Assignment 2: Feature Detection and Image Stitching

Jayesh Kawli
Ezhilan Ilangovan

Submitted File Details

We are submitting **A2_Jayesh_Ezhilan.zip** which has the following list of files:

1. a2.cpp
2. a2, the Executable
3. Makefile
4. harris_cornersim1_mcfaddin.png – Harris corners of mcfaddin_1.png
5. harris_cornersim2_mcfaddin.png – Harris corners of mcfaddin_2.png
6. harris_cornersim1_sage.png – Harris corners of sage_1.png
7. harris_cornersim2_sage.png – Harris corners of sage_2.png
8. new_sage.png – stitched image of sage_1.png and sage_2.png
9. new_mcfaddin.png – stitched image of mcfaddin_1.png and mcfaddin_2.png
10. Some stitched image samples
11. TomasiCorner_mcfaddin.png – Tomasi corners of mcfaddin_1.png (Extra – refer to the last section)
12. TomasiCorner_sage.png – Tomasi corners of sage_1.png (Extra – refer to the last section)

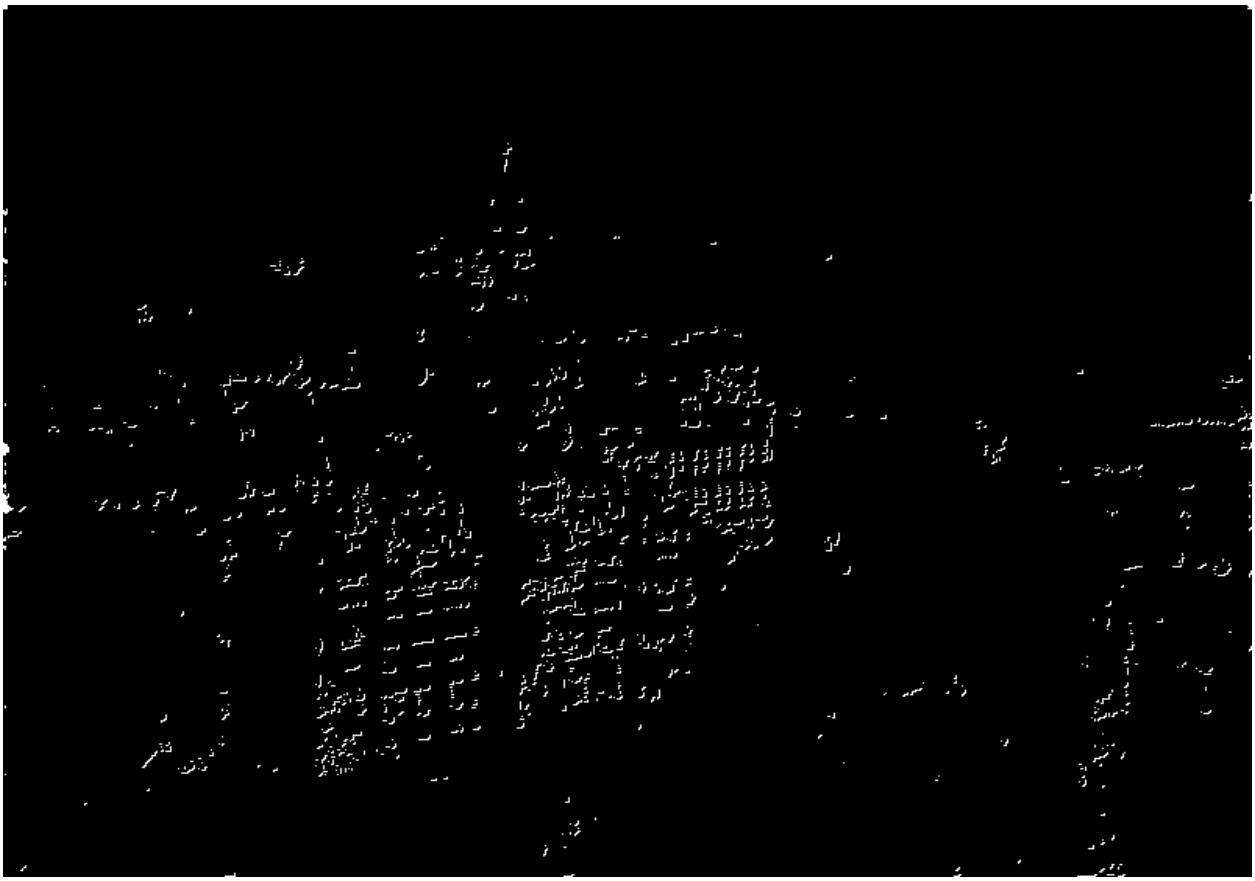
The command to run the executable: **./a2 stitched_image sage_1.png sage_2.png**

where, stitched_image is the output image and sage_1.png and sage_2.png are the input images. We are taking only 2 images at a time to stitch.

