Digital Image Processing Project5

By

賴知榆
310512025

CODE (DEMO.PY)

```
from plot_util import plot_and_save
from copy import deepcopy
import os, sys
import numpy as np
import cv2, math

class gray_img:
    def __init__(self, img_path) -> None:
        self.imsize = None
        self.img = self.read_img(img_path)

    def read_img(self, img_path):
        type = ['tif', 'jpg', 'jpeg', 'png']
        img_type = img_path.split('.')[-1]
        if img_type in type:
            img = cv2.imread(img_path, 0)
            img = np.array(img / 255, dtype=np.float32)
            self.imsize = img.shape
        return img

    def gaussian_blur(self, img):
        short_dim = self.imsize[0] if self.imsize[0] <
self.imsize[1] else self.imsize[1]
        blur_img = cv2.GaussianBlur(img, ksize=(3, 3),
sigmaX=short_dim, sigmaY=short_dim)
        return blur_img

    def sobel_operator(self, img):
        from scipy import ndimage
        # Sobel gradient (d)
        gx = np.array([ [ -1,  0, 1],
                        [ -2,  0, 2],
                        [ -1,  0, 1]], dtype='float32')
        gy = np.array([ [ 1,  2,  1],
                        [ 0,  0,  0],
                        [-1, -2, -1]], dtype='float32')
        gx = cv2.filter2D(img, -1, kernel=gx)
        gy = cv2.filter2D(img, -1, kernel=gy)
        # sobel = sqrt(gx^2 + gy^2)
        G = np.hypot(gx, gy)
        G = G / G.max()
        # edge direction
        theta = np.arctan2(gy, gx)

        return (G, theta)

    def non_max_suppression(self, img, D):
        img = deepcopy(img) * 255
        M, N = img.shape
        Z = np.zeros((M,N), dtype=np.int32)
        angle = D * 180. / np.pi
        angle[angle < 0] += 180


        for i in range(1,M-1):
            for j in range(1,N-1):
                try:
                    q = 255
                    r = 255

                    #angle 0
                    if (0 <= angle[i,j] < 22.5) or (157.5 <=
angle[i,j] <= 180):
                        q = img[i, j+1]
                        r = img[i, j-1]
                    #angle 45
                    elif (22.5 <= angle[i,j] < 67.5):
                        q = img[i+1, j-1]
                        r = img[i-1, j+1]
                    #angle 90
                    elif (67.5 <= angle[i,j] < 112.5):
                        q = img[i+1, j]
                        r = img[i-1, j]
                    #angle 135
                    elif (112.5 <= angle[i,j] < 157.5):
                        q = img[i-1, j-1]
                        r = img[i+1, j+1]

                    if (img[i,j] >= q) and (img[i,j] >= r):
                        Z[i,j] = img[i,j]
                    else:
                        Z[i,j] = 0

                except IndexError as e:
                    pass
        return Z / 255.0

