OpenCV

05 幾何轉換

06 影像強化

Ex:05-01 scaling

```
import numpy as np
import cv2

img1 = cv2.imread( "Lenna.bmp", -1 )
nr, nc = img1.shape[:2]
scale = eval( input( "Please enter scale: " ) )
nr2 = int( nr * scale )
nc2 = int( nc * scale )
img2 = cv2.resize( img1, ( nr2, nc2 ), interpolation = cv2.INTER_LINEAR )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Image Scaling", img2 )
cv2.waitKey( 0 )
```

Ex:05-02 rescaling

```
import numpy as np
import cv2

img = cv2.imread( "Baboon.bmp", 0 )
nr1, nc1 = img.shape[:2]
nr2, nc2 = nr1 // 4, nc1 // 4
img1 = cv2.resize( img, ( nr2, nc2 ), interpolation = cv2.INTER_NEAREST )
img1 = cv2.resize( img1, ( nr1, nc1 ), interpolation = cv2.INTER_NEAREST )
img2 = cv2.resize( img, ( nr2, nc2 ), interpolation = cv2.INTER_LINEAR )
img2 = cv2.resize( img2, ( nr1, nc1 ), interpolation = cv2.INTER_NEAREST )
img3 = cv2.resize( img, ( nr2, nc2 ), interpolation = cv2.INTER_CUBIC )
img3 = cv2.resize( img2, ( nr1, nc1 ), interpolation = cv2.INTER_NEAREST )
cv2.imshow( "Original Image", img )
cv2.imshow( "Nearest Neighbor", img1 )
cv2.imshow( "Bilinear", img2 )
cv2.imshow( "Bicubic", img3 )
cv2.waitKey( 0 )
```

Ex:05-03 rotation

```
import numpy as np
import cv2
```

```
img1 = cv2.imread( "Lenna.bmp", -1 )
nr2, nc2 = img1.shape[:2]
rotation_matrix = cv2.getRotationMatrix2D( ( nr2 / 2, nc2 / 2 ), 30, 1 )
img2 = cv2.warpAffine( img1, rotation_matrix, ( nr2, nc2 ) )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Image Rotation", img2 )
cv2.waitKey( 0 )
```

Ex:05-04 Flip

```
import numpy as np
import cv2

img = cv2.imread( "Baboon.bmp", -1 )
img1 = cv2.flip( img, 0 )
img2 = cv2.flip( img, 1 )
cv2.imshow( "Original Image", img )
cv2.imshow( "Flip Vertically", img1 )
cv2.imshow( "Flip Horizontally", img2 )
cv2.waitKey( 0 )
```

Ex:05-05 Affine

```
import numpy as np
import cv2

img1 = cv2.imread( "Poker.bmp", -1 )
nr, nc = img1.shape[:2]
pts1 = np.float32( [ [ 160, 165 ], [ 240, 390 ], [ 270, 125 ] ] )
pts2 = np.float32( [ [ 190, 140 ], [ 190, 375 ], [ 310, 140 ] ] )
T = cv2.getAffineTransform( pts1, pts2 )
img2 = cv2.warpAffine( img1, T, ( nc, nr ) )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Affine Transform", img2 )
cv2.waitKey( 0 )
cv2.imwrite( "O.bmp", img2 )
```

Ex:05-06 perspective

```
import numpy as np
import cv2

img1 = cv2.imread( "Gallery.bmp", -1 )
nr, nc = img1.shape[:2]
pts1 = np.float32( [ [ 795, 350 ], [ 795, 690 ], [ 1090, 720 ], [ 1090, 250 ] ] )
pts2 = np.float32( [ [ 0, 0 ], [ 0, 500 ], [ 650, 500 ], [ 650, 0 ] ] )
T = cv2.getPerspectiveTransform( pts1, pts2 )
img2 = cv2.warpPerspective( img1, T, ( 650, 500 ) )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Perspective Transform", img2 )
cv2.waitKey( 0 )
```

Ex:05-07 平移

```
import numpy as np
import cv2

img = cv2.imread("lena.bmp")

height, width = img.shape[:2]
x = 100
y = 200
M = np.float32([[1, 0, x],[0, 1, y]])
move = cv2.warpAffine(img, M, (width, height))

cv2.imshow( "Original Image", img)
cv2.imshow( "move", move)
cv2.waitKey()
cv2.destroyAllWindows()
```

