CHAPTER 5

Local Image Features

An object is separated from its background in an image by an *occluding contour*. Draw a path in the image that crosses such a contour. On one side, pixels lie on the object, and on the other, the background. Finding occluding contours is an important challenge, because the outline of an object—which is one cue to its shape—is formed by occluding contours. We can expect that, at occluding contours, there are often substantial changes in image brightness. There are other important causes of sharp changes in image brightness, including sharp changes in albedo, in surface orientation, or in illumination. Each can provide interesting information about the objects in the world. Occluding contours carry shape information; sharp changes in albedo carry texture information; sharp changes in surface orientation tell us about shape; and illumination changes might tell us where the sun is. All this means it is useful to find and reason about sharp changes in image intensity.

Sharp changes in brightness cause large image gradients. Section 5.1 describes methods to extract image gradients. One important use of gradients is to find *edges* or **edge points**, where the brightness changes particularly sharply (Section 5.2.1). The edge points produced tend to be sensitive to changes in contrast (i.e., the size of the difference in brightness across the edge), which can result from changes in lighting. Often, it is helpful to use the orientation of the gradient vector (Section 5.2.2), which does not depend on contrast. For example, at corners, the image gradient vector swings sharply in orientation.

Corners are important, because they are easy to match from image to image. At a corner, we expect to see strong image gradients that turn fast locally, and this cue yields a corner detector (Section 5.3). If we can describe a neighborhood around a corner, we can match descriptions across images. Such matching is an important basic subroutine in computer vision. Applications include: estimating a homography that will cause images to overlap (and so form a mosaic), Section 12.1.3; estimating the fundamental matrix, Section 7.1; reconstructing points in 3D from multiple views, Section 8.2.3; registering a 3D model with one or more images, Chapter 19. We must first find a natural size for a neighborhood around a corner, which we do by looking for the blob that best describes the local gray levels (Section 5.3.2). Once we have that neighborhood, there are two natural constructions that build representations of the orientation field in the neighborhood; the resulting features yield very well-behaved matchers (Section 5.4).

5.1 COMPUTING THE IMAGE GRADIENT

For an image $\mathcal{I}$, the gradient is

$$\nabla \mathcal{I} = (\frac{\partial \mathcal{I}}{\partial x}, \frac{\partial \mathcal{I}}{\partial y})^T,$$

which we could estimate by observing that

$$\frac{\partial \mathcal{I}}{\partial x} = \lim_{\delta x \to 0} \frac{\mathcal{I}(x + \delta x, y) - \mathcal{I}(x, y)}{\delta x} \approx \mathcal{I}_{i+1,j} - \mathcal{I}_{i,j}.$$

By the same argument, $\partial \mathcal{I}/\partial y \approx \mathcal{I}_{i,j+1} - \mathcal{I}_{i,j}$. These kinds of derivative estimates are known as *finite differences*. Image noise tends to result in pixels not looking like their neighbors, so that simple finite differences tend to give strong responses to noise. As a result, just taking one finite difference for x and one for y gives noisy gradient estimates. The way to deal with this problem is to smooth the image and then differentiate it (we could also smooth the derivative).

The most usual noise model is the *additive stationary Gaussian noise* model, where each pixel has added to it a value chosen independently from the same Gaussian probability distribution. This distribution almost always has zero mean. The standard deviation is a parameter of the model. The model is intended to describe thermal noise in cameras and is illustrated in Figure 5.1.

Smoothing works because, in general, any image gradient of significance to us has effects over a pool of pixels. For example, the contour of an object can result in a long chain of points where the image derivative is large. As another example, a corner typically involves many tens of pixels. If the noise at each pixel is independent and additive, then large image derivatives caused by noise are a local event. Smoothing the image before we differentiate will tend to suppress noise at the scale of individual pixels, because it will tend to make pixels look like their neighbors. However, gradients that are supported by evidence over multiple pixels will tend not to be smoothed out. This suggests differentiating a smoothed image (Figure 5.2).

5.1.1 Derivative of Gaussian Filters

Smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel. This fact is most easily seen by thinking about continuous convolution.

First, differentiation is linear and shift invariant. This means that there is some kernel—we dodge the question of what it looks like—that differentiates. That is, given a function $I(x, y)$,

$$\frac{\partial I}{\partial x} = K_{(\partial/\partial x)} * * I.$$

Now we want the derivative of a smoothed function. We write the convolution kernel for the smoothing as S. Recalling that convolution is associative, we have

$$(K_{(\partial/\partial x)} * * (S * * I)) = (K_{(\partial/\partial x)} * * S) * * I = (\frac{\partial S}{\partial x}) * * I.$$

This fact appears in its most commonly used form when the smoothing function is a Gaussian; we can then write

$$\frac{\partial \left(G_\sigma * * I\right)}{\partial x} = (\frac{\partial G_\sigma}{\partial x}) * * I,$$

that is, we need only convolve with the derivative of the Gaussian, rather than convolve and then differentiate. As discussed in Section 4.5, smoothed derivative

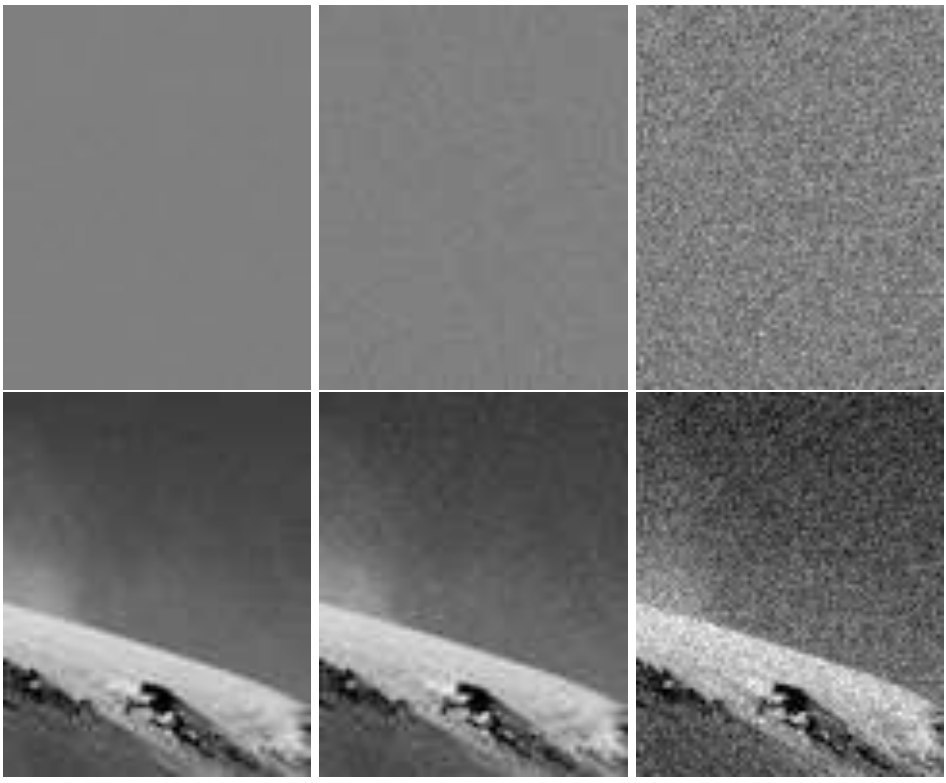

FIGURE 5.1: The **top row** shows three realizations of a stationary additive Gaussian noise process. We have added half the range of brightnesses to these images to show both negative and positive values of noise. From left to right, the noise has standard deviation 1/256, 4/256, and 16/256 of the full range of brightness, respectively. This corresponds roughly to bits zero, two, and five of a camera that has an output range of eight bits per pixel. The **lower row** shows this noise added to an image. In each case, values below zero or above the full range have been adjusted to zero or the maximum value accordingly.

filters look like the effects they are intended to detect. The x-derivative filters look like a vertical light blob next to a vertical dark blob (an arrangement where there is a large x-derivative), and so on (Figure 4.15). Smoothing results in much smaller noise responses from the derivative estimates (Figure 5.2).

The choice of σ used in estimating the derivative is often called the *scale* of the smoothing. Scale has a substantial effect on the response of a derivative filter. Assume we have a narrow bar on a constant background, rather like the zebra's whisker. Smoothing on a scale smaller than the width of the bar means that the filter responds on each side of the bar, and we are able to resolve the rising and falling edges of the bar. If the filter width is much greater, the bar is smoothed into the background and the bar generates little or no response (Figure 5.3).

