TECHNISCHE UNIVERSITÄT DRESDEN

IFA LS-AT

Fakultät Elektrotechnik und Informationstechnik Institut für Automatisierungstechnik

Bachelorarbeit

zum Thema

Image Based Visual Servoing for Aerial Robot

vorgelegt von Pablo Rodríguez Robles
im Studiengang Luft- und Raumfahrt, Jg. 2014
geboren am 28.02.1996 in León, Spain

Betreuer: Dipl.-Ing. Chao Yao
Verantwortlicher Hochschullehrer: Prof. Dr. techn. Klaus Janschek
Tag der Einreichung: 27.04.2018

Aufgabenstellung

Test der PDF-Integration

Achtung

Auch wenn die Möglichkeit besteht, die eingescannte Aufgabenstellung als PDF zu integrieren, muss in **einem einzureichendem Exemplar** die Aufgabenstellung **im Original** eingebunden werden.

TECHNISCHE UNIVERSITÄT DRESDEN

IFA LS-AT

Fakultät Elektrotechnik und Informationstechnik Institut für Automatisierungstechnik

Image Based Visual Servoing for Aerial Robot

Hier muss der Text für die deutsche Kurzfassung inklusive eines aussagekräftigen Bildes eingefügt werden.

Betreuer:	Dipl.-Ing. Chao Yao
Hochschullehrer:	Prof. Dr. techn. Klaus Janschek
Tag der Einreichung:	27.04.2018

BACHELORARBEIT Bearbeiter: Pablo Rodríguez Robles

TECHNISCHE UNIVERSITÄT DRESDEN

IFA LS-AT

Fakultät Elektrotechnik und Informationstechnik Institut für Automatisierungstechnik

Image Based Visual Servoing for Aerial Robot

Tutor:	Dipl.-Ing. Chao Yao
Supervisor:	Prof. Dr. techn. Klaus Janschek
Day of Submission:	27.04.2018

STUDENT RESEARCH THESIS

Author: Pablo Rodríguez Robles

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

Abkürzungen

CV	Computer Vision
DD	Data Dictionary
DFD	Data Flow Diagram
DOF	Degree of Freedom
GAS	Global Asymptotic Stability
GPS	Global Positioning System
IBVS	Image Based Visual Servoing
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
OS	Operative System
PBVS	Position Based Visual Servoing
PD	Proportional Derivative
ROS	Robot Operative System
ROS	Structure Analysis
SRS	Software Requirement Specification
SVS	Self Visual Servoing
TUD	Technische Universität Dresden
UAV	Unmanned Aerial Vehicle
VS	Visual Servoing

1 Introduction

1.1 Motivation and Background

During the last decade, the use of *Unmanned Aerial Vehicles* (UAVs) have spread among very different applications. Flying robots can be very helpful to improve the way some tasks are already achieved by terrestrial platforms. For example, object transportation, environment mapping or surveillance. At the Institute of Automation Engineering[1] of the Technical University of Dresden, a new robotic platform is being developed in cooperation with the Institute of Solid Mechanics[2] to investigate the use of flying robots in aerial manipulation.

When dealing with manipulation of objects, it is desired that the aerial robot adopts a certain pose with respect to the target before the manipulation process really starts. The present work deals with the development of a *Visual Servoing* (VS) control system that helps an aerial robot to acquire the desired pose by means of image data.

A monocular monochrome camera is planed to be the only available on board sensor. For the controller proposed the feedback is directly computed from image features rather than estimating the robot's pose and using the pose errors as control input.

Vision results to be a passive (in contrast to GPS) and cheap sensor (in contrast to LIDAR). Visual odometry performs well navigating in GPS denied environments like indoors, but its not appropriated to regulate the relative navigation of the vehicle with respect to a target.

In order to integrate the visual servoing algorithm into the future modular robot system, the algorithm has been designed and tested on a under-actuated conventional quadrotor. The aerial robot is implemented within the ROS framework, where the visual servoing controller developed for this thesis is also integrated. Instead of using real hardware the complete system is simulated using the Gazebo robotics simulator.

[1] Technische Universität Dresden. Institut für Automatisierungstechnik. Dresden, Germany
[2] Technische Universität Dresden. Institut für Festkörpermechanik. Dresden, Germany

1.2 Aims and Objectives

The aim of this work is to implement and test a VS control algorithm for a quadrotor, which could be later used by the Flypulator project. This includes the review of the state of the art with regard to Visual Servoing, the design of a solution and a prototypical implementation with in the ROS framework and simulation with Gazebo of a test case.

The present thesis documents comprehensively the theoretical background, implementation details and results of the conducted work through the following structure. In Chapter 2 the theoretical background and state of the art of Visual Servoing is presented. Chapter 3 contains the *Software Requirements Specification* of the system developed as well as its decomposition by *Structured Analysis* method. Chapter 4 describes the solution developed and the algorithms to be tested. Chapter 5 deals with the system implementation, testing and validation. Finally, Chapter 6 contains the system validation experiments and Chapter **??** presents the conclusions of the project and suggests future improvements and research paths.

2 Theoretical Background and State of the Art

2.1 Reference Frames

Throughout this work, different frames of reference are used to indicate the pose of the robot or the position of some point in space or in the image. In this section, the different reference frames that will be used in the rest of the text are presented.

- *Inertial frame* (I). Coincident with the Gazebo's `world` frame.
- *Body frame* (B). Coincident with the robot's URDF description `base_link` frame. The x-axis pointing in the frontal direction of the robot and the z-axis pointing upwards.
- *Camera frame* (C). Centered in the camera center with the x-axis pointing right in the image and the y-axis pointing downwards in the image. As a result, the z-axis points where the camera is pointing, in this case downwards. Coincident with the `downward_cam_optical_frame` frame.
- *Image plane* (P). Corresponds to the perspective projection of a physical point in the camera sensor following the pinhole camera model. The origin of this frame is in the top-left corner of the image, the x-axis points right and the y-axis points downwards. See also Figure 2.2
- *Tag frame* (T). Centered in the tag center with the x-axis pointing upwards in the image and the y-axis pointing right in the image. See also Figure 5.3.
- *Object frame* (O). Centered in the tag center with$x$-axis pointing right in the image and the y-axis pointing downwards in the image.

Figure 2.1: Reference frames used in this work

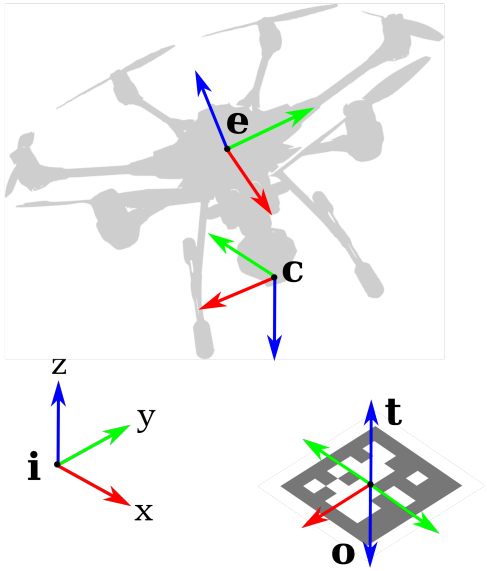
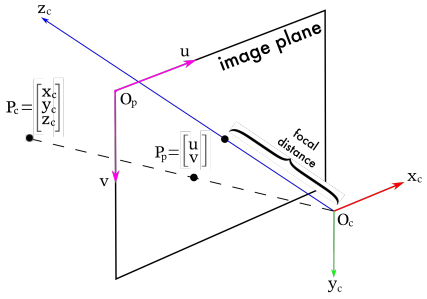


Figure 2.2: Camera image plane reference frame



2.2 Visual Servoing Theoretical Basics

In this section the theoretical background of visual servo controllers is briefly discussed. The different parts of a Visual Servoing scheme are presented and the basic strategy is illustrated. It is usual in the literature to take [7] and [8] as the main reference when it comes to the theoretical setup of the discipline. As a result, the following description is based on these popular sources[1].

Visual Servoing is defined in the literature as the use of computer vision data to control the motion of a robot. The image data comes from a camera, which can observe the robot fixed in the space or moving with the robot. The latter approach is know as eye-in-hand Visual Servoing and is the selected one for the case of this work.

Visual servo controllers accomplish their task of reaching a certain pose by trying to minimize the following error $\boldsymbol{e}(t)$

$$\boldsymbol{e}(t) = \boldsymbol{s}(\boldsymbol{m}(t), \boldsymbol{a}) - \boldsymbol{s}^* \tag{2.1}$$

Here, $\boldsymbol{m}(t)$ is a set of image measurements (e.g. the image coordinates of the interest points or the image centroid of an object), that is, information computed from the image data. With the help of these measurements a vector of k visual features, $\boldsymbol{s}(\boldsymbol{m}(t), \boldsymbol{a})$ is obtained, in which $\boldsymbol{a}$ is a vector containing different camera parameters. In contrast, $\boldsymbol{s}^*$ defines a set of desired features.

For the present case, where the target is not moving, $\boldsymbol{s}^*$ and the changes in $\boldsymbol{s}$ depend only on the camera motion.

There exist two main variants of Visual Servoing depending on how the features vector $\boldsymbol{s}$ is defined. On the one hand, Image Based Visual Servoing (IBVS) takes as $\boldsymbol{s}$ a set of features already available within the image data. It can be seen as a control of the features in the image plan such that moving the features to a goal configuration implicitly results in the task being accomplished (see [11]). On the other hand, Position Based Visual Servoing (PBVS) considers for $\boldsymbol{s}$ a set of 3D parameters that must be estimated from the image data. Once the parameters are available, pose estimation is conducted and the Visual Servoing task results in a cartesian motion planing problem.

Using the PBVS approach leads to the necessity of camera calibration and

[1] The interested reader should visit the Lagadic research group home page (http://www.irisa.fr/lagadic), pioneers in the area.

estimation of the flying robot pose, these are two big disadvantages for the application intended in this work. On the other side, IBVS needs no camera calibration and allows the robot to achieve the pose desired without any pose estimation process. Resulting in a convenient method for cheap systems.

A simple velocity controller can be arranged in the following way. Let $\boldsymbol{v}_c = (v_c, \boldsymbol{\omega}_c)$ be the spatial velocity of the camera, with v_c the instantaneous linear velocity of the origin of the camera frame and $\boldsymbol{\omega}_c$ the instantaneous angular velocity of the camera frame, as a result the temporal variation of the features can be expressed as

$$\dot{\boldsymbol{s}} = \boldsymbol{L}_{\boldsymbol{s}} \boldsymbol{v}_c \tag{2.2}$$

Where $\boldsymbol{L}_{\boldsymbol{s}} \in \mathbb{R}^{k \times 6}$, the feature Jacobian, acts as interaction matrix relating the camera velocity and the change in the visual features.

The time variation of the error to be minimized can be obtained by combining 2.1 and 2.2

$$\dot{\boldsymbol{e}} = \boldsymbol{L}_{\boldsymbol{e}} \boldsymbol{v}_c \tag{2.3}$$

with $\boldsymbol{L}_{\boldsymbol{e}} = \boldsymbol{L}_{\boldsymbol{s}}$. The input for such a controller is the camera velocity $\boldsymbol{v}_c$, which, using 2.3, can be set in such a way that an exponential decrease of the error is imposed (i.e. $\dot{\boldsymbol{e}} = -\lambda \boldsymbol{e}$)

$$\boldsymbol{v}_c = -\lambda \boldsymbol{L}_{\boldsymbol{e}}^{+} \boldsymbol{e} \tag{2.4}$$

Here, $\boldsymbol{L}_{\boldsymbol{e}}^{+} \in \mathbb{R}^{k \times 6}$ is the Moore-Penrose pseudoinverse of $\boldsymbol{L}_{\boldsymbol{e}}$. It is computed as $\boldsymbol{L}_{\boldsymbol{e}}^{+} = (\boldsymbol{L}_{\boldsymbol{e}}^T \boldsymbol{L}_{\boldsymbol{e}})^{-1} \boldsymbol{L}_{\boldsymbol{e}}^T$, provided that $\boldsymbol{L}_{\boldsymbol{e}}$ is of full rank 6. Imposing this condition leads to $\|\dot{\boldsymbol{e}} - \lambda \boldsymbol{L}_{\boldsymbol{e}}^T \boldsymbol{L}_{\boldsymbol{e}} \boldsymbol{e}\|$ and $\|\boldsymbol{v}_c\|$ being minimal. Note that for the special case of $k = 6$, if $\boldsymbol{L}_{\boldsymbol{e}}$ is nonsingular, it is possible to obtain a simpler expression using the matrix inversion $\boldsymbol{v}_c = -\lambda \boldsymbol{L}_{\boldsymbol{e}}^{-1} \boldsymbol{e}$.

When implementing real systems it is not possible to know perfectly either $\boldsymbol{L}_{\boldsymbol{e}}$ or $\boldsymbol{L}_{\boldsymbol{e}}^{+}$. Thus, an approximation of these two matrices is introduced, noted with the symbol $\widehat{\boldsymbol{L}_{\boldsymbol{e}}}$ for the approximation of the error interaction matrix and $\widehat{\boldsymbol{L}_{\boldsymbol{e}}^{+}}$ for the approximation of the pseudoinverse of the interaction matrix. Inserting

this notation in the control law we obtain

$$\boldsymbol{v}_c = -\lambda \widehat{\boldsymbol{L}_{\boldsymbol{e}}^{+}} \boldsymbol{e} \tag{2.5}$$

Once the basic appearance of a visual servo controller has being presented, the goal is to ask the following questions: How should $\boldsymbol{s}$ be chosen? What is the form of $\boldsymbol{L}_{\boldsymbol{s}}$? How should we estimate $\widehat{\boldsymbol{L}_{\boldsymbol{e}}^{+}}$?

In the simplest approach, the vector $\boldsymbol{s}$ is selected as a set of image-plane points, where $\boldsymbol{m}$ are the set of coordinates of these image points and $\boldsymbol{a}$ the camera intrinsic parameters. Later in this work, a more complex definition for the image features vector $\boldsymbol{s}$ will be used.

Concerning the interaction matrix $\boldsymbol{L}_{\boldsymbol{s}}$

The Interaction Matrix

The interaction matrix, which relates the camera velocity to the change of the visual features, is strictly related to the the camera model. A camera model describes the correspondence between objects in 3D space and their appearance in a 2D image. Here, the pinhole camera model is used.

The camera image capture is a procedure which projects a 3D point from its coordinates in the camera frame, $\boldsymbol{X} = (X, Y, Z)$, to a 2D image point with coordinates $\boldsymbol{x} = (x, y)$. From this geometry we have

$$\begin{cases} x = X/Z = (u - c_u)/f\alpha \\ y = Y/Z = (v - c_v)/f \end{cases} \tag{2.6}$$

where $\boldsymbol{m} = (u, v)$ gives the coordinates of the image point in pixel units, and $\boldsymbol{a} = (c_u, c_v, f, \alpha)$ is the set of camera intrinsic parameters: c_u and c_v are the coordinates of the principal point, f is the focal length, and α is the ratio of the pixel dimensions. For a feature point: $\boldsymbol{s} = \boldsymbol{x} = (x, y)$.

