.1

الف) در این قسمت ابتدا در تابع generateRandomMtrix یک ماتریس را به صورت رندوم با randint میسازیم که دو آرگومان اول حد پایین و بالای اعداد را مشخص کرده و در آرگومان دوم ابعاد را میدهیم. سپس تابع convolve را تعریف میکنیم به این صورت که هر درایه ی ماتریش از جمع ضرب نقطه ای درایه های کرنل با ماتریس اصلی بدست می آید. در قسمت بعد کرنل های Gx , Gy را تعریف کرده. سپس ماتریس رندوم 10 در 10 میسازیم و آن را به صورت افقی و عمودی در کرنل هایی که تعریف کردیم کانوالو میکنیم. در قسمت بعد هم ماتریس magnitude را ساخته و همچنین alpha هم همان ماتریس جهت است که از فرمول های زیر بدست آمده اند.

$$M(x,y) = \|\nabla f\| = \mathrm{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

$$\alpha(x,y) = \mathrm{dir}(\nabla f) = \mathrm{atan2}(g_y, g_x)$$

ب) دراین قسمت پس از خواندن عکس و بردن آن به gray scale، کرنل گاوسی را تعریف میکنیم. بعد هم عکس را در حالتی که فیلتر گوسی نخورده و در جهت افقی و عمودی عملگر sobel روی آن اعمال شده را خروجی داده :

در ادامه هم ابتدا فیلتر گوسی را اعمال میکنیم و sobel میزنیم :

همان طور که دیده میشود تصویر با فیلتر گوشی محوتر است و نویز های کمتری دارد یعنی قسمت هایی مثل سوییشرت و دیوار را بهتر لبه یابی کرده است پس نتیجه میگیریم فیلتر گوسی باعث کاهش نویز و تقویت لبه میشود.

در ادامه مقدار magnitude و گرادیان را بدست می آوریم.

ج) در این قسمت از تابع opencv استفاده میکنیم که ارگومان های آن به ترتیب:

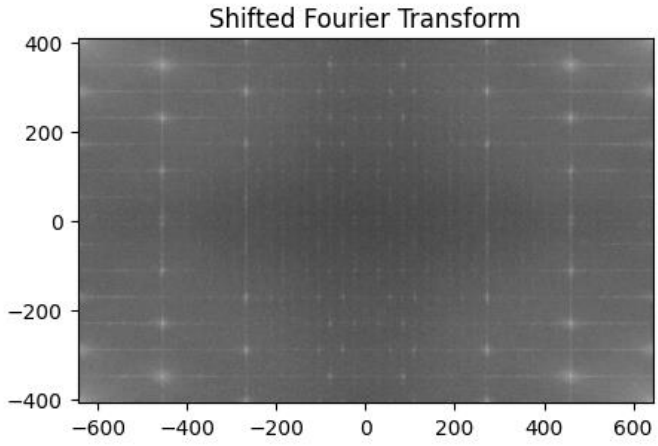
عکس ورودی ،

عمق تصویر که مقادیر متفاوتی میتواند داشته باشد CV_8U یعنی عکس پیکسل های آن اعداد صحیح 8 بیتی هستند،

مشتق در راستای x و y دو آرگون بعدی هستند که اگر 0 باشد یعنی مشتق نمیگیریم و اگر 1 باشد یعنی مشتق مرتبه اول داریم،

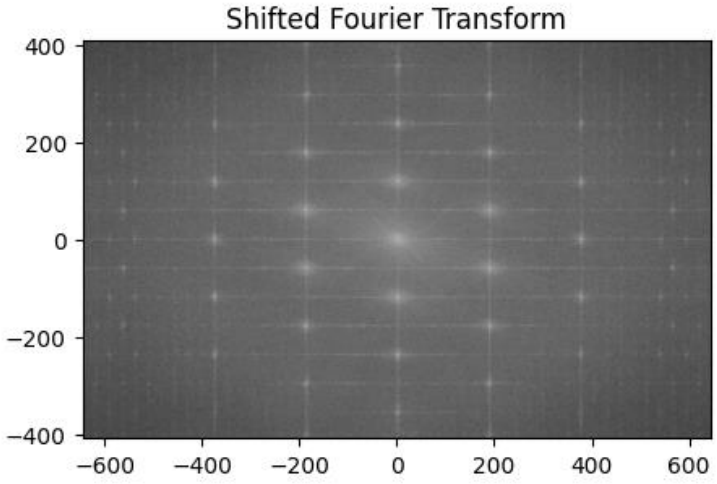
درآخر هم اندازه ی کرنل سوبل را میدهیم.

