دانشگاه علم و صنعت

تمرین سوم مبانی بینایی کامپیوتر

نام و نام خانوادگی:

فرناز خوش دوست آزاد

شماره دانشجویی:

99521253

نام استاد:

دکتر محمدرضا محمدی

(1

به سوالات زیر پاسخ دهید.(15 نمره)

a.برای تصویری مانند(I(x,y، بردار گرادیان ∇I(x, y) را محاسبه کنید.(نوشتن روابط کافی است)

پاسخ:

بردار گرادیان برای تابع $I(x,y)$ نشان دهنده جهت و شدت تغییرات تابع در هر نقطه است. بردار گرادیان به صورت زیر محاسبه می‌شود.

$$\nabla I = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

- G_x است (x) نشان‌دهنده تغییر شدت تصویر نسبت به جهت افقی.
- G_y است (y) نشان‌دهنده تغییر شدت تصویر نسبت به جهت عمودی.

برای حساب کردن تقریبی Gx در عمل، معمولاً از فیلترهایی مانند Sobel یا Prewitt استفاده می کنیم که به صورت زیر هستند:

- کرنل Sobel برای G_x:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I(x,y)$$

- کرنل Sobel برای G_y:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I(x,y)$$

که در اینجا، عملگر $*$ نمایانگر عمل کانولوشن (پیچش) است.

برای مثال، اگر تابع $I(x,y) = y^2x^2$ باشد، محاسبه‌ی بردار گرادیان به شکل زیر است:

$$\frac{\partial I}{\partial x} = \frac{\partial}{\partial x}(x^2 + y^2) = 2x$$
$$\frac{\partial I}{\partial y} = \frac{\partial}{\partial y}(x^2 + y^2) = 2y$$

پس بردار گرادیان برای این تابع خواهد بود:

$$\nabla I(x,y) = (2x, 2y)$$

پس از محاسبه Gx و Gy، می‌توان اندازه و جهت بردار گرادیان را به صورت زیر محاسبه کرد:

- اندازه بردار گرادیان: $|\nabla I| = \sqrt{G_x^2 + G_y^2}$
- جهت بردار گرادیان: $\theta = \arctan\left(\frac{G_y}{G_x}\right)$

این اطلاعات به ما در تشخیص لبه‌ها، نقاط برجسته، و ویژگی‌های دیگر در تصاویر کمک می‌کند.

b.چرامحاسبه این بردار مفید می باشد؟

پاسخ:

محاسبه بردار گرادیان و استخراج ویژگی‌های مبتنی بر گرادیان از تصاویر در بسیاری از کاربردهای پردازش تصویر و بینایی ماشین اهمیت زیادی دارد. در اینجا به چند دلیل که نشان می‌دهد چرا این محاسبات برای ما مفید هستند، اشاره می‌کنیم:

1. تشخیص لبه‌ها که در آنها تغییر ناگهانی در شدت (روشنایی) رخ می‌دهد. این تغییرات معمولاً نشان‌دهنده تغییرات ساختاری در صحنه، مانند مرزهای اشیاء، نواحی با بافت متفاوت، یا سایه‌ها هستند که برای ویژگی‌یابی اشیاء، تجزیه و تحلیل ساختاری، و تعقیب حرکت کاربرد دارد.
2. تقویت ویژگی‌ها در تصاویر به ویژه در مواردی که نیاز به شناسایی و تمایز نقاط کلیدی یا الگوهای خاص در تصویر است. این ویژگی‌ها می‌توانند در شناسایی و دسته‌بندی اشیاء، تشخیص چهره، و بینایی رایانه‌ای استفاده شوند.

3. بهبود ناوبری و درک محیط در رباتیک و وسایل نقلیه خودران، دانستن جایگاه لبه‌ها و تفاوت‌های ساختاری در محیط می‌تواند به بهبود الگوریتم‌های ناوبری و تصمیم‌گیری کمک کند. تشخیص لبه می‌تواند در تشخیص موانع، تعیین مسیر، و تخمین عمق مفید باشد.

4. پردازش تصویر و بهبود کیفیت از طریق شارپ‌کردن، حذف نویز، و کنتراست بخشی کمک کند. این تکنیک‌ها می‌توانند در بهبود تصاویر پزشکی، تصاویر ماهواره‌ای، و عکاسی دیجیتال مفید باشند.

5. تحلیل بافت و شناسایی الگو که می‌توان از آنها برای تفکیک و دسته‌بندی انواع مختلف بافت‌ها در تصاویر استفاده کرد.

6. ساخت نقشه‌های ویژگی برای یادگیری عمیق که از طریق کانوولوشن‌های متعدد و لایه‌های استخراج ویژگی به دست می‌آیند، اغلب حاوی اطلاعات گرادیان هستند. این اطلاعات می‌توانند به شبکه‌های عصبی کمک کنند تا ویژگی‌های مهم را برای تشخیص، دسته‌بندی، و سایر وظایف پردازش تصویر یاد بگیرند.

C.اندازه گرادیان تعریف شده روی صفحه تصویر (x,y) محاسبه کنید.(نوشتن روابط کافی است)

پاسخ:

- اندازه بردار گرادیان: $|\nabla I| = \sqrt{G_x^2 + G_y^2}$

لذا می توان گفت از فرمول زیر برای اندازه ی گرادیان هر تصویر استفاده می شود.

$$|\nabla I(x,y)| = \sqrt{\left(\frac{I(x+1,y) - I(x-1,y)}{2}\right)^2 + \left(\frac{I(x,y+1) - I(x,y-1)}{2}\right)^2}$$

D.جهت گرادیان تعریف شده روی صفحه تصویر(x,y) را محاسبه کنید.(نوشتن روابط کافی است)

پاسخ:

- جهت بردار گرادیان: $\theta = \arctan\left(\frac{G_y}{G_x}\right)$

که می توان در آن Gx و Gy را جایگذاری کرد.

E.نحوه استفاده از بردار گرادیان را در آشکار ساز لبه Canny توضیح دهید. (مراحل اصلی آن و مزایای آن نسبت به رویکردهای جایگزین را بنویسید).

پاسخ:

و مراحل شامل

1- در ابتدا باید تصویر رنگی را به جهت لبه یابی بهتر به یک تصویر سطح خاکستری تبدیل کرد.

2- نویز را از تصویر دریافتی حذف کرد. بدلیل اینکه فیلتر گوسین از یک ماسک ساده برای حذف نویز استفاده می کند لبه یاب کنی در مرحله اول برای حذف نویز آن را بکار میگیرد.

3- در یک تصویر سطح خاکستر جایی را که بیشترین تغییرات را داشته باشند به عنوان لبه در نظر گرفته می شوند و این مکانها با گرفتن گرادیان تصویر با استفاده عملگر سوبل بدست می آیند. سپس لبه های مات یافت شده به لبه های تیزتر تبدیل می شوند.

4- برخی از لبه های کشف شده واقعا لبه نیستند و در واقع نویز هستند که باید آنها توسط حد آستانه هیسترزیس فیلتر شوند.هیسترزیس از دو حد آستانه بالاتر (Th) و حد آستانه پایین تر (Tl) استفاده کرده و کنی پیشنهاد می کند که نسبت آستانه بالا به پایین سه به یک باشد.

این روش بیشتر به کشف لبه های ضعیف به درستی می پردازد و کمتر فریب نویز را می خورد و از بقیه روش ها بهتر است. تضعیف نویز، پیدا کردن نقاطی که بتوان آنها را به عنوان لبه در نظر گرفت و جذب نقاطی که احتمال لبه بودن آنها کم است از دیگر خوبی های این الگوریتم می باشد.

F.از عملگر لاپلاسین میتوان برای تشخیص لبه استفاده کرد. اما غالبا در عمل برای تشخیص لبه از همان عملگرهای sobel و canny استفاده میشود. چرا عملگر لاپلاسین عملگر خوبی برای تشخیص لبه نیست؟ 3 دلیل را ذکر کنید.

پاسخ:

1. حساسیت بالا به نویز

عملگر لاپلاسین به نویز بسیار حساس است و حتی تغییرات کوچک ناشی از نویز نیز می‌توانند به عنوان لبه تشخیص داده شوند. این امر می‌تواند منجر به نتایج نامطلوب و تشخیص‌های غلط بسیاری شود، مخصوصاً در تصاویر با کیفیت پایین یا تصاویری که دارای میزان زیادی نویز هستند.

2. عدم توانایی در تشخیص جهت لبه

در حالی که عملگرهایی مانند Sobel و Canny جهت لبه‌ها را تشخیص می‌دهند (که می‌تواند برای برخی کاربردها مفید باشد)، عملگر لاپلاسین یک عملگر بدون جهت است. این بدان معناست که لاپلاسین

فقط حضور یا عدم حضور یک لبه را تشخیص می‌دهد و نمی‌تواند اطلاعاتی در مورد جهت آن فراهم کند. این محدودیت می‌تواند در تحلیل‌های پیچیده‌تر تصویر محدودکننده باشد.