Ex:05-08 旋轉

```
import cv2

img = cv2.imread("lena.bmp")
```

```
height, width = img.shape[:2]

M = cv2.getRotationMatrix2D((width/2,height/2),45,0.6)

rotate = cv2.warpAffine(img, M, (width, height))

cv2.imshow( "Original Image", img)
cv2.imshow( "rotation", rotate)
cv2.waitKey()
cv2.destroyAllWindows()
```

Ex:05-09 Affine

```
import cv2
import numpy as np

img = cv2.imread("lena.bmp")
rows, cols, ch = img.shape

p1 = np.float32([[0, 0],[cols-1, 0],[0, rows-1]])
p2 = np.float32([[0, rows*0.33],[cols*0.85, rows*0.25],[cols*0.15, rows*0.7]])

M = cv2.getAffineTransform(p1, p2)
dst = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow( "Original", img)
cv2.imshow( "result", dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

Ex:06-01 image negative

```
import numpy as np
import cv2

def image_negative( f ):
    g = 255 - f
    return g

def main( ):
    img1 = cv2.imread( "Lenna.bmp", -1 )
    img2 = image_negative( img1 )
    cv2.imshow( "Original Image", img1 )
    cv2.imshow( "Image Negative", img2 )
    cv2.waitKey( 0 )

main( )
```

Ex:06-02 gamma

```
import numpy as np
import cv2

def gamma_correction( f, gamma = 2.0 ):
    g = f.copy( )
    nr, nc = f.shape[:2]
    c = 255.0 / ( 255.0 ** gamma )
    table = np.zeros( 256 )
    for i in range( 256 ):
        table[i] = round( i ** gamma * c, 0 )
    if f.ndim != 3:
        for x in range( nr ):
            for y in range( nc ):
                g[x,y] = table[f[x,y]]
    else:
        for x in range( nr ):
            for y in range( nc ):
                for k in range( 3 ):
                    g[x,y,k] = table[f[x,y,k]]
```

```
    return g

def main( ):
    img = cv2.imread( "Museum.bmp", 0 )
    img1 = gamma_correction( img, 0.1 )
    img2 = gamma_correction( img, 0.2 )
    img3 = gamma_correction( img, 0.5 )
    cv2.imshow( "Original Image", img )
    cv2.imshow( "Gamma = 0.1", img1 )
    cv2.imshow( "Gamma = 0.2", img2 )
    cv2.imshow( "Gamma = 0.5", img3 )
    cv2.waitKey( 0 )
```

Ex:06-03 beta

```
import numpy as np
import cv2
import scipy.special as special

def beta_correction( f, a = 2.0, b = 2.0 ):
    g = f.copy( )
    nr, nc = f.shape[:2]
    x = np.linspace( 0, 1, 256 )
    table = np.round( special.betainc( a, b, x ) * 255, 0 )
    if f.ndim != 3:
        for x in range( nr ):
            for y in range( nc ):
                g[x,y] = table[f[x,y]]
    else:
        for x in range( nr ):
            for y in range( nc ):
                for k in range( 3 ):
                    g[x,y,k] = table[f[x,y,k]]
    return g

def main( ):
    img = cv2.imread( "Building.bmp", 0 )
    img1 = beta_correction( img, a = 0.5, b = 0.5 )
    img2 = beta_correction( img, a = 2.0, b = 2.0 )
```

```
    cv2.imshow( "Original Image", img )
    cv2.imshow( "Beta Correction (a = b = 0.5)", img1 )
    cv2.imshow( "Beta Correction (a = b = 2.0)", img2 )
    cv2.waitKey( 0 )

main( )
```

Ex:06-04 convolution

```
import numpy as np

x = np.array( [ 1, 2, 4, 3, 2, 1, 1 ] )
h = np.array( [ 1, 2, 3, 1, 1 ] )
y = np.convolve( x, h, 'full' )
y1 = np.convolve( x, h, 'same' )
print( "x =", x )
print( "h =", h )
print( "Full Convolution y =", y )
print( "Convolution y =", y1 )
```

Ex:06-05 convolution 2D

```
import numpy as np
from scipy.signal import convolve2d

x = np.array( [ [1, 1, 1], [1, 1, 1], [1, 1, 1] ] )
h = np.array( [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] )
y = convolve2d( x, h, 'same' )
print( "x =" )
print( x )
print( "h =" )
print( h )
print( "Convolution y =" )
print( y )
```

Ex:06-06 average filter

