به نام خدا

گزارش تمرین سری دوم درس بینایی کامپیوتر
نام مدرس: دکتر محمدرضا محمدی

فرزان رحمانی ۹۹۵۲۱۲۷۱

مکمل توضیحات داخل نوت بوک و کامنت های کد وجود دارد.

سوال ۱

الف) با مراجعه به اسلاید ها فیلتر مطلوب را پیدا کردم و تابع هایی که داخل نوت بوک داده شده بود را کامل کردم و در نهایت با توجه به فرمول اسلاید های درس اندازه و جهت ماتریس را محاسبه کردم و نشان دادم. برای بعضی از توابع مانند کشیدن نمودار از سرچ در گوگل استفاده کردم.

ب) فیلتر گوسی را دخال اسلاید درس یافته و تعریف کردیم و مانند قسمت الف پیاده سازی کردم.

ج)

Python:

cv.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]]) -> dst

Parameters

src	input image.
dst	output image of the same size and the same number of channels as src .
ddepth	output image depth, see **combinations**; in the case of 8-bit input images it will result in truncated derivatives.
dx	order of the derivative x.
dy	order of the derivative y.
ksize	size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
scale	optional scale factor for the computed derivative values; by default, no scaling is applied (see **getDerivKernels** for details).
delta	optional delta value that is added to the results prior to storing them in dst.
borderType	pixel extrapolation method, see **BorderTypes**. **BORDER_WRAP** is not supported.

برای اجرای عملگر Sobel در OpenCV، ابتدا تصویر مورد نظر را بارگیری می کنیم. سپس با استفاده از تابع cv2.Sobel، گرادیان های افقی و عمودی را محاسبه می کنیم. پارامترهای تابع Sobel شامل تصویر واقعی، نوع داده خروجی، ترکیب گرادیان (افقی یا عمودی) و سایز کرنل هستند.(پارامتر های دیگری هم دارد که از آن ها استفاده نکردیم ولی در بالا توضیح آن ها آمده است.) به عنوان مثال:

```

import cv2

img = cv2.imread('afshin.jpg',0)

grad_x = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)

grad_y = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=3)
```

در اینجا، تصویر ورودی با استفاده از تابع imread به صورت خاکستری بارگیری شده است. سپس با استفاده از تابع Sobel، گرادیان های افقی و عمودی با سایز کرنل ۳*۳ محاسبه شده است. پارامتر CV_64F نوع داده خروجی را نشان میدهد که در اینجا ۶۴ بیت اعشاری است.

Sobel Operator

```
Mat grad_x, grad_y;
Mat abs_grad_x, abs_grad_y;

Sobel(src_gray, grad_x, ddepth, 1, 0, ksize, scale, delta, BORDER_DEFAULT);

Sobel(src_gray, grad_y, ddepth, 0, 1, ksize, scale, delta, BORDER_DEFAULT);
```

- We calculate the "derivatives" in *x* and *y* directions. For this, we use the function **Sobel()** as shown below: The function takes the following arguments:
  - *src_gray*: In our example, the input image. Here it is *CV_8U*
  - *grad_x* / *grad_y* : The output image.
  - *ddepth*: The depth of the output image. We set it to *CV_16S* to avoid overflow.
  - *x_order*: The order of the derivative in **x** direction.
  - *y_order*: The order of the derivative in **y** direction.
  - *scale*, *delta* and *BORDER_DEFAULT*: We use default values.

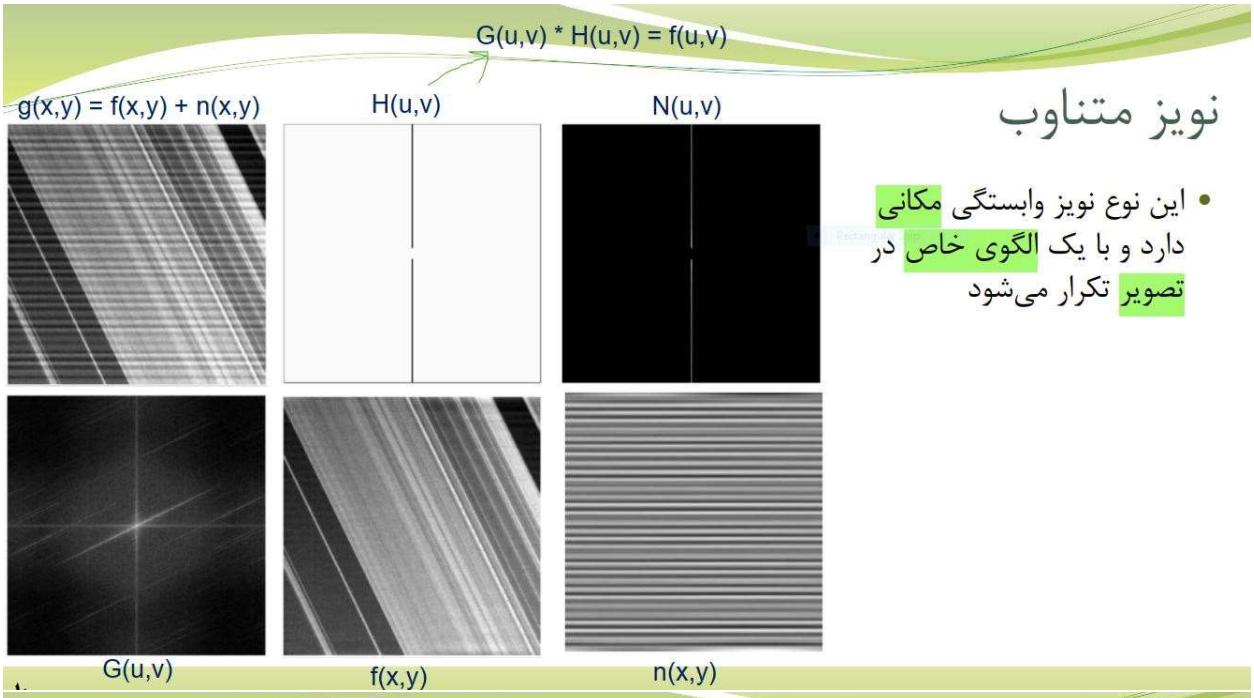
Notice that to calculate the gradient in *x* direction we use: $x_{order} = 1$ and $y_{order} = 0$. We do analogously for the *y* direction.

