BBM 418 - Computer Vision Laboratory

Assignment #2 - Image Stitching with Keypoint Descriptors

Emirhan Topcu

21827899

Department of Computer Engineering

Hacettepe University

Ankara, Turkey

emirhantopcu50@gmail.com

Overview

In this assignment, the goal is using keypoint descriptors to merge sequential images into a panoramic picture. A homography matrix needs to be calculated for this purpose. We used RANSAC algorithm to find an optimal value for homography matrix.

Feature Extraction and Feature Matching

Since SIFT and SURF algorithms are patented in the current version of OpenCV library, I started with ORB algorithm for feature extraction. The function in the OpenCV library for this purpose returns keypoint and descriptor arrays.

```
def findMatches(img1, img2):
    # finding matches with orb detector
    surf_detector = cv2.ORB_create()
    kp1f, des1 = surf_detector.detectAndCompute(img1, None)
    kp2f, des2 = surf_detector.detectAndCompute(img2, None)
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)
    img3 = cv2.drawMatches(img1, kp1f, img2, kp2f, matches, None, flags=2)
    cv2.imshow("img3", img3)
    cv2.waitKey(0)
    return matches, kp1f, kp2f
```

Then I used Brute Force Matching in the OpenCV library for Feature Matching.

Finding Homography

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

I found out that, this matrix equation to find homography can be solved as an Eigenproblem. In NumPy library, I found two different functions for this problem. Numpy.linalg.svd() function gave me the homography matrices that were giving correct solutions on images so I used it.

```python
def findHomography(samples, keypoints_img1, keypoints_img2):
    # finding homography matrix for given sample of matches
    a_matrix = []
    # put every match in 'A' matrix in correct format
    for match in samples:
        img1_idx_h = match.queryIdx
        img2_idx_h = match.trainIdx

        source_point = keypoints_img1[img1_idx_h].pt
        destination_point = keypoints_img2[img2_idx_h].pt
        row1 = [source_point[0], source_point[1], 1.0, 0.0, 0.0, 0.0,
                -(destination_point[0] * source_point[0]),
                -(destination_point[0] * source_point[1]),
                -(destination_point[0])]
        row2 = [0.0, 0.0, 0.0, source_point[0], source_point[1], 1.0,
                -(destination_point[1] * source_point[0]),
                -(destination_point[1] * source_point[1]),
                -(destination_point[1])]
        a_matrix.append(row1)
        a_matrix.append(row2)
    a_matrix = np.matrix(a_matrix)
    u, s, vh = np.linalg.svd(a_matrix)  # numpy function for solving eigen pr
    homography = np.reshape(vh[8], (3, 3))  # rearranging the smallest eigenv
    homography = homography/homography[2, 2]  # dividing the matrix by its ri
    return homography
```

I created the “A” matrix on the equation and put inputs in correct format. Then I used the function to find homography matrix. In order to make the right bottom value equal to “1”, I divided the matrix by that value.

Using RANSAC

Feature extraction algorithms we use does not give proper results every time. A lot of matches they find are not correct. We need to use an algorithm like RANSAC for this problem. Its purpose is to calculate an homography with at least four random match samples and try that homography value on every match the feature extraction functions had found. The homography matrix which gives the highest number of correct results is the true homography matrix. I would say it is a complicated trial and error algorithm, but it works.

```
def RANSACforOptimalHomography(matches, kp1, kp2):
    # ransac algorithm for finding opimal homography matrix
    inlier_error = 1
    max_inliers = 0
    optimal_homography = None
    for i in range(1000):
        print("Max inliers = ", max_inliers)
        four_random_matches = random.sample(matches, 4)  # take 4 random samples in A matrix and calculate homography
        homography = findHomography(four_random_matches, kp1, kp2)
        inlier_count = 0
        for match in matches:  # try this homography for every match in the matches list
            img1_index = match.queryIdx
            img2_index = match.trainIdx

            source_point_ransac = kp1[img1_index].pt
            destination_point_ransac = kp2[img2_index].pt

            source_point_matrix = np.asarray([source_point_ransac[0], source_point_ransac[1], 1.0]).T
            estimated_point = np.dot(homography, source_point_matrix)
            estimated_point_x = estimated_point[0, 0]
            estimated_point_y = estimated_point[0, 1]
            estimated_point_array = np.array((estimated_point_x, estimated_point_y))
            destination_point_array = np.array(destination_point_ransac)
            # estimated point is the one calculated with the homography
            # calculate euclidian distance between estimated point and destination point of the match
            geometric_distance = np.linalg.norm(destination_point_array - estimated_point_array)
            if geometric_distance < inlier_error:
                # if the geometric distance between two points is neglectable, increase inlier count
                inlier_count = inlier_count + 1
        # if inlier count for current homography matrix is the greatest that homography should be the optimal one
        if inlier_count > max_inliers:
            max_inliers = inlier_count
            optimal_homography = homography
    return optimal_homography
```