.2

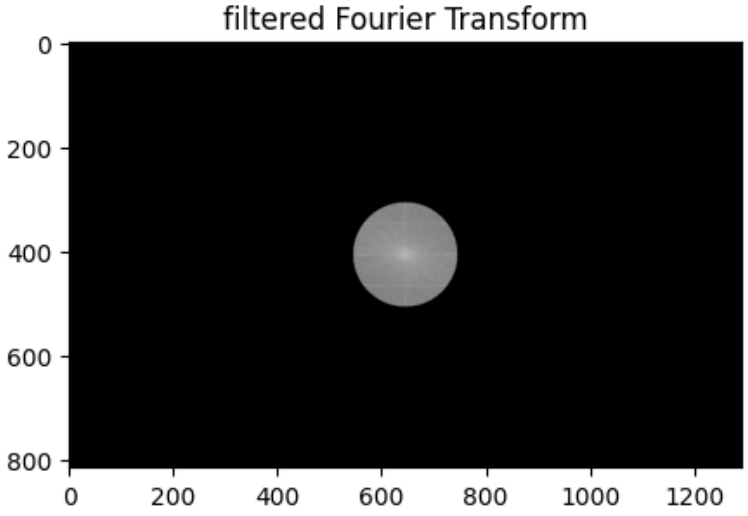


الف) ابتدا تصویر را خوانده و آن را gray scale میبریم.
سپس تبدیل فوریه آن را با تابع fft.fft2 بدست می آوریم
که در شکل میتوان دید:

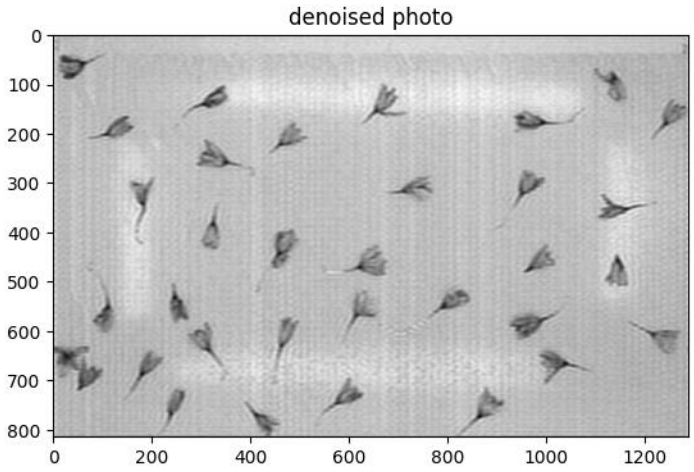
در ادامه فیلتر را با تابع fft.fftshift شیفت میدهیم به این صورت که نقاط وسط صفحه نشان دهنده ی فرکانس های کمتر باشد:



حالا باید قسمت های اطراف دایره سفید وسط که نشان دهنده ی فرکانس های بیشتر است را 0 کنیم تا نویزهای متناوبمان را از بین ببریم. برای این کار یک mask تعریف میکنیم که در قسمت های داخل دایره مقدار 1 و خارج آن مقدار0 دارد. بعد با ضرب کردن این ماتریس در تبدیل فوریه امان شکل زیر بدست می آید.



این فوریه را دوباره شیفت میدهیم تا به حالت اولیه برگردد با تابع fft.ifftshift
بعد با استفاده از فوریه بدست آمده و تابع fft.ifft2 تصویر فیلتر شده را بدست می آوریم.