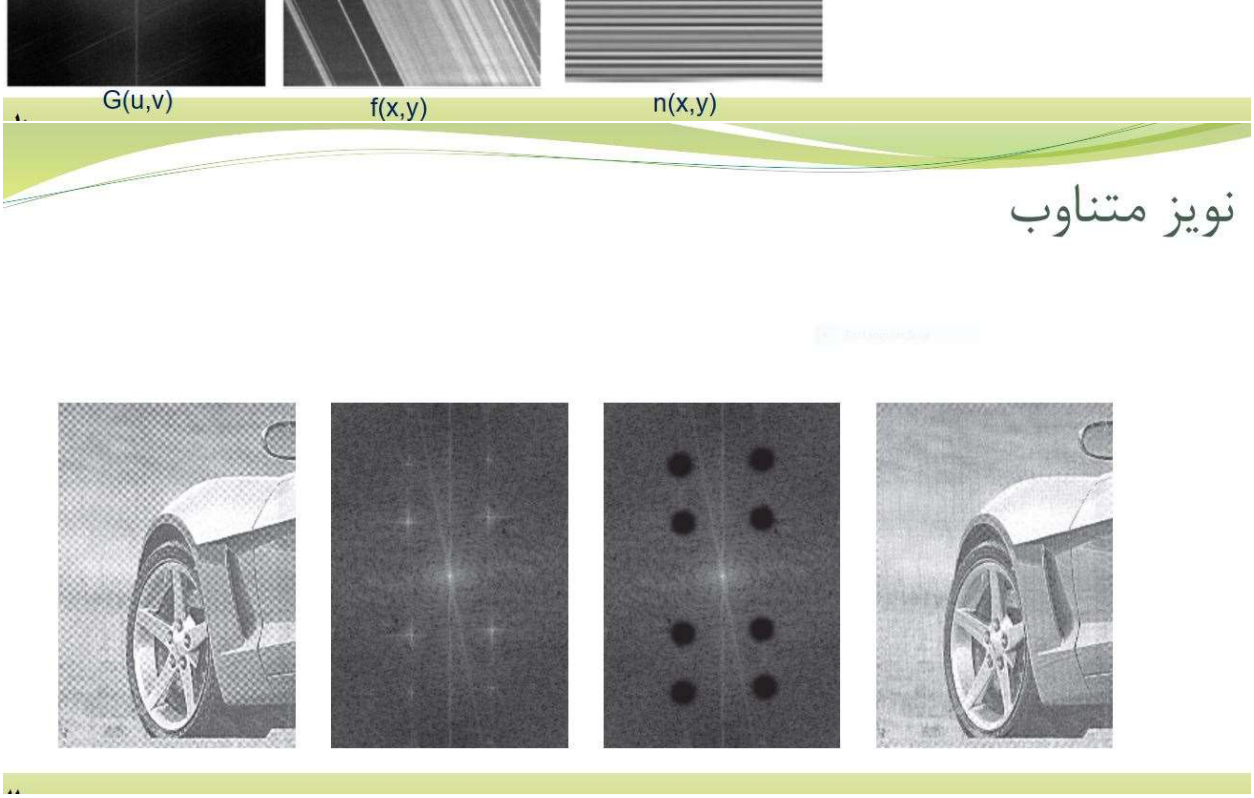
توضیح کامل تر در لینک زیر:

https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gacea54f142e81b6758cb6f375ce782c8d

سوال ۲

الف)





منابع:

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

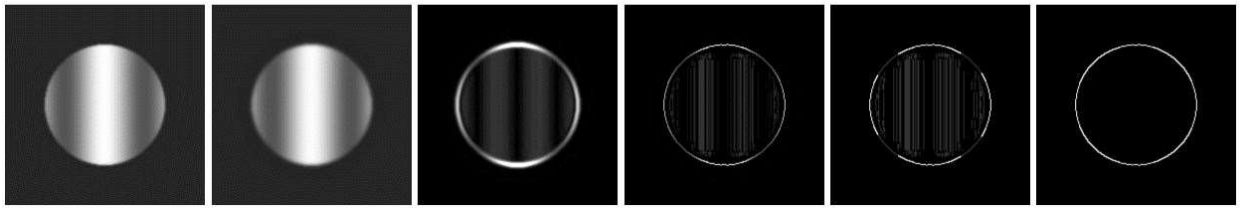
https://mathematica.stackexchange.com/questions/110914/how-to-use-2d-fourier-analysis-to-clean-the-noise-in-an-image

ابتدا تصویر saffron.jpg را با استفاده از تابع imread میخوانیم و سپس تبدیل FFT آن را با استفاده از تابع fft2 بدست می آوریم. روش کار با آن را با سرچ پیدا کردیم. سپس با رسم مقدار مطلق تبدیل FFT(شیفت هم می دهیم تا فرکانس های پایین تصویر به وسط بیایند)، نویزهای موجود در تصویر را شناسایی کرده (با توجه به اینکه زورنه های متناوب فرکانس بالایی دارند همان طور که در نوت بوک دیده می شود فرکانس های بالا را mask کردیم) و با حذف آن‌ها و شیفت به حالت اولیه و پس از آن تبدیل عکوس فوریه IFFT تصویر پاک شده را بدست می‌آوریم و پس از نرمالایز کردن آن را نمایش می دهیم. همان طور که می بینیم با سه روش مختلف فیلتر کردن در فضای فرکانس را انجام داده ایم.

ب)

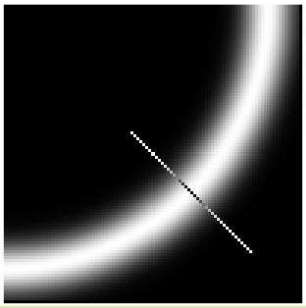
لبه‌یاب Canny

- یکی از پرکاربردترین و موفق‌ترین روش‌های لبه‌یابی است که از ۴ گام اساسی تشکیل می‌شود:
  - هموار کردن تصویر با استفاده از فیلتر گاوسی smooth
  - محاسبه گرادیان
  - حذف مقادیر غیربیشینه
  - آستانه‌گذاری دو مرحله‌ای

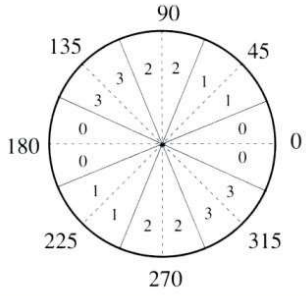


حذف مقادیر غیربیشینه

- هر پیکسل که در راستای گرادیان خود دارای مقدار غیربیشینه باشد حذف می‌شود
- جهت گرادیان به ۴ گروه تقسیم می‌شود و همسایگی ۳×۳ است

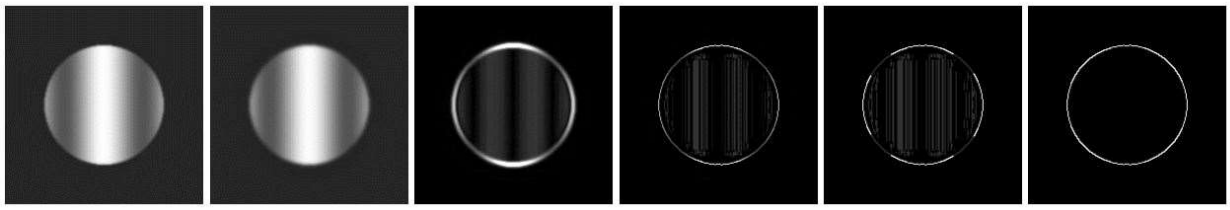


0	0	0	0	1	2	1	3
0	0	0	1	2	1	3	1
0	0	2	1	2	1	1	0
0	1	3	2	1	1	0	0
0	3	2	1	0	0	0	0
2	3	2	0	0	1	0	1
2	3	2	0	1	0	2	0



آستانه‌گذاری دوسطحی

- هر پیکسلی که اندازه گرادیان آن کوچکتر از T_1 باشد به عنوان غیرلبه معرفی می‌شود
- هر پیکسلی که اندازه گرادیان آن بزرگتر از T_2 باشد به عنوان لبه معرفی می‌شود
- پیکسل‌هایی که اندازه گرادیان آنها بین T_1 و T_2 باشد تنها در صورتی به عنوان لبه معرفی می‌شوند که به یک پیکسل لبه به صورت مستقیم یا از طریق پیکسل‌هایی که اندازه گرادیان آنها بین T_1 و T_2 است متصل باشند



برای لبه‌یابی از تابع Canny استفاده می‌کنیم. روش کار با آن را با سرچ پیدا کردیم. این تابع دارای سه پارامتر است: عکس مرود نظر، حداقل و حداکثر مقدار threshold. حداقل و حداکثر threshold برای تشخیص لبه‌ها در مرحله آستانه گذاری دو مرحله ای به کار می روند. پارامتر threshold را با تغییر دادن مقادیر آن، بهینه‌سازی کرده‌ایم. پارامتر های دیگری هم دارد که از آن ها استفاده نکردیم ولی در زیر توضیح آن ها آمده است.

