پاسخ سوال اول

عملگر Roberts در سال 1963 توسط Lawrence Roberts معرفی شد.ایده اصلی این عملگر تقریب زدن گرادیان های یک تصویر را که از طریق مشتق گیری گسسته که با محاسبه ی مجموع مربعات تفاوت بین پیکسل های محاور مورب به دست میآید،است.

در ابتدا باید تصویر را با دو کرنل زیر کانوالو کرد:

0	+1
-1	0

+1	0
0	-1

اگر هر پیکسل تصویر اصلی را با I(x,y) نشان دهیم و هر پیکسل حاصل از کانوالو عکس اصلی با کرنل اول را $G_Y(x,y)$ و هر پیکسل حاصل از کانوالو عکس اصلی با کرنل دوم را $G_x(x,y)$ نشان دهیم گرادیان به صورت زیر قابل محاسبه است.

$$\nabla I(x,y)=G(x,y)=\sqrt{G_x^2+G_Y^2}$$

و جهت گرادیان نیز به صورت زیر محاسبه میشود:

$$\Theta(x,y)=\arctan\left(\frac{G_y(x,y)}{G_x(x,y)}\right)-\frac{3\pi}{4}.$$

مزایا:

نسبت به سوبل پیاده سازی راحت تری دارد، در واقع ساده ترین عملگر لبه یابی است،محاسبات بسیار سریعی دارد و گرادیان دو بعدی میگیری از تصویر. ولی ممکن است در بعضی مواقع دقیق نباشد و نسبت به نویز بسیار حساس است.

هر دوی سوبل و رابرت بر پایه گرادیان هستند.

تصویر زیر نتیجه ی عملگر سوبل است:

و تصویر زیر نتیجه ی عملگر رابرت است:

منابع:

لینک اول

لینک دوم

پاسخ سوال دوم

1. لاپلاسین به طرز غیرقابل قبولی به نویز حساس است.
2. Magnitudeلاپلاسین لبه های دوتایی ایجاد میکند.
3. نمیتواند جهت لبه را تشخیص بدهد.

منبع:

لینک

لینک

پاسخ سوال سوم

اگر magnitude گرایان isotropic باشد یعنی در هر جهتی لبه ها را تشخیص میدهد.لاپلاسین یه عملگر isotropic است زیرا وزن در تمام جهت ها یکسان است.

در فصل 3 کتاب گونزالو به این صورت اشاره شده است که:

The kernels of choice in applications such as those just mentioned are *circularly symmetric* (also called *isotropic*, meaning their response is independent of orientation).

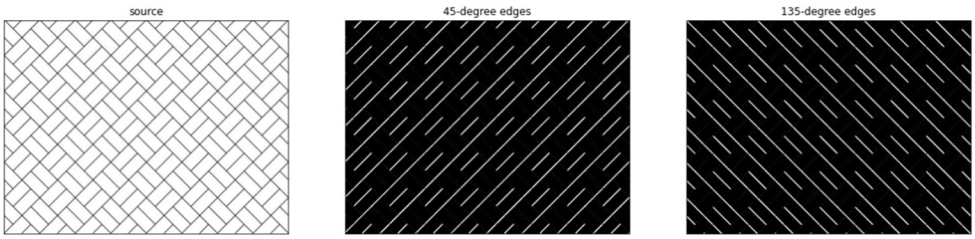
منبع:

لینک

گزارش سوال چهارم

در این سوال از لینک زیر کمک گرفتم و دانشی که از سوبل داشتم با مقایسه با کرنل های سوبل و امتحان کردن کرنل های متفاوت به پاسخ درست برای این سوال دست یافتم.

```python
def get_45_edges(image):
    '''
    Returns the image which shows the 45-degree edges.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        edges_45 (numpy.ndarray): The 45-degree edges of input image.
    '''
    kernel = None
    edges_45 = image.copy()

    #Writer your code here
    kernel=np.array([[2,1,0],[1,0,-1],[0,-1,-2]],np.float)
    edges_45=cv2.filter2D(image,-1,np.float32(kernel))
    return edges_45

def get_135_edges(image):
    '''
    Returns the image which shows the 135-degree edges.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        edges_135 (numpy.ndarray): The 135-degree edges of input image.
    '''
    kernel = None
    edges_135 = image.copy()
    #Writer your code here
    kernel=np.array([[0,-1,-2],[1,0,-1],[2,1,0]],np.float)
    edges_135=cv2.filter2D(image,-1,np.float32(kernel))

    return edges_135
```



منبع:

لینک

گزارش سوال پنجم

در این سوال من از فیلتر گاووسی که در تمرین سوم پیاده سازی کردم،استفاده کردم.

```python
def gaussian_kernel(size, sigma=1):
    '''
    Calculates and Returns Gaussian kernel.

    Parameters:
        size (int): size of kernel.
        sigma(float): standard deviation of gaussian kernel

    Returns:
        gaussian: A 2d array shows gaussian kernel
    '''
    #Writer your code here
    gaussian = None
    gaussian = np.zeros((size,size), np.float)

    ####### your code ########
    coe=1/(2*((np.pi)*(sigma**2)))
    co2_std=2*(sigma**2)
    for i in range(size):
        for j in range(size):
            gaussian[i,j]+=(coe*np.exp((-(i**2+j**2)/co2_std)))

    return gaussian
```

در مراحل بعدی ابتدا کرنل سوبل را پیاده سازی کردم.که در اسلاید های درس مقادیر عددی آن معرفی شده بود.

```python
def sobel_filters(image):
    '''
    finds the magnitude and orientation of the image using Sobel kernels.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        (magnitude, theta): A tuple consists of magnitude and orientation of the image gradients.
    '''
    #Writer your code here


    ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float)
    kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float)
    Ix = cv2.filter2D(np.float32(image),-1,np.float32(kx))
    Iy = cv2.filter2D(np.float32(image),-1,np.float32(ky))

    magnitude = np.hypot(Ix, Iy) #Result is equivalent to Equivalent to sqrt(x1**2 + x2**2), element-wise
    magnitude = magnitude / magnitude.max() * 255
    theta = np.arctan2(Iy, Ix)

    return magnitude, theta
```

نتیجه ی حاصل از کرنل سوبل دارای لبه های ضخیم است،با فانکشن بعدی و با اطلاعاتی که از این لینک به دست آوردم،با فانکشن بعدی آن را نازک تر کردم.

```
    #Writer your code here
    M, N = image.shape
    Z = np.zeros((M,N), dtype=np.int32)
    angle = theta * 180. / np.pi
    angle[angle < 0] += 180


    for i in range(1,M-1):
        for j in range(1,N-1):
                q = 255
                r = 255


                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                    q = image[i, j+1]
                    r = image[i, j-1]

                elif (22.5 <= angle[i,j] < 67.5):
                    q = image[i+1, j-1]
                    r = image[i-1, j+1]

                elif (67.5 <= angle[i,j] < 112.5):
                    q = image[i+1, j]
                    r = image[i-1, j]

                elif (112.5 <= angle[i,j] < 157.5):
                    q = image[i-1, j-1]
                    r = image[i+1, j+1]

                if (image[i,j] >= q) and (image[i,j] >= r):
                    Z[i,j] = image[i,j]
                else:
                    Z[i,j] = 0

    return Z
```

در فانکشن بعدی برای رفع گسستی های ایجاد شده در لبه ها استفاده کردیم.در این فانکشن از دو حد استفاده شده که با تغییر متعدد این حدها به دنبال بهترین نتیجه بودیم که عدد 70 و 38 را انتخاب کردم در نهایت.با بررسی و مقایسه هر یک مقدار آنها را 0 یا 255 تغییر دادم(با توجه به اینکه در چه بازه ایی قرار دارند)

```
#Writer your code here
M, N = image.shape
result = np.zeros((M,N), dtype=np.int32)

weak = np.int32(20)
strong = np.int32(255)

strong_i, strong_j = np.where(image >= highThreshold)
zeros_i, zeros_j = np.where(image < lowThreshold)

weak_i, weak_j = np.where((image <= highThreshold) & (image >= lowThreshold))

result[strong_i, strong_j] = strong
result[weak_i, weak_j] = weak
final=result.copy()
for i in range(1, M-1):
    for j in range(1, N-1):
        if (result[i,j] == weak):
                if ((result[i+1, j-1] == strong) or (result[i+1, j] == strong) or (result[i+1, j+1] == strong)
                    or (result[i, j-1] == strong) or (result[i, j+1] == strong)
                    or (result[i-1, j-1] == strong) or (result[i-1, j] == strong) or (result[i-1, j+1] == strong)):
                    final[i, j] = strong
                else:
                    final[i, j] = 0

return result,final
```

در نهایت در تابع انتهایی توابع نوشته شده را ترکیب کردم و نتیجه رضایت بخش بود:


Input image
Blur
Gradient Intensity
Non Max Suppression
Double Treshold
Output image

منبع:

لینک