برای استفاده از canny، اول عکس را میدهیم بعد هم treshhold اول و دوم این یعنی جاهایی که مقدار پیکسل کمتر از trshhold1 باشد، جز لبه ها در نظر گرفته نمیشود و اگر بیشتتر از treshhold2 باشد، یعنی یک strong edge است و در واقع احتمال لبه بودنش زیاد است، جاهایی که بین این treshhold1 و treshhold2 هستند، اگر به یک strong edge بچسبند جز لبه محاسبه میشوند.خروجی:

ج) در این قسمت مانند سوال 1 با توجه به توابعی که تعریف کرده بودیم مقدار گرادیان و magnitude را محاسبه می کنیم.

.3

الف) در فیلتر های پایین گذر تاثیر فرکانس های بالا کاهش پیدا کرده یا از بین میرود. بنابراین برای تار و هموار و از بین بردن نویزهای تصویر به کار میرود. مثال : فیلتر های میانه و گاوسی و میانگین.

در فیلتر های بالا گذر تاثیر فرکانس های پایین کاهش پیدا کرده و یا از بین میرود. بنابراین برای شارپ شدن تصویر، و بهتر شدن لبه استفاده میشود. مثال : فیلتره های لاپلاسی، سوبل.

ب) چون لبه های برجسته شده اند پس فیلتر بالاگذر زده ایم.

ج) نویز ضرب شونده یعنی نویزی که در تصویر اصلی ضرب میشود و ممکن است در اثر تغییرات نوری یا نوع سنسور باشد. این نوع از نویز میتواند تصویر را به شکل های مختلفی تغییر دهد بنابراین uniform نیست. از مثال های این نوع نویز میتوان به نویز دانه و نقطه ای اشاره کرد. برای از بین بردن این نوع نویز ها میتوان از فیلتر non local means filter استفاده کرد که در آن تصویر به بخش های کوچک تقسیم شده و با شباهت این قسمت ها نویز را حذف میکند. از راه دیگر میتوان به فیلترینگ متقارن هم اشاره کرد.

در نویز جمع شونده یک مقدار رندوم به تصویر اشافه میشود و uniform است.از مثال های این نوع نویز میتوان به نویز سفید، گاوسی اشاره کرد. برای از بین بردن این نوع نویز ها میتوان از فیلتر میانه و میانگین و گاوسی استفاده کرد.

د) نویز نمک و فلفل زمانی ایجاد میشود که به صورت رندوم و بدون الگو پیکسل ها یا 255 یا 0 بشوند. این نویز میتواند در اثر خرابی سنسورها ایجاد شود. برای حذف این نویز میتوان از فیلتر میانه استفاده کرد.

.4

الف)

$$F(u,v) = \sum_0^{M-1}\sum_0^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$$F(0,0) = \sum_0^{M-1}\sum_0^{N-1} f(x,y)e^{0} = \text{مجموع مقادیر تصویر}$$

ب)

$$F(0,0) = 9$$

$$F(0,1) = 4e^{-J2\pi(0)} + 0 + 3e^{-J2\pi(\frac{1}{2})} + 2e^{-2J\pi(\frac{1}{2})} = -1$$

$$F(1,0) = 4 + 0 + 3e^{0} + 2e^{-2\pi(\frac{1}{2})} = 5$$

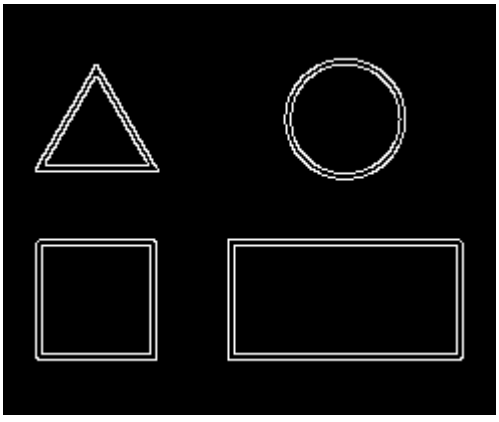
$$F(1,1) = 4 + 0 + 3\,e^{-J3\pi(\frac{1}{2})} + 3e^{-J2\pi(1)} = 3$$

