대응관계 설정 실습 1

저자 : 김천룡 , 저자 : 곡우환

Corresponding relations in Computer Vision 1

Nathan Jin , and Mike Gu

요 약

OpenCV를 이용하여 두 입력 이미지의 동일 특징 점 영역을 히스토그램 단위로 분석한다.

Abstract

In the report, we used OpenCV and image's histogram to check and analyze two special point area in two image.

Key words

OpenCV, histogram, normalize, compare, gradient, grayscale

Ⅰ. 서 론

컴퓨터비전 분야에서 대응관계 관련 문제는 꾸준하게 연구되고 있다. 다양한 대응관계 관련 알고리즘이 개발되면서, 대응관계 검출 및 인식 연구의 성능을 크게 개선하고 있고 응용 분야에서도 활발하고 있다.

데이터 세트: 두 개의 3024*4032의 컬러 이미지.

Ⅱ. 알고리즘 구현

본 실험의 과정은 4부분을 나누어 있다.

1. 두 입력영상의 동일 코너 점 영역을 저장.
2. 그래디언트 및 Grayscale 분포의 히스토그램을 저장.
3. 두 영상의 특징 점 집합에 대해 히스토그램 거리를 각각 계산.
4. 최소 거리의 특징 점 쌍을 직선으로 연결.

두 입력영상의 동일 코너 점 영역을 저장:

본 실험의 초반에 마우스 클릭으로 좌표 값을 저장하고 그 점부터 옆으로 코너영역을 잡기를 시도했는데 마우스 클릭으로 좌표 값을 메인 함수로 전달하는 과정에 문제가 생겨서 좌표 값을 하나하나 입력한다.

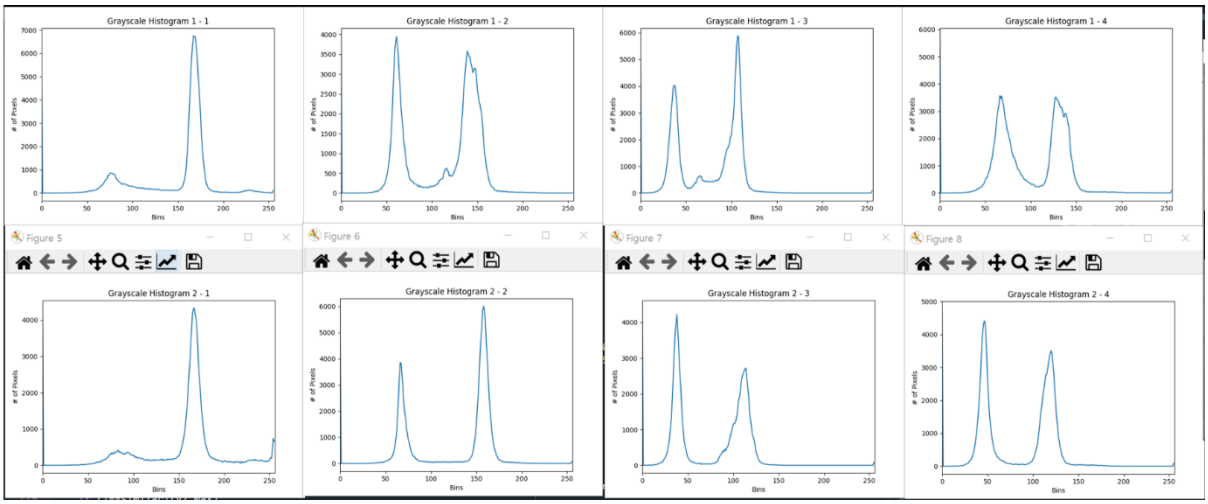
```
def Draw_RectText(img,x1,x2,y1,y2,text):
    cv2.rectangle(img,(x1,y1),(x2,y2),(0,0,255),5)
    cv2.putText(img,text,(x1,y1),cv2.FONT_HERSHEY_COMPLEX,4,(255,255,255),4)

first=cv2.imread('1st.jpg')
first=cv2.cvtColor(first,cv2.COLOR_BGR2GRAY)

dst11=first[607:923,1153:1557]
Draw_RectText(first,1153,1557,607,923,'1')

dst12=first[1677:2093,15:373]
Draw_RectText(first,15,373,1577,2093,'2')

dst13=first[3401:3751,1553:1967]
Draw_RectText(first,1553,1967,3401,3751,'3')

dst14=first[2163:2679,2683:3011]
Draw_RectText(first,2683,3011,2163,2679,'4')

second=cv2.imread('2nd.jpg')
second=cv2.cvtColor(second,cv2.COLOR_BGR2GRAY)

dst21=second[986:1308,2390:2694]
Draw_RectText(second,2390,2694,986,1308,'1')

dst22=second[950:1300,650:1000]
Draw_RectText(second,650,1000,950,1300,'2')

dst23=second[3316:3684,567:846]
Draw_RectText(second,567,846,3316,3684,'3')

dst24=second[3395:3793,2316:2620]
Draw_RectText(second,2316,2620,3395,3793,'4')
```