3. تشخیص لبه‌های دوگانه
عملگر لاپلاسین تمایل دارد لبه‌های "دوگانه" تولید کند، به این معنا که در جایی که یک لبه واقعی وجود دارد، دو پاسخ مجاور تولید می‌شود. این امر می‌تواند در تصاویری که لبه‌های دقیق مهم هستند، سبب سردرگمی شود. عملگرهایی مانند Sobel و Canny با استفاده از روش‌هایی برای نازک‌سازی و پیگیری لبه، این مشکل را بهتر مدیریت می‌کنند.

به همین دلایل، عملگرهای Sobel و Canny غالباً ترجیح داده می‌شوند، زیرا آن‌ها با فراهم کردن مکانیزم‌های بهتر برای کاهش نویز، تعیین جهت لبه، و تولید خطوط لبه دقیق‌تر، عملکرد بهتری در بسیاری از کاربردهای واقعی ارائه می‌دهند.

(2

نوتبوک Q2.ipnyb را با توجه به سوالات زیر تکمیل کنید.

a.تصویر jpg.1 و jpg.2 را از پوشه ی Q2 بخوانید. تبدیل فوریه هر تصویر را محاسبه کنید و دامنه و فاز هر تصویر را نمایش دهید. (5 نمره)

پاسخ:

پس از خواندن تصاویر، می توانیم از کتابخانه های fft برای تبدیل فوریه استفاده کنیم. در اولین مرحله برای استفاده از تبدیل فوریه بهتر است از تصویر رنگی را به تصویر خاکستری تبدیل کنیم زیرا برای تصاویر رنگی باید تبدیل فوریه را برای هر کانال رنگی به طور جداگانه حساب کرد که این کار منجر به ایجاد سه طیف دامنه ی جداگانه می شود و این ممکن است باعث شود دامنه به خوبی نمایش داده نشود. زیرا اساسا بزرگی سه کانال رنگی را نشان می دهد. لذا منطقی است که از تصویر خاکستری استفاده کنیم.

تجزیه و تحلیل دامنه و پاسخ های فازی تصویر در مقیاس خاکستری به درک محتوای فرکانس و تغییرات فضایی در تصویر کمک می‌کند.

در زمینه تحلیل فوریه و تجسم طیف دامنه، "نقاط نور" به طور معمول فرکانس‌های پایین را نشان می‌دهند، در حالی که "مناطق تاریک" با فرکانس‌های بالا مطابقت دارند.

فرکانس های پایین:

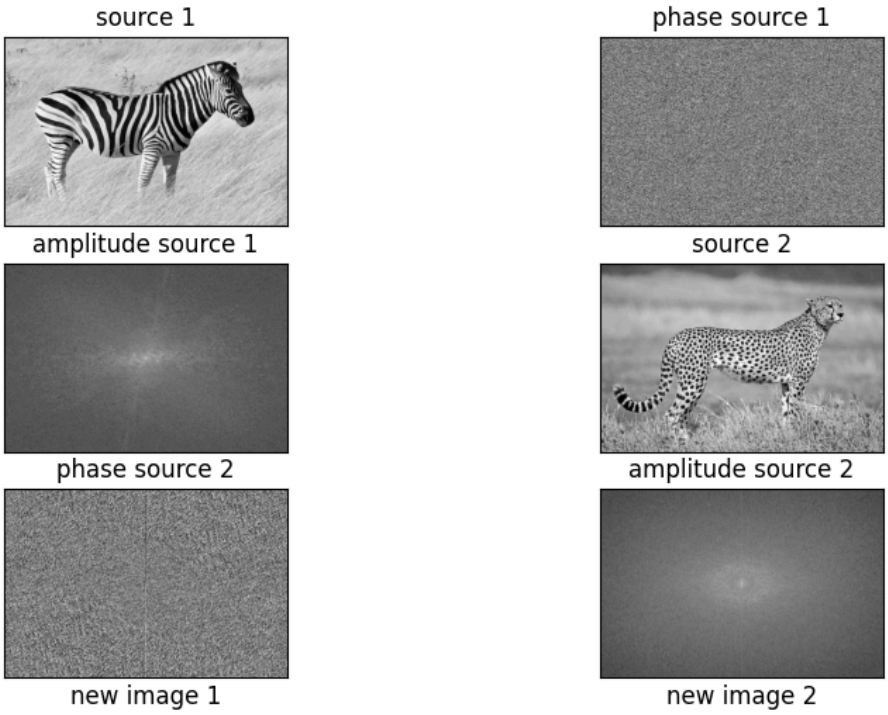
اجزای فرکانس پایین در یک تصویر با الگوها یا ساختارهای صاف و به آرامی متغیر مطابقت دارند. این مؤلفه‌های فرکانس پایین، ویژگی‌های کلی تصویر، مانند روشنایی کلی، گرادیان‌های در مقیاس بزرگ، و خطوط گسترده را ثبت می‌کنند. در حوزه فوریه، فرکانس های پایین با ضرایب نزدیک به مرکز طیف نمایش داده می شوند.[3]

فرکانس های بالا:

اجزای فرکانس بالا در یک تصویر نشان دهنده نوسانات یا تغییرات سریع با جزئیات دقیق هستند. این اجزای فرکانس بالا ویژگی‌های محلی، لبه‌های تیز، جزئیات بافت و الگوهای ظریف را ثبت می‌کنند. در حوزه فوریه، فرکانس‌های بالا با ضرایب دورتر از مرکز طیف نشان داده می‌شوند.

بنابراین با توجه به توضیحات بالا برای تصویر 1، یعنی گورخر است، بنظر می‌رسد که تغییرات شدت روشنایی افقی بصورت smooth یا غیر شدید هستند، و در بقیه ی نقاط به نسبت حالت افقی در میانه تصویر تغییرات شدت بیشتری دارند.

و در مورد تصویر 2، یعنی پلنگ، به نظر می‌رسد که تغییرات شدت ابتدا کم اما دوباره زیاد می‌شود، و دوباره کم، به نوعی انگار تغییرات روی خود پلنگ کم و در مرز پلنگ و اطراف زیاد و دوباره کم می‌شود.



b.جای فاز و دامنه ی دو تصویر را عوض کنید و تبدیل فوریه معکوس بگیرید. سپس تصاویر حاصل را نمایش دهید. از این آزمایش چه نتیجه ای می گیرید؟ (5 نمره)

پاسخ:

تغییر فاز دو تصویر در حالی که دامنه آنها را بدون تغییر نگه داریم، منجر به اثراتی در پردازش تصویر می‌شود. که در ادامه به آن ها اشاره کرده ایم:

انتقال بافت:

با جابجایی اطلاعات فاز بین دو تصویر، امکان انتقال ویژگی های بافت یک تصویر به تصویر دیگر وجود دارد. این می‌تواند در برنامه هایی مانند انتقال سبک، که در آن بافت یک تصویر به محتوای تصویر دیگر اعمال می‌شود، مفید باشد.

فیوژن تصویر:

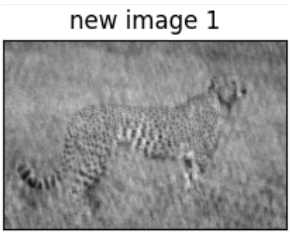
تعویض فاز بین تصاویر می‌تواند برای تکنیک های ترکیب تصویر استفاده شود. با ترکیب دامنه یک تصویر با فاز تصویر دیگر، می‌توانید تصویر جدیدی ایجاد کنید که جزئیات ساختاری هر دو تصویر ورودی را حفظ کند.

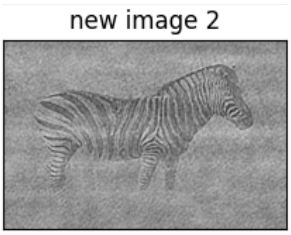
جلوه های بصری:

تغییر فاز تصاویر می تواند منجر به جلوه های هنری یا اعوجاج بصری شود. به عنوان مثال، تعویض فاز تصاویر طبیعی با نویز تصادفی می تواند تصاویر انتزاعی یا سورئالیستی ایجاد کند.

استحکام در برابر فشرده سازی:

در برخی موارد، اصلاح فاز یک تصویر با حفظ دامنه آن می‌تواند تصویر را در تکنیک های فشرده سازی قوی تر کند. این به این دلیل است که ادراک انسان نسبت به تغییرات در ساختار تصویر (قدرت) حساس تر از تغییرات فاز تصویر است.





نتیجه گیری:

از این آزمایش می‌توان نتیجه گرفت که فاز تصویر نقش مهمی در تعیین ویژگی‌های ساختاری و مکانی اجزای تصویر دارد. در حالی که دامنه اطلاعاتی در مورد شدت فرکانس‌های موجود در تصویر می‌دهد، فاز اطلاعاتی در مورد چگونگی ترکیب این فرکانس‌ها برای تشکیل تصویر نهایی را فراهم می‌کند. تصاویر با فاز جابجا شده نشان دهنده این موضوع هستند که با از دست دادن فاز اصلی، ویژگی‌های اصلی تصویر ممکن است به طور قابل توجهی تغییر کند.

