HW 3 report

Group 5

0516222 許芳瑀 0516209 呂淇安 0516326陳廷達

1. Introduction

In this work, we stitch two photos(same objects of different views) together in order to get their panoramic image. In the first, we need to find out two pictures' interest points and feature description. So we can know the same features in two pictures. Because the two photos were taken in different views, we need to calculate the homography matrix by using RANSAC algorithm.

2. Implementation procedure

(a) Interest points detection & feature description by SIFT

We use the built-in functions in cv2 to get the interest points and descriptions of img1 and img2.

```
img1 = cv2.imread(imname1)
Gimg1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.imread(imname2)
Gimg2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(Gimg1, None)
kp2, des2 = sift.detectAndCompute(Gimg2, None)
```

(b) Feature matching by SIFT features

We choose brute-force method to find two points in img2 with the minimum L2 distance between img1.

```
class BFMATCH():

    def Best2Matches(self, des1, des2):
        self.match = []
        idx1 = 0
        for p1 in des1:
            best_m = []
            temp0 = cv2.DMatch(idx1, 0, math.sqrt((p1 - des2[0]).T.dot(p1 - des2[0])))
            temp1 = cv2.DMatch(idx1, 1, math.sqrt((p1 - des2[1]).T.dot(p1 - des2[1])))
            if temp0.distance < temp1.distance:
                best_m.append(temp0)
                best_m.append(temp1)
            else:
                best_m.append(temp1)
                best_m.append(temp0)

            idx2 = 0
            for p2 in des2:
                dis = math.sqrt((p1-p2).T.dot((p1-p2)))
                if dis < best_m[0].distance:
                    best_m[0].trainIdx = idx2
                    best_m[0].distance = dis
                elif dis < best_m[1].distance:
                    best_m[1].trainIdx = idx2
                    best_m[1].distance = dis
                idx2 = idx2 + 1
            idx1 = idx1 + 1
            self.match.append(best_m)
        return self.match
```

And doing ratio test, if the distance of point 1 is much smaller than distance of point 2. It means it is the match point.

```
for m in Mymatches:
    if m[0].distance < 0.8*m[1].distance:
        temp.append((m[0].trainIdx, m[0].queryIdx))
        Mm.append(m[0])
Mymatches = np.asarray(temp)
```

After getting the match result, show the picture with matching points.

```
MATCH = sorted(MATCH, key=lambda x: x.distance)

thirty_match = MATCH[:30]


Match_picture = cv2.drawMatches(img1,kp1,img2,kp2, thirty_match, Match_picture, flags=2)
```

(c) RANSAC to find homography matrix H

```
RSC = RANSAC(thresh = 10.0, n_times = 1000, points = 4)
H, Lines = RSC.ransac(CorList = CorList)
```

