2D Edge Detection

Last lecture I introduced the Canny's basic criteria for edge detection. Let's now look at the 2D case, and the particular filter that he uses.

Recall from your probability background the definition of a Gaussian function,

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

which has mean zero and variance σ^2. Canny uses the filter:

$$f(x) = \frac{d}{dx} G(x) \ .$$

Using a derivative of a Gaussian to detect edges was not a Canny's idea. It had been used in many other researchers previously.[1] Canny shows this filter does a good job of satisfying his various criteria.

One way to think of this $f(x)$ is that it smooths an image with a Gaussian (to reduce noise), and then takes the derivative to look for the edge in the smoothed image. That is,

$$(I * \frac{d}{dx} G)(x) \ = (\frac{d}{dx}(I * G))(x)$$

where we are using the fact that a derivative is a convolution and convolution is commutative and associative.

Recall that to find the edge, Canny looks for a (large) maximum in $I * f(x)$. Thus, for $f(x) = \frac{d}{dx}G)(x)$, looking for a maximum is equivalent to looking for a zero value of $\frac{d}{dx} I * f(x) = I * \frac{d^2}{dx^2} G)(x)$. The zero value was used last lecture when we discussed how well we can localize the edge. I mention it again here because there are now two derivatives here and it is easy to get confused. There is a derivative of a Gaussian, used in the filter definition. And there is a derivative of the filter output which is used to find a maximum in the filter output.

2D convolution

Canny edge detection is done on 2D images, so we need to generalize our definition of convolution and edge detection to 2D. Let $I(x, y)$ be an image and let $f(x, y)$ be some other function. Then the discrete 2D convolution is

$$(I * f)(x, y) \ \equiv \ \sum_{-\infty}^{\infty} \sum_{-\infty}^{\infty} I(x - u, y - v) \ f(u, v)$$

Again, one can deal with image boundaries either by padding with zeros or make the functions periodic, e.g.

$$I(x, y) = I(x \bmod N_x, y \bmod N_y) \ .$$

Continuous 2D convolution is defined similarly

$$(I * f)(x, y) \ \equiv \ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(x - u, y - v) \ f(u, v) du dv. \qquad (1)$$

[1] One of the most famous is by Marr and Hildreth. See D. Marr, E. Hildreth, "Theory of Edge Detection", Proc. Royal Soc. Lond. (Series B), 1980.

2D Gaussian and directional derivative

Suppose we have a 2D image $I(x,y)$ and there is an intensity edge at the line $ax + by = c$, that is, the underlying image has two different values on opposite sides of this line. We would like to detect the points (x,y) that lie along this line. If there is little noise, then problem is easy to solve. We compute the gradient of the image $(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ or a discrete approximation of it, and look where the maximum magnitude of the gradient occurs. This is similar to the 1D case.

In practice, though, there is noise. To reduce noise, it is common to blur the image prior to computing the gradient. (This is similar to what we discussed above in the 1D case.) We blur with a 2D Gaussian. A 2D Gaussian is defined to be just the product of two 1D Gaussians:

$$G(x,y) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{x^2}{2\sigma^2}}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{y^2}{2\sigma^2}} = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Observe that a 2D Gaussian is radially symmetric (i.e. $x^2 + y^2 = c$ is a circle).

Directional Derivative

In the absence of noise, the edge will be perpendicular to the gradient direction, $\nabla(G * I)(x,y)$ and the magnitude $|\nabla(G * I)(x,y)|$ will have a peak along the edge, i.e. along the line marking the boundary between the two constant intensity regions. This suggests, as in the 1D case, that a 2D edge detector should try to find the points (x,y) where $|\nabla(I * G)(x)|$ achieves a maximum in the direction of $\nabla(I * G)(x)$. To clarify: we do not look for isolated points where $|\nabla(I * G)(x)|$ achieves a local 2D maximum. Rather, we look for a maximum of the magnitude of the gradient in the direction of the gradient, i.e. a directional derivative.

We need to compute the gradient $\nabla(I * G)(x,y)$. How do we do this ?

$$\nabla G(x,y) = (\frac{\partial G(x,y)}{\partial x}, \ \frac{\partial G(x,y)}{\partial y})$$

So by the associativity of convolution,

$$\nabla(G(x,y) * I(x,y)) = (\nabla G(x,y)) * I(x,y)$$

so

$$(\nabla G(x,y)) * I(x,y) = (\frac{\partial G(x,y)}{\partial x} * I(x,y), \ \frac{\partial G(x,y)}{\partial y} * I(x,y) \ )$$

so we see that we can compute $\nabla G(x,y) * I(x,y)$ using the filters $\frac{\partial G(x,y)}{\partial x}$ and $\frac{\partial G(x,y)}{\partial y}$.

