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

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

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

This is a probability density function, namely it is positive everywhere and

$$\int_{-\infty}^{\infty} G(x)dx = 1.$$

It has mean zero and variance σ^2, that is,

$$\mu = \int_{-\infty}^{\infty} xG(x)dx = 0, \qquad \int_{-\infty}^{\infty} (x-\mu)^2 G(x)dx = \sigma^2$$

We start by considering a simpler function

$$g(x) = e^{-\frac{x^2}{2}}$$

and its derivative

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

One way to think of this $f(x)$ is that it smooths an image with a Gaussian shape (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 $g(x)$, 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

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

2D Edge Detection

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, there is noise. As before, to reduce noise, we blur the image prior to computing the gradient. Define:

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

which is radially symmetric, i.e. $x^2 + y^2 = c$ is a circle. A 2D Gaussian $G(x,y)$ with standard deviation σ^2 is formed in the same way, namely as the product of two 1D Gaussians $G(x)$ and $G(y)$.

In the absence of noise, the edge will be perpendicular to the gradient direction, and the magnitude $|\nabla(g * I)(x,y)|$ will have a peak as you move across the edge. 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.

How do we compute the gradient $\nabla(I * g)(x,y)$?

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

so by the associativity of convolution,

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

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 $\frac{\partial g(x,y)}{\partial y}$.

Non-maxima suppression

Here we define a simple scheme for finding the local maxima of the filtered image (following textbook by Trucco and Verri, rather than Canny's original paper which is more complicated). After having computed $\nabla g * I(x,y)$, 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 one concludes 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 (x, y) gives you a set of pixels 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.) 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 or "hit" rate (good) and false alarm rate (bad).

Corner detection (also known as "interest points")

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 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:

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

You could just check this condition directly, by computing the sum for the eight neighbors. However, this would only allow you to have integer values of Δx and Δy.

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

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

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

The image gradient[1] is

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

and so we have

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

One subtlety: Before we made the linear approximation, we were considering two neighborhoods, one around (x_0, y_0) and the other around $(x_0{+}\Delta x,\ y_0{+}\Delta y)$. Once we make the linear approximation, we consider only pixels in the neighborhood around (x_0, y_0). [2]

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 will have more to say about it next lecture.

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

[2]Of course, to compute the underlying Gaussian blur and the derivative, pixels beyond the neighborhood are ultimately involved. But these pixels don't show up explicitly in the formulas.