Naman Jain Vimal Kumar
1001670153

REPORT

PROBLEM 1.1:

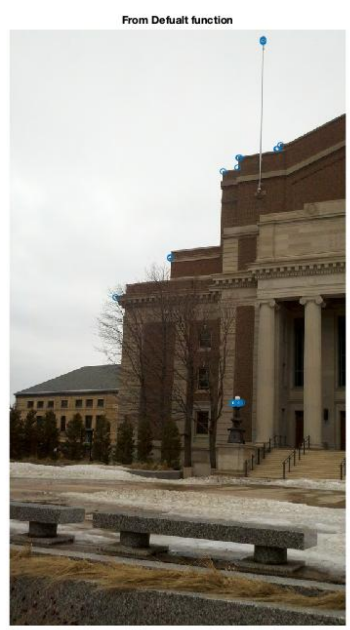
Write a MATLAB function [pts] = extract_corners(img, THRESH) that returns the locations of all corner points in an image, img, using the Harris corner detection algorithm. Use THRESH as the threshold on the response value of the detector. pts is a vector containing the locations of all corner points determined using non-maximal suppression. You may use the MATLAB function imgradientxy to compute the image derivatives.

Implementation:

- Returns edge points from the image.
- THRESH is the 'N' number of edge points required.
- img is the image path.
- Implementation is showed commented in the program.

To run:

$pts1 = extract\_corners(\text{'}Data/mall1.jpg\text{'}, 30)$



Note:

- Results for different images has been stored in **Output** folder with P_1.1 <img> .jpg
- Uncomment the end code to get the plots.

Naman Jain Vimal Kumar
1001670153

PROBLEM 1.2:

Write a MATLAB function [mpts1,mpts2] = match corners(img1,img2,pts1,pts2,WSIZE) that takes as input two images, img1 and img2, along with their detected corner locations, pts1 and pts2 (found using the extract corners function), and a window size, WSIZE. The output of the function will be two vectors, mpts1 and mpts2, containing the location of corresponding corners in the two images. Specifically, mpts1 and mpts2 contain the location of the corresponding corners in img1 and img2 found by computing the normalized cross correlation. The normalized cross correlation is defined as

$$NCC = \frac{(f - \bar{f})(g - \bar{g})}{\sqrt{\sum(f - \bar{f})^2 \sum(g - \bar{g})^2}}$$

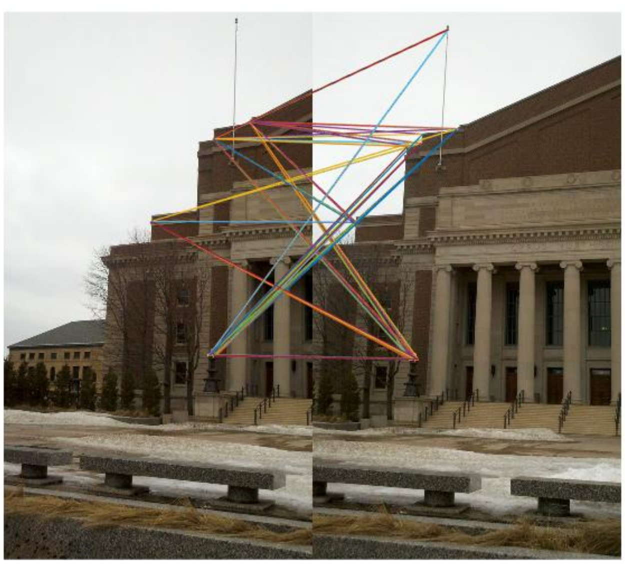
where f is a corner point in the first image, g is a corner point in the second image, and $\bar{f}$ and $\bar{g}$ are the mean intensity values within a window of size WSIZE centered at f and g in their respective images. The NCC is computed for every pair of corner points where the larger the value of the correlation the more likely that the corner points are matches.

Implementation:

- Returns matched points from the image1 and image2.
- pts1 and pts2 are points returned for img1 and img2 from extract_corners function.
- Img1 and img2 are the image paths.
- WSIZE is the window size [converted to nearest odd number size]
- Implementation is showed commented in the program.

To run:

$[mpts1_1, mpts1\_2] \ = \ match\_corners('Data/mall1.jpg', 'Data/mall2.jpg', pts1, pts2, 5);$



Note:

- Uncomment the end code to get the plots.

