CS543: Computer Vision **Spring 2017**

Problem Set 2

Chongxin Luo (cluo5) *Handed In: March 1, 2017*

1. **A feature tracker**

   (a) **Keypoint Selection**

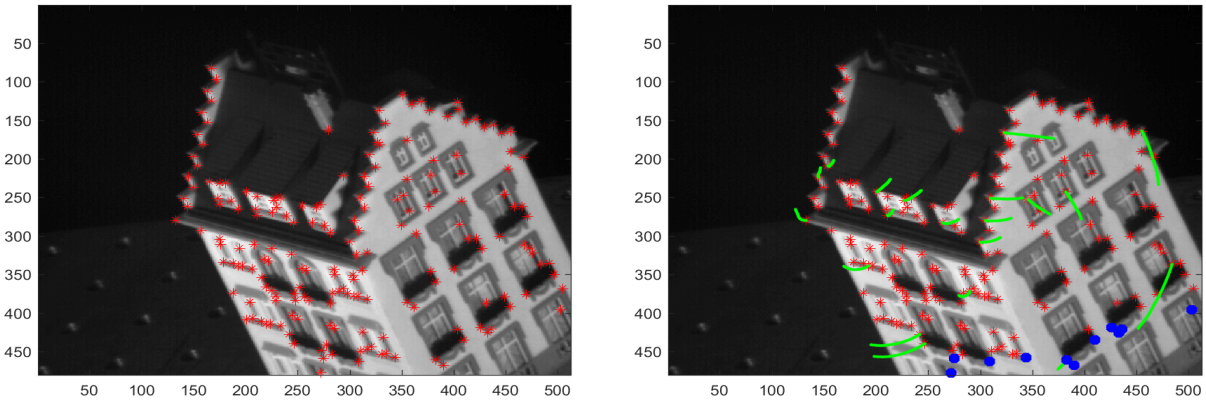
   In this section, we implemented a Harris Detector in order to perform keypint selection on input image. The Harris Detector was implemented according to the following steps:

   i. Smooth input image using Gaussian filter, find x and y directional gradient of the image I_x, and I_y.
   ii. Square the directional gradient to obtain I_x^2, I_y^2, and I_xI_y.
   iii. Apply Gaussian filter on three square graident output to obtain $g(I_x^2)$, $g(I_y^2)$, and $g(I_xI_y)$.
   iv. compute Harris score for each pixel with following formula: $g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$.
   v. non-maxima supression over image by finding the local maximum over $5x5$ blocks with threshold value $\tau = 0.002$.

   A total of 187 points were found using the method described above, and the result image with keypoints is shown below in figure 1 part (a). The red $*$ in the image is the key points that being found using this method.

   (b) **Tracking**

   In this section, the KLT tracking algorithm was implemented in order to track the key points over 50 frames of input images. For each critical points and each two image frame pairs, we started by computing the directional gradient I_x and I_y around the critical point in a 15x15 grid, we computed the initial time gradient by $I_t = I(x_i', y_i', t+1) - I(x, y, t)$. Then, we performed the point transformation prediction and solved the u, v vector by using the equation (4) from homework documentation, and finally we made the transformation of the point according to value u, v to obtain a new set of prediction points (x_{i+1}', y_{i+1}'). This procedure was repeated 25 times to obtain the final prediction of a single critical point between two frames. The final result of the key point tracking is shown in the figure 1 part (b) below. All 187 points are being tracked, and 20 points are being randomly selected and displayed on the image. The green line indicate the point's movement over 50 frames, the red $*$ indicate the point's initial location, and blue $*$ indicate the point moved out of image during the transformation.



(a) Key points generated with Harris Method

(b) Key points tracking using TLK

Figure 1: Key points generation and tracking

From the result image above, we can see that in image (a), the algorithm successfully generated a set of key points on the input image. The key points are mainly on the corner of the image texture. in the image (b), it shows 20 random key points tracking path and the set of points that moved out of the image during transformation. The algorithm successfully tracked all key points, which can be shown as the key points path as a clockwise rotation of the image, and the key points on the lower side of the image moves out of the image during transformation and being marked out as blue.

2. **Shape alignment**

In this section, we implemented a shape alignment algorithm, which given two input images and the algorithm aligns the shape on the first image onto the shape on the second image. The algorithm uses a combination of ICP algorithm and affine transformations to find the alignment. It performs the following steps:

(a) using matlab $find()$ function to extract the pixel points of shape on both images.

(b) find the mean and standard deviation for both point sets, and transform the point set from the first image to have the same mean and standard deviation with the point set from the second image.