(3

مراحل زیر را در نوتبوک Q3.ipynb پیش ببرید.

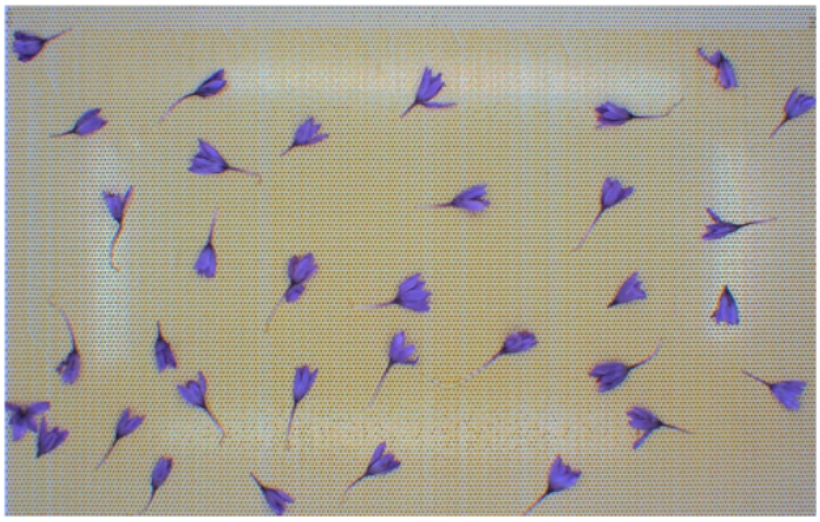
a.ابتدا تصویر saffron.jpg را بخوانید و نویز تصویر را با تبدیل FFT حذف کنید. در حذف نویز از تمام مراحل خروجی گرفته و رسم کنید و علت کار خود را نیز توضیح دهید. (15 نمره)

راهنمایی: همانطور که در تصویر مشاهده می شود خط تولید دارای یک سری روزنه هایی است که به صورت متناوب در پس زمینه قرار دارند. سعی کنید آنها را به عنوان نویز تشخیص داده و حذف کنید.

پاسخ:

در ابتدا تصویر را می خوانیم که در آن خواهیم داشت:

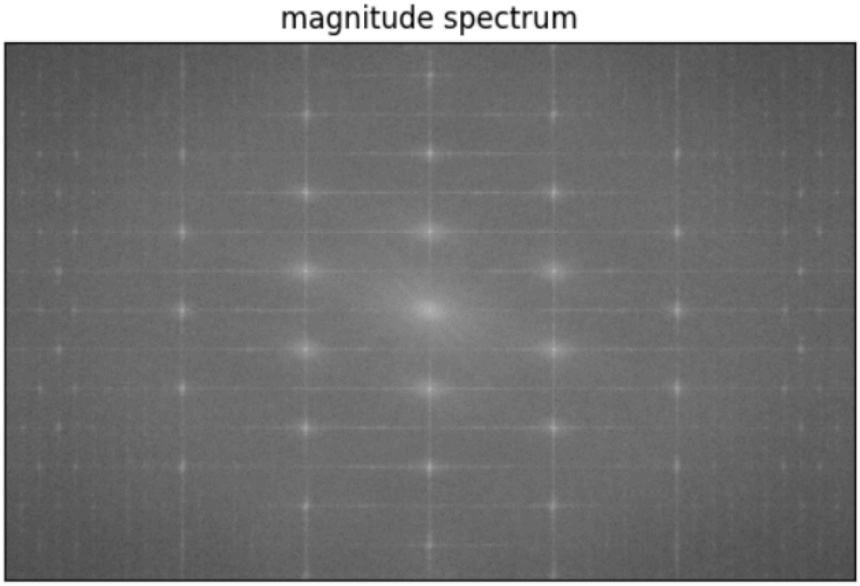
```python
image = cv2.imread('saffrun.jpg')
gray = cv2.imread('saffrun.jpg', cv2.IMREAD_GRAYSCALE)
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(rgb)
plt.axis('off')  # Hide axis
plt.show()
```



برای حذف نویز با تبدیل فوریه (FFT) مراحل زیر باید طی شوند:

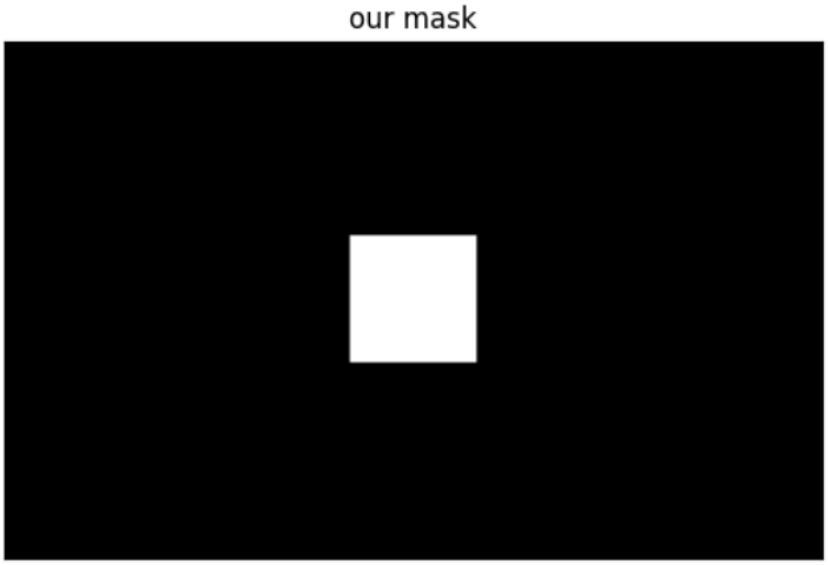
- اعمال FFT، تبدیل فوریه:

FFT تصویر ورودی را محاسبه می‌کنیم تا دامنه فرکانس و فاز آن را بدست آوریم.



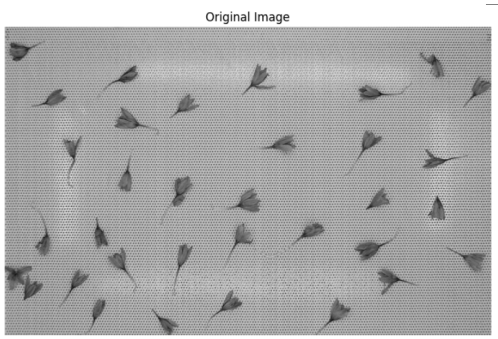
- فیلتر کردن:

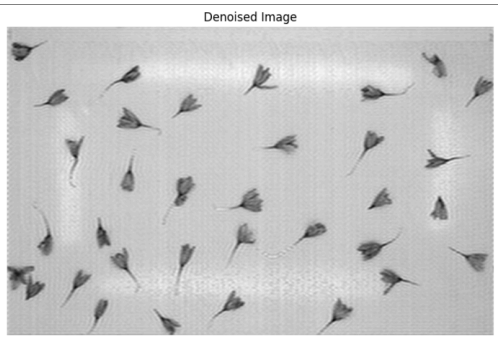
یک threshold در نظر می‌گیریم و به کمک آن فرکانس های بالا (که احتمال نویز بودن آنها زیاد است) را حذف می‌کنیم در حالی که اجزای فرکانس پایین حفظ می‌شو. با انتخاب threshold نامناسب باعث کاهش کیفیت تصویر نیز ممکن است بشود، بنابراین اگر عدد خیلی کوچکی انتخاب شود بخش های زیادی از خود تصویر را هم از دست می‌دهیم، و اگر عدد خیلی بزرگی انتخاب شود، نویز ها را کاور نمی‌کند که 100 مناسب است.



- FFT معکوس:

برای به دست آوردن تصویری که نویز آن حذف شده یک FFT معکوس انجام می‌دهیم. (نکته: اعمال تبدیل فوریه روی یک تصویر و سپس اعمال تبدیل فوریه رویش، به ما تصویر اصلی را خواهد داد.)

