Insitut Eurécom
2229 route des Crêtes, B.P. 193
06904 Sophia-Antipolis
France

Research Report N. RR-02-067

Account of an experimental implementation of a completely automatic camera calibration framework based on the Deriche-Giraudon corner extractor and Zhang's calibration algorithm.

Emmanuel Garcia, Jean-Luc Dugelay

March 2002

E-mail: garciae@eurecom.fr, dugelay@eurecom.fr

1 Introduction

1.1 Context

For many purposes in computer vision (e.g. reconstructing 3-D objects, and in particular realistic faces, as is one of our concerns) there is a need for camera calibration. Here we focus exclusively on the problem of calibrating a single camera (that is, estimating its intrinsic parameters). While the theory of this problem is well-known, it is still not easy to solve in practice. We wanted an algorithm as simple as possible from the practical point of view (i.e. from the point of view of the eventual end-user that would have to do the calibration). Many tools exist on internet that can achieve such calibration, using some more or less sophisticated patterns to be shown to the camera. But we have not found any completely automatic tool (i.e. where the user has no points to click after having acquired calibration images). We do not claim that such tools do not exist but we have not found any. Thus we had to implement one. We used Zhang's implementation of his own calibration algorithm [3, 4] and we implemented what was lacking to have a completely automatic calibration program.

1.2 Zhang's calibration algorithm

Zhang's calibration algorithm is an easy algorithm from the practical point of view since it requires only to take a few pictures of a pattern attached to a plane surface. An implementation of this algorithm is even provided as a microsoft windows executable file which takes as inputs the location of features detected in the calibration images, and which outputs the intrinsic parameters of the camera [4].

However what is lacking here is the preliminary detection of features in the images. In the example presented on [4], those images consist of black squares aligned in rows and columns (figure 1), and the features to be detected consist in the corners of those squares.

In [3] Zhang gives a hint as to how he detected the corners of the squares in those images: he fitted lines to the edges of the squares and computed the corners as the intersection of those lines. Yet no details are given concerning this basic but crucial operation and no fully automatic software is provided.

We chose to use the same pattern as in Zhang's experiments, and thus we had to cope with two tasks:

- Automatically locate the corners in the calibration images
- Refine the location of those corners with a sub-pixel precision

After that, the calibration itself is simply carried out by feeding Zhang's calibration program with the detected corner locations.

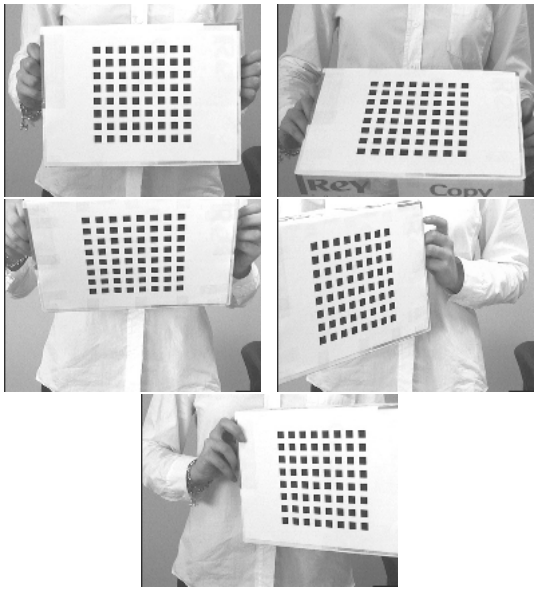

Figure 1: Set of 5 calibration images used: one front view of the calibration pattern, and four other views where the pattern is shifted and rotated in the four directions (up, down, left, right), so as to have an extensive coverage of the camera's field of view.

1.3 Preliminary approximate corner detection

We have separated the problem of detecting corners in two steps because the problem of refining the location of a corner with sub-pixel accuracy is mostly a

mathematical problem that depends little on the specific configuration of corners in the images, whereas the problem of roughly locating those corners is more an algorithmic problem that depends on the particular configuration of corners in the images used (each corner must be identified within this configuration, e.g. must be identified with its row and column indices in our case).

The rough detection of corners could be made manually, or partially manually. But we aimed at a fully automatic extraction of corners, and a significant part of this article is devoted to explaining the algorithms we developped.

1.4 Sub-pixel corner detection

The sub-pixel localization of corners can be done starting from a rough estimating, and using various algorithms. For instance, we could try to fit lines to the edges of a square and compute the corners as the intersections of those lines, as Zhang did, which takes advantage of the particular configuration of corners, or we could refine the location of corners independently.

