Applied Computer Vision (CS-696)
Homework Assignment 4

Name – Dhaval Harish Sharma **Red ID – 824654344**

Question:

The goal of this assignment is to create a local feature matching algorithm using techniques described in Szeliski chapter 4.1. The pipeline we suggest is a simplified version of the famous SIFT pipeline. The matching pipeline is intended to work for instance-level matching -- multiple views of the same physical scene.

Solution:

The three major steps of the local feature matching algorithm are listed and explained as follows:

(1) Interest Point Detection:

In order to create features to match images against, interest points, the position of the features, must be selected. Corners are often good interest points to select since they are different than their neighboring points. To select interest points I used the Harris corner detector algorithm as mentioned in the class notes. The Harris Corner Detection algorithm is explained as follows:

Harris Corner Detection:

The Harris Corner detector works by finding the x and y derivatives of the images. This is accomplished by filtering each image with the Sobel and its transpose. Next the derivatives are squared to form two squared derivatives images and an x-derivative times y-derivative image. The underlying mathematics relies on eigenvalues to describe how corner-like a given patch of an image is, however, this math can be simplified to use the squared derivatives to speed up computations. Pixels that are in a patch with a high corner score are considered distinct and get passed onto the next stage. In order to only operate on single pixels, non-maxima suppression is also run which finds the maximum of patches with scores above the threshold.

(2) Local Feature Description:

After interest points are selected, features must be created from those points. For local feature description, I used SIFT-like features. The steps are as follows:

(i) Estimate the gradients of the image using filters, one filter for each of the eight gradient directions.

(ii) For each interest point, create a 4x4 grid around it. Each of the cells contains a histogram made up of the gradients of the pixels in the cell.

(iii) For simplicity, each pixel only contributes its gradient to the orientation it is closest to.

(iv) This grid of histograms is flattened into a vector to be used as the feature for an interest point.

(3) Feature Matching:

We match points using the Nearest Neighbor Distance Ratio (NNDR). For each point in image 1, we calculate the proximity of the point's feature descriptor to the descriptors of all the points in image 2. We examine the ratio between the distance to the point "closest " to point A (in terms of the distance of feature descriptors) and the second closest point. If the ratio is sufficiently low (a threshold that we can manipulate as a parameter) then we can safely assume the match is a valid one. The lower the ratio, the higher the

confidence is of the match being a good one. I ended up setting the threshold at about 0.7, as this was not prohibitively low to eliminate all points but was also rather discerning about which matches to accept. A higher threshold will mean more matches will be made, but less accuracy can be guaranteed.
(i) Input: 'features1' and 'features2' are the n x feature dimensionality features from the two images.
(ii) For each feature in image 1, the nearest neighbor distance ratio is computed with respect to all features in image 2.
(iii) If the ratio is less than a threshold value (0.7), the nearest neighbor of the feature is considered a match and included in the result.
(iv) Sort the matches so that the most confident ones are at the top of the list.

The code files are as follows:

[get_interest_points.m]

```
% Initializing the parameters
alpha = 0.04;
gaussian = fspecial('Gaussian', [25 25], 1);
[gx, gy] = imgradientxy(gaussian);

% Applying the filter to the image
ix = imfilter(image, gx);
iy = imfilter(image, gy);

% Suppress gradients near the edges
ix([(1:feature_width) end-feature_width + (1:feature_width)], :) = 0;
ix(:, [(1:feature_width) end-feature_width + (1:feature_width)]) = 0;
iy([(1:feature_width) end-feature_width + (1:feature_width)], :) = 0;
iy(:, [(1:feature_width) end-feature_width + (1:feature_width)]) = 0;

large_gaussian = fspecial('Gaussian', [25 25], 2);
ixx = imfilter(ix.*ix, large_gaussian);
ixy = imfilter(ix.*iy, large_gaussian);
iyy = imfilter(iy.*iy, large_gaussian);
har = ixx.*iyy - ixy.*ixy - alpha.*(ixx+iyy).*(ixx+iyy);
thresholded = har > 10*mean2(har); % Adaptive threshold

sliding = 1;

switch sliding
    case 0
        components = bwconncomp(thresholded);
        width = components.ImageSize(1);
        x = zeros(components.NumObjects, 1);
        y = zeros(components.NumObjects, 1);
        confidence = zeros(components.NumObjects, 1);
        for ii = 1:(components.NumObjects)
            pixel_ids = components.PixelIdxList{ii};
            pixel_values = har(pixel_ids);
            [max_value, max_id] = max(pixel_values);
            x(ii) = floor(pixel_ids(max_id) / width);
            y(ii) = mod(pixel_ids(max_id), width);
            confidence(ii) = max_value;
        end
```

```
    case 1
        har = har.*thresholded;
        har_max = colfilt(har, [feature_width feature_width], 'sliding', @max);
        har = har.*(har == har_max);
        [y, x] = find(har > 0);
        confidence = har(har > 0);
end
```

