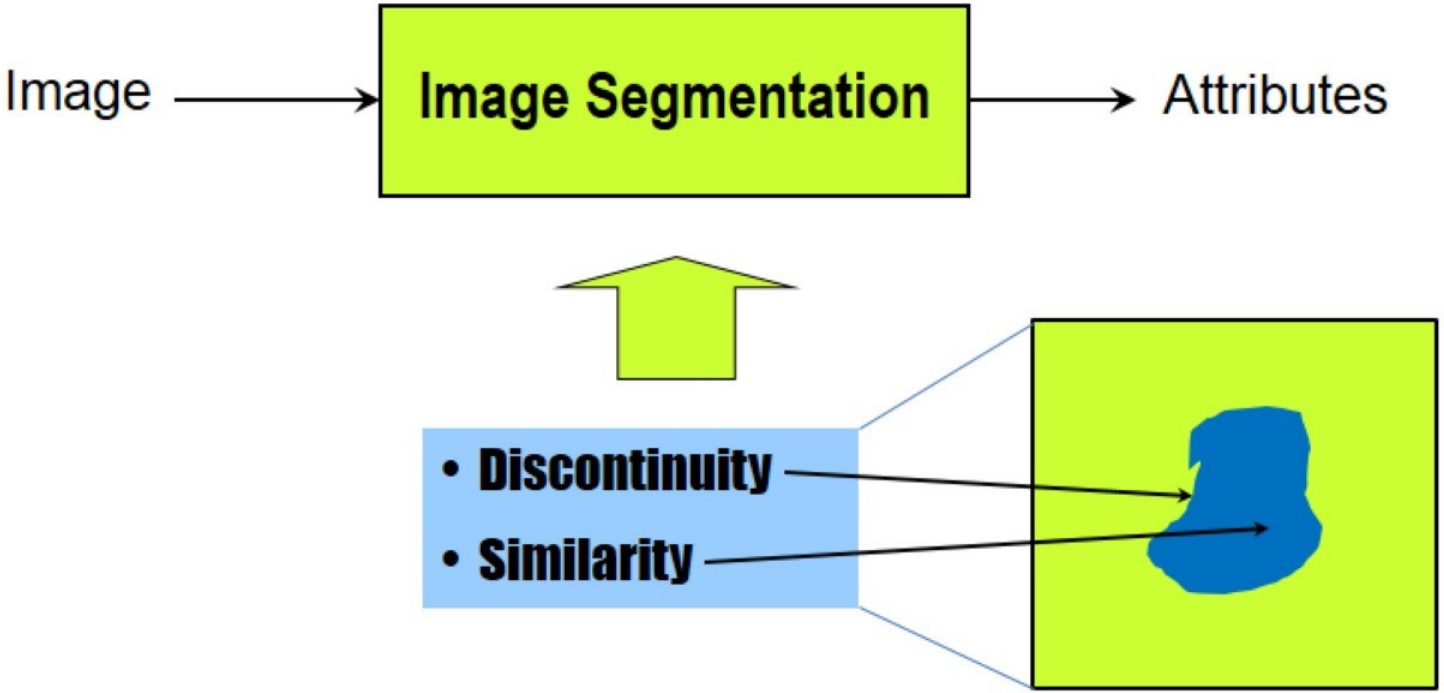
# Image Processing

## Image Segmentation (Part I)

Pattern Recognition and Image Processing Laboratory (Since 2012)

# Introduction



# Point, Line, and Edge Detection

The most common way to look for discontinuities is to run a mask through the image.

| | | |
|---|---|---|
| $z_1$ | $z_2$ | $z_3$ |
| $z_4$ | $z_5$ | $z_6$ |
| $z_7$ | $z_8$ | $z_9$ |

image

x

| | | |
|---|---|---|
| $w_1$ | $w_2$ | $w_3$ |
| $w_4$ | $w_5$ | $w_6$ |
| $w_7$ | $w_8$ | $w_9$ |

mask

$$= \quad R = \sum_{i=1}^{9} w_i z_i$$

---

The response of mask is defined with respect to its center.

