3D-PC: Semiautomatic Generation of Three-dimensional Databases from Images on Personal Computers

by

Gerardo José Lemus Rodriguez
Ingeniero Mecánico-Electricista
UNAM, 1992

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1994

© Massachusetts Institute of Technology 1994. All rights reserved.

Author ....................................................... 
Department of Electrical Engineering and Computer Science
June 14, 1994

Certified by .............................................
Andrew B. Lippman
Associate Director, MIT Media Laboratory
Thesis Supervisor

Accepted by ...............
Frederic R. Morgenthaler
Chairman, Committee on Graduate Students

NOV 16 1994
3D-PC: Semiautomatic Generation of Three dimensional Databases from Images on Personal Computers

by

Gerardo José Lemus Rodriguez
Ingeniero Mecánico-Electricista
UNAM, 1992

Submitted to the Department of Electrical Engineering and Computer Science on June 14, 1994, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis shows a method to obtain tridimensional structure from a video sequence. In practice a set of control points placed on objects in the image are traced and correlated as the camera moves through the scene. A 3-D estimate of the scene is made from the displacement of the points. This model can then be rendered from a variety of point of views with original video overlaid onto it. The point of the thesis is to test the operation of the system on a PC.

Thesis Supervisor: Andrew B. Lippman
Title: Associate Director, MIT Media Laboratory
For my Family
Contents

1 Introduction ................................. 8
   1.1 Project Overview .......................... 8
   1.2 Problem .................................. 9
       1.2.1 Approach ............................. 10

2 Structure and shape from Images .......... 11
   2.1 Imaging Geometry .......................... 11
       2.1.1 Camera Model ......................... 12
       2.1.2 Interior Orientation .................... 13
       2.1.3 Reference Frame Transformation ......... 14
   2.2 Stereopsis ................................ 15
       2.2.1 3-D modeling ........................... 16
       2.2.2 Absolute and Relative Orientation ......... 17
       2.2.3 Matching Points ....................... 18
   2.3 Structure from motion ..................... 24
       2.3.1 Recursive Estimation of Motion, Structure, and Focal Length 25

3 Implementation ............................... 29
   3.1 Feature Points Tracking .................. 31
   3.2 Structure and Motion Recovery .......... 33
   3.3 Extended Stereo Algorithm and Surface Interpolation .............. 33
   3.4 Manual geometric modeling and Texture mapping ................. 35

4 Results ................................... 37
4.1 Tracking and Estimation of Structure .......................... 37
4.2 Preliminary Results using Extended Stereo Algorithm .......... 38
4.3 Manual Modeling ............................................. 40

5 Conclusions ................................................. 44

A ................................................................. 46
A.1 Marr Hildreth edge detector ..................................... 46
A.2 Quaternions .................................................. 47
   A.2.1 Definition .............................................. 47
A.3 Least Squares Fitting of Surfaces .............................. 51
A.4 Extended Kalman Filtering ................................... 52
   A.4.1 EKF for Recursive estimation of motion, structure and focal
         length .................................................. 53

B MATLAB Code ...................................................... 55
B.1 Interactive Control Points Tracking ............................ 55
   B.1.1 menu3d.m .............................................. 55
B.2 Structure Estimation using EKF ............................... 72
   B.2.1 estimate.m ............................................. 72
   B.2.2 est_x.m ............................................... 78
   B.2.3 h.m ................................................... 79
   B.2.4 dh.m .................................................. 80
   B.2.5 p_set.m .............................................. 83
B.3 Manual Geometric Modeling .................................... 84
   B.3.1 polyg1.m .............................................. 84
   B.3.2 polyg1.m .............................................. 86
   B.3.3 pl_fit.m ............................................... 88
   B.3.4 getpoly.m ............................................. 89
B.4 Model Integration .............................................. 89
   B.4.1 plane6.m .............................................. 90
B.5 Stereo Approach ............................................. 92
  B.5.1 epi_mod.m ........................................... 92
  B.5.2 corr.m ............................................. 98
  B.5.3 dispar.m .......................................... 102
  B.5.4 dispar.m .......................................... 104
  B.5.5 orstereo.m ...................................... 106
  B.5.6 edge_zer.m .................................... 108

B.6 Miscellaneous Functions ..................................... 109
  B.6.1 bin2nan.m ...................................... 109
  B.6.2 gaussian.m .................................... 109
  B.6.3 hessian.m ....................................... 110
  B.6.4 marr_hil.m .................................... 110
  B.6.5 max_loc.m ..................................... 111
  B.6.6 quat2mat.m .................................... 111
  B.6.7 quat_mul.m .................................... 111
  B.6.8 red.m .......................................... 112
List of Figures

2-1 (a) Pinhole camera model. (b) Pinhole camera model with the focal length relocated ........................................ 13
2-2 Model of central projection ........................................ 14
2-3 Changing coordinate systems in 3-D space ..................... 15
2-4 Coordinate systems of two images ................................ 16
2-5 Pencil of epipolar lines for two images. (a) Frame 1. (b) Frame 10. 19
2-6 Set of rectified images. (a) Frame 1. (b) Frame 10. ............... 20

3-1 Block diagram of model extraction from video using manual segmentation 30
3-2 Block diagram of model extraction from video ................ 32

4-1 Control Points tracked. (a) Frame 1. (b) Frame 45. ............. 38
4-2 (a) and (b) A pair of stereo images and their corresponding epipolar lines. (c) and (d) A pair of stereo images rectified. (e) Disparity Map. 39
4-3 Structure Estimation for the wall and the floor. ................. 40
4-4 Modeling a plane. (a) Selection of the points. (b) Selection of the corresponding surface. (c) Warping of the selected surface onto the three-dimensional plane. ...................................... 41
4-5 3D model rendered from different views with an orthographic projection. 42

A-1 (a) Original figure. (b) Edge detection with $\sigma = 1.5$. (c) Edge detection with $\sigma = 3$ ........................................ 48
Chapter 1

Introduction

1.1 Project Overview

The goal of this thesis is the creation of a three-dimensional model of a scene derived from the succession of images in a video sequence. To do this one needs at least 2 images of the scene, as is the case with stereoscopic imaging, and one needs to correlate points in similar objects in both views. Given camera position and parameters (such as focal length) the depth of each correlated point can be determined and from that a segmented model can sometimes be built.

One of the most successful methods to solve this problem where the camera position is not known involves making a recursive estimate of the 3D point locations using an extended Kalman filter (EKF) as demonstrated by Azarbayejani [3, 2] and Brown [6]. Horn showed [16] that six feature points overdetermine the relative orientation between two camera positions, making it possible to determine the structure and motion of the camera with a sequence of still images.

An issue is the location and tracking of the points through the entire sequence. Tomasi did this automatically [32] but others have done this interactively. In cases where the model can be created with some hand-seeding of the database, an interactive solution will suffice. In effect, the operator monitors the correlation process and corrects it if the estimate goes awry.

Having thus obtained the depths and positions of each of the control points in
the scene, the next issue is segmenting the images into objects that can then be manipulated and displayed using normal 3-D rendering techniques. Further, the original video can be mapped onto the faces of the objects in the model permitting a realistic image synthesized from perspectives that the original camera never took. For example, the motion of the camera can be stabilized, a new path can be synthesized, new objects can be placed realistically into the scene and camera parameters can be altered in the rendering.

Segmentation of the scene, however, like correlation is not automatic. One can imagine how such an operation can be made to work but the technique is not perfect: geometric objects that track the control points are added to the database and either split or merged until there is a reasonable, representative number of useful shapes. For example, six points in a vertical plane near the front of the object field can be rendered as the faces of two squares, as a set of triangles, or as a single concave and convex figure. Heuristics that make assumptions about the underlying objects must be used to clarify this.

1.2 Problem

The core of the work in this thesis involves investigating this analysis algorithm and the associated rendering system for use in a real estate application. A movie of a house is made by literally sweeping the camera through the space without even looking through the viewfinder. This drives the model-building component. On replay, the goal is to allow a prospective buyer to walk freely through the space. Thus we have available a few optimizations that make the task tractable: (1) nothing in the scene is moving except the camera. Thus the control points themselves will remain static in space. (2) The elements of the house are simple and can be derived from a simple, cubic, "Cape Cod" style New England home. This simplifies the "triangle versus square" issue noted above. Finally, (3) manual intervention is reasonable - the data will be used more often than it will be gathered.

Finally, this work will be ported to a personal computer platform for use in real
applications.

The key problems that remain are the responsiveness of the system and the manner in which the heuristics can be applied to house-mapping.

1.2.1 Approach

Once the problem is defined, the task becomes the selection of the tools which will allow us to solve it. Using the techniques explained above it will be possible to recover the position of the camera relative to the first frame. At this point, two options can be considered; the first implies an approach similar to the one used by Azarbayejani [3, 2], which fits geometric objects to the few points tracked; the second employs the camera parameters obtained from the same algorithm to perform a 3D reconstruction of all the points in each frame using stereo algorithms. This second option represents a better alternative. The automatization of the model generation is simplified when a larger amount of data is available; in such a way the system may use less human interaction to specify the use of geometric basic elements (geons [27]).

With the geometric model available, computer graphics algorithm are useful in generating the different kinds of images required: rendering novel views of the buildings, modificating the model, adding or removing elements, and infinite other applications our imagination allows.
Chapter 2

Structure and shape from Images

Our ultimate goal is to create a 3-D model of a scene from a sequence of images. This is also one of the goals of visual processing at a ‘middle level’. The process of extracting the 3-D structure of a scene is usually referred to as shape-from-X since many sources of information can be used for the task [17]. One image can be used as a source, as the case of the shape-from-shade method [14] (an active area of research), using prior knowledge about the scene and interaction with a user, depth-from-focus [26], or using special cameras like laser rangefinders\(^1\).

Although it may not be the simplest 3-D extraction method, employing two or more images does not require special and expensive cameras; the data gathering can be done with any camera or videocamera\(^2\).

2.1 Imaging Geometry

In this section mathematical models and descriptions are developed for the geometric data related to scene structure, relative camera position and camera geometry. All these elements are used later for the 3-D estimation process.

\(^1\)A detailed explanation of some other methods can be found in [29].
\(^2\)the videocamera is more suitable for our purposes, as we would like to videotape buildings exteriors and interiors.
2.1.1 Camera Model

To produce images of a scene in a specified plane (called image plane or focal plane), imaging devices employ a set of lenses which are usually modeled as pinhole (Fig. 2-1(a), pinhole camera). This model not only places the focal plane behind the lens but it also reverses the image, so a more convenient way to represent the camera geometry is relocating the focal plane to a symmetrical position with respect to the lens center (Fig. 2-1(b), center of projection, COP). Perpendicular to the retinal plane and originating at the COP we traced a line denominated optical axis. The intersection of the optical axis with the image plane is called principal point (PP); the distance from the COP to the PP is the focal length \(f\). We have some options for the selection of the coordinate system which can be centered at the COP or at PP Fig. 2-2. *If the coordinate system is centered at the COP, the projection model [4, 9, 29, 14, 22]. Fig. 2-2 can be modeled as

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} = \begin{bmatrix}
  X_C \\
  Y_C
\end{bmatrix} \frac{f}{Z_C}
\]

(2.1)

where \((X_C, Y_C, Z_C)\) is the 3-D location of a point in the camera reference frame and \((u, v)\) the projection of it on the image plane. (2.1) can be rewritten linearly as

\[
\begin{bmatrix}
  U \\
  V \\
  S
\end{bmatrix} = \begin{bmatrix}
  f & 0 & 0 & 0 \\
  0 & f & 0 & 0 \\
  0 & 0 & 1 & 0
\end{bmatrix} \begin{bmatrix}
  X_C \\
  Y_C \\
  Z_C \\
  1
\end{bmatrix}
\]

(2.2)

where

\[
u = U/S \quad v = V/S \quad \text{if } S \neq 0
\]

(2.3)

These equations allow us to interpret \(U, V\) and \(S\) as the projective coordinates of a point in the retina. If \(S = 0\), the 3-D point is in the retinal plane of the camera.

\[
\begin{bmatrix}
    u \\
    v
\end{bmatrix} = \begin{bmatrix}
    X_C \\
    Y_C
\end{bmatrix} \frac{1}{(1 + Z_C \beta)}
\]  

(2.4)

with \( \beta = 1/f \), keeping \( u \) and \( v \) as the projective coordinates of a point in the image plane. Geometrically, (2.1) and (2.4) are identical, but the former fixes the system origin at the PP rather than the COP Fig. 2-2; it also uses \( \beta \) as a model parameter instead of as the focal length \( f \). This model has the advantage of being much better behaved that (2.1) when the focal length becomes large and can easily represent the orthographic projection (the limit when \( f \to \infty \) for (2.1) by simply defining \( \beta = 0 \) in (2.4)).

### 2.1.2 Interior Orientation

We have already placed the origin of the coordinate system at the principal point and the Z-axis parallel to the optical axis. However, we still need to know these parameters for our camera. The information we need is obtained through the process of internal orientation [14]. Measurements of radial distortion of the lens can also be included if they are significant enough. Several methods were developed to determine the interior orientation of a camera, such as Collimator array, Images of Spheres,
Figure 2-2: Model of central projection

Vanishing Points [11]. The Vanishing points method [33] is the easiest method to implement since it only needs the picture of a cube. It computes the vanishing points (vp, three possible for a cube [13]) of the cube’s edges, then finds a point (P) such that \( P \rightarrow vp1, P \rightarrow vp2 \) and \( P \rightarrow vp3 \) are orthogonal; \( P \) is the location of the COP.

2.1.3 Reference Frame Transformation

What happens when we change the world reference frame? In Fig. 2-3 we moved from an old coordinate system centered at \( O \) to the new coordinate system centered at \( C \) by a rotation \( R \) followed by a translation \( T = CO \). The coordinate frame transformation equation is:

\[
\begin{bmatrix}
X_C \\
Y_C \\
Z_C
\end{bmatrix} =
\begin{bmatrix}
t_x \\
t_y \\
t_z
\end{bmatrix} + R
\begin{bmatrix}
X_O \\
Y_O \\
Z_O
\end{bmatrix}
\]  

(2.5)

The matrices \( R \) and \( T \) describe the position and orientation of the camera with respect to the new world coordinate system and they are called the *extrinsic* parameters of the camera. However, some other kind of transformation equations can be
Figure 2-3: Changing coordinate systems in 3-D space

derived from (2.5); we now introduce

\[ \begin{bmatrix}
X_C \\
Y_C \\
Z_C
\end{bmatrix} = \begin{bmatrix}
t_X \\
t_Y \\
t_Z
\end{bmatrix} + \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & \beta
\end{bmatrix} \begin{bmatrix}
X_O \\
Y_O \\
Z_O
\end{bmatrix} \]

(2.6)

a transformation with some interesting properties which will be very useful for the recursive estimation process [4].

2.2 Stereopsis

In Stereopsis, two or more images are taken from different viewpoints. Looking at the images, points located closer to the camera will change their position significantly compared to points located at a great distance from the camera. The change (disparity) can be measured from one view to the other and is used to reconstruct the 3-D shape of the scene from it. Usually the disparity is not found for all the points, so the few points reconstructed three-dimensionally are used to interpolate the rest.
2.2.1 3-D modeling

In this section it is shown how to reconstruct a 3-D structure from the disparity. First we can suppose the projections of a 3-D point in the two images are known. The point \( M \) has \( M_f = [X_f, Y_f, Z_f]^T \) coordinates in the coordinate system of the first image and is \( M_s = [X_s, Y_s, Z_s]^T \) in the coordinate system of the second image; \( [u_f, v_f]^T \) is the projection of \( M \) in the first image and \( [u_s, v_s]^T \) the projection in the second image. Then in the coordinate system of the first image (Fig. 2-4), \( [X_f, Y_f, Z_f]^T \) is

\[
\begin{bmatrix}
X_f \\
Y_f \\
Z_f
\end{bmatrix} =
\begin{bmatrix}
1 + \alpha \beta & u_f \\
1 + \alpha \beta & v_f \\
0 & Z_s
\end{bmatrix} \tag{2.7}
\]
(2.7) is similar to (2.4) solving for \([X_f, Y_f, Z_f]^T\). The \([X_s, Y_s, Z_s]^T\) point in the second image corresponds to a modification of the coordinate system (see 2.1.3)

\[
\begin{bmatrix}
X_s \\
Y_s \\
Z_s
\end{bmatrix} =
\begin{bmatrix}
t_X \\
t_Y \\
t_Z
\end{bmatrix} +
\begin{bmatrix}
X_f \\
Y_f \\
Z_f
\end{bmatrix}
\] (2.8)

\([u_s, v_s]^T\) are obtained from substituting (2.8) into (2.4). If we also substitute (2.7) into (2.8) and (2.4), a function \([u_s, v_s]^T = f(u_f, v_f, \beta, \alpha = Z_f, R, T)\) is derived; solving for \(\alpha\),

\[
\alpha = -\frac{(1 + \beta t_z + \beta r_{s1} u_f + \beta r_{s2} v_f) v_s - t_y - r_{s1} u_f - r_{s2} v_f}{(r_{s1} u_f \beta^2 + r_{s2} v_f \beta^2 + \beta r_{s3}) v_s - r_{s2} v_f \beta - r_{s3} - r_{s1} u_f \beta}
\] (2.9)

or

\[
\alpha = -\frac{(1 + \beta t_z + \beta r_{s1} u_f + \beta r_{s2} v_f) u_s - t_x - r_{i1} u_f - r_{i2} v_f}{(r_{s1} u_f \beta^2 + r_{s2} v_f \beta^2 + \beta r_{s3}) u_s - r_{i2} v_f \beta - r_{i3} - r_{i1} u_f \beta}
\] (2.10)

both equations are equally valid and either one can be used to compute \(\alpha\), the depth we were searching. In this general form the depth depends of \(u_f, v_f, \beta, R, T\) and one of \(u_s\) or \(v_s\) (the relative orientation and the disparity); \(\alpha\) can be computed for each location where these parameters are known and then use (2.7) to get the 3-D location at each point. (2.9) and (2.10) look very complicated and the disparity is not straightforward. Current stereo systems make use of simplifying constraints considering \(R\) equal to the identity matrix and \(T\) equal to a translation along the X-axis. It is better to consider the general case since we will use this result later.

