Machine Vision and Applications (2010) 21:949–957
DOI 10.1007/s00138-009-0202-2

SHORT PAPER

A topological approach to finding grids in calibration patterns

Chang Shu · Alan Brunton · Mark Fiala

Received: 15 March 2006 / Revised: 13 February 2009 / Accepted: 13 May 2009 / Published online: 27 October 2009

Abstract This paper describes a technique for finding regular grids in the images of calibration patterns, a crucial step in calibrating cameras. Corner features located by a corner detector are connected using Delaunay triangulation. Pairs of neighboring triangles are combined into quadrilaterals, which are then topologically filtered and ordered. Both triangular and quadrilateral elements are represented by a single mesh data structure, which allows us to exploit the strong topological constraints in a regular grid. The complexity of the algorithm is linear to the number of corners. Experiments show that the method is able to handle images with severe radial distortions and partial pattern occlusions. Implemented on a conventional desktop, grid matching can be done in real time. The method is also applicable to marker detections for augmented reality and robot navigation.

Keywords Pattern matching · Grid finding · Delaunay triangulation · Computational geometry

1 Introduction

The process of calibrating a camera involves determining the intrinsic parameters, such as focal length, and extrinsic parameters, such as the position and orientation of the camera with respect to a coordinate frame. This process often relies on the use of calibration patterns with known geometry. Although it is possible to calibrate cameras using only natural scenes, a process known as *autocalibration*, when accuracy and reliability are demanded, one has to resort to the use of calibration patterns.

The first stage in calibration is to extract features from the images of patterns and match them with positions in the patterns. Once the correspondences between points in the images and points in the pattern are established, the second stage is to solve for the intrinsic and extrinsic parameters by certain numerical procedures [1]. Published work has been concentrated on the second stage, for example [2–4]. Public domain code is also available [5,6]. In contrast, there has been little study on the first stage. There, manual or user assisted methods are often used to solve the correspondence problem. This seemingly straightforward problem affects the overall performance of a calibration system [7]. In many applications, for example, in multiple camera systems and augmented reality systems, there is an increasing need for an automatic pattern detector that works rapidly and is robust to various conditions.

The most commonly used calibration pattern is the checkerboard pattern. Its alternating black and white squares make strong corner features. In this paper, we use a variant of the checkerboard pattern, as shown in Fig. 1. The three circles are used to define a coordinate system on the checkerboard.

One approach to processing the checkerboard pattern is to find edges and fit lines to them. Corners are found by intersecting lines. The drawback of this approach is that edges are in general curved due to radial distortions. Furthermore, the subsequent ordering of the corners into a regular grid can be complex and unreliable. Soh et al. [8] used a regular grid pattern. They used attributed relational graph matching to arrange the centers of the black squares into a regular grid, though little detail was given in the paper. The OpenCV function, *cvFindChessBoardCornerGuesses*, available as an open source code, uses a checkerboard pattern without the orientation marker. However, it requires the whole checkerboard

C. Shu (✉) · A. Brunton · M. Fiala
Institute for Information Technology,
National Research Council Canada,
Montreal Road, Building M-50,
Ottawa, ON K1A 0R6, Canada
e-mail: chang.shu@nrc-cnrc.gc.ca

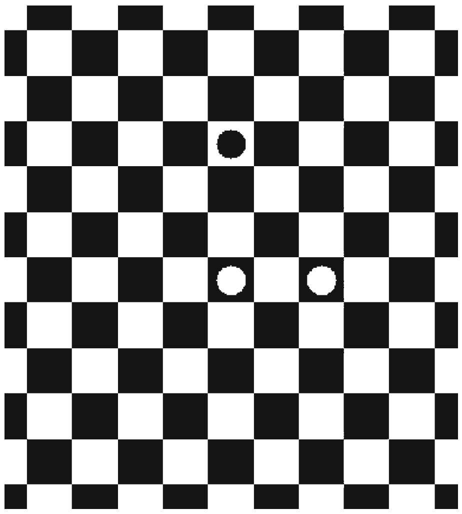


Fig. 1 Checkerboard calibration pattern

to be present in the images and all the squares to be successfully detected. This is a severe limitation in many real world situations, for examples, under poor lighting conditions or with highly distorted lens.

In this paper, we propose a method that exploits the topological structure of the checkerboard pattern. The main idea is to use Delaunay triangulation to connect the corner points. Neighboring pairs of triangles with similar colors are merged into quadrilaterals that match the squares in the pattern. Topological information embedded in the images of the pattern is captured in a mesh data structure which facilitates efficient manipulation and traversal. Figure 2 illustrated the main steps of the method.

The idea of matching features using both appearance and topological information was proposed by Tell and Carlsson [9] in the context of wide baseline matching. To match feature points in different images, the intensity profiles around the point in question are computed. Feature points are then matched based on their intensity profiles and using the fact that the cyclical order of the profiles should be preserved on a planar surface. However, this method is not applicable to our problem because the intensity profiles around each corner point on the checkerboard are identical. The method works best for natural scenes in which feature points tend to have distinguishable intensity profiles.