We have tried both types of algorithms. As for the line fitting, we have fitted a gaussian line profile to an edge-detected image. As for the independent refining of the location of each corner, we have used the theoretical results developped by Deriche and Giraudon in [1].

1.5 Results

The output of those experiments consist in:

- A fully automatic algorithm to roughly detect corner in images featuring a specific pattern of black squares used for camera calibration
- An actual implementation and evaluation of the Deriche-Giraudon theory of corner localization outlining practical details of utmost importance
- A comparison between our implementation of the Deriche-Giraudon algorithm and our implementation of the algorithm used by Zhang (fitting lines to the edges of the squares)

2 Approximate feature localization

The images provided as an example with the implementation of Zhang's calibration algorithm that we used feature a white background on which are 64 identical black squares regularly arranged in 8 rows and 8 columns (figure 1).

The features in those images are the 4 corners of each of the 64 squares. No software to extract those features robustly, accurately and completely automatically was provided, so we implemented such an algorithm in three parts:

- Finding black regions of the image that are candidates for the black squares and representing their boundary as a chain
- Rejection of all black regions that are not images of the squares of interest and localization of the four corners of the remaining black regions
- Ordering of the squares' images (within the 8×8 grid structure) and subsequent ordering of the corners

2.1 Location and representation of black regions

In order to locate the candidates for the images of black squares, we find the black connex regions of the image and represent them as chains of pixels according to the following operations.

2.1.1 Binarization

In order to locate black squares on the white background, we first do a simple binarization of the image using the mean of the minimum and maximum grey level values present in the image as a threshold (figure 2). Depending on the level of noise present in the image it might be useful to do a low-pass filtering before that. In our case it was not necessary.

2.1.2 Labeling connex components

The next step consists in labeling the connex components of the binary image, assuming that the black

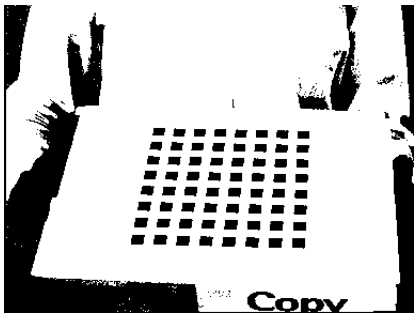

Figure 2: Binarized image.

squares are some of those components. This is done assuming a 4-connexity of black pixels. The aim of the following operations is to decide whether a given connex component is the image of a black square. We first discard non-convex then, non-quadrilaterals, non-parallelograms, and then possible parallelograms that are not the image of an observed square of the calibration pattern.

2.1.3 Computing boundary

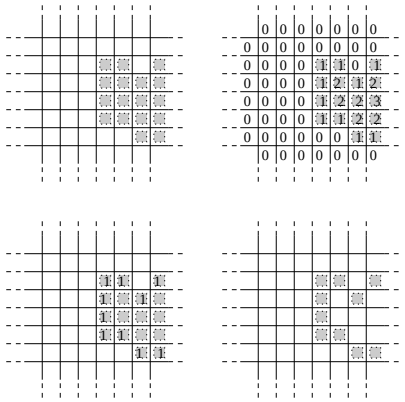

Figure 3: Distance map and boundary.

In order to manipulate the connex components of the binary image more easily, we represent them as a more abstract data than a set of pixels. At first we compute the boundary of a given connex component as a set of pixels, which defines it completely. To do this we compute a distance map of pixels with respect to the outside of the current connex component and we keep the pixels that are at distance 1.

2.1.4 Expressing boundary as a chain

An easier representation of the boundary (for our purpose) is as a chain of pixels. We chose to start with the left-most pixel on the top of the boundary of the current connex component and start running the boundary to the right. This operation discards the knowledge of possible holes inside the connex component. So we only keep the outer boundary.