### 2.2.2 Absolute and Relative Orientation

In 2.1.3 we showed how to change from one system coordinate to another. That is allowed as long as \(R\) and \(T\) are known. The recovery of these parameters is called *absolute orientation* if three-dimensional coordinates of \(M_f\) and \(M_s\) are known; Horn [14, 15] proved a closed solution for this problem exists and is overdetermined if the
three-dimensional coordinates of three (or more) points are known. The *relative orientation* problem appears only when the projections of these points are available; Horn [16] stated that the projection of six points overdetermine it, but the non-linear nature of the projective equations makes it impossible to find a closed form solution to this problem.

2.2.3 Matching Points

As Huttenlocher [17] says

> The geometry of stereo vision is the ‘easy part’, now we must consider the problem of how to identify pairs of images points $p_i$ and $p_i'$ that corresponds to projections of the same scene points $p$ into the left and right images.

We have two images and need to know which pairs of points correspond to the same 3-D point. There are many problems associated with matching points between two images; for example, a point in one image may not have a corresponding point in the other (the result of occlusion, windowing of the scene, etc.). Additional considerations should be made, the strongest of them being the *epipolar constraint*.

**Epipolar Constraint**

Recalling the set of equations (2.9) and (2.10), we notice we do not need to know both $u_s$ and $v_s$ to compute $\alpha$; only one suffices. An explanation is illustrated in Fig. 2-4; projecting $COP_s$ onto the first image gives a point denominated epipole ($ep_f$). Similarly, the projection of $COP_f$ onto the second image creates $ep_s$. The line traced from $ep_s$ to the $M$ projection onto the second image is called an epipolar line (the same line is also formed from the projection of the line $COP_f \rightarrow M$ onto the second image). The correct matched point to $[u_f, v_f]^T$ (the corresponding $[u_s, v_s]^T$) *has to lie along that epipolar line*. In general, a point imaged on the epipolar line in

---

3When more that three points are available, least-square methods are used to deal with noise. The method also constrains $R$ to be orthonormal.
Figure 2-5: Pencil of epipolar lines for two images. (a) Frame 1. (b) Frame 10.

the first image can only be imaged on the corresponding epipolar line in the second image (if it is imaged at all). The epipolar constraint is very useful; it allows us to perform a matching operation unidimensionally, instead of two-dimensionally. In Fig. 2-5 the pencil of correspondent lines for two images is shown; for real images some vertical disparity happens.

To simplify the matching process, the epipolar lines in one image should be parallel to each other \(^4\). That can be done either by taking the pictures with cameras which are only translated along the X-axis, or processing and rectifying the pair of pictures \([9]\) (see Fig. 2-6).

Normalized Correlation

Among the most used techniques to match points, the normalized correlation is one of the oldest and most investigated ones. Assuming the epipolar lines are parallel, the basic idea to match a point of coordinates \([u_0, v_0]^T\) in the first image, is to consider a rectangular window of size \((2P + 1) \times (2N + 1)\) centered at \([u_0, v_0]\) and compute its

\(^4\text{Since all the epipolar lines cross the epipole, it means the epipole }\to \infty, \text{ or the projection of the COP of the other image does not project onto the image plane.}\)
correlation with the second image along the row $v_2 = v_0$

$$C_{12}(\tau) = \frac{1}{K} \sum_{u_1 = -N}^{N} \sum_{v_1 = -P}^{P} (I_1(u_1 + u_0, v_1 + v_0) - \bar{I}_1(u_0, v_0))(I_2(u_1 + u_0 + \tau, v_1 + v_0) - \bar{I}_2(u_0 + \tau, v_0))$$

(2.11)

where

$$K = (2N + 1)(2P + 1)\sigma_1(u_0, v_0)\sigma_2(u_0 + \tau, v_0)$$

(2.12)

$$\bar{I}_1(u_0, v_0) = \frac{1}{(2N + 1)(2P + 1)} \sum_{u_1 = -N}^{N} \sum_{v_1 = -P}^{P} I_1(u_1 + u_0, v_1 + v_0)$$

(2.13)

$$\sigma_1^2(u_0, v_0) = \frac{1}{(2N + 1)(2P + 1)} \sum_{u_1 = -N}^{N} \sum_{v_1 = -P}^{P} (I_1(u_1 + u_0, v_1 + v_0) - \bar{I}_1(u_0, v_0))^2$$

(2.14)

$\bar{I}_1(u_0, v_0)$ and $\bar{I}_2(u_0, v_0)$ are the mean intensity at point $[u_0, v_0]^T$ in the first and second images, while $\sigma_1^2(u_0, v_0)$ and $\sigma_2^2(u_0, v_0)$ are the standard deviation at the same point. Ideally, $C_{12}$ would have only one maximum point (a possible match); but that is not always the case. $C_{21}$ can also be computed, exchanging the roles of the first and the second image in (2.11) \(^5\).

\(^5\)Note (2.11) is not symmetric with respect to the images.
Algorithm of the method:

1. Compute $C_{12}$ iterating from a $\tau_{\text{min}}$ to a $\tau_{\text{max}}$.

2. Keep values of $\tau$ which maximizes $C_{12}$.

3. Keep $\tau$ if $C_{12}$ is bigger than a preestablished threshold.

4. If $C_{12}$ has various maxima, recompute $C_{12}$ only for those points using a larger window to disambiguate.

5. Repeat process with $C_{21}$.

6. Conserve places where $C_{21}$ and $C_{12}$ agree.

Here the tradeoff for the stereo imaging system can easily be seen: a larger baseline $^6$ will provide a larger disparity and a better depth estimation; however, the matching process becomes more arduous since the size of the matching area is bigger (requiring more computations) and several ambiguous matches can be found for a single point.

The normalized correlation method can be refined if, instead of correlating the intensity of the images, the correlation is performed using the convolution of the images with the Laplacian of a Gaussian (Nishihara's refinement, [9]). Nishihara showed this has the effect of sharpening the peak of the autocorrelation function (thus facilitating the detection of maximum points) on the regions of constant sign.

One of the drawbacks relies on the assumption of constant disparity inside the processed window; an assumption often violated near edges or discontinuities. It does not work well when a large amount of foreshortening is present $^7$ or if large regions with similar intensities are present ($C_{12}$ is a very flat curve).

$^6$the distance between origins of coordinate systems

$^7$when the perspective of the two cameras is very different, i.e. one camera is located closer to the object than the other, the first camera will image a large object while the projection on the second camera will be an smaller version of the object; in this case correlation methods do not work properly.
Marr-Poggio-Grimson

The Marr-Poggio-Grimson algorithm uses a completely different point of view with respect to the normalized correlation method [12]; it tries to match key features in the images (discontinuities, edges). They used mechanisms of human stereo vision to propose a computational theory of depth perception.

The features of the images are extracted using the Marr-Hildreth edge detector [17]\(^8\). This operation renders two binary images, therefore the matching process can be performed with less complex operations (compared with the normal correlation method). The correspondence strategy follows a multiscale process; first it matches coarse features, and then repeats the correspondence process at finer resolutions using the previous match to guide the matching. The parameter \(\sigma\) of the operator \(\nabla^2 G\) (the Laplacian of a Gaussian) controls the coarseness of the feature extraction; i.e., a large \(\sigma\) is used for coarse matching and a small \(\sigma\) is used for finer matching. Grimson used operators with a size (\(\omega\)) of 9, 18, 36 and 72 picture elements (doubling the size of \(\sigma\)) for his implementation. The outline of the algorithm is:

1. **Loop over levels:** iterate from a coarse level (choosing a large \(\sigma\)) to the finer levels.

2. **Convolution:** convolve the images with the corresponding \(\nabla^2 G\) (\(\sigma\) defined by \{refnum:loop\}).

3. **Zero-crossings:** locate horizontal positive and negative zero crossings on the images.

4. **Loop over fixation position:** iterate \(d\) from some \(d_{\text{min}}\) to a \(d_{\text{max}}\) using a preestablished increment.

5. **Matching:**

   (a) **Feature point matching:** for each feature point in one zero-crossing description, a possible match with the same sign is searched in a preestablished

---

\(^8\)See A.1 for a more detailed description of the Marr-Hildreth method
area.

(b) **Figural continuity:** the figural continuity constraint is applied to eliminate most of the incorrect matches. In the edge figures, contours are located; contiguously matched *segments* are pruned if the length exceeds a length determined *a priori* from $\omega$ [12].

(c) **Disparity map update:** the disparity of matched points is added to the global disparity map.

6. **Disambiguation:** for the remaining points with more than one match, a disambiguation is done using the results of coarser levels. If ambiguous matches still remain, the points are discarded from the disparity map.

7. **Loop:** proceed to the finer level of representation, looping to step 1.

The algorithm is tuned to deal with parallel epipolar lines; thus the disparity $d$ becomes a change along only the horizontal axis. For step 5a matches are searched inside a region defined by $\omega$ and the disparity $d$:

$$\{(x',y) \mid x + d - \omega \leq x' \leq x + d + \omega\}$$

(2.15)

Sometimes an extension to the matching region is done to allow some vertical disparity ($\pm \epsilon$, which for practical purposes always can be present even if the images were rectified); the region becomes

$$\{(x',y') \mid x' + d - \omega \leq x' \leq x + d + \omega; y - \epsilon \leq y' \leq y + \epsilon\}$$

(2.16)

This method works well to determine the disparity of edges (thus a 3-D reconstruction of those edges) and it would solve the previous problems established for the normal correlation method.
Other Methods.

The problem of obtaining a correspondence between two profiles can be formulated as a path-finding problem on a two-dimensional plane. The matching process between two binary images (outputs of an edge detector) is considered a dynamic programming problem. A more detailed description of this method is found in [9, 29, 14]. There are several methods available, although our intention is not to describe them all, merely to describe the ones used during the implementation.

2.3 Structure from motion

The structure from motion methods use the idea of taking a sequence of images of an object or scene and use the motion of the object with respect to the camera to reconstruct its three-dimensional shape. All of them need to track the object over the whole sequence of images. As we have already seen how difficult it is to match points along epipolar lines, imagine how difficult it is to do so for all the images specially without the epipolar constraint! Generally the methods are feature based and their input is a set of two dimensional image coordinates of P feature points [3, 2, 1, 5, 14, 34, 22, 32, 17]. All of the feature-based motion estimation methods try to solve the relative orientation problem (see 2.2.2 [14, 16]), a nonlinear problem solved minimizing a nonlinear objective function. One of the most successful methods for solving this problem involves making a recursive estimate of the 3D point locations using an extended Kalman filter (EKF) as demonstrated by Azarbayejani [3, 2]. Azarbayejani mentions in [4] other methods which have a faster convergence rate than the EKF by using batch processes with more complex computational procedures; this tradeoff always exists between recursive and batch processes.
2.3.1 Recursive Estimation of Motion, Structure, and Focal Length

We have to identify the parameters the recursive estimation will recover; in equation (2.4) the model of central projection was modified to be represented by $\beta$ instead of $f$. In the usual model (2.1) $f$ is only a scale factor from world units to pixels, while $Z_C$ encodes both the camera imaging geometry and depth. Modifying $\beta$ alters the imaging geometry independent of the representation of object depth. Had we known the value of $f$, those concerns would have been of no value, but for estimation purposes it is very important to represent the model with $\beta$. In the case this camera parameter is unknown, we can add it to the estimation.

$$\text{(camera)} = (\beta) \quad (2.17)$$

Given the set of $P$ feature points, we would like to estimate their 3-D position. If we parameterize each feature point as in (2.7), the 3-D pointwise structure can be represented with one parameter $\alpha_i$. For the $P$ points

$$\text{(structure)} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_P \end{pmatrix} \quad (2.18)$$

this vector is sufficient for the representation of all of them. In 2.1.3 we introduced (2.6); in that equation, the $t_y$ and $t_x$ components correspond to directions parallel to the image plane, while $t_z$ corresponds to a direction along the optical axis. Even with a "wide angle" lens, the sensitivity to motion along the optical axis (Z-axis) is very small; in the orthographic cases the sensitivity is null. If instead of estimating $t_z$, the product $t_z\beta$ is used, the model does not degenerate when large focal lengths
are used\(^9\). The parameter of the translational points are

\[
(\text{translation}) = \begin{pmatrix}
t_X \\
t_Y \\
t_Z\beta
\end{pmatrix}
\]

(2.19)

To recover \(t_Z\) a simple division is enough, as long as \(\beta \neq 0\).

The rotation \(\mathbf{R}\) can be represented as an orthonormal matrix, or a unit quaternion\(^{10}\). A unit quaternion has three degrees of freedom and it is easier to handle, but it is very problematic to include the quaternion estimation directly into the EKF. To solve this problem, Euler angles are used to estimate interframe rotation at each frame; then they are composed with an external rotation quaternion (which maintains an estimate of global rotation). For the EKF we estimate the Euler angles

\[
(\text{interframe rotation}) = \begin{pmatrix}
\omega_X \\
\omega_Y \\
\omega_Z
\end{pmatrix}
\]

(2.20)

Externally, we compose these Euler angles into a quaternion \(\hat{q}\)

\[
(\text{global rotation}) = \begin{pmatrix}
q_0 \\
q_1 \\
q_2 \\
q_3
\end{pmatrix}
\]

(2.21)

used in the linearization at each step.

Until now we have counted \(7 + P\) parameters (6 for motion, 1 for camera geometry and \(P\) for structure). For each point, we have \(2P\) measurements, therefore we need

\(^9\)Azarbayejani showed in [4] the product \(t_Z\beta\) remains observable, and therefore also estimable even for orthographic projections.

\(^{10}\)See A.2.
at least 7 points to estimate correctly all the parameters. Previous method counted
incorrectly the number of parameters, yielding a undetermined problem. However,
the scale can not be estimated unless an additional constraint is added, which can be
one of the static parameters \{\alpha_i\}.

The complete state vector \( x \)

\[
x = \begin{pmatrix}
    t_x \\
    t_y \\
    t_z \beta \\
    \omega_x \\
    \omega_y \\
    \omega_z \\
    \beta \\
    \alpha_1 \\
    \vdots \\
    \alpha_p
\end{pmatrix}
\]

(2.22)

is the one used in a EKF implementation, where the measurement vector contains the
image locations of features in each new frame. The dynamics models in the EKF are
chosen trivially as an identity transform (\( \Phi[n] \)) plus noise (\( \xi[n] \), modeled as Gaussian
distributed noise for the motion and the camera parameters):

\[
x[n + 1] = \Phi[n]x[n] + \xi[n]
\]

(2.23)

The measurement equation is obtained by combining equations (2.4), (2.7) and
(2.6) defining:

\[
y_1[n] = \begin{bmatrix} u_i[n] \\ v_i[n] \end{bmatrix} = h_i \left( \begin{bmatrix} u_i[1] \\ v_i[1] \end{bmatrix}, R[n], t[n] \right)
\]  \hspace{1cm} (2.24)

\(u_i[n]\) and \(v_i[n]\) are the projections of the point \(i\) (1 to \(P\)) at the \(n^{th}\) frame (the first being labeled \(n = 1\)) and correspond to the measurements required for the EKF; \(R[n]\) and \(t[n]\) are the current translation parameters for the \(n^{th}\) frame. Including all the projected coordinates (the measurements) in a single formula and including the model uncertainty as the addition of a random variable \(\eta[n]\) modeled as Gaussian distributed white noise yields:

\[
y[n] = \begin{bmatrix} y_1[n] \\ \vdots \\ y_P[n] \end{bmatrix} = h(x[n]) + \eta[n]
\]  \hspace{1cm} (2.25)

With (2.25) and (2.23) any EKF algorithm \(^{11}\) can be used, taking into account that it should extract the Euler angles after each estimation and combine them with the external quaternion. The result of the EKF should be the values of the state vector \(x[n]\) for each \(n\), solving the problem of relative orientation. In some cases, an iterative EKF can be used to improve the result of the operation.

