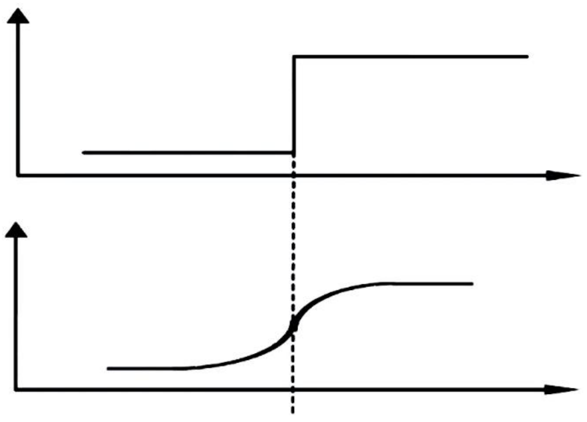
Lesson 10 Image Processing---Edge Detection

1.Edge Detection Introduction

Edge detection is fundamental technique in image processing and computer vision, which aims at identifying edges in a digital image at which the image brightness changes sharply. Sharp changes in image usually reflect important events and changes in properties. The edge is as the picture shown.



Edge detection greatly reduces the amount of data, removes irrelevant information, and preserves the important structural properties of the image. Edge detection is divided into two types.

1） Based on search: The boundary is detected by finding the maximum and minimum values in the first derivative of the image, and it usually locates in the direction with the largest gradient. The representative algorithms are the Sobel operator and the Scharr operator.

2） Based on zero-crossing: the boundary is found by searching the second-order derivative zero-crossing of the image, which is usually the Laplacian zero-crossing point or the zero-crossing point represented by the nonlinear difference, and the representative algorithm is the Laplacian operator.

2.Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986 and considered as the best algorithm of edge detection. Canny edge detection will go through 4 steps, including Noise Reduction, Finding Gradient Magnitude and Direction of the Image, Non-maximum Suppression and Hysteresis Thresholding.

◆ **Noise Reduction**

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image. For detailed operation, please refer to the file in "**4. OpenCV Computer Vision Lesson->Lesson 3 Image Processing---Smoothing**".

◆ **Finding Gradient Magnitude and Direction of the Image**

In math, gradient is a vector, indicating that the directional derivative of the function at a certain point reaches the maximum along this direction, that is, the function changes the fastest along the direction at this point, and the rate of change is the greatest.

In the image, gradient represents the degree and direction of gray value change, and the edge refers to the position where the gray intensity changes the most. Gradient direction is always perpendicular to edges.

Sobel filter is used to calculate the magnitude and direction of the gradient. The Sobel operator G_X is the first derivative in the horizontal direction, which is used to detect the edge in the Y-axis direction, while G_Y is the the first derivative in the vertical direction, which is used to detect the edge in the X-axis direction.

The formula for calculating the gradient size is as follows

$$G = \sqrt{(G_X^2 + G_Y^2)}$$

The formula for calculating the gradient direction is as follows

$$\theta = arctan\frac{G_Y}{G_X}$$

◆ **Non-Maximum Suppression**

Non-Maximum Suppression (NMS) is that reserve local maximum and suppress all the values apart from local maximum. In simple terms, all the pixels of the image will be detected. If the gradient intensity of a point is greater than the pixels in the positive and negative directions of its gradient direction, the point is retained; otherwise, the point is suppressed.

Canny edge detection algorithm perform non-maximum suppression along the gradient direction not the edge direction.

◆ Hysteresis Thresholding

This stage decides which are really edges. For this, we need two threshold values, “**minVal**” and “**maxVal**”. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

3.Operation Steps

This routine will perform the edge detection.

Before operation, please copy the routine “**edge_detection.py**”and sample picture “**luna.jpg**” in “4.OpenCV->Lesson 8 Image Processing --- Edge

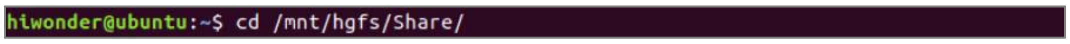
Detection->Routine Code" to the shared folder.

For how to configure the shared folder, please refer to the file in "**2. Linux Basic Lesson->Lesson 3 Linux Installation and Source Replacemen**t".

Note: the input command should be case sensitive and the keywords can be complemented by "Tab" key.