    def threshold(self, img, lowThresholdRatio=0.04, 
highThresholdRatio=0.10):
        img = deepcopy(img) * 255
        highThreshold = img.max() * highThresholdRatio
        lowThreshold = img.max() * lowThresholdRatio

        M, N = img.shape
        res = np.zeros((M,N), dtype=np.float)

        weak = np.int32(100)
        strong = np.int32(255)

        strong_i, strong_j = np.where(img >= highThreshold)
        zeros_i, zeros_j = np.where(img < lowThreshold)

        weak_i, weak_j = np.where((img <= highThreshold) & 
(img >= lowThreshold))

        res[strong_i, strong_j] = strong
        res[weak_i, weak_j] = weak
        res = res / 255.0

        strong_img = np.zeros((M,N), dtype=np.float)
        weak_img   = np.zeros((M,N), dtype=np.float)
        strong_img[strong_i, strong_j] = strong
        weak_img[weak_i, weak_j] = weak

        return (res, weak, strong, weak_img, strong_img)

    def hysteresis(self, img, weak, strong=255):
        img = np.array(deepcopy(img) * 255, dtype=np.int)
        M, N = img.shape
        for i in range(1, M-1):
            for j in range(1, N-1):
                if (img[i,j] == weak):
                    try:
                        if ((img[i+1, j-1] == strong) or 
(img[i+1, j] == strong) or (img[i+1, j+1] == strong)
                            or (img[i, j-1] == strong) or 
(img[i, j+1] == strong)
                            or (img[i-1, j-1] == strong) or 
(img[i-1, j] == strong) or (img[i-1, j+1] == strong)):
```

```
                            img[i, j] = strong
                        else:
                            img[i, j] = 0
                    except IndexError as e:
                        pass
        return img / 255.0

def main():

    img_name = 'Kid at playground.tif'
    # img_name = 'test.jpeg'
    gray = gray_img(img_name)

    # Gaussian blur with, sigma = (0.5% of the shortest
dimension of the image)
    blur_img = gray.gaussian_blur(gray.img)

    # Sobel operator for computing gradient vectors
    sobel_img, angle = gray.sobel_operator(blur_img)

    # Non-maxumum suppression
    NMS_img = gray.non_max_suppression(sobel_img, angle)

    # Threshold
    th_img, weak, strong, weak_img, strong_img =
gray.threshold(NMS_img, lowThresholdRatio=0.04,
highThresholdRatio=0.10)

    # Hysteresis
    hysteresis_img = gray.hysteresis(th_img, weak, strong)

    # Canny edge detection in opencv
    src = np.array(gray.img * 255, dtype=np.uint8)
    cv2_canny = cv2.Canny(src, threshold1=100, threshold2=200)

    img_list = []
    name_list = []
    img_list.append(gray.img)
    name_list.append('origin')
    img_list.append(blur_img)
    name_list.append('blur_img')
    img_list.append(sobel_img)
    name_list.append('sobel_img')
    img_list.append(NMS_img)
```

