Segmentation I: Edges and Lines

Prof. Eric Miller
elmiller@ece.tufts.edu
Segmentation

• Problem of breaking an image up into regions are are “interesting” as defined by the nature of the application
  – People walking through an intersection
  – Robots maneuvering through a building
  – Automatic method for red-eye removal
• Many different types of problems and even more solution
• In this class we look at two types of segmentation problems
  – Finding edges and lines from filter outputs
  – Finding regions of interest based on grayscale
Overview

• Last time we developed filters that somehow “find” edges
• Really, the output of these filters carry information about edges
  – Gradient: output is “large” close to edge
  – Laplacian: zero crossings indicate edge position
• Still a lot of work to do to go from filter outputs to edge map
Basic gradient method

• Filter image horizontally and vertically using Sobel, Prewitt, etc to get two images
  \[ y_v[m, n] \] \hspace{1cm} vertical edge image
  \[ y_h[m, n] \] \hspace{1cm} horizontal edge image

• Edges occur where gradient magnitude is large
  \[ e_{mag}[m, n] = \sqrt{y_v^2[m, n] + y_h^2[m, n]} \]

• Also get direction information
  \[ e_{dir}[m, n] = \tan^{-1}\left( \frac{y_v[m, n]}{y_h[m, n]} \right) \]
Basic gradient (cont)

- Edges = those pixels where the gradient is large
- Suggest that we threshold based on magnitude

\[
e[m, n] = \begin{cases} 
1 & e_{mag} > \text{threshold} \\
0 & \text{else}
\end{cases}
\]

threshold: \( \tau \max_{m,n} e_{mag}[m,n] \quad 0 < \tau < 1 \)

- See Matlab
Canny Edge Detector

Many real world methods have been proposed, but Canny is the most accepted

1. Smooth with Gaussian
2. Compute gradients as before
3. Non-maximum suppression:
4. Hysteresis thresholding

Matlab implementation

```matlab
edge_image = edge(image, 'canny', [th_low, th_high], sigma);
```

Low and high thresholds
(see two slides ahead)

Gaussian smoothing width
Non-maximum suppression

Note: I think the book is wrong here on pp. 242-243!

- Move along suspected edges
- Look in perpendicular direction
- If ?? is smaller than neighbors in normal direction, then set to zero
- Makes for thinner edges
Hysteresis thresholding

Single threshold methods either leave lots of artifacts (if threshold is too low), or do not capture the whole edge (if threshold is too high). Canny used two threshold approach that actually tracks the edge as well

• Start at pixel with large gradient magnitude
• March along gradient direction
• Keep as “real” edges all those pixels that exceed some threshold, \texttt{th\_high}, in Matlab
• Keep as real edges all pixels next to those that area between low threshold, \texttt{th\_low}, and \texttt{th\_high}
Laplacian Analysis

Recall that for Laplacian we are looking for zero crossings between extremes of the filter output.

Generally, straight Laplacian is too noisy.

Laplacian after Gaussian smoothing (LoG) is typically used.
Matlab Options

No Gaussian smoothing

```matlab
>> lap = fspecial('laplacian',0);
>> edge_image = edge(f,'zerocross',lap);
```

With Gaussian smoothing

```matlab
>> log = fspecial('log',sz,sigma);
% Size = kernel width
% sigma = Gaussian width

>> edge_image = edge(f,'zerocross',log);
```
Finding lines

• Need to link edge pixels together into lines
• Given a binary image with “1” = edge, how do we find all of the lines in the image?
• Common tool is called the Hough transform
Hough Basics

Start with the equation of line in slope-intercept form

\[ y = ax + b \]

Transform space is defined in terms of the slope and intercept

- Any point, \((x,y)\), on the original line becomes a whole line in \((a,b)\) space
- So if we did this for all the points on the original line, in \((a,b)\) space we would see a ton of lines all intersecting at one \((a,b)\) location
Example

• True line: \( y = 2x-1 \) meaning \( a = 2, \ b = -1 \)
• Some points on this line:
  - \( x = 0 \quad y = -1 \)
  - \( x = 1 \quad y = 1 \)
  - \( x = -1 \quad y = -3 \)
• In \((a,b)\) space our equation is \( b = -ax+y \)
• Evaluate for points along the “true” line
  - \( b=-1 \)
  - \( b = -a+1 \)
  - \( b = a-3 \)
Hough continued

• Idea: take each possible edge point in the image and create a line in \((a,b)\) space.

• “Add” up all of the lines

• Where there is a “real” line in the image, the sum in \((a,b)\) space will be large since all of the \((a,b)\) lines will intersect at the “true” \((a,b)\).
Illustration

- Lots of implementation details
- Can only deal with a “grid” in \( a-b \) space so we have to discretize everything
- Lines do not intersect exactly so we have to deal with approximation accuracy.
- What about vertical lines where \( a \) would be infinite?
Last problem first

- Slope-intercept is not the right equation to use.
- Polar form of a line is better

All points \((x,y)\) on a line distance \(r\) from the origin and making an angle \(\theta\) with the \(x\) axis satisfy

\[x \cos \theta + y \sin \theta = r\]

- No more vertical line problem
- Work in \((r, \theta)\) space in stead of \((a,b)\)
Discretization

• Consider a “table” of radius-angle pair counters each set to 0 initially

• For each possible edge pixel
  – For each $r$ in the table
    • Find the angle in the table that comes closest to satisfying $x \cos \theta + y \sin \theta = r$
    • Add 1 to the radius-angle counter at that table location

• In the end, the counters with the most values represent likely lines

• So we still have to threshold!
Hough and Matlab

• A four step process
  – Find the edges using your favorite method
  – Find the Hough transform using `hough`
  – Find the peaks of the Hough transform using `houghpeaks`
  – Find the line segments in the image using `houghlines`

• See `help` on these commands and the Matlab session for this class
Matlab code

% Step 1: Find the edges using your favorite method
edges = edge(f,'canny',[],3);

% Step 2: Compute the Hough transform
[H T R] = hough(edges,'thetaresolution',0.5,'rholeresolution',0.5);
    H = Hough transform matrix
    T = theta bins
    R = rho bins
    theta- and rhoresolution: sampling densities

% Step 3: Find the peaks
P = houghpeaks(H,15,'threshold',ceil(0.1*max(H(:))));
    P = peak data structure. See help for details
    15 = max number of peaks we are looking for
    threshold = minimum value to be considered a "peak"

% Step 4: Find the line segments associated with those peaks
lines = houghlines(edges,T,R,P,'fillgap',10,'minlength',5);
    lines = line data structure. See help for details
    fillgap = min spacing between two lines for them to be considered distinct
    minlength = minimum length of a line for it to be considered a real line
Matlab Topics

• Use of `edge` to filter images for edge information specifically using `canny`, `Laplacian`, and `Laplacian of Gaussian` methods

• Use of `hough`, `houghpeaks`, and `houghlines` for finding lines in images