Python:

```
cv.Canny( image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]] ) -> edges
cv.Canny( dx, dy, threshold1, threshold2[, edges[, L2gradient]]                  ) -> edges
```

Parameters

image	8-bit input image.
edges	output edge map; single channels 8-bit image, which has the same size as image .
threshold1	first threshold for the hysteresis procedure.
threshold2	second threshold for the hysteresis procedure.
apertureSize	aperture size for the Sobel operator.
L2gradient	a flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default L_1 norm $= \|dI/dx\| + \|dI/dy\|$ is enough (L2gradient=false).

overload دیگر:

Python:

cv.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]) -> edges

cv.Canny(dx, dy, threshold1, threshold2[, edges[, L2gradient]]) -> edges

Parameters

dx	16-bit x derivative of input image (CV_16SC1 or CV_16SC3).
dy	16-bit y derivative of input image (same type as dx).
edges	output edge map; single channels 8-bit image, which has the same size as image .
threshold1	first threshold for the hysteresis procedure.
threshold2	second threshold for the hysteresis procedure.
L2gradient	a flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default L_1 norm $= \|dI/dx\| + \|dI/dy\|$ is enough (L2gradient=false).

لینک های توضیح کامل:

https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de

https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

ج) برای بدست آوردن گرادیان از تابع cv2.Sobel استفاده می کنیم. پارامتر های این تابع به شرح زیر هستند:

- ddepth: عمق تصویر خروجی. در اینجا برابر با -۱ قرار داده شده است که به معنای استفاده از عمق تصویر ورودی است.

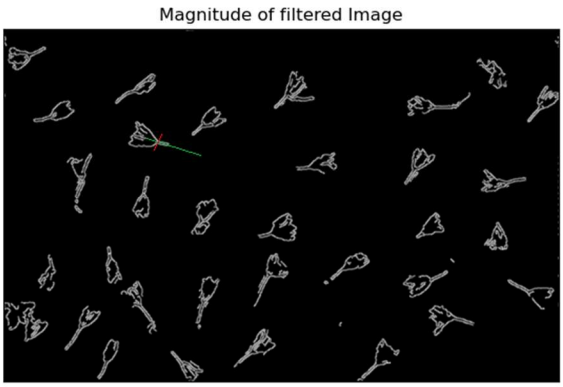
- dx و dy: مشخص می کند که گرادیان در جهت x یا y باید محاسبه شود. در اینجا dx=1 و dy=0 قرار داده شده است که به معنای محاسبه گرادیان در جهت x است.

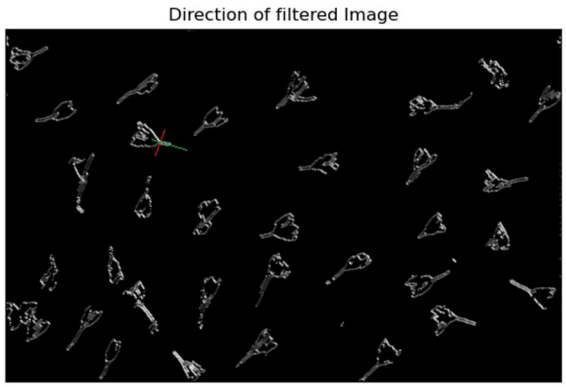
- ksize: سایز هسته فیلتر سوبل. در اینجا برابر با ۳ قرار داده شده است.

پس از به دست آوردن گرادیان ها از فرمول های اسلاید برای محاسبه جهت و اندازه گرادیان می توان استفاده کرد.

برای بدست آوردن نقطه برش ساقه از گلبرگ، می توان از روش های مختلفی استفاده کرد. یکی از روش های معمول، استفاده از تبدیل هاف (Hough Transform) است که با استفاده از آن می توان خطوط را در تصویر پیدا کرد. در اینجا به دلیل عدم وجود زاویه خاص بین ساقه و گلبرگ، روش دیگری برای بدست آوردن نقطه برش پیشنهاد نمی شود.

د) برای بدست آوردن نقطه برش ساقه از گلبرگ، میتوان با استفاده از جهت گرادیانهای بدست آمده، خطوطی را که متناظر با ادامه ساقه است، تشخیص داد. سپس با استفاده از این خطوط، نقطهای که برش ساقه را نشان میدهد را تعیین کرد. این روش با استفاده از الگوریتم های تشخیص خط و روشهای تخمین خط مانند رگرسیون خطی، قابل انجام است. همچنین با مقایسه‌ی گرادیان‌های گلبرگ و گرادیان‌های ساقه و تغییر جهتی هم که دارند می‌توان یک روش و الگوریتمی برای پیدا کردن نقطه برش ساقه از گلبرگ پیدا کرد.





با استفاده از جهت گرادیان های بدست آمده، می توانیم نقطه برش ساقه از گلبرگ را بیابیم. برای این کار، می توانیم از روش زیر استفاده کنیم:

۱. با استفاده از تابع cv2.Sobel، گرادیان های x و y را بدست آورده و سپس ماژول گرادیان را با استفاده از تابع cv2.magnitude محاسبه می کنیم.

۲. سپس جهت گرادیان ها را با استفاده از تابع cv2.phase و arctan2 به دست می آوریم.

۳. در نهایت، با استفاده از جهت گرادیان های بدست آمده، محاسبات لازم را برای پیدا کردن نقطه برش ساقه انجام داد.

برای پیدا کردن نقطه برش ساقه، می توان به صورت زیر عمل کرد:

۱. با استفاده از تابع cv2.inRange، پس زمینه عکس را حذف کنید.

۲. سپس با استفاده از تابع cv2.findContours، لبه های تصویر را پیدا کنید.

۳. سپس با استفاده از تابع cv2.fitLine، خطی را برای لبه های پیدا شده بدست آورید.

۴. با استفاده از خط بدست آمده، نقطه برش ساقه را محاسبه کنید.

