CS558-A Computer Vision Homework 4

- Prasad Naik (CWID: 20016345)
- Atharv Subhekar (CWID: 20015840)

File Structure: The submitted file contains 2 zip files, one for the source code and one for generated output.

Programming Language: Python. Note: The code in this homework uses certain keywords which requires Python version 3.10 or above. Please run this code with Python 3.10 or above.

Libraries Used:

- OpenCV: Reading and writing images.
- Numpy: Numerical operations like mean. Also used for matrix manipulation.
- Random: Generating random samples from data.
- OS: File Manipulation
- Scipy: Library for mathematical operations
- Sk Image: Image data operations. Plot Matches and affine transform.

Problem 1: Feature Detection

Inputs:

- - Input Image: uttower_left.jpg and uttower_right.jpg.
- - Parameters : Sigma for Gaussian Filter = 1
- - Parameters : Window Size for patch similarity = 9*9
- - Max number of iterations: Default value of 100.

Functions:

- **gaussian_filter (input_image, sigma = 1):** Function for passing gaussian filter over every pixel in the image.
- **partial_derivative (input_image):** Function to calculate the first derivatives and second derivatives.
- **corner_response(input_image):** A function to calculate the corner response function using the second derivatives of the input image. This function returns a list of corner location coordinates.
- **corners(input_image):** A function to store the first 1000 corners from the corner response function. This function returns a list of [R,x,y] where R is the cornerness, x is the x-coordinate and y is the y-coordinate.
- **NonMaximumSuppression(input_image):** A function to suppress the corner key points in a 3*3 window. Removing neighboring pixels if they are lower than the center pixel.
- **SSD(final_image1, img1, final_image2, img2):** A function for calculating keypoint similarity. Pixel wise comparison between the final images and using the original image pixel intensity values for calculate the difference. This function returns a list of similarities between all the keypoints.

- **patch_SSD(final_image1, img1, final_image2, img2):** A function for calculating window wise patch similarity. Window wise comparison between the final images and

using the original image pixel intensity values for calculate the difference. This function

returns a list of similarities between all the keypoints.

- **patch_NCC(final_image1, img1, final_image2, img2):** A function for calculating

  window wise patch similarity. Window wise comparison between the final images and using the original image pixel intensity values for calculate the difference. This function returns a list of similarities between all the keypoints.

- **Plot(sim_list, img1, img2, s):** Function to plot match all the keypoints between image one and image 2.
- **main():** Main function to load images and call the gaussian_filter followed by NonMaximumSuppression function. Calling these functions initiates the entire process.

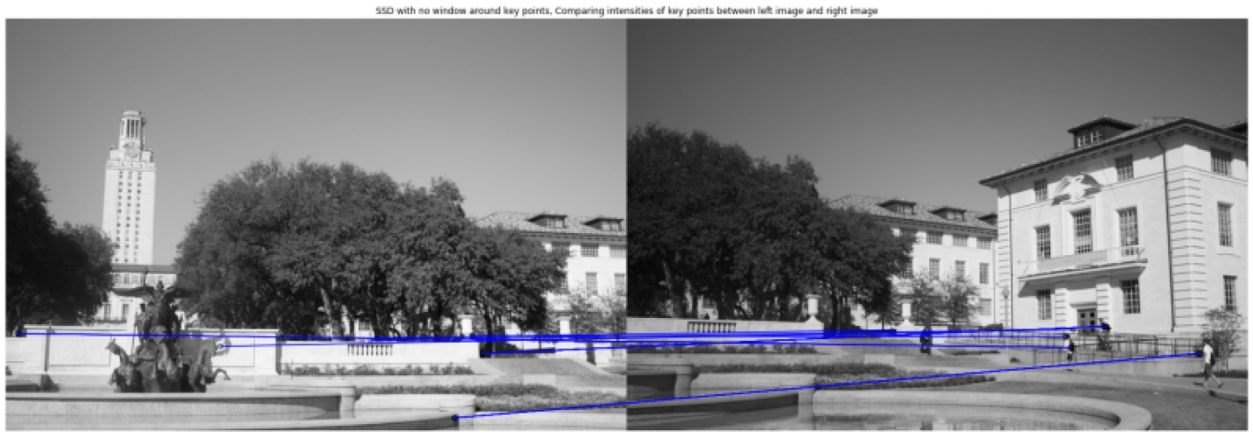
Outputs:
Left Corner Image

Non Maximum Suppression (Left Corner Image)

