به نام خدا

درس مبانی بینایی کامپیوتر

تمرین سری ششم

مدرس درس:
جناب آقای دکتر محمدی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۸/۲۲

سوال ۱:

فرض کنید یک مثلث قائم‌الزاویه در اختیار است که لبه‌های آن به دست آمده است. علاوه بر لبه‌های اضلاع، ۱۰۰ لبه‌ی دیگر نیز به دلیل نویزی بودن محیط به دست آمده است. برای ضلع وتر ۱۲۰ لبه و برای دو ضلع عمود بر هم نیز به ترتیب ۸۰ و ۶۰ لبه یافت شده است. اگر بخواهیم با احتمال ٪۹۰ ضلع وتر را با الگوریتم RANSAC به دست بیاوریم، حداقل چند بار باید الگوریتم اجرا شود؟ برای احتمال ٪۹۹ چطور؟ (۲۰ نمره)

پاسخ ۱:

طبق الگوریتم RANSAC، رابطه زیر را داریم:

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

در رابطه فوق، k نشان دهنده تعداد دفعات تکرار الگوریتم، p احتمال یافتن مجموعه درست و w بیانگر نسبت نقاط inlier به کل نقاط می‌باشد. در اینجا برای ضلع وتر، ۱۲۰ نقطه inlier داریم و باقی نقاط که در مجموع ۲۴۰ نقطه می‌شوند، جز نقاط outlier هستند. بنابراین مقدار w به شکل زیر محاسبه می‌شود:

$$w = \frac{120}{120+240} = \frac{1}{3}$$

حال برای یافتن مجموعه درست با احتمال ٪۹۰، مقدار k را به دست می‌آوریم:

$$k = \frac{\log(1-0.9)}{\log(1-(\frac{1}{3})^2)} = 19.55$$

بنابراین برای یافتن وتر با احتمال ٪۹۰، لازم است این الگوریتم ۲۰ بار تکرار شود. حال مقدار k را برای احتمال ٪۹۹ محاسبه می‌کنیم:

$$k = \frac{\log(1-0.99)}{\log(1-(\frac{1}{3})^2)} = 39.1$$

با توجه به جواب حاصل، تعداد تکرار لازم برای یافتن وتر با احتمال ٪۹۰، ۴۰ بار می‌باشد.

سوال ۲:

داخل نوتبوک پیوست شده، تصویر LineDetection.jpg را بخوانید. با استفاده از کتابخانه OpenCV و تابع‌های آماده برای الگوریتم Hough ،خطوطی را روی آن پیدا کنید که حداقل ۲۵۰ رای آورده باشد. (۱۰ نمره)

مشاهده می‌شود که تمام خانه‌های مربعی مستقلا جدا شده‌اند، اما اگر بخواهیم طول خطوط پیدا شده را نیز در پیدا شدن خطوط و همچنین بیشترین فاصله نقاط لبه را از یکدیگر نیز دخیل کنیم، از الگوریتم Probabilistic Transform Hough استفاده می‌شود. پارامترهای این تابع را طوری تنظیم نمایید که روی همان تصویر نخست، بهترین نتیجه را بیابید. این پارامترها را در گزارش خود ذکر نمایید. (۱۰ نمره)

*امتیازی: برای حل سوال تبدیل Hough را از پایه پیاده سازی کنید و به‌جای فراخوانی تابع آماده، تابع پیاده‌سازی شده را فراخوانی کنید و خروجی‌های بدست آمده از تابع آماده را با خروجی‌های تابع خودتان مقایسه کنید. (۱۵ نمره)

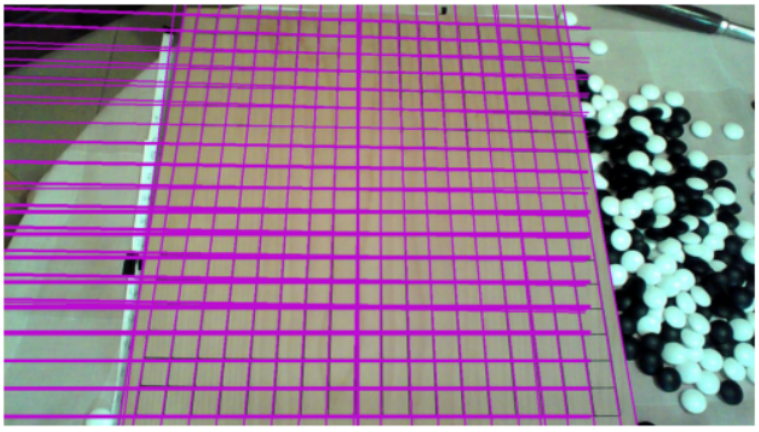
پاسخ ۲:

- در ابتدا تصویر را خوانده و سپس لبه‌های آن را با استفاده از لبه‌یاب canny پیدا می‌کنیم. سپس با استفاده از تبدیل Hough، خط‌های موجود در تصویر را به دست می‌آوریم. ورودی‌های این تابع به ترتیب لبه‌ها، مقدار ρ، θ و حد آستانه می‌باشد و خروجی آن، خط‌های یافت شده است. سپس به ازای هر خط یافت شده، ρ و θ آن را یافته و پس از محاسبه x_1، x_2، y_1 و y_2، آن خط را رسم می‌کنیم و تصویر را نمایش می‌دهیم. در واقع ρ و θ را به فضای x و y می‌بریم. کد و نتیجه حاصل از این بخش به شرح تصویر زیر است.

```
#TODO
path = r'E:\University\Term7\FCV\Homeworks\HW6\Images'
img = cv2.imread(os.path.join(path,"LineDetection.jpg"))
canny_img = cv2.Canny(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), 50, 150)
lines = cv2.HoughLines(canny_img, 1, np.pi/180, 250)
for line in lines:
    p, t = line[0]
    x1 = int(p * np.cos(t) + 1000*(-np.sin(t)))
    y1 = int(p * np.sin(t) + 1000*(np.cos(t)))
    x2 = int(p * np.cos(t) - 1000*(-np.sin(t)))
    y2 = int(p * np.sin(t) - 1000*(np.cos(t)))
    cv2.line(img,(x1,y1),(x2,y2),(209,12,187),2)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

✓ 0.4s



لینک مورد استفاده در این سوال:

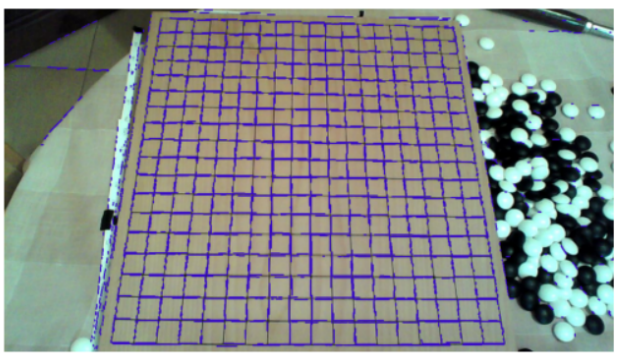
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

- در روش قبل پارامترهای خطوط را دریافت کرده و تمام نقاط را پیدا می‌کند؛ اما در این روش، مستقیما دو نقطه پایانی خطوط را برمی‌گرداند. پارامترهای ورودی این تابع، به ترتیب تصویر باینری (که همان تصویر لبه‌یابی شده است)، مقادیر ρ و θ، اندازه حد آستانه و دو پارامتر اختیاری که یکی برای تعیین کردن خطوط با طول حداقل اندازه آن مقدار، که یعنی خطوطی با طول برابر یا بیشتر از آن پیدا شوند و ماکزیمم فاصله بین خطوط می‌باشد.

```
#TODO
img = cv2.imread(os.path.join(path,"LineDetection.jpg"))
canny_img = cv2.Canny(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), 50, 150)
lines = cv2.HoughLinesP(canny_img,1,np.pi/180,90, 15, 3)
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img,(x1,y1),(x2,y2),(197,10,55),2)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

✓ 0.2s



همانطور که در شکل مشخص است، بهترین حالات پارامترهای این تابع را به ترتیب ۹۰ برای حد آستانه، ۱۵ برای حداقل طول پاره‌خط و ۳ برای ماکزیمم فاصله بین خطوط برآورد شده است.

- در اینجا ابتدا تابع Hough را پیاده‌سازی می‌کنیم تا مقدار ρ و θ را به ما بدهد. برای این کار یک accumulator را با مقادیر اولیه ۰، initialize می‌کنیم. سپس به ازای هر پیکسل در تصویر و θ در range ۰ تا ۱۸۰، رابطه $\rho = xcos(\theta) + ysin(\theta)$ را به دست می‌آوریم و مقدار accumualtor مربوط به آن نقطه را ۱ واحد افزایش می‌دهیم. سپس تابع را فراخوانی می‌کنیم و مانند بخش اول، مقادیر خط‌های یافت شده در فضای ρ و θ را به دست می‌آوریم. سپس در حلقه for نوشته شده، فضای ρ و θ را به فضای x و y تبدیل می‌کنیم. کدهای مربوط

به این بخش، در تصویر زیر می‌باشد.