Our objective is to provide a calibration pattern matching method that is both robust and efficient. It should work reliably under different lighting conditions, occlusions, and quality of lens. It should also be fast enough so that it can be used for real-time applications such as augmented reality and robot navigation.

The rest of this paper is organized as follows. Section 2 discusses connecting feature points using Delaunay triangulation and its associated data structures. Section 3 discusses combining triangles into quadrilaterals. In Sect. 4 we introduce a technique that filters out illegal quadrilaterals. In Sect. 5 we provide details of walking through the quadrilateral mesh to match them with the squares in the pattern. Experimental results are presented in Sect. 6. Finally, conclusions and discussions are presented in Sect. 7.

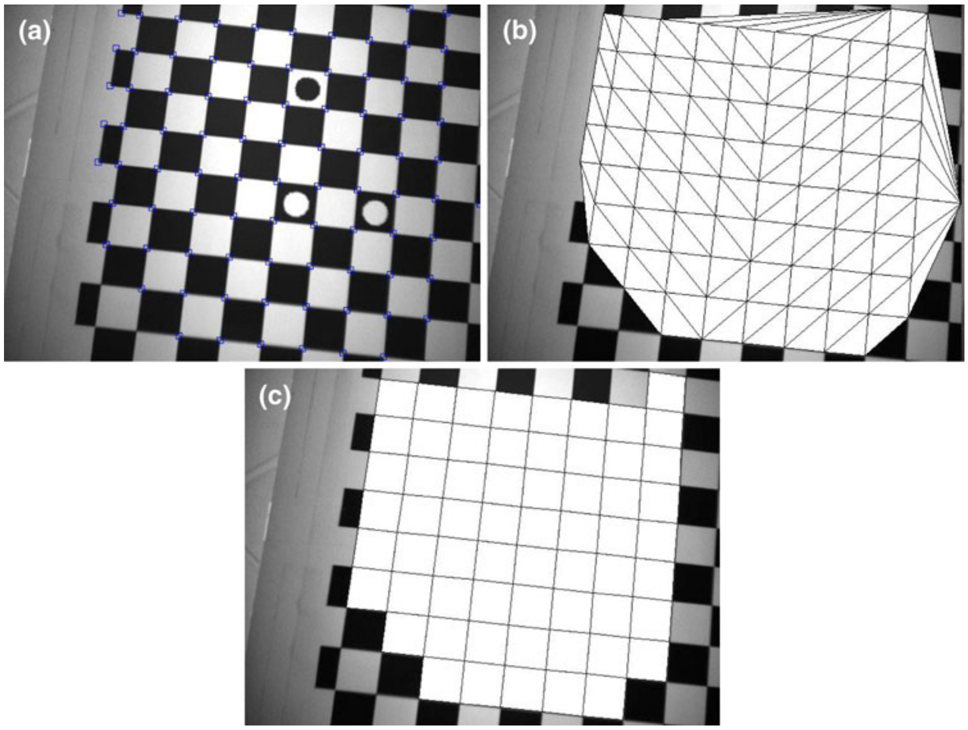


Fig. 2 The grid finding process. **a** Corner detection; **b** Delaunay triangulation of the corners; **c** Merge neighboring triangles to form quads

2 Feature point connectivity

We begin by finding corners in the images. Here, we use Harris' corner detector [10], which is based on thresholding on the eigenvalues of a gradient correlation matrix at each pixel. The checkerboard pattern gives strong corners. The positions of these corners can be found with subpixel accuracy.

Let the corner points in the image be denoted by $\{n_k\}$, and the corner points in the pattern be denoted by $\{N_k\}$. Our problem is to find a topologically compatible map $\Phi : I \rightarrow P$ such that $\Phi(n_k) = N_k$, where I is a subset of $\{n_k\}$ and P is a subset of $\{N_k\}$. By topologically compatible, we mean that there exists a graph *GI* on I and a graph *GP* on P such that *GI* is homomorphic to *GP*.

An obvious choice for the graph *GP* is the regular grid that connects each corners of the square. Now, our problem is reduced to finding the graph *GI* that matches *GP*.

We start with connecting the feature points with Delaunay triangulation, and we will show that the triangular mesh thus generated can be transformed into a regular grid.

2.1 Delaunay triangulation

Given a set of points in the plane, Delaunay triangulation is the unique triangulation of the point set in which the circumcircle of every triangle is empty. The dual of Delaunay triangulation is the Voronoi diagram with the point set as the Voronoi sites. This means that in a Delaunay triangulation triangles are always formed with points that are close to each other.