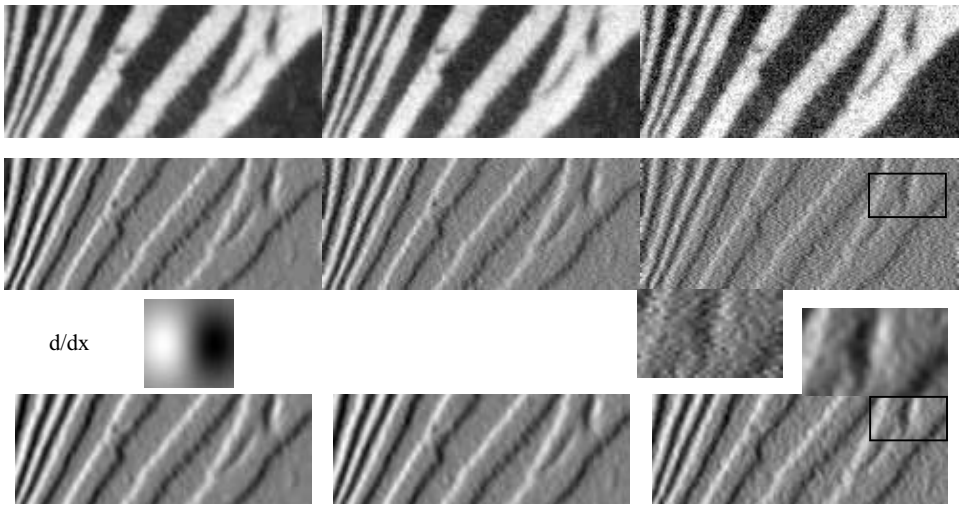

FIGURE 5.2: Derivative of Gaussian filters are less extroverted in their response to noise than finite difference filters. The image at **top left** shows a detail from a picture of a zebra; **top center** shows the same image corrupted by zero mean stationary additive Gaussian noise, with $\sigma = 0.03$ (pixel values range from 0 to 1). **Top right** shows the same image corrupted by zero mean stationary additive Gaussian noise, with $\sigma = 0.09$. The second row shows the finite difference in the x-direction of each image. These images are scaled so that zero is mid-gray, the most negative pixel is dark, and the most positive pixel is light; we used a different scaling for each image. Notice how the noise results in occasional strong derivatives, shown by a graininess in the derivative maps for the noisy images. The final row shows the partial derivative in the x-direction of each image, in each case estimated by a derivative of Gaussian filter with σ one pixel. Again, these images are scaled so that zero is mid-gray, the most negative pixel is dark, and the most positive pixel is light; we used a different scaling for each image. The images are smaller than the input image, because we used a 13×13 pixel discrete kernel. This means that the six rows (resp. columns) on the top and bottom of the image (resp. left and right) cannot be evaluated exactly, because for these rows the kernel covers some points outside the image; we have omitted these values. Notice how the smoothing helps reduce the impact of the noise; this is emphasized by the detail images (between the second and final row), which are doubled in size. The details show patches that correspond from the finite difference image and the smoothed derivative estimate. We show a derivative of Gaussian filter kernel, which (as we expect) looks like the structure it is supposed to find. This is not to scale (it'd be extremely small if it were).

5.2 REPRESENTING THE IMAGE GRADIENT

There are two important representations of the image gradient. The first is to compute edges, where there are very fast changes in brightness. These are usually seen as points where the magnitude of the gradient is extremal (Section 5.2.1). The second is to use gradient orientations, which are largely independent of illumination intensity (Section 5.7).

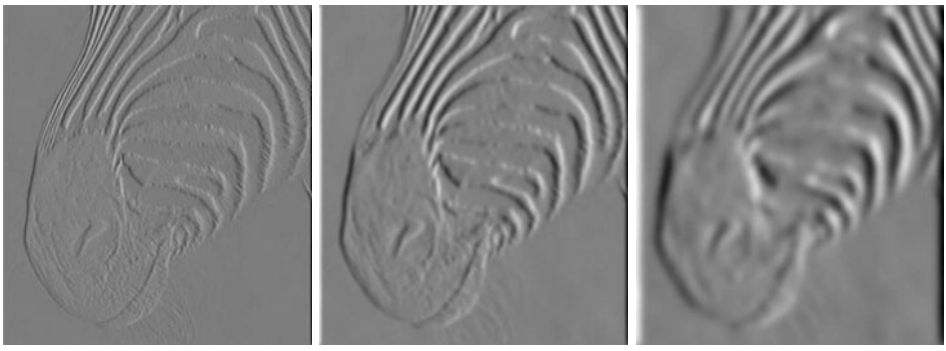

FIGURE 5.3: The scale (i.e., σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra obtained using a derivative of Gaussian filter with σ one pixel, three pixels, and seven pixels (**left** to **right**). Note how images at a finer scale show some hair, the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.

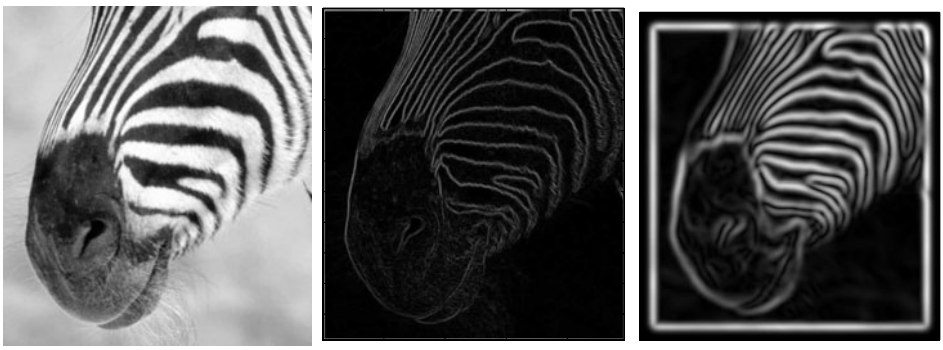

FIGURE 5.4: The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure of a zebra at different scales. At the **center**, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 1$ pixel; and on the **right**, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 2$ pixel. Notice that large values of the gradient magnitude form thick trails.

5.2.1 Gradient-Based Edge Detectors

We think of sharp changes in image intensity as lying on curves in the image, which are known as *edges*; the curves are made up of *edge points*. Many effects can cause edges; worse, each effect that can cause an edge is not guaranteed to cause an edge. For example, an object may happen to be the same intensity as the background, and so the occluding contour will not result in an edge. This means that interpreting edge points can be very difficult. Nonetheless, they are worth finding.

In the most common method for finding edges, we start by computing an

```
Form an estimate of the image gradient
Compute the gradient magnitude
While there are points with high gradient
magnitude that have not been visited
  Find a start point that is a local maximum in the
    direction perpendicular to the gradient
    erasing points that have been checked
  While possible, expand a chain through
    the current point by:
      1) predicting a set of next points, using
           the direction perpendicular to the gradient
      2) finding which (if any) is a local maximum
           in the gradient direction
      3) testing if the gradient magnitude at the
           maximum is sufficiently large
      4) leaving a record that the point and
           neighbors have been visited
      record the next point, which becomes the current point
  end
end
```

Algorithm 5.1: Gradient-Based Edge Detection.

estimate of the gradient magnitude. The gradient magnitude is large along a thick trail in the image (Figure 5.4), but occluding contours are curves, so we must obtain a curve of the most distinctive points on this trail.

There is clearly no objective definition, and we can proceed by reasonable intuition. The gradient magnitude can be thought of as a chain of low hills. Marking local maxima would mark isolated points—the hilltops in the analogy. A better criterion is to slice the gradient magnitude along the gradient direction, which should be perpendicular to the edge, and mark the points along the slice where the magnitude is maximal. This would get a chain of points along the crown of the hills in our chain. Each point in the chain can be used to predict the location of the next point, which will be in a direction roughly at right angles to the gradient at the edge point (Figure 5.5). Forming these chains is called *nonmaximum suppression*. It is relatively straightforward to identify the location of these chains at a resolution finer than that of the pixel grid (Figure 5.5).

