Sobel Filters

Note that smoothing the image before applying a Sobel filter typically gives better results.

Even thresholding the Sobel filtered image cannot usually create precise, i.e., 1-pixel wide, edges.

Idea: The pixels with local maxima of the intensity gradient should be more precise edge indicators.

We can use second-derivative methods such as Laplacian Filters for finding these maxima.

Laplacian Filters

Idea:

- Smooth the image,
- compute the second derivative of the (2D) image,
- Find the pixels where the brightness function "crosses" 0 and mark them.

We can actually devise convolution filters that carry out the smoothing and the computation of the second derivative.

Laplacian Filters

Obviously, the second derivatives in vertical and horizontal directions can then be computed using the following convolution filters:

\[
\begin{bmatrix}
1 \\
-2 \\
 1
\end{bmatrix}
L_i

\begin{bmatrix}
1 & -2 & -1
\end{bmatrix}
L_j
\]

The second derivative (Laplacian; $\nabla^2$) of a 2D function is defined as the sum of its partial derivatives in each dimension.

How can we create a single convolution filter that computes the Laplacian in one step?

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]
Laplacian Filters

This noise intolerance of Laplacian filters requires the input image to be smoothed before processing (e.g., with a Gaussian filter).

More efficient: Note that convolution is an associative and commutative operation.

Then for an input image \( A \), a Gaussian filter \( G \), and a Laplacian filter \( L \), we have:

\[
(A \ast G) \ast L = A \ast (G \ast L) = (G \ast L) \ast A
\]

Instead of convolving \( A \) with \( G \) and then convolving the result with \( L \), we can first convolve \( G \) with \( L \) and then convolve the result with \( A \).

Laplacian Filters

Then we can efficiently build a single, small filter \( G \ast L \) that performs both smoothing and Laplacian filtering within a single convolution.

A filter of this type is called a Laplacian of Gaussian (LoG).

Detection of Zero Crossings

Fig 5.11 Method for computing zero crossings in a matrix. If any of the values \( b, c, \) or \( d \) differs from \( a \) in its sign, and its absolute difference from \( a \) exceeds a given threshold \( z \), then the upper-left pixel with value \( a \) is marked as a zero crossing.
Laplacian Filters

Fig 5.12: Results of edge detection in the example image using the second-derivative approach based on the small (3x3 pixel) Laplacian filter $\nabla^2 f$ (left panel) and the larger (9x9 pixel) Laplacian filter $\nabla^2 f$ (right panel).

Gaussian Edge Detection

As you know, one of the problems in edge detection is the **noise** in the input image. Noise creates lots of local intensity gradients that can trigger unwanted responses by edge detection filters. We can reduce noise through (Gaussian) smoothing, but there is a **trade-off**:

Stronger smoothing will remove more of the noise, but will also add uncertainty to the location of edges. For example, if two parallel edges are close to each other and we use a strong (i.e., large) Gaussian filter, these edges may be merged into one.

Canny Edge Detector

The **Canny edge detector** is a good approximation of the optimal operator, i.e., the one that maximizes the product of signal-to-noise ratio and localization.

Let $I[i, j]$ be our input image.

We apply our usual Gaussian filter with standard deviation $\sigma$ to this image to create a smoothed image $S[i, j]$: $S[i, j] = G[i, j; \sigma] * I[i, j]$

In the next step, we take the smoothed array $S[i, j]$ and compute its gradient. We apply 2x2 filters to approximate the vertical derivative $P[i, j]$ and the horizontal derivative $Q[i, j]$: $P[i, j] \approx (S[i+1, j] - S[i, j] + S[i+1, j+1] - S[i, j+1])/2$ $Q[i, j] \approx (S[i, j+1] - S[i, j] + S[i+1, j+1] - S[i+1, j])/2$

As you see, we average over the 2x2 square so that the point for which we compute the gradient is the same for the horizontal and vertical components (the center of the 2x2 square).

