INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Vision Based Pose Computation from Landmarks: an application to Quadrotors

André Filipe Marques da Silva

Dissertação para obter o grau de Mestre em

Engenharia Electrotécnica e de Computadores

Outubro de 2011

Abstract

Worldwide, UAVs have become an important tool for the realization of different tasks that, otherwise, would have to be performed by humans in, sometimes, difficult and dangerous conditions. Even so, there is plenty of space for improving the technical capabilities and functionalities of these vehicles. Particularly, this thesis aims at the development of a module for providing pose error measurements to help stabilize the vehicle's position when it is airborne. This module is composed by lightweight hardware components that can easily be attached to an UAV and uses visual information acquired by a camera for error estimation. The first part of this thesis presents the theoretical background needed to the rest of the work developed. Consisting of some basic computer vision concepts such as camera models and image transformations and the description and analysis of features detectors and descriptors algorithms. The second part presents the developed solution, its implementation and the experimental results. The developed module was attached to a robotic arm and, by closing the loop with a control feedback law, the tests performed shown that the end effector's pose error is correctly estimated and can be driven to zero.

Keywords

Unmanned Aerial Vehicles; Visual Pose Estimation; Features Detectors; Features Descriptors

Resumo

Em todo o mundo, variadas tarefas têm vindo, progressivamente, a ser realizadas por veículos aéreos não tripulados que, caso contrário, teriam de ser realizadas por humanos, muitas vezes em condições difíceis e perigosas. Ainda assim, muito pode ser feito para melhorar as capacidades e funcionalidades técnicas destes veículos. Mais especificamente, nesta tese pretendeu-se desenvolver um módulo para calcular medidas de erro de pose para ajudar a estabilizar a posição do veiculo quando este está em vôo. Este módulo é composto por componentes de hardware muito leves que podem ser facilmente colocados a bordo de um veículo aéreo e usa informação visual capturada por uma câmara para estimar o erro de pose. Na primeira parte da tese descrevem-se as metodologias teóricas necessárias para o trabalho realizado. Este, composto por alguns conceitos básicos de processamento de imagem, como o modelo de uma câmara e transformações de imagens e algoritmos de detecção e descrição de pontos de interesse. Na segunda parte propõe-se uma solução para o problema em causa, a sua implementação e os resultados experimentais. O módulo desenvolvido foi fixado a um braço robótico e, ao fechar o *loop* de controlo, os testes realizados mostram que o erro da pose do *end effector* é correctamente estimado e pode ser levado para zero.

Palavras Chave

Veículos Aéreos Não Tripulados; Estimação Visual da Pose; Detecção de Pontos de Interesse; Descrição de Pontos de Interesse

Acknowledgments

This thesis represents a large amount of dedication, sacrifice and effort. Not only from myself, but for all of those who were around me and were an important part of my life in the last few months.

First of all, I would like to thank Professor Carlos Silvestre for giving me the opportunity to integrate his team, undertake this thesis and study this mixing area of control and image processing for which I have particular interest.

I want to thank Bruno Cardeira, Rita Cunha, David Cabecinhas and André Oliveira for their practical and pragmatic advices concerning the implementation issues that I, eventually, stumbled into.

A special acknowledgement, also, to Professor João Paulo Costeira for having the time and the patience to review with me all my homography decomposition procedure, and giving me that last bit of hope that I needed to solve a problem that I was having for a long, long time.

This thesis, would also not be possible without the precious help from Doctor César Silva and all the personnel at Reverse Engineering. They advised and guided me through a large part of my work. They helped me to understand some important technical concepts and to gain some practical intuition.

Pedro Santos, Henrique Silva, Jorge Ribeiro: you were my colleagues, my brothers in arms and my friends during this important stage of our lifes. Both those intellectual conversations and those relaxing moments during lunch time were essential as well. Thank you and best of luck with your thesis.

To my parents, it has been a long road, with highs and lows. With good, excellent moments, and others not so good. Now, I can say all of them helped me to grow and became the person I am today, thank you. Of course, I also want to acknowledge the rest of the family for standing by me and supporting me during these tough months.

To my closest friends in the personal sphere and to those special people who know who they are, I cannot express my gratitude in words. Without those coffee and cookies times, trips, concerts, movies and laughs, what is left of my mental sanity would not have survived.

Last but not least, I would like to thank to all the people, family and friends, for having the patience to deal with me in those worst moments and also, for actually trying to understand what I was talking about, when I was explaining to them what were my thesis subject.

October, 2011

André Silva

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
AST	Accelerated Segment Test
COM	Computer On Module
DOF	Degrees Of Freedom
DSP	Digital Signal Processor
FAST	Features from Accelerated Segment Test
FOV	Field Of View
FPS	Frames Per Second
G-C	Gumstix-and-Camera
IMU	Inertial Measurement Unit
ISR	Institute for Systems and Robotics
OS	Operating System
PUMA	Programmable Universal Machine for Assembly
RANSAC	Random Sample Consensus
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
TI	Texas Instruments
UAV	Unmanned Aerial Vehicle

1 Introduction

Contents

1.1 Context, Motivation and objectives

The work performed under the scope of this thesis is motivated by the project Advanced Interactive Robotic Tools for the Inspection of Critical Infrastructures (AIRTICI) [3]. The main objective of this project is to develop advanced robotic tools and techniques for the inspection of critical infrastructures, like bridges and dams, for example. These infrastructures require a periodic monitoring programs, in order to evaluate preservation conditions and to detect possible damages due to aging or meteorologic conditions. Considering, for example, the case of dams, the monitoring program, normally consists of a team of technicians who have to examine, possibly using climbing equipment or mobile platforms, all the infrastructure looking for damages, such as cracks. A more practical and safer alternative to this possibly dangerous activity, would be to use an unmanned aerial vehicle (UAV) that could capture images from the whole infrastructure surface and send them to a base station. This would require to maintain at least one person fully focused on teleoperating the vehicle, and another one analyzing the images looking for cracks. This has many drawbacks: the teleoperator has to either remain always within eyesight of the UAV or have video feedback of its surroundings to properly maneuver it without colliding with obstacles, the range of teleoperation is limited and it is known that a human cannot maintain his fully level of attention for long continuous periods. If the vehicle were autonomous, being able to navigate over a path that covered all the infrastructure surface, all the referred drawbacks would be overcome. Plus, it would be possible for a operator to control more than one vehicle at once. An autonomous UAV as the one referred is one of the possible solutions that can be developed by the end of the AIRTICI project.

For a UAV to navigate over all the infrastructure surface, it will need a navigation, guidance and control system. One of the most basic functionalities of these systems is to stabilize the vehicle around a desired position. The present thesis aims at developing a module that helps on this activity.

1.2 Problem Description

Using only a camera and a reference image captured at the desired position, the developed algorithm will be able to calculate the error of translation and rotation, from the current pose to the desired one. With this information, it is possible to take the error to zero and, consequently, the vehicle to the desired pose. One of the requirements for this to work is that the scene captured at the reference pose has to be planar. Nevertheless, in a man made environment, and even more when talking about big infrastructures such as dams, this type of scene is very abundant. Consider the following example of usage: the vehicle is teleoperated into a desired pose where a wall is seen by the camera. Due to winds and other factors, the vehicle will be dragged from the desired pose. The developed algorithm is initiated, and errors start being calculated. With these errors, the vehicle will stabilized around the desired pose using control techniques.

1.3 Proposed Solution

The first main issue is related to the overload capacity of an UAV, the hardware where the developed solution is implemented has be lightweight enough for the vehicle to carry it. The Gumstix Overo Fire Computer-On-Module (COM) together with the Caspa VL camera module will be used. The COM consists of a board as small as a stick of gum that contains the essential components that form a computer (CPU, RAM and storage memory).

Considering specifically the pose estimation problem, several computer vision algorithms will be used. First, features from the reference image are analyzed and characterized using features detection and description algorithms. Then, using the same algorithm, each frame is analyzed to determine if the reference scene is present on the current frame. If it is, an homography matrix that describes the transformation from the current frame to the reference one can be calculated. From this homography it is possible to calculate the rotation matrix and the translation vector that takes the vehicle from its current pose into the desired one. Finally, using an appropriate control law, the vehicle will converge into the reference pose.

1.4 Related Work

Vision based control of unmanned aerial vehicles is an active area of investigation. Several works exist on the topic with different approaches, and all have advantages and disadvantages.

Altüg et al. used a camera located on the ground to estimate the pose of a quadrotor in order to achieve a stable flight [4]. Later, they extended their work by using a second camera on-board the quadrotor[5]. Earl and D'Andrea estimated the quadrotor by using a Kalman filter with on board gyroscopes measurements and off board vision sensor measurements [6]. Regarding the challenge of flying a quadrotor indoors, Romero et al., proposed a simple vision system using off board computation hardware [7]. All these solutions require a ground station to process the visual information, making the vehicle dependent upon a continuous connection with ground.

Concerning on board only solutions, Mondragón et al., Martinez et al. and Bourquardez et al. works estimate the pose of an UAV at real time using an on board camera [8] [9] [10]. Similarly to the approach chosen in this thesis, they explore the information obtained by the projective transformation of planar opbjects into a calibrated camera. But the plane that is captured is previously known, such as a landmark or an helipad for example.

1.5 Main Contributions

With the work performed under the scope of this thesis a module was developed. This module, which we called Gumstix-and-Camera module, is composed by ultra-compact and lightweight hardware: a Gumstix Computer-On-Module, for performing all the computations, and a Caspa VL camera, for image capturing. Together with the developed algorithm , this module is able to compute the camera pose error in relation to a reference image, without the need of special visual marks or of any other type. The only restriction is that the scene captured at the reference image is planar. Finally, the Gumstix-and-Camera module is attached to a robotic arm and is used to close the control loop driving the arm end effector into the pose corresponding to the reference image, making the pose error converge to zero.

1.6 Thesis Outline

The remaining of this thesis is organized as follows. Chapter 2 provides basic concepts from computer vision, introduces the Pinhole Camera Model, the notion of image transformations including planar homography and the steps to the calculation of translation and rotation from the homography matrix. Chapter 3 discusses the algorithms available to detect and describe features on images. Chapter 4 describes the algorithm developed and the hardware used to test it. Chapter 5 shows the results of the tests realized. Finally, on Chapter 6, the results are discussed and final remarks are provided along with possible future work.

2 3-D Structure from Images

Contents

2.1 Introduction

One of the main requirements to control a rigid body using a vision based system is the ability to extract three-dimensional (3-D) information from two-dimensional (2-D) data. That is, using data that lies on a 2-D space, such as images acquired from a camera, determine the 3-D relations between such images.

To have any knowledge of how one image is related to another, it is first necessary to understand how one single image is formed. That is, it is necessary to know the mathematical relationship between the coordinates of a 3-D point and its projection onto the image plane. The most basic model that represents this relationship is known as the pinhole camera model [11]. It describes the camera aperture as a point and assumes that no lenses are used to focus light. Although its simplicity it can often be used to describe how a camera represents a 3-D scene. But, sometimes the camera suffers from some strong non-idealities such as radial and tangential distortions. This causes the effect of seeing curved lines that in reality are straight. So, besides determining the pinhole camera model, this effect also has to be taken into account. After having a model that copes with these effects, the camera is said to be *calibrated*.

The next step involves the determination of how the 2-D coordinates from two images of the same scene relate to each other. With this purpose, the notion of 2-D image transformations are introduced. When the scene being captured is planar, one transformation which is of particular interest is the *planar homography*. If the camera is calibrated, the planar homography describes exactly how the points in one image project into the other.

Once the planar-homography is known, it is possible to extract some 3-D information about the camera pose when the images were taken. The procedure is known as planar-homography decomposition [12] and has as input the planar homography matrix. It provides as outputs two possible solutions for the rotation matrix and the translation vector up to a scale factor. This describes the change of camera pose from one image into the other.

The remaining of this chapter is organized as follows. Section 2.2 shows how the pinhole camera model is constructed in a bottom-up approach and shows the image distortion effect. Section 2.3 describes the 2-D image transformations from the most basic to the planar-homography and how it can be calculated. Finally, section 2.4 explains the planar-homography decomposition method and how to choose the right solution from the two given.

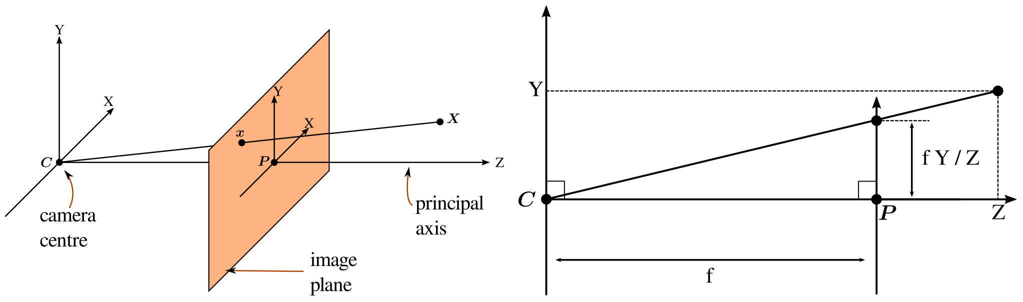
2.2 Pinhole Camera Model

In this section we define the most basic model used to describe how a camera acquires an image: the pinhole camera model[11].

Perfect pinhole model. Image formation is modeled as a central projection of points in space into a plane (see Figure 2.1). Consider a reference frame attached to the camera which we designate as *camera frame*. This frame is defined in a way that the centre of projection, C, is the origin of an Euclidean coordinate system and the projection plane is orthogonal to the Z axis in $Z = f$. A point in space with coordinates $\boldsymbol{X} = (X, Y, Z)^T$ is mapped into the projection plane to a point $\boldsymbol{x} = (x, y)^T$ that is defined as the intersection of the line that contains $\boldsymbol{X}$ and $\boldsymbol{C}$ with the projection plane. Using the notion of similar triangles it is easy to see that:

$$\boldsymbol{x} = (x, y)^T = (fX/Z, fY/Z)^T \tag{2.1}$$

which is the mapping from world to image coordinates.



(a) perspective view **(b)** side view

Figure 2.1: Pinhole camera model.

The centre of projection is called the *camera centre*. The projection plane e called the *image plane*. The line from the camera centre perpendicular to the image plane is called the *principal axis*. The point where the principal axis meets the image plane is designated as *principal point*.

Mapping from world to image plane using homogeneous coordinates In order to be able to simply express the central projection as a linear mapping between the world and image points we use homogeneous coordinates. From now this is the notation that we will use: space and image points are represented by their corresponding homogeneous vectors, $\boldsymbol{X} = (X, Y, Z, 1)^T$ and $\boldsymbol{x} = (x, y, 1)^T$, respectively. So, (2.1) may be written as

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

which can be written in a more compact manner as

$$\boldsymbol{x} = \frac{1}{Z} \mathsf{P} \boldsymbol{X} \tag{2.3}$$

which defines the camera matrix as

$$\mathsf{P} = \mathsf{diag}(f, f, 1)[\mathtt{I}|\mathbf{0}]. \tag{2.4}$$

.

The perfect pinhole model just described is only valid when some strong assumptions hold true. In order to be used to model real cameras some modifications have to be made. These modifications are described next.

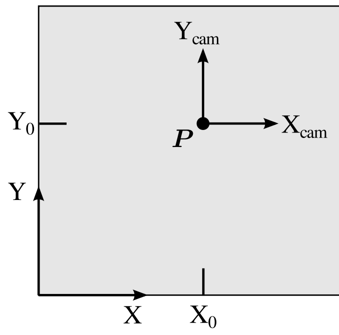


Figure 2.2: Principal point offset

Principal point offset. In the expression (2.2) it is assumed that the origin of the coordinates is at the the principal point. But, in practice, that may not be the case. So, a more general mapping (see Figure 2.2) can be defined as:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z + p_x \\ fY/Z + p_y \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.5}$$

where $(p_x, p_y)^T$ are the coordinates of the principal point. We can define the matrix K as

$$\mathsf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

which is called the *camera calibration matrix* and write (2.5) more concisely as

$$\boldsymbol{x} = \frac{1}{Z}\mathsf{K}[\mathtt{I}|\mathbf{0}]\boldsymbol{X}. \tag{2.7}$$

CCD cameras and finite projective camera. Until now we have assumed that that the image coordinates are Euclidean coordinates having the same scale in both axial directions. But, particularly in CCD cameras, that is not, generally, the case. If image coordinates are measured in pixels there is the possibility of having unequal scale factors in each direction. So, if we define m_x and m_y to be the number of pixels per unit distance in the x and y directions, the transformation from world to pixel coordinates is obtained by multiplying (2.5) on the left by an extra factor $\mathsf{diag}(m_x, m_y, 1)$ and we redefine

$$\mathsf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

where $\alpha_x = fm_x$ and $\alpha_y = fm_y$ are the focal length of the camera in the x and y direction respectively. Using the same logic, $x_0 = m_x p_x$ and $y_0 = m_y p_y$ are the principal point coordinates in terms of pixel dimensions.

Although not so common, it is also possible that the x and y axis are not perpendicular. If that is the case we can consider a more general form of the calibration matrix

$$\mathsf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

where the extra parameter s is called the *skew* parameter.

Radial Distortion. Although, in the camera model described so far, the important property of projecting 3-D straight lines to 2-D straight lines on images holds, it is common that this is not the case in real cameras. An effect, known as radial distortion, causes straight lines to become curved (Figure 2.3). This effect tends to be stronger when the camera lens aperture is bigger.

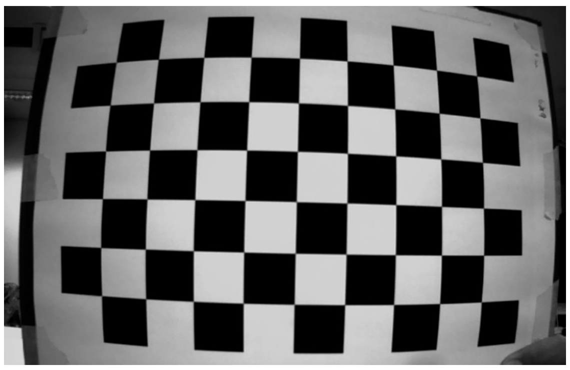


Figure 2.3: Radially distorted image

In order to minimize or, ideally, eliminate this effect, and to define the pinhole camera model, a method known as Camera Calibration has to be performed. The procedure executed to calibrate all the cameras used along this project is explained in Appendix A.2.

2.3 Planar Homographies

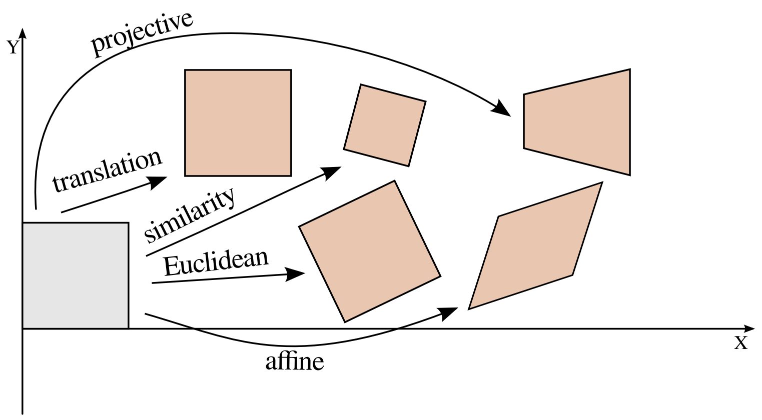


Figure 2.4: 2D transformations

2.3.1 2D image transformations

Consider two images of an object that are captured by a camera from two different positions. The image of the object is not the same in both images. Nevertheless, the two images may be related by a

2D transformation [13], i.e., a transformation that occurs in the 2D plane (see Figure 2.4). A set of 2D transformations is described next. It starts with the transformations with least degrees of freedom, that preserve more objects properties, and incrementally evolves into the more general transformations.

Translation. A translation can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$, being $\boldsymbol{t} = (t_1, t_2)^T$ the translation vector. This transformation can be rewritten in the matrix form as

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \boldsymbol{x} \tag{2.10}$$

A translation preserves orientation, lengths, angles, parallelism and straight lines.

2D Euclidean transformation. A 2D Euclidean transformation is a combination of a rotation and a translation. That is why it is also known as *translation + rotation transformation*. It is modeled as

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \boldsymbol{x} \tag{2.11}$$

where

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.12}$$

is an orthogonal matrix $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$. θ is the rotation angle. It preserves lengths, angles, parallelism and straight lines.

Scaled Rotation. The *scaled rotation* or *similarity transform* also changes the scale. It is written as

$$\boldsymbol{x}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \boldsymbol{x} \tag{2.13}$$

It preserves angles, parallelism and straight lines.

Affine. In the affine transformation the matrix $\boldsymbol{R}$ can take any form. It is written as

$$\boldsymbol{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{x} \tag{2.14}$$

parallelism is no longer preserved.

Homography. This is the transformation with more interest to the study done in this report. It is also known as *projective* or *perspective transform*. It has the form

$$\boldsymbol{x}' = \boldsymbol{H}\boldsymbol{x} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \boldsymbol{x} \tag{2.15}$$

The matrix $\boldsymbol{H}$ is homogeneous and, as so, is defined up to a scale factor. Two matrices that differ only by a scale represent the same transformation. The individual coordinates are given by:

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}}x \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}y. \tag{2.16}$$

Homographies only preserve straight lines.

Table 2.1 resumes the 2D transformations and their properties.