그래디언트 및 Grayscale 분포의 히스토그램을 저장:

OpenCV의 calcHist 함수로 Grayscale분포의 히스토그램을 하나하나 저장하고 출력한다. 히스토그램 비교의 정확성을 보완하기 위해 Normalize 함수로 저장한 히스토그램을 정규화 한다.

```
h11=cv2.calcHist([dst11],[0],None,[256],[0,256])
Draw_Histogram(h11,'Grayscale Histogram 1 - 1')
h11=cv2.normalize(h11, h11, 0, 1, cv2.NORM_MINMAX, -1)

h12=cv2.calcHist([dst12],[0],None,[256],[0,256])
Draw_Histogram(h12,'Grayscale Histogram 1 - 2')
h12=cv2.normalize(h12, h12, 0, 1, cv2.NORM_MINMAX, -1)

h13=cv2.calcHist([dst13],[0],None,[256],[0,256])
Draw_Histogram(h13,'Grayscale Histogram 1 - 3')
h13=cv2.normalize(h13, h13, 0, 1, cv2.NORM_MINMAX, -1)

h14=cv2.calcHist([dst14],[0],None,[256],[0,256])
Draw_Histogram(h14,'Grayscale Histogram 1 - 4')
h14=cv2.normalize(h14, h14, 0, 1, cv2.NORM_MINMAX, -1)

h21=cv2.calcHist([dst21],[0],None,[256],[0,256])
Draw_Histogram(h21,'Grayscale Histogram 2 - 1')
h21=cv2.normalize(h21, h21, 0, 1, cv2.NORM_MINMAX, -1)

h22=cv2.calcHist([dst22],[0],None,[256],[0,256])
Draw_Histogram(h22,'Grayscale Histogram 2 - 2')
h22=cv2.normalize(h22, h22, 0, 1, cv2.NORM_MINMAX, -1)

h23=cv2.calcHist([dst23],[0],None,[256],[0,256])
Draw_Histogram(h23,'Grayscale Histogram 2 - 3')
h23=cv2.normalize(h23, h23, 0, 1, cv2.NORM_MINMAX, -1)

h24=cv2.calcHist([dst24],[0],None,[256],[0,256])
Draw_Histogram(h24,'Grayscale Histogram 2 - 4')
h24=cv2.normalize(h24, h24, 0, 1, cv2.NORM_MINMAX, -1)
```

```python
def Draw_Histogram(hist,name):
    plt.figure()
    plt.title(name)
    plt.xlabel("Bins")
    plt.ylabel("# of Pixels")
    plt.plot(hist)
    plt.xlim([0,256])
    plt.show()
```



두 영상의 특징 점 집합에 대해 히스토그램 거리를 각각 계산:

OpenCV의 compareHist 함수로 정규화 되는 히스토그램들을 비교한다. 여기에 compareHist 함수안에 HISTCMP_CORREL을 사용해서 0~1인 비교 결과 값이 나온다. 이 값이 클수록 이미지가 서로 비슷한다. 그리고 max 값을 저장하고 이것으로 아래의 최소 거리의 특징 점 쌍의 연결에 사용한다.

```
similarity11=cv2.compareHist(h11,h21,cv2.HISTCMP_CORREL)
similarity12=cv2.compareHist(h11,h22,cv2.HISTCMP_CORREL)
similarity13=cv2.compareHist(h11,h23,cv2.HISTCMP_CORREL)
similarity14=cv2.compareHist(h11,h24,cv2.HISTCMP_CORREL)
similarity1=np.array([similarity11,similarity12,similarity13,similarity14])
similarity1_max=max(similarity11,similarity12,similarity13,similarity14)

similarity21=cv2.compareHist(h12,h21,cv2.HISTCMP_CORREL)
similarity22=cv2.compareHist(h12,h22,cv2.HISTCMP_CORREL)
similarity23=cv2.compareHist(h12,h23,cv2.HISTCMP_CORREL)
similarity24=cv2.compareHist(h12,h24,cv2.HISTCMP_CORREL)
similarity2=np.array([similarity21,similarity22,similarity23,similarity24])
similarity2_max=max(similarity21,similarity22,similarity23,similarity24)

similarity31=cv2.compareHist(h13,h21,cv2.HISTCMP_CORREL)
similarity32=cv2.compareHist(h13,h22,cv2.HISTCMP_CORREL)
similarity33=cv2.compareHist(h13,h23,cv2.HISTCMP_CORREL)
similarity34=cv2.compareHist(h13,h24,cv2.HISTCMP_CORREL)
similarity3=np.array([similarity31,similarity32,similarity33,similarity34])
similarity3_max=max(similarity31,similarity32,similarity33,similarity34)

similarity41=cv2.compareHist(h14,h21,cv2.HISTCMP_CORREL)
similarity42=cv2.compareHist(h14,h22,cv2.HISTCMP_CORREL)
similarity43=cv2.compareHist(h14,h23,cv2.HISTCMP_CORREL)
similarity44=cv2.compareHist(h14,h24,cv2.HISTCMP_CORREL)
similarity4=np.array([similarity41,similarity42,similarity43,similarity44])
similarity4_max=max(similarity41,similarity42,similarity43,similarity44)
```

최소 거리의 특징 점 쌍을 직선으로 연결:

먼저 Honcat 으로 실험 데이터 및 두개의 3024*4032의 이미지를 좌우로 연결한다. 순환문으로 각각 코너영역의 서로 대응한 제일 큰 similarity값을 찾고 line 함수로 직선으로 연결한다.

```
similarity_max=np.array([similarity1_max,similarity2_max,similarity3_max,similarity4_max])

second_point=np.array([[((2390+2694)//2)+3024,(986+1308)//2],[((650+1000)//2)+3024,(950+1300)//2],
                   [((567+846)//2)+3024,(3316+3684)//2],[((2316+2620)//2)+3024,(3395+3793)//2]])

result=cv2.hconcat([first,second])

for i in (similarity_max):
    if (i==similarity1_max):
        num=0
        for j in (similarity1):
            num+=1
            if(j==similarity1_max):
                cv2.line(result,((1153+1557)//2,(607+923)//2),(second_point[num-1][0],second_point[num-1][1]),(0,0,255),thickness=5,line
    if (i==similarity2_max):
        num=0
        for j in (similarity2):
            num+=1
            if(j==similarity2_max):
                cv2.line(result,((15+373)//2,(1677+2093)//2),(second_point[num-1][0],second_point[num-1][1]),(0,0,255),thickness=5,lineTy
    if (i==similarity3_max):
        num=0
        for j in (similarity3):
            num+=1
            if(j==similarity3_max):
                cv2.line(result,((1553+1967)//2,(3401+3751)//2),(second_point[num-1][0],second_point[num-1][1]),(0,0,255),thickness=5,li
    if (i==similarity4_max):
        num=0
        for j in (similarity4):
            num+=1
            if(j==similarity4_max):
                cv2.line(result,((2683+3011)//2,(2163+2679)//2),(second_point[num-1][0],second_point[num-1][1]),(0,0,255),thickness=5,li
```

그래디언트 히스토그램을 구하는과정:

Step1: Smooth image with Gaussian filtering

```
def gaussian(image,ksize):
    sigma =  0.3 *((ksize-1)*0.5-1) + 0.8
    pad = ksize//2
    out_p = padding(image,ksize)
    h = image.shape[0]
    w = image.shape[1]
    c = image.shape[2]
    kernel = np.zeros((ksize,ksize))
    for x in range(-pad,-pad+ksize):
        for y in range(-pad,-pad+ksize):
            kernel[y+pad,x+pad] = np.exp(-(x**2+y**2)/(2*(sigma**2)))
    kernel /= (sigma*np.sqrt(2*np.pi))
    kernel /=  kernel.sum()
    tmp = out_p.copy()
    for y in range(h):
        for x in range(w):
            for z in range(c):

                out_p[pad+y,pad+x,z] = np.sum(kernel*tmp[y:y+ksize,x:x+ksize,z])


    out = out_p[pad:pad+h,pad:pad+w].astype(np.uint8)
    # print(out)

    return out
✓ 0.5s
```

