دانشگاه صنعتی امیرکبیر

دانشکده مهندسی پزشکی

گزارش پردازش تصویر

تمرین سری ششم

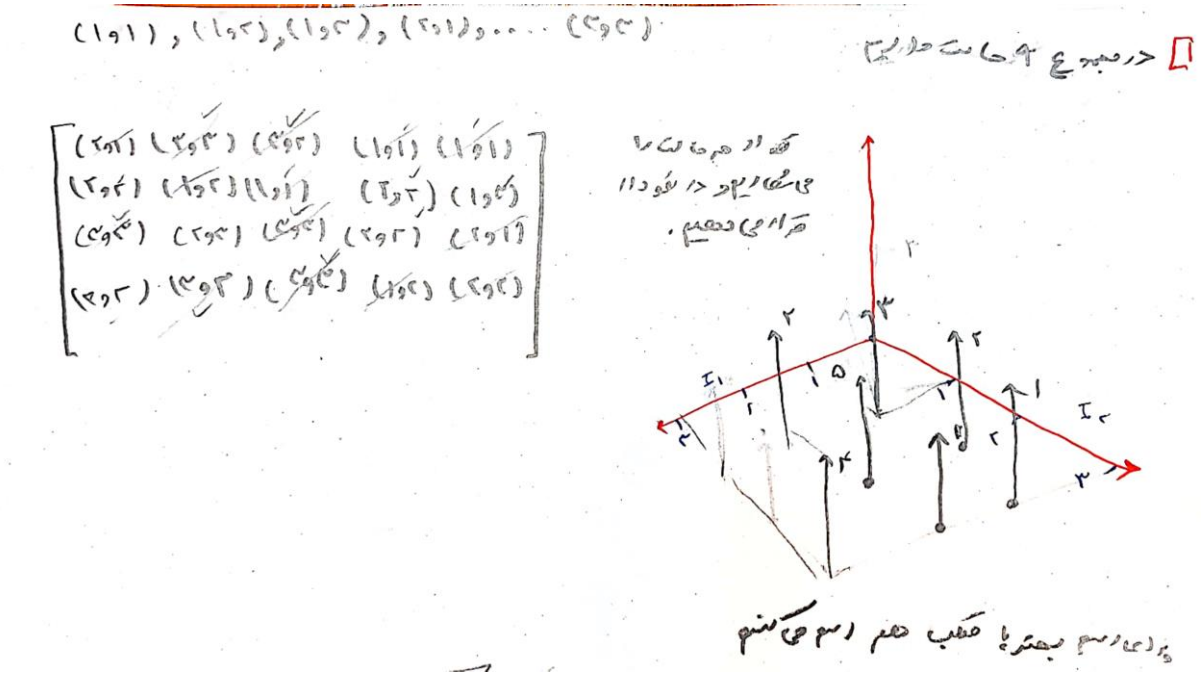
دانشجو

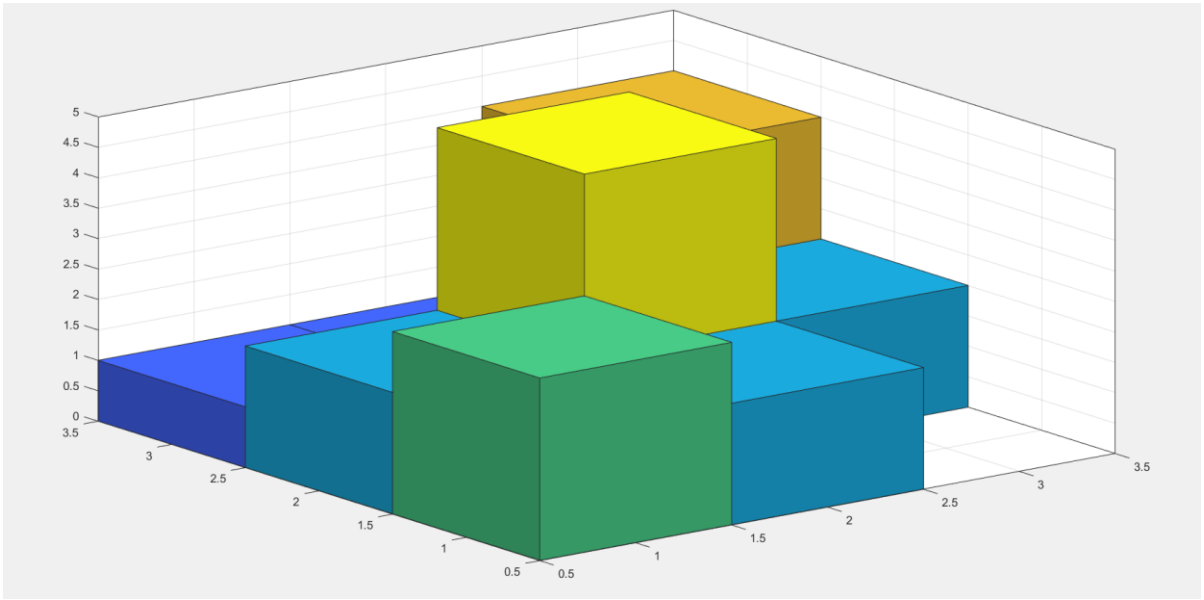
سیدابوالفضل مرتضوی ۹۸۳۳۰۶۳

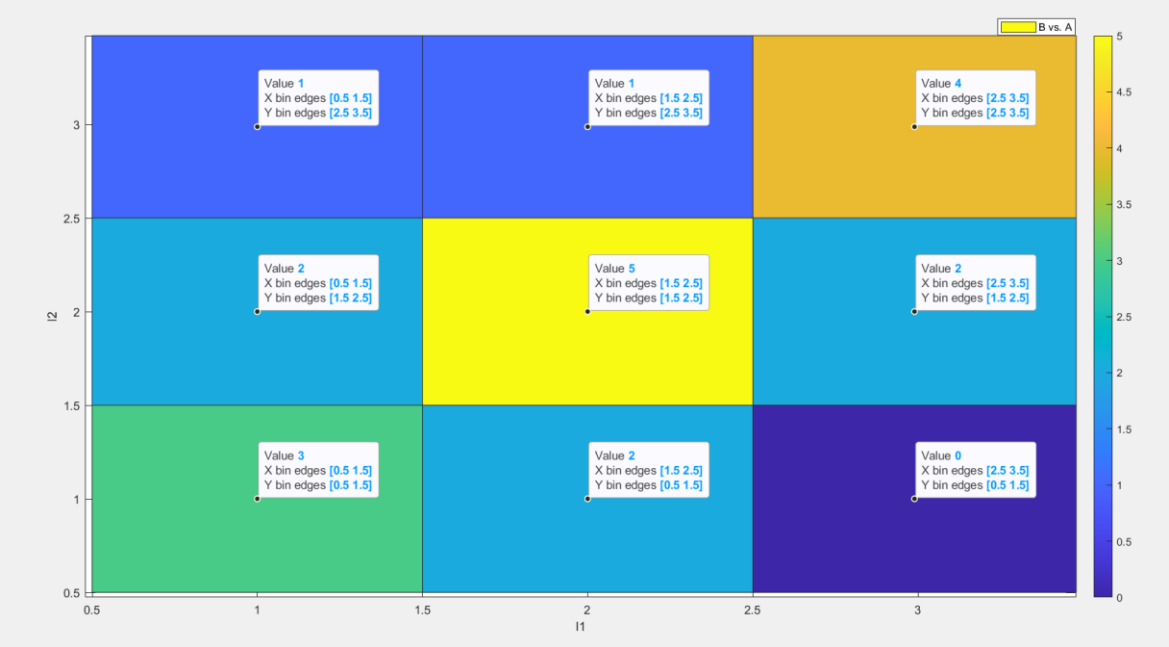
دی ۱۴۰۱

فهرست مطالب

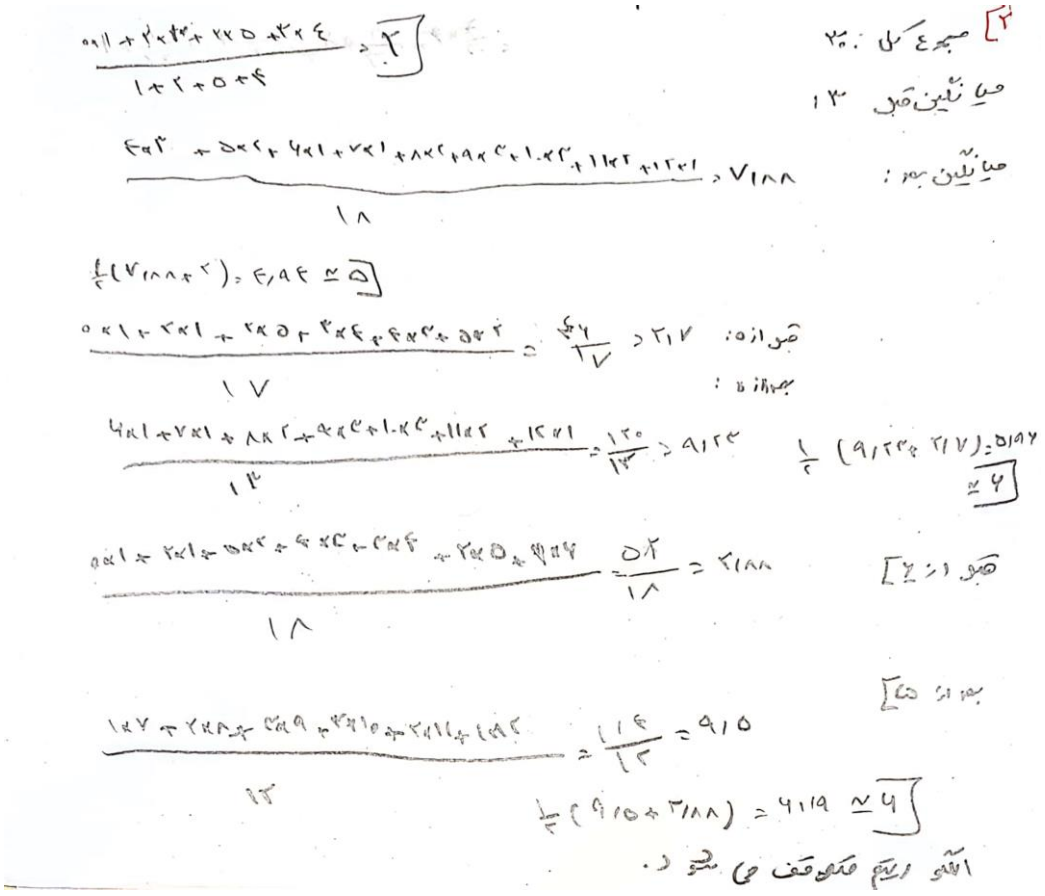