Table 2.1: 2D transformations. The *Preserves* column is incremental from bottom to top, for example, affine transformation preserves parallelism and also straight lines

Transformation	DOF	Matrix	Preserves
Translation	2	$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$	orientation
Euclidean	3	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$	lenghts
Scaled	4	$\begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}$	angles
Affine	6	$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix}$	parallelism
Homography	8	$\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$	straight lines

2.3.2 Planar Homography

Consider two images of points p on a 2D plane P in 3-D space (Figure 2.5). Assuming that the optical centre of the camera never passes through the plane, it is possible to define a transformation that maps a image $\boldsymbol{x}_1$ of a point $p \in P$ into the second image $\boldsymbol{x}_2$ of the same point. To this transformation we will call *planar homography* [12].

Definition: Epipolar constraint Consider two orthonormal reference frames with origins o_1 and o_2 located at the optical centres of the cameras that took the two different images of the same scene. For simplicity, and without loss of generality, assume the world frame to be one of the cameras. If we call the 3-D coordinates relative to the camera frame of a point p, $\boldsymbol{X}_1 \in \mathbb{R}^3$ and $\boldsymbol{X}_2 \in \mathbb{R}^3$, then the following relation holds

$$\boldsymbol{X}_2 = \boldsymbol{R}\boldsymbol{X}_1 + \boldsymbol{t} \tag{2.17}$$

where $\boldsymbol{R}$ is a 3x3 rotation matrix and $\boldsymbol{t}$ is a 3x1 translation vector that relate the two reference frames. Now, if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the homogeneous coordinates of the two images of p and assuming that the camera is calibrated, (2.17) can be rewritten as

$$\lambda_2\boldsymbol{x}_2 = \boldsymbol{R}\lambda_1\boldsymbol{x}_1 + \boldsymbol{t} \tag{2.18}$$

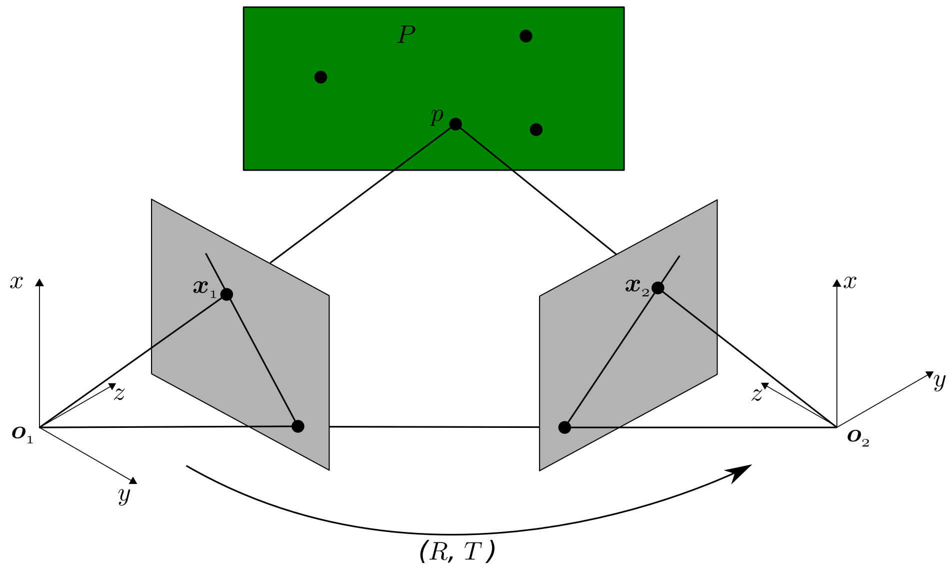


Figure 2.5: Planar homography

because $\boldsymbol{X}_i = \lambda_i \boldsymbol{x}_i$, $i = 1, 2$. Defining the skew-symmetric matrix $\mathsf{S}(\boldsymbol{t})$ that is constructed from the translation vector $\boldsymbol{t}$

$$\mathsf{S}(\boldsymbol{t}) = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}, \tag{2.19}$$

the depths λ_1 and λ_2 can be eliminated. Firstly by premultiplying both sides of (2.18) by $\mathsf{S}(\boldsymbol{t})$

$$\lambda_2 \mathsf{S}(\boldsymbol{t})\boldsymbol{x}_2 = \mathsf{S}(\boldsymbol{t})\boldsymbol{R}\lambda_1\boldsymbol{x}_1 \tag{2.20}$$

which eliminates the translation vector because multiplying by $\mathsf{S}(\boldsymbol{t})$ is equivalent to apply the cross product, so $\mathsf{S}(\boldsymbol{t})\boldsymbol{t} = \boldsymbol{t} \times \boldsymbol{t} = \varnothing$. And secondly, since the vector $\mathsf{S}(\boldsymbol{t})\boldsymbol{x}_2 = \boldsymbol{t} \times \boldsymbol{x}_2$ is perpendicular to the vector $\boldsymbol{x}_2$, premultiplying (2.20) by $\boldsymbol{x}_2^T$ and considering that $\lambda_1 > 0$ leads to

$$\boldsymbol{x}_2^T \mathsf{S}(\boldsymbol{t})\boldsymbol{R}\boldsymbol{x}_1 = 0 \tag{2.21}$$

which finally eliminates the depths λ_1 and λ_2. The matrix

$$\boldsymbol{E} \doteq \mathsf{S}(\boldsymbol{t})\boldsymbol{R} \tag{2.22}$$

is called the *essential matrix*. Equation(2.21) is called the ***epipolar constraint*** or *essential constraint*.

□

As stated before, two images of the same point satisfy the epipolar constraint. However, if the points belong to a 2D plane in the 3-D space further constraints apply.

Defining the unit normal vector of the plane P with respect to the first camera frame as $\boldsymbol{n} = [n_1, n_2, n_3]^T$ and the distance from the plane P to the optical centre of the first camera as $d > 0$ the following relation holds

$$\boldsymbol{n}^t\boldsymbol{X}_1 = n_1X + n_2Y + n_3Z = d \quad \Leftrightarrow \quad \frac{1}{d}\boldsymbol{n}^t\boldsymbol{X}_1 = 1, \quad \forall \boldsymbol{X}_1 \in P \tag{2.23}$$

Applying equation (2.23) in equation (2.17) we get

$$\boldsymbol{X}_2 = \boldsymbol{R}\boldsymbol{X}_1 + \boldsymbol{t} = \boldsymbol{R}\boldsymbol{X}_1 + \boldsymbol{t}\frac{1}{d}\boldsymbol{n}^t\boldsymbol{X}_1 = \left(\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{n}^t\right)\boldsymbol{X}_1. \tag{2.24}$$

Defining the matrix

$$H = \left(\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{n}^t\right) \tag{2.25}$$

which represents the transformation from $\boldsymbol{X}_1$ to $\boldsymbol{X}_2$ leads to

$$\boldsymbol{X}_2 = H\boldsymbol{X}_1. \tag{2.26}$$

Matrix H is called the *planar homography matrix* and, as it can be seen, depends on the motion parameters $(\boldsymbol{R}, \boldsymbol{t})$ and the structure parameters $(\boldsymbol{n}, d)$. Because the translation vector is scaled by the distance d in the term $\frac{1}{d}\boldsymbol{t}\boldsymbol{n}^t$, it can only be expected to extract the direction. but not the module of the vector.

Considering image coordinates we can write

$$\lambda_1\boldsymbol{x}_1 = \boldsymbol{X}_1, \quad \lambda_2\boldsymbol{x}_2 = \boldsymbol{X}_2, \quad \boldsymbol{X}_2 = H\boldsymbol{X}_1 \tag{2.27}$$

and obtain

$$\lambda_2\boldsymbol{x}_2 = H\lambda_1\boldsymbol{x}_1 \quad \Leftrightarrow \quad \boldsymbol{x}_2 \sim H\boldsymbol{x}_1 \tag{2.28}$$

This last equation is known as *planar homography mapping*[12].

2.3.3 Calculation of the Planar Homography Matrix

Premultiplying both sides of equation (2.28) by the skew-symmetric matrix $\mathsf{S}(\boldsymbol{x}_2) \in \mathbb{R}^{3\times 3}$ leads to

$$\mathsf{S}(\boldsymbol{x}_2)H\boldsymbol{x}_1 = 0 \tag{2.29}$$

which is a linear equation in H. As so, the entries of H can be stacked as a vector of the form

$$H^s \equiv [H_{11}, H_{21}, H_{31}, H_{12}, H_{22}, H_{32}, H_{13}, H_{23}, H_{33}]^T \tag{2.30}$$

to rewrite equation (2.29) as

$$\boldsymbol{a}^T H^s = 0 \tag{2.31}$$

with $\boldsymbol{a} \equiv \boldsymbol{x}_1 \otimes \mathsf{S}(\boldsymbol{x}_2) \in \mathbb{R}^{9\times 3}$ representing the Kronecker product (see appendix A.3). Because $\mathsf{S}(\boldsymbol{x}_2)$ is of rank 2, also $\boldsymbol{a}$ is of rank 2. So, in order to determine uniquely (up to a scale factor) the homography matrix it is necessary to know at least 4 pairs of images from points that lie on the same plane and that no three of them are collinear. This way, defining $\chi \equiv \left[\boldsymbol{a}^1, \boldsymbol{a}^2, \boldsymbol{a}^3, \boldsymbol{a}^4\right]^T \in \mathbb{R}^{12\times 9}$ which is of rank 8, the following system

$$\chi H^s = 0 \tag{2.32}$$

can be solved using standard least-squares estimation, or any other equivalent method to determine H up to a scale factor which has the form

$$H_L \equiv \lambda H = \lambda\left(\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{n}^t\right). \tag{2.33}$$

Random Sample Consensus Although, the first naive approach would be to apply a standard least-squares estimation, this would probably lead to a bad estimation of the homography matrix. This would be due to the fact that we will be working with images that contain noise, and it can exist points in one image that are mismatched with the points on the other image, depending on the algorithm used to describe and match the points (see Chapter 3).

A more reliable approach, and that is adopted, is to use Random Sample Consensus (RANSAC) [14]. Basically, this algorithm assumes that in the set of matched points, there are inliers (points correctly matched), and outliers (points not matched correctly) that will lead to a bad estimation when using standard least-squares estimation, because this method tries to fit optimally all the points. With RANSAC a random subset of the points pairs e used as input and least-squares estimation is applied into these hypothetical inliers. The homography estimated is then tested against the remaining points pairs, if a pair fits well enough it is also considered an hypothetical inlier. The homography is considered reasonably good if enough points pairs are considered as inliers. If so, the homography is reestimated using all the new hypothetical inliers. Finally, this new homography reliability is evaluated by using the error of the inliers with this homography. The procedure is repeated a fixed number of times. The homography that has a better reliability is considered the true one.

$\square$

Determined the homography that better describes the transformation from one image to the other, to recover the original homography matrix, it is necessary to determine the scale factor λ

Determination of the homography scale factor Defining $u = \frac{1}{d}\boldsymbol{R}^T\boldsymbol{t} \in \mathbb{R}^3$ we have

$$H_L^T H_L = \lambda^2 \left(\boldsymbol{I} + u\boldsymbol{n}^T + \boldsymbol{n}u^T + \|u\|^2\boldsymbol{n}\boldsymbol{n}^T\right) \tag{2.34}$$

Note that the vector $\boldsymbol{u}\times\boldsymbol{n} = \mathsf{S}(\boldsymbol{u})\boldsymbol{n}$ is orthogonal to $\boldsymbol{u}$ and $\boldsymbol{n}$, is an eigenvector of $H_L^T H_L$ and $H_L^T H_L(\mathsf{S}(\boldsymbol{u})\boldsymbol{n}) = \lambda^2(\mathsf{S}(\boldsymbol{u})\boldsymbol{n})$. This implies that $|\lambda|$ is a singular value of H_L. Furthermore, defining $v = \|u\|\boldsymbol{n}$, $w = u\|u\| \in \mathbb{R}^3$ and considering the matrix

$$Q = u\boldsymbol{n}^T + \boldsymbol{n}u^T + \|u\|^2\boldsymbol{n}\boldsymbol{n}^T = (w+v)(w+v)^T - ww^T \tag{2.35}$$

it is possible to see that it has a positive, a negative and a zero eigenvalue, except when $u \sim \boldsymbol{n}$, Q will have two repeated zero eigenvalues. Either way, λ^2 is the second-largest eigenvalue of $H_L^T H_L$.

Normalization of the planar homography matrix So, if $\{\sigma_1, \sigma_2, \sigma_3\}$ are the singular values of H_L the *normalized planar homography matrix* is obtained using

$$H = \pm\frac{H_L}{\sigma_2(H_L)}. \tag{2.36}$$

The correct sign is determined knowing that the following condition has to be true

$$(\boldsymbol{x}_2^j)^T H \boldsymbol{x}_1^j < 0, \quad \forall j = 1, 2, ..., n. \tag{2.37}$$

because $\lambda_2^j \boldsymbol{x}_2^j = H\lambda_1^j \boldsymbol{x}_1^j$ and $\lambda_1^j, \lambda_2^j > 0$. This condition is known as the *positive depth constraint*.

2.4 From Planar Homography to 3-D Displacement - Planar Homography Decomposition

The procedure to decompose the planar homography matrix into its elements $\boldsymbol{R}$, $\frac{1}{d}\boldsymbol{t}$ and $\boldsymbol{n}$ as explained in [12] is described here.

Using the symmetric matrix $H^T H$, it can be decomposed into the form

$$H^T H = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{2.38}$$

where $\boldsymbol{V}$ is an orthogonal matrix and $\boldsymbol{\Sigma}$ is the diagonal matrix that contains the three eigenvalues of $\boldsymbol{H}^T\boldsymbol{H}$, $\boldsymbol{\Sigma} = \text{sigma}\{\sigma_1^2, \sigma_2^2, \sigma_3^2\}$. Because the matrix $\boldsymbol{H}$ is assumed to be already normalized $\sigma_2 = 1$. Denominating the three column vectors of $\boldsymbol{V}$ as $[v_1, v_2, v_3]$ leads to

$$H^T H v_1 = \sigma_1^2 v_1, \quad H^T H v_2 = \sigma_2^2 v_2, \quad H^T H v_3 = \sigma_3^2 v_3. \tag{2.39}$$

So, and because

$$H^T H = \left(\boldsymbol{I} + \frac{1}{d}\boldsymbol{t}\boldsymbol{n}^T + \boldsymbol{n}(\frac{1}{d}\boldsymbol{t})^T + \|\frac{1}{d}\boldsymbol{t}\|^2 \boldsymbol{n}\boldsymbol{n}^T\right), \tag{2.40}$$

we can conclude that v_2 is orthogonal both to $\boldsymbol{n}$ and $\boldsymbol{t}$. Also, because $H = \left(\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{n}^t\right)$, the length of v_2 is preserved under the map of H, i.e. $\|Hv_2\|^2 = \|\boldsymbol{R}v_2\|^2 = \|v_2\|^2$. Furthermore, the unit vectors defined as

$$u1 \equiv \frac{\sqrt{1-\sigma_3^2}v_1 + \sqrt{\sigma_1^2 - 1}v_3}{\sqrt{\sigma_1^2 - \sigma_3^2}}, \quad u2 \equiv \frac{\sqrt{1-\sigma_3^2}v_1 - \sqrt{\sigma_1^2 - 1}v_3}{\sqrt{\sigma_1^2 - \sigma_3^2}} \tag{2.41}$$

also have its lengths preserved under the map H. Because the vectors v_2, u_1 and u_2 lengths are preserved under the map H so any vectors of the subspaces

$$S_1 = \text{span}\{v_2.u_1\}, \quad S_2 = \text{span}\{v_2.u_2\} \tag{2.42}$$

will have its lengths preserved. Finally, notice that, if $\boldsymbol{n}$ is the normal to the subspace $S_i, i = 1, 2$ then

$$Rv_2 = Hv_2, \quad Ru_i = Hu_i, \quad R(\mathsf{S}(v_2)u_i) = \mathsf{S}(Hv_2)Hu_i. \tag{2.43}$$

So, defining the matrices

$$\begin{aligned} \boldsymbol{U}_1 &= [v_2, u_1, \mathsf{S}(v_2)u_1], \quad \boldsymbol{W}_1 = [Hv_2, Hu_1, \mathsf{S}(Hv_2)Hu_1], \\ \boldsymbol{U}_1 &= [v_2, u_1, \mathsf{S}(v_2)u_1], \quad \boldsymbol{W}_1 = [Hv_2, Hu_1, \mathsf{S}(Hv_2)Hu_1]. \end{aligned} \tag{2.44}$$

equations (2.43) can be rewritten as

$$\boldsymbol{R}\boldsymbol{U}_1 = \boldsymbol{W}_1, \quad \boldsymbol{R}\boldsymbol{U}_2 = \boldsymbol{W}_2. \tag{2.45}$$

This indicates that both subspaces, S_1 and S_2, can lead to a solution of the decomposition problem. Besides the option of the two subspaces, there is also an ambiguity in the sign of the term $\frac{1}{d}\boldsymbol{t}$. So, this

Table 2.2: The four possible solutions to the decomposition of the planar homography matrix

Solution 1	$\boldsymbol{R}_1 = \boldsymbol{W}_1\boldsymbol{U}_1^T$ $\boldsymbol{n}_1 = \mathsf{S}(\boldsymbol{v}_2)\boldsymbol{u}_1^T$ $\frac{1}{d}\boldsymbol{t}_1 = (H - \boldsymbol{R}_1)\boldsymbol{n}_1$	Solution 3	$\boldsymbol{R}_3 = \boldsymbol{R}_1$ $\boldsymbol{n}_3 = -\boldsymbol{n}_1$ $\frac{1}{d}\boldsymbol{t}_3 = -\frac{1}{d}\boldsymbol{t}_1$
Solution 2	$\boldsymbol{R}_2 = \boldsymbol{W}_2\boldsymbol{U}_2^T$ $\boldsymbol{n}_2 = \mathsf{S}(\boldsymbol{v}_2)\boldsymbol{u}_2^T$ $\frac{1}{d}\boldsymbol{t}_2 = (H - \boldsymbol{R}_2)\boldsymbol{n}_2$	Solution 4	$\boldsymbol{R}_4 = \boldsymbol{R}_2$ $\boldsymbol{n}_4 = -\boldsymbol{n}_2$ $\frac{1}{d}\boldsymbol{t}_4 = -\frac{1}{d}\boldsymbol{t}_2$

sums up to four possible solutions in the decomposition problem of the planar homography matrix into the terms $\{\boldsymbol{R}, \frac{1}{d}\boldsymbol{t}, \boldsymbol{n}\}$. These solutions are summarized in Table 2.2.

The problem now, is to determine the correct solution. Since the camera only sees points that are in front of it, two solutions can be eliminated by applying the positive depth constraint, i.e. by imposing $\boldsymbol{n}^T e_3 = n_3 > 0$. This way, only two possible solution are left. Now, several techniques can be applied. For example, a third image of the same plane or a second plane on the same images can be used [15]. Either way, a new homography matrix will be available. After decomposing this second homography and getting two new possible solutions, it is only a question of finding the common solution on the two sets of solutions.

3 Image Features

Contents

3.1 Introduction

As seen in chapter 2, if a planar scene is being captured by a camera, it is necessary to have the correspondence between the 2-D coordinates of points to extract the information about the camera pose change between images. This chapter deals with the problem of automatically finding good points to track and finding their correspondents across images.

Common requirements of the applications that require a visual tracking system are the system to be robust and fast enough to be computed in real time. Although some systems work with optical flow, we are only interested in feature-based visual tracking because optcal flow is prone to long-term drift, as the motion estimates and therefore the errors are integrated over time, and is only feasible for smooth motion.

Although different, all feature-based approaches start with two essential steps: interest points detection and features description.

Mainly, three types of points detectors can be found among the literature:

- *Corner Detectors*: Where one of the first low-level features used to image analysis. The most well known approach is the Harris Corner Detector [16]. It uses the second-order moment image gradient matrix to calculate a score that indicates if the current point should be considered a corner or not. This method is also the basis for some other corner detector that were developed later. Other approaches, for example, use the determinant of the Hessian matrix [17] or measure the change of direction in the local gradient field [18]. Although there are some variants that are, normally corner detectors are not scale invariant.

- *Blob Detectors*: These approaches try to find blobs instead of single corners using local extrema of the responses of certain filters. One of the first used was the Laplacian of a Gaussian which was shown to be scale invariant when applied to to multiple image scales [19]. Then, in order to develop a faster detector, a filter based on differences of Gaussians was used [20]. Because they are separable, they are faster to compute. In order to speed up even more the process, the authors of [21] proposed the Fast Hessian detector which is based on approximations of the Hessian matrix at different scales that are very efficient to compute.

- *Affine-Invariant Detectors*: These type of detectors are more recent. They aim to the detections techniques that are invariant to affine changes. These detector are more robust and provide higher repeatability to strong affine changes but are much more expensive to compute [20].

On the field of feature descriptors there are a few approaches that stand out. There are some earlier ones such as [22], that use derivatives to achieve rotation invariance, or [23] that make use of derivatives of Gaussians of different order. But, is the work of Lowe [20] that really stands out. Its Scale-Invariant Feature Transform (SIFT) work is probably the most well-known and widely used descriptor. It is invariant to changes in scale and rotation. To achieve that, it assigns a local reference frame in relation to a dominant scale and rotation previously computed around the feature point, using local histograms on a square grid. Further refinements of Lowe's work have been made, such as using PCA to reduce descriptor size and, therefore, reducing the matching time [24]. Bay et al. [21] introduced Speeded-Up Robust Features (SURF) as a faster alternative to SIFT. It uses similar SIFT approaches but makes use of integral images and approximations of the expensive Gaussian filters using response box filters to speed up the computations. More recently, Calonder et al. [1], developed the Binary Robust Independent Elementary Features (BRIEF), which uses only binary strings as an efficient alternative for feature point description. It makes use of the Hamming distance which is very efficient to compute when compared to the L_2 norm usually used. Other approaches such as [25] and [26] exist that use trained classifiers but they have the drawback of previously requiring a training phase.

