Edges

- ➢Edge Detection
- ➢Contour Segmentation
- ➢Hough Transform
- ➢Least Squared Error
- ➢Random Sample Consensus (RANSAC)

1

Edges

- An approach to segmentation.
- The analysis of the discontinuities in an image.

- No correct answer?
- An alternative to region based processing.

2

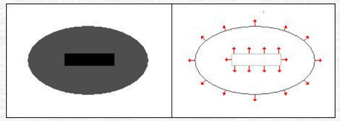
Edge Detection – Topics

- 1^{st} derivative edge detection
- 2^{nd} derivative edge detection
- Multispectral edge detection
- Image sharpening

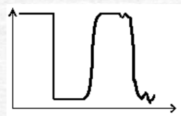
3

Edge Detection – What is an edge?

- Where brightness changes abruptly
- Edges have
  - Magnitude (Gradient)
  - Direction (Orientation)

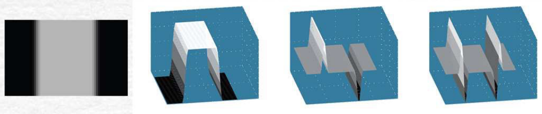


- Edge Profiles
  - Step
  - Real
  - Noisy



4

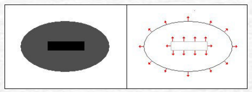
Edge Detection – 1st derivative definitions



- Calculus…
  - Rate of change in two directions
  - Vector variable:
    - Gradient Magnitude
    - Orientation (0º is East)

$$\nabla f(i,j) = \sqrt{\left(\frac{\delta f(i,j)}{\delta i}\right)^2 + \left(\frac{\delta f(i,j)}{\delta j}\right)^2}$$

$$\phi(i,j) = \arctan\left(\frac{\delta f(i,j)}{\delta j}, \frac{\delta f(i,j)}{\delta i}\right)$$



5

Edge detection – Digital images

- Derivatives work on continuous functions
  - Map every point in the input image to the output
- Discrete domain
  - Differences
  - Orthogonal

6

Edge detection – 1st derivative – Roberts

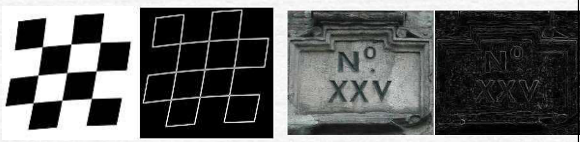
$$\delta_1(i,j) = f(i,j) - f(i+1,j+1)$$

$$\delta_2(i,j) = f(i,j+1) - f(i+1,j)$$

- Convolution Masks

$$h_1(i,j) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad h_2(i,j) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- Sensitivity to Noise
- Binary Images



7

Edge detection – Digital images

- Need to define the partial derivates so that they:
  1. Cross at a single middle point
  2. Preferably cross at the centre of a pixel
  3. Evaluate points which are not too close together
  4. Deal with some degree of image noise

8

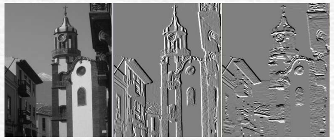
Edge detection – 1st derivative – Compass

- Compass edge detectors:
  - Partial derivatives defined for a number of orientations (typically 8)
  - Prewitt:

$$h_1(i,j)=\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix}\; h_2(i,j)=\begin{bmatrix}0&1&1\\-1&0&1\\-1&-1&0\end{bmatrix}\; h_3(i,j)=\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}\; h_4(i,j)=\begin{bmatrix}-1&-1&0\\-1&0&1\\0&1&1\end{bmatrix}$$

$$h_5(i,j)=\begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}\; h_6(i,j)=\begin{bmatrix}0&-1&-1\\1&0&-1\\1&1&0\end{bmatrix}\; h_7(i,j)=\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}\; h_8(i,j)=\begin{bmatrix}1&1&0\\1&0&-1\\0&-1&-1\end{bmatrix}$$

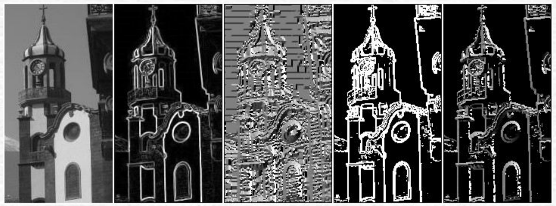
  - Use $h_3(i,j)$ and $h_1(i,j)$ to derive gradient and orientation



Edge detection – 1st derivative – Compass

- Sobel

$$h_1(i,j)=\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}\; h_3(i,j)=\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$$