Based on RANSAC algorithm, we choose 4 match points to find homography matrix and calculate the distance between line and points. After doing above steps many times, we keep the best result to do next part.

```
def ransac(self, CorList):
    MaxLines = []
    AnsH = None
    Clen = len(CorList)
    for iter_1 in range(self.n_times):
        testP = []
        RanPoints = []
        ## pick up 4 random points
        Cor1 = CorList[random.randrange(0, Clen)]
        Cor2 = CorList[random.randrange(0, Clen)]
        Cor3 = CorList[random.randrange(0, Clen)]
        Cor4 = CorList[random.randrange(0, Clen)]
        RanPoints.append((Cor1, Cor2, Cor3, Cor4))# = np.vstack((Cor1, Cor2, Cor3, Cor4))

        RanPoints = np.vstack((Cor1, Cor2, Cor3, Cor4))

        ## cal H
        H = self.CalH(RanPoints)
        ## Cal line
        Lines = []
        for iter_2 in range(Clen):
            d = self.GeoDis(points = CorList[iter_2], H = H)
            if d < 5:
                Lines.append(CorList[iter_2])

        if len(Lines) > len(MaxLines):
            MaxLines = Lines
            AnsH = H

    return AnsH, MaxLines
```

```
def CalH(self, RanPointsssssss):
    AsmList = []

    for iter_pts in RanPointsssssss:
        p1 = np.array([iter_pts.item(0), iter_pts.item(1), 1])
        p2 = np.array([iter_pts.item(2), iter_pts.item(3), 1])
        a2 = [0, 0, 0, -p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2), p2.item(1) * p1.item(0), p2.item(1) * p1.item(
        a1 = [-p2.item(2) * p1.item(0), -p2.item(2) * p1.item(1), -p2.item(2) * p1.item(2), 0, 0, 0, p2.item(0) * p1.item(0), p2.item(0) * p1.item(
        AsmList.append(a1)
        AsmList.append(a2)

    AsmMtx = np.matrix(AsmList)
    U, Sigma, Vt = np.linalg.svd(AsmMtx)

    pre_H = np.reshape(Vt[8], (3, 3))
    H = (1 / pre_H.item(8)) * pre_H
    return H
```

```
def GeoDis(self, points, H):
    point1 = np.transpose(np.array([points[0], points[1], 1]))
    Estm = np.dot(H, point1)
    Estm2 = (1 / Estm.item(2)) * Estm

    point2 = np.transpose(np.array([points[2], points[3], 1]))
    Err = point2 - Estm2
    return np.linalg.norm(Err)
```

(d) Warp image to create panoramic image

We wrap img1 onto img2 with homography matrix we got in previous step and fix some pixels by linear interpolation.

```
    def warp(self, img, H, outputShape = None):
        outputShape = tuple(outputShape)
        NewImg = self.Realwarp(img, H, outputShape)
        return NewImg

    def Realwarp(self, img, H, outputShape):
        MP = self.CaPrTr(
            np.arange(outputShape[0], dtype = np.float32),
            np.arange(outputShape[1], dtype = np.float32)
        )
        MP = MP.reshape(outputShape[0], outputShape[1], 2)
        MP = self.perspectiveTransform(MP, inv(np.array(H)))
        return np.swapaxes(self.reMP(img, MP), 0, 1)

    def CaPrTr(self, *arrays):
        la = len(arrays)
        dtype = np.result_type(*arrays)
        arr = np.empty([la] + [len(a) for a in arrays], dtype = dtype)
        for i, a in enumerate(np.ix_(*arrays)):
            arr[i, ...] = a
        return arr.reshape(la, -1).T


    def perspectiveTransform(self, MP, M):
        t1 = ((M[0, 0] * MP[:, :, 0]) + (M[0, 1] * MP[:, :, 1]) + M[0, 2]) / (
                    (M[2, 0] * MP[:, :, 0]) + (M[2, 1] * MP[:, :, 1]) + M[2, 2])
        t2 = ((M[1, 0] * MP[:, :, 0]) + (M[1, 1] * MP[:, :, 1]) + M[1, 2]) / (
                    (M[2, 0] * MP[:, :, 0]) + (M[2, 1] * MP[:, :, 1]) + M[2, 2])
        return np.stack((t1, t2), axis = 2)
```

