EDGE DETECTION

INEL 6088 - Computer Vision

Davies Ch. 5, Jain et. al. Ch. 5
Sec. 4.2 Szelinski
ideal

\[\sim \text{real}\]
Edge point – point at which the local intensity changes significantly

Edge fragment – edge (point) and orientation

Edge detector – produces a set of edges (points or fragments)

Edge linking – orders a list of edges

Contour – list of edges

Edge following – searching the (input) image to determine contours
Real image

Edge detector produces:
• correct edges
• false edges (false positives)
• can miss edges (false negatives)
EDGE DETECTION USING GRADIENT

Simple way

\[ G[f(x,y)] = [G_x \ G_y]^T \]

\( G_x = \delta f/\delta x \approx f[i,j+1]-f[i,j] \approx \text{gradient at } [i, j+{1/2}] \)

\( G_y = \delta f/\delta y \approx f[i+1,j]-f[i,j] \approx \text{gradient at } [i+{1/2}, j] \)

To get the gradient at the same point \([i+{1/2}, j+{1/2}]\), use:

\[
G_x = \begin{bmatrix}
-1 & 1 \\
-1 & 1 \\
\end{bmatrix} \quad G_y = \begin{bmatrix}
1 & 1 \\
-1 & -1 \\
\end{bmatrix}
\]
Steps:
- filtering
- enhancement
- detection
- sometimes, localisation

Roberts: \( P5 = |P5 - P9| + |P8 - P6| \)

Sobel:
\[
M = \sqrt{s_x^2 + s_y^2}
\]
\[
s_x = (P3 + cP6 + P9) - (P1 + cP4 + P7)
\]
\[
s_y = (P1 + cP2 + P3) - (P7 + cP8 + P9)
\]

Prewitt: Sobel with \( c = 1 \)
\[ F_{r,c} = f_{r,c} \ast g_{r,c} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} f_{i,j} g_{r-i,c-j} \]

\( g_x = \text{conv2}(C, S_y) = \)

\[
\begin{array}{cccccccc}
2 & 5 & 19 & 64 & 110 & 91 & 29 \\
3 & 8 & 18 & 52 & 127 & 148 & 60 \\
3 & 30 & 68 & 65 & 43 & 31 & 12 \\
2 & 22 & 60 & 77 & 33 & -23 & -19 \\
-1 & -25 & -60 & -62 & -45 & -25 & -6 \\
-5 & -30 & -78 & -129 & -160 & -125 & -41 \\
\end{array}
\]

\[ 5 + 2 \times 25 + 32 - (2 + 2 + 15) = 68 \]
\[
S_x = S_y' \\
\begin{bmatrix}
2 & 1 & 13 & 32 & 14 & -33 & -29 \\
7 & 4 & 34 & 90 & 77 & -94 & -118 \\
13 & 30 & 56 & 99 & 121 & -129 & -190 \\
\end{bmatrix}
\]

\[
g_y = \text{conv2}(S_y, C) = \\
\begin{bmatrix}
18 & 72 & 90 & 79 & 75 & -151 & -183 \\
19 & 67 & 98 & 86 & 41 & -153 & -158 \\
13 & 24 & 58 & 73 & 40 & -97 & -111 \\
4 & 2 & 15 & 25 & 16 & -27 & -35 \\
\end{bmatrix}
\]

\[
G(i,j) = \sqrt{g_x^2 + g_y^2} \\
\begin{bmatrix}
2.8284 & 5.0990 & 23.0217 & 71.5542 & 110.8873 & 96.7988 & 41.0122 \\
7.6158 & 8.9443 & 38.4708 & 103.9423 & 148.5194 & 175.3283 & 132.3782 \\
13.3417 & 42.4264 & 88.0909 & 118.4314 & 128.4134 & 132.6725 & 190.3786 \\
18.1108 & 75.2861 & 108.1665 & 110.3177 & 81.9390 & 152.7416 & 183.9837 \\
19.0263 & 71.5122 & 114.9087 & 106.0189 & 60.8769 & 155.0290 & 158.1139 \\
13.9284 & 38.4187 & 97.2008 & 148.2228 & 164.9242 & 158.2214 & 118.3300 \\
\end{bmatrix}
\]
uint8 version of G=

\[
\begin{array}{cccccccc}
3 & 5 & 23 & 72 & 111 & 97 & 41 \\
8 & 9 & 38 & 104 & 149 & 175 & 132 \\
13 & 42 & 88 & 118 & 128 & 133 & 190 \\
18 & 75 & 108 & 110 & 82 & 153 & 184 \\
19 & 72 & 115 & 106 & 61 & 155 & 158 \\
14 & 38 & 97 & 148 & 165 & 158 & 118 \\
6 & 10 & 31 & 72 & 109 & 101 & 49 \\
\end{array}
\]