Edge detection – 1st derivative – Compass

$$\nabla f(i,j) = \sqrt{\left(\frac{\delta f(i,j)}{\delta i}\right)^2 + \left(\frac{\delta f(i,j)}{\delta j}\right)^2}$$

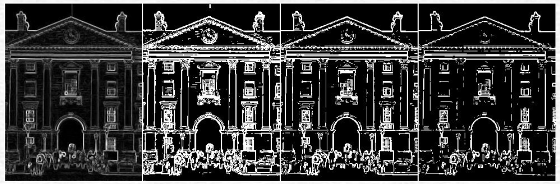
$$\phi(i,j) = \arctan\left(\frac{\delta f(i,j)}{\delta j}, \frac{\delta f(i,j)}{\delta i}\right)$$

$$\nabla f(i,j) = \left|\frac{\delta f(i,j)}{\delta i}\right| + \left|\frac{\delta f(i,j)}{\delta j}\right|$$

```
Mat horizontal_derivative, vertical_derivative;
Sobel( gray_image, horizontal_derivative, CV_32F,1,0 );
Sobel( gray_image, vertical_derivative, CV_32F,0,1 );
Mat abs_gradient, l2norm_gradient, orientation;
abs_gradient = abs(horizontal_derivative) +
                              abs(vertical_derivative);
cartToPolar(horizontal_derivative,vertical_derivative,
                              l2norm_gradient,orientation);
```

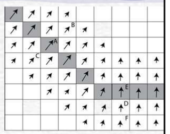

Edge detection – 1st derivative – Thresholding

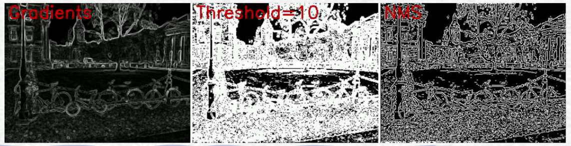
- Simple thresholding
  - Too many points
  - Too few points



Edge detection – 1st derivative – Non maxima suppression

- Use orientation information
- Algorithm:
  - Quantise edge orientations
  - For all points (i,j)
    - Look at the 2 points orthogonal to edge
    - if gradient(i,j) < gradient(either of these 2 points)
      - output(i,j) = 0
      - else output(i,j) = gradient(i,j)
- Now thresholding can be used… or hysteresis thresholding (detailed later…)





Edge detection – 1st derivative – NMS

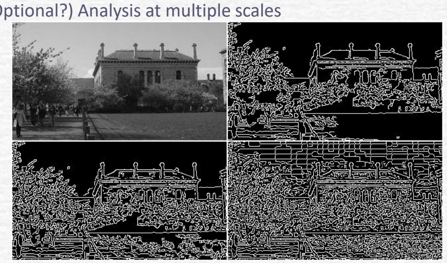
```
nms_result = gradients.clone();
for (int row=1; row < gradients.rows-1; row++)
 for (int column=1; column < gradients.cols-1; column++)
 {
  float curr_gradient = gradients.at<float>(row,column);
  float curr_orientation = orientations.at<float>(row,column);
  // Determine which neighbours to check
  int direction = (((int) (16.0*(curr_orientation)/(2.0*PI))+15)%8)/2;
  float gradient1 = 0.0, gradient2 = 0.0;
  switch(direction)
  {
  case 0:
   gradient1 = gradients.at<float>(row-1,column-1);
   gradient2 = gradients.at<float>(row+1,column+1);
   break;
```


Edge detection – 1st derivative – NMS

```
  case 1:
   gradient1 = gradients.at<float>(row-1,column);
   gradient2 = gradients.at<float>(row+1,column);
   break;
  case 2:
   gradient1 = gradients.at<float>(row-1,column+1);
   gradient2 = gradients.at<float>(row+1,column-1);
   break;
  case 3:
   gradient1 = gradients.at<float>(row,column+1);
   gradient2 = gradients.at<float>(row,column-1);
   break;
  }
  if ((gradient1 > curr_gradient) || (gradient2 > curr_gradient))
   nms_result.at<float>(row,column) = 0.0;
 }
```


Edge detection –Canny

- Combination of first and second derivative detectors
- Second derivative intended to improve localisation
- Thresholding incorporated so result is binary
- (Optional?) Analysis at multiple scales

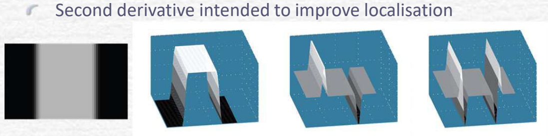


Edge detection – Canny algorithm

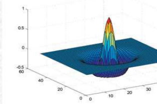
1. Convolve image with Gaussian
2. Compute the first derivative gradients and orientations
3. Apply non-maxima suppression
4. Apply a double threshold to find strong & weak edge points.
5. Threshold edges with hysteresis
   - Remove any weak edges points not connected to strong edge points.

