Problem 1:

Harris Corner Detector

1. The majority of implementation is the formula(I use K = 0.05 and the paper recomend us to use k between 0.04 to 0.06):

R = detM - $k(traceM)^2$ = $\lambda_1\lambda_2 - 0.05(\lambda_1 + \lambda_2)^2$

Where M = $\sum w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$, R is the score to return for the function.

2. The W(x,y) is the window function and I used gaussian here. Ix and Iy is the gradient along x and y direction that can be derived with sobel filter.

3. At last, I set a bar for the score such that only 200 points at most, for example, can be returned for my output.

Problem 2:

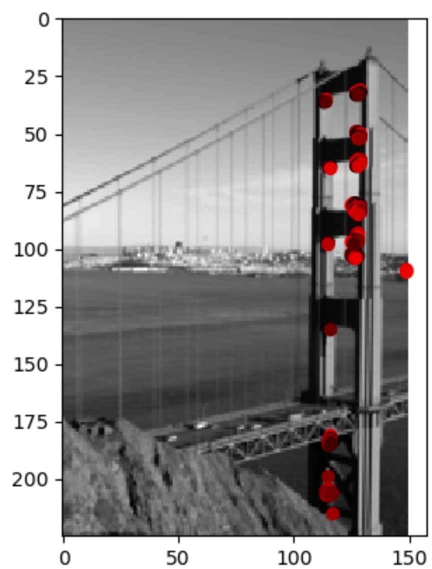
Feature descriptor:

set width: 2*scale + 1 spacial width: 3* set width

1. For each set_width * set_width(here 3*3) grid around the interest point, I created a histogram of 8 bins that spread evenly from -180 to 180 degree and stored the orientation energy(here I just use the magnitude of gradient from canny_max function). Therefore, a vector with length 8 is created for each 3*3 grid.

2. For each 9*9 grid, we will have 9 numbers of 3*3 grid. Therefore, each 9*9 grid around a interested point, we have a vector of length 9*8 = 72.

A sample output with Golden Gate with 100 Interest points:



Problem 3**:**

Feature Matching

Naive Approach:

1. Given two extracted features, I traversed each feature for image 0 and find its nearest neighbour in image1. Return their ratio as a metric for the matching.

2. This naive approach takes at least O(n^2) time complexity to traverse each point.

KD tree:

this code use this wiki page as reference : https://en.wikipedia.org/wiki/K-d_tree

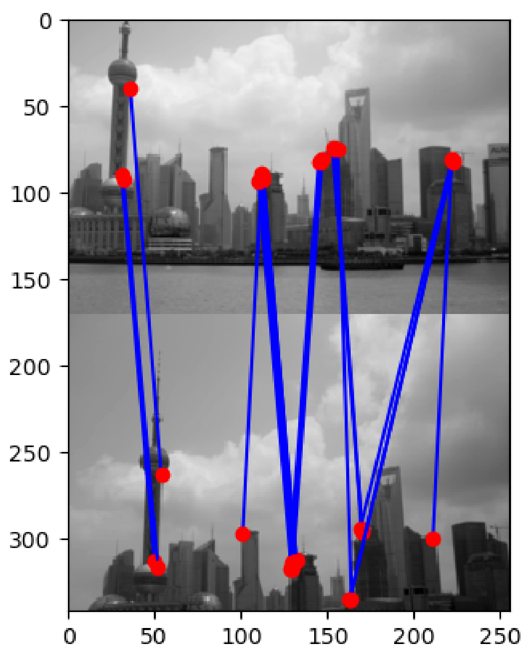
1. Build a tree such that each split I changed the axis to next dimension. In this way, the tree split the whole space.

2. When we search the nearest neighbour, simply go to the left subtree if our input vector's value on k dimension is less than the value in the node, right otherwise. I use the same method that record the nearest and second nearest distance and use these ratio for the score.

3. This KD tree first takes O(nlgn) to build the tree and when finding nearest neighbour, it takes O(lgn) for each feature point.

