

به نام خدا

استاد: دکتر محمدرضا محمدی
درس مبانی بینایی کامپیوتر

نام: فاطمه زهرا بخشنده
شماره دانشجویی: 98522157

گزارش تمرین 6:

سوال اول:

اگر w نسبت تعداد نقاط Inlier به تمام نقاط باشد و p احتمال یافتن یک مجموعه از نقاط بدون Outlier باشد، باتوجه به آنکه برای تشخیص خط نیاز به 2 نقطه داریم پس احتمال آنکه یک مجموعه کاملاً از نقاط Inlier تشکیل شده باشد برابر است با w^2 پس اگر k تعداد تکرار باشد، احتمال آنکه هیچ مجموعه درستی انتخاب نشده باشد برابر است با:

$$1 - p = (1 - w^2)^k$$

اگر از طرفین لگاریتم بگیریم میتوان تعداد تکرار یعنی k را بدست آورد پس داریم:

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

تعداد کل نقاط را حساب می کنیم. 80 نقطه از ضلع قائم اول، 60 نقطه از ضلع قائم دوم، 120 نقطه از وتر و 100 نقطه نویزی دیگر پیدا شده است.

$$80 + 60 + 120 + 100 = 360$$

احتمال اینکه نقطه ای که انتخاب می کنیم مربوط به وتر مثلث باشد برابر است با:

$$w = \frac{120}{360} = \frac{1}{3}$$

احتمال اینکه هردو نقطه انتخابی از خط وتر باشند تقریباً برابر است با:

$$w^2 = \frac{1}{9}$$

1- اگر بخواهیم با احتمال 90% ضلع وتر را با الگوریتم Ransak بدست آوریم:

$$k = \frac{\log(1 - 0.9)}{\log(1 - \frac{1}{9})} = \frac{\log(0.1)}{\log(\frac{8}{9})} \approx 19.5 \approx 20$$

یعنی با حداقل 20 بار اجرا کردن الگوریتم می توانیم 90% احتمال دهیم که وتر را پیدا می کنیم.

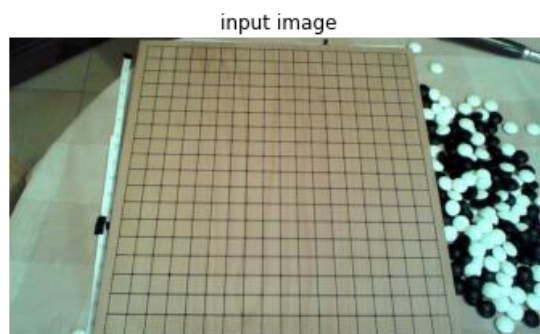
2- اگر بخواهیم با احتمال 99% ضلع وتر را با الگوریتم Ransak بدست آوریم:

$$k = \frac{\log(1 - 0.99)}{\log(1 - \frac{1}{9})} = \frac{\log(0.01)}{\log(\frac{8}{9})} \approx 39.1 \approx 40$$

یعنی با حداقل 40 بار اجرا کردن الگوریتم می توانیم 99% احتمال دهیم که وتر را پیدا می کنیم.
می بینیم که تعداد تکرار 2 برابر شد. اگر یک 9 دیگر نیز اضافه کنیم (99.9%)، این تعداد، 3 برابر می شود و به همین ترتیب.

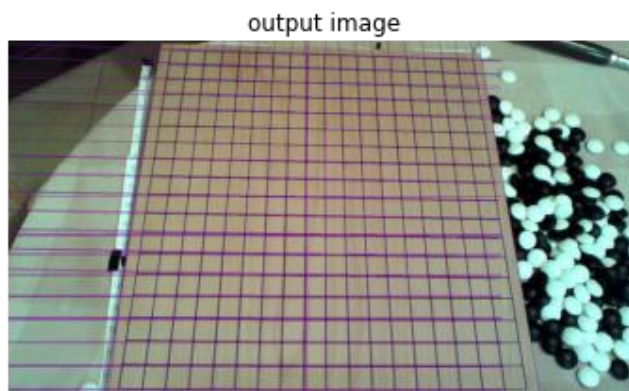
سوال دوم:

ابتدا تصویر را خوانده و نمایش می دهیم:



یک نسخه از تصویر را در مقیاس خاکستری در gray می ریزیم و روی آن لبه یاب Canny را اعمال می کنیم تا همه لبه های تصویر یافت شوند. خروجی را در edges نگه می داریم.