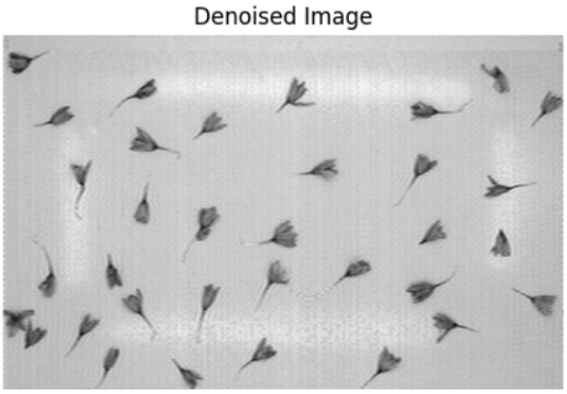


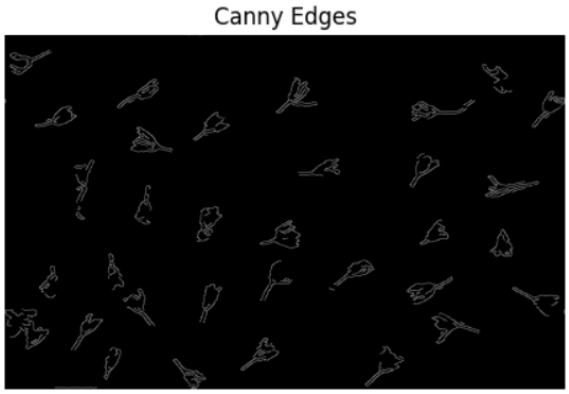


b.لبه یاب canny را بر روی خروجی مرحله a اجرا کنید. برای لبه یاب از توابع آماده استفاده کنید. تمام پارامترهای تابع که مقداردهی می شوند با ذکر دلیل توضیح داده شوند. مطلوب است در خروجی این مرحله فقط گلهای زعفران بمانند. (10نمره)

پاسخ:

در لبه یاب Canny 2 آستانه داریم، که در اینجا نیز ورودی های تابع ما هستند.
آستانه 1 یا ورودی اول، مقدار "آستانه پایین" را تعریف می‌کند. این اولین آستانه مورد استفاده در مرحله "آستانه گذاری دو مرحله‌ای" الگوریتم تشخیص لبه Canny است. هر پیکسل دارای اندازه گرادیان شدت روشنایی است، پیکسلی که اندازه بالاتر از آستانه 1 داشته باشد، به عنوان پیکسل لبه قوی در نظر گرفته می‌شود.
و آستانه 2 پارامتر مقدار "آستانه بالا" را در مرحله "آستانه گذاری دو مرحله‌ای" مشخص می‌کند. این دومین آستانه مورد استفاده در این مرحله از الگوریتم تشخیص لبه Canny است. هر پیکسلی که اندازه گرادیان شدت روشنایی آن کمتر از آستانه 2 باشد، حذف می‌شود، مگر اینکه به یک پیکسل لبه قوی متصل باشد.
آشکارساز لبه Canny بدین صورت کار می‌کند که ابتدا فیلتر گاوسی را روی تصویر ورودی برای " هموار کردن تصویر " اعمال می‌کند و نویز ها را تا حد امکان حذف می‌کند، سپس از "محاسبه گرادیان" روی تصویر برای یافتن اندازه و جهت لبه استفاده می‌کند. سپس، برای نازک کردن لبه‌ها، "حذف مقادیر غیربیشینه" که به آن "مهار غیر حداکثری" هم می‌گویند را اعمال می‌کند و در نهایت پیوند لبه‌ها را با "آستانه گذاری دو مرحله‌ای" انجام می‌دهد.

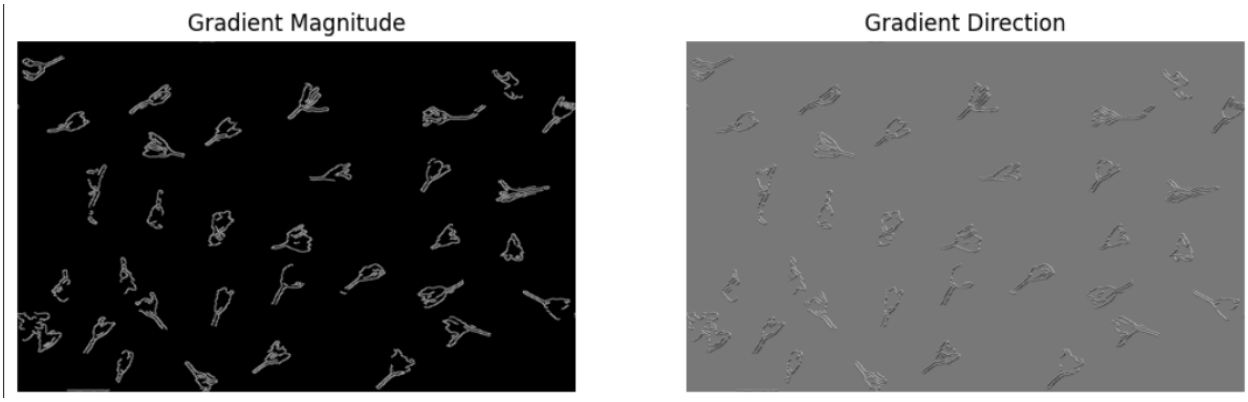




c.از تصویر بدست آمده(مرحله قبل b)گرادیان بگیرید و با استفاده از تابع arctan2 جهت گرادیان های بدست آمده را محاسبه کنید. (5 نمره)

پاسخ:

ابتدا برای محاسبه گرادیان تصویر در جهت x و y از یک فیلتر گرادیان، مانند Sobel استفاده می‌کنیم، سپس مقدار و جهت شیب ها را به ترتیب با استفاده از توابع قدر مطلق و arctan2 محاسبه می‌کنیم.



d.امتیازی: با استفاده از جهت گرادیان های بدست آمده، راه حلی برای بدست آوردن نقطه برش ساقه از گلبرگ ارائه دهید. (5 نمره)

پاسخ:

(4

به سوالات زیر در رابطه با تبدیل فوریه پاسخ دهید.

a.سه مثال از روشها یا ابزارهای مورد استفاده در Computer Vision را ارائه دهید که در آنها تحلیل فوریه نقش مهمی را ایفا میکند، یا برای حل یک مسئله، یا برای کارآمدتر کردن محاسبات به کار میرود. برای هر یک از مثال های خود، علت و فایده آن را توضیح دهید. (5 نمره)

پاسخ:

تحلیل فوریه یکی از ابزارهای مهم در پردازش تصویر و بینایی ماشین است که از طریق تبدیل فوریه، سیگنال‌ها یا تصاویر را از دامنه زمانی یا فضایی به دامنه فرکانسی تبدیل می‌کند. این تحول امکان بررسی ویژگی‌های فرکانسی تصاویر را فراهم می‌آورد و در موارد زیر کاربرد فراوان دارد:

1. فیلتر کردن تصاویر

تحلیل فوریه در فیلتر کردن تصاویر برای حذف نویز یا بهبود ویژگی‌های خاصی مانند لبه‌ها نقش مهمی دارد. برای مثال، در حذف نویز گوسی از تصویر، می‌توان تصویر را به فضای فرکانسی منتقل کرد و سپس فرکانس‌های بالا که معمولاً حاوی اطلاعات نویز هستند، کاهش داده شوند. این کار باعث می‌شود که تصویری صاف‌تر و بدون نویز در فضای واقعی بدست آید.

2. تشخیص الگو

در تشخیص الگوها و تجزیه و تحلیل محتوای تصاویر، استفاده از تبدیل فوریه می‌تواند بسیار مفید باشد. با تجزیه تصویر به مولفه‌های فرکانسی مختلف، می‌توان ویژگی‌هایی که در فرکانس‌های خاصی بیان می‌شوند را شناسایی کرد. برای مثال، تحلیل فرکانسی می‌تواند در تشخیص الگوهای بازگشتی مانند بافت‌ها یا شبکه‌های عصبی کاربرد داشته باشد که به دنبال تشخیص ویژگی‌هایی در سطوح مختلف دقت هستند.

3. ترمیم تصویر

در ترمیم تصاویر (Image Restoration) که شامل حذف تاری یا خطاهای دیگر از تصویر است، تحلیل فوریه کمک شایانی می‌کند. مثلاً، اگر تصویری دچار تاری شده باشد به دلیل حرکت دوربین یا موضوع، با استفاده از مدل‌های فرکانسی می‌توان اثر این تاری را به صورت یک فیلتر در فضای فرکانسی مدل کرد و سپس این فیلتر را برعکس نمود تا شفافیت تصویر بازیابی شود.

این روش‌ها و ابزارها نمونه‌هایی از کاربردهای تبدیل فوریه در بینایی ماشین هستند که نه تنها به حل مشکلات کمک می‌کنند بلکه باعث افزایش کارآمدی و دقت سیستم‌های مربوطه می‌شوند. استفاده از تحلیل فوریه به ویژه در مواقعی که نیاز به بررسی جامع و عمیق‌تر فرکانس‌های موجود در تصویر است، ایده‌آل می‌باشد.

b.اگر F(u,v) تبدیل فوریه تصویر f(x,y) باشد، حاصل F(0,0) را به دست آورید. (روابط ریاضی محاسبه را بنویسید) (5 نمره)

پاسخ:

در تبدیل فوریه دو بعدی، F(u, v) که تبدیل فوریه تصویر f(x, y) است، به صورت زیر تعریف می‌شود:

$$f(x,y)e^{i2\pi(ux+vy)-}\,dx\,dy\int_{\infty-}^{\infty}\int_{\infty-}^{\infty}=F(u,v)$$

مقدار F(0, 0)، که به آن مؤلفه DC یا متوسط تصویر گفته می‌شود، به این صورت محاسبه می‌شود:

$$f(x,y)\,dx\,dy\int_{\infty-}^{\infty}\int_{\infty-}^{\infty}=F(0,0)$$

توجه داشته باشید که در معادله برای (F(0, 0)، عبارت(e^{-i 2\pi (0x + 0y) به 1 تبدیل می‌شود و در نتیجه از معادله حذف می‌گردد.

