Computer Vision

Ch.9 Object Detection #2

Prof. Po-Yueh Chen (陳伯岳)
E-mail: pychen@cc.ncue.edu.tw
Ext: 8440

NCUE CSIE

Edge Detection & Line Extraction

- Background about image edge
- Image gradient
- **Canny edge detector**
- **Hough line transform for line extraction**

Canny Edge Detector (1/21)

Gradient

Canny

The performance of canny edge detector is superior in general to the others discussed so far.

Canny Edge Detector (2/21)

Canny's approach is based on three basic objectives:

1.Low error rate.

All edges should be found, and there should be no false responses.

2.Edge points should be well localized.

The edges located must be as close as possible to the true edges.

- That is, the distance between a point marked as an edge by the detector and the center of the true edge should be minimum.

3. Single edge point response.

The detector should return only one point for each true edge point.

- The detector should not identify multiple edge pixels where only a single edge point exists.

Canny Edge Detector (3/21)

- **Steps of Canny edge detection**
  1. Smooth the input image with a Gaussian filter
  2. Compute the gradient magnitude and angle images
     - ➢ You can use Sobel, Prewitt, or Scharr filters.

Original image

Smoothed image

Gradient image

Canny Edge Detector (4/21)

- ✓ Gradient image typically contains wide ridges around the local maxima.
- ✓ The next step is to thin those ridges.

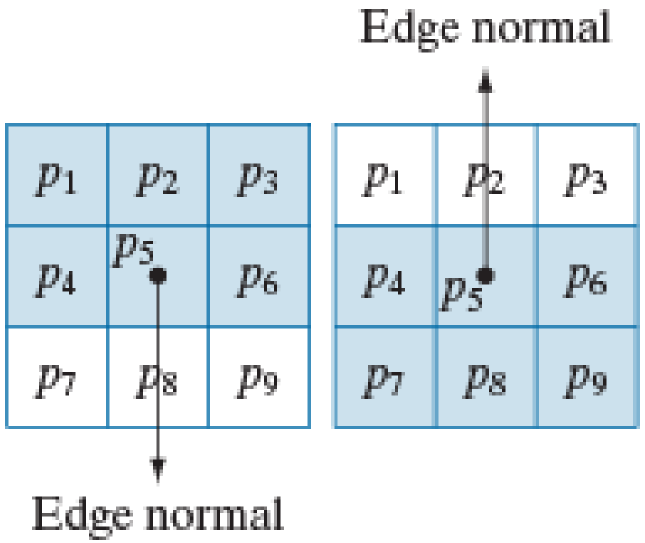
Wide ridge

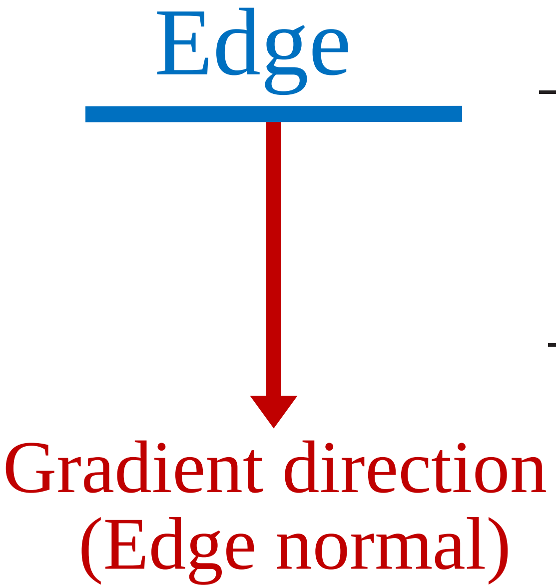
Canny Edge Detector (5/21)

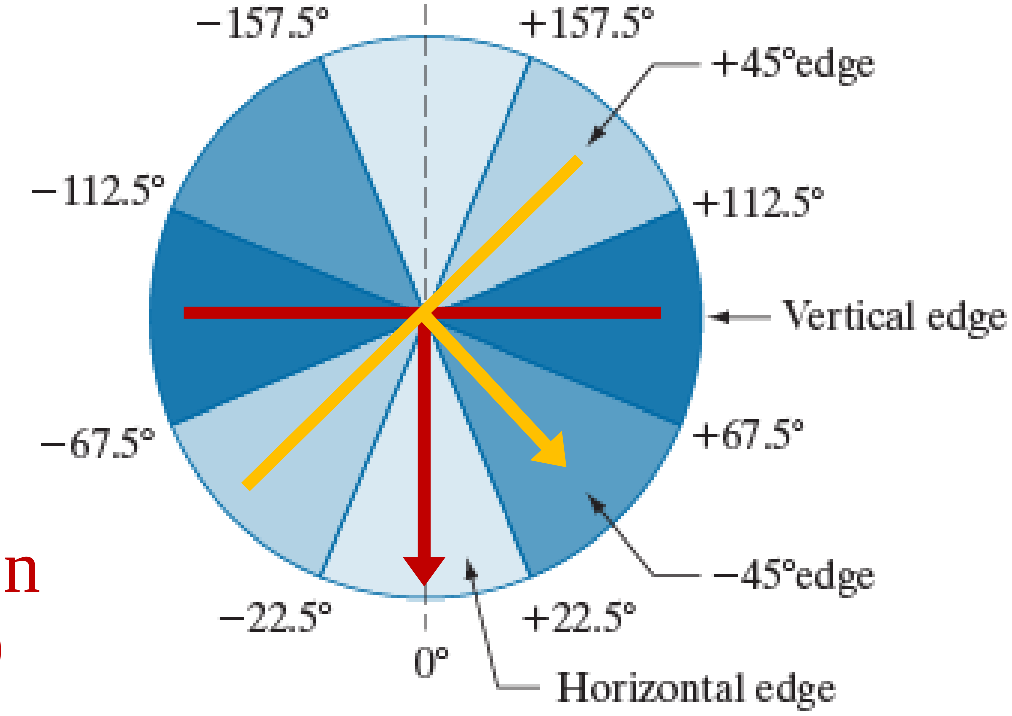
- **Steps of Canny edge detection**

3. Apply non-maximal suppression (NMS) to the gradient magnitude image to eliminate false responses.

    a. classify the gradient direction into four: horizontal, vertical, $+45^{\circ}$, and -45°



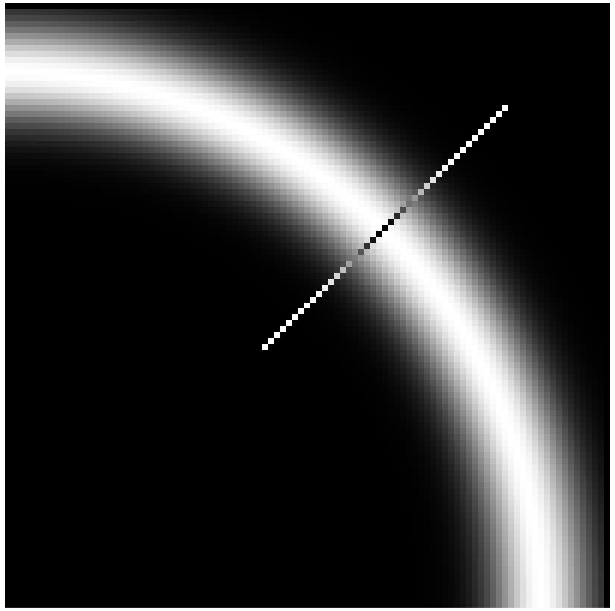


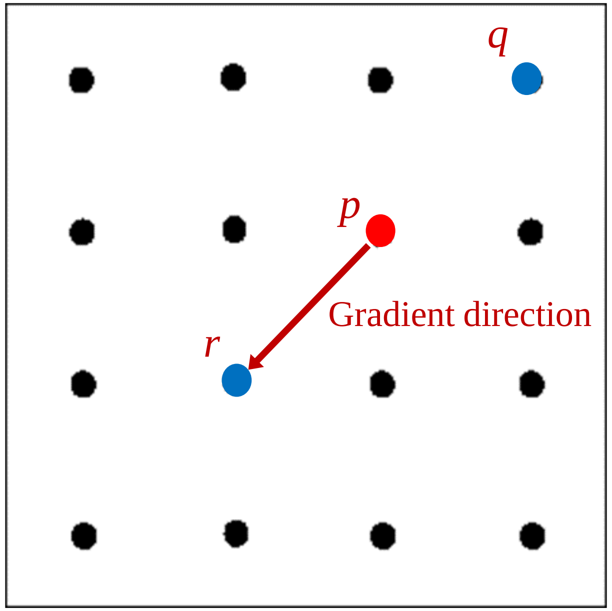


Canny Edge Detector (6/21)

- **Steps of Canny edge detection**

3. Apply non-maximal suppression (NMS) to the gradient magnitude image to eliminate spurious response.
   b. For a point *p*, if its gradient magnitude is not more than both of the neighbors (*q* and *r*) along the gradient direction (i.e., *p* is not max), set the gradient of *p* to 0 (suppression).





Canny Edge Detector (7/21)

- **Steps of Canny edge detection**
  3. Apply non-maximal suppression (NMS) to the gradient magnitude image to eliminate spurious response.

Directly conduct thresholding on the gradient image

Multiple responses for an edge → Bad localization

With non-maximal suppression

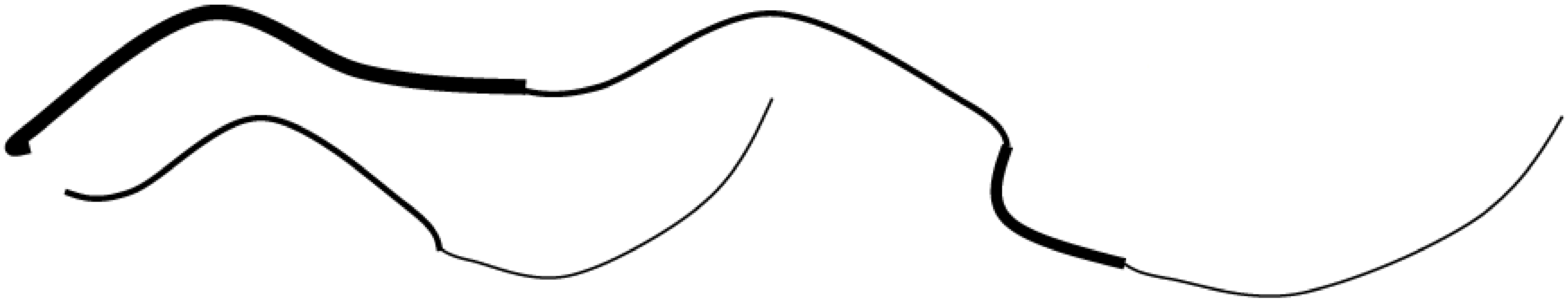
Well located single response for each edge point

Canny Edge Detector (8/21)

- **Steps of Canny edge detection**

4. Use double thresholding and connectivity analysis to detect and link edges.

a. Use high/low threshold to detect strong/weak edge pixels
b. Strong ones are marked as valid edge pixels immediately
c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.
→This is based on the connectivity of edges.



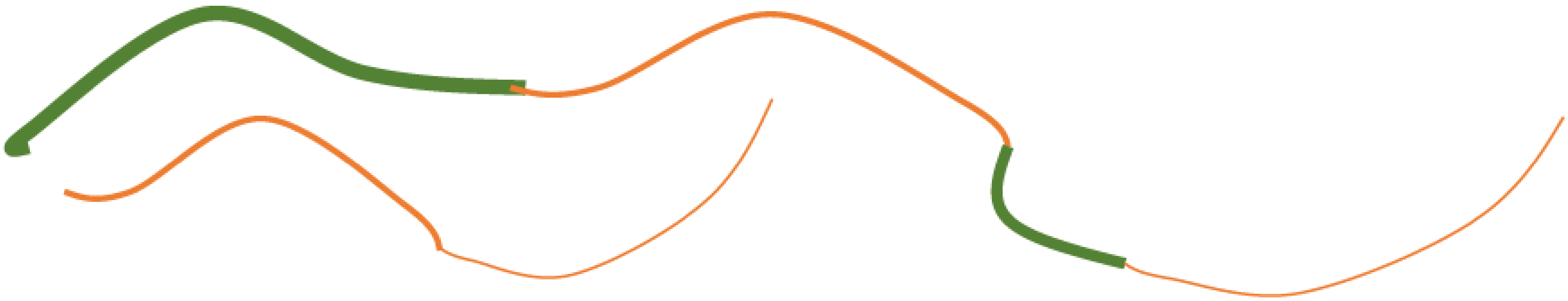
The thickness indicates the gradient magnitude

Canny Edge Detector (9/21)

- **Steps of Canny edge detection**

4. Use double thresholding and connectivity analysis to detect and link edges.

a. Use high/low threshold to detect strong/weak edge pixels
b. Strong ones are marked as valid edge pixels immediately
c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.
→This is based on the connectivity of edges.



