Khulna University of Engineering & Technology

Khulna-9203

-: ASSIGNMENT REPORT :-

Course Code : CSE 6239
Course Title : Computer Vision

-Submitted by-

Name : Md. Tahmid Hasan
ID : 1907565
Semester : July-2019
Dept. of CSE
KUET

-Submitted To-

Dr. Sk. Mohammad Masudul Ahsan
Professor
Dept. of CSE
KUET

Submission Date: 16 Nov 2019

I have used "Python 3" for coding language. I used some built-in library: **cv2** (Open CV), **numpy**, **math**, **pyplot** from **matplotlib**. In most cases, I wrote my own function to do my work.

Image input and preprocessing:

1. Take color image as input
2. Resize the image (400×600 pixels)
3. Convert to grayscale image

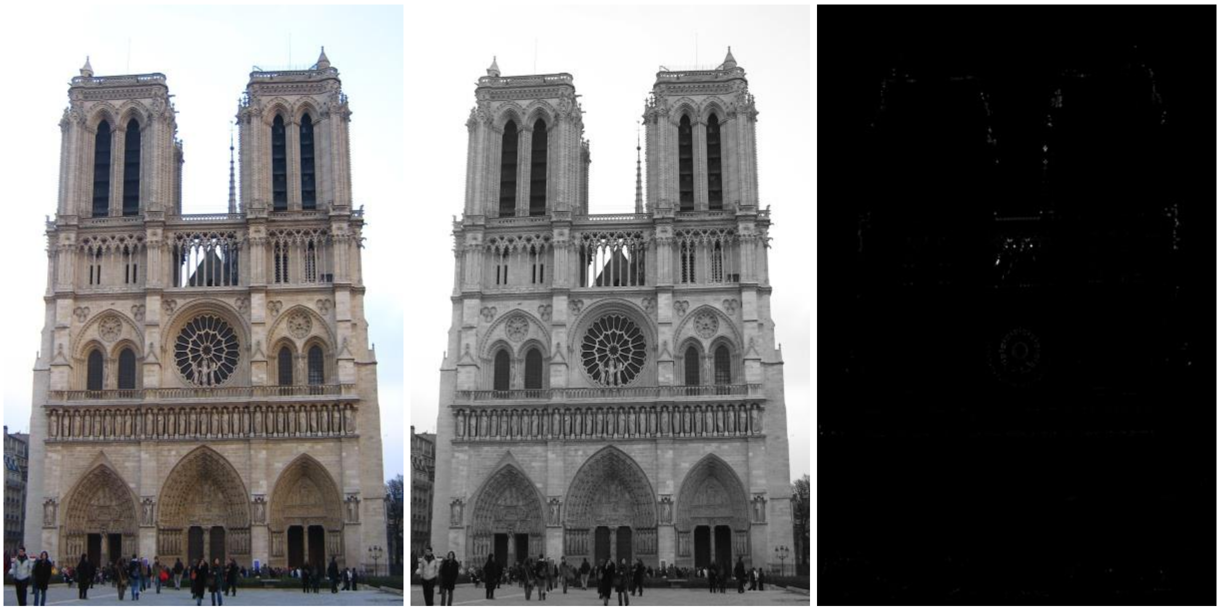
1. Find corners using Harris corner detector

To find corners using Harris corner detector, I followed these steps:

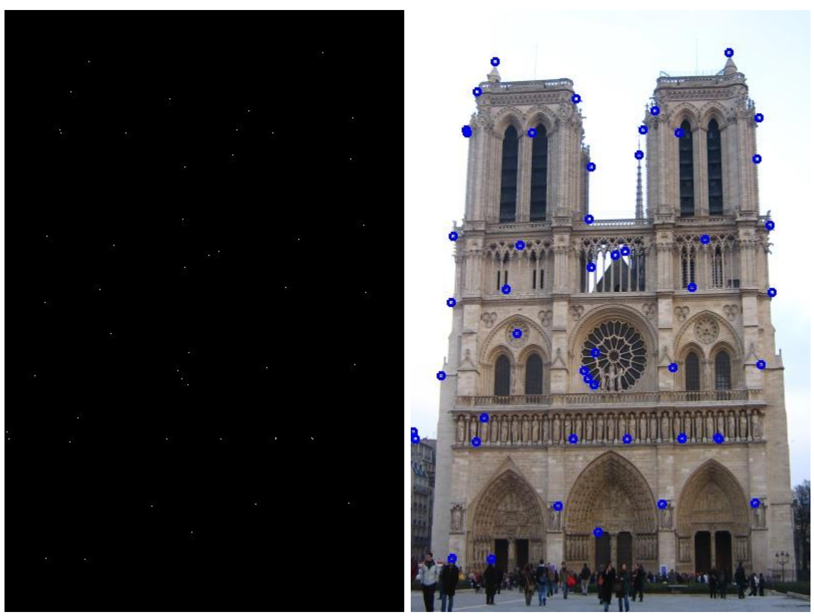
a. Take grayscale image as input
b. Find x and y derivative (d_x, d_y) of the image:
   Convolve image $f(x,y)$ with Sobel 3×3 kernel to get d_x and d_y.
   $d_x = f(x,y) * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ and $d_y = f(x,y) * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
c. Calculate ${d_x}^2$, ${d_y}^2$ and $d_x d_y$ for M:
   $M = \begin{bmatrix} {d_x}^2 & d_x d_y \\ d_x d_y & {d_y}^2 \end{bmatrix}$ where, ${d_x}^2 = d_x d_x$ and ${d_y}^2 = d_y d_y$
d. Apply Gaussian on M:
   Convolve with Gaussian (σ = 0.75) kernel of size 6σ+1
   $M = \sum_{x,y} g(x,y) * \begin{bmatrix} {d_x}^2 & d_x d_y \\ d_x d_y & {d_y}^2 \end{bmatrix} = \begin{bmatrix} {S_x}^2 & S_x S_y \\ S_x S_y & {S_y}^2 \end{bmatrix}$
e. Calculate determinant and trace:
   Determinant, $det = ({S_x}^2 \times {S_y}^2 - (S_x S_y)^2)$ and Trace, $trace = {S_x}^2 + {S_y}^2$
f. Calculate response:
   Response, $R = det - \kappa(trace)^2$ Here, κ is a constant. Suitable value for $\kappa = 0.4$ to 0.6
g. Apply threshold on R:
   High response means strong corner. I used threshold = 100000000.
   R > threshold = corner
h. Apply non-maximum suppression:
   I used an Adaptive Non-Maximum Suppression (ANMS) technique to get a limited number of final key-points which are strongest in their neighbor. Set a patch size 5×5 and choose the highest response value (key-point) from patch. If total number of key-points exceeds limit (I set, limit = 50), increase patch size by 4 and repeat the process.
i. Mark key-points:
   Finally mark the key-points in main image and store the key-point coordinates as list. This list will reduce computation time in next stages.

Input & Output:

For input image 1:



(A) (B) (C)



(D) (E)

Figure 1.1: (A) Resized input image; (B) Grayscale image; (C) Response values over threshold (threshold = 100000000); (D) Chosen points after Adaptive Non-Maximum Suppression applied (50 points have chosen); (E) Marked chosen key-points.

For input image 2:

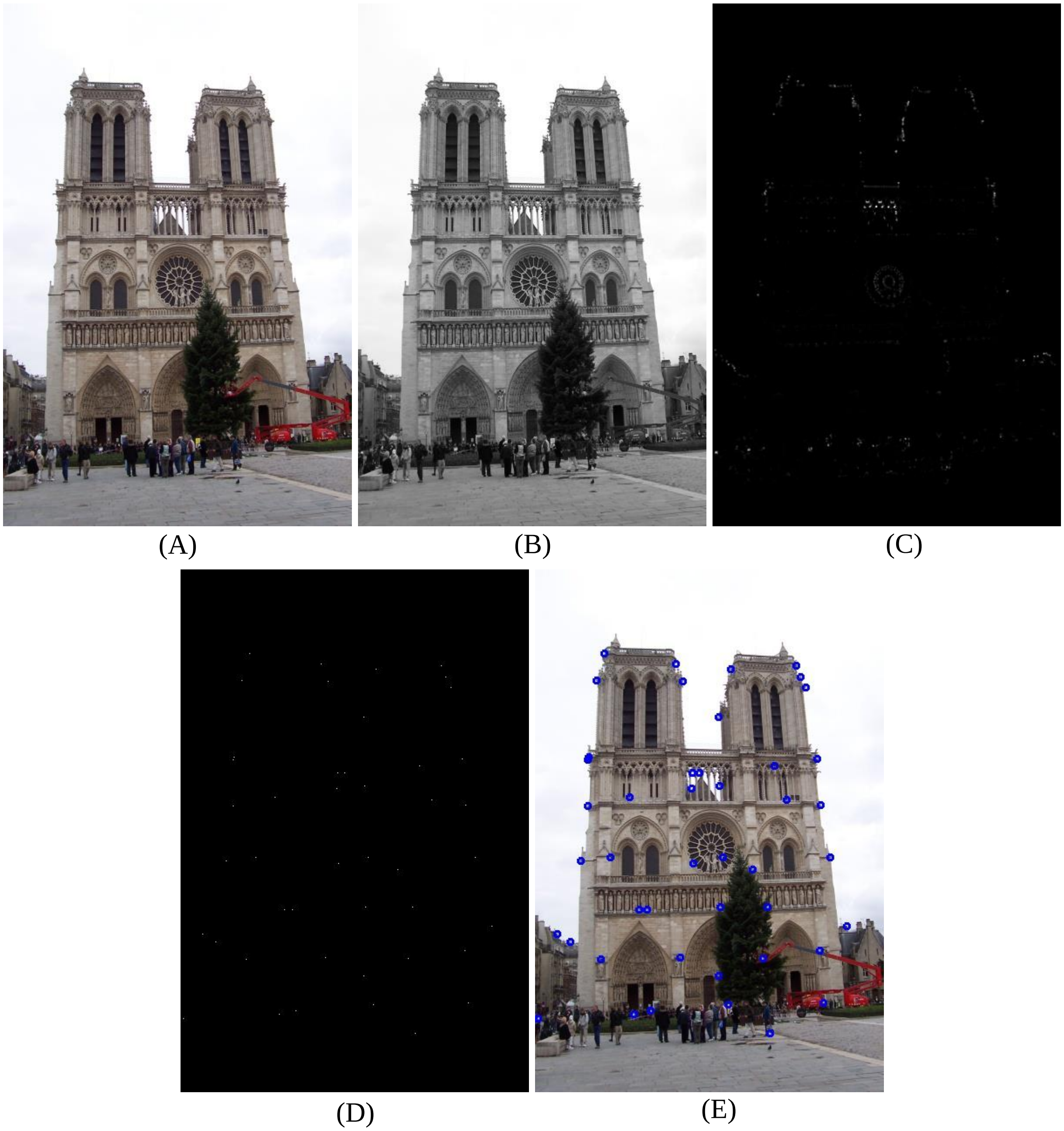


Figure 1.2: (A) Resized input image; (B) Grayscale image; (C) Response values over threshold (threshold = 100000000); (D) Chosen points after Adaptive Non-Maximum Suppression applied (45 points have chosen); (E) Marked chosen key-points.

2. Use SIFT like descriptor

To describe a keypoint so that it can use for matching, I used a SIFT like descriptor. I didn't consider scale factor. To do this, I followed these steps:

a. Take d_x, d_y and key-point list as input (from 1)

b. Calculate magnitude and orientation from d_x and d_y:

Magnitude = $\sqrt{{d_x}^2 + {d_y}^2}$ and Orientation = $\tan^{-1}(d_y/d_x)$

Orientations are in degrees ranges from 0 to 360.

c. Create patch:
Create a 16×16 (256 pixels) patch around each key-point

d. Divide patch into portion:
Divide each patch into 4×4 (=16) portion. So, each portion has 4×4=16 pixels.

e. Create container:
Create 8 containers for each portion. These 8 containers represent 8 degrees (0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°). [360° = 0°]

f. Value distribution:
According to orientation of each pixel in a portion, distribute the magnitude values among containers. Here, distribution is done by linear distribution. For example:
If, magnitude = 150 and orientation = 105° then it lies between 90° and 135° (90° < 105° < 135°). So, container of 90° will get $\frac{135-105}{135-90} \times 150 = 100$ and container of 135° will get 150-100 = 50.

g. Descriptor:
For a key-point, we will get 16×8=128 containers. Store these values (descriptor value) with coordinates.

h. Normalization:
Perform Euclidean normalization on descriptor values (without coordinates)

Descriptor value, $\boldsymbol{d} = \boldsymbol{d} / \sqrt{\sum_{i=1}^{128} {d_i}^2}$ where, $\boldsymbol{d} = (d_1, d_2, \dots d_{128})$

i. Create bar chart:
Create a bar chart from descriptor values. X-axis represents orientation and Y-axis represents magnitude (descriptor values).