When square tiles of a checkerboard pattern are imaged through a camera, they become general quadrilaterals. Consider a black or a white tile in the image with four corners A, B, C, and D, as shown in Fig. 3. The circumcircle of the triangle $\Delta(A, B, C)$ does not contain other feature points. In fact, with the exception of point D, other feature points

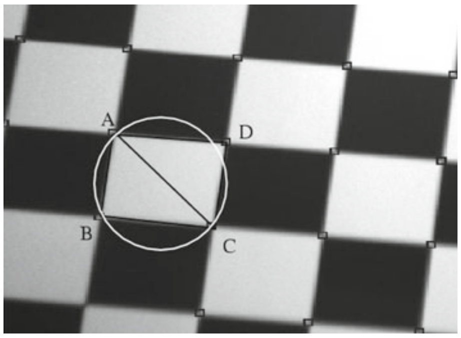


Fig. 3 The empty circumcircle test for Delaunay triangulation. Note the point D is outside circumcircle of $\Delta(A, B, C)$ and other corners are even further away. This ensures the interior colors of the Delaunay triangles are either black or white

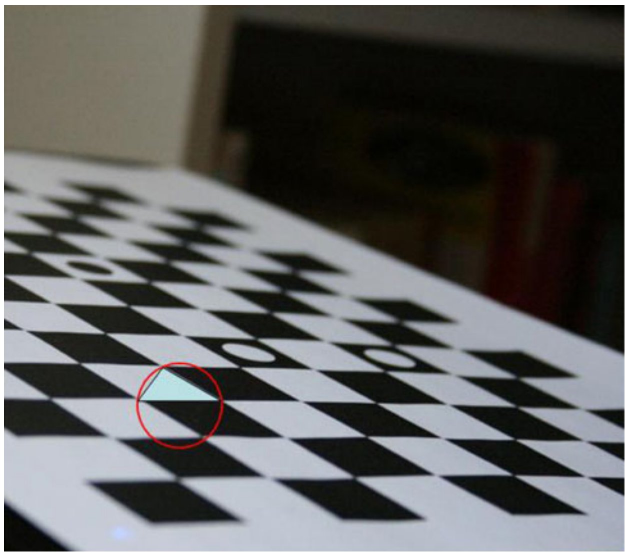


Fig. 4 An example image of low viewing angle where a triangle formed by diagonalizing does not have an empty circum circle

are quite far away from the circumcircle. Similarly, the circumcircle of triangle $\Delta(A, C, D)$ is also empty. Therefore, $\Delta(A, B, C)$ and $\Delta(A, C, D)$ are in the final Delaunay triangulation. The situation is the same in the other tiles where Delaunay triangles are formed by joining one of the diagonals of the tiles. Only in extreme cases, when the camera is pointed to the calibration pattern at a very small angle, the Delaunay triangulation may make triangles with their three corners belong to different tiles. In these cases, the corner finding is not reliable either, and therefore these images should not be used for calibration. Figure 4 shows an example of this case where the viewing angle is about 10° from the pattern plane.

Delaunay triangulation is a well studied structure in computational geometry. Guibas et al. [11] gives an incremental algorithm which runs in $(n \log n)$ time. We use an early algorithm by Watson [12]. Although its worst case performance is $O(n^2)$, in practice it runs close to linear time. We choose Watson's algorithm because its implementation is simple. For a survey of algorithms for Delaunay triangulation, see Bern and Eppstein [13]. Shewchuk [14] compares various algorithms from an implementation viewpoint.

3 Combining triangles

We wish to combine a pair of neighboring triangles to form a quadrilateral that corresponds to a black or a white tile. As shown in Sect. 2, the Delaunay triangulation will usually generate triangles by diagonalizing the tiles. From the design of the checkerboard pattern, it is clear that among the three edge-neighbors of a triangle, only one can have the same

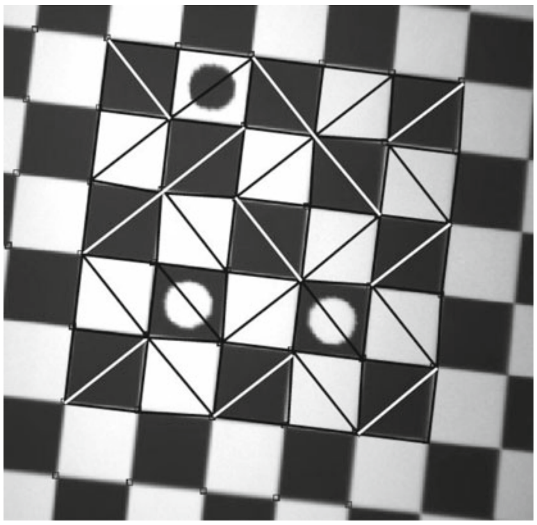


Fig. 5 Matching triangles. After Delaunay triangulation, each triangle has only one edge-neighbor that has the same color