\(^{11}\)Refer to A.4.
Chapter 3

Implementation

Recalling the main goal of this thesis, a method for a semiautomatic generation of tridimensional structures should be developed. The immediate application is to allow real-estate companies to show to prospective buyers a model of the house letting them “travel” around it without physically being present. The emphasis then is in the almost automatic generation of the model. Holtzman [13] implemented a highly interactive system for the creation of models from pictures. Azarbayejani [2] used the recursive estimation from motion to semiautomatically extract 3-D object models; his method requires the manual tracking of the feature points and the manual segmentation of images into regions which belong to the same object (Fig. 3-1). The last procedures need a complex and time consuming user-interface especially if the representation of the model requires the segmentation into many objects [2] used only one building, while for real-estate purposes we require the modeling of both exteriors and interiors of houses which can be composed of dozens of objects).

It seems we cannot rely completely (yet) on the automatic feature points tracking; however, this task is very easily performed by humans (although very boring) and an operator could monitor the correlation procedure.

The recursive estimation of motion, structure and focal length (REMSF) yields very good estimates of the feature points three dimensional location. Again the tracking problem begins to affect us: if we want to obtain a dense representation of the structure, we are required to track the same number of points, which is very difficult if
Figure 3-1: Block diagram of model extraction from video using manual segmentation
we do not constrain the problem; manual tracking of the points become unmanageable as the number of points increases. Why would we like a denser representation of the structure? Because it is always better for the model generation if we have the 3-D position of hundreds of points instead of dozens, thus facilitating the fitting of basic tridimensional objects.

We can solve that tradeoff recalling most of the structure from motion methods which solve the relative orientation problem. Then instead of using only the tracked points for the 3-D modeling, we focus on the relative orientation of each frame against the others and adapt the structure determination to be a Stereo problem (a more researched problem). For Stereo algorithms, surface interpolation or the approximation process is necessary to obtain a good estimate of 3-D shape as not all the points have a corresponding match. At this point we can constrain the possible shapes (therefore the interpolation methods) to be polygons (because we are not depicting natural scenes) in order to simplify the process. The last step would be to create texture maps and overlay them into the 3-D model, to create the illusion of a real model.

The complete structure of the method is shown in Fig. 3-2.

3.1 Feature Points Tracking

Feature points should be selected and tracked for the sequence (although they can be added and removed interactively if an occlusion happens). One would like to select easily tracked features; for example, corners [10]. They can either be manually selected, or selected using a simple algorithm tuned to find corners. An example of feature points selection scheme would be the identification of extreme points of the Hessian of an image

$$Hessian(I) = \begin{vmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{vmatrix}$$

(3.1)

The tracking of the feature points is done using normalized correlation\(^1\). An interesting idea could be the physical collocation of traceable features on the scene (control

\(^1\)Refer to section 2.2.3
Figure 3-2: Block diagram of model extraction from video
dots), i.e. put red dots everywhere in a room and use the color to correlate them along the sequence. This approach is invalid when the video is already taken or if it is difficult to put the control dots in the scene (which could be the case when videotaping a building’s exterior).

3.2 Structure and Motion Recovery

As explained in 2.3, many algorithms are available to extract the 3-D structure and motion from a sequence of images. For our purposes, batch processes are a good option as we do not have time restrictions. For this implementation, an iterative EKF was chosen assuring a better convergence of the results.

3.3 Extended Stereo Algorithm and Surface Interpolation

As we said in the beginning of this chapter, we can use Stereo algorithms to obtain a denser 3-D representation of the object. A possibility is to chose two frames and apply any good algorithm. From 2.2 we noticed the Steropsis tradeoff: for large baselines the depth is more accurate, but it is more difficult to match points. However, we have available dozens (even hundreds) of frames. Is there any way we can use that extra information? From frame to frame the correspondent baseline is small and with the epipolar constrain the correspondence process is less difficult. Multiscale Stereo matchers use information of coarser levels to guide the correspondence process at finer levels; a similar process in time could be applied. The correspondence between recent frames can be used to help the matching process for later frames. A similar procedure is used in [23]; using multiple images taken using a lateral motion, Okutomi used the multiple baselines available to obtain a dense Disparity map. In this case we would generalize the process to take into account any kind of movement.

We can arbitrarily define any frame to be the initial one used for the estimation and designate it $F_1$. This frame will have a Depth Map $\alpha(F_1)$; each point of the frame
\((u_1, v_1)\) has an associated depth such that the 3-D location of that point satisfies the equation (2.7) (which parameterizes the 3-D location of a point only with the point projection on the frame and its depth \(\alpha\)).

An algorithm to utilize all the extra information could be (if the number of available frames is \(N\), arbitrarily using a step of \(s\) frames):

1. Initially, set \(j = 1 + s\).

2. Set depth map \(\alpha(F_1)\) arbitrarily to 0.

3. A stereo algorithm is applied between \(F_1\) and \(F_j\) *taking into account* the depth map.

4. Update the depth map, increment \(j = j + s\) if we still have available frames and got to step \(\{2\}\).

The value of \(s\) can be set arbitrarily to 1 or another fixed number; or it can be derived from a predetermined length of baseline \(^2\). After all the iterations, a very refined depth map \(\alpha(F_1)\) should be achieved, corresponding to the stereo pair \(F_1\) and \(F_N\), but with a more efficient process (less ambiguous matches).

The critical step in this algorithm is \(\{3\}\); a successful stereo method should be selected. Most of the stereo algorithms require as an input a pair of *rectified* images such that the correlation process is simplified to search along epipolar lines which correspond to the rows of the images. A detailed description of step \(\{3\}\) is:

1. Rectify the pair of images.

2. Match points

3. Correlate matched points in the rectified images to the one in the original pair \((F_1\) and \(F_j\)).

\(^2\)Suppose the movement between frames is very small, then we can take a large \(s\) without problems. But if the movement is large, then an small \(s\) should be used; the size of \(s\) can be derived from the length of the baseline.
(a) If a depth map is available, project the location of points in the rectified $F_j$ based on the depth map $\alpha(F_1)$.

(b) If the projection of the point does not lie on the correspondent epipolar line, find the closest point of the epipolar line to the projected point.

(c) Realize the search of the matched point in the epipolar line in a region close to the projected point.

4. Compute depth map of matched points.

5. Apply an interpolation method for the remaining points, and compute depth map of all the points ($\alpha(F_1)$).

The interpolation process depends upon the selected Stereo Method. Considering the nature of this problem, approximations with polygons are quite acceptable and desirable.

3.4 Manual geometric modeling and Texture mapping

The Extended Stereo Algorithm and the Surface Interpolation can be substituted with a simpler method if human interaction is allowed. In this case, given the set of tracked control points, the user selects subsets of points which belong to a surface. The parameters of the surface are then obtained using least square methods and the three-dimensional locations of the control points.

For a more realistic view of the 3-D model, the original video can be mapped onto the surface thus allowing a generation of images from different perspectives to the ones found in the original sequence.

First the texture maps are generated from the video. This process could be considered the inverse operation to computer graphics rendering. The rendering process consists of matching each pixel location in the new image with a 3-D location on the scene giving the pixels the color of the object (information contained in the texture
map). Thus if we want to reverse the process, the color of the images is used to create the texture maps which later will be used for the generation of novel sequences.

Taking into account the selected surfaces, a region of the original video is manually segmented and overlaid onto a specific surface. The segmented video overlaid onto the surface forms a 3D region in space. This 3D region can be projected in the remaining frames of the sequence yielding a mask which can be used to retrieve additional information. This extra information can be used to complement the texture maps [2].
Chapter 4

Results

As the title mentions, the main goal of this thesis is to create three-dimensional databases from video on personal computers, specifically the IBM PC and its clones. All the programming was done using MATLAB on a PC 386. The video sequence was captured using a commercial camcorder on a Hi-8 tape and digitized in D1 format. The D1 sequence was then converted into DAT format and ported into MATLAB’s MAT file format. About six points were tracked per selected surface and their 3D position was computed using the EKF algorithm previously described. The principal point of the image was assumed to be in the center of the image \(^1\). For the geometry modeling two methods were attempted: the first using manual segmentation of regions [2] and the second using Stereo Algorithms.

4.1 Tracking and Estimation of Structure

To track all the control points, normalized correlation was used and manually supervised and corrected for cases involving incorrect estimates. In Fig. 4-1 two different frames and their tracked control points are shown. The estimation of Structure was performed after all the points were tracked (although it can be done in real-time); the speed for the convergence depended upon the selected noise covariances, taking from

\(^{1}\)A very bold assumption; a better procedure is to use any of the camera calibration processes to get the correct parameters
2 to 19 iterations to fine tune the estimation. The results of an experiment using control points only in the floor and the wall are shown in Fig. 4-3: the principal point (pp) and the image plane are also depicted. The fact that the 3D positions of the floor control points appear to be between the pp and the image plane is explained realizing that this image plane is a virtual one and not the real one which should be symmetric and have an inverted image of the object. Qualitatively, the performance of the method is correct and works well for scene modeling purposes. Currently, tracking of features in a sequence consisting of 50 frames takes about 2 hours to complete (with human intervention), due to the large amount of memory required to handle images (720 × 480).

4.2 Preliminary Results using Extended Stereo Algorithm

This method was not fully developed. The weakness of this method was the same as the weakness of the Stereo Algorithms: the correlation process, which resulted in
Figure 4-2: (a) and (b) A pair of stereo images and their corresponding epipolar lines. (c) and (d) A pair of stereo images rectified. (e) Disparity Map.
Figure 4-3: Structure Estimation for the wall and the floor.

a greater task than expected for the scope of this thesis. Partial results can be seen in Fig. 4-2. Intermediate steps to obtain images, adequate as inputs for any Stereo Algorithm, are shown in Fig. 4-2 (a)-(d); in this case a pair of images with parallel epipolar lines is generated from a pair of images from the sequence. A disparity map from this rectified pair of images was created using the correlation method previously explained in 2.2.3 (see Fig. 4-2 (e)). The darker dots correspond to points located away from the camera, the lighter dots correspond to points closer to the camera, while the white regions correspond to unmatched dots). As can be seen, the Disparity map gives an estimation of the depth of each point in the scene. However, there is still a lot of work to be done before better results can be obtained. Some parts of the Marr-Poggio-Grimson algorithm were implemented, but it was not completed in order to dedicate more efforts to the manual modeling.

4.3 Manual Modeling

Manual Modeling offers immediate results, although the implementation of the user interface is not trivial and requires complex technical capacitiation as well as a large
Figure 4-4: Modeling a plane. (a) Selection of the points. (b) Selection of the corresponding surface. (c) Warping of the selected surface onto the three-dimensional plane.
Figure 4-5: 3D model rendered from different views with an orthographic projection.
variety of modeling tools. The modeling tools implemented in this thesis are quite simple. In Fig. 4-4 the modeling steps are shown:

1. Selection of Control dots which belong to a surface.

2. Selection of the texture corresponding to that surface.

3. Projection of the texture onto a fitted surface using least square methods.

4. Repetition of above steps for each desired surface.

There are some nuisances, especially in the borders and corners. The intersection of the floor with the wall (Fig. 4-5) is not connected but that can be solved using more complex surfaces for the model, such as polygons with two faces instead of two surfaces only. The selection of the texture for one surface is not restricted to be from one frame; it can be manually selected to be from the frame which contains more detail of the specific surface. In this way, a model containing information from the entire sequence can be generated.

Once the model is ready, many transformations can be realized to it. MATLAB limitations do not allow real-time rendering, but the 3D model can be ported to another software package more suitable to that task. In Fig. 4-5 three different orthographic views of the model are shown; the model is incomplete because control points were not tracked for all the surfaces.
Chapter 5

Conclusions

Currently, machine vision algorithms are not powerful enough to allow the automatic reconstruction of a scene from a series of still images. There are many unresolved topics which have been intensively researched since the beginning of computer vision study. Some researchers believe computer vision should not focus its goal in obtaining complete models of the surrounding world, although the applications of that task are very interesting, ranging from the generation of topographical maps to reengineering mechanical structures. A shortcut for this process consist of permitting an interactiveness between the computer and a human operator. If this approach is to be used, the methodology to create models would be to use any of the current CAD-CAM and/or computer graphics systems and add the basic techniques of 3D structure estimation; that would be much easier than implementing the basic techniques and then creating a new CAD-CAM system.

The performance of the model generation on a PC-386 is acceptable; although rendering of the images can improve using especial hardware for that purpose.

There are still many issues which should be researched. The automatization of the whole process is still a very desirable property if this method is to be used with a large quantity of video sequences. Again we face the main problem in Structure from video and Stereo Algorithms; correlation between frames. A better tracking algorithm for the control points should be designed. Instead of using the Hessian of the image to identify the corners, possibly an algorithm tuned to find corners
(taking into consideration occlusion) could be used. Using a similar idea to feature correlation, corners are found in one frame and are matched only with the corners of following frames reducing the search area. As previously mentioned, after the feature points are located and correlated in the entire sequence the structure and the camera parameters are estimated and can be used to find epipolar lines from frame to frame to limit the search to be unidimensional instead of em two dimensional.

Knowledge from the scene can also be used to do the geometric modeling; if the scene is composed mainly of regular objects (as man made constructions), surface fitting of polygons can be used for the image interpolation (instead of splines or geons).

Methods to obtain the structure from video should be extended to consider the full model of an object whereas the current ones need to correlate some control points between the first and the \( n^{th} \) frame; when the frame contains no information related to the first frame (perhaps the first frame has the picture of the front of the house and the current one is the picture of the back) they do not work. The first frame is then used as a reference frame for the estimation. An improvement would be to consider multiple reference frames in such a way that the structure does not depend from the first frame.
Appendix A

A.1 Marr Hildreth edge detector

The basis of the Marr-Hildreth edge detection scheme is the detection of zero crossings of the convolution of the operator $\nabla^2 G$ (A.1) (The Laplacian of the Gaussian) with an image [12, 17, 9, 29]. The reason why they chose that operator was not gratuitous, after extensive investigation they found a number of biological systems appeared to use operators similar to $\nabla^2 G$ in their low-level processing.

$$\nabla^2(x, y) = \left(\frac{x^2 + y^2}{\sigma^2} - 2\right) e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (A.1)$$

(A.1) can also be approximated by a difference of two Gaussians; in the one dimensional case:

$$DOG(\sigma_e, \sigma_i) = \frac{1}{\sqrt{2\pi} \sigma_e} e^{\frac{-x^2}{2\sigma_e^2}} - \frac{1}{\sqrt{2\pi} \sigma_i} e^{\frac{-x^2}{2\sigma_i^2}} \quad (A.2)$$

is an approximation to the second derivative where $\sigma_e \leq \sigma_i$ (generally $\sigma_i/\sigma_e \approx 1.6$). In the discrete case, we notice the Laplacian and the Gaussian are linear operations; thus the same result could be obtained first convolving the image with $G_\sigma$ (a separable operator) and then applying the discrete approximation of a Laplacian:

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-20</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
The size of the discrete version of the Gaussian depends upon the selected \( \sigma \); the size of the complete window is computed as:

\[
\omega = \sigma 2\sqrt{2}; \tag{A.3}
\]

The separable filter \( G_\sigma \) is obtained sampling a Gaussian:

\[
G_\sigma(x) = \frac{1}{\sqrt{2\pi \sigma}} e^{\frac{-x^2}{2\sigma^2}} \tag{A.4}
\]

The next step to make the edge detection consist of identifying the contours where the sign of the Laplacian of the Gaussian changes; this can be done defining a binary image (with value of one if the sign is positive and 0 if the sign is negative). From the binary image a point is selected as an edge element if it is 1 and one of it's eight neighbors is 0. The Marr-Grimson-Poggio stereo algorithm requires the detection of positive and negative zero crossings (with a positive or negative slope). An edge element in this case will be a positive zero crossing if it is equal to 1 and its left neighbor is 0; a negative zero crossing will have a binary value of one and a right neighbor 0. An example can is depicted in Fig. A-1.