Taking the time derivative of the projection Equation 2.6, we obtain

$$\begin{cases} \dot{x} = \dot{X}/Z - X\dot{Z}/Z^2 = (\dot{X} - x\dot{Z})/Z \\ \dot{y} = \dot{Y}/Z - Y\dot{Z}/Z^2 = (\dot{X} - y\dot{Z})/Z \end{cases} \tag{2.7}$$

The velocity of the 3D point can be related to the spatial velocity of the camera using the equation for the velocity in a non-inertial reference frame

$$\dot{\boldsymbol{X}} = -\boldsymbol{v}_c - \omega_c \times \boldsymbol{X} \Leftrightarrow \begin{cases} \dot{X} = -v_x - \omega_y Z + \omega_z Y \\ \dot{Y} = -v_y - \omega_z X + \omega_x Z \\ \dot{Z} = -v_z - \omega_x Y + \omega_y X \end{cases} \tag{2.8}$$

Introducing 2.8 in 2.7 and grouping terms, it can be written

$$\begin{cases} \dot{x} = -v_x/Z + xv_z/Z + xy\omega_z - (1+x^2)\omega_y + y\omega_z \\ \dot{y} = -v_y/Z + yv_z/Z + xy\omega_z - (1+y^2)\omega_x + x\omega_z \end{cases} \tag{2.9}$$

using matrix notation

$$\dot{\boldsymbol{x}} = \boldsymbol{L_x} \boldsymbol{v}_c \tag{2.10}$$

where the interaction matrix that relates the camera velocity $\boldsymbol{v}_c$ to the velocity of the image point $\dot{\boldsymbol{x}}$ is

$$\boldsymbol{L_x} = \begin{bmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix} \tag{2.11}$$

In Equation 2.11, the value Z corresponds to the depth of the point relative to the camera frame. As a result, any Visual Servoing scheme using this form of the interaction matrix must provide an estimation of this value. Furthermore, the camera intrinsic parameters are necessary to compute x and y. Therefore, it is not possible to use directly $\boldsymbol{L_x}$, but an approximation $\widehat{\boldsymbol{L_x}}$ is to be used.

Approximation of the Interaction Matrix

When the current depth Z of each point is known, there is no need of approximation and $\widehat{\boldsymbol{L_e}^+} = \boldsymbol{L_e}^+$ for $\boldsymbol{L_e} = \boldsymbol{L_x}$ can be used. However, this approach requires the estimation of Z for all iterations of the scheme control (see [15]), which may be conducted by means of pose estimation methods.

A second alternative is to use $\widehat{\boldsymbol{L}_{\boldsymbol{e}}^{+}} = \boldsymbol{L}_{\boldsymbol{e}^*}^{+}$, where $\boldsymbol{L}_{\boldsymbol{e}^*}$ is the value of $\boldsymbol{L}_{\boldsymbol{e}}$ for the desired position ($\boldsymbol{e} = \boldsymbol{e}^* = 0$) (see [11]). Here, the depth parameter only needs to be estimated once for every point.

2.2.1 Image Moments as Visual Features

As in other computer vision problems, the choice of visual features has a strong influence on the performance of an visual servo controller. An image feature is a piece of information extracted from an image and may be helpful to solve a certain task. Depending on the problem, different kind of features can be interesting for the user. These features are a specific structure in the image such as points, edges or closed contours, which are detected by means of operations on their pixel neighborhood.

Usually, the desired characteristics [39] for a visual feature are: locality (robust to occlusion and cluttering), invariance to transformations such as rotation or scaling, robustness against noise or compression and efficiency (so they can be computed in real-time). In addition to that, for a visual servoing task it is convenient that the set of features is decoupled, so it can be used to control independently all the degrees of freedom of the robot.

Image moments are the result of a particular weighted average of the pixels' intensities of an image or a combination of various of these moments. They are useful to describe an object after segmentation, and have been used in computer vision for patter-recognition tasks [14].

Before the introduction of image moments, other visual features such as points from corner detectors, Fourier descriptors or light intensity where used to carry visual servoing tasks. Image moments result of particular interest for planar objects when the desired configuration is such that the camera plane and the object are parallel. They are easy to compute, robust to noise and its interaction matrix can be computed for any case thanks to the analytical method presented in [9]. The inclusion of spherical image moments as visual features has the advantage that they are invariant under rotation of the camera frame [4].

Image moments have also the advantage of being intuitive for simple cases. The simplest example of image moments are the area and centroid coordinates of an object.

For a greyscale image with pixel intensities $I(x, y)$, the raw image moments m_{ij} are computed as

$$m_{ij} = \sum_{k=1}^{n} x_k^i y_k^j \tag{2.12}$$

While entered moments are given by

$$\mu_{ij} = \sum_{k=1}^{n} (x_k - x_g)^i (y_k - y_g)^j \tag{2.13}$$

Where for the area $a = m_{00}$ and for the centroid coordinates $\{\bar{x_g}, \bar{y_g}\} = \{m_{10}/m_{00}, m_{01}/m_{00}\}$. Any moment defined in this way, is known to be invariant to 2D translational motion.

In this work, the normalized area and normalized centroid coordinates of a planar target are used as visual features.

2.2.2 Robot Operating System (ROS)

The Robot Operating System (ROS)[2] is an open-source, operating system for robots. It provides all the capabilities expected from any operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes and package management [35].

ROS works as a peer-to-peer network of processes, called *nodes*, connected through the ROS communication infrastructure. ROS implements several communication interfaces such as *services*, *topics* and a *Parameter Server*.

As a result, ROS works as a framework where the user can implement the desired robotic system modularly as a graph connecting several independent processes. The main ROS concepts are [34]:

- *Nodes*: Nodes are processes that perform a certain computation. The robotic system is decomposed in functionalities running in each of the in different nodes. For example, one node processes de camera information, other node controls the wheel motors and a third node is responsible of the path planning.

[2] www.ros.org

- *Master*: The ROS Master builds and maintains the name registration and lookup of the rest of components of the computation graph, so it allows the identification between nodes to establish the desired communication.
- *Messages*: Messages are the data transfered between nodes. A message is a data structure comprising typed fields. The standard primitive types (integer, float point, boolean, etc.) can be combined in nested structures or arrays.
- *Topics*: Topics are an asynchronous streaming of data between nodes following a publish/subscribe semantic. For example, data coming from a sensor. A node can send out a message through a topic publishing into it and an interested node can receive the message by subscribing the topic. A topic can use only a unique type of message. Topics work as an abstraction layer to separate the data production from the data manipulation using different nodes that communicate between each other without depending one on the other.
- *Services*: A constant stream of data is not always the desired communication case. It is also possible to desire a request/reply interaction between to nodes. For example to start the motors of a robot. This kind of communication is conducted using services. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply.
- *Parameter Server*: The Parameter Server allows data to be stored in a centralized data structure similar to a dictionary, where it can be accessed by any node.

In the present work, the system designed works as a set of nodes establishing communication between the *action interface* described in the Section 5.4, which subscribe to the sensor topics of the robot and publish control command to the command topic of the robot.

2.3 State of the Art

2.3.1 Visual Servoing for Aerial Robots

In this section, the literature on the use of Visual Servoing to control de translation and yaw motion of aerial robots is analyzed. The focus of this review is on the IBVS approach, where image feature errors between the current and target image are mapped to actuator inputs through the inverse of the Jacobian matrix. The different approaches presented complete the basic IBVS formulation with alternatives on feature selection, Jacobian matrix construction and different control strategies to make the error converge to zero, and with it, the relative pose between UAV and target object will converge to the desired one.

The task is to move the camera attached quadrotor to match the observed image features with the predefined desired image features obtained from a stationary object.

In [6], the visual servo controller presented is very simple, using feature points and a simple proportional law to control the kinematics of the robot. The VS technique is not explained in detail since it uses the MATLAB Visual Servoing Toolbox[3]. However, the paper is a good reference when it comes to the system description of the most simple approach to follow for the control of a quadrotor using visual servoing: The desired velocities on the image plane are computed with the visual information, transformed to the robot body frame and later to motor speeds that provide the robot with this motion. In this case, a PD controller is used to make the robot follow the chosen velocities.

The Visual Servoing research was originally developed for serial manipulators. In order to translate these techniques to aerial robots there are different aspects that may be considered, derived of the fact that a quadrotor is an under-actuated system:

- Feature depth estimation.
- Dynamics of the system leading to a high coupling between camera motion and target (visual feature) motion on the image plane.

To cope with these problems different solutions have been proposed and

[3] https://sourceforge.net/projects/vstoolbox/

applied for the usual configuration of a quadrotor helicopter, with the camera mounted on its platform (eye-in-hand configuration).

In the past, several approaches have been taken to face the problem of depth estimation, such as: partial pose estimation (see [23]), adaptive control (see [31]) or estimation of the Jacobian using quasi-Newton methods (see [32]). Nowadays, the most frequent approach is the use of a partitioned control that allows the uncoupling of translational and rotational degrees of freedom[4].

[13] is one of the pioneering works to cope with the visual servo control of under-actuated systems as a quadrotor. Introduces the idea of controlling the full dynamics of the vehicle. The approach requires separate measurements of linear and angular velocities which are not needed in classical IBVS, since for this case the kinematic velocities are used directly as control variables. In particular the method presented, needs only a single inertial direction. For the case of quadrotors, which usually require hovering applications, this is the vertical axis acceleration, provided by a filtered IMU signal. Due to the rotational ego-motion of the camera, the dynamics of the image point involve the angular velocity as well as the velocity linear velocity of the vehicle. This dependence destroys the explicit triangular cascade structure of rigid body motion expressed in the inertial frame. However, under certain conditions on the camera geometry, it is possible to recover a passivity-like property (from virtual input to the ping error) sufficient to apply a backstepping control design for the non-linear dynamical system. The paper the formulates that only image geometry that preserves the passivity-like properties of the body fixed frame dynamics of a rigid object in the image space are those of a spherical camera. Thus, introducing the use of spherical image moments[5].

For serial manipulators, a low-level actuator is usually employed to compensate the dynamic behavior of the system, making possible to control it with velocity commands as a first order system. For a full-dynamics quadrotor model accounting for aggressive maneuvers this is not at all the case, since it is a fourth order system. Thus, other strategies are usually considered to

[4]However, it is important to bear in mind that due to the under-actuated dynamics of an usual 4 DOF quadrotor, it is only possible to control the three linear velocities and the yaw (rotational) speed. While the other two rotational speeds, roll and pith, are used to move the thrust vectors of the propellers and thus generate movement on the horizontal plane.

[5]To use spherical moments is not necessary to implement physically a spherical camera, just to compute numerically the spherical moments of the image.

control such a robot (see [27] for a comprehensive description of the control of the dynamics of a quadrotor and its trajectory generation in space). Already in the basic literature on Visual Servoing (see [8]) the authors suggest that kinematic control may not be enough for the case of aggressive maneuvers.

Image moments are invariant to some transformations like scale, 2D translation or 2D rotation. This has been intensively used in pattern recognition. Introducing visual features based on image moments allows to design a decoupled control scheme when the object is parallel to the image plane (see [40]). Later, it is possible to generalize this property for the case of a non-parallel position with respect to the image plane.

In the work [12], a IBVS controller for the full dynamics of a quadrotor system around the hovering position is proposed. The biggest difficulty to develop dynamic controllers based on IBVS is the high coupling in the image Jacobian between the rotational and translational dynamics. Something that does not happen in the case of PBVS. The paper introduces a new visual error formulation to improve the conditioning of the Jacobian matrix, following the idea of [13]. Image moment features are augmented with the information provided by the IMU and used in conjunction with a non-linear controller.

Common strategy is to use to different loops to separate the control problem. The inner attitude loop runs at high gain using IMU measurements, while the outer loop runs at low gain with the visual input of the camera taking care of translation. The visual servo controller (outer loop) provides the attitude control (inner loop) the desired targets and outer loop ensures the stability of the system. An advantage of this approach is the possibility of reuse the IBVS scheme in other platforms since the low-level control of the specific material equipment of the aerial robot is not involved in it.

The paper [3] focuses in the design of kinematic controllers. The use of zero and first-order image moments as visual features is considered inappropriate for aggressive maneuvers due to its lack of robustness for this case, where Global Asymptotic Stability (GAS) cannot be guaranteed. For this reason, first-order spherical moments are introduced. Different control schemes are proposed in the paper. First, one controller using perspective projections is able of regulating the translation of the system thanks to the decoupled relationship between image and task space obtained. For the case of the camera image plane being parallel to the target plane, it is possible to control the translation of the robot independently of its rotation in a way equivalent to PBVS but without any pose estimation involved. The target depth is introduced as an initial data,

so no depth estimation is performed. The strategy is easy to implement and provides a good result within the assumed geometrical limitation.

To overcome the limitations of the simple model presented, the work introduces order control schemes that include spherical projections as visual features and non-linear control techniques that improve the behavior for the case of aggressive maneuvers, although still around the hovering position.

In [5], present two kinematic controllers, one based on points as visual features for IBVS and other implementing a Hybrid Visual Servoing scheme that requires partial pose estimation.

[17] designs a controller for the dynamics of a quadrotor within the IBVS approach. Although spherical moments are able to guarantee GAS, they generate trajectories that are not adequate when transforming to Cartesian coordinates. In this work, the method presented allows creating trajectories that are not only convenient in the image plane (i.e. where a usual IBVS controller is considered) but also in the Cartesian space. A projection of the perspective image moment features. augmented with inertial information to better control the dynamics, on a virtual plane dependent on the pitch and roll of the robot is proposed. Thus, the paper extends the approach followed in [3], here perspective projection was considered useful for a flight perpendicular to the target, but replaced by spherical moments to provide the GAS that it was not able to provide. In this way, a decoupled linear link between image and Cartesian coordinates is obtained using perspective image moments for the case in which the robot does not navigate perpendicular to the target. Finally a non-linear controller is derived for the system.

The authors in [2], propose two improvements to the work presented in the previous paragraph. For most of the controllers in the literature the camera velocity is required. This comes usually from IMU estimations, but due to the usual lack of GPS of the robots the noisy measurements of the IMU may not be enough. The paper proposed a visual flow and non-linear observer strategy to overcome this inconvenience. Secondly, in the standard formulation of the Jacobian matrix a estimation of the visual feature depth is necessary, something which is expensive to include in the controller. This is usually solved tanks to the use of image moments and a predefined target depth. The authors include here a controller to deal with this uncertainty.

2.3.2 Visual Servoing for Aerial Manipulators

In this section a perspective of Visual Servoing applied to aerial manipulators is presented. The main available literature is analyzed and the common approaches followed in it highlighted. An the end of the section, Table 2.1 summarizes the strategies commented with regard to the main characteristics of each method.

Some publications related to the University of Pennsylvania GRASP Laboratory[6] (see [41] and [42]) have studied the vision-based localization and servoing of quadrotors in grasping and perching tasks. However, the emphasis of these publications lays on the generation of dynamically-feasible trajectories in the image space, thus second order system control is performed instead of the most common kinematic control strategy. These vehicles are not manipulators in the sense of the rest of the approaches presented in this section, since the main task here is hanging from structures and grasping targets by means of aggressive, thus dynamical, maneuvers. In addition to that, the actuator used is not a high-DOF serial manipulator, but a 1 DOF gripper.

In the last years, a concrete aerial manipulator architecture has been popularized, for example in the context of the European projects ARCAS[7] and AEROARMS[8]. These aerial manipulators have usually the task of collecting an structural element from its initial position and fly it to a final position, where the element is used to assemble a strut structure. The main configuration of such a robot is an under-actuated rotary-wing aircraft (usually a quadrotor) and a robotic serial manipulator arm. Different degrees of freedom (DOF) are used for the arm and different camera placements are considered.