```
import numpy as np
import cv2
```

```
img1 = cv2.imread( "Lenna.bmp", -1 )
img2 = cv2.blur( img1, ( 5, 5 ) )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Average Filtering", img2 )
cv2.waitKey( 0 )
```

Ex:06-07 gaussian

```
import numpy as np
import cv2

img1 = cv2.imread( "Lenna.bmp", -1 )
img2 = cv2.GaussianBlur( img1, ( 5, 5 ), 0 )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Gaussian Filtering", img2 )
cv2.waitKey( 0 )
```

Ex:06-08 gradient

```
import numpy as np
import cv2

def Sobel_gradient( f, direction = 1 ):
    sobel_x = np.array( [ [-1,-2,-1], [ 0, 0, 0], [ 1, 2, 1] ] )
    sobel_y = np.array( [ [-1, 0, 1], [-2, 0, 2], [-1, 0, 1] ] )
    if direction == 1:
        grad_x = cv2.filter2D( f, cv2.CV_32F, sobel_x )
        gx = abs( grad_x )
        g = np.uint8( np.clip( gx, 0, 255 ) )
    elif direction == 2:
        grad_y = cv2.filter2D( f, cv2.CV_32F, sobel_y )
        gy = abs( grad_y )
        g = np.uint8( np.clip( gy, 0, 255 ) )
    else:
        grad_x = cv2.filter2D( f, cv2.CV_32F, sobel_x )
        grad_y = cv2.filter2D( f, cv2.CV_32F, sobel_y )
        magnitude = abs( grad_x ) + abs( grad_y )
        g = np.uint8( np.clip( magnitude, 0, 255 ) )
```

```
    return g

def main( ):
    img = cv2.imread( "Osaka.bmp", -1 )
    gx  = Sobel_gradient( img, 1 )
    gy  = Sobel_gradient( img, 2 )
    g   = Sobel_gradient( img, 3 )
    cv2.imshow( "Original Image", img )
    cv2.imshow( "Gradient in x", gx )
    cv2.imshow( "Gradient in y", gy )
    cv2.imshow( "Gradient", g )
    cv2.waitKey( 0 )

main( )
```

Ex:06-09 Laplacian

```
import numpy as np
import cv2

def laplacian( f ):
    temp = cv2.Laplacian( f, cv2.CV_32F ) + 128
    g = np.uint8( np.clip( temp, 0, 255 ) )
    return g

def main( ):
    img1 = cv2.imread( "Osaka.bmp", -1 )
    img2 = laplacian( img1 )
    cv2.imshow( "Original Image", img1 )
    cv2.imshow( "Laplacian", img2 )
    cv2.waitKey( 0 )

main( )
```

Ex:06-10 Composite Laplacian

```
import numpy as np
import cv2
```

```
def composite_laplacian( f ):
    kernel = np.array( [ [0, -1, 0], [-1, 5, -1], [0, -1, 0] ] )
    temp = cv2.filter2D( f, cv2.CV_32F, kernel )
    g = np.uint8( np.clip( temp, 0, 255 ) )
    return g

def main( ):
    img1 = cv2.imread( "Osaka.bmp", -1 )
    img2 = composite_laplacian( img1 )
    cv2.imshow( "Original Image", img1 )
    cv2.imshow( "Composite Laplacian", img2 )
    cv2.waitKey( 0 )

main( )
```

Ex:06-11 unsharp

```
import numpy as np
import cv2

def unsharp_masking( f, k = 1.0 ):
    g = f.copy( )
    nr, nc = f.shape[:2]
    f_avg = cv2.GaussianBlur( f, ( 15, 15 ), 0 )
    for x in range( nr ):
        for y in range( nc ):
            g_mask = int( f[x,y] ) - int( f_avg[x,y] )
            g[x,y] = np.uint8( np.clip( f[x,y] + k * g_mask, 0, 255 ) )
    return g

def main( ):
    img1 = cv2.imread( "Osaka.bmp", -1 )
    img2 = unsharp_masking( img1, 10.0 )
    cv2.imshow( "Original Image", img1 )
    cv2.imshow( "Unsharp Masking", img2 )
    cv2.waitKey( 0 )

main( )
```

Ex:06-12 bilateral filter

```
import numpy as np
import cv2

img1 = cv2.imread( "Jenny.bmp", -1 )
img2 = cv2.bilateralFilter( img1, 11, 50, 50 )
cv2.imshow( "Original Image", img1 )
cv2.imshow( "Bilateral Filtering", img2 )
cv2.waitKey( 0 )
```