There are too many of these chains to come close to being a reasonable representation of object boundaries. In part, this is because we have marked maxima of the gradient magnitude without regard to how large these maxima are. It is more usual to apply a threshold test to ensure that the maxima are greater than some lower bound. This in turn leads to broken edge curves. The usual trick for dealing with this is to use *hysteresis*; we have two thresholds and refer to the *larger* when starting an edge chain and the *smaller* while following it. The trick often results in an improvement in edge outputs. These considerations yield Algorithm 5.1. Most

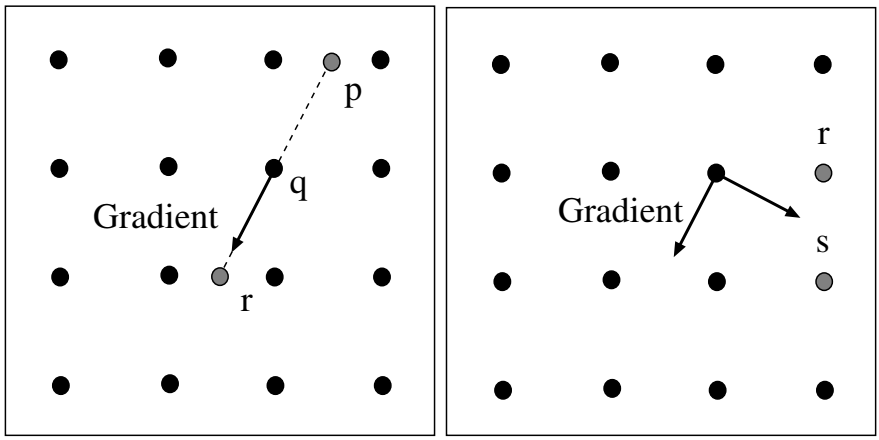

FIGURE 5.5: Nonmaximum suppression obtains points where the gradient magnitude is at a maximum *along the direction of the gradient*. The figure on the **left** shows how we reconstruct the gradient magnitude. The dots are the pixel grid. We are at pixel $\boldsymbol{q}$, attempting to determine whether the gradient is at a maximum; the gradient direction through $\boldsymbol{q}$ does not pass through any convenient pixels in the forward or backward direction, so we must interpolate to obtain the values of the gradient magnitude at $\boldsymbol{p}$ and $\boldsymbol{r}$. If the value at $\boldsymbol{q}$ is larger than both, $\boldsymbol{q}$ is an edge point. Typically, the magnitude values are reconstructed with a linear interpolate, which in this case would use the pixels to the left and right of $\boldsymbol{p}$ and $\boldsymbol{r}$, respectively, to interpolate values at those points. On the **right**, we sketch how to find candidates for the next edge point given that $\boldsymbol{q}$ is an edge point; an appropriate search direction is perpendicular to the gradient, so that points $\boldsymbol{s}$ and $\boldsymbol{t}$ should be considered for the next edge point. Notice that, in principle, we don't need to restrict ourselves to pixel points on the image grid, because we know where the predicted position lies between $\boldsymbol{s}$ and $\boldsymbol{t}$. Hence, we could again interpolate to obtain gradient values for points off the grid.

current edgefinders follow these lines.

5.2.2 Orientations

As the light gets brighter or darker (or as the camera aperture opens or closes), the image will get brighter or darker, which we can represent as a scaling of the image value. The image $\mathcal{I}$ will be replaced with $s\mathcal{I}$ for some value s. The magnitude of the gradient scales with the image, i.e., $\|\nabla\mathcal{I}\|$ will be replaced with $s\|\nabla\mathcal{I}\|$. This creates problems for edge detectors, because edge points may appear and disappear as the image gradient values go above and below thresholds with the scaling. One solution is to represent the *orientation* of image gradient, which is unaffected by scaling (Figure 5.7). The gradient orientation field depends on the smoothing scale at which the gradient was computed. Orientation fields can be quite characteristic of particular textures (Figure 5.9), and we will use this important property to come up with more complex features below.

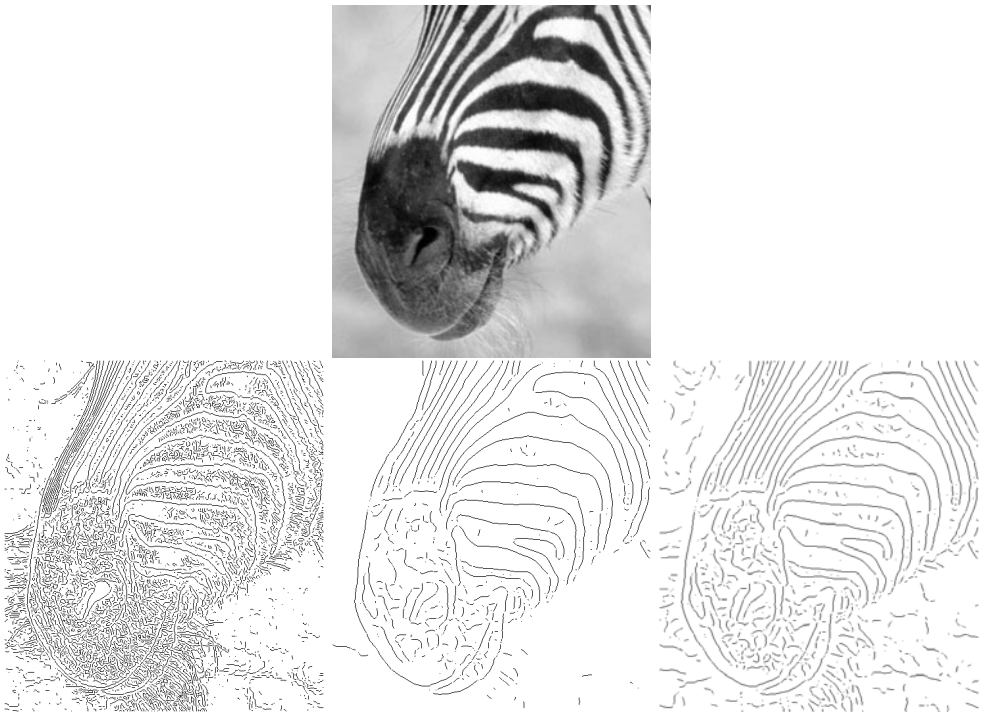

FIGURE 5.6: Edge points marked on the pixel grid for the image shown on the **top**. The edge points on the **left** are obtained using a Gaussian smoothing filter at σ one pixel, and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point. The edge points at the **center** are obtained using a Gaussian smoothing filter at σ four pixels, and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point. The edge points on the **right** are obtained using a Gaussian smoothing filter at σ four pixels, and gradient magnitude has been tested against a low threshold to determine whether a point is an edge point. At a fine scale, fine detail at high contrast generates edge points, which disappear at the coarser scale. When the threshold is high, curves of edge points are often broken because the gradient magnitude dips below the threshold; for the low threshold, a variety of new edge points of dubious significance are introduced.

5.3 FINDING CORNERS AND BUILDING NEIGHBORHOODS

Points worth matching are corners, because a corner can be *localized*, which means we can tell where a corner is. This motivates the more general term *interest point* often used to describe a corner. In this view, corners are interesting because we can tell where they are. Place a small window over a patch of constant image value. If you translate the window in any direction, the image in the window will not change significantly. This means you cannot give a reliable estimate of the location of the window from its gray levels. Similarly, if you translate a window up and down an edge, the image in the window doesn't change, so you cannot estimate location along the edge (this observation used to be known as the *aperture problem*). But

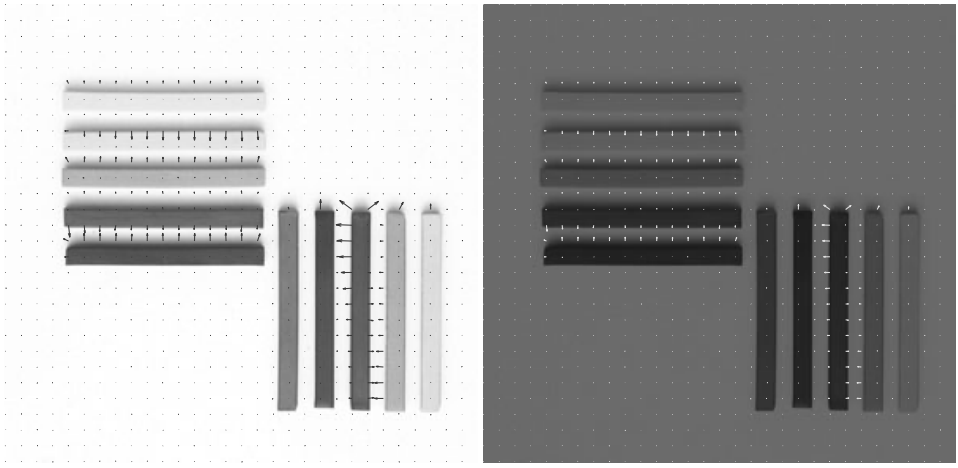

FIGURE 5.7: The magnitude of the image gradient changes when one increases or decreases the intensity. The orientation of the image gradient does not change; we have plotted every 10th orientation arrow, to make the figure easier to read. Note how the directions of the gradient arrows are fixed, whereas the size changes. *Philip Gatward © Dorling Kindersley, used with permission.*

with a corner, any movement of the window changes the image in the window (i.e., the patch of image around the corner is not *self-similar*), so you can estimate the location of the corner. Corners are not the only type of local image structure with this property (Section 5.3.2)

There are many ways of representing a neighborhood around an interesting corner. Methods vary depending on what might happen to the neighborhood. In what follows, we will assume that neighborhoods are only translated, rotated, and scaled (rather than, say, subjected to an affine or projective transformation), and so without loss of generality we can assume that the patches are circular. We must estimate the radius of this circle. There is technical machinery available for the neighborhoods that result from more complex transformations, but it is more intricate; see Section 5.6.