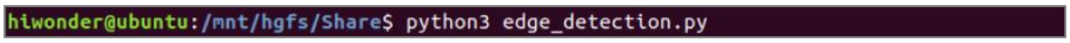
1） Open virtual machine and start the system. Click "", and then "" or press "**Ctrl+Alt+T**" to open command line terminal.

2） Input command "**cd /mnt/hgfs/Share/**" and press Enter to enter the shared folder.



```
hiwonder@ubuntu:~$ cd /mnt/hgfs/Share/
```

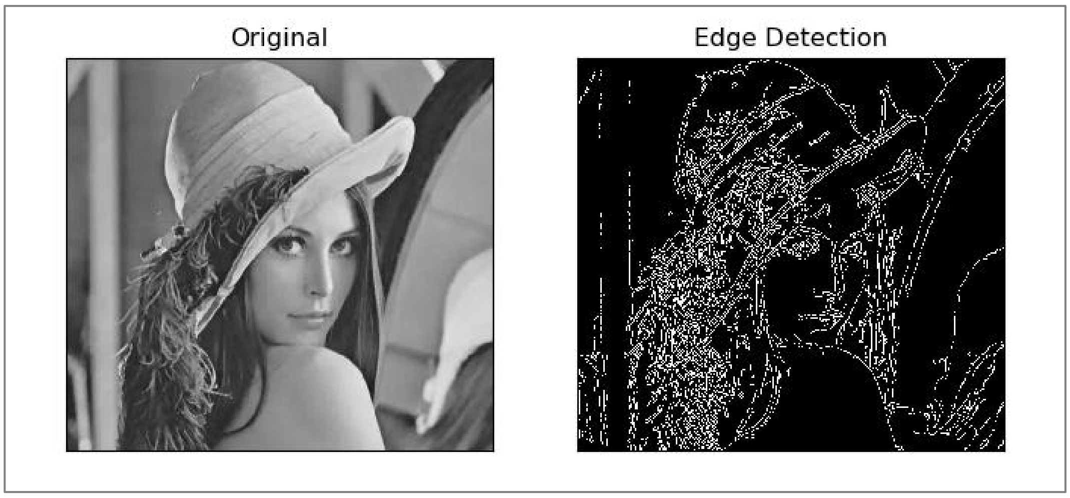
3） Input command "**python3 edge_detection.py**" and press Enter to run the routine.



```
hiwonder@ubuntu:/mnt/hgfs/Share$ python3 edge_detection.py
```

4.Program Outcome

The final output image is as follow.



5.Program Analysis

The routine “**edge_detection.py**” can be found in “**4. OpenCV Computer Vision Lesson->Lesson 8 Image Processing---Edge Detection->Routine Code**”.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# read the image
img = cv2.imread('luna.jpg')

# Canny edge detection
lowThreshold = 1
max_lowThreshold = 80
canny = cv2.Canny(img, lowThreshold, max_lowThreshold)

# image display
plt.figure(figsize=(8, 5), dpi=100)
plt.rcParams['axes.unicode_minus'] = False
plt.subplot(121), plt.imshow(img, cmap=plt.cm.gray), plt.title("Original")
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(canny, cmap=plt.cm.gray), plt.title("Edge Detection")
plt.xticks([]), plt.yticks([])
plt.show()

```

5.1 Image Processing

◆ **Import Module**

Firstly, import the required module through import statement.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

◆ **Read Image**

Call **imread()** function in cv2 module to read the image

```python
img = cv2.imread('luna.jpg')
```

The parameter in the bracket is the name of the image.

◆ **Set Threshold**

Set two thresholds to determine the final edge.

```
lowThreshold = 1
max_lowThreshold = 80
```

◆ **Edge Detection**

Call Canny() function in cv2 module to perform edge detection on the specific image.

```
canny = cv2.Canny(img, lowThreshold, max_lowThreshold)
```

The format of Canny() function is as follow.

```
Canny( image, threshold1, threshold2)
```

The first parameter "**image**" is the input image.

The second parameter "**threshold1**" is the low threshold "**minVal**"

The third parameter "**threshold2**" is high threshold "**maxVal**".

5.2 Image Display

◆ **Create Custom Figure**

Call figure() function in matplotlib.pyplot module to create a custom figure for displaying the final output image.

```
plt.figure(figsize=(8, 5), dpi=100)
```

The format of the figure() function is as follow.