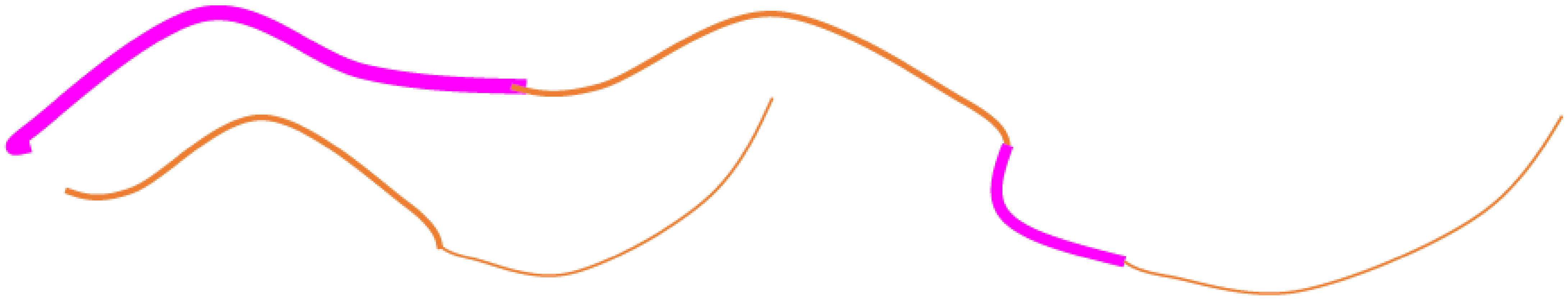
The thickness indicates the gradient magnitude

Canny Edge Detector (10/21)

- **Steps of Canny edge detection**

4. Use double thresholding and connectivity analysis to detect and link edges.

a. Use high/low threshold to detect strong/weak edge pixels
b. Strong ones are marked as valid edge pixels immediately
c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.
→This is based on the connectivity of edges.



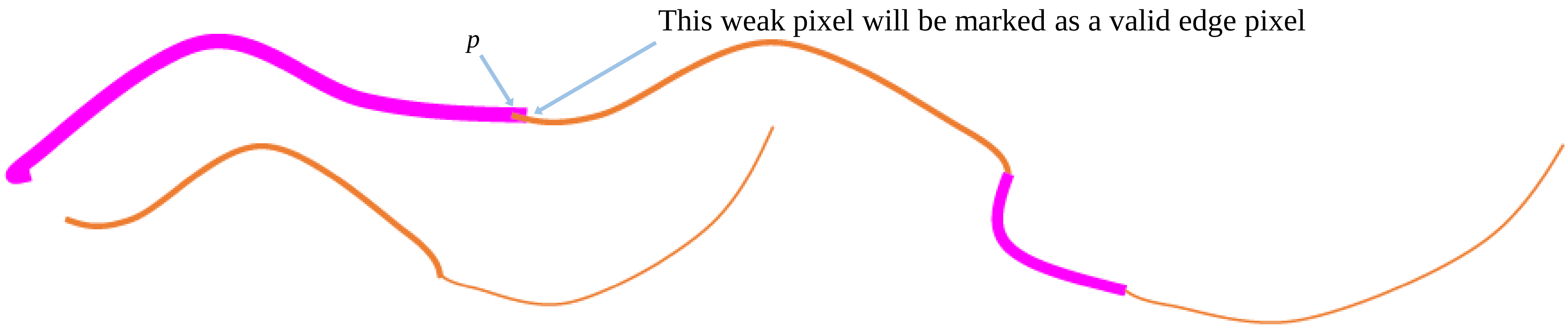
The thickness indicates the gradient magnitude

Canny Edge Detector (11/21)

- **Steps of Canny edge detection**

4. Use double thresholding and connectivity analysis to detect and link edges.

a. Use high/low threshold to detect strong/weak edge pixels
b. Strong ones are marked as valid edge pixels immediately
c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.
→This is based on the connectivity of edges.



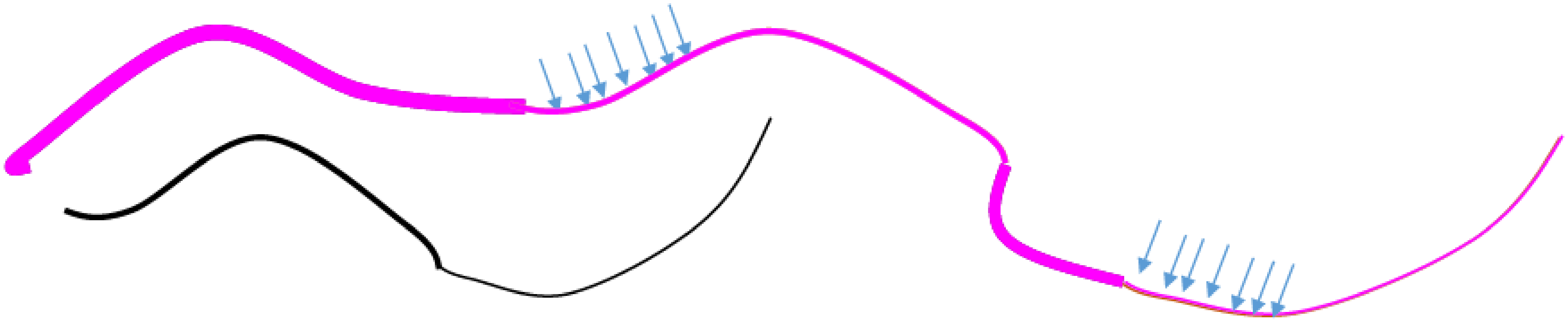
The thickness indicates the gradient magnitude

Canny Edge Detector (12/21)

- **Steps of Canny edge detection**

4. Use double thresholding and connectivity analysis to detect and link edges.

   a. Use high/low threshold to detect strong/weak edge pixels
   b. Strong ones are marked as valid edge pixels immediately
   c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.
      →This is based on the connectivity of edges.



The thickness indicates the gradient magnitude

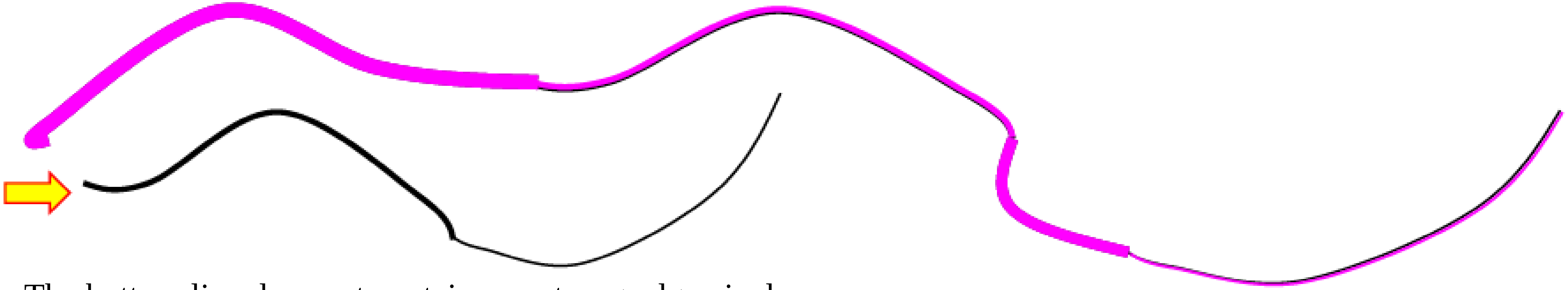
⇨ These weak pixels are marked as valid edge pixels one by one.

Canny Edge Detector (13/21)

- **Steps of Canny edge detection**

4. Use double thresholding and connectivity analysis to detect and link edges.

   a. Use high/low threshold to detect strong/weak edge pixels
   b. Strong ones are marked as valid edge pixels immediately
   c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.
      →This is based on the connectivity of edges.



- The bottom line does not contain any strong edge pixel.
- As the result, the pixels on the bottom line will not be marked as edge pixels.

The thickness indicates the gradient magnitude

Canny Edge Detector (14/21)

- **Steps of Canny edge detection**

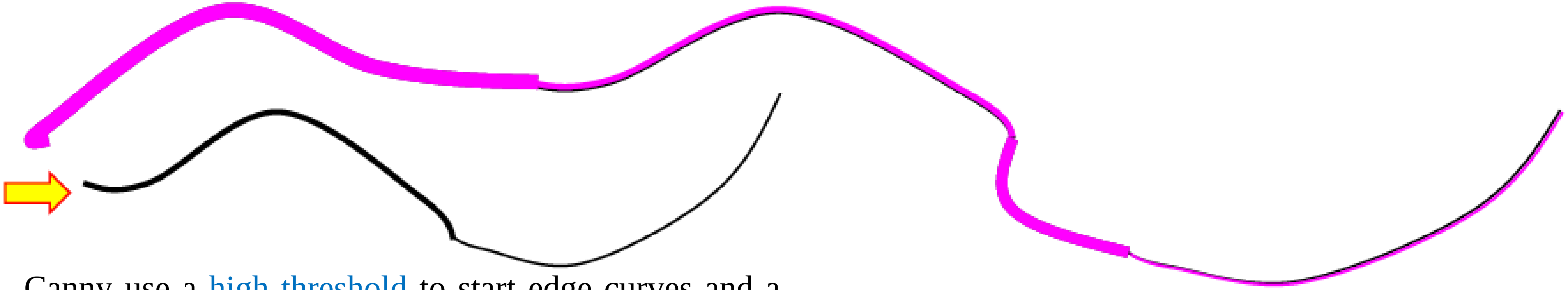
4. Use double thresholding and connectivity analysis to detect and link edges.

   a. Use high/low threshold to detect strong/weak edge pixels

   b. Strong ones are marked as valid edge pixels immediately

   c. For a valid edge pixel *p*, mark all the weak ones that are connected to *p* (8-connectivity) as valid edge pixels.

   →This is based on the connectivity of edges.



- Canny use a high threshold to start edge curves and a low threshold to continue them.
- Canny recommended a ratio of high : low threshold between 2:1 and 3:1.

The thickness indicates the gradient magnitude

Canny Edge Detector (15/21)

➢ Summary

1. Smooth the input image with a Gaussian filter.
2. Compute the gradient magnitude and angle images (Scharr, Sobel, Prewitt).
3. Apply non-maximal suppression (NMS) to the gradient magnitude image to eliminate spurious response.
4. Use double thresholding and connectivity analysis to detect and link edges.

Canny Edge Detector (16/21)

➢ Code (Review)

Syntax:

✓ **Canny recommended a ratio of high : low threshold between 2:1 and 3:1.**

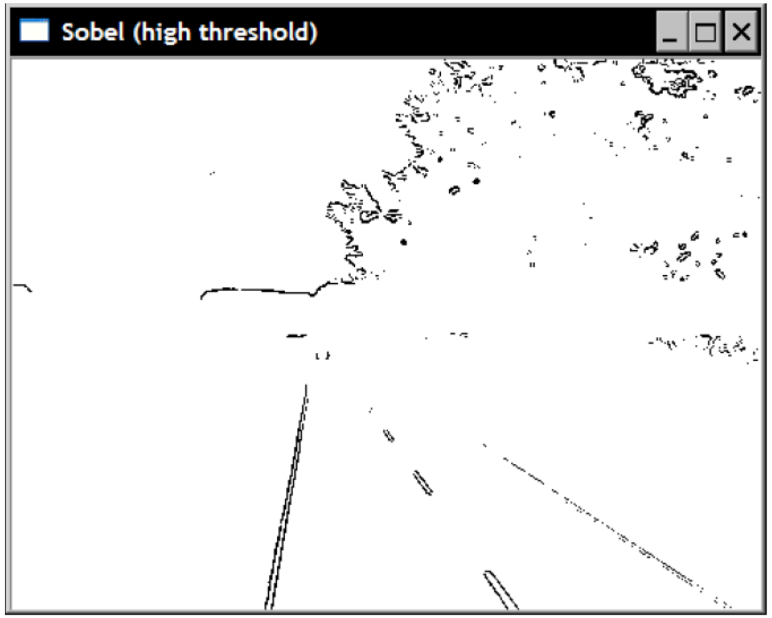
Canny(src, edges, threshold1, threshold2, apertureSize, L2gradient=false);

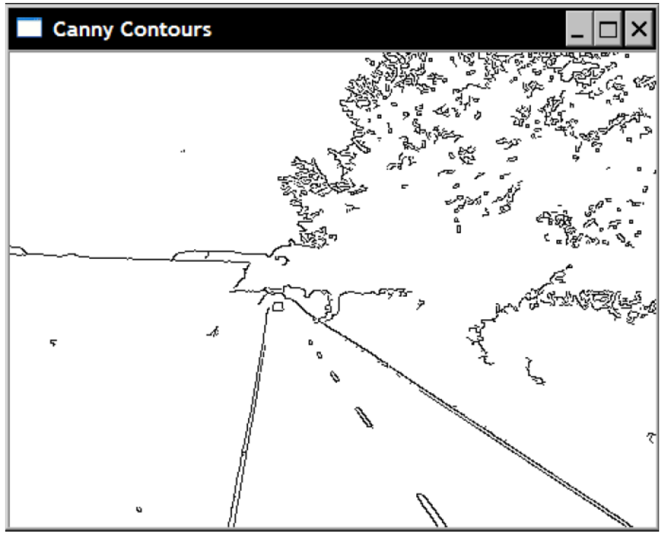
src – Source, an 8-bit single-channel image.
edges – output edge map; it has the same size and type as image .
threshold1 – First threshold for the hysteresis procedure.
threshold2 – Second threshold for the hysteresis procedure.
apertureSize – Aperture size for the Sobel() operator.

L2gradient – A flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether L_1 the default norm $= |dI/dx| + |dI/dy|$ is enough (L2gradient=false).

Canny Edge Detector (17/21)

➢ Example





Canny (image, contour, 125, 350)

Canny Edge Detector (18/21)

➢ Example

After smoothing, “irrelevant” features can be reduced.