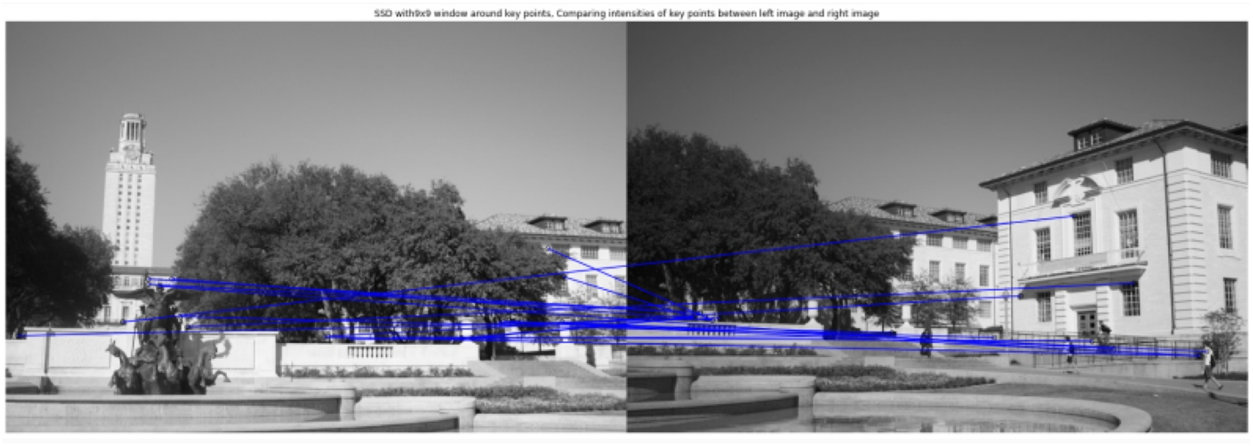
Right Corner Image

Non Maximum Suppression (Right Corner Image)

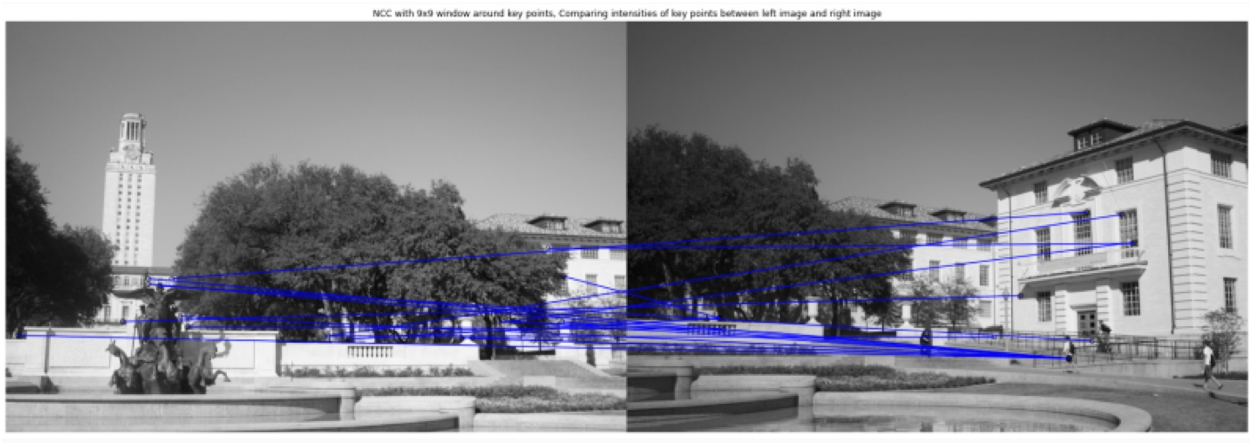
Keypoint SSD


SSD with no window around key points, Comparing intensities of key points between left image and right image