### A.2 Quaternions

Some of the many ways to represent rotation include: Euler angles, Gibbs vector, Cayley-Klein parameters, Pauli spin matrices, axis and angle, orthonormal matrices and Hamiltons's quaternions [16, 14]. Unit quaternions are much simpler to use for some tasks than orthonormal rotation matrices, as quaternions are easier to normalize than enforce orthonormality in a matrix.

#### A.2.1 Definition

A quaternion \( \hat{q} \) can be thought as a vector with four components:

\[
\hat{q} = [q_0, q_1, q_2, q_3]^T \tag{A.5}
\]
Figure A-1: (a) Original figure. (b) Edge detection with $\sigma = 1.5$. (c) Edge detection with $\sigma = 3$
as a composite of a scalar and a vector with three components, or as an "hypercomplex" number with one real part and three imaginary parts:

\[ \dot{q} = q_0 + i q_1 + j q_2 + k q_3 \]  
(A.6)

**Products of Quaternions**

Multiplication of quaternions can be defined in terms of the products of their components; if

\[
\begin{align*}
  i^2 &= -1, \quad j^2 = -1, \quad k^2 = -1; \\
  ij &= k, \quad jk = i, \quad ki = j; \\
  ji &= -k, \quad kj = -i, \quad ik = -j;
\end{align*}
\]  
(A.7)

the product \(\dot{r} \dot{q}\) (where \(\dot{r} = r_0 + i r_1 + j r_2 + k r_3\)) can be expanded as the product of an \(4 \times 4\) orthogonal matrix and a vector as follows:

\[
\dot{r} \dot{q} = \begin{bmatrix}
  r_0 & -r_1 & -r_2 & -r_3 \\
  r_1 & r_0 & -r_3 & r_2 \\
  r_2 & r_3 & r_0 & -r_1 \\
  r_3 & -r_2 & r_1 & r_0
\end{bmatrix} \dot{q}
\]  
(A.8)

In general, \(\dot{r} \dot{q} \neq \dot{q} \dot{r}\).

**Dot Products of Quaternions**

The dot product of two quaternions is the sum of the products of corresponding components:

\[
\dot{r} \cdot \dot{q} = r_0 q_0 + r_1 q_1 + r_2 q_2 + r_3 q_3
\]  
(A.9)

The dot product of a quaternion with itself is the square of its magnitude;

\[
\dot{q} \cdot \dot{q} = \|\dot{q}\|^2
\]  
(A.10)
so unit quaternions are the ones with magnitudes equal to 1.

Composition of Rotations

A very interesting property is the composition of rotations as multiplications of quaternions, i.e., if \( \hat{q} \) and \( \hat{r} \) are unit vectors which represent rotations, and \( \hat{r} \) is a rotation applied after \( \hat{q} \), \( \hat{r}\hat{q} \) represents the overall rotation.

Unit quaternions and rotations

In [16] Horn shows that the orthonormal matrix \( \mathbf{R} \) corresponding to the unit quaternion \( \hat{q} \) is:

\[
\mathbf{R} = \begin{bmatrix}
(q_0 + q_1 - q_2 - q_3)^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\
2(q_1q_2 + q_0q_3) & (q_0 - q_1 + q_2 - q_3)^2 & 2(q_2q_3 - q_0q_1) \\
2(q_1q_3 - q_0q_2) & 2(q_1q_2 + q_0q_3) & (q_0 - q_1 - q_2 + q_3)^2
\end{bmatrix}
\]  (A.11)

He also showed the rotation of an angle \( \theta \) about an axis given by the unit vector \( \hat{u} = [u_x, u_y, u_z]^T \) is represented as:

\[
\hat{q} = \cos(\theta/2) + \hat{u}\sin(\theta/2)
\]  (A.12)

Considering the Euler angles as rotations about the axis X, Y and Z, the overall rotation would be the multiplication of the corresponding angle rotation quaternion about each axis. If the angles are small, an approximation can be used:

\[
\delta\hat{q} \approx \begin{bmatrix}
\sqrt{\hat{q} - \epsilon} \\
\omega_x/2 \\
\omega_y/2 \\
\omega_z/2
\end{bmatrix}
\]  (A.13)

where

\[
\epsilon = (\omega_x^2 + \omega_y^2 + \omega_z^2)/4
\]  (A.14)

where \( \omega_x, \omega_y \) and \( \omega_z \) are the Euler angles.
A.3 Least Squares Fitting of Surfaces

If $a = Mb$, where $M$ is an $m \times n$ matrix, $a$ is a vector with $m$ components, and $b$ is a vector with $n$ components, Horn in [14] derives a least squares solution for the case when $k > n$ measurements $\{a_i\}$ and $\{b_i\}$ are available. The matrices $A$ and $B$ are formed adjoining the vectors $\{a_i\}$ and (The $i^{th}$ column of the matrix $A$ is $a_i$). Then

$$A = MB$$

(A.15)

Since the measurements are not perfect, an error vector $e$ with $m$ components is estimated as:

$$e_i = a_i - Mb_i$$

(A.16)

and adjoining the $k$ vectors:

$$E = A - MB$$

(A.17)

The solution to the minimization of $E$ is when:

$$M = AB^T(BB^T)^{-1}$$

(A.18)

where $B^T(BB^T)^{-1}$ is called the pseudoinverse of the nonsquare matrix $B$.

For plane fitting of a plane, the measurements (a set of 3-D points) should be given in (A.17) format. A plane can be parameterized as an unit vector $\hat{u} = [u_X, u_Y, u_Z]$ normal to the plane and one point pertaining to the plane $c = [c_X, c_Y, c_Z]$. The distance $e_i$ from the desired plane to one of the measurement points $p_i = [p_{iX}, p_{iY}, p_{iZ}]$ is:

$$e_i = (p_i - c) \cdot \hat{u} = (p_{iX} - c_X)u_X + (p_{iY} - c_Y)u_Y + (p_{iZ} - c_Z)u_Z$$

(A.19)

c_X and c_Y can be arbitrarily set to:

$$c_X = \frac{\sum_{i=1}^{k} p_{iX}}{k}$$

(A.20)

$$c_Y = \frac{\sum_{i=1}^{k} p_{iY}}{k}$$

(A.21)
̂u is restricted to be a unit vector (a non linear characteristic). Another approach is to use a parallel vector v to ̂u such that v = âu where a is an scalar, and fix one of the components of v \textsuperscript{1}. then the new error becomes \( e_{i_{\text{new}}} = a e_i = (p_i - c) \cdot v \), which is minimized with the same M matrix as in (A.17). For illustration purposes, lets suppose the selected component was \( v_z \). Then we have to find the values of \( c_z \), \( v_x \) and \( v_y \). From (A.19):

\[
ae_i = p_{iz} v_z - (v_x, v_y, c_z) \cdot (-p_{ix} + c_x, -p_{iy} + c_y, v_z) \quad (A.22)
\]

Then \( a_i = p_{iz} v_z \), \( b_i = [-p_{ix} + c_x, -p_{iy} + c_y, v_z]^T \) and \( M = [v_x, v_y, c_z] \). Adjoining the \( k \) vectors, matrix \( M \) can be obtained with the procedure shown above, giving the rest of the parameters needed to represent a plane (normalizing v to form the unit vector ̂u).

### A.4 Extended Kalman Filtering

Kalman filtering is a solution for the problem of parameter estimation; given several noisy observations, recovery of the information is required. A dynamic system can be described by the evolution of its state vector. When the state vector cannot be directly measured, indirect (usually corrupted by noise) observations should be done. Denoting the state vector as \( x \) and the measurement vector by \( y \), a dynamic system (in discrete time form) can be described as:

\[
x[n + 1] = \Phi(x[n]) + \xi[n] \quad (A.23)
\]

\[
y[n + 1] = h(x[n]) + \nu[n] \quad (A.24)
\]

\textsuperscript{1}Special care should be taken not to choose a component which is close to zero. The variance for each X, Y and Z component in the set \( P_i \) is computed, and the component with least variance is choosen.
where the noises $\xi[n]$ and $\nu[n]$ are usually modelled as Gaussian white noise:

$$E[\xi[n]] = 0 \quad E[\xi[n]\xi^T[n]] = Q[n]$$  \hfill (A.25)

$$E[\nu[n]] = 0 \quad E[\nu[n]\nu^T[n]] = R[n]$$  \hfill (A.26)

In practice the noise parameters are determined on the basis of "experience and intuition"\(^2\). The standard Kalman filter assumes the transformations $\Phi$ and $h$ are linear. If that is not the case, the Extended Kalman Filter (EKF) approach is used. Basically the extension consist of updating a linearization around the previous state estimate (considering the linear terms of a Taylor approximation).

### A.4.1 EKF for Recursive estimation of motion, structure and focal length

In this specific implementation, $\Phi$ is chosen to be an Identity matrix ($I$), while $h$ is a nonlinear relation. The estimate of the state vector $\hat{x}[n_1|n_2]$ represents the linear least squares estimate (LLSE) of $x[n_1]$ based upon $y[n]$ such that $n \leq n_2$. The update equations are:

$$\hat{x}[n|n] = \hat{x}[n|n - 1] + K[n](y[n] - h(\hat{x}(n|n - 1)))$$  \hfill (A.27)

$$P[n|n] = (I_{KC} - K[n]C[n])P[n|n - 1]$$  \hfill (A.28)

where $I_{KC}$ is the identity matrix with the same size as $K[n]C[n]$ and $P[n_1|n_2]$ is defined as the error covariance of the LLSE:

$$P[n_1|n_2] = E[(x[n_1] - \hat{x}[n_1|n_2])(x[n_1] - \hat{x}[n_1|n_2])^T]$$  \hfill (A.29)

\(^2\)They are 'guessed'.
The Kalman gain optimized to minimize the LLSE is computed from

\[ K[n] = P[n|n - 1]C^T[n](C[n]P[n|n - 1]C^T[n] + R[n])^{-1} \]  

(A.30)

\( C[n] \) are the linear components of the Taylor expansion:

\[ C[n] = \left[ \frac{\partial h}{\partial x} \right]_{x = \hat{x}[n|n-1]} \]  

(A.31)

The prediction step is:

\[ \hat{x}[n + 1|n] = \Phi[n] \hat{x}[n|n] = \hat{x}[n|n] \]  

(A.32)

\[ P[n + 1|n] = \Phi[n]P[n|n]\Phi^T[n] + Q[n] = P[n|n] + Q[n] \]  

(A.33)

In this particular implementation, some additional computations are required; before (A.27) and (A.32), the Euler angles in \( \hat{x} \) are extracted and the rotation is composed (see A.2) into an external quaternion.

For a detailed description of Kalman filtering and its implementation, please refer to [6, 34].
Appendix B

MATLAB Code

All the programming was done using MATLAB 4.0 with the Signal Processing and Image Processing Toolboxes in a PC-386.

B.1 Interactive Control Points Tracking

This section consist of only one program which lets the user interactively track the control points.

B.1.1 menu3d.m

function menu3dd(action,in1,in2);
% menu3dd(action,in1,in2);
% This version is to be used with a PC.
% the sequence of pictures should be saved
% as a sequence of mat files, each one of them containing
% the frame as fi.

if nargin<1,
    action='start';
end;

10

55
% Global variables.

global PIC_DAT PIC_NAM MENU3D_DAT PIC1 PIC2 TEXT1_H TEXT2_H er_mode

if strncmp(action, 'start'), strcmp(action, 'refresh'),
    % graphics initialization
    % It seems the next line does not work on a PC
    if strcmp(action, 'refresh');
        clg;
        'pase por refresh !'

    % SUBSTITUTE WITH GLOBAL PARAMETERS !!!
    hessian_flag = 0;
    colormap(gray(256));
    frames = 49;
    nfeat = 15;
    w=20;
    win = 4;
    win2 = 14;
    rib = 15;

elseif strncmp(action, 'start');
    figure('Pos', [.02 .05 0.95 0.45], 'name', '3D reconstruction', ... 
        'Num', 'off', 'Units', 'normalized', 'backingstore', 'off');

    % The correct next line should inquire the levels of gray.
    % Also, the number of frames should be saved in some other place

    % Hessian is not used now.

    hessian_flag = 0;
    colormap(gray(256));
    frames = 49;
    nfeat = 15;
    w=20;
    win = 4;
    win2 = 14;
    rib = 15;
% The purpose of er_mode is to avoid the constant "redrawing" of a
% frame after moving a feature.
er_mode = 'background';

% In MENU3D_DAT all the variables we need from action to action are found;

MENU3D_DAT = zeros([1 20]);
MENU3D_DAT(1) = 1; % Left side frame = 1
MENU3D_DAT(2) = 2; % Right side frame = 2 (n-1)

% PIC_NAM = '/mas/garden/work/repo-man/gerardo/';

%%% This part should be changed, using uigetfile to allow the user to select a
%%% sequence.

PIC_NAM = '/gerardo/thesis/sequences/';
PIC_DAT = zeros([n_feat*2 frames]);
end;

% The next lines define the position of the graphical interface

% To select a frame
im_text_1=uicontrol('Style', 'edit', 'Position', [.4 .02 .04 .07],... 'Units', 'normalized', 'String', num2str(MENU3D_DAT(1)),...
'CallBack', 'menu3d(''Set'',1);'

im_text_2=uicontrol('Style', 'edit', 'Position', [.75 .02 .04 .07],...
'Units', 'normalized', 'String', num2str(MENU3D_DAT(2)),...
'CallBack', 'menu3d(''Set'',2);'

acc_im_1=uicontrol('Style', 'Pushbutton', 'Position', [.02 .4 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Accept'',1), 'String', 'Accept');

acc_im_2=uicontrol('Style', 'Pushbutton', 'Position', [.91 .4 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Accept'', 2)', 'String', 'Accept');

tr_im_1=ucicontrol('Style', 'Pushbutton', 'Position', [.02 .33 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Track'', 1)', 'String', 'Track');

tr_im_2=ucicontrol('Style', 'Pushbutton', 'Position', [.91 .33 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Track'', 2)', 'String', 'Track');

next_im_1=ucicontrol('Style', 'Pushbutton', 'Position', [.24 .02 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Next'', 1,1)', 'String', 'Next');

prev_im_1=ucicontrol('Style', 'Pushbutton', 'Position', [.31 .02 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Next'', 1,2)', 'String', 'Prev');

next_im_2=ucicontrol('Style', 'Pushbutton', 'Position', [.59 .02 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Next'', 2,1)', 'String', 'Next');

prev_im_2=ucicontrol('Style', 'Pushbutton', 'Position', [.659 .02 .07 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''Next'', 2,2)', 'String', 'Prev');

close_button=ucicontrol('Style', 'Pushbutton', 'Position', [.02 .02 .12 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''done''), 'String', 'Done');

refresh_b=ucicontrol('Style', 'Pushbutton', 'Position', [.87 .02 .12 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''refresh''), 'String', 'Refresh');

feat_file=ucicontrol('Style', 'Popup', 'Position', [.02 .9 .18 .07],...
'Units', 'normalized', 'Callback', 'menu3d(''feat.p''), 'String', ...
'Save Control Points | Load Control Points');

% Picture 1
pos_ax = [0.15 0.1 0.35 .8; .5 .1 .35 .8];

pic_1_ax=axes('Position',pos_ax(1,:), 'AspectRatio',[0.646 1], ...
    'visible','off');

if strcmp(action,'start'),
    % For a PC, the next modification should be done.
    % maybe the same code can be used for both, including a flag or something
    temp_str  = [PIC_NAM,'cubo',int2str(MENU3D_DAT(1))];
    temp_str_2 = ['load ',temp_str];
    eval(temp_str_2);
    PIC1 = zer_pad(fi,rib,rib,rib,rib); clear fi;
end;

im1_han=imshow(PIC1);

hold on;

numbers = zeros([n_feat,3]);
for i=1:n_feat
    if i<=9
        numbers(i,1:3)=[' ',num2str(i), ' '];
    else
        numbers(i,1:3)=[' ',num2str(i)];
    end;
end;

if hessian_flag == 1
    hes = abs(hessian(PIC1));
    [i,j]=size(hes);
    w_.=w+rib;
    hes = hes(w_:i-w_,w_:j-w_);
    temp=zeros([i,j]);
    temp(w_:i-w_,w_:j-w_) = hes;
    hes = temp;
    clear temp;

end;
xy = zeros([n_feat 2]);

h = ones([n_feat 1]);
for co = 1:n_feat;
    if max(max(hes)) > 0;
        [i,j] = max_loc(hes);
        hes(i-w:i+w,j-w:j+w) = zeros(2*w+1);
        xy(co,:)=[i,j];
    end
end

xy_=xy';
else
    [r c] = size(PIC1);
    rc = min([r c]);
    xy_ = [1:n_feat', 1:n_feat]*rc/2/n_feat;
    xy_ = xy_';
end;

if strcmp(action,'start');