Canny parameters:

$$T_L = 0.04, T_H = 0.1$$

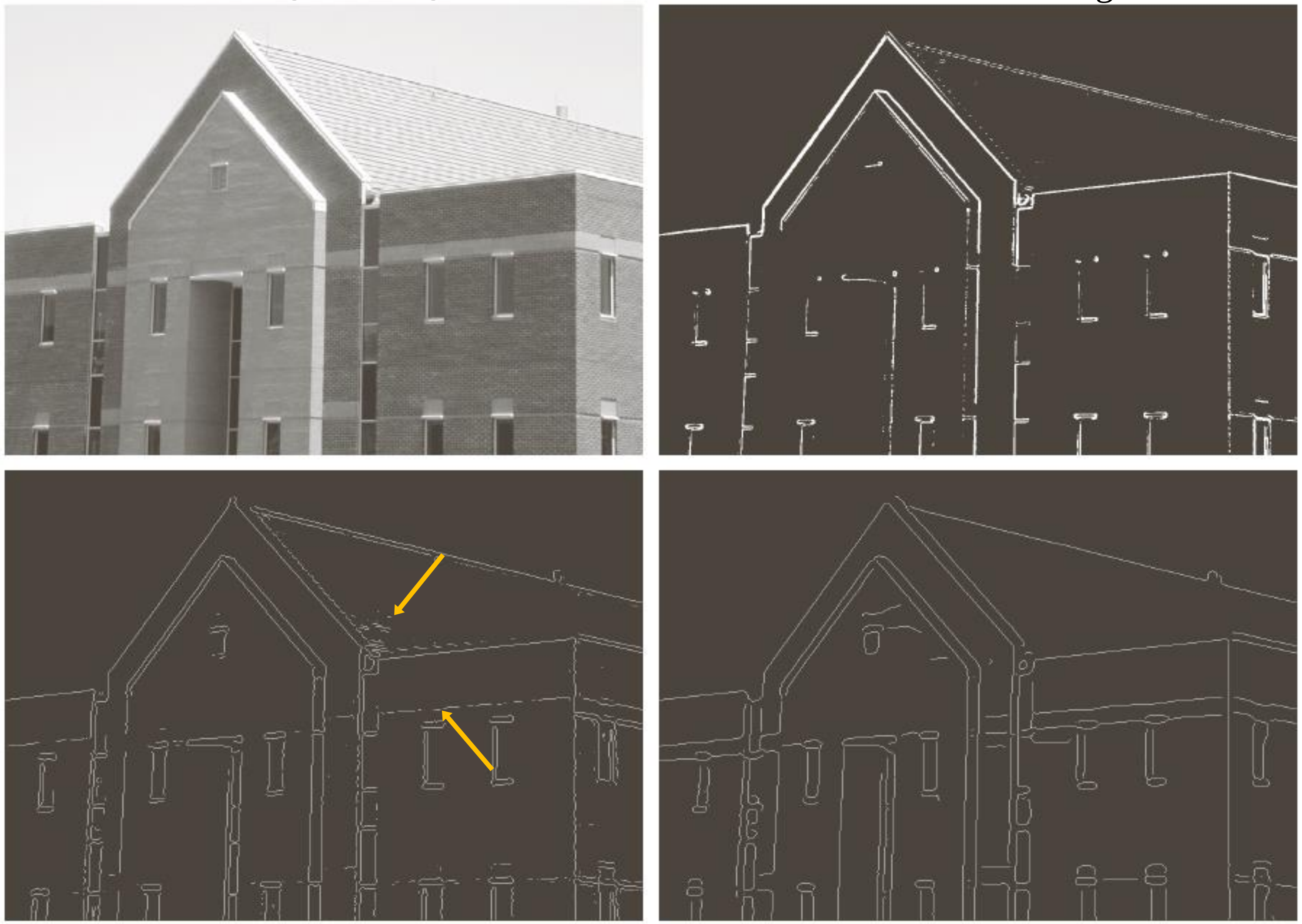
Kernel size: 25×25

Canny algorithm is a good choice for edge detection.

- Better continuity, thinness, and straightness

(a) Original image

(b) Threshold gradient of the smoothed image



(c) Marr-Hildreth

(d) Canny

Canny Edge Detector (19/21)

➢ Example

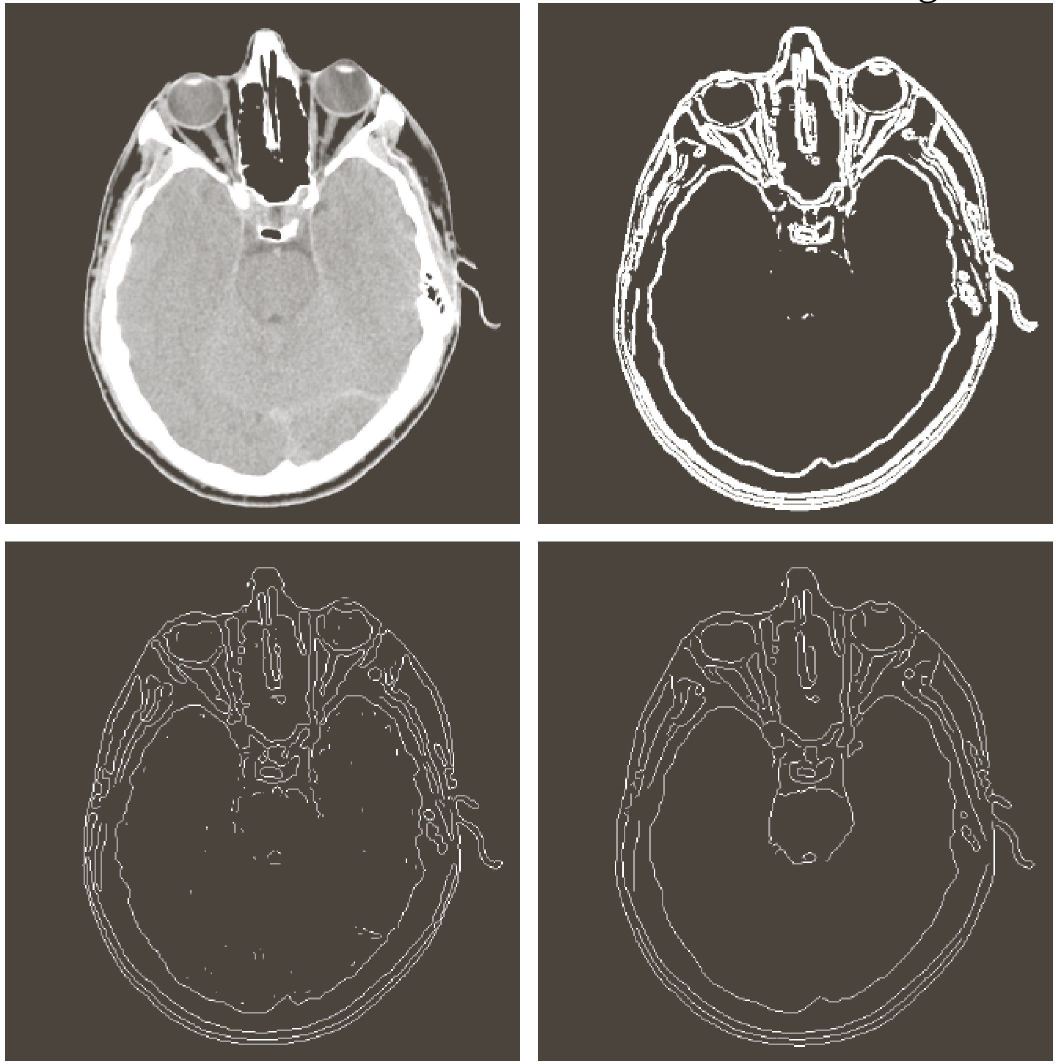
Canny parameters:

$T_L = 0.05, T_H = 0.15$

Kernel size: 13×13

(a) Original image

(b) Threshold gradient of the smoothed image

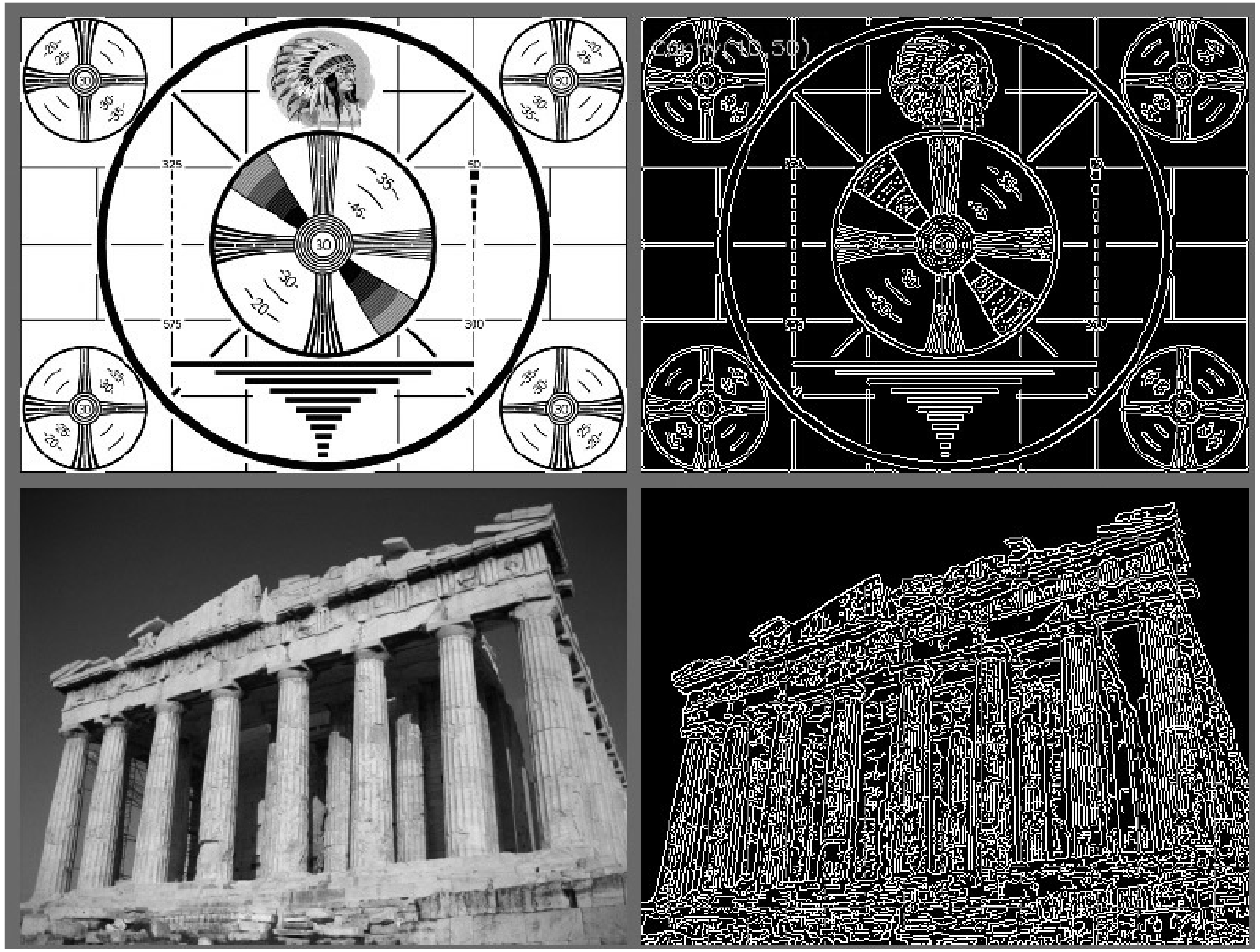


(c) Marr-Hildreth

(d) Canny

Canny Edge Detector (20/21)

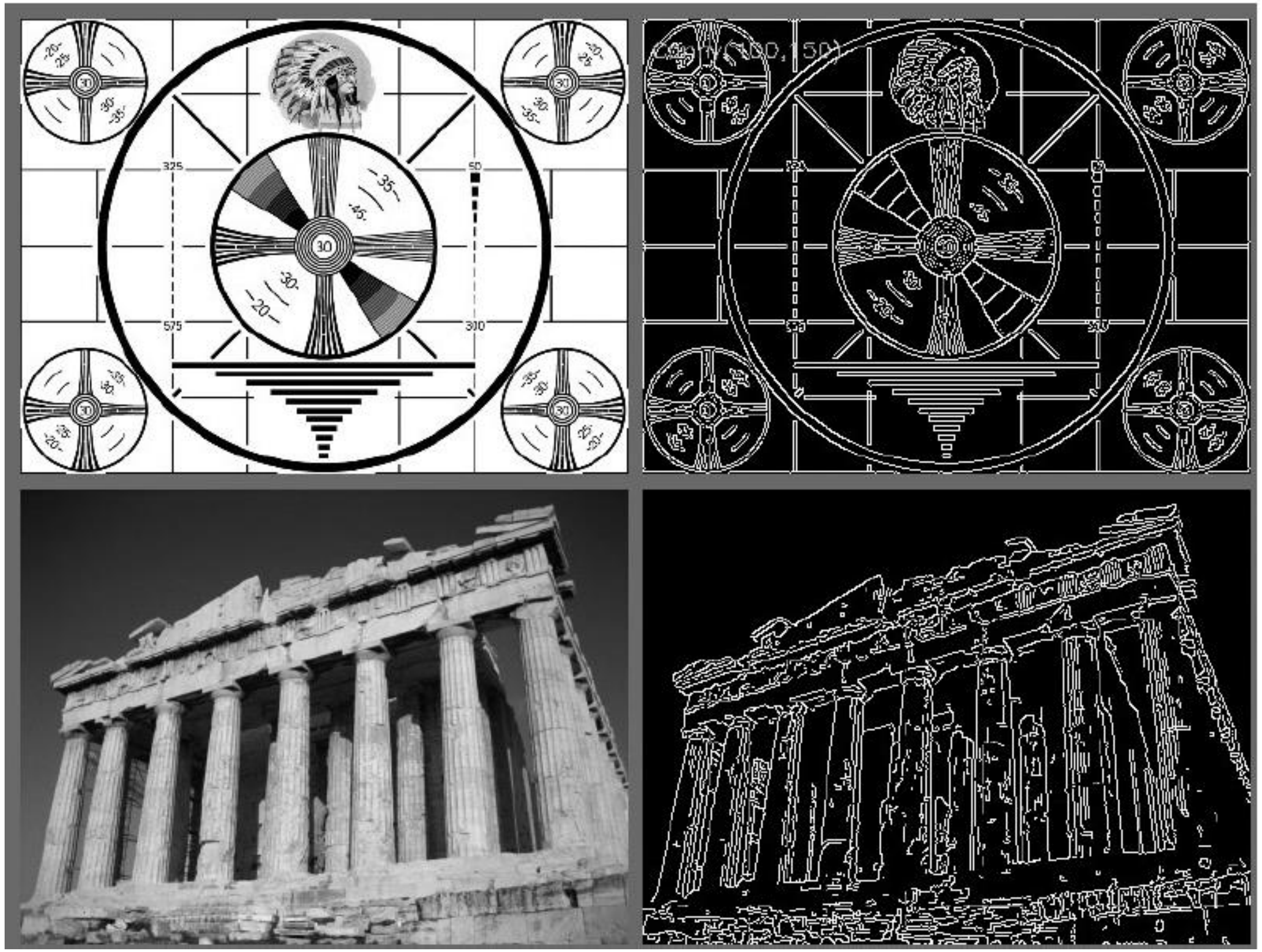
- **Example**



Canny (image, contour, 10, 50)