5.3.1 Finding Corners

One way to find corners is to find edges, and then walk the edges looking for a corner. This approach can work poorly, because edge detectors often fail at corners. At sharp corners or unfortunately oriented corners, gradient estimates are poor because the smoothing region covers the corner.

At a corner, we expect two important effects. First, there should be large gradients. Second, in a small neighborhood, the gradient orientation should swing sharply. We can identify corners by looking at variations in orientation within a

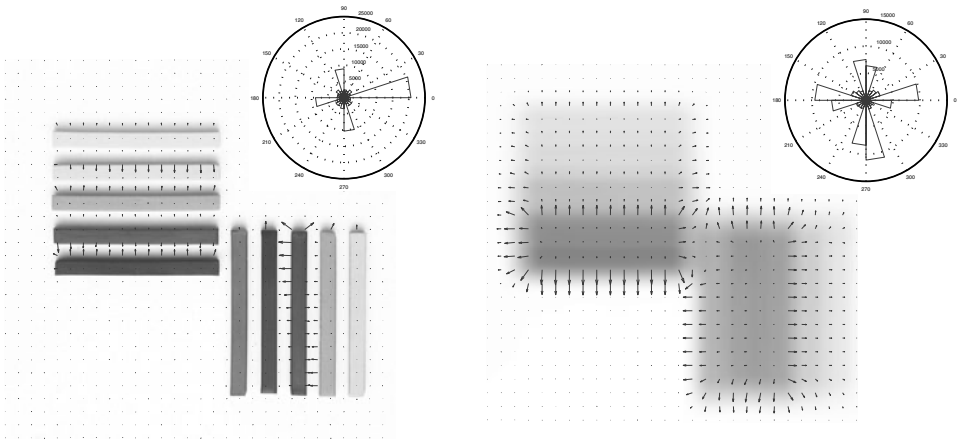

FIGURE 5.8: The scale at which one takes the gradient affects the orientation field. We show the overall trend of the orientation field by plotting a rose plot, where the size of a wedge represents the relative frequency of that range of orientations. **Left** shows an image of artists pastels at a fairly fine scale; here the edges are sharp, and so only a small set of orientations occurs. In the heavily smoothed version on the **right**, all edges are blurred and corners become smooth and blobby; as a result, more orientations appear in the rose plot. *Philip Gatward © Dorling Kindersley, used with permission.*

window. In particular, the matrix

$$\begin{aligned}\mathcal{H} &= \sum_{window} \left\{(\nabla I)(\nabla I)^T\right\} \\ &\approx \sum_{window} \left\{ \begin{array}{ll} (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial x} * *\mathcal{I}) & (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) \\ (\frac{\partial G_\sigma}{\partial x} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) & (\frac{\partial G_\sigma}{\partial y} * *\mathcal{I})(\frac{\partial G_\sigma}{\partial y} * *\mathcal{I}) \end{array} \right\}\end{aligned}$$

gives a good idea of the behavior of the orientation in a window. In a window of constant gray level, both eigenvalues of this matrix are small because all the terms are small. In an edge window, we expect to see one large eigenvalue associated with gradients at the edge and one small eigenvalue because few gradients run in other directions. But in a corner window, both eigenvalues should be large.

The *Harris corner detector* looks for local maxima of

$$\det(\mathcal{H}) - k(\frac{\text{trace}(\mathcal{H})}{2})^2$$

where k is some constant (Harris and Stephens 1988); we used 0.5 for Figure 5.10. These local maxima are then tested against a threshold. This tests whether the product of the eigenvalues (which is $\det(\mathcal{H})$) is larger than the square of the average (which is $(\text{trace}(\mathcal{H})/2)^2$). Large, locally maximal values of this test function imply the eigenvalues are both big, which is what we want. Figure 5.10 illustrates corners found with the Harris detector. This detector is unaffected by translation and rotation (Figure 5.11).

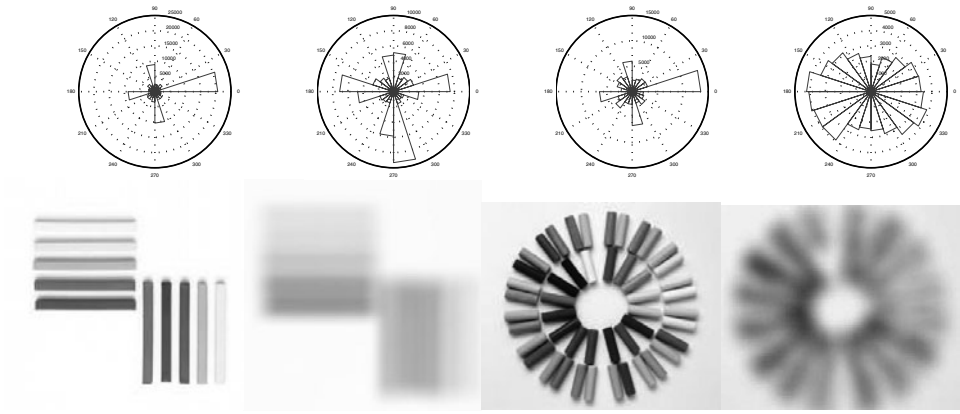

FIGURE 5.9: Different patterns have quite different orientation histograms. The **left** shows rose plots and images for a picture of artists pastels at two different scales; the **right** shows rose plots and images for a set of pastels arranged into a circular pattern. Notice how the pattern of orientations at a particular scale, and also the changes across scales, are quite different for these two very different patterns. *Philip Gatward © Dorling Kindersley, used with permission.*

5.3.2 Using Scale and Orientation to Build a Neighborhood

To turn a corner into an image neighborhood, we must estimate the radius of the circular patch (equivalently, its scale). The radius estimate should get larger proportionally when the image gets bigger. For example, in a 2x scaled version of the original image, our method should double its estimate of the patch radius. This property helps choose a method. We could center a blob of fixed appearance (say, dark on a light background) on the corner, and then choose the scale to be the radius of the best fitting blob. An efficient way to do this is to use a Laplacian of Gaussian filter.

The *Laplacian* of a function in 2D is defined as

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

It is natural to smooth the image before applying a Laplacian. Notice that the Laplacian is a linear operator (if you're not sure about this, you should check), meaning that we could represent taking the Laplacian as convolving the image with some kernel (which we write as K_{∇^2}). Because convolution is associative, we have that

$$(K_{\nabla^2} * *(G_\sigma * *I)) = (K_{\nabla^2} * *G_\sigma) * *I = (\nabla^2 G_\sigma) * *I.$$

The reason this is important is that, just as for first derivatives, smoothing an image and then applying the Laplacian is the same as convolving the image with the Laplacian of the kernel used for smoothing. Figure 5.12 shows the resulting kernel for Gaussian smoothing; notice that this looks like a dark blob on a light background.

