Computer Vision (CE316 and CE866): Feature Detection and Description

Adrian F. Clark
⟨*alien@essex.ac.uk*⟩

We consider the detection of edges and corners in images. Motivated by the desire to match corners between images, we then explore ways of describing corners and then matching them. Finally, we consider briefly detectors and descriptors for more general features in images.

Contents

List of Figures

1 Introduction

We now have enough image processing 'equipment' to be able consider two of the mainstays of computer vision, edge and corner detection. We shall then consider more general feature detection. For a long time, edge detection was regarded as one of the main problems in computer vision, at least partly because humans find that line drawings of scenes are easy to interpret and processing lines rather than entire images is presumed to be easier for computers too — though the author doesn't follow this logic himself. Nevertheless, edge detection remains a topic of interest as the problem certainly has not been solved, especially for natural scenes.

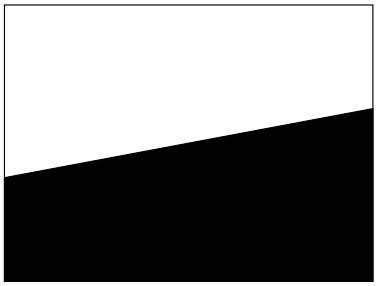


Figure 1: The aperture problem

Edge detection, no matter how well done, suffers from a significant drawback for image interpretation. As Figure 1 shows, when an edge between (say) a dark object and a white background fills the field of view, it is impossible to determine which part of an edge is associated with which part of the object; this is known as the *aperture problem*. Thus, for image analysis, edges are actually of limited value. In practice, it is much more useful to identify the locations of *corners* in an image, as they are related to characteristic features of the objects that created them.

Notwithstanding the above comments, we shall first consider the edge detector due to John Canny, which probably remains the most-used edge detector in image analysis and computer vision. We shall then move on to consider the problem of corner detection. The first scheme we shall look at, due to Moravec, is fairly easy to understand and represented the state of the art in around 1987. Subsequent work in the UK by Harris and Stephens circa 1989 produced a somewhat more effective corner detector which remains competitive with the best, though still vastly inferior to the human visual system.

2 The Canny edge detector

The development of the Canny edge detector has an Essex connection. A researcher called Mike Brady was a lecturer in Essex's Department of Computer Science and he had a PhD student called Libor Spacek who was working on the problem of edge detection. Part-way through Libor's research programme, Mike took up an appointment at MIT where he got a Masters student, John Canny, to take a somewhat cut-down approach to the careful maths underlying Libor's edge detector. This was published in an MIT report around 1984, then in a journal paper [Canny, 1986] — and was taken up by most of the companies and institutions researching computer vision in the USA. Incidentally, Mike returned to the UK a little later in the 1980s as a professor in the Department of Engineering Science at Oxford and has become one of the prime movers of vision research in the UK. He moved from robot vision into medical imaging research in the 1990s and was knighted a few years ago. Libor was a member of academic staff in CSEE until his retirement a few years ago.

The Canny edge detector is based around three principles:

1. it should respond only to edges, and all edges should be found;

2. edges should be found in the correct places; and

3. multiple edges should not be found where only a single edge exists.

Canny considered how these three criteria could be obtained at a step edge in an image. Without going into the mathematics, he ended up with the following five-step algorithm.

Step 1. Convolve the image with a Gaussian-shaped mask (see Figure 2) to smooth the image and reduce the effects of noise. The s.d. of the Gaussian controls the amount of smoothing obtained, so this part of the algorithm can be tailored to the characteristics of the data.

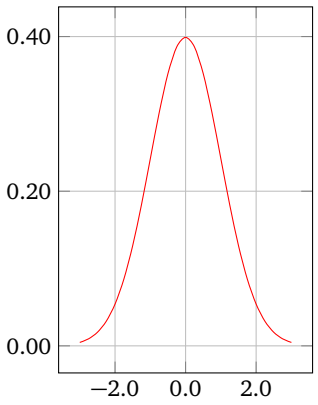


Figure 2: Profile across a Gaussian mask

Step 2. Find differences in the horizontal and vertical directions, averaging over 2×2 squares of the image S:

$$H(x,y) = \frac{(S(x,y+1) - S(x,y)) + (S(x+1,y+1) - S(x+1,y))}{2} \tag{1}$$

$$V(x,y) = \frac{(S(x+1,y) - S(x,y)) + (S(x+1,y+1) - S(x,y+1))}{2} \tag{2}$$

Step 3. Find the magnitudes and directions of these gradients:

$$M(x,y) = \sqrt{V(x,y)^2 + H(x,y)^2} \tag{3}$$

$$\theta(x,y) = \tan^{-1} \frac{V(x,y)}{H(x,y)} \tag{4}$$

Note that, when calculating $\theta(x,y)$ on a computer, you need to use the `atan2` function, which returns a value in the range $-\pi$ to $+\pi$, rather than `atan`, which returns a value in the range $-\pi/2$ to $+\pi/2$: `atan2` takes into account the signs of both of its arguments in determining the angle.

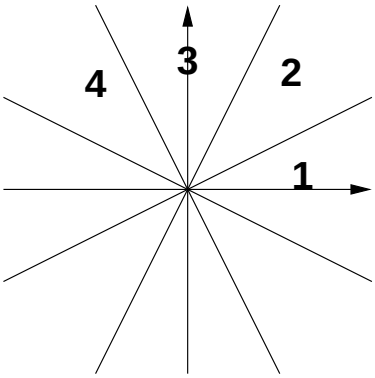


Figure 3: Quantization of $\theta(x,y)$ into four directions

Step 4. A broad edge in the image will tend to produce two edge responses, one at either side. This violates our third criterion, so something must be done to reduce any broad lines in $M(x,y)$. The approach taken is to perform *non-maximum suppression*, *i.e.* to remove all those parts of the edge except where there is the greatest local change. This is carried out in two stages:

- Firstly, $\theta(x,y)$ is quantised into four values that indicate roughly the direction of the edge gradient (Figure 3).
- Then, for each 3×3 neighbourhood in $M(x,y)$, the value at x,y is compared with its neighbours along the four possible gradient directions and, if $M(x,y)$ is less than any neighbour, it is set to zero.

The result is that any broad edges in $M(x,y)$ are thinned, usually to be a single pixel in width.

Step 5. Even after non-maximum suppression, there will usually be many false edge fragments, due to texture and noise in the image. These typically have much poorer contrast than edges obtained from the boundaries of objects, which is what computer vision is more interested in. To reduce the effects of these false edge segments, we attempt to link them together. In the Canny detector, we use a thresholding strategy which employs two thresholds, τ_L and τ_H, where typically $\tau_H \approx 2\tau_L$. This scheme is sometimes called *hysteresis thresholding*. Simply thresholding $M(x,y)$ using a sensibly-chosen value for τ_H will reduce the number of false edge segments but the true edge segments will inevitably contain gaps. The idea is to try to join up edge segments.

When the contour formed by following successive pixels with values $> \tau_H$ (shown in red in Figure 4) ends, the hysteresis thresholding algorithm looks in the 3×3 region around that pixel for any neighbours whose value is $> \tau_L$ (shown in black in Figure 4). If so, it continues the edge to that pixel. This process continues until there are no suitable neighbours or the another edge segment with value $> \tau_H$ has been found.

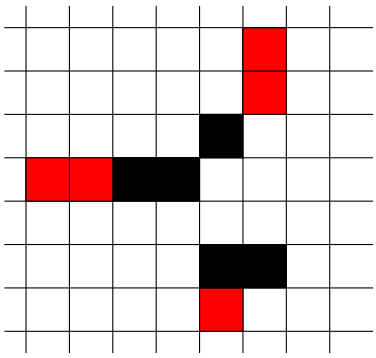


Figure 4: Linking together edge segments

As we can see from Figure 6, Canny's edge detector gives better results than the Laplacian or Sobel detectors — which it should, of course, since it is somewhat more sophisticated. However, we can also see that the Canny result is far from perfect on images of natural scenes. For robot vision in research laboratories, where illumination tends to be strong and the boundaries of objects straight and well-defined, it is fairly effective.