Output:

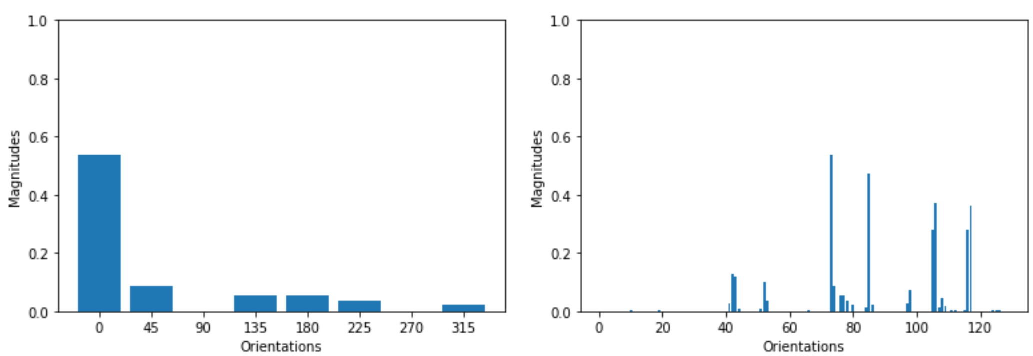


Figure 2: Bar chart of a portion in a patch (Left); and full patch (Right) of a key-point

3. Find matches

I have used three different technique to match the key-points from similar 2 images. Matching steps are below:

a. Take 2 key-point descriptor lists kd1 and kd2 as input (from 2)
b. Compare each point in kd1 with each point in kd2 and calculate matching score
c. Apply threshold on matching score to pick best match
d. Draw line between best match points

The three techniques to calculate matching score are described here:

Euclidean distance:

a. Measure distance:

   $dist = \sqrt{\sum_{i=1}^{128}\left(d_{1_i} - d_{2_i}\right)^2}$ where, d_1 from kd1 and d_2 from kd2

b. Store minimum distance (matching score):

   $minDist = \min\,(minDist, dist)$

   Using this, find the best possible match

c. Apply threshold:

   After finding best possible match, apply threshold to choose those pair as best match or not. I use 0.6 as threshold. Matching score less than threshold have chosen.

Cosine similarity:

a. Measure similarity ($\cos\theta$):

   $sim = \cos\theta = \frac{\boldsymbol{d_1}.\boldsymbol{d_2}}{\|\boldsymbol{d_1}\|\|\boldsymbol{d_2}\|}$ where, $\boldsymbol{d_1}$ from kd1 and $\boldsymbol{d_2}$ from kd2

   and $\|\boldsymbol{d}\| = \sqrt{\boldsymbol{d}.\boldsymbol{d}}$

b. Store maximum similarity (matching score):

   $maxSim = \max\,(maxSim, sim)$

   Using this, find the best possible match

c. Apply threshold:

   After finding best possible match, apply threshold to choose those pair as best match or not. I use 0.75 as threshold. Matching score greater than threshold have chosen.

Nearest Neighbor Distance Ratio (NNDR):

a. Find best and second-best possible match using Euclidean distance
b. Measure ratio (matching score):

   $R = \frac{NND1}{NND2}$ [Nearest Neighbor Distance (NND) = Euclidean distance]

c. Apply threshold:

   After calculating ratio, apply threshold to choose the best possible match as final best match or not. I use 0.85 as threshold. Matching score less than threshold are good match (Yellow) and others are poor match (Red).

