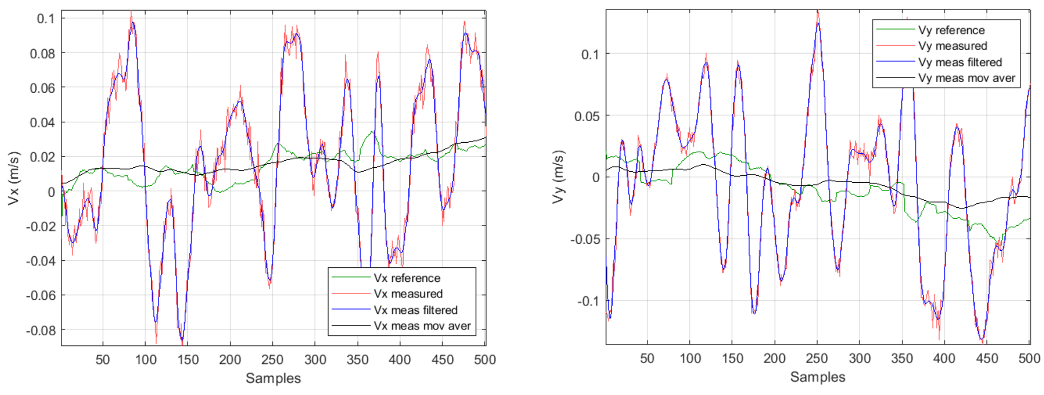


Figure 5.46: QTM data for target and drone trajectory in Cartesian space - PBVS simulated for a target moving diagonally in real-time model



a) x component

b) y component

Figure 5.47: Reference and measured velocity x (a)) and y (b)) components relative to drone frame - PBVS simulated for a target moving diagonally in real-time model

Chapter 6

Conclusions

In this project the implementation of visual servo control algorithms to Parrot AR Drone 2.0 was investigated. Both IBVS and PBVS are combined with PID control strategy to test target tracking scenarios. The drone was able to track a generic target either in a static or in movement condition. However, a time delay was found between the transmitting data protocols which compromises the visual servo control performance. There is a lag of 61 samples between the image and the other drone data acquisition, which introduces a time-delay on tracking the reference trajectory.

This increases the probability of the target being outside the camera's field of view, specially when it is moving. But the centroid tracking algorithm developed could address this problem in a way that the visual servo control could still work and the drone could still catch the target. Plus, this algorithm could prevent situations where there are faults in the image acquisition process.

Even though the visual servo control laws error does not converge to zero, as expected, the drone could make several attempts on reaching the target. The statistics showed that when the target is motionless, there are small deviations from zero error and there is no clear benefit on saturating the error. Additionally, the QTM position data reveals drone capacity on tracking the target moving in different directions, but also the high difficulty of the drone on keeping the same position in space, specially in the y direction of drone frame. This explained why a much better performance is achieved when the target is moving diagonally.

On the other hand, the tracking performance of the drone on following the velocity reference was found to be very unsatisfactory. The drone velocity could catch up the trend of the reference, but there are big discrepancies on their magnitude. Possible errors might have been introduced by the image-based algorithms used to estimate the horizontal velocities onboard the quadrotor, because they are not able to detect the coupling effect between its translational and rotational motion.

Based on the experimental results obtained, it was not possible to conclude about any advantage of one visual servo approach over the other, due to the time delay. But considering that the estimations of target's position are quite accurate when comparing to QTM measurements and looking into the simulator results, the PBVS approach seems more attractive to *Eye in the Sky* project applications. The main advantage is considered to be the easy conversion between Cartesian space coordinates and any geographic coordinate system.

Regarding future developments of the proposed solution for the *Eye in the Sky* project applications, the laboratory testing conditions could be improved, so more realistic results for the visual servoing implementation are acquired. The controller and image processing could be implemented in an on-board control unit to obtain a decentralized approach that could be tested in a open-field where constant communication with the UAV cannot be guaranteed. Also, this would avoid the time delay problem concerning the Wi-fi control. In case this is still an option, the video streaming protocol should be changed to UDP.

Moreover, the use of a rotating camera would be preferred so the underactuated degrees of freedom of the drone could be addressed on the visual servoing approach, at the expense of an increased complexity. This way a larger area in space could be monitored.

