Sub-pixel Corner Location Using Edge Intersections

Tim Yates
MSCS, St. Olaf College

May 18, 2012

1 Introduction

Accurate feature detection is an important but often overlooked aspect of camera calibration. Most calibration techniques involve the approximation of camera parameters and pose using a set of known point correspondences across the images. A very common technique is finding corners on a planar checkerboard pattern: the simple geometry and (relatively) detectable features are very helpful.

This paper describes an approach to locate corners with subpixel precision from multiple checkerboard patterns in an image.

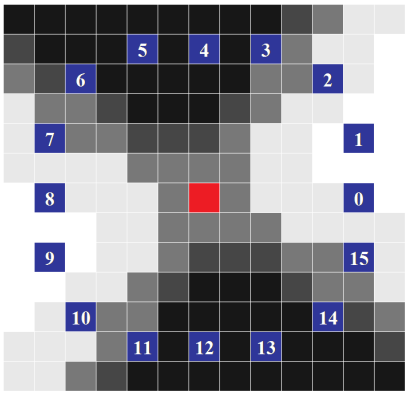

Figure 1: ChESS measures the pixel values in a circular region around the center and computes a response that measures how much those pixels "look like" a corner.

2 Procedure

Our corner finding procedure has two main parts: locating likely corners using the ChESS feature detector, and using data from that step to intersect lines fitted to edges.

2.1 Corner approximation

The ChESS feature detector (Bennett and Lasenby) calculates a value for each pixel indicating the likelihood that it is a the center of a chessboard corner. Once these corner responses are calculated over the whole image, we choose a set of most likely corner locations with a) a high-pass filter that requires the response to meet a certain threshold, and b) non-maximum suppression to choose the pixel with the highest value in its immediate neighborhood.

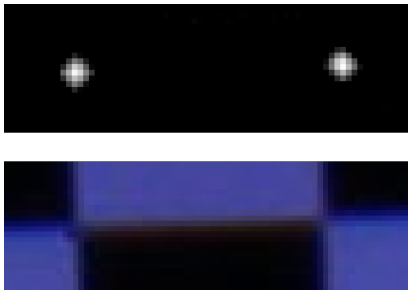

Figure 2: A corner in the image (bottom) and its ChESS response before filtering (top).

An alternative to non-maximum suppression is to calculate the center of mass of a region of nonzero pixels. On its own, this method can provide a sub-pixel estimate of the corner locations—an approach we will use as a comparison.

Next, we need to assign each corner pixel to a location on a target. First, we cluster the points using the iterative k-means algorithm in order to group them by target. In case any "corner-like" features in the image produce extraneous points, we discard corners whose distance from the cluster mean is greater than twice the average. Since such outliers are rare, this approach is robust enough for real data.

Next, we determine the ordering of corners into rows and columns by brute force: First we find the four outer corners of the board by picking the combination that gives the greatest area out of all the points. Note that, ideally, these four points would make up the convex hull. Due to image distortion and sampling error, however, some points along the outer edge might not lie inside this area. To ensure a consistent orientation, the point at the origin of the board is always the one nearest to the image ori-

gin by Manhattan distance—as a consequence, the board should not rotated by more than $\frac{\pi}{2}$ in the image. The area is calculated such that the points are always in clockwise order (or counterclockwise, if the y axis is positive in the upward direction), allowing us to distinguish rows from columns.

After we have determined the outer corners and their orientation, we need to order the inner corners by rows and columns. We know that the number of corners N is a product of the number of rows n_r and columns n_c. We can easily find paired factors of N using trial division, which gives us all the candidate values for n_r and n_c. Using the four outer corners, we can estimate the homography $\mathbf{H}$ that maps the coordinates on the target to the coordinates on the image (see (Zhang 2000)), assuming that all corners are located at lattice points in the target coordinate system, so that the outer corners are at $[0\ 0]^T$, $[n_c\ 0]^T$, $[n_c\ n_r]^T$, and $[0\ n_r]^T$. For each candidate pair (n_r, n_c), we can compute a corresponding homography and project the expected corner locations:

$$\mathbf{p}_{ij} = \mathbf{H} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \quad \text{for all } 0 \le i \le n_c, 0 \le j \le n_r$$

Next, we match the real and expected corner points according to distance. We choose the pair (n_r, n_c) that gives the lowest sum of residual distances (and thus best matches the true board dimensions), and order the corners by their associated target coordinates.

2.2 Edge Detection

We want to precisely locate corners by fitting lines to the edges of each square. A plot of intensity values along a row of pixels near a typical edge is shown in Figure 3.

