تمرین Features

آریا ابراهیمی ۹۸۲۲۷۶۲۱۷۵

چکیده

در این تمرین، به بررسی ویژگی‌ها و الگوریتم های مختلف شناسایی آن‌ها می‌پردازیم. در بخش اول، الگوریتم گوشه یاب Harris بررسی شده و آنرا روی تصویری با scale های متفاوت بررسی می کنیم. به این نتیجه می‌رسیم که الگوریتم Harris در scale های مختلف تصویر به خوبی عمل نمیکند در نتیجه الگوریتم SIFT را معرفی کرده و با استفاده از آن نقاطی کلیدی در چند تصویر از یک فضا را به دست آورده و آن‌ها را روی یکدیگر align میکنیم تا به تصویر Panaroma موقعیت برسیم.

۱- تحلیل تکنیکال

برای این تمرین از کتابخانه OpenCV Contrib استفاده شده است که دارای الگوریتم SIFT است. (در OpenCV از یک ورژن به بعد الگوریتم های SIFT و SURF قابلیت استفاده ندارند)

۱.۱.۱. یکی از ویژگی هایی که میتواند در تصاویر تمایز ایجاد کند، گوشه ها می‌باشند. الگوریتم هریس، الگوریتمی است که گوشه های تصویر را براساس مقادیر ویژه پیدا می کند. اگر دو مقدار ویژه بزرگ در یک ناحیه داشته باشیم نشان‌دهنده آن است که در این ناحیه یک گوشه داریم.
گام های الگوریتم هریس به صورت زیر می‌باشد:

۱. در گام اول باید مشتق های تصویر را در راستای افقی و عمودی پیدا کنیم. این کار را میتوان با اعمال فیلتر Robert یا Sobel بر روی تصویر انجام داد. در این تمرین از فیلتر Sobel استفاده شده است. مشتق ها را Ix و Iy می‌نامیم که اولی مشتق در راستای x و دیگری مشتق در راستای y است.
۲. در گام بعد، I_x^2 و I_y^2 و I_xI_y را محاسبه کرده و سپس پنجره هایی را در نظر گرفته و در ناحیه آنها، مجموع سه قسمت را حساب می کنیم و ماتریس هریس را تشکیل میدهیم.
۳. بعد از تشکیل ماتریس برای هر پنجره، برای یافتن اینکه ناحیه‌ای گوشه است یا خیر، مقدار R را به دست آورده و اگر این مقدار بزرگ و مثبت باشد، نشان‌دهنده گوشه است. اگر منفی شود یعنی یک مقدار ویژه بزرگ وجود دارد و در نتیجه نشاندهنده لبه است و اگر کوچک باشد نشاندهنده plain است.
۴. در گام آخر یک threshold در نظر گرفته و نقاطی که مقدار بزرگتری از threshold دارند به عنوان نقاط کلیدی در نظر گرفته می‌شوند.
۵. در آخر در مکان نقاط کلیدی مشخص شده، روی تصویر اصلی دایره هایی را در نظر میگیریم تا نقاط قابل تفکیک باشند.

$$R = det(H) - \alpha\, tr(H)^2$$

نکته ای که باید در نظر داشت این است که از آنجایی که میخواهیم نقاط را با دایره های رنگی مشخص کنیم، تصویر را ابتدا gray کرده و دوباره آنرا به RGB تبدیل میکنیم تا نقاط را بتوان به صورت واضح نمایش داد.