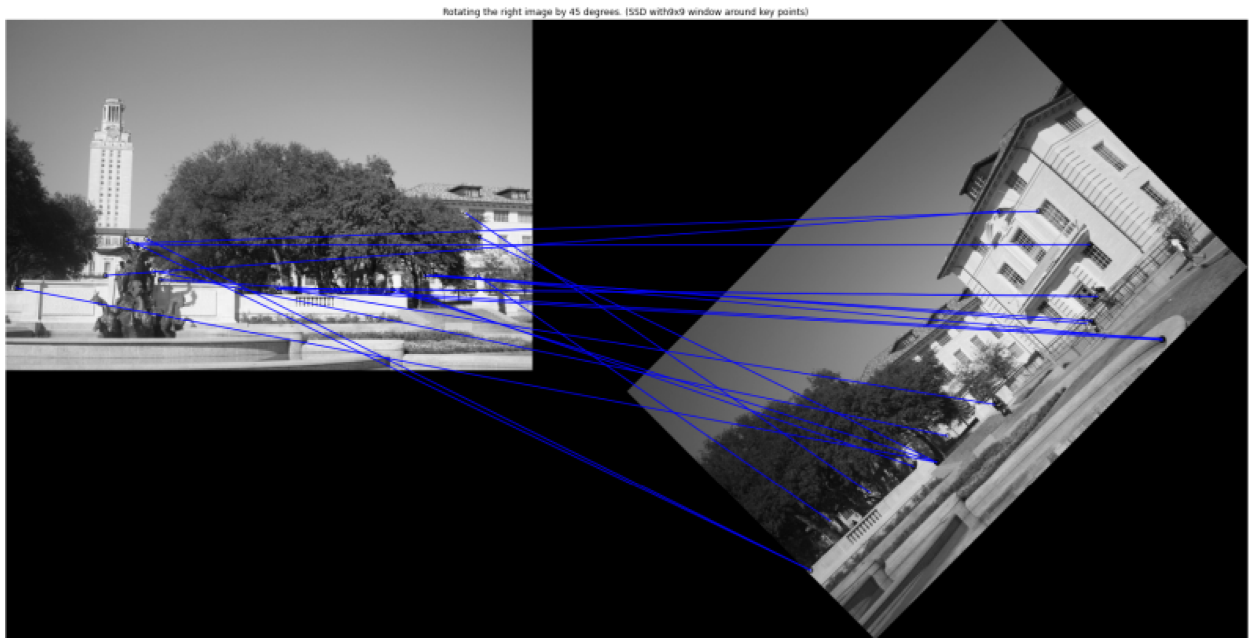
Patch SSD


SSD with9x9 window around key points, Comparing intensities of key points between left image and right image

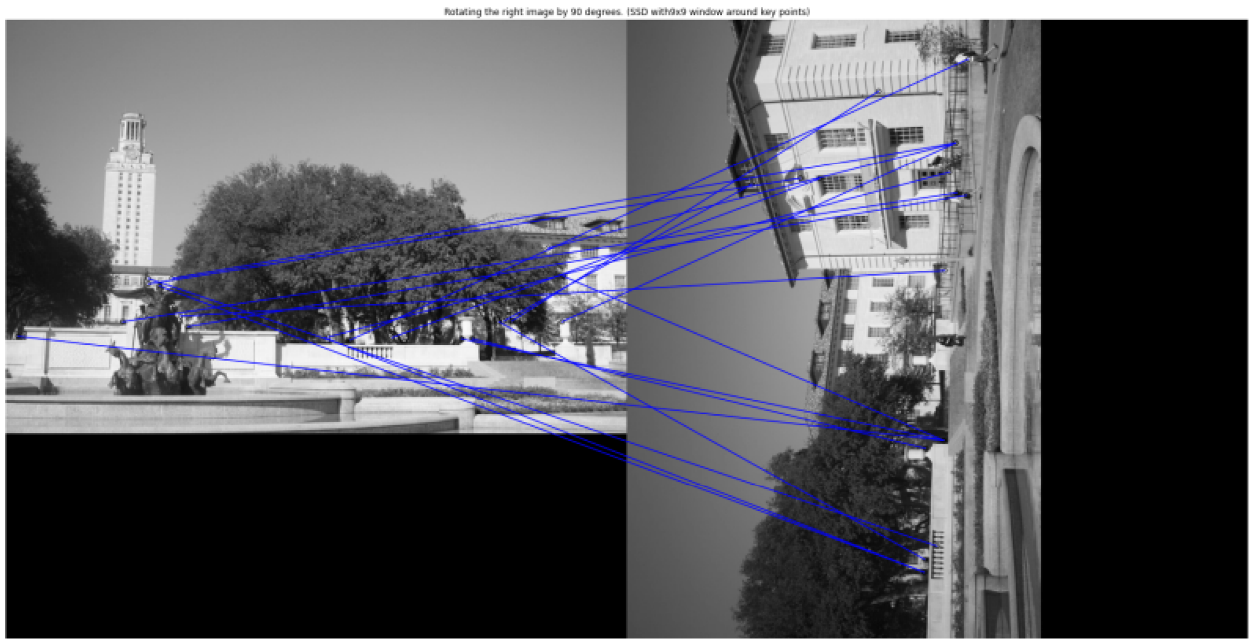
Patch NCC


NCC with 9x9 window around key points, Comparing intensities of key points between left image and right image