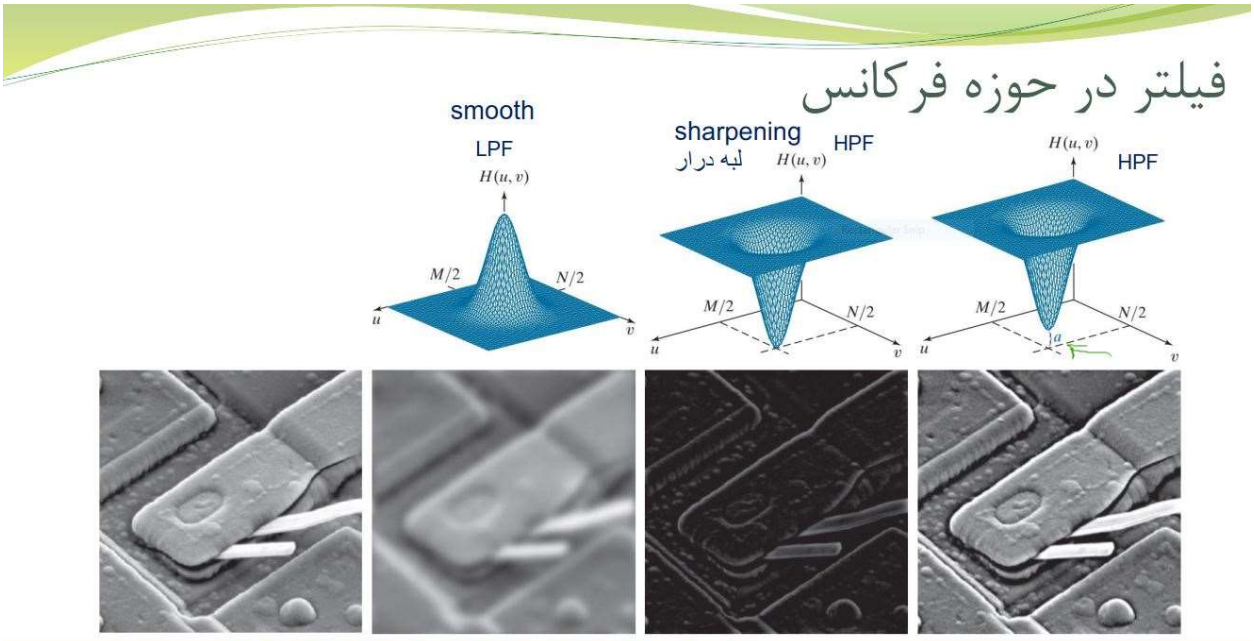
توجه: این روش فقط در صورتی کارآمد است که ساقه گلبرگ به صورت عمودی و یکنواخت باشد. در غیر این صورت، نتایج به دقت کافی نخواهند داشت.

سوال ۳

الف)

- فیلتر پایین گذر(LPF): این فیلتر تمرکز خود را برای حذف فرکانس‌های بالا یا نویز های بالاتر از یک حداکثر مشخص می‌کند. با داشتن یک فیلتر پایین گذر، فرکانس‌های بالاتر از حداکثر مشخص، حذف می‌شوند و فقط فرکانس‌های پایین تر و با تاثیر کمتر بر تصویر باقی می‌مانند. این فیلتر در پردازش تصویر به منظور حذف نویز و بدون کیفیت‌کردن(smoothing) تصویر کاربرد دارد.
- فیلتر بالاگذر(HPF): در برخی موارد، فرکانس‌های پایین تر از یک حداقل مشخص، نامطلوب هستند و فیلتر بالاگذر برای حذف آنها استفاده می‌شود. این فیلتر با حفظ فرکانس‌های بالاتر، فرکانس‌های پایینتر را حذف می‌کند. فیلتر بالاگذر در پردازش تصویر به منظور تشدید لبه‌ها و حذف نویز(sharpening) استفاده می‌شود.

از این دو نوع فیلتر در پردازش تصویر برای بهبود کیفیت تصویر، کاهش نویز و تشدید لبه‌ها استفاده می‌شود. همچنین، در پردازش سیگنال‌های دیجیتال نیز، فیلتر پایینگذر و بالاگذر برای تمیز کردن سیگنال از نویز و تشدید وضوح آنها استفاده می‌شود.



HPF and LPF stand for High Pass Filter and Low Pass Filter.

When it comes to processing signals, filtering is a key aspect that helps in shaping the characteristics of the signal. Low-pass and high-pass filters are two commonly used types of filters that work in opposite ways to filter signals. Low-pass filters, as the name suggests, allow low-frequency signals to pass through while attenuating high-frequency signals. On the other hand, high-pass filters allow high-frequency signals to pass through while attenuating low-frequency signals.

Low pass filter: Low pass filter is the type of frequency domain filter that is used for smoothing the image. It attenuates the high-frequency components and preserves the low-frequency components.

High pass filter: High pass filter is the type of frequency domain filter that is used for sharpening the image. It attenuates the low-frequency components and preserves the high-frequency components.

Applications of Low-Pass and High-Pass Filters:

Low-pass and high-pass filters find applications in a variety of fields including audio processing, image processing, communication systems, and biomedical signal processing. Understanding the characteristics of these filters and their applications is essential for signal-processing engineers and researchers.

similarities between the two filters:

- Both filters are used to remove unwanted frequency components from a signal.
- Both filters have a cut-off frequency, which is the frequency at which the filter begins to attenuate the signal.
- The steepness of the cut-off slope depends on the order of the filter, with higher-order filters having steeper slopes.
- Both filters can introduce phase shifts, which can affect the time-domain characteristics of the signal.
- Both filters are used in a variety of applications including audio processing, image processing, communication systems, and biomedical signal processing.

Difference between Low pass filter and High pass filter:

Low pass filter	High pass filter
It is used for smoothing the image.	It is used for sharpening the image.
It attenuates the high frequency.	It attenuates the low frequency.
Low frequency is preserved in it.	High frequency is preserved in it.
It allows the frequencies below cut off frequency to pass through it.	It allows the frequencies above cut off frequency to pass through it.
It consists of resistor that is followed by capacitor.	It consists of capacitor that is followed by a resistor.
It helps in removal of aliasing effect.	It helps in removal of noise.
$G(u, v) = H(u, v) . F(u, v)$	$H(u, v) = 1 – H'(u, v)$

Conclusion: *Low-pass and high-pass filters are two fundamental types of filters used in signal processing. These filters have opposite characteristics and are used to filter out unwanted frequency components from a signal. Understanding the characteristics of these filters and their applications can help in designing effective signal-processing systems.*

منابع:

https://www.geeksforgeeks.org/difference-between-low-pass-filter-and-high-pass-filter/

https://www.tutorialspoint.com/dip/high_pass_vs_low_pass_filters.htm

ب) فیلتر بالا گذر (HPF) چرا که لبه های تصویر را آشکار کرده است و تصویر sharp تر شده است و فرکانس های بالای آن تقویت شده اند و فرکانس های پایین آن تضعیف.

ج)

Additive noise and multiplicative noise are just models of how noise corrupts our data.

One very common model is the additive noise model, where we have our 'true' data vector s[n], (which we are trying to ascertain), being corrupted by a noise vector, v[n]. What we are given, is x[n], where:

$$x[n]=s[n]+v[n]$$

This is called the 'additive' noise model, because as you can see, noise is added to our true signal, giving us x[n]. There are many ways in which we can remove this noise in an additive model, such as filtering, (which is a form of averaging if you like). This is a very common type of noise model. If I am talking, and you are talking over me, your voice can be modeled as additive to my voice. You voice would be additive 'noise', that is added to my voice, which in this vain example is the 'true' signal, (although this would be disputed in a heated argument between two people). In a more objective example, thermal noise from a microphone's electronics can also be modeled as an additive noise, which is added to a voice signal that it has received. Many things can be modeled as additive types of noises.