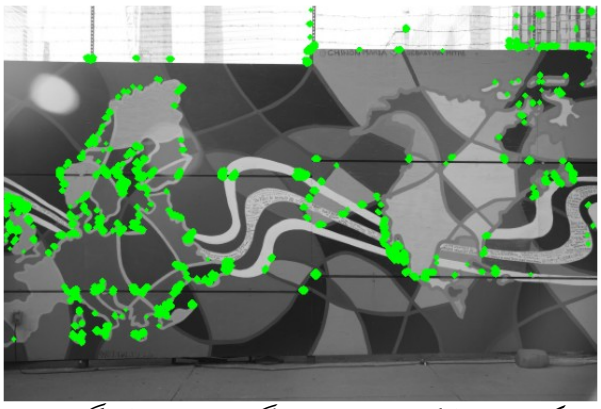
۱.۲.۱. و ۱.۲.۲ در این دو تمرین خواسته شده است تا با استفاده از الگوریتم SIFT ، نقاط کلیدی را در تصاویر پیدا کرده و سپس آنها را روی یکدیگر align کنیم و تصویری همانند تصویر Panaroma تشکیل دهیم. تابعی به نام match پیاده سازی شده است که دو تصویر را به عنوان ورودی دریافت کرده و الگوریتم SIFT را با استفاده از پیاده سازی آن در کتابخانه OpenCV روی ورودی ها انجام میدهد. در ادامه با استفاده از تابع BFMatcher ، نقاط کلیدی را با یکدیگر match میکنیم و در نهایت، یک threshold در نظر گرفته و فقط آنهایی که match خوبی هستند را نگهداری میکنیم. برای نمایش اینکه کدام نقاط با یکدیگر match شده اند، از تابع drawMatchesKnn استفاده میکنیم که دو تصویر را کنار یکدیگر قرار میدهد و نقاط مچ شده

را با یک خط به یکدیگر متصل میکند. نتایج در قسمت تحلیل نتایج قابل مشاده هستند.

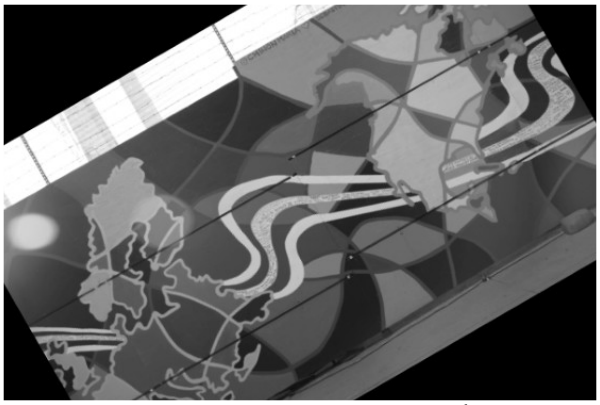
در گام بعد تابع Panaroma برای align کردن تصاویر پیاده سازی شده است. این تابع از تابع stitch استفاده میکند که فقط دو تصویر را align میکند. تابع Panaroma در هر iteration، نتیجه قبل را با تصویر جدیدی align میکند. تابع stitch دو تصویر را به عنوان ورودی گرفته و تقریبا همانند تابع match عمل میکند. ابتدا SIFT را برای دو تصویر انجام میدهد و نقاط کلیدی خوب را پیدا میکند و درنهایت با استفاده از تابع findHomography که در OpenCV پیاده سازی شده است، نقاطی را که باید روی هم قرار بگیرند را پیدا میکند. در تابع wrap با استفاده از homography به دست آمده، translation مورد نیاز روی تصویر انجام میشود تا بتوان نقاط کلیدی را روی هم align کرد. نتایج این بخش نیز در قسمت تحلیل نتایج قابل مشاهده است.

۲- تحلیل نتایج

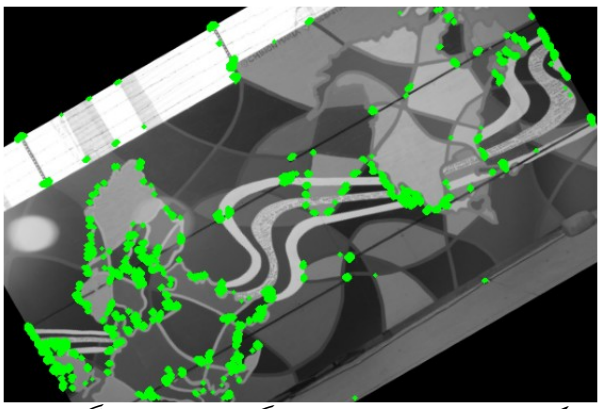
۲.۱.۱



شکل ۱- نقاط کلیدی پیدا شده (گوشه ها) توسط الگوریتم هریس برای تصویر اصلی.