Finally, a more realistic scenario and a more complex environment could be created to test the visual servo control.

Bibliography

[1] U.S. Army. 'Eyes of the Army' U.S. Army Roadmap for UAS 2010-2035. *Federation Of American Scientists*, 2010. ISSN \URL-Online{http://www.acq.osd.mil/usd/Roadmap Final2.pdf}.

[2] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019. doi: 10.1109/ACCESS.2019.2909530.

[3] A. Kumar, K. Sharma, H. Singh, S. G. Naugriya, S. S. Gill, and R. Buyya. A drone-based networked system and methods for combating coronavirus disease (COVID-19) pandemic. *Future Generation Computer Systems*, 2021. ISSN 0167739X. doi: 10.1016/j.future.2020.08.046.

[4] W. Krüll, R. Tobera, I. Willms, H. Essen, and N. Von Wahl. Early forest fire detection and verification using optical smoke, gas and microwave sensors. In *Procedia Engineering*, 2012. doi: 10.1016/j.proeng.2012.08.208.

[5] D. Bohdanov. Quadrotor UAV Control for Vision-based Moving Target Tracking Task. *ProQuest Dissertations and Theses*, 2012.

[6] M. G. Popova. Visual Servoing for a Quadrotor UAV in Target Tracking Applications. Technical report, University of Toronto, 2015.

[7] E. Frew, T. McGee, Z. W. Kim, X. Xiao, S. Jackson, M. Morimoto, S. Rathinam, J. Padial, and R. Sengupta. Vision-based road-following using a small autonomous aircraft. In *IEEE Aerospace Conference Proceedings*, 2004. ISBN 0780381556. doi: 10.1109/AERO.2004.1368106.

[8] G. Flandin, F. Chaumette, and E. Marchand. Eye-in-hand/eye-to-hand cooperation for visual servoing. *Proceedings-IEEE International Conference on Robotics and Automation*, 2000. ISSN 10504729. doi: 10.1109/ROBOT.2000.846442.

[9] J. M. Daly, Yan Ma, and S. L. Waslander. Coordinated landing of a quadrotor on a skid-steered ground vehicle in the presence of time delays. 2011. doi: 10.1109/iros.2011.6094488.

[10] F. Chaumette and S. Hutchinson. Visual servo control. II. Advanced approaches [Tutorial]. *IEEE Robotics & Automation Magazine*, 14(1):109–118, mar 2007. ISSN 1070-9932. doi:

10.1109/MRA.2007.339609. URL https://hal.inria.fr/inria-00350638http://ieeexplore.ieee.org/document/4141039/.

[11] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, mar 2004. ISBN 9780521540513. doi: 10.1017/CBO9780511811685.

[12] P. I. Corke and S. A. Hutchinson. A new hybrid image-based visual servo control scheme. *Proceedings of the IEEE Conference on Decision and Control*, 2000. ISSN 01912216. doi: 10.1109/CDC.2000.914182.

[13] E. Malis, F. Chaumette, and S. Boudet. 2-1/2-D visual servoing. *IEEE Transactions on Robotics and Automation*, 1999. ISSN 1042296X. doi: 10.1109/70.760345.

[14] J. Chen, D. M. Dawson, W. E. Dixon, and A. Behal. Adaptive homography-based visual servo tracking for a fixed camera configuration with a camera-in-hand extension. *IEEE Transactions on Control Systems Technology*, 2005. ISSN 10636536. doi: 10.1109/TCST.2005.852150.

[15] N. P. Papanikolopoulos and P. K. Khosla. Adaptive Robotic Visual Tracking: Theory and Experiments. *IEEE Transactions on Automatic Control*, 1993. ISSN 15582523. doi: 10.1109/9.210141.

[16] F. Liu, C. Shen, G. Lin, and I. Reid. Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016. ISSN 01628828. doi: 10.1109/TPAMI.2015.2505283.

[17] W. Chen, Z. Fu, D. Yang, and J. Deng. Single-image depth perception in the wild. In *Advances in Neural Information Processing Systems*, 2016.