Taking into account the good results documented in the literature regarding SURF and BRIEF , these are the main two approaches that are tested in the project and described on this section.

The remaining of this chapter is organized as follows. Section 3.2 describes the SURF algorithm as proposed by [21] . Section 3.3 explains the main differences of BRIEF when compared to SURF.

3.2 SURF - Speeded-Up Robust Features

3.2.1 Integral Images

The definition of *integral image*, or *summed area table*, was first introduced in [27] with the objective of making computer graphics textures tractable. In the current scope it will be useful to allow the fast computation of box type convolution filters.

Integral images is in an algorithm to calculate very efficiently the sum of values in a rectangular subset of an image. The integral image $I_\Sigma(x,y)$, with the same dimensions of the input image I, is constructed in such a way that each entry represents the sum of all input image pixels within a rectangular region delimited by the pixel at the coordinates (x,y) and the origin.

$$I_\Sigma(x,y) = \sum_{i=0}^{i\leq x}\sum_{j=0}^{j\leq y} I(i,j) \tag{3.1}$$

With this foundation, the calculation of the sum of the intensities over any rectangular area in very efficient. For example, considering the rectangular area $[\mathrm{ABCD}]$ in Figure 3.1, the sum of intensities Σ will be given by:

$$\Sigma = \mathrm{A} - \mathrm{B} - \mathrm{C} + \mathrm{D} \tag{3.2}$$

Hence, the calculation time is constant for rectangular areas of any size and takes only one sum and two differences.

3.2.2 Interest points detection

In the task of finding an object in an image, instead of searching for the object as a whole, it is usual to search for objects interest points. This type of approach is chosen for several reasons, from which the main

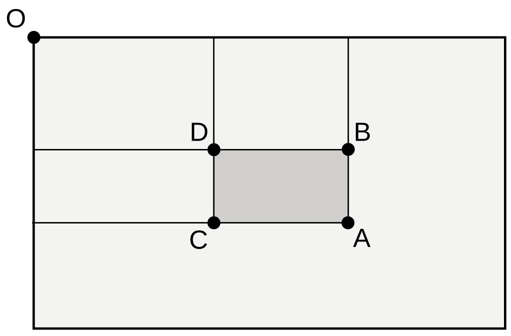


Figure 3.1: Using integral images, the calculation time of the sum of the pixels in the rectangular area delimited by $[\mathrm{ABCD}]$ is independent of its size and takes only one sum and two differences

ones are the computational cost of searching in such a high-dimensional data as the one stored in images, and the high level of redundancy incorporated, because pixels do not move independently and have a high level of correlation. There are several methods to define and detect interest points, ranging from the ones that consider corners as interest points to the ones that consider blobs instead. In this section the *fast Hessian* detector [21] will be decribed. Like Figure 3.2 shows, it detects blob-like features. For more details on others interest points detectors please refer to [28].

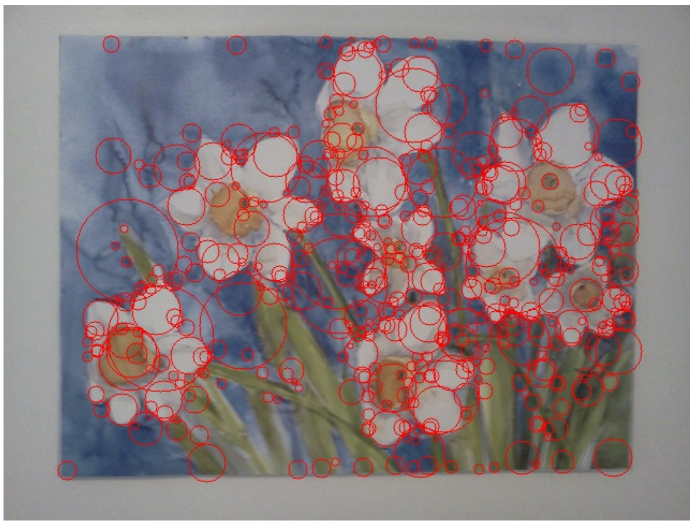


Figure 3.2: An example of the type of features that the Fast Hessian Detector detects

Fast Hessian Detector The fast Hessian detector detects blob-like features. It is based on the Hessian Matrix, which at scale σ is defined as follows:

$$\mathcal{H}(x,y,\sigma) = \begin{bmatrix} \frac{\delta^2}{\delta x^2}G(\sigma) * I(x,y) & \frac{\delta}{\delta x}\frac{\delta}{\delta y}G(\sigma) * I(x,y) \\ \frac{\delta}{\delta x}\frac{\delta}{\delta y}G(\sigma) * I(x,y) & \frac{\delta^2}{\delta y^2}G(\sigma) * I(x,y) \end{bmatrix}, \tag{3.3}$$

It is know that, in the continuous case, Gaussians are optimal for scale-space analysis [29, 30]. However, all Hessian-based detectors have a downside in general: when working with discrete images, Gaussians have to be discretized as well, as consequence there is a loss of repeatability under image rotations around odd multiples of $\frac{\pi}{4}$. Nevertheless, the Hessian matrix is chosen because the repeatability rate is still very high in any rotation angle [21]. As discretized Gaussian filters are already non-ideal and the convolutions are still very expensive, they are approximated by box filters (Figure 3.3). This way, when used together with integral images, the computations can be performed in constant time. Although this is an even stronger approximation, the performance is comparable or even better than with the discretized and cropped Gaussian [21].

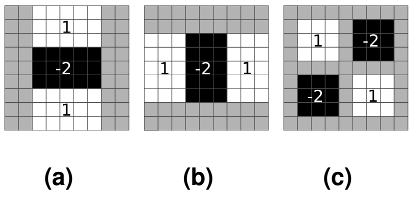


Figure 3.3: Box filters approximating Gaussians second order derivatives

Once the approximated Hessian matrix using box filters is calculated it is necessary to determine how 'strong' is the current point to classify it as of interest or not. For this purpose, the determinant of the exact Hessian matrix is used as a score measure. Because only the approximated Hessian matrix is available, an approximation for the determinant is also required:

$$\det(\mathcal{H}_{approx}) = D_{xx}(\sigma)D_{yy}(\sigma) - (wD_{xy}(\sigma))^2 \simeq \det[\mathcal{H}(x,y,\sigma)] \tag{3.4}$$

where D_{xx}, D_{yy} and D_{yy} are the results of convolving the image with the filters in Figure 3.3. The factor w takes the value 0.9 and is used to approximate more precisely $\det[\mathcal{H}(x,y,\sigma)]$.

Scale Invariance The previous calculations are performed at different scales because interest points may be compared between images where they are seen at different scales. The scale space is implemented as an image pyramid. Without box filters, usually, image pyramid is built by repeatedly smooth the image with a Gaussian and then sub-sample it in order to achieve a higher level of the pyramid. Fortunately, with box filters and integral images, there is not the need to filter the image iteratively and sub-sample it. Instead, it is the filter that is up-scaled and applied at exactly the same speed on the original image. The filters represented in Figure 3.3 are the initial scale layer, which is referred as scale $s = 1.2$. The higher levels of the pyramid are reached by gradually applying bigger filters. The scale space is divided into octaves. Each octave is a series of filter response map obtained by convolving the same input image with a filter of increasing size. Due to the discrete nature of box filters, for two successive levels, its size must be increased by a minimum of two pixels in order to maintain the presence of the central pixel.

The first octave starts with the 9×9 filter and then filter with sizes 15×15, 21×21 and 27×27 are applied. For each new octave, the filter size increase is doubled. So, in the second octave the size increase will be of 12 and so on. This lead to the octaves represented in Figure 3.4. The last octave is only calculated if the original image size is still larger than the corresponding filter sizes. Although, even more octaves can be analyzed, the number of interest points decays very quickly [21], so it is not worth the computational cost.

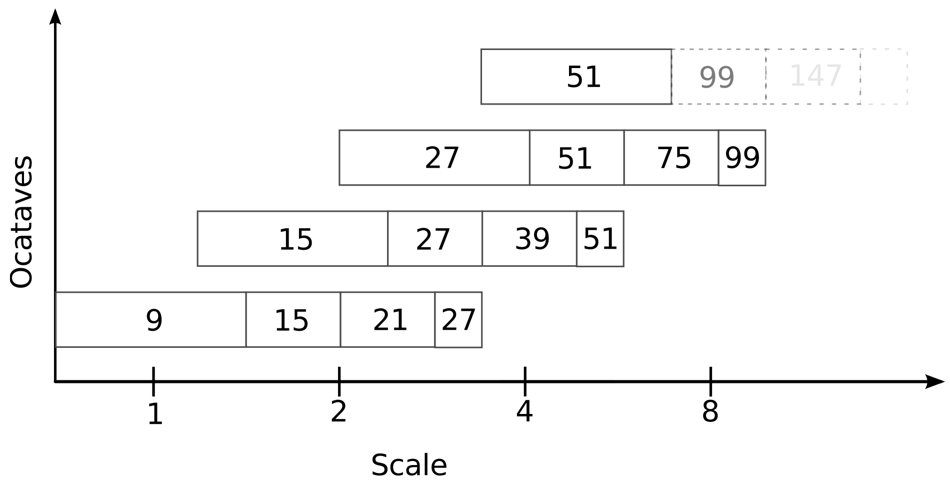


Figure 3.4: Graphical representation of the filter side lengths for four different octaves. The octaves overlap in order to cover all possible scales seamless

Interest point classification In order to classify a point a of interest or not a non-maximum suppression in a $3 \times 3 \times 3$ is applied in scale and image space (Figure 3.5). Each sample is compared to its 8 neighbors in the current image and the 18 neighbors of the “bigger” and “smaller” images in scale space. If it has the biggest score (Hessian matrix determinant) of its neighbors then it is considered an interest point, otherwise it is discarded.

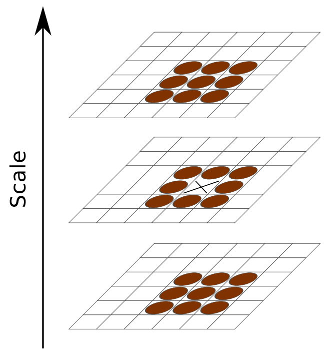


Figure 3.5: Graphical representation of the $3 \times 3 \times 3$ neighborhood

Once it is classified as an interest point, the location is refined to subpixel accuracy by fitting a parabola to the sample point and its immediate neighbors [31].

3.2.3 Interest points descriptors

After the detection of the interest points, it is necessary to assign a description to each one in order to identify and distinguish them from each other and to match them across images. Ideally, a good descriptor will provide a description to every point that is unique but identical for all possible views of the same point.

This ideal case is real hard to achieve, if not impossible. Nevertheless, local descriptors use information about the texture of the interest points neighborhood to distinguish them as much as possible from each other. Specifically, the SURF descriptor, described next, tends to perform very well in this task. The SURF descriptor is constructed in three step which are now described.

Interest point orientation assignment To achieve rotation invariance, the first step is to assign an orientation to the point so that, when it is seen from another perspective, it can be correctly matched. For this purpose, and to take advantage of the use of integral images, Haar wavelet filters are used (Figure 3.6). Being s the scale at which the interest point was detected, filters of size $4s$ are used and wavelet responses in x and y directions with a sampling step of s are calculated around a circular neighborhood of radios $6s$.

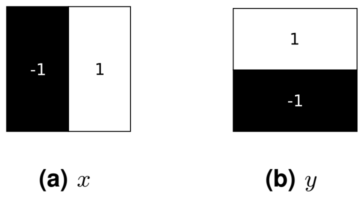


Figure 3.6: Haar wavelet filters

The filter responses are weighted with a Gaussian of $\sigma = 2s$ centred at the point and represented in Cartesian coordinates system. Using a orientation sliding window of $\frac{\pi}{3}rad$ (Figure 3.7), the sum of all responses within the window are calculated. The orientation of the window that has the greatest summed value is the orientation that is assigned to the point.

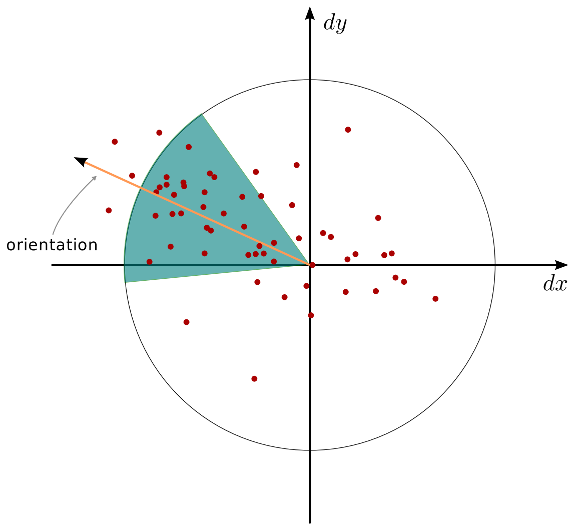


Figure 3.7: Sliding window used to assign the orientation

Descriptor vector Although there are few variations, the standard SURF descriptor consists of a vector with 64 entries. To build this vector, the first step is to construct a square region of size $20s$, centered at the interest point and with the orientation selected in the previous step.

This region is split up into 4×4 square sub-regions. Using Haar wavelets filters with size $2s$, the filter responses at 5×5 equally spaced sample points are calculated in the x and y direction. Note that these directions are defined in relation to the square region orientation. But instead of rotating the image itself, the filter responses are calculated in the unrotated image and then interpolated.

After weighting the filter responses using a Gaussian with $\sigma = 3.3$ centred at the interest point, four sums in each sub-region are calculated: the sums of dx and dy and, to have information about the polarity of the intensity changes (Figure 3.8), the sums of $|dx|$ and $|dy|$.

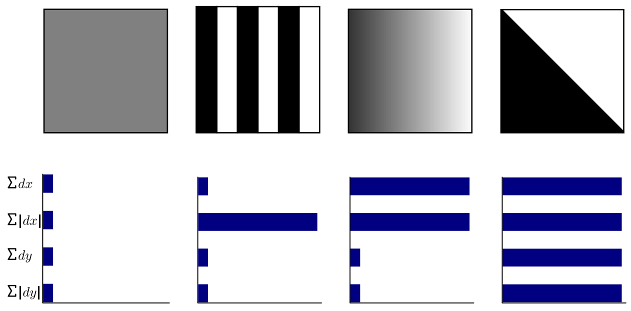


Figure 3.8: Descriptor entries of a sub-region.

3.2.4 Interest points matching

An interest point in one image, is considered to *match* another interest point in other image if they are close enough in the *nearest-neighbor* sense. The most widely used algorithm for nearest-neighbor is the kd-tree [32].

The kd-tree algorithm uses a binary tree in which every node is a k-dimensional point. Each non-leaf node can be considered as a generator of a hyperplane that divides the space into two subspaces. To define this hyperplane, one of the k-dimensions of the space is associated to the node and the hyperplane is defined in such a way that it passes through the node and is perpendicular to the associated dimension's axis. Points to the left of the hyperplane belong to the the left subtree and points to right belong to the right subtree. An example of a kd-tree for a 2-dimensional space is illustrated in Figure 3.9.

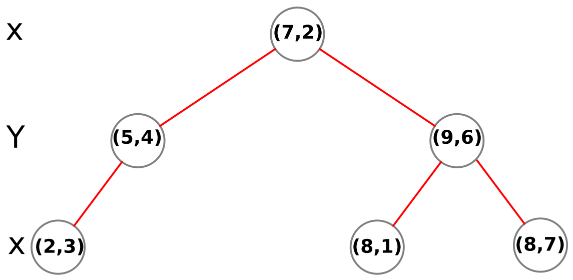


Figure 3.9: An example of a kd-tree

The canonical method to add an element to the tree consists on selecting, from the group of points to be inserted, the median one. Then, at each tree level, we cycle through the space dimensions used to select the hyperplanes. We move down to the left subtree if the point coordinate of the selected dimension is lower then the current node's one, or to the right subtree otherwise, until we reach a leaf. This will lead to a balanced tree. For example, the point (3,5) would be the right child of the node (5,4) on the tree of Figure 3.9.

Once the tree is constructed, one can find the nearest neighbor of a point by performing the following steps:

1. Start at the root node and move down the tree recursively, in the same way as if the point was being inserted.
2. Once on a leaf node, save it as the “current best“.
3. Now, go back the recursion of the tree, performing the following steps at each node:
    (a) If the current node is closer than the “current best“, then it becomes the “current best”.
    (b) Now, check whether there could be any points on the other side of the splitting plane that are closer to the search point than the “current best”. Conceptually, this is done by intersecting the splitting hyperplane with a hypersphere around the search point that has a radius equal to the current nearest distance. To do that and since the hyperplanes are all axis-aligned, simply check if the difference between the splitting coordinate of the search point and current node is less than the distance (overall coordinates) from the search point to the current best.
        i. If the hypersphere crosses the plane, there could be nearer points on the other side of the plane, so we must move down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search.
        ii. If the hypersphere does not intersect the splitting plane, then continue walking up the tree, and the entire branch on the other side of that node is eliminated.
4. Once this process is completed for the root node, then the search is complete.

Finding the nearest neighbor is an O(log N) operation in the case of randomly distributed points and considering that the tree is balanced. This works well for exact nearest neighbor search in low-dimensional data. Unfortunately, when the number of points is only slightly higher than the number of dimensions, the algorithm is only slightly better than a linear search of all of the points.

Several modifications, like [33], have been made to this algorithm to improve its performance with the trade-off of finding only the approximate nearest neighbor instead of the exact one.

For the current work we chose to use the FLANN library [34]. It is a C++ library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

3.3 BRIEF - Binary Robust Independent Elementary Features

In this section, the main differences of BRIEF from SURF are described.

3.3.1 BRIEF descriptor

The BRIEF descriptor differs from the SURF descriptor in the sense that it directly computes binary strings from image patches. Each bit is obtained by comparing intensities of pairs of points. Also, by constructing the descriptor this way, it enables the possibility of using the Hamming distance to compare strings. The use of this distance has the advantage of only requiring a bitwise XOR operation and a bit count which can be performed much faster than other standard distance measures.

Patch smoothing Because BRIEF tests are performed in single pixel pairs, if they are performed in the original patch without any form of preprocessing, the performance will be very sensitive to image noise. To avoid this dependence, the image patch is previously smoothed with a Gaussian kernel. This way, the descriptor stability and repeatability are increased. The tests performed in [1] show that using a Gaussian kernel with variance of 2 leads to the best results, and this will be the value used.

Intensity test The test τ to determine each descriptor's bit is defined as follows:

$$\tau(\mathbf{p};\boldsymbol{x},\boldsymbol{y}) := \begin{cases} 1 & \text{if } \mathbf{p}(\boldsymbol{x}) < \mathbf{p}(\boldsymbol{y}) \\ 0 & \text{otherwise} \end{cases}, \tag{3.5}$$

where $\mathbf{p}$ is the $S \times S$ image patch and $\mathbf{p}(\boldsymbol{x})$ is the smoothed pixel intensity at coordinates $\boldsymbol{x} = (u,v)^T$. So, if the pixel intensity at $\boldsymbol{x}$ is lower than the one at $\boldsymbol{y}$ the correspondent bit will be 1, otherwise it will be 0.

Spatial arrangement of tests Defined the individual test, it is necessary to decide the patch positions at which the tests will be performed. The authors of [1] tested several sampling geometries (Figure 3.10) and found that the one that leads to the highest recognition rate is the one that samples the tests positions from an isotropic Gaussian distribution, $(X,Y) \sim$ i.i.d. Gaussian$(0, \frac{1}{25}S^2)$. (Figure 3.10(b)). The locations following this distribution are calculated on the initialization phase of the algorithm, from there, the locations will be the same for every feature descriptor that is calculated.

Descriptor bitstring Finally, having the location pairs $(\boldsymbol{x},\boldsymbol{y})$ defined, the n_d-dimensional descriptor string is defined as

$$f_{n_d}(\mathbf{p}) := \sum_{1 \le i \le n_d} 2^{i-1}\tau(\mathbf{p};\boldsymbol{x}_i,\boldsymbol{y}_i). \tag{3.6}$$

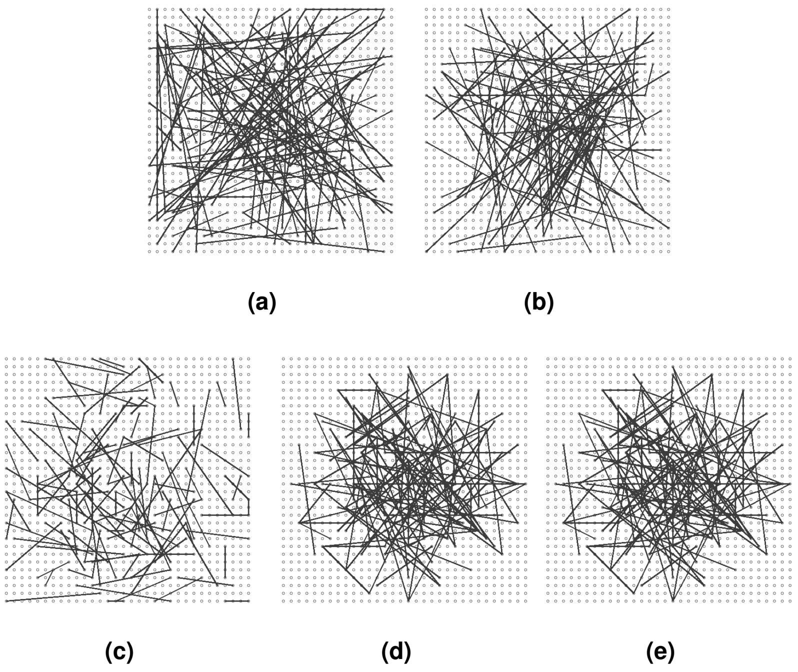


Figure 3.10: Different approaches to choosing the test locations. All except the righmost one are selected by random sampling. Showing 128 tests in every image. Figures adapted from [1].