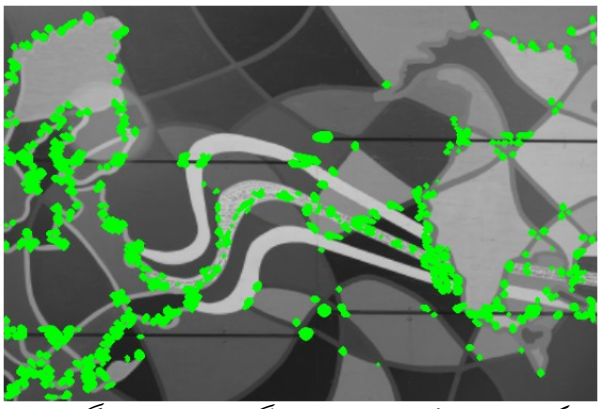


شکل ۲- تصویر با ۴۰ درجه چرخش.



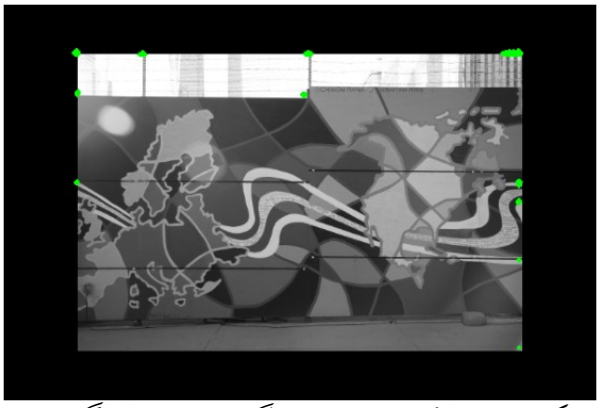
شکل ۳- نقاط کلیدی پیدا شده (گوشه ها) توسط الگوریتم هریس برای تصویر با ۴۰ درجه چرخش.

شکل ۴- تصویر زوم شده با scale برابر با ۱.۵



شکل ۵- نقاط کلیدی پیدا شده (گوشه ها) توسط الگوریتم هریس برای تصویر زوم شده.

شکل ۶- تصویر unzoom شده با scale برابر با ۰.۷۵.

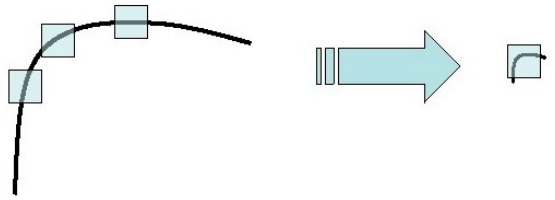


شکل ۷- نقاط کلیدی پیدا شده (گوشه ها) توسط الگوریتم هریس برای تصویر unzoom شده.

شکل ۸- تصویر unzoom شده با scale برابر با ۰.۵.

شکل ۹- نقاط کلیدی پیدا شده (گوشه ها) توسط الگوریتم هریس برای تصویر unzoom شده.

میتوان مشاهده کرد که الگوریتم هریس برای rotation خوب عمل میکند ولی برای scale های متفاوت تصویر، خوب عمل نمیکند و باید پارامتر فاصله و یا پنجره آنرا تغییر داد تا بتواند نقاط را دوباره شناسایی کند. با دقت در شکل ۱۰ میتوان به این قضیه پی برد. در یک اسکیل با یک اندازه، یک نقطه میتواند گوشه باشد در صورتی که در یک scale دیگر ممکن است لبه تشخیص داده شود.



شکل ۱۰- الگوریتم کوشه یاب هریس برای شناسایی کوشه ها در scale های متفاوت خوب عمل نمیکند.

این باعث میشود تا به سراغ الگوریتم های SIFT و SURF برویم که scale invariant هستند.

۲.۲.۱

در این بخش، به بررسی نتایج الگوریتم SIFT میپردازیم. در ابتدا فقط ویژگی ها را در چند تصویر با استفاده از SIFT پیدا کرده و آنها را با یکدیگر align میکنیم. در ادامه از SIFT استفاده کرده و تصاویر را با استفاده از ویژگی هایشان، روی هم قرار داده و تصویری همانند تصویر پاناروما میسازیم.

نتیجه match کردن ویژگی ها برای تصاویر sl, sm و sr در شکل ۱۱ و شکل ۱۲ قابل مشاهده است.

همانطور که مشاهده میشود اکثر ویژگی ها به درستی با یکدیگر تطابق پیدا کرده اند.