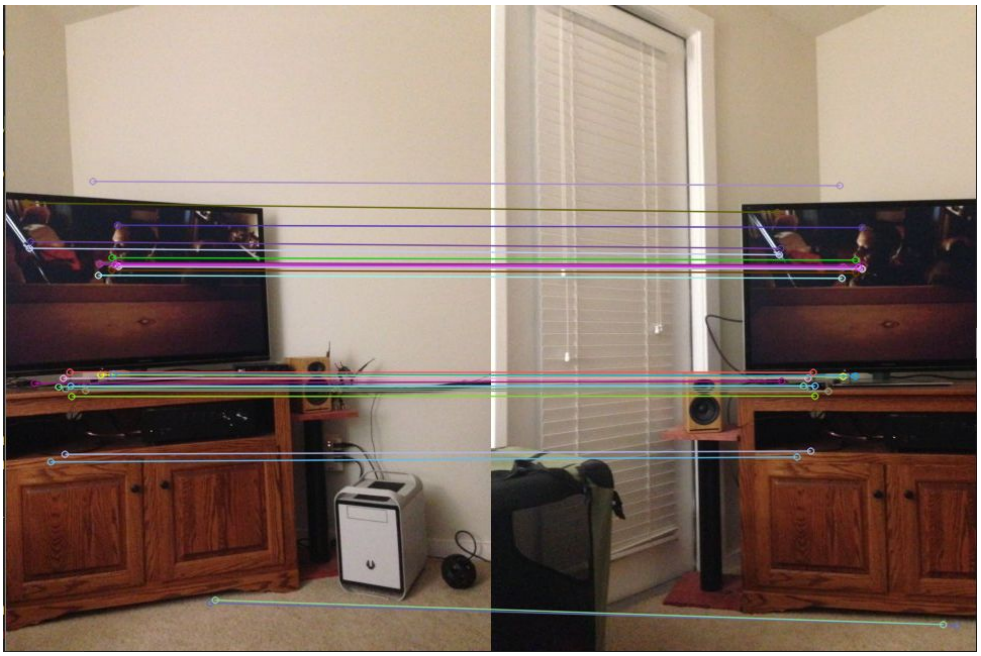
```
def reMP(self, src, MP):
    RMP = np.zeros((MP.shape[0],MP.shape[1],src.shape[2]),dtype=np.uint8)
    height = MP.shape[0]
    width = MP.shape[1]
    print("Doing mapping and bilinear interpolation...\n")
    for i in range(height):
        for j in range(width):
            x = MP[i][j][1]
            y = MP[i][j][0]
            x1 = int(x)
            x2 = x1+1
            y1 = int(y)
            y2 = y1+1
            if x1 >= 0 and y1 >= 0 and x2 < src.shape[0] and y2 < src.shape[1]:
                # Bilinear interpolation
                RMP[i][j] = (src[x1][y1] * (x2 - x)*(y2 - y) + src[x1][y2] * (x2 - x) * (y - y1) + src[x2][y1] * (x - x1) * (y2 - y) + src[x2][y2]
    return RMP
```

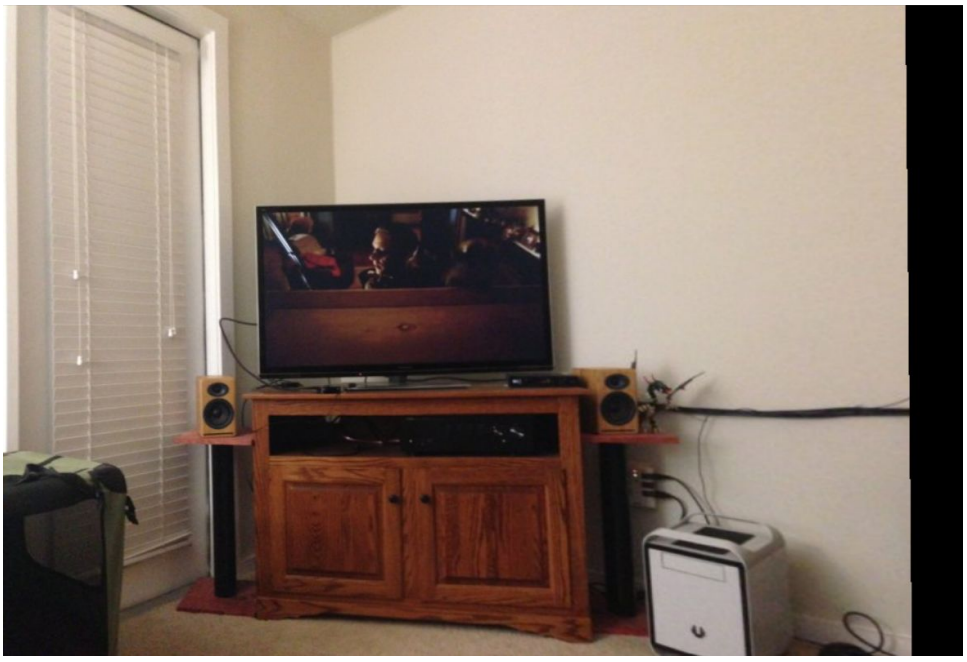
The last step is stitch two pictures to get the panoramic image.

```
leftest_overlap = img2.shape[1]
for i in range(0,img2.shape[0]):
    for j in range(0,img2.shape[1]):
        if any(v != 0 for v in ResultImg[i][j]):
            leftest_overlap = min(leftest_overlap, j)
# to the left
for i in range(0,img2.shape[0]):
    for j in range(0,img2.shape[1]):
        if any(v != 0 for v in ResultImg[i][j]): # overlapped pixel
            # Linear(alpha) Blending
            alpha = float(img2.shape[1]-j)/(img2.shape[1]-leftest_overlap)
            ResultImg[i][j] = (ResultImg[i][j] * (1-alpha) +  img2[i][j] * alpha).astype(int)
        else:
            ResultImg[i][j] = img2[i][j]
```