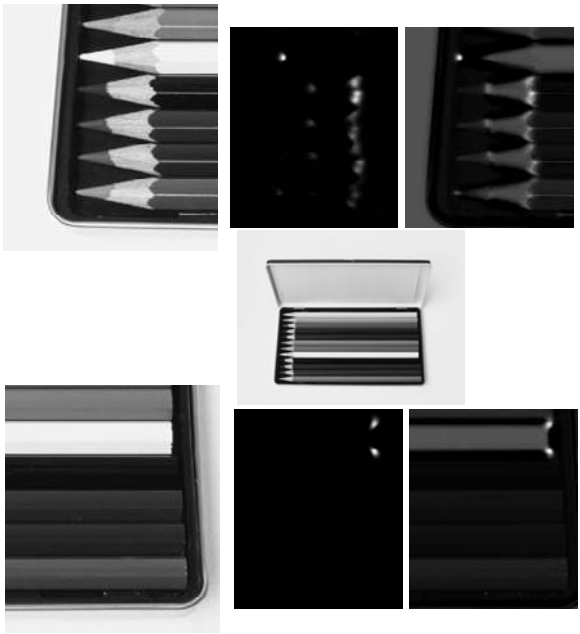

FIGURE 5.10: The response of the Harris corner detector visualized for two detail regions of an image of a box of colored pencils (**center**). **Top left**, a detail from the pencil points; **top center**, the response of the Harris corner detector, where more positive values are lighter. The **top right** shows these overlaid on the original image. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). Note that the detector is affected by contrast, so that, for example, the point of the mid-gray pencil at the top of this figure generates a very strong corner response, but the points of the darker pencils do not, because they have little contrast with the tray. For the darker pencils, the strong, contrasty corners occur where the lead of the pencil meets the wood. The **bottom** sequence shows corners for a detail of pencil ends. Notice that responses are quite local, and there are a relatively small number of very strong corners. *Steve Gorton © Dorling Kindersley, used with permission.*

Imagine applying a smoothed Laplacian operator to the image at the center of the patch. Write $\mathcal{I}$ for the image, ∇^2_σ for the smoothed Laplacian operator with smoothing constant σ, $\uparrow_k \mathcal{I}$ for the the image with size scaled by k, (x_c, y_c) for the coordinates of the patch center, and (x_{kc}, y_{kc}) for the coordinates of the patch center in the scaled image. Assume that upscaling is perfect, and there are no effects resulting from the image grid. This is fair because effects will be small for the scales of interest for us. Then, we have

$$(\nabla^2_{k\sigma} \uparrow_k \mathcal{I})(x_c, y_c) = (\nabla^2_\sigma \mathcal{I})(x_{kc}, y_{kc})$$

(this is most easily demonstrated by reasoning about the image as a continuous function, the operator as a convolution, and then using the change of variables formula for integrals). Now choose a radius r for the circular patch centered at

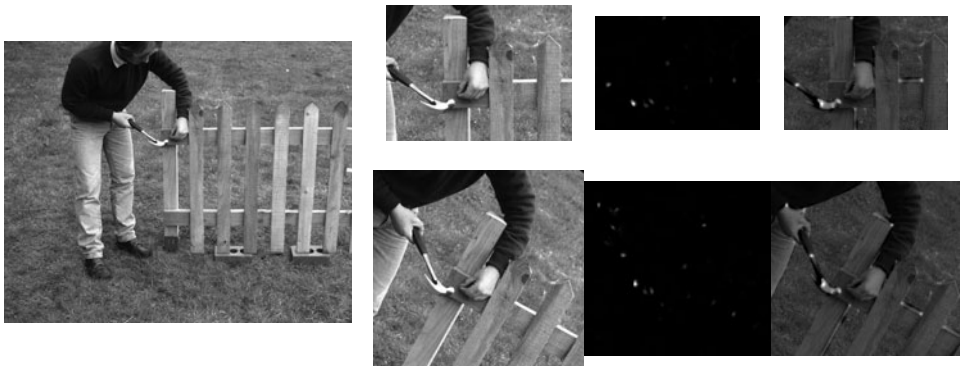

FIGURE 5.11: The response of the Harris corner detector is unaffected by rotation and translation. The **top row** shows the response of the detector on a detail of the image on the **far left**. The **bottom row** shows the response of the detector on a corresponding detail from a rotated version of the image. For each row, we show the detail window (**left**); the response of the Harris corner detector, where more positive values are lighter (**center**); and the responses overlaid on the image (**right**). Notice that responses are quite local, and there are a relatively small number of very strong corners. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). The arm and hammer in the top row match those in the bottom row; notice how well the maps of Harris corner detector responses match, too. © *Dorling Kindersley, used with permission.*

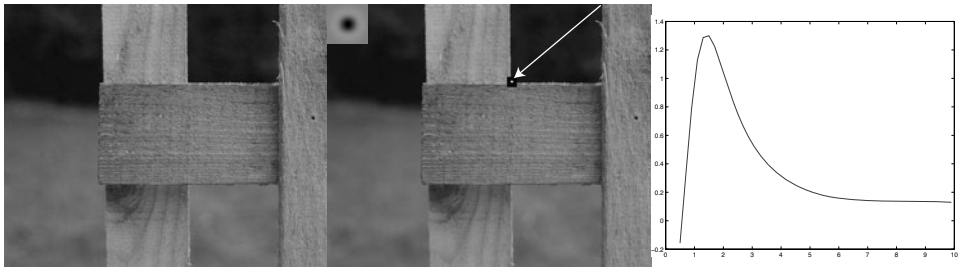

FIGURE 5.12: The scale of a neighborhood around a corner can be estimated by finding a local extremum, in *scale* of the response at that point to a smoothed Laplacian of Gaussian kernel. On the **left**, a detail of a piece of fencing. In the center, a corner identified by an arrow (which points to the corner, given by a white spot surrounded by a black ring). *Overlaid* on this image is a Laplacian of Gaussian kernel, in the **top right** corner; dark values are negative, mid gray is zero, and light values are positive. Notice that, using the reasoning of Section 4.5, this filter will give a strong positive response for a dark blob on a light background, and a strong negative response for a light blob on a dark background, so by searching for the strongest response at this point as a function of scale, we are looking for the size of the best-fitting blob. On the **right**, the response of a Laplacian of Gaussian *at the location of the corner*, as a function of the smoothing parameter (which is plotted in pixels). There is one extremal scale, at approximately 2 pixels. This means that there is one scale at which the image neighborhood looks most like a blob (some corners have more than one scale). © *Dorling Kindersley, used with permission.*

Assume a fixed scale parameter k
Apply a corner detector to the image $\mathcal{I}$
Initialize a list of patches
For each corner detected
  Write (x_c, y_c) for the location of the corner
  Compute the radius r for the patch at (x_c, y_c) as
    $r(x_c, y_c) = \underset{\sigma}{\text{argmax}} \ \nabla^2_\sigma \mathcal{I}(x_c, y_c)$
    by computing $\nabla^2_\sigma \mathcal{I}(x_c, y_c)$ for a variety of values of σ,
    interpolating these values, and maximizing
  Compute an orientation histogram $H(\theta)$ for gradient orientations within
    a radius kr of (x_c, y_c).
  Compute the orientation of the patch θ_p as
    $\theta_p = \underset{\theta}{\text{argmax}} \ H(\theta)$. If there is more than
    one theta that maximizes this histogram, make one copy of the
    patch for each.
  Attach (x_c, y_c, r, θ_p) to the list of patches for each copy

Algorithm 5.2: Obtaining Location, Radius and Orientation of Pattern Elements Using a Corner Detector.

(x_c, y_c), such that

$$r(x_c, y_c) = \underset{\sigma}{\text{argmax}} \ \nabla^2_\sigma \mathcal{I}(x_c, y_c)$$

(Figure 5.12). If the image is scaled by k, then this value of r will be scaled by k too, which is the property we wanted. This procedure looks for the scale of the best approximating blob. Notice that a Gaussian pyramid could be helpful here; we could apply the same smoothed Laplacian operator to different levels of a pyramid to get estimates of the scale.

We can generalize this method, too, to detect interest points. Write $(\boldsymbol{x}, \sigma)$ for a triple consisting of a point and a scale around that point. We would like to detect such triples in a way that (a) when the image is translated, the triples translate, too and (b) when the image is scaled, the triples scale. This can be given a formal meaning. If $\mathcal{I}'(\boldsymbol{x}) = \mathcal{I}(\lambda\boldsymbol{x} + \boldsymbol{c})$ is a scaled and translated image, then for each point $(\boldsymbol{x}, \sigma)$ in the list of neighborhoods for $\mathcal{I}$, we want to have $(\lambda\boldsymbol{x} + \boldsymbol{c}, \lambda\sigma)$ in the list of neighborhoods for $\mathcal{I}'$. This property is referred to as *covariance* (although the term invariance is widely but incorrectly used).