1. `set the initial pixel P to the left-most pixel on the top of the boundary`
2. `set the initial direction D to be E`
3. `do until all boundary pixels are marked`
   (a) `mark the current pixel P`
   (b) `find the first unmarked pixel P' around P begining with the pixel in direction -D and counting clock-wise`
   (c) `set the current direction D to the direction from P to P'`
   (d) `set the current pixel P to P'`

2.2 Keeping only images of the black squares of interest

Once the black regions that could be images of the black squares of the calibration pattern are identified, we proceed by successively discarding regions that do not satisfy an increasing number of conditions until only images of squares remain. For this we make use of the fact that those images of squares should roughly be parallelograms (assuming that in the neighbourhood of a square, the perspective projection of the camera is close to a parallel projection). Since they should be parallelograms, they should be quadrilaterals, and before that they should be convex. Thus we first discard non-convex black

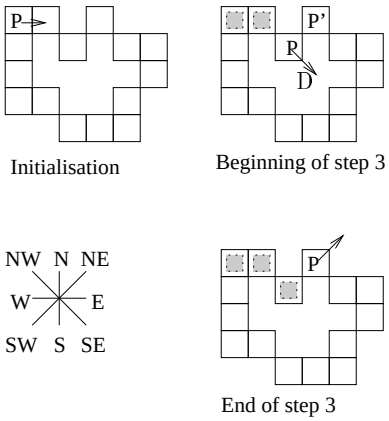

Figure 4: Representing a contour as a chain.

regions, then non-quadrilateral ones, and then non-parallelograms (figure 5).

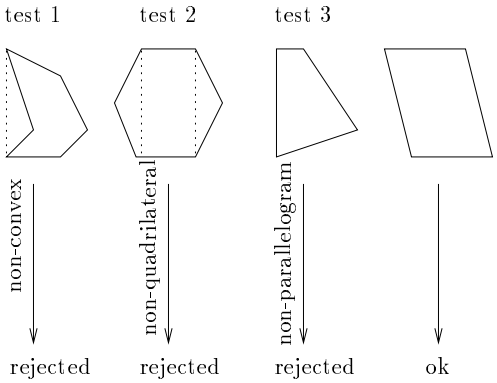

Figure 5: Step-by-step rejection of non-parallelogram shapes.

2.2.1 Discarding non-convex regions

We compute the convex hull of the boundary of a connex component by discarding the pixels that are inside it. This can simply be done by considering the pixels immediately before and after a given pixel in the chain representing the boundary assuming we know the orientation of chain (which we know). When removing a pixel, we examine whether it was far from the boundary of the convex hull or not. If it was we discard the current connex component. If it is very close (we have chosen 4 pixels at most) we don't discard the connex component assuming it is almost convex and the apparent non-convexity may be due to noise or the discrete nature of the image.

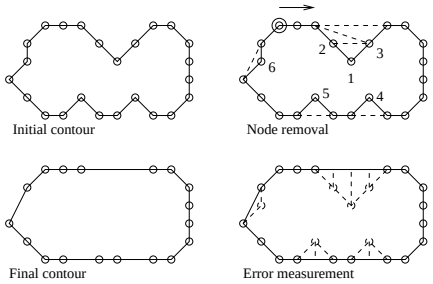

Figure 6: Computing convex hull.

2.2.2 Discarding non-quadrilaterals

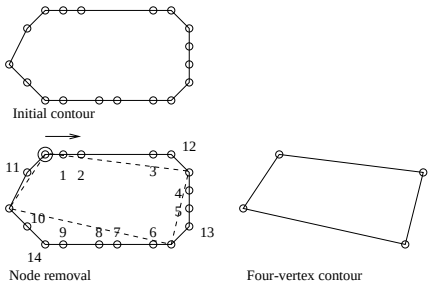

Figure 7: Keeping 4 vertices.

After discarding non-convex chains, we discard those that are not quadrilaterals, i.e. those that do not contain exactly four points. Actually, some chains that contain more than four points can be seen as quadrilaterals if, for example, one extra point is almost on the line joining two other points. So, to decide whether a chain represents a quadrilateral or not we first delete all those points that may be on a line joining their two neighbours. We have decided that one point is on a line when it is at most 4 pixels away from it. Note that this automatically discards all connex components that are 4 or less than 4 pixels

wide which is a good thing since there might be small components due to noise for example.

2.2.3 Discarding non-parallelograms

The images of the observed squares should roughly be quadrilateral assuming that the perspective projection of the camera can locally (i.e. around a given square) be seen as a parallel projection. Therefore we discard every quadrilateral in the image that is not a parallelogram.

To do this we have used a particular criterion for parallelism. We have computed a 2D homography associated with a quadrilateral. Such an homography can be expressed as a 3×3 matrix H operating on 2D homogeneous coordinates.

In our case we have computed an homography H that transforms a system of homogeneous coordinates into another. The target system of coordinates if the base system of coordinates of the image plane where a pixel (x, y) has homogeneous coordinates $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$. The source system of coordinates is a system of coordinates where the corners of the quadrilateral have coordinates $\begin{bmatrix} \pm 1 & \pm 1 & 1 \end{bmatrix}^T$.