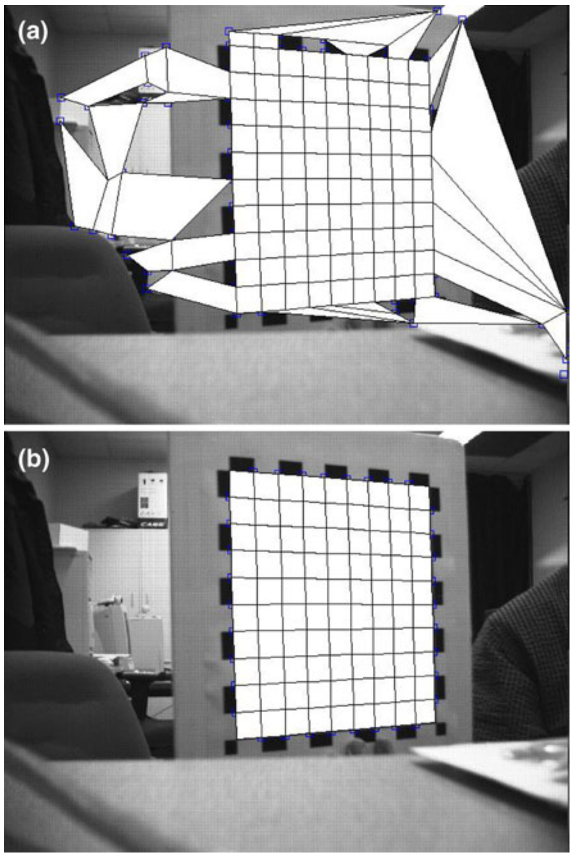


Fig. 6 The effect of topological filtering. **a** Before filtering, spurious quads exist because of corners found in the background; **b** After filtering, illegal quads are eliminated

color (see Fig. 5). In practice, we check for triangles with similar average colors. Note that the three tiles with circles satisfy the average color criterion.

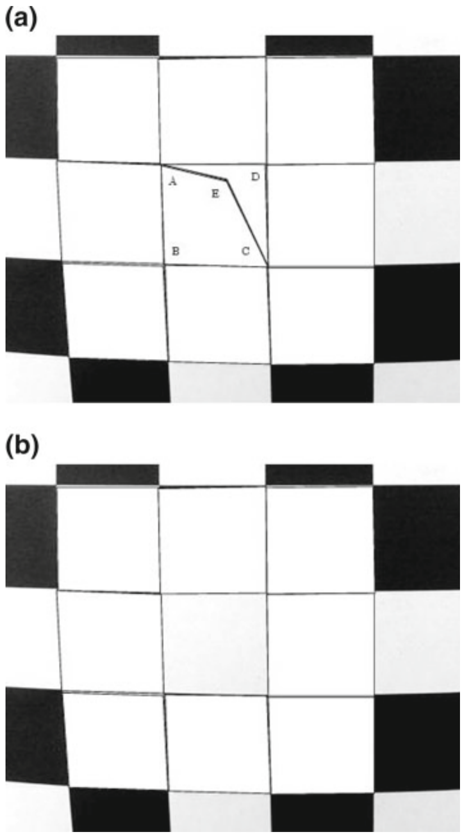


Fig. 7 Topological filtering removes false quads in the interior of the pattern. **a** Because of a noisy point E in the center white tile, two false quads ABCE and AECD are formed; **b** The two false quads are removed because they both have degree five vertices

Due to distortions, the pixels covered by a triangle may not be the same color. Since this situation always happens near the boundary of the triangle, we can simply ignore pixels with a small distance from the triangle edges.

4 Topological and geometric filtering

When the pattern in an image does not fill up the whole frame, corners may be found outside the pattern in the environment scene. These corners will be part of the triangulation and some of the triangles may pass the color test and eventually be merged into quads (Fig. 6a). Occasionally, due to noise, there are mismatched triangles inside the pattern area (Fig. 7a). These quads have to be pruned.

Notice that a proper node in a regular grid mesh has edge-degree of 2, 3, or 4. If a node has edge-degree more than 4, it is an illegal node. A quadrilateral that has two illegal nodes is removed from the mesh. Most of the illegal quads can be pruned by the topological filtering, since the chance of an arbitrary scene that contains feature points with regular grid topology is not very high.

It is possible that some illegal quads still exist after the topological filtering. These quads are removed geometrically. We assume that only the squares in the checkerboard pattern are used for calibration. Although in theory the image of a square can be an arbitrary quadrilateral, it has to be under extreme low viewing angles that a square has a large distortion. Therefore, quads that significantly deviate from parallelogram are pruned. In our implementation, we examine the lengths of the opposing edges. If the ratio between the longer edge and the shorter edge is greater than 10, we prune away the quad. Our experimental experiences show this is a conservative threshold.

Once a set of legal quads are identified, they can be rectified by using homography. Those quads that are far from squares are further removed. In Fig. 2, four rectangular quads in the upper-left corner are removed this way. Finally, isolated groups of small number of quads are also removed because they cannot be ordered to contribute to the calibration.