Step2: Calculate gradient with Sobel operator

```
def sobel_operator(img):
    r, c = img.shape
    new_image = np.zeros((r, c))
    new_imageX = np.zeros(img.shape)
    new_imageY = np.zeros(img.shape)
    s_suanziX = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    s_suanziY = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
    for i in range(r-2):
        for j in range(c-2):
            new_imageX[i+1, j+1] = abs(np.sum(img[i:i+3, j:j+3] * s_suanziX))
            new_imageY[i+1, j+1] = abs(np.sum(img[i:i+3, j:j+3] * s_suanziY))
            new_image[i+1, j+1] = (new_imageX[i+1, j+1]*new_imageX[i+1,j+1] + new_imageY[i+1, j+1]*new_imageY[i+1,j+1])**0.5

    return np.uint8(new_image)
✓ 0.7s
```

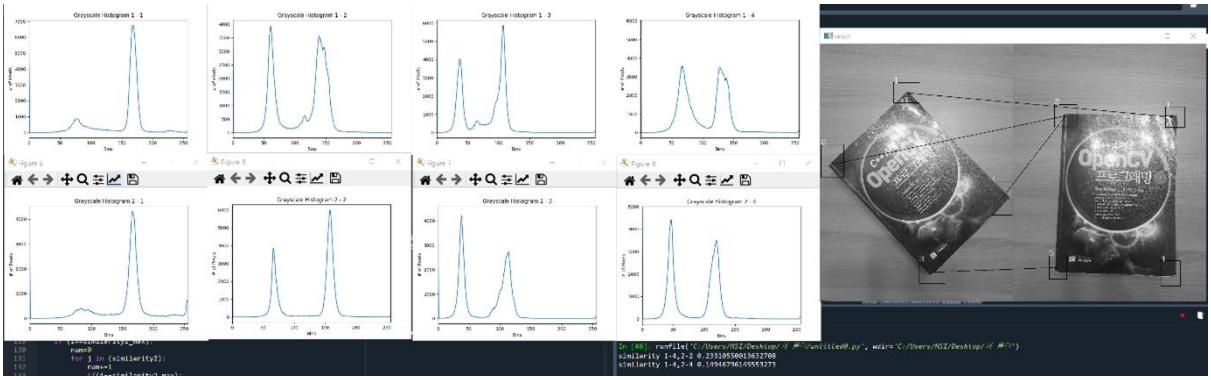
(그래디언트 값 출력 결과)

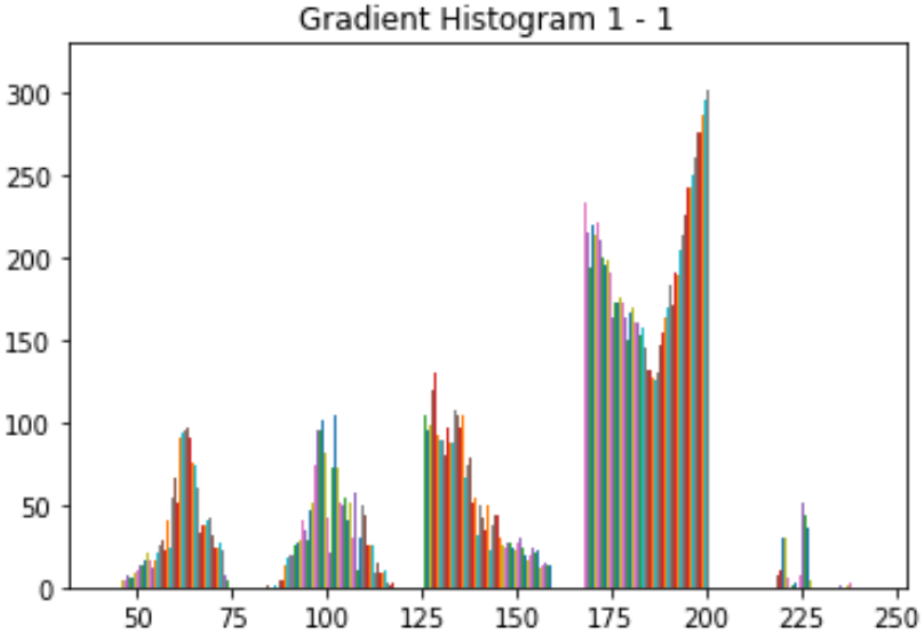
cv project1.ipynb > gray1 (316, 404)