1- از تابع آماده HoughLines در OpenCV استفاده می کنیم که خروجی لبه یابی شده یعنی edges، Angle resolution بر حسب رادیان (1) و Distance resolution بر حسب پیکسل ($\frac{\pi}{180}$) و threshold که مقدار 250 دارد را به آن می دهیم. خروجی آن یک آرایه از r و theta هاست. برای کشیدن خط ها روی تصویر، روی این آرایه یک loop می زنیم. برای گرفتن x و y نقطه ابتدا و انتهای خط مورد نظر، از تابع polar to cartesian استفاده می کنیم که بالاتر آن را تعریف کردم. این تابع، ورودی rho و theta را می گیرد و x1 و y1 و x2 و y2 را خروجی می دهد. سپس با استفاده از cv2.line هر خط را روی تصویر می کشیم. نتیجه:

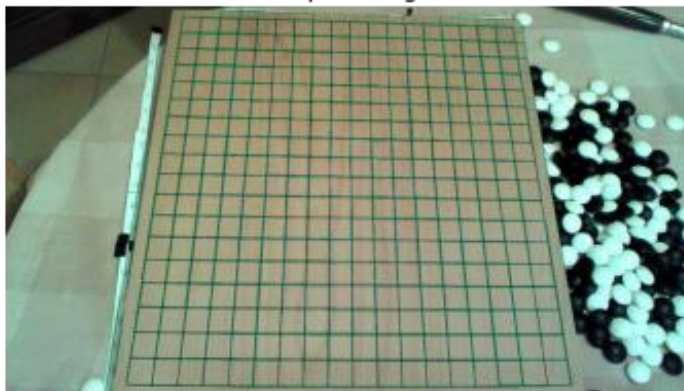


که مشابه نتیجه مورد انتظار است.

2- می خواهیم طول خطوط پیدا شده را نیز در پیدا شدن خطوط و همچنین بیشترین فاصله نقاط لبه از یکدیگر را نیز دخیل کنیم، پس از الگوریتم Probabilistic Hough Transform استفاده می کنیم. این

تابع دو پارامتر اضافه تر minLineLength (حداقل طول خط مجاز) و maxLineGap (بیشترین فاصله مجاز خطوط برای پیوستن به هم) را می گیرد. در استفاده از این تابع سعی می کنیم مقدار threshold ، maxLineGap و minLineLength را طوری انتخاب کنیم که نتیجه خوبی بگیریم. مقدار این پارامتر ها را به ترتیب 100 و 3 و 18 در نظر گرفتیم. نتیجه پس از کشیدن خط ها روی تصویر، به صورت زیر است:

output image



3- بخش امتیازی: برای این بخش، تابع hough_transform را پیاده سازی کردم که دقیقا مشابه تابع HoughLines در OpenCV ، edges و threshold را به عنوان ورودی گرفته، و rho و theta خط هایی که بیش از threshold رای آورده اند را بر می گرداند. الگوریتم پیاده سازی شده تقریبا مانند سودوکد موجود در اسلاید هاست با این تفاوت که در الگوریتم موجود در اسلاید فقط rho و theta مربوط به خطی که بیشترین رای را آورده را به عنوان خروجی می دهد. اما تابع hough_transform ، rho و theta همه خط هایی که بیشتر از threshold رای آورده اند را به عنوان خروجی بر می گرداند. خروجی این تابع با همان $\text{threshold} = 250$ به صورت زیر است:

output image

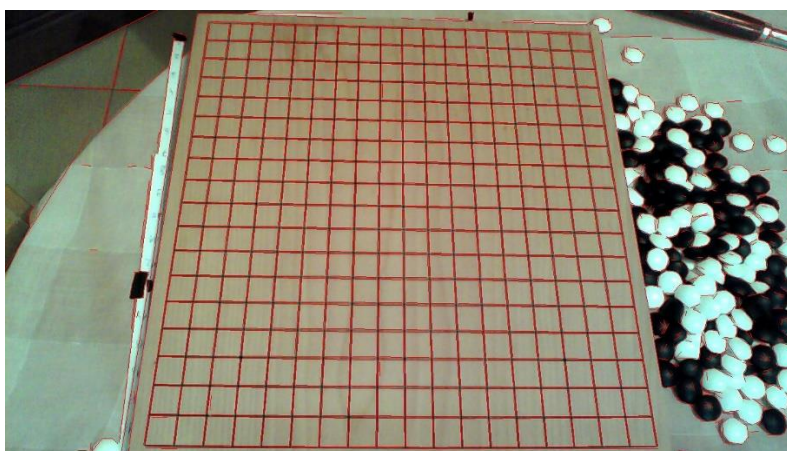


که به خوبی خط ها را پیدا کرده است. و مشابه خروجی تابع آماده HoughLines است.

