Neha Bisht
University at Buffalo
Department of Computer Science and Engineering
CSE 473/573 - Computer Vision and Image Processing
Fall 2021
Project #2

Task 1 Report
cv version used is 3.4.2.17

Image Stitching

The goal of this task is to stitch two images (named "left.jpg" and "right.jpg") together to construct a panorama image.

Steps -

1. Extract key points
2. Match Keypoint
3. Estimate Homography
4. Warp & Stitch Images

1. Extract key points

To extract the key points the image is converted to greyscale to reduce the complexity. The detector I used to find the key points and features of the image is SIFT detector as its invariant to scale. The getKeyPointsNFeatures(image) function will take an image as input and return key points and features as output.

```
#function will return the key points and features of the image
def getKeyPointsNFeatures(image):
    grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    siftDescriptor = cv2.xfeatures2d.SIFT_create()
    (keyPts, features) = siftDescriptor.detectAndCompute(image, None)
    return (keyPts, features)
```

grayImage	-> gray scale Image of input image
siftDescriptor	-> SIFT filter object
keyPts	-> key points of the image
features	-> features of the image

2. Match Keypoint

Our target is to find the two matching features in the right image for every feature in the left image. To implement this we will compare features using KNN distance where k=2.

Firstly, we find the 2-norm distance between each key feature in the left image with every key feature in the right image. After getting the 2-norm distance for every point we will sort the matches in ascending order of the norm distance. As we know, the lesser the norm distance more is the resemblance between the two features. After getting the two best matches for a feature we will use ratio testing to filter the good matches.

According to the ratio test, the top 2 best matches should not be ambiguous. Let sortedDist store the norm distance in the ascending order between a point in the left image and all the points in the right image then the sortedDistances(0) and sortedDistances(1) are the two best matches for a feature. Check the below condition to filter out the good matches.

```
if sortedDistances(0) < $\eta$ * sortedDistances(1) then
        Good match add in the goodFeatures list
else
        Ignore ambiguous match
```

Here η is just a ratio. Smaller η filters out more matching candidates.

```python
#store the good matches
goodMatches = []
for i in range(len(featuresLeft)):
    distances = []
    for j in range(len(featuresRight)):
        distanceInfo = {"leftIndex": i,
                        "rightIndex": j,
                        "normDist": np.sum(np.square(np.subtract(featuresLeft[i],featuresRight[j])))}
        distances.append(distanceInfo)
    sortedDistances = []
    sortedDistances = sorted(distances, key = lambda x:x["normDist"])
    #best two matches of a feature
    firstMatch = sortedDistances[0]["normDist"]
    secondMatch = sortedDistances[1]["normDist"]
    #ratio check
    if firstMatch < ratioFilter*secondMatch:
        goodMatches.append(sortedDistances[0])
```

goodMatches	-> stores the good features
ratioFilter	-> 0.07
distances	-> stores leftIndex, rightIndex and distance of leftIndex from rightIndex
sortedDistances	-> stores distances, sorted by normDist
firstMatch	-> best match for a feature
secondMatch	-> second best match for a feature

3. Estimate Homography

Now we will find the homography matrix which is the mapping between two projection planes with the same center of projection.

$$p' = Hp$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

In order to solve H, at least 4 key points matching pairs are needed. To find 4 key points we will use RANSAC (Random sample consensus). The homography matrix with 4 inliers will be the required homography.

Steps to find the inliers

1. We will select 4 random points from good matches to form the homography matrix for this getRandomMatches(goodMatches) method is used.

```
#function will return 4 random matches
def getRandomMatches(goodMatches):
    randomMatches = []
    random.seed(4)
    n=len(goodMatches)
    vx1 = [random.randint(0,n-1) for i in range(4)]
    i=0
    while i<4:
        randomMatch = goodMatches[vx1[i]]
        if randomMatch not in randomMatches:
            randomMatches.append(randomMatch)
        i=i+1
    return randomMatches
```

2. After getting the four random matches will form the homography matrix.

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n\text{x}9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\substack{\mathbf{h} \\ 9\text{x}1}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\substack{\mathbf{0} \\ 2n\text{x}1}}$$