Since the homography H is defined up to a scale factor, it has 8 degres of freedom, which can be computed linearly using the correspondance between the 4 pairs of coordinates, each providing 2 contraints.

Our criterion for parallelism is $C = \frac{\sqrt{H_{13}^2 + H_{23}^2}}{|H_{33}|}$. When it is zero, the quadrilateral is a parallelogram, and the higher its value, the more distorted the quadrilateral is. We have empirically set the threshold for parallelism to 0.15.

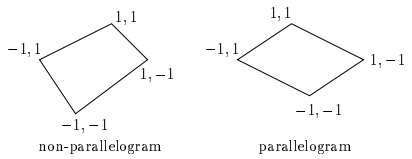

Figure 8: Representing a quadrilateral by an homography.

2.2.4 Removing outliers

After the previous steps, only connex components of the binary image that are quadrilaterals or almost quadrilaterals remain. Still, some quadrilaterals might not be images of the black squares of interest. So we have designed another step to discard another type of potential outliers.

For this we consider the 4 neighbouring squares of each square and decide whether those neighbours are in a direction and at a distance consistent with the average configuration of a square.

It is explained in 2.3.3 how the 4 neighbours of a square are found.

2.3 Ordering of the 8×8 squares

2.3.1 Computing orientation

In order to characterize the location and orientation of a quadrilateral we compute a system of coordinates attached to it using its associated homography H (see above for definition of H).

By definition H applied to homogeneous coordinates $\begin{bmatrix} \pm 1 & \pm 1 & 1 \end{bmatrix}^T$ yields the homogeous coordinates of the 4 corners of the quadrilateral associated to H in the image plane.

Now we will set the center of the quadrilateral to have coordinates $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ in the source system coordinates of H. Applying H to those coordinates yields to image coordinates of the center of the quadrilateral.

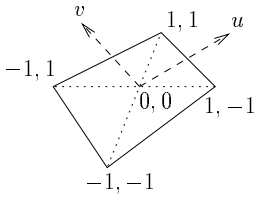

Figure 9: Local system of axes of a quadrilateral.

The axes of the system of coordinates associated to the quadrilateral have been defined as $\frac{\partial x}{\partial u_1}(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T)$ and $\frac{\partial x}{\partial u_2}(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T)$ where $x = \begin{bmatrix} x_1 & x_2 & 1 \end{bmatrix}^T = H.\begin{bmatrix} u_1 & u_2 & 1 \end{bmatrix}^T$

This can easily be explicited in terms of the coefficients of H.

2.3.2 Aligning squares

Once we have computed a local system of coordinates for each quadrilateral we make the first axes of all the quadrilaterals point in the same direction, and the same for their second axes, which may require to swap axes or to reverse one or two axes for some quadrilaterals.

To decide whether we swap or not the axes of some quadrilaterals, we do a two-level quantization of their direction using a Lloyd-Max algorithm.

To completely define the algorithm let us define the quantization space, the nature of the centroids and the metric used to cluster the variables in the quantization space.

The parameter space for one quadrilateral is the set $\{0, 1\}$, 0 meaning for example that the axes of the quadrilateral are not to be swapped, and 1 that they are to be swapped. Associated to this parameter is the pair formed by the directions of the two axes of the quadrilateral, swapped or not depending on the value of the parameter. The directions are expressed as angles defined modulo π.

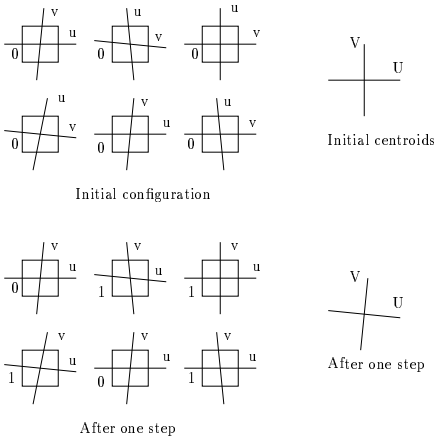

Figure 10: Two-level quantization of the quadrilaterals in order to align their first and second axes consistently.

The centroids are angles, each angle being defined modulo π. There are 2 centroids, one corresponding to the mean direction of the first axes of the quadrilaterals (after possible swapping of their two axes), and the other corresponding to the mean direction of the second axes of the qudrilaterals.