شکل ۱۱- تطابق ویژگی های شناسایی شده با الگوریتم SIFT برای دو تصویر sl و sm.

شکل ۱۲- تطابق ویژگی های شناسایی شده با الگوریتم SIFT برای دو تصویر sm و sr.

شکل ۱۳- تصویر نهایی تابع panaroma که هر سه تصویر را روی هم تطابق داده است.

شکل ۱۴- تصویر خروجی تابع panaroma برای قسمت آخر.

همانطور که مشاهده میشود، الگوریتم SIFT به خوبی در شناسایی ویژگی های عمل میکند و در شرایط نوری متفاوت هم میتواند ویژگی های منحصر به فردی شناسایی کند. (شکل ۱۳)

۲.۲.۲ برای این تمرین، سه تصویر از داخل دانشکده گرفته شده است و با استفاده از همان تابع تمرین قبل، با استفاده از SIFT ویژگی هایشان پیدا شده است و با استفاده از تابع Homography نقاطی که باید روی هم align شوند پیدا شده اند. نتیجه در شکل ۱۴ قابل مشاهده است.

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

def show_img(*args, figsize=10, is_gray=True, title=None, fontsize=12):
    if isinstance(figsize, int):
        figsize = (figsize, figsize)
    images = args[0] if type(args[0]) is list else list(args)
    cmap=None
    if not is_gray:
        images = list(map(lambda x: cv2.cvtColor(x, cv2.COLOR_BGR2RGB), images))
    else:
        cmap = 'gray'
    plt.figure(figsize=figsize)
    for i in range(1, len(images)+1):
        plt.subplot(1, len(images), i)
        if title is not None:
            plt.title(title[i-1], fontsize=fontsize)

        plt.imshow(images[i-1], cmap=cmap)
        plt.axis('off')

img = cv2.imread('img.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def harris(img, window_size, alpha ,threshold):

    gray_bgr = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    gauss = cv2.GaussianBlur(img,(3,3),0)

    height = img.shape[0]
    width = img.shape[1]
    result = np.zeros((height,width))

    Ix = cv2.Sobel(gauss, cv2.CV_64F, 1, 0, ksize=3)
    Iy = cv2.Sobel(gauss, cv2.CV_64F, 0, 1, ksize=3)

    Ixx = np.square(Ix)
    Iyy = np.square(Iy)
    Ixy = Ix * Iy

    index = int(window_size/2)

    for y in range(index, height-index):
        for x in range(index, width-index):
            WIxx = np.sum(Ixx[y-index:y+1+index, x-index:x+1+index])
            WIyy = np.sum(Iyy[y-index:y+1+index, x-index:x+1+index])
            WIxy = np.sum(Ixy[y-index:y+1+index, x-index:x+1+index])

            M = np.array([[WIxx,WIxy],[WIxy,WIyy]])

            det = np.linalg.det(M)
            trace = np.matrix.trace(M)
            R = det - alpha*(trace**2)

            result[y-index, x-index]=R

    cv2.normalize(result, result, 0, 1, cv2.NORM_MINMAX)

    for x in range(index, height-index):
        for y in range(index, width-index):
            value = result[x, y]

            if value > threshold:
                cv2.circle(gray_bgr, (y, x), radius=2, color=(0,255,0), thickness=-1)


    return gray_bgr

scale_1 = harris(img, 5, 0.04, 0.18)
show_img(scale_1)
plt.show()

def rotate(img, angle):
    center = img.shape[1]//2, img.shape[0]//2
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(img, M, (img.shape[1], img.shape[0]))
    return rotated

rotated_1 = rotate(img, 30)
show_img(rotated_1)
plt.show()

scale_2 = harris(rotated_1, 5, 0.04, 0.512)
show_img(scale_2)
plt.show()

def scale(img, scale):
    center = img.shape[1]//2, img.shape[0]//2
    M = cv2.getRotationMatrix2D(center, 0, scale)
    scaled = cv2.warpAffine(img, M, (img.shape[1], img.shape[0]))
    return scaled

scaled_1 = scale(img, 1.5)
show_img(scaled_1)
plt.show()

scale_3 = harris(scaled_1, 5, 0.04, 0.33)
show_img(scale_3)
plt.show()

scaled_2 = scale(img, 0.75)
show_img(scaled_2)
plt.show()