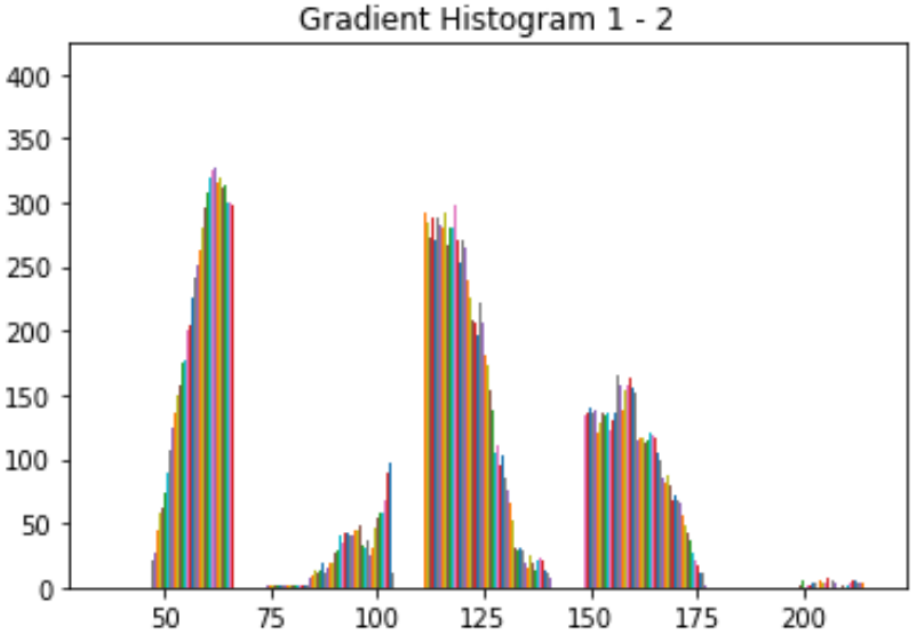
> SLICING

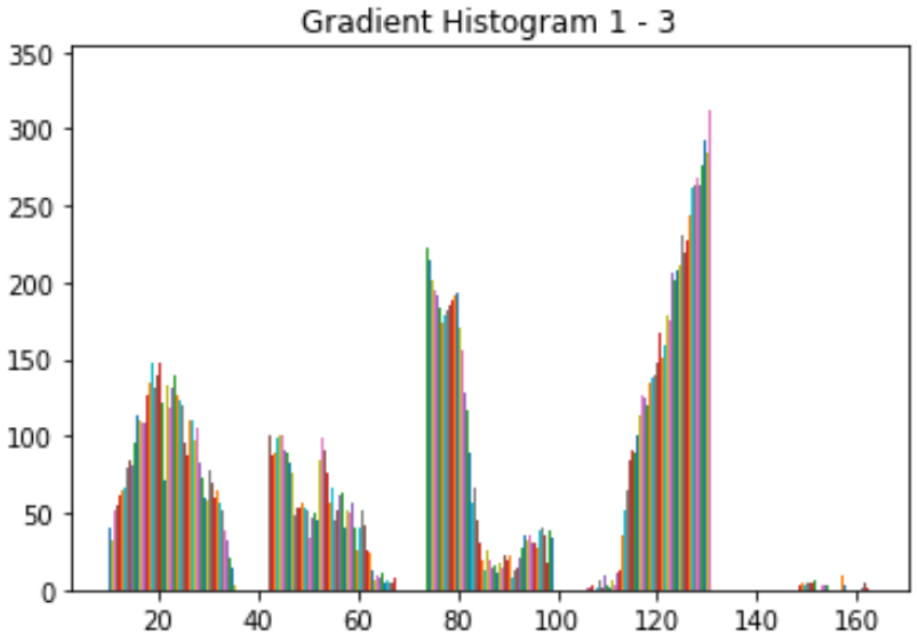
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	55	72	72	72	72	72	72	71	65	60	64	70	72	72	72	72	72	72
1	72	96	96	96	96	96	96	87	63	45	59	85	96	96	96	96	96	96
2	72	96	96	96	95	97	101	84	37	12	34	83	104	104	104	103	103	104
3	72	96	96	96	97	108	123	106	44	10	41	105	132	132	131	130	131	133
4	72	96	96	93	94	114	147	143	94	61	89	142	161	160	158	157	159	163
5	72	96	96	88	74	91	145	165	150	139	150	167	171	171	169	168	169	172
6	72	96	96	87	67	81	140	169	169	169	170	171	172	172	173	172	170	170
7	72	96	96	93	90	113	152	168	167	167	168	170	171	173	173	173	171	168
8	72	96	96	96	104	131	159	167	167	167	167	169	171	171	170	170	169	168
9	72	96	96	96	104	131	157	166	167	167	168	168	169	169	168	168	168	169
10	72	96	96	96	103	129	154	164	165	166	167	168	168	168	168	168	168	170
11	72	96	96	96	103	128	154	164	165	165	166	167	167	167	168	168	168	169
12	72	96	96	96	103	128	155	165	166	166	166	166	166	166	168	169	169	170
13	72	96	96	96	103	129	157	166	167	168	169	169	169	170	171	171	172	173
14	72	96	96	96	103	130	158	168	169	171	172	173	174	175	175	176	178	179
15	72	96	96	96	103	130	158	169	170	170	171	173	176	175	172	175	179	181
16	72	96	96	96	103	130	158	167	168	167	166	168	171	168	162	164	170	172
17	72	96	96	96	103	130	157	167	167	165	164	163	163	159	154	154	159	161
18	72	96	96	96	103	130	156	166	166	163	161	159	156	153	152	152	153	155
19	72	96	96	96	103	128	154	163	163	161	158	157	154	152	153	154	153	153
20	72	96	96	96	103	127	152	159	159	157	156	156	155	156	159	159	157	157
21	72	96	96	96	103	127	151	157	157	157	158	158	158	159	161	162	161	160