3 The Moravec corner detector

Like so many other image processing operators, the Moravec corner detector is based around the processing of a region with a mask of coefficients. In this case, the strategy is to devise a mask with the property that convolution with the mask should produce a maximum in its output when it is centred on a corner, and a large decrease when the mask moves away from the corner.

If we think about this, we'll see that there are three main cases, shown in Figure 5:

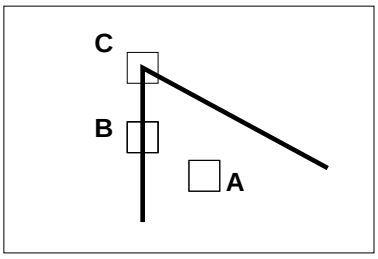


Figure 5: The difference cases considered in the Moravec corner detector

A: if the region of the image is fairly uniform, all shifts result in a small change in response;

B: if the region straddles an edge, a shift along the edge produces a small change but a shift perpendicular to the edge produces a large change;

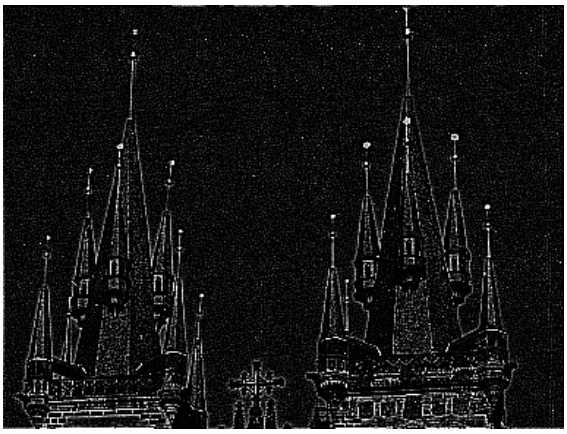
C: at a corner or isolated point, all shifts produce a large change.

If we write $I(x,y)$ as the value of the pixel at x,y and $W(u,v)$ as the coefficient in the mask for some values of (u,v), then we can write this as

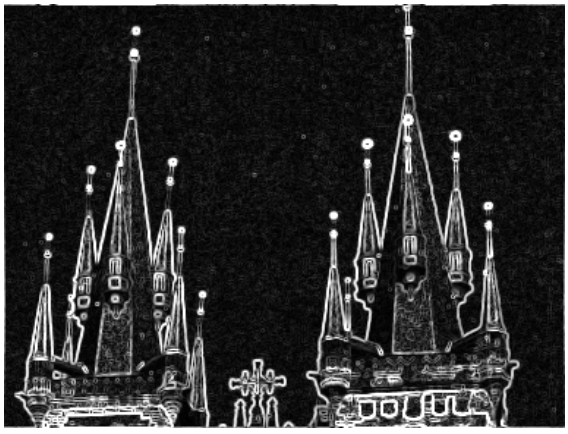
$$E(x,y) = \sum_{u,v} W(u,v)\,(I(x+u,y+v) - I(u,v))^2 . \qquad (5)$$

If we consider shifts in x,y of $(1,0)$, $(1,1)$, $(0,1)$ and $(-1,1)$, then we explore the same four directions that we used when quantising $\theta(x,y)$ in the Canny edge detector (Figure 3). Hence, if we look for local *maxima* in $\min(E)$ above some threshold value, we should be able to identify corners.