9	5
-1	3

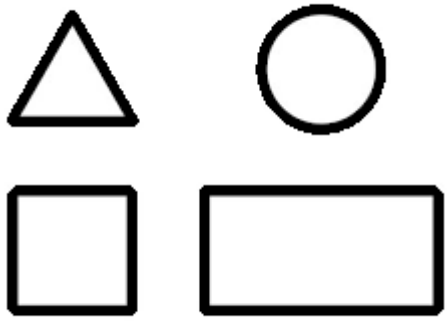
5)

الف) ابتدا تصویر را خوانده و به gray scale میریم.

ب) سپس با استفاده از canny لبه ها را پیدا میکنیم خروجی:



بعد با استفاده از تابع findContours لبه های بسته را پیدا میکنیم که در ارگومان اول عکس ورودی را گرفته درارگومان دوم mode را مشخص میکنیم که در این مثال RETR_EXTERNAL میگذاریم که یعنی لبه های خارجی را پیدا میکند در ارگومان بعدی هم نحوه ی تخمین لبه ها را مشخص میکنیم که در این مثال از CHAIN_APPROX_SIMPLE استفاده میکنیم تا تنها نقاط پایانی خطوط را خروجی بدهد. بعد با استفاده از تابع drawContours خطوط موجود در شکل را میکشیم که اول عکس ورودی را داده بعد contour ها و -1 میدهیم تا همه ی خطوط را بکشد و در ارگومان های بعدی رنگ و فونت را مشخص میکنیم. خروجی:



در آخر با تابع approxPolyDP که به ارگومان آن contours ، آرگومان دوم آن دقت تقریب را مشخص میکند و در اینجا ما طول خم را بدست می آوریم با تابع arclength که آرگومان اول contour ها و آرگومان دوم آن نشان دهنده ی بسته بودن خطوط است. و درصد خطا را 4 در نظر میگیریم. آرگومان سوم دو تابع approxPolyDP هم نشان دهنده ی بسته بودن خطوط است.

بعد به اضای هر نقاطی که در محاسبه کردیم با تابع drawMarker روی تصویر اصلی آن را مشخص میکنیم و بعد اگر 3 نقطه داشتیم یعنی مثلث اگر 4 نقطه داشتیم یا مربع یا مستطیل و اگر 6 و بیشتر نقطه داشتیم دایره داریم.

در تصویر با استفاده از تابع putText نام شکل را می نویسیم.و برای تشخیص مستطیل و مربع از تابع boundingRect استفاده میکنیم که نقاط را داده و به ما طول و عرض را میدهد اگر برابر بودند یعنی مربع داریم در غیر این صورت مستطیل داریم.