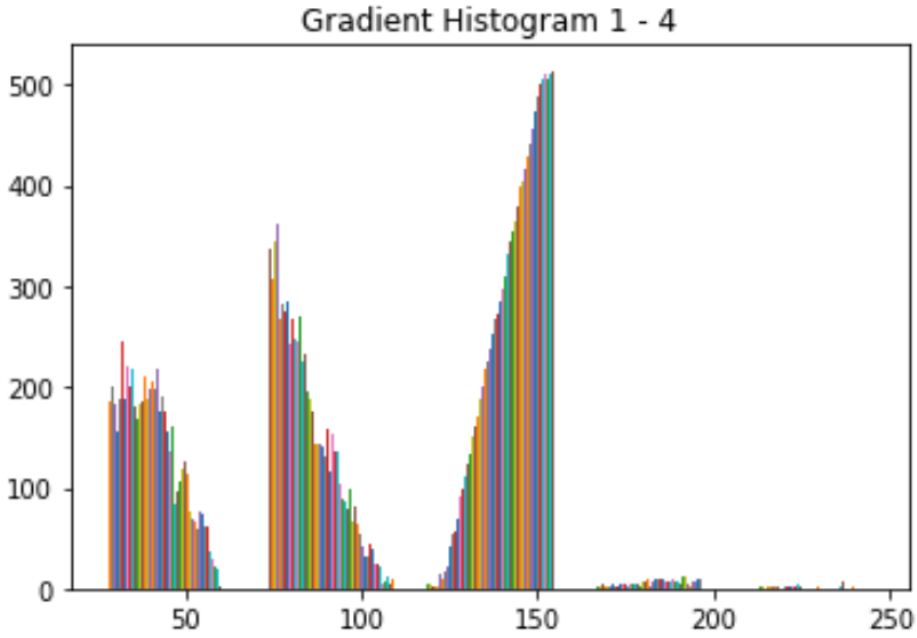
실험 결과 출력:

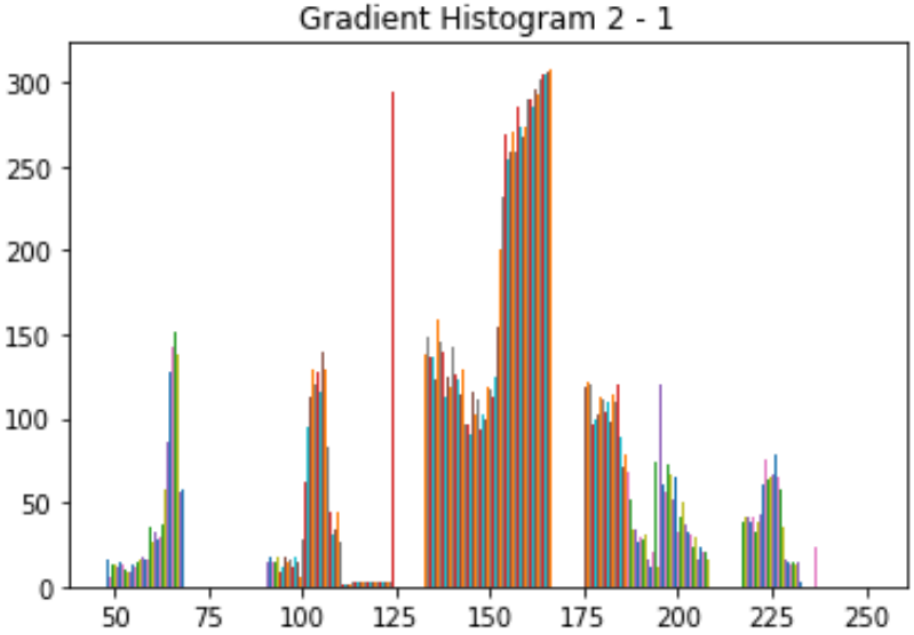


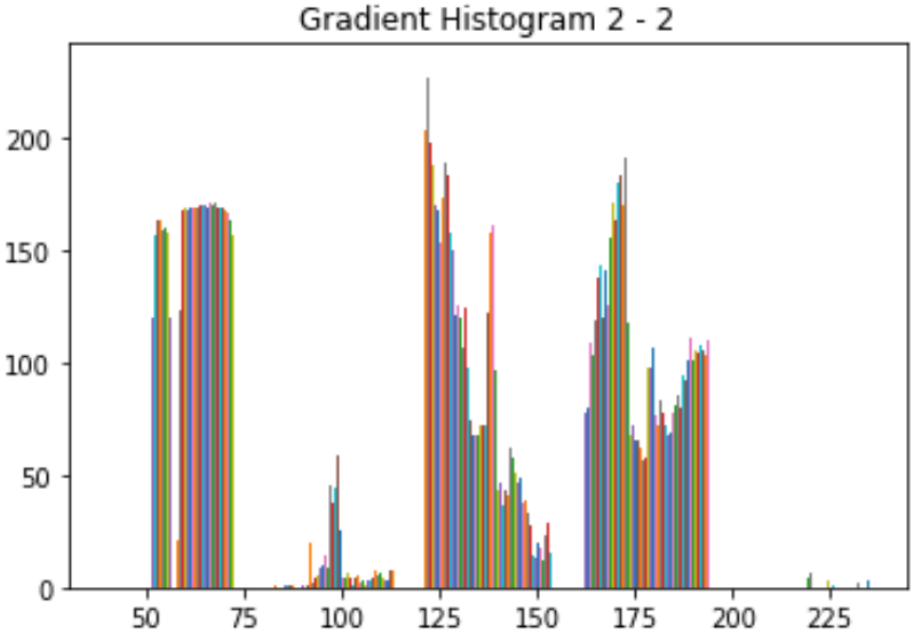

Gradient Histogram 1 - 1
300
250
200
150
100
50
0
50
75
100
125
150
175
200
225
250