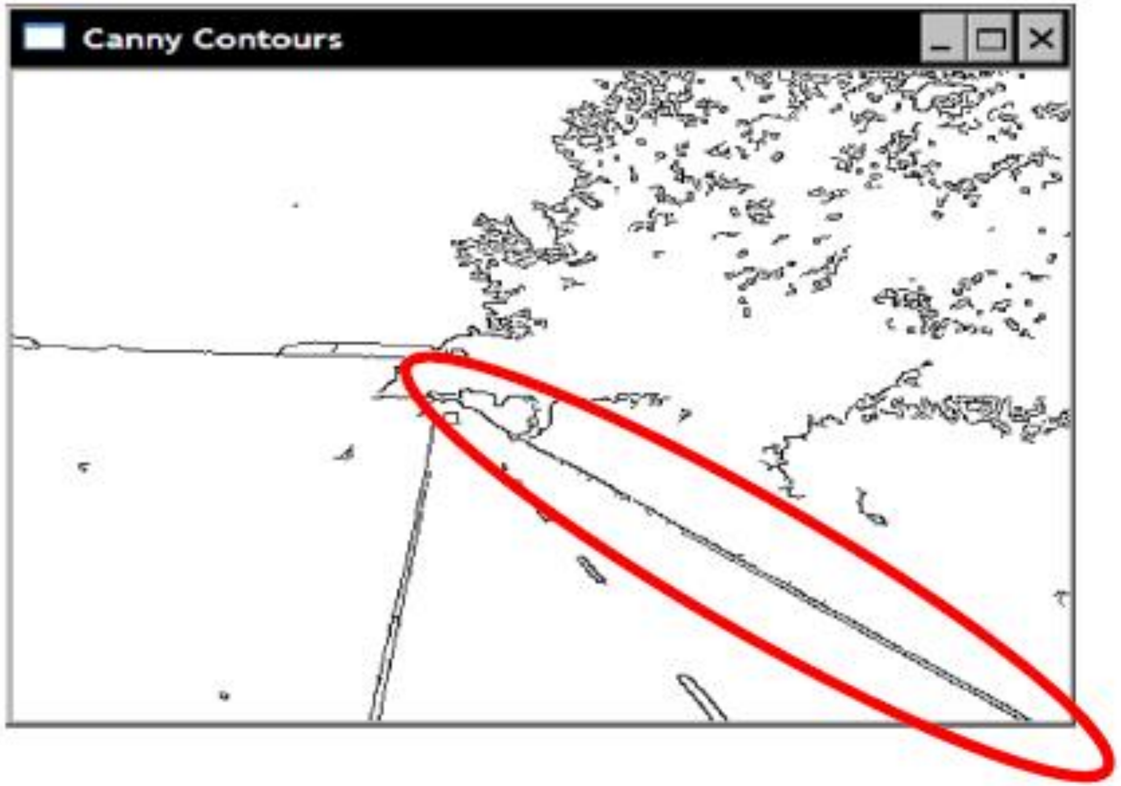
Canny Edge Detector (21/21)

- **Example**



Canny (image, contour, 100, 150)

Hough transform for line extraction



How do we detect lines from edges?

Hough transform (1/11)

General equation of a straight line：

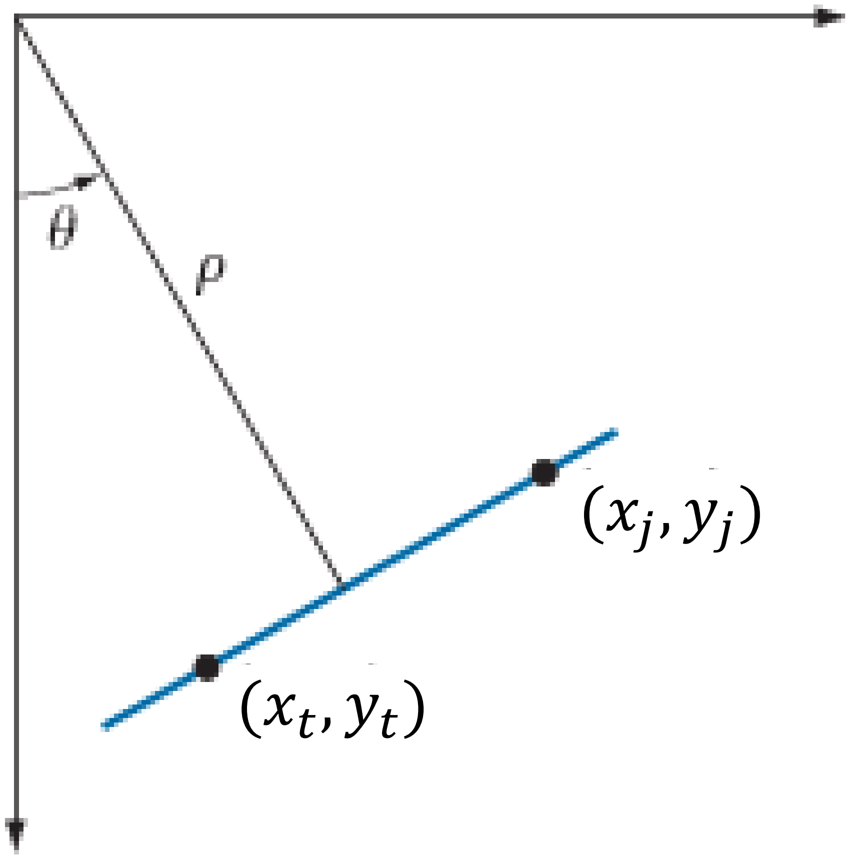
$$ax + by + c = 0$$

Use the normal representation of a line：

$$x\cos\theta + y\sin\theta - \rho = 0$$

θ: The angle of the line normal.

ρ: The distance between the line and the origin.

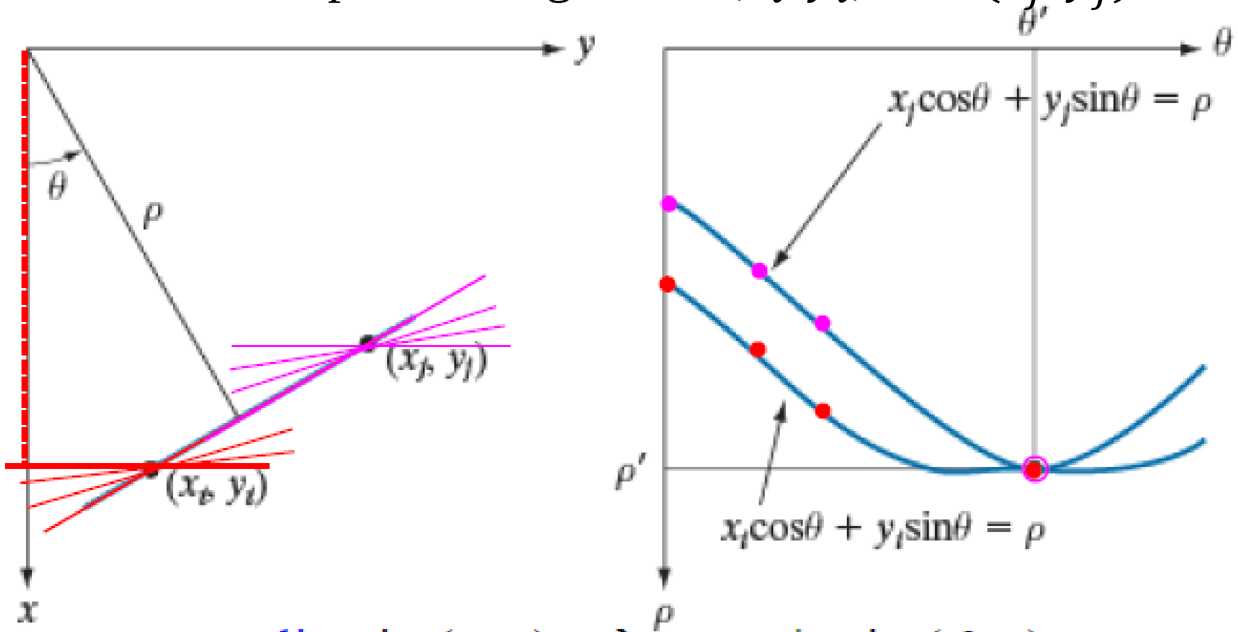


Hough transform (2/11)

- ✓ A line in the *xy*-plane corresponds to a point in the $\theta\rho$-plane.
- ✓ For a point (x, y), we can find many lines passing through it.
- ✓ Each curve in the right figure represents the family of lines that pass through a particular point (x_i, y_i)[or (x_j, y_j)].
- ✓ The intersection point (θ, ρ) corresponds to the line that pass through both (x_i, y_i) and (x_j, y_j).

θ: The angle of the line normal.

ρ: The distance between the line and the origin.



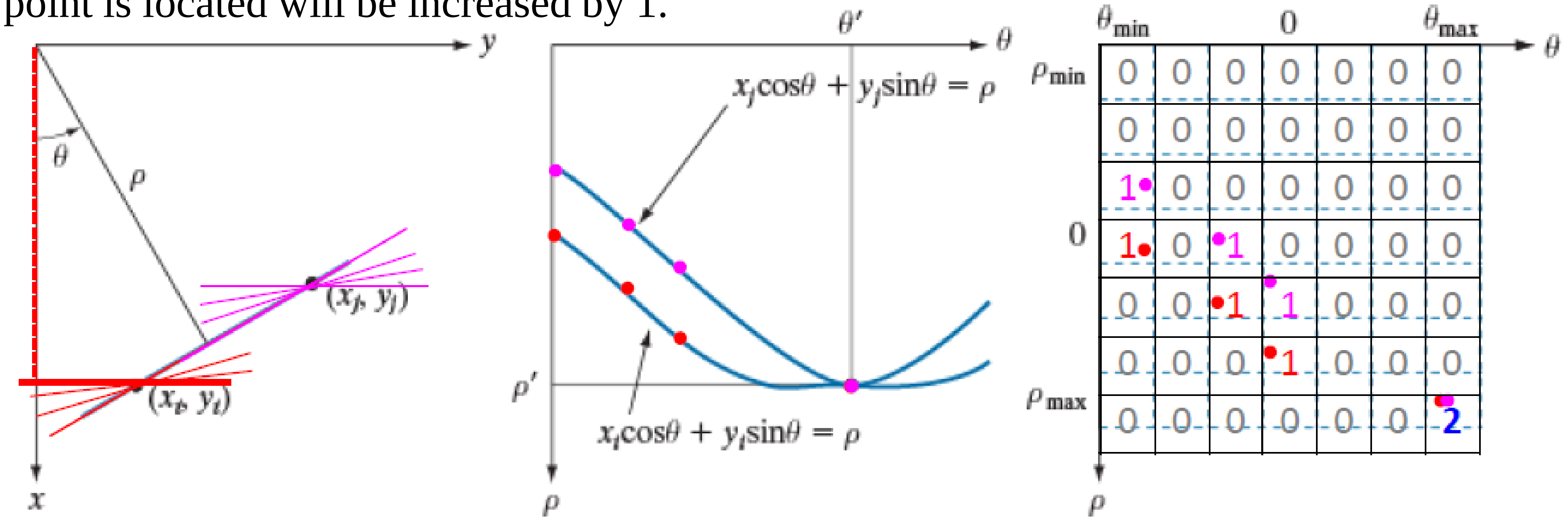
a line in (x, y) → a point in (θ, ρ)

$x\cos\theta + y\sin\theta - \rho = 0$

Hough transform (3/11)

➢ Finding lines by Hough Transform

- ✓ Divide the $\theta\rho$-plane into so-called accumulator cells.
- ✓ For a point (x, y), find the family of lines that pass through it.
- ✓ Each line in xy-plane corresponds to a point in the $\theta\rho$-plane.
- ✓ The cell where a point is located will be increased by 1.