تمرین اول






B vs. A
Value 1
X bin edges [0.5 1.5]
Y bin edges [2.5 3.5]
Value 1
X bin edges [1.5 2.5]
Y bin edges [2.5 3.5]
Value 4
X bin edges [2.5 3.5]
Y bin edges [2.5 3.5]
Value 2
X bin edges [0.5 1.5]
Y bin edges [1.5 2.5]
Value 5
X bin edges [1.5 2.5]
Y bin edges [1.5 2.5]
Value 2
X bin edges [2.5 3.5]
Y bin edges [1.5 2.5]
Value 3
X bin edges [0.5 1.5]
Y bin edges [0.5 1.5]
Value 2
X bin edges [1.5 2.5]
Y bin edges [0.5 1.5]
Value 0
X bin edges [2.5 3.5]
Y bin edges [0.5 1.5]
I1
I2

تمرین دوم



تمرین سوم

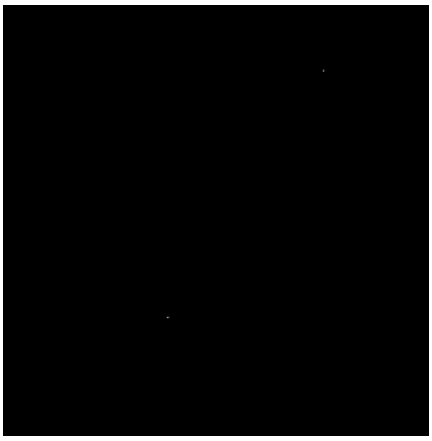
۳-۱ بخش ۱

انجام شد.