PROBLEM 1.3:

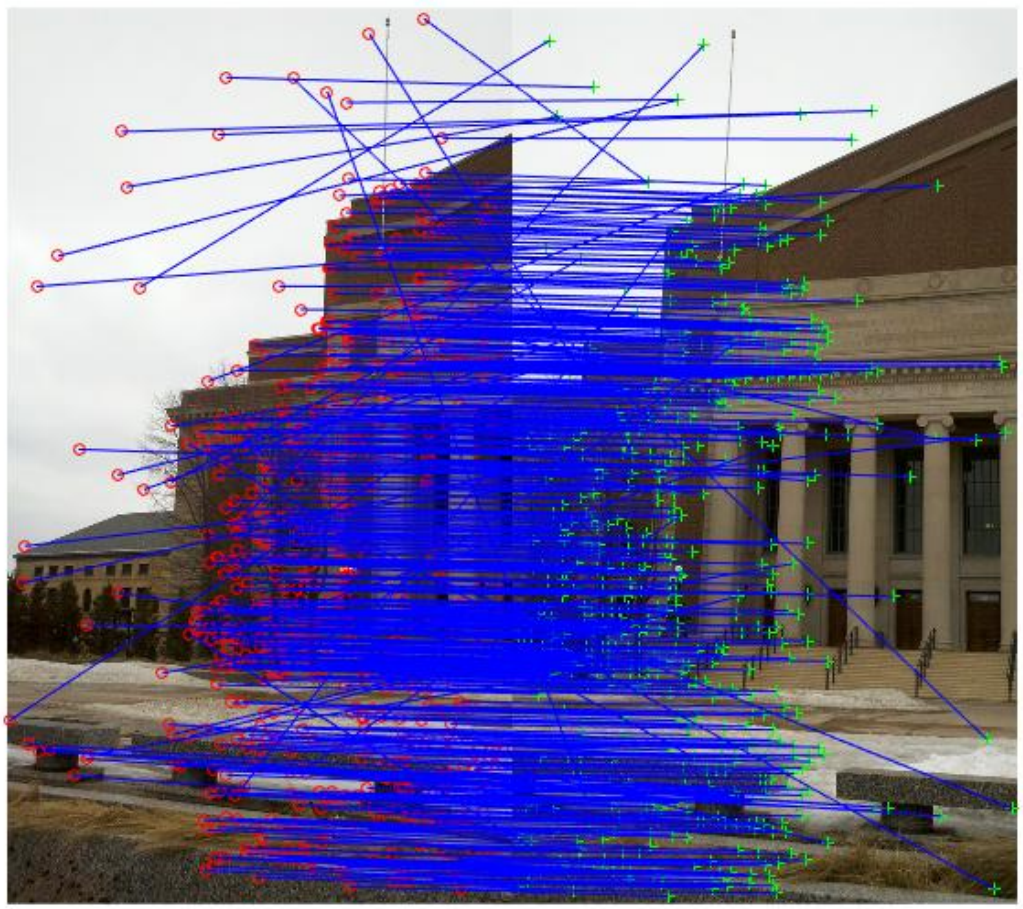
Use the vl_sift function from the SIFT toolbox available at www.vlfeat.org to detect the SIFT features in an input image. Match the obtained SIFT features from two images using the vl_ubcmatch function. In the PDF report, submit two images marking the location of the detected features and the matches found. You can show the matches using arrows in the two images placed side by side, or mark the position of the features using text labels on the image. Use the input images 'mall1.jpg' and 'mall2.jpg' for finding the features.

Implementation:

- Install VLFeat from www.vlfeat.org
- Img1 and img2 are the image paths.

To run:

- $sift\_comparision('Data/mall1.jpg','Data/mall2.jpg')$;



PROBLEM 2.1:

Write a MATLAB function H = compute homography(mpts1, mpts2) that takes as input two N ×3 vectors of homogeneous image coordinates, and returns as output the 3 × 3 homography matrix such that mpts2 = H mpts1 using least squares fitting. Note that the number of input points, N, must be greater than or equal to four in order to estimate the homography.

Implementation:

- Returns Homography matrix such that T(mpts2) = H * T(mpts1)
- Mpts1 and mpts2 are matched points generated by 2 images from match_corners function.
- Implementation is showed commented in the program.

To run:
$H = compute\_homography(mpts1, mpts2)$;

Proof:

```
>> a = [randi(10, 4,2), ones(4,1)]

a =

     9     8     1
     5     6     1
     4     4     1
     7     2     1

>> b = [randi(10, 4, 2), ones(4, 1)]

b =

     6     3     1
     3     5     1
     1     7     1
     8     4     1

>> H = compute_homography(a, b)

H =

   -3.8323   -1.0000   17.8944
    0.6273   -1.3478   -7.1615
   -0.2298   -0.3789    1.0000

>> X = H * transpose(a)

X =
```

```
 -24.5963   -7.2671   -1.4348  -10.9317
 -12.2981  -12.1118  -10.0435   -5.4658
  -4.0994   -2.4224   -1.4348   -1.3665

>> convertToHomogenous(transpose(X))

ans =

    6.0000    3.0000    1.0000
    3.0000    5.0000    1.0000
    1.0000    7.0000    1.0000
    8.0000    4.0000    1.0000

>> b

b =

     6     3     1
     3     5     1
     1     7     1
     8     4     1
```

Here **b** and **ans** are same.