# Point, Line, and Edge Detection

- **Point Detection**

  An isolated point is detected at the location on which the mask is centered if

$$|R| \geq T .$$

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

← A mask for point detection.

---

$T$ is a specified threshold.

# Point, Line, and Edge Detection

- **Point Detection: MATLAB code**

```
f = imread('test_pattern_with_single_pixel.tif');
figure(1); imshow(f);

w = [-1 -1 -1; -1  8 -1; -1 -1 -1];

g = abs(imfilter(double(f), w));
T = max(g(:));
g = g >= T;

figure(2); imshow(g);
```

# Point, Line, and Edge Detection

- **Line Detection**

  This mask responds more strongly to lines (one pixel thick) oriented horizontally.

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 2 | 2 | 2 |
| -1 | -1 | -1 |

# Point, Line, and Edge Detection

- Line Detection

| | | |
|---|---|---|
| -1 | -1 | 2 |
| -1 | 2 | -1 |
| 2 | -1 | -1 |

+45°

| | | |
|---|---|---|
| -1 | 2 | -1 |
| -1 | 2 | -1 |
| -1 | 2 | -1 |

Vertical

| | | |
|---|---|---|
| 2 | -1 | -1 |
| -1 | 2 | -1 |
| -1 | -1 | 2 |

- 45°

# Point, Line, and Edge Detection

- **Line Detection: MATLAB code**

```
>> f = imread('wirebond_mask.tif');
>> figure(1); imshow(f);
>> w = [ 2 -1 -1; -1  2 -1; -1 -1  2];
>> g = abs(imfilter(double(f), w));
>> figure(2); imshow(g, [ ]);
>> gtop = g(1:120, 1:120);
>> gtop = pixeldup(gtop, 4);
>> figure(3); imshow(gtop, [ ]);
>> gbot = g(end-119:end, end-119:end);
>> gbot = pixeldup(gbot, 4);
>> figure(4); imshow(gbot, [ ]);
>> g = abs(g);
>> figure(5); imshow(g, [ ]);
>> T = max(g(:));
>> g = (g >= T);
>> figure(6); imshow(g);
```

# Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

The edge detection is the most common approach for detecting meaningful **discontinuities** in intensity values.

Such discontinuities are detected by using **first- and second-order derivatives**.

# Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

The gradient of a 2-D function, $f(x,y)$, is defined as the vector.

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

# Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

$$\nabla f = mag(\nabla f) = \left[G_x^2 + G_y^2\right]^{1/2} = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2}$$

$$\nabla f \approx |G_x| + |G_y|$$

---

$$\alpha(x, y) = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

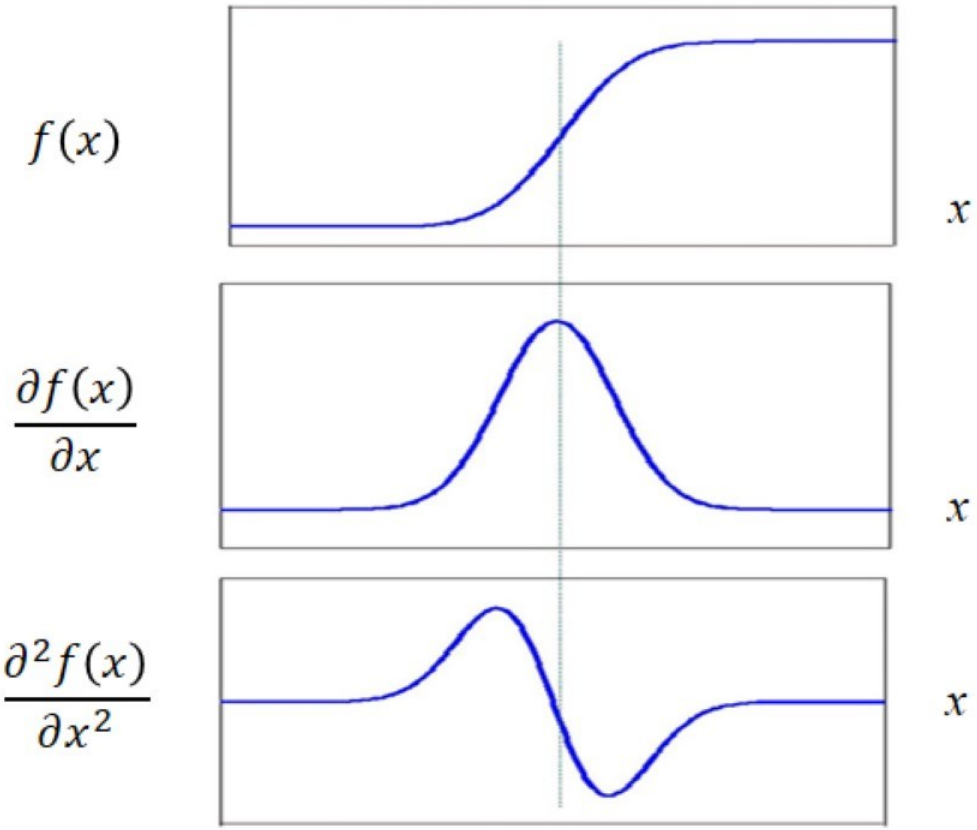
# Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