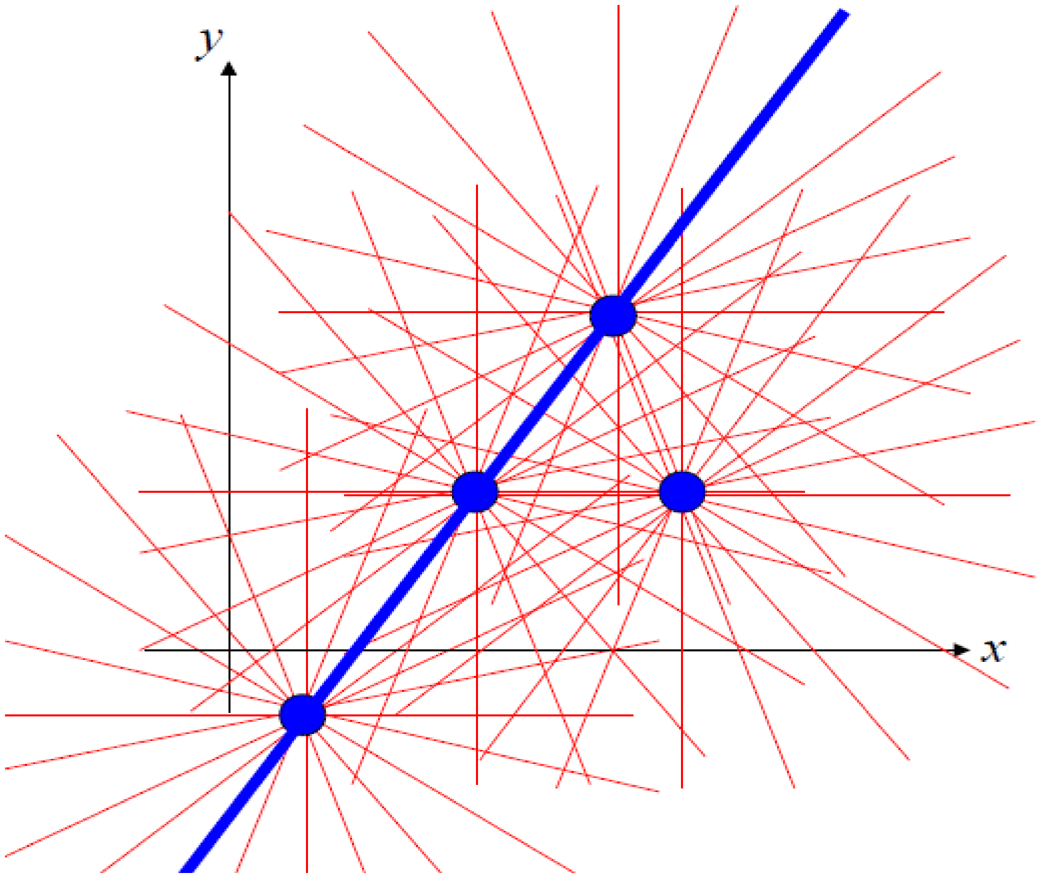


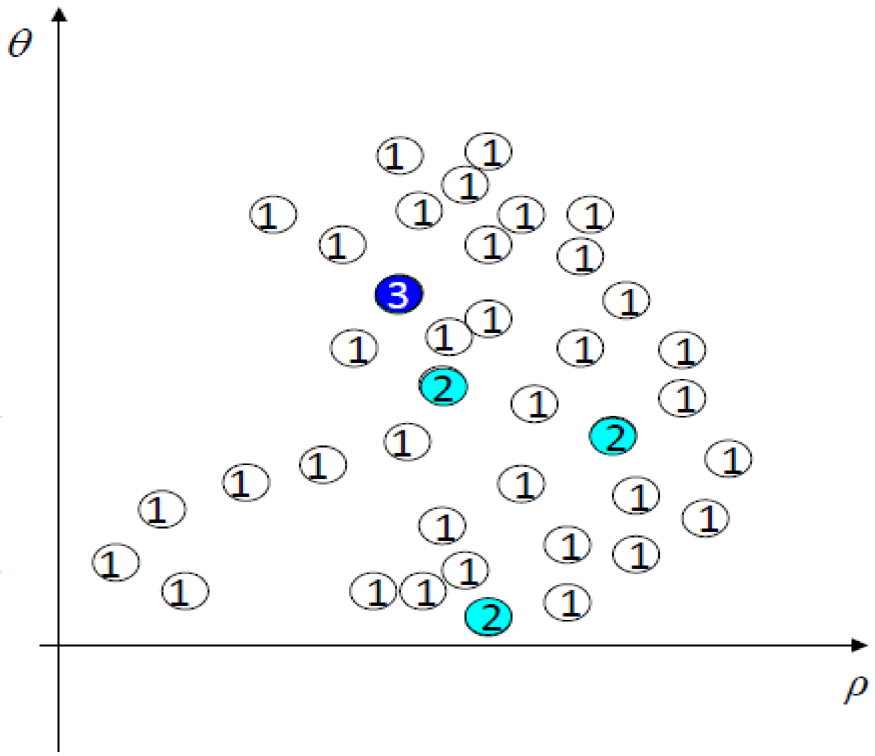
a line in (x, y) → a point in (θ, ρ)

Hough transform (4/11)

➢ **Finding lines by Hough Transform**

a line in (x, y) → a point in (θ, ρ)





Hough transform (5/11)

➢ Code - HoughLines

Syntax:

HoughLines(image, lines, rho, theta, threshold, srn, stn)

image − 8-bit, single-channel binary source image. The image may be modified by the function.
lines − Output vector of lines. Each line is represented by a two-element vector (θ,ρ)
rho − Distance resolution of the accumulator in pixels.
theta − Angle resolution of the accumulator in radians.
threshold − Accumulator threshold parameter. Only those lines are returned that get enough votes (> threshold).
srn − For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive.

stn − For the multi-scale Hough transform, it is a divisor for the distance resolution theta.

Hough transform (6/11)

➢ Code - HoughLinesP

Syntax:

➢ Finds line segments in a binary image using the **probabilistic** Hough transform.

HoughLines(image, lines, rho, theta, minLineLength, maxLineGap)

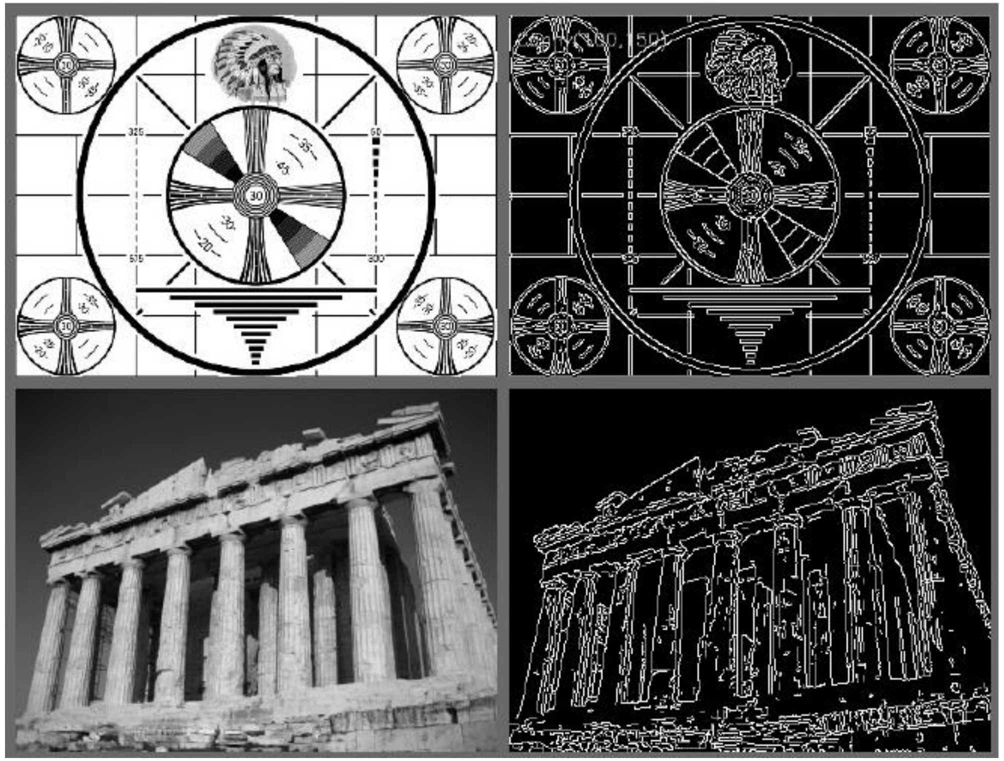
image − 8-bit, single-channel binary source image. The image may be modified by the function.
lines − Output vector of lines. Each line is represented by a 4-element vector (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) are the ending points of each detected line segment.
rho − Distance resolution of the accumulator in pixels.
theta − Angle resolution of the accumulator in radians.
threshold − Accumulator threshold parameter. Only those lines are returned that get enough votes (> threshold).

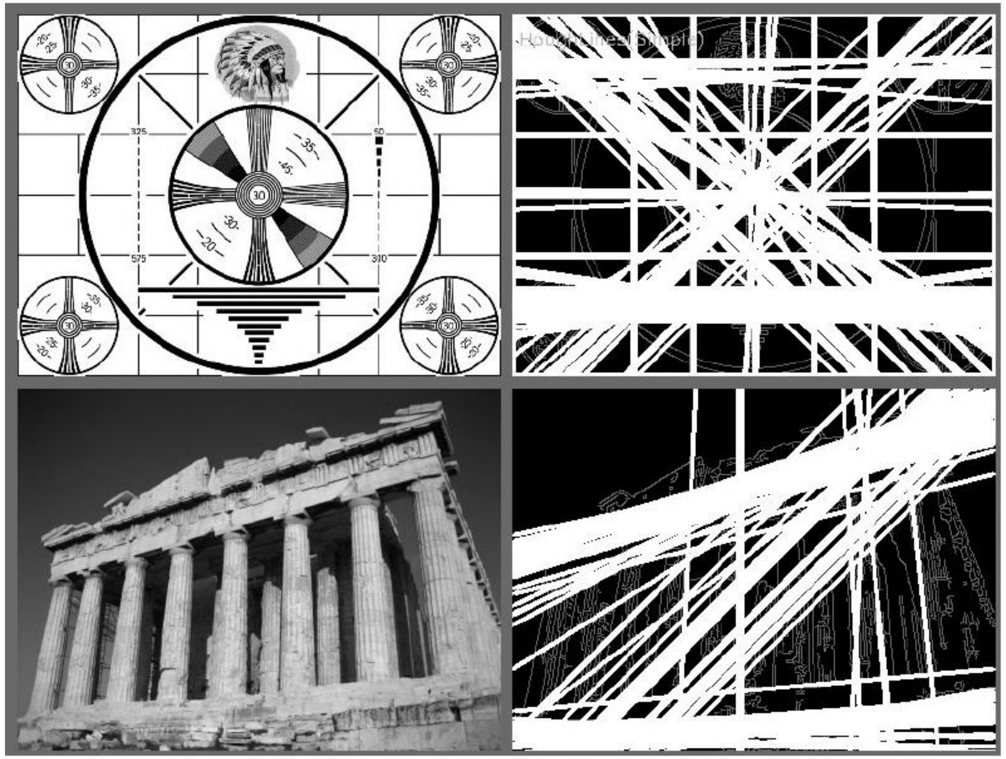
minLineLength − Minimum line length. Line segments shorter than that are rejected.

maxLineGap − Maximum allowed gap between points on the same line to link them.