    % Just for a while, I am going to use:
    xy = [1:15; 1:15]*30+20;
    xy_ = xy';
    PIC_DAT(:,MENU3D_DAT(1))=xy(:,);
    PIC_DAT = xy(:,)*ones([1 frames]);
end;

temp=reshape(PIC_DAT(:,MENU3D_DAT(1)),2,length(PIC_DAT(:,MENU3D_DAT(1)))/2);
TEXT1_H = text(temp(2,:),temp(1,:),numbers, ...'
erasemode',er_mode);
pic1_han = line(temp(2,:),temp(1,:),'linestyle','o', ...'
erasemode','background');
set(gca,'visible','off') ;
% Picture n

if strcmp(action,'start'),
    temp_str = [PIC_NAM,'cubo',int2str(MENU3D_DAT(2))];
% PIC2 = dat2mat(temp_str);
    temp_str_2 = ['load ',temp_str];
    eval(temp_str_2);
    PIC2 = zer_pad(fi,rib,rib,rib,rib); clear fi;
end;

pic_2_ax=axes('Position',pos_ax(2,:), 'AspectRatio',[0.646 1], ...
    'visible','off');

im2_han=imshow(PIC2);
hold on;
if strcmp(action,'start'),
    PIC_DAT(:,MENU3D_DAT(2))=xy(:,);
end;
temp=reshape(PIC_DAT(:,MENU3D_DAT(2)),2,length(PIC_DAT(:,MENU3D_DAT(2))))/2;

TEXT2_H = text(temp(2,:),temp(1,:),numbers, ...
    'erasemode',er_mode);
pic2_han = line(temp(2,:),temp(1,:), 'linestyle','o', ... 
    'erasemode','background');
set(gca,'visible','off');

set(pic1_han,'ButtonDownFcn','menu3d(''down'',1)');
set(pic2_han,'ButtonDownFcn','menu3d(''down'',2)');
drawnow;

if strcmp(action,'start'),

61
MENU3D_DAT = [1; 2; pic1_han; pic2_han; pic_1_ax; pic_2_ax; 0; ...
   im1_han; im2_han; frames; im_text_L; im_text_2; 0; 0; 0; 0; ...
   n_feat; win; win2; rib; feat_file];
else
   MENU3D_DAT = [MENU3D_DAT(1); MENU3D_DAT(2); pic1_han; pic2_han; pic_1_ax; pic_2_ax; 0; ...
   im1_han; im2_han; frames; im_text_L; im_text_2; 0; 0; 0; 0; ...
   n_feat; win; win2; rib; feat_file];
end;

elseif strcmp(action,'down'),
   % assumes that a line was clicked

   if (in1==1),
      line_handle = MENU3D_DAT(3);
      ax = MENU3D_DAT(5);
      text_h = TEXT1_H;
   else  % assume in1 == 2 otherwise (might not be true)
      line_handle = MENU3D_DAT(4);
      ax = MENU3D_DAT(6);
      text_h = TEXT2_H;
   end;

   if (ax^=gca),
      axes(ax);
      % drawnow discard;
   end;

   % Obtain coordinates of mouse click location in axes units
   pt=get(gca,'currentpoint');
   x=pt(1,1);
   y=pt(1,2);

   set(line_handle,'Color','r','erasemode','xor');
   for i=1:length(text_h); set(text_h(i),'erasemode','xor'); end
% find closest point on line to mouse click loc (call it fixed_x, fixed_y)
line_x = get(line_handle, 'XData');
line_y = get(line_handle, 'YData');
dist = (line_x-x).^2 + (line_y-y).^2;
[temp, i] = min(dist);
fixed_x = line_x(i);
fixed_y = line_y(i);

% drawnow;

MENU3D_DAT(7) = i;

% set(gcf, 'WindowButtonDownFcn', sprintf('menu3d(''move'',%g',in1));
set(gcf, 'WindowButtonDownFcn', sprintf('menu3d(''up'',%g',in1));

elseif strcmp(action, 'move'),

if (in1 == 1),
    line_handle = MENU3D_DAT(3);
    ax = MENU3D_DAT(5);
    text_h = MENU3D_DAT(8);
else  % assume in1 == 2 otherwise (might not be true)
    line_handle = MENU3D_DAT(4);
    ax = MENU3D_DAT(6);
    text_h = MENU3D_DAT(8);
end;

if (ax == gca),
    axes(ax);
% drawnow discard;
end;
pt=get(gca,'currentpoint');
x=pt(1,1);
y=pt(1,2);

ydat = get(line_handle,'Ydata');
xdat = get(line_handle,'Xdata');
ydat(MENU3D_DAT(7))=y;
xdat(MENU3D_DAT(7))=x;
set(line_handle,'YData',ydat,'Xdata',xdat);
for i=1:length(text_h); set(text_h(i),'position',[xdat(i) ydat(i)]); end

% i=MENU3D_DAT(7);
% set(text_h(i),'position',[xdat(i) ydat(i)]);
xy = [ydat',xdat'];

PIC_DAT(:,MENU3D_DAT(7))=xy(:,);

elseif strcmp(action,'up'),

if (in1==1),
    line_handle = MENU3D_DAT(3);
    ax = MENU3D_DAT(5);
    text_h = TEXT1_H;

else % assume in1 == 2 otherwise (might not be true)
    line_handle = MENU3D_DAT(4);
    ax = MENU3D_DAT(6);
    text_h = TEXT2_H;
end;

pt=get(gca,'currentpoint');
x=pt(1,1);
y=pt(1,2);

ydat = get(line_handle,'Ydata');
xdat = get(line_handle,'Xdata');
ydat(MENU3D_DAT(7)) = y;
xdat(MENU3D_DAT(7)) = x;
set(line_handle,'color','y','erasemode','background');
set(line_handle,'YData',ydat,'XData',xdat);
for i = 1:length(text_h); set(text_h(i), 'position',[xdat(i) ydat(i)],...
    'erasemode','er_mode'); end

elseif strcmp(action,'redraw'),

    if (in1==1),
        text_h = TEXT1_H;
        im_han = MENU3D_DAT(8);
        line_handle = MENU3D_DAT(3);
        temp_pic=PIC1;
    else  % assume in1 == 2 otherwise (might not be true)
        im_han = MENU3D_DAT(9);
        line_handle = MENU3D_DAT(4);
        text_h = TEXT2_H;
        temp_pic=PIC2;
    end

    set(im_han,'Cdat',temp_pic);
    temp=round(reshape(PIC_DAT(:,MENU3D_DAT(in1)), ... 1 2,length(PIC_DAT(:,MENU3D_DAT(in1)))/2));
    set(line_handle,'xdat',temp(2,:),'ydat',temp(1,:));
    for j = 1:length(text_h); set(text_h(j), 'position',[temp(2,j) temp(1,j)]);end;

    % drawnow;
elseif strcmp(action,'Set'),

    if (in1==1),


line_handle = MENU3D_DAT(3);
ax = MENU3D_DAT(5);
text_h = TEXT1_H;
im_han = MENU3D_DAT(8);
im_text = MENU3D_DAT(11);
else % assume in1 == 2 otherwise (might not be true)
    line_handle = MENU3D_DAT(4);
    ax = MENU3D_DAT(6);
    text_h = TEXT2_H;
    im_han = MENU3D_DAT(9);
    im_text = MENU3D_DAT(12);
end;

min_fr=1;
max_fr=MENU3D_DAT(10);
fr=str2num(get(im_text,'string'));
if isempty(fr), % handle non-numeric input into field
    set(im_text,'string',num2str(MENU3D_DAT(in1)));
else
    if (fr>max_fr),
        fr=max_fr;
    end;
    if (fr<min_fr),
        fr=min_fr;
    end;
    MENU3D_DAT(in1)=round(fr);
end

temp_str = [PIC_NAM,'cubo',int2str(MENU3D_DAT(in1))];
% PIC2 = dat2mat(temp_str);
temp_str_2 = ['load ',temp_str];
eval(temp_str_2);
rib = MENU3D_DAT(20)
pic = zer_pad(f1,rib,rib,rib,rib); clear f1;
temp=round(reshape(PIC_DAT(:,MENU3D_DAT(in1)),2,...
length(PIC_DAT(:,MENU3D_DAT(in1))))/2));

set(line_handle,'xdat',temp(2,:),'ydat',temp(1,:));
for j=1:length(text_h); set(text_h(j), 'position', [temp(2,j) temp(1,j)]); end;

set(im_text, 'string', int2str(MENU3D_DAT(in1)));
set(im_han, 'Cdata', pic);

if (in1==1),
    PIC1=pic;
else    % assume in1 == 2 otherwise (might not be true)
    PIC2=pic;
end;
elseif strcmp(action, 'Next'),

if (in1==1),
    line_handle = MENU3D_DAT(3);
    ax = MENU3D_DAT(5);
    text_h = TEXT1_H;
    im_han = MENU3D_DAT(8);
    im_text = MENU3D_DAT(11);
else    % assume in1 == 2 otherwise (might not be true)
    line_handle = MENU3D_DAT(4);
    ax = MENU3D_DAT(6);
    text_h = TEXT2_H;
    im_han = MENU3D_DAT(9);
    im_text = MENU3D_DAT(12);
end;

if (in2==1)
    i=rem(MENU3D_DAT(in1)+1,MENU3D_DAT(10)+1);
    if i==0,i=1;end;

67
else
    i=MENU3D_DAT(in1)-1;
    if (i<=0)
        i=MENU3D_DAT(10);
    end;
end;
MENU3D_DAT(in1) = i;

    temp_str = [PIC_NAM,'cubo',int2str(MENU3D_DAT(in1))];
% PIC2 = dat3mat(temp_str);
    temp_str_2 = ['load ',temp_str];
    eval(temp_str_2);
    rib = MENU3D_DAT(20)
    pic = zer_pad(fl,rib,rib,rib,rib); clear fl;

    temp=round(reshape(PIC_DAT(:,MENU3D_DAT(in1)),2,...
            length(PIC_DAT(:,MENU3D_DAT(in1)))/2));

    set(line_handle,'xdat',temp(2,:), 'ydat',temp(1,:));
    for j=1:length(text_h); set(text_h(j), 'position',[temp(2,j) temp(1,j)]);end;

    set(im_text,'string',int2str(i));
    set(im_han,'Cdata',pic);

if (in1==1),
    PIC1=pic;
else  % assume in1 == 2 otherwise (might not be true)
    PIC2=pic;
end;

% *****
% Menu('Track',in1);
else if strcmp(action,’Accept’),

    if (in1==1),
        line_handle = MENU3D_DAT(3);
        ax = MENU3D_DAT(5);
        text_h = TEXT1_H ;
        im_han = MENU3D_DAT(8) ;
        im_text = MENU3D_DAT(11);
        temp_pic = PIC1;
    else   % assume in1 == 2 otherwise (might not be true)
        line_handle = MENU3D_DAT(4);
        ax = MENU3D_DAT(6);
        text_h = TEXT2_H;
        im_han = MENU3D_DAT(9);
        im_text = MENU3D_DAT(12);
        temp_pic = PIC2;
    end;
    temp = get(line_handle,’xdat’);
    temp2 = get(line_handle,’ydat’);
    temp =[temp2; temp];
    temp =temp(:);
    PIC_DAT(:,MENU3D_DAT(in1))=round(temp);
    set(im_han,’Cdat’,temp_pic);

    drawnow;

else if strcmp(action,’Track’),

    if (in1==1),
        line_handle = MENU3D_DAT(3);
        ax = MENU3D_DAT(5);
        text_h = TEXT1_H ;
        im_han = MENU3D_DAT(8) ;
        im_text = MENU3D_DAT(11);
        temp_pic = PIC1;

else  % assume in1 == 2 otherwise (might not be true)
    line_handle = MENU3D_DAT(4);
    ax = MENU3D_DAT(6);
    text_h = TEXT2_H;
    im_han = MENU3D_DAT(9);
    im_text = MENU3D_DAT(12);
    temp_pic = PIC2;
end;

frames = 29;
n_feat = MENU3D_DAT(17);
win = MENU3D_DAT(18);
win2 = MENU3D_DAT(19);
rib = MENU3D_DAT(20);
%  w=20;
%  win = 4;
%  win2 = 14;
%  rib = 15;

match = zeros(2*win+1);
corre = match;
    xy=reshape(PIC_DAT(:,MENU3D_DAT(1)),2,length(PIC_DAT(:,MENU3D_DAT(1))))/2);
%  temp=reshape(PIC_DAT(:,MENU3D_DAT(in1)),2,length(PIC_DAT(:,MENU3D_DAT(in1))))/2);
    xy_=xy';
    xy = xy';
    f = MENU3D_DAT(in1);

for co = 1:n_feat;
    i=xy(co,1);
    j=xy(co,2);
    match = PIC1(i−win2:i+win2,j−win2:j+win2);

    i=xy_(co,1);
    j=xy_(co,2);
for coi = -win:win
    for coj = -win:win

        i_ = (i+coi-win2):(i+coi+win2);
        j_ = (j+coj-win2):(j+coj+win2);

        \[ [i_', j_'] \]
        \[
        \text{size(PIC1)}
        \]
        \[
        \text{min([i_', j_'])}
        \]
        \[
        \text{max([i_', j_'])}
        \]
        \[
        \text{prod(min([i_', j_'])} \geq [1, 1]) \& \text{prod(max([i_', j_'])} \leq \text{size(PIC2)}
        \]
        if \[
        \text{prod(min([i_', j_'])} \geq [1, 1]) \& \text{prod(max([i_', j_'])} \leq \text{size(PIC2)},
        \]
        \[
        \text{corre(coi+win+1,coj+win+1) = sum(sum(abe(match-PIC2(i,j))))};
        \]
        \[
        \text{corre(coi+win+1,coj+win+1) = cor2(match,PIC2(i,j))};
        \]
        \[
        \text{end};
        \]
        \[
        \text{end};
        \]
        \[
        \text{end};
        \]

        [imax,jmax] = max_loc(corre);

        \[
        \text{subplot(2,3,f)};
        \]
        \[
        \text{imax,jmax}
        \]
        \[
        \text{PIC_DAT(co*2-1:co*2,f)=[ imax+i-win-1,jmax+j-win-1]'};
        \]
        \[
        \text{xy_}(co,1)=imax+i-win-1;
        \]
        \[
        \text{xy_}(co,2)=jmax+j-win-1;
        \]
        \[
        \text{xy(co,1:2)}
        \]
        \[
        \text{xy_}(co,1:2)
        \]
        \[
        \text{end};
        \]
        \[
        \text{PIC_DAT}
        \]
        \[
        \text{menu3d( 'redraw', in1)};
        \]

    \[
    \text{elseif strcmp(action, 'feat_p'),}
    \]
    \[
    \text{val = get(MENU3D_DAT(21), 'Value')}
    \]
    \[
    \text{if val ==1;}
    \]
    \[
    \text{[filename, path] = uiputfile( '*.mat', 'Save Feature Points');}
    \]
    \[
    \text{factor = length(PIC1);}
    \]
pp = size(PIC1)/2;
eval(['save ', path, filename, ' PIC_DAT rib factor pp']);

elseif val ==2:

[filename, path] = uigetfile('*.*.mat', 'Load Feature Points');
eval(['load ', path, filename]);
temp=round(reshape(PIC_DAT(:,MENU3D_DAT(1)),... 2,length(PIC_DAT(:,MENU3D_DAT(1))))/2));
set(MENU3D_DAT(3),'xdat',temp(2,:), 'ydat',temp(1,:));
for j=1:length(TEXT1_H); set(TEXT1_H(j), 'position', [temp(2,j) temp(1,j)]);end;
temp=round(reshape(PIC_DAT(:,MENU3D_DAT(2)),... 2,length(PIC_DAT(:,MENU3D_DAT(2))))/2));
set(MENU3D_DAT(4),'xdat',temp(2,:), 'ydat',temp(1,:));
for j=1:length(TEXT2_H); set(TEXT2_H(j), 'position', [temp(2,j) temp(1,j)]);end;
menu3d('redraw',1);
menu3d('redraw',2);
end;
elseif strcmp(action,'done'),
save name PIC_DAT
clf reset;
clear global SIGDEMO2_DAT
close;
end

B.2 Structure Estimation using EKF

Once the data of the control points is available, the EKF computes the focal length, motion and structure with the following functions.