Multiplicative noise on the other hand is still a model, but in this model our true data samples are being multiplied by noise samples, like so:

$$x[n]=s[n]v[n]$$

One common way to remove multiplicative noise is to actually transform it into an additive model, and then apply everything we know from the additive noise reduction field. We can do this easily via taking the logarithm of the signal, filtering, then inverse log transform. Thus, we can do:

$$x[n] = s[n]v[n] \implies \log(x[n]) = \log(s[n]v[n]) = \log(s[n]) + \log(v[n])$$

At this point we have an additive model once more. Now, we can filter as we normally might, to remove or reduce log(v[n]), and then simply take log^{-1} of the result, yielding an estimate of s[n], our true signal.
One example of multiplicative noise is in illumination differences between images, which is solved in the above way. The non-uniform illumination across an image can be modeled as a pixel-by-pixel multiplication of the image by an illumination mask. This is also known as homomorphic filtering by the way. Anytime you can model a corrupting phenomenon as *multiplying* your clean data signal, you can use this multiplicative model.

نویز جمع شونده و ضرب شونده دو نوع نویز شناخته شده در پردازش سیگنال هستند. در ادامه توضیحات بیشتری درباره هر نوع نویز و راهکارهای حذف آنها ارائه خواهیم داد:

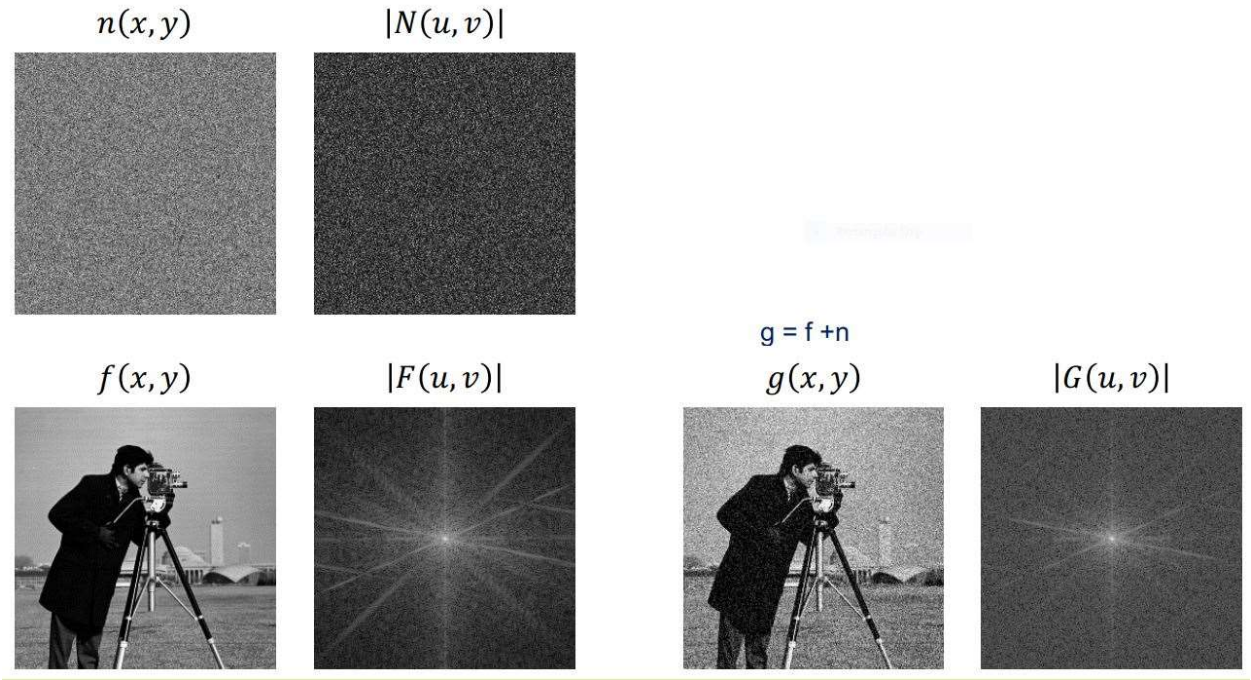
- نویز جمع شونده :(Additive Noise) این نوع نویز به سیگنال اصلی اضافه می‌شود و معمولا به صورت ناشی از خطاهای مربوط به محیط یا سیستم است. مثال‌هایی از نویز جمعشونده عبارتند از نویز امواج رادیویی، نویز دمایی، نویز هوا و نویز الکتریکی. برای حذف نویز جمعشونده می‌توان از روش‌هایی مانند فیلترینگ و استفاده از الگوریتم‌های پردازش سیگنال استفاده کرد.

کاهش نویز

- مدل نویز جمع‌شونده:

$$g(x,y) = f(x,y) + n(x,y)$$

- دستگاه‌های تصویربرداری مختلف دارای مدل‌های نویز متفاوتی هستند
- نویز گاوسی متداول‌ترین نویز است



- نویز ضرب شونده :(Multiplicative Noise) در این نوع نویز، میزان نویز با مقدار سیگنال اصلی متغیر است و برخلاف نویز جمعشونده که به سیگنال اضافه می‌شود، به صورت ضرب در سیگنال اعمال می‌شود. این نوع نویز معمولا به صورت ناشی از خطاهای سیستمی به وجود می‌آید. مثال‌هایی از نویز ضربشونده عبارتند از نویز روی مدارهای بالابر، نویز روی تصاویر ناشی از نویزسازی حسگر، و نویز در موجودیت‌های فیزیکی مانند جرم و تنش. برای حذف نویز ضربشونده می‌توان از روش‌هایی مانند مدل‌سازی آماری سیگنال و بهینه‌سازی پارامترهای سیستم استفاده کرد.

در کل، روش‌های مختلفی برای حذف هر دو نوع نویز وجود دارد که از آنها می‌توان به عنوان راهکارهایی برای حذف هر نوع نویز استفاده کرد. به عنوان مثال، برای حذف نویز جمعشونده، می‌توان از فیلترهای پایین گذر استفاده کرد که فرکانس‌های بالای سیگنال را حذف می‌کنند و نویز را کاهش می‌دهند. همچنین، الگوریتم‌های پردازش سیگنال مانند رگرسیون و کاهش بعد نیز می‌توانند به خوبی برای حذف نویز جمعشونده عمل کنند.

در مورد حذف نویز ضربشونده، روش‌های مختلفی از جمله استفاده از الگوریتم‌های پردازش تصویر مانند فیلترینگ و پردازش فرکانسی، مدل‌سازی آماری و بهینه‌سازی پارامترهای سیستم وجود دارند. به طور کلی، برای حذف هر دو نوع نویز، می‌توان از ترکیبی از روش‌های مختلف استفاده کرد تا بهترین نتیجه را به دست آورد.

در هر صورت، حذف نویز از سیگنال‌های پردازش تصویر مهم است و در بسیاری از برنامه‌های کاربردی به عنوان یک گام ضروری در پردازش تصویر استفاده می‌شود.