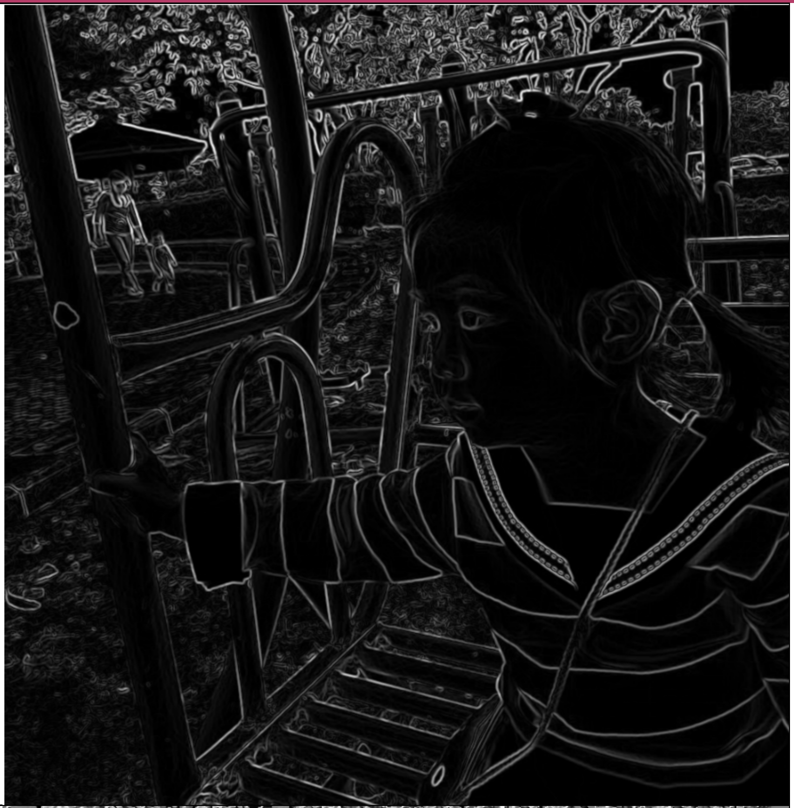
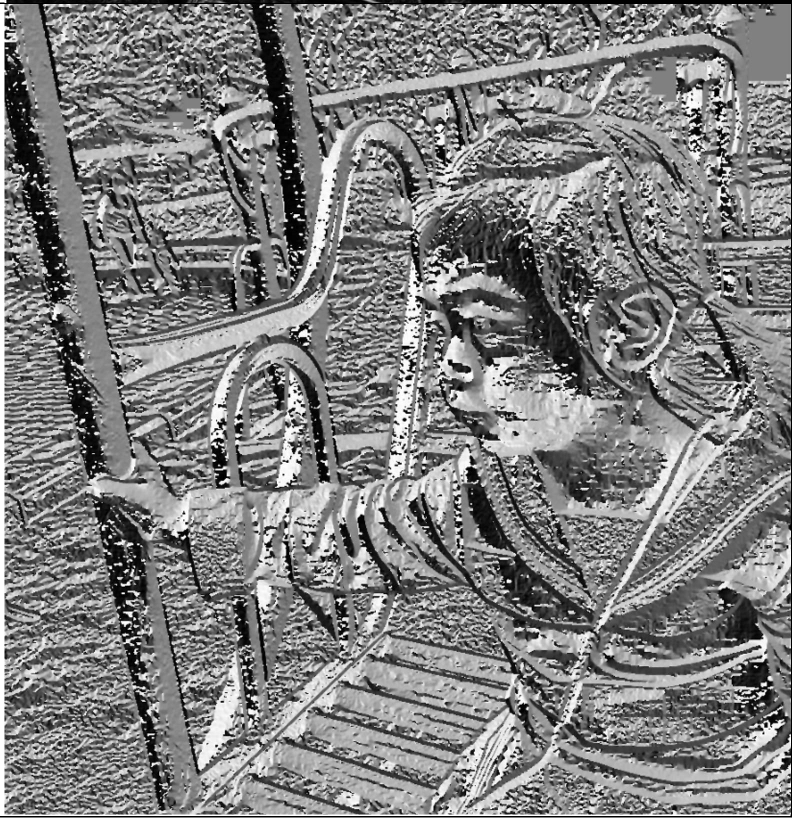
```python
    name_list.append('NMS_img')
    img_list.append(th_img)
    name_list.append('th_img')
    img_list.append(weak_img)
    name_list.append('weak_img')
    img_list.append(strong_img)
    name_list.append('strong_img')
    img_list.append(hysteresis_img)
    name_list.append('hysteresis_img')
    img_list.append(cv2_canny)
    name_list.append('cv2_canny')
    plot_and_save(
        'Canny edge detection',
        img_list,
        name_list,
        type='gray',
        dpi=200,
        plot=True,
        save=True,
        col_size=3,
        show_axis=0
    )

if __name__ == "__main__":
    main()
```