B.2.1 estimate.m

% Script used to get the estimation of the 3D location of the
% control points. Uses as input the 2d locations of the
% control points as given by MENU3D
% The whole procedure parallels Ali's article.

% plot_fag = 1 if a graphic representation of the estimation
% is going to be shown.
plot_flag =1;

iterations = 600;

% tolerance, valid from 0 to 1
tolerance = 0.001;

%load m_5.26

% Obsolete, use uigetfile
load est_5.30; rib = 0; PIC_DAT = P;
% WARNING -- is using est_5.30; which is a combination
% of all the previous tracked control dots.
% (stored in P)
% CHECK ******************

initial = 1;
% factor = 720;
% factor used to \( \approx \) 720.

xy_p = PIC_DAT;
[dummy frames] = size(xy_p)
dum = [497; 720]*ones([1 dummy/2]);
%dum = [497; 720]/2*ones([1 dummy/2]);
dum = reshape(dum,1,dummy)^*ones([1 frames]);

% scale xy_p (for better convergence)
xy_p = (PIC_DAT -rib -dum/2)/factor;
% Here the process begins: All the above stuff is used only
% to generate an adequate xy_p matrix.
% that matrix is in the followinf format:
%
% frame -> 1 2 3 ..... n feature
% xy_p = [ x x x ... x ] 1
% [ y y y ... y ] 1
% [ ... ]
% [ x x x ... x ] p
% [ y y y ... y ] p
%

% Begin iterative extended Kalman Filter,

count = 0;
iter = (1==1);
prev_beta = 10000;

% The following is an implementation of the ITERATED
% Kalman filter.
%
% It still does not handle additions and removal
% of control points

while iter

    count = count +1;
n=length(xy_p(:,1))/2;

% WARNING those values should be changed for better performance.
% Right now they are only GUESSED !!!!
    noise = 0.1;     % 10 %
confidence = 0.1; % low confidence for initial values

% State Matrix
A = eye(n+7);

% Assumptions and Initial conditions:
% covariance R and Q are diagonal, with some noise ± ? pixels
R = noise*eye(n*2);
Q = noise*eye(n+7);

P_p = zeros(n+7);
P_p(1:6,1:6) = eye(6);
% P_p = eye(n+7);
% P_p(1:7,1:7) = zeros(7); % first six elements are zero since the initial % state is choosen.
P_p = P_p*confidence; % Since we do not know the initial values, % the variance should be high.

% Initial values:
q = [1 0 0 0]'; % No rotation.
x_p = zeros([n+7 1]); % nothing new.

% IN CASE OF INVERTED ESTIMATION (i.e., if the final beta is % negative, use a negative initial x_p).
x_p(7) = 0.001; %

q_const = q;
x_const = x_p;

% Set initial state
if count == 1
    betas = zeros([n+7 frames]);
    q_s = zeros([4 frames]);
end;
q = q;
count
if count > 1
    x_p = betas(:,frames);
    x_p(1:3,:) = [0 0 0];
end;

for i = 1:frames
c = i;
    XY = xy_p(:,i);
    if i == 1,
        XY_0 = XY;
        p = p_set(XY_0,x_p,q);
    end;

% LINEARIZATION:
    C = dh(x_p,q,0.0001,XY_0);

% KALMAN ESTIMATION
    [x,x_n,K,P,P_n,q] = est_x(x_p,P_p,A,C,Q,R,p,q,XY,XY_0);

    x_p = x_n;
    P_p = P_n;

% 3-D estimation
    p = p_set(XY_0,x,q);

    betas(:,i) = x;
    q_s(:,i) = q;

% Below code only use to show graphically how the estimation is going.
    p_xyz = reshape(p,3,length(p)/3);
    drawnow
    p = p_set(XY_0,x,q);

    if plot_flag == 1
\( y_{\text{seen}} = h(x_n,q,XY_0); \)
\( y_{\text{sn}} = \text{reshape}(y_{\text{seen}},2,\text{length}(y_{\text{seen}})/2); \)
\( p_{\text{xyz}} = \text{reshape}(p,3,\text{length}(p)/3); \)
\( px = p_{\text{xyz}}(1,:); \)
\( py = p_{\text{xyz}}(2,:); \)
\( pz = p_{\text{xyz}}(3,:); \)
\([s1,s2]=\text{size}(\text{PIC\_DAT});\)
\( xy = \text{reshape}(xy_p(:,i),2,s1/2); \)
\( \text{if count}^t_i == 1 \)
\( \quad \text{subplot}(1,2,2) \)
\( \quad y_{\text{seen\_line}} = \text{line}(y_{\text{sn}}(1,:),y_{\text{sn}}(2,:), 'linestyle', 'o', 'color', 'red', 'erasemode', 'background'); \)
\( \quad xy_{\text{line}} = \text{line}(xy(1,:),xy(2,:), 'linestyle', 'o', 'erasemode', 'background'); \)
\( \quad \text{subplot}(1,2,1);p_{\text{xyz\_line}} = \text{line}(px,py,pz, 'linestyle', 'o', 'erasemode', 'background'); \text{view}(3); \)
\( \quad \text{axis} = ([-500 500 -500 500 -500 500]); \)
\( \quad \text{else} \)
\( \quad \text{set}(y_{\text{seen\_line}}, 'xdata', y_{\text{sn}}(1,:), 'ydata', y_{\text{sn}}(2,:)); \)
\( \quad \text{set}(xy_{\text{line}}, 'xdata', xy(1,:), 'ydata', xy(2,:)); \)
\( \quad \text{set}(p_{\text{xyz\_line}}, 'xdata', px, 'ydata', py, 'zdata', pz); \)
\( \quad \text{end}; \)
\( \text{end}; \)
\( \text{end}; \)
\( \text{iter} = \text{abs}(\text{prev\_beta}-\text{betas}(7,i))/\text{abs}(\text{betas}(7,i))>\text{tolerance}; \)
\( \text{prev\_beta} = \text{betas}(7,i) \)

\% CHANGE FOR UIPUTFILE
\% save pr_6_7 betas q_s rib XY_0

end;

\% save pr_5_26 betas q_s rib
function [x,x_n,K,P,n,q]=est_x(x,p,P,p,A,C,Q,R,p,q,y,y_0);

% function [x,x_n,K,P,n,q]=est_x(x,p,P,p,A,C,Q,R,p,q,y,y_0)
% x(t+1) = Ax(t) + w(t)
% y(t) = h(x(t)) + v(t)
% Q = Ew(t)w'(t)
% R = Ev(t)v'(t)
% x_p = x(t|t-1)
% P_p = P(t|t-1)
% p = structure
% q = rotation
% Outputs:
% x = updated x.
% x_n = predicted a
% K = Kalman Gain
% 
% Used for the EKF, evaluates the estimated x and K.
% See Kalman filter.

% Evaluate Kalman Gain:
K = P_p*C'*inv(C*P_p*C' + R);
I = eye(size(K*C));

% For this implementation only: extract quaternion:
epsilon = (x_p(4)^2 + x_p(5)^2 + x_p(6)^2)/4;
d_q = [(1-epsilon)^(1/2); x_p(4:6)/2];
q = quat_mult(q,d_q);
q = q/norm(q);
%q(2:4) = -q(2:4);

% Zero euler angles
x_p(4:6) = [0 0 0]';
% Update x
x = A*x_p + K*(y - h(x_p,q,y_0));
%x = A*x_p + K*(y - C*x_p);

P = (I-K*C)*P_p;

%[x(4) x(5) x(6)]

% Predict x

% For this implementation only: extract quaternion:
epsilon = (x(4)^2+x(5)^2+x(6)^2)/4;
d_q = [(1-epsilon)^(-1/2);x(4:6)/2];
q = quat_mult(q,d_q);
q = q/norm(q);
%q(2:4)=−q(2:4);

% Zero euler angles
x(4:6) = [0 0 0]';

x_n = A*x;
P_n = A*P*A' + Q;

---

B.2.3 h.m

function y=h(x,q,y_0);
% function y=h(x,q,y_0)
% x = state vector;
% q = external quaternion;
% y_0 = initial values of y;
% Used in EKF
% Computes the transformation h
%

% n = number of features
n = length(x) - 7;

% extract quaternion from the euler angles in the state vector
% and compose with external
% quaternion.
epsilon = (x(4)^2 + x(5)^2 + x(6)^2)/4;
d_q = [(1 - epsilon)^(-1/2), x(4:6)/2];
q_ = quat_mult(q, d_q);
q_ = q_/norm(q_);

% Get rotation matrix from the quaternion.
R = quat2mat(q_);

y = zeros([n + 2, 1]);
Ibeta = eye(3);
Ibeta(3, 3) = x(7);
beta = x(7);

% For all the feature points, compute transformation.
% See ALi's article.

for c = 1:n
    c_bis = 3*(c-1)+1;
c_b_b = 2*(c-1)+1;
u = y_0(c_b_b);
v = y_0(c_b_b+1);
XYZ = [u v 0]' + x(7+c)*[u*beta + v*beta + 1]';
% p(c_bis:(c_bis+2)) = XYZ;

% XYZc = [x(1) x(2) x(3)]' + Ibeta*R*p(c_bis:(c_bis+2));
XYZc = [x(1) x(2) x(3)]' + Ibeta*R*XYZ;

80
\[ uv = \frac{[XYZc(1) \, XYZc(2)]^*1}{1 + XYZc(3)} \];
\[ y(c_b_b:(c_b_b+1)) = uv; \]

end

**B.2.4 dh.m**

function H=dh(x,q,delta,yo);
% function H=dh(x,q,delta,yo);
% This function gives the matrix H required for the
% EKF. Is similar to Ali's C code.
% 
% See: Extended Kalman Filter Estimation.

\[ n = \text{length}(x) - 7; \]
\[ H = \text{zeros}([2*n \, \text{length}(x)]); \]

\[ Q0 = q(1); \]
\[ Q1 = q(2); \]
\[ Q2 = q(3); \]
\[ Q3 = q(4); \]

\[ q00 = Q0*Q0; \quad q01 = Q0*Q1; \quad q02 = Q0*Q2; \quad q03 = Q0*Q3; \]
\[ q11 = Q1*Q1; \quad q12 = Q1*Q2; \quad q13 = Q1*Q3; \]
\[ q22 = Q2*Q2; \quad q23 = Q2*Q3; \]
\[ q33 = Q3*Q3; \]

\[ c0 = 2*(q01 + q23); \quad c1 = 2*(q01 - q23); \]
\[ c2 = 2*(q02 + q13); \quad c3 = 2*(q02 - q13); \]
\[ c4 = 2*(q03 + q12); \quad c5 = 2*(q03 - q12); \]

\[ k0 = q00 + q11 - q22 - q33; \]
\[ k1 = q00 - q11 + q22 - q33; \]
\[ k2 = q00 - q11 - q22 + q33; \]
beta = x(7);
dTdO = zeros([9 1]);

for count = 1:n

    alpha = x(7+count);

    cx = 2*(count-1) + 1;
    cy = 2*count;

    y1 = yo(cx);
    y2 = yo(cy);

    ph1 = (1. + alpha * beta) * y1;
    ph2 = (1. + alpha * beta) * y2;
    ph3 = alpha;

    Rchph1 = k0*ph1 - c5*ph2 + c2*ph3;
    Rchph2 = c4*ph1 + k1*ph2 - c1*ph3;
    Rchph3 = (-c3)*ph1 + c0*ph2 + k2*ph3;

    pc1 = Rchph1 + x(1);
    pc2 = Rchph2 + x(2);
    pc3B = beta*Rchph3 + x(3);

    m0 = 1. / (1. + pc3B);
    m1 = (-m0 * m0 * pc1);
    m2 = (-m0 * m0 * pc2);

    H(cx,1) = m0;
    H(cx,3) = m1;
    H(cy,2) = m0;
    H(cy,3) = m2;

    % /* compute d(transform)/d(omega) */
\[ dToD(1) = ph2*c2 + ph3*c5; \]
\[ dToD(2) = -ph1*c2 + ph3*k0; \]
\[ dToD(3) = -ph1*c5 - ph2*k0; \]
\[ dToD(4) = -ph2*c1 - ph3*k1; \]
\[ dToD(5) = ph1*c1 + ph3*c4; \]  
\[ dToD(6) = ph1*k1 - ph2*c4; \]
\[ dToD(7) = beta*(ph2*k2 - ph3*c0); \]
\[ dToD(8) = beta*(-ph1*k2 - ph3*c3); \]
\[ dToD(9) = beta*(ph1*c0 + ph2*c3); \]
\[
% /* compute dh/d(omega) */
H(cx,4) = m0 * dToD(1) + m1 * dToD(7);
H(cx,5) = m0 * dToD(2) + m1 * dToD(8);
H(cx,6) = m0 * dToD(3) + m1 * dToD(9);
H(cx,4) = m0 * dToD(4) + m2 * dToD(7);
H(cx,5) = m0 * dToD(5) + m2 * dToD(8);
H(cx,6) = m0 * dToD(6) + m2 * dToD(9);
\]
\[
v1 = alpha*(k0*y1 - c5*y2);
v2 = alpha*(c4*y1 + k1*y2);
v3 = Rchph3 + alpha*beta*(-c3*y1 + c0*y2);
\]
\[
H(cx,7) = m0*v1 + m1*v3;
H(cx,7) = m0*v2 + m2*v3;
\]
\[
% SMKF_PX(Hx,p) = 1.;
% SMKF_PY(Hy,p) = 1.;
\]
\[
v1 = beta*y1;
v2 = beta*y2;
v3 = 1.;
Rv1 = (k0*v1 - c5*v2 + c2*v3);
Rv2 = (c4*v1 + k1*v2 - c1*v3);
Rv3 = beta*(-c3*v1 + c0*v2 + k2*v3);
H(cx,7+count) = m0*Rv1 + m1*Rv3;
\]
H(cy,7+count) = m0*Rv2 + m2*Rv3;

end

B.2.5 p_set.m

function p=p_set(y,x,q);

% function y=p_set(y,x,q)

n = length(x) - 7;

epsilon = (x(4)*2 + x(5)*2 + x(6)*2)/4;

d_q = [(1-epsilon)^(1/2); x(4:6)/2];

q_ = quat_mult(q_d_q);

q_ = q_/norm(q_);

R = quat2mat(q_);

p=zeros([n*3 1]);

Ibeta = eye(3);

Ibeta(3,3)=x(7);

beta = x(7);

Roc = inv(Ibeta^R);

for c=1:n

c_bis = 3*(c-1)+1;

c_b_b = 2*(c-1)-1;

u = x(c_b_b);

v = u^b + c;

XYZ = [u v 0]' + a/2; * u^beta + v^beta + 1]';
\% \text{ri} = [y(c_b\_b)*(1+x(7+c)*x(7)) y(c_b\_b+1)*(1+x(7+c)*x(7)) x(7+c)]';
\% XYZ = Roc*([\text{ri} - [x(1) x(2) x(3)]']);

\text{p(c_bis:(c_bis+2))}=XYZ;
end

\section*{B.3 Manual Geometric Modeling}

The manual selection of planes is realized using the next functions.

\subsection*{B.3.1 \texttt{polyg1.m}}

\begin{verbatim}
function dummy = polyg(frame_num,num_planes);
\% function dummy = polyg(frame_num,num_planes);
\% frame_num = frame where the texture map will be used
\% num_planes = number of surfaces to be fitted.

\% OK, is a little silly do to that, later I will ask
\% the number of surfaces AT THE END.

frame_name = 'cubo';
path_name = '\gerardo\thesis\sequences\';
\%frame_num = 1;

\% load m_5.26;
\% load est_5.30; rib =0; PIC\_DAT = P;
filter = '*.mat';
filter = 'est_5.30.mat'

eval(['load ',path_name,frame_name,int2str(frame_num)]);
[fil, path] = uigetfile(filter, 'Load matched points')
eval(['load ',path,fil]);
rib =0; PIC\_DAT = P;
\end{verbatim}
clf
imshow(fi);
pp = size(fi)/2;
factor =length(fi);

hold on;
len_pic = size(PIC_DAT);

temp=round(reshape(PIC_DAT(:,frame_num)−rib,2,length(PIC_DAT(:,1))/2));
h_feat = plot(temp(2,:),temp(1,:),’o’);

colormap(gray(256));

p_x =[];
p_y =[];
s_xy=[];
s_f=[];
feat=[];

for count = 1:num_planes

[poly_x,poly_y,feat_1] =getpoly(h_feat,’r’);
p_x = [p_x;poly_x];
p_y = [p_y;poly_y];
feat = [feat; feat_1’];
s_f = [s_f,length(feat_1)];
s_xy = [s_xy;length(poly_x)];
h_feat = plot(temp(2,:),temp(1,:),’o’);

end;

filter_2 = ’pol_6_1.mat’
[fil,path] = uiputfile(filter_2,’Save polygons’);
eval([’save ’,path,fil,’ p_x p_y s_xy s_f feat num_planes frame_num’]);
B.3.2  polyg1.m

function dummy = polyg(frame_num,num_planes);
% function dummy = polyg(frame_num,num_planes);
% frame_num = frame where the texture map will be used
% num_planes = number of surfaces to be fitted.

