
BBM418 Introduction to Computer Vision Lab.

Assignment 2

Image Stitching with Keypoint Descriptors

Ayşe Balci
21726985
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21726985@cs.hacettepe.edu.tr

Overview

In this assignment, I merged sub-images by using keypoint description methods SIFT, and ORB and obtain a final panorama image that including all scenes in the sub-images.

1 Part 1 - Feature Extraction

Feature detection is the making a local decision at every image point to see if there is an image feature of the given type existing in that point.

1.1 Feature Extraction with SIFT

SIFT is a feature extraction method where image content is transformed into local feature coordinates that are invariant to translation, scale and other image transformations.

```
def extractFeatureWithSIFT(img):  
  
    # extract features of image  
    extractor = cv2.SIFT_create()  
    kp, des = extractor.detectAndCompute(img, None)  
  
    return kp, des
```

1.2 Feature Extraction with ORB

```
def extractFeatureWithORB(img):  
  
    # extract features of image  
    extractor = cv2.ORB_create()  
    kp, des = extractor.detectAndCompute(img, None)  
    return kp, des
```

2 Part 2 - Feature Matching

Descriptors are compared across the images, to identify similar features. For two images we may get a set of pairs (X_i, Y_i) (X_i', Y_i') , where (X_i, Y_i) is a feature in one image and (X_i', Y_i') its matching feature in the other image.

2.1 Feature Matching with SIFT

I create the Brute Force matcher and here I used the K- nearest neighbor matches which yields the Matches based on the similarity distances and let us further filter this out only considering if the distance between the two matches as good if the distance between them is 75

```
def matchFeatures(feature1, feature2):
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(feature1, feature2, k=2)

    good = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good.append(m)

    return good
```

3 Part 3 - . Finding Homography

The homography is an isomorphism of projective spaces and has been historically used to explain and study the difference in the appearance of two planes observed from different points of view. Homographies can be represented as a 3 x 3 matrix which can be left-multiplied with any homogeneous point in the original image to describe where that point lies in the transformed image.

RANSAC algorithm is a general parameter estimation approach to compensate for a large proportion of outliers in the data.

$$\begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \Leftrightarrow \begin{cases} uh_1 + vh_2 + h_3 = u' \\ uh_4 + vh_5 + h_6 = v' \\ uh_7 + vh_8 + h_9 = 1 \end{cases} \Leftrightarrow \begin{bmatrix} 0 & 0 & 0 & -u & -v & -1 & u'u & v'u & v' \\ u & v & 1 & 0 & 0 & 0 & -u'u & -u'v & -u' \\ -v'u & -v'v & -v' & u'u & u'v & u' & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow A\mathbf{h} = \mathbf{0}$$

I randomly picked four good matches, compute a homography from these four. The vh matrix is returned in a transposed form. Added an extra constraint $|H|=1$ to avoid the obvious solution of H being all zeros, then I used `np.linalg.svd` for matrix and selected the last singular vector of A.

```
def calculate_homography(point1, point2):

    # calculate homography matrix from 4 points
    A = np.array([
        [point1[0][0][0], point1[0][0][1], 1, 0, 0, 0, -point2[0][0][0] *
         point1[0][0][0], -point2[0][0][0] * point1[0][0][1], -point2[0][0][0][0]],
```

```

[0, 0, 0, point1[0][0][0], point1[0][0][1], 1, -point2[0][0][1] *
point1[0][0][0], -point2[0][0][1] * point1[0][0][1], -point2[0][0][1]],

[point1[1][0][0], point1[1][0][1], 1, 0, 0, 0, -point2[1][0][0] *
point1[1][0][0], -point2[1][0][0] * point1[1][0][1], -point2[1][0][0]],
[0, 0, 0, point1[1][0][0], point1[1][0][1], 1, -point2[1][0][1] *
point1[1][0][0], -point2[1][0][1] * point1[1][0][1], -point2[1][0][1]],

[point1[2][0][0], point1[2][0][1], 1, 0, 0, 0, -point2[2][0][0] *
point1[2][0][0], -point2[2][0][0] * point1[2][0][1], -point2[2][0][0]],
[0, 0, 0, point1[2][0][0], point1[2][0][1], 1, -point2[2][0][1] *
point1[2][0][0], -point2[2][0][1] * point1[2][0][1], -point2[2][0][1]],

[point1[3][0][0], point1[3][0][1], 1, 0, 0, 0, -point2[3][0][0] *
point1[3][0][0], -point2[3][0][0] * point1[3][0][1], -point2[3][0][0]],
[0, 0, 0, point1[3][0][0], point1[3][0][1], 1, -point2[3][0][1] *
point1[3][0][0], -point2[3][0][1] * point1[3][0][1], -point2[3][0][1]]
])

u, s, vh = np.linalg.svd(A)
homography = (vh[-1, :] / vh[-1, -1]).reshape(3, 3)
return homography