CODE (PLOT_UTIL.PY)

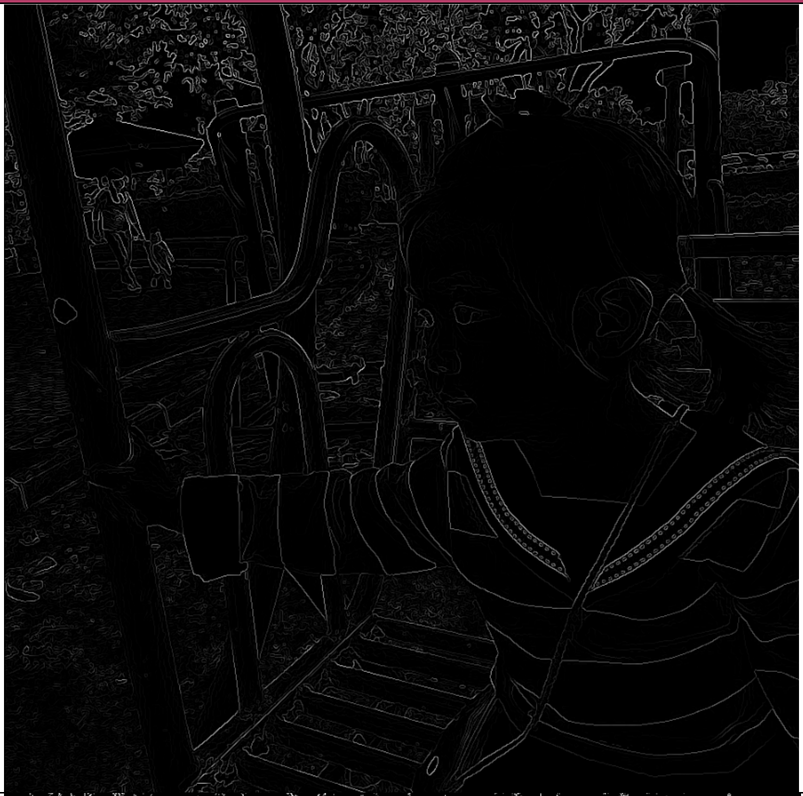
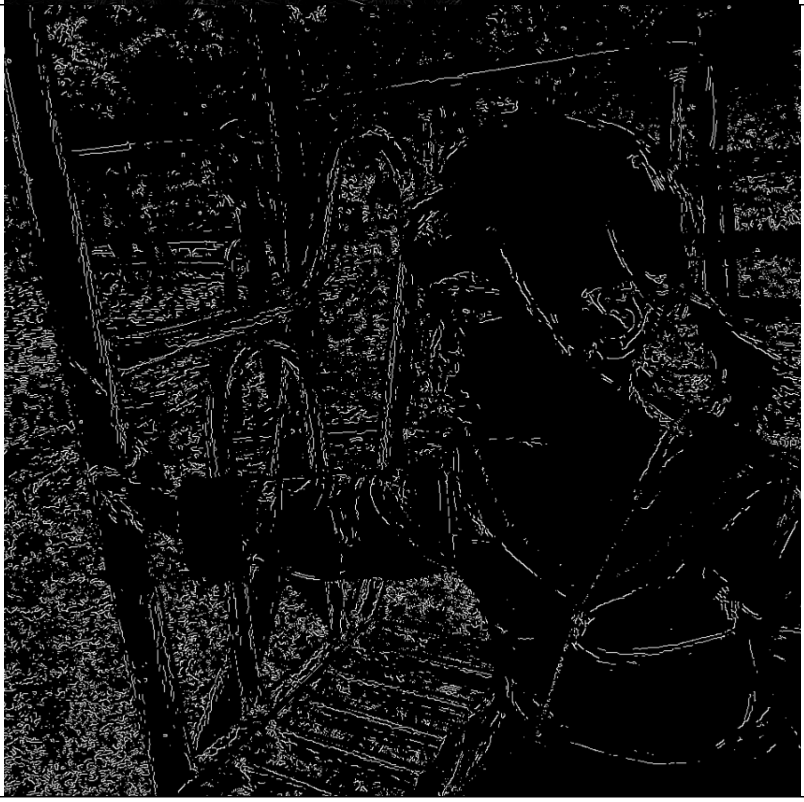
```
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt

def plot_and_save(demo_name, imgs, filenames, type='gray',
dpi=200, plot=True, save=True, col_size=4, show_axis=1):
    if type == 'rgb':
        type = None
    size = len(imgs)
    if save:
        root = 'results'
        result_path = os.path.join(root, demo_name)
        imgs_path = os.path.join(result_path, 'imgs')
        if not os.path.exists(root):
            os.mkdir(root)
        if not os.path.exists(result_path):
            os.mkdir(result_path)
        if not os.path.exists(imgs_path):
            os.mkdir(imgs_path)
        for i in range(size):
            img_path = imgs_path + '/' + filenames[i] + '.png'
            plt.figure()
            plt.title(filenames[i])
            if not show_axis: plt.axis('off')
            plt.tight_layout()
            plt.imshow(imgs[i], cmap=type)
            plt.savefig(img_path, bbox_inches='tight',
dpi=dpi)
        plt.close('all')

    if plot:
        plt.figure(figsize=(16, 10))
        img_in_row = col_size
        r = size // img_in_row + (size % img_in_row > 0)
        c = img_in_row
        for i in range(size):
            plt.subplot(r, c, i+1)
            plt.title(filenames[i])
            if not show_axis: plt.axis('off')
            plt.imshow(imgs[i], cmap=type)
        plt.suptitle(demo_name + ' images')
```