% OK, is a little silly do to that, later I will ask
% the number of surfaces AT THE END.

frame_name = 'cubo';
path_name = '\gerardo\thesis\sequences\';
% frame_num = 1;

% load m_5_26;
% load est_5_30; rib =0; PIC_DAT = P;
filter = '.*.mat';
filter = 'est_5_30.mat'

eval(['load ',path_name,frame_name,int2str(frame_num)]);
[fil, path] = uigetfile(filter, 'Load matched points')
eval('load ',path,fil);
rib =0; PIC_DAT = P;

clg
imshhow(fli);
pp = size(fli)/2;
factor =length(fli);

hold on;
en_pic = size(PIC_DAT);

temp=round(reshape(PIC_DAT(:,frame_num) - rib,2,length(PIC_DAT(:,1))/2));
h_feat = plot(temp(2,:),temp(1,:),’o’);

colormap(gray(256));

p_x =[];
p_y =[];
s_xy=[];
s_f=[];
feat=[];

for count = 1:num_planes

[poly_x,poly_y,feat_1] = getpoly(h_feat,’r’);
p_x = [p_x;poly_x];
p_y = [p_y;poly_y];
feat = [feat; feat_1];
s_f = [s_f,length(feat_1)];
s_xy = [s_xy;length(poly_x)];
h_feat = plot(temp(2,:),temp(1,:),’o’);

end;

filter_2 = ’pol_6_1.mat’
[fil,path] = uiputfile(filter_2,’Save polygons’);
eval([’save ’,path,fil,’ p_x p_y s_xy s_f feat num_planes frame_num’]);

---

**B.3.3 pl_fit.m**

```matlab
function [u,c] = pl_fit(x,y,z);

% This function fits a plane to the given points.
% u is a unitary vector, and c is a 3-D point which define the plane.
% Following code to be done into a function: input x,y,z, output u,c.
```
% Compute means.
x_m = mean(x);
y_m = mean(y);

% to constrain the least squares, initially nz = 0.5.
nz = 0.5;

% Compute matrices A and B (see Robot Vision Appendix)
B = [x-x_m;y-y_m;-ones(size(x))*nz];
A = [-nz*z];

M = A*B'*inv(B*B');
nx = M(1);
ny = M(2);
z_m = M(3);

u = [nx ny nz]/norm([nx ny nz]);
c = [x_m y_m z_m];

---

**B.3.4 getpoly.m**

```matlab
function [poly1_x,poly1_y,feat_1] = getpoly(h_feat,col);

% function [poly1_x,poly1_y,feat_1] = getpoly(h_feat,color);
% h_feat - handle of the dots to be selected.
% color - color used to plot the selected dots.
% This function gives back the locations of the polygon
% and the points selected to form the plane.

button = 1;
hold on;

while button == 1;
    [x,y,button]=ginput(1);
    if button == 1;
```
line_x=get(h_feat,'XData');
line_y=get(h_feat,'YData');
dist=(line_x-x).^2 + (line_y-y).^2;
[t,p,ind] = min(dist);
feat_1 = [feat_1 point];
plot(line_x(point),line_y(point),'x','erasemode','background','color',col);
end;
end;

[poly1_x,poly1_y] = getline;

% find closest point on line to mouse click loc (call it fixed_x, fixed_y)
%line_x=get(h_feat,'XData');
%line_y=get(h_feat,'YData');
%dist=(line_x-x).^2 + (line_y-y).^2;

B.4 Model Integration

the next code is used to show model which uses all the data generated with the above programs.

B.4.1 plane6.m

% The function of this script is to get the correct polygons
% to generate a model of the scene.

% first, load parameters
% Change for input.
filter = '*.mat';
filter = 'pr_5_28.mat'
[fil, path] = uigetfile(filter, 'Load Structure Estimation')
eval(['load ',path,fil]);
rib = 0;

filter = '*.mat';
filter = 'pol_6_1.mat'
[fil, path] = uigetfile(filter, 'Polygons')
eval(['load ', path, fil]);

frame_name = 'cubo';
path_name = '\users\gerardo\thesis\sequences\';
eval(['load ', path_name, frame_name, int2str(frame_num)]);
fi = (red(red(fi)));
reduc = 4;
resolution = 20;

clg

% for now, factor = 720.
factor = 720;
pp.x = 720/2;
pp.y = 487/2;

% Generate the 3D points from those parameters:
beta = betas(7, 1);
p = p.set(XY_0, betas(:, 1), q_s(:, 1));
p_xyz = reshape(p, 3, length(p)/3);
temp = p_xyz(1, :);

% correct for xy translation.
p_xyz(1, :) = p_xyz(2, :);
p_xyz(2, :) = temp;
p_xyz(3, :) = p_xyz(3, :);

% "Warp" polygons

for count = 1:nvm_planes
feat1 = get_cont(feat,s_f,count);
poly1_x = get_cont(p_x,s_xy,count);
poly1_y = get_cont(p_y,s_xy,count);

pxyz = p_xyz(:,feat1);
p1_x = (poly1_x-pp_x)/factor;
p1_y = (poly1_y-pp_y)/factor;
[xp1,yp1,zp1] = ren_p(p1_x,p1_y,pxyz,resolution,beta);
rect = rect_p(poly1_x,poly1_y,reduce);

% If the image was reduced two times:
%bw = bin2nan(~roipoly(f1,poly1_x/reduc,poly1_y/reduc))+1;
bw = roipoly(f1,poly1_x/reduc,poly1_y/reduc);

end;

%h2 = warp(xp2,yp2,zp2,flipud(imcrop(256.*bw,rect)),256);
view(0,-90);axis('auto');axis('equal');axis('off');

B.5 Stereo Approach

All the code used to generate the preliminary using the stereo approach is enclosed in this section.

B.5.1 epi_mod.m

This code is used to identify the epipolar lines, so it is possible to generate an stereo pair of rectified pictures.
function dummy = epipolar(flag,t,step);
% function dummy = epipolar(flag,t,step);
%
% flag = 0 if loading of images is unnecessary,
% 1 if loading of images is necessary
% t = reduction rate. (final_size = size/t)
% step = interval between epipolar lines.
% This script is used to find the epipolar lines of
% the sequence.
% modified for use in PC

% Obsolete code. Change to reduce amount of lines

% To get epipolar lines, first it is necessary to load
% the values of the estimated betas and q.s (From motion_est)
%load exp_4_18_pr5 % contains betas and q.s, iterated
  factor = 720; % from exp_4_6 100 times.

% For PC use:
%
% Obsolete: change fot uigetfile
load pic.dat;

if flag == 1;

load \gerardo\thesis\sequences\cubo1
f1 = imresize(fi,1/t,'bilinear');

load \gerardo\thesis\sequences\cubo10
f2 = imresize(fi,1/t,'bilinear');
clear fi;
else
load work_space;
end;

% If the frames used are 1 and 31, it looks kinda nice :)
frame = 10;

% Operations to determine the "poles"
% (the projection of the center of projection of the
% other frames in each frame.
Ibeta = eye(3);
Ibeta(3,3)=betas(7,frame);
beta = betas(7,frame);

% Select R
R = quat2mat(q.s(:,frame));
tr=betas(1:3,frame);
f=1/betas(7,frame);
tr(3)=tr(3)*f;

% XYZ is the location in frame 1 coordinates of the other c.o.p.
% uv_1 is the projection of XYZ (a "pole").
XYZ = tr+R*[0 0 -f];
uv_1 = [XYZ(1) XYZ(2)]'*1/(1+XYZ(3)*beta);

% XYZf is the 3D location in the frame n coordinates of the c.o.p. 1
% uv_f is the pole in frame n coordinates.
XYZf = inv(R)*[0 0 -f]' - tr;
uv_f = [XYZf(1) XYZf(2)]'*1/(1+XYZf(3)*beta);

% In case we use edge detection.
sigma = 3;

% u and c are the coordinates of the principal point
% WE ARE SUPPOSING u and V ARE exactly at the center of the image,
% It can be false, and the vanishing points method should be used to
% locate it.

% t is a factor needed in case we use reduce images.

u = 497/2;
v = 720/2;
u_v_fac = u/v;
[x,z]=edge_det(f2,sigma);
%image(z*256);

max_p = round(u/v/2*100);

size_f1 = size(f1);

% Get the epipolar lines which crosses the principal point (0,0), or (u,v).
    x = (factor*[uv_f(2); 0]+v)/t;
y = (factor*[uv_f(1); 0]+u)/t;
ang_rad = atan((y(1)-y(2))/(x(1)-x(2)));
ang_deg = ang_rad *360/2/pi

if flag == 1
    fig_rot2 = imrotate(f2,ang_deg,'bilinear','crop');
    fig_rot2 = imrotate(f2,ang_deg,'bilinear');
end;

% Modification:
% If we use fig_rot2 WITHOUT cropping, then the principal point
% corresponds to:
    uv_pp = size(fig_rot2);
u = uv_pp(1)*t/2;
v = uv_pp(2)*t/2;
x_int_2 = zeros(uv_pp);
y_int_2 = zeros(uv_pp);

c1g;
subplot(121);
%imshow(f2); ang_rad = 0;
imshow(fig_rot2);
colormap(gray(256));
hold on;
R2 = [cos(ang_rad), sin(ang_rad); -sin(ang_rad) cos(ang_rad)];

% The following lines serve to fill the image with the epipolar
% lines.

% str is another scaling function which will allow me to cover
% the entire image with the epipolar lines.

y1 = [0.5, 1]*inv([[uv_f(2); -0.5],[1,1]])*[uv_f(1);-u_v_fac/2]
y2 = [0.5, 1]*inv([[uv_f(2); -0.5],[1,1]])*[uv_f(1);u_v_fac/2]
y3 = [0.5, 1]*inv([[uv_f(2); 0.5],[1,1]])*[uv_f(1);-u_v_fac/2]
y4 = [0.5, 1]*inv([[uv_f(2); 0.5],[1,1]])*[uv_f(1);u_v_fac/2]

h_min = min([y1 y2 y3 y4]);
h_max = max([y1 y2 y3 y4]);

range_h = (h_max-h_min);

str = 1.15;
if 1==1
    for i = 1:step:uv_pp(1)

% h = -u/v/2 + (i-1)/size_f1(1)*u/v*str;
% h = -u_v_fac/2 + (i-1)/size_f1(1)*u_v_fac*str;
\[ h = \frac{(i-1)}{uv_{pp}(1) \cdot \text{range}_h + h_{\text{min}}} \]

\[ x = [uv_{f}(2); 0.5]; \]
\[ y = [uv_{f}(1); h]; \]
\[ \text{angulo} = \text{atan}((y(1) - y(2))/(x(1) - x(2))); \]

\[ xy = R2*[x'; y']; \]
\[ x = xy(1,:); \]
\[ y = xy(2,:); \]

\[ x = (\text{factor} \cdot x + v)/t; \]
\[ y = (\text{factor} \cdot y + u)/t; \]

\% plot(x,y);

\[ \text{recta} = \text{inv}([x', [1; 1]]) \cdot y'; \]
\[ m = \text{recta}(1); \]
\[ b = \text{recta}(2); \]
\[ x_{\text{int}, 2(i,:)} = 1:uv_{pp}(2); \]
\[ y_{\text{int}, 2(i,:)} = m \cdot (1:uv_{pp}(2)) + b; \]

\% plot(x_{\text{int}, 2(i,:)}, y_{\text{int}, 2(i,:)});
\% pause

end;
end;
axis('auto');

% clear f2 x z;

% [x,z] = edge_det(f1,sigma);
% image(z*256);
XYZ_f = R*[0 0 0]' + tr;
uv = [XYZ_f(1) XYZ_f(2)]'*t/(1+XYZ_f(3)*beta);
x = (factor*[uv_1(2); uv(2)]+v)/t;
y = (factor*[uv_1(1); uv(1)]+u)/t;
ang_rad = atan((y(1)-y(2))/(x(1)-x(2)));
ang_deg = ang_rad *360/2/pi;

if flag == 1
  % fig_rot1 = imrotate(fl,ang_deg,'bilinear','crop');
  fig_rot1 = imrotate(fl,ang_deg,'bilinear');
end;
subplot(122);
%imshow(fl);ang_rad = 0;
imshow(fig_rot1);
colormap(gray(256));
hold on;
R2 = [cos(ang_rad), sin(ang_rad); -sin(ang_rad) cos(ang_rad)];
uv_pp_1 = size(fig_rot1);
u_1 = uv_pp_1(1)*t/2;
v_1 = uv_pp_1(2)*t/2;

x_int_1 = zeros([uv_pp(1) uv_pp_1(2)]);
y_int_1 = zeros([uv_pp(1) uv_pp_1(2)]);

if 1==1
  for i = 1:step:uv_pp(1)
    % h = -u/v/2 + (i-1)/size_fl(1)*u/v*str;
    % h = -u_v_fac/2 + (i-1)/size_fl(1)*u_v_fac*str;
    h = (i-1)/uv_pp(1)*range_h+h_min;
    XYZ_f = R*[h 0.5 0]' + tr;
    % XYZ_f = inv(R)*(i/150 0 0)'-t);
    uv = [XYZ_f(1) XYZ_f(2)]'*/(1+XYZ_f(3)*beta);
    x = [uv_f(2); uv(2)];
\[ y = [uv_f(1); uv(1)]; \]

\[ xy = R2*[x'; y']; \]
\[ x = xy(1,:); \]
\[ y = xy(2,:); \]

\[ x = (factor*x+v_1)/t; \]
\[ y = (factor*y+u_1)/t; \]

\% plot(x,y);

\[ recta = inv([x',[1;1]])*y'; \]
\[ m = recta(1); \]
\[ b = recta(2); \]
\[ x_{int.1}(i,:) = 1:uv.pp.1(2); \]
\[ y_{int.1}(i,:) = m*(1:uv.pp.1(2))+b; \]

\% plot(x_{int.1}(i,:),y_{int.1}(i,:));

axis('auto');
end;
end;

save w_5.5 f1 f2 fig_rot1 fig_rot2 x_int_1 y_int_1 x_int_2 y_int_2

---

**B.5.2 corr.m**

Performs the Normalized correlation technique.

---

% This file contains the functions needed to create the rectified versions
% of two frames. It need the output of the file EPIPOLAR.M
I am fed up with the MArr-Poggio-Grimson Algorithm, so I am going to try the correlation technique.

