Computer Vision
Dr. Mohammadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW6

۱.

① w = نسبت تعداد نقاط inlier به تمام نقاط

p = احتمال یافتن یک مجموعه از نقاط بدون outlier

$$k = \frac{\log(1-p)}{\log(1-w^2)} \qquad w = \frac{120}{120+100+80+60} = \frac{1}{3}$$

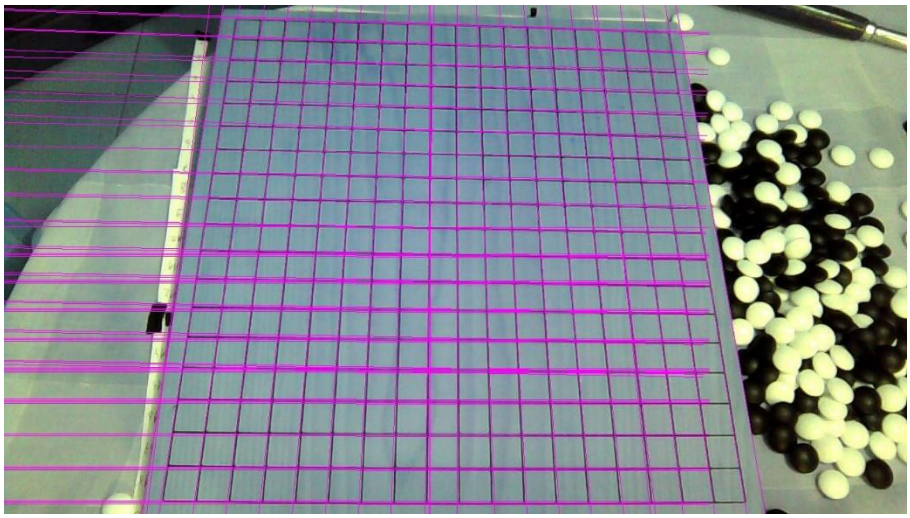
$$p=0.9 \rightarrow k = \frac{\log(1-0.9)}{\log\left(1-\frac{1}{9}\right)} = \frac{-1}{-0.05} = 20 \text{ بار}$$

$$p=0.99 \rightarrow k = \frac{\log(1-0.99)}{-0.05} = \frac{-2}{-0.05} = 40 \text{ بار}$$

۲. برای قسمت اول ابتدا تصویر را graysclae می‌کنیم. سپس با استفاده از لبه‌یاب canny لبه‌های تصویر را شناسایی کرده و ذخیره می‌کنیم. سپس با استفاده از تابع آماده‌ی HoughLines خطوطی که حداقل ۲۵۰ رای آورده‌اند را ذخیره می‌کنیم. ورودی این تابع لبه‌های تصویر، θ و ρ و threshold هستند. حد آستانه همان پارامتر تعیین‌کننده‌ی تعداد نقاط رای‌دهنده است. سپس برای رسم خطوط از یک حلقه‌ی for استفاده کرده و ρ, θ را به فضای x, y برده و برای هر خط دو نقطه را تعیین کرده و آن خط را رسم می‌کنیم. در انتها تصویر با خطوط رسم‌شده را ذخیره می‌کنیم. (لینک)

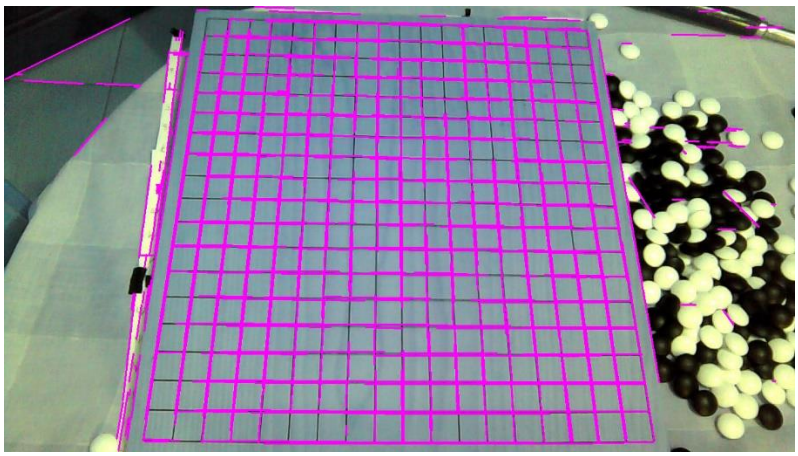
```
# https://www.geeksforgeeks.org/line-detection-python-opencv-houghline-method
img = cv2.imread("LineDetection.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 250)

for line in lines:
    x, th = line[0]
    x0 = x * np.cos(th)
    y0 = x * np.sin(th)
    p1 = (int(x0 + 1000  * (-np.sin(th))), int(y0 + 1000 * (np.cos(th))))
    p2 = (int(x0 - 1000 * (-np.sin(th))), int(y0 - 1000 * (np.cos(th))))
    cv2.line(img, p1, p2, (255, 0, 255), 1)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img, cmap="gray")
cv2.imwrite("q2_1.jpg",img)
```