[get_features.m]

```
% Initializing the features
num_points = size(x, 1);
features = zeros(num_points, 128);

% Calculating the small and large gaussian
small_gaussian = fspecial('Gaussian', [feature_width feature_width], 1);
large_gaussian = fspecial('Gaussian', [feature_width feature_width], feature_width / 2);

% Applying the gradient to the image
[gx, gy] = imgradientxy(small_gaussian);
ix = imfilter(image, gx);
iy = imfilter(image, gy);

get_octant = @(x,y) (ceil(atan2(y, x) / (pi / 4)) + 4);

orients = arrayfun(get_octant, ix, iy);
mag = hypot(ix, iy);
c_size = feature_width / 4;

for ii = 1:num_points
    frame_x_range = (x(ii) - 2 * c_size): (x(ii) + 2 * c_size - 1);
    frame_y_range = (y(ii) - 2 * c_size): (y(ii) + 2 * c_size - 1);
    frame_mag = mag(frame_y_range, frame_x_range);
    frame_mag = frame_mag.*large_gaussian;
    frame_orients = orients(frame_y_range, frame_x_range);

    % Looping through each cell in the frame
    for xx = 0:3
        for yy = 0:3
            cell_orients = frame_orients(xx * 4 + 1:xx * 4 + 4, yy * 4 + 1:yy * 4 + 4);
            cell_mag = frame_mag(xx * 4 + 1:xx * 4 + 4, yy * 4 + 1:yy * 4 + 4);
            for o = 1:8
                f = cell_orients == o;
                features(ii, (xx * 32 + yy * 8) + o) = sum(sum(cell_mag(f)));
            end
        end
    end
end
features = diag(1./sum(features, 2)) * features; % Normalize feature vectors

end
```

[match_features.m]

```
% Take a threshold value
threshold = 0.7;
dist_matrix = pdist2(features1, features2, 'euclidean');

[sorted_dist_matrix, indices] = sort(dist_matrix, 2);
inverse_confidences = (sorted_dist_matrix(:, 1)./sorted_dist_matrix(:, 2));
confidences = 1./inverse_confidences(inverse_confidences < threshold);

% Matching the features
matches = zeros(size(confidences, 1), 2);
matches(:, 1) = find(inverse_confidences < threshold);
matches(:, 2) = indices(inverse_confidences < threshold, 1);
%>

% Sort the matches so that the most confident onces are at the top of the
% list. You should probably not delete this, so that the evaluation
% functions can be run on the top matches easily.
[confidences, ind] = sort(confidences, 'descend');
matches = matches(ind,:);
```

The input images are as follows:

The output of these input images is as follows:

(1) Case 1 (Connected Components):