We have already established that, at a particular point (given by our corner detector), we get a covariant scale estimate by choosing the local maximum *in scale* of the response of the Laplacian of Gaussian. We can build an interest point detector directly out of a Laplacian of Gaussian, by identifying local extrema *in position and scale* of the operator (if this looks slow to you, keep in mind that a Gaussian pyramid could speed up the process). Each such extremum is a triple $(\boldsymbol{x}, \sigma)$ with the properties we want. These points are different from the points

Assume a fixed scale parameter k
Find all locations and scales which are local extrema of
  $\nabla^2_\sigma \mathcal{I}(x, y)$ in location (x, y) and scale σ forming a list of triples (x_c, y_c, r)
For each such triple
  Compute an orientation histogram $H(\theta)$ for gradient orientations within
    a radius kr of (x_c, y_c).
  Compute the orientation of the patch θ_p as
    $\theta_p = \underset{\theta}{\text{argmax}}\ H(\theta)$. If there is more than one θ that
    maximizes this histogram, make one copy of the patch for each.
  Attach (x_c, y_c, r, θ_p) to the list of patches for each copy

Algorithm 5.3: Obtaining Location, Radius, and Orientation of Pattern Elements Using the Laplacian of Gaussian.

obtained by using a corner detector and then estimating scale. Corner detectors respond to a corner structure at the point of interest; the Laplacian of Gaussian looks for structures that look like a circular blob of a particular scale centered at the point of interest. Corner detectors tend to produce neighborhoods where the estimate of the center is very accurate, but the scale estimate is poor. These are most useful in matching problems where we don't expect the scale to change much. Laplacian of Gaussian methods produce neighborhoods where the estimate of the center is less accurate, but the scale estimate is better. These are most useful in matching problems where large changes of scale might appear.

As we have seen, orientation histograms are a natural representation of image patches. However, we cannot represent orientations in image coordinates (for example, using the angle to the horizontal image axis), because the patch we are matching to might have been rotated. We need a reference orientation so all angles can be measured with respect to that reference. A natural reference orientation is the most common orientation in the patch. We compute a histogram of the gradient orientations in this patch, and find the largest peak. This peak is the reference orientation for the patch. If there are two or more peaks of the same magnitude, we make multiple copies of the patch, one at each peak orientation. The whole process is summarized in Algorithms 5.2 and 5.3. These estimates of patch neighborhoods are remarkably well behaved (Figure 5.13).

5.4 DESCRIBING NEIGHBORHOODS WITH SIFT AND HOG FEATURES

We know the center, radius, and orientation of a set of an image patch, and must now represent it. Orientations should provide a good representation. They are unaffected by changes in image brightness, and different textures tend to have different orientation fields. The pattern of orientations in different parts of the patch is likely to be quite distinctive. Our representation should be robust to small errors in the center, radius, or orientation of the patch, because we are unlikely to estimate these exactly right.

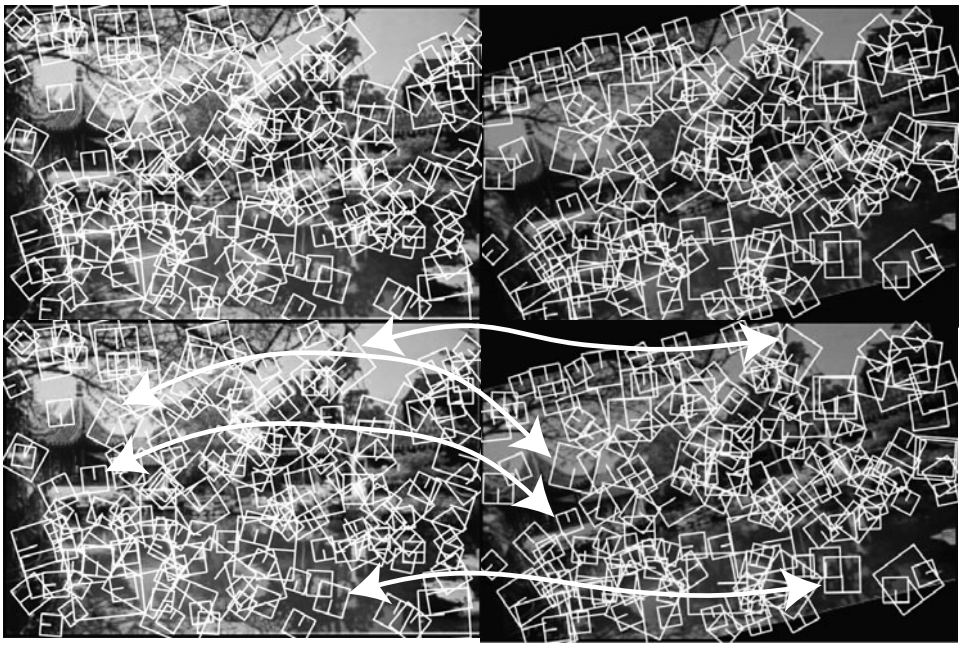

FIGURE 5.13: This figure shows local patches recovered using a method similar to that described in the text (the details of the corner detector were different). These patches are plotted as squares, rather than as circles. The location of the patch is the center of the square. The reference orientation of the patch is given by the line segment in the square, and the scale is the size of the square. The image on the **right** has been scaled, rotated, and translated to produce the image on the **left**. Notice that (a) most of the patches on the right have corresponding patches on the left and (b) the corresponding patches are translated, rotated, and scaled versions of the original patches. You can check this by looking at the grayscale version of the image. We have shown some of the many corresponding pairs of patches (**below**; the large white arrows). *This figure was originally published as Figure 1 of "Object recognition from local scale-invariant features" D.G. Lowe, Proc. IEEE ICCV, 1999* © *IEEE 1999.*

You should think of these neighborhoods as being made up of pattern elements. In this case, the pattern elements will be orientations, but we will use this trick again for other kinds of pattern element. These elements move around somewhat inside the neighborhood (because we might not get the center right), but if most elements are there and are in about the right place, then the neighborhood has the right properties. We must build features that can make it obvious whether the pattern elements are present, and whether they are in about the right place, but are not affected by some rearrangement.

The most obvious approach is to represent the neighborhood with a histogram of the elements that appear there. This will tell us what is present, but it confuses too many patterns with one another. For example, all neighborhoods with vertical stripes will get mixed up, however wide the stripe. The natural approach is to take histograms locally, within subpatches of the neighborhood. This leads to a very

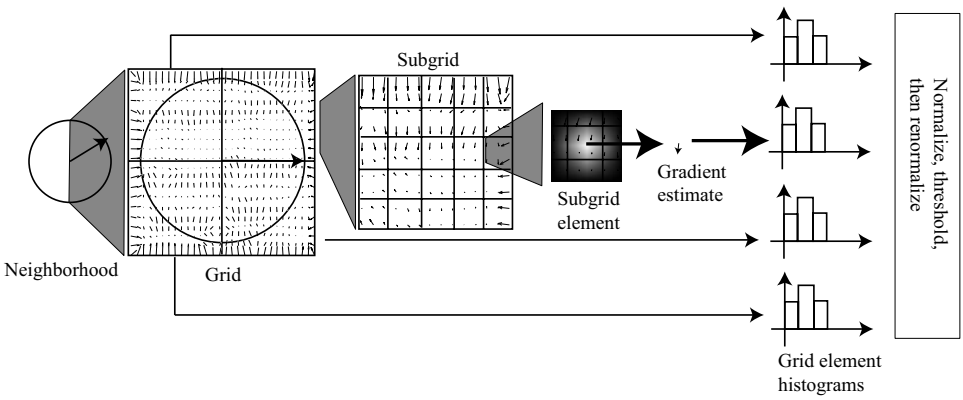

FIGURE 5.14: To construct a SIFT descriptor for a neighborhood, we place a grid over the rectified neighborhood. Each grid is divided into a subgrid, and a gradient estimate is computed at the center of each subgrid element. This gradient estimate is a weighted average of nearby gradients, with weights chosen so that gradients outside the subgrid cell contribute. The gradient estimates in each subgrid element are accumulated into an orientation histogram. Each gradient votes for its orientation, with a vote weighted by its magnitude and by its distance to the center of the neighborhood. The resulting orientation histograms are stacked to give a single feature vector. This is normalized to have unit norm; then terms in the normalized feature vector are thresholded, and the vector is normalized again.

important feature construction.

5.4.1 SIFT Features

We can now compute a representation that is not affected by translation, rotation, or scale. For each patch, we rectify the patch by translating the center to the origin, rotating so the orientation direction lies along (say) the x-axis, and scaling so the radius is one. Any representation we compute for this rectified patch will be invariant to translations, rotations, and scale. Although we do not need to rectify in practice—instead, we can work the rectification into each step of computing the description—it helps to think about computing descriptions for a rectified patch.