```

I tested how good this homography is by checking how many of the good matches are consistent with the homography. Good matches that are consistent with the homography are called inliers, and those that aren't are called outliers. We count the number of outliers for this homography. Then, we repeat this 1000 times (picking a set of four good matches anew each time), in each iteration deriving a homography and counting the number of outliers associated with it. We keep the homography with the smallest number of outliers.

For the numerical calculation issues in transformation algorithm I applied normalization process. I computed the homography by normalized transformation from the 4 points pairs, for each source and destination point pair, calculated distance by the homography. The criterion I used to distinguish inliers from outliers is based on a maximum absolute threshold from which an inlying pair of corresponding points. I choosed the threshold value as 5 to determine data points that are fit well by model. The ransac funtion returns the *best_homography_model_parameters_which_best_fit_the_data(or None if no good model is found)*.

```

def ransac(src_pts, dst_pts, threshold=5, maxIters=1000):
    # ransac algorithm for finding best homography matrix

    src_pts = np.dstack((src_pts, np.ones((src_pts.shape[0], src_pts.shape[1]))))
    dst_pts = np.dstack((dst_pts, np.ones((dst_pts.shape[0], dst_pts.shape[1]))))
    best_count_matches = 0

    for iteration in range(maxIters):
        # get four random points
        random_indices = np.random.randint(0, len(src_pts) - 1, 4)
        random_kp_src, random_kp_dst = src_pts[random_indices], dst_pts[random_indices]

        # calculate a homography
        homography = calculate_homography(random_kp_src, random_kp_dst)
        count_matches = 0

        for i in range(len(src_pts)):

```

```

        unnorm = np.dot(homography, src_pts[i][0])
        normalized = (unnorm / unnorm[-1])

        # calculate norm
        norm = np.linalg.norm(normalized - dst_pts[i][0])
        if norm < threshold:
            count_matches += 1

    if count_matches >= best_count_matches:
        # update best homography if better match found
        best_count_matches = count_matches
        homography_best = homography
    return homography_best

```

4 Part 4 - Merging by Transformation

In the final step, the homography matrix is applied to the image to wrap and fit those images and merge them into one. I applied a function called create panorama for two images. I created a base image by array's first element. Then I created a loop for dataset all images. In the loop I send base image with next image in dataset to function, and equalize base image to created panorama image.