The authors of [1] performed several tests with $n_d = 128$, 256 and 512. They conclude that, for image pairs with short baseline, using $n_d = 256$ yields near optimal results and that for all other cases $n_d = 512$ perform better.

3.3.2 Feature detector

Calonder et al., do not propose any special feature detector, any option available on the literature can be used. The chosen one was FAST (Features from Accelerated Segment Test) [2].

FAST is based on the Accelerated Segment Test (AST). This test, which is simplified version of SUSAN (Smallest Univalue Segment Assimilating Nucleus) [35], consists on defining a circle of radius r around the candidate point (Figure 3.11). Then, if at least n contiguous pixels are all brighter or all darker than the center pixel by at least t, the point is considered a feature. Specifically, FAST's tests shown that better results are achieved by using $r = 3$ (circunference of 16 pixels) and $n = 9$. The order by which the circle pixels are tested can speed up considerably the overall performance. This order is chosen by building a decision tree from the ID3 algorithm [36] applied to a training set of images. Contrasting to SURF's Fast Hessian Detector, that detects blob-like features, FAST detects corner-like features.

3.3.3 Scale and Rotation Invariance

On the original work from Calonder et al., BRIEF is not invariant to scale change or rotation. To achieve that, a pyramidal approach was used. When the template image is being processed, several copies of

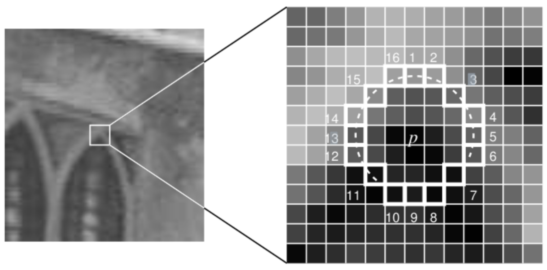


Figure 3.11: Illustration of the test circle. The pixel at p is the center of a candidate corner. The arc indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the t. Figure adapted from [2].

the original image are generated by scaling it and rotating within a 360 degrees span. To each of these generated imaged FAST and BRIEF algorithms are applied. This will lead to several descriptors of the same features but rotated and scaled, thus, achieving scale and rotation invariance.

4 Developed Algorithm and System Architecture

Contents

4.1 Introduction

In this chapter, the practical implementation of the work developed is addressed. It will be explained how all the theoretical knowledge acquired is used to propose an algorithm that successfully estimates a camera pose, the hardware where it was implemented and respective architecture.

Concerning the main algorithm developed, it is divided into two main parts. The first one, consists in detecting the template plane present on a video stream. The two features detectors and descriptors described before were implemented in order to do so. By doing this, it will be possible to compare them, regarding computational efficiency, robustness and precision (see Chapter 5). The second main part of the developed algorithm consists of using the coordinates of the frame already detected on the video stream to estimate the pose of the camera that is acquiring it. This is done by calculating the homography matrix that maps the coordinates of the template plane detected on the current frame into the coordinates of the template plane on the reference frame. With this matrix, the homography decomposition method is applied and two possible solutions, each one consisting of a translation vector and a rotation matrix, are calculated. The true solution is chosen by, either using the IMU data of the vehicle where the camera is attached, or by using a third image containing the reference plane too but captured at a different pose.

Apart from some conceptual tests performed on the Matlab software due to easiness of development, all the code was written using the C++ language. Although being a low level language, it is much more computationally efficient. Because, when working with images, great part of the calculations are performed using matrices and C++ does not natively provide an API to easily use these data structures, the OpenCV library [37] was also used. It is a computer vision library that is free for academic and professional use and provides an API with several computer vision algorithm already implemented.

Regarding the hardware where the code will run, because one of the final objectives of the developed algorithm is to be used on board of aerial vehicles, it was necessary to define a commitment between cost, computation power and size. It is required that the hardware is lightweight enough, but at the same time powerful enough to, hopefully, run the code at real time speed. Considering these requirements, the hardware that seemed more appropriate was the Gumstix Overo Fire. It belongs to the COM (Computer On Module) hardware class. Basically, it has an ARM Cortex-A8 CPU, RAM memory, storage memory and networking capabilities, all on a single board with an approximate size of a gum stick. It runs the Linux OS, and many expansion boards to extend the capabilities are available. For capturing video, firstly the e-cam50 camera module (see Appendix A.1) was considered, but due to better technical specifications the Caspa VL was chosen.

Although, as already stated, the main objective is to use the developed application on board of aerial vehicles, and a quadrotor is available at ISR labs. This was not the test bench used. Getting a quadrotor in the air requires a team of several people, and this is not practical to perform every time there is the need test some modifications on the code. Instead, a robotic arm was used. It is more practical to operate, requires only a single person and is conceptually identical to estimate the pose of a quadrotor, or the pose of the camera attached to the arm's end effector. The robotic arm available on the university facilities is the PUMA 500 model from Unimation with 6-DOF.

The remaining of this chapter is organized as follows. Section 4.2 contains a flowchart of the developed algorithm and explain the order of the steps performed. Sections 4.3, 4.4 and 4.5 describes with more detail all the hardware used. Finally, Section 4.6 shows all the hardware is connected, it provides a global view of the testing setup architecture.

4.2 Developed Algorithm

The developed algorithm works as follows (see flowchart on Figure 4.1): the program starts acquiring frames from the camera. When the camera is at the desired pose the user sends a command and the template image is stored and learned by the chosen method (either SURF, BRIEF or TAG features are extracted). Then the main loop starts. A new frame is acquired and the features are extracted. Now, the features of the current frame are matched with the template frame features. Matched features coordinates are undistorted using the camera calibration matrix and distortion coefficients. The homography matrix is calculated using RANSAC and then decomposed into rotation matrix and translation vector. From the two possible solutions, the correct one is chosen based, either on a previously second frame taken from the same template or on the vehicle IMU. From the correct solution, the control command is calculated using the chosen control law and fed into the vehicle actuators. This procedure is repeated until the user requests the program to be stopped.

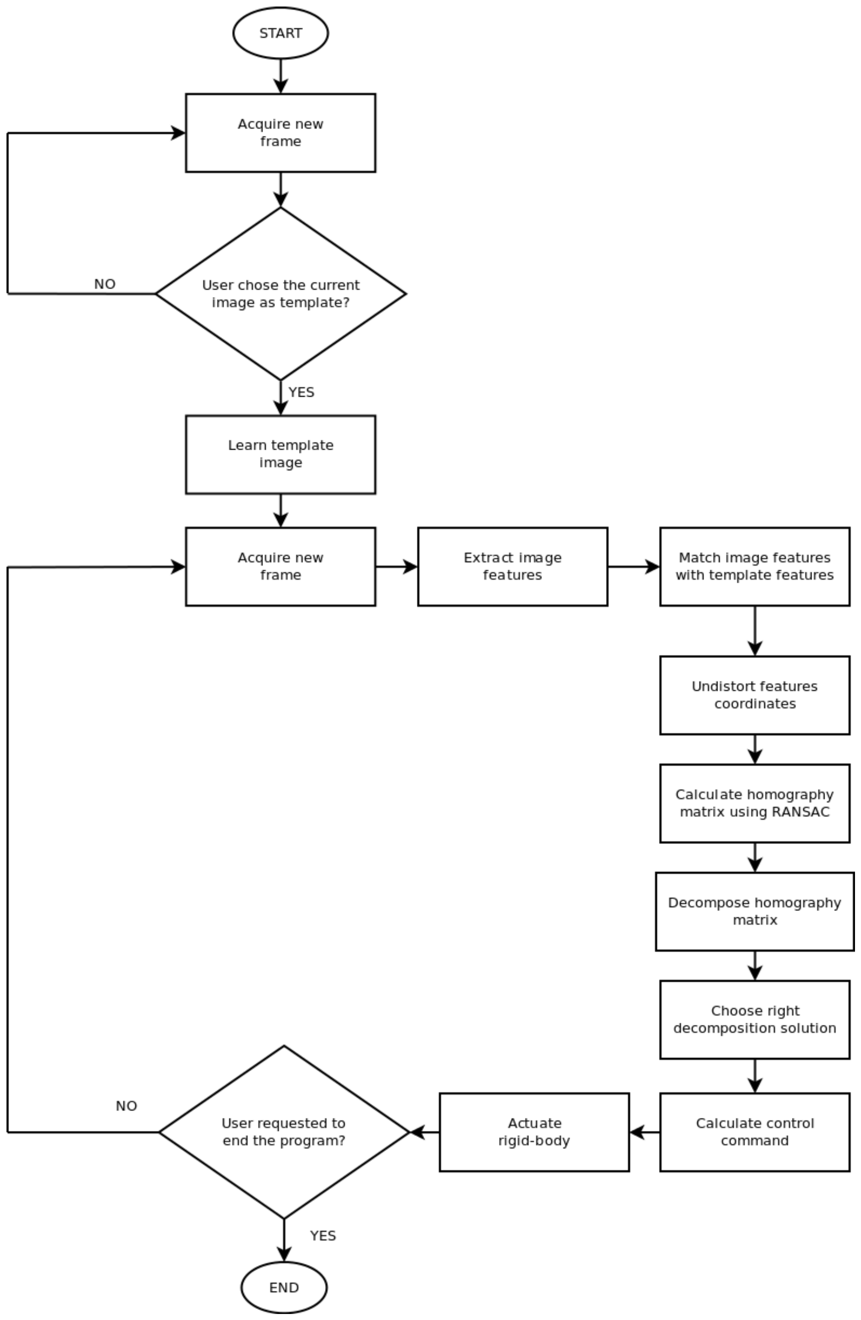

START
Acquire new frame
User chose the current image as template?
NO
YES
Learn template image
Acquire new frame
Extract image features
Match image features with template features
Undistort features coordinates
Calculate homography matrix using RANSAC
Decompose homography matrix
Choose right decomposition solution
Calculate control command
Actuate rigid-body
User requested to end the program?
NO
YES
END

Figure 4.1: Developed algorithm flowchart.

4.3 Gumstix Overo Fire

The Gumstix series computers are very small, general purpose computers for embedded systems applications shiped with Linux 2.6 operating system. The name "gumstix" is due to the fact that each one of these computers are a COM (Computer On Module) with dimensions really close to a common stick of gum. The model used in this particular work is the Gumstix Overo Fire COM (Figure 4.2). It consists of a OMAP3530 processor from Texas Instruments (TI) based on ARM Cortex-A8 architecture with a working frequency of 600Mhz a DSP. Also, it features a 256MB RAM unit, a 256MB flash drive and Bluetooth and WiFi connectivity.

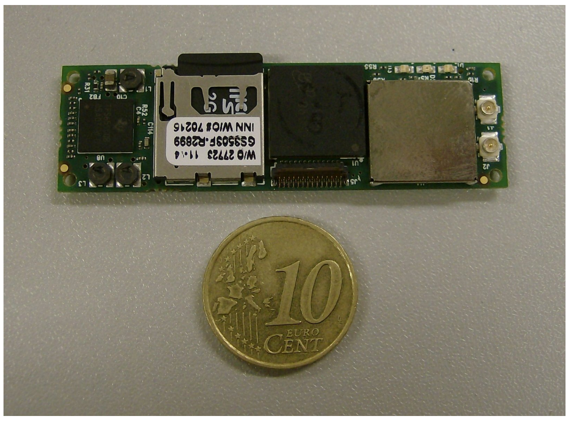


Figure 4.2: Gumstix Overo Fire COM board

To provide the COM basic communication interfaces and expand its functionalities small expansion boards exist that are attached to the main COM board. The model chosen was the Tobi expansion board (Figures 4.3 and 4.4). It supports numerous interfaces such as 10/100baseT Ethernet, DVI-D (HDMI), USB OTG mini-AB, USB host standard A, Stereo audio in/out, USB Serial Console among others.

All these flexible features and processing capacities make the Gumstix Overo Fire a powerful and versatile COM very suitable for our project.

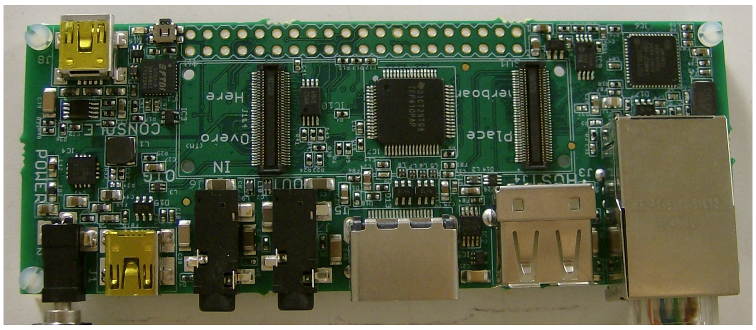


Figure 4.3: Tobi expansion board

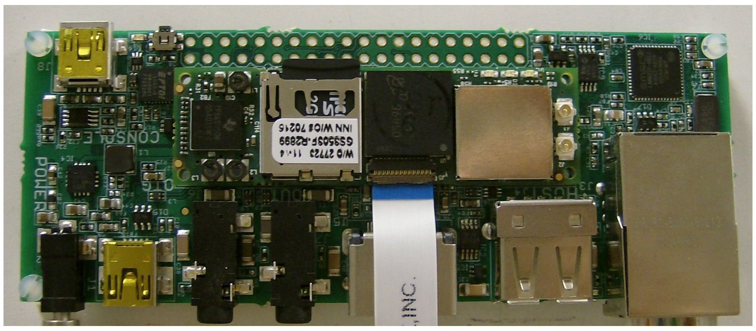


Figure 4.4: Gumstix Overo Fire attached to the Tobi expansion board

4.4 Caspa VL Camera

To integrate imaging capacities with the Gumstix Overo Fire, a Caspa VL Camera module is connected to it (Figure 4.5). It consists of an ultra-compact camera which weights only 22.9 g. It can capture images with up to 720×480 pixels of resolution with a framerate of 60 fps. Due to its technical features and to the fact that it is an affordable and lightweight hardware, it was chosen to be used in this project.

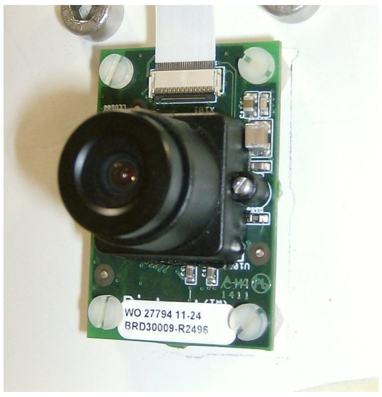


Figure 4.5: The Caspa VL camera module

The mechanical support that hold the Overo Fire together with the Caspa VL Camera is hand made and it is shown in Figure 4.6. This module provides the protection and robustness needed to easily attach and detach the Gumstix to a vehicle.

This set will, from now on, be called the *Gumstix-and-Camera module* (G-C module).

4.5 PUMA 560 Robotic Arm

After simulating the algorithms behavior, it is necessary to test them with real data in a real context. Performing these tests with real data directly on the Quadrotor could be very dangerous, as any malfunction could lead to damages or even total loss of the UAV. A more controlled and secure environment consists of assembling the Gumstix-and-Camera module on a robotic arm for tests purpose. With this platform, it is possible to test all the algorithms, to analyze their performance, improve them and correct any errors

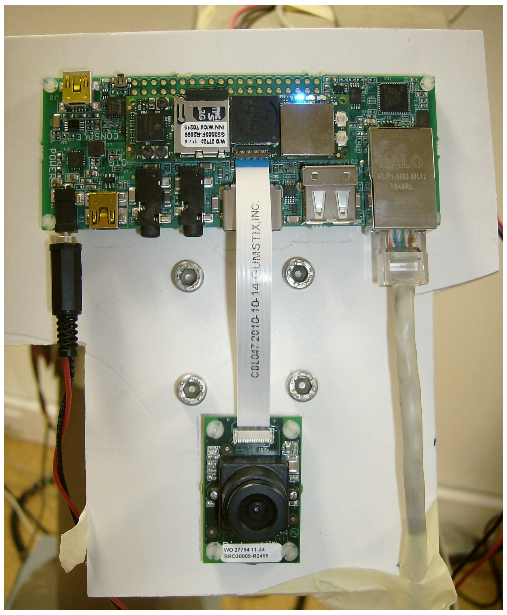


Figure 4.6: The Gumstix-and-Camera module

without jeopardizing the hardware.

The robotic arm available for tests on ISR lab is the PUMA (Programmable Universal Machine for Assembly) 560 robot (Figure 4.7). It is a chain of links and joints creating a 6-DOF (degrees of freedom) robotic arm with a DC servomotor controlling each joint. Encoders at each motor allow to acquire reads of each joint's position. It is similar to a human torso, shoulder, arm, and wrist and has 86 cm reaching and 2.5 Kg payload capacity. Table 4.1 shows each joint limits.

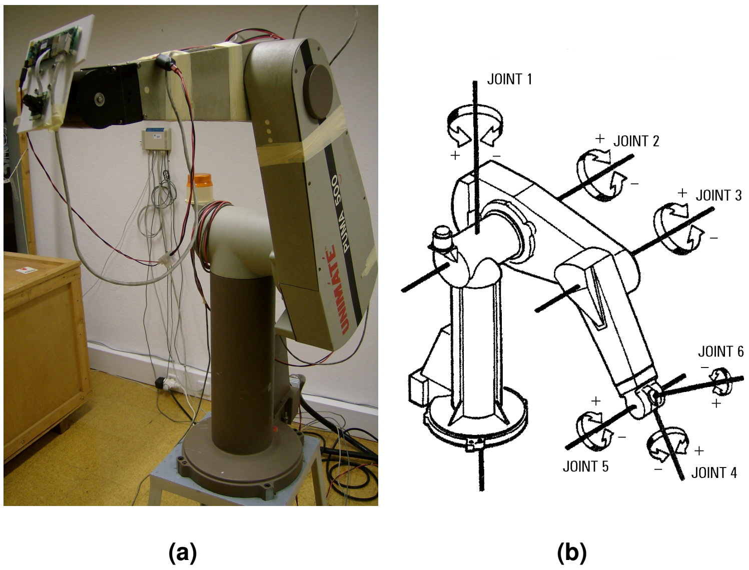


Figure 4.7: The Puma 500 Robotic Arm. The end-effector was disassembled and in its place, the Gumstix-and-Camera module, was attached.

Connected to the robotic arm is the Mark III Controller computer (Figure 4.8) which is the master com-

Table 4.1: Joint limits

Joint	Limit
Joint 1	$320°$
Joint 2	$250°$
Joint 3	$270°$
Joint 4	$280°$
Joint 5	$200°$
Joint 6	$532°$

ponent of the electric system and the arm controller. All signals to and from the robot pass through this computer. It supports the VAL programming language to perform real-time calculation to control the position of the robot. It also has status indicators and controls to power and command the arm in the front panel (Figure 4.9).

Originally, this was the only mean of controlling the robot. Currently, in order to make the system more easily modifiable and easy to integrate with other programs and algorithms, a common PC running Linux OS is connected to the controller computer using a serial port. This enables and facilitates the integration with programs written in C or C++, for example, as it is the case in this project.

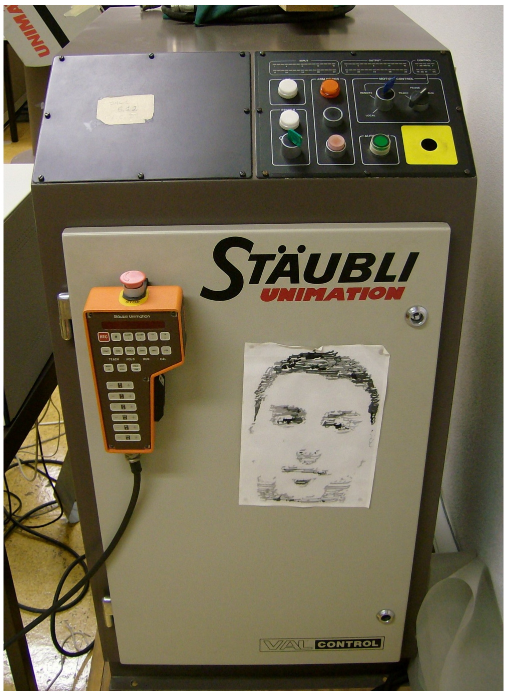


Figure 4.8: The controller computer.

When working with PUMA it was possible to understand that its operation conditions are not ideal anymore. Maybe due to the fact that it is a 15 years old robotic arm and no regular maintenance has been made. For example, when a single joint is ordered to move the entire arm “shakes” a bit. Also, if the end effector is ordered to go to a specific position, and then joints readings are requested, the reported position has an error in the order of centimeters.