```
figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs)
```

The first parameter "**num**" is the only identifier of the image i.e. the serial number of the picture (number) or the name (string).

The second parameter "**figsize**" is the width and height of the image in inch.

The third parameter "**dpi**" is the resolution of the image i.e. the number of pixels by inch

The fourth parameter "**facecolor**" is the background color.

The fifth parameter "**edgecolor**" is the frame color

The sixth parameter "**frameon**" determines whether to draw the picture, and it is "**True**" by default.

The seventh parameter "**FigureClass**" is used to select the custom figure when generating the image

The eighth parameter "**clear**" determines whether to clear all the original images.

The ninth parameter "****kwargs**" represents other properties of the image.

◆ **Modify matplotlib Configuration**

matplotlib is plotting library of Python. User can access and modify matplotlib configuration options through parameter dictionary "**rcParams**".

```
plt.rcParams['axes.unicode_minus'] = False
```

The codes above are used to manipulate the display of the normal characters.

◆ **Set the Parameter of Image Display**

Call subplot(), imshow() and title() functions in matplotlib.pyplot modules to designate the position, color and headline of the subplot in the Figure.

```
plt.subplot(121), plt.imshow(img, cmap=plt.cm.gray), plt.title("Original")
```

1） subplot() function is used to set the position of the subplot, and the function format is as follow.

```
subplot(nrows, ncols, index, **kwargs)
```

The first parameter “**nrows**” and the second parameter “**ncols**” respectively are the number of row and column of subplot.

The third parameter “**index**” is the index position. Index starts at 1 in the upper left corner and increases to the right.

When both the row and column are less than “**10**”, these two values can be abbreviated to an integer. For example, the meaning of “subplot(1, 2, 1)” and “subplot(121)” are the same, both representing the image is divided into one row and 2 columns, and the subplot is in the first place i.e. 1st row, 1st column.

2） imshow() function is used to set the color of subplot, and its format is as follow.

```
imshow(X, cmap=None)
```

The first parameter “**X**” is the image data.

The second parameter “**cmap**” is the colormap, RGB(A) color space by default.

3） title() function is used to set the title of the subplot. The parameter in the bracket is the name of the subplot and the function format is as follow.

```
title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)
```

The first parameter “label” is the title composed of string.

The second parameter “fontdict” is the property of the font, and the current parameter refers to dictionary.

The third parameter “loc” is the position of the title. It can be “left”, “center” or “right”, and “center” by default.

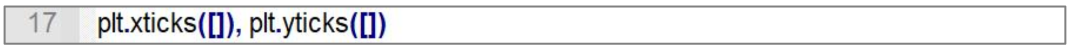
The fourth parameter “pad” is the padding distance (inside margin) between the tile and the subplot, “6.0” by default.

The fifth parameter “y” is the vertical distance between the title and the subplot, and the unit is the percentage of the height of the subplot. The default value is "None", that is, the position of the title is automatically determined to avoid overlapping with other elements. "1.0" means the title is at the top of the subplot.

The sixth parameter "**kwargs" is the text object keyword property, which is used to determine the appearance of the text, such as font, text color, etc.

◆ **Set Axis Tick**

Call xticks() and yticks() function in matplotlib.pyplot module to set the tick and tag of X and Y axis. As the coordinate axis is not required in image display in this routine, the list is set as none that is the coordinate axis will not be displayed.



```
plt.xticks([]), plt.yticks([])
```

The format of xticks() function is as follow.

```
xticks(ticks=None, labels=None, **kwargs)
```

When the parameter is none, the function will return the current tick and tag of X axis. Otherwise the function is used to set the current tick and label of X axis.

The first parameter "**ticks**" is a list of the positions of the X-axis ticks. If the list is empty, the X-axis ticks will be cleared.

The second parameter “**labels**” is the label of X-axis tick. Only when parameter “ticks” is not none, can this parameter be passed.

The third parameter “****kwargs**” is used to control the appearance of the tick and label.

The format of yticks() is the same as that of xticks(). The difference lies in the controlled object.

- **Display Image**

Call show() function in matplotlib.pyplot module to display the image on the window.

```
plt.show()
```

The complete codes of image display part are as follow.

```
plt.figure(figsize=(8, 5), dpi=100)
plt.rcParams['axes.unicode_minus'] = False
plt.subplot(121), plt.imshow(img, cmap=plt.cm.gray), plt.title("Original")
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(canny, cmap=plt.cm.gray), plt.title("Edge Detection")
plt.xticks([]), plt.yticks([])
plt.show()
```