Hough transform (7/11)

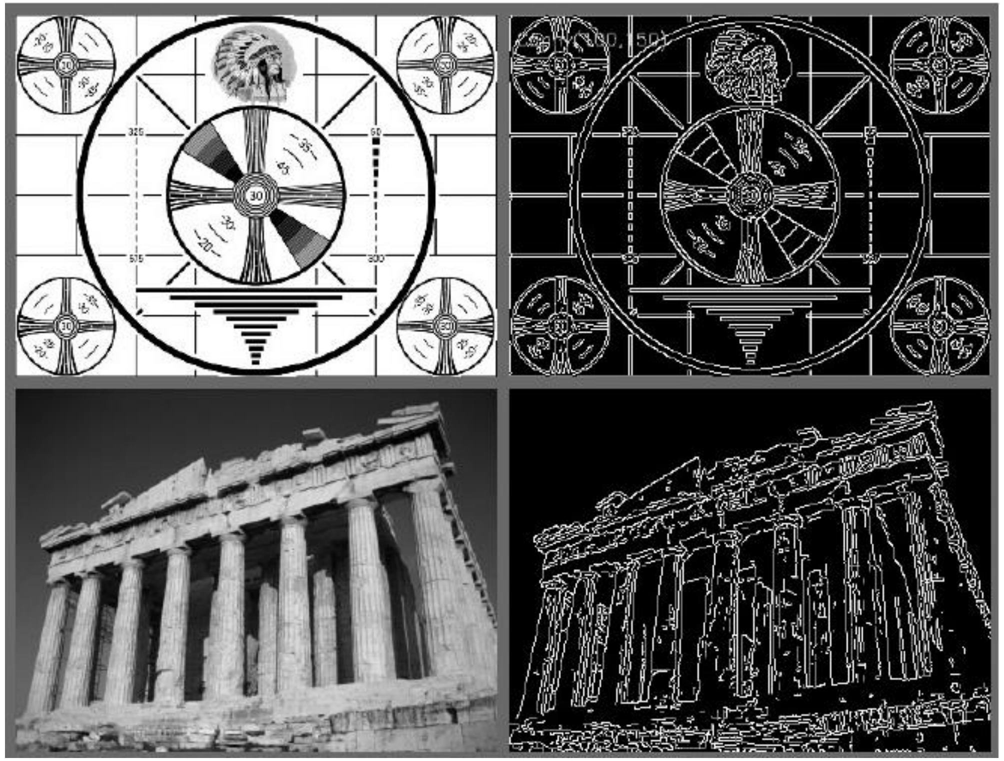
➢ Example - HoughLines

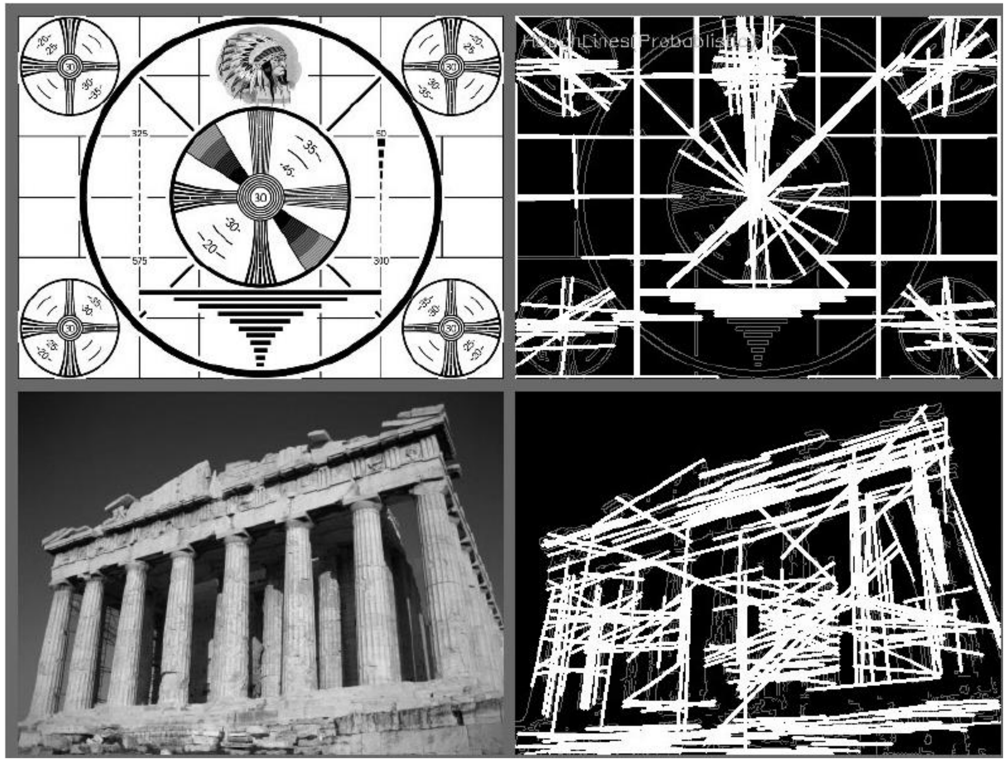




Hough transform (8/11)

➢ Example - HoughLinesP





Hough transform (9/11)

- **Example**

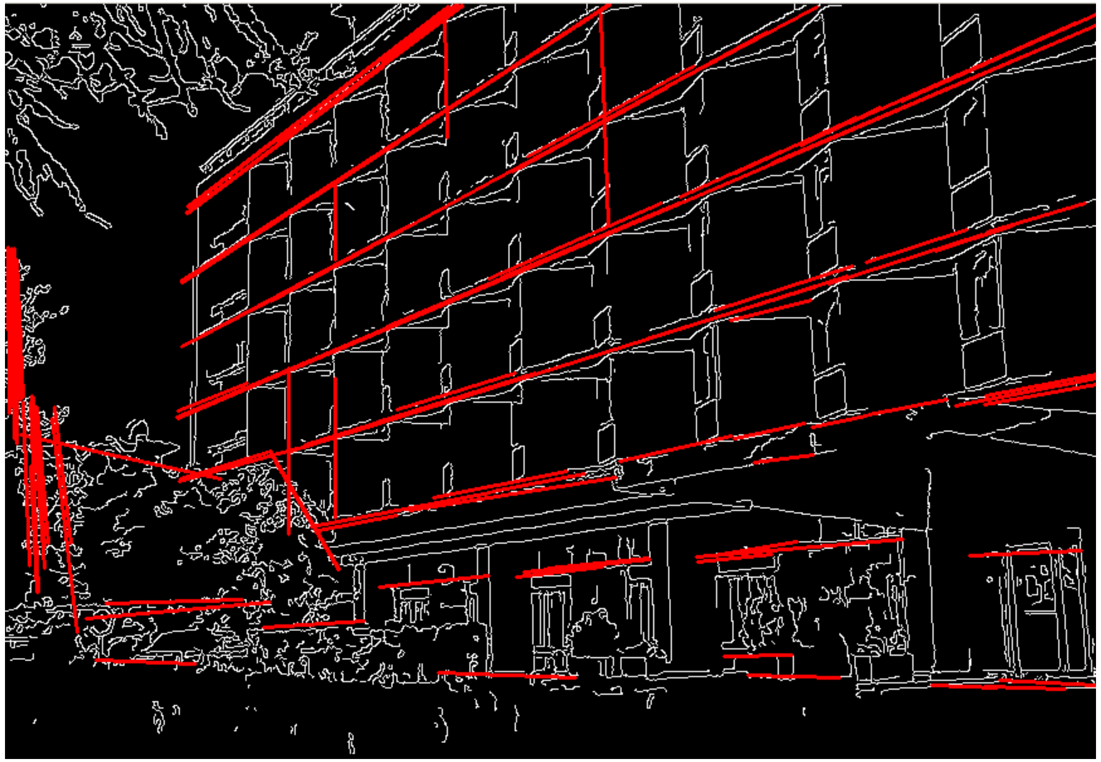
Hough transform (10/11)

➢ Demo Code – HoughLineP Demo

```
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
using namespace cv;
using namespace std;
int main(int argc, char** argv)
{
Mat src, dst, color_dst;
if( argc != 2 || !(src=imread(argv[1], 0)).data)
return -1;
Canny( src, dst, 50, 200, 3 );
cvtColor( dst, color_dst, COLOR_GRAY2BGR );
vector<Vec4i> lines;
HoughLinesP( dst, lines, 1, CV_PI/180, 80, 30, 10 );
for( size_t i = 0; i < lines.size(); i++ )
{
line( color_dst, Point(lines[i][0], lines[i][1]),
Point( lines[i][2], lines[i][3]), Scalar(0,0,255), 3, 8 );
}
namedWindow( "Source", 1 );
imshow( "Source", src );
namedWindow( "Detected Lines", 1 );
imshow( "Detected Lines", color_dst );
waitKey(0);
return 0;
}
```

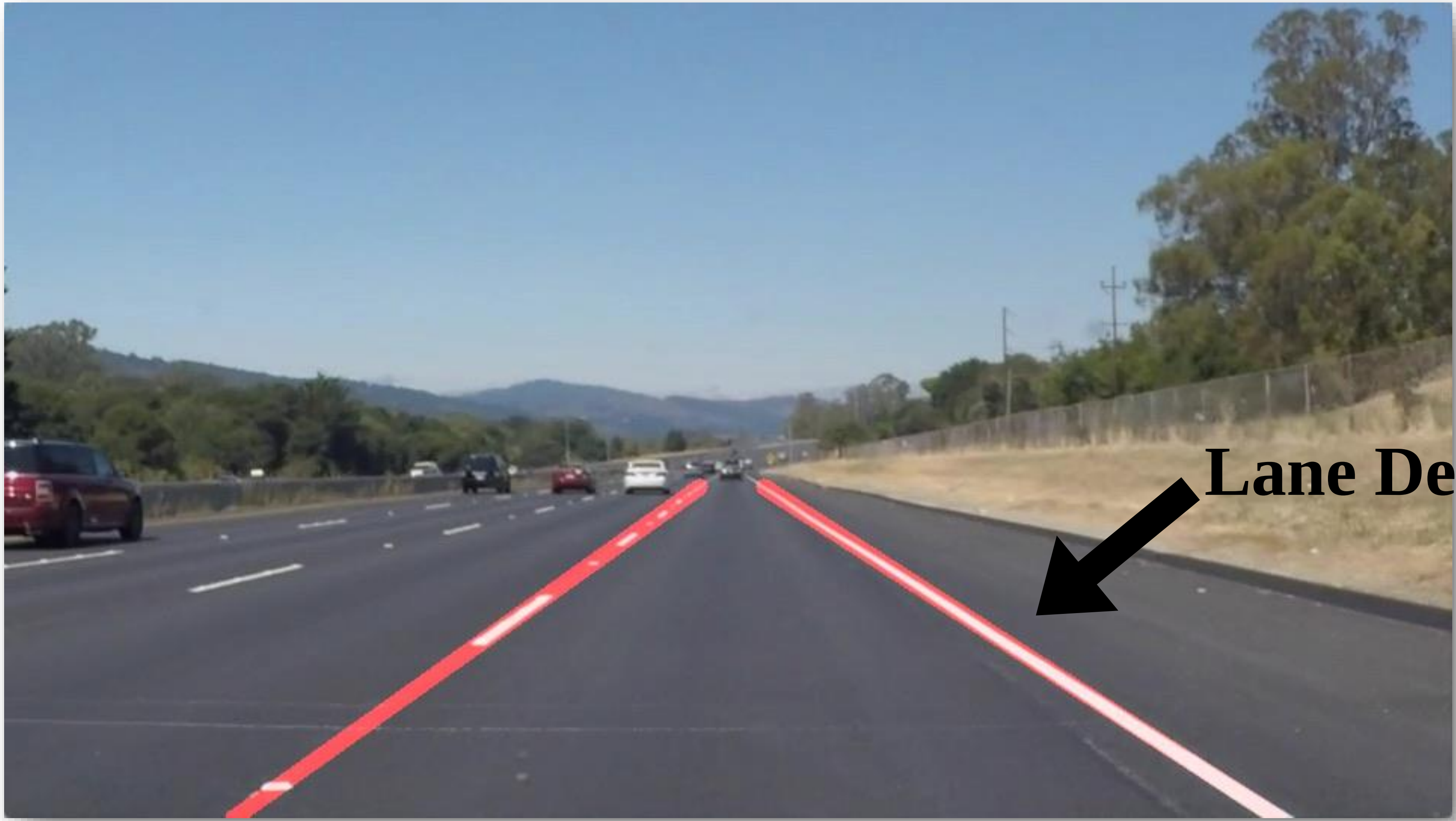
Hough transform (11/11)

- ➢ **Example - HoughLineP**



Practice

Application – Lane Line Detection (1/4)



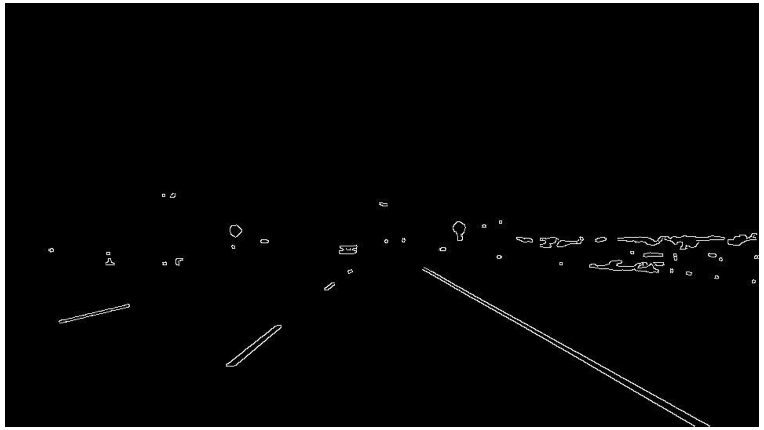
Lane Detection

Application – Lane Line Detection (2/4)

