Indiana University Bloomington

CSCI-B657 Computer Vision

Spring, 2016

Assignment 2

Author:
Jayalakshmi Sureshkumar
jsureshk@indiana.edu
Hao Peng
penghao@iu.edu
Raghuveer Krishnamurthy Kanchibail
rkanchib@indiana.edu
February 29, 2016

1 Part 1

To run our program for this part, you should first go into `a2-images/part1_images` as follows:

```
$ cd a2-images/part1_images
```

1.1 Question 1

Our program accepts 5 parameters as show in the following example, we set the ratio between the closest and second closest distance as 0.75:

```
$ ../../a2 part1 0.65 bigben_2.jpg bigben_3.jpg
```

The output is as shown in Figure 1.

Figure 1: Sample output of question 1

The match result is not perfect because the two images have very different illumination and background, below is a test on our own images with similar background, as shown in Figure 2. The result is much better.

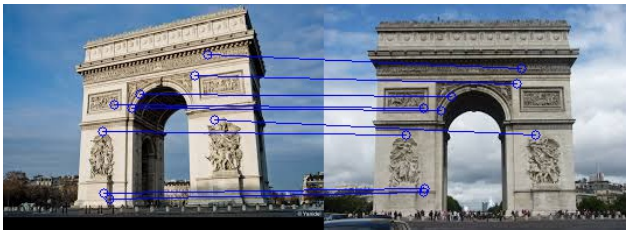


Figure 2: Output of our own images for question 1

1.2 Question 2

Our program for this question runs in the following way(replace `*.jpg` with a sequence of the names of images you want to retrieve from):

```
$ ../../a2 part1 0.65 bigben_2.jpg bigben_3.jpg *.jpg
```

And the output is as shown in Figure 3:

```
[jsureshk@silo part1_images]$ ../../a2 part1 0.65 bigben_2.jpg bigben_3.jpg bigb
en_6.jpg bigben_7.jpg bigben_8.jpg bigben_10.jpg bigben_12.jpg bigben_13.jpg big
ben_14.jpg bigben_16.jpg
bigben_3.jpg : 26
bigben_14.jpg : 9
bigben_6.jpg : 6
bigben_8.jpg : 4
bigben_12.jpg : 2
bigben_16.jpg : 1
bigben_7.jpg : 1
bigben_10.jpg : 0
bigben_13.jpg : 0
```

Figure 3: result of image retrieve

1.3 Question 3

For this question, we have to run the program for 10 times. Each time it will output the top 10 out of 100 matching images with the number of corresponding matches and the precision. We then calculate the accuracy of our program by averaging those 10 accuracies.

One example for our program to retrieve images similar to `bigben_2.jpg` is as follows:

```
$ ../../a2 part1 0.65 bigben_2.jpg *.jpg
```

The output is as shown in Figure 4. We can see that the accuracy for retrieving `bigben_2.jpg` is 50%.

The accuracy for 10 attractions is shown in Table 1. The average accuracy of our image retrieve system is 24%.

```
[jsureshk@silo part1_images]$ ../../a2 part1 0.65 bigben_2.jpg *.jpg
bigben_2.jpg : 2843
bigben_3.jpg : 26
bigben_14.jpg : 9
bigben_6.jpg : 6
empirestate_27.jpg : 4
tatemodern_14.jpg : 4
bigben_8.jpg : 4
londoneye_13.jpg : 3
trafalgarsquare_15.jpg : 3
sanmarco_1.jpg : 3
Precision :50
```

Figure 4: precision of image retrieve for bigben

Accuracy for 10 attractions for Q3	
query image	accuracy(%)
bigben_2.jpg	50
colosseum_15.jpg	20
eiffel_7.jpg	10
empirestate_25.jpg	10
londoneye_16.jpg	20
louvre_14.jpg	20
notredame_19.jpg	30
sanmarco_1.jpg	10
tatemodern_2.jpg	30
trafalgarsquare_22.jpg	30

Table 1: accuracy of image retrieve for 10 attractions

1.4 Question 4

In this question, we need to specify parameter k and w in our program, which means we reduce the dimension of the SIFT descriptor vectors from 128 to just k. We call the reduced vector as summary vector.