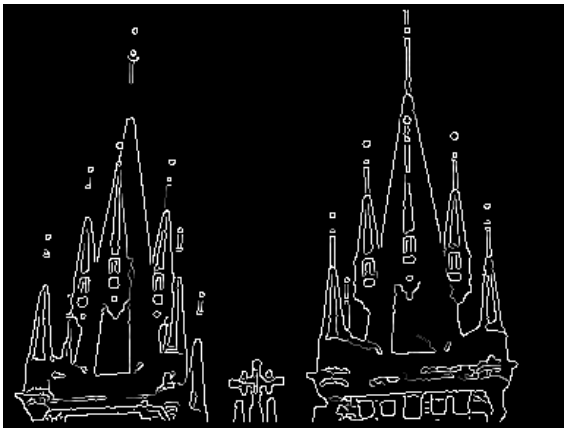
(a) Original image



(b) Result of Laplacian



(c) Result of Sobel



(d) Result of Canny

(e) Result of Harris & Stephens

(f) Result of SUSAN

Figure 6: Comparing the results of different edge and feature detectors

4 The corner detector of Harris and Stephens

As mentioned above, the Moravec detector represents the state of the art in around 1987. Subsequent researchers have improved on this, in particular in the detector due to [Harris and Stephens, 1988], which combines the ideas of Moravec and Canny has been the most widely-used corner detector since the early 1990s. The advance due to Harris and Stephens seems fairly straightforward, given our knowledge of Canny's edge detector: rather than considering shifted patches, Harris and Stephens considered the direction of the derivatives at the corner directly.

Consider a patch of a grey-scale image $I(u,v)$ shifted by (x,y). The weighted sum-squared difference is given by (5). If one expands $I(u+x,v+y)$ as a Taylor series then, if I_x and I_y are the partial derivatives of I in the x- and y-directions respectively,

$$I(u+x,v+y) \approx I(u,v) + xI_x(u,v) + yI_y(u,v)$$

which means that

$$E(x,y) \approx \sum_{u,v} w(u,v)\left(xI(u,v) + yI(u,v)\right)^2.$$

If we calculate the matrix of partial derivatives at point (x,y)

$$\mathbf{A} = \begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix}$$

then

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} \mathbf{A} \begin{pmatrix} x \\ y \end{pmatrix}$$

will have a large variation irrespective of direction x and y. Rather than calculate $S(x,y)$, Harris and Stephens calculate $\det(\mathbf{A}) - \kappa \mathrm{trace}^2(\mathbf{A})$, where κ is a tuning parameter usually in the range 0.04–0.15, to determine whether a corner is present. (For more mathematically-inclined readers, this is an approximation to the eigen decomposition of $\mathbf{A}$.) The output from Harris and Stephens is shown in Figure 6. These days, it is able to run at video rate on off-the-shelf hardware.

5 Other corner detectors

Work subsequent to that of Harris and Stephens has produced more sophisticated detectors, two of which are worthy of mention here. The first is the *smallest univalue segment assimilating nucleus* ("SUSAN" — a strained acronym if there ever was) operator of Steve Smith and Mike Brady (again) [Smith and Brady, 1997]. An example of its output is also shown in Figure 6.

Secondly, Edward Rosten and colleagues at Cambridge developed the FAST corner detector [Rosten, Porter and Drummond, 2010]. This is based on a particularly simple observation: at a corner, more than half the pixels will be dark or light, as illustrated in Figure 7. Although some sophistication is required to make FAST work well, a C implementation is easily able to operate at video rates.

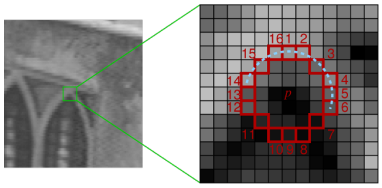


Figure 7: The FAST corner detector (from http://mi.eng.cam.ac.uk/~er258/work/fast.html)

6 Describing corners

Detecting corners may help us detect the boundaries of a feature (an obstacle in front of a robot, say) but that is of limited value. It is much more useful if one is able to relate the positions found in one frame of a video sequence to those in the next frame; or, with a pair of cameras looking at a scene, which feature found in the image from the left camera matches a feature found in the right camera. To be able to do that, one needs to be able to *describe* corners in some way, and then see how similar the descriptors are to those calculated from different frames or cameras.

The best identifying characteristic of a corner is its internal angle, and here at Essex we have shown [Kanwal, Bostanci and Clark, 2014] that this can be found from the values calculated as part of the Harris and Stephens corner detector. This forms the basis of a way of *describing* and *matching* corners — and our technique is able to do so for thousands of corners at video rate on modest hardware. However, our technique does not cope well with changes of orientation or scale, such as when a robot is moving rapidly towards an obstacle. For real-time vision, the current state of the art are the BRIEF [Calonder et al., 2010] and ORB ("oriented BRIEF") [Rublee et al., 2011] descriptors used with a conventional corner detector.