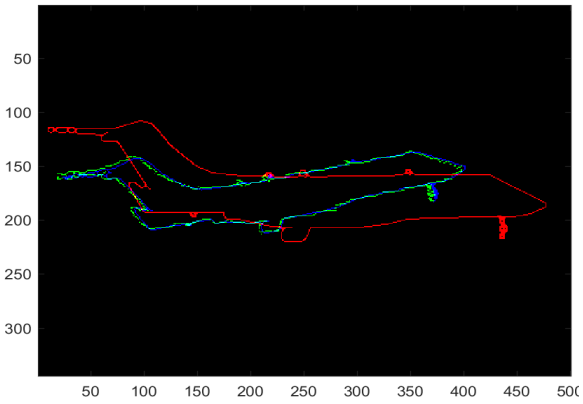
(c) using matlab $bwdist()$ function to find the closest points for each point from the image one set, they considered as a paired matching point

(d) Stack all points to one matrix and solve the affine transformation, which returns a single affine transformation matrix (with least square method)

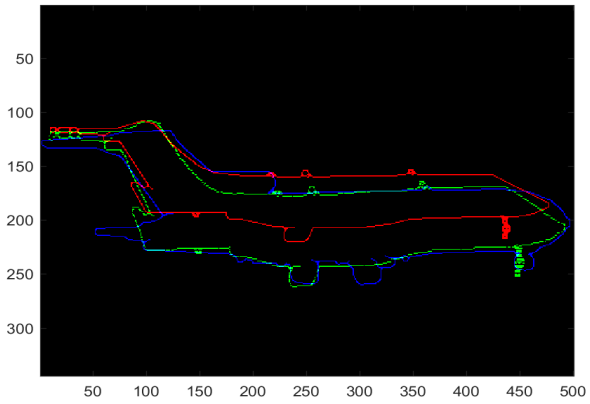
(e) Transform image point set 1, and repeat step (c)-(e) until reach a threshold.

The threshold that we used in this implementation is changing of average distance between paired point from two iterations, if the change is less than 0.01, we consider the algorithm converged to a good result.

The tree images below are the aliened output from the given matlab function *align_shape*. The red figure is original image, blue figure is the target image, and the green figure is the output image after alignment. The alignment error result and running time are shown in the table below.

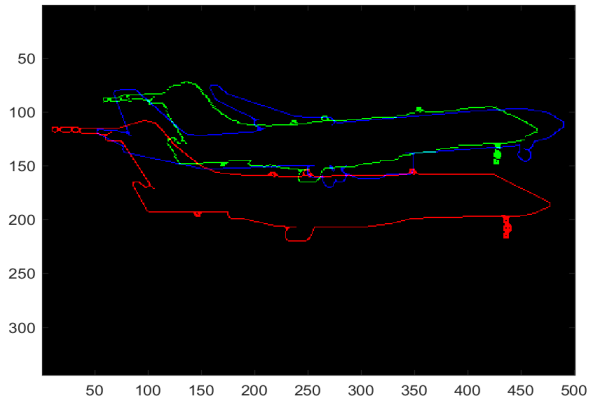


(a) Alignment of object 2 to object 2t



(b) Alignment of object 2 to object 3

Figure 2: Image alignment output



(a) Alignment of object 2 to object 1

Figure 3: Image alignment output Conti.

Alignment images	error	run time(s)
obj2 to obj2t	1.144	0.255
obj2 to obj3	4.922	0.299
obj2 to obj1	7.252	0.146

Table 1: Alignment error and run time

From the results shown above, we can see that the algorithm performs very well with image only contains translation displacement. When the target images have more than just translation displacement, the algorithm can still align the shape to a relatively close position, however, it cannot achieve a near 100% alignment for images has more than translation displacement. This might have to do with the affine transformation method that we used in the implementation, which the affine transformation only covers 6 degrees of freedoms, which cannot fully cover all types of transformations.

3. **Object instance recognition**

(a) **Keypoint matching**
We simply checking all keypoint descriptor from the given set, and comparing them with g, if they match, we return the keypoint descriptor. The pseudo-code is shown below:

```
KeypointMatch():
  for i ← 1 to n:
      if g == f_i:
          return f_i
  return "no match"
```

(b) **Object alignment**
The object alignment can be completed by performing the following steps :

i. Transform image one keypoint to origin with scale of 1, and orientation of 0 degrees, which is transform the keypoint of the image one such that $u_1 = 0, v_1 = 0, s_1 = 1, \theta_1 = 0$.
ii. Perform the same transformation for the center and four corner points of the box container.
iii. Transform the keypoint to match the orientation of the keypoint in the image 2.
iv. Perform the same transformation for the center and four corner points of the box container.

After this procedure, the layout of the box container in the first image should match with the box container in the second image.