In general, we denote the (first) directional derivative in direction θ by

$$\begin{aligned} G^1_\theta(x,y) &\equiv lim_{s\rightarrow 0}\frac{1}{s}(G(x + s\cos\theta, y + s\sin\theta) - G(x,y)) \\ &= \cos\theta\frac{\partial G}{\partial x} + \sin\theta\frac{\partial G}{\partial y} \end{aligned}$$

In particular, we wish to see where there is a maximum in the direction of the gradient. Define $\theta(x,y)$ as the angular direction of the gradient at (x,y), where

$$\nabla G(x,y) * I(x,y) = (\ \cos\theta|\nabla G(x,y) * I(x,y)| \ , \ \sin\theta \ |\nabla G(x,y) * I(x,y)|).$$

To see if we have a maximum in $G^1_\theta(x,y) * I(x,y)$, we take the directional derivative again and see if it is zero (or sufficiently near zero). For fixed $x, y, \theta(x, y0$, we define

$$\begin{aligned} G^2_\theta(x,y) &\equiv lim_{s\to 0}\ \frac{1}{s}(G^1_\theta(x + s\cos\theta, y + s\sin\theta) - G^1_\theta(x,y)) \\ &= \cos^2\theta\ \frac{\partial^2 G(x,y)}{\partial x^2} + \sin^2\theta\ \frac{\partial^2 G(x,y)}{\partial y^2}\ +\ 2\cos\theta\sin\theta\frac{\partial^2 G(x,y)}{\partial x \partial y} \end{aligned}$$

A maximum in $G^1_\theta(x,y)$ in direction θ is indicated by a zero value in $G^2_\theta(x,y)$.

Note that we can compute $G^2_\theta(x,y) * I(x,y)$ rather easily, using first and second derivative of Gaussian filters, namely derivatives in the x and y directions. We use the first derivative filters to compute $\theta(x,y)$. Then we use $\theta(x,y)$ to take combinations of the second directional derivative filters.

Non-maxima suppression

Some textbooks (such as Trucco and Verri, which is used in many Computer Vision courses) take a simpler computational approach to finding the maximum of the directional derivative of $\nabla G * I$. After one computes $\nabla G * I$, they simply check the neighboring pixels of (x,y) which are in *approximate* direction of the gradient and see if their values of $|\nabla G * I(x,y)|$ are less than those at (x,y). Since we are working on a square pixel grid, we need to say what we mean by neighbor here. Trucco and Verri quantize the "directions" θ into 45 degree steps, e.g. $[-22.5^o, 22.5^o], (22.5^o, 67.5^o), etc$, namely θ intervals centered on the directions of the eight neighbors in the square grid. For each pixel, they check the two neigbhoring pixels in the quantized θ direction. For example, if the gradient is in direction $\theta = 18^o$, this lies in interval $[-22.5^o, 22.5^o]$ and so the immediate neighbors are pixels located at $(x \pm 1, y)$. Or, if the gradient is in direction $\theta = 25^o$, this lies in interval $(22.5^o, 67.5^o]$, and so the immediate neighbors are pixels located at $(x+1, y+1)$ and $(x-1, y-1)$.

If either of the two neighbors has an $|\nabla G * I|$ value that is greater than that of the pixel (x,y), then it follows that the pixel (x,y) doesn't have a maximum of $|\nabla G * I|$ in direction $\theta(x,y)$. Repeating this test for all pixels gives you a set of pixels (x,y) which are candidates for being considered edges.

Just finding a local maximum is not sufficient, however, since noise alone can produce local maxima. We only want to find local maxima of $\mid \nabla(G(x,y) * I(x,y)) \mid$ that are sufficiently large. What is large? There is no simple answer to this unfortunately. If you choose a threshold to be too low, they you may "detect" edges that aren't there, that is, you might think there is an edge where in fact there is only noise. (This is called a "false alarm" in Signal Detection Theory[2].) On the other hand, if you choose the threshold to be too high, then you might miss an edge that is in fact there. (This is called a "miss" in Signal Detection Theory.) By lowering/raising the threshold, you will raise both the detection rate (good) and false alarm rate (bad).