scale_4 = harris(scaled_2, 5, 0.04, 0.18)
show_img(scale_4)
plt.show()

scaled_3 = scale(img, 0.5)
show_img(scaled_3)
plt.show()

scale_5 = harris(scaled_3, 5, 0.04, 0.3)
show_img(scale_5)
plt.show()

sl = cv2.imread('sl.jpg')
sm = cv2.imread('sm.jpg')
sr = cv2.imread('sr.jpg')

def match(img1, img2):
    sift = cv2.SIFT_create(300)
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)

    good_matches = []
    for m, n in matches:
        if m.distance < 0.75*n.distance:
            good_matches.append([m])

    result = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good_matches, None,
flags=cv2.DrawMatchesFlags_DEFAULT)
    return result, (good_matches, kp1, kp2)

a1, _ = match(sl, sm)
a2, _ = match(sm, sr)
a3, _ = match(sl, sr)

show_img(a1, figsize=30)

show_img(a2, figsize=30)

def wrap(img1,img2,H):
    rows1, cols1 = img1.shape[:2]
    rows2, cols2 = img2.shape[:2]

    list_of_points_1 = np.float32([[0,0], [0,rows1], [cols1,rows1],
[cols1,0]]).reshape(-1,1,2)
    temp_points = np.float32([[0,0], [0,rows2], [cols2,rows2],
[cols2,0]]).reshape(-1,1,2)
    list_of_points_2 = cv2.perspectiveTransform(temp_points, H)
    list_of_points = np.concatenate((list_of_points_1, list_of_points_2), axis=0)

    [x_min, y_min] = np.int32(list_of_points.min(axis=0).ravel() - 0.5)
    [x_max, y_max] = np.int32(list_of_points.max(axis=0).ravel() + 0.5)
    translation_dist = [-x_min, -y_min]
    H_translation = np.array([[1, 0, translation_dist[0]], [0, 1,
translation_dist[1]], [0,0,1]])

    output_img = cv2.warpPerspective(img1, H_translation.dot(H), (x_max-x_min, y_max-
y_min))
    output_img[translation_dist[1]:rows1+translation_dist[1],
translation_dist[0]:cols1+translation_dist[0]] = img2

    return output_img

def align_size(img1,img2):
    if(img1.shape!=img2.shape):
        maxHeight = img1.shape[0] if img1.shape[0] >= img2.shape[0] else img2.shape[0]
        maxWidth = img1.shape[1] if img1.shape[1] >= img2.shape[1] else img2.shape[1]
        img1 = cv2.copyMakeBorder(img1, 0, maxHeight - img1.shape[0], 0, maxWidth -
img1.shape[1], cv2.BORDER_CONSTANT, value=[0,0,0])
        img2 = cv2.copyMakeBorder(img2, 0, maxHeight - img2.shape[0], 0, maxWidth -
img2.shape[1], cv2.BORDER_CONSTANT, value=[0,0,0])
    return img1,img2

def stitch(img1,img2):

    img1, img2 = align_size(img1, img2)
    sift = cv2.SIFT_create()
    bf = cv2.BFMatcher()

    kp_1, desc_1 = sift.detectAndCompute(img1, None)
    kp_2, desc_2 = sift.detectAndCompute(img2, None)

    matches = bf.knnMatch(desc_1, desc_2, k=2)
    good_matches = []

    for m, n in matches:
        if m.distance < 0.75*n.distance:
            good_matches.append(m)

    src_pts = np.float32([ kp_1[m.queryIdx].pt for m in good_matches
]).reshape(-1,1,2)
    dst_pts = np.float32([ kp_2[m.trainIdx].pt for m in good_matches
]).reshape(-1,1,2)

    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    wrap_im = wrap(img1, img2, H)

    return wrap_im

def panaroma(images):
    res = images[0]
    for i in range(1, len(images)):

        res = stitch(res, images[i])
    return res

images = [sl, sm, sr]
res = panaroma(images)
show_img(res, figsize=30, is_gray=False)

img1 = cv2.imread('1.jpg')
img2 = cv2.imread('2.jpg')
img3 = cv2.imread('3.jpg')

res = panaroma([img3, img2, img1])

show_img(res, figsize=30, is_gray=False)
```