Today’s topics:

- Recap about convolutions
- Recap about edge detection
- Coding assignment on edge detection
Convolutions are:

- Operator $\ast$ mapping image and kernel to images: $I_{\text{out}} = k \ast I_{\text{in}}$
Convolutions are:

- Operator $\ast$ mapping image and kernel to images: $I_{out} = k \ast I_{in}$

- Local: $I_{out}[i, j]$ depends only on neighbors of $I_{in}[i, j]$
Convolutions are:

- Operator $\ast$ mapping image and kernel to images: $I_{out} = k \ast I_{in}$
- Local: $I_{out}[i, j]$ depends only on neighbors of $I_{in}[i, j]$
- Linear: $k \ast (\alpha I_1 + \beta I_2) = \alpha(k \ast I_1) + \beta(k \ast I_2)$
Convolutions are:

- Operator $\ast$ mapping image and kernel to images: $l_{\text{out}} = k \ast l_{\text{in}}$

- Local: $l_{\text{out}}[i, j]$ depends only on neighbors of $l_{\text{in}}[i, j]$

- Linear: $k \ast (\alpha l_1 + \beta l_2) = \alpha (k \ast l_1) + \beta (k \ast l_2)$

- Associative: $(k_1 \ast (k_2 \ast l)) = ((k_1 \ast k_2) \ast l)$
Convolutions are:

- Operator $\ast$ mapping image and kernel to images: $I_{\text{out}} = k \ast I_{\text{in}}$

- Local: $I_{\text{out}}[i, j]$ depends only on neighbors of $I_{\text{in}}[i, j]$

- Linear: $k \ast (\alpha I_1 + \beta I_2) = \alpha(k \ast I_1) + \beta(k \ast I_2)$

- Associative: $(k_1 \ast (k_2 \ast I)) = ((k_1 \ast k_2) \ast I)$

- Shift invariant: $\text{shift}(k \ast I) = k \ast \text{shift}(I)$
Convolutions are:

- Operator \( \ast \) mapping image and kernel to images: \( I_{\text{out}} = k \ast I_{\text{in}} \)
- Local: \( I_{\text{out}}[i, j] \) depends only on neighbors of \( I_{\text{in}}[i, j] \)
- Linear: \( k \ast (\alpha I_1 + \beta I_2) = \alpha(k \ast I_1) + \beta(k \ast I_2) \)
- Associative: \( (k_1 \ast (k_2 \ast I)) = ((k_1 \ast k_2) \ast I) \)
- Shift invariant: \( \text{shift}(k \ast I) = k \ast \text{shift}(I) \)

\[
I'(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} K(i, j)I(x - i, y - j)
\]
Image Filtering, Convolution

- Image filtered by convolving with a filter kernel
- Convolution denoted by “*”

\[ I_{output} = k \ast I_{input} \]

http://en.wikipedia.org/wiki/Kernel_(image_processing)
Edge Detection

http://vision.cs.arizona.edu/nvs/research/image_analysis/edge.html
Edges in images are areas with strong intensity contrasts

- Change is measured by derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero

http://www.pages.drexel.edu/~weg22/edge.htm
Gradient Method

Gradient Vector

\[ \mathbf{g}(x, y) = \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} = \begin{bmatrix} (k_x \ast f)(x, y) \\ (k_y \ast f)(x, y) \end{bmatrix} \]

Gradient Magnitude

\[ |\mathbf{g}| = \left( g_x^2 + g_y^2 \right)^{1/2} \]

Direction

\[ \theta = \tan^{-1}\left( \frac{g_y}{g_x} \right) \]
Image gradient?

Usual continuous derivatives:
\[
\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x + \epsilon, y) - f(x, y)}{\epsilon}
\]

Discrete approximation:
\[
\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}
\]

Translated in image convolutions:
\[
\frac{\partial I}{\partial x} = [-1 \quad 1] \ast I
\]
Sobel kernel

Approximate of the 2D derivative of an image

\[
\begin{align*}
\kappa_x &= \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\
\kappa_y &= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}
\end{align*}
\]
Prewitt kernel

Approximate of the 2D derivative of an image

\[ k_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad k_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \]
Gradient Thresholding
Canny Edge Detection
Canny Edge Detection

Combine noise reduction and edge enhancement.

1. Apply derivative of Gaussian filter
2. Non-maximum suppression
   • Thin multi-pixel wide “ridges” down to single pixel width
3. Hysteresis
   • Accept all edges over low threshold that are connected to edge over high threshold
Derivative of Gaussian kernel

- Need smoothing to reduce noise prior to taking derivative

\[
\begin{array}{cccccc}
0.0121 & 0.0261 & 0.0337 & 0.0261 & 0.0121 \\
0.0261 & 0.0561 & 0.0724 & 0.0561 & 0.0261 \\
0.0337 & 0.0724 & 0.0935 & 0.0724 & 0.0337 \\
0.0261 & 0.0561 & 0.0724 & 0.0561 & 0.0261 \\
0.0121 & 0.0261 & 0.0337 & 0.0261 & 0.0121 \\
\end{array}
\]

- We can use derivative of Gaussian filters
  - because differentiation is convolution, and convolution is associative:

\[
D \ast (G \ast I) = (D \ast G) \ast I
\]
Non-maximum suppression

- The edge direction angle is rounded to one of four angles representing vertical, horizontal and the two diagonals.
- Select the single maximum point across the width of an edge.
- Maximum: The gradient magnitudes of the two neighbors in edge normal direction are smaller.

courtesy of G. Loy
Hysteresis

Idea: real objects usually define continuous edges, noise is disrupted instead

In practice define two thresholds $T_{\text{low}} < T_{\text{high}}$ and classify each a gradient pixel $G$:

- if $G < T_{\text{low}}$ then it’s definitely not an edge
- if $G > T_{\text{high}}$ then it’s definitely a strong edge
- if $T_{\text{low}} < G < T_{\text{high}}$ then it is a weak edge if and only if it is connected to any strong edge through other weak edges
Hysteresis

Strong edges only  
> $T_{\text{high}}$

Weak edges  
> $T_{\text{low}}$

Gap is gone

Strong + connected weak edges

courtesy of G. Loy
Coding assignment this week

Coding assignment available on same repository as last week (https://github.com/tavisualcomputing/viscomp2022) under Exercises/W3

To avoid merging issues with your solution from last week, every week you can pull the new exercise using the version_control.ipynb notebook. (Or you can always clone the repository again)

Assignments:

1. Implement gradient thresholding edge detection

2. Implement Canny edge detection