The usual implementation considers the simultaneous control (at the velocity level) of the mobile platform (i.e. quadrotor) and the manipulator for such a grasping task. Since the sum of the 4 DOF of a quadrotor plus the multiple-DOF of a serial manipulator leads to a redundant system, the possibility of choosing degrees of freedom is used to realize different subtasks (e.g. joint limit reaching prevention). The Visual Servoing controller chosen generates velocity inputs both for the manipulator joints (i.e. $\dot{\boldsymbol{q}}$) and for the quadrotor (i.e. translational velocity $\boldsymbol{v}$ and rotational velocity ω_z). The use of a weighted pseudo-inverse allows to favor the control of the mobile platform when the distances to the target are bigger and increase the manipulator mobility when

[6] https://www.grasp.upenn.edu
[7] http://www.arcas-project.eu
[8] http://www.arcas-project.eu

it is close to the target.

[25] propose a quadrotor equipped with a 5 DOF arm. Traditional Visual Servoing distinguishes between two different classes of camera configuration: eye-to-hand (fixed in the workspace) and eye-on-hand (mounted on the mobile platform). In this paper a new configuration for the camera is presented, called onboard-eye-to-hand, i.e. the camera is placed on-board of the robot while it observes the manipulator. In this way, the manipulator can accomplish large rotations while the target is not left out of the camera field of view, as happens in the eye-in-hand configuration. Furthermore, for the case of eye-in-hand configuration, during assembly tasks the manipulator end-effector can contact or impact with objects and damage or obstruct the camera. Thanks to the onboard-eye-to-hand camera configuration the paper is able to introduce a variation of the IBVS approach, called Self Visual Servoing (SVS). Where the error nullified comes directly form the image itself (hence the adjective self) an there is no need for a target image. The servo controller implemented has two different tasks. The main task is positioning the feature points at a target position on the target object and the second one the end-effector motion. The error formulation decouples both tasks and a weighted pseudo-inverse is used to provide a different gain for the arm joints rates $\dot{\boldsymbol{q}}$ than for the UAV velocities $\boldsymbol{v}$ and ω_z control.

Image moments as features for the Visual Servoing are proposed in [24] for the previous system. Furthermore, aerial manipulators have to cope with the change of the center of mass during flight due to the effect of suspended loads (see [30]). To achieve this behavior, low-level attitude controllers are usually designed to compensate this effects using Cartesian impedance control (see [21]) or an adaptive control approach is followed. In this case, the system includes a controller to reduce dynamic effects by vertically aligning the arm center of gravity to the multirotor gravitational vector, along with one that keeps the arm close to a desired configuration of high manipulability and avoiding arm joint limits. In [26], the author completes the robot with a nonlinear low-level controller that thanks to a integral approach allows the inclusion of the dynamic coupling of the UAV and the robotic arm, while the control of the system through the velocities provided by the IBVS high-level is maintained.

In this case the source (see [10]) includes as host a gantry used to emulate an UAV and a 6 DOF manipulator with an end-effector mounted camera (i.e. eye-in-hand). Coordination of redundant degrees of freedom by means of partitioned control. Visual servoing is used to drive the end-effector pose relative to a

target thanks to the use of feature points and their desired positions.

[22] uses a hybrid-control framework to take advantage of the main benefits of both IBVS and PBVS schemes to control a octorotor with a 6 DOF arm. Kinematic redundancy of the end-effector is used to accomplish secondary tasks and lead by a hierarchical task-composition algorithm, in conjunction with a smooth activation mechanism for the tasks.

DLR's work within ARCAS project (see [20]) uses of an helicopter and a 7 DOF manipulator. The helicopter is a bigger robot when compared to the rest of the systems, with more than 1 m from manipulator to center of gravity. Influence of the arm movement is significant to the helicopter flight and is actively compensated by the robot controller by means of a coordinated control of both elements. The paper discusses performance and accuracy in aerial manipulation, where the time it takes between the measurement of a position difference and its compensation using the manipulator or the flying platform is the main factor. Additionally, the work presents a multi-marker approach to compensate the possible occlusion of the target marker by the manipulator derived of the onboard-eye-to-hand camera configuration.

A combination of kinematic and dynamic models to develop a passivity-based adaptive controller which can be applied on both position and velocity control is proposed in [18]. Position control is used for waypoint tracking and landing, while velocity control is triggered for target servoing. The robot is a quadrotor with a 3 DOF arm and an eye-in-hand camera. The work uses IBVS with image moments as visual features and tries to solve two problems of the method when applied to aerial manipulators (under-actuated system). Firstly, movements of the manipulator produce movement of the camera, thus making probable that the target object is taken out of its field of view. For this reason a fisheye camera is used, so it is possible to introduce a bigger field of view. Secondly, the under-actuation of the robot is corrected introducing a image modification method. Velocity weighting of the Jacobian matrix in accordance of the situation is used for the simultaneous control of UAV and manipulator.

In [37], redundant manipulation and hierarchical control law are combined with a new variation of the IBVS that does not need the camera parameters. The system establishes as primary task the avoidance of obstacles, as well as several secondary tasks. The visual servo strategy is used to drive the arm end-effector to a desired position and orientation by using a camera attached to it. The configuration used is a 4 DOF quadrotor, which is equipped with a 6 DOF robotic arm. In the common IBVS approaches, Jacobian or interaction matrix,

which relates the camera velocity with the image feature velocities, depends on a priori knowledge of the intrinsic camera parameters. The paper presents a variation of IBVS called Uncalibrated IBVS, the approach uses the barycenter of the features as control points. The method recovers the coordinates of these control points and also the camera focal length, with this data a new formulation of the Jacobian is constructed. The system also compensates by means of a hierarchical algorithm the position of the manipulator. The implementation of this method withing the ARCAS project uses Gazebo and is available on the Internet under the name Kinton[9].

[9] https://devel.iri.upc.edu/pub/labrobotica/ros/iri-ros-pkg_hydro/metapackages/iri_visual_servo/

Reference	Vehicle	Arm's DOF	Camera Configuration	VS Type	Visual feature	Comment
[41] and [42]	quadrotor	1	eye-in-hand	IBVS	cylinder parameters	agressive maneuvers
[25]	quadrotor	5	onboard-eye-to-hand	SVS	points (no target)	-
[24]	quadrotor	5	onboard-eye-to-hand	SVS	perspective projection image moments (no target)	-
[26]	quadrotor	5	onboard-eye-to-hand	SVS	points (no target)	low-lever controller for dynamic coupling robot-arm
[10]	grantry	6	eye-in-hand	IBVS	points	gantry used to emulate an UAV
[22]	octortor	6	onboard-eye-to-hand	Hybrid VS	points	hierarchical task-composition algorithm, smooth task activation
[20]	helicopter	7	onboard-eye-to-hand	IBVS	points	discusses performance and accuracy, multi-marker approach
[18]	quadrotor	3	eye-in-hand	IBVS	corrected perspective projection image moments	adaptive controller for both position and velocity, fisheye camera
[37]	quadrotor	6	eye-in-hand	Uncalibrated IBVS	blobs' barycenters	hierarchical task-composition algorithm

Table 2.1: Overview of the different approaches for VS in flying manipulators

3 Software Requirements Specification and Structured Analysis

This chapter deals with the *Software Requirement Specification* (SRS) [16] and the *Structured Analysis* [38] of the system developed in this work. Thanks to these two procedures, the objectives that the system must fulfill and a decomposition of it into different functions are stated. This leads to a complete definition of the system.

The purpose of this work is to design a visual controller to provide an aerial robot the commands necessary to reach a desired pose with respect to a target object.

The visual servo controller developed is to be integrated into the *Flypulator* aerial manipulator, a fully-actuated aerial robot equipped with a monocular monochrome camera pointing downwards. Since it is not available for testing during the development of this work, the *hector_quadrotor*[1] (see [28]), an under-actuated aerial robot also equipped with a monocular monochrome camera pointing downwards, will be used.

3.1 Software Requirements Specification

In this section, the Software Requirement Specification [16] for the visual servo controller developed in this thesis is presented. The use of SRS helps to define the system that is being designed, tracking continuously that the product developed satisfies the needs of the user. Only when every requirement stated therein is fulfilled the implementation would be completed.

3.1.1 Product Perspective

The VS controller is to be used with an aerial robotic system based on the ROS framework. From the perspective of the robotic system, the VS controller

[1] http://wiki.ros.org/hector_quadrotor

will appear as a ROS node which publishes control commands through a ROS topic to the rest of the system.

The robotic system to be used with the controller is an aerial manipulator whose goal is to acquire a certain pose with respect to a target laying on the ground.

The system developed here is to interact with the camera hardware of the robot, a monocular monochrome camera pointing downwards. The output of the system are the control inputs of the aerial robot system, a kinematic control is adopted, being these inputs the linear and angular velocities of the vehicle. These inputs interact with the already implemented inner control loop for the attitude of the robotic system and are the main input for the outer control loop, the one in charge of the translation.

With the help of the visual servo controller, the robot is able to achieve a desired pose with respect to a target using only the visual information provided by the camera, so no pose computation is involved.

3.1.2 User Characteristics

The product developed in this thesis will be used as part of a ROS-based system, thus the expected user is a designer willing to implement an Image Based Visual Servoing control strategy for his/her robotic system. The user should be familiarized with the ROS framework and the system will need the structure and interfaces of any standard ROS product.

3.1.3 Assumptions and Dependencies

The software has been tested on the following platform. Forward or backward support is not guaranteed on a different set-up.

- ROS version: ROS Indigo[2]
- Operating System: Ubuntu 14.04[3] Trusty Tahr, 64 bit

[2] http://wiki.ros.org/indigo
[3] http://releases.ubuntu.com/14.04/

3.1.4 Functional Requirements

The functional requirements describe what the system must do to complete the overall task:

- **F1**: *Compute desired visual features.* For the desired pose with respect to the target, compute a vector of image features.
- **F2**: *Compute current visual features.* For the current pose with respect to the target, compute a vector of image features from the target.
- **F3**: *Compute feature difference.* Compute the difference between the current feature vector and the desired feature vector to be used as error for the control law.
- **F4**: *Provide velocity commands to the low level controller.* Control input based on image data so that the aerial robot achieves the desired pose with respect to the target.Velocity control based on the error between current and desired image features, no pose estimation.
- **F5**: *Provide feedback to the user and terminate the visual servoing task.* The system must be able to tell the user whether the target pose has been already achieved or not, by publishing the current error of the controller. Once the desired pose has been achieved, it must notify the user and disconnect from the robot.

3.1.5 Qualitative Requirements

- **A1**: All components are working reliably.
- **A2**: The software is sufficiently fast, modular and modifiable.
- **A3**: The implementation is transparent and comprehensible.
- **A4**: Control inputs must provide stable and smooth flight maneuvers.
- **A5**: Robot must be able to start from different initial positions.
- **A6**: Algorithm must be fast enough to allow real time control of the aerial robot.
- **A7**: The implementation should follow the style guide of ROS [36].

3.1.6 General Constraints

- The environment must be sufficiently illuminated for the camera to work.
- The observed object must be planar and continuous and provided to the program as a binary image obtained by segmentation algorithms.
- The target pose must be provided by a sufficient number of features.
- The target must be always in the filed of view of the camera, so features can be extracted an control input computed.
- Testing computer is a MacBook Pro[4] (Early 2015) with 2.7 GHz Intel Core i5 processor, 8 GB 1867 MHz DDR3 memory and Intel Iris Graphics 6100 1536 MB graphics. Linux OS is run using Oracle VM VirtualBox[5] (Version 5.1.14 r112924) with 5 GB base memory and two processors.

[4] https://everymac.com/systems/apple/macbook_pro/specs/macbook-pro-core-i5-2.7-13-early-2015-retina-display-specs.html
[5] www.virtualbox.org

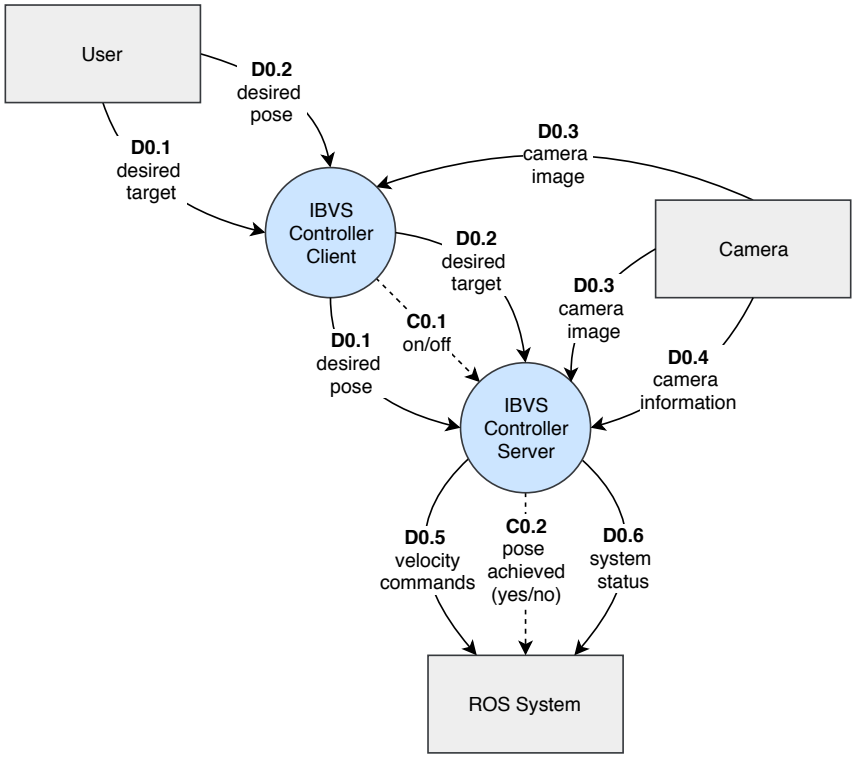
3.2 Structured Analysis

The *Structured Analysis* [19] is a formal method to describe relationships of functional type in complex processes. In order to do that, it views the system from the perspective of data flowing through it. The Structured Analysis is a top-down model, meaning that it analyses the system data flows through successive decomposition. The result of the Structured Analysis of a system is a set graphical diagrams representing the different processes or functions applied to the data and the outputs stored by the system.

3.2.1 Context Diagram

The *Context Diagram* shows the interfaces between the system developed in this work and its environment.

Figure 3.1: Context Diagram



3.2.2 Level A: IBVS Controller

Level A is the first level below the context diagram and contains the main functions (defined in Section 3.1.4 as F1, F2, F3, F4 and F5) of the system developed, the *IBVS Controller Server*. In Figure 3.2 a *Data Flow Diagram* (DFD) is used to describe the interaction between data and processes. A Data Flow Diagram [19] is a decomposition of the system developed resultant of the subjective perspective of the design engineer about it.

Figure 3.2: Data Flow Diagram - Level A

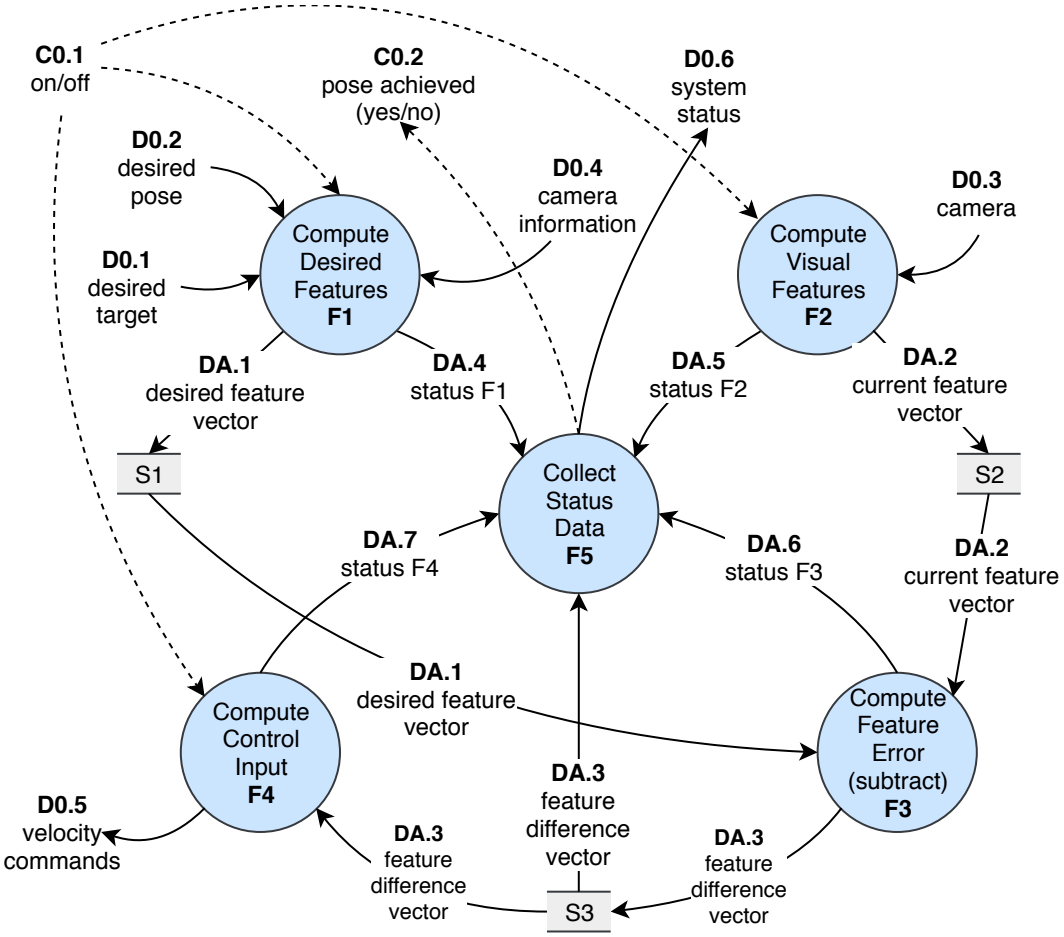


Table 3.1 contains the *Data Dictionary* (DD) for the DFD of Level A. The Data Dictionary [19] is a detailed representation of all data flows involved in a Data Flow Diagram. Where D = data flow and C = control flow.

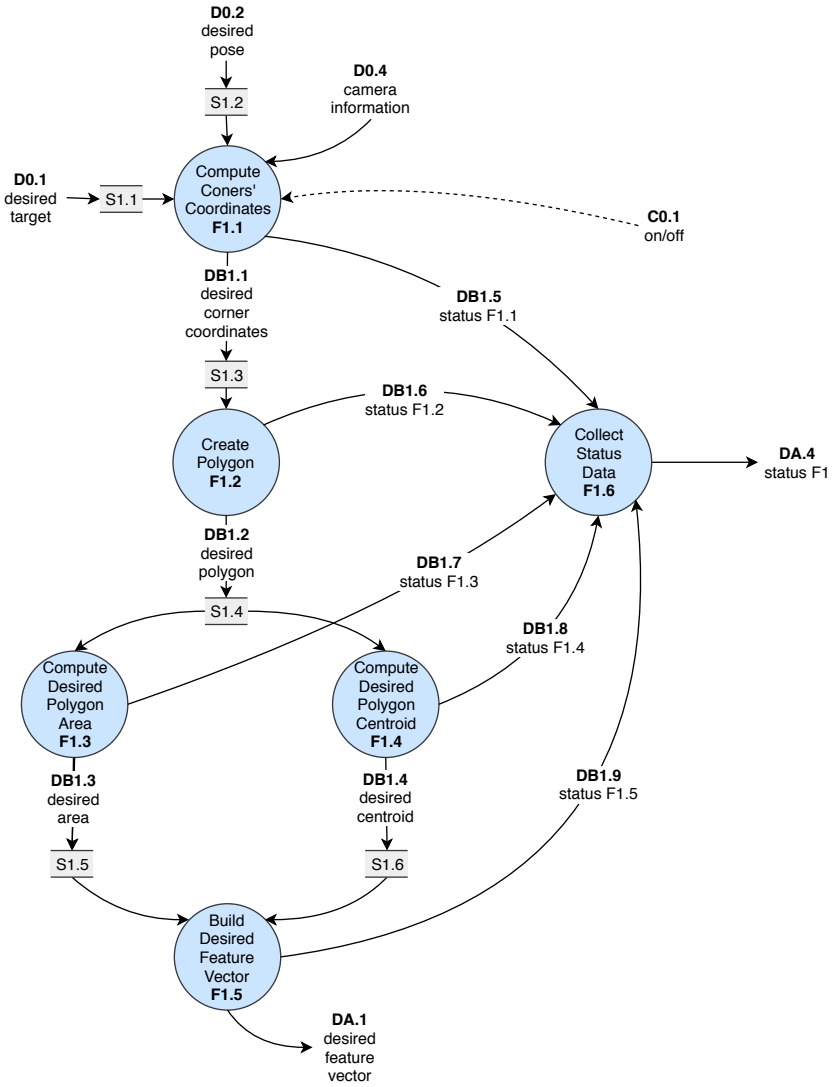
Table 3.1: Data Dictionary for Level A

Flow	Type	Description
C0.1	C	User input to start or stop the system
C0.2	C	Information about pose achieved or not
D0.1	D	User defined desired target ID
D0.2	D	User defined desired camera pose w.r.t. the target
D0.3	D	Camera image data
D0.4	D	Camera information (sensor data, camera model, etc.)
D0.5	D	Velocity commands in the E frame
D0.6	D	System status
DA.1	D	Feature vector for desired pose
DA.2	D	Feature vector from current camera image data
DA.3	D	Features difference vector used for control law
DA.4	D	System status from process F1
DA.5	D	System status from process F2
DA.6	D	System status from process F3
DA.7	D	System status from process F4
S1	Datastore	Desired feature vector
S2	Datastore	Current feature vector
S3	Datastore	Feature difference vector

3.2.3 Level B1: Compute Desired Visual Features

Figure 3.3 contains the Data Flow Diagram that describes the inner work of the process *F1: Compute desired visual features*.

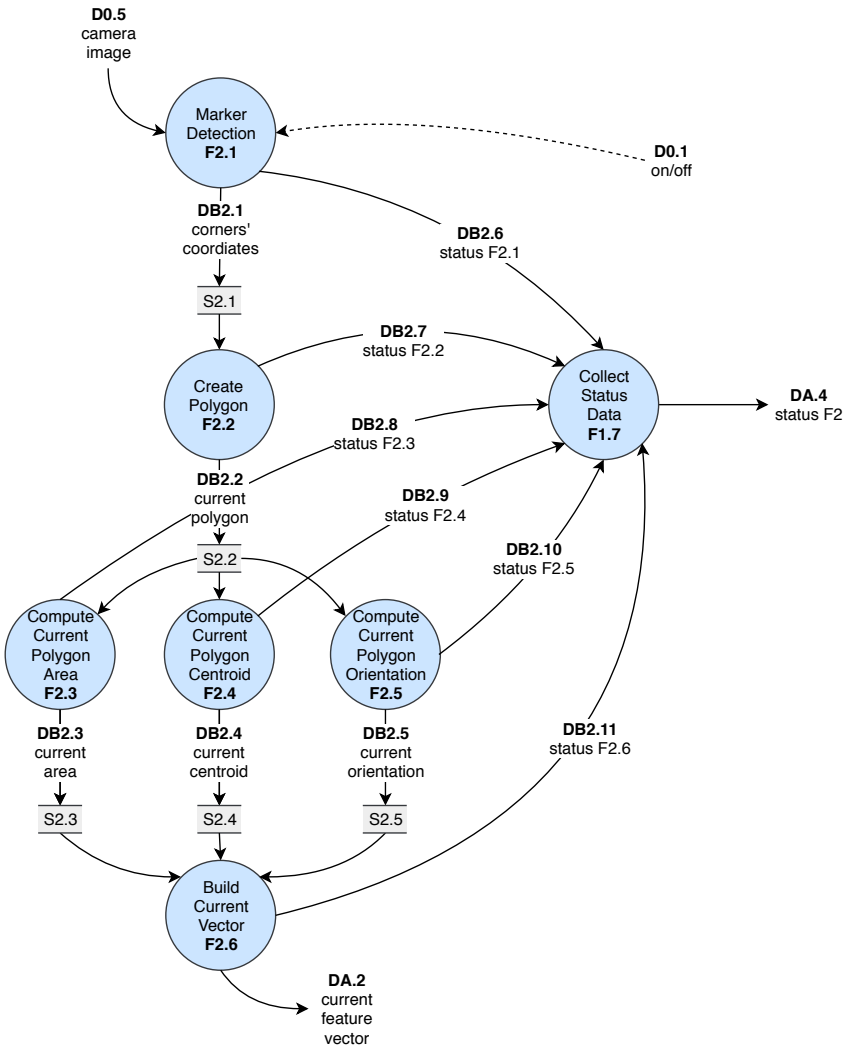
Figure 3.3: Data Flow Diagram - Level B1



3.2.4 Level B2: Compute Current Visual Features

Figure 3.4 contains the Data Flow Diagram that describes the inner work of the process *F2: Compute current visual features*.

Figure 3.4: Data Flow Diagram - Level B2



3.2.5 Level B3: Compute Control Law

Figure 3.5 contains the Data Flow Diagram that describes the inner work of the process *F4: Compute control law.*

Figure 3.5: Data Flow Diagram - Level B3

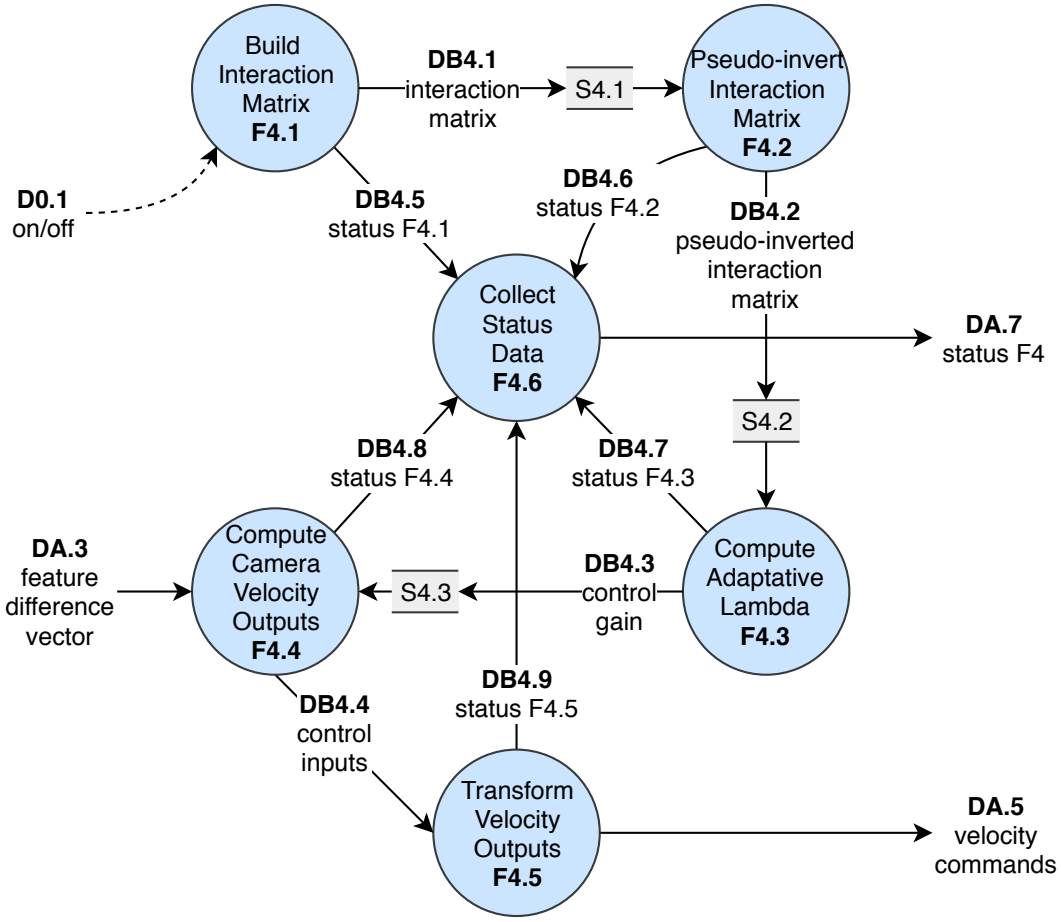


Table 3.2: Data Dictionary for Level B

Flow	Type	Description
DB1.1	D	Desired corners' coordinates in the image plane
DB1.2	D	Desired polygon object
DB1.3	D	Desired polygon area
DB1.4	D	Desired polygon centroid coordinates in the image plane
DB1.5	D	System status from process F1.1
DB1.6	D	System status from process F1.2
DB1.7	D	System status from process F1.3
DB1.8	D	System status from process F1.4
DB1.9	D	System status from process F1.5
DB2.1	D	Marker corners' coordinates in the image plane
DB2.2	D	Current polygon object
DB2.3	D	Current area of the polygon
DB3.4	D	Current polygon centroid coordinates in the image plane
DB2.5	D	Current angle of orientation of the target
DB2.6	D	System status from process F2.1
DB2.7	D	System status from process F2.2
DB2.8	D	System status from process F2.3
DB2.9	D	System status from process F2.4
DB2.10	D	System status from process F2.5
DB2.11	D	System status from process F2.6
DB4.1	D	Visual servoing interaction matrix
DB4.2	D	Pseudo-inverted visual servoing interaction matrix
DB4.3	D	Control gain λ
DB4.4	D	Velocity control inputs in camera frame
DB4.5	D	System status from process F4.1
DB4.6	D	System status from process F4.2
DB4.7	D	System status from process F4.3
DB4.8	D	System status from process F4.4
DB4.9	D	System status from process F4.5

Table 3.3: Datastore Dictionary for Level B

Flow	Type	Description
S1.1	Datastore	Target marker ID
S1.2	Datastore	geometry_msgs/Pose of frame C w.r.t. frame O
S1.3	Datastore	Coordinates of the desired marker's corners in the image plane
S1.4	Datastore	Desired polygon object
S1.5	Datastore	Desired polygon area
S1.6	Datastore	Desired polygon centroid coordinates in the image plane
S2.1	Datastore	Marker corners' coordinates in the image plane
S2.2	Datastore	Current target polygon object
S2.3	Datastore	Current polygon area
S2.4	Datastore	Current polygon centroid coordinates in the image plane
S2.5	Datastore	Current polygon orientation angle
S4.1	Datastore	Visual servoing interaction matrix
S4.2	Datastore	Pseudo-inverted Visual Servoing interaction matrix
S4.3	Datastore	Control gain λ

4 System Design

In Chapter 2 the theoretical basis of visual servo controllers and ROS systems were presented. Later, in Chapter 3, the desired product was defined thanks to the *Software Requirements Specification* and the *Structured Analysis*. In this chapter, the system design adopted to achieve the above described product is presented.

In order to complete its task, the system must use the five functions defined in Section 3.1.4, which can be divided in the following steps:

- **Step 1**. Receive the desired target from the user.
- **Step 2**. Receive the desired pose with respect to the target from the user.
- **Step 3**. Compute the visual features from the desired pose to be used as desired features $\boldsymbol{s}^*$.
- **Step 4**. Detect the desired target.
- **Step 5**. Compute the visual features form the current pose to be used as current features $\boldsymbol{s}$.
- **Step 6**. Compute the feature difference to be used as control error and send to the user as feedback.
- **Step 7**. Compute the interaction matrix for the current features.
- **Step 8**. Compute the control linear velocity inputs in the camera frame.
- **Step 9**. Transform from the control linear velocity inputs from the camera frame to the robot frame.
- **Step 10**. Compute the target rotation around the z-axis.
- **Step 11**. Compute the control angular velocity input around the z-axis.

- **Step 12**. Send the velocity command to the low-level controller of the robot.
- **Step 13**. Stop when the control error has achieved a certain tolerance.

For *Step 1* and *Step 2*, a ROS action interface has been designed. The action interface implementation is explained in detail in the Section 5.4.

The target used is an AprilTag (see Section 5.3), a squared marker placed on the ground. Several markers can be present in the scene and the user must be able to select them. This feature is useful for a manipulator, since different targets may me necessary for its task. In a similar way, the user must be able of specifying the desired pose, so it can vary depending on the task to be conducted.

For *Step 3*, the target's corners (3D coordinates in meters w.r.t. the object frame O) can be obtained, since the dimensions of the target are know. Using the desired pose, the corner coordinates are transformed to the camera frame C (3D coordinates in meters) and projected into the image plane (2D coordinates w.r.t. the camera frame C in pixels). With the pixel information, the desired image moment features are computed as described in Section 2.2.1 so the $\boldsymbol{s}^*$ vector is obtained. Here the desired area of the target and the desired centroid coordinates are used as features.

Using the current image, the target is detected thanks to the detector described in Section 5.3, so *Step 4* is satisfied.

Once the desired target is detected its corner's coordinates are obtained already in the image plane (2D coordinates in pixels w.r.t. the camera frame C), so the current vector image moment features $\boldsymbol{s}$ can also be computed for *Step 5*.

For *Step 6*, the feature difference is computed as the subtraction $\boldsymbol{s} - \boldsymbol{s}^*$ and is given as feedback tanks to the action interface. The iteration matrices necessary for *Step 7* were presented in Section 2.2.1.

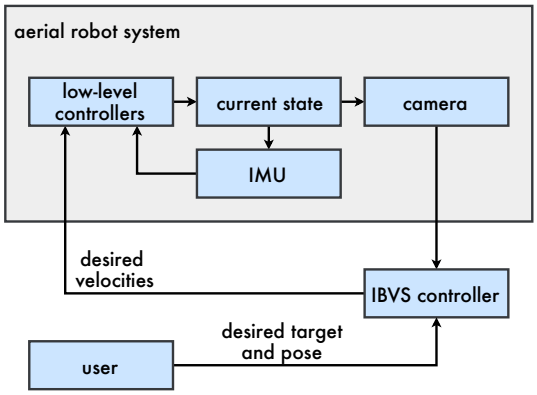
The yaw rotation of the aerial robot is controlled by means of a proportional controller. The rotation of the target can be also computed (*Step 10*) from the corner's coordinates as

$$\theta_z = \arctan\left(\frac{tr_i - tl_i}{tr_j - tl_j}\right) \tag{4.1}$$

Once the feature vectors have been computed, the control error can be obtained as $\boldsymbol{e} = \boldsymbol{s} - \boldsymbol{s}^*$ and together with θ_z are used to compute the control velocities (*Step 8* and *Step 11*) as described in Section 4.1.

The camera velocities are transformed to the robot frame *Step 9* following the procedure explained in Section TODO and used as input for the low level controller of the robot.

Figure 4.1: Interaction between the IBVS controller and the low-level controllers



Putting all together, when the aerial robot simulation is started the action server for the visual servoing task is started. The robot can carry out any other mission, such as navigate to the target's position. Once the target is in the field of vision of the robot, the action client can be started. A desired pose with respect to the target is commanded and the client sends the goal to the server. The server receives the goal and creates a new visual servoing task for it. The desired features are computed taking into account the desired pose and the detector keeps tracking the AprilTag to get its corners and compute the current visual features. The visual feature difference is used as error for a proportional controller, which computes the required camera velocity to get an exponential decrease of the error. The computed velocities are transformed to into the robot frame and published to the low-level control velocity command. The low level control imposes this velocities on the robot while maintaining its stability. When a certain tolerance is reached, the robot is stopped and the client is notified about the final result and then shuts down. The robot is now able to start other mission and the visual servo controller available for a new request.

The visual servoing control of the robot requires then three components:

- The *low-level* controllers of the aerial robot.
- The visual servoing controller implemented within the *action interface*.
- The *client* that uses the action interface to set a new visual servoing task.

4.1 Visual Servoing Algorithm Description

In this section, an IBVS scheme for the control of the translation and yaw rotation kinematics [3] of an aerial robot is presented. The implementation details of this algorithm as a ROS system are given in Chapter 5.

Given an planar target parallel to the image plane, the visual feature vector $\boldsymbol{s} = (x_n, y_n, a_n)$ is defined such that

$$a_n = Z^* \sqrt{\frac{a^*}{a}} \qquad x_n = a_n x_g \qquad y_n = a_n y_g$$

Where a is the area of the object in the image (given in pixels), (x_g, y_g) are its centroid coordinates (given in pixels), a^* the desired area, and Z^* the desired depth between the camera and the target (given in meters). The normalization of the initial quantities leads to a better numerical stability in the computation of the interaction matrix, as noted in Section 2.2.1.

The relationship between the relative motion of the camera and the object and the feature kinematics is given by

$$\dot{\boldsymbol{s}} = \boldsymbol{L}_v \boldsymbol{v} + \boldsymbol{L}_\omega \boldsymbol{\omega} \tag{4.2}$$

Here, the linear and angular velocities of the camera (expressed in the camera frame) are $\boldsymbol{v}$ and $\boldsymbol{\omega}$. The iteration matrix is separated in two matrices, $\boldsymbol{L}_v$ related to translation and $\boldsymbol{L}_\omega$ related to rotation. The desired image feature is denoted by $\boldsymbol{s}^*$, and the visual error is defined by $\boldsymbol{e} = \boldsymbol{s} - \boldsymbol{s}^*$.

As already mentioned in Section 2.2, linear exponential stability is imposed on the error kinematics to ensure an exponential decoupled decrease for $\boldsymbol{e}$

(i.e. $\dot{\boldsymbol{e}} = -\lambda \boldsymbol{e}$, with λ a positive gain). Now $\boldsymbol{e}$ can be used to control the translational[1] degrees of freedom using the following control input

$$\boldsymbol{v} = -(\boldsymbol{L_v})^{-1}(\lambda \boldsymbol{e} + \boldsymbol{L_\omega}\boldsymbol{\omega}) \text{ with } \lambda > 0 \tag{4.3}$$

Generally, the interaction terms $\boldsymbol{L_v}$ and $\boldsymbol{L_\omega}$ depend nonlinearly on the state of the system and cannot be reconstructed exactly from the observed visual data. To cope with this situation the system can be linearized around an equilibrium position.

For an aerial robot where the camera is pointing downwards towards the target object, it is possible to approximate $\boldsymbol{L_v} \approx -\boldsymbol{I}_3$, since the camera image plane is parallel to the target plane. Furthermore, the motion of the robot is smooth and slow, so the value of $\boldsymbol{L_\omega}\boldsymbol{\omega}$ is small compared with the error $\lambda \boldsymbol{e}$.

The convenience of the selected features lays on the decoupling for the control of the degrees of freedom, i.e. each feature is used to compute a different degree of freedom. For a planar object which remains parallel to the image plane, its centroid's coordinates allow the control of the velocities in this. While the area of the object is used to control the perpendicular distance between the object plane and the image plane. As a result, the following approximation is valid

$$\boldsymbol{v} = \lambda \boldsymbol{e} \text{ with } \lambda > 0 \tag{4.4}$$

In order to control the yaw motion of the system, it is possible to compute the rotation θ_z around z with respect to the target. Yaw control is achieved through the following law

$$\boldsymbol{\omega_z} = \lambda \theta_z \text{ with } \lambda > 0 \tag{4.5}$$

The method presented in this section has some limitations, since it depends on the geometry of the target and considers only smooth and slow trajectories. Any aggressive maneuver, or a case in which the parallel target assumption is invalidated, makes the approximations taken fail.

[1] Remember that for the case of a quadrotor, it is an underactuated system. Thus, it is only possible to control its translation and yaw.

In case that this algorithm was to be implemented into a fully-actuated aerial robot, it would be necessary to limit the degrees of freedom of the robot so it stays always parallel to the target. Thus, roll and pitch would not be considered. This restriction is no too heavy since the purpose of this work is acquiring a certain pose with respect to a target laying on the ground.

In order to consider more aggressive maneuvers, the dynamics of the system must be taken into account. Several algorithms have been proposed for this purpose [29] [17] [5]. To cope with the with the limitation of the target being parallel to the image plane, a virtual plane approach [46] can be introduced.

For aerial manipulators, it would be possible to use the above mentioned algorithm to place the robot near the target. Later, an additional visual servoing algorithm could be used to control the arm to conduct the manipulation task. However, in order to take advantage of a fully actuated mobile platform during the manipulation task, it would be specially interesting to use a weighted interaction matrix [37] to control not only the arm but also the mobile platform thanks to a partitioned control.

5 System Implementation

In this chapter, the implementation details of the system previously described are given.

All the elements of the system have been implemented as ROS components using its C++ API. The code is organized as a stack called `flypulator_vs` and composed of five ROS packages described below. All the source code is available on GitHub[1] under a GPLv3 license[2]. The content of each of the following ROS packages is the following:

- `flypulator_vs_controller`: Contains the action server for the visual servo controller. The node receives a desired marker and pose, conducts the corresponding visual servoing task and publishes the velocities to the robot command topic. Additionally the action server publishes feedback about the current task.
- `flypulator_vs_demo`: Contains the ROS launch files to run all the components needed for a demonstration of the controller.
- `flypulator_vs_gazebo`: Contains the URDF files for the markers and the world used during the simulation.
- `flypulator_vs_msgs`: Contains the descriptor of the action interface.
- `flypulator_vs_task`: Contains the action client for the visual servo controller. The node is used to get the user input for the selection of the desired marker and pose. The desired pose is specified in a text file, while the desired marker is selected by the user by clicking on one of the currently available makers in the camera image. The user input is published so the action server can receive it.

[1] https://github.com/PabloRdrRbl/flypulator_vs
[2] https://github.com/PabloRdrRbl/flypulator_vs/blob/master/LICENSE

All the C++ code follows the ROS Style Guide [36] and has been conveniently documented using the Doxygen[3] documentation generator tool.

In the following sections, the most relevant aspects of the implementation are discussed. These are:

- How are the hardware and environment simulated?
- How is the target detection and tracking conducted?
- How is the visual servoing algorithm implemented?
- How is the system integrated as a ROS component?

5.1 Gazebo Simulation

It has been already explained how ROS works as an operating system for the robot, allowing the complete implementation of a robotic system. This is valid not only for real hardware, but also for simulated robots. In order to do that, an additional tool is needed to work along with ROS. This is the roll of the Gazebo robotics simulator.

Gazebo is a 3D dynamic simulator used to simulate robotic environments. It allows the kinematic and dynamic simulation of robots and other bodies, indoor and outdoor environments with different illumination conditions and textures and the emulation of sensors such as cameras or light rangers. Additionally, using the Gazebo plug-in's it is possible to model more complex phenomenas including, aerodynamics or controllers.

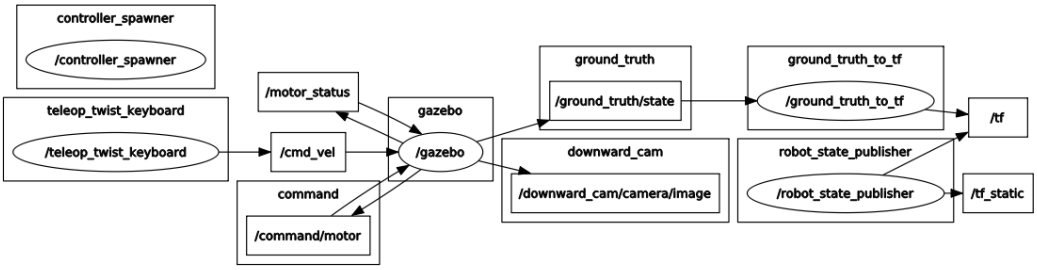
Gazebo works as an additional ROS node. It subscribes to the topics containing the robot's current state and publishes in other topics the new state of the robot. An example of this architecture is shown in Figure 5.1.

In order to use a robot in *Gazebo*, it is necessary to convert its CAD files to the *Unified Robot Description Format* (URDF). This is an XML format for representing a robot model. A similar procedure is also followed to introduce any other model in the simulation, for example the target markers.

Once the robot and the markers have been spawned in the desired world (Gazebo's scenes) the simulator publishes data such as the camera images or the robot state and subscribes to the velocity commands and motor status

[3] http://www.stack.nl/~dimitri/doxygen/

Figure 5.1: Example of a simple communication between Gazebo and the rest the ROS environment of a quadrotor simulation



of the robot. In this manner, it is possible to simulate not only the robot dynamics due to the input commands, but also the sensor data readings of the robot for this state (e.g. the camera image).

For this work, the quadrotor `hector_quadrotor`[4] has been used. It contains not only the robot description in form of URDF files, but also the low-level controllers necessary to command the robot via kinematic inputs. The desired linear and angular velocities are published to the `/cmd_vel` topic the low-level controllers, which compute the necessary motor inputs and allow the control ot the translation and yaw of the robot. Additional, the quadrotor has an on-board camera pointing downwards, which is simulated by Gazebo, who publishes into two topics[5]:

- `/downward_camera/camera/image`: Contains the camera image simulated by Gazebo.
- `/downward_camera/camera/info`: Contains the camera intrinsic parameters.

The `flypulator_vs_gazebo` package contains all launch files to start the Gazebo simulation under the directory `/launch`. The `hector_quadrotor` stack is installed separately and called by the launch file `start.launch`. Any other robot could be spawned in a similar manner.

[4] http://wiki.ros.org/hector_quadrotor

[5] In the code the topics are called respectively `/camera/image` and `/camera/info` for compatibility with other robots. The mentioned names are remapped where the visual servo controller is launched.

5.2 ViSP Visual Servoing Framework

ViSP[6] (*Visual Servoing Platform*) is a visual servoing open source framework for the development of visual tracking and visual servoing systems created by the Inria Lagadic[7] team. ViSP allows the computation and tracking of visual features, the design of control laws for a robotic system and other computer vision tools [43].

