Lab 3 : computer vision

Image Mosaics

April 27, 2019

Team

Dahlia Chehata 27

Fares Mehanna 52

TABLE OF CONTENTS

Overview

In this exercise, you will implement an image stitcher that uses image warping and homographies to automatically create an image mosaic. We will focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps. For simplicity, we will specify corresponding pairs of points manually using mouse clicks.

Getting correspondence

<table>
<tr><th>Automatic</th><th>Manual</th></tr>
<tr><td>

```
def auto_correspondence(image1,image2):

    ' get coresspondance points between two given
images using (oriented BRIEF) keypoint detector and
descriptor extractor'
    ' better than SIFT descriptors'

    orb = cv.ORB_create()
    keypoint1, descriptor1 =
orb.detectAndCompute(image1, None)
    keypoint2, descriptor2 =
orb.detectAndCompute(image2, None)

    bf = cv.BFMatcher(cv.NORM_HAMMING,
crossCheck=True)  # creates a matcher

    # Match descriptors.
    matches = bf.match(descriptor1, descriptor2)  #
matches the two descriptors
    matches = sorted(matches, key=lambda x:
x.distance)  # sort matches where best matches come
first

    points1 = []  # list of correspondence points in
first image
    points2 = []  # list of correspondence points in
second image

    # loop on matches and fills  points1  and points2
```