It uses

% dispar to get the correlation between two images.

frame = 11;
factor = 720;
r1 = 121;
c1 = 180;
sigma = 0.75;
% For that, I still need the rectified versions of the two images.
% right now, for memory sake, an smaller image is used.
% epi_line(1,4,1,frame,'w_5_18');

% Change this line with an uigetfile
load w_5_18

% This line too.
load pic_dat

% load w_5_5

% To change the disparity map, change out2 and out1:
out2 = round(int4_mod(fig_rot2,x_int_2,y_int_2));
out1 = round(int4_mod(fig_rot1,x_int_1,y_int_1));

x=marr_hild(sigma);
out2 = conv2_b(out2,x,'same');
out1 = conv2_b(out1,x,'same');

subplot(121);image(out1);
subplot(122); image(out2);

drawnow
% pause

ff = 11;
dis_fin = 10;
n = 6;
m = 6;
[d12 c12] = dispar(out1, out2, n, m, dis_fin, -1);
[d21 c21] = dispar(out2, out1, n, m, dis_fin, -1);

%[d12 c12] = disp_f(out1, out2, n, m, dis_fin, -1);
%[d21 c21] = disp_f(out2, out1, u, m, dis_fin, -1);

save xyxy d12 d21

subplot(121); image(d12+32);
subplot(122); image(d21+32);
colormap(gray);
drawnow

[r c] = size(d12);
[x, y] = meshgrid( 1:c, 1:r);

% check when c12(t) = c21(t).

dd = interp2(x, y, d21, x - d21, y, 'nearest');
mask_nan = (~abs(dd + d12) <= 1) | abs(dd) == dis_fin;
% mask_nan = bin2nan(~abs(dd + d21) == 0) + 1;
% d12_m = (bin2nan(mask_nan) + 1) * d12;
d12_m = (ff * mask_nan) + (mask_nan + 1) * d12;

%***************************************************************************
% INSERT SURFACE INTERPOLATION: BLOCK PROCESSING.*
% **************************************************************

% Make some space.

```matlab
clear out1 out2 fig_rot_1 fig_rot_2 c12 c21;
```

```matlab
% d12_m = blkproc(d12_m,[20 20],’grid_fcn’);
mode = ’nearest’
```

% Return disparity measure to "unrectified" mapping.

```matlab
y2 = 2*y–y_int_2;
d12_m = interp2(x,y,d12_m,x,y2,mode);
```

```matlab
d12_mr = imrotate(d12_m,–ang_rad_1*360/2/pi,mode,’crop’);
```

% AN APPROXIMATION :: cos(teta) is equivalent to the displacement.

```matlab
% d12_mr = d12_m.*(bin2nan(mask_nr)+1)*cos(ang_rad_1);
```

% Transformation function:

```matlab
R = quat2mat(q_s(:,frame));
tr = betas(1:3,frame);
f = 1/betas(7,frame);
beta = betas(7,frame);
tr(3)=tr(3)*f;
```

% Crop image to original size.

```matlab
% [r1 c1]=size(f1);
rr = round((r–r1)/2);
cc = round((c–c1)/2);
d12_mr = d12_mr(rr+1:rr+r1,cc+1:cc+c1);
```

```matlab
mesh(flipud(d12_mr));view(–30,75);
```

```matlab
[uf,vf]=meshgrid(1:c1,1:r1);
uf = (uf–c1/2)*t/factor;
```
vf = (vf − r1/2)*t/factor;
us = uf+d12_mr*t/factor;

alpha = −((1+beta*tr(3)+beta*R(3,1)*uf+beta*R(3,2)*vf).*us−tr(1)−R(1,1)*uf−R(1,2)*vf);
al_den = ((R(3,1)*uf*beta*beta+R(3,2)*vf*beta*beta+beta*R(3,3)).*us ... -R(1,2)*vf*beta−R(1,3)−R(1,1)*uf*beta);
save XYXY d12_mr d12 d21

B.5.3 dispar.m

function [Disparity, Corr_max] = disparity(out1,out2,N,P,dis_fin,thr)
% function [Disparity, Corr_max] = disparity(out1,out2,N,P,dis_fin,thr)
% out1 — image1
% out2 — image2
% N — horizontal size of the search window
% P — vertical size of the search window (size = 2P+1 x 2N+1)
% dis_fin = seaching limits (pixels)
% thr — threshold (between 0 and 1)
%
% We made a funtion out of corr tec.m. It needs the RECTIFIED versions
% of two images.
% This file contains the functions needed to create the rectified versions
% of two frames. It need the output of the file EPIPOLAR.M
%
% I am fed up with the MArr— Poggio—Grimson Algorithm, so I am goin to try
% the correlation technique.
%
% Refer to Faugeras’ “3D Computer Vision”

% dis_fin = the final amount of disparity used for the process.
%dis_fin = 10;
% so disparity goes from −dis_fin to dis_fin.
% Define the size of the window:
\% N = 4;
\% P = 4;
wn = 2*N+1;
wp = 2*P+1;

out1 = zer_pad(out1,dis_fin,dis_fin,0,0);
out2 = zer_pad(out2,dis_fin,dis_fin,0,0);

av_window = ones([wn, wp]);

% Get the "average" Pictures:
av_out1 = conv2(out1,av_window/wn/wp,'same');
av_out2 = conv2(out2,av_window/wn/wp,'same');

% Get the "variance" pictures:
sig_out1 = conv2((out1 - av_out1).^2,av_window/wn/wp,'same') .^ (0.5);
sig_out2 = conv2((out2 - av_out2).^2,av_window/wn/wp,'same') .^ (0.5);

% First, we are going to get C12(t)
% <<See Faugeras>>
% So the figure 2 is the one which moves.

% To do the "sweeping", we need:
% for bigger images, we have to do everything with
% block processing. DO IT AS SOON AS THIS THING WORKS!
I1_I1ave = out1 - av_out1;
I2_I2ave = out2 - av_out2;

% By now, there are some variables we do not need;
% So for the sake of the memory, let's erase them:
clear av_out1 av_out2 fig_rot_1 fig_rot_2 x_int_1 x_int_2 y_int_1 y_int_2;

Disparity = zeros(size(I1_I1ave));
Corr_max = zeros(size(I1_I1ave));

for i=1:dis_fin;
    I2_I2ave = rota(I2_I2ave,'l',1);
    sig_out2 = rota(sig_out2,'l',1);
end;

% finally, the "sweeping"

for t = -dis_fin:dis_fin;
    K = (wn*wp*sig_out1.*sig_out2);
    Corr = conv2(I1_I1ave.*I2_I2ave,av_window,'same')./K;
    subplot(121);imshow(I2_I2ave - I1_I1ave); % subplot(122);imshow(Disparity);
    drawnow;

    Disparity = (max(Corr_max,abs(Corr)) == Corr).*t + ... 
    (max(Corr_max,abs(Corr)) == Corr).*Disparity;

    d = [d,Corr(147,95)];
    Corr(147,95);
    Corr_max = max(abs(Corr_max),abs(Corr));
    I2_I2ave = rota(I2_I2ave,'r',1);
    sig_out2 = rota(sig_out2,'r',1);
end;

[r c]=size(I2_I2ave);
Disparity = Disparity.*(bin2nan(!(Corr_max>thr))+1);
Disparity = Disparity(:,dis_fin+1:c-dis_fin);
Corr_max = Corr_max(:,dis_fin+1:c-dis_fin);

---

B.5.4 dispar.m

function [Disparity, Corr_max] = disparity(out1,out2,N,P,dis_fin,thr)
% function [Disparity, Corr_max] = disparity(out1,out2,N,P,dis_fin,thr)
% out1 -- image1
% out2 -- image2


% N - horizontal size of the search window
% P - vertical size of the search window (size = 2P+1 x 2N+1)
% dis_fin = seaching limits (pixels)
% thr - threshold (between 0 and 1)
%
% We made a function out of corr_tec.m. It needs the RECTIFIED versions
% of two images.
% This file contains the functions needed to create the rectified versions
% of two frames. It need the output of the file EPIPOLAR.M
%
% I am fed up with the MArr—Poggio—Grimson Algorithm, so I am goin to try
% the correlation technique.
%
% Refer to Faugeras' "3D Computer Vision"

% dis_fin = the final amount of disparity used for the process.
%dis_fin = 10;
% so disparity goes from -dis_fin to dis_fin.

% Define the size of the window:
%N = 4;
%P = 4;
wn = 2*N+1;
wp = 2*P+1;

out1 = zer_pad(out1,dis_fin,dis_fin,0,0);
out2 = zer_pad(out2,dis_fin,dis_fin,0,0);

av_window = ones([wn, wp]);

% Get the "average" Pictures:
av_out1 = conv2(out1,av_window/wn/wp,'same');
av_out2 = conv2(out2,av_window/wn/wp,'same');

% Get the "variance" pictures:
sig_out1 = conv2((out1−av_out1).^2,av_window/wn/wp,'same').^(0.5);
% First, we are going to get C12(t)
% <See Faugeras>
% So the figure 2 is the one which moves.

%To do the "sweeping", we need:
% for bigger images, we have to do everything with
% block processing. DO IT AS SOON AS THIS THING WORKS!
I1_I1ave = out1-av_out1;
I2_I2ave = out2-av_out2;

% By now, there are some variables we do not need;
% So for the sake of the memory, let's erase them:
clear av_out1 av_out2 fig_rot_1 fig_rot_2 x_int_1 x_int_2 y_int_1 y_int_2;

Disparity = zeros(size(I1_I1ave));
Corr_max = zeros(size(I1_I1ave));

for i=1:dis_fin;
    I2_I2ave = rota(I2_I2ave,'l',1);
    sig_out2 = rota(sig_out2,'l',1);
end;
% finally, the "sweeping":
for t = -dis_fin:dis_fin;
    K = (wn*wp*sig_out1.*sig_out2);
    Corr = conv2(I1_I1ave.*I2_I2ave,av_window,'same')./K;
% subplot(121);imshow(I2_I2ave - I1_I1ave);
% subplot(122);imshow(Disparity);
drawnow;
% pause;

Disparity = (max(Corr_max,abs(Corr))==Corr).*t + (~(max(Corr_max,abs(Corr))==Corr)).*Disparity;
d = [d,Corr(147,95)];
Corr(147,95);
Corr_max = max(abs(Corr_max), abs(Corr));
I2_l2ave = rota(I2_l2ave, 'r', 1);
sig_out2 = rota(sig_out2, 'r', 1);
end;
[r c]=size(I2_l2ave);
Disparity = Disparity.*bin2nan(~(Corr_max>thr))+1;
Disparity(:,dis_fin+1:c−dis_fin);
Corr_max = Corr_max(:,dis_fin+1:c−dis_fin);

\section*{B.5.5 orstereo.m}
Uses the Marr-Poggio-Grimson algorithm to do the stereo matching.

function out = or_stereo(in, w, disparity, ver_disparity);
% function out = or_stereo(in, w);
% Using Marr-Poggio-Grimson algorithms, this function returns
% a bitmap where the possibles matches are. See
% article.

w_int = round(w);
temp_l = in;
temp_r = in;
out = in;
for count = 1:w;
    temp_l = rota(temp_l, 'l', 0);
    temp_r = rota(temp_r, 'r', 0);
    out = out|temp_l|temp_r;
end;

    temp_l = out;
    temp_r = out;
    for count = 1:ver_disparity;
        temp_l = rotv(temp_l, 'u', 0);
\begin{verbatim}
temp_r = rotv(temp_r,'d',0);
out = out|temp_l|temp_r;
end;

% It is necessary to use a better mode to do the next step.
% Maybe getting the functio "rota" with rotations greater than 1.

if ~(disparity == 0)
    if disparity > 0; s = 'r'; end;
    if disparity < 0; s = 'l'; end;
    for i= 2:disparity
        out = rota(out,s,0);
    end;
end;
\end{verbatim}

\textbf{B.5.6 edge_zer.m}

function [x,xp,zn,zhp,zhn,zz]=edge_det(figure,sigma);
% function [x,zp,zn,zhp,zhn,zz]=edge_det(figure,sigma)
% This function returns:
% binary figure zz (unscaled) with all zero crossings,
% " zh p or n with horizontal zero crossings,
% " zn " negative " 
% " zp " positive " 
% and a matrix x (scaled and with an offset of 128) as the output
% of a convolution with the marr--hildreth operator.

x=marr_hild(sigma);
% Obtains the Marr--hildreth operator with a predefined sigma.
x=conv2(figure,x,'same');
y=fix(x/max(max(x))*128)+128;
% x = result of the convolution with the Marr–Hildreth operator.

y=sign(y-128);
% The purpose of the following code is to obtain the zero-crossings.
% If a pixel has a positive value (1 on the y matrix), and it is
% surrounded by at least one pixel with negative value (-1 on the y
% matrix) it is considered an edge.

temp = conv2(y,[-1,1],'same');

zn = temp == 2;
zp = temp == -2;

30
temp = conv2(y,[-1,1]', 'same');
zhn = temp == 2;
zhp = temp == -2;
%zh = (abs(temp) == 2);
zz = zhn | zhp | zp | zn;

B.6 Miscellaneous Functions

This section consist of all the functions which are shared by the above programs.

B.6.1 bin2nan.m

function nanmat = bin2nan(bin);
% function nanmat = bin2nan(bin);
% This is a function to get a matrix of NaN given a
% binary matrix. Where bin = 1; nanmat = NaN, otherwise
% nanmat = 0.

110
\[ r \cdot c = \text{size(bin)}; \]
\[ \text{nanmat} = \text{zeros}([r \cdot c]); \]
for \( i = 1 : c \)
  for \( j = 1 : r \)
    if \( \text{bin}(j, i) == 1 \)
      \[ \text{nanmat}(j, i) = \text{NaN}; \]
    end;
  end;
end;

\section*{B.6.2 \ \texttt{gaussian.m}}

\begin{verbatim}
function x = gaussian(sig)
\%
\% function x = gaussian(sig)
\%
\% returns a gaussian with sigma defined by the user

x_min = - fix(4*sig);
\[ x_{\text{max}} = - x_{\text{min}}; \]
\[ n = x_{\text{max}}^2 + 1; \]
\[ \text{row} = x_{\text{min}} : x_{\text{max}}; \]
\[ \text{column} = - \text{row}'; \]
\[ Y = \text{column} \cdot \text{ones}([1 \ n]); \]
\[ X = \text{ones}([1 \ n])' \cdot \text{row}; \]
\[ \% x = (X.2 + Y.2 - 2*\text{sig}^2) * \exp(-(X.2 + Y.2) / (2*\text{sig}^2)) / (\text{sig}^4*2*\text{pi}^2*\text{sig}^2); \]
\[ x = \exp(-(X.2 + Y.2) / (2*\text{sig}^2)); \]
\[ x = x / \text{sum} (\text{sum}(x)); \]
\end{verbatim}
B.6.3  hessian.m

function hes = hessian(Pic);

%function hes = hessian(Pic);
% This function returns the determinant of the hessian at
% each point.

dxx = conv2(Pic, [1, -2, 1], 'same');
dyy = conv2(Pic, [1, -2, 1]', 'same');
dxy = conv2(Pic, [-1 0 1; 0 0 0; 1 0 -1], 'same');

hes = dxx.*dyy - dxx.*dxy;

B.6.4  marr_hil.m

function x=marr_hil(d)
% function x=marr_hil(sigma)
% returns an N x N matrix of the sampling of a laplacian of
% a Gaussian.

laplacian = [1 4 1; 4 -20 4; 1 4 1];
x=gaussian(d);
x=conv2(laplacian,x);

B.6.5  max_loc.m

function [i,j]=max_loc(M);
% function [i,j]=max_loc(M);
% M = matrix;
% [i,j] - location of the maximum element of M
B.6.6  **quat2mat.m**

```matlab
function R = quat2mat(q);

% function R = quat2mat(q);
% Returns the correspondent R rotation matrix
% given q (quaternion)

R = [ (q(1)^2 + q(2)^2 - q(3)^2 - q(4)^2) 2*(q(2)*q(3) - q(1)*q(4)) 2*(q(2)*q(4) + q(1)*q(3)); ... 
     2*(q(2)*q(3) + q(1)*q(4)) (q(1)^2 - q(2)^2 + q(3)^2 - q(4)^2) 2*(q(3)*q(4) - q(1)*q(2)); ... 
     2*(q(2)*q(4) - q(1)*q(3)) 2*(q(3)*q(4) + q(1)*q(2)) (q(1)^2 - q(2)^2 - q(3)^2 + q(4)^2) ];
```

B.6.7  **quat_mult.m**

```matlab
function rq = quat_mult(r,q);

% function rq = quat_mult(r,q);
% quaternion multiplication.

% temp = r;
% r = q;
% q = temp;

R=[r(1), -r(2), -r(3), -r(4) ; ... 
   r(2), r(1), -r(4), r(3) ; ... 
   r(3), r(4), r(1), -r(2) ; ... 
   r(4), -r(3), r(2), r(1)];
```
rq = R*q;

---

### B.6.8 red.m

function G = red(x)
% function G = red(x)
% Given an image x, this function reduces the size of the image to 1/4,
% but first lowpassing the image to avoid aliasing.

[rows columns]=size(x);

% Low pass filter:
w = [1 4 6 4 1]/16;
reg=2;
G = conv2(x,w);
G = conv2(G,w');

% Sampling:
G = G((reg+1):(rows+reg),(reg+1):(columns+reg));
%reg=5;
%G = G((reg):(rows),(reg):(columns));

for i=1:(rows/2)
    G(i,:) = G(i*2-1,:);
end

G = G(1:rows/2,1:columns);

for i=1:(columns/2)
    G(1:rows/2,i) = G(1:(rows/2),i*2-1);
end
G = G(1:rows/2,1:columns/2);
G = round(G);
Bibliography


Acknowledgments

I would like to thank everyone who helped me to finish this thesis, my advisor Andrew Lippman who suggested this interesting problem and gave me lots of freedom to handle it; Ali Azarbayejani for some hints and part of the original code (which correspond to \texttt{dh.m}).

I would also like to thank my friends here at MIT, in México and the runaway brains (including the Totus Camotus club, as well as the infamous G.H. followers). And special thanks to Kate Baty who kindly offer to proofread this thesis and for being that special human being.

To complete the S.M. studies I was sponsored by the Dirección General de Asuntos del Personal Académico de la Universidad Autónoma Nacional de México.