.6

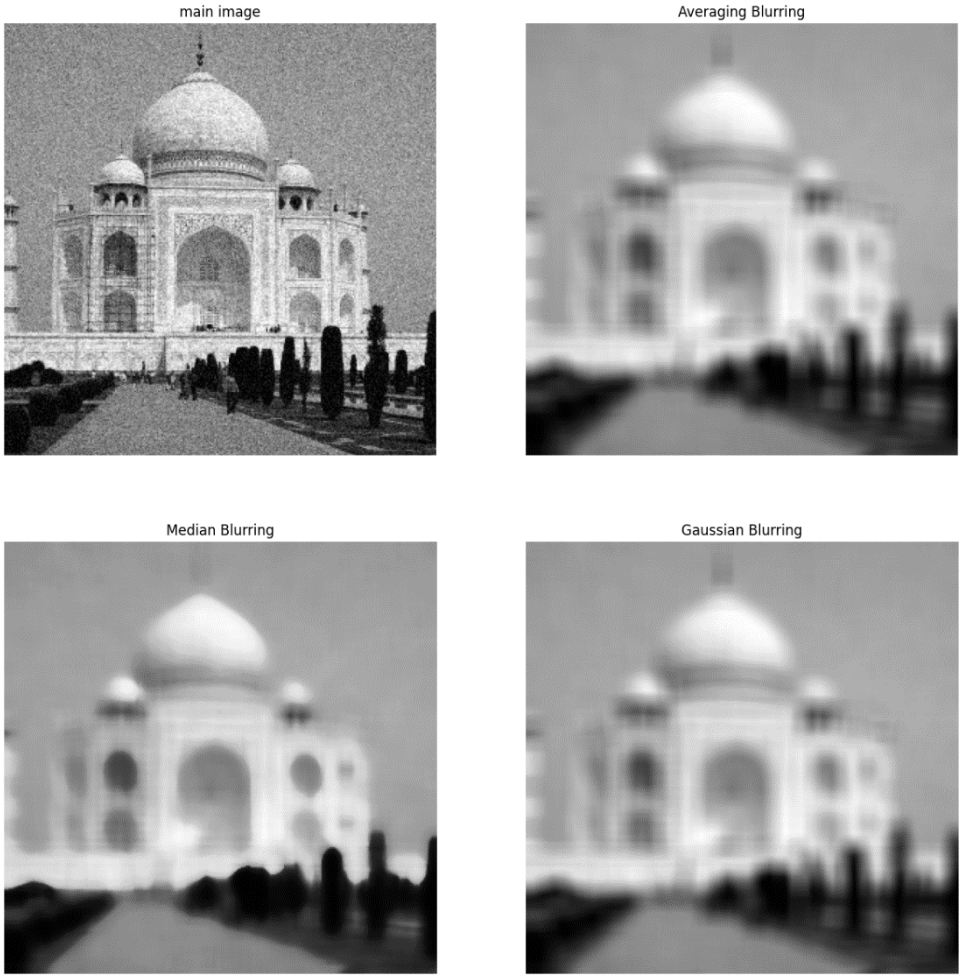
الف) ابتدا برای تابع reflect101 با تابع pad در آرگومان اول عکس ورودی در آگومان دوم طول و عرض های پدینگ و آرگومان سوم mode را میدهد که نوع پدینگ را مشخص میکند.

در تابع Average_Blurring عکس را با یک کرنل که هر عضو آن 1/n*n است و n همان طول کرنل است کانوالو کنیم و باعث میشود که در واقع میانگین بگیریم.

در Median_blurring هم هر بخش از عکس را که اندازه ی کرنل است میانه میگیریم (مانند کانوالو کردن با این تفاوت که میانه را هر بار می گیریم)

با افزایش سایز کرنل هموارسازی افزایش میابد و جزییات تصویر از بین میرود. اما با کاهش اندازه کرنل جزییات بهتر دیده میشود.

خروجی :



ب) یک فیلتر غیرخطی است که برای smooth کردن تصویر در حالی که حاشیه‌ها را حفظ می‌کند استفاده می‌شود. این فیلتر فاصله مکانی و تفاوت شدت بین پیکسل‌ها را در هنگام فیلتر کردن تصویر در نظر می‌گیرد. این فیلتر در نظر می‌گیرد که هر پیکسل خروجی، میانگین وزن‌داری از پیکسل‌های همسایه‌اش است که وزن هر پیکسل براساس فاصله مکانی و تفاوت شدت بین آن‌ها تعیین می‌شود. همچنین این فیلتر ترکیبی از فیلتر میانه و گاوسی است.

فرمول و پارامتر ها:

$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

I^{filtered} که نشان دهنده ی عکس فیلتر شده است.

I که عکس اولیه است.

x که نشان دهنده ی مختصاتی از عکس است که قرار است فیلتر شود.

Ω پنجره ای است که مرکز آن x است.

f_r تابع کرنل برای نرم‌کردن اختلافات درجات خاکستری

g_s تابع کرنل برای نرم‌کردن اختلافات موقعیتی بین پیکسل‌ها

ب) در این قسمت فرمول زیر را پس از پدینگ زدن پیاده سازی کرده ایم:

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|I(i,j) - I(k,l)\|^2}{2\sigma_r^2}\right),$$

$$I_D(i,j) = \frac{\sum_{k,l} I(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)},$$