- **Preprocess**

Video of dash cam

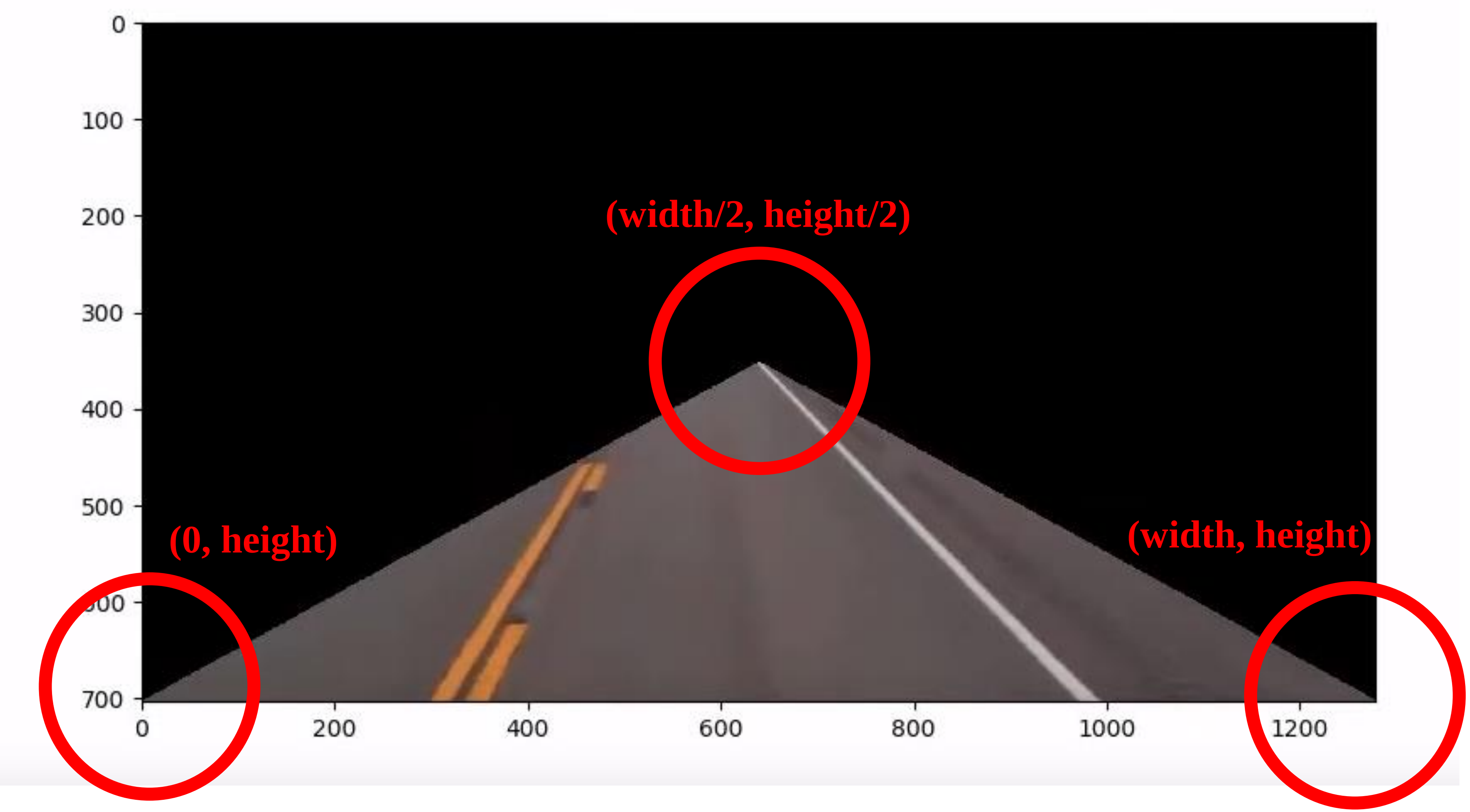
Gray



Canny

Application – Lane Line Detection (3/4)

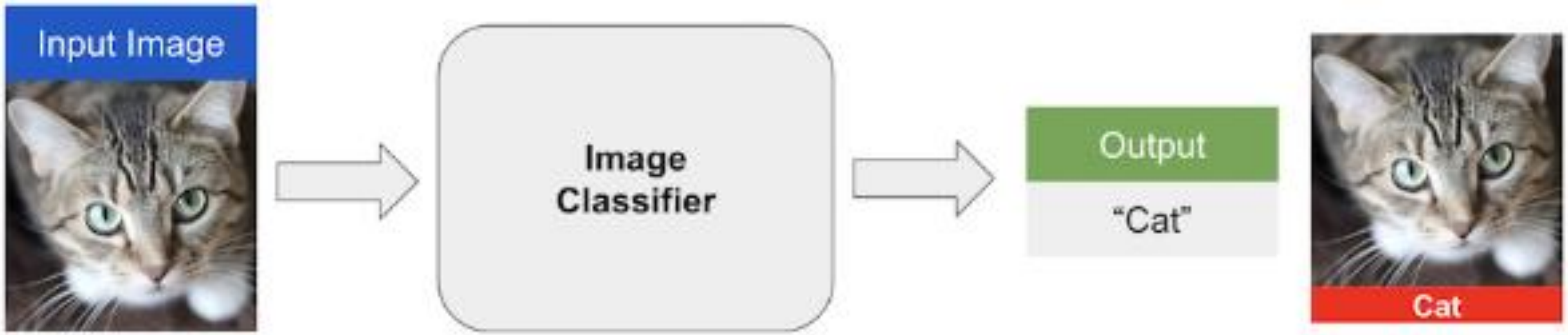
- RoI (Region of Interest) setup

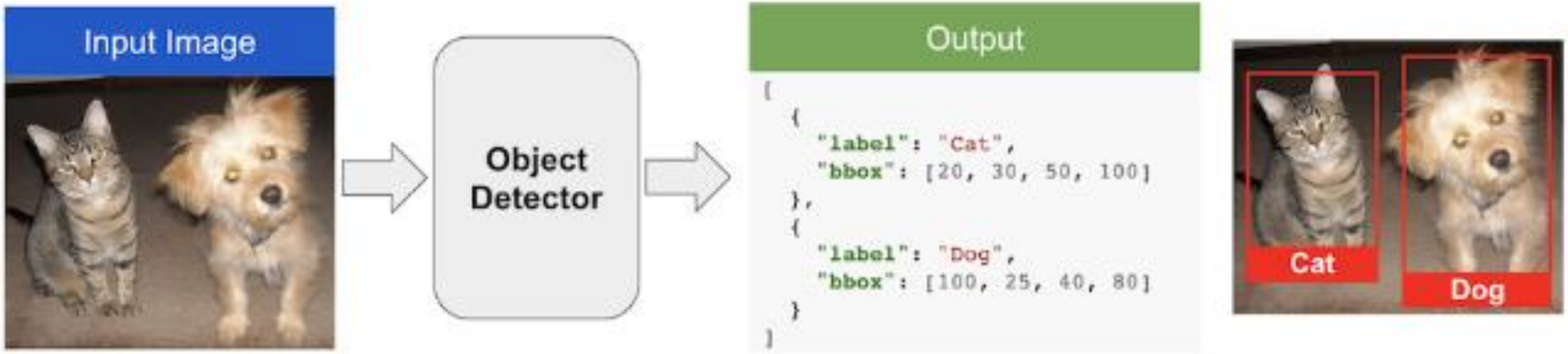


Lane line Detection (4/4)

- Hough – HoughLinesP

Object Detection (1/5)





Object Detection (2/5)

- ✓ Typically, image classification is used in applications where there is only one object in the image.

- ✓ There could be multiple classes (e.g. cats, dogs, etc.) but usually, there is only one instance of that class in the image.

1. Find bounding boxes containing objects such that each bounding box has only one object.

2. Classify the image inside each bounding box and assign it a label.

Object Detection (3/5)

➢ Sliding Window Approach

✓ Example of Human face

Original Image

Target Objects

Result

Object Detection (4/5)

➢ Sliding Window Approach

✓ Example of Human face

By sliding window to search the area of original image.

Object Detection (5/5)

➢ Code - matchTemplate

➢ Compares a template against overlapped image regions.

matchTemplate(image, templ, result, method, mask)

image – Image where the search is running.

templ – Searched template. It must be not greater than the source image and have the same data type.

result – Map of comparison results. It must be single-channel 32-bit floating-point.

method – Parameter specifying the comparison method, *see TemplateMatchModes*.

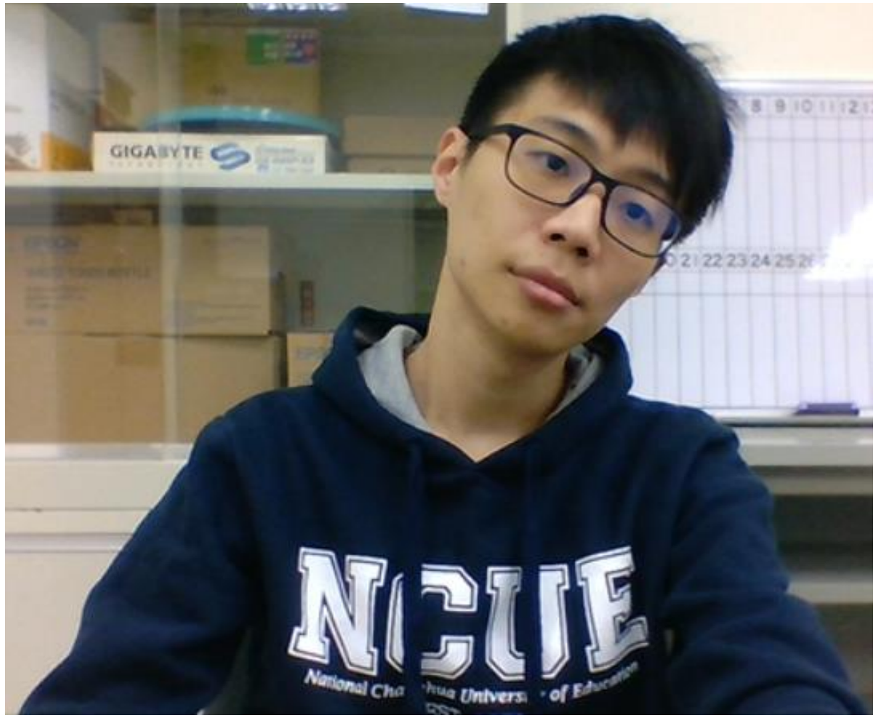
mask – Mask of searched template. It must have the same datatype and size with templ.
It is not set by default. Currently, only the TM_SQDIFF and TM_CCORR_NORMED methods are supported.

Practice

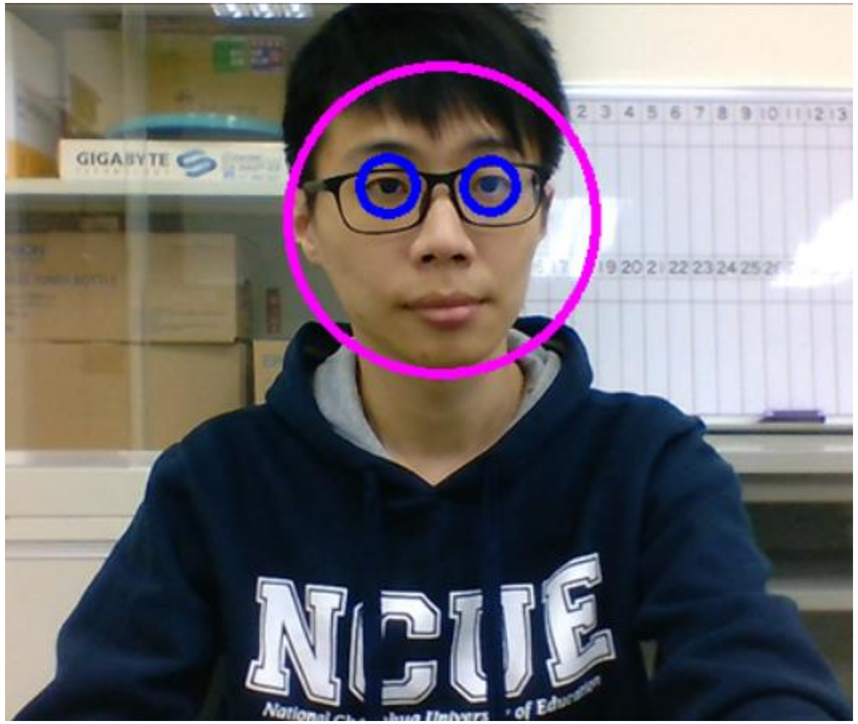
Demo code:
https://docs.opencv.org/trunk/d8/ded/samples_2cpp_2tutorial_code_2Histograms_Matching_2MatchTemplate_Demo_8cpp-example.html#a16

Cascade Classifier (1/8)

- **Haar Feature-based Cascade Classifier for Object Detection**
  - ✓ Example of Human face



Video



Tracking result

Cascade Classifier (2/8)

➢ Haar Feature-based Cascade Classifier for Object Detection

✓ CascadeClassifier

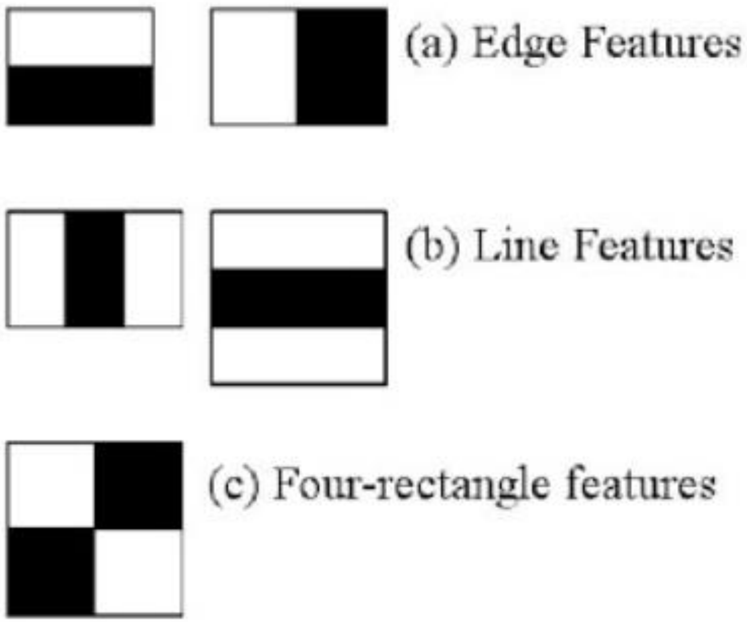
- Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

- It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Cascade Classifier (3/8)

➢ Haar Feature-based Cascade Classifier for Object Detection

- Here we will work with face detection.
- Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier.
- Then we need to extract features from it. For this, Haar features shown in the right image are used.
- They are just like our convolutional kernel.
- Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



Haar method

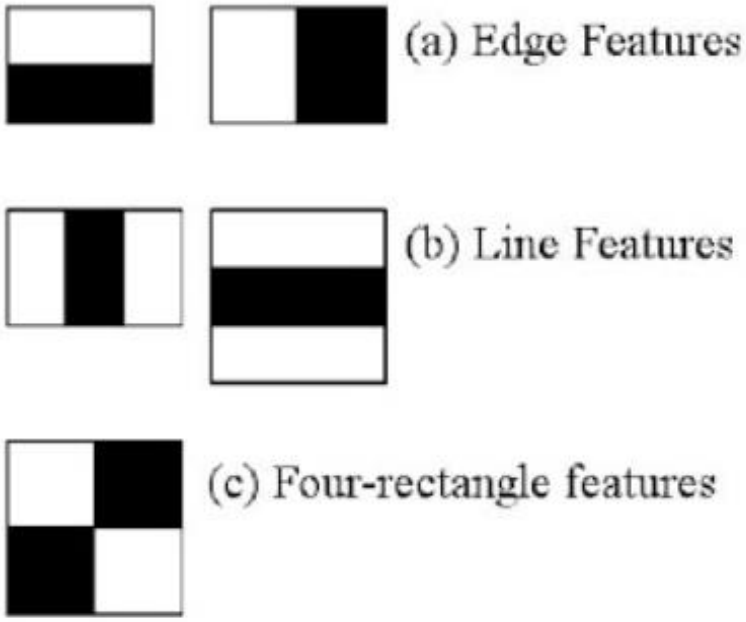
Cascade Classifier (4/8)

➢ Haar Feature-based Cascade Classifier for Object Detection

- Now, all possible sizes and locations of each kernel are used to calculate lots of features.
  (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features).

