Digital Image Processing Project 5

Name:彭晨益

Student ID: 310512054

(a)Source Code

```
import os
import sys
import cv2
import copy
import numpy as np

from tqdm import tqdm
from PIL import Image
import matplotlib.pyplot as plt

def check_folder(path):
    if not os.path.exists(path):
        os.makedirs(path)
    return path

class CannyEdgeDetector():

    def __init__(self, img_path):
        self.img = cv2.imread(img_path, 0)
        self.norm_img =  self.img / 255.
        self.sigma = min(self.img.shape[:2]) * 0.005
        self.gaus_img  = self.gaussian_smooth(self.norm_img, (3,3), self.sigma)
        self.magnitude, self.angle = self.sobel_filter(self.gaus_img)
        self.Gn = self.non_maximum_supress(self.magnitude, self.angle)
        self.thres_result, self.gnh, self.gnl = self.hysteresis_thres(self.Gn, 0.01,
0.1)

    def gaussian_smooth(self, image, kernel_size: tuple, sigma: float):
        blur_img = cv2.GaussianBlur(image, kernel_size, sigma)
        return blur_img

    def sobel_filter(self, image):
        # Ms(x, y) = sqrt(Gx**2 + Gy**2)
        # alpha(x, y) = Gangle
```

```
        Gx_filter = np.array([[-1,-2,-1]
                              ,[ 0, 0, 0]
                              ,[ 1, 2, 1]])
        Gy_filter = np.array([[-1, 0, 1]
                              ,[-2, 0, 2]
                              ,[-1, 0, 1]])

        Gx = cv2.filter2D(image, ddepth=-1, kernel=Gx_filter)
        Gy = cv2.filter2D(image, ddepth=-1, kernel=Gy_filter)

        # Extract the angle from the Gx, Gy
        n, m = Gx.shape[:2]
        angle = np.zeros(Gx.shape)
        for i in range(n):
            for j in range(m):
                angle[i][j] =  np.arctan2(Gy[i][j], Gx[i][j])

        magnitude = np.sqrt(Gx**2 + Gy**2)

        return magnitude, angle

    def non_maximum_supress(self, magnitude_map, direction_map):
        nonmaxima_supress_img = np.zeros(magnitude_map.shape)
        n, m = magnitude_map.shape[:2]

        # solve the corner case during the processing
        # make each pixel has complete neighbors
        magnitude_map_pad = np.pad(magnitude_map, pad_width=1, mode='constant',
constant_values=0)

        for i in range(1, n+1):
            for j in range(1, m+1):
                degrees = np.rad2deg(direction_map[i-1][j-1])
                # decide the pixel is belong to which domain d1~d4
                # D1: horizontal edge
                if (degrees <= 22.5 and degrees > -22.5) or (degrees > 157.5 and
degrees <= 180) or (degrees >= -180 and degrees <= -175.5):
                    # compare top and down
                    if magnitude_map_pad[i][j] > magnitude_map_pad[i-1][j] and
magnitude_map_pad[i][j] > magnitude_map_pad[i+1][j]:
                        nonmaxima_supress_img[i-1][j-1] = magnitude_map_pad[i][j]
                # D2: -45 degree edge
```

```
                elif (degrees <= 67.5 and degrees > 22.5) or (degrees > -157.5 and
degrees <= -112.5):
                    # compare top-left and down-right
                    if magnitude_map_pad[i][j] > magnitude_map_pad[i-1][j-1] and
magnitude_map_pad[i][j] > magnitude_map_pad[i+1][j+1]:
                        nonmaxima_supress_img[i-1][j-1] = magnitude_map_pad[i][j]
                # D3: vertical edge
                elif (degrees <= 112.5 and degrees > 67.5) or (degrees > -112.5 and
degrees <= -67.5):
                    # compare left and right
                    if magnitude_map_pad[i][j] > magnitude_map_pad[i][j+1] and
magnitude_map_pad[i][j] > magnitude_map_pad[i][j-1]:
                        nonmaxima_supress_img[i-1][j-1] = magnitude_map_pad[i][j]
                # D4: 45 degree edge
                elif (degrees <= 157.5 and degrees > 112.5) or (degrees > -67.5 and
degrees <= -22.5):
                    # compare top-right and down-left
                    if magnitude_map_pad[i][j] > magnitude_map_pad[i-1][j+1] and
magnitude_map_pad[i][j] > magnitude_map_pad[i+1][j-1]:
                        nonmaxima_supress_img[i-1][j-1] =
magnitude_map_pad[i][j]

        return nonmaxima_supress_img

    def hysteresis_thres(self, Gn, thres_low, thres_high):
        # Recommend ratio of high and low is T_high:T_low = 2:1 to 3:1
        gnh = np.zeros(Gn.shape)
        gnl = np.zeros(Gn.shape)
        output_image_pad = np.pad(gnl, pad_width=1, mode='constant',
constant_values=0)
        n, m = Gn.shape[:2]
        for i in range(1, n+1):
            for j in range(1, m+1):
                pixel = Gn[i-1, j-1]
                if pixel >= thres_high:
                    gnh[i-1, j-1] = 1
                    output_image_pad[i, j] = 1
                elif pixel >= thres_low:
                    if output_image_pad[i-1, j] == 1 or output_image_pad[i+1, j] == 1
or output_image_pad[i, j-1] == 1 or output_image_pad[i, j+1] == 1 \
                        or output_image_pad[i-1, j-1] == 1 or output_image_pad[i-1,
j+1] == 1 or output_image_pad[i+1, j-1] == 1 or output_image_pad[i+1, j+1] == 1:
```

```
                        output_image_pad[i, j] = 1
                    gnl[i-1, j-1] = 1

        return output_image_pad[1:n+1, 1:m+1], gnh, gnl

    def visualize(self):
        cv2.imshow("Original image", self.norm_img)
        cv2.imshow("gaussian smooth", self.gaus_img)
        cv2.imshow("sobel", self.magnitude)
        cv2.imshow("nonmaximun supress", np.float64(self.Gn))
        cv2.imshow("gnh", self.gnh)
        cv2.imshow("gnl", self.gnl)
        cv2.imshow("hysteresis threshold", self.thres_result)
        cv2.waitKey()
        cv2.destroyAllWindows()

    def save_single_img(self, img, img_title, save):
        if save:
            check_folder("result")
            image = Image.fromarray((img*255))
            if image.mode == "F":
                    image = image.convert('L')
            image.save("result"+ '/' +img_title + ".jpg", dpi=(200.0, 200.0, 0))
        else:
            pass

def main():
    # class CannyEdgeDetector will do the Canny edge detection
    # you can get the detection result by call -> img.thres_result
    img = CannyEdgeDetector("Kid at playground.tif")

    save_image_list = [img.magnitude, img.angle, img.Gn, img.gnl, img.gnh,
img.thres_result]
    save_image_title = ['gradient magnitude', 'gradient angle', 'nonmaxima
suppressed', 'G_NL', 'G_NH', 'Final Edge Map']

    for i in range(len(save_image_list)):
        img.save_single_img(save_image_list[i], save_image_title[i], save=True)

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

(b) Plot images of the gradient magnitude and gradient angle:

Magnitude:

Angle:

(c) Plot nonmaxima suppressed image g_N (x,y) as well as images of g_{NL}(x,y) and g_{NH}(x,y):

g_N (x,y):

g_{NL}(x,y):

$g_{NH}(x,y)$

(d) Plot final edge map e(x,y):