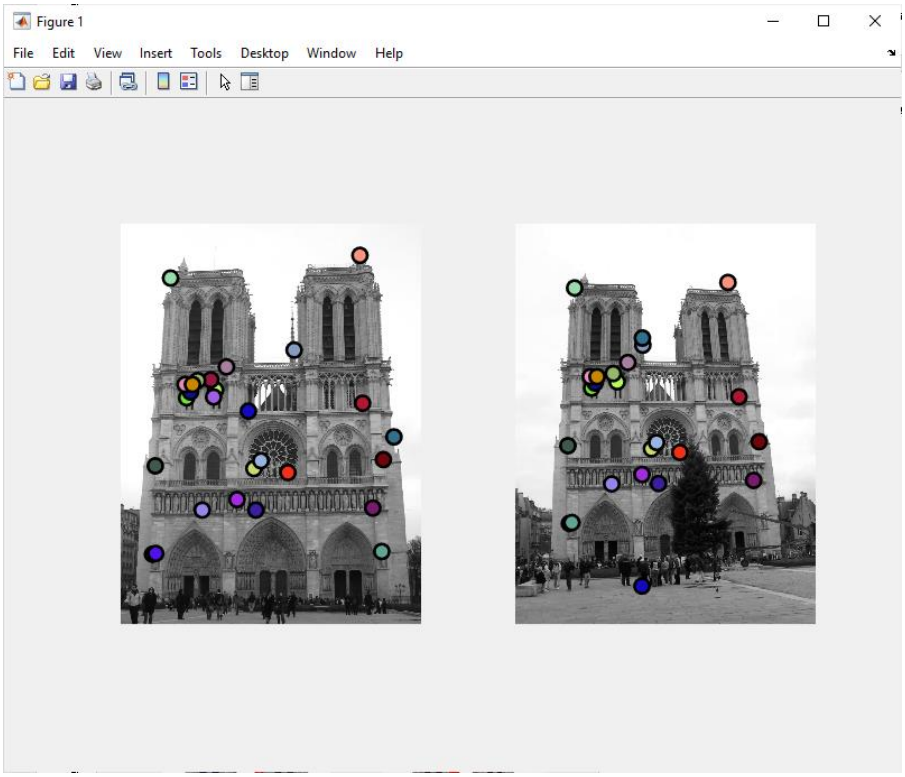


Figure 1
File Edit View Insert Tools Desktop Window Help

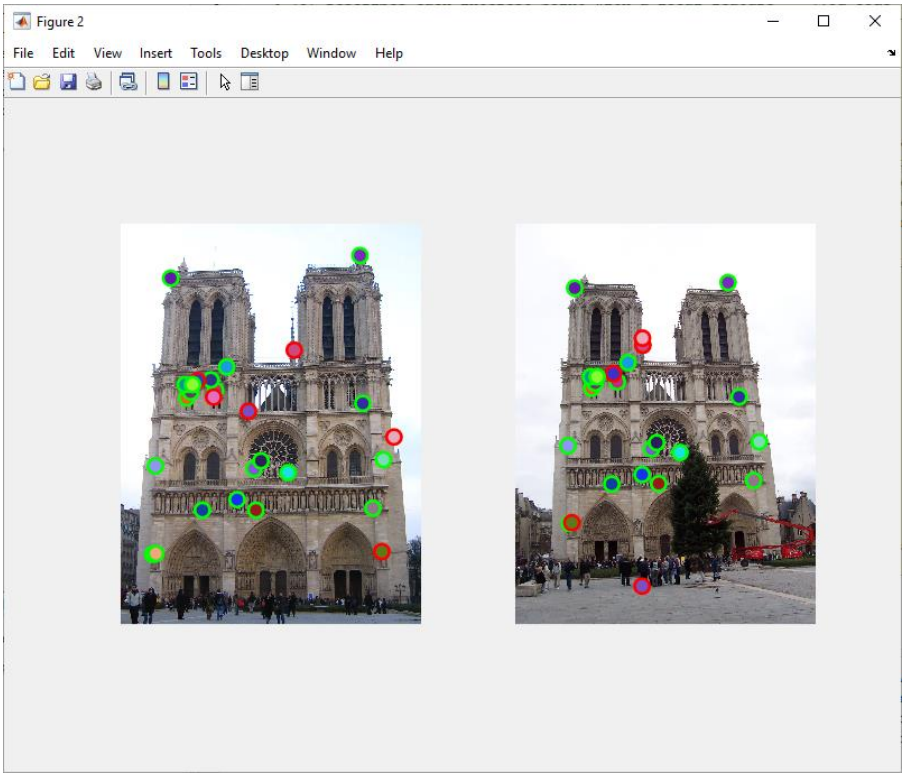
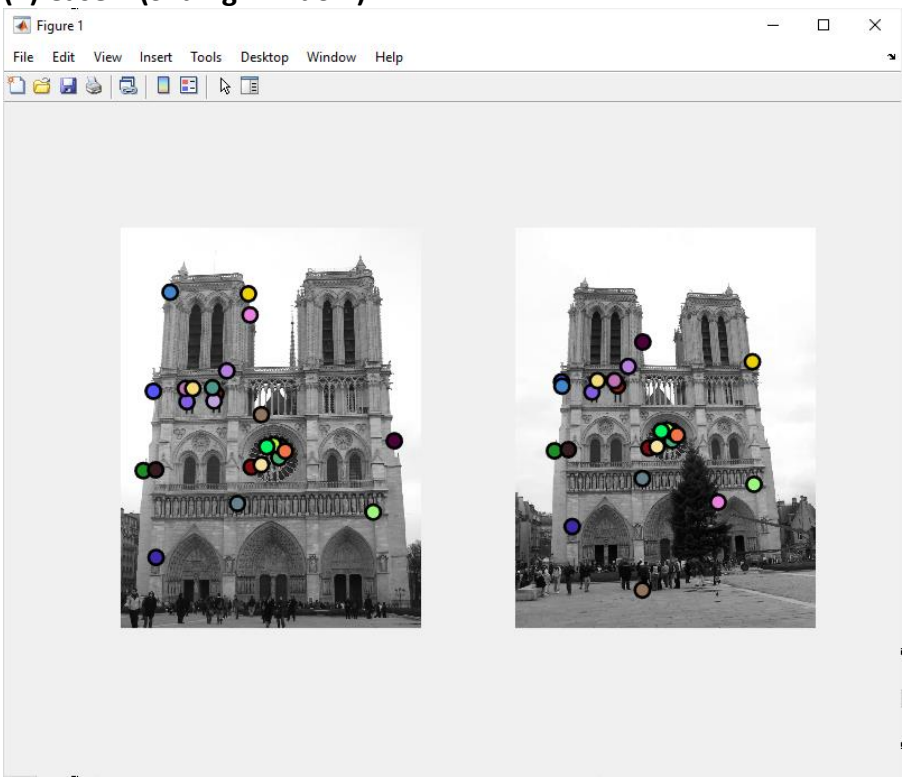
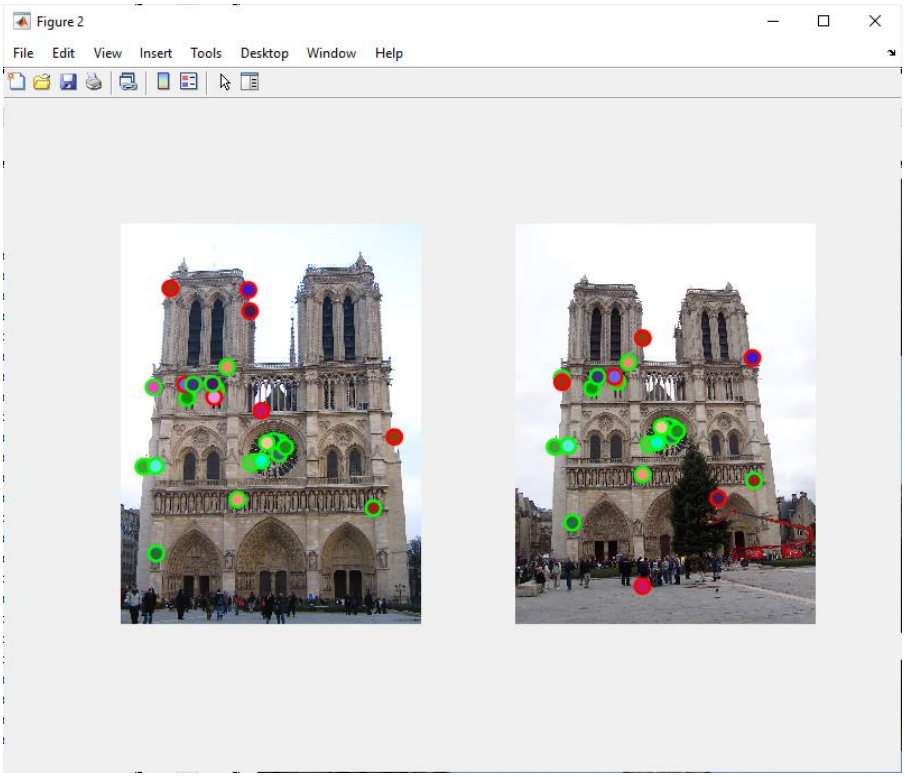