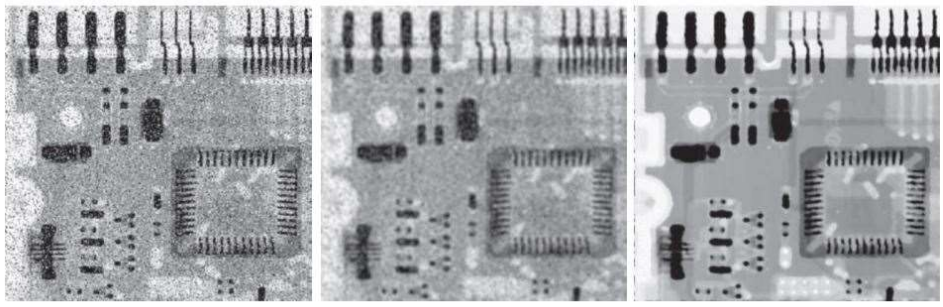
منابع:

https://chat.openai.com/chat/
https://dsp.stackexchange.com/questions/13895/which-domain-used-for-denoising-additive-and-multiplicative-noises

د)

نویز نمک و فلفل

- این نوع نویز برخلاف نویزهای بررسی شده، جمع‌شونده نیست
- فیلترهای هموارساز خطی نمی‌توانند این نوع نویز را به خوبی برطرف کنند
- فیلترهای مرتبه‌ای می‌توانند عملکرد بهتری داشته باشند



فیلتر میانه Median filter

- فیلتر میانه یک فیلتر غیرخطی است که بر اساس مرتب‌سازی پیکسل‌های درون کرنل و جایگزینی مقدار میانه بجای پیکسل مرکزی عمل می‌کند

	10	11	15	8	7
	7	10	50	12	10
	9	14	12	13	11
	10	16	14	15	14
	8	11	10	10	9

		11	12	12		
		12	14	13		
		11	13	12		

In salt and pepper noise (sparse light and dark disturbances),[33] also known as impulse noise,[34] pixels in the image are very different in color or intensity from their surrounding pixels; the defining characteristic is that the value of a noisy pixel bears no relation to the color of surrounding pixels. Generally this type of noise will only affect a small number of image pixels. When viewed, the image contains dark and white dots, hence the term salt and pepper noise. Typical sources include flecks of dust inside the camera and overheated or faulty CCD elements.

Salt-and-pepper noise, also known as **impulse noise**, is a form of noise sometimes seen on digital images. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels.

An effective noise reduction method for this type of noise is a median filter[1] or a morphological filter.[2] For reducing either salt noise or pepper noise, but not both, a contraharmonic mean filter can be effective.[3]

نویز نمک و فلفل (salt and pepper noise) یکی از انواع نویزها است که در تصاویر به طور تصادفی در برخی از پیکسل‌ها به وجود می‌آید و باعث افت کیفیت و روند تصویر می‌شود. در این نوع نویز، برخی از پیکسل‌های تصویر به صورت تصادفی سفید یا سیاه می‌شوند.

برای حذف نویز نمک و فلفل، می‌توان از فیلتر میانه‌گیر (median filter) استفاده کرد. این فیلتر برای هر پیکسل، مقدار میانه را پس از مرتب سازی از مجموعه‌ای از پیکسل‌های اطراف آن پیشنهاد می‌دهد. با این کار، پیکسل‌های مشکل‌دار که با نویز نمک و فلفل دچار شده‌اند، با پیکسل‌های دیگر جایگزین می‌شوند و تصویر بدون نویز تولید می‌شود.

این فیلتر به دلیل سادگی و قابلیت اعمال به تصاویر با ابعاد مختلف، به عنوان یک روش موثر برای حذف نویز نمک و فلفل شناخته شده است. با این حال، برای حذف نویز در تصاویر با فرکانس‌های بالاتر، مانند تصاویر پرتره، ممکن است نیاز به استفاده از فیلترهای پیچیده‌تر باشد.

منابع:

https://en.wikipedia.org/wiki/Noise_reduction
https://en.wikipedia.org/wiki/Salt-and-pepper_noise
https://chat.openai.com/chat/

سوال ۴

در حل این سوال از روابط زیر استفاده میکنیم:

تبدیل فوریه گسسته 2D

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{+j2\pi(ux/M+vy/N)}$$

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)}$$

$$Magnitude = |F(u,v)| = \sqrt{Re^2(u,v) + Im^2(u,v)}$$

$$Phase = \varphi(u,v) = atan2\big(Im(u,v), Re(u,v)\big)$$

Subject:( ) Year: Month: Day:

۴ـ الف) رابطه تبدیل فوریه: $F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$

نقطه مبدا تبدیل فوریه تصویر ($u=0$, $v=0$): $F(0,0)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{0}{M}+\frac{0}{N}\right)}$

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

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

اگر $F(0,0)$ را بر MN تقسیم کنیم (نرمال سازی) نشان دهنده میانگین مقادیر پیکسل های تصویر است.

$$\frac{F(0,0)}{MN} = \text{میانگین مقادیر تصویر}$$

ب)

$$f=\begin{bmatrix}۴ & ۰\\ ۳ & ۲\end{bmatrix} \quad F=\begin{bmatrix}۹ & ۵\\ -۱ & ۳\end{bmatrix}$$

x ↓ y ، u → ، v ↓

$M=۲ \quad N=۲$

$$f(x,y)=\frac{۱}{۲\times۲}\sum_{u=0}^{۱}\sum_{v=0}^{۱} F(u,v)\,e^{+j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

TANIN

Subject:( ) Year: Month: Day:

$$F(0,0)=\sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)\underbrace{e^{-j2\pi(0+0)}}_{1}=\sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)$$

$$F(0,0)=4+0+2+2=9$$

$$F(1,0)=\sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)\underbrace{e^{-j2\pi\left(\frac{x}{2}+0\right)}}_{e^{-j\pi x}}=f(0,0)e^{0}+$$

$$f(1,0)e^{-j\pi}+f(0,1)e^{0}+f(1,1)e^{-j\pi}=4+2-2=4$$

$$F(0,1)=\sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)\underbrace{e^{-j2\pi\left(0+\frac{y}{2}\right)}}_{e^{-j\pi y}}=4\times 1+0\times 1+$$

$$2\times -1+2\times -1=-1$$

$$F(1,1)=\sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)e^{-j2\pi\left(\frac{x}{2}+\frac{y}{2}\right)}=f(0,0)e^{0}+$$

$$f(1,0)e^{-j\pi}+f(1,0)e^{-j\pi}+f(1,1)e^{-j2\pi}=4+0-2+2$$

$$F(1,1)=2$$

$$e^{-j\pi}=\cos(-\pi)+j\overbrace{\sin(-\pi)}^{0}=-1+0=-1$$

$$e^{-j2\pi}=\underbrace{\cos(-2\pi)}_{1}+j\underbrace{\sin(-2\pi)}_{0}=+1$$

TANIN

الف) برابر با جمع مقادیر پیکسل های تصویر است.
تبدیل فوریه تصویر، تبدیلی است که تصویر را از دامنه زمان به دامنه فرکانس تبدیل می‌کند. این تبدیل می‌تواند برای تحلیل و پردازش تصاویر مفید باشد. برای تبدیل فوریه تصویر به دامنه فرکانس، ابتدا تصویر را به صورت ماتریس اعداد مختلط نمایش می‌دهیم.

نقطه مبدا تصویر، معمولا به عنوان پایه‌ای برای اعمال تبدیل فوریه در نظر گرفته می‌شود. در این حالت، رابطه تبدیل فوریه تصویر به صورت زیر است:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

در این رابطه، F(u,v) نمایش‌دهنده‌ی مقدار تبدیل فوریه در دامنه فرکانس u و v است، و f(x,y) نمایش‌دهنده‌ی مقدار تصویر در موقعیت (x,y) است. علاوه بر این، M و N به ترتیب عرض و ارتفاع تصویر را نشان می‌دهند و j نمایش‌دهنده‌ی واحد مختلط است.