Output:

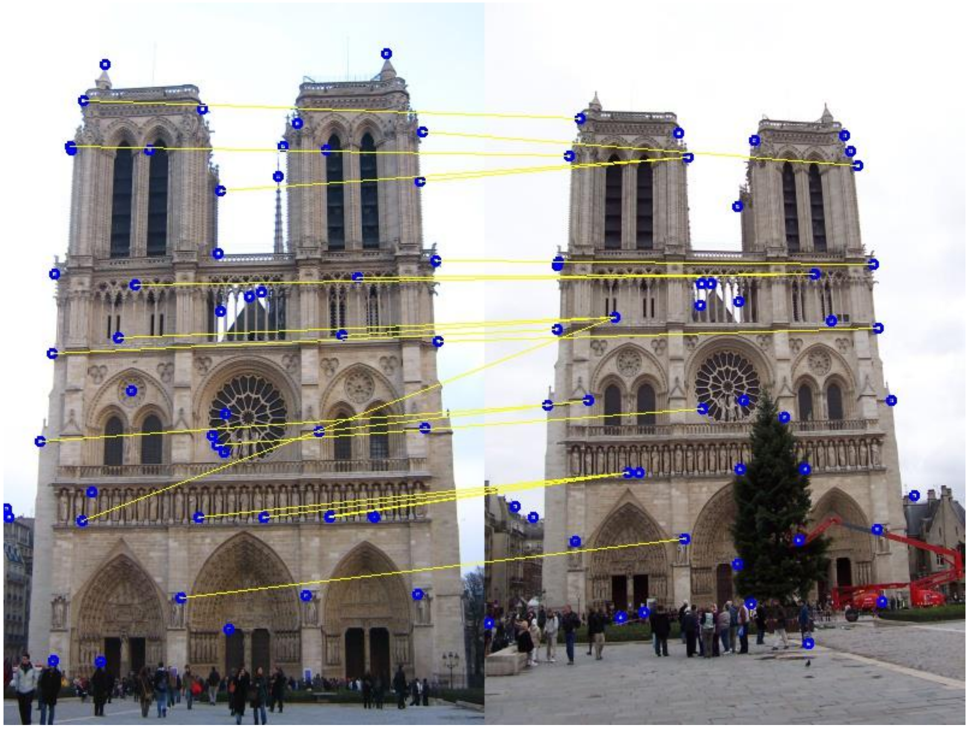


Figure 3.1: Matching using Euclidean distance. Under threshold (<0.6) 21 pair matched (yellow line).

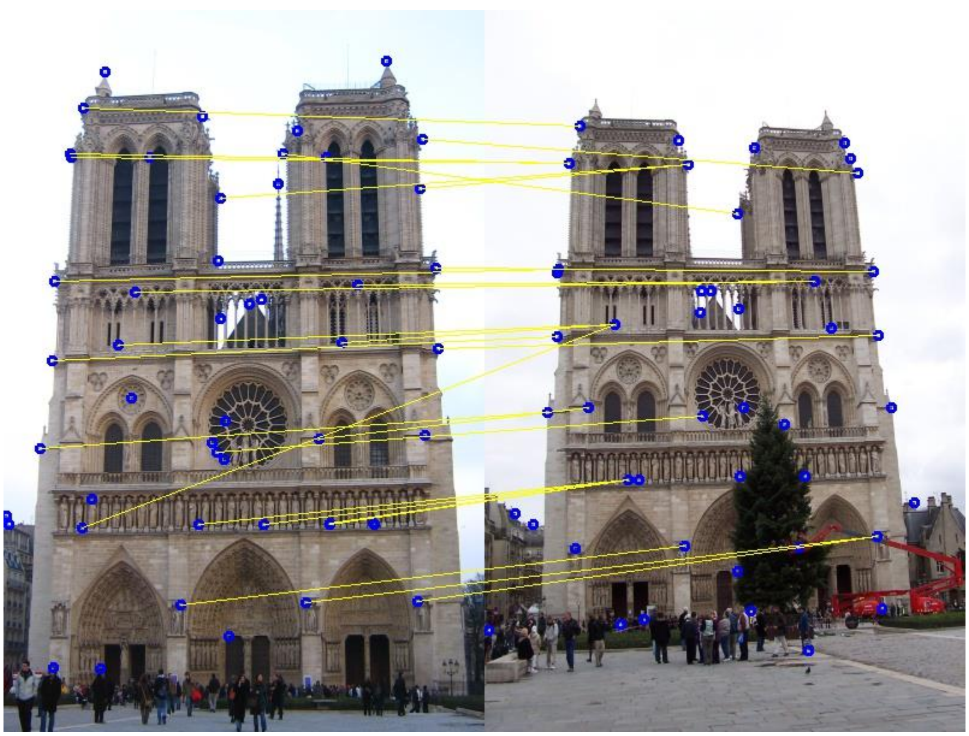


Figure 3.2: Matching using cosine similarity. Above threshold (>0.75) 26 pair matched (yellow line).