Figure 2
File Edit View Insert Tools Desktop Window Help

```
>> proj4
Saving visualization to vis.jpg
(  486,  852) to (  518,  804) g.t. point   34 px. Match error    4 px.  correct
(  164, 1692) to (  276, 1522) g.t. point   17 px. Match error    3 px.  correct
(  476,  888) to (  388,  836) g.t. point    4 px. Match error  121 px.  incorrect
(  338,  892) to (  388,  836) g.t. point    3 px. Match error    3 px.  correct
(  682, 1254) to (  692, 1144) g.t. point   67 px. Match error   10 px.  correct
(  462,  800) to (  498,  762) g.t. point   40 px. Match error    1 px.  correct
(  418, 1466) to (  490, 1322) g.t. point   65 px. Match error    5 px.  correct
(  542,  734) to (  572,  706) g.t. point   32 px. Match error   18 px.  correct
(  654,  960) to (  644, 1840) g.t. point    4 px. Match error  942 px.  incorrect
(  180, 1688) to (  288, 1518) g.t. point    5 px. Match error    2 px.  correct
(  358,  862) to (  406,  810) g.t. point   33 px. Match error    7 px.  correct
( 1344, 1208) to ( 1238, 1108) g.t. point   56 px. Match error    9 px.  correct
(  692, 1466) to (  728, 1320) g.t. point   11 px. Match error    6 px.  correct
( 1290, 1456) to ( 1210, 1304) g.t. point   44 px. Match error    1 px.  correct
(  330,  824) to (  382,  778) g.t. point   66 px. Match error   13 px.  correct
( 1224,  164) to ( 1080,  300) g.t. point   19 px. Match error    4 px.  correct
(  888,  648) to (  648,  618) g.t. point  154 px. Match error  196 px.  incorrect
(  394,  804) to (  498,  762) g.t. point   31 px. Match error   68 px.  incorrect
(  596, 1412) to (  644, 1274) g.t. point   56 px. Match error   10 px.  correct
(  178, 1240) to (  268, 1128) g.t. point   16 px. Match error    3 px.  correct
(  856, 1274) to (  836, 1162) g.t. point   19 px. Match error    7 px.  correct
( 1238,  920) to ( 1136,  880) g.t. point   20 px. Match error    2 px.  correct
(  718, 1214) to (  718, 1112) g.t. point   24 px. Match error   17 px.  correct
(  256,  280) to (  302,  328) g.t. point   15 px. Match error    4 px.  correct
(  368,  824) to (  416,  778) g.t. point   63 px. Match error   14 px.  correct
( 1398, 1092) to (  646,  582) g.t. point   69 px. Match error  742 px.  incorrect
( 1336, 1678) to (  288, 1518) g.t. point    2 px. Match error  974 px.  incorrect
21 total good matches, 6 total bad matches
Saving visualization to eval.jpg
```

(2) Case 2 (Sliding Window):






```
>> proj4
Saving visualization to vis.jpg
(  488,  852) to (  520,  804) g.t. point   34 px. Match error    4 px.  correct
(  810, 1178) to (  796, 1082) g.t. point   28 px. Match error    4 px.  correct
(  668, 1226) to (  678, 1120) g.t. point   46 px. Match error    9 px.  correct
(  168,  838) to (  236,  784) g.t. point   55 px. Match error   12 px.  correct
(  478,  888) to (  390,  836) g.t. point    3 px. Match error  121 px.  incorrect
(  340,  892) to (  390,  836) g.t. point    3 px. Match error    3 px.  correct
(  182, 1688) to (  290, 1518) g.t. point    5 px. Match error    2 px.  correct
(  816, 1120) to (  798, 1036) g.t. point   36 px. Match error    8 px.  correct
( 1400, 1092) to (  648,  582) g.t. point   68 px. Match error  742 px.  incorrect
(  782, 1112) to (  770, 1028) g.t. point    3 px. Match error    2 px.  correct
( 1292, 1456) to ( 1212, 1304) g.t. point   42 px. Match error    1 px.  correct
(  470,  820) to (  504,  778) g.t. point   54 px. Match error    4 px.  correct
(  116, 1244) to (  200, 1134) g.t. point    3 px. Match error    4 px.  correct
(  254,  332) to (  238,  806) g.t. point   56 px. Match error  455 px.  incorrect
(  544,  734) to (  574,  706) g.t. point   34 px. Match error   18 px.  correct
(  842, 1144) to (  822, 1054) g.t. point   55 px. Match error   17 px.  correct
(  662,  448) to ( 1030, 1394) g.t. point  101 px. Match error  974 px.  incorrect
(  180, 1240) to (  270, 1128) g.t. point   16 px. Match error    3 px.  correct
(  720,  958) to (  646, 1840) g.t. point   55 px. Match error  941 px.  incorrect
(  332,  824) to (  504,  778) g.t. point   66 px. Match error  124 px.  incorrect
(  654,  340) to ( 1204,  682) g.t. point   30 px. Match error  645 px.  incorrect
(  720, 1214) to (  720, 1112) g.t. point   22 px. Match error   17 px.  correct
(  748, 1122) to (  742, 1034) g.t. point   36 px. Match error    5 px.  correct
(  598, 1412) to (  646, 1274) g.t. point   55 px. Match error   10 px.  correct
(  370,  824) to (  418,  778) g.t. point   61 px. Match error   14 px.  correct
18 total good matches, 7 total bad matches
Saving visualization to eval.jpg
```

The other additional input images are as follows:

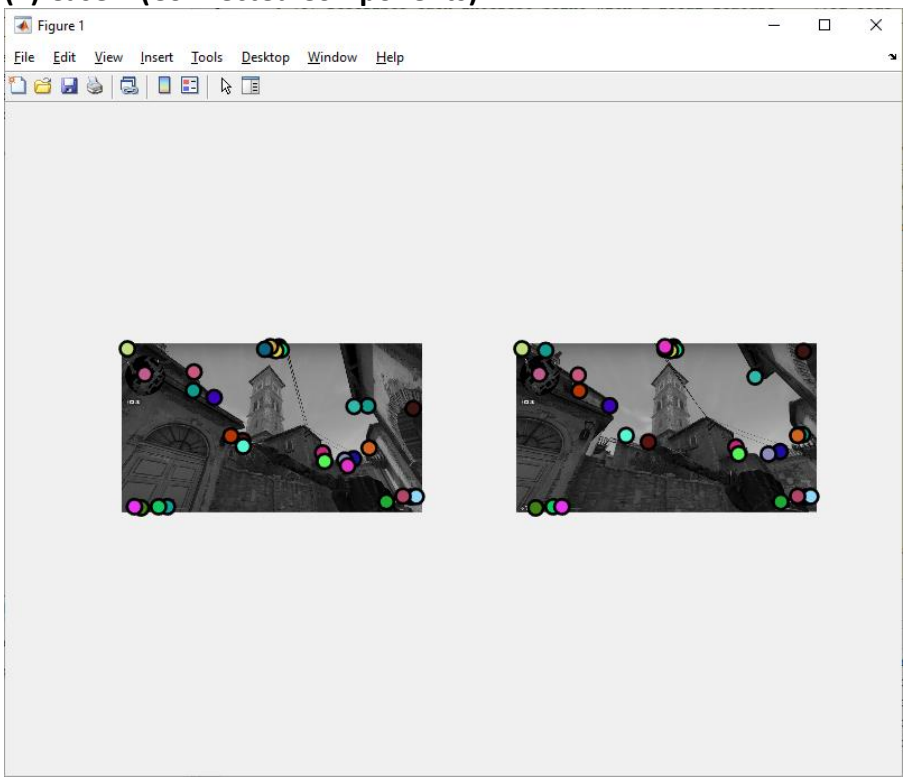
For these additional images, I have added the ground-truth values. The steps for doing that are as follows:

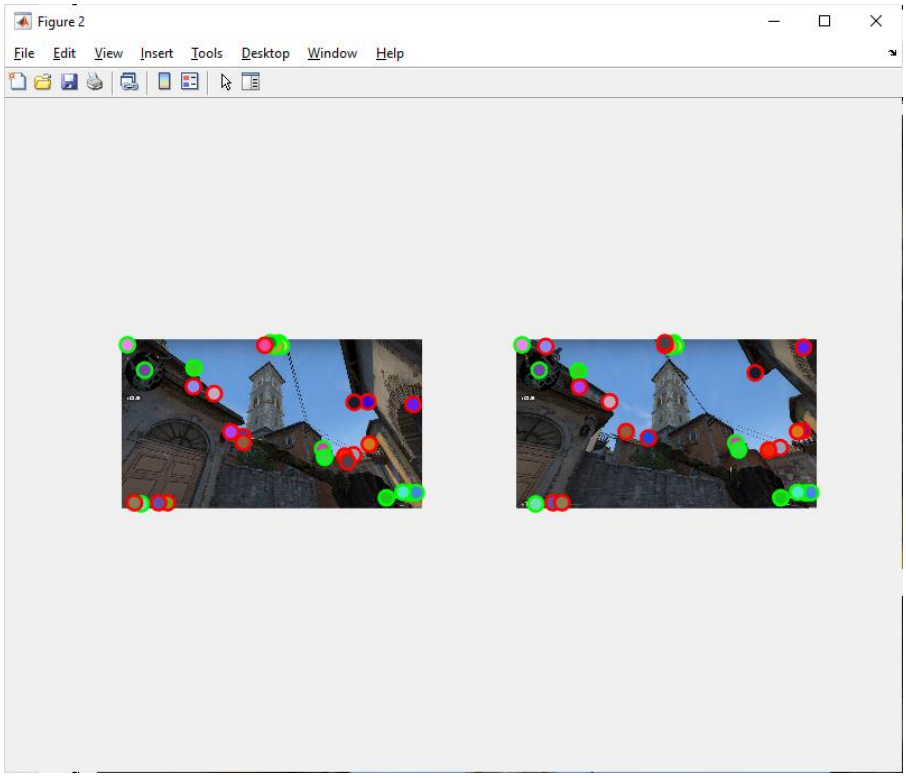
(1) Running the collect_ground_truth_corr() file. The input of this file will be the above two images.

(2) After running the file, both the images will be displayed, and we have to mark the corresponded points on

both the images.
(3) After marking the corresponded points we need to click on the negative-ordinate above or left to stop it.
(4) Then the script will be stopped and all the points will be saved in an output file.
(5) We will be using this output file as ground-truth value while running the evaluate_correspondence.m file to generate the evaluation image.