Edge detection – Canny – uses 2^{nd} derivative?

- According to the original paper Canny is supposed to be a combination of first and second derivative detectors
  - Second derivative intended to improve localisation



- Uses a filter like the Laplacian (of Gaussian)

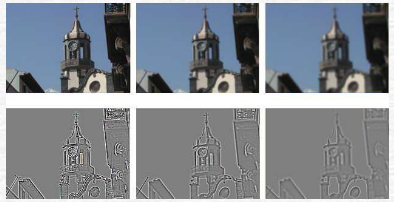


$$h(i,j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Need to fine zero-crossings.

Edge detection –Canny

- Also according to the original paper the image is supposed to be analysed at multiple scales
  - Smooth image using different Gaussians
  - Look for correspondences across scale to identify significant discontinuities

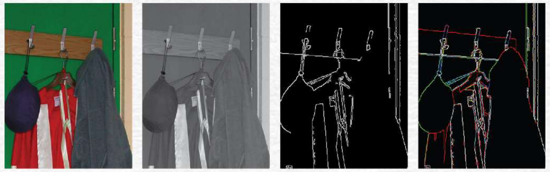


Edge detection – advanced – Canny


```
Canny( gray_image, binary_edges, 100, 200 );
```

Multispectral edge detection

- Grey levels are ordered, colours are not.
- Multispectral edge detection is addressed in 3 ways:
  1. Output fusion
     - Combine edges



  2. Multidimensional gradient methods
     - Combine gradients
  3. Vector methods
     - Distance between colours in 3D

21

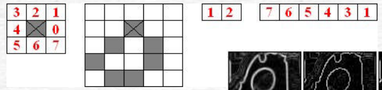
Contour Segmentation – Topics

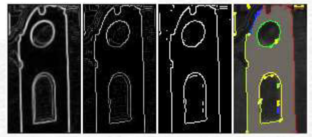
- Edge data representations
- Border Detection
- Line segment extraction

22

Contour Segmentation – Boundary Chain Codes

- Each chain contains
  - Start point
  - A list of orientations

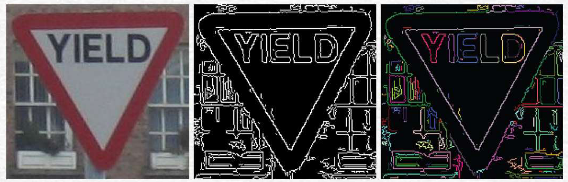




- Features!
  - Orientation & Scale dependent
  - Slightly position dependent
  - Can be smoothed to reduce boundary noise
  - Difficulties obtaining consistent representation from image.

23

Contour Segmentation – Boundary Chain Codes



24

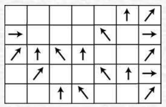
Contour Segmentation – Boundary Chain Codes