- For each feature calculation, we need to find the sum of the pixels under white and black rectangles.

- But among all these features we calculated, most of them are irrelevant.

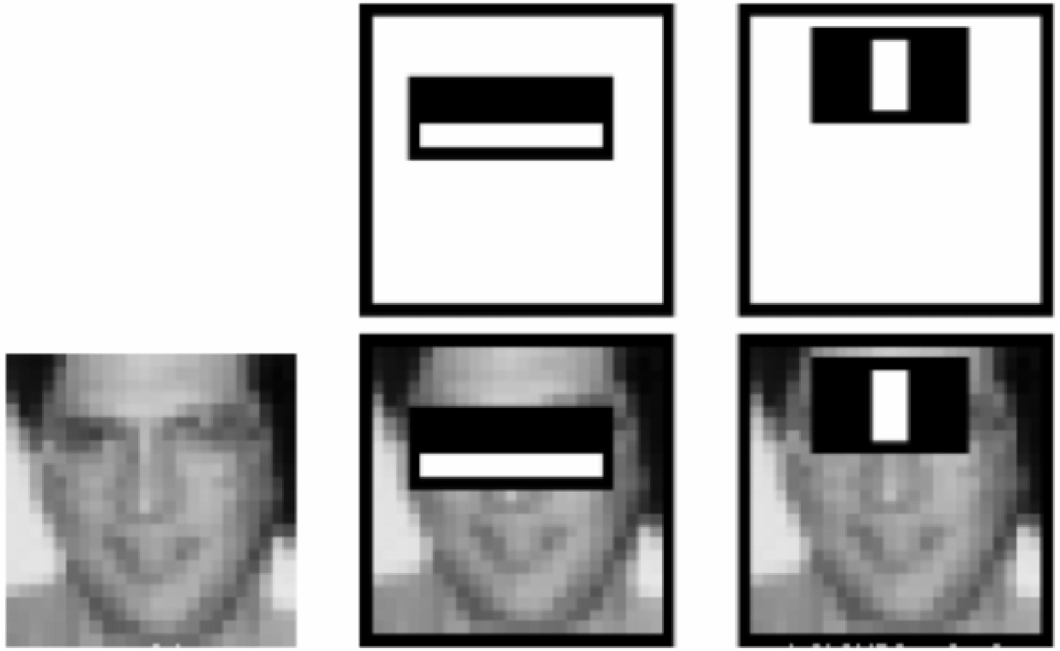


Haar method

Cascade Classifier (5/8)

➢ Haar Feature-based Cascade Classifier for Object Detection

- For example, consider the image on the right.
- The top row shows two good features.
- The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks.
- The second feature selected relies on the property that the eyes are darker than the bridge of the nose.
- But the same windows applied to cheeks or any other place is irrelevant.



Face Features

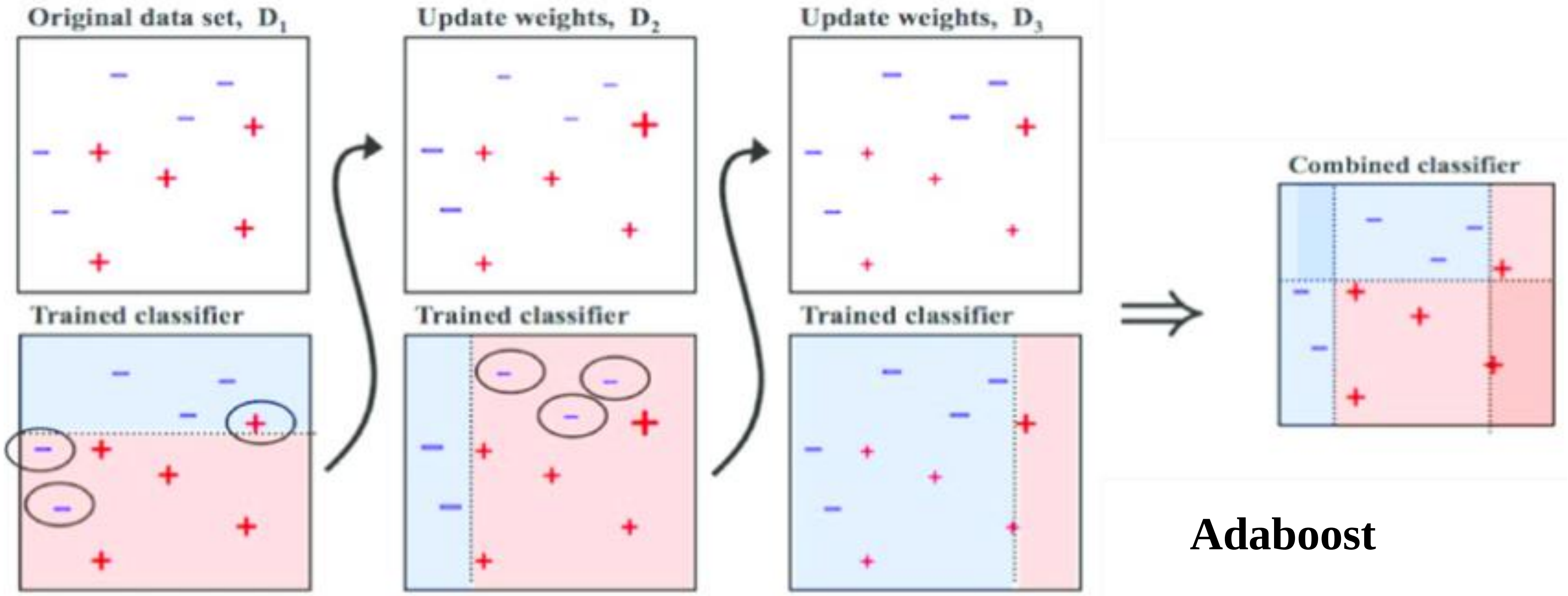
✓ So how do we select the best features out of 160,000+ features?

Cascade Classifier (6/8)

➢ AdaBoost

✓ So how do we select the best features out of 160,000+ features?

➢ It is achieved by **Adaboost (Adaptive Boosting)**.

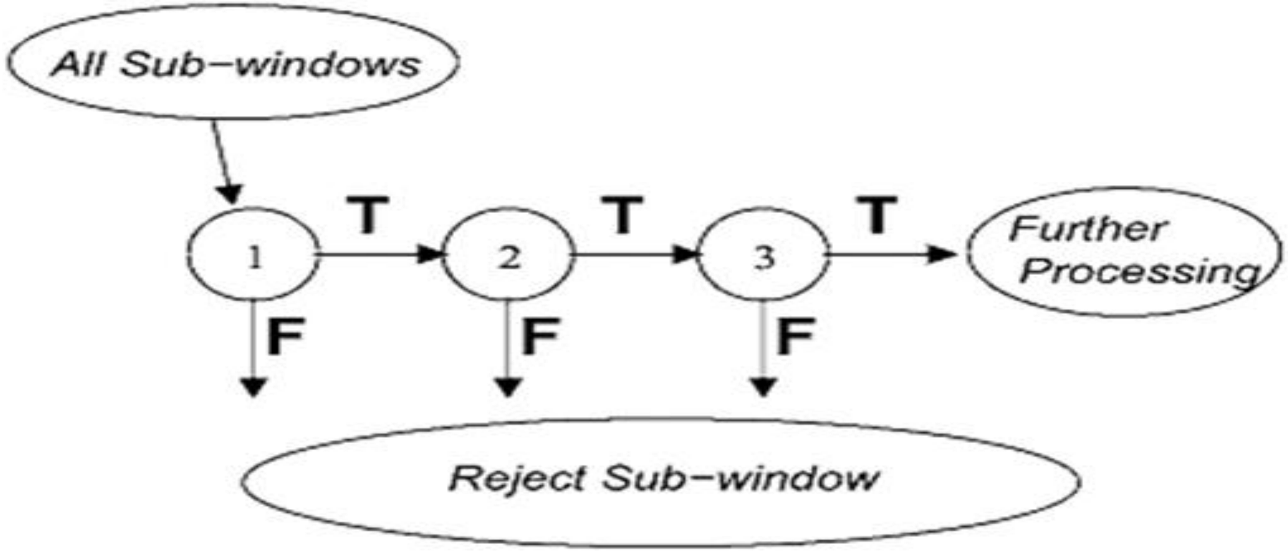


Adaboost

Cascade Classifier (7/8)

➢ Haar Feature-based Cascade Classifier for Object Detection

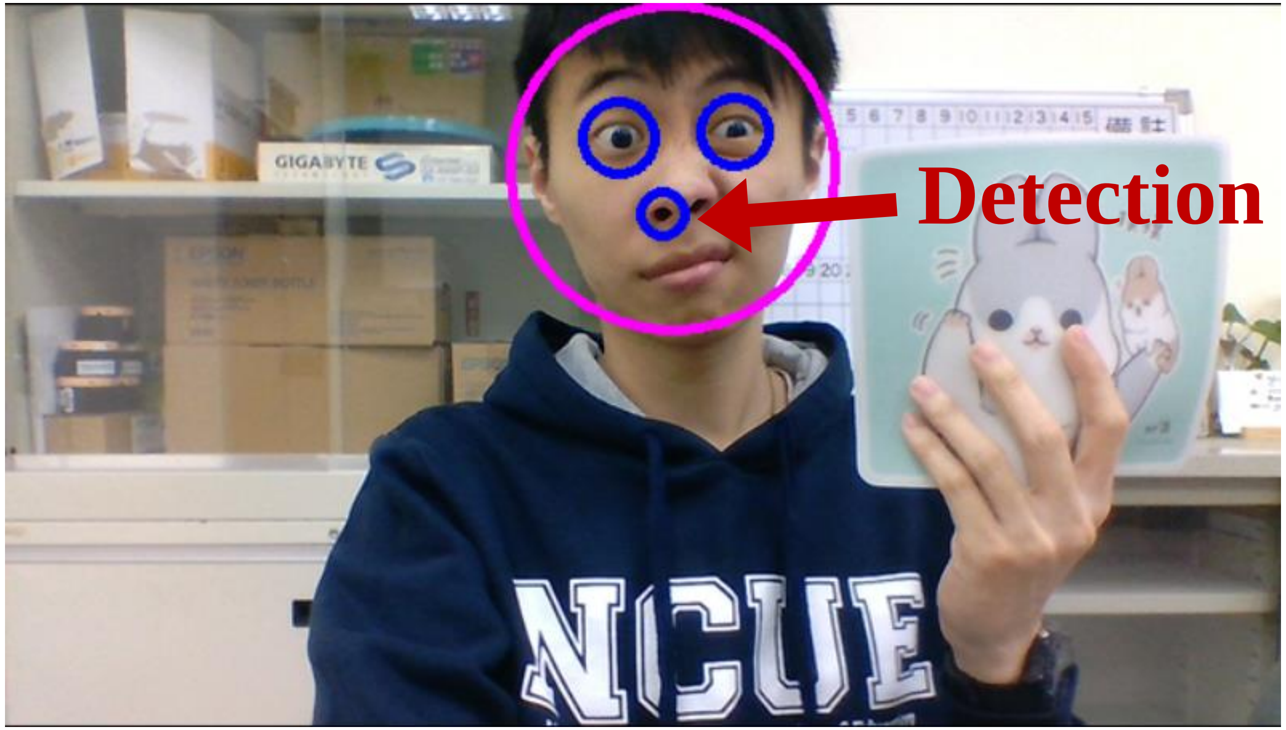
- For this, we apply each feature on all the training images.
- For each feature, it finds the best threshold which will classify the faces to positive and negative.
- Obviously, there will be errors or misclassifications.
- We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images.



Cascade Classifier

Cascade Classifier (8/8)

- **Haar Feature-based Cascade Classifier for Object Detection**
  - ✓ Problem of detection error.



Detection error.

Practice

Demo code: https://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html

Thanks!

Any questions?