PROBLEM 2.2:

Write a MATLAB function H = compute homography ransac(mpts1,mpts2,THRESH) that uses the RANSAC algorithm to reject the outlier matches in mpts1 and mpts2, and then uses only the inliers to find H. Since H can be computed using four points, at each iteration the of algorithm you must sample four matches and 2 use these corresponding pairs to determine the candidate Hˆ . Use the symmetrical reprojection error as the distance function to determine the inliers. The symmetrical reprojection error is defined as

$$d = ||mpts2 - H*mpts1|| + ||mpts1 - H^{-1}*mpts2||.$$

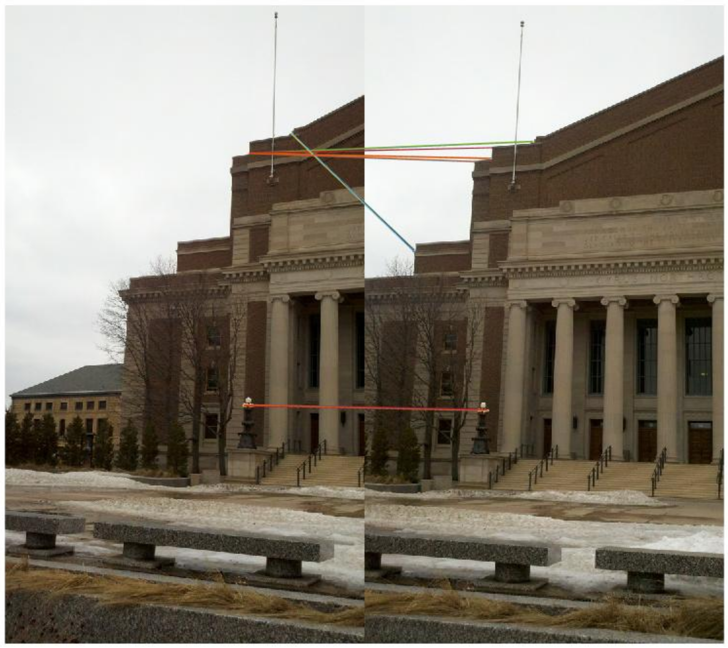
*Since mpts1 and mpts2 are homogeneous coordinates, you have to normalize H*mpts1 and* H^{-1}**mpts2 before computing the norm. To decide if a point is an inlier, use THRESH as a threshold for the error. If THRESH is 0, then use 2.5σ as the threshold where σ is the standard deviation of the error. Try out different values of THRESH to see which ones give reasonable performance. Use only the inliers found to compute the return value using the least squares compute homography function. Additional information can be found in [1]. In particular, refer to steps 1-3 of Algorithm 4.6 in section 4.8 'Automatic computation of a homography'.*

Implementation:

- Returns Homography matrix such that T(mpts2) = H * T(mpts1) based on RANSAC generated inliers.
- Mpts1 and mpts2 are matched points generated by 2 images from match_corners function.

- THRESH is the value compared with d.

To run:
$H = compute\_homography\_ransac(mpts1, mpts2, 200, 'Data/mall1.jpg', 'Data/mall2.jpg');$



PROBLEM 3.1:

After estimating the homography between two images, they can be stitched together using Laplacian pyramid blending. Write a MATLAB function blend = blend images(img1,img2,mask1,mask2) that takes as input two images and their corresponding binary masks and blends them together into a single output image using five pyramid levels. For additional information, see section 3.5.5 in [2] (available online).

Implementation:

- Returns gaussian series of blended images.
- Img1 and img2 are path to respective images.
- Only single mask has been used. [As said by Prof.]
- Actual implementation explained in the code.
- Mask with respect to img1.

To run:
$mask = [ones(979, 276)];$
$mask = [mask1, zeros(979, 276)];$
$blend\_images('Data/mall1.jpg', 'Data/mall2.jpg', mask);$