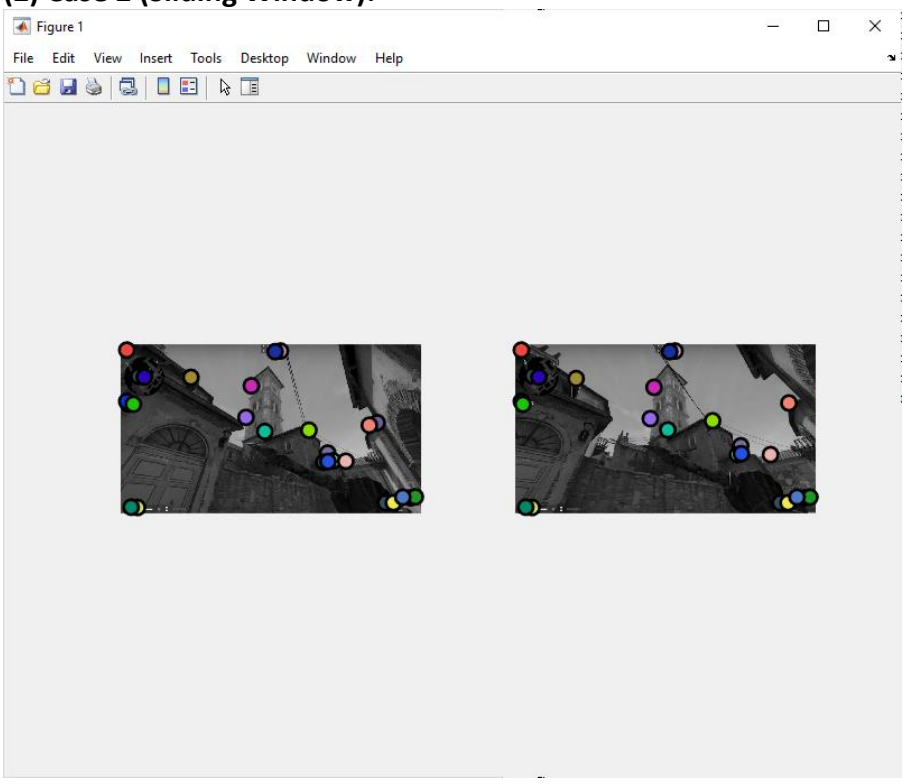
(1) Case 1 (Connected Components):

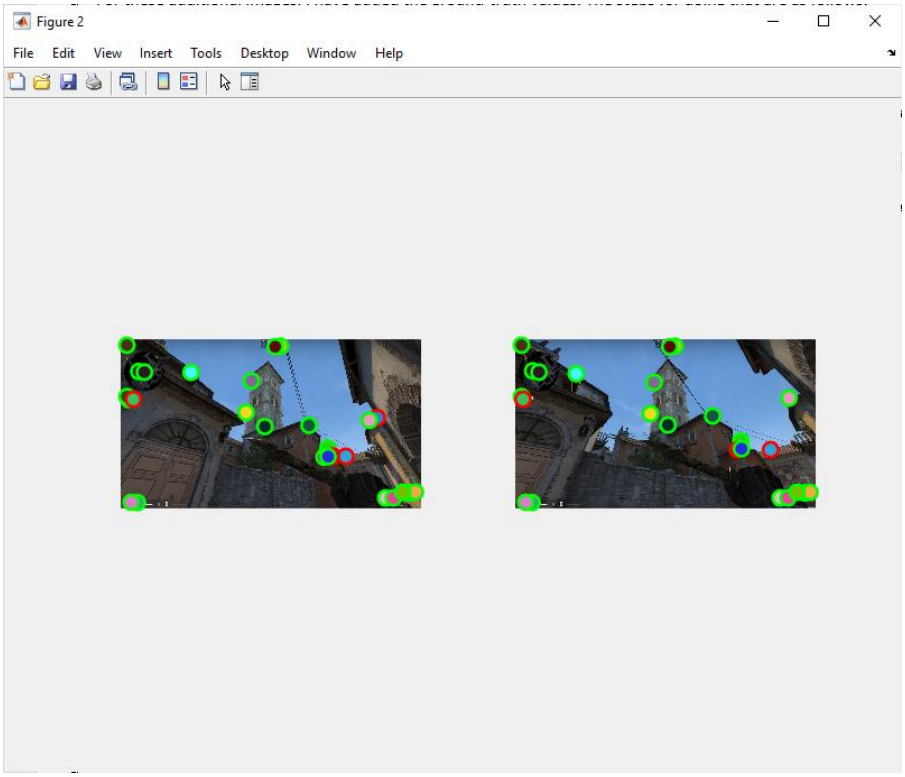




```
>> proj4
Saving visualization to vis.jpg
( 1020,   44) to ( 1020,   44) g.t. point   10 px. Match error    2 px.  correct
( 1692, 1014) to ( 1692, 1014) g.t. point    5 px. Match error   12 px.  correct
(  986,   48) to (  986,   48) g.t. point   13 px. Match error    2 px.  correct
(  920,  558) to (  974,  548) g.t. point    4 px. Match error    2 px.  correct
( 1850,  982) to ( 1850,  982) g.t. point   41 px. Match error    6 px.  correct
(  834,  266) to (  886,  276) g.t. point   24 px. Match error    5 px.  correct
(   38, 1046) to (   38, 1046) g.t. point    4 px. Match error    3 px.  correct
( 1486,  404) to ( 1528,  214) g.t. point  154 px. Match error  129 px.  incorrect
( 1638,  502) to ( 1444,  700) g.t. point    3 px. Match error  510 px.  incorrect
(  114,  206) to (  110,  202) g.t. point    8 px. Match error    7 px.  correct
(   66, 1042) to (   66, 1042) g.t. point   25 px. Match error    3 px.  correct
( 1882,  976) to ( 1882,  978) g.t. point    9 px. Match error    4 px.  correct
(  698,  600) to (  400,  318) g.t. point  169 px. Match error  465 px.  incorrect
(  294, 1044) to (  292, 1044) g.t. point  253 px. Match error    5 px.  incorrect
(  460,  304) to (  190,   46) g.t. point  109 px. Match error  347 px.  incorrect
( 1322,  696) to ( 1440,  650) g.t. point   27 px. Match error   19 px.  correct
( 1226,  574) to ( 1320,  530) g.t. point   45 px. Match error   33 px.  incorrect
(  800,  468) to (  862,  476) g.t. point    2 px. Match error    3 px.  correct
(  448,  212) to (  388,  220) g.t. point   17 px. Match error   10 px.  correct
( 1284,  700) to (  598,  406) g.t. point   55 px. Match error  844 px.  incorrect
( 1202,  550) to ( 1260,  490) g.t. point   12 px. Match error    7 px.  correct
( 1298,  754) to ( 1420,  706) g.t. point    2 px. Match error    3 px.  correct
( 1802,  978) to ( 1800,  978) g.t. point    3 px. Match error    5 px.  correct
16 total good matches, 7 total bad matches
Saving visualization to eval.jpg
```