  The Laplacian of 2-D function is formed from second derivatives as follows:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

# Point, Line, and Edge Detection



# Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

The Laplacian has some drawbacks:

- It is sensitive to noise, its magnitude produces double edge.
- It is unable to detect edge direction.

However, the Laplacian can be a powerful complement when used in combination with other edge-detection techniques.

# Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

  The basic idea behind edge detection is to find places in an image where the intensity changes rapidly, using one of two general criteria.

  1. Find places where the first derivative of the intensity is greater in magnitude than a specified threshold.

  2. Find places where the second derivative of the intensity has a zero-crossing.

# Point, Line, and Edge Detection

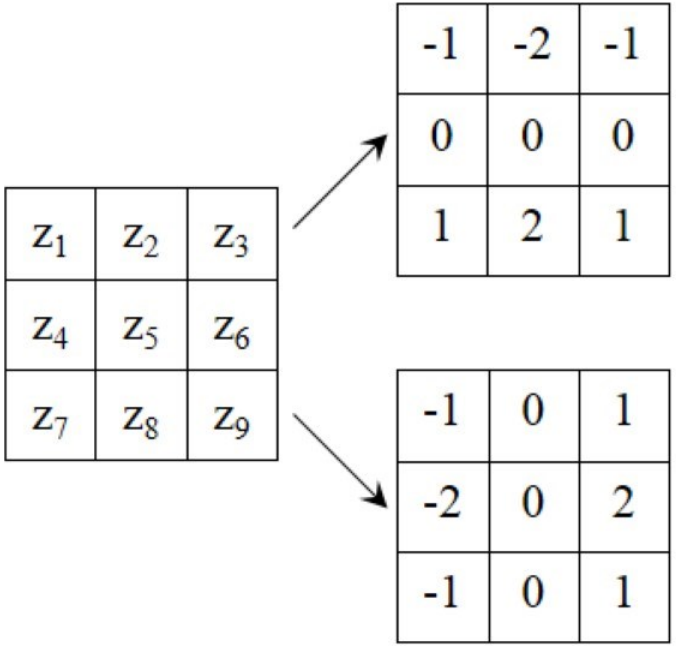
- IPT function: edge

[g, t] = edge(f, 'method', parameters)

- Sobel Edge Detector
- prewitt Edge Detector
- Roberts Edge Detector
- Laplacian of a Gaussian Detector
- Zero-crossing Detector
- Canny Edge Detector