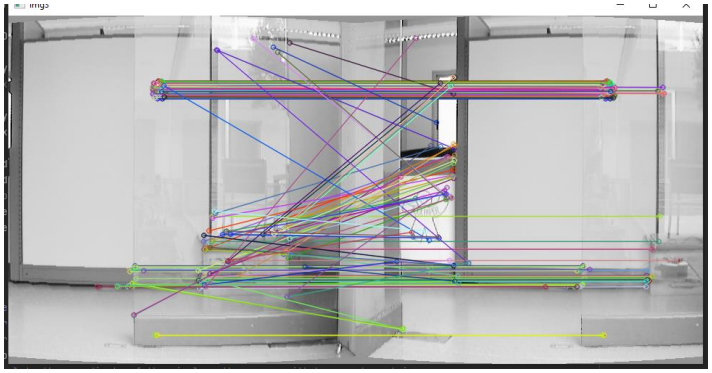
Merging by Transformation

I started with creating an empty canvas for that matter. I calculated its size by applying homography matrix on the corner pixels of the to be stitched image. I copied the foundation image on this canvas then applied the transformation on every pixel on the to be stitched image.

```
def stitchImages(first_image, second_image):
    m, kp1, kp2 = findMatches(second_image, first_image)
    h = RANSACforOptimalHomography(m, kp1, kp2)

    image1_shape_y = first_image.shape[0]
    image1_shape_x = first_image.shape[1]

    image2_shape_y = second_image.shape[0]
    image2_shape_x = second_image.shape[1]
    # calculating the sufficient canvas size for stitching
    dr_x, dr_y = dotHomoghraphy(h, image2_shape_x, image2_shape_y)
    ur_x, ur_y = dotHomoghraphy(h, image2_shape_x, 0)
    # creating a blank canvas and pasting the foundation image on it
    shape = (image1_shape_y, int(max(dr_x, ur_x)))
    canvas = np.zeros(shape, dtype=int)
    paste(canvas, first_image, (0, 0))
    # now we have a canvas with foundation image on left

    # to warp the second image and stitch it on the canvas
    for y in range(image2_shape_y):
        for x in range(image2_shape_x):
            # iterate through each pixel on the image that will be stitched
            new_coord_x, new_coord_y = dotHomoghraphy(h, x, y)
            # calculate the coordinate of the pixel on the canvas with homography matrix
            if inShape(int(new_coord_x), int(new_coord_y), shape):
                # if found pixel is in the dimensions of the canvas print it
                canvas[int(new_coord_y), int(new_coord_x)] = second_image[y, x]

    return canvas.astype(np.uint8)
```

Results

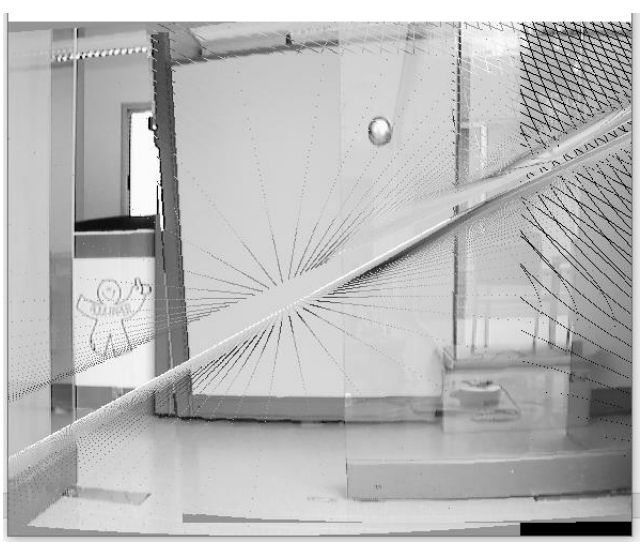
I could not complete this assignment, so my results are not completely correct.

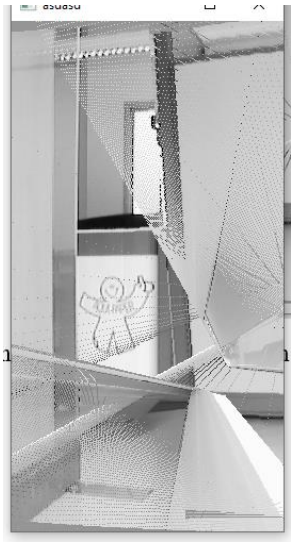


This is the feature point matching of the two images on the first folder of the dataset. Their file names are “cyl_image05.png” and “cyl_image06.png” to be exact.

This is the result of them merging. It seems correct and promising. But on next iterations, it gets messy. Sometimes it gives partially correct results like below.

But in general, when there are a lot of mismatches on the two images or when there are not enough matches, RANSAC algorithm can not find a proper homography matrix so outputs get real messy like below.





On each iteration it gets worse and on the iteration it could not find a single homography, the program crashes.

I really could not understand what the problem is and I ran out of time. I thought the problem was merging the two images together so I tried different algorithms for that but they also did not work.