The way in which BRIEF works is surprisingly simple. Given a patch of size $S \times S$ encompassing a corner in the image I, some N locations (x_i, y_i) in the patch are chosen and a series of 'tests' are made for different combinations of i and j:

$$\tau = \begin{cases} 1 & \text{if } I(x_i, y_i) < I(x_j, y_j) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Each of these comparisons yields a single binary number, so it is easy to combine these into a string of bits — typical values of N are 128, 256 and 512. (You will see when we consider machine learning techniques that there are similarities between this approach and the one used by the WISARD 'neural network.') Different approaches to choosing the N locations in the patch encompassing the corner yield descriptors with slightly different performances; it seems that selecting the positions randomly is as good a scheme as any other, though one has to be careful to ensure the random number generator is re-seeded correctly whenever a descriptor is calculated.

Comparing the bit-strings from different patches is done by calculating the *Hamming distance*, the number of positions for which the bit-strings differ, and this can be done using the exclusive-OR (XOR) operator — which is why BRIEF descriptors can be matched at video rates on modern hardware.

The problem with BRIEF is that it is affected badly by rotations in the image — the randomly-chosen places at which the comparisons take place do not lie in similar parts of the rotated corner. This is circumvented to some extent in ORB: it positions the corner in the centre of the patch, then computes the intensity-weighted centroid of the patch. The direction of the line from this corner point to the centroid gives the orientation.

To improve the rotation invariance, moments are computed at x and y positions which lie in a circular region of radius r, where $2r < S$ so that the circle lies within the patch, starting from the line from the corner to the centroid. Matching of ORB descriptors is also done by XOR.

Both BRIEF and ORB are available within OpenCV, but rather than being part of the core library they are part of the contributed code (don't ask me why!). This means that they are not necessarily available in all OpenCV installations; and in particular, they are not currently available in our software labs.

7 SIFT and related techniques

David Lowe's *Scale-Invariant Feature Transform* (SIFT) [Lowe, 2004] detector actually preceded the development of the corner descriptors described above. SIFT was the first workable operator that was able to calculate and match consistent features that were reasonably *independent of scale and orientation* — in fact, it quickly became the *de facto* standard for feature detection and is still the yardstick against which we measure the performance of other feature detectors.

SIFT finds characteristic features that are independent of scale and orientation and, for each of them, calculates a descriptor of (typically) 128 floating-point numbers. There may be several thousand such features in an image. Comparing objects in two images then comes down to finding groups of similar descriptors that have all undergone the same transformation. An example of matching SIFT features is shown in Figure 8.

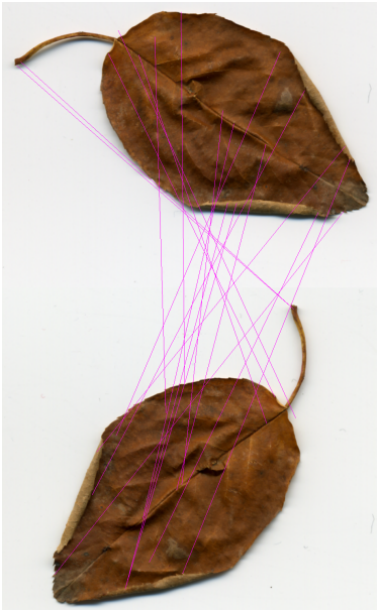


Figure 8: An example of feature matching using SIFT

How can one compare a pair of SIFT features, **A** and **B**? Several ways are in common use:

Euclidean distance: This is practically the same as the sum-squared difference we saw in earlier:

$$E = \sqrt{\sum_i (A_i - B_i)^2} \tag{7}$$

Manhattan distance: This is a quicker-to-calculate alternative to E:

$$M = \sum_i |A_i - B_i| \tag{8}$$

It is easy to show that $E \leq M$ (think of Pythagoras's theorem).