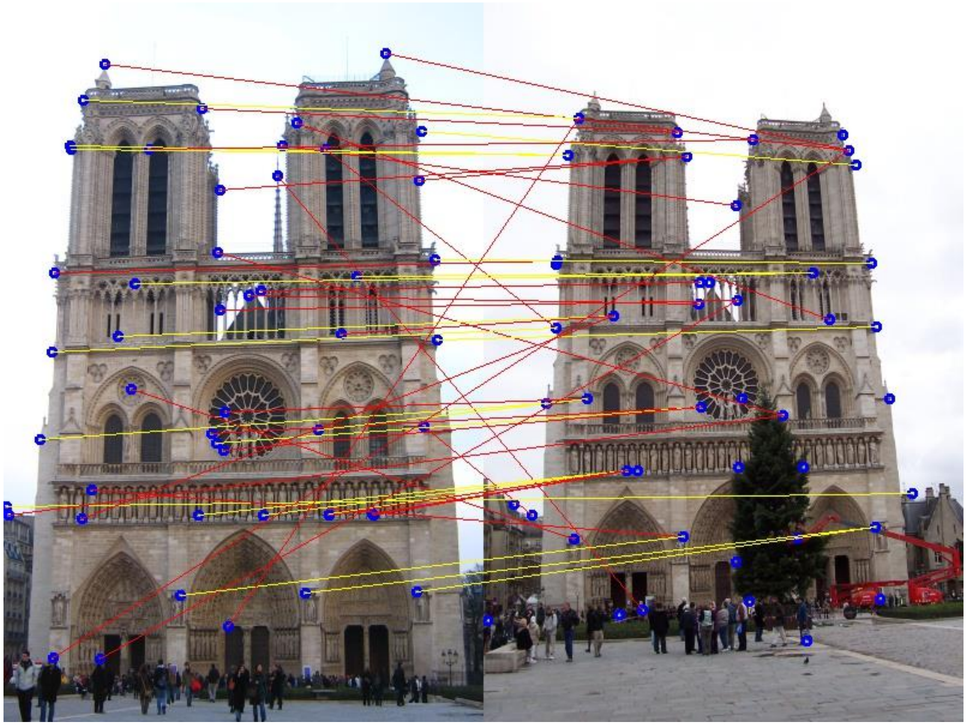


Figure 3.3: Matching using Nearest Neighbor Distance Ratio (NNDR). Under threshold (<0.85) 20 pair marked as good match (yellow line) and others are marked as poor match (red line).

4. Manually/Autometicaly choose at least 4 matching pair and represent as Ah=0 form

I chose 4 matching pair manually.

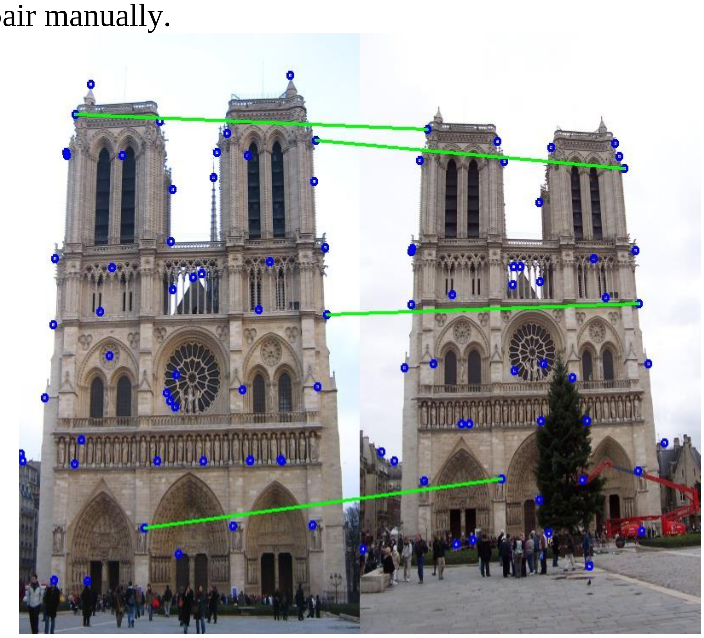


Figure 4: Marked chosen matching pairs

Chosen points:

```
(x, y): ( 66 ,  81 )   (xp, yp): ( 79 ,  96 )
(x, y): ( 348 ,  107 )  (xp, yp): ( 310 ,  135 )
(x, y): ( 361 ,  281 )  (xp, yp): ( 327 ,  270 )
(x, y): ( 147 ,  494 )  (xp, yp): ( 166 ,  445 )
```