We emphasize the importance of topological testing. Whenever possible, we perform topological tests ahead of geometric tests, because topological tests are noise free. Determining topological information from geometric and appearance information always involves the use of thresholds which are difficult to choose. Different lighting conditions and different cameras may require different thresholds. We cannot eliminate the use of thresholds, but we can make use of the most reliable information, the topological information, first.

Figure 6 shows the effect before and after topological and geometric filtering when corners are found in the background. Figure 7 illustrates topological filtering removes false illegal quads caused by a noisy point in the interior of the pattern.

5 Ordering quads: topological walking

Once we have a quadrilateral mesh, and after the topological filtering, we have a graph with a regular grid topology. The next step is to map nodes of the quads to the corners of the pattern. We label each node in the graph *GI* with the 2D coordinates of the corner points N_i from the pattern. To do this, we start from an arbitrary quad and fix the grid coordinates of its nodes. Without loss of generality, we assume that the grid coordinates of the pattern's corner are integers. Then the neighbors of the first quad can be labeled. This process is repeated in a flood-fill fashion. When all the nodes of the quads are labeled, we can find the reference quads and transform the labeled coordinates according to the reference orientation.

Referring to Fig. 1, we define the pattern origin as the horizontally centered white circle. The x axis points toward the other white circle. To determine orientation, however, we need only to find the black circle and the white circle that lies on the same grid line. The second white circle is redundant but we use it for double checking the correctness of the labeling.

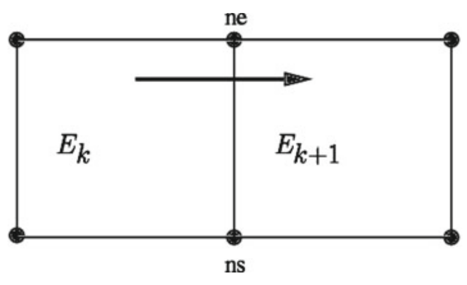


Fig. 8 Propagating node coordinates

Since every quad is visited exactly once, the algorithm is $O(n)$, where n is the number of the quads which is linear to the number of corners.

We illustrate the process of labeling node coordinates using an example of propagating element node coordinates from left to right (Fig. 8). Suppose we have fixed the coordinates in element E_k, now we walk into its right neighbor element E_{k+1} through E_k's second edge $e = (ns, ne)$. Let $N[j]$ be the node with index j in element E_{k+1}. As nodes ns and ne are shared by both E_k and E_{k+1}, we can use them to fix the orientation of E_{k+1} with respect to that of E_k. We first find the node index i of ns in E_{k+1}. Then the pattern coordinates of the nodes of E_{k+1} are assigned as follows:

$$N[i].x = ns.x$$
$$N[i].y = ns.y$$
$$N[(i+1) \bmod 4].x = ns.x + 1$$
$$N[(i+1) \bmod 4].y = ns.y$$
$$N[(i+2) \bmod 4].x = ns.x + 1$$
$$N[(i+2) \bmod 4].y = ns.y + 1$$
$$N[(i+3) \bmod 4].x = ns.x$$
$$N[(i+3) \bmod 4].y = ns.y + 1$$

This orientation operation is equivalent to re-ordering the nodes in a quad based on its neighbors.

Note that the coordinate labeling is done using pure topological information. Hence, the algorithm works even if the edges are curved due to radial distortions.

6 Experiments

The technique was tested by capturing image sequences of the calibration pattern with 12 different cameras. In each sequence the camera was moved relative to the pattern through a range of distances and orientations. The camera/pattern pose was moved by hand to several positions where snapshot images were captured while keeping the intrinsic parameters unchanged. In our case Zhang's calibration algorithm [4] was used which requires several views. Therefore, a sequence of images was needed to provide a sequence of image correspondence sets for each calibration run. The sequences were then processed on a Pentium III desktop computer with a Windows C++ program which performed the grid finding, with the result being a set of point correspondences for most of the frames in the sequence. The successfully extracted point correspondences from a sequence were then input to the OpenCV calibration function *cvCalibrateCamera*, which implements the Zhang's algorithm.

A total of 28 sequences with 12–52 frames, with three different resolutions were captured. Standard video (NTSC), IEEE 1394 (Firewire) and USB cameras were used. The 12 cameras and the resolution modes used are: two color and two greyscale PointGrey DragonFly firewire cameras (640 × 480 pixels), a low-cost single board greyscale NTSC camera (416 × 484 pixels), three Intel Easycam Webcams (320 × 240 pixels) captured in greyscale, an NTSC camcorder Sharp VL-AH150U captured at 320 × 240 pixels through the NTSC input of a Intel IPro webcam, a wireless security NTSC camera also captured at 320 × 240 pixels by the Intel IPro webcam, and the camera input itself from the Intel IPro (320 × 240), and a Telemax webcam (320 × 240 pixels).