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours( binary_edge_image, contours, hierarchy,
            CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE );
```

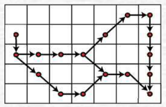
```
for (int contour_number=0;
      (contour_number<contours.size()); contour_number++)
{
  Scalar colour( rand()&0xFF, rand()&0xFF, rand()&0xFF );
  drawContours( display_image, contours, contour_number,
                          colour, 1, 8, hierarchy );
}
```


Contour Segmentation

- Represent an edge image as a **graph**:
  - Node n_i
    - Corresponds to pixel x_i
    - Associated cost $s(x_i)$ and orientation $\phi(x_i)$
  - n_i, n_j are connected by an **arc** if
    - Pixels x_i, x_j are 8-connected neighbours
    - $\phi(x_i)$ and $\phi(x_j)$ match the local border:
      - x_j must be one of 3 neighbours in the direction $[\phi(x_i)-\pi/4, \phi(x_i)+\pi/4]$
      - $| \phi(x_i) - \phi(x_j) | < \pi/2$

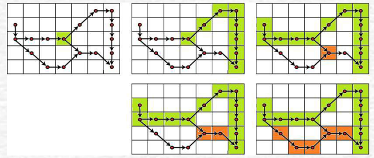


- When expanding edge chains consider
  - Strength of edges
    - E.g. $(\max_{image} s(x_k)) - s(x_i)$
  - Border curvature
    - diff[$\phi(x_i) - \phi(x_j)$]



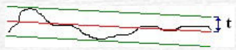
Contour Segmentation

- Purpose: Segment all edge chains in an image
- Algorithm:
  1. Search for the strongest (unused) node in the graph. If none go to step 5.
  2. Expand all the edges in front of the specified edge
  3. Expand all the edges behind of the specified edge
  4. If the edge chain consists of > 3 pixels store it. Go to step 1.
  5. Modify edge chains to fill small breaks and to remove duplicate contours.
  6. Repeat step 5 until stable.



Contour Segmentation – Segment sequences

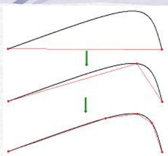
- Boundary
  - Sequence of segments
- Segment
  - Start and end points: x_1 and x_2
  - Type: straight line, some type of curve, etc.
  - Other parameters
- Accurate polygonal representation
  - Must define acceptable tolerances
  - Vertices – where?



Contour Segmentation – Straight line extraction example

Contour Segmentation – Obtaining polygonal segments

- Recursive boundary splitting
  - Split at furthest point
  - Keep going until with tolerance



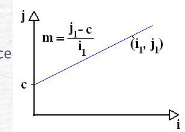
```
vector<vector<Point>> approx_contours(contours.size());
for (int contour_number=0;
        (contour_number<contours.size()); contour_number++)
    approxPolyDP( Mat(contours[contour_number]),
            approx_cont ours[contour_number], 3, true );
```

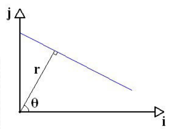

Hough transform

- Direct transformation from image space to probability of the existence of some feature…
  - Lines
  - Circles
  - Generalised shapes

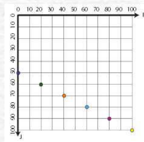
Hough Transform – Line detection

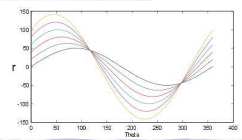
- Line equation:
  - $j = m.i + c$
    - Lines in Hough space
    - What about $i = c$



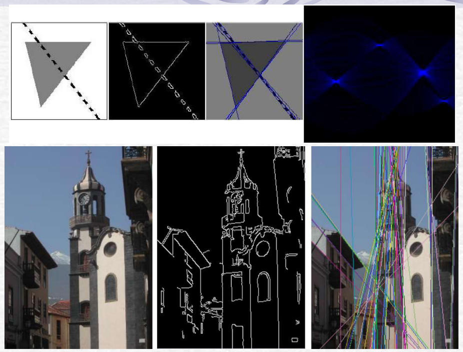


  - $r = i.\cos\theta + j.\sin\theta$
    - Sinusoidal curves





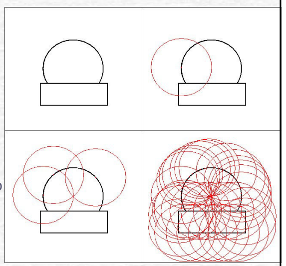
Hough Transform – Line detection



33

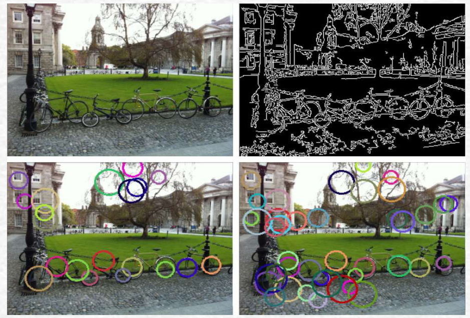
Hough Transform – Circle detection

- Equation of a circle $(i-a)^2 + (j-b)^2 = r^2$
  - Assume constant radius *r*
- Transform
  - From Image space (x, y)
  - To Hough space (a, b)
- Algorithm
  - Initialise accumulator to 0
  - For every edge point
    - Increment cells in accumulator corresponding to all possible circle centers
  - Search for Maximums



34

Hough Transform – Circle detection



35

Hough transform – in OpenCV

- Hough for lines:

```
vector<Vec2f> hough_lines;
HoughLines( binary_edge_image, hough_lines, 1, PI/200.0, 60);
```

- Probabilistic Hough for line segments:

```
vector<Vec4i> line_segments;
HoughLinesP( binary_edge_image, line_segments, 1.0,
                                          PI/200.0, 20, 20, 5);
```

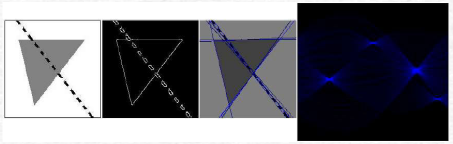
- Hough for circles:

```
vector<Vec3f> circles;
HoughCircles( gray_image, circles, CV_HOUGH_GRADIENT,
                                          2,20,300,20,5,15);
```


36

Hough Transform – Resolution of Hough Space

- Circle
  - (a, b) could be anywhere up to *r* outside the image space
  - Can detect the circle centre to higher precision than the image points
- Lines
  - $-\text{dist}(0,0, M,N) \leq r \leq +\text{dist}(0,0, M,N)$
  - $-\pi \leq \theta \leq \pi$
  - Precision of s and θ application dependent



37

Hough Transform – Efficient Implementation

- Use edge orientations
  - Restrict the mapping into space
- Use half the accumulator
  - For lines only
- Multi-resolution
  - Process at a small resolution
  - Higher resolution to find accurate data
- Problem
  - What if the size of the circle is unknown
  - 3-D Accumulator??

38

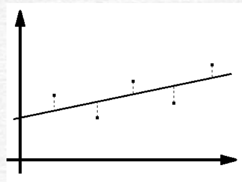
Other techniques for estimating pose (lines ++)

- Common to have a number of data points which may
  - form part of a feature (such as a straight line) OR
  - represent possible matches with some known object features
- We need to determine the best possible position or pose of this feature or object
- Techniques:
  - Least squared error solution
  - RANSAC

39

Least squared error solution

- Linear fit which best matches the data.
- For a straight line it minimises the sum of the vertical residuals



  - Line equation: $y = m.x + c$
  - First compute the slope m
  - Then the intercept c

40

Least squared error solution

Given (x_i, y_i) where $i = 1..N$

$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$ $\qquad$ $\mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i$

$\sigma_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)^2}$ $\qquad$ $\sigma_y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu_y)^2}$

Pearson's correlation coefficient: $\rho_{xy} = \frac{cov_{xy}}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{N}(x_i-\mu_x)(y_i-\mu_y)}{\sigma_x \sigma_y}$

$m = \rho_{xy}\frac{\sigma_y}{\sigma_x}$ $\qquad$ $c = \mu_y - m.\mu_x$

41

Least squared error solution – Problems

- Assumes line is not vertical
- Assumes that error distribution is normal
- Assumes that the points which should be included in the regression are known (i.e. segmented)
- Assumes no significant outliers

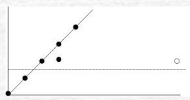


Figure 10.7: Influence of an outlier in least squares line fitting. With 6 valid data points and 1 gross outlier (white), the best line is shown in solid. Least squares, followed by discarding the worst outlier, reaches the dotted line after 3 discards [Fischler and Bolles, 1981]. © *Cengage Learning 2015.*

42

Random Sample Consensus (RANSAC)

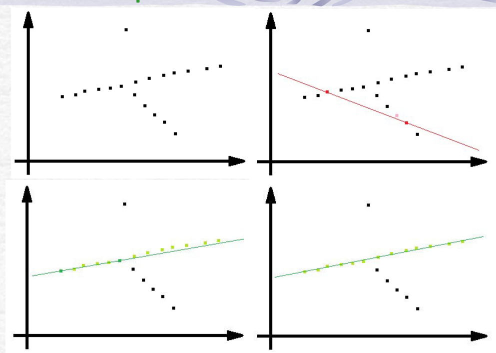
- Uses the minimum number of data points (m) to determine the model
  - For a straight line *m=2*

Technique

1. Randomly select the minimum number (m) of data points from the N data points (x_i, y_i) where $i = 1..N$
2. Determine the model from the selected data points
3. Determine how many data points are within some tolerance of the model – the consensus set
4. If the consensus set is not big enough (i.e. is smaller than some pre-set threshold) go back to step 1 (OR fail if tried too often).
5. If the consensus set was big enough, re-compute the model using all points in the consensus set.

43

RANSAC Example



44

RANSAC Example

Figure 10.8: Use of RANSAC in panoramic stitching (the original images are, of course, in color): (a) An image pair that overlaps. (b) Each dot represents an 'interest point', whose vector matches a point in the other image. (c) The illustrated dots are the RANSAC 'inliers'; they conform to the hypothesis of the best-performing candidate homography that is tried. (d) Prior to smoothing, the resulting overlap. *Courtesy of D. Lowe, M. Brown, University of British Columbia.*

RANSAC Examples

From http://web.engr.illinois.edu/~nanchen2/cs498dwh/proj5/