مؤلفه (F(0, 0) مجموع تمام مقادیر پیکسل‌ها (یا متوسط تصویر ضربدر تعداد کل پیکسل‌ها) را نشان می‌دهد. این مقدار نشان‌دهنده میزان روشنایی کلی تصویر است. در تصاویر دیجیتال، این محاسبه به صورت مجموع کل پیکسل‌های تصویر در نظر گرفته می‌شود:

$$f(x,y)\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} = F(0,0)$$

که (M) و (N) به ترتیب تعداد ردیف‌ها و ستون‌های تصویر هستند. این مجموع، که برابر با مجموع تمام مقادیر پیکسل‌ها در تصویر است، نمایانگر میزان کلی نور یا انرژی در تصویر است و برای محاسبات مختلفی مانند نرمال‌سازی تصویر یا مقایسه سطوح روشنایی بین تصاویر مختلف به کار می‌رود.

(5

نوتبوک Q5.ipynb را با توجه به موارد خواسته شده در نوتبوک تکمیل کنید. (15 نمره)

پاسخ:

این کد یک تابع به نام conv تعریف می‌کند که یک فیلتر کانولوشن را پیاده‌سازی می‌کند با استفاده از ضرب عنصر به عنصر و تابع np.sum() در NumPy برای محاسبه بهینه جمع وزن‌دار همسایگی در هر پیکسل.

این تابع دو آرگومان می‌گیرد:

image: یک آرایه numpy که تصویر ورودی را به شکل (Hi، Wi) نشان می‌دهد

kernel: یک آرایه numpy که هسته کانولوشن را به شکل (Hk، Wk) نشان می‌دهد

کد مقادیر Hi، Wi را برای ذخیره شکل تصویر ورودی و Hk، Wk را برای ذخیره شکل هسته اولیه می‌کند.

آرایه خروجی out را با مقادیر صفر با ابعاد مشابه تصویر ورودی مقداردهی اولیه می‌کند.
سپس تصویر ورودی با استفاده از پاد صفری پدینگ شده و به خوبی با مرزها رفتار می‌کند. تعداد پادینگ توسط نیم ارتفاع و عرض هسته مشخص می‌شود.
سپس کد در هر پیکسل تصویر ورودی حرکت کرده و منطقه تصویر متناظر با اندازه هسته را استخراج می‌کند. ضرب عنصر به عنصر بین منطقه استخراج شده و هسته انجام می‌شود و سپس نتایج جمع شده تا مجموع وزن‌دار محاسبه شود. سپس نتیجه در آرایه خروجی out ذخیره می‌شود و در نهایت، تابع آرایه خروجی نهایی را پس از پردازش همه پیکسل‌ها در تصویر ورودی برمی‌گرداند.
این کد یک بخش اساسی از پردازش تصویر است که در وظایفی مانند تشخیص لبه، ابهام و استخراج ویژگی استفاده می‌شود. این کد نحوه عملکرد فیلترهای کانولوشن را نشان می‌دهد تا تبدیلاتی را به تصاویر به صورت کارآمد اعمال کند.

```
### YOUR CODE HERE

for i in range(Hi):
    for j in range(Wi):
        # Extract the region of the image corresponding to the kernel
        region = padded[i:i+Hk, j:j+Wk]
        # Element-wise multiplication followed by summation
        out[i, j] = np.sum(region * kernel)


### END YOUR CODE
```

این کد یک تابع به نام gaussian_kernel تعریف می‌کند که یک هسته گاوسی پیاده‌سازی می‌کند. این تابع فرمول‌های فیلتر گاوسی را دنبال کرده و یک ماتریس فیلتر ایجاد می‌کند. در اینجا توضیحی از کد آمده است: این تابع یک تابع کرنل گاوسی تعریف می‌کند که با استفاده از فرمول‌های فیلتر گاوسی پیروی می‌کند و یک ماتریس فیلتر ایجاد می‌کند.
تابع گاوسی برای محاسبه مقادیر فیلتر از np.pi و np.exp استفاده می‌کند تا مقدار pi و exp را محاسبه کند.
تابع gaussian_kernel دو آرگومان می‌گیرد:
size: یک عدد صحیح که اندازه ماتریس خروجی را نشان می‌دهد
sigma: یک عدد اعشاری که برای محاسبه مقادیر فیلتر استفاده می‌شود.

تابع یک ماتریس filter_values با ابعاد (size، size) با مقادیر صفره ایجاد می‌کند.

دلتا به صورت (اندازه-1) / 2 محاسبه می‌شود.

سپس کد در هر فرایند انجام می‌شود:

برای i در محدوده(size) و j در محدوده(size)

x برابر با i - delta و y برابر با j - delta قرار می‌گیرد

مقدار فیلتر با استفاده از فرمول فیلتر گاوسی محاسبه می‌شود

مقدار نهایی در filter_values ذخیره می‌شود.

در نهایت، تابع مقادیر فیلتر نهایی را برمی‌گرداند پس از پردازش همه مقادیر در ماتریس خروجی.

این کد نشان‌دهنده محاسبه هسته گاوسی است که یک تکنیک پرکاربرد در پردازش تصویر برای اعمال افزایش نویز، ابهام و شارپنینگ است.

```
### YOUR CODE HERE
for i in range(size):
  for j in range(size):
    x = i - delta
    y = j - delta
    filter_values[i, j] = (1 /(2 * np.pi * sigma ** 2)) * np.exp(-(x ** 2 + y ** 2) / (2 * sigma**2))
### END YOUR CODE
```

تابع gradient که در کد بالا تعریف شده است، ورودی یک تصویر خاکستری به صورت آرایه‌ای numpy با ابعاد (H، W) می‌گیرد و مقدار و جهت گرادیان را برای هر پیکسل در تصویر ورودی برمی‌گرداند.

در این تابع، ابتدا آرایه‌های G و theta با ابعاد مشابه تصویر ورودی ایجاد می‌شوند. سپس گرادیان در جهت x و y با استفاده از توابع partial_x و partial_y محاسبه می‌شود. سپس مقدار گرادیان G برای هر پیکسل با استفاده از فرمول Gx2+Gy2\sqrt{Gx^2 + Gy^2}Gx2+Gy2 و جهت گرادیان theta با استفاده از فرمول (arctan2(Gy,Gx)\arctan2(Gy, Gx)arctan2(Gy,Gx) و به واحد درجه تبدیل شده و مقدار آن به باقی‌مانده تقسیم ۳۶۰ گردانده می‌شود.

در نهایت، زوج مقدار و جهت گرادیان برای هر پیکسل به عنوان خروجی تابع برگردانده می‌شود.

```
### YOUR CODE HERE
Gx = partial_x(img)
Gy = partial_y(img)
G = np.sqrt(Gx**2 + Gy ** 2)
theta = np.arctan2(Gy, Gx)
theta = np.degrees(theta) % 360
### END YOUR CODE
```

تابع non_maximum_suppression که در کد بالا تعریف شده است، با ورودی‌های تصویر مقدار گرادیان (G) و جهت گرادیان (theta) کار می‌کند و عملیات non-maximum suppression را انجام می‌دهد. این عملیات به ازای هر پیکسل، مقدار گرادیان آن پیکسل را با دو پیکسل همسایه در جهت گرادیان مقایسه می‌کند و اگر مقدار گرادیان پیکسل مورد نظر بزرگتر یا مساوی این دو پیکسل همسایه باشد، مقدار آن را حفظ می‌کند و در غیر این صورت مقدار آن را صفر قرار می‌دهد.

در این تابع، ابتدا ابعاد تصویر G گرفته شده و یک آرایه خروجی با ابعاد مشابه ایجاد می‌شود. سپس جهت گرادیان‌ها به نزدیک‌ترین زاویه ۴۵ درجه گرد می‌شود. سپس برای هر پیکسل در تصویر، مقدار جهت فعلی گرادیان یافته شده و دو همسایه مربوط به آن جهت از نقشه زاویه angle_map استخراج می‌شوند. سپس مقادیر گرادیان این دو همسایه محاسبه شده و با مقدار گرادیان پیکسل فعلی مقایسه می‌شوند. اگر مقدار گرادیان پیکسل فعلی بزرگتر یا مساوی این دو همسایه باشد، مقدار آن در تصویر خروجی حفظ می‌شود و در غیر این صورت مقدار آن به صفر تنزل می‌یابد.