۳-۲ بخش ۲و۳

```
white_seed_coor=[(385,344)]
gray_seed_coor=[(190,467)]
empty_img=np.zeros(MRI_img.shape)
empty_img[(385,344)]=255
empty_img[(190,467)]=100
```

دو نقطه در تصویر به عنوان دانه انتخاب شدند در شکل ۳.۱محل این نقاط نشان داده شده است.



شکل ۳.۱ محل نقاط

۳-۳ بخش ۴

```
def regionGrow(img, seeds, thresh, diffType,label):
    height, weight = img.shape
    seedMark = np.zeros(img.shape)
    seedList = []
    steps = 1
    seeds_grayscale = []
    seeds_avg_grayscale = 0
```

```
    for seed in seeds:
        seedList.append(seed)
        seeds_grayscale.append(img[seed[0], seed[1]])
    seeds_avg_grayscale = np.average(seeds_grayscale)

    connects = [ (0, -1), (1, 0),(0, 1), (-1, 0)]
    while(len(seedList)>0):
        currentPoint = seedList.pop(0)
        if diffType=='static':
            seedMark[currentPoint[0], currentPoint[1]] = label

            for i in range(4):
                tmpX = currentPoint[0] + connects[i][0]
                tmpY = currentPoint[1] + connects[i][1]
                if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= weight:
                    continue

                grayDiff = abs (img[tmpX, tmpY] - seeds_avg_grayscale)

                if grayDiff < thresh and seedMark[tmpX,tmpY] == 0:
                    seedMark[tmpX,tmpY] = label
                    seedList.append((tmpX,tmpY))


        if (diffType == "dynamic"):
            seeds_grayscale.append(img[currentPoint[0], currentPoint[1]])
            seeds_avg_grayscale = np.average(seeds_grayscale)
        seedMark[currentPoint[0], currentPoint[1]] = label

        for i in range(4):
            tmpX = currentPoint[0] + connects[i][0]
            tmpY = currentPoint[1] + connects[i][1]
            if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= weight:
                continue

            grayDiff = abs (img[tmpX, tmpY] - seeds_avg_grayscale)

            if grayDiff < thresh and seedMark[tmpX,tmpY] == 0:
                seedMark[tmpX,tmpY] = label
                seedList.append((tmpX,tmpY))

        steps +=1


    print ("Number of steps: ", steps)
    return seedMark
```

در این بخش تابع regionGrow با ۵ ورودی تعریف شد. یک ورودی برای تصویر،ورودی دوم برای مختصات دانه‌های انتخاب شده،ورودی سوم آستانه لازم برای مقایسه شباهت دو نقطه، ورودی چهارم، نوع آستانه،

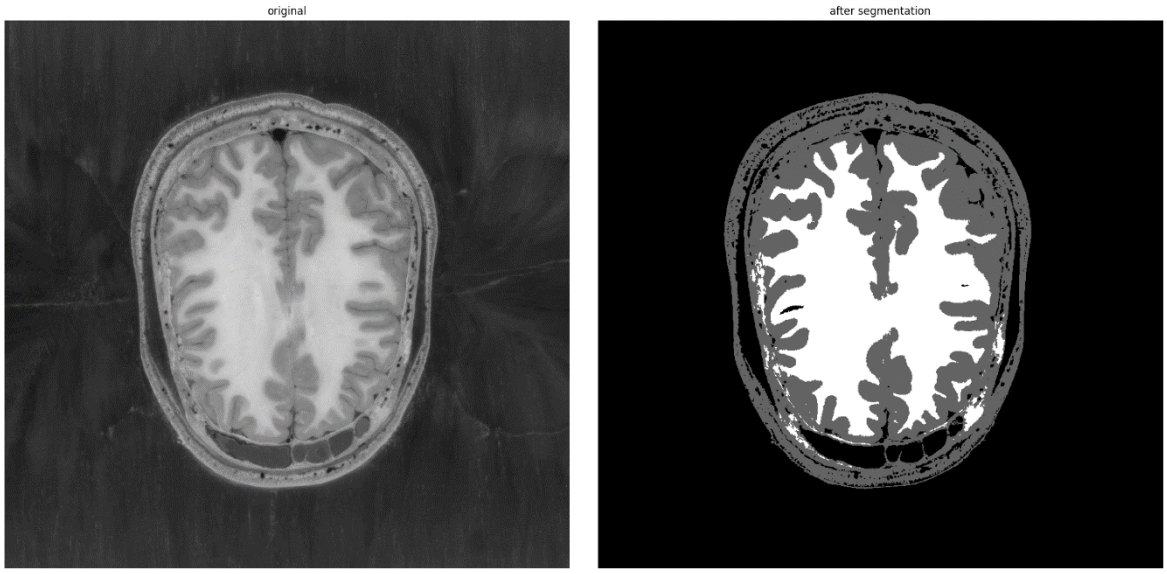
ورودی پنجم برای اندازه اینتنسیتی نهایی خروجی. بااین ورودی آخر می‌توان بین بخش سفید و خاکستری تفاوت قائل شد.