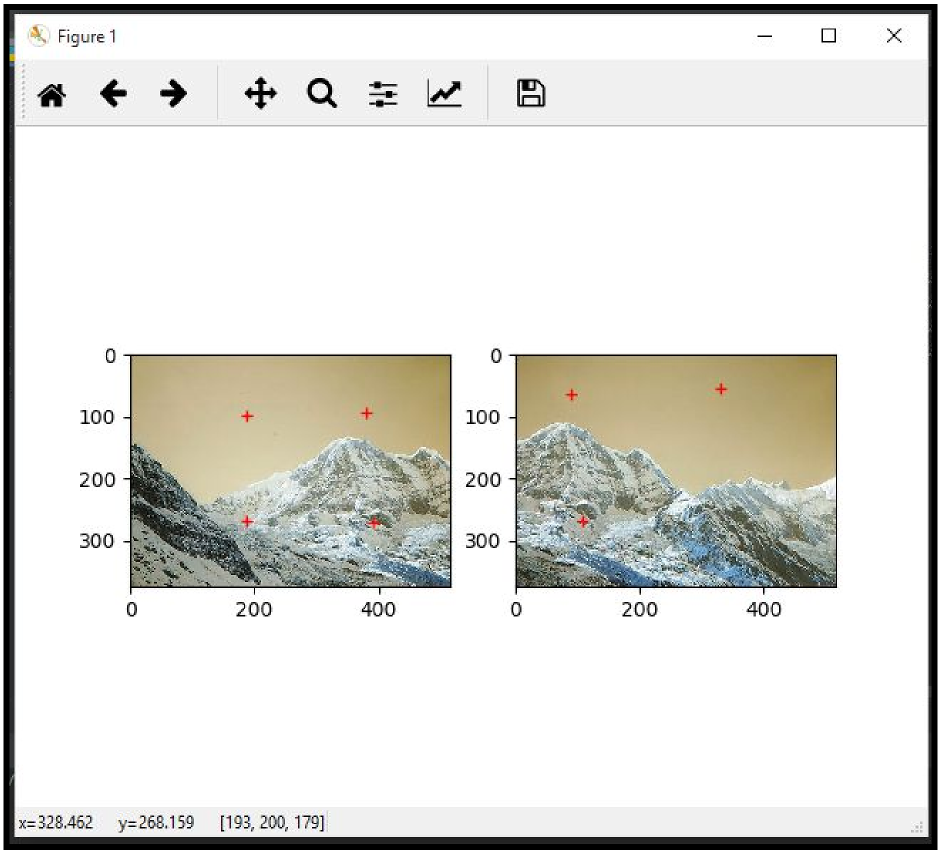
</td><td>

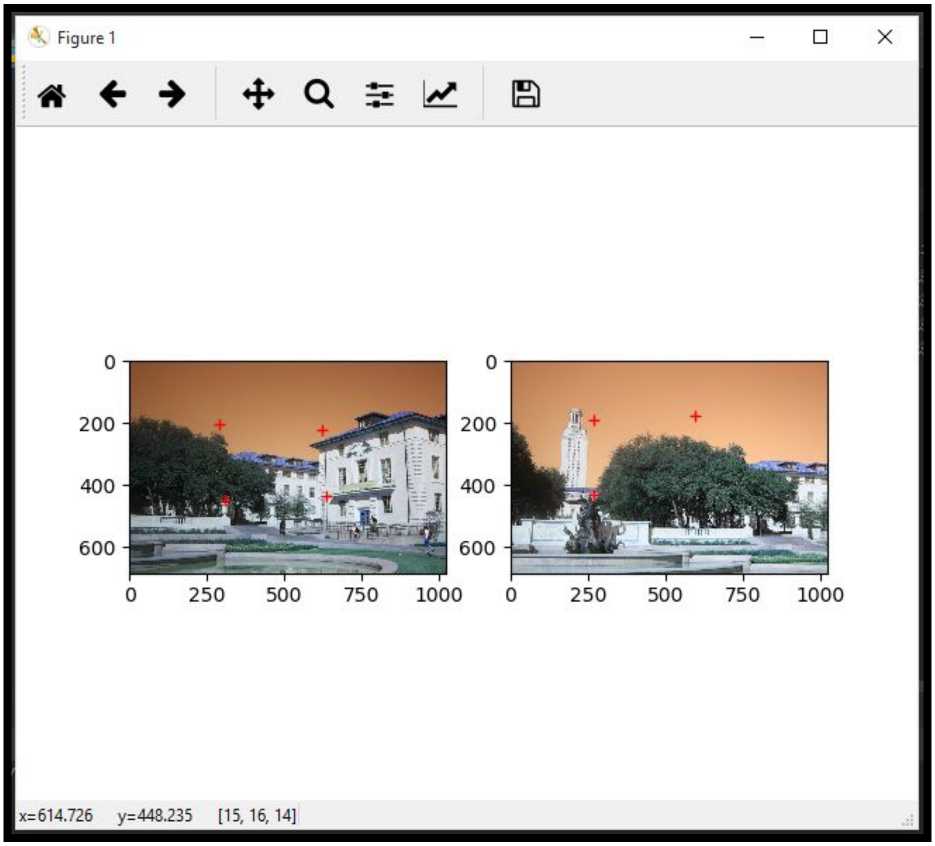
```
def manual_correspondence(image1, image2,
number_of_points):
    " Display images and select matching points "
    fig = plt.figure()
    fig1 = fig.add_subplot(1, 2, 1)
    fig2 = fig.add_subplot(1, 2, 2)

    # Display the image
    # to flip the image : #, origin='lower'
    fig1.imshow(image1)
    fig2.imshow(image2)
    plt.axis('image')

    p1 = np.zeros([(number_of_points // 2), 2])
    p2 = np.zeros([number_of_points // 2, 2])
    pts = plt.ginput(n=number_of_points,
timeout=0)
    p1_itr = 0
    p2_itr = 0
    for i in range(0, number_of_points):
        if i % 2 == 0:
            p1[p1_itr] = pts[i]
            print("p1 of index ", p1_itr, " is ",
pts[i])
            p1_itr += 1

        else:
            p2[p2_itr] = pts[i]
            print("p2 of index ", p2_itr, " is ",
```

</td></tr>
</table>

```
    for match in matches:
        index1 = match.queryIdx
        points1
.append((int(keypoint1[index1].pt[0]),
int(keypoint1[index1].pt[1])))
        index2 = match.trainIdx
        points2.append((int(keypoint2[index2].pt[0]),
int(keypoint2[index2].pt[1])))

    points1 = np.array(points1)
    points2 = np.array(points2)
    return points1, points2
```

```
pts[i])
            p2_itr += 1
    return p1, p2
```





- Computing the homography parameters
    - 8 unknown variables
    - we need at least 4 pair of points
    - homography format is that the last element in the matrix is 1
    - the resultant projective transformation is 3 X 3