Finally, the distance associated to the choice of a value, 0 or 1, for the parameter of a quadrilateral, and given a 2 centroids, is the sum of the distance between the first axis of the quadrilateral (considering possible swapping of the 2 axis depending on the value of the parameter) and the first centroid, and the distance between the other axis of the quadrilateral and the second centroid.

We iterate the Lloyd-Max algorithm, by clustering the quadrilaterals with some initial choice for the centroids, and then compute the new centroids until the clustering is fixed or at least 64 quadrilaterals don't change cluster (to avoid possible non-convergence in some cases, due to outliers). This requires very few iterations, typically 1 or 2. We must point out that computing the mean, modulo π, of a set of angles defined modulo π must be done carefully, since it is a bit tricky. We have used the fact that all the angles involved in the computation of a given centroid can be expressed as numbers in a same interval of length $\frac{\pi}{2}$.

To end with the description of our procedure for aligning the quadrilaterals in a consistent manner we should add that at the end with have swapped the 2 centroids, when needed, to ensure that the first one was associated to the most horizontal direction.

We have also given an orientation to the directions associated with the centroids so that the first one pointed to the right and the second one, up.

And finally we have also reversed the orientation of the axes of the quadrilaterals, when needed, so that their axes point in the same direction than their associated oriented centroids.

2.3.3 Looking for neighbours

Now that we have computed a local system of axes for each squares and that all those systems have axes pointing in almost the same direction we can consistently compute the neighbouring squares of each square in the 4 directions defined by the axes of their

local system of coordinates.

For example, to find the neighbour of one square, having a local system of coordinates (O, u, v), along the positive direction of its first axis u, we look for the square whose center is at minimum distance from O under some constraint. If (x, y) are the coordinates of another square's center in the system of coordinates (O, u, v), we set the distance of this center to be $d = x$ and impose the constraints $d > 0$ and $|y| < |x|$. We use similar definitions to find the neighbours of a square in all 4 directions.

In case we still have more than 64 squares, meaning there are still outliers, we try to discard them here. For this we compute the mean coordinates of a neighbour's center with respect to a square's local system of coordinates. And discard the squares whose neighbours are at coordinates farthest from the average.

2.3.4 Ordering squares

Now that we have identified, characterized and aligned all 64 squares, computing also their neighbours in all 4 directions, it is easy to order them by row and column. The corners of each squares are also roughly located and can be globally ordered in 16 rows and 16 columns. We assume that the first row is to the top and the first column to the left in the image. To decide what is top and left in a possibly slighly rotated pattern, we use the estimation of the average directions of the axes of the squares, computed earlier. Left is in the negative direction of the first axis and top in the positive direction of the second axis.

In order to ensure a consistent numbering of corners between several pictures of the same pattern, it is necessary that the pattern be not rotated too much along the line of view between the various pictures.

3 Sub-pixel feature localization

3.1 The Deriche-Giraudon algorithm

In order to refine the location of the corners found in the previous set of operations with a low precision (about one pixel or worse) we have implemented the theory developped by Deriche-Giraudon which is proven to give optimal theoretical results under some assumptions.

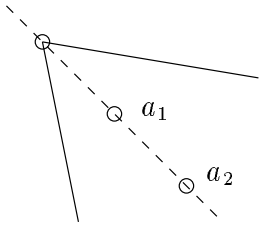

Figure 11: The Deriche-Giraudon corner detection algorithm.

We will only give a short presentation of this algorithm. For details and justifications one may read [1].

The first step consists in computing the determinant of the Hessian of the gaussian-filtered image, for two different values, σ_1 and σ_2, of the width of the gaussian filter.

This is expressed as $DET = I_{xx} I_{yy} - I_{xy}^2$ where I_{xx}, I_{yy} and I_{xy} represent the second-order derivatives of the gaussian filtered image.

Then, a local maximum of DET is looked for around a given roughly detected corner, for the two values of σ. This yields two different points, a_1 and a_2 (see figure 11), that both approximate the exact location of the corner.

It is shown in [1] that the exact location of the corner is along the line joining the two previous points and that in this point the Laplacian of the image is zero.

So, after locating the two local maxima of DET for the two possible values of σ, we find the point on the line joining those two points where the Laplacian of the image crosses zero.

This is straightforward implement once it is decided how to compute and interpolate the needed differential properties of the image. In fact, to compute either of the 2 criteria DET, we need to compute the derivatives of images. To locate a local maximum of such a criterion with sub-pixel accuracy we need to be able to interpolate it. Same thing for the computation of the Laplacian and the finding of its zero-crossing with sub-pixel accuracy.