۳-۴ بخش۵و۶و۷و۸

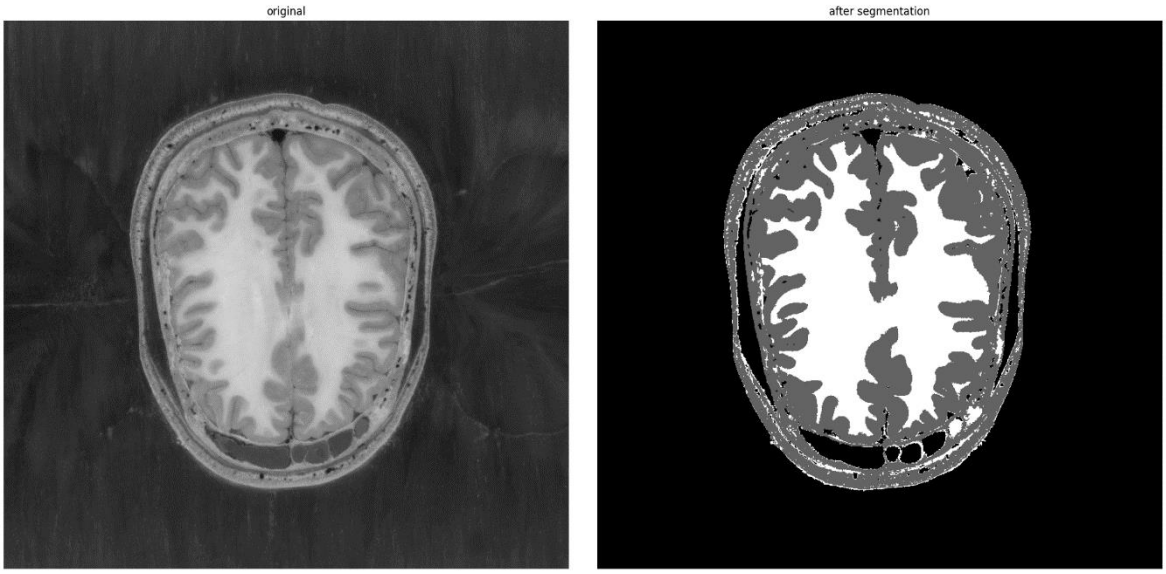
```
white_seed_coor=[(385,344)]
gray_seed_coor=[(190,467)]
thresh_w = 70
thresh_g = 30
seedMark_white = np.uint8(regionGrow(MRI_img, white_seed_coor, thresh_w,
thresh_type,255))
seedMark_gray= np.uint8(regionGrow(MRI_img, gray_seed_coor, thresh_g,
thresh_type,100))
seedmark_final = np.uint8(seedMark_gray + seedMark_white)
```

در این بخش برای بخش سفید و خاکستری هرکدام یکبار تابع اعمال شد. برای هرکدام label متفاوتی داده شد. سپس خروجی دو بخش باهم جمع شده و نتیجه نهایی در خروجی نمایش داده شد.

برای بخش متغیر مدت زمان لازم برای به دست آوردن خروجی بیشتر بود و تعداد گام‌های بیشتری هم طول کشید.



شکل ۳.۲ خروجی برای بخش ثابت



شکل ۳.۳ خروجی برای بخش متغیر

تعدادگام ها:

متغیر: بخش سفید:۱۹۷۶۵۷ بخش‌خاکستری:۱۱۹۳۱۲

ثابت: بخش سفید:۱۰۰۹۹۸ بخش‌خاکستری:۱۱۴۳۱۹

سوال ۴

۴-۱ بخش ۱

انجام شد.