In creating panorama step, firstly I created a function to find match points. I created 2 edge points array for 2 image, then computed transformation using homography matrix by using second edge point array. After that, created an array that contains all match points from first edge point array and calculated edge point arrays. Finally, found panorama edges coordinates by minimum and maximum of match points and round to integer.

```

def calculate_match_points(img1, img2, homography):
    h1, w1, _ = img1.shape
    h2, w2, _ = img2.shape

    # edge points from image 1
    edge_points_1 = np.float32([[0, 0], [0, h1], [w1, h1], [w1, 0]]).reshape(-1, 1, 2)

    # edge points from image 2
    edge_points_2 = np.float32([[0, 0], [0, h2], [w2, h2], [w2, 0]]).reshape(-1, 1, 2)

    # computes transformation using homography matrix
    new_points = []
    for point in edge_points_2:
        p = np.dot(homography, np.array([point[0][0], point[0][1], 1]))
        p = p / p[-1]
        new_points.append(p[:-1])
    new_points = np.array(new_points, dtype=np.float32).reshape(edge_points_2.shape)

    # match points
    matches = np.vstack((edge_points_1, new_points))

    # find new edges coordinate and round
    x_min, y_min = np.int32(matches.min(axis=0).ravel() - 0.5)
    x_max, y_max = np.int32(matches.max(axis=0).ravel() + 0.5)

    return x_min, y_min, x_max, y_max

```

I started the create panorama function by calculating the edge points above, then I calculate new height and weights by subtraction of edge points. Then I find the transformation matrix that will stitch the 2 images together based on their matching points by using homography matrix. Then with this matrix, I calculated new coordinates of panorama and fill it.

```
def create_panorama(img1, img2, homography):
    # stitch 2 images using homography matrix
    h1, w1, _ = img1.shape
    h2, w2, _ = img2.shape

    x_min, y_min, x_max, y_max = calculate_match_points(img1, img2, homography)

    new_height = y_max - y_min
    new_width = x_max - x_min

    H_translation = np.array([[1, 0, -x_min], [0, 1, -y_min], [0, 0, 1]])
    homography = np.dot(H_translation, homography)
    homography = np.linalg.inv(homography)

    coordinates = np.zeros((new_height, new_width, 2))

    for i in range(new_height):
        for j in range(new_width):
            unnorm = np.dot(homography, np.array([j, i, 1]))
            coordinates[i, j, :] = (unnorm / unnorm[-1])[:-1]

    # new coordinates after homography
    coordinates = coordinates.reshape(-1, 2)

    # join colors
    x = coordinates[:, 0]
    y = coordinates[:, 1]

    # for each pixel is x0, x1, y0, y1
    x0, y0 = np.floor(x).astype(int), np.floor(y).astype(int)
    x1, y1 = x0 + 1, y0 + 1

    x0 = np.clip(x0, 0, img2.shape[1] - 1)
    x1 = np.clip(x1, 0, img2.shape[1] - 1)
    y0 = np.clip(y0, 0, img2.shape[0] - 1)
    y1 = np.clip(y1, 0, img2.shape[0] - 1)

    # weighted sum of 4 nearest neighbours
    panorama = (img2[y0, x0].T * (x1 - x) * (y1 - y)).T \
        + (img2[y1, x0].T * (x1 - x) * (y - y0)).T \
        + (img2[y0, x1].T * (x - x0) * (y1 - y)).T \
        + (img2[y1, x1].T * (x - x0) * (y - y0)).T

    panorama = panorama.reshape(new_height, new_width, 3)
    panorama = panorama.astype(np.uint8)

    panorama[-y_min:h1 - y_min, -x_min:w1 - x_min] = img1

    return panorama
```