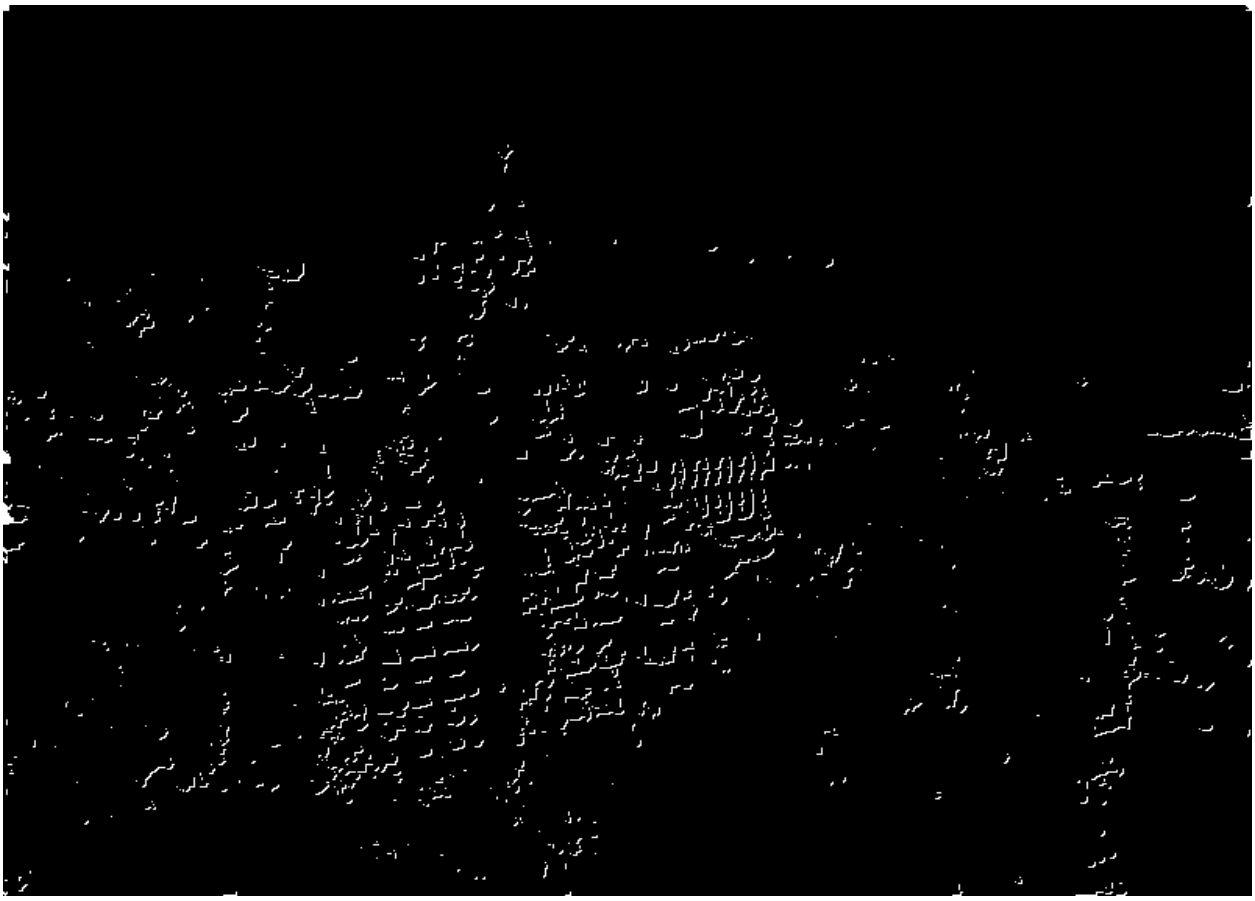
Harris Corner Detector

1. The first step in Harris Corner Detection is filtering the input images using a Gaussian filter. We are re-using the gaussian_filter function and convolve_seperable function from a1.cpp. We use a sigma of 3.0. The kernel size of the filter is 3X3.
2. The next step is finding the x-gradient and y-gradient of the images. We use the sobel_gradient_filter from a1.cpp. The filter that we are using in this function is [1 1 1] and [-1 0 1]. These filters are used alternatively as column and row filters to generate the x-gradient and y-gradient of the 2 input images.
3. R is calculated by the following formula: *R=((gaussxsquare[i][j]*gaussysquare[i][j]) - (pow(gaussxy[i][j],2)))- ((0.01)* pow((gaussxsquare[i][j]+gaussysquare[i][j]),2)).* The k value is taken as 0.01. Gaussian filter has a sigma of 1.0 and kernel size of 3X3.

4. The threshold value of R is taken to be 3000. After trying for various images, we decided 3000 to be optimum. If a point in the image has an R value greater than 3000, then we take it as a corner and store it in a vector of struct Coordinate.
5. '**points'** is the vector of struct Coordinate and has the list of corners in it. points is returned at the end of the function **find_corners_harris.**



Harris Corners of sage_1.png



Tomasi corners of sage_1.png

Descriptors Estimation

The size of the vector of corners is equal to the size of vector of descriptors. The following steps used to calculate the descriptors:

1. We are estimating the descriptors for every corner identified in the image in the function **invariant_descriptors.** This function takes in the image and its vector of corners. We are calculating 128-dimensional descriptors for every corner detected in the image. We are storing the descriptors in a vector of struct Descriptor.
2. The size of the vector of corners is equal to the size of vector of descriptors.
3. For every coordinate in the vector of coordinates (corners) of image 1 and image 2, we take 16 4X4 windows around the coordinate (4 windows of size 4X4 to the left top, left bottom, right top and right bottom to the corner).
4. We calculate the magnitude and gradient of each point in all the considered 4X4 windows in the function **thresh.**
   magnitude = sqrt(pow((r-l),2)+pow((b-t),2))
   gradient = atan((r-l)/(b-t)) double(180/PI)*