The calibration pattern from Fig. 1 was printed out onto a small (20 × 25cm) and large (150 × 150cm) sheet, and each mounted on a flat panel. 14 of the 28 sequences were filmed aiming at the smaller pattern and the rest viewed the larger pattern.

For each frame, a list of correspondences between image and world (pattern) coordinate points are created, one from each quad node. Since the goal was to accurately find correspondences, the results are presented as how often the grid in the image was successfully found. For each sequence, the grid finding success (manually checked) and the number of extracted correspondences is reported. Table 1 details the success in finding these correspondences. Each row represents a single sequence from a camera. For example, the first row shows that there are 20 frames in the sequence captured by one of the Color Dragonfly camera, 95% of the frames had a grid successfully located, and on average 93 correspondences were found in each frame. A frame is failed if the program cannot find a set of topologically correct quads. Of out 28 sequences of images tested using a variety of cameras under different lighting conditions and poses, only three sequences have successful rates lower than 30% where there are not enough located patterns for calibration. All successfully located frames have enough correspondences for estimating a homograph matrix.[1] Note that we do not need all the frames to be successfully located in order to calibrate the camera. Figure 9 shows a few sample frames from our test data. The grids that are found by the program are painted white. Note that our method works when the pattern is partially occluded in the images. The OpenCV grid finder fails in this situation. In fact, it fails to locate the grids in every frame in Fig. 9 due to either pattern occlusion or missing corners. Being able to handle pattern occlusion is important for calibrating lens distortions where we need to move the pattern close to the camera to fill the frame.

Table 1 Results of automatic grid finding on 28 sequences

Camera	Frames	Located (%)	Points/grid
Color Dragonfly A	20	95	93
Color Dragonfly B	20	75	81
Color Dragonfly B	20	100	94
Greyscale Dragonfly A	20	50	93
Greyscale Dragonfly B	12	100	72
Low Cost NTSC	52	83	75
Intel EasyCam A	32	81	69
Intel EasyCam A	32	56	57
Intel EasyCam A	32	100	68
Intel EasyCam A	32	91	75
Intel EasyCam A	32	97	49
Intel EasyCam A	32	100	55
Intel EasyCam B	32	94	76
Intel EasyCam B	32	84	48
Intel EasyCam B	32	72	58
Intel EasyCam C	32	31	66
Intel EasyCam C	32	84	61
Intel EasyCam C	32	19	62
Camcorder/IPro	32	19	68
Camcorder/IPro	32	50	75
Camcorder/IPro	32	85	54
Wireless/IPro	32	88	65
Wireless/IPro	32	16	70
Wireless/IPro	32	63	71
Intel IPro	32	94	56
Intel IPro	32	94	61
Telemax Webcam	32	69	64
Telemax Webcam	32	13	36

The *Located (%)* field indicates how many of the frames in the sequence out of a total of *Frames* frames had a grid successfully located in them. The *Points/grid* field indicates the average number of correspondences found per successfully located grid

[1] Zhang's algorithm requires at least four correspondences in each frame to estimate a homography matrix.