Using a threshold of 0.4 to detect the edge:

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]
Filtered with 7x7 gaussian filter

Figure 5.4: A comparison of various edge detectors. (a) Original image. (b) Filtered image. (c) Simple gradient using $1 \times 2$ and $2 \times 1$ masks, $T = 32$. (d) Gradient using $2 \times 2$ masks, $T = 64$. (e) Roberts cross operator, $T = 64$. (f) Sobel operator, $T = 225$. (g) Prewitt operator, $T = 225$. 
unfiltered

Figure 5.5: A comparison of various edge detectors without filtering. (a) Original image. (b) Simple gradient using $1 \times 2$ and $2 \times 1$ masks, $T = 64$. (c) Gradient using $2 \times 2$ masks, $T = 64$. (d) Roberts cross operator, $T = 64$. (e) Sobel operator, $T = 225$. (f) Prewitt operator, $T = 225$. 
Filtered, noisy

Figure 5.6: A comparison of various edge detectors on a noisy image. (a) Noisy image. (b) Filtered image. (c) Simple gradient using $1 \times 2$ and $2 \times 1$ masks, $T = 32$. (d) Gradient using $2 \times 2$ masks, $T = 64$. (e) Roberts cross operator, $T = 64$. (f) Sobel operator, $T = 225$. (g) Prewitt operator, $T = 225$. 
Unfiltered, noisy

Figure 5.7: A comparison of various edge detectors on a noisy image without filtering. (a) Noisy image. (b) Simple gradient using $1 \times 2$ and $2 \times 1$ masks, $T = 64$. (c) Gradient using $2 \times 2$ masks, $T = 128$. (d) Roberts cross operator, $T = 64$. (e) Sobel operator, $T = 225$. (f) Prewitt operator, $T = 225$. 
EDGE DETECTORS THAT USE THE SECOND DERIVATIVE

Two ways:
• Laplacian
• Second directional derivative

Change in sign indicates edge
\[ \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]

\[ \frac{\partial^2 f}{\partial x^2} f[i, j+1] - 2f[i, j] + f[i, j-1] \]

\[ \frac{\partial^2 f}{\partial y^2} f[i+1, j] - 2f[i, j] + f[i-1, j] \]

\[ \nabla^2 \approx \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \]

Combined to get

\[ \nabla^2 \approx \begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix} \]

Also used:
A sample image containing a vertical step edge.

Figure 5.9: The response of the Laplacian to a vertical step edge.
A sample image containing a vertical ramp edge.

Figure 5.10: The response of the Laplacian to a vertical ramp edge.
SECOND DIRECTIONAL DERIVATIVE

- Second derivative computed in the direction of the gradient
- Implement using the formula:

\[
\frac{\partial^2}{\partial n^2} = \frac{f_x^2 f_{xx} + 2 f_x f_y f_{xy} + f_y^2 f_{yy}}{f_x^2 + f_y^2}
\]
Problem with Laplacian and second-derivative-operator:
- very sensitive to noise
- small peaks in first derivative produce zero-crossing in the second derivative.

Solution: Filter out noise before edge enhancement

**Laplacian of Gaussian (LoG)**

- Smoothing: Gaussian smoothing
- Enhancement: Second-derivative edge enhancement
- Detection: zero-crossing in the second derivative with a corresponding large peak (i.e. above some threshold) in the first derivative
- If desired, use linear interpolation to locate the edge with sub-pixel resolution
The output of the LoG operator, \( h(x, y) \) is given by:

\[
h(x, y) = \nabla^2 [g(x, y) * f(x, y)]
\]

\[
h(x, y) = [\nabla^2 g(x, y)] * f(x, y)
\]

\[
\nabla^2 g(x, y) = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \exp \left(-\frac{x^2 + y^2}{2\sigma^2}\right)
\]

Figure 5.11: The inverted Laplacian of Gaussian function, \( \sigma = 2 \), in one and two dimensions.
Two ways of doing LoG:

- Gaussian smoothing followed by laplacian
- Convolution of image with a linear filter that is the laplacian of a gaussian filter

To obtain real edges, it might be necessary to combine information from filters of different sizes. The problem of combining edges obtained from different size operators still remains.
### $5 \times 5$ Laplacian of Gaussian mask

\[
\begin{bmatrix}
0 & 0 & -1 & 0 & 0 \\
0 & -1 & -2 & -1 & 0 \\
-1 & -2 & 16 & -2 & -1 \\
0 & -1 & -2 & -1 & 0 \\
0 & 0 & -1 & 0 & 0 \\
\end{bmatrix}
\]