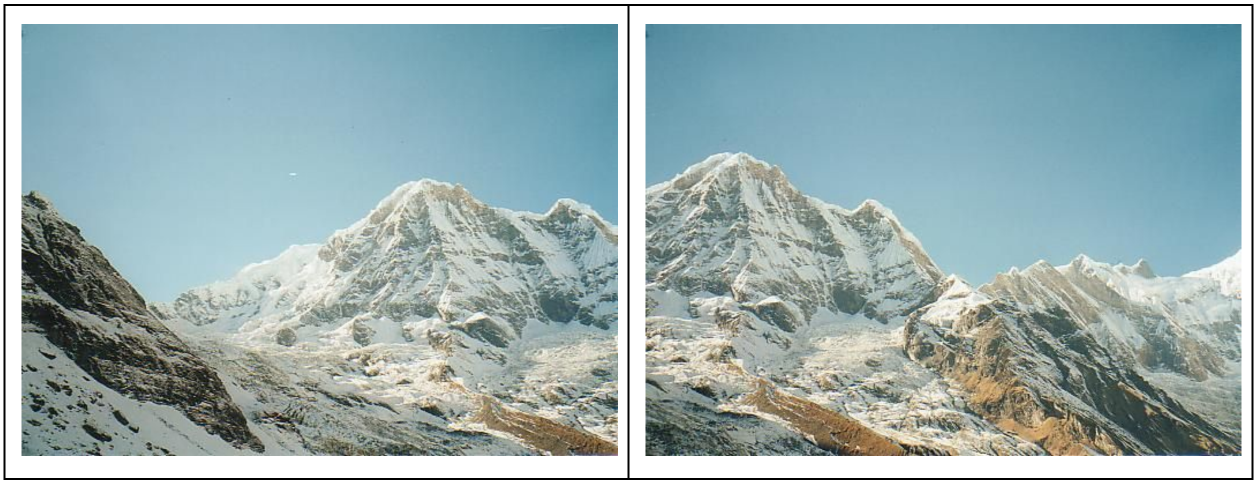
```
def calculate_homography(points1, points2):
    """
     takes a pair of points list , calculates the Homography matrix, returns
3x3 matrix"
     points1 is array of correspondence points  ≥ 4 of image 1
     points2 is array of correspondence points  ≥ 4 of image 2
    """

    B = points2
```

```
    A = np.zeros([2 * points1.shape[0], 8])  # construct A from input
points
    for i in range(points1.shape[0]):  # constructs A from given points of
image 1
        A[i * 2] = points1[i, 0], points1[i, 1], 1, 0, 0, 0, -points1[i, 0]
* points2[i, 0],  -points1[i, 1] * points2[i, 0]

        A[i * 2 + 1] = 0, 0, 0, points1[i, 0], points1[i, 1], 1, -points1[i,
0] * points2[i, 1], -points1[i, 1] * points2[i, 1]

    B = B.flatten().reshape(-1, 1)  # flatten and reshape to be one column
for dimension suitability

    H = np.linalg.lstsq(A, B, rcond=None)[0]  # returns H

    H = np.append(H, 1)  # puts 1 at the end of H

    H = np.reshape(H, [3, 3])  # return H as matrix of shape 3x3
    return H
```

- Warping between image planes
    - forward warping
    - The transformed coordinates are sub-pixel values, sample the pixel values from nearby pixels.
    - For color images, warp each RGB channel separately and then stack together to form the output
    - inverse warping to remove holes

Screenshots of the output mosaic

Manual : result__manual with ransac_ransac_loops=10



- Warp one image into a frame region in the second image. To do this, let the points from the one view be the corners of the image you want to insert in the frame, and let the corresponding points in the second view be the clicked points of the frame (quadrilateral) into which the first image should be warped

- Extra credits section
  - RANSAC

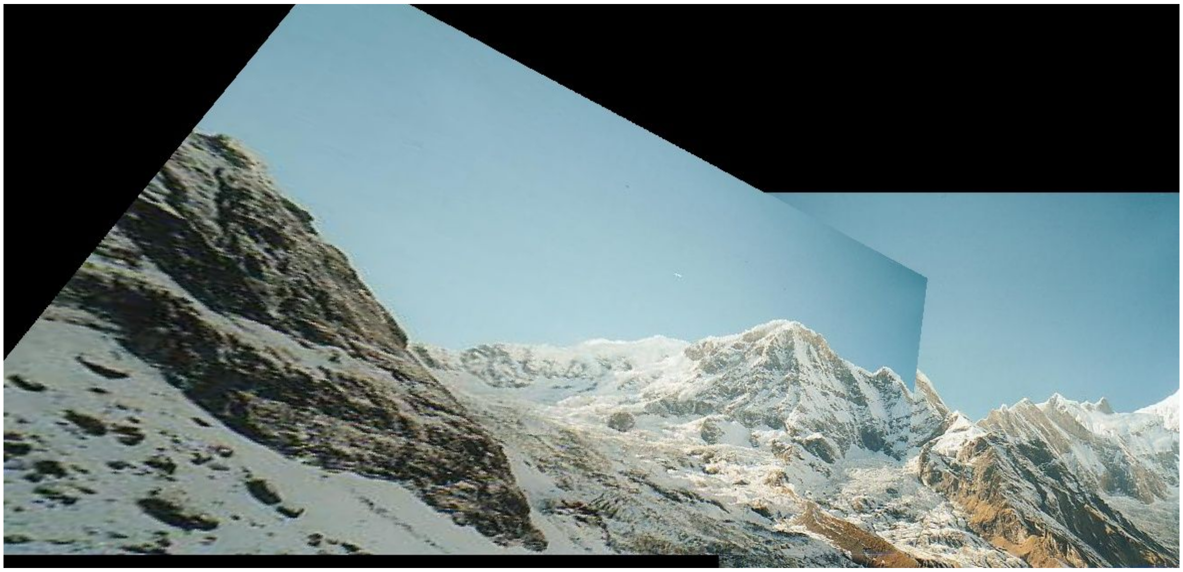
```
def ransac(points1, points2, threshold, iterations):
    "takes correspondence points and for each 4 random pairs it
calculates h "
    " and counts inliners from a given threshold and keeps the
best h"
    max_inliners = 0
    best_h = None

    for i in range(iterations):
        inliners = 0
        randp = np.zeros([4, 2])
        randp_ = np.zeros([4, 2])
        H = None
        for j in range(4):
            random_index = random.randrange(0,
points1.shape[0], 1)  # picks random points from the given set

            randp[j] = points1[random_index]
            randp_[j] = points2[random_index]

        H = Homography.calculate_homography(randp, randp_)  #
calculates h from the 4 random points

        for j in range(points1.shape[0]):
            error = ransac_error(points1[j], points2[j], H)

            if (error < threshold):
                inliners += 1

        if (inliners > max_inliners):
            max_inliners = inliners
            best_h = H
            # return the H and the number of inliners for H
    return best_h, max_inliners
```