After that for each SIFT descriptor i in the query image, we find the candidate vectors whose summary vectors are identical to i's summary vector and then we calculate the distances between vector i and all the candidate vectors in 128 dimensional space.

One example to test our implementation for image matching is as follows(0.65 is the threshold used in `findMatch` function, 3 and 100 is used to set parameters k and w):

```
$ ../../a2 part1 0.65 sanmarco_3.jpg sanmarco_5.jpg 3 100
```

The output is in Figure 5.

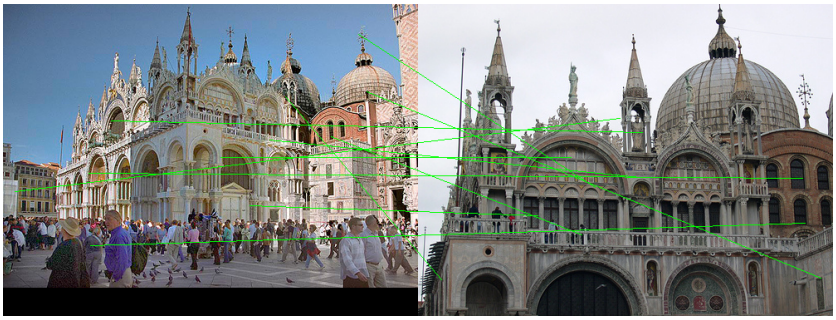


Figure 5: image matching using summary vector

Another example to test our implementation for image retrieve on all 100 images is shown in Figure 6(0.65 is the threshold, 3 and 100 is used to set parameters k and w). The running efficiency is slightly better than what we did in question 3.

```
[jsureshk@silo part1_images]$ ../../a2 part1 0.65 sanmarco_1.jpg 3 100 *.jpg
sanmarco_1.jpg : 1799
louvre_13.jpg : 3
tatemodern_14.jpg : 2
bigben_16.jpg : 2
bigben_7.jpg : 2
tatemodern_9.jpg : 2
notredame_3.jpg : 1
sanmarco_20.jpg : 1
sanmarco_19.jpg : 1
empirestate_15.jpg : 1
Precision :30
```

Figure 6: precision of image retrieve for bigben

The accuracy for 10 attractions is shown in Table 2. The average accuracy of our image retrieve system is 22%, which is similar to the one in question 3.

Accuracy for 10 attractions in Q4	
query image	accuracy(%)
bigben_2.jpg	10
colosseum_15.jpg	40
eiffel_7.jpg	10
empirestate_25.jpg	30
londoneye_16.jpg	10
louvre_14.jpg	10
notredame_19.jpg	20
sanmarco_3.jpg	40
tatemodern_2.jpg	30
trafalgarsquare_22.jpg	30

Table 2: accuracy of image retrieve for 10 attractions

2 Part 2

To run our program for this part, you should first go into `a2-images/part2_images` as follows:

2.1 Question 1

We implemented this problem using inverse warping, which means for each coordinate in the output image, we use the inverse transformation to calculate its corresponding coordinate in the original image and copy its value to the output image.

We use the given matrix to transform the `lincoln.jpg` to Figure 7 using the following command:

```
$ cd a2-images
$ ../../a2 part2 lincoln.jpg
```

2.2 Question 2

First, we get all possible matches between two images using `findMatches()` in part1. We sampled 100 times in our **RANSAC** algorithm, each time we

Figure 7: Warped image of lincoln.jpg

randomly chose four pairs from the match points and calculate the transformation matrix using the linear system solver, we also counted the number of points that agree with this matrix.

Then we use the matrix which has the maximum number of inliers to transform the test image into a warped image that appears to have been taken in the first camera's coordinate system. Our implementation was tested in question 3.