Minimize $||Ah - 0||^2$, can be solved with SVD

$$U\Sigma V^T = SVD(A)$$

Solution: $\hat{h}$ should be the last row of V^T

getHomographyMatrix(randomMatches, keyPoints) will return the homography matrix for the random matches and key points.

```python
#function will return the homography matrix for matches
def getHomographyMatrix(randomMatches, keyPtsLeft, keyPtsRight):
    A = []
    for match in randomMatches:
        x, y = keyPtsLeft[match["leftIndex"]].pt
        u, v = keyPtsRight[match["rightIndex"]].pt
        A.append([-x, -y, -1, 0, 0, 0, u*x, u*y, u])
        A.append([0, 0, 0, -x, -y, -1, v*x, v*y, v])
    A = np.asarray(A)
    U, S, Vh = np.linalg.svd(A)
    H = np.ndarray.reshape(Vh[8], (3, 3))
    H = (1/H.item(8)) * H
    return H
```

keyPtsLeft	-> left image key points
keyPtsRight	-> right image key points
H	-> homography matrix

3. After getting the homography matrix we will check for inliers. If the homography matrix has at least 4 inliers we are considering that as the correct homography.

```python
#finding homography using RANSAC algorithm
max_inliers = 0
maxIterations = 5000
homographyMatrix = []
for i in range(maxIterations):
    randomMatches = getRandomMatches(goodMatches)
    tempHomography = getHomographyMatrix(randomMatches, keyPtsLeft, keyPtsRight)
    inliers = 0
    for match in goodMatches:
        #getting the coordinates of the key point
        x1,y1 = keyPtsLeft[match["leftIndex"]].pt
        x2,y2 = keyPtsRight[match["rightIndex"]].pt

        #using point1 of left image will try to find point2 using homography matrix
        point1 = np.matrix([[x1], [y1], [1]])
        point2 = np.matrix([[x2], [y2], [1]])
        estimatedPoint = np.dot(tempHomography, point1)
        if estimatedPoint[2] != 0:
             estimatedPoint = estimatedPoint/estimatedPoint[2]

        #checking the difference between the estimatedPoint and point2
        difference = np.linalg.norm(estimatedPoint - point2)
        if difference < 1:
            inliers += 1

    if inliers > max_inliers:
        max_inliers = inliers
        homographyMatrix = tempHomography

    if max_inliers >= 4:
        break;
```

Steps to check if a point is an inlier -

1. Take a point in the left image
2. Using homography matrix find the estimated point of the right image
3. If the difference between the estimated point and correct point in the right image is less than the threshold consider the point as an inlier

4. Warp & Stitch Images

For stitching and wrapping, we will use the homography matrix to stitch the left and the right image. To achieve this we used perspectiveTransform and warpPerspective method of cv2.

```
#wrapping image using homography matrix
left_height = left_img.shape[1]
left_width = left_img.shape[0]
right_height = right_img.shape[1]
right_width = right_img.shape[0]

right_frame = np.float32([[0, 0],
                           [0, right_width],
                           [right_height, right_width],
                           [right_height, 0]]).reshape(-1, 1, 2)
left_frame = np.float32([[0, 0],
                          [0, left_width],
                          [left_height, left_width],
                          [left_height, 0]]).reshape(-1, 1, 2)
right_transformed = cv2.perspectiveTransform(left_frame, homographyMatrix)

final_image_frame = np.vstack((right_frame, right_transformed))
[minx, miny] = np.int32(final_image_frame.min(axis=0).flatten())
[maxx, maxy] = np.int32(final_image_frame.max(axis=0).flatten())

translation_dist = [-minx, -miny]
h_translation = np.array([[1, 0, translation_dist[0]],
                          [0, 1, translation_dist[1]],
                          [0, 0, 1]])
product = h_translation.dot(homographyMatrix)
wrapped_img = cv2.warpPerspective(left_img,
                                  product,
                                  (maxx - minx, maxy - miny))
wrapped_img[translation_dist[1]:right_width + translation_dist[1],
            translation_dist[0]:right_height + translation_dist[0]] = right_img
return wrapped_img
```

PARAMETERS USED

S.No.	Parameter	Value
1	Ratio η	0.07
2	RANSAC threshold	1
3	Detector	SIFT
4	RANSAC max iterations	5000