### $17 \times 17$ Laplacian of Gaussian mask

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -2 & -3 & -3 & -3 & 0 & 2 & 4 & 2 & 0 & -3 & -3 & -3 & -2 & -1 & 0 & 0 & 0 \\
-1 & -1 & -3 & -3 & -3 & -3 & 0 & 4 & 10 & 12 & 10 & 4 & 0 & -3 & -3 & -3 & -2 & -1 & 0 \\
-1 & -1 & -3 & -3 & -3 & -3 & 2 & 10 & 18 & 21 & 18 & 10 & 2 & -2 & -3 & -3 & -2 & -1 & 0 \\
-1 & -1 & -3 & -3 & -3 & -3 & 4 & 12 & 21 & 24 & 21 & 12 & 4 & -3 & -3 & -3 & -2 & -1 & 0 \\
-1 & -1 & -3 & -3 & -3 & -3 & 2 & 10 & 18 & 21 & 18 & 10 & 2 & -2 & -3 & -3 & -2 & -1 & 0 \\
-1 & -1 & -3 & -3 & -3 & -3 & 0 & 4 & 10 & 12 & 10 & 4 & 0 & -3 & -3 & -3 & -2 & -1 & 0 \\
0 & -1 & -2 & -3 & -3 & -3 & 0 & 2 & 4 & 2 & 0 & -3 & -3 & -3 & -2 & -1 & 0 & 0 & 0 \\
0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 \\
0 & 0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & 0 & 0 \\
0 & 0 & -1 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Figure 5.12: Some useful Laplacian of Gaussian masks [146].
LoG Results

Smoothing causes blurring
large $\sigma$: better noise filtering but more blurring – can cause edge merging

By applying filters of different sizes and analysing the results, better edge detection can be accomplished.
An alternative approach is to fit a function to the image and then detect the edges in the function…

Figure 5.14: A graphical representation of the continuous image intensity function.
Facet model: fit a function only in the local neighborhood of each pixel.
Example: bicubic polynomial

\[ f(x, y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 + k_7 x^3 + k_8 x^2 y + k_9 xy^2 + k_{10} y^3 \]

Figure 5.16: An example of the coordinate system for the facet model using a 5 \times 5 neighborhood. The continuous intensity function is approximated at every pixel location only within this neighborhood. The array indices are marked on the pixels for clarity. Note that pixel \([i, j]\) lies in the center of the neighborhood.
Masks for computing the coefficients of the bicubic approximation.
Figure 5.17: Masks for computing the coefficients of the bicubic approximation [103].
\[ f'_\theta(x, y) = \frac{\partial f}{\partial x} \cos \theta + \frac{\partial f}{\partial y} \sin \theta \]

\[ f''_\theta(x, y) = \frac{\partial^2 f}{\partial x^2} \cos^2 \theta + 2 \frac{\partial^2 f}{\partial x \partial y} \cos \theta \sin \theta + \frac{\partial^2 f}{\partial y^2} \sin^2 \theta \]

The angle may be chosen to be the angle of the approximating plane:

\[ \sin \theta = \frac{k_3}{\sqrt{k_2^2 + k_3^2}} \]

\[ \cos \theta = \frac{k_2}{\sqrt{k_2^2 + k_3^2}} \]

\[ f'''_\theta(x_0, y_0) = 2(3k_7 \cos^2 \theta + 2k_8 \sin \theta \cos \theta + k_9 \sin^2 \theta)x_0 \]

\[ + 2(k_8 \cos^2 \theta + 2k_9 \sin \theta \cos \theta + 3k_{10} \sin^2 \theta)y_0 \]

\[ + 2(k_4 \cos^2 \theta + k_5 \sin \theta \cos \theta + k_6 \sin^2 \theta) \]
There is an edge at \((x_0, y_0)\) if for some \(\rho\), \(|\rho| < \rho_0\) where \(\rho_0\) is the length of the side of a pixel,

\[
\begin{align*}
  x_0 &= \rho \cos \theta \\
  y_0 &= \rho \sin \theta \\
  f''(x_0, y_0) &= 6(k_{10} \sin^3 \theta + k_9 \sin^2 \theta \cos \theta + k_8 \sin \theta \cos^2 \theta + k_7 \cos^3 \theta)\rho \\
  &\quad + 2(k_6 \sin^2 \theta + k_5 \sin \theta \cos \theta + k_4 \cos^2 \theta) \\
  &= A\rho + B
\end{align*}
\]

\[f''(x_0, y_0; \rho) = 0\]

and

\[f'(x_0, y_0; \rho) \neq 0\]
Figure 5.18: An enlarged view of the pixel at the center of the approximated function. \((x_0, y_0)\) is the location of the edge as determined by Equations 5.41 and 5.42. This pixel will be marked as an edge pixel since the location of the edge falls within its boundaries.