Angle between vectors: If you have experience of the maths of 3D graphics or vector analysis, this approach will make sense. If you think of **A** and **B** as vectors, then their scalar product is given by

$$\mathbf{A}.\mathbf{B} = \sum_i A_i B_i = |\mathbf{A}|\,|\mathbf{B}| \cos\theta$$

where $|\mathbf{A}|$ is the length of A *etc.* and θ is the angle between the vectors. Re-arranging, we obtain

$$\cos\theta = \frac{\sum_i A_i B_i}{|\mathbf{A}|\,|\mathbf{B}|} \tag{9}$$

The last of these alternatives is widely used in information retrieval, and the author's own experience is that it works best on the problems he has worked on.

Hence, to find matches between, say, objects in successive frames of a video, the first step is to calculate the SIFT features for each frame. Then, taking each feature of the first frame in turn, one compares it with *all* the features in the second frame and chooses the one with the lowest score from any of the criteria described above as being the best match. Clearly, the computation time scales as $O(N^2)$, so is slow to perform.

SIFT features can be computed quickly on systems equipped for general-purpose GPU calculations; but there are not at the time of writing good GPU-based implementations of the matching stage, so overall SIFT is quite slow to work with. Even though we have gone to some lengths to make computation as efficient as possible, in some of our research, SIFT feature computation takes several days of number-crunching to perform on GPU-equipped machines. This restricts the size of the problem we are able to work on.

There are a number of successor techniques to SIFT, such as Luc van Gool's SURF (*Speeded-Up Robust Features*) [Bay et al., 2008] which use slightly different approaches that allow the operator to run more quickly. In fact, in our research work, we recently compared about a dozen of these feature detectors [Bostanci, Kanwal and Clark, 2014] and found that Wolfgang Förstner's SPOF [Förstner, Dickscheid and Schindler, 2009] is the most effective overall.

The main problem with SIFT, SURF and most related techniques is that they *avoid* identifying features in the vicinity of corners to ensure they are reproducible when there is motion between frames — but corners are often the features that are of most interest to the vision researcher! For example, in our research into mobile robots and navigation aids for the visually impaired, we really need to know the boundaries of objects so that navigation around them is possible. So where are SIFT *etc*. used? They are extremely good for tasks such as stitching overlapping images into panoramas — if you have used *Photoshop* or related tools to do this, you have probably used SIFT — tracking features between images, and so on. They also form the mainstay of so-called "structure from motion" techniques, which allow 3D reconstructions to be calculated from (large numbers of) photographs.

References

Bay, Herbert et al. (2008). "SURF: Speeded Up Robust Features". In: *Computer Vision and Image Understanding* 110.3, pp. 346–359.

Bostanci, G. E., N. Kanwal and A. F. Clark (2014). "Spatial Statistics for Image Features for Performance Comparison". In: *IEEE Transactions on Image Processing* 23.1, pp. 153–162.

Calonder, Michael et al. (2010). "BRIEF: Binary Robust Independent Elementary Features". In: *Proceedings of the 11th European Conference on Computer Vision*, pp. 778–792.

Canny, John F. (1986). "A computational approach to edge detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.6, pp. 679–698.

Förstner, W., T. Dickscheid and F. Schindler (2009). "Detecting interpretable and accurate scale-invariant keypoints". In: *Proceedings of the 12th International Conference on Computer Vision*, pp. 2256–2263.

Harris, C. and M. Stephens (1988). "A combined corner and edge detector". In: *Proceedings of the 4th Alvey Vision Conference*. http://www.bmva.org/bmvc/1988/avc-88-023.pdf, pp. 147–151.

Kanwal, N., G. E. Bostanci and A. F. Clark (2014). "Matching Corners Using the Informative Arc". In: *IET Proceedings on Computer Vision* 8.3, pp. 245–253.

Lowe, David (2004). "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2. http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf, pp. 91–110.

Rosten, Edward, Reid Porter and Tom Drummond (2010). "Faster and better: A machine learning approach to corner detection". In: *IEEE Trans. Pattern Analysis and Machine Intelligence* 32, pp. 105–119.

Rublee, Ethan et al. (2011). "ORB: An Efficient Alternative to SIFT or SURF". In: *Proceedings of the 2011 International Conference on Computer Vision*, pp. 2564–2571.

Smith, Stephen M. and J. Michael Brady (1997). "SUSAN—A New Approach to Low Level Image Processing". In: *Int. J. Comput. Vision* 23 (1), pp. 45–78.