منابع: اسلاید و [لینک](#)

سوال سوم:

خروجی این cell تصویر زیر است:



ابتدا تصویر LineDetection.jpg را در مقایس gray خوانده است. سپس یک object از کلاس LineSegmentDetector در OpenCV ساخته است. پاره خط ها را در تصویر gray scale با استفاده از فراخوانی متد detect از این object پیدا کرده است. سپس با فراخوانی متد drawSegments که تصویر اصلی (رنگی) و خط ها را به عنوان ورودی می گیرد، پاره خط های پیدا شده را روی تصویر رنگی کشیده است. در آخر نیز تصویر نهایی در UNKNOWN.jpg ذخیره شده است.

کلاس استفاده شده در واقع برای تشخیص پاره خط است. که آن را به عنوان LSD می شناسیم. الگوریتم تبدیل هاف در برخی مثال ها نمی تواند به خوبی عمل کند. دقت آن وابسته به موارد زیادی مانند مرحله لبه یابی است، لبه یابی هم حساس به نویز است که همین امر باعث ایجاد محدودیت هایی میشود. همچنین در الگوریتم Hough با هر بار اجرا کردن به یک نتیجه جدید می رسمیم که دلیل آن، داشتن ماهیت تصادفی است.

LSD معمولاً نتایج دقیقی تری از Hough ارائه میدهد چون از جهت گرادیان نیز استفاده می کند. همانطور که در خروجی LSD مشاهده می شود، خیلی تمیزتر و دقیقتر از Hough توانسته پاره خط ها را پیدا کند. هدف از الگوریتم LSD یافتن نقاط ابتدا و انتهای پاره خط های موجود در تصویر است. در واقع هر پاره خط بجای 2 پارامتر توسط 4 پارامتر مشخص میشود. مزیت اصلی الگوریتم LSD آن است که به خوبی از جهت گرادیان استفاده میکند.

در الگوریتم LSD تصویر ورودی از طیف خاکستری است، ولی در الگوریتم Hough عکس ورودی به صورت باینری است و همچنین الگوریتم LSD بدون تنظیم پارامتر های اضافه برای کارکردن طراحی شده است، ولی الگوریتم Hough دارای چندین پارامتر (فاصله رزولوشن، زاویه رزولوشن، threshold، حداقل طول خط مجاز و حداکثر گپ بین خطوط) است.

الگوریتم LSD بیشتر برای تشخیص پاره خط ها بدون پارامتر های اضافی مفید و کاربردی است و الگوریتم Hough برای تشخیص خط های خاص تر به دلیل داشتن پارامترهای مختلف، کاربردی تر است زیرا با تنظیم و ترکیب هر یک از این پارامترها باهم (به طور مثال حداکثر گپ خط و حداقل طول خط) می توان خط های خاص مورد نظر خود را بدست آورد.

پیچیدگی زمانی الگوریتم LSD خطی است اما پیچیدگی زمانی الگوریتم تبدیل Hough به پارامتر های آن بستگی دارد. Hough قادر به تعیین نقاط انتهایی یک بخش خط نیست و می تواند فقط خطوطی را که از کل تصویر عبور می کنند شناسایی کند. بنابراین برای اینکه hough بتواند نقاط انتهایی یک بخش را شناسایی کند ، باید روش های تقسیم بندی نیز اعمال شود. که همین امر نیز سربار زمانی دارد.

منابع: [لینک](#) و [لینک](#) و [لینک](#)

سوال چهارم:

مدل رنگ CMYK: در اغلب پرینترها از 4 جوهر با رنگ های فیروزه ای، بنفش روشن، زرد و سیاه استفاده میشود. علت استفاده از جوهر سیاه آن است که چاپ کردن رنگ سیاه با استفاده از 3 جوهر هزینه بر است.

از فرمول های موجود در اسلاید برای تبدیل RGB به CMYK و بالعکس استفاده می کنیم. در تبدیل مدل رنگ ها باید به scale هرکدام توجه کنیم، مثلاً برای تبدیل RGB به CMYK، قبل از اعمال فرمول، باید ابتدا R و G و B تقسیم بر مقیاس خود (255) شوند و هرکدام از C و M و Y و K هم پس از تبدیل باید ضربدر مقیاس خود (100) شوند. همچنین در تابع اول باید حالت 0 و 0 و 0 مشکی را جداگانه در نظر بگیریم تا مخرج ما صفر نشود.