خروجی:

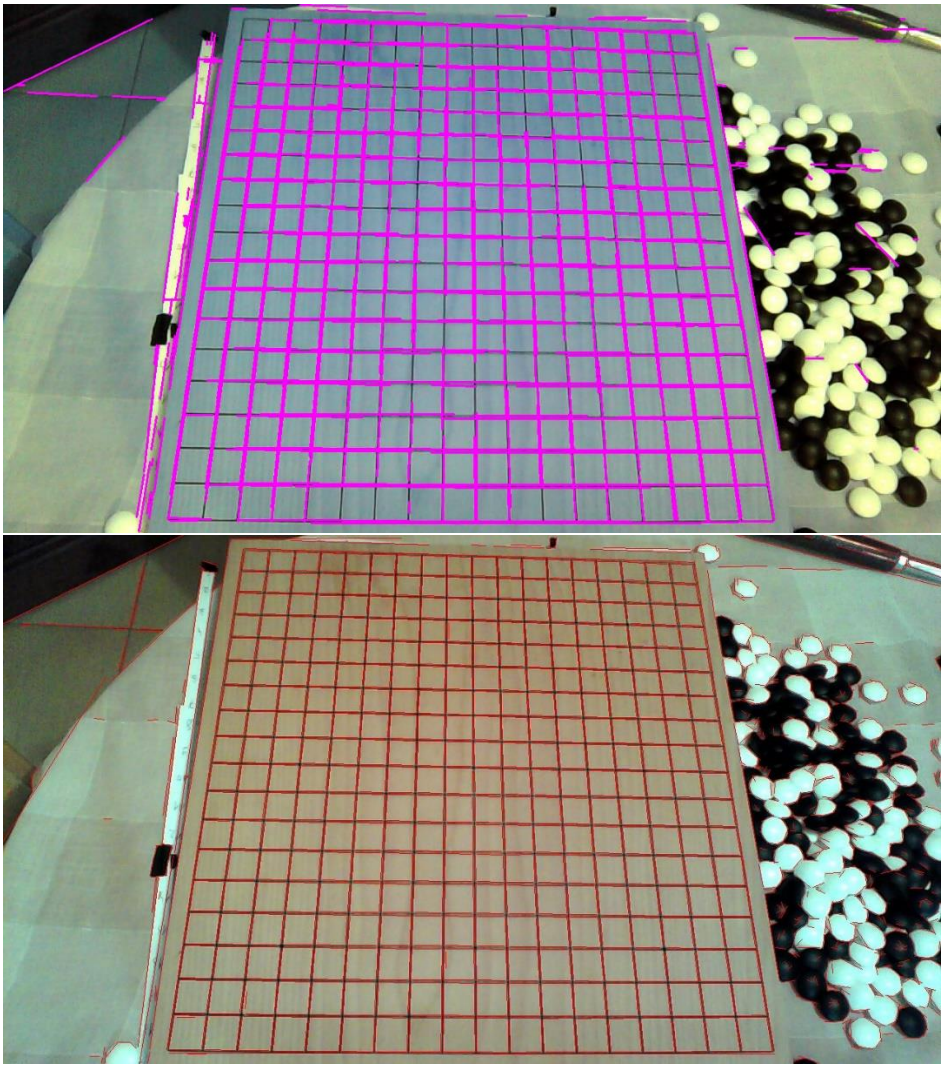


در بخش دوم، همان مراحل را طی کرده و این بار از تابع HoughLinesP استفاده کرده و نقاط ابتدا و انتهای خط را هم به ما می‌دهد. سپس برای هر خط نقاط ابتدا و انتهای خط را روی تصویر رسم می‌کنیم و تصویر خروجی را ذخیره می‌کنیم. پارامترهای ورودی این تابع علاوه بر پارامترهای قبلی دو پارامتر minLineLength و maxLineGap هم اضافه شده که اولی مینیمم طول مجاز خطوط یافت شده و دومی ماکسیمم فاصله‌ی مجاز بین خطوط است.

```
#TODO
img = cv2.imread("LineDetection.jpg")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)

lines = cv2.HoughLinesP(
            edges, # Input edge image
            1, # Distance resolution in pixels
            np.pi/180, # Angle resolution in radians
            threshold=100, # Min number of votes for valid line
            minLineLength=5, # Min allowed length of line
            maxLineGap=10 # Max allowed gap between line for joining them
            )

for points in lines:
    x1, y1, x2, y2 = points[0]
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 2)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img, cmap="gray")
```



۳. از الگوریتم LSD برای پیدا کردن پاره‌خط استفاده شده است. زیرا تبدیل هاف در بعضی جاها خوب عمل نمی‌کند و باید پیوستگی نقاط نیز لحاظ شود. لذا مزیت آن این است که با استفاده از ایده‌ی جهت گرادیان پاره‌خط‌های موجود در تصویر را پیدا می‌کند. در این کد ابتدا تصویر را به صورت grayscale خوانده و سپس با استفاده از تابع createLineSegmentDetector پاره-خط‌ها را تشخیص داده و با استفاده از تابع drawSegment پاره‌خط‌ها را در تصویر رسم کرده است.