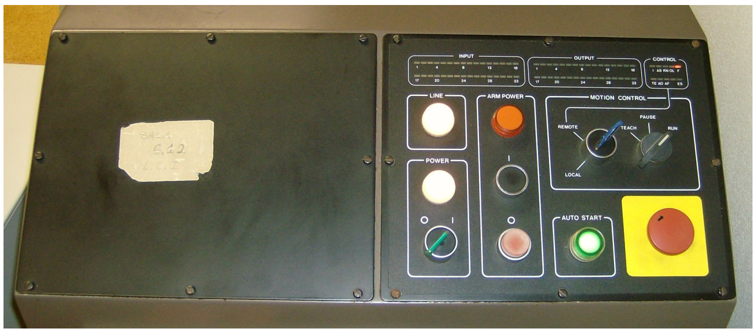


Figure 4.9: The controller computer front panel.

4.6 Overall Architecture

The architecture of the test bench created is illustrated in Figure 4.10. As already stated, the controller computer is only used as interface to the robotic arm. Is the PUMA PC that does the calculations regarding the inverse kinematics (conversion from the desired arm pose in the Cartesian Space to the individual joints desired positions). This PC receives the desired positions from the main laptop. By its turn, the main laptop is connected to the G-C module. Two options exist, the main laptop can do all the calculations for estimating the camera pose and the G-C module is only used as a simple network camera that sends the captured images over the network, or, all the calculations are performed on the G-C modules and the main laptop only operates as an interface. Either way, after each frame is processed, the mail laptop send the desired next position to the PUMA PC that, by its turn, order the PUMA arm to move.

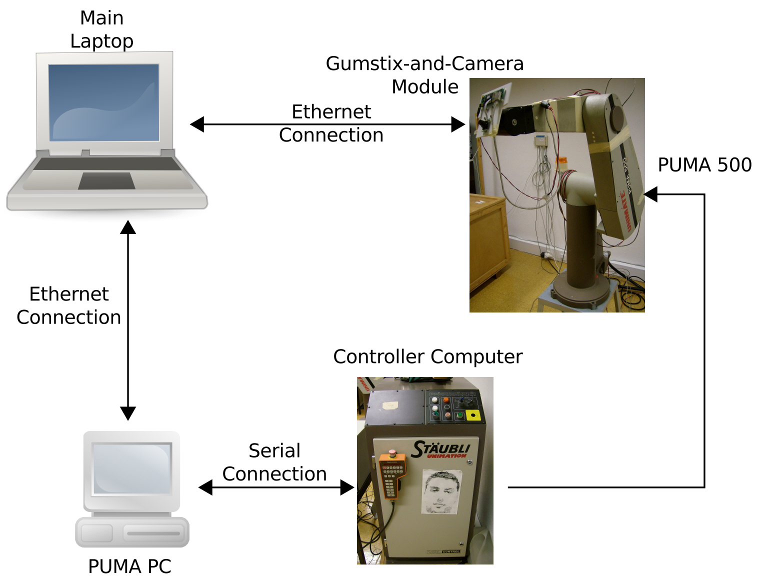


Figure 4.10: Gumstix Overo Fire attached to the Tobi expansion board

5 Experimental Results

Contents

5.1 Introduction

To demonstrate the capabilities of the system developed, several tests were performed using the test bench described in Chapter 4. First, synthetic videos that simulate what a real camera would see when it moves around with a plane always inside the FOV were generated. These videos are useful to show that all the calculations are correct since the exact ground truth data is available. Next, videos trying to replicate the synthetic ones, were captured using a real camera. This time, no ground truth data is available. These videos were used, essentially to test the behavior and robustness of the two features extractors and descriptors, SURF and BRIEF. The third test set consists of videos captured using the Caspa VL attached to the PUMA arm end-effector. While the videos were being captured, a log containing the readings of the arm joints positions was saved. Although not exact, this will be used as a form of ground truth for comparison purposes. The final test shows that closing the loop using the control law chosen, it is possible to make the arm end effector converge to the pose at which the template image was taken.

Considering computation times, table 5.1 presents the time needed to process one frame of 752×480 pixels resolution either by using SURF or BRIEF on a Dual Core at 2.2GHz PC and on the Gumstix. SURF requires much more time than BRIEF. On PC real-time speed is achieved, but on Gumstix, even when choosing brief, only an average of 3-4 fps.

Table 5.1: Computation times

Algorithm	PC	Gumstix
SURF	256 ms	5 s
BRIEF	27 ms	330 ms

Regarding SURF, 85% of processing time is spent calculating the features descriptors, thus representing the bottleneck. Almost all the operations on this task are floating point, which are not very efficiently performed on the Gumstix CPU.

5.2 Simulation

5.2.1 Synthetic Dataset

We want to simulate what a real camera sees if it travels along a certain trajectory. By defining the signals of rotation and translation over time, we can calculate the respective homography that describes the transformations suffered by the image seen by the camera.

So, to build the dataset we need to generate a movie that shows what a real camera would capture. We start by defining the “real camera”, i.e., defining the camera calibration matrix:

$$\mathsf{K} = \begin{bmatrix} 547.09 & 0 & 330.11 \\ 0 & 547.77 & 250.60 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

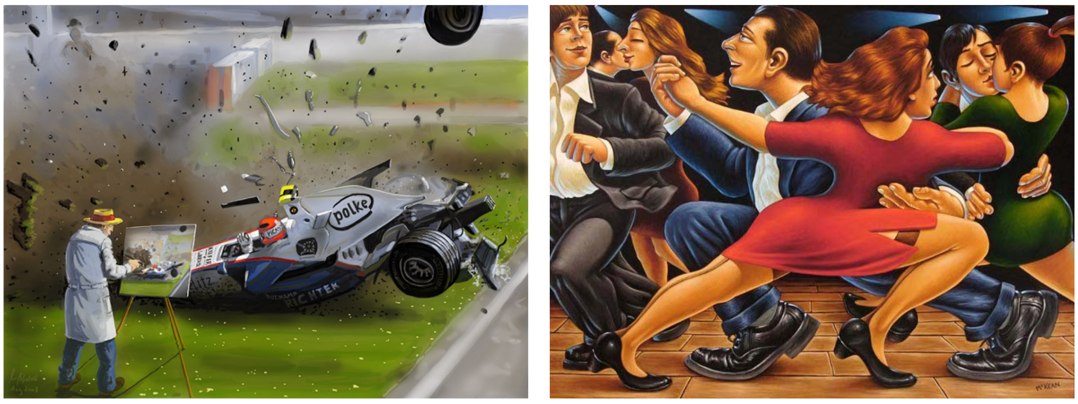
Any matrix could be chosen, but we decided to use one similar to a true calibration matrix of a webcam available in the lab. Next, a template image is chosen. Then, starting from this image, the homographies that correspond to the rotations and translations chosen are calculated using equation 2.25. To each of these homographies the following transformation is applied to take into account that the image is uncalibrated:

$$T = \mathsf{K}H\mathsf{K}^{-1}. \tag{5.2}$$

These consecutive perspective transformations are applied to the template image. And, this way, the movie frames are generated.

The purpose of these movies is to test the correctness and the robustness of the solution given by the algorithm developed. Which is possible because the true exact ground truth is available (actually is chosen) and can be used to calculate the error of the solution found by the algorithm developed.

The actual dataset is composed of 10 movies, 5 different trajectories each applied to two different template images. The template images are the ones in Figure 5.1.



(a) Scene 1 **(b)** Scene 2

Figure 5.1: The two scenes used to generate the synthetic datasets.

The the purpose of five different trajectories is to evaluate the behavior of the algorithm under various conditions. These are represented in Figure 5.3 and described next:

- **Pure Rotation dataset:** The camera is rotated along its y axis.
- **Scale Change dataset:** The camera is moved along its z axis without rotating. This causes the template on the image to get consecutively small. To test the scale invariance of the descriptors.
- **Pure Rotation Z dataset:** The camera is rotated along its z axis. To test rotation invariance of the descriptors.
- **Perspective Distortion dataset:** Starting perpendicular to the object, the camera moves down in an arc resulting in strong perspective distortion

- **Unconstrained dataset:** Different types of movements, with the constrain of the template remaining in the FOV.

It worth to mention that because the axis-angle representation was chosen, to avoid the degenerate case, none of the trajectories contain a pose where there is no rotation. For example, in the Scale Change dataset there is a constante rotation along the z axis of 10 degrees.

Some examples of the movie frames are shown in Figure 5.2.

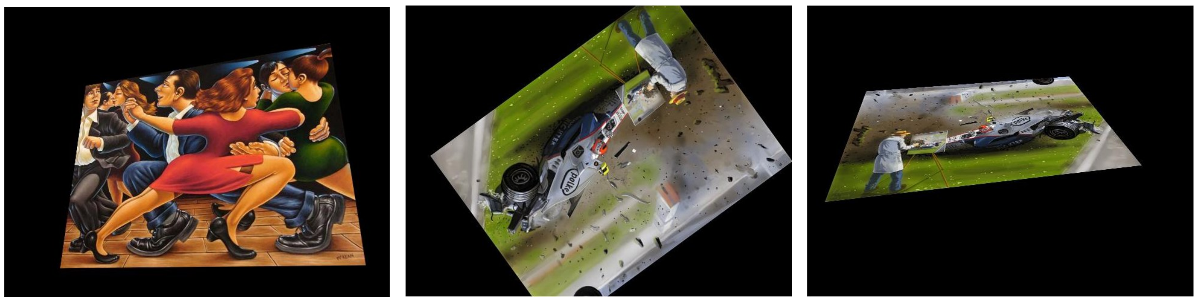


Figure 5.2: Some examples of dataset frames.

5.2.2 Results Analysis

In order to evaluate the precision of the algorithm, some performance indexes had the be defined. Considering the translation vector error, and taking into account that, by construction, the scale factor is always unknown, we can only evaluate the correctness of the direction. So the angle between the true and calculated translation vector was used as the error measurement. For the rotation, the axis-angle representation was chosen. The error between the true and the calculated rotation angle, as well as the angle between the true and the calculated rotation axis, are used as error measurements.

The three graphics for each of the five trajectories are represented in Figures 5.4, 5.5 and 5.6. The results were pretty similar for both template images so, the errors represented are the average of both templates errors. Both SURF and BRIEF descriptors were tested. If the plots are not continuous and there are points missing it means that the algorithm failed to detect the template in the correspondent frame.

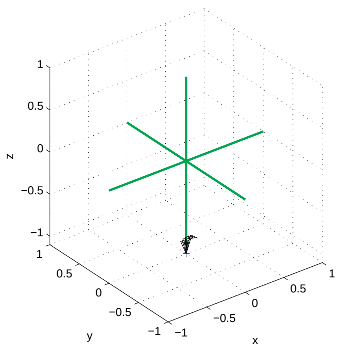
Regarding the Pure Rotation dataset on Figures 5.10(a), 5.10(c) and 5.4(e) it can be seen that the translation error is relatively small, between 1 and 7 degrees for both BRIEF and SURF. The rotation axis error is higher when the angle of rotation tends do zero. But, at the same time, because the rotation angle itself tends to zero along with the rotation angle error, this becomes negligible.

In the Scale Change data set (Figures 5.10(b), 5.10(d) and 5.4(f)), both BRIEF and SURF remain more or less precise until the template is reduced by a scale factor of 0.4. The translation error oscillates between 1 and 7 degrees and, once again, the rotation axis error is negligible. From a scale factor of 0.4 and beyond, BRIEF fails to detect the template and the SURF estimation becomes more unstable, although still reliable.

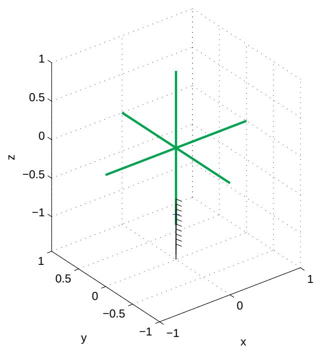
Considering the Pure Rotation Z dataset (Figures 5.5(a), 5.5(c) and 5.5(e)), BRIEF and SURF are both very invariant to rotations, with translation error between 0 and 4 degrees and rotation error almost always zero. Still, SURF performed slightly better around rotation angles of 90 degrees.

The Perspective Distortion dataset in Figures 5.5(b), 5.5(d) and 5.5(f) shows that with angles greater than 50 degrees the estimations became unreliable. For angles between 0 and 50 degrees, translation errors are no greater than 13 degrees, rotation axis errors are almost always lower than 15 degrees, just like the rotation angle error. SURF estimation are little more accurate.

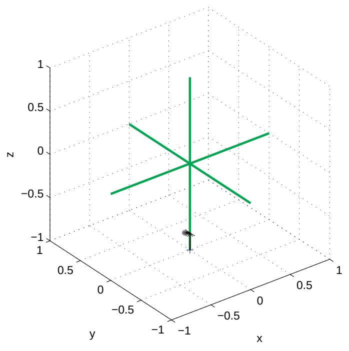
Finally, the Unconstrained data set (Figures 5.6(a), 5.6(b) and 5.6(c)) it is used just to see how the estimations performance on a trajectory that mixes some of the former conditions. The errors are almost always low, except around frame 40 where a mix of strong scale change with perspective distortion is used. Because in this case the SURF estimation is still accurate, we can conclude that was scale change that caused BRIEF to fail.



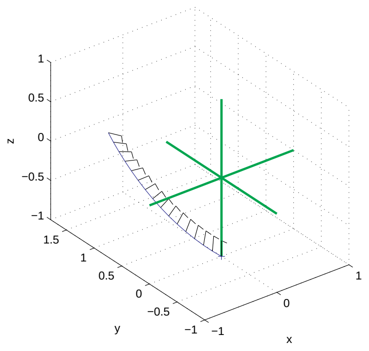
(a) Pure Rotation dataset



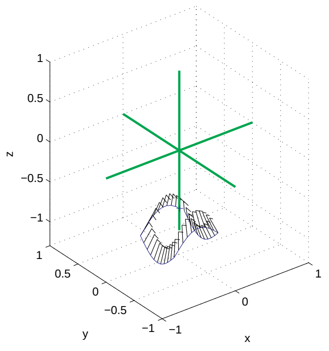
(b) Scale Change dataset



(c) Pure Rotation Z dataset

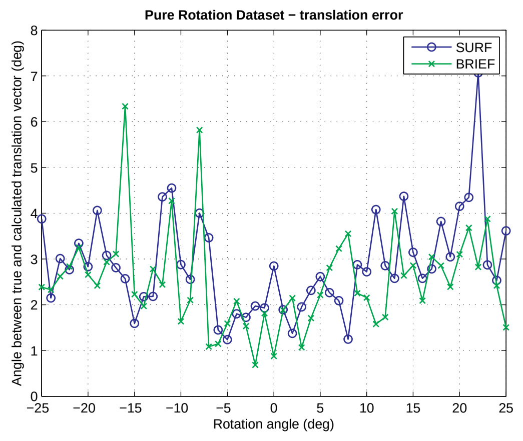


(d) Perspective Distortion dataset

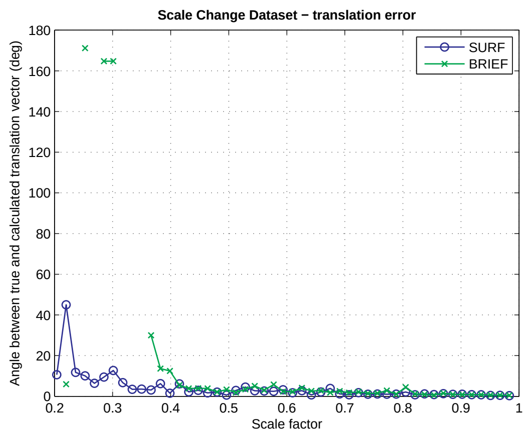


(e) Unconstrained dataset

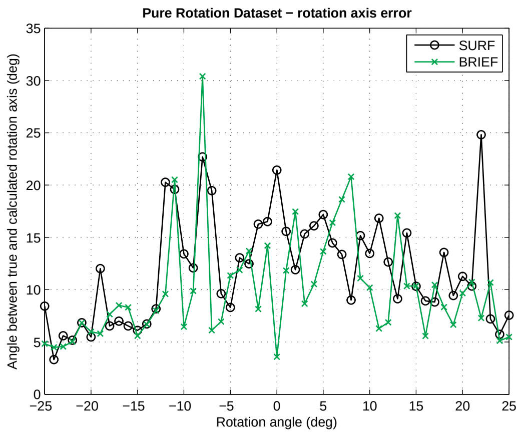
Figure 5.3: The datasets trajectories used to perform the tests.



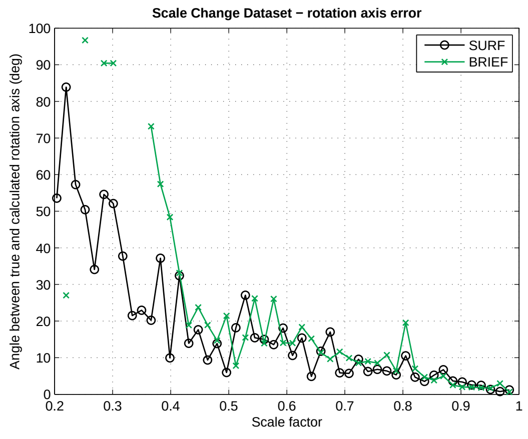
(a)



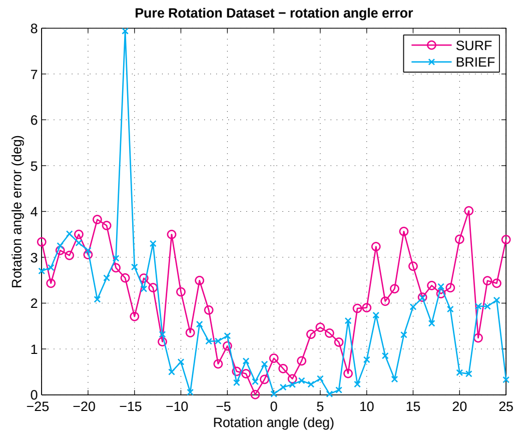
(b)



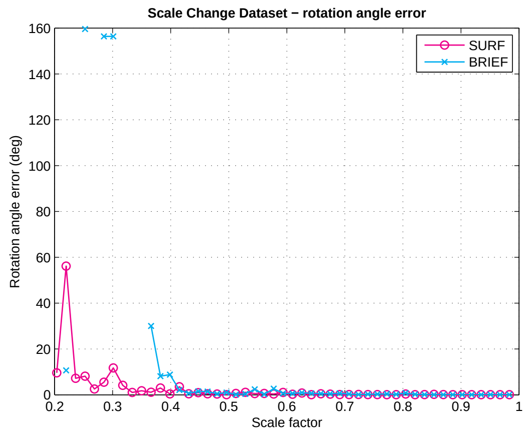
(c)



(d)

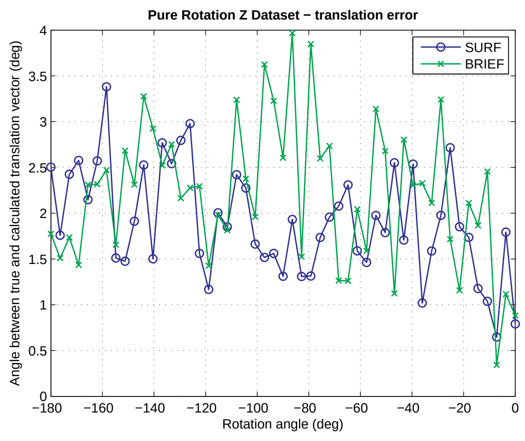


(e)

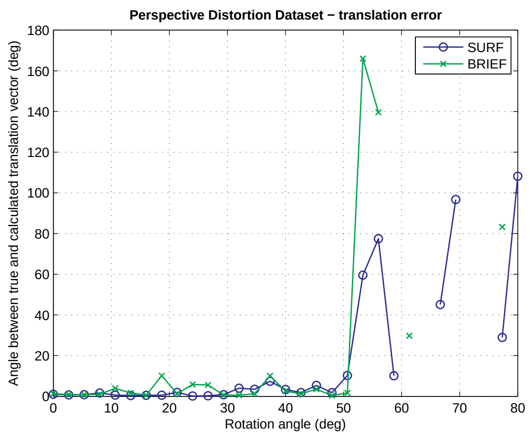


(f)

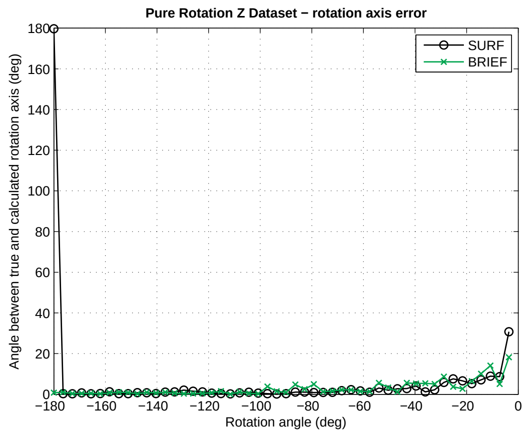
Figure 5.4: Error of pose detection under several different conditions.