در نهایت، تصویر خروجی که شامل نتایج non-maximum suppression است، به عنوان خروجی تابع برگردانده می‌شود.

```
### BEGIN YOUR CODE

angle_map = {
    0: ((0, 1), (0, -1)),       # East and West
    45: ((1, 1), (-1, -1)),     # Northeast and Southwest
    90: ((1, 0), (-1, 0)),      # North and South
    135: ((1, -1), (-1, 1)),    # Northwest and Southeast
    180: ((0, 1), (0, -1)),     # East and West
    225: ((1, 1), (-1, -1)),    # Northeast and Southwest
    270: ((1, 0), (-1, 0)),     # North and South
    315: ((1, -1), (-1, 1))     # Northwest and Southeast
}

for y in range(1, H-1):
    for x in range(1, W-1):
        current_angle = theta[y, x]
        neighbors = angle_map[current_angle]
        n1 = G[y + neighbors[0][0], x + neighbors[0][1]]
        n2 = G[y + neighbors[1][0], x + neighbors[1][1]]

        # Suppress the pixel value if it's not a local maximum
        if G[y, x] >= n1 and G[y, x] >= n2:
            out[y, x] = G[y, x]
        else:
            out[y, x] = 0

### END YOUR CODE
```

تابع get_neighbors که در کد بالا تعریف شده است، با ورودی‌های موقعیت یک پیکسل (y، x) و ابعاد تصویر (H، W)، اندیس‌های همسایه‌های معتبر پیکسل (y، x) را برمی‌گرداند. همسایه‌های معتبر باید شرایط زیر را ارضا کنند:

0 <= i و i < H

0 <= j و j < W

(i، j) != (y، x)

در این تابع، یک لیست خالی به نام neighbors ایجاد می‌شود. سپس برای هر dy در [-1، 0، 1] و dx در [-1، 0، 1]، همسایه‌های معتبر پیکسل (y، x) محاسبه می‌شوند. در هر مرحله، پیکسل مرکزی خود (y، x) را نادیده می‌گیریم. سپس با افزودن dy و dx به موقعیت y و x، موقعیت جدید ny و nx محاسبه می‌شود. سپس اطمینان حاصل می‌شود که موقعیت‌های جدید داخل محدوده‌های تصویر هستند و در صورت اینکه شرایط بالا برقرار باشند، این همسایه به لیست neighbors اضافه می‌شود.
در نهایت، لیست neighbors که شامل اندیس‌های همسایه‌های معتبر پیکسل (y، x) است، به عنوان خروجی تابع برگردانده می‌شود.

```
### YOUR CODE HERE

for dy in [-1, 0, 1]:
    for dx in [-1, 0, 1]:
        # Skip the center pixel itself
        if dy == 0 and dx == 0:
            continue
        ny, nx = y + dy, x + dx
        # Ensure the new coordinates are within the image boundaries
        if 0 <= ny < H and 0 <= nx < W:
            neighbors.append((ny, nx))
### END YOUR CODE
```

تابع link_edges که در کد بالا تعریف شده است، با ورودی‌های تصویر دودویی strong_edges و weak_edges کار می‌کند و همسایه‌های لبه‌های ضعیف که به لبه‌های قوی متصل هستند را پیدا

کرده و آن‌ها را به هم متصل می‌کند. در این تابع، به ازای هر پیکسل در strong_edges، جستجوی اولویت از پیکسل‌های متصل در weak_edges انجام می‌شود. در اینجا یک پیکسل (a، b) به یک پیکسل (c، d) متصل است اگر (a، b) یکی از هشت پیکسل همسایه (c، d) باشد.

در این تابع، ابعاد تصویر strong_edges گرفته شده و اندیس‌های پیکسل‌های غیر صفر در strong_edges استخراج می‌شود. سپس یک آرایه جدید به نام edges با ابعاد (H، W) ایجاد می‌شود. سپس یک آرایه visited با ابعاد (H، W) ایجاد شده و یک صف queue تهیه می‌شود.

سپس برای هر پیکسل در اندیس‌ها، آن پیکسل به صف اضافه شده، مقدار edges برای آن پیکسل به True تغییر می‌یابد و visited برای آن پیکسل به True تغییر می‌کند. سپس تا زمانی که صف خالی نشود، از انتهای صف یک پیکسل گرفته شده و همسایه‌های آن پیکسل با استفاده از تابع get_neighbors محاسبه می‌شود. برای هر همسایه، اگر هنوز بازدید نشده بود و در weak_edges مقدار آن برابر با True بود، مقدار edges برای آن همسایه به True تغییر می‌یابد، visited برای آن همسایه به True تغییر می‌کند و آن همسایه به صف اضافه می‌شود.

در نهایت، تصویر edges که شامل لبه‌های متصل شده است، به عنوان خروجی تابع برگردانده می‌شود.

و در نهایت این عکس، عکس نهایی ما می باشد و عکس های دیگر در کد موجود می باشند.

(6

میخواهیم از الگوریتم RANSAC برای یافتن پارامترهای یک دایره در تصویر استفاده کنیم. در صورتی که بدانیم تنها 40 درصد از لبه های تصویر مربوط به دایره مورد نظر است و بخواهیم با احتمال بالای 0.99 به پارامترهای صحیح دست پیدا کنیم، به چند تکرار نیاز است؟ (5 نمره)

پاسخ:

رای محاسبه تعداد تکرارهای مورد نیاز الگوریتم RANSAC برای رسیدن به احتمال موفقیت مشخص (در اینجا 0.99 یا 99٪)، می‌توانیم از فرمول زیر استفاده کنیم:

$$\frac{\log(1-P)}{\log(1-w^n)} = N$$

که در آن:

- P احتمال موفقیت مطلوب است (در این مثال 0.99).
- w نسبت نقاط داده‌های درست در داده‌ها است (در این مثال 40٪ یا 0.4).
- n تعداد نقاط لازم برای مدل‌سازی یک دایره، که در اینجا 3 نقطه است (چرا که برای تعریف یک دایره نیاز به حداقل سه نقطه است).

با جایگذاری این مقادیر در فرمول:

$$\frac{\log(1-0.99)}{\log(1-0.4^3)} = N$$

$$0.064 = {}^{3}0.4$$

$$\frac{\log(0.01)}{\log(1-0.064)} = N$$

$$\frac{\log(0.01)}{\log(0.936)} = N$$

$$\frac{4.60517-}{0.06614-} = N$$

$$69.59 \approx N$$

بنابراین، تقریباً 70 تکرار برای رسیدن به احتمال موفقیت 99٪ در یافتن پارامترهای صحیح دایره در یک تصویر با 40٪ نقاط مربوط به دایره، لازم است.

(7

a.میدانیم برای تشخیص خط از الگوریتم Hough و LSD استفاده می شود. این دو روش را از جنبه های مختلف با هم مقایسه کنید. (حداقل سه مورد را بررسی کنید) (5 نمره)

پاسخ:

الگوریتم‌های Hough Transform و LSD(Line Segment Detector) دو روش متفاوت برای تشخیص خطوط در تصاویر هستند. در اینجا به مقایسه این دو روش از سه جنبه مختلف پرداخته می‌شود:

1. روش و دقت:

- **Hough Transform**: این روش یک تکنیک مبتنی بر رأی‌گیری در فضای پارامتری است. برای هر نقطه در تصویر، یک منحنی در فضای پارامتری ترسیم می‌شود و هرگاه این منحنی‌ها در نقطه‌ای تقاطع پیدا کنند، نشان‌دهنده وجود یک خط در تصویر است. این روش برای تشخیص خطوط بلند و واضح مناسب است و می‌تواند حتی در حضور نویز نیز عملکرد خوبی داشته باشد.

- **LSD : Line Segment Detector** یک الگوریتم مبتنی بر گرادیان است که به طور مستقیم خطوط را در تصویر تشخیص می‌دهد و قادر است قطعات خط کوتاه را نیز تشخیص دهد. این الگوریتم به دنبال الگوهای محلی است و برای تشخیص خطوط در پیکسل‌های نزدیک یکدیگر به کار می‌رود.

2. پیچیدگی محاسباتی و سرعت:

- **Hough Transform**: به دلیل نیاز به محاسبه و رأی‌گیری در فضای پارامتری، معمولاً سرعت پایین‌تری دارد و منابع محاسباتی بیشتری مصرف می‌کند، خصوصاً در تصاویر بزرگ یا زمانی که دقت بالایی مورد نیاز است.

- **LSD**: این الگوریتم به طور کلی سریع‌تر و کم هزینه‌تر از Hough Transform است، زیرا به جای محاسبه در فضای پارامتری، مستقیماً بر روی پیکسل‌های تصویر کار می‌کند و بر اساس تغییرات شدت محلی عمل می‌کند.