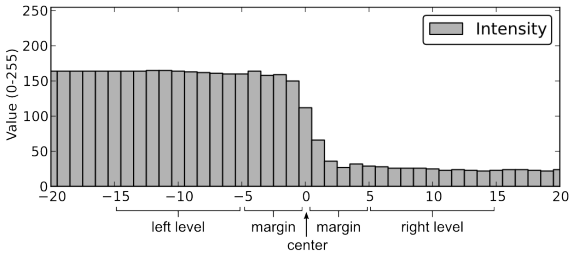

Figure 3: Intensity values along a row of pixels near a checkerboard edge. The white and black levels are computed from the average of the two regions to the right and left of the center.

Since these edges typically follow a smooth, symmetric curve between black and white, we can define the exact location of the edge to be the point where the intensity curve crosses midway between the black and white levels.[1] The tricky part of this approach is choosing the black and white levels: arbitrary constants (such as 0 and 100% intensity) will perform poorly if the image exposure is not controlled precisely, and historgram-based methods (minimum and maximum, quantiles, or some similar statistic) will not correctly handle variations in lighting across an image. Instead, we take the average of two regions on either side of the center, as illustrated in Figure 3.

When sufficiently spaced (so as not to include any of the edge pixels) and taken over an appropriate number of pixels, these should give reliable, local white and black levels. The actual midway point (hereafter known as the *edge threshold*) is the average of these two levels, and can be computed for any center pixel i:

$$T(i) = \frac{1}{2}(L(i) + R(i))$$

where

$$L(i) = \frac{1}{w} \sum_{j=i-m-w}^{i-m} I_i$$

$$R(i) = \frac{1}{w} \sum_{j=i+m}^{i+m+w} I_i$$

where w is the width of the white and black regions, m is the margin between the center pixel and those regions, and I_i is the intensity at pixel i. The resulting function[2] is plotted in Figure 4

Since we are looking for points where the intensity function crosses the threshold function, we can just take the difference:

$$D(i) = T(i) - I_i$$

and look for zero crossings. This difference function looks is shown in Figure 4.

To determine the exact locations of zero crossings, we need to interpolate. Since the intensity gradient is roughly constant near the edge, we used linear interpolation for simplicity. We did not investigate the use of higher-order splines, although this might be advisable.

[1] This differs from most edge detection approaches, where the edge location is defined as the point with the maximum intensity gradient. For the simple black-white edges in our problem, both approaches should give nearly the same result. Additionally, our approach only requires interpolation of the intensity and not the gradient.

[2] Note that we can compute this function efficiently for all pixels by computing the cumulative sum once and using the fact that $\sum_a^b x_i = \sum_0^b x_i - \sum_0^a x_i$.

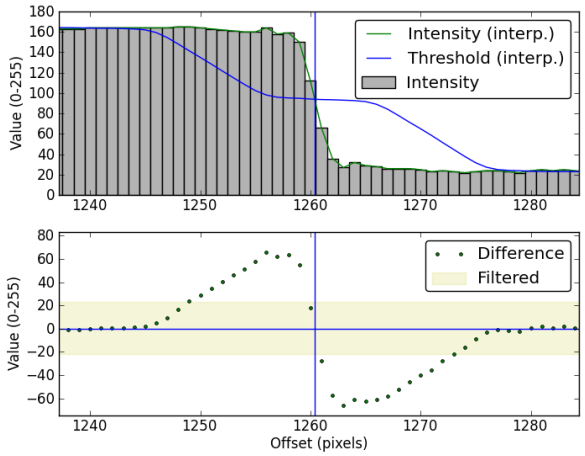

Figure 4: The threshold function around an edge (top) and the difference between it and the intensity (bottom). The interpolated zero crossing of the difference function is where the edge lies. To filter out zero crossings in non-edge regions, we require that they be flanked by "high" and "low" values on either side.

Since this difference is near zero inside the white and black regions, we need some method to filter out all the extraneous zero crossings due to noise in these areas. There are several options here. The approach we took was to look for strong signals by defining "high" and "low" values and requiring that they follow each other (in either order) with only one intervening zero crossing, as illustrated in Figure 4.

Finally, note that our edge detector only works in one dimension. To find all the edges in the image, we run the detector over each row and column of pixels and aggregate all the edge points.

2.3 Edge Intersection

Once we have corner estimates and edge points, we want to refine the corner estimates by fitting lines to the horizontal and vertical edges near a corner and finding their intersection. To do this, we need to assign the edge points to horizontal and vertical segments. The easiest way to achieve this is to use the estimated homographies to back-project the edge points from image coordinates to target coordinates. This is simply a matter of multiplying them by the homography's matrix inverse:

$$a \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{H}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

where u and v are image coordinates, x and y are target coordinates, and a is an arbitrary scale factor that can be eliminated by dividing by the last element of the result. Once x and y are known, we can find the coordinates i and j of the nearest corner by simply rounding to the nearest integer. The edge pixel is part of the vertical segment if $|x - i| < |y - j|$, and part of the horizontal segment otherwise (Figure 5).