The accuracy that we can hope for when practically using the Deriche-Giraudon algorithm depends greatly on the methods used for computing derivatives and for interpolating images. No hint about what methods are more suitable in practice is given in [1]. So we hereafter present the methods we implemented.

3.1.1 Gaussian-filtering of an image

Filtering an image $I(k,l)$ with a Gaussian filter $G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{\sigma^2}}$ simply consists in convolving I with G, which can be done in the frequency domain by multiplication. The resulting image is

$$I_\sigma(k,l) = \sum_{p,q} I(p,q)G(k-p,l-q)$$

.

3.1.2 Interpolating an image

Our choice of an image interpolation scheme was largely based on the comprehensive presentation of image interpolation provided by [2]. We chose to use a cubic B-spline interpolation of images with a support of 7 pixels in each dimension of the image. We used the code provided in [2] to efficiently compute interpolation coefficients.

3.1.3 Computing the spatial derivatives of an image

Given an image $I(k,l)$ and an interpolating function ϕ such that the interpolated value of I at coordinates (x,y) is

$$I(x,y) = \sum_{k,l} I(k,l)\phi(x-k,y-l)$$

one can compute the first derivative of I with respect to x at any point as

$$\frac{\partial I}{\partial x}(x,y) = \sum I(k,l)\frac{\partial \phi}{\partial x}(x-k,y-l)$$

(and similarly for other or higher order deriatives).

Thus, assuming the previous interpolation scheme (cubic B-spline interpolation) one can deduce a consistent way of computing derivatives of an image.

However we did not compute derivatives in this way, because we had to compute the derivatives of Gaussian-filtered images. In this case we can simply compute the derivatives of the smoothed image by convolving the initial image with the derivatives of the Gaussian filter (which can be computed analytically). So the derivatives of the smoothed image with respect to x would be

$$\frac{\partial I_\sigma}{\partial x}(k,l) = \sum_{p,q} I(p,q)\frac{\partial G}{\partial x}(k-p,l-q)$$

which again can be computed efficiently in the frequency domain.

3.1.4 Finding a local maximum in an image

Finding a local maximum in an image is easy, but finding it with sub-pixel accuracy is not a well-defined problem since the location of such a local maximum depends mainly on the interpolation scheme chosen for the image and on the optimization procedure.

For interpolation we have chosen the aforementionned cubic B-spline interpolation scheme. For the optimization procedure we thought about a few algorithms (e.g. gradient descent, recursive searching in a 3×3 neighbourhood of decreasing scale). We finally used the following iterative algorithm:

- From an estimate (x,y) of the location of the maximum (with non-integer coordinates) compute (by interpolation) the values of the image at coordinates $(x+i, y+j)$ where i and j are in $\{-1,0,1\}$
- Compute the 2-dimensional quadratic function of (i,j) that best approximates the previous 9 interpolated values of the image
- Analytically find the local maximum of this quadratic function and take it as the estimate of the location of the maximum for the next step

The stopping criterion is a threshold on the difference between the estimate of the maximum at the

beginning and at the end of a step (10^{-6} for instance). In a few cases, there would be an oscillation between two values. To prevent this we multiply the amplitude of the correction to be made, by a factor initially equal to one and that decreases exponentially.

In cases where the quadratic function has no analytical maximum but a minimum, we take the correction to be the opposite of the correction that would have to be applied to move towards the minimum.

In cases where the correction to be made has either coordinate greater than 1 (or lower than -1), we reduce it to so that the absolute value of its coordinates be at most 1.

Finally, the computing of the best-fitting quadratic function assumes a mean squared error criterion, and the data are not all given the same weight (1 for the central point, 0.7 for the edge points, and 0.5 for the corner points).

3.1.5 Finding the zero-crossing of the Laplacian

If a_1 is the estimate of the corner's location obtained for the smallest value of σ, say σ_1, and a_2 is the estimate obtained for the largest value $sigma_2$, the best estimate of the corner's location lies on the semi-line $[a_2, a_1)$ and where the Laplacian of the image is zero.

To find this point we have computed the value of the Laplacian at point a_2, then found a point on the semi-line $[a_2, a_1)$ where the sign of the Laplacian is opposite to its sign at a_2, and found the zero-crossing of the Laplacian by dichotomy.