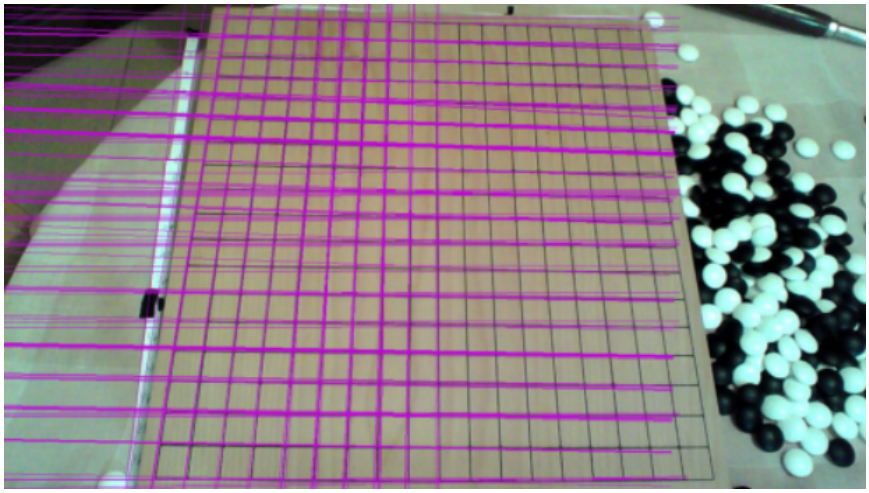
```python
def Hough_algorithm(img):
    w = img.shape[0]
    h = img.shape[1]
    thetas = np.arange(0, 180, 1)
    cos = np.cos(np.deg2rad(thetas))
    sin = np.sin(np.deg2rad(thetas))
    rhos = round(math.sqrt(w**2 + h**2))
    H = np.zeros((2 * rhos, len(thetas)), dtype=np.uint8)
    edges = np.where(img == 255)
    xys = list(zip(edges[0], edges[1]))
    for p in range(len(xys)):
        for t in range(len(thetas)):
            rho = int(round(xys[p][1] * cos[t] + xys[p][0] * sin[t]))
            H[rho, t] += 1
    return H
```