3. دقت و شرایط استفاده:

- **Hough Transform**: در شرایطی که خطوط در تصویر به خوبی تعریف شده‌اند و ممکن است کمی نویز داشته باشند، بسیار مؤثر است. می‌تواند خطوط را حتی اگر بخش‌هایی از آن‌ها ناپدید شده باشد، تشخیص دهد.
- **LSD**: برای تصاویری که خطوط کوتاه و یا خطوط با جهت‌های مختلف دارند مناسب است. به خوبی می‌تواند خطوط را در تصاویر با وضوح بالا و با دقت زیاد تشخیص دهد، اما در حضور نویز، ممکن است نتایج نامطلوبی تولید کند.

به طور کلی، انتخاب بین این دو روش بستگی به نوع داده‌ها، شرایط تصویر و نیازهای خاص پروژه دارد. Hough ممکن است برای تشخیص خطوط طولانی در حضور نویز مناسب باشد، در حالی که LSD برای تشخیص دقیق و سریع خطوط در تصاویر با جزئیات بالا ایده‌آل است.

b.تصویر jpg.1 را در نظر بگیرید. دایره های موجود در این تصویر را تشخیص دهید و آنها را حذف کنید. برای حل این قسمت از توابع cv2.HoughCircles و cv2.circle استفاده نمایید. (5 نمره)

تصویر خروجی	تصویر ورودی
	

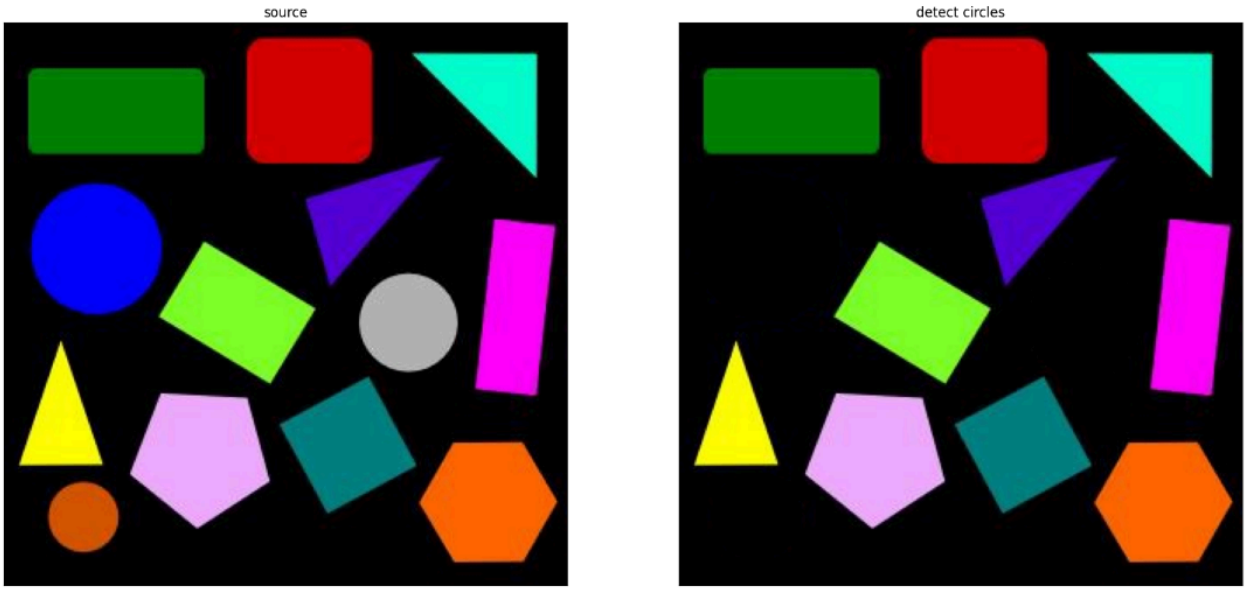
پاسخ:

```
#Writer your code here
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# gray = cv2.medianBlur(gray, 5)
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 0.85, 40,
                           param1 = 60, param2 = 40, minRadius = 10, maxRadius = 110)

if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, :]:
        cv2.circle(out_img, (i[0], i[1]), i[2] + 5, (0, 0, 0), -1)
```

تابع remove_circles که در کد بالا تعریف شده است، تصویر ورودی را دریافت کرده و دایره‌های موجود در تصویر را حذف کرده و تصویر حاصل را برمی‌گرداند. در این تابع، ابتدا یک کپی از تصویر ورودی تهیه می‌شود. سپس با تبدیل تصویر ورودی به تصویر خاکستری (gray)، از تابع cv2.HoughCircles برای تشخیص دایره‌ها در تصویر استفاده می‌شود.

در تابع cv2.HoughCircles، پارامترهای مختلفی مانند نوع روش تشخیص دایره، دقت، پارامترهای الگوریتم، شعاع حداقل و حداکثر دایره و ... تنظیم می‌شوند. اگر دایره‌ها در تصویر شناسایی شوند (circles is not None)، آن‌ها را به نزدیک‌ترین مقدار صحیح 16 بیتی گرد می‌کنیم و برای هر دایره، یک دایره با شعاع بیشتر از اندازه مشخص شده (+5) با رنگ سیاه رسم می‌کنیم.

در نهایت، تصویر حاصل که دایره‌ها از آن حذف شده‌اند، به عنوان خروجی تابع برگردانده می‌شود.



c.تصویر خروجی مرحله قبل را در نظر بگیرید. با استفاده از الگوریتم Hough خطوط موجود در تصویر را به دست آورید و با خطوط سفید رنگ بر روی آن نمایش دهید. برای حل این قسمت از توابع cv2.canny و cv2.HoughLinesP و cv2.Lines استفاده نمایید.(5 نمره)

امتیازی: علاوه بر قسمت قبل نتیجه خواسته شده را با استفاده cv2.HoughLines پیاده سازی کنید.(5 نمره)

تصویر خروجی	تصویر ورودی
	

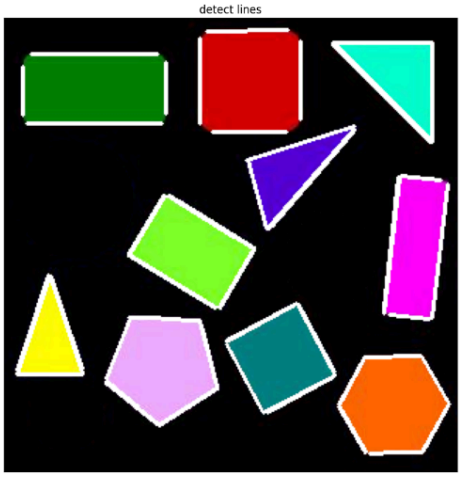
پاسخ:

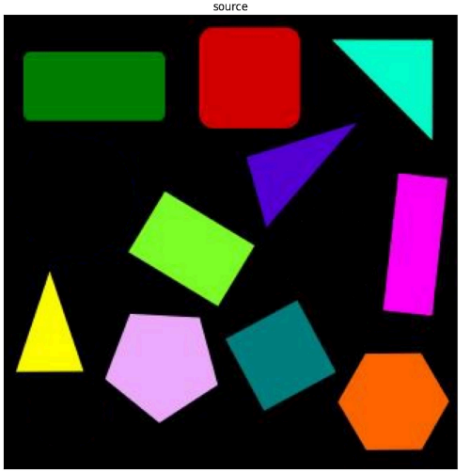
```python
#Writer your code here
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply Gaussian blur to help reduce noise and improve edge detection
blurred = cv2.GaussianBlur(gray, (3, 3), 0)

edges = cv2.Canny(blurred, 50, 150, apertureSize=3)
lines = cv2.HoughLinesP(edges, 1, np.pi/(360*180) , threshold=20, minLineLength=10, maxLineGap=10)
if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(out_img, (x1, y1), (x2, y2), (255, 255, 255), 2)
```

در این خط کد، برای هر خطی که توسط تابع cv2.HoughLinesP شناسایی شده است، نقاط آن خط را گرفته و یک خط با رنگ سفید و ضخامت 2 پیکسل بین این دو نقطه رسم می‌شود. این کار با استفاده از تابع cv2.line انجام می‌شود.

در نهایت، تصویر حاصل که خطوط در آن شناسایی شده‌اند، به عنوان خروجی تابع برگردانده می‌شود.

