به نام خدا

درس مبانی بینایی کامپیوتر

تمرین سری پنجم

مدرس درس:
جناب آقای دکتر محمدی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۸/۱۱

سوال ۱ :

مراحل خواسته شده را در نوت بوک Q1 پیاده سازی کنید.

- الف) ابتدا تصویر img_01.jpg را بخوانید و نویز تصویر را با تبدیل FFT حذف کنید. در حذف نویز از تمام مراحل خروجی گرفته و رسم کنید. (۳۰ نمره)
راهنمایی: همانطور که در تصویر مشاهده می شود خط تولید دارای یک سری روزنه‌هایی می باشد که به صورت متناوب در پس‌زمینه قرار دارند سعی کنید ان ها را به عنوان نویز تشخیص داده و حذف کنید.

- ب) لبه‌یاب canny را بر روی خروجی مرحله (الف) اجرا کنید. برای لبه‌یاب می‌توانید از کتابخانه آماده استفاده کنید. تمام پارامتر های تابع که مقداردهی می‌شوند با ذکر دلیل توضیح داده شوند. (مطلوب است در خروجی این مرحله فقط گل‌های زعفران بمانند) (۱۵ نمره)

- ج) از تصویر بدست آمده گرادیان بگیرید و با استفاده از تابع arctan2 جهت گرادیان‌های بدست آمده را محاسبه کنید. (۱۰ نمره)

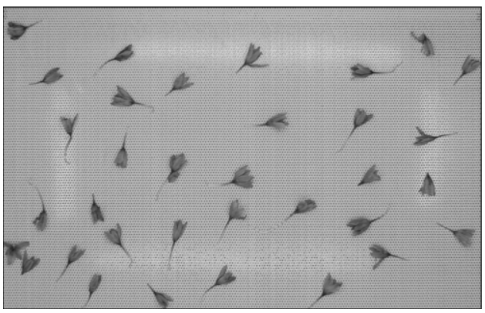
- د) با استفاده از جهت گرادیان‌های بدست آمده، راه‌حلی برای بدست آوردن نقطه برش ساقه از گلبرگ ارائه دهید. (۵ نمره)

پاسخ ۱:

- الف) نکته: برای حل این سوال، از لینک زیر کمک گرفته شد:

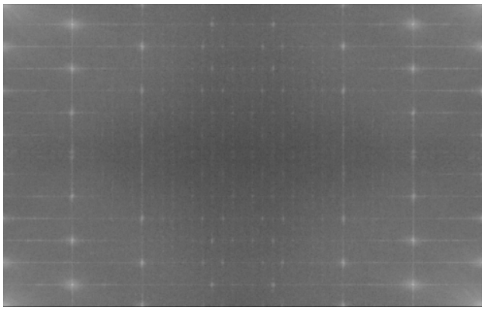
`https://scipy-lectures.org/intro/scipy/auto_examples/solutions/plot_fft_image_denoise.html`

در این بخش، ابتدا تصویر را خوانده و نمایش می‌دهیم که تصویر زیر حاصل می‌شود:



تصویر اولیه

سپس با استفاده دستور fft2، تبدیل فوریه تصویر را به دست می‌آوریم تا نقاط نویزی را پیدا کنیم. همانطور که در تصویر زیر نیز مشخص است، نقاط نویزی این تصویر به صورت متناوب در تمام پس‌زمینه پخش هستند.

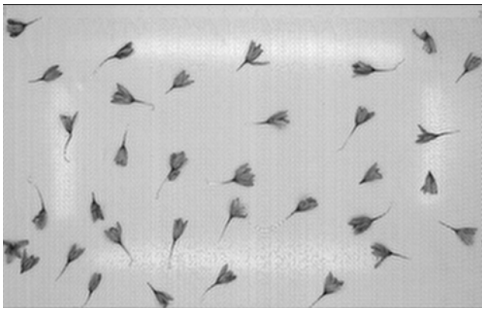


تصویر پس از اعمال تبدیل فوریه

سپس با صفر کردن این مقادیر نویز که شامل یک مستطیل افقی و یک مستطیل عمودی می‌شوند، نویز تصویر خود را حذف می‌کنیم. به این دلیل کل صفحه را صفر نکردیم چون گوشه‌های تصویر را به علت اطلاعات زیادشان، نمی‌توانیم حذف کنیم. تصویر حاصل از این بخش، به شرح زیر می‌باشد:

تصویر پس از صفر کردن مقادیر نویز

در انتها، با استفاده از یک تبدیل فوریه معکوس، تصویر خود را رفع نویز می‌کنیم که نتیجه زیر را به ما می‌دهد:



تصویر پس از صفر کردن مقادیر نویز

کد زده شده در این بخش نیز، در ادامه آورده شده است.

```python
# read original image
path = r'E:\University\Term7\FCV\Homeworks\HW5\Images'
original_image = cv2.imread(os.path.join(path,"img_01.jpg"),0)
w, h = original_image.shape[0] , original_image.shape[1]
plt.imshow(original_image, cmap='gray')
plt.show()
# apply 2D fourier transform
fourier_transform = np.fft.fft2(original_image)
plt.imshow(np.log(1+np.abs(fourier_transform)), cmap='gray')
plt.show()
# set zero the noisy points
k = 0.1
fourier_transform[int(w*k):int(w*(1-k))] = 0
fourier_transform[:,int(h*k):int(h*(1-k))] = 0
plt.imshow(np.log(1+np.abs(fourier_transform)), cmap='gray')
plt.show()
# apply ifft2 for 2D fourier transform
ifft2_image = np.real(np.fft.ifft2(fourier_transform))
plt.imshow(ifft2_image, cmap='gray')
plt.show()
```

✓ 1.9s

کد بخش الف

- ب) در این بخش با استفاده از تابع canny کتابخانه cv2، لبه‌های تصویر را تشخیص می‌دهیم. پارامترهای ورودی این تابع، عبارت هستند از تصویر ورودی، حد آستانه اول، حد آستانه دوم، اندازه کرنل عملگر sobel و یک boolean است که مشخص می‌کند از کدام معادله برای محاسبه استفاده کنیم. اگر این مقدار true باشد، از معادله $Edge\_Gradient(G) = \sqrt{g_x^2 + g_y^2}$ که دقیق‌تر است، استفاده می‌کند و در صورت false بودن، از معادله $Edge\_Gradient(G) = |g_x| + |g_y|$ که از دقت کمتری برخوردار هست، استفاده می‌کند. همچنین حد آستانه اول برای این است که می‌گوییم هر پیکسلی که اندازه گرادیانش از T_1 کوچک‌تر بود، غیر لبه است و هر پیکسلی که اندازه گرادیانش از T_2 بزرگ‌تر بود، لبه است. همچنین پیکسل‌هایی که بین این دو مقدار قرار دارند نیز در صورت متصل بودن مستقیم یا غیر مستقیم به لبه، لبه محسوب می‌شوند و در غیر این‌صورت، غیر لبه است. کد مربوط به این بخش و نتیجه آن، در تصویر زیر آورده شده است:

```
array = np.zeros((32,32),dtype=int)
array[31][2] = 1

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(14, 14), sharex=True, sharey=True)
for aa in (ax1, ax2, ax3):
    aa.set_axis_off()
ax1.imshow(abs(array), cmap='gray')
ax1.set_title('Original image')
ax2.imshow(abs(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax2.set_title('Frequency spectrum')
ax3.imshow(np.real(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax3.set_title('Real part')

array[31][2] = 0
array[31][3] = 1
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(14, 14), sharex=True, sharey=True)
for aa in (ax1, ax2, ax3):
    aa.set_axis_off()
ax1.imshow(abs(array), cmap='gray')
ax1.set_title('Shifted image')
ax2.imshow(abs(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax2.set_title('Frequency spectrum')
ax3.imshow(np.real(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax3.set_title('Real part')
✓ 0.4s
```