✓ 0.2s

```python
path = r'E:\University\Term7\FCV\Homeworks\HW5\Images'
image = cv2.imread(os.path.join(path,"lineDetection.jpg"))
edges = cv2.Canny(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY),50,150)
H = Hough_algorithm(edges)
edges = np.where(H > 200)
xys = list(zip(edges[0], edges[1]))
for i in range(0, len(xys)):
    x1 = int(np.cos(np.deg2rad(xys[i][1]))*xys[i][0] + 1000*(-np.sin(np.deg2rad(xys[i][1]))))
    y1 = int(np.sin(np.deg2rad(xys[i][1]))*xys[i][0] + 1000*(np.cos(np.deg2rad(xys[i][1]))))
    x2 = int(np.cos(np.deg2rad(xys[i][1]))*xys[i][0] - 1000*(-np.sin(np.deg2rad(xys[i][1]))))
    y2 = int(np.sin(np.deg2rad(xys[i][1]))*xys[i][0] - 1000*(np.cos(np.deg2rad(xys[i][1]))))
    cv2.line(image,(x1,y1),(x2,y2),(209,12,187),1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

✓ 43.7s

نتیجه حاصل از این بخش، تصویر زیر است:



همانطور که مشخص است، تابع پیاده‌سازی شده توسط خودمان، عملکرد ضعیف‌تری نسبت به تابع آماده HoughLine دارد.

نکته: در حل این سوال، از لینک زیر کمک گرفته شد.

https://github.com/Hank-Tsou/Hough-Transform-Line-Detection/blob/master/hough_line_transform.py

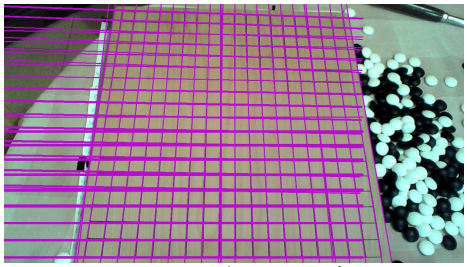
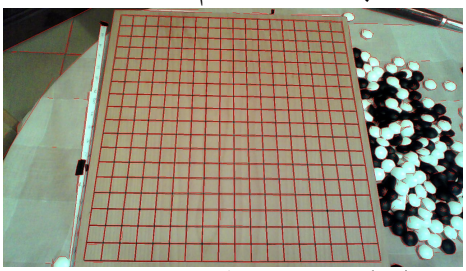
سوال ۳:

کد نوشته شده در نوتبوک چه کاری انجام می‌دهد؟ نتیجه را با سوال قبل روی همان تصویر مقایسه نمایید. (۲۰ نمره)

پاسخ ۳:

در این کد، به جای استفاده از الگوریتم Hough، از الگوریتم LSD برای تشخیص خطوط استفاده شده است. در این روش، از اندازه گرادیان استفاده نمی‌کند و از جهت گرادیان استفاده می‌کند. به این

صورت که نقاطی را که جهت گرادیانشان یکسان هستند و کنار هم هستند را به عنوان یک پاره خط در نظر می‌گیرد. با استفاده از این الگوریتم، مثلا مشکلی که الگوریتم Hough بر روی شانه داشت (اینکه خطوط افقی عمود بر دانه‌های شانه هم به عنوان خط در نظر می‌گرفت) نیز حل می‌شود و علاوه بر آن، مانند Hough دیگر به ما خط نمی‌دهد، بلکه یک پاره خط می‌دهد. ابتدا حاصل خروجی تصویر با هر یک از الگوریتم‌های Hough و LSD را نمایش داده و سپس به تفسیر آن‌ها می‌پردازیم. تصویر سمت چپ، نتیجه الگوریتم Hough و تصویر سمت راست خروجی LSD می‌باشد.

همانطور که در تصویر بالا نیز مشخص است، در روش Hough یک‌سری خط تشخیص داده شده‌اند؛ اما در روش LSD، پاره خط تشخیص داده شده است که مرزها را برای ما آشکارتر می‌کند. به علاوه در روش Hough، به ازای یک خط در تصویر، چندین خط تشخیص داده شده است؛ اما در روش LSD اینگونه نیست و دقت بیشتری دارد. به این دلیل که روش LSD از جهت گرادیان استفاده می‌کند و به علاوه هر پاره خط به جای ۲ پارامتر، با ۴ پارامتر مشخص می‌شود که همه این‌ها باعث افزایش دقت ما تا حد خوبی می‌شود و خطها به صورت واضح‌تری تشخیص داده می‌شوند. از دیگر تفاوت‌های این دو تصویر، می‌توان به این موضوع اشاره کرد که الگوریتم LSD، خطهای موجود در پس‌زمینه تصویر که جزئیات هستند را هم به عنوان خط تشخیص داده است، در حالی که الگوریتم Hough، فقط خطهای موجود در صفحه شطرنجی را تشخیص داده است.

سوال ۴:

دو تابعی طراحی نمایید که تبدیل RGB به CMYK و بلعکس را انجام دهد. (توجه نمایید که مقیاس RGB ۲۵۵ و CMYK درصد می‌باشد.) (۲۰ نمره)

پاسخ ۴:

- ابتدا رابطه تبدیل RGB به CMYK را می‌نویسیم:

$$k = 1 - max(R, G, B)$$
$$C = \frac{(1-R-K)}{1-k}$$
$$M = \frac{(1-G-K)}{1-k}$$
$$Y = \frac{(1-B-K)}{1-k}$$

سپس چون در اینجا RGB scale و CMYK scale را به ما داده است، ابتدا R، B و G را بر RGB_scale تقسیم می‌کنیم تا عددی بین ۰ و ۱ به ما بدهد. چرا که رابطه‌های بالا،

در حالت نرمالیزه بودن صادق هستند. سپس با استفاده از رابطه‌های بالا، کد خود را برای تبدیل RGB به CMYK می‌نویسیم. البته باید توجه کنیم چون مقیاس CMYK هم به ما داده است، باید حاصل معادلات بالا را در CMYK_scale ضرب کنیم. کد و نتیجه مربوط به این بخش را در تصویر زیر مشاهده می‌نمایید.