Here we had a large choice of algorithms to compute the Laplacian. And no doubt this choice has an impact on the final results. For instance when we need to value of the Laplacian in a specific location we can directly compute the interpolated second-order derivatives from the initial image, as explained in 3.1.3. But we can also compute the Laplacian at integer coordinates and then interpolate it using our interpolation scheme. What we did is the latter, but we did not compute the Laplacian from the image, but from a Gaussian-filtered image (in order to reduce the influence on the second-order derivatives of the noise present in the image).

3.2 Alternative algorithm

Zhang took advantage of the particular configuration of corners in order to locate then with sub-pixel accuracy: he fitted lines to the edges of the black squares and set the corners to be the intersection of those lines. This accuracy of this method is as good as the accuracy with which the lines are fitted to the edges of the squares.

Although no technical details are given in [3] regarding the line-fitting process, we have implemented it in this way:

- For each edge of each square, consider its two extremities x_1 and x_2 (initially taken as the rough estimate for the location of corners)
- Compute the norm of the gradient of the Gaussian-filtered image in the direction orthogonal to the edge $[x_1, x_2]$
- Move x_1 and x_2 so as to maximize the correlation between the gradient image and a Gaussian line profile
- Compute the line going through the final x_1 and x_2 for each edge, and set the final detected corners to be the intersections of those lines

Given an edge $[x_1, x_2]$ and a gradient norm image E, the correlation between this image and a Gaussian line profile of width σ attached to the edge $[x_1, x_2]$ is computed as

$$C = \sum_x E(x) e^{-\frac{d(x,(x_1,x_2))^2}{\sigma^2}}$$

where x has integer coordinates and is limited to be in the neighbourhood of the segment $[x_1, x_2]$ (e.g. at most 4 or 5 pixels away from it) and where $d(x, (x_1, x_2))$ represents the distance between x and the line (x_1, x_2).

It is easy to analytically differentiate C with respect to x_1 and x_2 and to maximize it using an optimal step gradient descent algorithm (at each step we move both x_1 and x_2 along a direction orthogonal to $[x_1, x_2]$ and we recompute the gradient norm according to the new orientation of $[x_1, x_2]$, even if it has only slightly changed).

4 Results

The accuracy of the corner detection cannot be evaluated in itself except in the case of synthetic images where the exact location of the corners is known and can be compared with the estimated location of the corners. Such images can be built but the validity of the results they provide depends on the validity of the algorithm with which they have been synthesized. In particular we should note that while such images are mostly black and white, pixels near the edges of the black squares should be represented with a gray level computed according to some algorithm which may not exactly reflect the way images are acquired by a real camera.

We did not evaluate the accuracy of the corner detection in itself, but we did evaluate the reprojection error after calibration with Zhang's calibration algorithm.

4.1 Calibration algorithm used

After the extraction of corners in a set of 5 images, we feed Zhang's calibration algorithm with this data and get the estimated calibration parameters of the camera with which the images have been acquired, along with the position in 3-D space of the calibration object for each image.

Knowing the projection parameters of the camera and the location of the calibration object in 3-D space (and the location of the corners of the squares drawn on it) it is possible to project in the image the corners whose location are known in 3-D space and to compare them with the corners estimated from the image. This yields the so-called reprojection error.

Before proceeding with the numerical results, let us mention that we could have used another pattern (than black squares) or another algorithm similar to Zhang's but that would also take into account the possible tangential distortion of the camera, if needed.

4.2 Numerical results

For the sake of comparison we have tested our two corner extraction implementations on a set of 5 images actually used by Zhang in [3, 4]. Thus, table 1 shows the reprojection error when using corners detected with 3 different algorithms on the images of [3, 4]. In experiment 1, we simply used the corners detected by Zhang, available on [4]. In experiment 2 we used our own implementation of a line-fitting algorithm on Zhang's images. In experiment 3 we used our implementation of the Deriche-Giraudon algorithm.

We have defined the reprojection error E as in [3], that is as the square root of the mean squared error between detected (x_i) and reprojected (x'_i) corners:

$$E = \sqrt{\frac{1}{n}\sum_i (x_i - x'_i)^2} = \sqrt{\bar{x}^2 + \sigma^2}$$

where $\bar{x}$ and σ are the mean and standard deviation of $x_i - x'_i$ respectively.

The parameters for the line-fitting algorithm are:

- σ_1: width of the Gaussian line profile to be fitted
- σ_2: width of the Gaussian filter applied to the image "before" computing the gradient image

The parameters for the Deriche-Giraudon algorithm are:

- σ_1: width of the Gaussian filter applied to the image before computing the first DET criterion (see 3.1)
- σ_2: width of the Gaussian filter applied to the image before computing the second DET criterion
- σ_3: width of the Gaussian filter applied to the image before computing its Laplacian

Tables 2 and 3 respectively show results obtained on synthetic images and on the real images of figure 1.