Interest point detection ("corners")

Edges are very useful for marking boundaries between regions in an image. One limitation with edges, though, is that we cannot use them to reliably identify *single points*, for example, points that

[2]SDT is a general theory used in many experimental fields that involve detecting signals in the presence of noise

we might try to match from one image to another. Technically, as we will see next, the problem is that edges have a well defined intensity gradient direction and so the intensity is by definition constant *along* the edge. As such, neighboring points along an edge are difficult to distinguish from each other.

It is common to refer to points that is locally distinctive as *corners*, though we don't necessarily need them to be the corner of anything. The word "corner" just suggests that there are two edges coming together at a sharp angle. A more general notion of a corner is just that the intensities in a small neighborhood (say 7×7) of a pixel (x_0, y_0) are different from those in the neighborhood of a nearby pixel $(x_0 + \Delta x, y_0 + \Delta y)$. Again, note this is typically *not* the case with edges, since the intensities in a neighborhood tend to be constant as you move along the edge.

We set up the problem of finding locally distinctive points ("corners") formally as follows. The intensities at a point (x_0, y_0) are locally distinctive if the following "error" is large, relative to the distance $|(\Delta x, \Delta y)|$ to the local neighbor:

$$e(x_0, y_0, \Delta x, \Delta y) = \sum_{(x,y) \in Nbd(x_0,y_0)} (I(x,y) - I(x + \Delta x, y + \Delta y))^2.$$

You could just check this condition directly, by computing the sum for the eight neighbors. I am going to present another way to do it, however, which has basically the same computational cost, but which generalizes to other fundamental problems (e.g. that we will see next lecture).

We approximate $I(x + \Delta x, y + \Delta y)$ with a Taylor series to first order:

$$I(x + \Delta x, y + \Delta y) \approx I(x, y) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y.$$

which is fine as long as $(\Delta x, \Delta y)$ are very small here, say ± 1 pixel, and $I(x, y)$ has been smoothed. Typically one smooths with a 2D Gaussian before doing these operations, but I will not write this explicitly so that I keep the notation down.

The image gradient[3] is

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

and so we have

$$\begin{aligned} e(x_0, y_0, \Delta x, \Delta y) &\approx \sum_{(x,y) \in Ngd(x_0,y_0)} (\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y)^2 \\ &= \sum_{(x,y) \in Ngd(x_0,y_0)} ((\nabla I) \cdot (\Delta x, \Delta y))^2 \\ &= \sum_{(x,y) \in Ngd(x_0,y_0)} (\Delta x, \Delta y)(\nabla I)(\nabla I)^T (\Delta x, \Delta y)^T \\ &= (\Delta x, \Delta y)\{ \sum_{(x,y) \in Ngd(x_0,y_0)} (\nabla I)(\nabla I)^T \} (\Delta x, \Delta y)^T. \end{aligned}$$

There is a subtlety here which can be easily missed. Before we made the linear approximation, we were considering two neighborhoods, one around (x_0, y_0) and the other around $(x_0 + \Delta x,\ y_0 +$

[3]think of this as $\nabla(G * I)$ since I am not explicitly writing the blurring with G

Δy). Once we make the linear approximation, we consider only pixels in the neighborhood around (x_0, y_0). Of course, to compute the underlying Gaussian blur and the derivative, pixels beyond the neighborhood are ultimately involved.

The 2×2 matrix

$$\sum_{(x,y) \in Ngd(x_0,y_0)} (\nabla I)(\nabla I)^T = \begin{bmatrix} \sum(\frac{\partial I}{\partial x})^2 & \sum(\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ \sum(\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & \sum(\frac{\partial I}{\partial y})^2 \end{bmatrix}$$

is called the *second moment matrix* and we write it $\mathbf{M}$.

We said that for the intensities at a point (x, y) to be locally distinctive, $e(x, y)$ should be large relative to the step $(\Delta x, \Delta y)$. How can we quickly examine this condition for each (x, y) ? Since $\mathbf{M}$ is symmetric, it has two real eigenvalues and two the eigenvectors are orthogonal. Since $e(x_0, y_0) \geq 0$, the eigenvalues must be positive i.e. letting $(\Delta x, \Delta y)$ be an eigenvector with eigenvalue λ, we see $\lambda|(\Delta x, \Delta y)| \geq 0$.

For the error $e(x, y, \Delta x, \Delta y)$ to be much larger than $|(\Delta x, \Delta y)|$ for any $(\Delta x, \Delta y)$, we require that *both* eigenvalues must be much greater than zero. Thus, if we want to ensure that $e(x, y, \Delta x, \Delta y)$ is much larger than $|(\Delta x, \Delta y)|$, we just check if both eigenvalues are greater than some chosen threshold value.