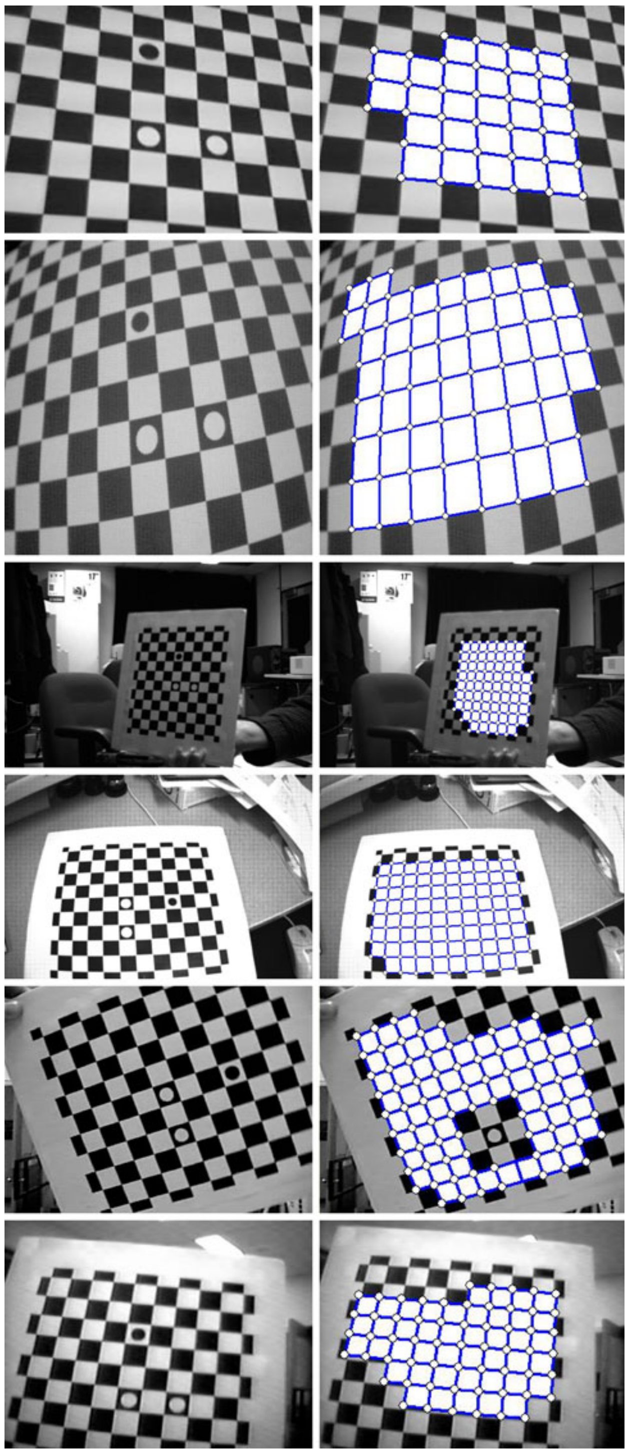


Fig. 9 Sample frames from the test sequences (*left column*) and grids found (*right column*)

Table 2 provides statistics for each frame of a sequence. For each frame, we report the number of corners found, the number of triangles after Delaunay triangulation, the number of quads found, the number of quads after pruning, the number of corresponding points, and the total processing time. Table 3 gives a breakdown of average time spent in each processing stage. These timing results show that our grid finding algorithm is useful for applications that require determining camera poses in real-time, such as in robot localization and augmented reality.

Figure 10 shows the results of grid finding under a variety of camera orientations. The camera is moved between 0° and 60° relative to the normal of the pattern plane, a very useful range for Zhang's algorithm [4] which requires the pattern planes at oblique angles to the camera.

Figure 11 shows the results of grid finding on a distorted image and unwarped image after calibration. Since our algorithm orders quads in topological space, the curved edges caused by lens distortion do not present a difficulty.

7 Conclusions

We have presented a method for finding regular grids from calibration patterns using Delaunay triangulation. By triangulating the feature points, we create connectivities between the isolating feature points. Working with a mesh data structure helped exploit the topological constraints easily and greatly enhanced the robustness of the algorithm. The mesh data structure also results in a linear time algorithm for matching the grids.

The technique of triangulating feature points and combining them to form quadrilaterals can be applied to general marker identification in augmented reality and robot navigation. The principle of this approach is that we search for topological regularities as well as appearance patterns. We may also design markers in such a way that their features have certain topological invariance under perspective transformation.

We use three circular dots in our pattern to assist determining the camera orientation. This scheme can easily be extended to design a marker system that uses more dots in the checkerboard pattern for encoding different markers. This is similar to the one developed at the Siemens Corporate Research [15]. However, the advantage of using the checkerboard pattern is that it gives more correspondences.

For future work, we plan to further improve the robustness and efficiency of the algorithm by adding corner validation mechanisms, more sophisticated topological filtering techniques, and optimization of the current algorithms. It is also interesting to compare the checkerboard-based calibration patterns with the circle-based patterns [16].

Although Delaunay triangulation is not projective invariant in general, in practice it is not a significant limitation, as our experiments have shown. However, finding out the theoretical angle bound at which Delaunay triangulation is invariant under projective triangulation is an interesting problem. We are currently investigating this problem.

Table 2 Processing stages in a sequence for each image frame

Frame	Corners	Triangles	Quads	Pruned quads	Points	Total time (ms)
0	35	48	24	24	31	52
1	36	54	25	24	31	53
2	43	65	32	30	38	65
3	43	65	32	30	38	65
4	43	66	32	30	38	63
5	46	69	34	33	40	68
6	43	65	32	30	38	64
7	43	65	33	30	38	64
8	43	65	32	30	38	66
9	44	66	32	31	39	66
10	45	67	33	32	40	67
11	45	67	33	32	40	65
12	46	68	33	33	41	67
13	53	84	38	38	45	77
14	55	88	42	40	49	81
15	56	89	43	41	50	82
16	57	94	43	41	50	83
17	57	90	44	42	50	82
18	59	97	45	43	52	87
19	61	99	46	45	53	87
20	63	107	47	45	54	92

Table 3 Average time spent in each processing stage

Stage	Average time per frame (ms)
Harris corner detector	29.5
Delaunay triangulation	1.6
Triangle merging	15.9
Topological filtering	19.2
Ordering quads	16.9
Total	83.1