تابع اول:

```
rgb_to_cmyk(25, 56, 25) => (55, 0, 55, 78)
```

تابع دوم:

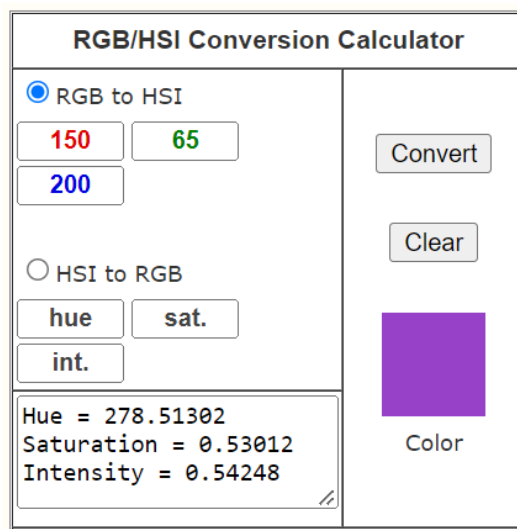
```
cmyk_to_rgb(55, 0, 55, 78) => (25, 56, 25)
```

منابع: [لینک](#)

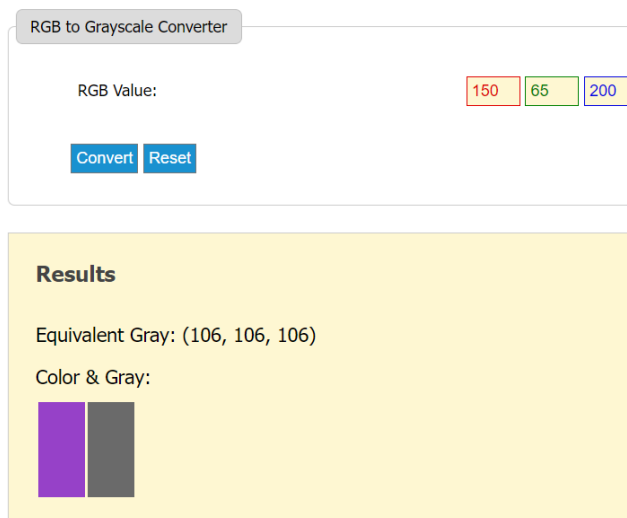
سوال پنجم:

این پارامتر ها را با توجه به فرمول های موجود در اسلاید ها از روی R و G و B بدست می آوریم. در این سوال نیز باید حواسمان به scale پارامتر ها باشد.

چند لینک تبدیل آنلاین مدل های رنگی که در منبع نوشتیم را بررسی کردم. در این تبدیل ها مقایس S و I و V و L از 1 است. من هم این پارامتر ها را با همین مقیاس حساب کردم. (می توان به صورت درصد هم بیان کرد). همچنین بعضی جاها پارامتر ها را به صورت اعشاری با تعداد رقم های مختلفی نشان داده و برخی جاها به صورت صحیح round کرده بود. برای مثال به این صورت:



من همه پارامتر ها را با 4 رقم اعشار round کردم. برای چک کردن تبدیل مقیاس خاکستری نیز از تبدیل این سایت استفاده کردم:



کد محاسبه پارامتر ها:

```
R, G, B = 150, 65, 200

if (R, G, B) == (0, 0, 0):
    H, S, I, V, L, Y = 0, 0, 0, 0, 0, 0
else:
    RGB_SCALE = 255

    # H
    num = (R-G) + (R-B)
    denom = 2 * (((R-G)**2 + (R-B) * (G-B))**0.5)

    theta = 180 * np.arccos(num / denom) / np.pi

    # If B>G then H = 360-Theta
    H = round(360 - theta if B > G else theta, 4)

    # S
    S = round(1 - (3 * min(R, G, B)) / (R + G + B), 4)

    # I
    I = round((R + G + B) / (3 * RGB_SCALE), 4)

    # V
    V = round(max(R, G, B) / RGB_SCALE, 4)

    # L
    L = round((max(R, G, B) + min(R, G, B)) / (2 * RGB_SCALE), 4)

    # Y
    Y = round(0.299 * R + 0.587 * G + 0.114 * B)
```

نتیجه تبدیل ها به صورت زیر است:

RGB: (150, 65, 200)

H = 278.513

S = 0.5301

I = 0.5425

HSI: (278.513, 0.5301, 0.5425)

V = 0.7843

L = 0.5196

Y = 106

Gray: (106, 106, 106)

منابع: [لینک](#) و [لینک](#) و [لینک](#)