- **MANUAL_CORRESPONDENCE, mosaic_builder.RANSAC_H,10**

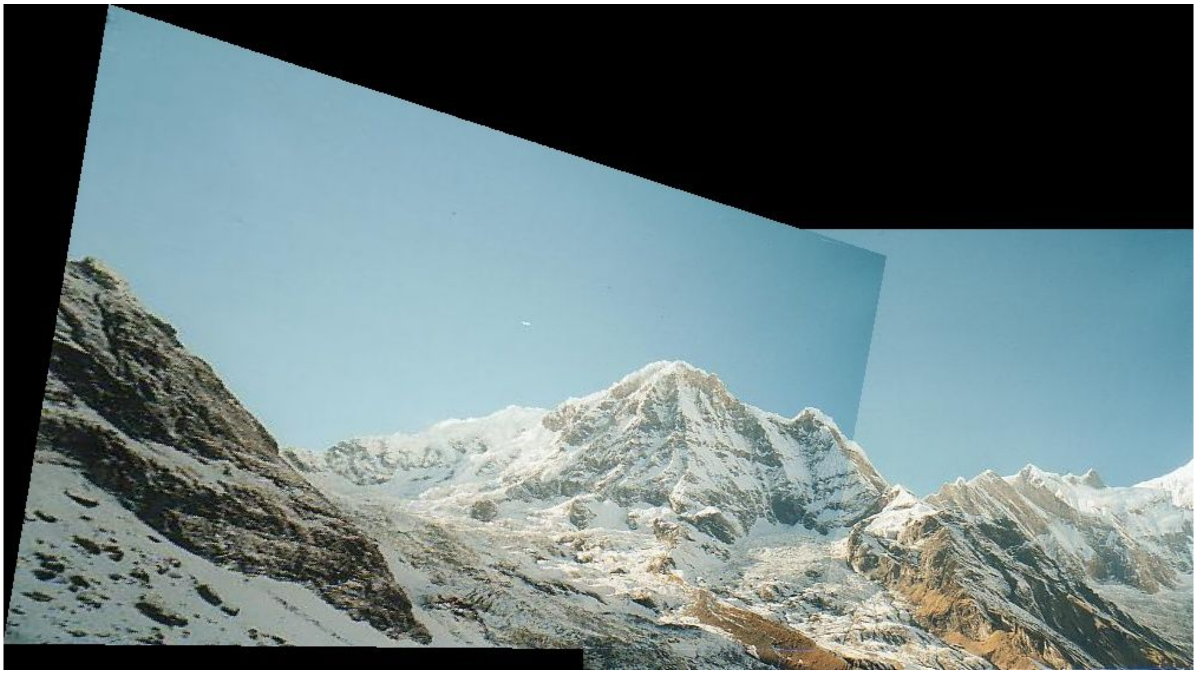
  shown before

- **AUTO_CORRESPONDENCE, mosaic_builder.RANSAC_H, 100**

AUTO_CORRESPONDENCE, mosaic_builder.RANSAC_H, 1000



AUTO_CORRESPONDENCE, mosaic_builder.RANSAC_H, 5000



AUTO_CORRESPONDENCE, mosaic_builder.NORMAL_H (without RANSAC)

- Automatic correspondence

  shown before

- Conclusion :

the first pair of images couldn't be formed under automatic correspondence without RANSAC

it robustly estimates the homography matrix from noisy correspondences and it successfully gives good results even when there are outlier (bad) correspondences given as input