Harris corner detector

One practical problem, though, is that computing the eigenvalues of $\mathbf{M}$ at all pixels can be slow, since computing the eigenvalues requires solving a quadratic equation ($a\lambda^2 + b\lambda + c = 0$, which requires computing a square root. We will check every pixel in an image to see if it is locally distinctive. We would like to avoid computing a square root at each pixel.

One trick that was proposed by Harris and Stevens[4] is to take advantage of a basic fact from matrix algebra that the determinant of a matrix with eigenvalues λ_1 and λ_2 is the product $\lambda_1\lambda_2$ and the trace is $\lambda_1 + \lambda_2$. The trick is to convert the conditions "both eigenvalues are large" into a condition on the determinant and trace of $\mathbf{M}$, both of which can easily be computed from $\mathbf{M}$ i.e. no need to compute the λ's explicitly.

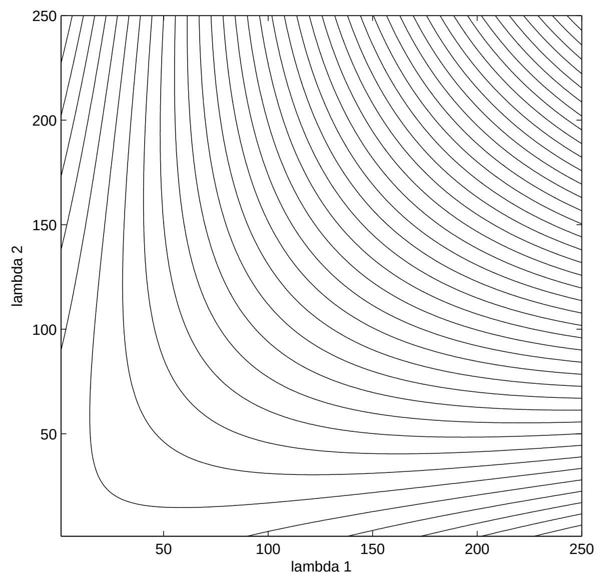
Harris and Stevens noticed that if k is small constant (say $k \approx 0.1$) then

$$\lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

is negative if one of the eigenvalues is 0, is small if both of the eigenvalues are small, but otherwise is large. (The figure on the next page shows the iso-values of this expression when $k = 0.1$. Thus the *Harris corner detector* or *Harris interest point* detector, as its now known, looks for points in the image where the above expression takes a value above some given threshold.

Notice that if the determinant is near 0 but the trace is much different from zero, then one of the eigenvalues must be large and the other must be small. In this case, there must be strong image intensity gradients in the neighborhood of the point, but the gradients would be in only one direction. (We will discuss this in more detail next lecture.) Thus, Harris and Stevens argued that using the trace and determinant of the second moment matrix could be used to detect "edges" as

[4] "A combined corner and edge detector" by C. Harris and M. Stevens, Proceedings of the 4th Alvey Vision Conference, 1988. Cited nearly 4000 times according to Google Scholar.



well as corners, where by "edges" they mean points whose neighborhood contains strong gradients that are all roughly in the same direction. (This now is a more general notion of an edge that we discussed earlier, just as Harris'es "corner" is a more general notion of a corner.)

One final observation (not made in the lecture, but read it anyhow because it is important): While the Harris corner detector gives locally distinctive points, it doesn't necessarily give points that can be matched from one image to the next. Suppose the Harris corner detector finds an interest point (x_0, y_0) in an image, and this point is the projection of some scene point (X, Y, Z). A second image taken with a different camera (with both different intrinsic and extrinsic parameters) may see the same 3D point, but it typically will be at a different image position (x'_0, y'_0). Moreover, the intensities in the local neighborhood of this point in the second image might be different from those in the neighborhood of (x_0, y_0) in the first image. There are several reasons for this. First, because the projection matrix will typically be different in the two cameras, the neighboring points in the two images might not correspond in a simple way. That is, the neighborhoods might be deformed versions of each other. Second, if the second camera is at a different position than the first and if the surface is not exactly Lambertian then the image irradiances from corresponding points might differ. Third, images have noise. Fourth, different cameras have different response curves i.e. exposure to digital value.

Thus, even though (x_0, y_0) is found as an interest point in the first image, there is no absolute guarentee that the corresponding point will be found to be an interest point in the other image. We will return to this issue a few lectures from now (e.g. when we get to RANSAC).