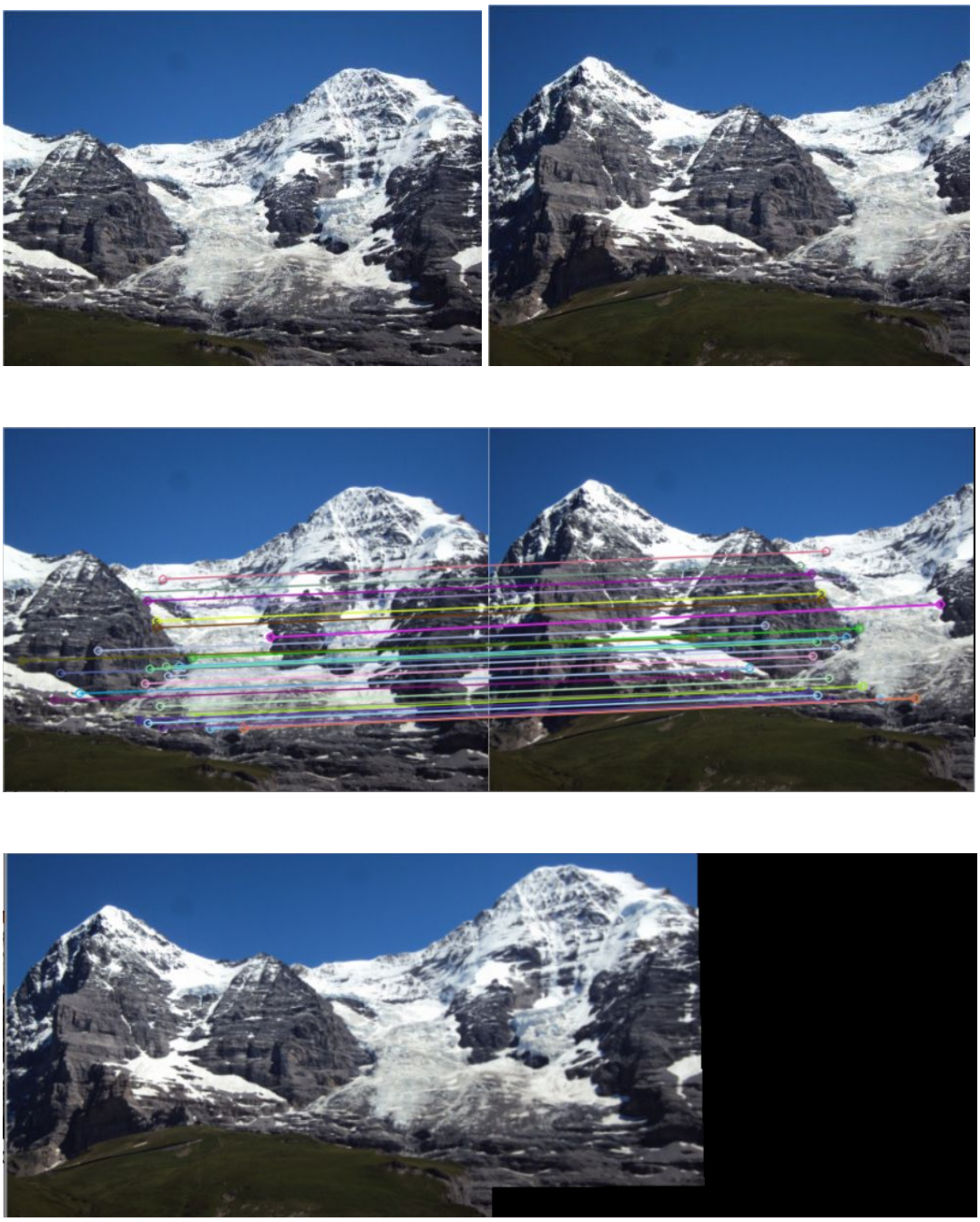
3. Experimental results (of course you should also try your own images)