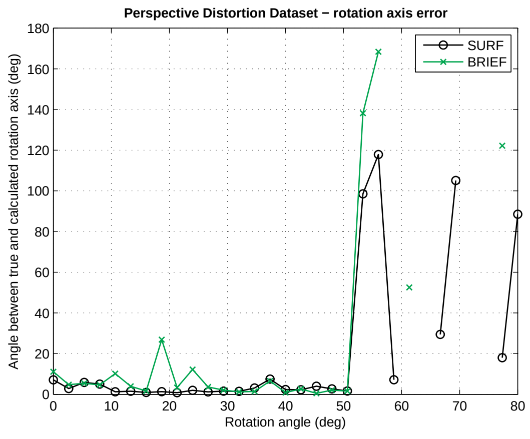
(a)



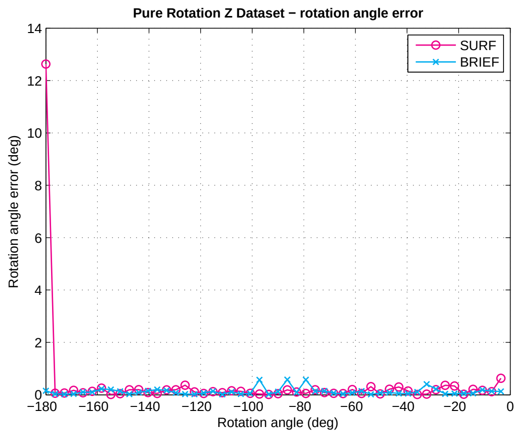
(b)



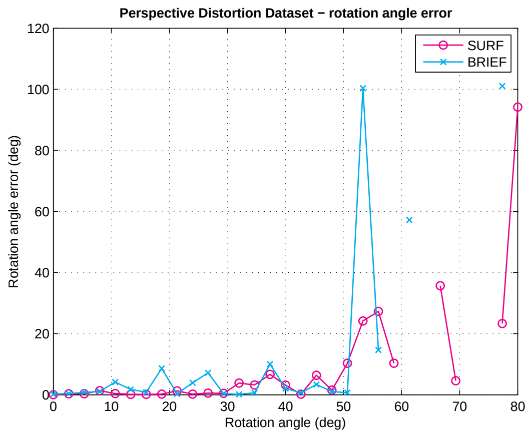
(c)



(d)

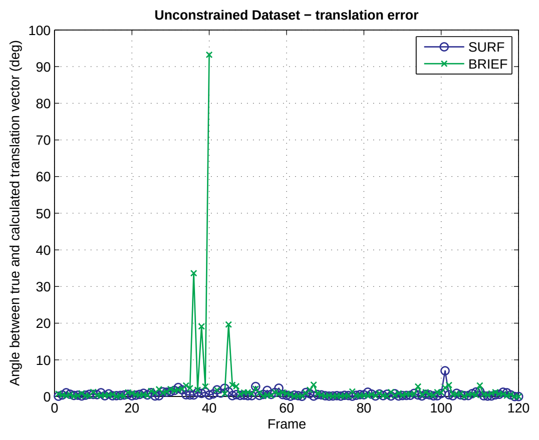


(e)

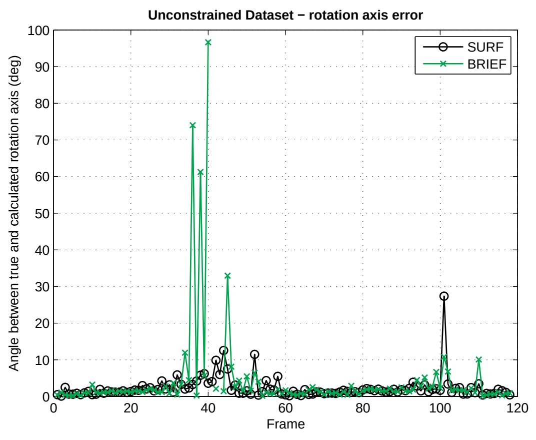


(f)

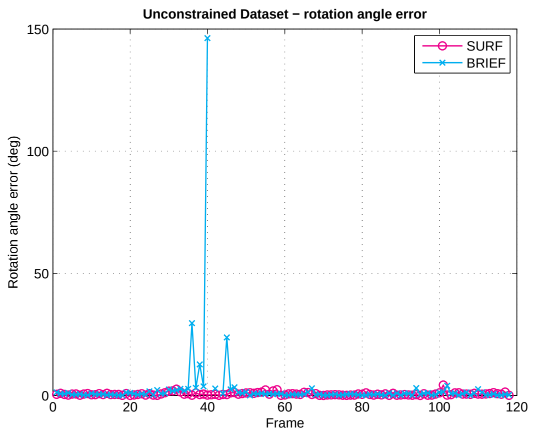
Figure 5.5: Error of pose detection under several different conditions.



(a)



(b)



(c)

Figure 5.6: Error of pose detection under several different conditions.

5.3 Real Videos tests

5.3.1 Real Dataset

In these tests, the videos were captured with a Logitech HD Pro Webcam C910 at 640x480 pixels resolution. This camera has little to no radial distortion and was calibrated using the method described in Appendix A.2 The purpose of this data set is to confirm with real data what we have seen with the synthetic videos of the previous experiment: that the algorithm developed can detect a template in a movie, calculate the homography that maps the present frame to the template one and determine the camera pose.

With this objective in mind, the template used was the one shown in Figure 5.7.

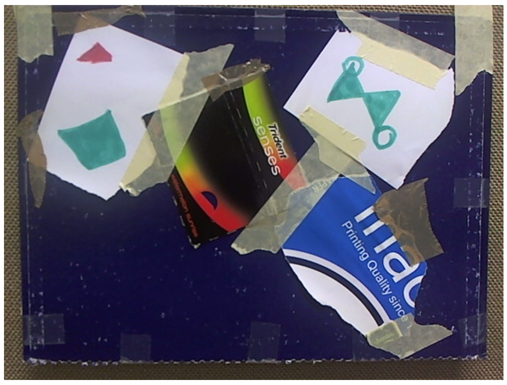


Figure 5.7: The image template.

To capture the movies, the camera was hand-held. Nevertheless, there was a special effort to try to make smooth trajectories and to replicate the same type of conditions present on the synthetic dataset, so the results could be comparable. As so, this real data set contain five movies, each one with a different type of trajectory, just like before: Pure Rotation, Scale Change, Pure Rotation Z, Perspective Distortion and Unconstrained. Some example of the frames captured are shown in Figure 5.8.

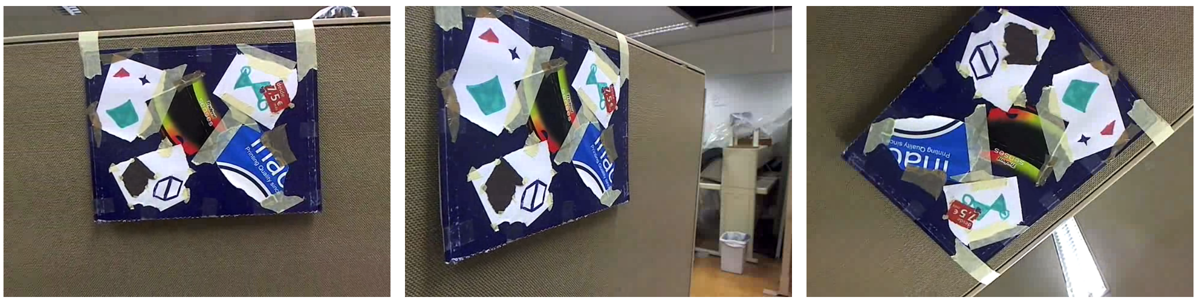


Figure 5.8: Some examples of dataset frames.

5.3.2 Results Analysis

In this experiment there is no ground truth data, i.e., the real trajectory traveled by the camera is unknown. Nevertheless, we have seen in the previous experiment that, if the template is well detected on the movie frames, the camera pose is correctly calculated. So, in this experiment, we will focus on the robustness of the SURF and BRIEF algorithms on detecting the template frame.

To this purpose, a few characteristics of the detectors were considered important: the smoothness of the pose solution, the number of features in the current frame that are matched with the features on the template frame and, from these, the number of the correctly matched features .

The number of features matched together with the number of the ones correctly matched can be an indicator of how reliable a pose estimation is. And, since all the trajectories traveled by the camera were smooth, if there are abrupt “jumps” on the pose estimation, we can conclude that at these points, the estimations are wrong or unreliable.

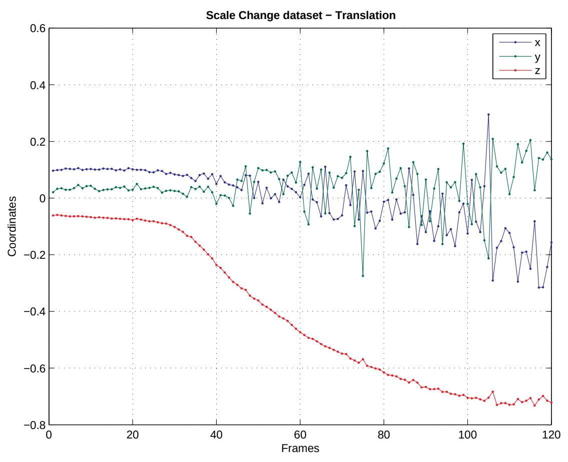
To evaluate the smoothness of the estimation, both translation and rotation vectors could be used, but we opted by only show here the translation vector, as the information obtained by the rotation vector is redundant (i.e., when the template is wrongly detected both rotation and translation estimations will be unreliable). The number of features matched is the number of features matched returned by SURF or BRIEF. Finally, because there is no ground truth between frames and it would be humanly impossible to do this manually in practical time, a matched feature is considered an outlier if it is excluded by the RANSAC algorithm when the homography is being estimated. That is, consider a current frame feature. Its coordinates are mapped to the template frame coordinates using the homography calculated. If the distance between the coordinates of the matched feature pair is greater than the RANSAC threshold, then this pair is considered an outlier.

In this section only the results that correspond to two datasets are shown. These were considered to be the ones that provide most information about the algorithms. The total of the results can be seen on Appendix A.4.

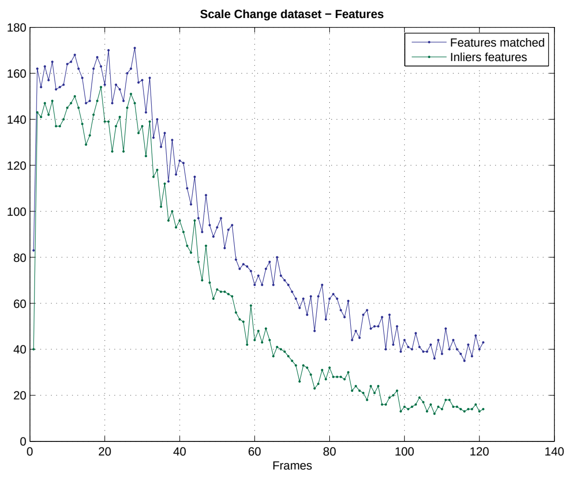
Discussion Figure 5.9 refers to the Scale Change dataset. Here, it can be seen that just like on previous experiment, SURF is more robust to scale change. BRIEF’s number of inliers features decay rapidly around frame 40, and from frame 75 the detection fails. SURF never lost the template and, on the furthest position, it still correctly detects and matches around 15 features which returns a sufficiently reliable estimation. Note that the SURF’s estimation of the z coordinate remain smooth.

Figure 5.10 shows the results from the Perspective Distortion dataset. Once again, BRIEF fails to detect the template sooner than SURF. When the number of inliers detected by SURF drops to a value below of more or less 12 the estimation is not reliable anymore as it can be seen by the interchange of estimation fails and sudden bumps since frame 90.

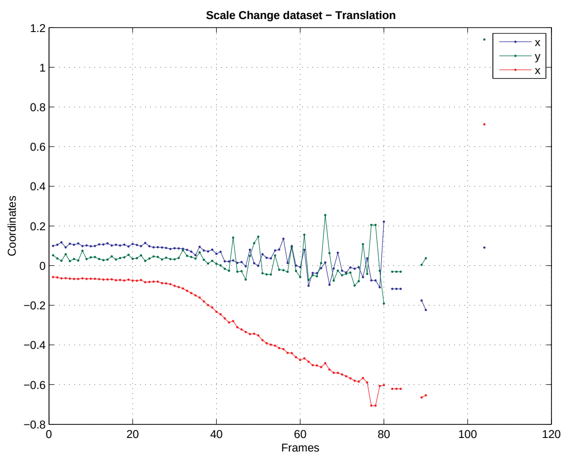
From these results it was possible to see that the algorithms behavior with real data is consistent with the behavior with synthetic data. It is also worth noticing that SURF always tends to detect an higher number of features, and the inlier ratio also tends do be higher. As so, SURF tends to produce more reliable estimation. This comes with a cost of computation time as SURF is much more time consuming than BRIEF.



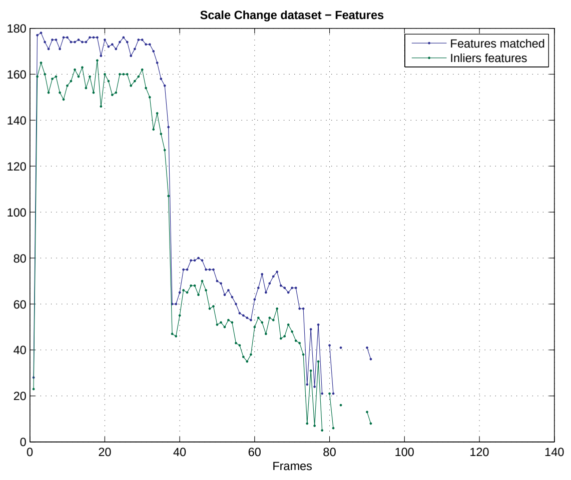
(a) SURF



(b) SURF

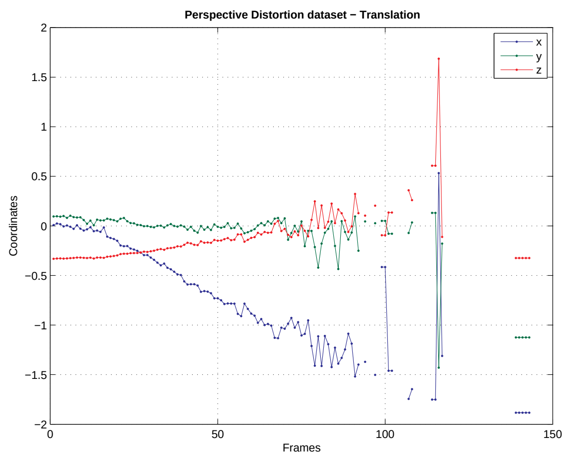


(c) BRIEF

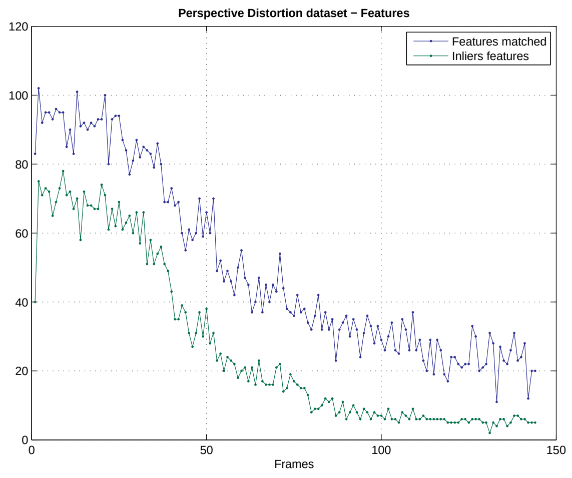


(d) BRIEF

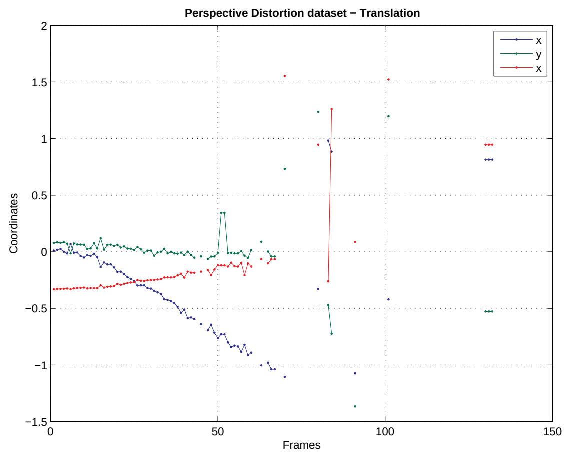
Figure 5.9: Coordinates and number of features along the Scale Change dataset frames.



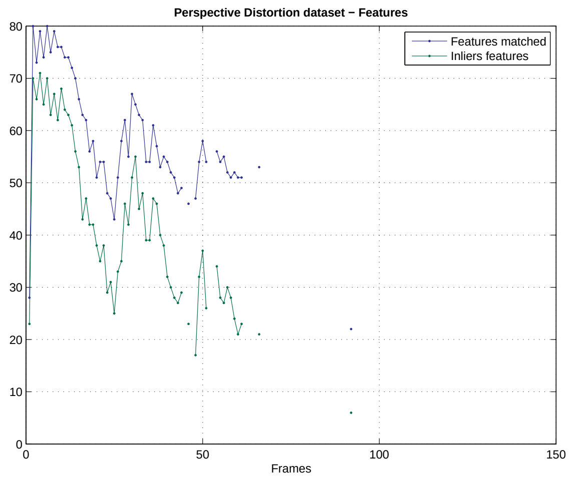
(a) SURF



(b) SURF



(c) BRIEF



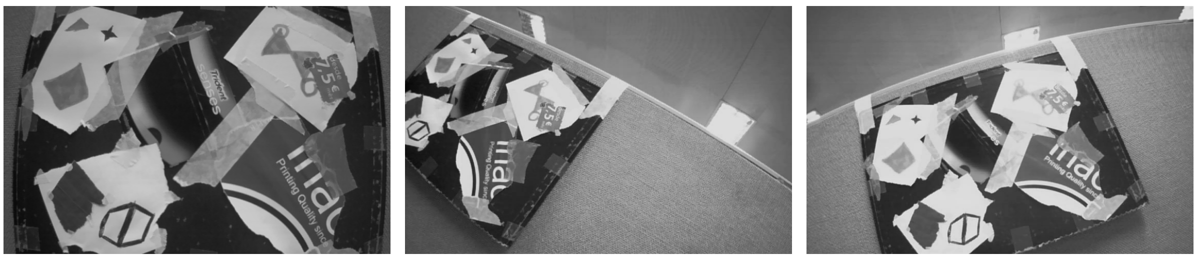
(d) BRIEF

Figure 5.10: Coordinates and number of features along the Perspective Distortion dataset frames.

5.4 Real Videos with Camera Attached to PUMA tests

5.4.1 Dataset

This, is an intermediate test between the previous one, and the next one where a control law will be used to drive the camera into the desired reference pose. Here, the Caspa VL camera was used, attached to the PUMA arm. While the video captured by the camera was being acquired, the joints readings from the arm encoders were also being saved into a log file. This test, serves mostly, to see if the pose estimation were in agreement with the PUMA readings. Figure 5.11 shows the template frame along with some example frames taken from the video captured.



(a) Template frame **(b)** Example frame **(c)** Other example frame

Figure 5.11: Template frame and some examples of dataset frames.

5.4.2 Results Analysis

With the video captured and the log file containing the PUMA readings, it is possible determine the trajectory traveled by the camera according to each of the data. However, some approximations have to be taken into account. First, the fact that the robotic arm has more than fifteen years and without any maintenance, led to the deterioration of the encoders and joints actuators, as so, the readings are not very accurate. Second, because there is no scale information, the scale factor was chosen by a trial and error approach, until the form of the trajectory estimated, matched the trajectory from the PUMA readings the best possible. This is, definitely, not a good scientific approach, but it served the purpose of confirming that the trajectory estimated has, at least, the “form” of the true trajectory. The result is shown in Figure 5.12.

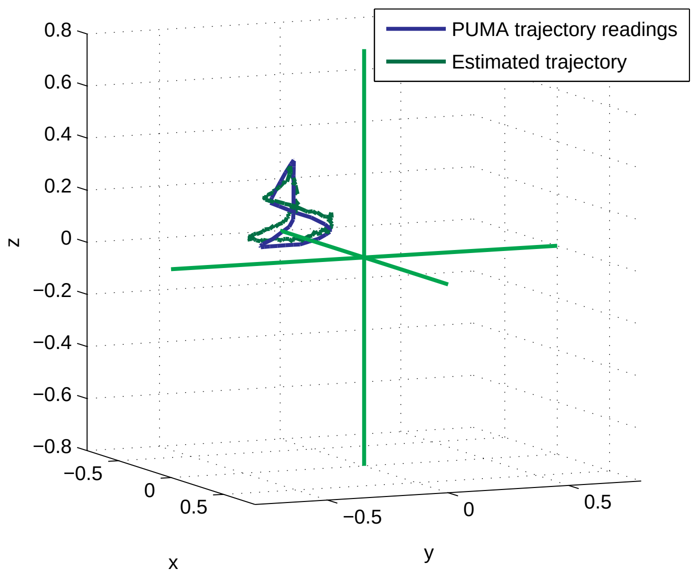


Figure 5.12: The trajectory traveled by the camera according to PUMA readings and the developed algorithm estimation.

5.5 Closed Loop Control tests

In order to fully demonstrate that the proposed solution allows the correct pose determination of a camera, a feedback control law was used to close the loop and drive the camera to the desired pose.

In this section, the control law used is described, some implementation considerations and modifications explained and the final results are presented.

5.5.1 Vision-Based control for rigid body stabilization and implementation on the PUMA 500 robotic arm

The following is a summary of Cunha et al. [38] to describe the control law used. Here the notation is a little different from the rest of the thesis in order to be in agreement with the original work.