```
def rgb_to_cmyk(r, g, b, RGB_SCALE = 255, CMYK_SCALE = 100):

    #TODO
    r = r / RGB_SCALE
    g = g / RGB_SCALE
    b = b / RGB_SCALE
    k = 1 - np.max([r, g, b])
    c = int((1 - k - r) * CMYK_SCALE / (1 - k))
    m = int((1 - k - g) * CMYK_SCALE / (1 - k))
    y = int((1 - k - b) * CMYK_SCALE / (1 - k))
    return c, m, y, int(k * 100)
```

✓ 0.2s

```
rgb_to_cmyk(25, 56, 25)
```

✓ 0.4s

```
(55, 0, 55, 78)
```

Expected Output: (55, 0, 55, 78)

- حال رابطه تبدیل CMYK به RGB را می‌نویسیم:

$$R = 255 * (1 - C) * (1 - K)$$
$$G = 255 * (1 - M) * (1 - K)$$
$$B = 255 * (1 - Y) * (1 - K)$$

معادله بالا نیز در حالت نرمالیزه صادق است، یعنی C، M، Y و K باید عددی بین ۰ و ۱ باشند. بنابراین این مقادیر را باید بر CMYK_scale تقسیم کنیم. سپس حاصل عبارات بالا را در RGB_scale ضرب کنیم تا مقادیر R، G و B را به صورت عددی بین ۰ تا ۲۵۵ به ما بدهد. کد مربوط به تابع زده شده در این بخش نیز، به شرح زیر می‌باشد.

```
def cmyk_to_rgb(c, m, y, k, CMYK_SCALE = 100, RGB_SCALE = 255):

    #TODO
    r = int(((1 - (c / CMYK_SCALE)) * (1 - (k / 100)) * RGB_SCALE))
    g = int(((1 - (m / CMYK_SCALE)) * (1 - (k / 100)) * RGB_SCALE))
    b = int(((1 - (y / CMYK_SCALE)) * (1 - (k / 100)) * RGB_SCALE))
    return r, g, b
```

✓ 0.2s

```
cmyk_to_rgb(55, 0, 55, 78)
```

✓ 0.2s

```
(25, 56, 25)
```

Expected Output: (25, 56, 25)

سوال ۵:

فرض کنید فضای رنگی یک پیکسل به صورت (R, G, B) = (150, 65, 200) باشد، پارامترهای HSI، V، L و Y را به صورت کد نویسی محاسبه و پیاده‌سازی نمایید. (۲۰ نمره)

پاسخ ۵:

در این بخش نیز فرمول‌های مربوط به هر یک از پارامترها را می‌نویسیم و شروع به پیاده‌سازی آن‌ها می‌کنیم:

$$\theta = \arccos\left(\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

$$H = \begin{cases} \theta & \text{if } B \leq G \\ 360 - \theta & \text{if } B < G \end{cases}$$

$$S = 1 - 3\frac{min(R,G,B)}{R+G+B}$$

$$I = \frac{R+G+B}{3}$$

$$V = max(R,G,B)$$

$$L = \frac{max(R,G,B) + min(R,G,B)}{2}$$

$$Y = 0.299R + 0.587G + 0.114B$$

حال با توجه به این فرمول‌ها، کد خود را پیاده‌سازی کرده و یک تست از آن می‌گیریم.

```python
def convert(R, G, B):
    theta = (np.arccos(((R-G) + (R-B))/(2 * math.sqrt(pow(R-G, 2)+((R-B)*(G-B)))))) * 180) / np.pi
    if B <= G:
        H = theta
    else:
        H = 360 - theta
    S = 1 - (3 * ((np.min([R, G, B]))/(R+G+B)))
    I = ((R+G+B) / 3) / 255
    V = np.max([R, G, B]) / 255
    L = ((np.max([R, G, B]) + np.min([R, G, B]))/(2)) / 255
    Y = (0.299 * R) + (0.587 * G) + (0.114 * B)
    print ("H is: ", H)
    print ("S is: ", S)
    print ("I is: ", I)
    print ("V is: ", V)
    print ("L is: ", L)
    print ("Y is: ", Y)
```

✓ 0.3s

```python
convert(25, 56, 25)
```

✓ 0.4s

```
H is:  120.00000000000001
S is:  0.2924528301886793
I is:  0.13856209150326798
V is:  0.2196078431372549
L is:  0.1588235294117647
Y is:  43.197
```