۴-۲ بخش ۲

```
def Event_Mouse(event, X_Corr, Y_Coor,flag,params) :
    # Left click mouse
    if event == cv.EVENT_LBUTTONDOWN :
        # Add seeds
        seeds.append((Y_Coor, X_Corr))
        # Draw solid dots
        cv.circle(Img_MRI, center = (X_Corr, Y_Coor), radius = 2,
```

```
                color = (0, 0, 255), thickness = -1)
        cv.circle(Img_MRI2, center = (X_Corr, Y_Coor), radius = 2,
                color = (0, 0, 255), thickness = -1)
def Select_seed(img):
    cv.namedWindow('img')
    cv.setMouseCallback('img', Event_Mouse)
    cv.imshow('img', img)

    while True :
        cv.imshow('img', img)
        if cv.waitKey(1) & 0xFF == ord('q') :
            break
    cv.destroyAllWindows()
```

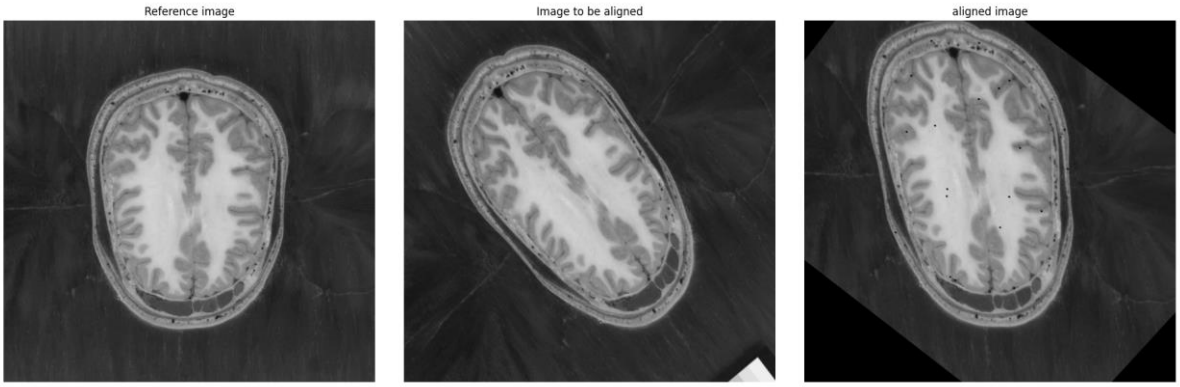
در این بخش تابع فوق تعریف شد. در هربار بااستفاده از کلیک چپ بخش‌هایی از تصویر انتخاب می‌شوند. این بخش‌ها با نقاط مشکی برروی تصویر نشان داده می‌شوند.

۳-۴ بخش۳

```
height, width = Img_MRI.shape
seeds = []
MRI_copy=Img_MRI.copy()
MRI2_copy=Img_MRI2.copy()

Select_seed(Img_MRI)
Select_seed(Img_MRI2)

Img_MRI_seeds = seeds[0:int(len(seeds)/2)]
Img_MRI2_seeds = seeds[int(len(seeds)/2):len(seeds)]

Img_MRI_seeds = np.float32(Img_MRI_seeds[:])
Img_MRI2_seeds = np.float32(Img_MRI2_seeds[:])

homography, mask = cv.findHomography(Img_MRI_seeds, Img_MRI2_seeds, cv.RANSAC)
transformed_MRI2 = cv.warpPerspective(Img_MRI2, homography, (width, height))
```

دراین بخش از تابع findHomography برای پیدا کردن تبدیل لازم و از تابع warpPerspective برای اعمال تبدیل به دست آمده برروی تصویر استفاده شد. برای انجام این مراحل نیاز به نقاطی از دو تصویر داریم. برای این کار با استفاده از کلیک موس چند نقطه از تصویر اول و تصویر دوم انتخاب کرده و درون لیست seed

ریخته می‌شود. سپس نصفه اول لیست به لیست مربوط به تصویر اول و بخش دوم به لیست مربوط به تصویر دوم داده شد. و درنهایت تبدیل انجام شد.