# Point, Line, and Edge Detection

- **Sobel Edge Detector**




$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

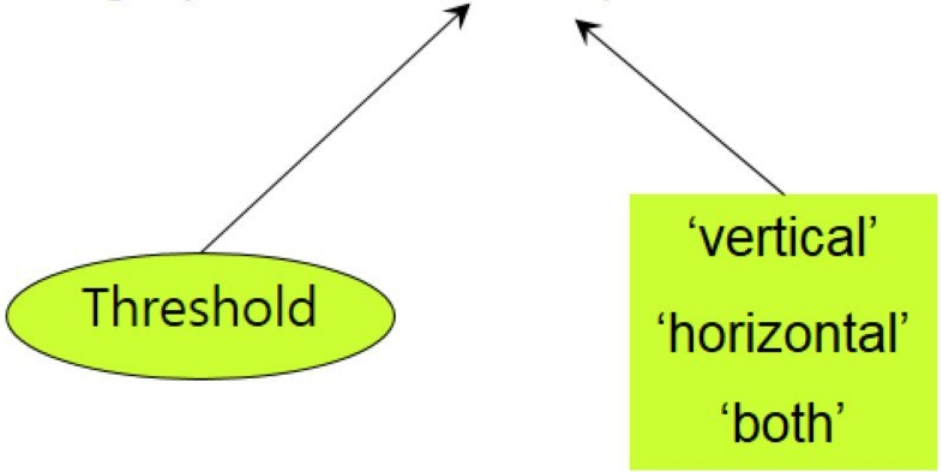
$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

$$\therefore g = \left[G_x^2 + G_y^2\right]^{1/2}$$

# Point, Line, and Edge Detection

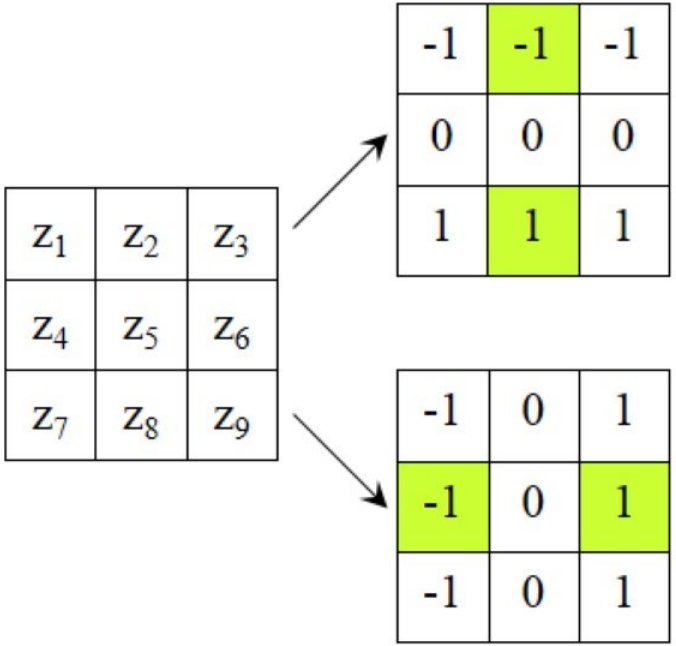
- IPT function: edge

[g, t] = edge(f, 'sobel', T, dir)



# Point, Line, and Edge Detection

- **Prewitt Edge Detector**




$$G_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

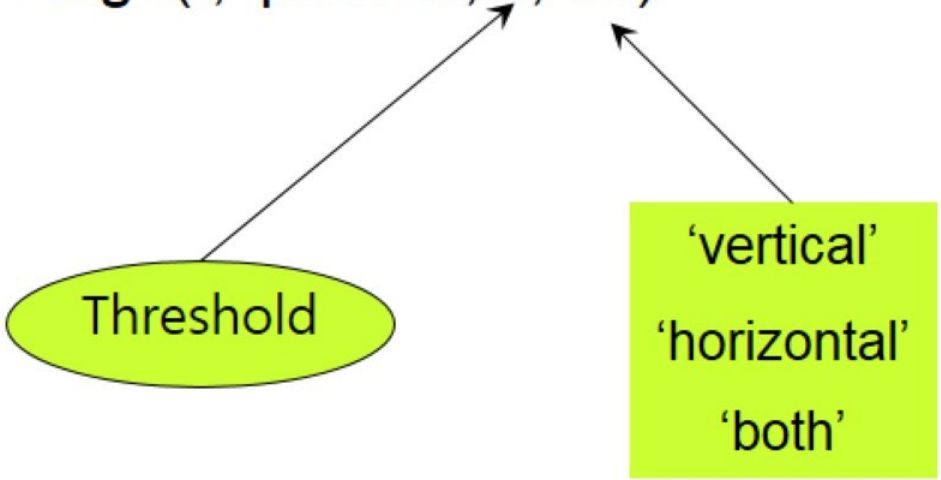
$$G_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