Considering a fully-actuated rigid-body, with an attached coordinate frame $\{B\}$, and an inertial frame $\{F\}$ attached to the feature plane, let $(\boldsymbol{p}, \boldsymbol{R}) = ({}^B\boldsymbol{p}_F, {}^B_F\boldsymbol{R})$ represent, respectively, the translation and rotation of the $\{F\}$ with respect to the body frame. The equation that describes the motion of the body over time can be written as:

$$\dot{\boldsymbol{p}} = -\boldsymbol{v} - \mathsf{S}(\omega)\boldsymbol{p} \tag{5.3}$$

$$\dot{\boldsymbol{R}} = -\mathsf{S}(\omega)\boldsymbol{R} \tag{5.4}$$

where $\boldsymbol{v}$ and $\omega \in \mathbb{R}^3$ are the linear and angular velocities, respectively. If the rigid-body desired pose is represented by $(\boldsymbol{p}^*, \boldsymbol{R}^*)$ which are considered constant over time, we can introduce the following error variables

$$\boldsymbol{p}_e = \boldsymbol{p} - \boldsymbol{p}^*, \qquad \boldsymbol{R}_e = \boldsymbol{R}\boldsymbol{R}^{*T} \tag{5.5}$$

and write the corresponding state equations

$$\dot{\boldsymbol{p}}_e = -\boldsymbol{v} - \mathsf{S}(\omega)(\boldsymbol{p}_e + \boldsymbol{p}^*) \tag{5.6}$$

$$\dot{\boldsymbol{R}}_e = \mathsf{S}(\omega)\boldsymbol{R}_e \tag{5.7}$$

Denoting the current and desired calibrated coordinates of the image features by $\boldsymbol{y}$ and $\boldsymbol{y}^*$, respectively, and by α the homography matrix scale factor (see Chapter 2), the control law developed can be written as

$$\boldsymbol{v} = \begin{cases} k_1(\alpha\boldsymbol{y} - \boldsymbol{y}^*) & \text{until } ||\alpha\boldsymbol{y} - \boldsymbol{y}^*|| < \gamma \\ k_2(\alpha\boldsymbol{y} - \boldsymbol{y}^*) - \hat{z}^*\mathsf{S}(\omega)\alpha\boldsymbol{y} & \text{afterwards} \end{cases} \tag{5.8}$$

$$\omega = \begin{cases} k_3\mathsf{S}(\boldsymbol{y}^*)\alpha\boldsymbol{y} & \text{until } ||\alpha\boldsymbol{y} - \boldsymbol{y}^*|| < \gamma \\ k_4\mathsf{S}^{-1}(\boldsymbol{R}_e\boldsymbol{M} - \boldsymbol{M}\boldsymbol{R}_e^T) & \text{afterwards} \end{cases} \tag{5.9}$$

with the update law for the estimate of the desired depth $\hat{z}^*$ given by

$$\dot{\hat{z}}^* = \begin{cases} 0 & \text{until } ||\alpha\boldsymbol{y} - \boldsymbol{y}^*|| < \gamma \\ k_z\boldsymbol{y}^*\mathsf{S}(\omega)\alpha\boldsymbol{y} & \text{afterwards} \end{cases} \tag{5.10}$$

where γ, k_1, k_2, k_3, k_4 and k_z are positive scalars. $\hat{z}^*$, is a estimator for the desired depth. This control law relies on an adaptive scheme and can be divided in two sequential objectives. First, it will drive the translation vector $\boldsymbol{p}$ to an arbitrarily small neighborhood of $\boldsymbol{p}^*$. Then, it will ensure the convergence of $(\boldsymbol{p}, \boldsymbol{R})$ to $(\boldsymbol{p}^*, \boldsymbol{R}^*)$ using a controller that enforces feature visibility by guaranteeing that the camera not only points towards the features, but also remains in front of them.

As stated on the original work, the described control law guarantees that the desired equilibrium point is an almost global attractor.

For the implementation on the PUMA robotic arm, a few considerations had to be made due to some PUMA limitations. The control law had to be discretized because the arm can only receive inputs in position and not in velocity.

Denoting the inertial arm frame by $\{A\}$, the position and orientation of the body frame $\{B\}$, with respect to $\{A\}$, can be written as

$$^{A}\boldsymbol{p}_B = ^{A}\boldsymbol{p}_F - ^{A}_{F}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{p}, \tag{5.11}$$

$$^{A}_{B}\boldsymbol{R} = ^{A}_{F}\boldsymbol{R}\boldsymbol{R}^T \tag{5.12}$$

respectively, where $(^{A}\boldsymbol{p}_F, ^{A}_{B}\boldsymbol{R})$ denotes the pose of $\{F\}$ expressed in $\{A\}$. Using (5.3) and (5.4), the kinematics for $^{A}\boldsymbol{p}_B$ and $^{A}_{B}\boldsymbol{R}$ can be written as

$$^{A}\dot{\boldsymbol{p}}_B = {}^{A}_{B}\boldsymbol{R}\boldsymbol{v} \tag{5.13}$$

$$^{A}_{B}\dot{\boldsymbol{R}} = {}^{A}_{B}\boldsymbol{R}\mathsf{S}(\omega) \tag{5.14}$$

Using the Euler method, the solution of (5.13) can be approximated by

$$^{A}\boldsymbol{p}_B(k+1) = ^{A}\boldsymbol{p}_B(k) - \Delta t(^{A}_{B}\boldsymbol{R}(k)\boldsymbol{v}(k)), \tag{5.15}$$

where Δt is the sampling time. The zero-order hold discretization of (5.14) is given by

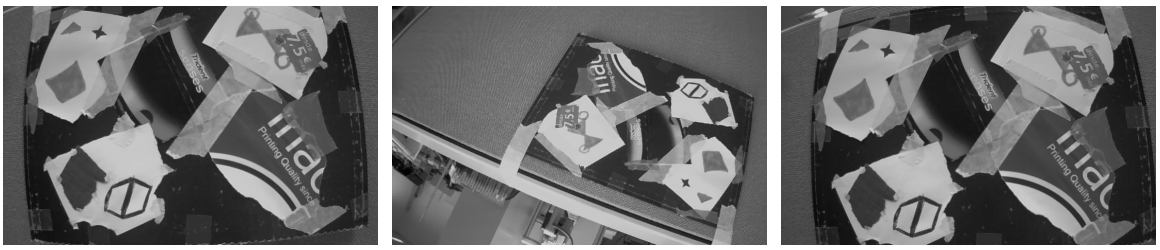
$$^{A}_{B}\boldsymbol{R}(k+1) = ^{A}_{B}\boldsymbol{R}(k)\ \mathrm{rot}(\Delta t\omega(k)) \tag{5.16}$$

Also, the true current pose of the camera had to be estimated by reading the PUMA joints encoders to know the current transformation from the arm frame to the camera frame. This was due to the fact that PUMA accepts position inputs expressed in the arm frame $\{A\}$ instead of in the feature frame $\{F\}$.

5.5.2 Results Analysis

The results are presented next. Figure 5.13 shows the template frame used, the image seen at the start pose, and the image seen at the final pose to where the PUMA end effector converged. As it can be seen, the image captured at the final pose is very similar to the reference template frame.

The evaluation over time of the error measure $||\alpha\boldsymbol{y} - \boldsymbol{y}^*||$ used on the control law is shown in Figure 5.14. Overall, the error tends to zero but, there are moments where its norm increases. This is caused by the way that the controller outputs are sent to the PUMA arm. As already stated, the controller is discretized. At each sample time, the controller outputs are calculated considering the sample rate and the PUMA actuators are activated to move the arm to next position according to the controller outputs. The trajectory that the arm travels between two sample positions is not externally controlled and depends on the internal joints controllers. So, the temporary increases on the error norm were predictable. If the arm could accept commands in linear and angular velocities, the control law could be used on its original version and it is expectable that the error norm curve thus becomes monotonically decreasing. It is also important to note that the final error is not zero. At the final pose, controller outputs were still being calculated and sent to PUMA, but the end-effector was not moving. This is caused by the low precision commands accepted by the PUMA arm. Nevertheless, according to the actuators readings, the translation error norm between the desired and the achieved pose is only 1.9 cm.



(a) Template frame **(b)** Image at start pose **(c)** Achieved image at final pose

Figure 5.13: Template frame and some examples of dataset frames.

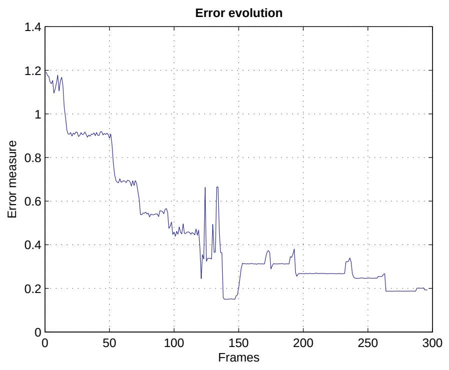


Figure 5.14: Error norm evolution over time

6 Conclusions

Contents

6.1 Conclusions

A solution to the problem of taking a rigid-body into a desired pose was developed. This solution only requires a image of planar scene acquired at the desired pose and a video stream captured by a camera on board of the vehicle containing the planar scene inside the FOV. Real-time performance was achieved when using a regular PC. Using the lightweight Gumstix-and-Camera hardware module an average of 3-4 fps are possible.

To achieve the described solution, a complete research was performed comprising several topics that are compiled together on the first chapters of this thesis. Concepts from computer vision such as the Camera Pinhole Model, image transformations like planar homographies and methods for the extraction of 3-D information from the planar homograhy matrix were learned. Also, it is explained how to calculate the homography that relates two images if features correspondence between images is available. To solve the problem of finding correspondence between image features, several algorithms were studied.

These algorithm are designed as image features extraction and description algorithms and mainly two were studied and implemented: SURF and BRIEF. From the tests performed, it was possible to conclude that both can be used to detect the reference planar scene on a video stream. Nevertheless, both have its own advantages and disadvantages. In general, SURF is more robust in extreme situations, like strong image scale changes or perspective distortion. Besides that, there are conditions where BRIEF totally fails detecting the scene and SURF estimation is not robust. But, at least SURF provides a estimate, which, if a control feedback law is used, can be sufficient to take the rigid-body into a pose where robustness is recovered. On the other hand, BRIEF wins with the incredible speed at which the required computations are performed. Plus, the fact that BRIEF fails sooner than SURF may not be a problem, since if the feedback control law is working properly the rigid-body pose error will probably not be very big and BRIEF will always succeed on detecting the reference scene. That is why, when considering applications requiring real time performance, BRIEF is chosen over SURF.

A test bench for testing the developed algorithm was built, consisting of synthetically generated video and true video captured from a PUMA robotic arm with the G-C module attached to its end effector. After long years of inactivity, the PUMA robotic arm was reanimated and its existence given a purpose. This allowed to verify the correctness of the algorithm, its performance and robustness. With this test bench it was possible to confirm that, when using an appropriate control law, the pose error tends to zero, thus demonstrating the correctness of the algorithm.

6.2 Future Work

Much more could have been done if time restrictions were inexistent.

The frame rate of the algorithm running on the G-C module can be improved if the integrated DSP is used to compute the Hamming distance, required for matching the BRIEF descriptors. Also, if Moore's law remains valid, new and more powerful Gumstix's like COMs will be available.

Further testing the algorithm with a real vehicle, like the quadrotor available at the ISR labs, would be interesting by not discretizing the control law and using its original version.

Regarding, the features detection. A different technique was studied, where circular TAG's (Figure 6.1) are used instead of the regular features extractors and descriptors algorithms. This has the drawback of requiring the scene to contain a TAG but, under the scope of the AIRTICI project this may not be an issue. On the other hand, this has the advantage of not being as computationally intensive as SURF or BRIEF algorithms. The TAG detection algorithm originally developed by Reverse Engineering company personell was studied and adapted to calculate homographies between images. Unfortunately, due to time restrictions, it was not integrated on the main developed algorithm.

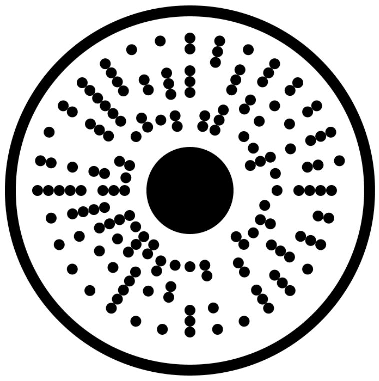


Figure 6.1: An example of a circular TAG.

Bibliography

[1] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," ECCV'10, 2010.

[2] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," European Conference on Computer Vision, 2006.

[3] Airtici: Advanced interactive robotic tools for the inspection of critical infrastructures. [Online]. Available: http://welcome.isr.ist.utl.pt/project/index.asp?accao=showproject&id_project=131

[4] E. Altüg, J. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," Robotics and Automation, 2002.

[5] E. Altüg, J. Ostrowski, and C. Taylor, "Quadrotor control using dual camera visual feedback," Robotics and Automation, 2003.

[6] M. Earl and R. D'Andrea, "Real-time attitude estimation techniques applied to a four rotor helicopter," Robotics and Automation, 2003.

[7] H. Romero, R. Benosman, and R. Lozano, "Stabilization and location of a four rotor helicopter applying vision," American Control Conference, 2006.

[8] I. F. Mondragón, P. Campoy, C. Martínez, and M. A. Olivares-Mendez, "3d pose estimation based on planar object tracking for uavs control," IEEE International Conference on Robotics and Automation, 2010.

[9] C. Martínez, I. F. Mondragón, M. A. Olivares-Méndez, and P. Campoy, "On-board and ground visual pose estimation techniques for uav control," Intelligence Robotics Systems, 2011.

[10] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck, "Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle," IEEE Transactions on Robotics, 2009.

[11] R. Hartley and A. Zisserman, Multiple View Geometry in computer vision. Cambridge University Press, 2003.

[12] Y. Ma, S. Soatto, J. Kosecká, and S. Sastry, An Invitation to 3-D Vision. Springer, 2004.

[13] R. Szeliski, Computer Vision: Algorithms and Applications. Springer, 2010.

[14] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," American Control Conference, 1981.

[15] O. Faugeras and F. Lustman, "Motion and structure from motion in a piecewise planar environment," International Journal of Pattern Recognition and Artificial Intelligence, 1988.

[16] C. Harris and M. Stephens, "A combined corner and edge detector," Proceedings of the 4th ALVEY vision conference, p. 147 151, 1988.

[17] P. R. Beaudet, "Rotationally invariant image operators," International joint conference on pattern recognition, 1978.

[18] L. Kitchen and A. Rosenfeld, "Gray-level corner detection," Pattern Recognition Letters, pp. 95 – 102, 1982.

[19] T. Lindeberg, "Scale-space theory: A basic tool for analysing structures at different scales," Journal of Applied Statistics, pp. 224 – 270, 1994.

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, pp. 91 – 110, 2004.

[21] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features surf," Computer Vision and Image Understanding, 2008.

[22] C. Schmid and R. Mohr, "Local greyvalue invariants for image retrieval," IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997.

[23] W. T. Freeman and E. H. Adelson, "The design and use of steerable filters," IEEE Transactions on Pattern Analysis and Machine Intelligence, 1991.

[24] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005.

[25] M. Özuysal, P. Fua, and V. Lepetit, "Fast keypoint recognition in ten lines of code," IEEE conference on computer vision and pattern recognition, 2007.

[26] S. Taylor, E. Rosten, and T. Drummond, "Robust feature mathing in 2.3us," IEEE conference on computer vision and pattern recognition, 2009.

[27] F. C. Crow, "Summed-area tables for texture mapping," Proceedings of the 11th annual conference on Computer graphics and interactive techniques, p. 207 212, 1984.

[28] S. Gauglitz, T. Höllerer, and M. Turk, "Evaluation of interest point detectors and feature descriptors for visual tracking," International Journal of Computer Vision, 2011.

[29] J. J. koenderink, “The structure of images,” Biological Cybernetics, 1984.

[30] T. lindeberg, “Scale-space for discrete signals,” PAMI, 1990.

[31] M. Brown and D. lowe, “Invariant features from interest point groups,” BMVC, 2002.

[32] J. H. Freidman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” ACM Trans. Math. Softw., 1977.

[33] C. Silpa-Anan and R. Hartley, “Optimised kd-trees for fast image descriptor matching,” CVPR, 2008.

[34] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” International Conference on Computer Vision Theory and Applications (VISAPP’09), 2009.

[35] S. M. Smith and J. M. Brady, “Susan - a new approach to low level image processing,” International Journal of Computer Vision, 1997.

[36] J. R. Quinlan, “Induction of decision trees,” Machine Learning, 1986.

[37] Opencv: Open source computer vision. [Online]. Available: http://http://opencv.willowgarage.com/

[38] R. Cunha, C. Silvestre, J. Hespanha, and A. P. Aguiar, “Vision-based control for rigid body stabilization,” Automatica, 2011.

[39] J.-Y. Bouguet. (2010, July) Camera calibration toolbox for matlab. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

A

Appendix

A.1 Steps aiming the practical implementation

During the course of this work, several implementation issues had to be resolved.

The OpenCV library is very complete and well designed. Nevertheless, some familiarization time with the coding style and functionalities was needed. It was necessary to get used with the Gumstix board. It is not a very common type of hardware, and developing code to run on it, is not the same as developing code to run on a common computer. Particularly, it was necessary to learn how to use different toolchains and to study the operative system options available. At first, a considerable amount of time was spent on learning the OpenEmbedded building framework. But a more lightweight and minimalistic Linux OS version was needed, so, one was developed with the essential help of André Oliveira. There was also the need to port some libraries, and debugging code is much more difficult.

Considering the camera modules, the first one chosen was the e-CAM50 from e-con Systems (Figure A.1). Time was spent trying to resolve some issues relative to the driver interface integration with the OpenCV library. Plus, when in comparison with the Caspa VL camera module, the technical specifications are weaker: 30 FPS at 640x480 resolution versus 60 FPS at 720x480 resolution. So, this option was left behind, and the Caspa VL adopted.

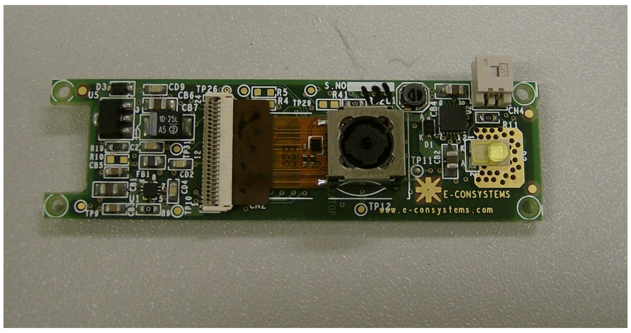


Figure A.1: The e-cam50 camera module.

Also, it was also necessary to understand the PUMA kinematics and software source code in order to modify it and add the functionality of receiving commands over the network.

Apart, from these principal issues mentioned, a few more existed. Nevertheless, it was expectable, as this was a more practical focused thesis.

A.2 Camera Calibration

The image in Figure A.4(a) shows that the lens mounted on the Caspa VL Camera module suffers from the effect of radial distortion. In order to reduce or, ideally completely eliminate this effect, the camera was calibrated. To this purpose, images of a planar checkerboard were acquired. The chosen method was the one from Jean-Yves Bouguet available online in the form of a MatLab toolbox [39]. This method uses as input the 3-D coordinates of the checkerboard corners and their 2-D point projections on the several images acquired. The checkerboard must be seen from as much different perspectives as possible and cover as much as possible all the field of view (FOV) of the camera. Some of these images are shown in Figure A.2.

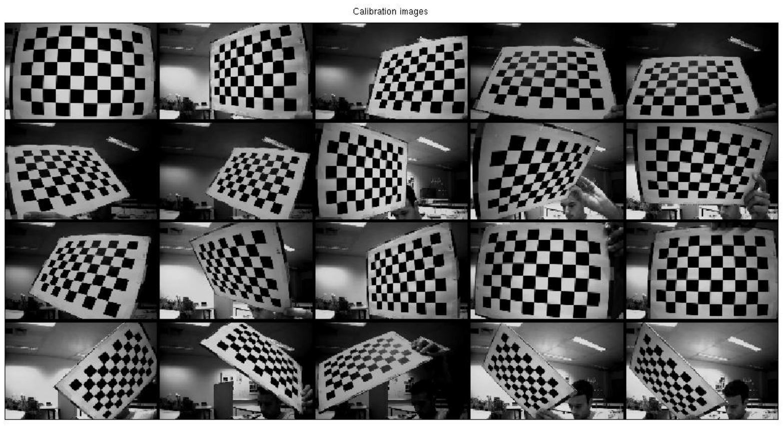


Figure A.2: Some examples of the calibration images used.

The 3-D coordinates are defined by considering a 3-D reference frame attached to the checkerboard. The origin is one of its corers, and the x-y plane coincides with the checkerboard plane. Knowing the dimensions of each square, and the number of squares, the 3-D coordinates of all the corners can be can be easily obtained. For the 2-D coordinates, the images acquired have to be processed. After the the four external corners are manually located by the user, the rest of the corners are automatically detected using corners detectors algorithms.

Having all the necessary coordinates defined, the main calibration step is performed using non linear optimization. This step minimizes the total re-projection error in the least square sense over all the calibration parameters. The non linear optimization algorithm used is the iterative gradient descent with an explicit computation of the Jacobian matrix.

The toolbox has many features and gives a good intuition about the calibration process. For example, after the calibration, it is possible to see a graphic (Figure A.3) with the estimated checkerboard poses when the images were taken.