کد بخش ب

- ج) در این بخش، ابتدا با استفاده از عملگر sobel، مشتق عمودی و افقی تصویر را محاسبه می‌کنیم که ورودی‌های این تابع، به ترتیب، تصویر ورودی، عمق تصویر، مشتق در راستای x و مشتق در راستای y هستند. سپس با استفاده از arctan2، جهت گرادیان‌ها را به دست می‌آوریم. ورودی‌های این تابع نیز مشتق افقی و عمودی هستند و این تابع جهت گرادیان را در مقیاس رادیان، می‌دهد. کد و نتیجه این بخش نیز، در تصویر زیر قابل مشاهده است.

```
g_y = cv2.Sobel(src=canny_image, ddepth=cv2.CV_64F, dx=0, dy=1)
#print(g_y)
g_x = cv2.Sobel(src=canny_image, ddepth=cv2.CV_64F, dx=1, dy=0)
#print(g_x)
arctan2_image = np.arctan2(g_x,g_y)
arctan2_image
✓ 0.5s
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

کد بخش ج

- د) ابتدا یک تصویر باینری مناسب از گل زعفران با استفاده از عملگر sobel، به دست می‌آوریم. سپس با استفاده از روش رای‌گیری گفته شده در کلاس و جهت‌های گرادیان به دست آمده، جاهایی که گرادیان خیلی سریع تغییر می‌کند را به دست می‌آوریم که در واقع همان ساقه زعفران می‌شود. سپس با استفاده از روش hough، نقاط ابتدا و انتهای این خطوط را پیدا کرده و ابتدای آن را برش می‌زنیم.

سوال ۲:

طیف فرکانسی تصویر شکل ۱ چگونه خواهد شد؟ (به صورت تقریبی شرح دهید)، اگر پیکسل روشن به اندازه یک پیکسل به سمت راست شیفت داده شود طیف فرکانسی چه تغییری می کند؟ (۱۵ نمره)

شکل ۱- تصویری به ابعاد ۳۲ * ۳۲

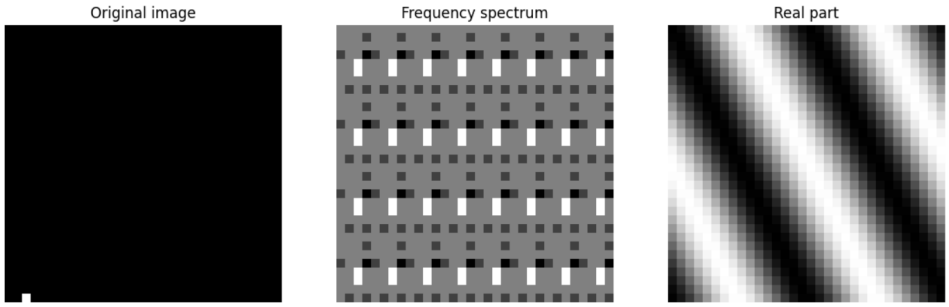
پاسخ ۲:

در تبدیل فوریه، میانگین کل پیکسل‌های تصویر، میزان تغییرات افقی، میزان تغییرات عمودی و همچنین شدت تغییرات، مورد بررسی قرار می‌گیرد. یعنی در واقع تبدیل فوریه به جای اینکه شدت روشنایی پیکسل را به ما بگوید، رفتار آن را توصیف می‌کند. بنابراین با توجه به توضیحات داده شده، کدی که برای این بخش زده شده و نتایج حاصل از آن، می‌توانیم بگوییم این تصویر در حوزه فرکانس، تغییرات افقی نسبتا آهسته و تغییرات عمودی سریع دارد. اگر این پیکسل یک واحد به سمت راست منتقل شود، میزان تغییرات افقی آن نیز افزایش می‌یابد اما تغییرات عمودی همچنان ثابت و زیاد است. همچنین همانطور که در تصویر نیز مشاهده می‌شود، با افزایش تغییرات افقی، تعداد خطوط بیشتر می‌شود و چون تغییرات افقی سریع‌تر است، ضخامت خطها نیز کمتر می‌شود.

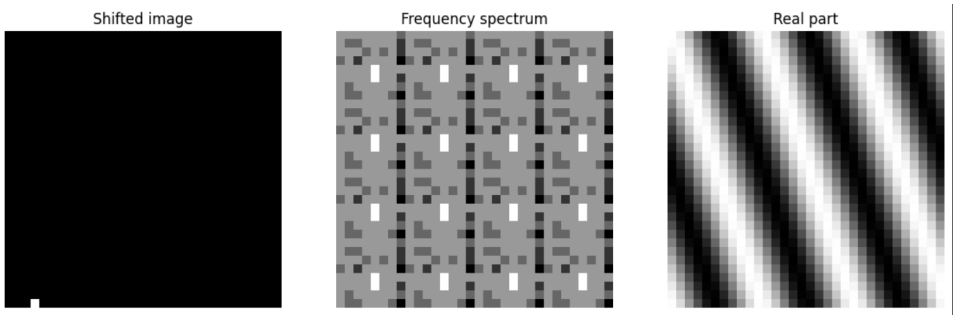
```python
array = np.zeros((32,32),dtype=int)
array[31][2] = 1

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(14, 14), sharex=True, sharey=True)
for aa in (ax1, ax2, ax3):
    aa.set_axis_off()
ax1.imshow(abs(array), cmap='gray')
ax1.set_title('Original image')
ax2.imshow(abs(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax2.set_title('Frequency spectrum')
ax3.imshow(np.real(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax3.set_title('Real part')