$$\therefore g = \left[G_x^2 + G_y^2\right]^{1/2}$$

# Point, Line, and Edge Detection

- **IPT function: edge**

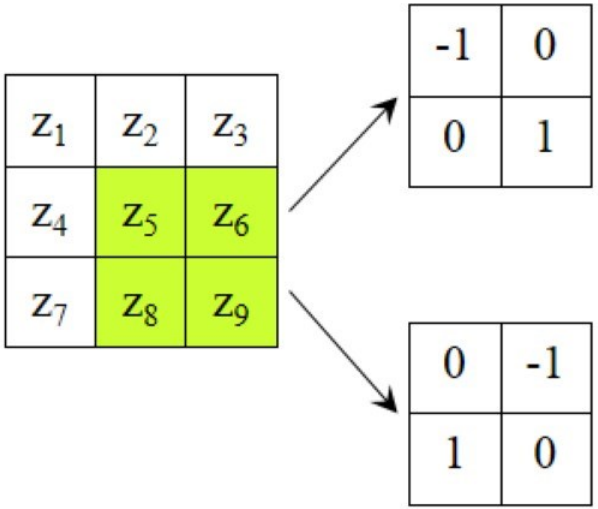
[g, t] = edge(f, ‘prewitt’, T, dir)




Prewitt detector is slightly simpler to implement computationally than the sobel detector, but it tends to produce somewhat noisier results.

# Point, Line, and Edge Detection

- Roberts Edge Detector




$$G_x = (z_9 - z_5)$$

$$G_y = (z_8 - z_6)$$

$$\therefore g = \left[G_x^2 + G_y^2\right]^{1/2}$$

# Point, Line, and Edge Detection

- **Laplacian of Gaussian (LoG) Detector/ Zero-Crossing Detector**

  The key concepts of these detectors are

  **1** Smoothing the image by using Gaussian function

$$h(r) = -e^{-\frac{r^2}{2\sigma^2}}$$

# Point, Line, and Edge Detection

- **Laplacian of Gaussian (LoG) Detector/ Zero-Crossing Detector**

The key concepts of these detectors are

**2** - Computing the Laplacian,

$$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}},$$

which yields a double-edge image.

- Finding the zero-crossing between the double edges.

# Point, Line, and Edge Detection

- **Laplacian of Gaussian (LoG) Detector/ Zero-Crossing Detector**

Zero-crossing detector is based on the same concept as the LoG method, but the convolution is carried out using a specified filter function.

```
H = fspecial('log', 40, 5);
mesh(H);
```

# Point, Line, and Edge Detection

- **Canny Edge Detection**

The method can be summarized as follows:

1. Noise reduction;
2. Gradient calculation;
3. Non-maximum suppression;
4. Double threshold;
5. Edge tracking by hysteresis.