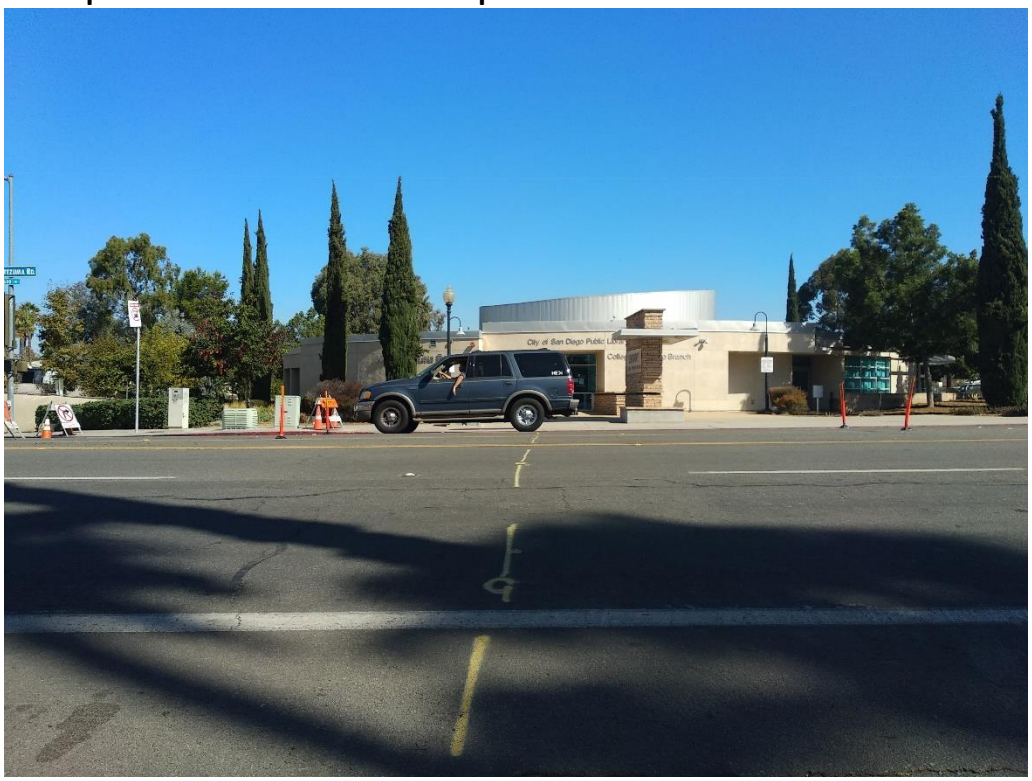
(2) Case 2 (Sliding Window):