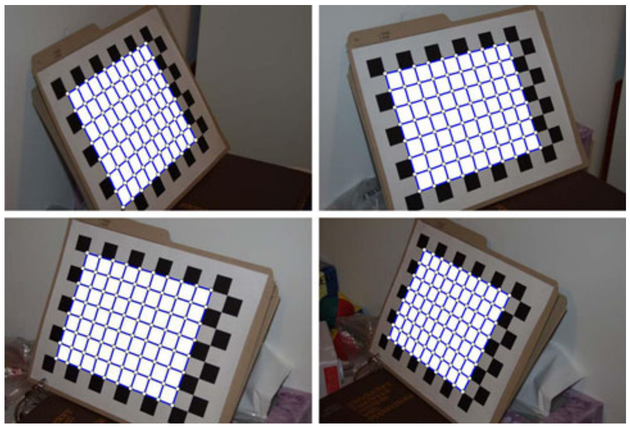
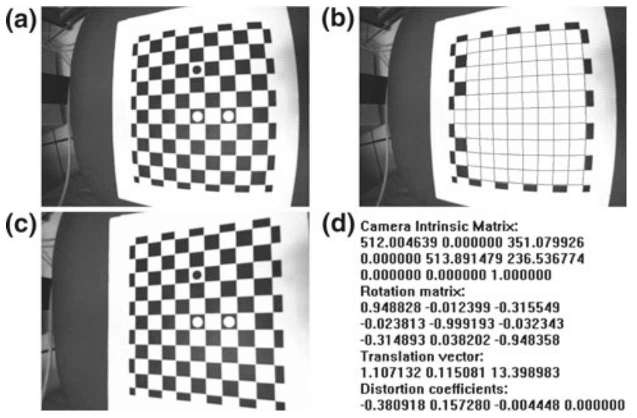


Fig. 10 Grid finding under different camera poses. The successfully identified quads are painted *white*



```
Camera Intrinsic Matrix:
512.004639 0.000000 351.079926
0.000000 513.891479 236.536774
0.000000 0.000000 1.000000
Rotation matrix:
0.948828 -0.012399 -0.315549
-0.023813 -0.999193 -0.032343
-0.314893 0.038202 -0.948358
Translation vector:
1.107132 0.115081 13.398983
Distortion coefficients:
-0.380918 0.157280 -0.004448 0.000000
```

Fig. 11 Results of calibration and unwarped image. **a** One of the original input images; **b** Result of the matched quads; **c** Results of the undistorted image after calibration; **d** Calibration results showing intrinsic and extrinsic camera parameters and lens distortion coefficients

The grid finding program together with a calibration program are available for downloading at http://www.scs.carleton.ca/~c_shu/Research/Projects/CAMcal/.

References

1. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge (1998)

2. Sturm, P.F., Maybank, S.J.: On plane-based camera calibration: a general algorithm, singularities, applications. In: IEEE Conference on Pattern Recognition and Computer Vision (CVPR99), vol. 1, pp. 432–437 (1999)
3. Tsai, R.Y.: A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE J. Robotics Autom. **3**(4), 323–344 (1987)
4. Zhang, Z.: A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. **22**(11), 1330–1334 (2000)
5. OpenCV: Open Source Computer Vision Library. http://www.intel.com/research/mrl/research/opencv/
6. Willson, R.: Tsai Camera Calibration Software. http://www-2.cs.cmu.edu/People/rgw/TsaiCode.html (1995)
7. Fiala, M., Shu, C.: Self-identifying patterns for plane-based camera calibration. Mach. Vis. Appl. **19**(4), 209–216 (2008)
8. Soh, L., Matas, J., Kittler, J.: Robust recognition of calibration charts. In: IEE 6th International Conference on Image Processing and its Applications, pp. 487–491 (1997)
9. Tell, D., Carlsson, S.: Combining appearance and topology for wide baseline matching. In: European Conference on Computer Vision (ECCV2002). Lecture Notes in Computer Science, vol. 2350, pp. 68–81 (2002)
10. Harris, C., Stephens, M.: A combined corner and edge detector. In: Alvey Vision Conference, pp. 147–151 (1988)
11. Guibas, L.J., Knuth, D.E., Sharir, M.: Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica **7**, 381–643 (1992)
12. Watson, D.F.: Computing the *n*-dimensional Delaunay tessellation with application to Voronoi polytopes. Comput. J. **24**(2), 167–172 (1981)
13. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. In: Du, D.Z., Hwang, F. (eds.) Computing in Euclidean Geometry, Lecture Notes Series on Computing, vol. 1, pp. 23–90. World Scientific, Singapore (1992)
14. Shewchuk, J.R.: Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In: First Workshop on Applied Computational Geometry, Philadelphia, Pennsylvania, pp. 124–133 (1996)
15. Zhang, X., Fronz, S., Navab, N.: Visual marker detection and decoding in AR systems: A comparative study. In: International Symposium on Mixed and Augumented Reality (ISMAR 2002), pp. 79–106 (2002)
16. Jiang, G., Quan, L.: Detection of concentric circles for camera calibration. In: International Conference on Computer Vision (ICCV'05), vol. 1, pp. 333–340 (2005)