This method estimates the coefficients for both tangential and radial distortion, as well as the linear projective intrinsic parameters for a distortion free equivalent camera. In other words, the output of the

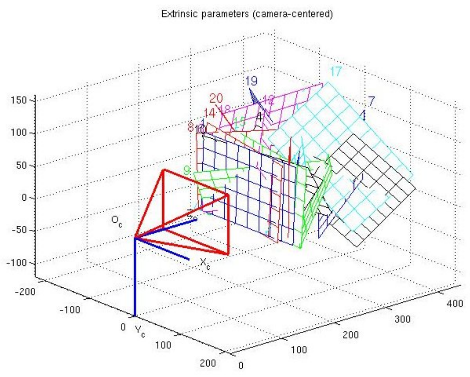


Figure A.3: Some examples of the calibration images used.

method is the camera calibration matrix together with the image distortion coefficients. An example of radially distorted image and its corrected version is illustrated in Figure A.4.

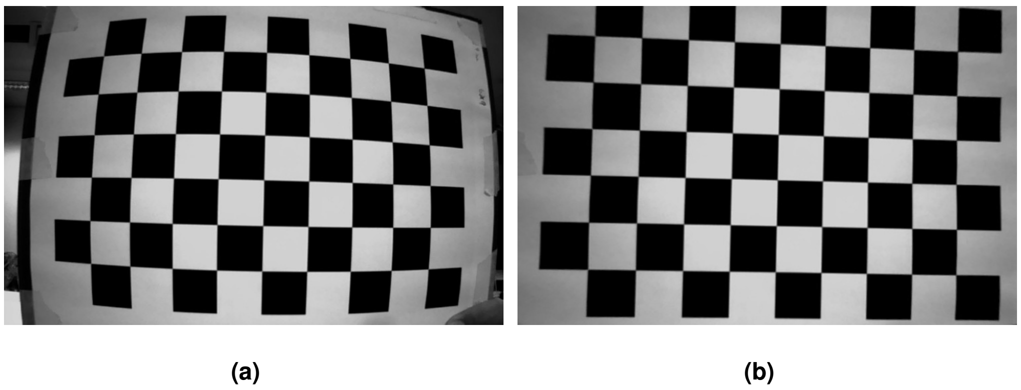


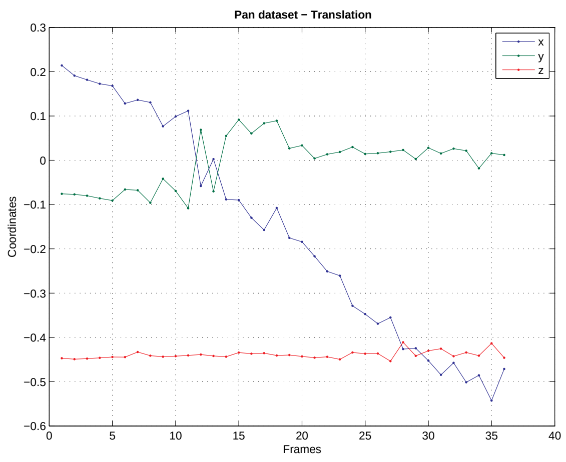
Figure A.4: Radially distorted image (a) compared to the undistorted image (b).

A.3 The Kronecker Product

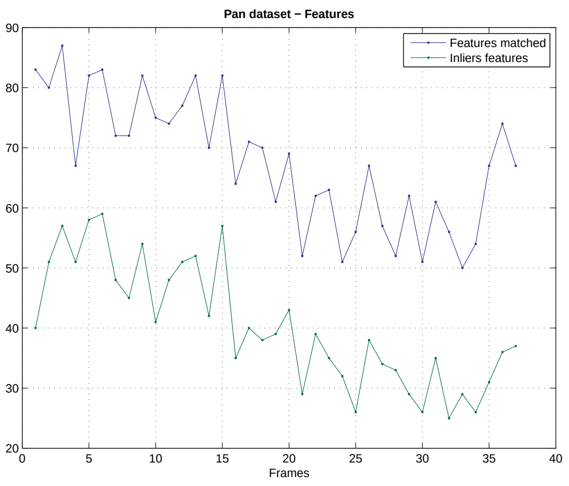
If A is an m-by-n matrix and B is a p-by-q matrix then, the Kronecker product $A \otimes B$, is the mp-by-nq block matrix

$$A \otimes B = \begin{bmatrix} a_{11}B & ... & a_{1n}B \\ : & ::: & : \\ a_{m1}B & ... & a_{mn}B \end{bmatrix}. \tag{A.1}$$

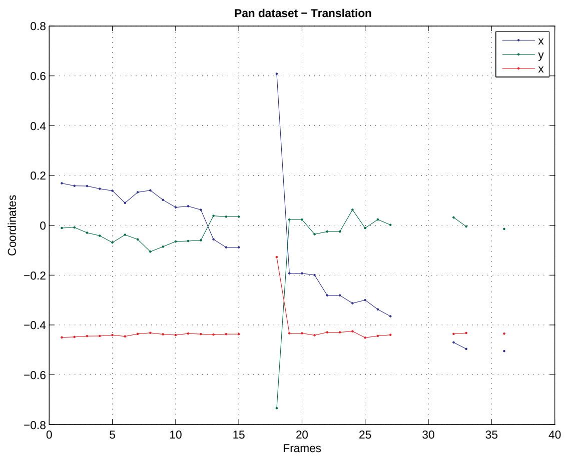
A.4 Real videos experiments graphics



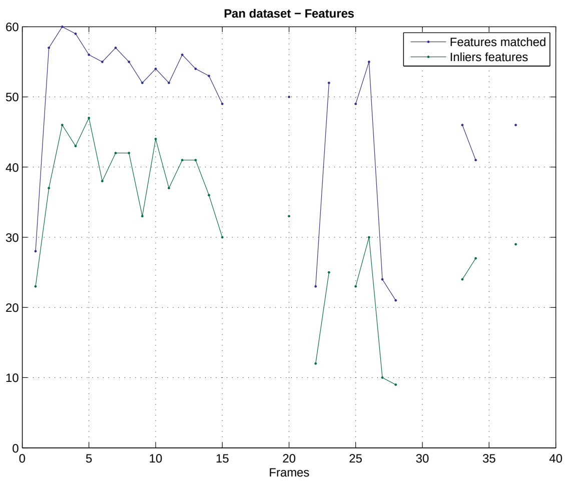
(a) SURF



(b) SURF

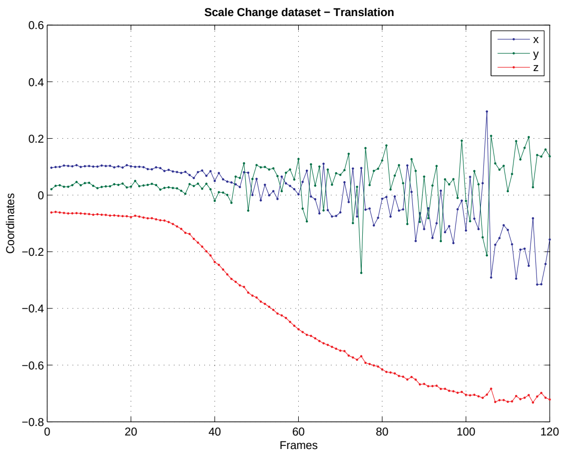


(c) BRIEF

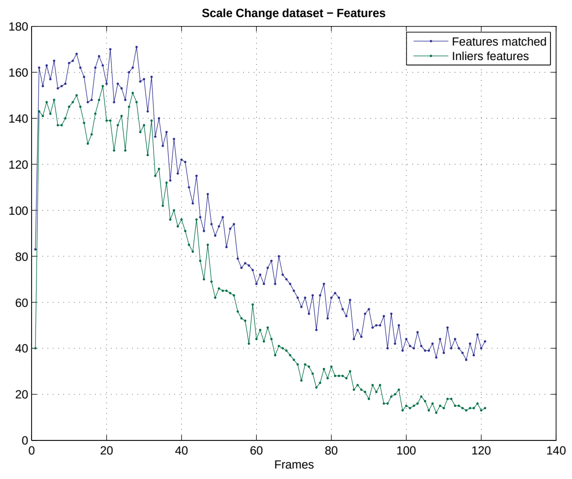


(d) BRIEF

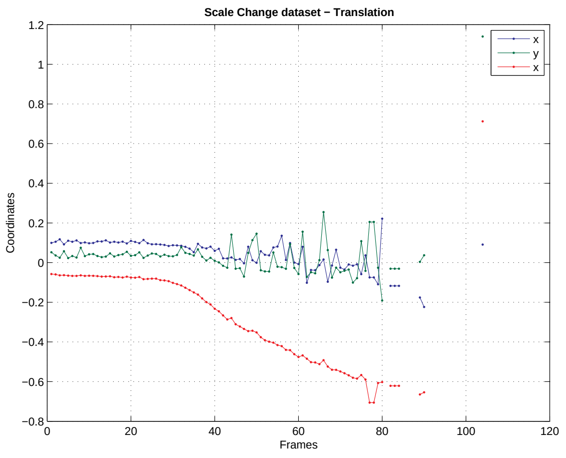
Figure A.5: Coordinates and number of features along the Pure Rotation dataset frames.



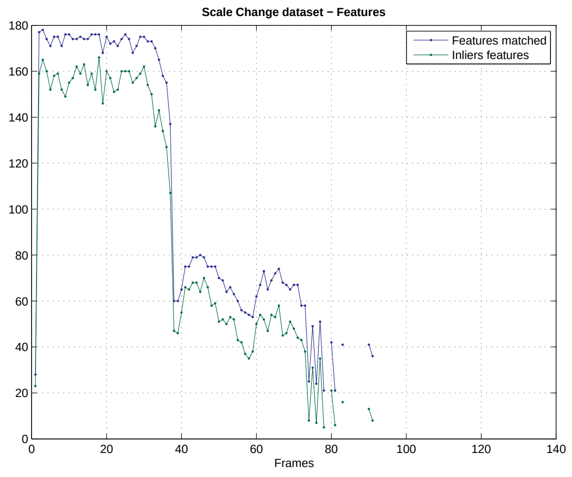
(a) SURF



(b) SURF

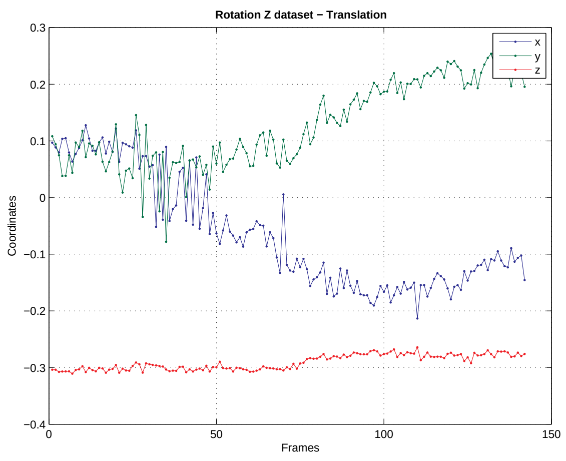


(c) BRIEF

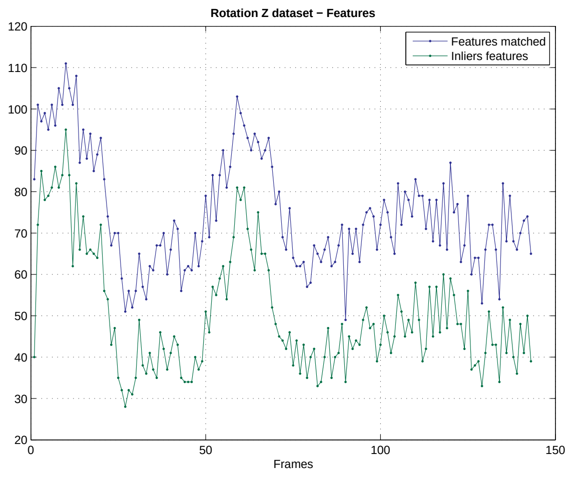


(d) BRIEF

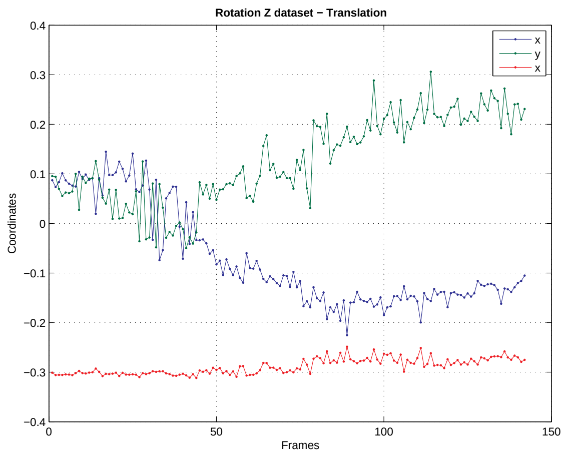
Figure A.6: Coordinates and number of features along the Scale Change dataset frames.



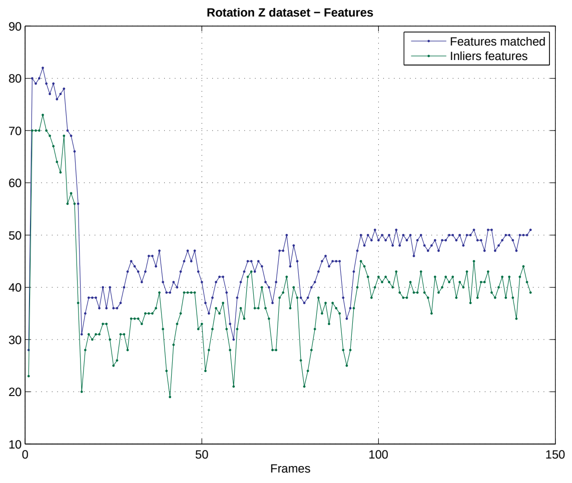
(a) SURF



(b) SURF

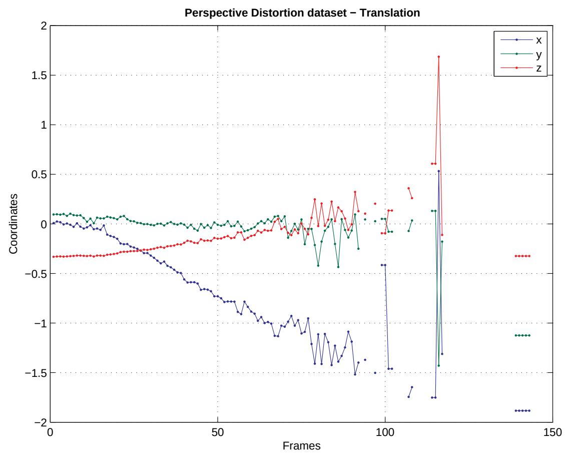


(c) BRIEF

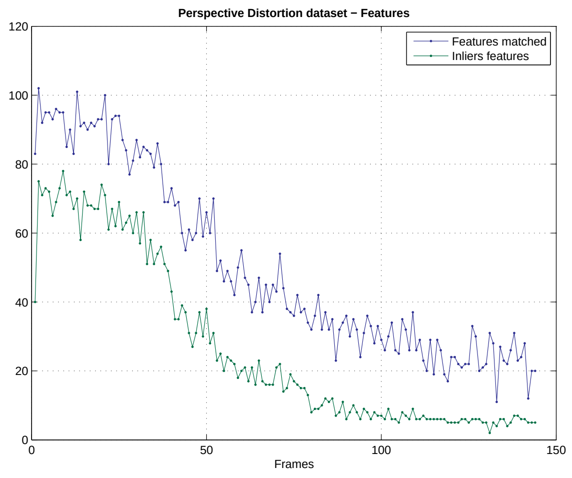


(d) BRIEF

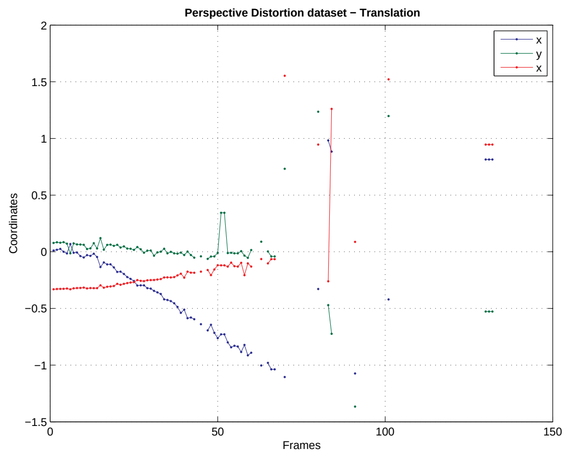
Figure A.7: Coordinates and number of features along the Pure Rotation Z dataset frames.



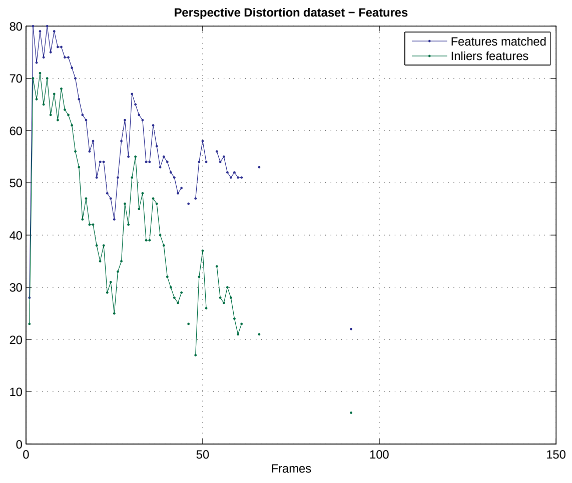
(a) SURF



(b) SURF

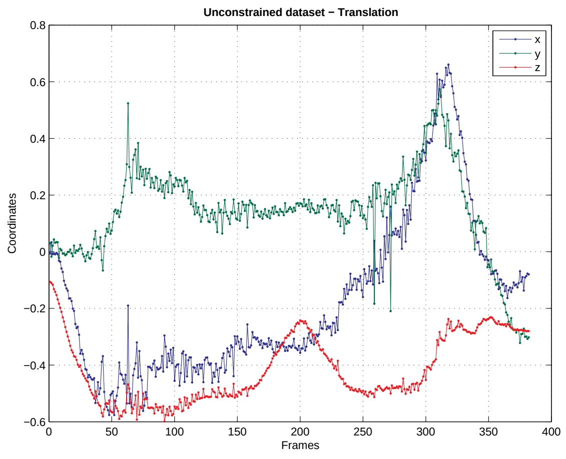


(c) BRIEF

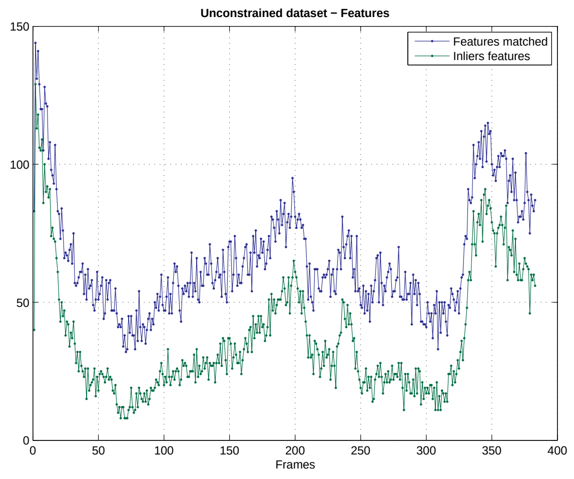


(d) BRIEF

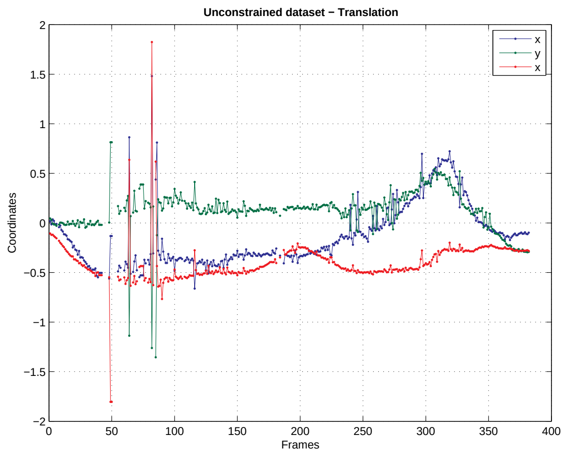
Figure A.8: Coordinates and number of features along the Perspective Distortion dataset frames.



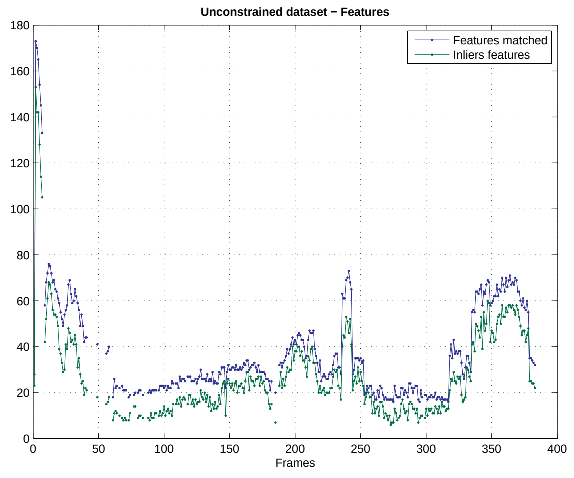
(a) SURF



(b) SURF



(c) BRIEF



(d) BRIEF

Figure A.9: Coordinates and number of features along the Unconstrained dataset frames.