---

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

# Point, Line, and Edge Detection

- **Canny Edge Detection: Step 1. Noise reduction**

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

# Point, Line, and Edge Detection

- **Canny Edge Detection: Step 2. Gradient calculation**

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

# Point, Line, and Edge Detection

- **Canny Edge Detection:** Step 3. Non-Maximum suppression

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

# Point, Line, and Edge Detection

- **Canny Edge Detection:** Step 4. Double threshold

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

# Point, Line, and Edge Detection

- **Canny Edge Detection:** Step 5. Edge tracking by hysteresis

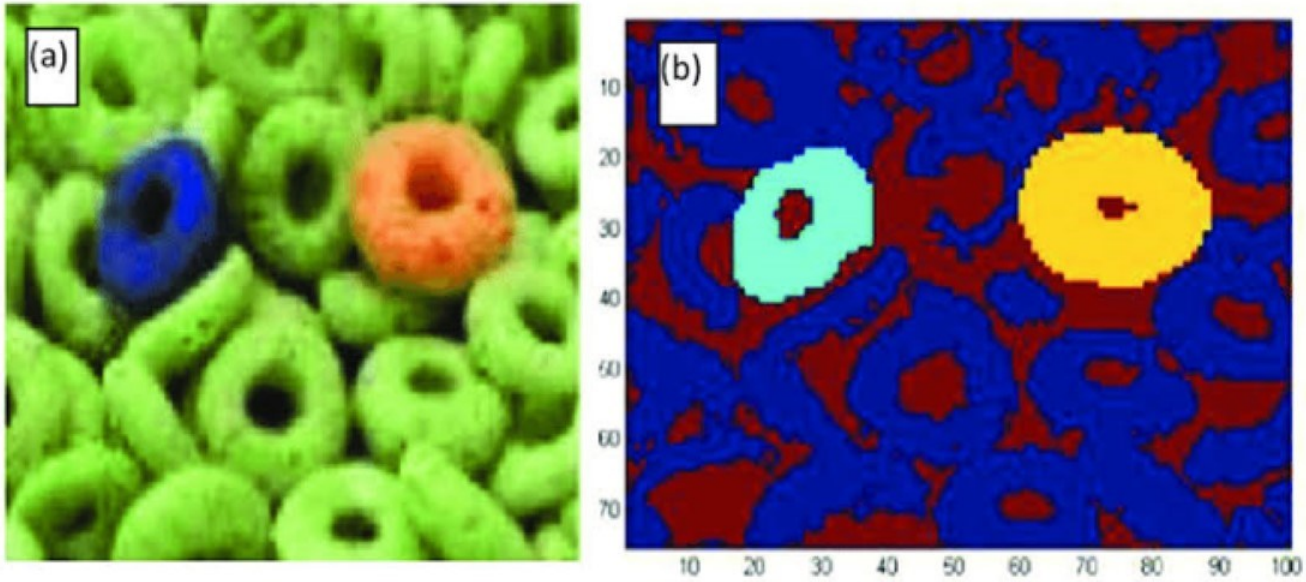
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