5. Depending upon the gradient angle, the magnitude is stored in the appropriate position of the vector of struct Descriptor. For e.g. if angle is between 0 and 45 then then we store magnitude at 0^{th} position of the vector.
6. We repeat step 5 for each corner coordinate in both images
7. Once above steps are done, we have a vector of 128 elements for every corner, that can uniquely identify that corner.

Translation Estimation

The translation in x and y direction of the two images is calculated by the function **translation_estimation.** This function takes in the 2 images along with its descriptors and corners.

translation_estimation(image1, image2, image1_descriptors, image2_descriptors, image1_coordinates, image2_coordinates);

The function returns a structure of type image_details. The returned structure will have a point in the image 1, a point in the image 2 and their corresponding x and y translation. The algorithm used is as follows:

1. We are comparing the first 10000 descriptors of image1 with the first 10000 descriptors of image 2.
2. The sum of the difference of the 128 dimensions of every 2 descriptors is calculated. The descriptors that have corresponds to the least sum is found. The corresponding coordinates in image 1 and image 2 are stored in a 2 dimensional array. The count of the corresponding x and y translation is noted in another 2-dimensional array.
3. From the count of the estimated translations that we maintain, we find the translation that occur the most and the corresponding coordinates in image 1 and image 2.
4. We store these details in a structure and return it to the stitch function that called translation_estimation function.

Stitching

Having found the translation, it's easy to stitch the 2 images, now. The stitched images of sage and mcfaddin images are shown below.

Stitched image of Sage_1 and Sage_2

Stitched image of mcfaddin_1 and mcfaddin_2

Extra

We have implemented Tomasi corner detector.

Tomasi Corner Detector

Tomasi Corner detector is the advanced version of Harris corner detector difference between two is that for the R value thus given by

R=det(M)-k*trace(M)^2

Where

det(M)=lambda1*lambda2
Trace(M) = lambda1+lambda2

However in Tomasi corner detector, it is calculated as

Min(lambda1,lambda2)

Consequentially, Tomasi performs better than the Harris corner detector. Even though it is just minor modification of harris, it gives better results for corner detection.