The library is written in C++ and distributed for different platforms. It is provides a `vpServo` class which contains all the elements of a classical visual servoing control algorithm. The Listing 5.1 contains a code snippet with a simple visual servoing task using ViSP.

The framework contains a class for each of the common visual features, including the computation of their interaction matrix as explained in Section 2.2.1. ViSP is thoroughly documented, so the reader is recommended to consult the online documentation[44] for further details. In the case of this work, the features used are implemented in the classes `vpFeatureMomentAreaNormalized` and `vpFeatureMomentGravityCenterNormalized` .

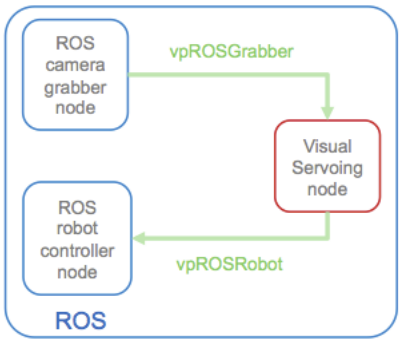
The communication between the library and ROS is established via the `visp_ros`[8] package. The class `vpROSGrabber` subscribes to the camera image and camera info topics and converts the image from the camera into a format compatible with ViSP. While the class `vpROSRobot` allows publishing directly to the velocity command topic. It allows ViSP to be run outside the ROS node if desired, although this would result in less integration of the algorithm. The implementation proposed integrates ViSP inside a ROS node, as described in Figure 5.2.

[6] https://visp.inria.fr
[7] http://team.inria.fr/lagadic/
[8] http://wiki.ros.org/visp_ros

Listing 5.1: Simple ViSP example

```
// Addtional code goes before

// Start a ViSP visual servoing task
vpServo task;

// Configure the visual servoing task where joint velocities
   are computed
task.setServo(vpServo::EYEINHAND_L_cVe_eJe);

// Interaction matrix is computed with the current visual
   features s
task.setInteractionMatrixType(vpServo::CURRENT);

// Set control gain
task.setLambda(1.0);

// Add feature
task.addFeature(s, s_star);

do
{
  // Compute the camera velocities
  v = task.computeControlLaw();

  // Update features
  s.buildFrom(cdMc);

  // Get the feature vector difference (error = s^2 - s_star^2)
  error =  ( task.getError() ).sumSquare();
} while (error > 0.0001);

// End visual servoing task
task.kill();
```

Figure 5.2: Architecture of ViSP running inside a ROS node [45]



5.3 AprilTag Markers

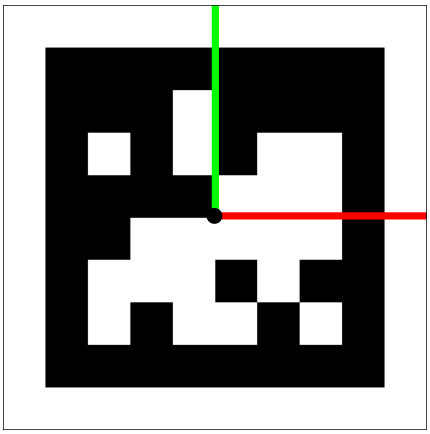
AprilTag[9] is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera [1].

In this work, AprilTag markers have been used as target. Any detected AprilTag can be used to compute the relative pose between the camera and the marker frame of reference. Since the IBVS algorithm does not use any pose computation, here only the marker detection is used. Once the marker has been detected, the detector returns a list of the camera coordinates of the corners of the marker ordered clock-wise and starting in the bottom-left cornet of the tag, as indicated in Figure 5.3.

The AprilTag detector is provided by the creators of the tags and implemented in different programming languages. Here the `vpDetectorAprilTag` detector wrapper provided in ViSP is used.

[9] https://april.eecs.umich.edu/software/apriltag/

Figure 5.3: Example of simple communication between Gazebo and ROS

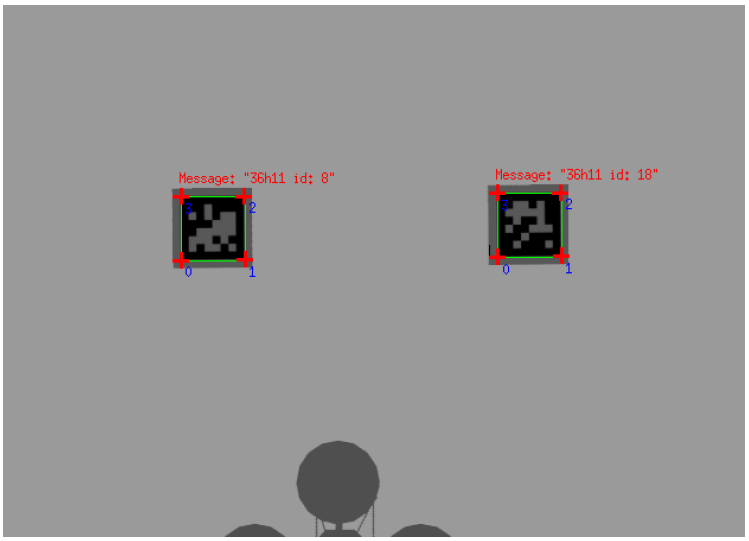


Listing 5.2: Simple AprilTag detection example

```
// Addtional code goes before

// The variable "I" contains the camera image

// Create detector for the 36h11 AprilTag family
vpDetectorAprilTag detector(vpDetectorAprilTag::TAG_36h11);

// Detect markers in image
detector.detect(I);

// Get the corners of the first marker in the list
// (counter-clockwise order)
std::vector<vpImagePoint> p = detector.getPolygon(0);
```

Once the four corners of the marker are detected, it is possible to compute the image features as described in Chapter 4. The marker tracking is not necessary, since it is not moving it is just detected every cycle of the control algorithm. The markers are arranged in families (the implementation uses the `36h11` family), so different makers can be present in the camera image. The detector detects all of them, so the user can access each of them using their respective ID. Figure 5.4 shows the detection of two different target during a simulation, as well as the tagets' IDs.

Figure 5.4: AprilTag detection in a multi-target image

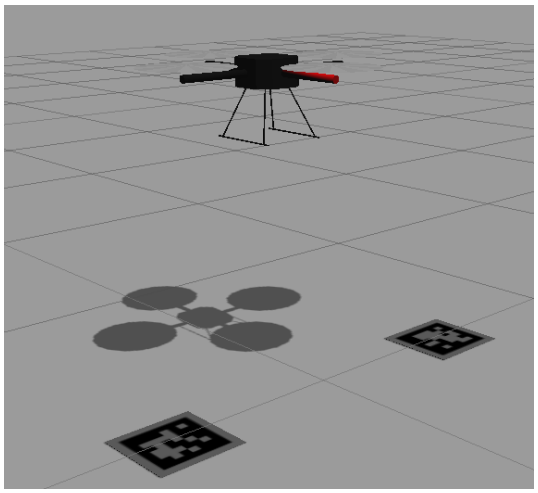


The possibility of having several markers is very useful for manipulation tasks, where several objects can be together. In this case, the system could select a target among all the present possibilities. In order to select a target, the system only needs the ID of the desired marker. This can be obtained from a user click, a data base relating IDs and objects or a text file.

In order to integrate the AprilTag's markers in Gazebo, a cube of dimensions 25cm × 25cm × 1mm was created in Blender[10]. For each marker, the AprilTag image was rescaled to the dimension of the cube's face and applied as texture of the cube using the *UV Unwrapping* tool. The Blender object was finally exported as a COLLADA file. All the Gazebo implementation of the AprilTags is in the `flypulator_vs_gazebo` package. The COLLADA file is spawned as URDF file using the `generic_spawn.launch` launch file.

[10] https://www.blender.org

Figure 5.5: Quadrotor flying over two AprilTags in a Gazebo simulation



5.4 ROS Action Interface

In this section, the action interface created for the visual servo controller is described. The purpose of this interface is to separate the task creation by the user and the task execution in the controller. The action interface is provided by ROS in the `actionlib`[11] package.

The action interface comprises two agents, the action client and the action server. The action server is always running, waiting for the client to send a new task (a new *action*).

An action [33] is an abstraction of a discrete behavior that moves the robot or runs for a longer time, producing feedback during the execution. Examples of possible actions are moving the arm of a robot to a certain pose, pointing the head of a robot into a desired point or performing a laser scan. In this case, the action is commanding the aerial robot towards the desired pose with respect to the target using a visual servoing algorithm.

Using an action interface has several advantages with respect to the other communication methods provided in ROS. An alternative implementation could be running the entire visual servo controller in a unique node. The node could be launched using a service and the feedback could be published into a topic. But services block the communication while there are running and shut down

[11] http://wiki.ros.org/actionlib

when they are finished, they are though to a short task like switching on the robot motors. While the action can be preempted in any moment, e.g. when another goal is sent by a other client, and the server keeps waiting for a new goal. Furthermore, the action interface allows the separation of the goal generation and the goal search. Services can be used to start the low-level controllers of the robot at the start of the simulation, since the low-level controllers are activated only one and keep running all the time. While the action allows the visual servoing task to be started several times during the robot's operation.

Each action is defined by three elements if form of messages on which the server and client communicate: the *goal*, *feedback* and *result*. These messages are created in de *action descriptor*. The action descriptor for the IBVS action is placed in the package `flypulator_vs_msgs` within the directory `/action` in a file called `AprilTagIBVS.action` and reproduced in the Listing 5.3. For each of the three elements, one or more variables are created, each of them defined by a message type and a name.

Listing 5.3: Action descriptor for the IBVS task

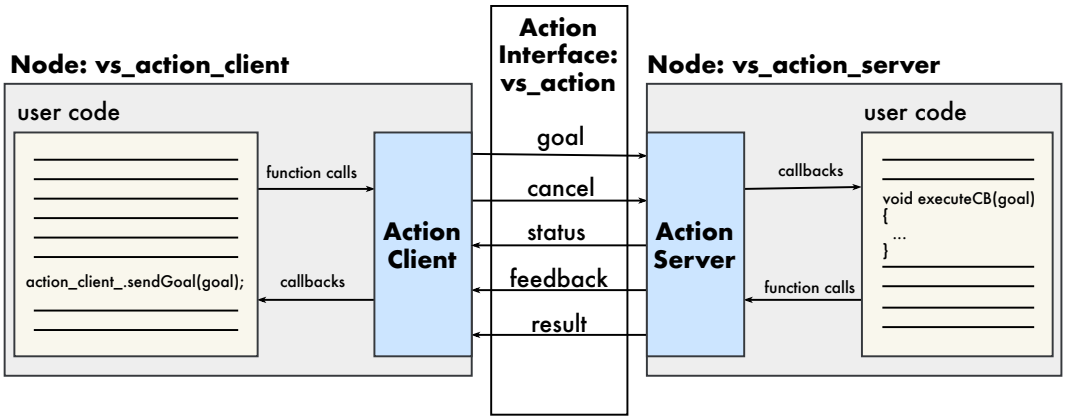
```
# Define the goal
string tag_id
geometry_msgs/Pose pose
---
# Define the result
float64 final_error
---
# Define a feedback message
float64 current_error
```

The action server is implemented within the `flypulator_vs_controller` as a node called `vs_action_server`. It contains the ViSP code to execute the IBVS task based on the goal provided to it. The code is structured around the class `VisualServoingServer`. The class constructor receives the parameters of the servo controller from the ROS Parameter Server and subscribes to the robot command and camera topics. Finally it starts the action server.

Every time that the action server receives a new request from the client, the method `executeCB` is used as callback, running the visual servoing task. During the servo task, the node publishes feedback in the `/vs_action/feedback`

topic, where the feedback is in this case the current error of the control law. When the tolerance of the control error has been achieved, the control law stops the robot and terminates and the server publishes the last control error in the `/vs_action/result`. Additionally, the server tells the client that the task was finished. So the client can shut down while the server keeps running waiting for another client request. In Figure 5.6, the basic architecture of the action interface is presented, while Figure 5.7 shows a part of the ROS network while the visual servoing controller is running.

Figure 5.6: Basic architecture of the implemented action interface



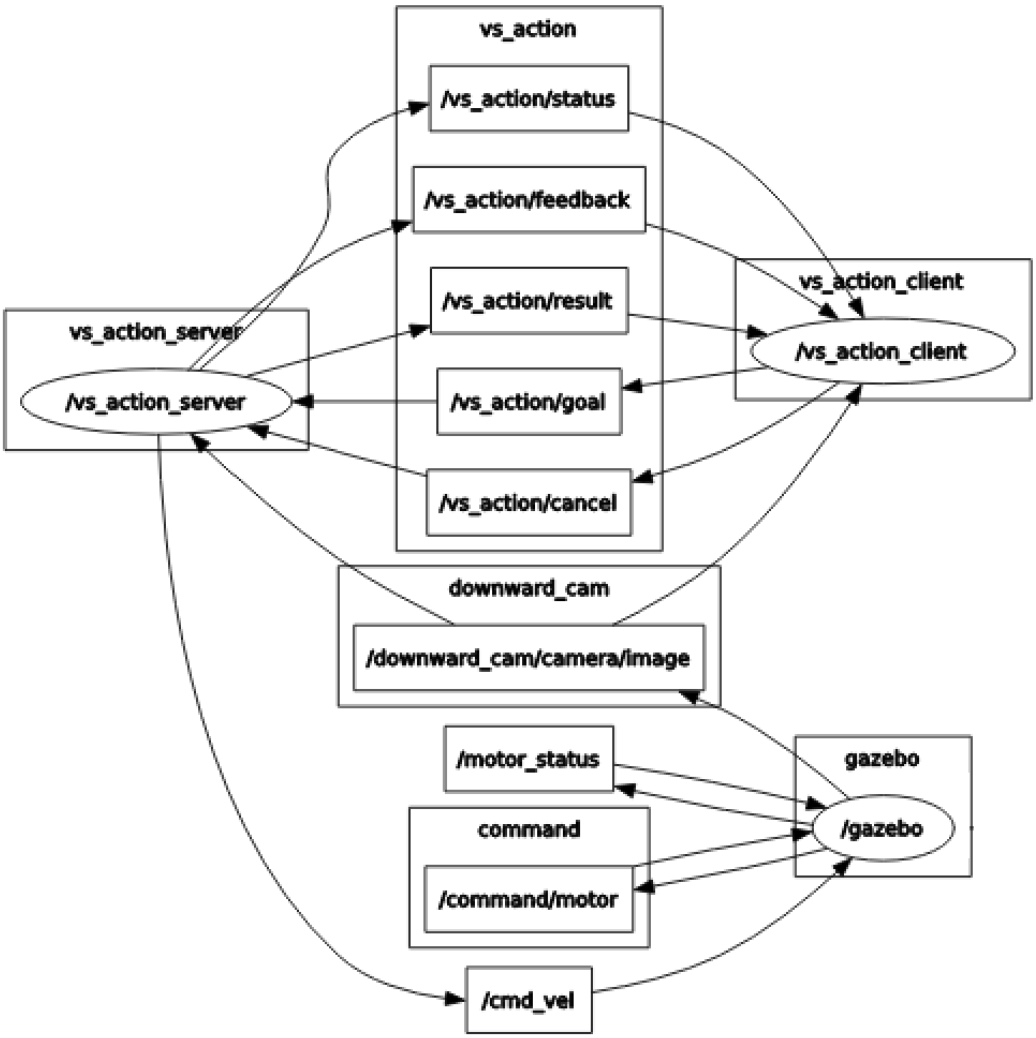
In order to set a concrete goal for the action server a client is necessary. The goal assignation is completely independent from the servo control implementation, i.e. the user can write a new client to assign the goal according to his/her respective needs. The only requirement is that the node runs a client using the class `SimpleActionClient` provided by the `actionlib` package and the action message type created in the action descriptor. A simple implementation of a client for the IBVS controller is presented in the Listing 5.4.