همانطور که می‌بینیم دقت این الگوریتم از تبدیل هاف بالاتر است و بعضی از خطوطی را که تبدیل هاف موفق به تشخیص آن‌ها نشده را تشخیص داده است.

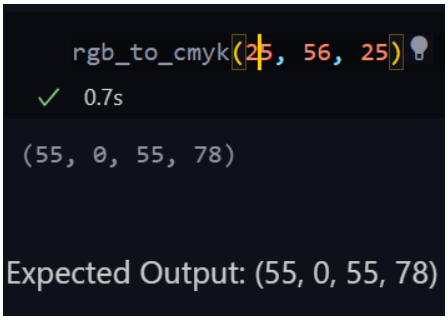
۴.

برای تبدیل rgb به cmyk از فرمول زیر استفاده می‌کنیم:

$$
\begin{aligned}
R' &= R/255 \\
G' &= G/255 \\
B' &= B/255 \\
\mathbf{K} &= 1-\max(R',G',B') \\
\mathbf{C} &= (1-R'-\mathbf{K})/(1-\mathbf{K}) \\
\mathbf{M} &= (1-G'-\mathbf{K})/(1-\mathbf{K}) \\
\mathbf{Y} &= (1-B'-\mathbf{K})/(1-\mathbf{K})
\end{aligned}
$$

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

  r = r / RGB_SCALE
  g = g / RGB_SCALE
  b = b / RGB_SCALE

  k = 1 - max(r, g, b)

  c = int(((1 - k - r) / (1 - k)) * CMYK_SCALE)
  m = int(((1 - k - g) / (1 - k)) * CMYK_SCALE)
  y = int(((1 - k - b) / (1 - k)) * CMYK_SCALE)
  k = int(k * CMYK_SCALE)


  return c, m, y, k
```

تطبیق خروجی تابع با مقدار مورد انتظار:



همچنین برای تبدیل cmyk به rgb از فرمول زیر استفاده می‌کنیم:

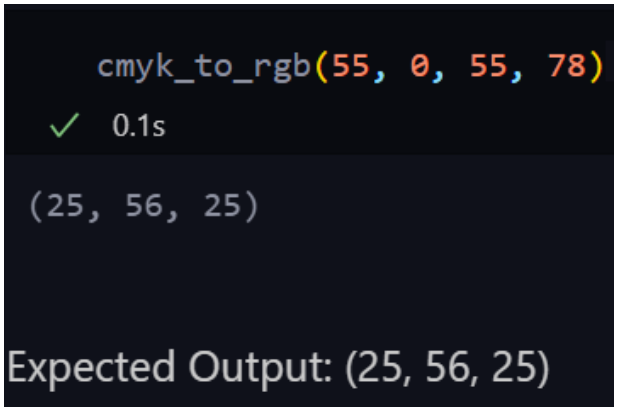
$$Red = 255 \times (1-C/100) \times (1-K/100)$$

$$Green = 255 \times (1-M/100) \times (1-K/100)$$

$$Blue = 255 \times (1-Y/100) \times (1-K/100)$$

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

  c = c / CMYK_SCALE
  m = m / CMYK_SCALE
  y = y / CMYK_SCALE
  k = k / CMYK_SCALE

  r = int(round(255 * (1 - c) * (1 - k), 2))
  g = int(round(255 * (1 - m) * (1 - k), 2))
  b = int(round(255 * (1 - y) * (1 - k), 2))

  return r, g, b
```

تطبیق خروجی تابع با مقدار مورد انتظار:



۵. با توجه به فرمول‌های اسلاید کلاسی داریم:

```
def HSI_V_L_Y(r, g, b):
    th = math.acos((r - b) + (r - g) / (2 * (math.sqrt(math.pow((r - g), 2) + (r
- b)*(g - b)))))
    th = math.degrees(th)
    if (b > g):
        H = 360 - th
    else:
        H = th

    S = 1 - 3 * (min(r, g, b) / (r + b + g))
    I = (r + b + g) / (3 * 255)

    V = max(r, g, b) / 255
    L = (max(r, g, b) + min(r, g, b)) / (2 * 255)

    Y = 0.299 * r + 0.587 * g + 0.114 * b

    return round(H, 2), round(S, 2), round(I, 2) , round(V, 2), round(L, 2),
round(Y, 2)
```