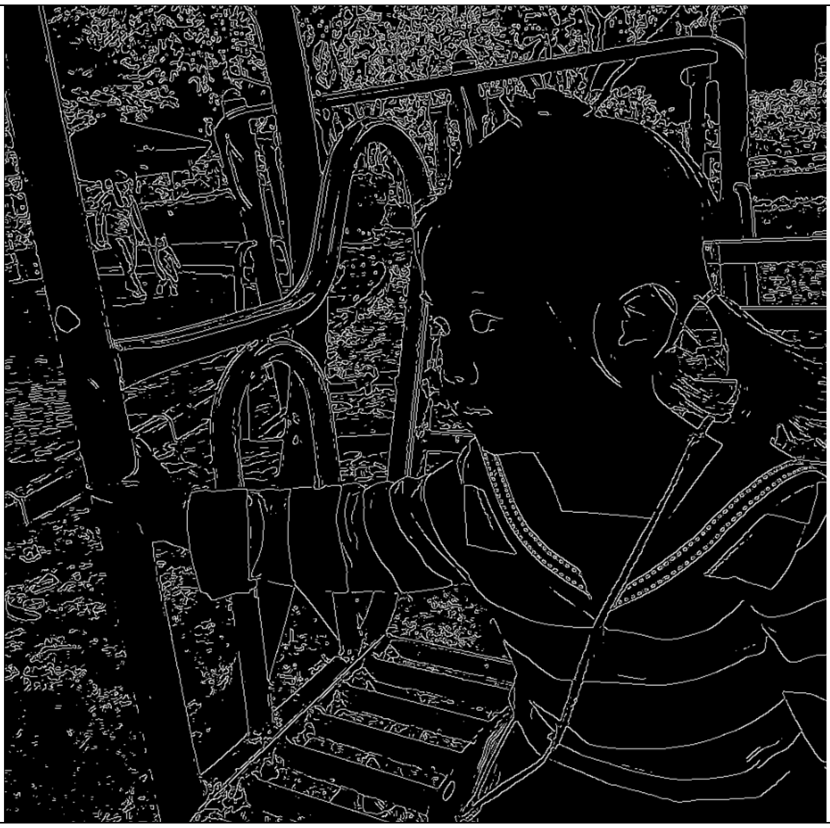
```python
        plt.tight_layout()
        plt.show()
        plt.close('all')

def show_hist(demo_name, gray_imgs, filenames, plot=True,
save=True, density=True):
    size = len(gray_imgs)
    if save:
        root = 'results'
        result_path = os.path.join(root, demo_name)
        hist_path = os.path.join(result_path, 'hist')
        if not os.path.exists(root):
            os.mkdir(root)
        if not os.path.exists(result_path):
            os.mkdir(result_path)
        if not os.path.exists(hist_path):
            os.mkdir(hist_path)
        for i in range(size):
            img_path = hist_path + '/' + filenames[i] + '.png'
            plt.figure()
            plt.title(filenames[i])
            plt.hist(gray_imgs[i].ravel(), 256, [0,256])
            plt.tight_layout()
            plt.savefig(img_path, bbox_inches='tight')
        plt.close('all')

    if plot:
        plt.figure(figsize=(16, 9))
        r = size // 4 + (size % 4 > 0)
        c = 4
        for i in range(size):
            plt.subplot(r, c, i+1)
            plt.title(filenames[i])
            plt.hist(gray_imgs[i].ravel(), 256, [0,256])
        plt.suptitle(demo_name + ' histograms')
        plt.tight_layout()
        plt.show()
        plt.close('all')

    hists = []
    for i in range(size):
        hists.append(np.histogram(gray_imgs[i].ravel(), 256,
[0, 256], density=density))
    return hists
```

IMAGES OF THE GRADIENT MAGNITUDE AND GRADIENT ANGLE

Magnitude	
Angle	

NONMAXIMA SUPPRESSED IMAGE GN(X,Y) AS WELL AS IMAGES OF GNL(X,Y) AND GNH(X,Y)

g_N	
g_{NL}	

g_{NH}	

FINAL EDGE MAP E(X,Y)

hysteresis_img