که تصویر آن در زیر مشخص شده است.





d.امتیازی: حال تصویر jpg.7 را بخوانید و با استفاده از تابع findCounters ,approxPolyDP نقاط گوشه شکل را پیدا کنید و علامت بزنید سپس تشخیص دهید که شکل متعلق به چه کلاسی است و با متنی این را در تصویر مشخص کنید.(10نمره)

پاسخ:

```python
circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1, minDist=20, param1=50, p
if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0,:]:
        cv2.circle(out_img, (i[0], i[1]), i[2], (255, 255, 0), 2)  # Draw circles in yellowl

contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
edge_img = np.zeros_like(image)
cv2.drawContours(edge_img, contours, -1, (0,125, 125), 3)  # Drawing all contours in green with
for cnt in contours:
    epsilon = 0.02 * cv2.arcLength(cnt, True)
    approx = cv2.approxPolyDP(cnt, epsilon, True)
    if len(approx) == 3:
        cv2.drawContours(out_img, [approx], 0, (255, 0, 0), 2)  # Draw triangles in blue
        print("blue shapes are triangle.")
    elif len(approx) == 4:
        x, y, w, h = cv2.boundingRect(approx)
        aspectRatio = float(w) / h
        if 0.8 <= aspectRatio <= 1.2:  # More square-like
            cv2.drawContours(out_img, [approx], 0, (0, 255, 0), 2)
            print("green shapes are square.")
        else:  # More rectangle-like
            cv2.drawContours(out_img, [approx], 0, (0, 0, 255), 2)
            print("red shapes are rectangle.")
    elif len(approx) > 8:
        area = cv2.contourArea(cnt)
        perimeter = cv2.arcLength(cnt, True)
        circularity = 4 * np.pi * (area / (perimeter * perimeter))
        if circularity > 0.7:  # Lower the threshold to catch more imperfect circles
            cv2.drawContours(out_img, [approx], 0, (255, 255, 0), 2)  # Draw circles in yellow'
        print('Pale blue shapes are circles.')
    for vertex in approx:
        x, y = vertex.ravel()
        cv2.circle(out_img, (x, y), 5, (10, 134, 100), -1)  # Vertex in red
```

در این خطوط کد، اشکال مستطیل‌های گرد و مربع‌های گرد در تصویر شناسایی می‌شوند و مرزهای این اشکال در تصویر مشخص می‌شوند.

در این بخش از کد، از تابع cv2.HoughCircles برای شناسایی دایره‌ها استفاده می‌شود. اگر دایره‌ها در تصویر شناسایی شوند، آن‌ها را با رنگ زرد و ضخامت 2 پیکسل رسم می‌کنیم.

با استفاده از تابع cv2.findContours، ابتدا لبه‌های تصویر با استفاده از تابع Canny و GaussianBlur به دست می‌آید. سپس با استفاده از این لبه‌ها، کانتورها (contours) در تصویر پیدا می‌شوند و در تصویر جدید edge_img با رنگ سبز و ضخامت 3 پیکسل رسم می‌شوند.

برای هر کانتور، ابتدا با استفاده از تابع cv2.approxPolyDP، یک تقریب چندضلعی از کانتور به دست می‌آید. سپس بر اساس تعداد ضلع‌های تقریبی، اقدامات زیر انجام می‌شود:

اگر تعداد ضلع‌ها برابر با 3 باشد، مثلث رسم می‌شود و پیام "اشکال آبی مثلث هستند." چاپ می‌شود.
اگر تعداد ضلع‌ها برابر با 4 باشد، ابتدا مستطیل تقریبی از کانتور محاسبه شده و نسبت ابعاد آن بررسی می‌شود. اگر نسبت ابعاد در بازه 0.8 تا 1.2 باشد (شبیه به مربع)، مربع رسم می‌شود و پیام "اشکال سبز مربع هستند." چاپ می‌شود. در غیر این صورت (شبیه به مستطیل)، مستطیل رسم می‌شود و پیام "اشکال قرمز مستطیل هستند." چاپ می‌شود.
اگر تعداد ضلع‌ها بیشتر از 8 باشد، ابتدا مساحت و محیط کانتور محاسبه شده و استوانه‌ایتی (circularity) بررسی می‌شود. اگر استوانه‌ایتی بیشتر از 0.7 باشد (دایره نزدیک به کامل)، دایره رسم می‌شود و پیام "اشکال آبی محوطه‌ای دایره‌ای هستند." چاپ می‌شود. سپس برای هر نقطه‌ی تقریبی، یک دایره قرمز با شعاع 5 پیکسل رسم می‌شود.

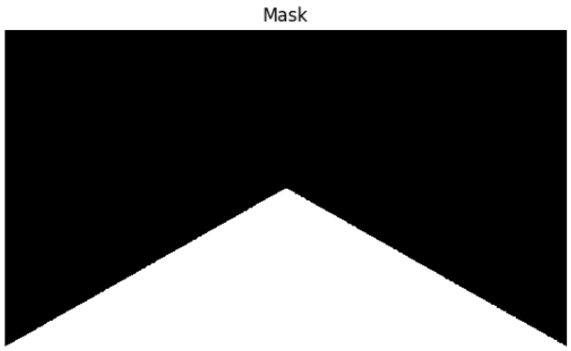
(8

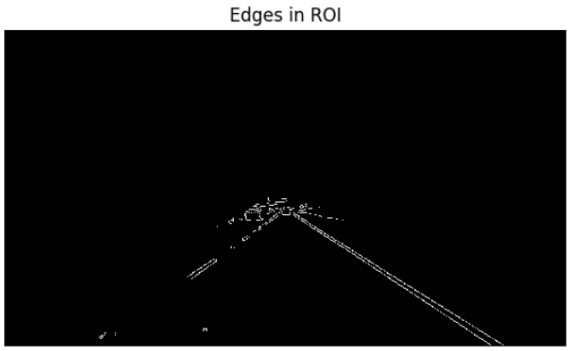
امتیازی : نوتبوک Q8.ipynb را با توجه به موارد خواسته شده تکمیل کنید. (10نمره)

پاسخ:

در ابتدا canny را از سوال 5 import می کنیم. که در اینجا من تمام کد سوال 5 را در یک سوال گذاشته ام. پس تصویر خود را خوانده و با canny لبه ها را به دست می آوریم و سپس یک mask را

در نظر می گیریم و تنها لبه های موجود در آن mask را در تصویر نشان می دهیم که به صورت زیر است.





در این تابع، امتیازات نقاط تصویر ورودی به فضای هاف تبدیل می‌شوند. از پارامترهای

ρ=x⋅cos(θ)+y⋅sin(θ)\rho = x \cdot \cos(\theta) + y \cdot

(\sin(\theta)ρ=x⋅cos(θ)+y⋅sin(θ) برای تبدیل نقطه (x,y)(x,y)(x,y) به یک تابع شبیه به سینوس در فضای هاف استفاده می‌شود.

در ابتدا، محدوده‌های ρ\rhoρ و θ\thetaθ تعیین می‌شوند. سپس مقادیر ρ\rhoρ و θ\thetaθ محاسبه و تعریف می‌شوند.

مقادیر cos(θ)\cos(\theta)cos(θ) و (sin(θ)\sin(\theta)sin(θ) ذخیره شده و تعداد θ\thetaθها محاسبه می‌شود.

یک آکومولاتور در فضای هاف با ابعاد مناسب ایجاد می‌شود. نقاط غیر صفر تصویر (مختصات x و y) استخراج می‌شوند.

برای هر نقطه (x,y)(x, y)(x,y) در تصویر، مقدار ρ\rhoρ متناظر با مقادیر θ\thetaθ محاسبه شده و آکومولاتور در مختصات متناظر افزایش می‌یابد.

در نهایت، آکومولاتور، مقادیر ρ\rhoρ و θ\thetaθ که به ترتیب آرایه‌های numpy با ابعاد (m,) (m ،(n,) و (n,) هستند، به عنوان خروجی تابع برگردانده می‌شوند.

```
    for x, y in zip(xs, ys):
        for theta_idx in range(num_thetas):
            rho = int(x * cos_t[theta_idx] + y * sin_t[theta_idx])
            rho_idx = np.searchsorted(rhos, rho)  # find index in rhos array
            accumulator[rho_idx, theta_idx] += 1

    ### END YOUR CODE
```

در این بخش از کد، تبدیل هاف روی ROI (منطقه تصویری) انجام می‌شود و سپس خطوط مربوط به لبه‌های موجود در تصویر تشخیص داده می‌شوند.

در ابتدا، تابع تبدیل هاف بر روی ROI اعمال می‌شود و مقادیر آکومولاتور، ρ\rhoρ و θ\thetaθ به دست می‌آیند.

سپس برای 20 بار، موقعیت بیشینه آکومولاتور یافته شده و مقادیر ρ\rhoρ و θ\thetaθ متناظر با آن استخراج می‌شوند.

بر اساس مقادیر ρ\rhoρ و θ\thetaθ، خطوط متناظر در فضای xyxyxy محاسبه می‌شوند. این خطوط با استفاده از شیب (aaa) و نقطه تلاقی با محور yyy (bbb) تعریف می‌شوند.

در این بخش از کد، بررسی می‌شود که آیا خطوط راست و چپ تشخیص داده شده‌اند یا خیر. اگر هر دو خط تشخیص داده شوند، حلقه متوقف می‌شود.

سپس بر اساس شیب خط، نقاط متناظر با آن در تصویر محاسبه می‌شوند و در صورتی که در محدوده خاصی از تصویر قرار داشته باشند، به لیست نقاط خط اضافه می‌شوند.

در نهایت، تصویر اصلی به همراه خطوط تشخیص داده شده بر روی آن نمایش داده می‌شود.

نتیجه نیز در آخر به صورت زیر می باشد.