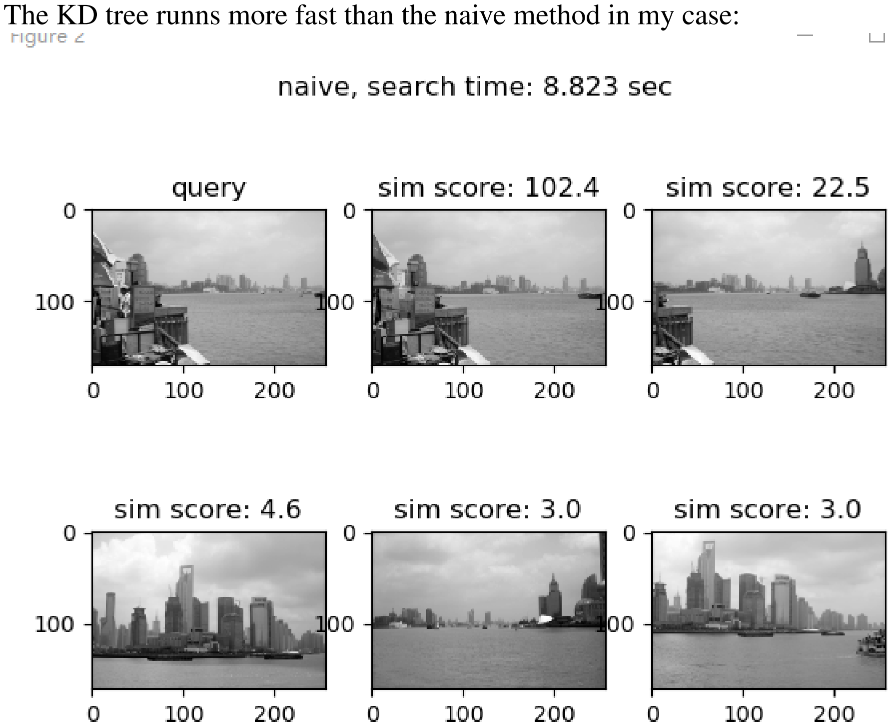
Sample Output:

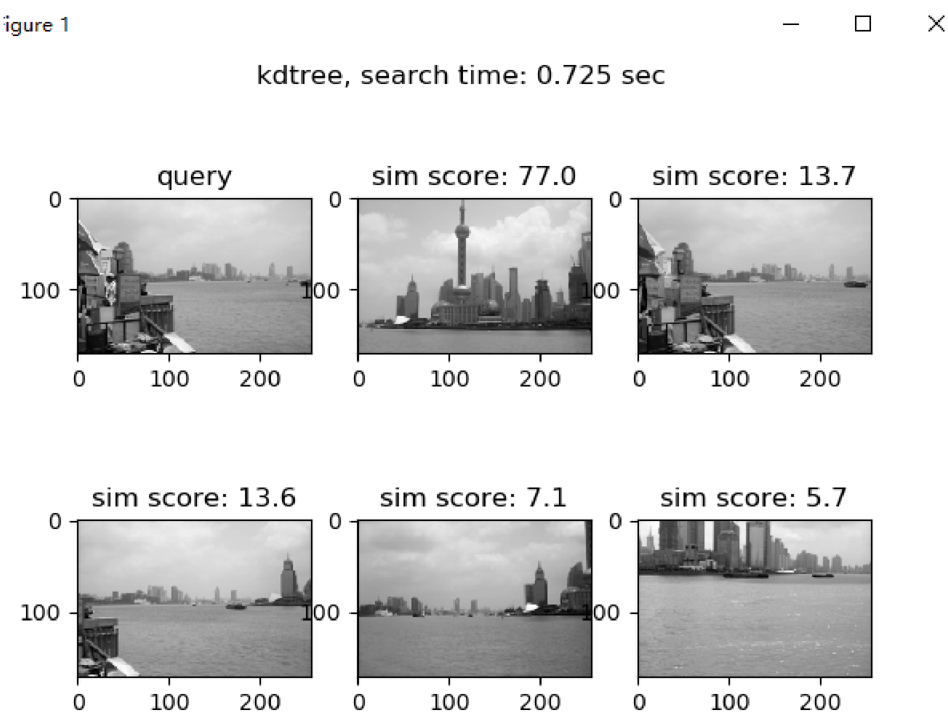
Feature matching with 100 features and threshould value = 0.95



Run time comparison:

The KD tree runns more fast than the naive method in my case:





Problem 4**:**

Hough Transform

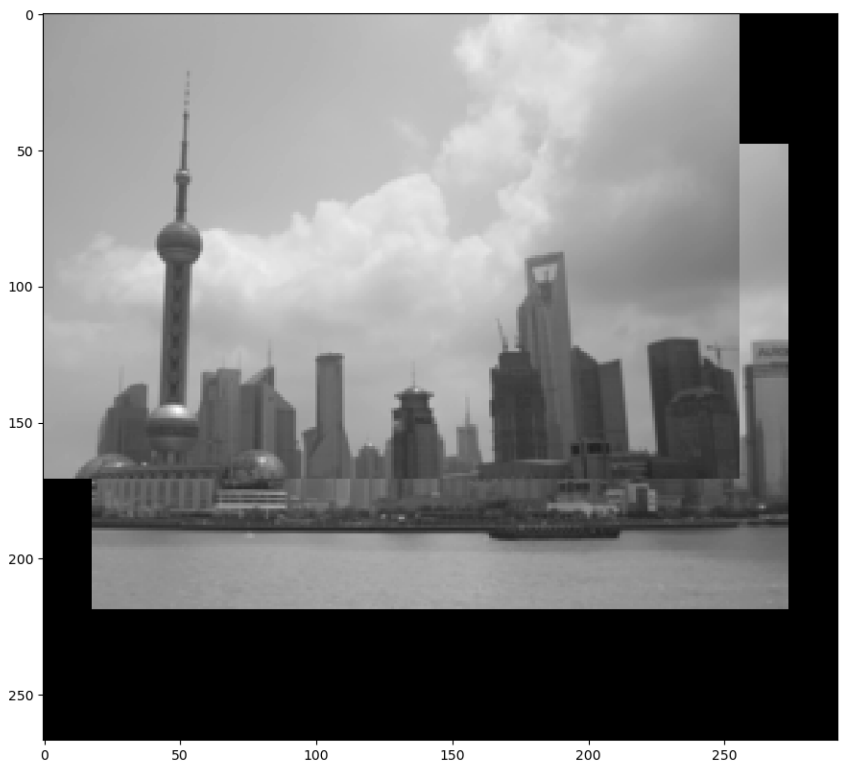
A simple steps that I used for implmenting this hough transform:

1. Store the difference of x-axis and y-axis into different list. The order of their difference is the same of xs0

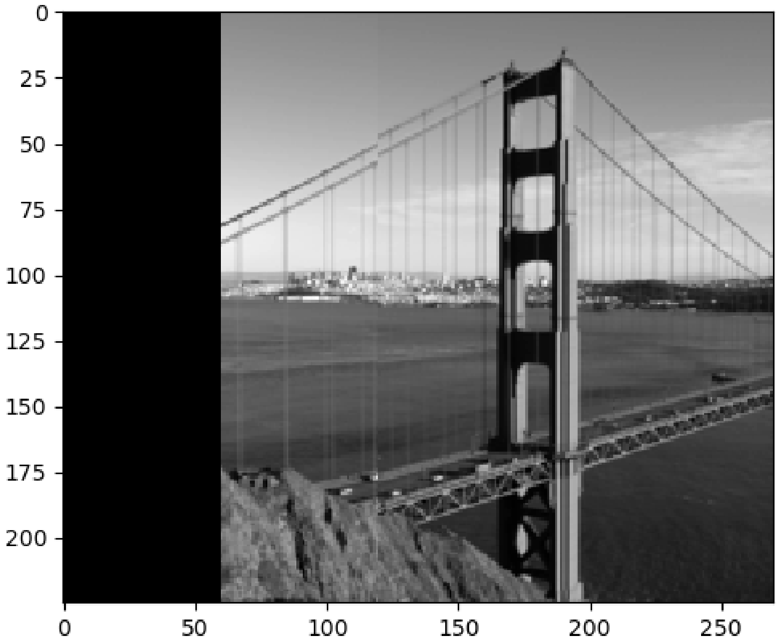
2. I use a dictionary in the formart of Dic[x_axisdiff/bin_width,y_axisdiff/bin_width] to represent for the hough space and to store the scores. After trials, I decide to use the bin width of size 3.

3. Get the combination of x_axisdiff and y_axisdiff with the most scores and return their indices.

A sample output with ShangHai with 100 Interest points and bin width of 3 look like this:



A sample output with Golden Gate with 100 Interest points and bin width of 3 look like this:



0
25
50
75
100
125
150
175
200
0
50
100
150
200
250