The main function is shown in below. All functions are calling from main step by step.

```
if __name__ == "__main__":
    dataset_dir = Path('cvc01passadis-cyl-pano01')
    imgs = [img for img in dataset_dir.glob("/*.png")]
    imgs = [cv2.imread(str(img)) for img in imgs]
    img_list = []
    base_image = imgs[0]

    for i in range(0, len(imgs)):
        src_img = base_image
        dst_img = imgs[i]



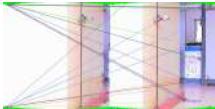

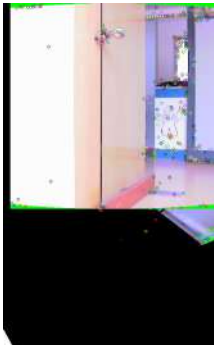

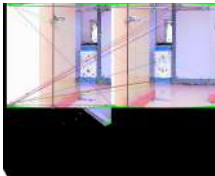

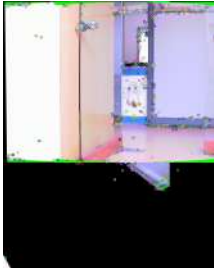
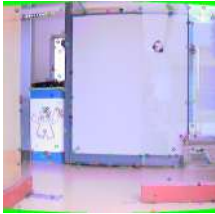
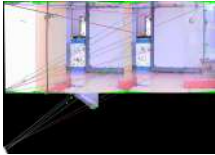
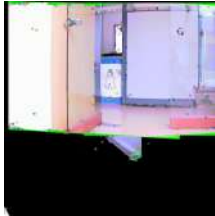
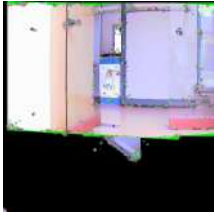

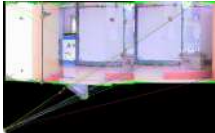
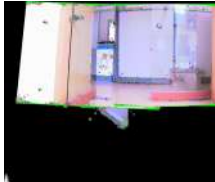
        kp1, des1 = extractFeatureWithSIFT(src_img)
        kp2, des2 = extractFeatureWithSIFT(dst_img)

        good = matchFeatures(des1, des2, kp1, kp2, src_img, dst_img, str(dataset_dir), str(i))

        if len(good) > 1:
            src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
            dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

            homography = findHomography(kp1, kp2, good)
            if homography is not None:
                print(i, ' is homography ')
                panorama = create_panorama(dst_img, src_img, homography)
                base_image = panorama
                cv2.imshow('output_image', base_image)
                cv2.waitKey(0)
            else:
                base_image = dst_img
                cv2.imshow('output_image', base_image)
                cv2.waitKey(0)
        else:
            base_image = dst_img
            cv2.imshow('output_image', base_image)
            cv2.waitKey(0)
```



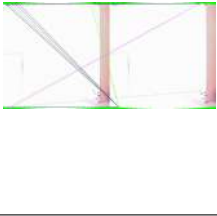



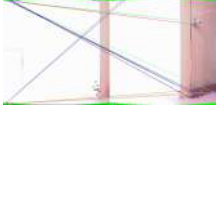




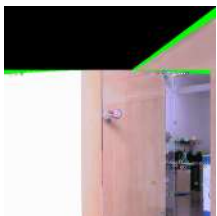

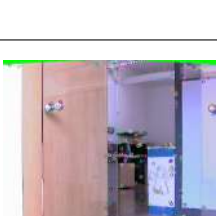

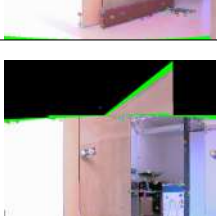






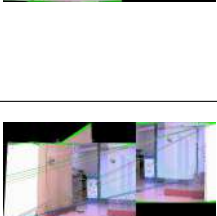


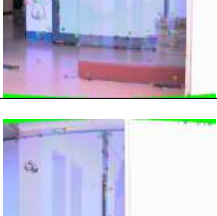


5 Successful Panoramas With SIFT - cvc01passadis-cyl-pano01





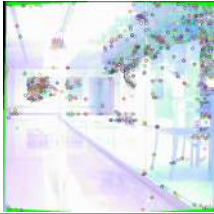
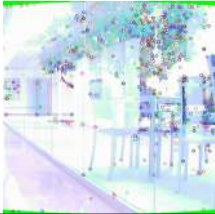
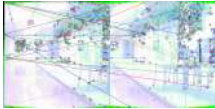