می‌توان رابطه تبدیل فوریه تصویر را برای نقطه مبدا، به روابط زیر ساده کرد:

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

ب)

$$F = \begin{bmatrix} 9 & 5 \\ -1 & 3 \end{bmatrix}$$

سوال ۵

الف) تصویر مربوط به یک تصویر ساده شامل سه شکل مستطیل، مربع و مثلث است و به راحتی با تابع imread آن را می خوانیم.

ب) تابع findContours در OpenCV برای پیدا کردن مرزهای شیء در تصاویر استفاده میشود. این تابع با گرفتن یک تصویر لیستی از مرزهای شیء(contours) را به صورتی آرایه از نقاط برمیگرداند.

در اینجا با استفاده از این تابع، مرز های هر شکل را پیدا کرده و رسم کرده ایم. قبل از استفاده از تابع findContours، با استفاده از تابع cv2.threshold، تصویر را به صورت دودویی (سیاه و سفید) کرده ایم. این پیش پردازش(theshold) لازم است و باید انجام دهیم چرا که ورودی تابع findContours باید یک عکس دودویی(binary image) باشد. همان طور که در خروجی دیده می شود مثلا برای مربع هم مربع داخلی و هم خارجی را به عنوان contour داریم می توان فقط مربع خارجی را در نظر بگیریم و برای این کار باید از پارمتر cv2.RETR_EXTERNAL به جای cv2.RETR_TREE استفاده کنیم.

What are contours?

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- Since OpenCV 3.2, **findContours()** no longer modifies the source image.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

Let's see how to find contours of a binary image:

```
import numpy as np
import cv2 as cv

im = cv.imread('test.jpg')
assert im is not None, "file could not be read, check with os.path.exists()"
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

See, there are three arguments in **cv.findContours()** function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy. Contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

Python:

cv.findContours(image, mode, method[, contours[, hierarchy[, offset]]]) -> contours, hierarchy

```
#include <opencv2/imgproc.hpp>
```

Finds contours in a binary image.

The function retrieves contours from the binary image using the algorithm **[240]** . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.

Note

Since opencv 3.2 source image is not modified by this function.

Parameters

image Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use **compare**, **inRange**, **threshold** , **adaptiveThreshold**, **Canny**, and others to create a binary image out of a grayscale or color one. If mode equals to **RETR_CCOMP** or **RETR_FLOODFILL**, the input can also be a 32-bit integer image of labels (CV_32SC1).

contours Detected contours. Each contour is stored as a vector of points (e.g. std::vector<std::vector<cv::Point> >).

hierarchy Optional output vector (e.g. std::vector<cv::Vec4i>), containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative.

Note

In Python, hierarchy is nested inside a top level array. Use hierarchy[0][i] to access hierarchical elements of i-th contour.

Parameters

mode Contour retrieval mode, see **RetrievalModes**

method Contour approximation method, see **ContourApproximationModes**

offset Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

How to draw the contours?

To draw the contours, **cv.drawContours** function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

- To draw all the contours in an image:

```
cv.drawContours(img, contours, -1, (0,255,0), 3)
```

- To draw an individual contour, say 4th contour:

```
cv.drawContours(img, contours, 3, (0,255,0), 3)
```

- But most of the time, below method will be useful:

```
cnt = contours[4]
cv.drawContours(img, [cnt], 0, (0,255,0), 3)
```

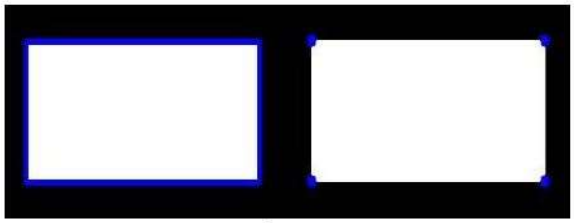
Contour Approximation Method

This is the third argument in **cv.findContours** function. What does it denote actually?

Above, we told that contours are the boundaries of a shape with same intensity. It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates ? That is specified by this contour approximation method.

If you pass **cv.CHAIN_APPROX_NONE**, all the boundary points are stored. But actually do we need all the points? For eg, you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what **cv.CHAIN_APPROX_SIMPLE** does. It removes all redundant points and compresses the contour, thereby saving memory.

Below image of a rectangle demonstrate this technique. Just draw a circle on all the coordinates in the contour array (drawn in blue color). First image shows points I got with **cv.CHAIN_APPROX_NONE** (734 points) and second image shows the one with **cv.CHAIN_APPROX_SIMPLE** (only 4 points). See, how much memory it saves!!!



image

منابع:

https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0
https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html

ج) با استفاده از تابع approxPolyDP، نقاط گوشه های هر شکل را پیدا کرده و آن ها را با علامت نقطه نشان داده ایم. سپس با بررسی تعداد گوشه های هر شکل، آن هارا به چه کلاس هایی (مستطیل، مربع و مثلث) و aspect ratio (تشخیص مربع از مستطیل) تعلق دارند، تشخیص داده ایم. در نهایت با استفاده از تابع putText، نام هر شکل را در تصویر نشان داده ایم. (توضیحات بیشتر در نوت بوک)

Python:

cv.approxPolyDP(curve, epsilon, closed[, approxCurve]) -> approxCurve