   (a)



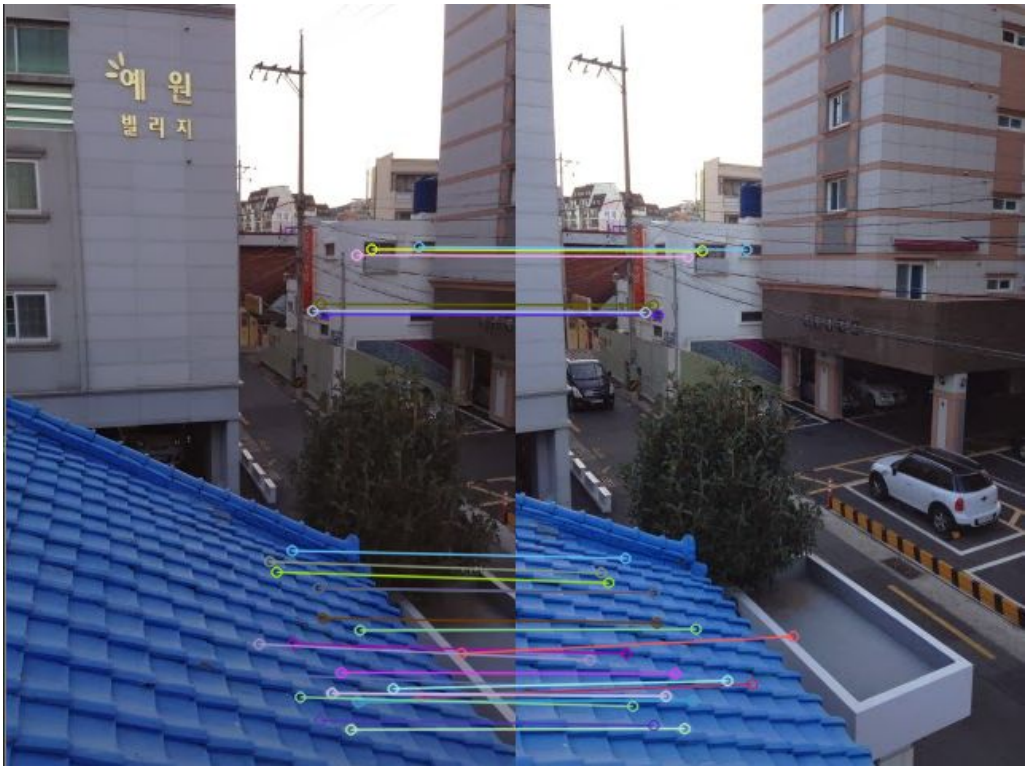


(b)

