Project 5

ID：311512064 name：鄧書桓

(a) Source codes:

import numpy as np

import cv2

import matplotlib.pyplot as plt

from PIL import Image

from random import randint

import math

import random

from scipy import ndimage

def gaussian\_kernel(size, sigma):

    size = int(size) // 2

    x, y = np.mgrid[-size:size+1, -size:size+1]

    normal = 1 / (2.0 \* np.pi \* sigma\*\*2)

    g = np.exp(-((x\*\*2 + y\*\*2) / (2.0\*sigma\*\*2))) \* normal

    return g

def sobel\_filters(img):

    # kernel

    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)

    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

    Ix = ndimage.filters.convolve(img, Kx)

    Iy = ndimage.filters.convolve(img, Ky)

    G = np.hypot(Ix, Iy)

    G = G / G.max()

    theta = np.arctan2(Iy, Ix)

    return (G, theta)

def non\_max\_suppression(img, D):

    M, N = img.shape

    Z = np.zeros((M, N), dtype=np.float32)

    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

def threshold(img):

    highThreshold = 0.1

    lowThreshold = 0.04

    M, N = img.shape

    res = np.zeros((M, N), dtype=np.float32)

    weak = np.float32(0.5)

    strong = np.float32(1)

    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))

    img\_weak = np.zeros((img.shape[0], img.shape[1]), dtype=np.float32)

    img\_strong = np.zeros((img.shape[0], img.shape[1]), dtype=np.float32)

    img\_weak[weak\_i, weak\_j] = weak

    img\_strong[strong\_i, strong\_j] = strong

    res[strong\_i, strong\_j] = strong

    res[weak\_i, weak\_j] = weak

    return (res, img\_weak, img\_strong)

def hysteresis(img, weak, strong=1):

    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

if \_\_name\_\_ == "\_\_main\_\_":

    # 位啥要-1

    image = cv2.imread('Kid at playground.tif', -1)

    normalized\_image = image/255

    # guassion LPF

    # 位啥size是31？

    GLP = gaussian\_kernel(31, sigma=5)

    img\_glp = cv2.filter2D(normalized\_image, -1, GLP)

    cv2.imshow("Gaussian", img\_glp)

    # sobel gradient calculation

    magnitude, theta = sobel\_filters(img\_glp)

    cv2.imshow("magnitude", magnitude)

    cv2.imshow("angle", theta)

    PILimage = magnitude\*255

    PILimage = Image.fromarray(PILimage.astype(np.uint8))

    PILimage.save("img/magnitude.png", dpi=(200, 200))

    PILimage = (theta+np.pi/2)/np.pi\*255

    PILimage = Image.fromarray(PILimage.astype(np.uint8))

    PILimage.save("img/theta.png", dpi=(200, 200))

    # nunmaxima suppression

    nun\_sup = non\_max\_suppression(magnitude, theta)

    cv2.imshow("suppression", nun\_sup)

    PILimage = nun\_sup\*255

    PILimage = Image.fromarray(PILimage.astype(np.uint8))

    PILimage.save("img/non\_max\_suppression.png", dpi=(200, 200))

    # hysteresis thresholding

    final\_img, gNL, gNH = threshold(nun\_sup)

    final = hysteresis(final\_img, 0.5, 1)

    cv2.imshow("final\_img", final\_img)

    cv2.imshow("gNL", gNL)

    cv2.imshow("gNH", gNH)

    cv2.imshow("final", final)

    cv2.waitKey(0)

    PILimage = final\_img\*255

    PILimage = Image.fromarray(PILimage.astype(np.uint8))

    PILimage.save("img/final\_img.png", dpi=(200, 200))

    PILimage = gNL\*255

    PILimage = Image.fromarray(PILimage.astype(np.uint8))

    PILimage.save("img/gNL.png", dpi=(200, 200))

    PILimage = gNH\*255

    PILimage = Image.fromarray(PILimage.astype(np.uint8))

    PILimage.save("img/gNH.png", dpi=(200, 200))

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

|  |  |
| --- | --- |
| magnitude | Angle |

(c) Plot nonmaxima suppressed image gN (x,y) as well as images of gNL(x,y) and gNH(x,y):

|  |  |
| --- | --- |
| gN (x,y) | gNL(x,y) |
| gNH(x,y) |

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

|  |
| --- |
|  |