```
#include <opencv2/imgproc.hpp>
```

Approximates a polygonal curve(s) with the specified precision.

The function **cv::approxPolyDP** approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. It uses the Douglas-Peucker algorithm http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm

Parameters

curve	Input vector of a 2D point stored in std::vector or **Mat**
approxCurve	Result of the approximation. The type should match the type of the input curve.
epsilon	Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.
closed	If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

د) برای ساخت یک دسته بند برای تشخیص اشکال، ویژگیهای مفید عبارتند از:

۱. تعداد گوشه‌ها: تعداد گوشه‌ها می‌تواند به عنوان یک ویژگی مفید برای تشخیص اشکال استفاده شود. به طور مثال، مثلث سه گوشه دارد، مستطیل و مربع چهار گوشه دارند.

۲. نسبت طول به عرض: نسبت طول به عرض نیز می‌تواند به عنوان ویژگی استفاده شود. برای مثال، در صورتی که نسبت طول به عرض برابر با ۱ باشد، شکل یک مربع است.

۳. نسبت پیرامون به مساحت: نسبت پیرامون به مساحت همچنین می‌تواند به عنوان ویژگی استفاده شود. این ویژگی برای تمام اشکال قابل استفاده است.

۴. و به طور کلی می توان ویژگی(feature) های مختلفی مانند تعداد ضلع ها و قطر ها ، اندازه ضلع ها و قطر ها ، مساحت، محیط، نسبت اضلاع و ... تعریف کنیم و از الگوریتم های یادگیری ماشین کلاسیک (machine learning) برای آموزش classifier استفاده کنیم.

برای پیاده‌سازی این ویژگي ها، می‌توان از تابع‌هایی مانند cv2.findContours و cv2.approxPolyDP استفاده کرد. به طور مثال، برای تعداد گوشه‌ها، می‌توان از تابع cv2.approxPolyDP استفاده کرد و تعداد نقاط گوشه را شمارش کرد. برای نسبت طول به عرض و نسبت پیرامون به مساحت، می‌توان از تابع cv2.boundingRect و cv2.arcLength استفاده کرد و سپس این ویژگی‌ها را با استفاده از الگوریتم دسته‌بندی مناسب (مانند SVM) در دسته‌بند قرار داد.

سوال ۶

الف) برای padding با reflect101 می‌توان از تابع np.pad استفاده کرد. طرز استفاده با سرچ قابل یادگیری است. برای این کار، ابتدا تصویر را به همراه سایز کرنل و نوع padding به تابع np.pad پاس می‌دهیم و تصویر جدیدی با padding مورد نظر برگشت داده می‌شود.

سپس با استفاده از سه فیلتر متوسط گیر، میانه و گوسی، تصاویر smoothing شده را به دست می‌آوریم. در هر کدام از این فیلترها، سایز کرنل تاثیر زیادی در خروجی دارد. با افزایش سایز کرنل، smoothing بیشتر صورت می‌گیرد و نویزهای تصویر بیشتر حذف می‌شود. البته در عین حال، جزئیات تصویر نیز کمتر قابل رؤیت خواهند بود.

ب) فیلتر دوطرفه یک فیلتر صاف کننده غیرخطی، حفظ لبه و کاهش نویز برای تصاویر است. شدت هر پیکسل را با میانگین وزنی مقادیر شدت پیکسل های مجاور جایگزین می کند. این وزن می تواند بر اساس توزیع گاوسی باشد. مهمتر از همه، وزن‌ها نه تنها به فاصله اقلیدسی پیکسل‌ها، بلکه به تفاوت‌های

رادیومتری (به عنوان مثال، تفاوت‌های محدوده، مانند شدت رنگ، فاصله عمق، و غیره) بستگی دارند. این باعث حفظ لبه های تیز می شود.و مزیت آن نسبت به فیلتر های میانگین، میانه و گوسی این است که در عین اینکه نویز ها را smooth میکند ولی لبه ها را حفظ میکند بر خلاف فیلتر های ذکر شده.
فرمول آن در زیر آمده است.

فیلتر دوسویه، فیلتری غیرخطی است که لبه‌های تصویر را حفظ می‌کند و به عنوان فیلتر نرم کننده و کاهنده نویز مورد استفاده قرار می‌گیرد. مقدار نهایی درجه خاکستری پیکسل در این فیلتر به صورت میانگین وزن‌دار درجات خاکستری پیکسل‌های همسایه محاسبه می‌شود. همسایگی در این فیلتر تنها به معنی نزدیکی مکانی نیست و پیکسل‌ها برای اینکه همسایه در نظر گرفته شوند باید درجه خاکستری مشابه نیز داشته باشند.

سمت چپ: تصویر اصلی. سمت راست: تصویر پردازش شده با فیلتر دو طرفه

تعریف [ویرایش]

رابطه فیلتر دو سویه [1]

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

و مقدار ضریب نرمال سازی W_p، طبق رابطه زیر محاسبه می‌شود

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

که

I^{filtered} تصویر فیلتر شده است.
I تصویر ورودی اصلی که باید فیلتر شود
x مختصات پیکسل فعلی که باید فیلتر شوند.
Ω پنجره ای است حول پیکسل x ، بنابراین $x_i \in \Omega$ پیکسل دیگری است؛
f_r تابع کرنل برای نرم‌کردن اختلافات درجات خاکستری (این تابع می تواند یک تابع گاوسی باشد).
g_s تابع کرنل برای نرم‌کردن اختلافات موقعیتی بین پیکسل‌ها(این تابع می تواند یک تابع گاوسی باشد).

Definition [edit]

The bilateral filter is defined as[1][2]

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

and normalization term, W_p, is defined as

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

where

I^{filtered} is the filtered image;

I is the original input image to be filtered;

x are the coordinates of the current pixel to be filtered;

Ω is the window centered in x, so $x_i \in \Omega$ is another pixel;

f_r is the range kernel for smoothing differences in intensities (this function can be a Gaussian function);

g_s is the spatial (or domain) kernel for smoothing differences in coordinates (this function can be a Gaussian function).

The weight W_p is assigned using the spatial closeness (using the spatial kernel g_s) and the intensity difference (using the range kernel f_r).[2] Consider a pixel located at (i, j) that needs to be denoised in image using its neighbouring pixels and one of its neighbouring pixels is located at (k, l). Then, assuming the range and spatial kernels to be Gaussian kernels, the weight assigned for pixel (k, l) to denoise the pixel (i, j) is given by

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

where σ_d and σ_r are smoothing parameters, and $I(i, j)$ and $I(k, l)$ are the intensity of pixels (i, j) and (k, l) respectively.

After calculating the weights, normalize them:

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

where I_D is the denoised intensity of pixel (i, j).

Left: original image. Right: image processed with bilateral filter

Parameters [edit]

- As the range parameter σ_r increases, the bilateral filter gradually approaches Gaussian convolution more closely because the range Gaussian widens and flattens, which means that it becomes nearly constant over the intensity interval of the image.
- As the spatial parameter σ_d increases, the larger features get smoothened.

Limitations [edit]

The bilateral filter in its direct form can introduce several types of image artifacts:

- Staircase effect – intensity plateaus that lead to images appearing like cartoons[3]
- Gradient reversal – introduction of false edges in the image.[4]

There exist several extensions to the filter that deal with these artifacts, like the scaled bilateral filter that uses downscaled image for computing the weights.[5] Alternative filters, like the *guided filter*,[6] have also been proposed as an efficient alternative without these limitations.

مقدار انحراف معیار در فضای رنگ: هر چه مقدار بیشتر باشد، رنگ های دورتر از یکدیگر شروع به مخلوط شدن می کنند. هر چه مقدار انحراف معیار کوچکتر باشد، لبه تیزتر است. وقتی که انحراف معیار به سمت بی نهایت میل می کند، معادله به تاری گاوسی(Gaussian blur) تمایل میکند.

مقدار انحراف معیار در فضای مختصات: هرچه مقدار آن بیشتر باشد، با توجه به اینکه رنگ آنها در محدوده sigmaColor(انحراف معیار در فضای رنگ) قرار دارد، پیکسل های بیشتری با هم ترکیب می شوند.

مقدار کم انحراف معیار فضایی و رنگی باعث می‌شود که فیلتر با تعداد کمتری پیکسل در نزدیکی پیکسل مورد نظر اعمال شود و در نتیجه تاثیرات نویز کمتری حذف شود. اما در عوض، با انحراف معیار کمتر، حفظ لبه‌ها و نواحی پر جزئیات تصویر نیز کمتر می‌شود و در نتیجه تصویر شبیه به یک تصویر ابرنواختری می‌شود.

مقدار زیاد انحراف معیار فضایی و رنگی باعث می‌شود که فیلتر با تعداد بیشتری پیکسل در نزدیکی پیکسل مورد نظر اعمال شود و در نتیجه تاثیرات نویز بیشتری حذف شود. اما با انحراف معیار زیاد، تاثیرات لبه‌ها و نواحی پر جزئیات تصویر بیشتر حفظ می‌شود و در نتیجه تصویر واضح‌تر و با جزئیات بیشتری به دست می‌آید. با این حال، مقدار زیاد انحراف معیار باعث ایجاد می‌شود که فیلتر خیلی بیشتر از حد لازم اعمال شود و در نتیجه ممکن است باعث ایجاد اثرات ناخواسته مانند ایجاد هاله‌های نوری حول لبه‌ها شود.

ج) با استفاده از لینک
https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html

طرز کار هر یک از فیلتر ها را در کتابخانه opencv یافتیم و اجرا کردیم.

سپس با مقایسه دریافتیم که نتایج یکسان بود.

پایان