# Point, Line, and Edge Detection

- Comparison of the Sobel, LoG, and Canny edge detectors

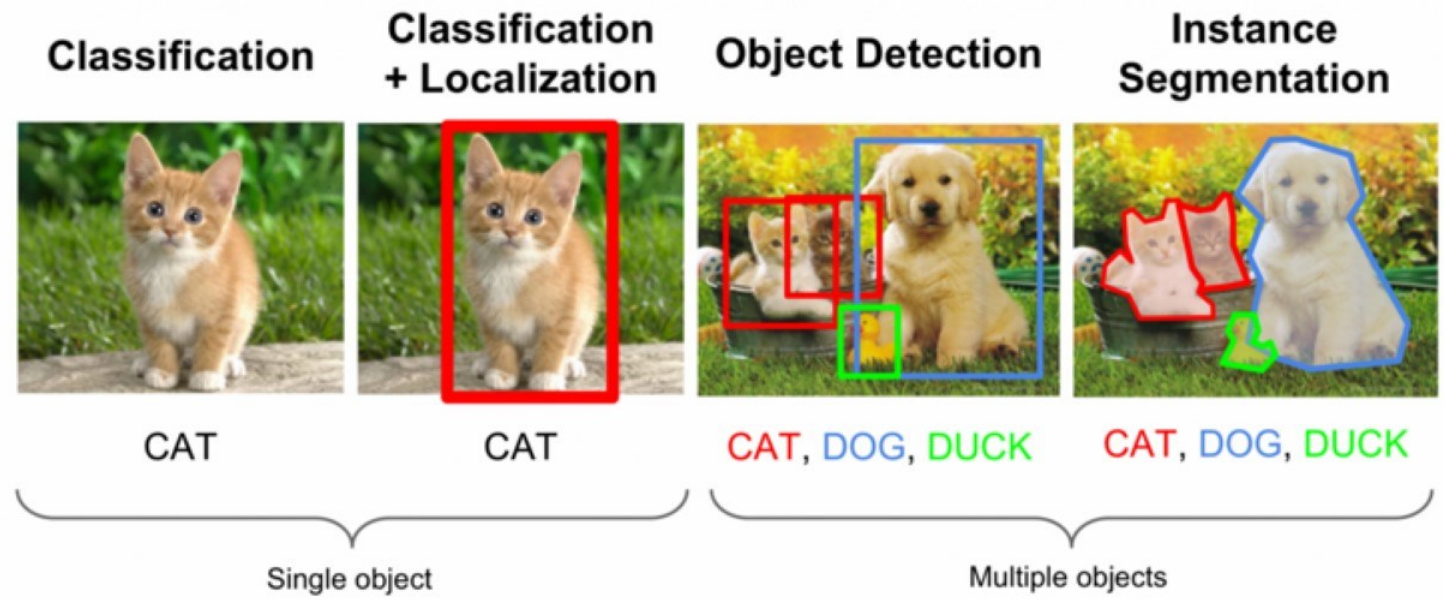
```
>> ex_edge  % See demo
```

# Applications of Image Segmentation



# Applications of Image Segmentation

# Applications of Image Segmentation



# Applications of Image Segmentation


The end of
part I

```
% ex_edge.m
% ------------------------------------------------------------

% An example of SOBEL edge detections.

% [g, t] = edge(f, 'method', parameters);

f = imread('lena.bmp');
T = []
[g1, t1] = edge(f, 'sobel', T, 'vertical');
figure(1); imshow(g1, []);

[g2, t2] = edge(f, 'prewitt', T, 'vertical');
figure(2); imshow(g2, []);

[g3, t3] = edge(f, 'roberts', T, 'vertical');
figure(3); imshow(g3, []);

sigma = 1;
[g4, t4] = edge(f, 'log', T, sigma);
figure(4); imshow(g4, []);

H = fspecial('sobel');
% [g5, t5] = edge(f, 'zerocross', T, H);
[g5, t5] = edge(f, 'zerocross');
figure(5); imshow(g5, []);
```

# Image Processing

## Workshop on Image Segmentation (Part I)

Pattern Recognition and Image Processing Laboratory (Since 2012)

# Workshop on Morphological Image Processing (Part I)

1. จาก MATLAB code: StepByStepCannyEdgeDetection ที่กำหนดให้ ให้นักศึกษาระบุส่วนของ code ที่ทำหน้าที่ประมวลผลดังต่อไปนี้