array[31][2] = 0
array[31][3] = 1
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(14, 14), sharex=True, sharey=True)
for aa in (ax1, ax2, ax3):
    aa.set_axis_off()
ax1.imshow(abs(array), cmap='gray')
ax1.set_title('Shifted image')
ax2.imshow(abs(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax2.set_title('Frequency spectrum')
ax3.imshow(np.real(np.fft.fftshift(np.fft.fft2(array))), cmap='gray')
ax3.set_title('Real part')
```

✓ 0.4s

کد مربوط به این بخش



نتایج تصویر اولیه



نتایج تصویر پس از انتقال

سوال ۳:

یک ماتریس کوچک (۵ * ۵)با یک لبه‌ی عمودی بسازید و سپس عملگر Sobel را روی آن اجرا نمایید و نتیجه را نشان دهید. (۲۰ نمره)

پاسخ ۳:

در ابتدا، یک ماتریس با لبه عمودی می‌سازیم. سپس عملگر sobel را یک بار در راستای افقی و یک بار در راستای عمودی، روی ماتریس خود، اجرا می‌کنیم. ورودی‌های تابع sobel، به ترتیب، تصویر ورودی، عمق تصویر خروجی، مشتق در راستای x، مشتق در راستای y و سایز kernel می‌باشد که می‌تواند ۱،۳،۵ و ۷ باشد. کد و خروجی مربوط به این بخش، در تصویر زیر آمده است:

```
vertical_matrix = np.zeros((5, 5))
for i in range(5):
    vertical_matrix[i, 0] = 1
sobely = cv2.Sobel(src=vertical_matrix, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=3)
print(sobely)
sobelx = cv2.Sobel(src=vertical_matrix, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=3)
print(sobelx)
```

✓ 0.5s

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
[[ 0. -4.  0.  0.  0.]
 [ 0. -4.  0.  0.  0.]
 [ 0. -4.  0.  0.  0.]
 [ 0. -4.  0.  0.  0.]
 [ 0. -4.  0.  0.  0.]]
```

کد و خروجی عملگر sobel

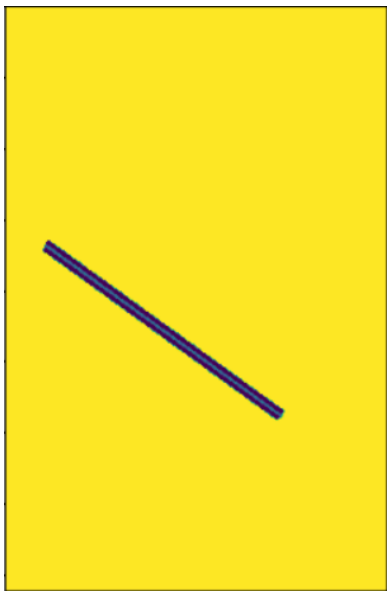
همانطور که در تصویر بالا نیز مشاهده می‌شود، در اینجا فقط مشتق افقی داریم و مشتق عمودی صفر است. البته نتیجه چندان دور از انتظار نبود؛ چرا که مقادیر ماتریس اولیه در راستای افقی تغییر می‌کردند و راستای عمودی ثابت هستند.

سوال ۴:

معادله خط تصویر img_02.jpg را با توجه به توضیحات داده شده در کلاس به دست آورید. (۱۵ نمره)

پاسخ ۴:

طبق توضیحات داده شده در کلاس، روش hough، بهترین روش برای تشخیص خط معرفی شد. در ابتدا با استفاده از لبه‌یاب canny، لبه‌های تصویر را مشخص می‌کنیم. ورودی‌های این تابع به ترتیب عبارت تصویر ورودی، حد آستانه اول و حد آستانه دوم می‌باشند. سپس با استفاده از تابع HoughLinesP، نقاط ابتدایی و انتهایی هر پاره‌خط را پیدا می‌کنیم. ورودی‌های این تابع نیز عبارت هستند از تصویر ورودی، ρ، θ و حد آستانه. در این شکل، به علت ضخیم بودن خط، دو لبه بالایی و پایینی خط، به عنوان خط پیدا شد که در تصویر زیر قابل مشاهده است:



خطوط تشخیص داده شده در شکل

در اینجا، من برای دقیق‌تر شدن جواب، معادله خطی که بین این دو خط قرار می‌گیرد را به دست آوردم. کد مربوط و نتیجه حاصل از این‌بخش نیز، در تصویر زیر آمده است:

```
path = r'E:\University\Term7\FCV\Homeworks\HW5\Images'
img = cv2.imread(os.path.join(path,"img_02.jpg"),0)
edges = cv2.Canny(img,400,800)
lines = cv2.HoughLinesP(edges,1,np.pi/180,100)
x0 = (lines[0, 0, 0] + lines[1, 0, 0]) / 2
y0 = (lines[0, 0, 1] + lines[1, 0, 1]) / 2
x1 = (lines[0, 0, 2] + lines[1, 0, 2]) / 2
y1 = (lines[0, 0, 3] + lines[1, 0, 3]) / 2
m = (y0 - y1) / (x0 - x1)
c = y0 - m * x0
print(f"y = -{m:.2f}x + {c:.2f}")
```

✓ 0.6s

```
y = -0.72x + 148.48
```