Then matrix A had formed as:

$$A = \begin{bmatrix} -66 & -81 & -1 & 0 & 0 & 0 & 5214 & 6399 & 79 \\ 0 & 0 & 0 & -66 & -81 & -1 & 6336 & 7776 & 96 \\ -348 & -107 & -1 & 0 & 0 & 0 & 107880 & 33170 & 310 \\ 0 & 0 & 0 & -348 & -107 & -1 & 46980 & 14445 & 135 \\ -361 & -281 & -1 & 0 & 0 & 0 & 118047 & 91887 & 327 \\ 0 & 0 & 0 & -361 & -281 & -1 & 97470 & 75870 & 270 \\ -147 & -494 & -1 & 0 & 0 & 0 & 24402 & 82004 & 166 \\ 0 & 0 & 0 & -147 & -494 & -1 & 65415 & 219830 & 445 \end{bmatrix}$$

Now represent as Ah=0 form:

$$\begin{bmatrix} -66 & -81 & -1 & 0 & 0 & 0 & 5214 & 6399 & 79 \\ 0 & 0 & 0 & -66 & -81 & -1 & 6336 & 7776 & 96 \\ -348 & -107 & -1 & 0 & 0 & 0 & 107880 & 33170 & 310 \\ 0 & 0 & 0 & -348 & -107 & -1 & 46980 & 14445 & 135 \\ -361 & -281 & -1 & 0 & 0 & 0 & 118047 & 91887 & 327 \\ 0 & 0 & 0 & -361 & -281 & -1 & 97470 & 75870 & 270 \\ -147 & -494 & -1 & 0 & 0 & 0 & 24402 & 82004 & 166 \\ 0 & 0 & 0 & -147 & -494 & -1 & 65415 & 219830 & 445 \end{bmatrix} \times \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

5. Find homography matrix H by solving the equation Ah=0

I solved the equation Ah=0 by Single Value Decomposition (SVD) and found the values of h.

a. Use SVD function:
   Calculate U, S, V^T = svd (A). Here, A is a 8×9 matrix. So, I got a 8×8 orthonormal matrix U, 1×9 eigen vector S and 9×9 orthonormal matrix V^T.
b. Get transpose:
   Transpose V^T matrix to get V.
c. Choose values of h
   Pick the last column values of V as h.
d. Reshape:
   Reshape the 1×9 vector to 3×3 matrix.
e. Divide whole matrix by 9^{th} value.

Now the homography matrix H is:

```
    [[ 9.38204871e-01  3.66398425e-02  1.56980824e+01]
H = [ 1.08284678e-01  8.56218677e-01  2.14285286e+01]
    [ 3.44889336e-04 -3.29460627e-05  1.00000000e+00]]
```