A *SIFT descriptor* (for Scale Invariant Feature Transform) is constructed out of image gradients, and uses both magnitude and orientation. The descriptor is normalized to suppress the effects of change in illumination intensity. The descriptor is a set of histograms of image gradients that are then normalized. These histograms expose general spatial trends in the image gradients in the patch but suppress detail. For example, if we estimate the center, scale, or orientation of the patch slightly wrong, then the rectified patch will shift slightly. As a result, simply recording the gradient at each point yields a representation that changes between instances of the patch. A histogram of gradients will be robust to these changes. Rather than histogramming the gradient at a set of sample points, we histogram local averages of image gradients; this helps avoid noise.

The standard SIFT descriptor is obtained by first dividing the rectified patch

into an $n \times n$ grid. We then subdivide each grid element into an $m \times m$ subgrid of subcells. At the center of each subcell, we compute a gradient estimate. The gradient estimate is obtained as a weighted average of gradients around the center of the cell, weighting each by $(1 - d_x/s_x)(1 - d_y/s_y)/N$, where d_x (resp. d_y) is the x (resp. y) distance from the gradient to the center of the subcell, and s_x (resp. s_y) is the x (resp. y) spacing between the subcell centers. This means that gradients make contributions to more than one subcell, so that a small error in the location of the center of the patch leads to a small change in the descriptor.

We now use these gradient estimates to produce histograms. Each grid element has a q-cell orientation histogram. The magnitude of each gradient estimate is accumulated into the histogram cell corresponding to its orientation; the magnitude is weighted by a Gaussian in distance from the center of the patch, using a standard deviation of half the patch.

We concatenate each histogram into a vector of $n \times n \times q$ entries. If the image intensity were doubled, this vector's length would double (because the histogram entries are sums of gradient magnitudes). To avoid this effect, we normalize this vector to have unit length. Very large gradient magnitude estimates tend to be unstable (for example, they might result from a lucky arrangement of surfaces in 3D so that one faces the light directly and another points away from the light). This means that large entries in the normalized vector are untrustworthy. To avoid difficulties with large gradient magnitudes, each value in the normalized vector is thresholded with threshold t, and the resulting vector is renormalized. The whole process is summarized in Algorithm 5.4 and Figure 5.14. Standard parameter values are $n = 4$, $m = 4$, $q = 8$, and $t = 0.2$.

Given an image $\mathcal{I}$, and a patch with center (x_c, y_c),
  radius r, orientation θ, and parameters n, m, q, k and t.
For each element of the $n \times n$ grid centered at (x_c, y_c) with spacing kr
  Compute a weighted q element histogram of the averaged
    gradient samples at each point of the $m \times m$ subgrid,
    as in Algorithm 5.5.
Form an $n \times n \times q$ vector $\boldsymbol{v}$ by concatenating the histograms.
Compute $\boldsymbol{u} = \boldsymbol{v}/\sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}$.
Form $\boldsymbol{w}$ whose i'th element w_i is $\min(u_i, t)$.
The descriptor is $\boldsymbol{d} = \boldsymbol{w}/\sqrt{\boldsymbol{w} \cdot \boldsymbol{w}}$.

Algorithm 5.4: Computing a SIFT Descriptor in a Patch Using Location, Orientation and Scale.

There is now extensive experimental evidence that image patches that match one another will have similar SIFT feature representations, and patches that do not will tend not to. SIFT features can be used to represent the local color pattern around a sample point, too. The natural procedure is to apply SIFT feature code to a color representation. For example, one could compute SIFT features for each of the hue, saturation, and value channels (HSV-SIFT; see Bosch *et al.* (2008)); for the opponent color channels (OpponentSIFT, which uses R-G and B-Y; see van de

Given a grid cell $\mathcal{G}$ for patch with center $\boldsymbol{c} = (x_c, y_c)$ and radius r

Create an orientation histogram
For each point $\boldsymbol{p}$ in an $m \times m$ subgrid spanning $\mathcal{G}$
  Compute a gradient estimate $\nabla\mathcal{I}\,|\boldsymbol{p}$ estimate at $\boldsymbol{p}$
    as a weighted average of $\nabla\mathcal{I}$, using bilinear weights centered at $\boldsymbol{p}$.
  Add a vote with weight $\|\nabla\mathcal{I}\|\frac{1}{r\sqrt{2\pi}}\exp\left(-\frac{\|\boldsymbol{p}-\boldsymbol{c}\|^2}{r^2}\right)$
    to the orientation histogram cell for the orientation of $\nabla\mathcal{I}$.

Algorithm 5.5: Computing a Weighted q Element Histogram for a SIFT Feature.

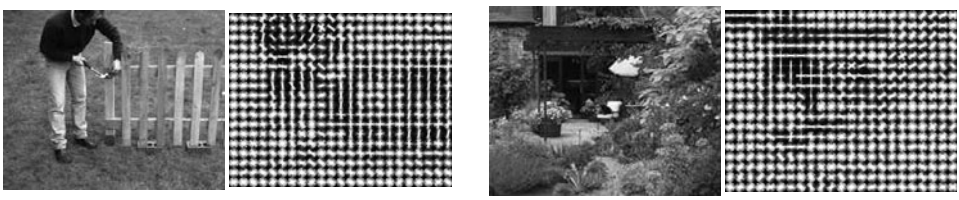

FIGURE 5.15: The HOG features for each the two images shown here have been visualized by a version of the rose diagram of Figures 5.7–5.9. Here each of the cells in which the histogram is taken is plotted with a little rose in it; the direction plotted is at right angles to the gradient, so you should visualize the overlaid line segments as edge directions. Notice that in the textured regions the edge directions are fairly uniformly distributed, but strong contours (the gardener, the fence on the **left**; the vertical edges of the french windows on the **right**) are very clear. This figure was plotted using the toolbox of Dollár and Rabaud. *Left:* © *Dorling Kindersley, used with permission. Right: Geoff Brightling* © *Dorling Kindersley, used with permission.*

Sande *et al.* (2010)); for normalised opponent color channels (C-SIFT, which uses $(R-G)/(R+G+B)$ and $(B-Y)/(R+G+B)$; see Abdel Hakim and Farag (2006); Geusebroek *et al.* (2001); or Burghouts and Geusebroek (2009)); and for normalized color channels (rgSIFT, which uses $R/(R+G+B)$ and $G/(R+G+B)$; see van de Sande *et al.* (2010)). Each of these features will behave slightly differently when the light falling on an object changes, and each can be used in place of, or in addition to, SIFT features.

5.4.2 HOG Features

The *HOG feature* (for Histogram Of Gradient orientations) is an important variant of the SIFT feature. Again, we histogram gradient orientations in cells, but now adjust the process to try and identify high-contrast edges. We can recover contrast information by counting gradient orientations with weights that reflect how significant a gradient is compared to other gradients in the same cell. This means that, rather than normalize gradient contributions over the whole neighborhood, we normalize with respect to nearby gradients only. Normalization could occur on a grid of cells that is different from the orientation subgrid, too. A single gradient

location might contribute to several different histograms, normalized in somewhat different ways; this means we will be relatively unlikely to miss boundaries that have low contrast.

Write $\|\nabla I_{\mathbf{x}}\|$ for the gradient magnitude at point $\mathbf{x}$ in the image. Write $\mathcal{C}$ for the cell whose histogram we wish to compute and $w_{\mathbf{x},\mathcal{C}}$ for the weight that we will use for the orientation at $\mathbf{x}$ for this cell. A natural choice of weight is

$$w_{\mathbf{x},\mathcal{C}} = \frac{\|\nabla I_{\mathbf{x}}\|}{\sum_{\mathbf{u}\in\mathcal{C}} \|\nabla I_{\mathbf{u}}\|}.$$

This compares the gradient magnitude to others in the cell, so that gradients that are large compared to their neighbors get a large weight. This normalization process means that HOG features are quite good at picking outline curves out of confusing backgrounds (Figure 5.15).

5.5 COMPUTING LOCAL FEATURES IN PRACTICE

We have sketched the most important feature constructions, but there is a huge range of variants. Performance is affected by quite detailed questions, such as the extent of smoothing when evaluating orientations. Space doesn't allow a detailed survey of these questions (though there's some material in Section 5.6), and the answers seem to change fairly frequently, too. This means we simply can't supply accurate recipes for building each of these features.