However, for this work a more complex client was written. When the client launches the current image seen by the camera is displayed, so the user can click the desired AprilTag. The desired pose is prescribed using the quaternion from the traget's reference system O to the camera optical frame C, as in the previous example. This implementation is contained in the package `flypulator_vs_client` within the class `VisualServoingClient`.

Listing 5.4: Simple action server for the IBVS controller

```
#include <ros/ros.h>
#include <actionlib/client/simple_action_client.h>
#include <flypulator_vs_msgs/AprilTagIBVSAction.h>

int main (int argc, char **argv)
{
    ros::init(argc, argv, "vs_action_client");

    // Create the action client
    // "vs_action": name of the action, do not change
    actionlib::SimpleActionClient<flypulator_vs_msgs::
       AprilTagIBVSAction> ac("vs_action", true);

    ac.waitForServer(); // Wait for the action server an
       infinite time

    flypulator_vs_msgs::AprilTagIBVSGoal goal; // Create goal

    // Marker ID
    goal.tag_id = "36h11 id: 18";

    // Pose - Position
    goal.pose.position.x = 0.0;
    goal.pose.position.y = 0.0;
    goal.pose.position.z = 1.0;

    // Pose - Orinetation (quaternion)
    goal.pose.orinetaion.x = 0.0;
    goal.pose.orinetaion.y = 0.0;
    goal.pose.orinetaion.z = 0.0;
    goal.pose.orinetaion.w = 0.0;

    ac.sendGoal(goal); // Send goal to the server

    // Waits for the result and shutsdown

    return 0; // Exit
}
```

Figure 5.7: Example of ROS network while running the robot simulation and the visual servo controller



5.5 Controller Parameter Server

The ROS Parameter Server[12] is used to allow to modify the controller's gain. As already explained in Section 2.2.2, the Parameter Server allows a the centralized data storage in form of dictionary structure.

New parameters can be added in form of a YAML[13] file. YAML is a human-readable data serialization language commonly used for configuration files. For the visual servo controller this file is located in the package `flypulator_vs_controller` within the directory `/params` in a file called `params.yaml` and reproduced in the Listing 5.5.

Listing 5.5: Configuration file for the parameters of the visual servo controller

```
use_adapt_lambda: true

lambda/lambda: 0.01

lambda/zero: 1.5
lambda/inf: 0.3
lambda/slope: 40.0

lambda/w: 0.2
```

The parameters are published under a given name space, in this case the `vs_parameter_server_namespace`. Additional levels can be defined within the YAML file. The desired parameters are written in the file which is loaded in the ROS network through the following line in the launch file (see Listing 5.6).

To access the data from a given node, the `ros::NodeHandle::param()` method is used.

[12] http://wiki.ros.org/Parameter%20Server
[13] http://yaml.org

Listing 5.6: ROS launch file for the visual servo controller action server

```
<?xml version="1.0"?>

<launch>

  <!-- Launch visual servo controller node -->
  <node pkg="flypulator_vs_controller"
        type="vs_controller"
        name="vs_action_server">

    <remap from="/cmd_vel" to="/cmd_vel"/>
    <remap from="/camera/image"
           to="/downward_cam/camera/image"/>
    <remap from="/camera/info"
           to="/downward_cam/camera/info"/>
  </node>

  <!-- Load visual servo controller parameters -->
  <rosparam ns="vs_parameter_server"
            file="$(find flypulator_vs_controller)/params/
               params.yaml"/>

</launch>
```

6 System Validation

In the present chapter the IBVS controller developed in this work will be validated. First, the performance of the system implementation will be tested. Secondly, it will be validated if the system proposed satisfies the requirements described in Chapter 3.

6.1 Controller Performance

In this section, the performance of the proposed control scheme is verified experimentally by means of Gazebo simulations of an aerial robot. All the experiments were undertaken using the `hector_quadrotor` aerial robot and the simulations run in the hardware configuration mentioned in Section 3.1.6.

6.1.1 Experimental Conditions

1. **Prototype description**. The aerial robot used for the text is a quadrotor. The vehicle is equiped with an IMU that allow its state estimation and feed the low level controllers. Two low-level controllers take care of teh attitude stabillization and the velocity control. The camera used for the visual servoing task is placed on board of the quadrotor and pointg downwards towards the target. The 2D visual information comming from the camera is used as the only input for the visual servo controller running ontop of the low-level controllers. The visual controller sends velocity commands to the velocity controller to control kinematicaly the translational and the yaw degrees of freedom. The velocity commands of the quadrotor are saturated to ensure that it stays in quasi-stationary flight regime. In case that the linear velocity vector is bigger that 0.2 m/s, the vector is scaled to this limit value. For the yaw angular velocity a saturation of 0.2 rad/s, so the vehicle can rotate half-revolution in the usual convergence time.

2. **Experimental environment**. For the simulation scenario the Gazebo's `empty_world` is used. The illummination conditions are the standard

conditions of this scenario, the only modification done to it is the target's placement and the robot's respawn. The target is placen in the origin of the `world` frame. The vehicle is navigated until its initial position using a joystick. The `tf`[1] transformations between the target's frame T and the camera frame C are used to ensure that the intitial pose is always correct.

3. **Experimental protocol**. Once the robot is in the desired initial condition the action client for the controller is called. The user clicks on the target and the visual servoing starts. The controller publishes the current error through the action interface feedback. Additional data used for the performance evaluation is published using a custom menssage. The robot state used for the validation come from the real robot (simulatio) state that Gazebo publishes into the topic `/gazebo/model_states`. All the interesting topics are recorded using `rosbag`[2]. Then, a Python script is used to open the bag file and format the necessary data into comma separated values. The resulting files are porcessed with Python to produce the plots.

   The tunning of the PID's gains was conducted in the following form: the proportional gain is increased until convergence is achieved without too much overshoot. Then the integral and proportional gains are adjusted to reduce the oscillations. Empirically, these last gains must be around a hounded times smaller. Since they will work only when the error is very small and only small corrections are necessary.

   The sampling ration of the high-level visual servoing controller is limited to 20 Hz due to the bottleneck of the computer simulation without GPU. While Gazebo solves the physics ODEs using CPU computing, the camera simulations are run in this other component. In the optimal conditions the computer should be able to run the simulation until achieving a 50 Hz rate. In case of changing this rate, a new PID tuning would be needed.

4. **Test cases**. Different initial conditions were considered to test the performance in all the possible cases and are presented below. For all the cases, the error convergence criteria used are 10^{-5} for the translation

[1] http://wiki.ros.org/tf
[2] http://wiki.ros.org/rosbag

error and 5×10^{-3} for the rotational one.The first was chosen empirically to have millimeter accuracy in the final pose and the second ensures an orientation error of 10^{-4} rad $\simeq 0.3$ deg.

Noisy camera input conditions were also test. It was founded that since all the feature computation is based on the AprilTag detection, it robustness makes that common camera Gaussian noise conditions ($\mu = 0$ and $\sigma = 0.1$) do not influence the results.

6.1.2 Large Initial Displacements

For the case of large initial displacements the initial pose was chosen to be $(x, y, z) \simeq (-0.9, -0.5, 2.0)$ m and $(\phi, \theta, \psi) \simeq (0.0, 0.0, 18)$ deg. With desired pose $(x, y, z) \simeq (0.0, 0.0, 1.0)$ m and $(\phi, \theta, \psi) \simeq (0.0, 0.0, 0.0)$ deg (camera hovering over the target at 1 m height of the ground). The results of this test case are presented in Figure 6.1 and identified with different letters form *a* to *f*.

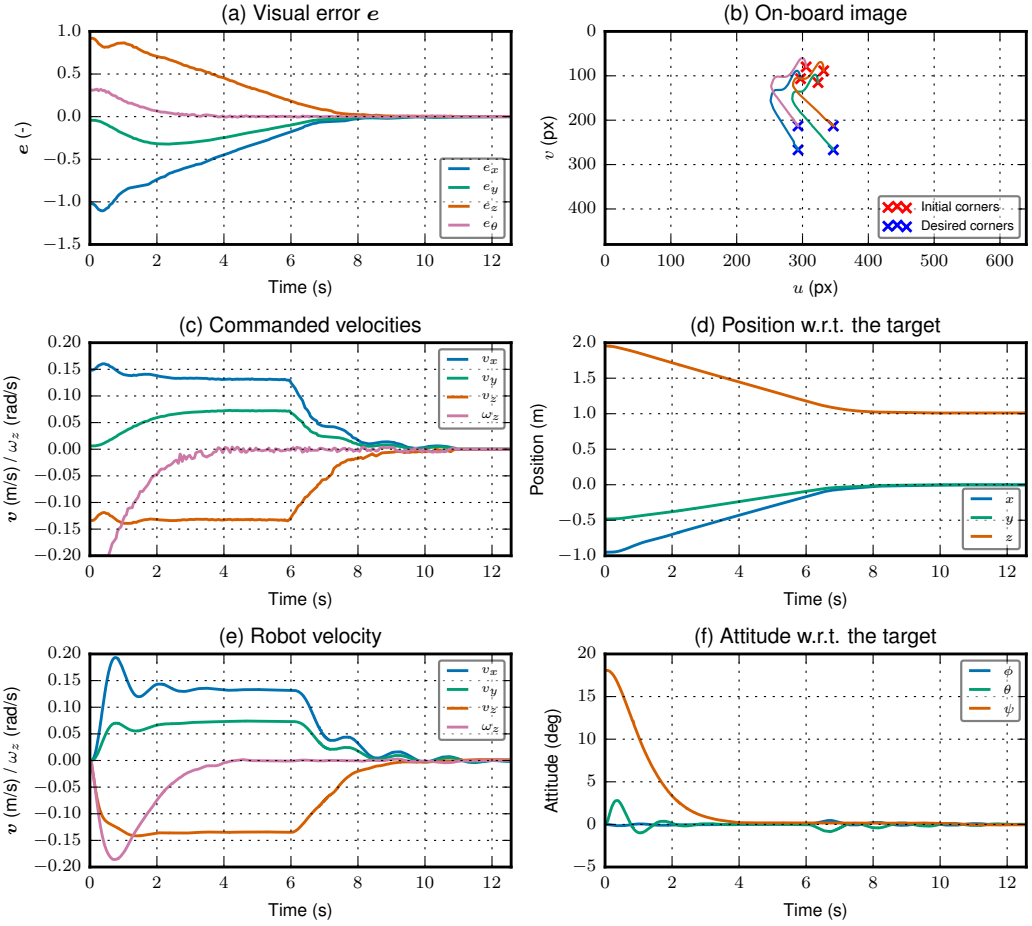
Due to the linear correspondence between image and task space, the system response shows a good asymptotic convergence for both image and task space. The convergence time being $t > 12$ s.

During the first second is possible to observe in the disturbance caused by the dynamic effects when the aerial robot is accelerating from zero to the commanded velocity. These dynamic effects are bigger in (e) the actual robot's velocity achieved by the low-level controllers than in (c) the commanded velocities by the high-level visual servoing.

Some oscillation still occurs before convergence, but this could be caused by the low-level controllers not being able to achieve in time the small velocities required (e.g. due to minimum velocity change of the motors and dynamic effects) for the final correction, since the accuracy imposed is very high.

The yaw control appears to oscillate a bit in (c) the commanded velocities, although (e) the achieved yaw angular velocity is very stable. The same happens in (f), where the orientation with respect to the target converges exponentially asymptotically to the desired one. The small disturbances in ϕ and θ are due to the changes of speed at the beginning and during convergence. Since they stay small, the assumption of image plane parallel to the target plane is still valid.

Figure 6.1: Visual servo controller test results for large displacements

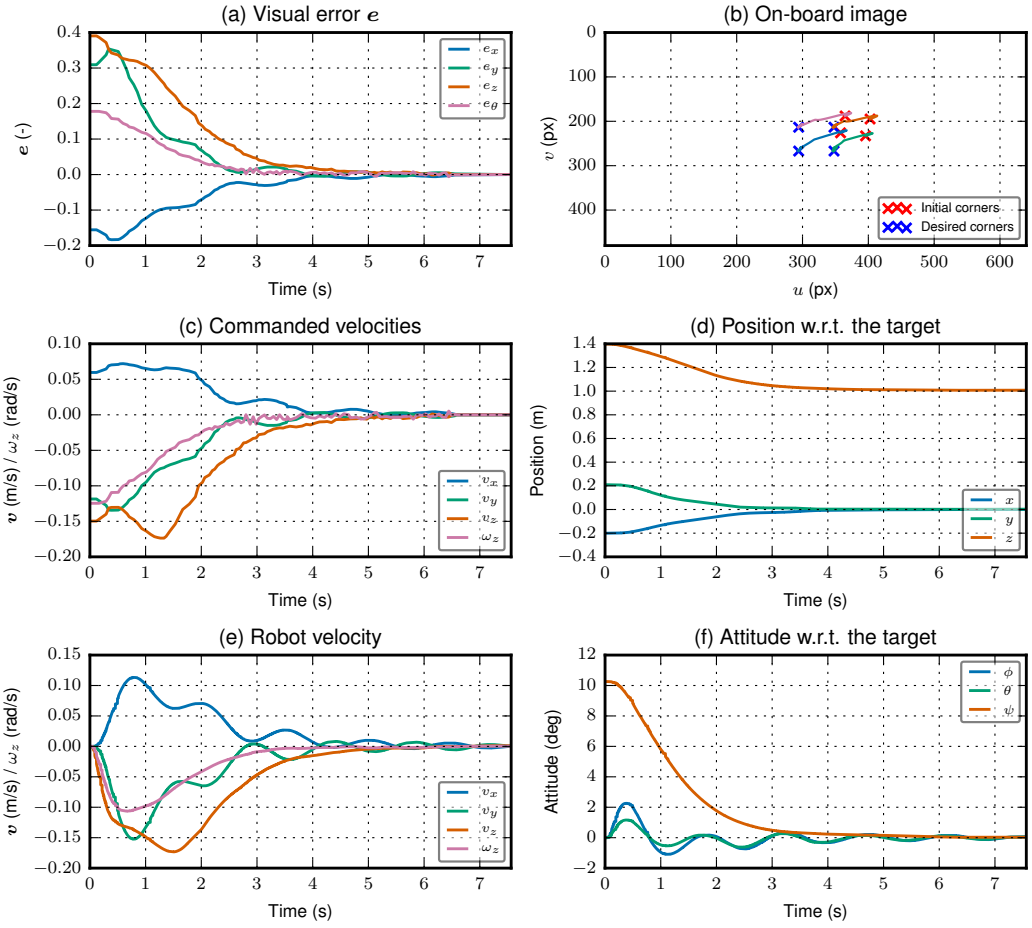


6.1.3 Small Initial Displacements

For the case of small initial displacements the initial pose was chosen to be $(x, y, z) \simeq (-0.2, 0.2, 1.4)$ m and $(\phi, \theta, \psi) \simeq (0.0, 0.0, 10)$ deg. With desired pose $(x, y, z) \simeq (0.0, 0.0, 1.0)$ m and $(\phi, \theta, \psi) \simeq (0.0, 0.0, 0.0)$ deg (camera hovering over the target at 1 m height of the ground). The results of this test case are presented in Figure 6.2 and identified with different letters form *a* to *f*. The results here are similar to the large displacements case. Convergence time is

lower here.