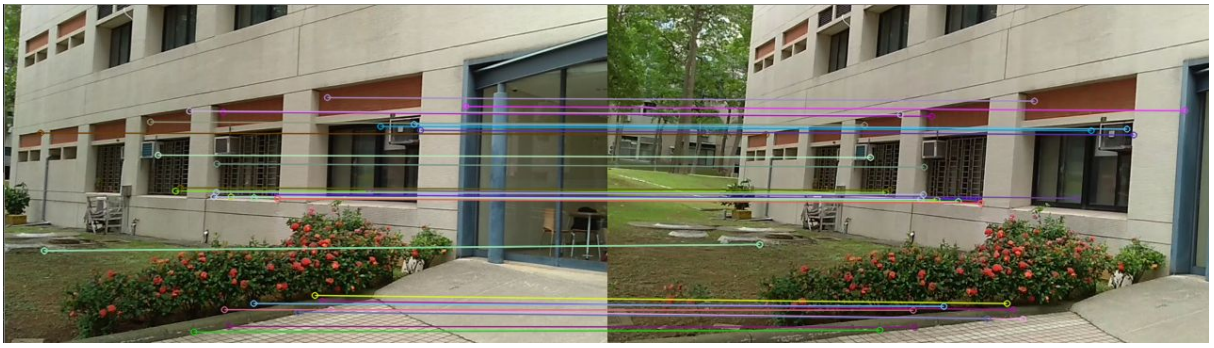


(c)

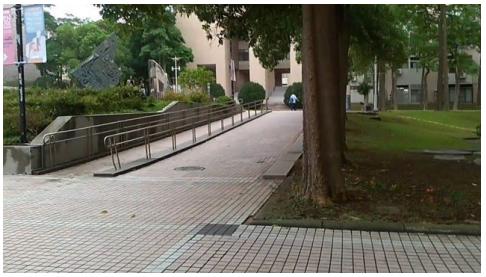

예 원
빌 리 지

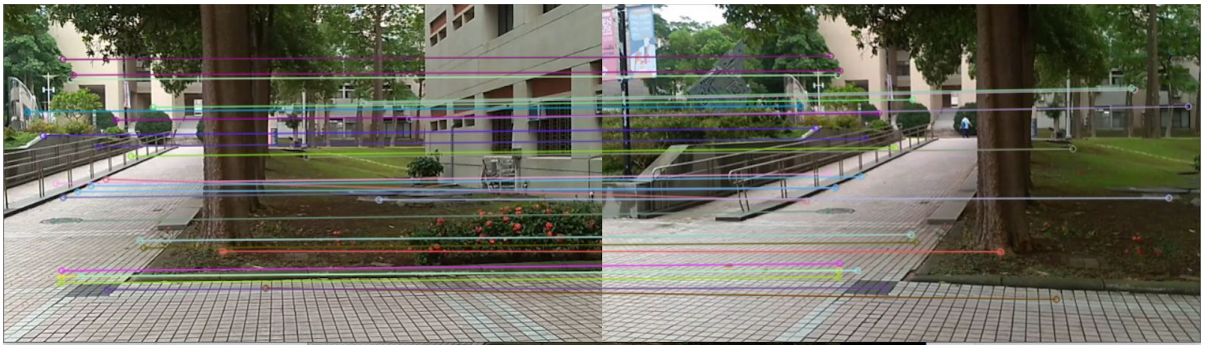

예 원
빌 리 지

(d)



(e)





4. Discussion (what difficulties you have met? how you resolve them?)
   (a) In part1 we want to use cv2.xfeatures2d, but this function only exits in older python version. Thus we install related packages, but it didn't work. We didn't know why it didn't work, so we create virtual environment and re-install many times. Finally it works successfully.
   (b) When stitching two pictures, we always suppose the image 1 is the left part of the panoramic image. So if we change the order of reading pictures, the result will not as good as the former one.

5. Conclusion

In this work, we implement simple image stitching and also tried to run it on our own pictures. This work help us be familiar with the transformation of different views with homography matrix and find out the features of pictures. These methods in computer vision are important concepts. We believe this experience is really good and meaningful for our future works.

6. Work assignment plan between team members.

Code implementation : 陳廷達、呂淇安
Parameter adjustment : 許芳瑀、陳廷達
Report : 呂淇安、許芳瑀