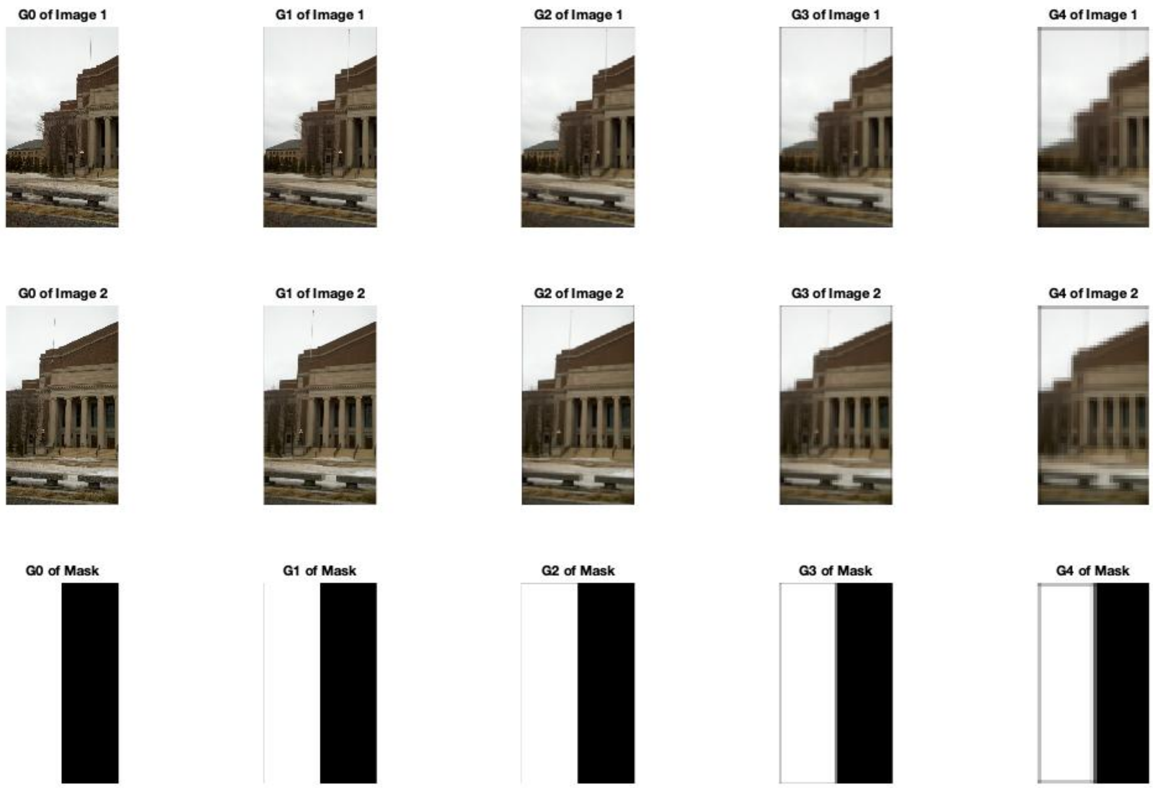


Figure 1 Gaussian

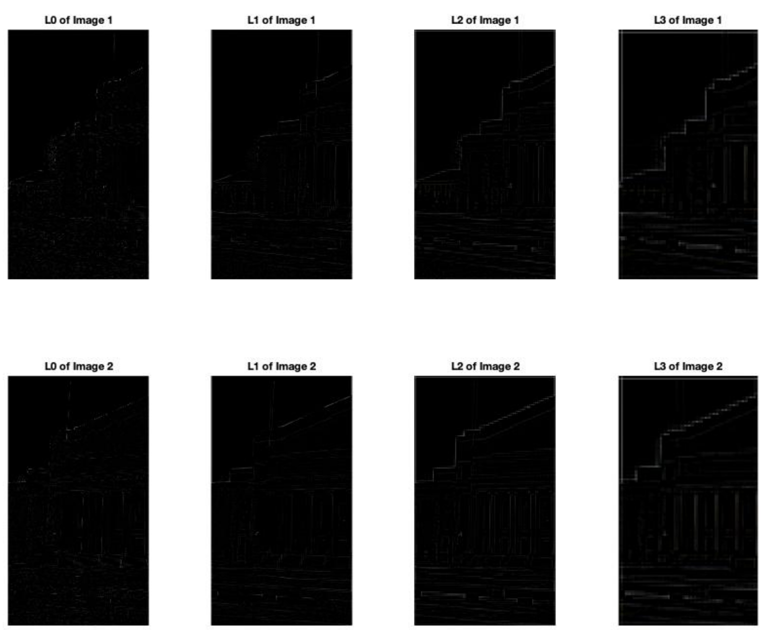


Figure 2 Laplacians

Figure 3 Blended Images

Note:

- mask2 = 1 – mask [Mask for second image]
- 5 levels of Gaussian pyramids are constructed.
- Uncomment the required code to generate plots.

PROBLEM 4.1:

After estimating the homography between two images, they can be stitched together using Laplacian pyramid blending. Write a MATLAB function blend = blend_images(img1,img2,mask1,mask2) that takes as input two images and their corresponding binary masks and blends them together into a single output image using five pyramid levels. For additional information, see section 3.5.5 in [2] (available online).

[Partially done]