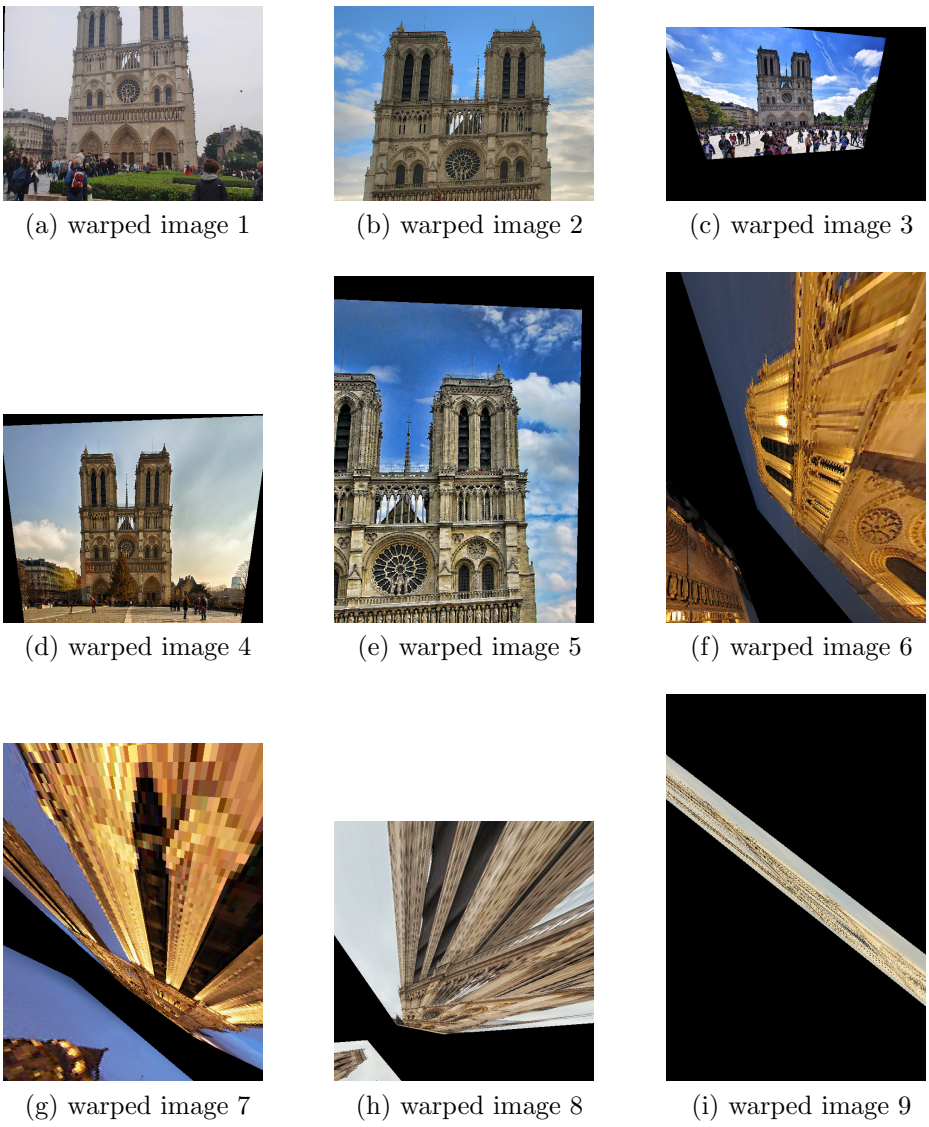
2.3 Question 3

We run the program with `3268706748_0d2c67f3c3_z_d.jpg` as input image as follows(please replace the `*.jpg` as a sequence of the remaining image in folder `seq1`) :

```
$ cd a2-images/part2_images
$ cd seq1
$ ../../../a2 part2 0.65 3268706748_0d2c67f3c3_z_d.jpg
*.jpg
```

And the output is as shown in Figure 8:

We can see that when random sampling 100 times with the threshold of `findMatches()` as 0.65, 5 out of 9 warped image are pretty good, while the other 4 makes little sense. This problem is due to the performance of our



(a) warped image 1 (b) warped image 2 (c) warped image 3

(d) warped image 4 (e) warped image 5 (f) warped image 6

(g) warped image 7 (h) warped image 8 (i) warped image 9

Figure 8: Image matching using summary vector

`findMatches()` function and the random sampling.

The warped picture with little sense means either that we sampled four wrong pairs or that we sampled four correct matches, but these matches are not perfectly close to each other, thus those output images make little sense.