Patch SSD with rotation (45):


Rotating the right image by 45 degrees. (SSD with9x9 window around key points)

Patch SSD with rotation (90):


Rotating the right image by 90 degrees. (SSD with9x9 window around key points)

Patch SSD with rotation (180):

Difference between SSD and NCC : SSD works fine until a dark film is overlated on the image. The main difference between both of them is, with NCC the brighter patches won't have advantage over the dark patches.

Performance : The correspondence points were not accurate as they were, when the image is not rotated. The more you rotate, the more inaccurate correspondences will be.

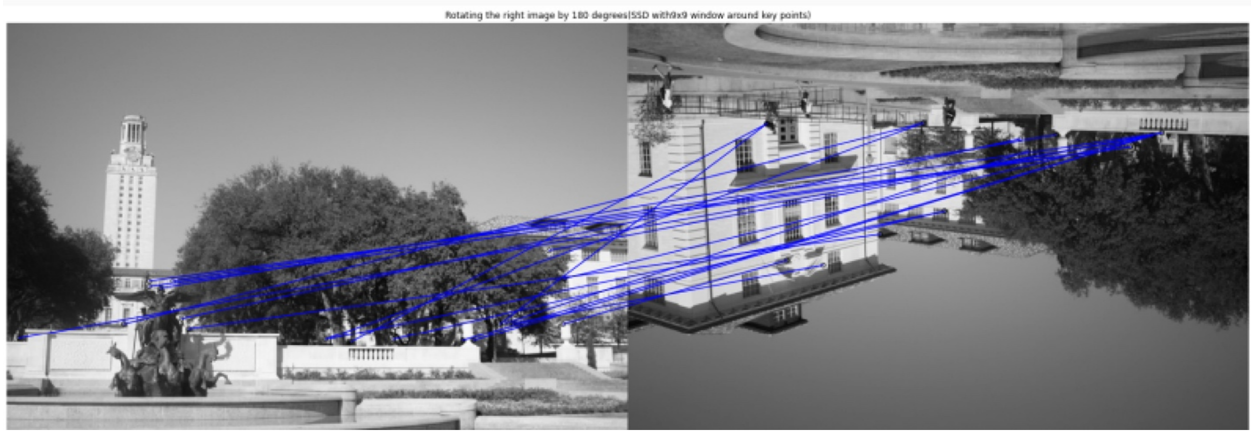
Solution :

- Image rectification might be able to solve this issue. It will align both the image in the same orientation.
- SIFT descriptors can also be used for this problem. They are invariant to rotations.

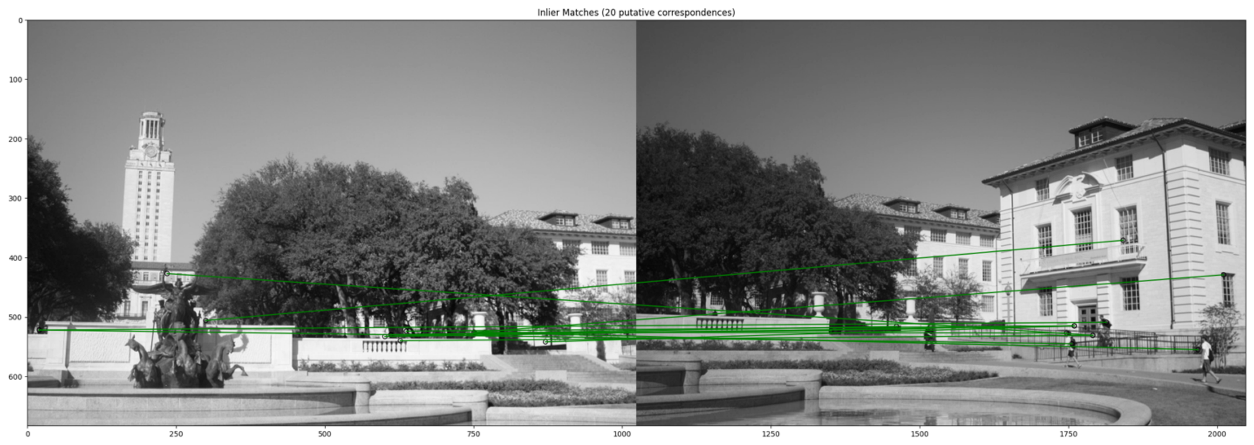
Problem 2: Two View Alignment Inputs:

• - Input Image: uttower_left.jpg and uttower_right.jpg. **Functions:**

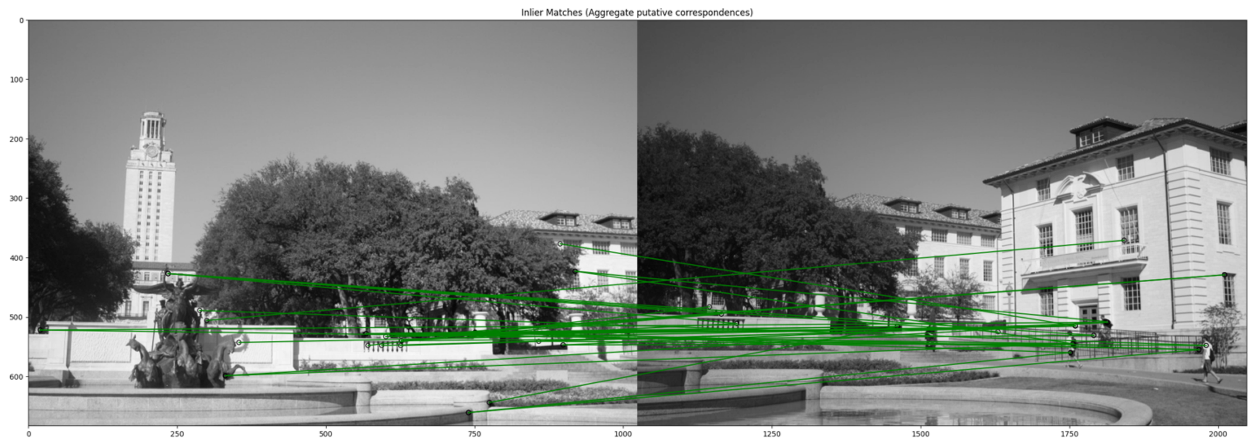
- **reprojection(input):** Function to randomly pick 3 samples from the best similarity list. This function returns the reprojected points and translation matrix.
- **Ransac(input, iterations = 100):** Function to implement ransac on reprojected points to find the optimal points with the most number of inliers.


Rotating the right image by 180 degrees(SSD with9x9 window around key points)

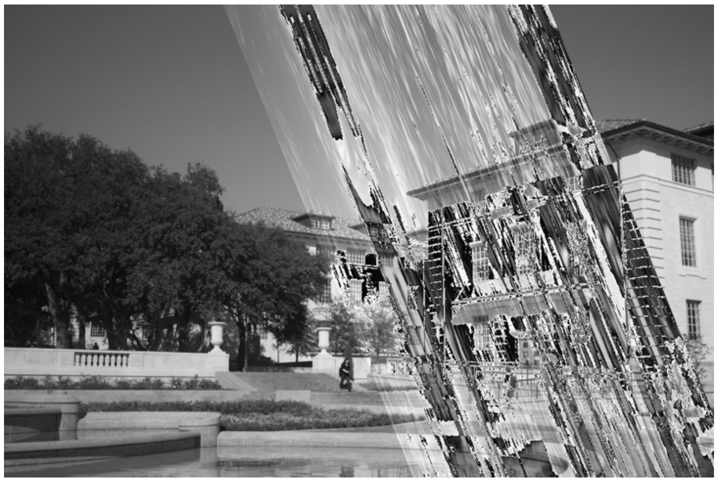
Outputs:
Inlier Matches best 20



Inlier Matches (aggregate of 20 best and 30 random correspondences)



Stiched Image



Observations:

NOTE : We tried performing RANSAC using our function as well as the built-in function from skimage library and the output is the same for both the cases.

The expected number of epoch train RANSAC on the keypoints is around 27 and the observed number of epochs is 4.After running the code multiple times, since it takes random 3 points from the putative correspondences, it might result in an exact output similar to the skimage output. Since the number of keypoints selected were top 1000 in 1st problem, we don't have enough correspondences to map the common parts between both the images. Since the tree is the common part between both the images, those keypoints were eliminated in problem one. This results in extreme warping as the model takes correspondences from the non similar sections of the image.