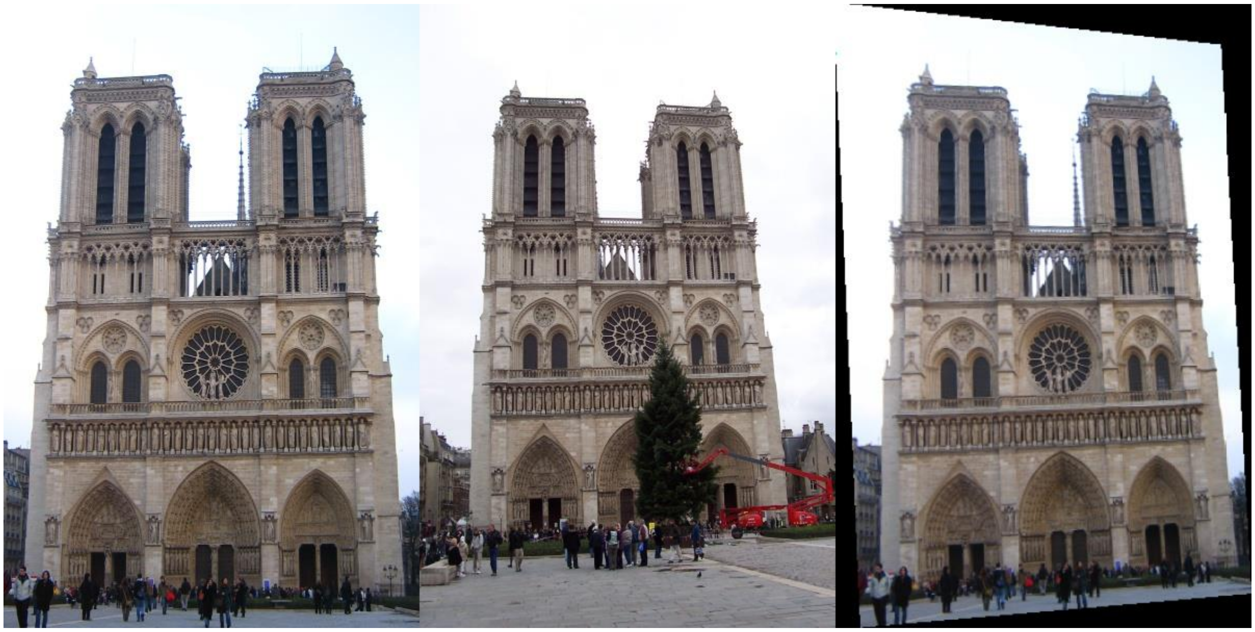
6. Apply H to transform a image using backward warping. Also show the differences

I solved the equation Ah=0 by Single Value Decomposition (SVD) and found the values of h.

a. Take first image, second image and H as input
b. Apply H on 4 corners of 1st image to get their transformed position.
   $\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ then $x' = x'/w'$ and $y' = y'/w'$
   Here, (x, y) for original position and (x', y') for new transformed position
   Calculate size of new matrix from the result values and create a new matrix.
c. Calculate the inverse matrix (H^{-1}) of H
d. Backward warping
   Apply H^{-1} on new matrix to get the position on 1st image from where it will pick the values.
   $\begin{bmatrix} x'' \\ y'' \\ w'' \end{bmatrix} = H^{-1} \times \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ then $x'' = x''/w''$ and $y'' = y''/w''$
   Here, (x'', y'') are calculated position on 1st image and (x', y') are new transformed position. Ignore (x'', y'') if they are out of bound in 1st image.
e. Bilinear interpolation:
   Position (x'', y'') may represent any position within $(x, y), (x + 1, y), (x, y + 1), (x + 1, y + 1)$ area. Let $a = x'' - x$ and $b = y'' - y$. So, (x'', y'') will get the value,
   $$(1 - a)(1 - b) * f(x, y) + a(1 - b) * f(x + 1, y) + (1 - a)b * f(x, y + 1) + ab * f(x + 1, y + 1)$$
   Here $f(x, y)$ returns the value of (x, y) position from 1st image.
f. Resize the transformed image same as 1st image.
g. Concate 1st, 2nd and transformed image horizontally to show. (All are in same size)
h. Find difference:
   Subtract transformed image from 2nd image.

Output:

Figure 6.1: Transformed image of 1st image



(A) (B) (C)

Figure 6.2: (A) 1st image; (B) 2nd image; (C) Transformed image

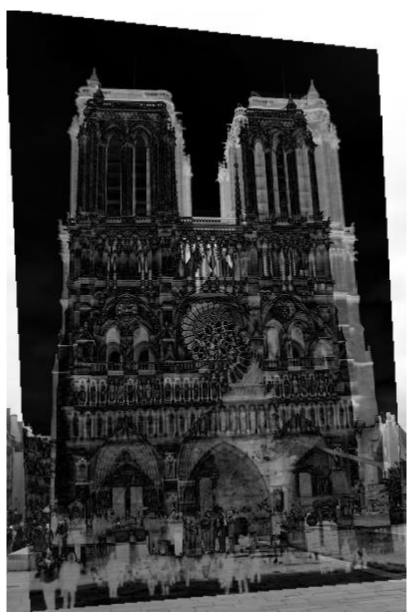


Figure 6.3: Difference between 2nd image and transformed image