Figure 5.19: An enlarged view of the pixel at the center of the approximated function. This pixel will not be marked as an edge pixel since the location of the edge does not fall within the pixel boundaries.
Figure 5.20: Edges obtained with facet model edge detector.
CANNY EDGE DETECTION

- Let $I[i,j]$ be the image. First use Gaussian smoothing:
  \[ S[i,j] = G[i,j; \sigma] * I[i,j] \]

- Find approx. gradient using $2 \times 2$ first-diff. approx.
  - x direction:
    \[ P[i,j] = (S[i,j+1] - S[i,j] + S[i+1,j+1] - S[i+1,j]) / 2 \]
  - y direction:
    \[ Q[i,j] = (S[i,j] - S[i+1,j] + S[i,j+1] - S[i+1,j+1]) / 2 \]
CANNY EDGE DETECTION (CONT)

- Magnitude and orientation: rectangular to polar conversion:
  - $M[i,j] = \sqrt{P[i,j]^2 + Q[i,j]^2}$
  - $\theta[i,j] = \arctan(Q[i,j]/P[i,j])$

- Non-maxima Suppression
  - Thins the ridges of gradient magnitude to one pixel
  - Passes a 3x3 neighborhood across the magnitude array $M[i,j]$ and replace pixels with 0 if not greater than neighbors
Possible gradient orientations are partitioned into the following sectors for non-maxima suppression

Replace pixel 0 with value 0 if it is not bigger than neighboring pixels in the (quantized) direction of the gradient
CANNY EDGE DETECTION (CONT)

- Thresholding
  - Apply two threshold $t_1$ and $t_2$, with $t_2 = 2 \times t_1$
  - This produces two threshold images $T_1$ & $T_2$
  - $T_2$ will have fewer edges, but might have gaps
  - When the end of a contour in $T_2$ is reached, look into $T_1$ at the 8-neighboring locations to see if there are further edge points that can be linked to the contour.
Test Image (256x256)
Canny with 7x7 Gaussian smoothing + gradient approx + NMS
Canny with $31 \times 31$ Gaussian smoothing
SEQUENTIAL METHODS

- Edge following:
  - scan image looking for strong edge
  - extend the edge in the proper direction by looking at neighboring edges
  - link if directions are compatible
  - look at large neighborhood to fill in missing edges
Figure 5.25: An illustration of edge following.
Corner Detection

Consider the spatial image gradient, \([E_x, E_y]^T\) (the subscripts indicate partial differentiation, e.g. \(E_x = \frac{\partial E}{\partial x}\)). Consider a generic image point \(p\) and a neighborhood \(Q\) of \(p\), and a matrix \(C\) defined as

\[
C = \left( \begin{array}{cc}
\sum E_x^2 & \sum E_x E_y \\
\sum E_x E_y & \sum E_y^2 \\
\end{array} \right)
\]

where the sums are taken over \(Q\). A rotation of the coordinate axes can diagonalized the matrix,

\[
C' = \begin{pmatrix}
\lambda_1 & 0 \\
0 & \lambda_2
\end{pmatrix}
\]

- \(\lambda_1 \approx \lambda_2 \approx 0\): gradient vanishes everywhere
- \(\lambda_1 > 0; \lambda_2 \approx 0\): \(Q\) contains an ideal step edge; eigenvector associated with \(\lambda_1\) gives the direction of the edge
- \(\lambda_1 > 0; \lambda_2 > 0\): \(Q\) contains a corner. The larger the \(\lambda\)'s the stronger the edges.
Corner detection algorithm

- Inputs: grayscale image $I$ and two parameters:
  - threshold $t$ on $\lambda_2$ (smallest eigenvalue)
  - size $N$ of square window (neighborhood), $2N + 1$
- Compute gradient over entire image
- For each image point $p$,
  1. form matrix $C$ over a $(2N \times 1) \times (2N \times 1)$ neighborhood $Q$ of $p$
  2. find $\lambda_2$
  3. if $\lambda_2 \geq t$, save the coordinates of $p$ into a list $L$
- Sort $L$ in decreasing order of $\lambda_2$
- Scan $L$ and for each point $p$ delete other points in $L$ that belong to $Q$. 