Figure 6.2: Visual servo controller test results for small displacements



6.2 Validation of the Requirements

In this section, the requirements for the developed system, which where imposed in Section 3.1 are validated.

6.2.1 Validation of the Functional Requirements

In Section 3.1.4, five different functional requirements where imposed. Now, it is discussed whether each of them was satisfied:

- **F1**: *Compute desired visual features.* Chapter 4 describes how the ROS node `vs_action_server` computes the visual features from the desired pose and the known target dimensions.
- **F2**: *Compute current visual features.* Chapter 4 describes how the ROS node `vs_action_server` computes the visual features from the current camera image and the detected target's corners.
- **F3**: *Compute feature difference.* Once the current and desired features are computed, the same node computes its difference.
- **F4**: *Provide velocity commands to the low level controller.* Once the desired camera velocities have been computed by the PID controller described in Chapter 5 are known, they are transformed to the robot's body frame E.
- **F5**: *Provide feedback to the user and terminate the visual servoing task.* Thanks to the action interface described in Section 5.4, the user receives feedback about the current error in the controller, as well as notification when the task succeeded or it was preempted.

6.2.2 Validation of the Qualitative Requirements

Section 3.1.5 described the qualitative requirements of the system proposed. Now, it is discussed whether each of them was satisfied:

- **A1**: *All components are working reliably.* The software has been tested using Gazebo simulations and its performance analyzed in Section 6.1.
- **A2**: *The software is sufficiently fast, modular and modifiable.* Modularity and modifiability has been greatly achieved thanks to the integration of the system in different ROS nodes and the use of an action interface to send tasks to the visual servoing high-level controller. Additionally, other visual features could be implemented thank to the use of the ViSP library.

- **A3**: *The implementation is transparent and comprehensible.* The implementation has been comprehensively documented.
- **A4**: *Control inputs must provide stable and smooth flight maneuvers.* The controller's velocity commands are saturated so the velocities are slow enough to satisfy the assumption of the image plane been parallel to the target. Under this conditions the trajectory is exponentially asymptotic.
- **A5**: *Robot must be able to start from different initial positions.* Different initial positions have been tested during the performance validation in Section 6.1.
- **A6**: *Algorithm must be fast enough to allow real time control of the aerial robot.* The software speed is not a problem. However, it has been identified a bottleneck for the test computer in the image simulation performed by Gazebo. The computer does not have a graphic performance enough to produce images at a rate higher that 20 Hz, thus limiting the controller to this rate, which still is enough to obtain the desired results.
- **A7**: *The implementation should follow the style guide of ROS.* All the code has been conveniently formated and documented following the mentioned style guide.

7 Conclusions

In this thesis, the system development of a Image Based Visual Servoing controller to control the translational and yaw degrees of freedom of an aerial robot has been described through all its phases.

The process started with a review of the theoretical background of a IBVS controller and an analysis of similar systems. Since the final goal of the system is to work in aerial manipulation, the use of visual servoing in aerial manipulators was also covered in this part.

In the requirements specification and system analysis, the project goals and objectives where translated into fix requirements for a later implementation. After at this stage, the desired system was defined and the next step would be design all the system components so that such requirements were satisfied.

Within the system design the concrete strategies and algorithms to achieve the imposed requirements where analyzed an developed. The target was chosen to be an AprilTag, whose corners can be detected reliably and form an square. Based on this information, the centroid of the target, its area and orientation where selected as visual features. Under the assumption of smooth maneuvers, it can be said that the image plane is always parallel to the target. Thanks to this property, the relationship between the image features and the camera velocities is decoupled. Making possible to control each of the velocities with one of the selected features.

The difference between the desired and current features is used as error for a PID controller. A PID controller computes the velocities that make the error zero and ensures an adequate system performance. The desired camera velocities are transformed to the body frame of the aerial robot and commanded to the low-level controllers. When the robot moves, the new image provides a new set of features, closing the loop.

During the implementation phase, the designed system was materialized through its implementation as a ROS component so it can interact with the rest of the robot. The visual servo controller was implemented as a ROS action interface. This allows to divide the servoing task in a client and a server. The server runs the controller itself, computing the visual features and the

correspondent camera velocities, while the client is used to set a new task goal (i.e. an AprilTag maker ID and desired pose) and waits for the server to finish. The system to detect several markers at the same time and use the one provided by the user. When the marker goes out of the field of view of the camera, a simple algorithm tries to go back towards the lost target.

Finally, the performance of the controller was tested, achieving the desired behavior and achieving convergence to the desired pose in ten to twenty seconds.

8 Future Work

The present work is part of the Flypulator project at the Institute of Automation Engineering of the Technische Universität Dresden. This aerial robot is going to be a fully-actuated hexarotor equipped with a multi-degree of freedom serial actuator, which will be used for aerial manipulation tasks.

The controller developed allows only the control of the translational and yaw degrees of freedom, so the next natural step would be to take into account the serial manipulator in the visual servoing task.

Due to this nature, taking into account the dynamic behavior of the aerial vehicle during strong maneuvers may not be important. Since aerial manipulations does not usually require them. However, due to the presence of the serial actuator arm, it would be recommended to consider its dynamics during the visual servoing . In this way the arm position can be controlled to minimize disturbances.

Partitioned control should be implemented to apply visual servoing for the control of the mobile platform and the arm. So depending on the distance to the target the camera velocities are applied using the mobile platform of moving the serial manipulator.

The current standard in visual servoing for aerial manipulators is a 4DOF plaform with a 6DOF arm. The weighted Jacobian matrix approach is combined with a priority task scheduler to take advantage of the over-actuation of the system and choose the degrees of freedom taking into account secondary task such as collision avoidance and joint limit reaching prevention.

Maintaining the camera on-board (eye-on-board) pointing to the end-effector would have the advantage of using the same camera also for visual flow. Otherwise, the camera could be placed in the end-effector to achieve a pure eye-in-hand configuration.

In order to achieve the control of six degrees of freedom, new visual features should be applied. Virtual random 3D point clouds with respect to an AprilTag of similar AR markers is the current strategy in all the aerial manipulators.

Using model based tracking could be used instead of markers, so the only think necessary to carry the visual servoing task would be the CAD model of

the object to be manipulated. Allowing an easier interaction of the robot with the environment.

References

[1] *AprilTag*. Apr. 20, 2018. URL: https://april.eecs.umich.edu/software/apriltag/ (cit. on p. 45).

[2] H. J. Asl et al. "Vision-based control of a flying robot without linear velocity measurements". In: 2015 (cit. on p. 16).

[3] O. Bourquardez et al. "Image-Based Visual Servo Control of the Translation Kinematics of a Quadrotor Aerial Vehicle". In: *IEEE Transactions on Robotics* (2009) (cit. on pp. 15, 16, 37).

[4] O. Bourquardez et al. "Stability and Performance of Image Based Visual Servo Control Using First Order Spherical Image Moments". In: 2006 (cit. on p. 10).

[5] Z. Ceren. "Image Based and Hybrid Visual Servo Control of an Unmanned Aerial Vehicle". In: *Journal of Intelligent & Robotic Systems* (2012) (cit. on pp. 16, 39).

[6] Z. Ceren et al. "Vision-based servo control of a quadrotor air vehicle". In: *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*. 2009 (cit. on p. 13).

[7] Chaumette, François and Hutchinson, Seth. "Visual servo control. I: Basic Approaches". In: *IEEE Robotics & Automation Magazine* (2006) (cit. on p. 6).

[8] Chaumette, François and Hutchinson, Seth. "Visual servo control, Part II: Advanced approaches". In: *IEEE Robotics and Automation Magazine* (2007) (cit. on pp. 6, 15).

[9] F. Chaumette. "Image Moments: A General and Useful Set of Features for Visual Servoing". In: *IEEE Transactions on Robotics* (2004). DOI:

10.1109/TRO.2004.829463. URL: http://ieeexplore.ieee.org/document/1321161/ (cit. on p. 10).

[10] T. W. Danko et al. "Evaluation of Visual Servoing Control of Aerial Manipulators Using Test Gantry Emulation". In: 2014 (cit. on pp. 18, 21).

[11] B. Espiau et al. "A new approach to visual servoing in robotics". In: *IEEE Transactions on Robotics and Automation* (1992) (cit. on pp. 6, 10).

[12] N. Guenard et al. "A Practical Visual Servo Control for an Unmanned Aerial Vehicle". In: *IEEE Transactions on Robotics* (2008) (cit. on p. 15).

[13] T. Hamel et al. "Visual Servoing of an Under-actuated Synamic Rigid-body System: An Image-based Approach". In: *IEEE Transactions on Robotics and Automation* (2002) (cit. on pp. 14, 15).

[14] Ming-Kuei Hu. "Visual pattern recognition by moment invariants". In: *IRE Transactions on Information Theory* (1962). DOI: 10.1109/TIT.1962.1057692 (cit. on p. 10).

[15] S. Hutchinson et al. "A Tutorial on Visual Servo Control". In: *IEEE Transactions on Robotics and Automation* (1996) (cit. on p. 9).

[16] *IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications*. URL: https://standards.ieee.org/findstds/standard/830-1998.html (visited on 11/14/2017) (cit. on p. 22).

[17] H. Jabbari et al. "Dynamic IBVS control of an underactuated UAV". In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2012 (cit. on pp. 16, 39).

[18] Suseong Kim et al. "Vision-Guided Aerial Manipulation Using a Multirotor With a Robotic Arm". In: *IEEE/ASME Transactions on Mechatronics* (2016) (cit. on pp. 19, 21).

[19] Klaus Janschek. *Mechatronic Systems Design*. 2011 (cit. on pp. 26–28).

[20] M. Laiacker et al. “High Accuracy Visual Servoing for Aerial Manipulation Using a 7 Degrees of Freedom Industrial Manipulator”. In: 2016 (cit. on pp. 19, 21).

[21] V. Lippiello et al. “Exploiting redundancy in Cartesian impedance control of UAVs equipped with a robotic arm”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 (cit. on p. 18).

[22] V. Lippiello et al. “Hybrid Visual Servoing With Hierarchical Task Composition for Aerial Manipulation”. In: *IEEE Robotics and Automation Letters* (2016) (cit. on pp. 19, 21).

[23] E. Malis et al. “2 frac12;D visual servoing”. In: *IEEE Transactions on Robotics and Automation* (1999) (cit. on p. 14).

[24] Rafik Mebarki et al. “Exploiting image moments for aerial manipulation control”. In: *ASME Dynamic Systems and Control Conference*. 2013 (cit. on pp. 18, 21).

[25] R. Mebarki et al. “Image-Based Control for Aerial Manipulation”. In: *Asian Journal of Control* (2014) (cit. on pp. 18, 21).

[26] R. Mebarki et al. “Image-based control for dynamically cross-coupled aerial manipulation”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014 (cit. on pp. 18, 21).

[27] D. W. Mellinger. “Trajectory Generation and Control for Quadrotors”. iUniversity of Pennsylvania, 2012 (cit. on p. 15).

[28] Johannes Meyer et al. “Comprehensive Simulation of Quadrotor UAVs using ROS and Gazebo”. In: (2012), to appear (cit. on p. 22).

[29] Ozawa, Ryuta and Chaumette, François. “Dynamic Visual Servoing with Image Moments for a Quadrotor Using a Virtual Spring Approach”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011 (cit. on p. 39).

[30] I. Palunko et al. “Agile Load Transportation : Safe and Efficient Load Manipulation with Aerial Robots”. In: *IEEE Robotics Automation Magazine* (2012) (cit. on p. 18).

[31] N. Papanikolopoulos et al. “Adaptive Robotic Visual Tracking”. In: *1991 American Control Conference*. 1991 (cit. on p. 14).

[32] Jenelle Armstrong Piepmeier. “A Dynamic Quasi-newton Method for Model Independent Visual Servoing”. AAI9953830. PhD thesis. Atlanta, GA, USA, 1999. ISBN: 0-599-56997-2 (cit. on p. 14).

[33] *ROS Communication Patterns*. URL: http://wiki.ros.org/ROS/Patterns/Communication (visited on 04/19/2018) (cit. on p. 48).

[34] *ROS Concepts*. URL: http://wiki.ros.org/ROS/Concepts (visited on 04/19/2018) (cit. on p. 11).

[35] *ROS Introduction*. URL: http://wiki.ros.org/ROS/Introduction (visited on 04/19/2018) (cit. on p. 11).

[36] *ROS Style Guide*. URL: http://wiki.ros.org/StyleGuide (visited on 11/15/2017) (cit. on pp. 24, 41).

[37] A. Santamaria-Navarro et al. “Uncalibrated Visual Servo for Unmanned Aerial Manipulation”. In: *IEEE/ASME Transactions on Mechatronics* (2017) (cit. on pp. 19, 21, 39).

[38] *Systementwurf mit Strukturierter Analyse*. German. 2016. URL: https://www.et.tu-dresden.de/ifa/fileadmin/user_upload/www_files/richtlinien_sa_da/Nutzeranforderungen_nach_IEEE.pdf (visited on 11/14/2017) (cit. on p. 22).

[39] Richard Szeliski. *Computer Vision*. Springer London, 2011. DOI: 10.1007/978-1-84882-935-0. URL: https://doi.org/10.1007/978-1-84882-935-0 (cit. on p. 10).

[40] O. Tahri et al. “Point-based and Region-based Image Moments for Visual Servoing of Planar Objects”. In: *IEEE Transactions on Robotics* (2005) (cit. on p. 15).

[41] J. Thomas et al. “Toward Image Based Visual Servoing for Aerial Grasping and Perching”. In: 2014 (cit. on pp. 17, 21).

[42] J. Thomas et al. "Visual Servoing of Quadrotors for Perching by Hanging From Cylindrical Objects". In: *IEEE Robotics and Automation Letters* (2016) (cit. on pp. 17, 21).

[43] *ViSP*. Apr. 20, 2018. URL: https://visp.inria.fr (cit. on p. 43).

[44] *ViSP 3.1.0 Documentation*. Apr. 20, 2018. URL: http://visp-doc.inria.fr/doxygen/visp-3.1.0/ (cit. on p. 43).

[45] *visp_ros package*. Apr. 20, 2018. URL: http://wiki.ros.org/visp_ros (cit. on p. 45).

[46] D. Zheng et al. "Image-Based Visual Servoing of a Quadrotor Using Virtual Camera Approach". In: *IEEE/ASME Transactions on Mechatronics* (2017) (cit. on p. 39).

Selbstständigkeitserklärung

Hiermit versichere ich, Pablo Rodríguez Robles, geboren am 28.02.1996 in León, Spain, dass ich die vorliegende Bachelorarbeit zum Thema

Image Based Visual Servoing for Aerial Robot

ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

Dipl.-Ing. Chao Yao

Weitere Personen waren an der geistigen Herstellung der vorliegenden Bachelorarbeit nicht beteiligt. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung zum nachträglichen Entzug des Diplomabschlusses (Masterabschlusses) führen kann.

Dresden, den 27.04.2018 .....

Unterschrift