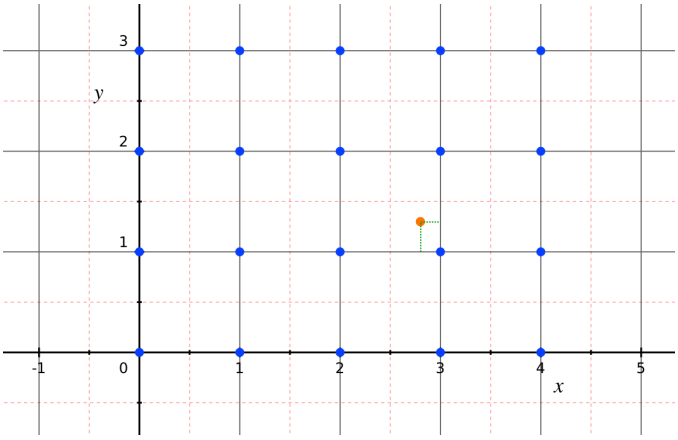

Figure 5: Edge points are assigned to segments based on their nearest corner and whether the horizontal or vertical distance is greater. In this case, the orange point would be assigned to the vertical segment of corner $(3, 1)$.

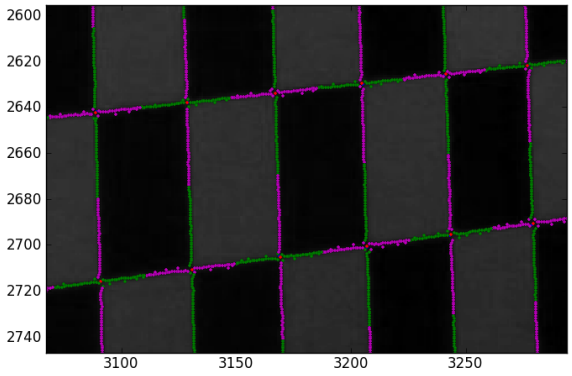

Figure 6: The resulting groups of edge pixels corresponding to each line segment, shown in alternating colors. Each corner has a single horizontal and vertical line segment centered around it. The slope and intercept of the segments are found using orthogonal regression.

Once the edges are grouped into segments (Figure 6), they are fitted to lines using orthogonal regression. The

Image Set	Mean Reprojection Error (px)
Edge Intersection	0.34
ChESS CoM	0.13
Previous Work	0.21

Table 1: Average reprojection errors when calibrating an image set using each set of corner estimates

corner point is given by the intersection of the horizontal and vertical segments corresponding to it:

$$x = \frac{\alpha_h - \alpha_v}{\beta_v - \beta_h}$$
$$y = \alpha_h + \beta_h x$$

where β_h and β_v are the slopes and α_h and α_v are the y-intercepts of the fitted horizontal and vertical lines.

3 Results

As a stand-in for the accuracy of our corner locations, we compared the average reprojection error when calibrating a data set whose accuracy, emperically, was limited by the noise in the corner estimates. We used corner estimates from our estimator, the ChESS estimator with center-of-mass locations, and work by a previous group on the same problem.

As is clear from Table 1, the edge intersection method was not able to improve on the performance of ChESS. We were also not able to meet the performance of the previous work on this problem

4 Assumptions & Limitations

There are several limitations to our edge intersection method. First, it does not explicitly account for lens distortion. In any real image, the chessboard edges are not perfectly straight, in part because of radial or tangential distortion of the lens system. We fit short line segments—rather than continuous lines along the chessboard—to minimize the bias that this introduces, but that comes with the tradeoff of increased variance due to the smaller sample sizes.

As noted earlier, we also make some assumptions about the mathematical structure of the edge: namely, that the edge gradient is roughly constant near the edge location (so that we can use linear interpolation) and that the edge is located at the intensity mark midway between the white and black values (so that we do not have to estimate the maximum gradient). Both of these assumptions are easily relaxed by adding higher-order spline interpolation, but that comes at a computational cost, plus the same bias-variance dilema as before.

Currently, our system also requires several parameters:

- The edge margin (which should be slightly greater than its expected "spread")
- A threshold (percentage of the extrema) for the high-pass edge filter
- A threshold (constant) for the high-pass ChESS response filter
- The number of targets in the image

References

Stuart Bennett and Joan Lasenby. ChESS: quick and robust detection of chess-board features. URL `http://www-sigproc.eng.cam.ac.uk/~sb476/publications/ChESS.pdf`.

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