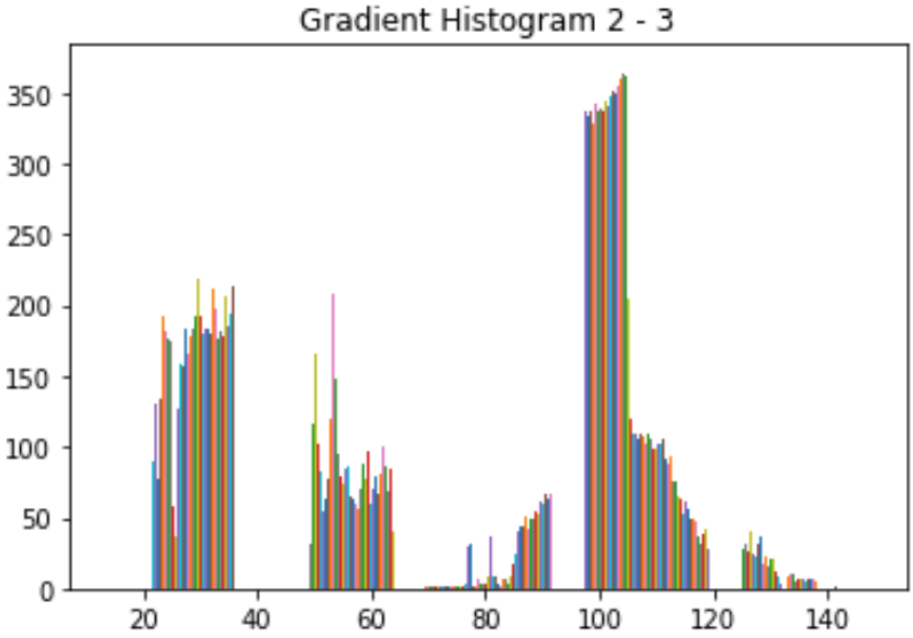

Gradient Histogram 1 - 2
400
350
300
250
200
150
100
50
0
50
75
100
125
150
175
200

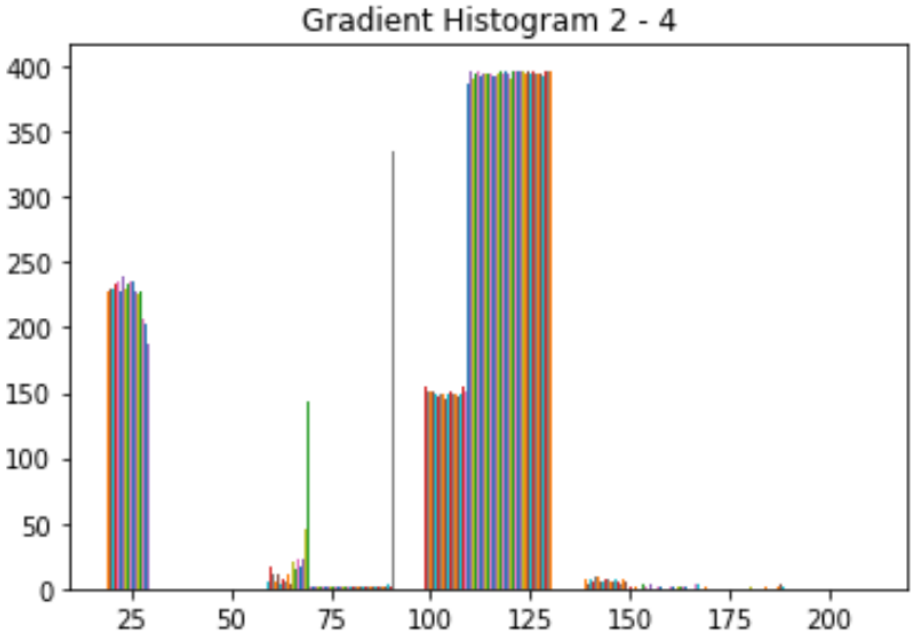

Gradient Histogram 1 - 3
350
300
250
200
150
100
50
0
20
40
60
80
100
120
140
160


Gradient Histogram 1 - 4
500
400
300
200
100
0
50
100
150
200
250


Gradient Histogram 2 - 1
300
250
200
150
100
50
0
50
75
100
125
150
175
200
225
250


Gradient Histogram 2 - 2
200
150
100
50
0
50
75
100
125
150
175
200
225


Gradient Histogram 2 - 3
350
300
250
200
150
100
50
0
20
40
60
80
100
120
140


Gradient Histogram 2 - 4
400
350
300
250
200
150
100
50
0
25
50
75
100
125
150
175
200

Ⅲ. 결 론

본 실험은 OpenCV을 이용하여 두 입력 이미지의 동일 특징 점 영역을 히스토그램 단위로 분석하였다. 서로의 대응관계를 찾는 알고리즘을 구현했는데 아직 많이 부족하다고 생각한다. 그래디언트 값의 히스토그램을 출력하고 각각 대응한 특징 점 영역의 히스토그램이 비슷하는 것을 확인했지만 이것으로 최종결과에 활용하지 못 했다. 그리고 상대적으로 OpenCV의 우수한 성능 및 활용성을 확인하였다.

Ⅳ. 부 록

https://opencv.org/

https://www.python.org/

https://numpy.org/

https://matplotlib.org/

https://github.com/jql956313805/ComputerVision

조원 역할 분담

곡우환:그래디언트 값을 계산하고 분석. 초기 Gray Histogram 및 similarity 계산의 테스트.

김천룡:Gray Histogram 및 similarity 계산의 보완. 조원의 분석결과로 종합적으로 결과출력.

College of Software, Chung-Ang University, Seoul, South Korea