5 Conclusion

We have applied the Deriche-Giraudon [1] corner detector to the specific problem of camera calibration,

	Exp. 1	Exp. 2	Exp. 3
Image Set	Zhang	Zhang	Zhang
Corner Detection	Zhang	Line-Fitting $\sigma_1 = 1.0$ $\sigma_2 = 2.0$	Deriche-Giraudon $\sigma_1 = 1.3$ $\sigma_2 = 3.9$ $\sigma_3 = 3.6$
Image 1	0.35	0.33	0.23
Image 2	0.23	0.25	0.28
Image 3	0.54	0.52	0.37
Image 4	0.24	0.26	0.22
Image 5	0.21	0.23	0.31

Table 1: Square root of the mean squared reprojection error for Zhang's images. Our implementation of the line-fitting algorithm gave results comparable to those obtained by Zhang. Our implementation of the Deriche-Giraudon algorithm did not give significantly better results and required intensive fine-tuning.

	Exp. 4	Exp. 5
Image Set	Synthetic	Synthetic
Corner Detection	Line-Fitting $\sigma_1 = 0.9$ $\sigma_2 = 0.9$	Deriche-Giraudon $\sigma_1 = 0.9$ $\sigma_2 = 3.7$ $\sigma_3 = 3.7$
Image 1	0.083	0.100
Image 2	0.082	0.098
Image 3	0.083	0.099
Image 4	0.082	0.098
Image 5	0.007	0.023

Table 2: Square root of the mean squared reprojection error for our synthetic images. The results are better than for the previous real images since the noise in synthetic images is only due to sampling and anti-aliasing algorithms and there were no lens distortion in the virtual camera.

and to the even more specific problem of camera calibration using images of a plane object on which is drawn a pattern consisting of black squares as described in [3, 4].

While implementing the corner detector we have noticed that the choice of the interpolation algorithm was very important and [1] gives no hint about it. In the light of [2] which provides a thorough study of interpolation in the context of images, we have chosen to use a cubic B-spline interpolation.

We compared our implementation of the Deriche-Giraudon algorithm with an implementation of a line-fitting algorithm that takes advantage of the particular configuration of the corners to detect in order to detect them with sub-pixel accuracy.

The results of both methods, in terms of reprojection error on real images, are comparable on low quality images. On images of higher quality, the line-fitting method is better, while the results of our implementation of the Deriche-Giraudon algorithm remain almost the same. This might indicate that the Deriche-Giraudon algorithm is limited by the choice that have to be made in its practical implementation (choice of image interpolation scheme, choice of derivatives computation scheme). In that case work should be directed towards experimenting what implementation choices give the best results. Those optimal choice certainly depend partly on the image acquisition device.

Finally, we have presented a fully automatic

	Exp. 6	Exp. 7
Image Set	Real	Real
Corner Detection	Line-Fitting $\sigma_1 = 0.9$ $\sigma_2 = 0.9$	Deriche-Giraudon $\sigma_1 = 1.0$ $\sigma_2 = 3.0$ $\sigma_3 = 2.0$
Image 1	0.09	0.15
Image 2	0.17	0.34
Image 3	0.11	0.31
Image 4	0.23	0.33
Image 5	0.17	0.20

Table 3: Square root of the mean squared reprojection error for our real images. The results are much better than with Zhang's images in the case of line-fitting and are similar in the case of the Deriche-Giraudon algorithm. In fact those real images are of much better quality than those used by Zhang.

scheme for roughly locating the corners in the specific pattern used for camera calibration. This is necessary before using any sub-pixel localization refinement algorithm, and this is not provided in [4].

References

[1] R. Deriche, G. Giraudon, *A Computational Approach for Corner and Vertex Detection*, IJCV, 10-2, pp. 101-124, 1993.

[2] P. Thévenaz, T. Blu, M. Unser, *Image Interpolation and Resampling*, Swiss Federal Institute of Technology, Lausanne, `http://bigwww.epfl.ch/publications/thevenaz9901.html`

[3] Z. Zhang, *A flexible new technique for camera calibration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330-1334, 2000.

[4] A Flexible New Technique for Camera Calibration, `http://research.microsoft.com/~zhang/calib/`