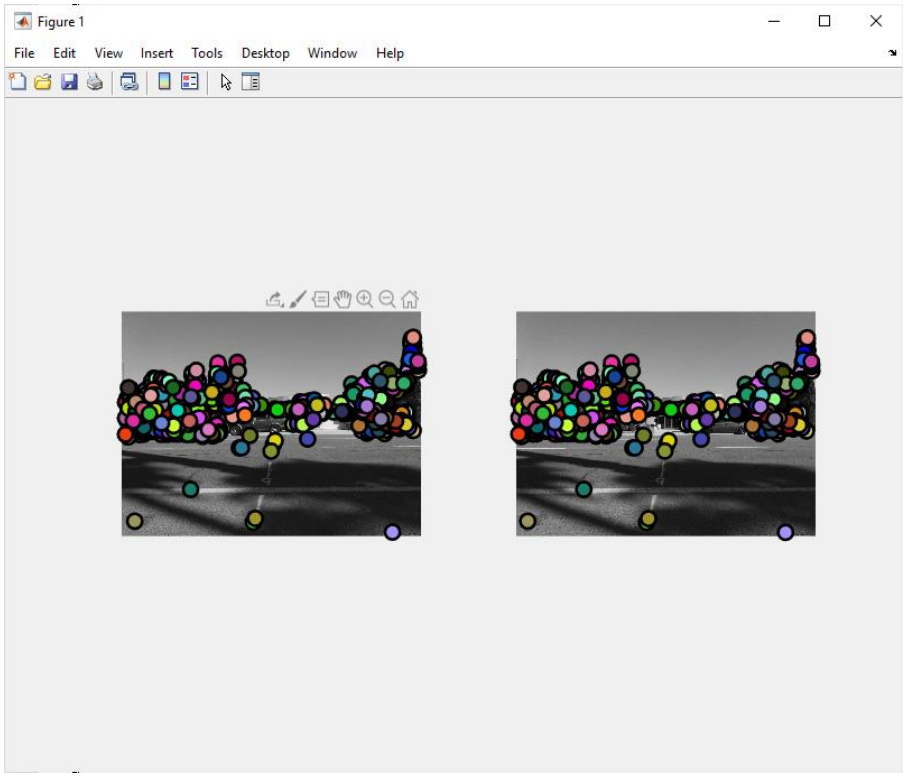


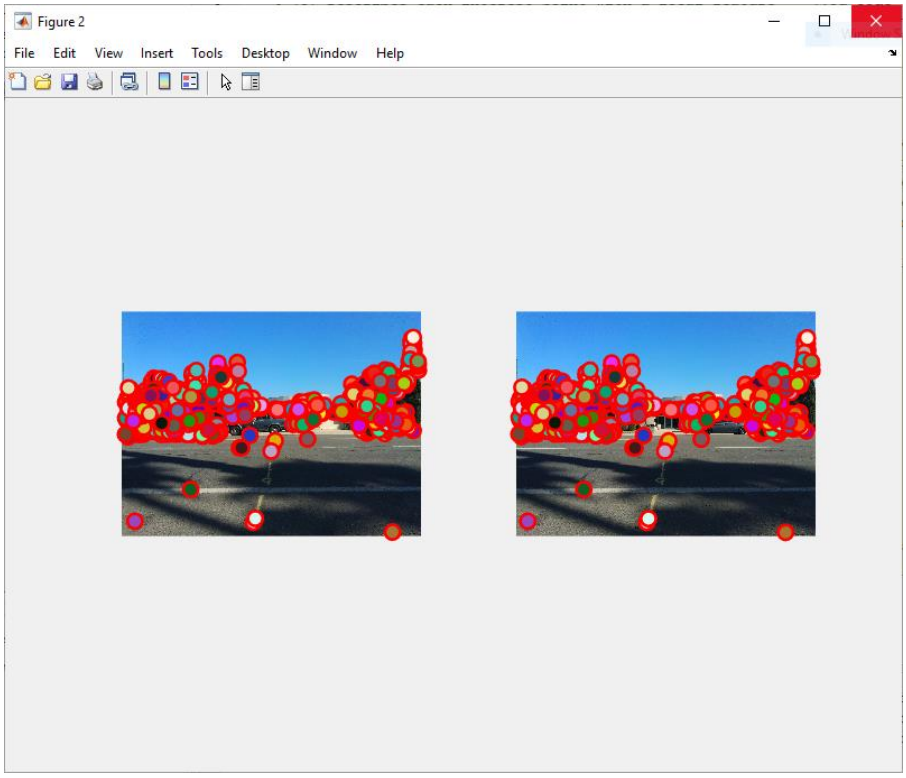

```
>> proj4
Saving visualization to vis.jpg
(   50,  384) to (   50,  384) g.t. point   23 px. Match error    7 px.  correct
(   40,  366) to (   40,  366) g.t. point    3 px. Match error    7 px.  correct
( 1694, 1014) to ( 1694, 1014) g.t. point    4 px. Match error   12 px.  correct
( 1022,   44) to ( 1022,   44) g.t. point    9 px. Match error    2 px.  correct
(  988,   48) to (  988,   48) g.t. point   15 px. Match error    2 px.  correct
(  922,  558) to (  976,  548) g.t. point    3 px. Match error    2 px.  correct
(  836,  266) to (  888,  276) g.t. point   26 px. Match error    5 px.  correct
(  108, 1044) to (  108, 1044) g.t. point   67 px. Match error    3 px.  correct
(  116,  206) to (  112,  202) g.t. point    7 px. Match error    7 px.  correct
( 1640,  502) to ( 1446,  700) g.t. point    1 px. Match error  510 px.  incorrect
(   40,   38) to (   40,   38) g.t. point    9 px. Match error    9 px.  correct
( 1744, 1014) to ( 1744, 1014) g.t. point   11 px. Match error    7 px.  correct
( 1852,  982) to ( 1852,  982) g.t. point   39 px. Match error    6 px.  correct
( 1324,  696) to ( 1442,  650) g.t. point   26 px. Match error   19 px.  correct
( 1884,  976) to ( 1884,  978) g.t. point    7 px. Match error    4 px.  correct
( 1300,  754) to ( 1422,  706) g.t. point    1 px. Match error    3 px.  correct
( 1590,  518) to ( 1750,  376) g.t. point    3 px. Match error    2 px.  correct
( 1804,  978) to ( 1802,  978) g.t. point    5 px. Match error    5 px.  correct
(  152,  210) to (  152,  210) g.t. point   35 px. Match error    6 px.  correct
(  802,  468) to (  864,  476) g.t. point    3 px. Match error    3 px.  correct
( 1358,  746) to ( 1422,  706) g.t. point    7 px. Match error   68 px.  incorrect
( 1440,  748) to ( 1632,  706) g.t. point   44 px. Match error   48 px.  incorrect
(   68, 1042) to (   68, 1042) g.t. point   27 px. Match error    3 px.  correct
(  450,  212) to (  390,  220) g.t. point   18 px. Match error   10 px.  correct
( 1204,  550) to ( 1262,  490) g.t. point   13 px. Match error    7 px.  correct
(   82,  384) to (   50,  384) g.t. point   17 px. Match error   26 px.  incorrect
( 1326,  750) to ( 1446,  700) g.t. point    7 px. Match error    5 px.  correct
23 total good matches, 4 total bad matches
Saving visualization to eval.jpg
```

The input for some additional examples are as follows:



The output for the same examples are as follows:





Conclusion:

(1) Notre Dame image pair: For case 1, 27 matches were found in total, of which 21 were correct, hinting at an accuracy of about 77% for this one image pair with the matching NNDR ratio threshold at 0.7 and for case 2, 25 matches were found in total, of which 18 were corrected.

(2) CSGO image pair: I have made the ground-truth values and generated the viz.jpg and eval.jpg. For case 1,

23 matches were found in total, of which 16 were corrected, hinting at an accuracy of about 70% for this one image pair with the matching NNDR ratio threshold at 0.7 and for case 2, 27 matches were found in total, of which 23 were corrected.

For image pairs, depending on the set-up of the images, the algorithm can have good or bad performance. As mentioned above, image pairs with large scale or orientation differences are harder to match for the algorithm since it doesn't take these parameters into account. In contrast, image pairs that are similar in scale and orientation are easier to match.