PRAKTIKUM 11

Segmentasi Citra
(*Image Segmentation*)

Materi:

- Thresholding dan deteksi tepi (*edge detection*)
- Contoh implementasi segmentasi citra pada Python

Tujuan Praktikum:

- Mahasiswa dapat mengimplementasikan teknik thresholding dan edge detection menggunakan Python

A. PENYAJIAN

Segmentasi Citra

Segmentasi citra merupakan proses mempartisi citra menjadi beberapa daerah atau objek. Umumnya segmentasi bertujuan untuk memisahkan suatu objek citra dengan objek lainnyadan atau memisahkan objek citra dengan latar belajang citra. Segmentasi citra didasarkan pada dua hal yaitu diskontinuitas dan kemiripan intensitas piksel. Pendekatan diskontinuitas disebut juga pendekatan berbasis edge (*edge-based*), sementara pendekatan kemiripan intensitas, seperti thresholding, clustering (contoh: k-means), klasifikasi, pendekatan teori graf (contoh: GrabCut), region growing, dan region splitting & merging.

Thresholding

Thresholding merupakan teknik yang paling dasar untuk melakukan segmentasi citra. Teknik thresholding dapat digunakan untuk melakukan segmentasi citra berdasaran warna tertentu atau berdasarkan suatu nilai piksel tertentu.

Deteksi Tepi (Edge Detection)

Deteksi tepi adalah operasi yang dijalankan untuk mendeteksi garis tepi yang membatasi dua wilayah citra homogen yang memiliki tingkat kecerahan yang berbeda. Edge adalah bagian dari citra dimana intensitas kecerahan berubah secara drastis. Tujuan dari deteksi tepi pada umumnya yaitu untuk mengurangi jumlah data pada citra pada langkah image processing berikutnya. Metode edge detection dibagi menjadi dua, yaitu *first order edge detection* dan *second order edge detection*.

First Order Derivative Edge Detection

Metode deteksi tepi yang termasuk kedalam first order derivative adalah Roberts operators, Prewitt operators, Sobel operators, dan First-order of gaussian (FDOG). Pada modul ini akan dicoba untuk operator Sobel, Prewitt, dan Roberts. Setiap operator memiliki row gradient dan column gradient.

Roberts Operator

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

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

$G_x = (z_9 - z_5)$ $G_y = (z_8 - z_6)$

Prewitt Operator

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

$G_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$ $G_y = (z_1 + z_6 + z_9) - (z_1 + z_4 + z_7)$

Sobel Operator

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

$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$ $G_y = (z_1 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$

Second Order Derivative Edge Detection

Canny detection

Materi lebih lanjut tentang Canny dapat dilihat pada file Canny Operator di LMS. Untuk melakukan canny, dilakukan 5 langkah berikut:

1. Smoothing, melakukan blurring untuk meghilangkan noise.
2. Menemukan gradien

   Algortime Canny pada dasarnya menemukan tepi dengan mencari area yang memiliki perubahan intensitas grayscale yang paling signifikan. Area ini ditemukan dengan mencari gradien. Jenis operator Sobel dapat digunakan untuk mendapatkan gradien.
3. Non-maximum suppresion

   Tujuan dari step ini adalah untuk meng-*convert* edge ‘blur’ dari step sebelumnya menjadi edge yang ‘sharp’. Algoritme ini dilakukan untuk setiap pixel pada citra gradient. Hanya local maxima yang ditandai sebagai edge.
4. Double thresholding

   Hasil dari non-maximum suppresion memang sudah mencirikan edge asli, tetapi bisa saja edge tersebut dihasilkan oleh noise. Maka dari itu thresholding dilakukan sehingga edge yang digunakan hanya edge yang paling kuat. Canny menggunakan dua threshold. Edge yang lebih kuat dari threshold tertinggi ditandai sebagai ‘strong’, edge yang lebih lemah dari threshold terendah ditandai sebagai ‘weak’.
5. Edge tracking menggunakan hysteresis

   Edge tracking dilakukan untuk melihat edge mana saja yang termasuk ke true edge. Tidak semua edge strong adalah true edge, begitupun tidak semua edge weak adalah noise. Edge tracking dapat diimplementasikan menggunakan analisis BLOB (Binary Large Object). Pixel edge dibagi kedalam BLOB connected neighborhood.

B. LATIHAN

Latihan Robert dan Sobel image Edge Detection

```
import numpy as np

import matplotlib.pyplot as plt


from skimage.data import camera

from skimage.filters import roberts, sobel, sobel_h, sobel_v, scharr, \
```

```
    scharr_h, scharr_v, prewitt, prewitt_v, prewitt_h, farid_v, farid_h


image = camera()
edge_roberts = roberts(image)
edge_sobel = sobel(image)


fig, ax = plt.subplots(ncols=2, sharex=True, sharey=True,
                       figsize=(8, 4))


ax[0].imshow(edge_roberts, cmap=plt.cm.gray)
ax[0].set_title('Roberts Edge Detection')


ax[1].imshow(edge_sobel, cmap=plt.cm.gray)
ax[1].set_title('Sobel Edge Detection')


for a in ax:
    a.axis('off')


plt.tight_layout()
plt.show()
```

Latihan Canny Edge Detector

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi


from skimage import feature



# Generate noisy image of a square
im = np.zeros((128, 128))
```

```
im[32:-32, 32:-32] = 1

im = ndi.rotate(im, 15, mode='constant')
im = ndi.gaussian_filter(im, 4)
im += 0.2 * np.random.random(im.shape)

# Compute the Canny filter for two values of sigma
edges1 = feature.canny(im)
edges2 = feature.canny(im, sigma=3)

# display results
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(8, 3),
                                    sharex=True, sharey=True)

ax1.imshow(im, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('noisy image', fontsize=20)

ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title(r'Canny filter, $\sigma=1$', fontsize=20)

ax3.imshow(edges2, cmap=plt.cm.gray)
ax3.axis('off')
ax3.set_title(r'Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()

plt.show()
```

TUGAS

Pelajari dan jalankan code Python mengenai image segmentation di bawah ini.
Apa keluaran dari program tersebut? Jelaskan cara kerja image segmentation pada code tersebut.

```
"""
=================================================
Comparing edge-based and region-based segmentation
=================================================

In this example, we will see how to segment objects from a background. We use
the ``coins`` image from ``skimage.data``, which shows several coins outlined
against a darker background.
"""

import numpy as np
import matplotlib.pyplot as plt

from skimage import data
from skimage.exposure import histogram

coins = data.coins()
hist, hist_centers = histogram(coins)

fig, axes = plt.subplots(1, 2, figsize=(8, 3))
axes[0].imshow(coins, cmap=plt.cm.gray)
axes[0].axis('off')
axes[1].plot(hist_centers, hist, lw=2)
axes[1].set_title('histogram of gray values')

######################################################################
#
# Thresholding
# ============
#
# A simple way to segment the coins is to choose a threshold based on the
# histogram of gray values. Unfortunately, thresholding this image gives a
# binary image that either misses significant parts of the coins or merges
# parts of the background with the coins:

fig, axes = plt.subplots(1, 2, figsize=(8, 3), sharey=True)

axes[0].imshow(coins > 100, cmap=plt.cm.gray)
axes[0].set_title('coins > 100')

axes[1].imshow(coins > 150, cmap=plt.cm.gray)
axes[1].set_title('coins > 150')

for a in axes:
    a.axis('off')

plt.tight_layout()

######################################################################
# Edge-based segmentation
# =======================
#
# Next, we try to delineate the contours of the coins using edge-based
# segmentation. To do this, we first get the edges of features using the
# Canny edge-detector.

from skimage.feature import canny
```

```
edges = canny(coins)

fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(edges, cmap=plt.cm.gray)
ax.set_title('Canny detector')
ax.axis('off')

######################################################################
# These contours are then filled using mathematical morphology.

from scipy import ndimage as ndi

fill_coins = ndi.binary_fill_holes(edges)

fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(fill_coins, cmap=plt.cm.gray)
ax.set_title('filling the holes')
ax.axis('off')


######################################################################
# Small spurious objects are easily removed by setting a minimum size for
# valid objects.

from skimage import morphology

coins_cleaned = morphology.remove_small_objects(fill_coins, 21)

fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(coins_cleaned, cmap=plt.cm.gray)
ax.set_title('removing small objects')
ax.axis('off')

######################################################################
# However, this method is not very robust, since contours that are not
# perfectly closed are not filled correctly, as is the case for one unfilled
# coin above.
#
# Region-based segmentation
# ========================
#
# We therefore try a region-based method using the watershed transform.
# First, we find an elevation map using the Sobel gradient of the image.

from skimage.filters import sobel

elevation_map = sobel(coins)

fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(elevation_map, cmap=plt.cm.gray)
ax.set_title('elevation map')
ax.axis('off')

######################################################################
# Next we find markers of the background and the coins based on the extreme
# parts of the histogram of gray values.

markers = np.zeros_like(coins)
markers[coins < 30] = 1
markers[coins > 150] = 2

fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(markers, cmap=plt.cm.nipy_spectral)
ax.set_title('markers')
ax.axis('off')
```

```
# ---------------------------------------------------------------------
# Finally, we use the watershed transform to fill regions of the elevation
# map starting from the markers determined above:

segmentation = morphology.watershed(elevation_map, markers)

fig, ax = plt.subplots(figsize=(4, 3))
ax.imshow(segmentation, cmap=plt.cm.gray)
ax.set_title('segmentation')
ax.axis('off')

# ---------------------------------------------------------------------
# This last method works even better, and the coins can be segmented and
# labeled individually.

from skimage.color import label2rgb

segmentation = ndi.binary_fill_holes(segmentation - 1)
labeled_coins, _ = ndi.label(segmentation)
image_label_overlay = label2rgb(labeled_coins, image=coins)

fig, axes = plt.subplots(1, 2, figsize=(8, 3), sharey=True)
axes[0].imshow(coins, cmap=plt.cm.gray)
axes[0].contour(segmentation, [0.5], linewidths=1.2, colors='y')
axes[1].imshow(image_label_overlay)

for a in axes:
    a.axis('off')

plt.tight_layout()

plt.show()
```

Referensi

https://scikit-image.org/docs/dev/auto_examples/applications/plot_coins_segmentation.html?highlight=image%20segmentation

https://scikit-image.org/docs/dev/auto_examples/edges/plot_edge_filter.html?highlight=edge

https://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html?highlight=canny