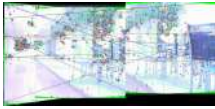






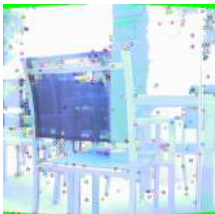
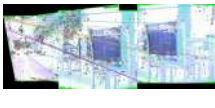


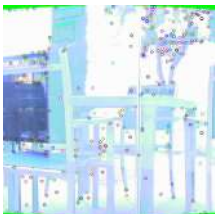

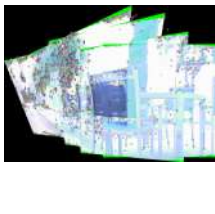
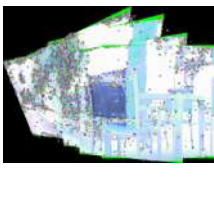
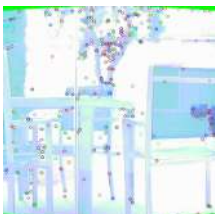
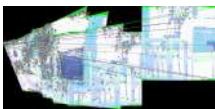
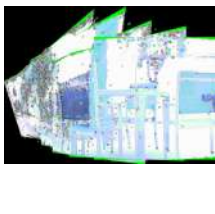
1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			

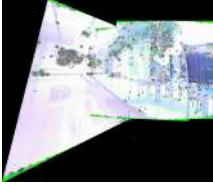

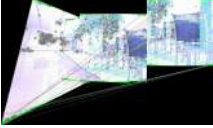
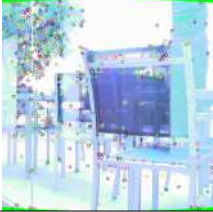
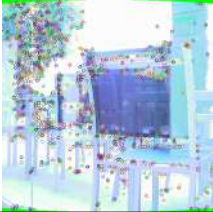
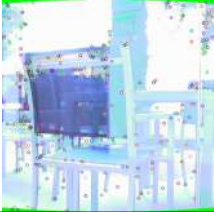


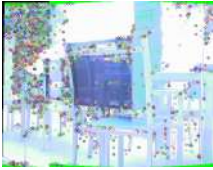
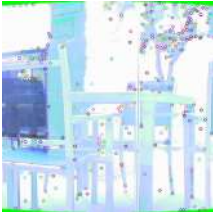


6 Successful Panoramas With ORB - cvc01passadis-cyl-pano01

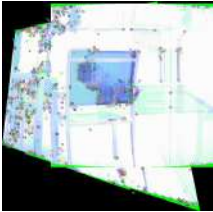

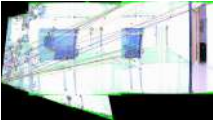



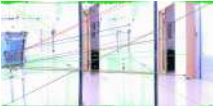
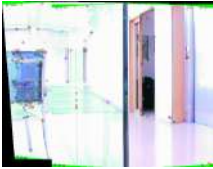
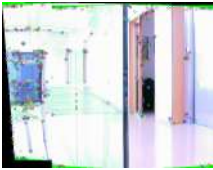


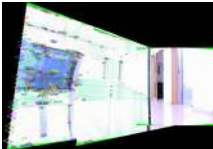
1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			
			
			

7 Successful Panoramas With SIFT - cvc01passadis-cyl-pano05

1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			
			
			
			

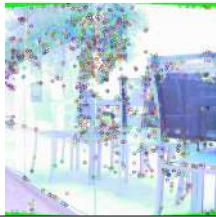



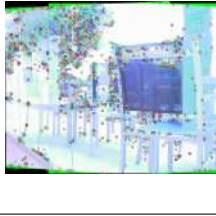
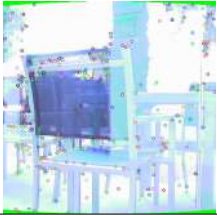