    1.1 Noise reduction % เพิ่มเติม code เพื่อแสดงผลลัพธ์ในส่วนนี้

    1.2 Gradient calculation % เพิ่มเติม code เพื่อแสดงผลลัพธ์ในส่วนนี้

    1.3 Non-maximum suppression % เพิ่มเติม code เพื่อแสดงผลลัพธ์ในส่วนนี้

    1.4 Double threshold % เพิ่มเติม code เพื่อแสดงผลลัพธ์ในส่วนนี้

    1.5 Edge tracking by hysteresis % เพิ่มเติม code เพื่อแสดงผลลัพธ์ในส่วนนี้

```
%% Canny Edge Detection




clear all;
close all;
clc; clf;
step = 0;


FILTER_SIZE = 5;
SIGMA = 14;

LOW_THRESHOLD_FACTOR = 0.1;
HIGH_THRESHOLD_FACTOR = 0.3;




im = imread('building.tif');
im = double(im);
step = step + 1;
figure(step);






gaussian_filter = fspecial('gaussian', FILTER_SIZE, SIGMA);



conv_im = conv2(im, gaussian_filter, 'same');
step = step + 1;
figure (step);






[gaussian_filter_x gaussian_filter_y] = gradient(gaussian_filter);
im_gradient_x = conv2(conv_im, gaussian_filter_x, 'same');
```

```
im_gradient_y = conv2(conv_im, gaussian_filter_y, 'same');
n_direction = atan2(im_gradient_y, im_gradient_x);
n_direction = n_direction*180/pi;

n_direction_dis = zeros(512, 512);
for i = 1   : 512
    for j = 1 : 512
        if ((n_direction(i, j) > 0 ) && (n_direction(i, j) < 22.5)
|| (n_direction(i, j) > 157.5) && (n_direction(i, j) < -157.5))
            n_direction_dis(i, j) = 0;
        end

        if ((n_direction(i, j) > 22.5) && (n_direction(i, j) < 67.5)
|| (n_direction(i, j) < -112.5) && (n_direction(i, j) > -157.5))
            n_direction_dis(i, j) = 45;
        end

        if ((n_direction(i, j) > 67.5 && n_direction(i, j) < 112.5)
|| (n_direction(i, j) < -67.5 && n_direction(i, j) > 112.5))
            n_direction_dis(i, j) = 90;
        end

        if ((n_direction(i, j) > 112.5 && n_direction(i, j) <=
157.5) || (n_direction(i, j) < -22.5 && n_direction(i, j) > -67.5))
            n_direction_dis(i, j) = 135;
        end
    end
end
step = step + 1;
figure(step);


step = step + 1;
figure(step);


step = step + 1;
figure(step);
imagesc(n_direction_dis); colorbar;




im_gradient_mag = sqrt(im_gradient_x.^2 + im_gradient_y.^2);
step = step + 1;
figure(step);
```

```
supressed_im = zeros(512, 512);
for i = 2  : 511
    for j = 2 : 511

        if (n_direction_dis(i, j) == 0)
            if (im_gradient_mag(i, j) > im_gradient_mag(i, j - 1) &&
im_gradient_mag(i, j) > im_gradient_mag(i, j + 1))
                supressed_im(i, j) = im_gradient_mag(i, j);
            else
                supressed_im(i, j) = 0;
            end
        end
        if (n_direction_dis(i, j) == 45)
            if (im_gradient_mag(i, j) > im_gradient_mag(i + 1, j -
1) && im_gradient_mag(i, j) > im_gradient_mag(i - 1, j + 1))
                supressed_im(i, j) = im_gradient_mag(i, j);
            else
                supressed_im(i, j) = 0;
            end
        end
        if (n_direction_dis(i, j) == 90)
            if (im_gradient_mag(i, j) > im_gradient_mag(i - 1, j) &&
im_gradient_mag(i, j) > im_gradient_mag(i + 1, j))
                supressed_im(i, j) = im_gradient_mag(i, j);
            else
                supressed_im(i, j) = 0;
            end
        end
        if (n_direction_dis(i, j) == 135)
            if (im_gradient_mag(i, j) > im_gradient_mag(i - 1, j -
1) && im_gradient_mag(i, j) > im_gradient_mag(i + 1, j + 1))
                supressed_im(i, j) = im_gradient_mag(i, j);
            else
                supressed_im(i, j) = 0;
            end
        end
    end
end
step = step + 1;
figure(step);
```

```
ThreshL = LOW_THRESHOLD_FACTOR * max(max(supressed_im));
ThreshH = HIGH_THRESHOLD_FACTOR * max(max(supressed_im));
thresh_im = zeros(512, 512);
for i = 1  : 512
    for j = 1 : 512
        if (supressed_im(i, j) < ThreshL)
            thresh_im(i, j) = 0;
        elseif (supressed_im(i, j) > ThreshH)
            thresh_im(i, j) = 1;
        else
            if (                                                ...
                  (supressed_im(i + 1, j) > ThreshH) ||         ...
                  (supressed_im(i - 1, j) > ThreshH) ||         ...
                  (supressed_im(i, j + 1) > ThreshH) ||         ...
                  (supressed_im(i, j - 1) > ThreshH)            ...
               )
                thresh_im(i, j) = 1;
            end
        end
    end
end
step = step + 1;
figure(step);
```