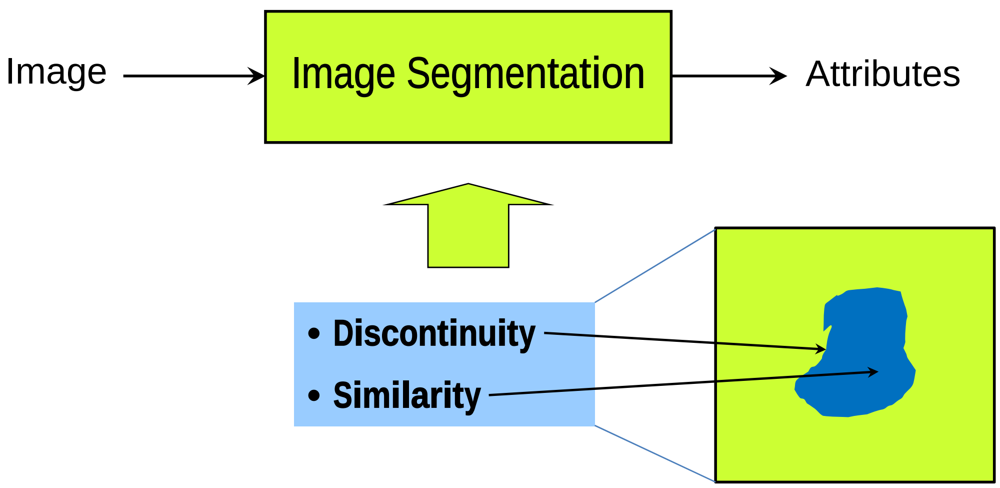
Image Processing

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.

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

Point, Line, and Edge Detection

- **Line Detection: Using Function edge**

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

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

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

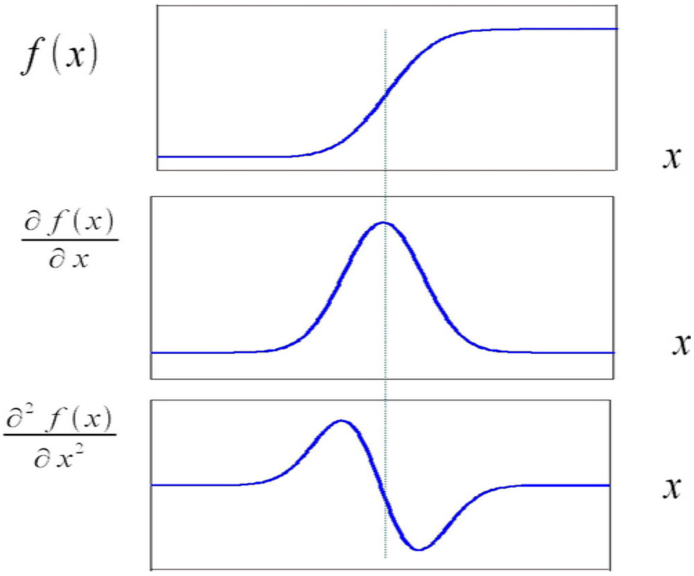
Point, Line, and Edge Detection

- **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**

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

-1	-2	-1
0	0	0
1	2	1

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

-1	0	1
-2	0	2
-1	0	1

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

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

Point, Line, and Edge Detection

- **IPT function: edge**

[g, t] = edge(f, ‘sobel’, T, dir)

Threshold

‘vertical’
‘horizontal’
‘both’

Point, Line, and Edge Detection

- **Prewitt Edge Detector**

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

-1	-1	-1
0	0	0
1	1	1

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

-1	0	1
-1	0	1
-1	0	1

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

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

Point, Line, and Edge Detection

- **IPT function: edge**

[g, t] = edge(f, ‘prewitt’, T, dir)

Threshold

‘vertical’

‘horizontal’

‘both’

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**

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

-1	0
0	1

$$G_x = (z_9 - z_5)$$

0	-1
1	0

$$G_y = (z_8 - z_6)$$

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

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

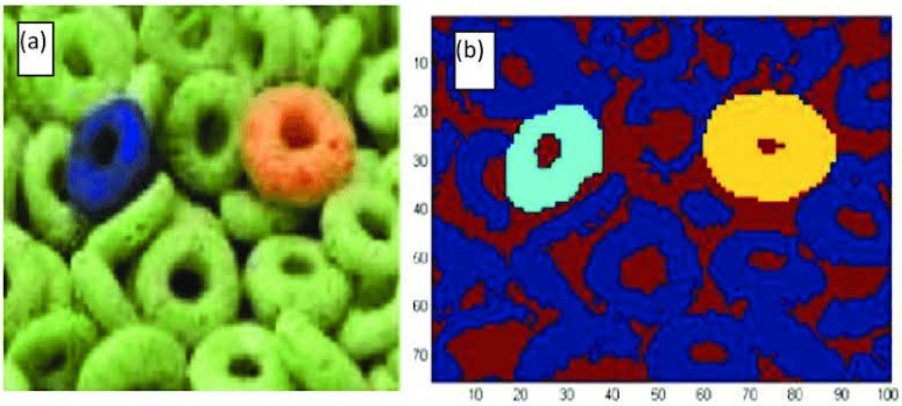
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

Point, Line, and Edge Detection

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

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

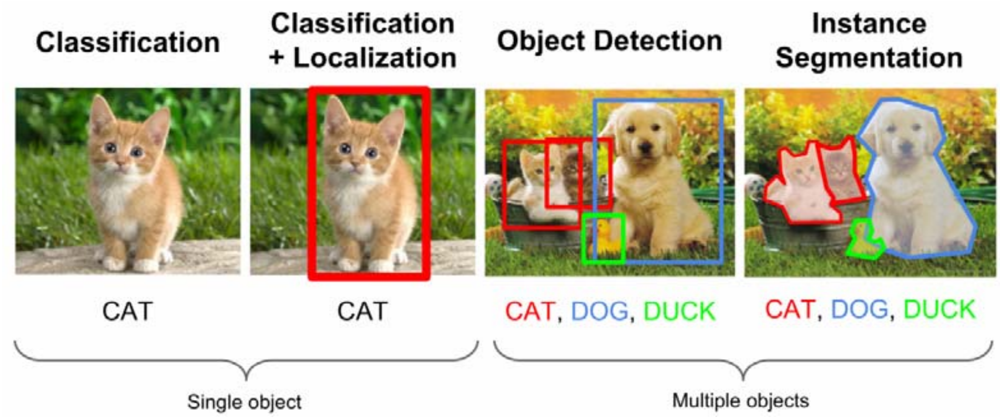
Applications of Image Segmentation



Applications of Image Segmentation

(A) Classification (B) Detection (C) Segmention

Applications of Image Segmentation



Applications of Image Segmentation