Fortunately, at time of writing, there are several software packages that provide good implementations of each of these feature types, and of other variations. Piotr Dollár and Vincent Rabaud publish a toolbox at `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`; we used this to generate several figures. VLFeat is a comprehensive open-source package that provides SIFT features, vector quantization by a variety of methods, and a variety of other representations. At time of writing, it could be obtained from `http://www.vlfeat.org/`. SIFT features are patented (Lowe 2004), but David Lowe (the inventor) provides a reference object code implementation at `http://www.cs.ubc.ca/~lowe/keypoints/`. Navneet Dalal, one of the authors of the original HOG feature paper, provides an implementation at `http://www.navneetdalal.com/software/`. One variant of SIFT is PCA-SIFT, where one uses principal components to reduce the dimension of the SIFT representation (Ke and Sukthankar 2004). Yan Ke, one of the authors of the original PCA-SIFT paper, provides an implementation at `http://www.cs.cmu.edu/~yke/pcasift/`. Color descriptor code, which computes visual words based on various color SIFT features, is published by van de Sande *et al.* at `http://koen.me/research/colordescriptors/`.

5.6 NOTES

Edges

There is a huge edge detection literature. The earliest paper of which we are aware is Julez (1959) (yes, 1959!). Those wishing to be acquainted with the early literature in detail should start with a 1975 survey by Davis (1975); Herskovits and Binford (1970); Horn (1971); and Hueckel (1971), who models edges and then detects the model. There are many optimality criteria for edge detectors, and rather more

"optimal" edge detectors. The key paper in this literature is by Canny (1986); significant variants are due to Deriche (1987) and to Spacek (1986). Faugeras' textbook contains a detailed and accessible exposition of the main issues Faugeras (1993). At the end of the day, most variants boil down to smoothing the image with something that looks a lot like a Gaussian before measuring the gradient. All edge detectors behave badly at corners; only the details vary.

Object boundaries are not the same as sharp changes in image values. There is a vast literature seeking to build boundary detectors; we can provide only some pointers. The reader could start with Bergholm (1987), Deriche (1990), Elder and Zucker (1998), Fleck (1992), Kube and Perona (1996), Olson (1998), Perona and Malik (1990*b*), or Torre and Poggio (1986). The best current boundary detector takes quite a lot of local information into account, and is described in Section 17.1.3.

The edges that our edge detectors respond to are sometimes called *step edges* because they consist of a sharp, "discontinuous" change in value that is sometimes modeled as a step. A variety of other forms of edge have been studied. The most commonly cited example is the *roof edge*, which consists of a rising segment meeting a falling segment, rather like some of the reflexes that can result from the effects of interreflections. Another example that also results from interreflections is a composite of a step and a roof. It is possible to find these phenomena by using essentially the same steps as outlined before (find an "optimal" filter, and do nonmaximum suppression on its outputs) (Canny 1986, Perona and Malik 1990*a*). In practice, this is seldom done. There appear to be two reasons. First, there is no comfortable basis in theory (or practice) for the models that are adopted. What particular composite edges are worth looking for? The easy answer—those for which optimal filters are reasonably easy to derive—is most unsatisfactory. Second, the semantics of roof edges and more complex composite edges is even vaguer than that of step edges. There is little notion of what one would *do* with roof edge once it had been found.

Corners, Neighborhoods, and Interest Points

The first corner detector we know of is due to Moravec (1980). Corner detectors are now very well studied (there is an excellent Wikipedia page that describes the various detectors and their relations at `http://en.wikipedia.org/wiki/Corner_detection`). The Harris and Stephens detector we described remains competitive. Important variants look at different eigenvalue criteria (Tomasi and Shi 1994); differential geometric criteria (Wang and Brady 1994); multiple scales (Lindeberg 1993); local self-similarity measures (Smith and Brady 1997, Trajkovic and Hedley 1998); and machine learning (Rosten *et al.* 2010).

For simplicity of exposition, we have elided corners and interest points (the other name under which corners are often studied). Interest points are usually thought of as a corner (or something like it) together with a neighborhood, covariant under some form of transformation. We like to see detecting the points and estimating their neighborhoods as distinct processes, though for strict covariance both the detector and the neighborhood estimator must be covariant. Various detectors are scale covariant (Mikolajczyk and Schmid 2002); affine covariant (Mikolajczyk and Schmid 2002); and illumination robust (Gevrekci and Gunturk 2009). The idea can

be extended to spatio-temporal representations (Willems *et al.* 2008, Laptev 2005). There are now detailed experimental studies of the performance of interest point detectors (Schmid *et al.* 2000, Privitera and Stark 1998, Mikolajczyk *et al.* 2005).

Descriptors

The tricks to describing neighborhoods seem to be: describe a local texture pattern within a covariant neighborhood; work with orientations, because they're illumination invariant; and use histograms to suppress spatial detail, working with more detail at the center than at the boundary. These tricks appear in numerous papers in a variety of forms (e.g., Schmid and Mohr (1997); Belongie *et al.* (2001); Berg *et al.* (2005)), but SIFT and Hog features now dominate. Comparisons between local descriptors seem to support this dominance (Mikolajczyk and Schmid 2005).

PROBLEMS

5.1. Each pixel value in 500×500 pixel image $\mathcal{I}$ is an independent, normally distributed random variable with zero mean and standard deviation one. Estimate the number of pixels that, where the absolute value of the x derivative, estimated by forward differences (i.e., $|I_{i+1,j} - I_{i,j}|$, is greater than 3.

5.2. Each pixel value in 500×500 pixel image $\mathcal{I}$ is an independent, normally distributed random variable with zero mean and standard deviation one. $\mathcal{I}$ is convolved with the $2k + 1 \times 2k + 1$ kernel $\mathcal{G}$. What is the covariance of pixel values in the result? There are two ways to do this; on a case-by-case basis (e.g., at points that are greater than $2k+1$ apart in either the x or y direction, the values are clearly independent) or in one fell swoop. Don't worry about the pixel values at the boundary.

5.3. We have a camera that can produce output values that are integers in the range from 0 to 255. Its spatial resolution is 1024 by 768 pixels, and it produces 30 frames a second. We point it at a scene that, in the absence of noise, would produce the constant value 128. The output of the camera is subject to noise that we model as zero mean stationary additive Gaussian noise with a standard deviation of 1. How long must we wait before the noise model predicts that we should see a pixel with a negative value? (Hint: You may find it helpful to use logarithms to compute the answer as a straightforward evaluation of $\exp(-128^2/2)$ will yield 0; the trick is to get the large positive and large negative logarithms to cancel.)

5.4. Show that for a 2×2 matrix $\mathcal{H}$, with eigenvalues λ_1, λ_2

(a) $\det\mathcal{H} = \lambda_1\lambda_2$

(b) $\text{trace}\mathcal{H} = \lambda_1 + \lambda_2$

PROGRAMMING EXERCISES

5.5. The Laplacian of a Gaussian looks similar to the difference between two Gaussians at different scales. Compare these two kernels for various values of the two scales. Which choices give a good approximation? How significant is the approximation error in edge finding using a zero-crossing approach?

5.6. Obtain an implementation of Canny's edge detector (you could try the vision home page; MATLAB also has an implementation in the image processing toolbox), and make a series of images indicating the effects of scale and contrast thresholds on the edges that are detected. How easy is it to set up the edge detector to mark only object boundaries? Can you think of applications where

this would be easy?

5.7. It is quite easy to defeat hysteresis in edge detectors that implement it; essentially, one sets the lower and higher thresholds to have the same value. Use this trick to compare the behavior of an edge detector with and without hysteresis. There are a variety of issues to look at:

(a) What are you trying to do with the edge detector output? It is sometimes helpful to have linked chains of edge points. Does hysteresis help significantly here?

(b) Noise suppression: We often wish to force edge detectors to ignore some edge points and mark others. One diagnostic that an edge is useful is high contrast (it is by no means reliable). How reliably can you use hysteresis to suppress low-contrast edges without breaking high-contrast edges?

5.8. Build a Harris corner detector; for each corner, estimate scale and orientation as we have described. Now test how well your list of neighborhoods behaves under rotation, translation, and scale of the image. You can do this by a simple exercise in matching. For each test image, prepare a rotated, translated, and scaled version of that image. Now you know where each neighborhood should appear in the new version of the image — check how often something of the right size and orientation appears in the right place. You should find that rotation and translation cause no significant problems, but large scale changes can be an issue.