شکل ۴.۱ خروجی بخش ۳

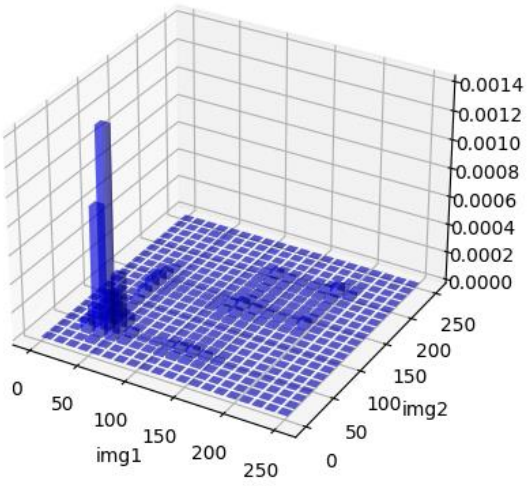
چون نقاط توسط کاربر انتخاب می‌شوند، ممکن است که نقاط دقیقا منطبق با یک دیگر نباشند و چند پیکسل باهم تفاوت داشته باشند. برای همین تصویر خروجی دقیقا با تصویر اصلی یکسان نیست. برای دقت بیشتر ۱۰ نقطه انتخاب شد.

۴-۴ بخش ۴و۵

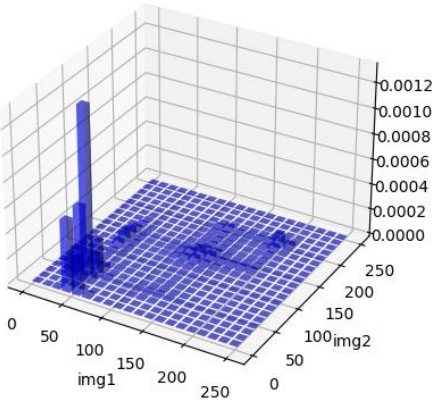
```
def joint_histogram(img1 , img2):

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x = img1.ravel()
    y = img2.ravel()
    hist, xedges, yedges = np.histogram2d(x, y, bins=20, range=[[0, 255], [0,
255]], density=True)

    xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25)
    xpos = xpos.flatten('F')
    ypos = ypos.flatten('F')
    zpos = np.zeros_like(xpos)

    dx =10* np.ones_like(zpos)
    dy = dx.copy()
    dz = hist.flatten()
```

```
    ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', zsort='average',
alpha=0.4)
    ax.set_xlabel('img1')
    ax.set_ylabel('img2')
    plt.show()
```

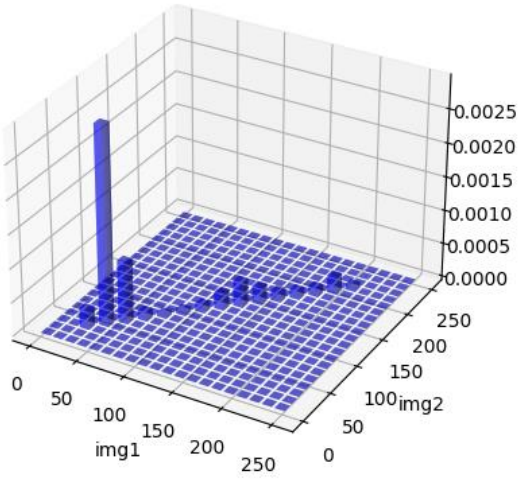
در این بخش تابع فوق برای رسم هیستوگرام نوشته شد. و خروجی در تصاویر بعدی نشان داده شده است.



شکل ۴.۲هیستوگرام دو تصویر



شکل ۴.۳ هیستوگرام تصویر منطبق شده باتصویر اصلی



شکل ۴.۴ هیستوگرام تصویر اصلی با خودش

همانطور که در تصاویر فوق قابل مشاهده است هیستوگرام تصویر با خودش به صورت قطری است. اما هیستوگرام تصویر اصلی با دو تصویر دیگر قطری نیست که نشان دهنده عدم تطابق کامل دو تصویر است. هیستوگرام تصویر اصلی و تصویر دوم قبل از رجیستریشن عناصر کمتری روی قطر اصلی دارد اما بعد از اعمال رجیستریشن تعداد عناصر روی قطر اصلی افزایش می‌یابد.