[18] D. Lee, H. Lim, H. J. Kim, Y. Kim, and K. J. Seong. Adaptive image-based visual servoing for an underactuated quadrotor system. *Journal of Guidance, Control, and Dynamics*, 2012. ISSN 15333884. doi: 10.2514/1.52169.

[19] B. P. Larouche and Z. H. Zhu. Position-based visual servoingin robotic capture of moving target enhanced by Kalman filter. *International Journal of Robotics and Automation*, 30(3):267–277, 2015. ISSN 08268185. doi: 10.2316/Journal.206.2015.3.206-4230.

[20] Y. Ma, J. Kosecka, S. Soatto, S. Sastry, and J. Ko. *An Invitation to 3-D Vision: From Images to Models*. 2006. ISBN 978-1-4419-1846-8. doi: 10.1007/978-0-387-21779-6.

[21] F. Chaumette and S. Hutchinson. Visual servo control. I. Basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, dec 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.250573. URL http://ieeexplore.ieee.org/document/4015997/.

[22] P.-J. Bristeau, F. Callou, D. Vissière, and N. Petit. The Navigation and Control technology inside the AR.Drone micro UAV. In *IFAC Proceedings Volumes*, volume 44, pages 1477–1484, jan 2011. doi: 10.3182/20110828-6-IT-1002.02327. URL https://linkinghub.elsevier.com/retrieve/pii/S1474667016438188.

[23] David Escobar Sanabria. AR Drone Simulink Development-Kit V1.1, 2020. URL https://www.mathworks.com/matlabcentral/fileexchange/43719-ar-drone-simulink-development-kit-v1-1.

[24] S. Piskorski, N. Brulez, P. Eline, and F. D'Haeyer. AR.Drone Developer Guide. page 133, 2012. URL https://jpchanson.github.io/ARdrone/ParrotDevGuide.pdf.

[25] Rui Moura Coelho. Real-time video stream capture on Simulink from Parrot ARDrone 2.0, 2019.

[26] P. Marmaroli, X. Falourd, and H. Lissek. A comparative study of time delay estimation techniques for road vehicle tracking. *11th French Congress of Acoustics and 2012 Annual IOA Meeting*, 2012.

[27] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart. Mav navigation through indoor corridors using optical flow. In *2010 IEEE International Conference on Robotics and Automation*, pages 3361–3368, 2010. doi: 10.1109/ROBOT.2010.5509777.

Appendix A

Centroid tracking algorithm

Algorithm 1: Centroid Tracking

```
Require: Centroid,CentroidSentPrev;
  Out = 0;
  TargetIsOut1 = 0;
  TargetIsOut2 = 0;
  OutPrev = Out;
  CentroidPrev = Centroid;
  while Image On = TRUE do
    OutPrev = Out;
    CentroidPrev = Centroid;
    if Centroid = 0 then
      Out = 1
    else
      Out = 0
    end if;
    TargetIsOut1Prev = TargetIsOut1;
    TargetIsOut2Prev = TargetIsOut2;
    if $\|$Centroid $-$ CentroidPrev$\|$ $>$X and Out = 0 and TargetIsOut1Prev = 0 and TargetIsOut2Prev =
    0 then
      TargetIsOut1 = 1;
      TargetIsOut2 = 0;
    else if $\|$Centroid $-$ CentroidSentPrev$\|$ $>$X and TargetIsOut1Prev = 1 then
      TargetIsOut1 = 1;
      TargetIsOut2 = 1;
    else if Out = 1 and OutPrev = 0 and TargetIsOut1Prev = 0 then
      TargetIsOut1 = 1;
      TargetIsOut2 = 0;
    else if (Out = 1 and OutPrev = 1) or (Out = 1 and OutPrev = 0 and TargetIsOut1Prev = 1) then
      TargetIsOut1 = 1;
      TargetIsOut2 = 1;
    else
      TargetIsOut1 = 0;
      TargetIsOut2 = 0;
    end if
    if TargetIsOut1 = 1 and TargetIsOut2 = 1 then
      CentroidSent = CentroidSentPrev;
    else if TargetIsOut1 = 1 and TargetIsOut2 = 0 then
      CentroidSent = CentroidPrev;
    else
      CentroidSent = Centroid;
    end if
    return CentroidSent
  end while
```