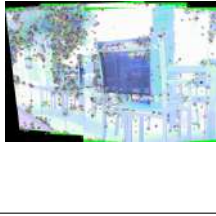
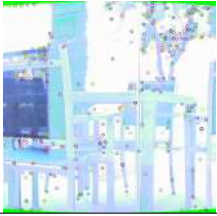


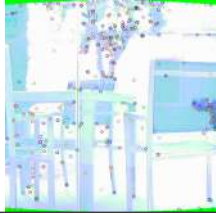
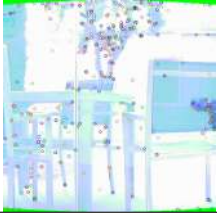


1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			
			
			
			






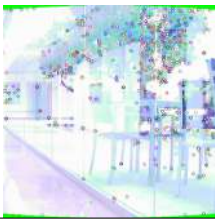

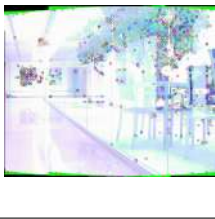
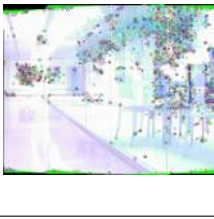






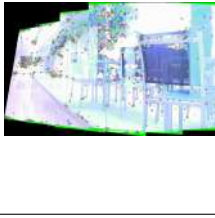
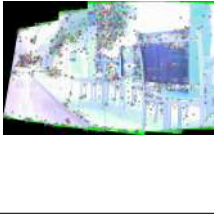
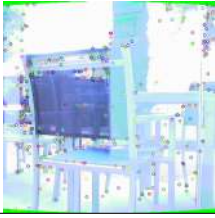

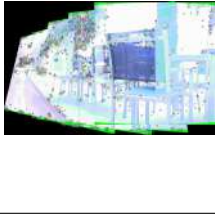
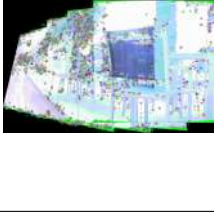


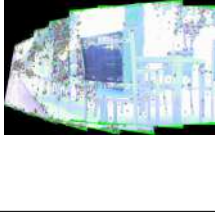
1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			

1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			














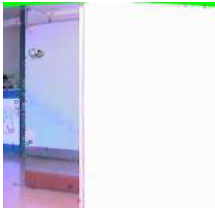


8 Successful Panoramas With ORB - cvc01passadis-cyl-pano05

1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			
			
			
			



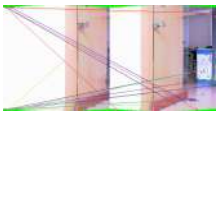



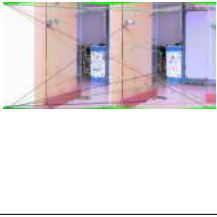



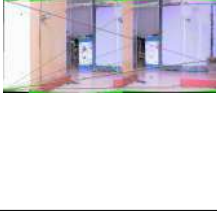
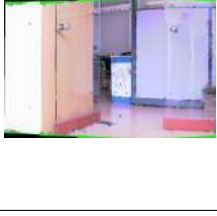
1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			

1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			
			
			

9 Successful Panoramas With SIFT - cvc01passadis-cyl-pano07

1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			
			

10 Successful Panoramas With ORB - cvc01passadis-cyl-pano07

1'st SIFT Features	2'nd SIFT Features	Feature Matching	Panorama Images
			
			
			

11 Conclusion

Runtime and accuracy comparison of the description methods (SIFT/SURF and ORB)

Description methods	Runtime Comparison (min)	Accuracy Comparison (%)
SIFT/SURF	11.5	59.09
ORB	5.7	46.04

To sum up, ORB is the fastest algorithm while SIFT/SURF performs the best.

References

Your references here.

[1] https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html

[2] <https://medium.com/all-things-about-robotics-and-computer-vision/homography-and-how-to-calculate-it-8abf3a13ddc5>