Once we have computed $P[i, j]$ and $Q[i, j]$, we can also compute the **magnitude** $m$ and **orientation** $\alpha$ of the gradient vector at position $[i, j]$: $m[i, j] = \sqrt{P[i, j]^2 + Q[i, j]^2}$ $\alpha[i, j] = \arctan(\frac{Q[i, j]}{P[i, j]})$

This is exactly the information that we want – it tells us where edges are, how significant they are, and what their orientation is. However, this information is still very **noisy**.
Canny Edge Detector

The first problem is that edges in images are not usually indicated by perfect step edges in the brightness function. Brightness transitions usually have a certain slope that extends across many pixels. As a consequence, edges typically lead to wide lines of high gradient magnitude. However, we would like to find thin lines that most precisely describe the boundary between two objects. This can be achieved with the nonmaxima suppression technique.

Canny Edge Detector

We first assign a sector $\zeta[i, j]$ to each pixel according to its associated gradient orientation $\alpha[i, j]$. There are four sectors with numbers 0, ..., 3:

$$
\begin{align*}
0^\circ & \quad 90^\circ \\
225^\circ & \quad 180^\circ \\
135^\circ & \quad 45^\circ \\
315^\circ & \quad 270^\circ 
\end{align*}
$$

Canny Edge Detector

If the value of $m[i, j]$ is less than either of the $m$-values in these two neighboring positions, set $E[i, j] = 0$. Otherwise, set $E[i, j] = m[i, j]$. The resulting array $E[i, j]$ is of the same size as $m[i, j]$ and contains values greater than zero only at local maxima of gradient magnitude (measured in local gradient direction). Notice that a contour in an image always induces an intensity gradient that is perpendicular to the orientation of the contour. Thus, this nonmaxima suppression technique thins the detected edges to a width of one pixel.

Canny Edge Detector

However, usually there will still be noise in the array $E[i, j]$, i.e., non-zero values that do not correspond to any edge in the original image. In most cases, these values will be smaller than those indicating actual edges. We could then simply use a threshold $\theta$ to eliminate the noise, as we did before. However, this could still leave some isolated edge outputs caused by strong noise and some gaps in detected contours. We can improve this process by using hysteresis thresholding.

Canny Edge Detector

Hysteresis thresholding uses two thresholds - a low threshold $\theta_L$ and a high threshold $\theta_H$. Typically, $\theta_H$ is chosen so that $2\theta_L \leq \theta_H \leq 3\theta_L$. In the first stage, we label each pixel in $E[i, j]$ as follows:

- If $E[i, j] > \theta_H$ then the pixel is an edge,
- If $E[i, j] < \theta_L$ then the pixel is no edge (deleted),
- Otherwise, the pixel is an edge candidate.
Canny Edge Detector

In the second stage, we check for each edge candidate \( c \) whether it is connected to an edge via other candidates (8-neighborhood). If so, we turn \( c \) into an edge. Otherwise, we delete \( c \), i.e., it is no edge. This method fills gaps in contours and discards isolated edges that are unlikely to be part of any contour.

Canny Edge Detector - Example

Let us look at the following (already smoothed) sample image and perform Canny edge detection on it:

```
220 215 210 205 200 195 190 185 180 175 170 165 160 155 150 145 140 135 130 125 120 115 110 105 100 95 90 85 80 75 70 65 60
```

grayscale image intensity values of same image

Canny Edge Detector - Example

Applying the vertical and horizontal gradient filters gives us the following results:

```
P[i, j]  Q[i, j]

2  -25 -30.5 -46.5 -41.5 -70 -75
30  -30 -30.5 -45.5 -41.5 -70 -41.5
30  -30 -30.5 -45.5 -41.5 -70 -75
50  -25 -30.5 -46.5 -41.5 -70 -75
```

```
Q[i, j]

0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

Based on \( P[i, j] \) and \( Q[i, j] \), we can now compute the magnitude and orientation of the gradient:

```
m[i, j]  \( \alpha[i, j] \)

0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

This allows us to compute the sector \( \zeta[i, j] \) for each gradient angle \( \alpha[i, j] \):

```
\zeta[i, j]

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Remember the directions:

```
1 0 3
2 [i, j] 2
3 0 1
```
Finally: hysteresis thresholding ($\theta_L = 30$, $\theta_H = 60$):

Hysteresis stage 1:
- $1 = \text{edge}$; $0 = \text{no edge}$;
- $? = \text{edge candidate}$

Hysteresis stage 2:
- $1 = \text{edge}$; $0 = \text{no edge}$;

Finally we visualize the detected edge pixels ($0 = \text{black}$, $1 = \text{white}$) and indicate the precise edge locations (the pixels' lower right corners) in the original image: