ECE661: Homework 4
Fall 2018

Gopikrishnan Sasi Kumar
PUID 0030843999
gsasikum@purdue.edu

September 27, 2018

1 Introduction

The goal of the homework is to extract interest points from pairs of photos, and then automatically establish correspondences between the interest points in the two images. Interest points is in general a scale-space concept. Algorithms like SIFT and SURF are used to determine interest points. But when scale-space is not considered, corners in an image can be used as interest points. Harris corner detector is one of the most popular algorithms for corner detection. The objectives of the home work are

1. To first implement the Harris corner detector and apply it to pairs of images of the same scene taken from different viewpoints.
2. To determine the correspondences between the interest points thus found out using two similarity measures – SSD (Sum of Squared Differences) and NCC (Normalized Cross Correlation).
3. To subsequently use the SIFT operator for extracting the interest points and to establish correspondences between the points in the pairs of images.

These tasks are to be carried out for 4 pairs of images – 2 of which are provided in the homework task statement, and 2 of my own. These pairs of images shall be named Image pair 1 to 4, with 1, 2 being the former, and 3, 4 the latter.

2 Harris Corner Detector

2.1 Theory

A corner is a pixel in the vicinity of which the gray levels have significant variations in at least two different directions. The Harris corner detector is invariant to in-plane rotations of the image. The Harris corner detection works as follows.

1. Haar filters $H_x(\sigma)$ and $H_y(\sigma)$ for the required scale (σ) are generated. These are operators with half of the elements 1's and the other half -1's, with order $M \times M$, where M is the smallest even integer greater than 4σ. The filters for $\sigma = 1.2$ are as follows.

$$H_x(\sigma = 1.2) = \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{pmatrix}$$

$$H_y(\sigma = 1.2) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

2. Haar filters are convolved (or rather correlated) with the image to get the x and y derivatives (d_x and d_y) of the gaussian-blurred image at scale σ.

3. The matrix C is computed for each pixel as:

$$C = \begin{pmatrix} \sum {d_x}^2 & \sum d_x d_y \\ \sum d_x d_y & \sum {d_y}^2 \end{pmatrix} \tag{1}$$

where the summations are for a $5\sigma \times 5\sigma$ neighborhood around each pixel. This C has a full rank for genuine corners. It works based on the fact that if a particular pixel is not at a corner (it is at an edge, or at a point which is neither a corner nor an edge), then either the derivatives are zero, or the y-derivative is a constant times the x-derivative. Hence, the matrix C will not be of full rank.
If λ_1 and λ_2 are the two eigen values of C ($\lambda_1 \geq \lambda_2$), $\frac{\lambda_2}{\lambda_1} \geq 0.1$ for the pixel to be a corner.

4. Instead of carrying out the eigen decomposition of C, we compute the harris response:

$$R = \det(C) - k(trace(C))^2 \tag{2}$$

where k is the empirical constant set to 0.05, and the determinant and trace of C are computed as:

$$\det(C) = \sum {d_x}^2 \sum {d_y}^2 - \left(\sum d_x d_y\right)^2 \tag{3}$$

$$trace(C) = \sum {d_x}^2 + \sum {d_y}^2 \tag{4}$$

5. The correspondences are established between the corners determined in the 2 images based on the SSD (Sum of Squared Differences) and NCC (Normalized Cross Correlation) metrics, based on their gray levels in an $(M+1) \times (M+1)$ window, as follows.

$$SSD = \sum_i \sum_j |f_1(i,j) - f_2(i,j)|^2 \tag{5}$$

$$NCC = \frac{\sum_i \sum_j (f_1(i,j) - m_1)(f_2(i,j) - m_2)}{\sqrt{\left[\sum_i \sum_j (f_1(i,j) - m_1)^2\right]\left[\sum_i \sum_j (f_2(i,j) - m_2)^2\right]}} \tag{6}$$

where f_1 is image 1, f_2 is image 2, m_1 is the mean of f_1's gray levels within the window, and m_2 is the mean of the gray levels of f_2 in the window.

2.2 Implementation Notes

- The σ value is configurable, and the code works for multiple values of σ in a single execution, with the values input as a list.

- d_x and d_y matrices obtained by convolving with the Haar operators are multiplied element-wise using numpy to generate ${d_x}^2$, ${d_y}^2$ and $d_x d_y$ matrices.
- The sum over the $5\sigma \times 5\sigma$ neighborhood around each of the pixels is carried out by convolving (or rather correlating) with a kernel of size $5\sigma \times 5\sigma$ with all the elements equal to 1. This method simplifies implementation.
- The Harris response values R (equation 2) at each of the pixels are used to sort the pixels, and the pixels with the largest values of R are identified as corners. The number of corners identified in the image is configurable in the code, and is set as a large value, so as to enable detection of a rich set of corners.
- The size of the window used for computing SSD and NCC to determine the correspondences between images is set as a configurable value. But a window of size 21×21 is set as the default in the work. In case the corner point happens to be near the image boundaries such that a window of the required size goes beyond the boundaries, the pixels at the boundary are replicated beyond the boundaries and the window of the required size is generated.
- f_1, f_2, m_1 and m_2 for the window for each of the corner pixels in the images for computing SSD and NCC are saved into a list and used for computations, instead of accessing the big image matrices each time a new pair of corners is to be assessed for closeness.
- The SSD and NCC are computed for each of the corner-pairs (one corner from each of the images), and the pairs with the highest closeness (smallest SSD / largest NCC) give the correspondences. Each of the corners in the first image is paired with each of the corners in the second image, and the SSD and NCC metrics are computed. The pair that shows the most closeness is identified as a valid pair. All such valid pairs are sorted according to the closeness metric, and the pairs that exhibit the most closeness among the list of such pairs are concluded to be valid correspondences. The number of valid correspondences is configurable, so as to avoid cluttering of the image that shows the matching. Also, the match is displayed as a connected line between the matched corners.

3 SIFT

3.1 Theory

SIFT stands for Scale Invariant Feature Transform. It tries to find the pixels in the image (interest-points) that are significantly invariant to scale, orientation and illumination. It uses the Difference of Gaussian / DoG (which is an approximation to the Laplacian of Gaussian / LoG). The extraction of the SIFT features is carried out as follows.

1. Find all the extrema in the DoG pyramid by comparing each point in the DoG with 8 points in the immediate 3×3 neighborhood at the same scale, and the 9 points in each of the 3×3 neighborhoods at the scale levels just above and just below (26 neighbors in total). This gives the local maxima and minima in the (x, y, σ) space.
2. As σ increases from a lower octave to a higher octave, the DoG points represent increasingly coarse sampling of the original image. Hence, in order to determine the precise positions of the maxima with respect to the original image, we use the Taylor series expansion of $D(x, y, \sigma)$ – the DoG values at x, y, at scale σ. If $\vec{x_0} = (x_0, y_0, \sigma_0)^T$ is the location of an extremum found out in the first step,

$$D(\vec{x_0} + \vec{h}) \approx D(\vec{x_0}) + J^T(\vec{x_0})\vec{h} + \frac{1}{2}\vec{h}^T H(\vec{x_0})\vec{h} \tag{7}$$

where J and H are the gradient vector and Hessian respectively estimated at $\vec{x_0}$, and $\vec{h}$ is the incremental difference of the true-minimum from $\vec{x_0}$. At the true extremum, $\frac{\partial D(\vec{x})}{\partial \vec{x}} = 0$. Using this with equation 7 gives:

$$\vec{h} = -H^{-1}(\vec{x_0})J(\vec{x_0}) \tag{8}$$

3. Subsequently we weed out the weak extrema by thresholding $|D(\vec{x})|$ at the extrema (typical threshold is 0.03). We also weed out those extrema that are at an edge in the image using the same technique that we used for Harris corner detection (at the same σ at which the extremum was discovered).

4. Find out a dominant local orientation for each extremum that remains after the previous step. For this, we first find the gradient magnitude and orientation of the Gaussian smoothed image at the scale σ of the extremum over a $K \times K$ neighborhood around the extremum. Then we construct a histogram of the orientation values using 36 bins spanning the full 360° range, with the gradient magnitudes as weights for the corresponding orientations. The bin where the histogram peaks gives the dominant orientation. A parabola may be fit to the peak and its immediate neighbors for better accuracy. The descriptors are calculated with respect to this dominant local orientation to make them invariant to in-plane rotations of the image.

5. At the scale of each extremum from the previous step, the 16×16 neighborhood around the extremum is divided into 16 '4×4' cells. The magnitudes of the gradients in the 16×16 neighborhood are weighted by a Gaussian with scale half the width of the neighborhood in order to reduce the importance of the points are farther from the extremum. For each of the 16 cells, an 8-bin orientation histogram is generated for the 16 pixels with gradient orientations (with respect to the dominant local orientation) weighted by the gradient magnitudes. Stringing together these 16 8-bin histograms gives a 128-dimensional vector at each retained extremum in the DoG pyramid. This 128-element vector is normalised to a unit vector to make it invariant to illumination changes. This is the SIFT descriptor for the interest point.

3.2 Implementation notes

- The SIFT algorithm in OpenCV was used for detecting features.
- The Euclidean distance between the 128-dimensional feature descriptors were used to determine the closeness between pairs of feature points.
- The number of features to be detected is maintained as a variable that is configurable through the script header. This value is kept high so as to detect a fairly large number of feature points.
- After determining the correspondences, feature pairs are sorted based on the closeness between them (which is essentially the Euclidean distance between their descriptors). The pairs with the highest degree of closeness are displayed with a line connecting them in order to show the match. The number of pairs displayed is configurable, and limiting the number avoids the image from getting cluttered to the point of nothing being visible.

4 Results

A total of 4 pairs of images are used in the work. 2 of them are a part of the homework question (image pairs 1 and 2), and 2 are photos taken by me (image pairs 3 and 4). The results are generated for multiple scales (σ values).

4.1 Image pair 1

The input pair of images is shown in figure 1. Harris corner detection was carried out for $\sigma = [0.6, 1.2, 1.8, 2.4]$. The corners detected are shown in figures 2, 3, 4 and 5. The correspondence-matching based on SSD at different scales are shown in figures 6, 7, 8 and 9. The same based on NCC at different scales are shown in figures 10, 11, 12 and 13. It is observed that for smaller scales the algorithm is able to find smaller features in the image, which are not identified at the larger scales. Also, good correspondence-match is observed using both SSD and NCC.
The features identified using SIFT are shown in figure 14. The correspondence-matching for the same

is shown in figure 15. It is observed that SIFT gives a much better and richer set of features and the correspondences are much more robust.

Figure 1: Image Pair 1

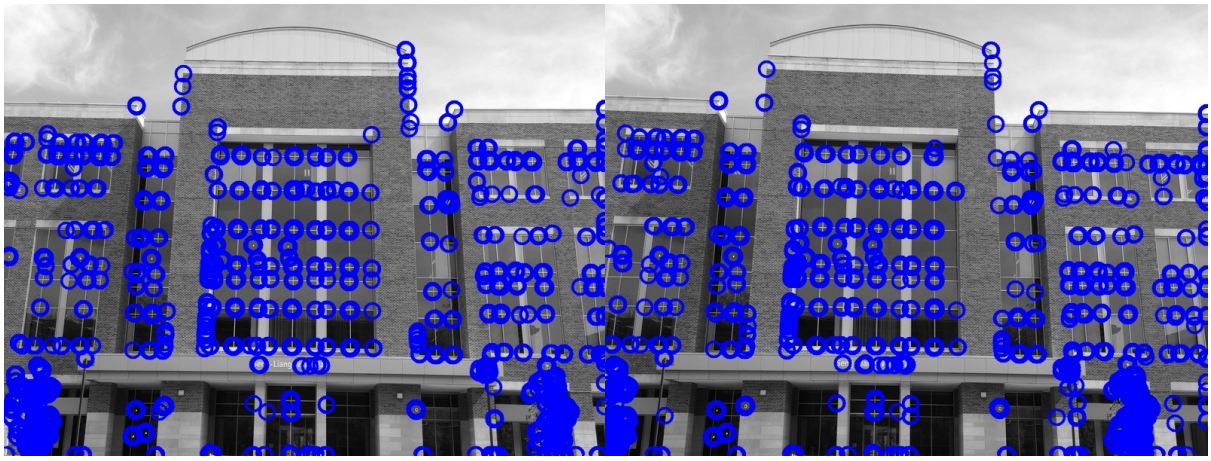

Figure 2: Harris corners identified at $\sigma = 0.6$ for Image pair 1

Figure 3: Harris corners identified at $\sigma = 1.2$ for Image pair 1

Figure 4: Harris corners identified at $\sigma = 1.8$ for Image pair 1

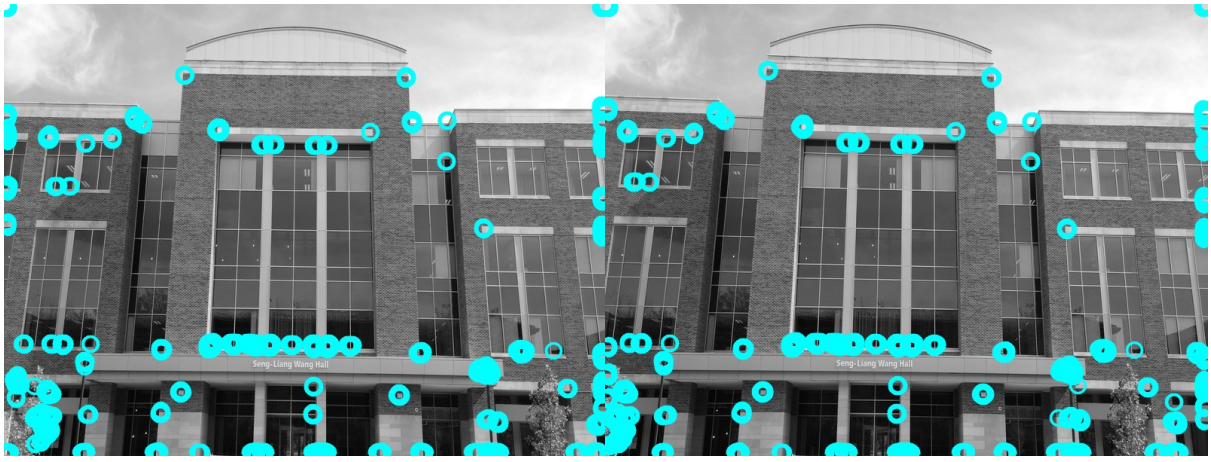

Figure 5: Harris corners identified at $\sigma = 2.4$ for Image pair 1

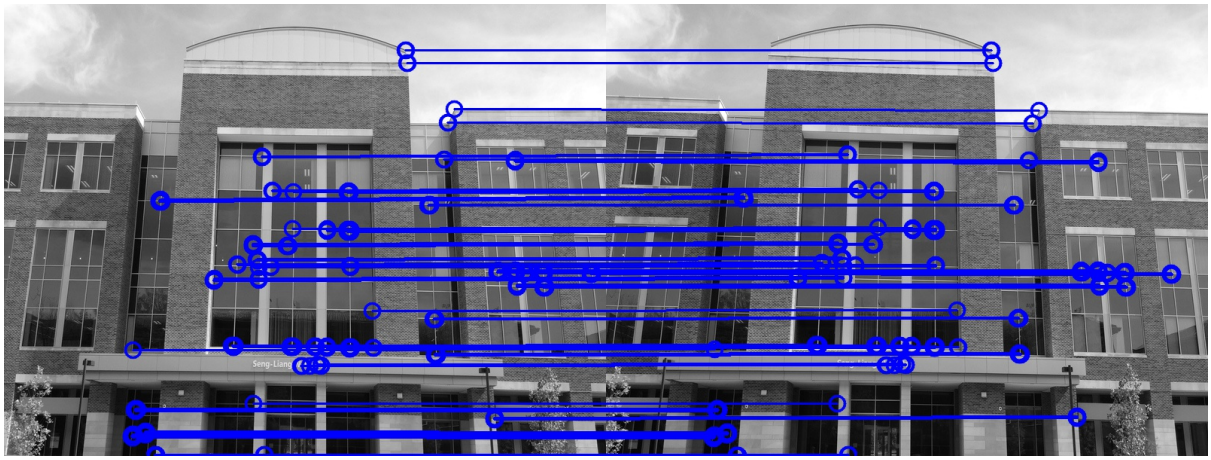

Figure 6: Correspondence-matching of corners at $\sigma = 0.6$ based on SSD for Image pair 1

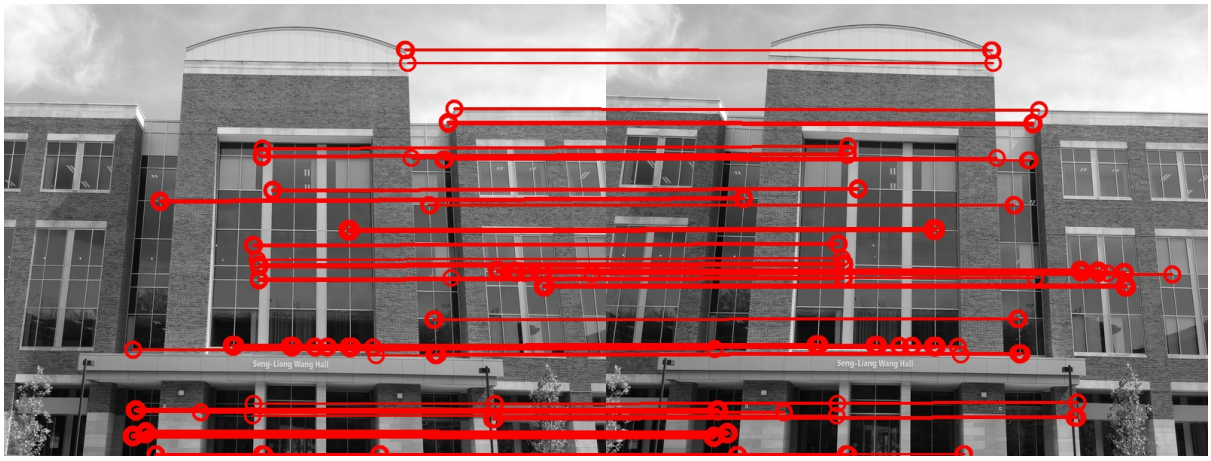

Figure 7: Correspondence-matching of corners at $\sigma = 1.2$ based on SSD for Image pair 1

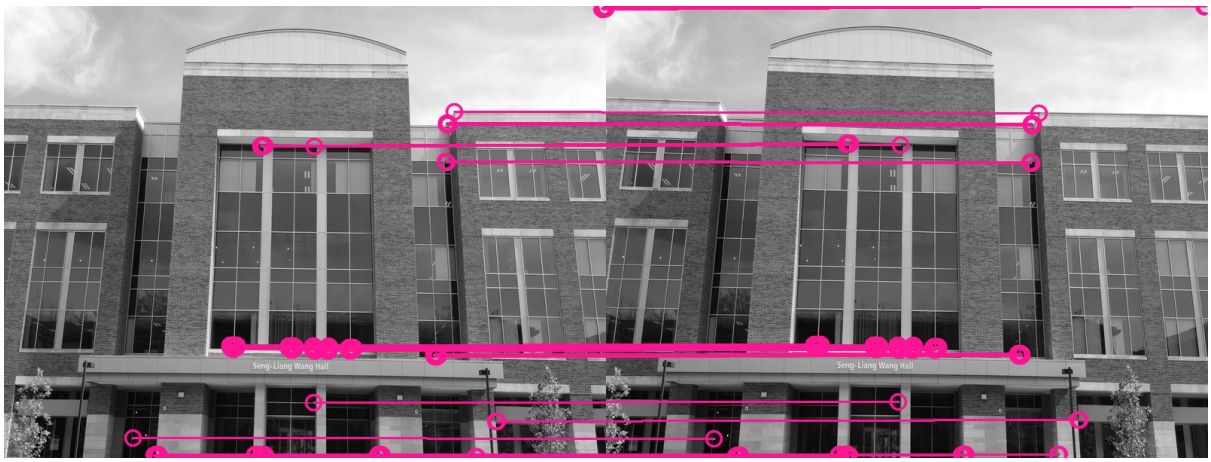

Figure 8: Correspondence-matching of corners at $\sigma = 1.8$ based on SSD for Image pair 1

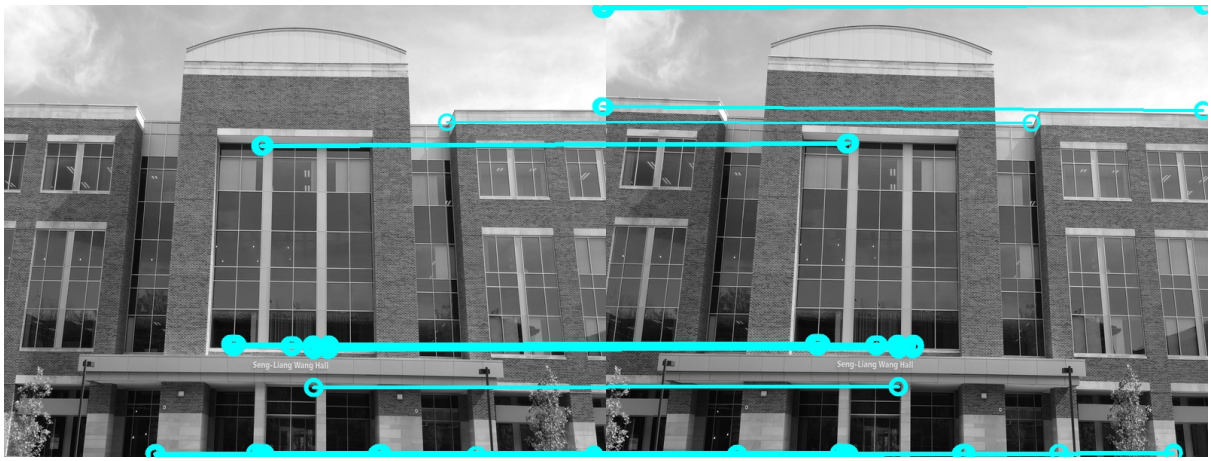

Figure 9: Correspondence-matching of corners at $\sigma = 2.4$ based on SSD for Image pair 1

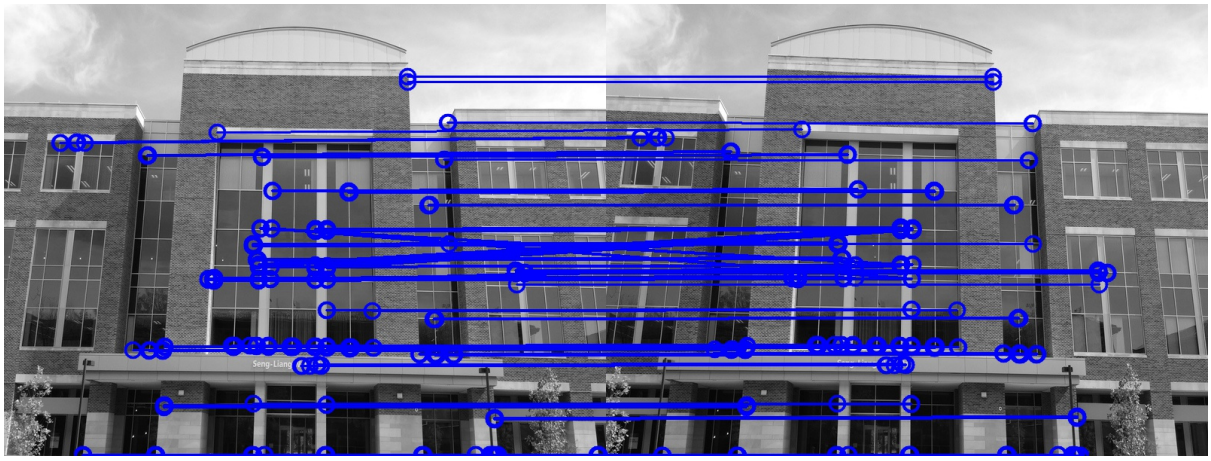

Figure 10: Correspondence-matching of corners at $\sigma = 0.6$ based on NCC for Image pair 1

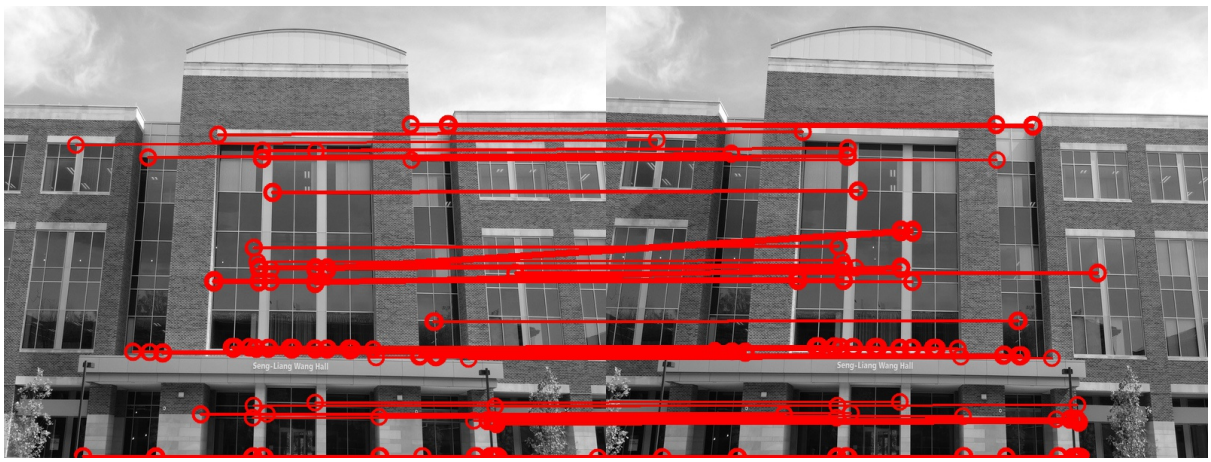

Figure 11: Correspondence-matching of corners at $\sigma = 1.2$ based on NCC for Image pair 1

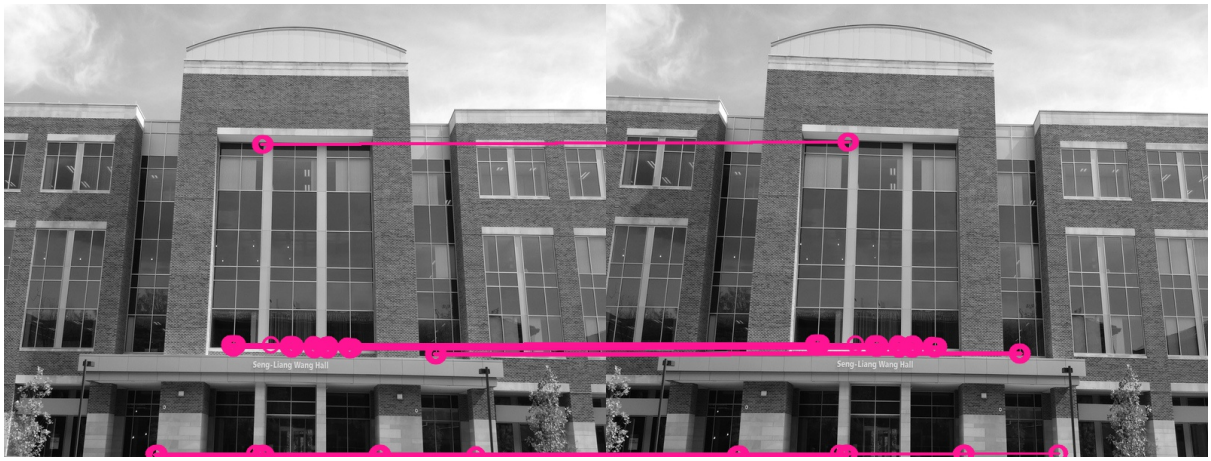

Figure 12: Correspondence-matching of corners at $\sigma = 1.8$ based on NCC for Image pair 1

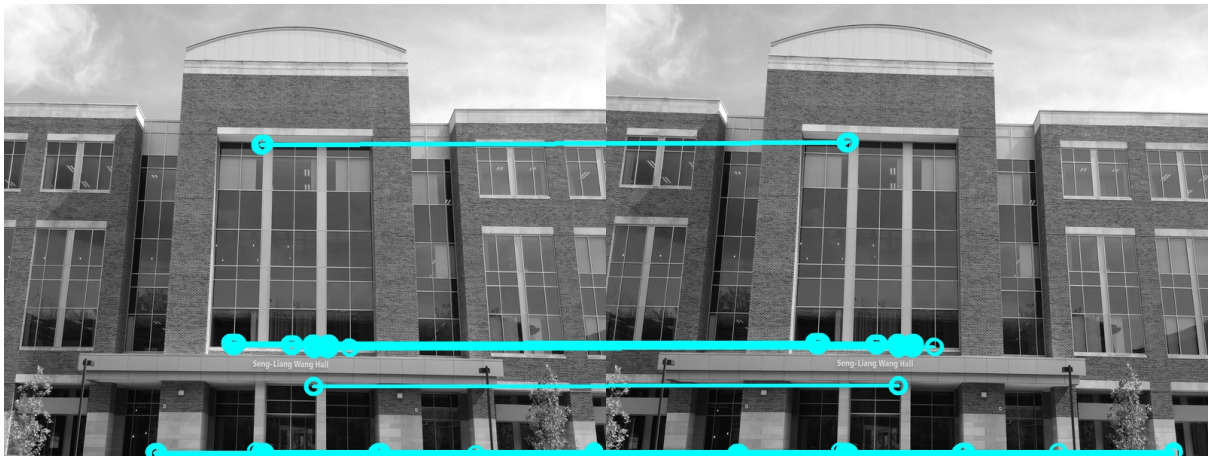

Figure 13: Correspondence-matching of corners at $\sigma = 2.4$ based on NCC for Image pair 1

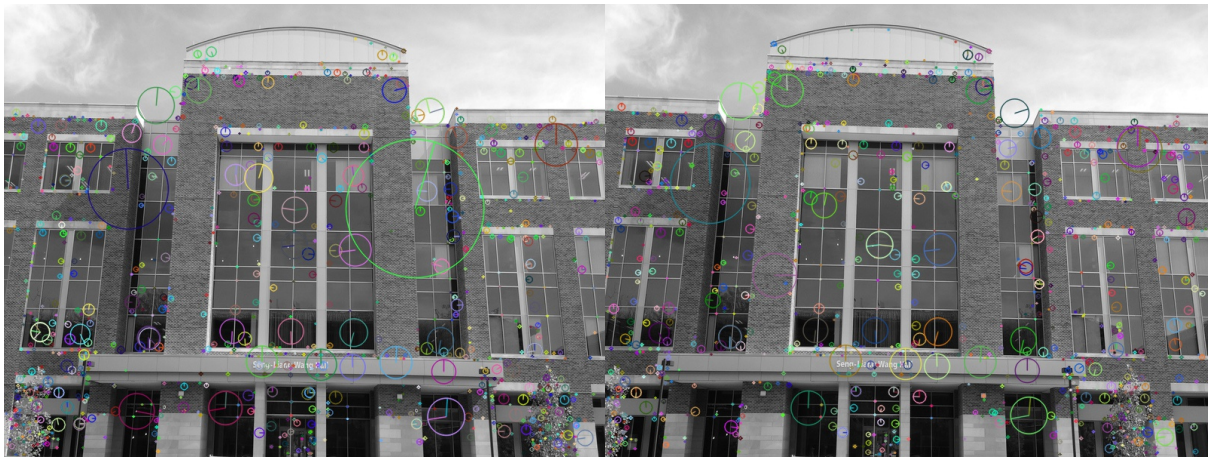

Figure 14: SIFT features identified for Image pair 1

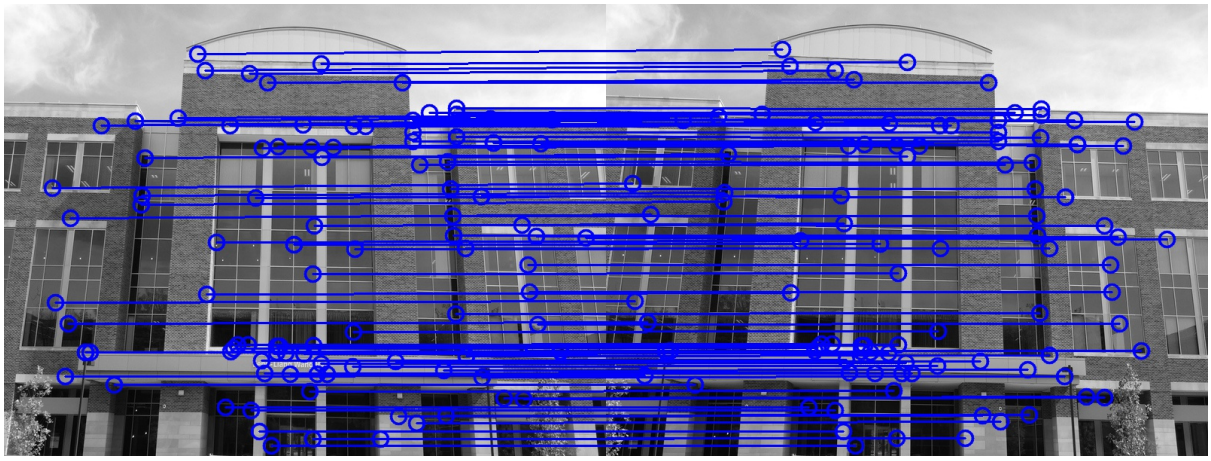

Figure 15: Correspondence-matching of SIFT features for Image pair 1

4.2 Image pair 2

Similar results for image pair 2 are shown in figures 16 to 30. The scales for Harris corner detector are [0.6, 1.2, 1.8, 2.4]. It is observed that the smaller scales of 0.6 and 1.2 result in detection of the pebbles on the ground as corner points, and it is only at the higher scales of 1.8 and 2.4 that the edges on the writing "U-HAUL" on the truck get detected. SIFT performs really well, apart from some pebbles on the ground getting detected and matched to pebbles at some other locations in the second image.

Figure 16: Image Pair 2

Figure 17: Harris corners identified at $\sigma = 0.6$ for Image pair 2

Figure 18: Harris corners identified at $\sigma = 1.2$ for Image pair 2

Figure 19: Harris corners identified at $\sigma = 1.8$ for Image pair 2

Figure 20: Harris corners identified at $\sigma = 2.4$ for Image pair 2

Figure 21: Correspondence-matching of corners at $\sigma = 0.6$ based on SSD for Image pair 2

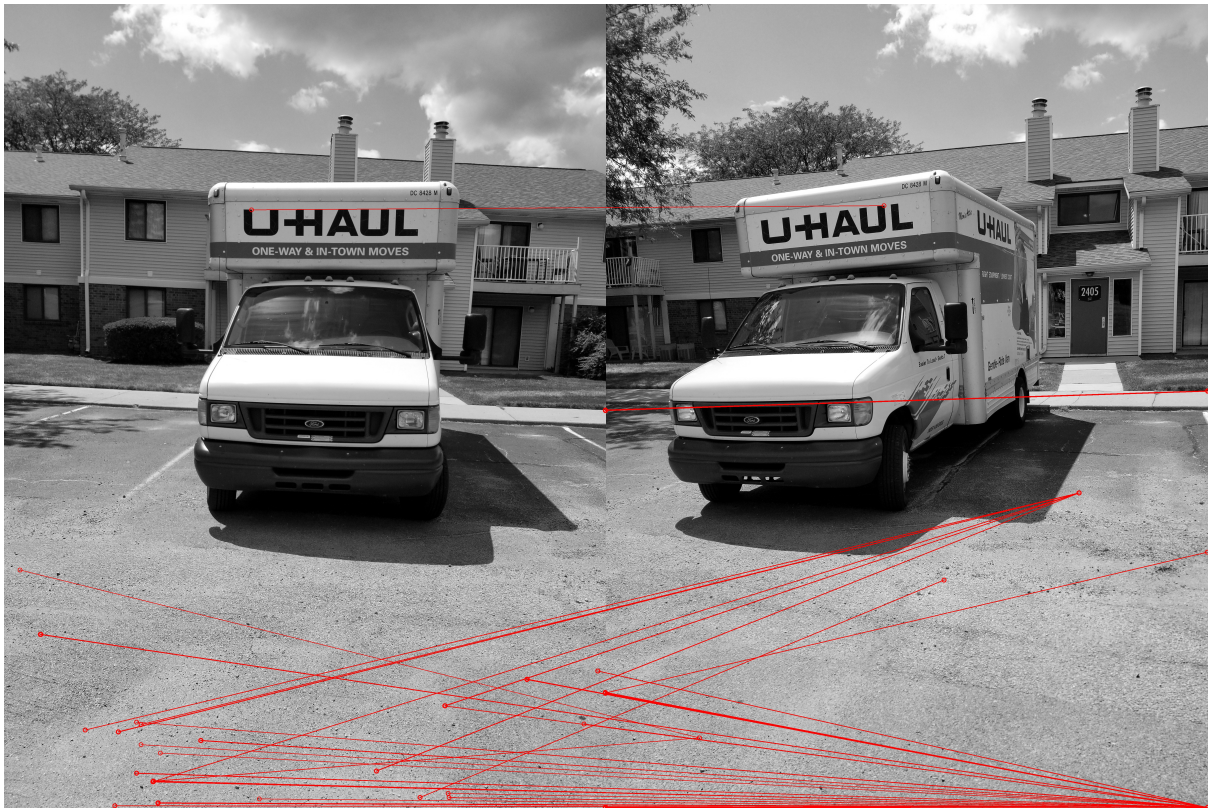

Figure 22: Correspondence-matching of corners at $\sigma = 1.2$ based on SSD for Image pair 2

Figure 23: Correspondence-matching of corners at $\sigma = 1.8$ based on SSD for Image pair 2

Figure 24: Correspondence-matching of corners at $\sigma = 2.4$ based on SSD for Image pair 2

Figure 25: Correspondence-matching of corners at $\sigma = 0.6$ based on NCC for Image pair 2

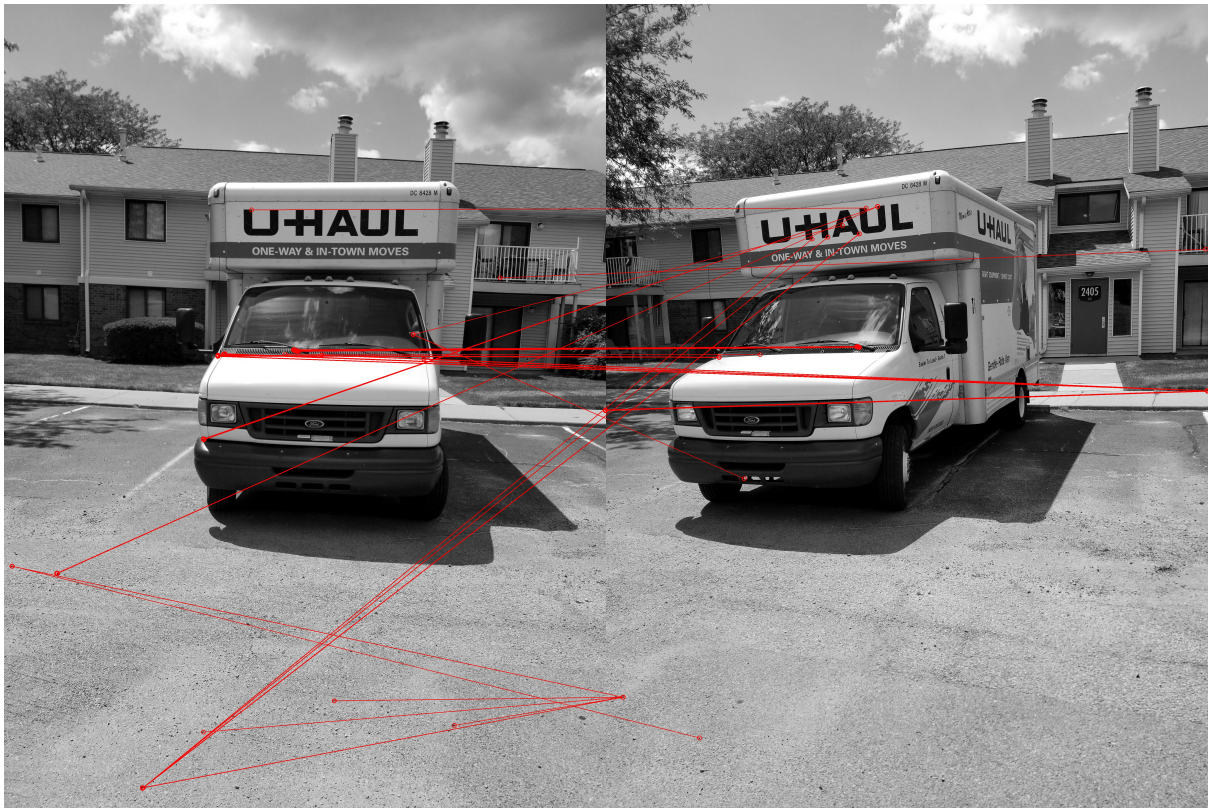

Figure 26: Correspondence-matching of corners at $\sigma = 1.2$ based on NCC for Image pair 2

Figure 27: Correspondence-matching of corners at $\sigma = 1.8$ based on NCC for Image pair 2

Figure 28: Correspondence-matching of corners at $\sigma = 2.4$ based on NCC for Image pair 2

Figure 29: SIFT features identified for Image pair 2

Figure 30: Correspondence-matching of SIFT features for Image pair 2

4.3 Image pair 3

Similar results for image pair 3 are shown in figures 31 to 45. The scales for Harris corner detector are [0.3, 0.6, 1.2, 2.4]. It is observed that the smaller values of the scale detect much finer corners in the images. Harris corner correspondence matching works well for both SSD and NCC. With SSD, no correspondences are seen for the car at the largest scale. This is because all the strongest correspondences in the lot are between the portions of the sky that are matched. The SIFT correspondences are observed to be extremely robust and work really well for the pair of images.

Figure 31: Image Pair 3

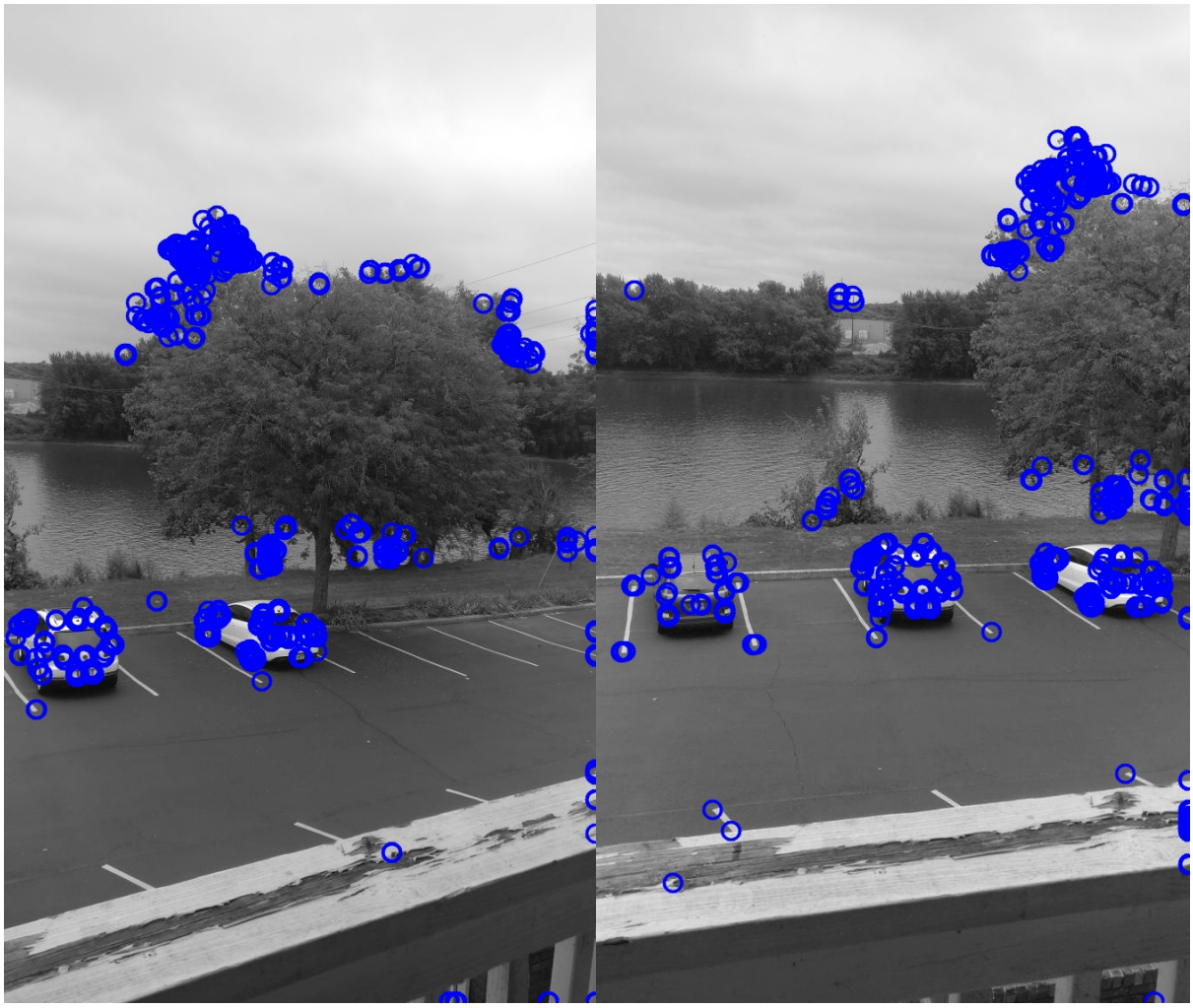

Figure 32: Harris corners identified at $\sigma = 0.3$ for Image pair 3

Figure 33: Harris corners identified at $\sigma = 0.6$ for Image pair 3

Figure 34: Harris corners identified at $\sigma = 1.2$ for Image pair 3

Figure 35: Harris corners identified at $\sigma = 2.4$ for Image pair 3

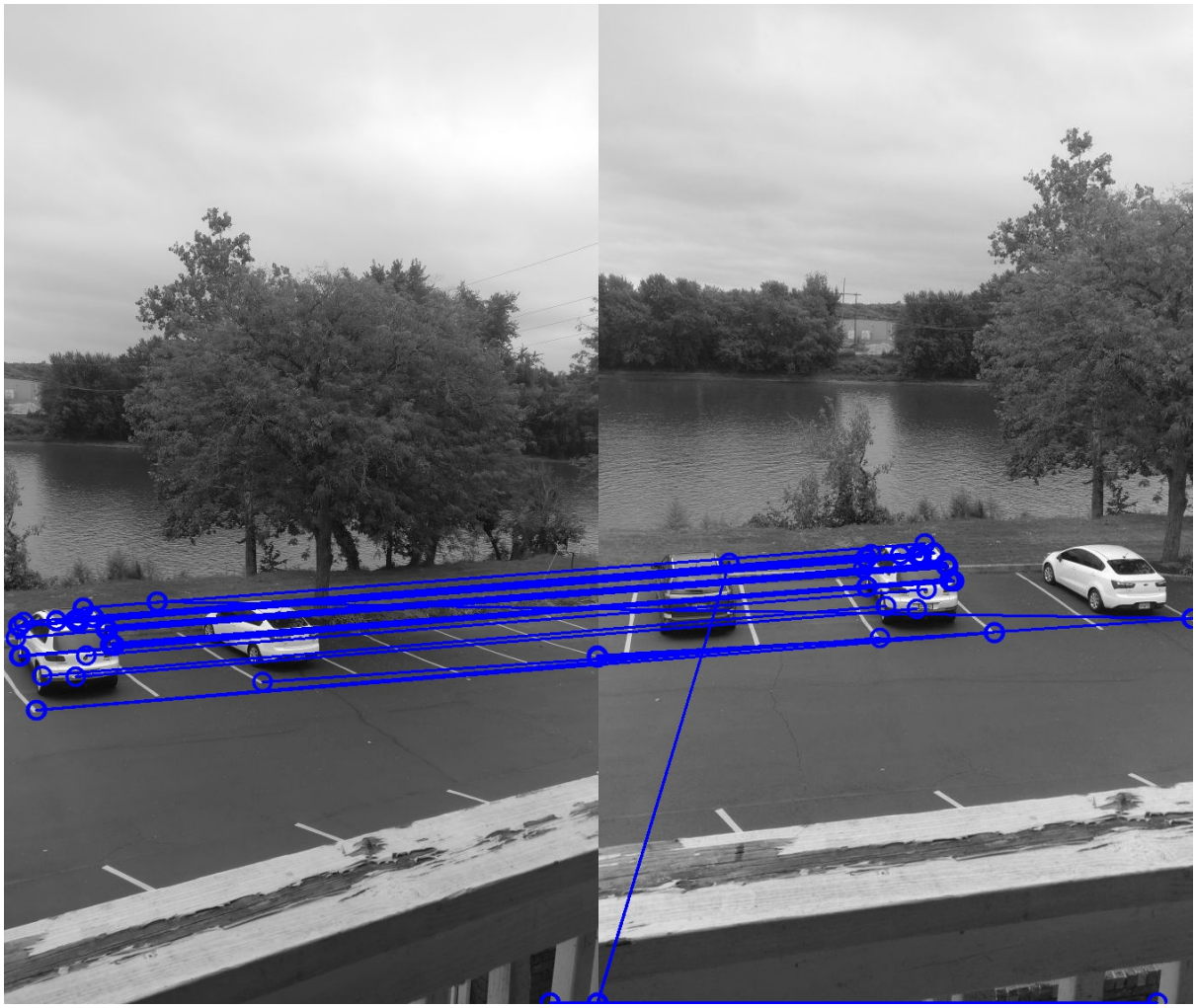

Figure 36: Correspondence-matching of corners at $\sigma = 0.3$ based on SSD for Image pair 3

Figure 37: Correspondence-matching of corners at $\sigma = 0.6$ based on SSD for Image pair 3

Figure 38: Correspondence-matching of corners at $\sigma = 1.2$ based on SSD for Image pair 3

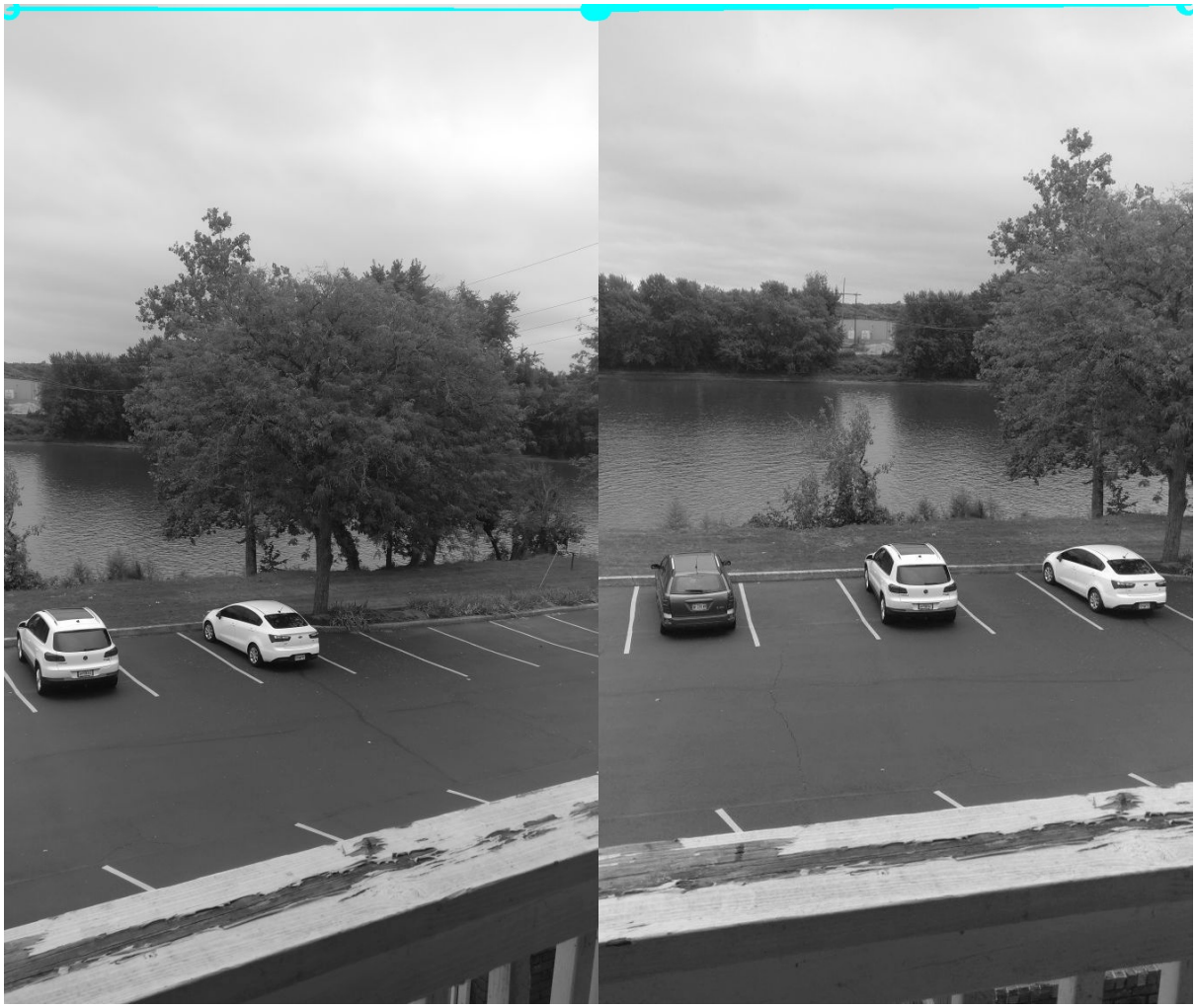

Figure 39: Correspondence-matching of corners at $\sigma = 2.4$ based on SSD for Image pair 3

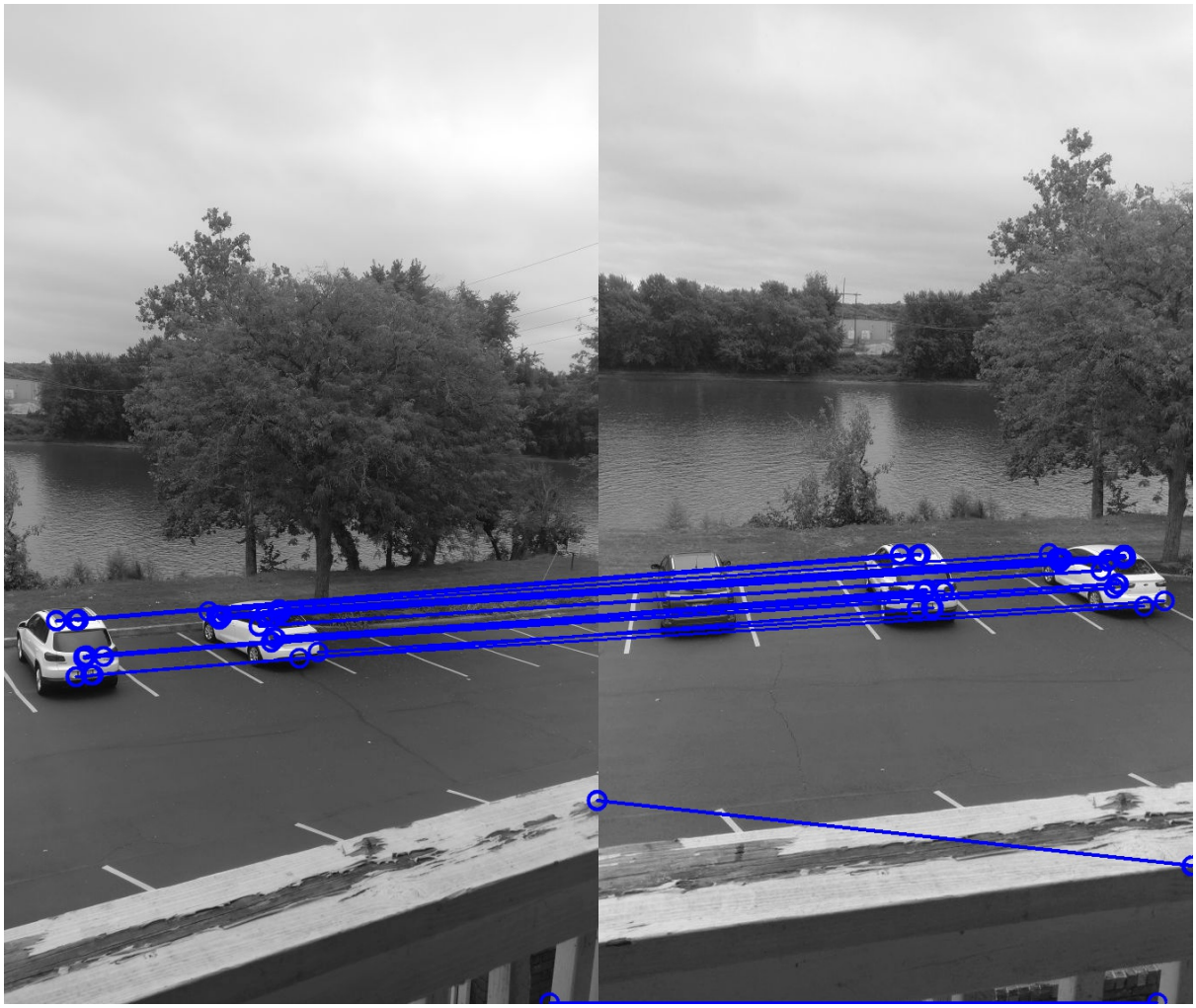

Figure 40: Correspondence-matching of corners at $\sigma = 0.3$ based on NCC for Image pair 3

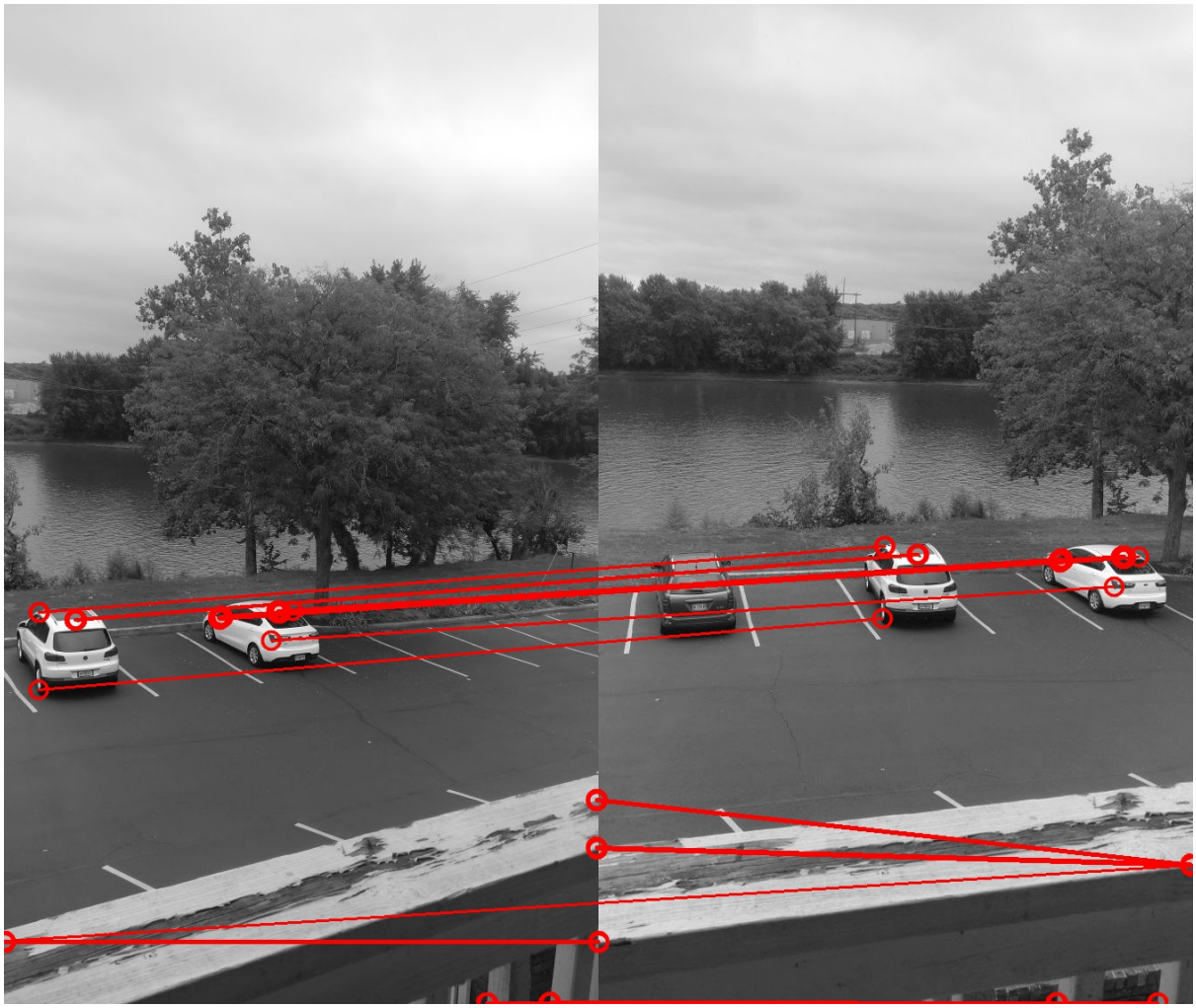

Figure 41: Correspondence-matching of corners at $\sigma = 0.6$ based on NCC for Image pair 3

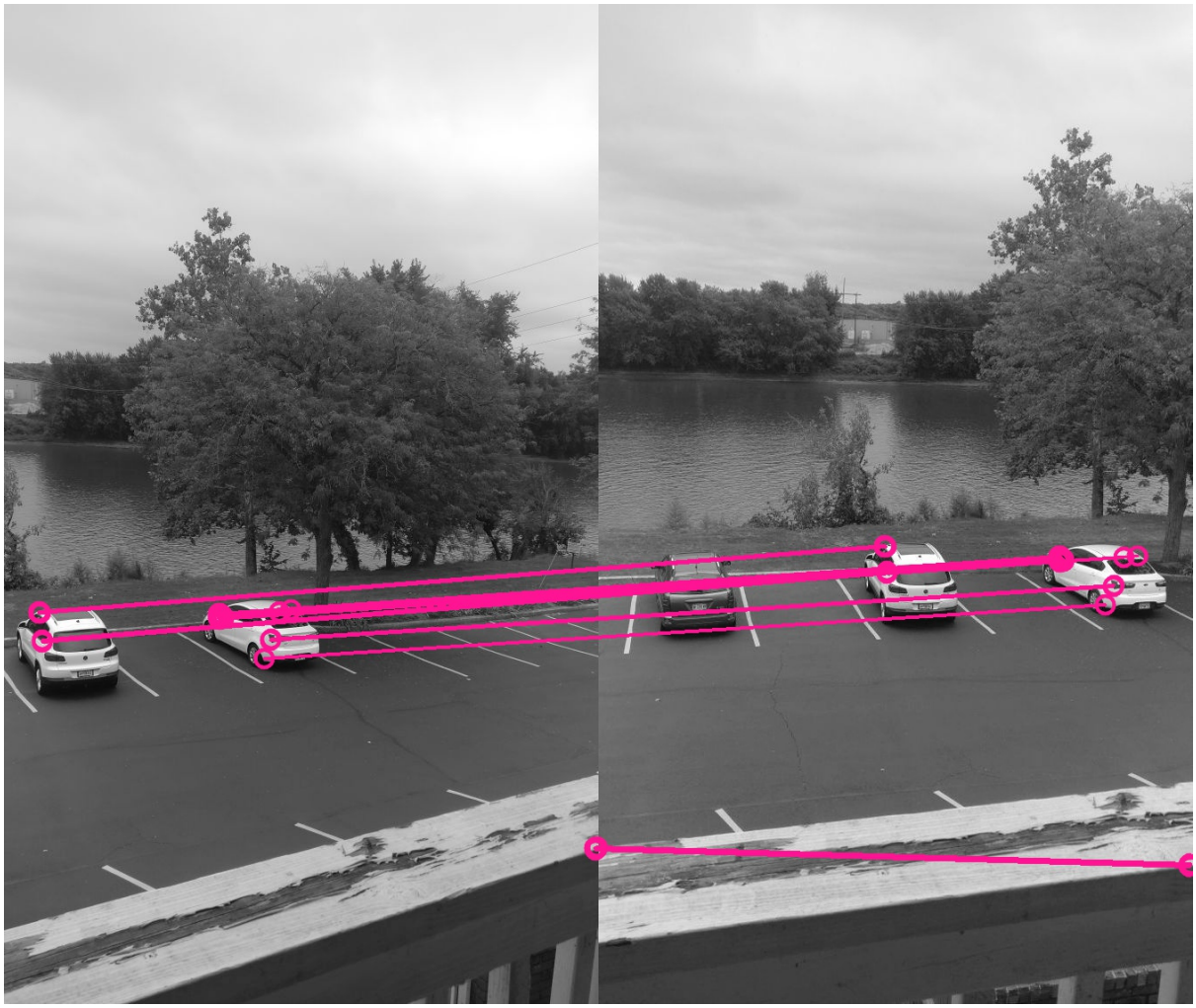

Figure 42: Correspondence-matching of corners at $\sigma = 1.2$ based on NCC for Image pair 3

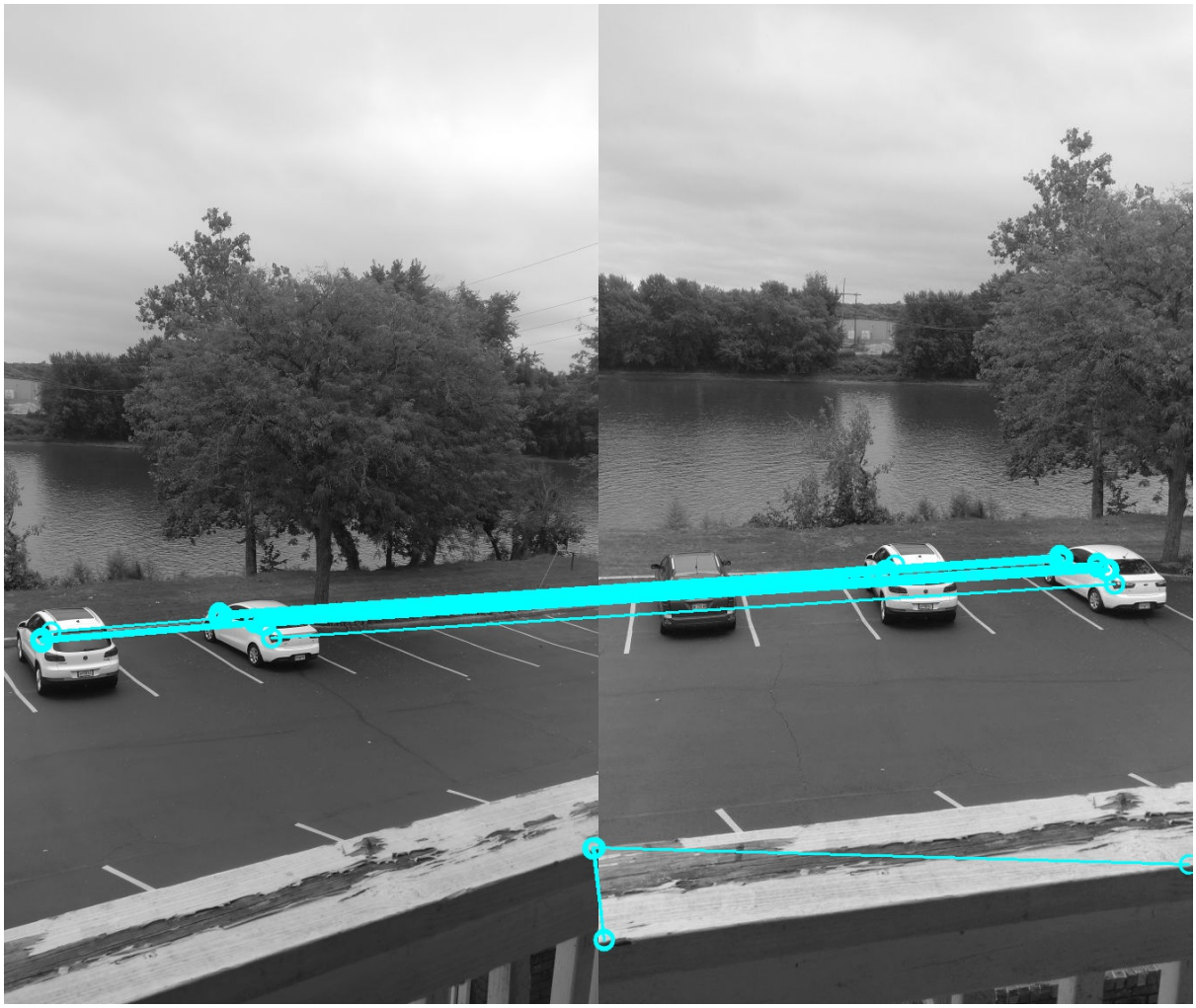

Figure 43: Correspondence-matching of corners at $\sigma = 2.4$ based on NCC for Image pair 3

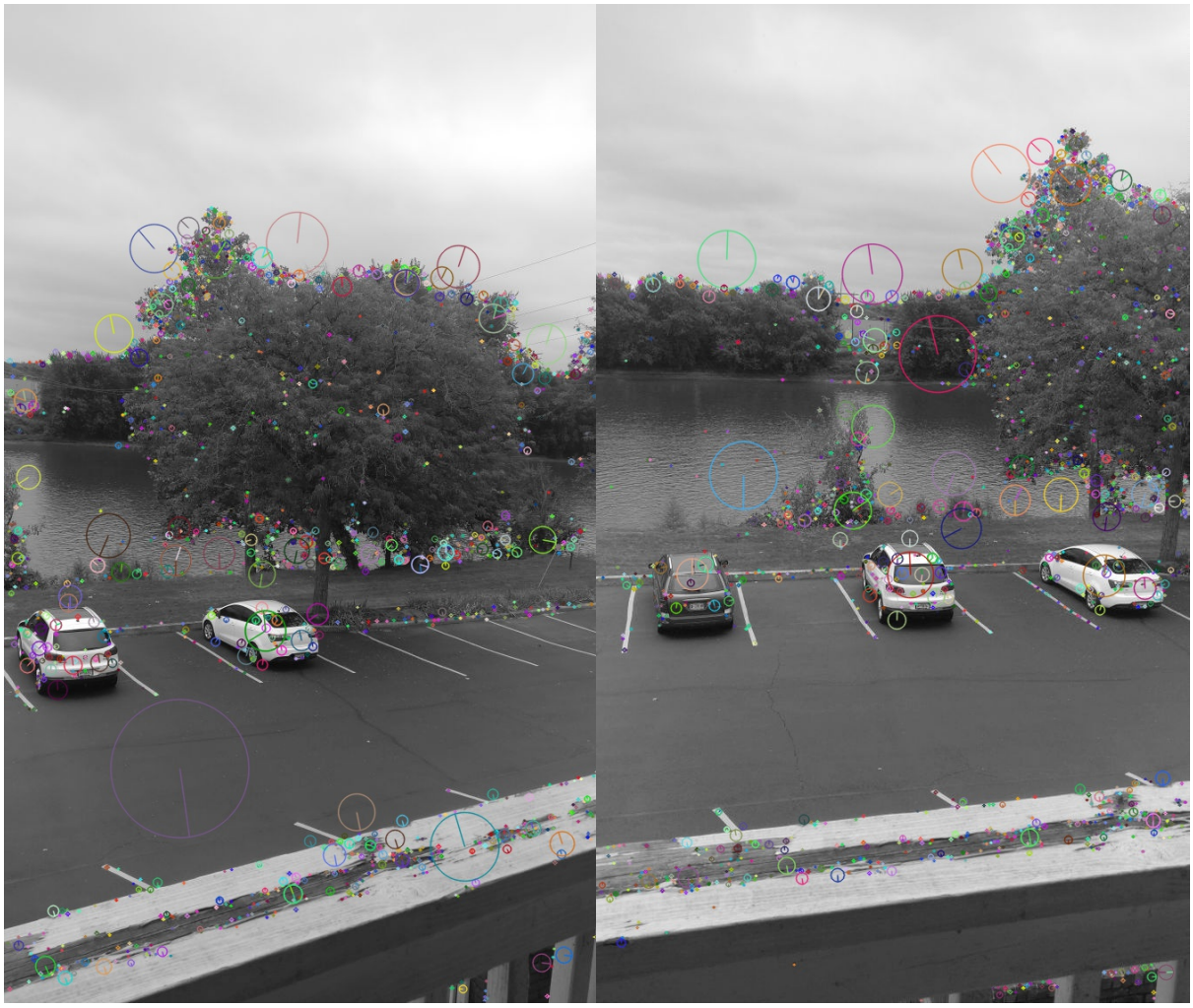

Figure 44: SIFT features identified for Image pair 3

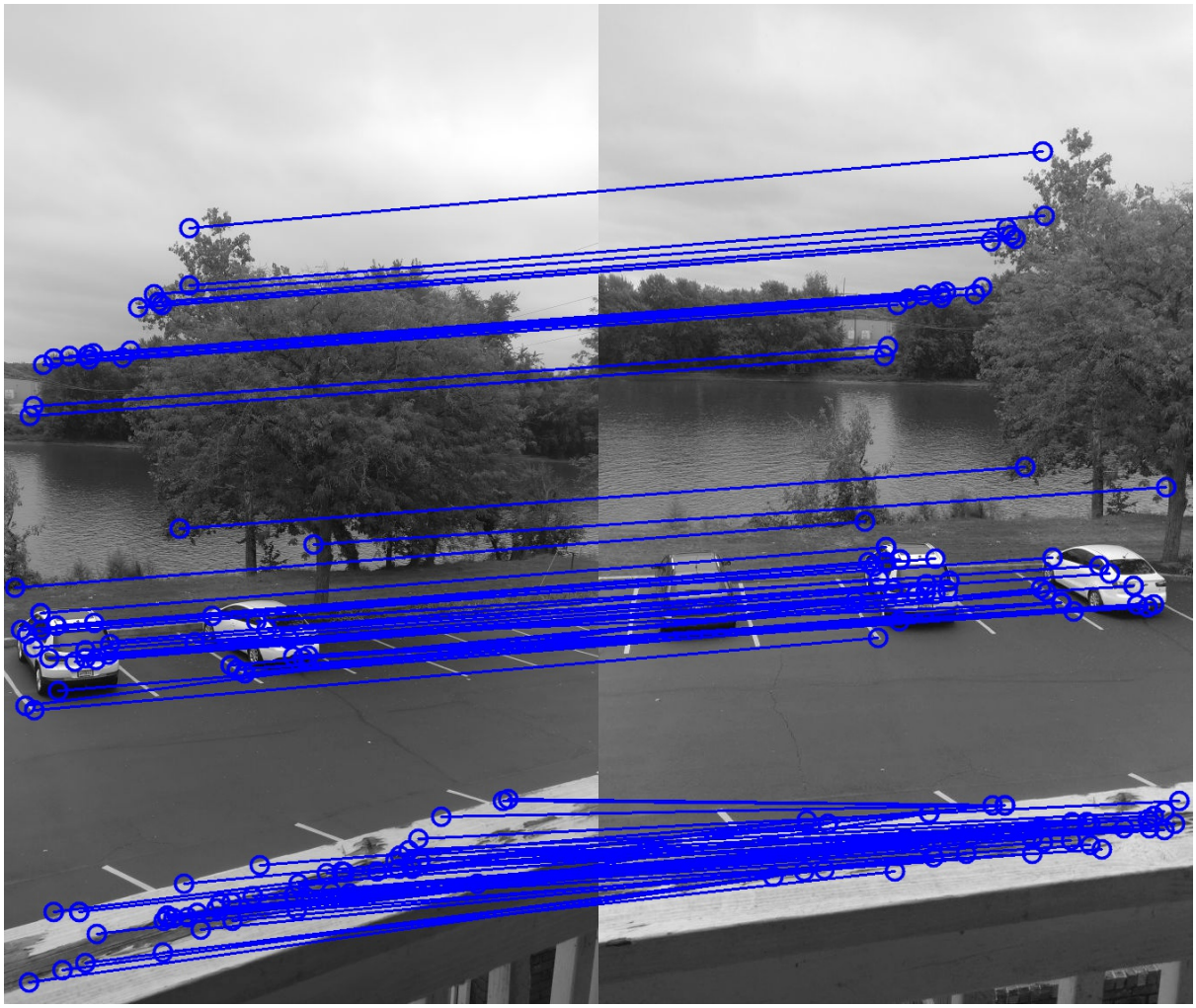

Figure 45: Correspondence-matching of SIFT features for Image pair 3

4.4 Image pair 4

Similar results for image pair 4 are shown in figures 46 to 60. The scales for Harris corner detector are [0.3, 0.6, 1.2, 2.4]. It is observed that Harris corner detection works well for the lower values of the scale. Even though correspondence matching works fairly well, there are some erroneous correspondences too. On the other hand, SIFT works extremely well in comparison.

Figure 46: Image Pair 4

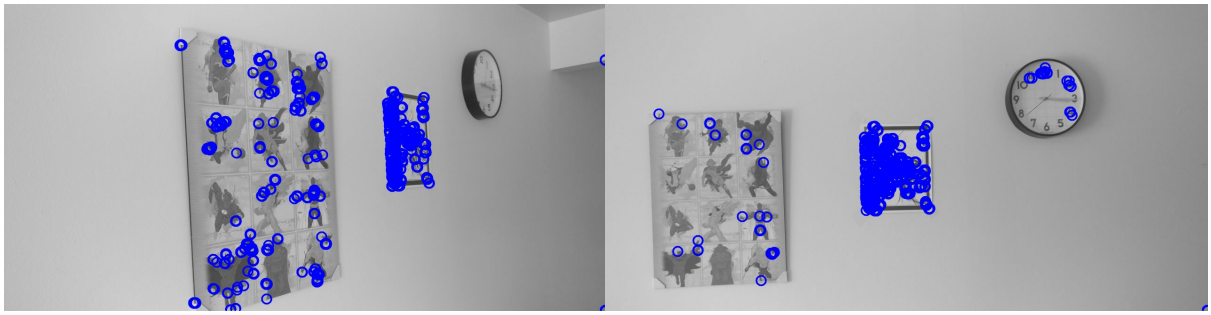

Figure 47: Harris corners identified at $\sigma = 0.3$ for Image pair 4

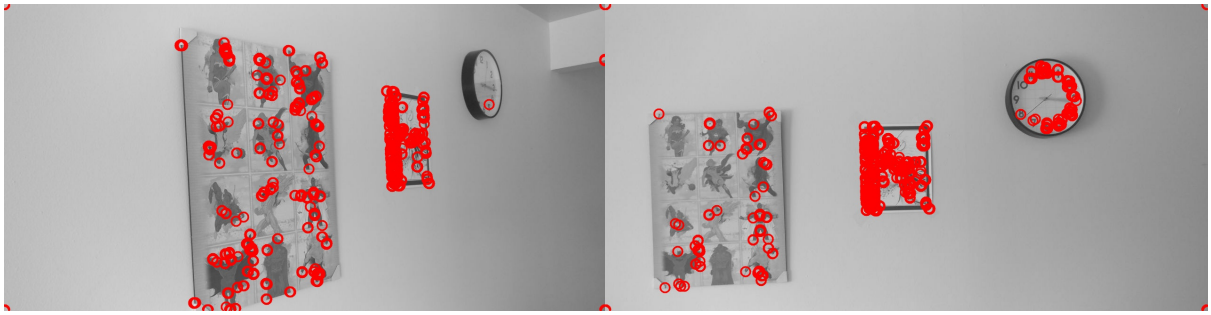

Figure 48: Harris corners identified at $\sigma = 0.6$ for Image pair 4

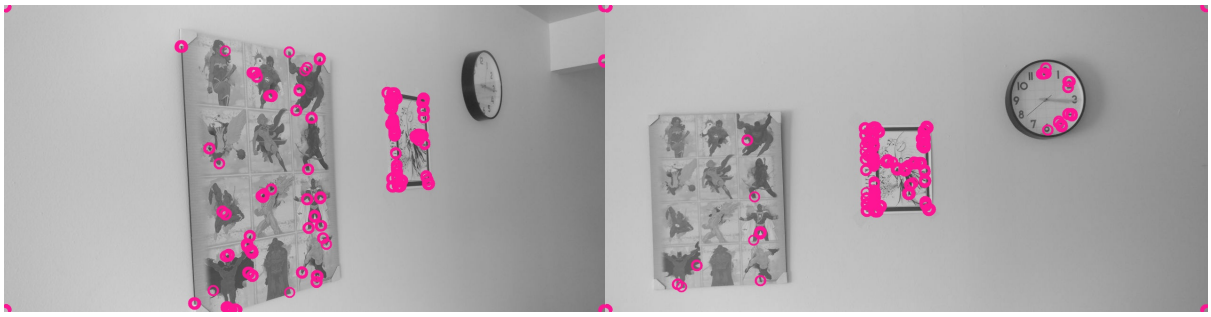

Figure 49: Harris corners identified at $\sigma = 1.2$ for Image pair 4

Figure 50: Harris corners identified at $\sigma = 2.4$ for Image pair 4

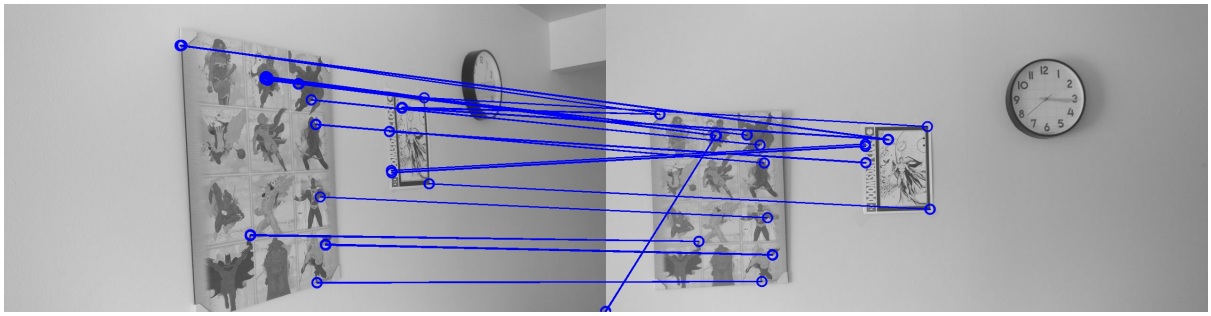

Figure 51: Correspondence-matching of corners at $\sigma = 0.3$ based on SSD for Image pair 4

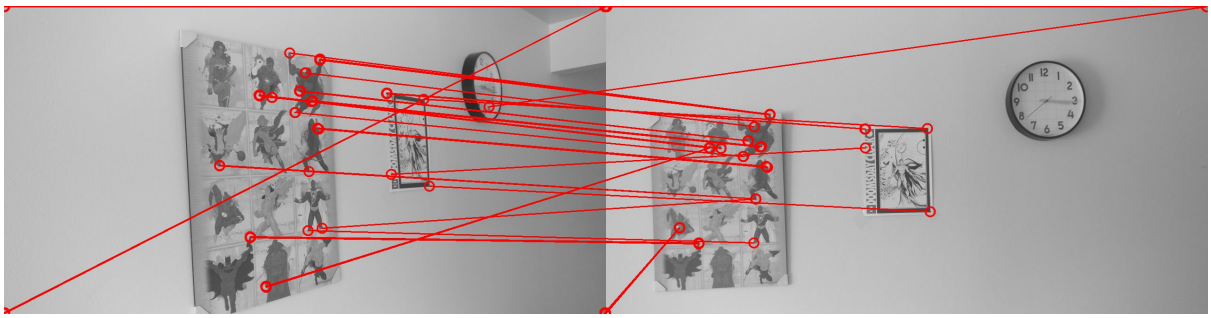

Figure 52: Correspondence-matching of corners at $\sigma = 0.6$ based on SSD for Image pair 4

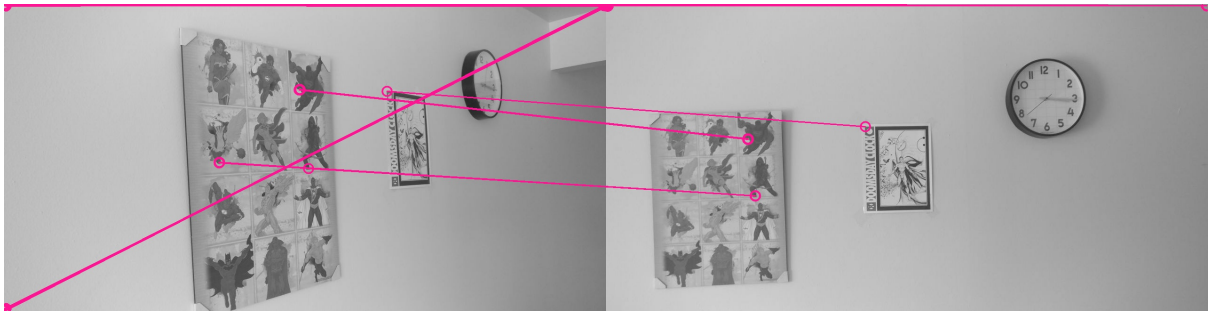

Figure 53: Correspondence-matching of corners at $\sigma = 1.2$ based on SSD for Image pair 4

Figure 54: Correspondence-matching of corners at $\sigma = 2.4$ based on SSD for Image pair 4

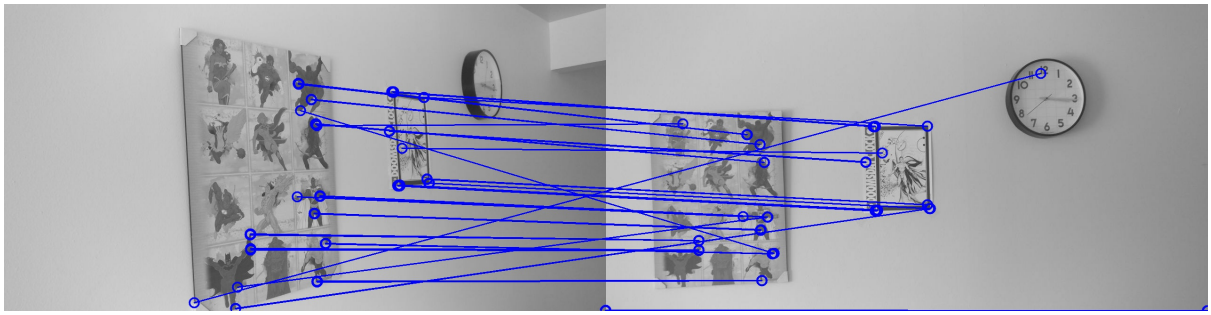

Figure 55: Correspondence-matching of corners at $\sigma = 0.3$ based on NCC for Image pair 4

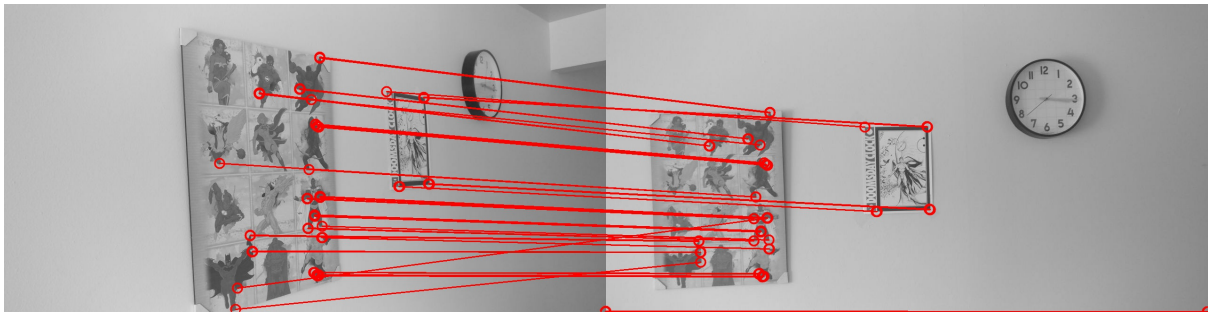

Figure 56: Correspondence-matching of corners at $\sigma = 0.6$ based on NCC for Image pair 4

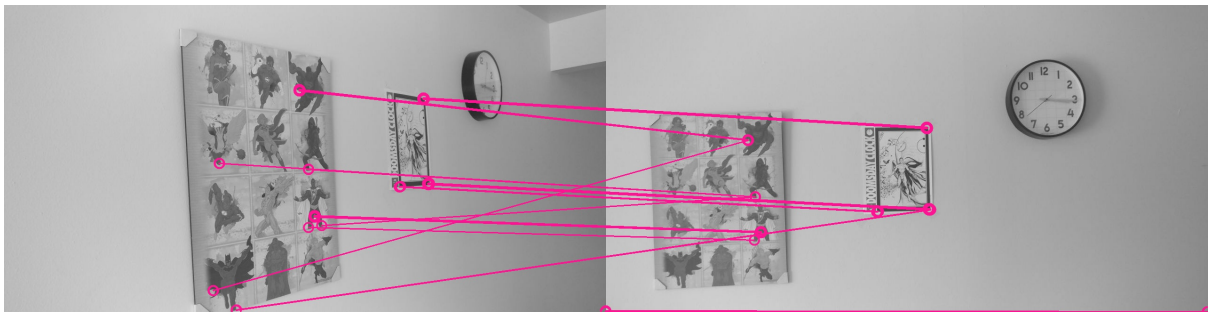

Figure 57: Correspondence-matching of corners at $\sigma = 1.2$ based on NCC for Image pair 4

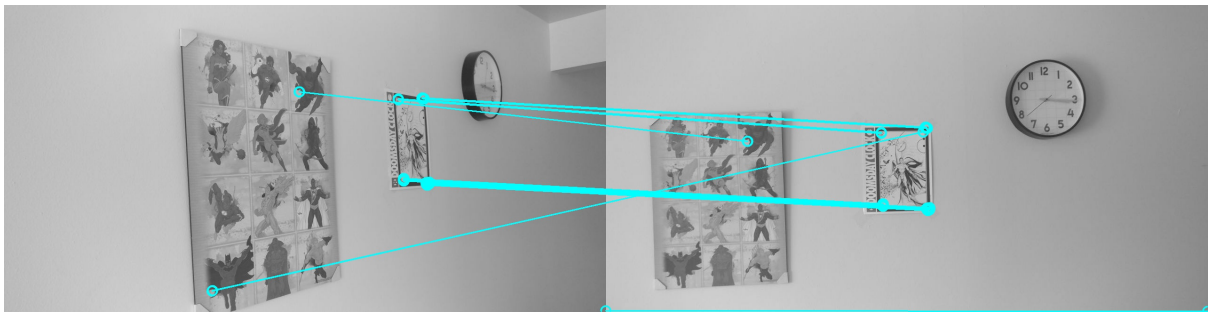

Figure 58: Correspondence-matching of corners at $\sigma = 2.4$ based on NCC for Image pair 4

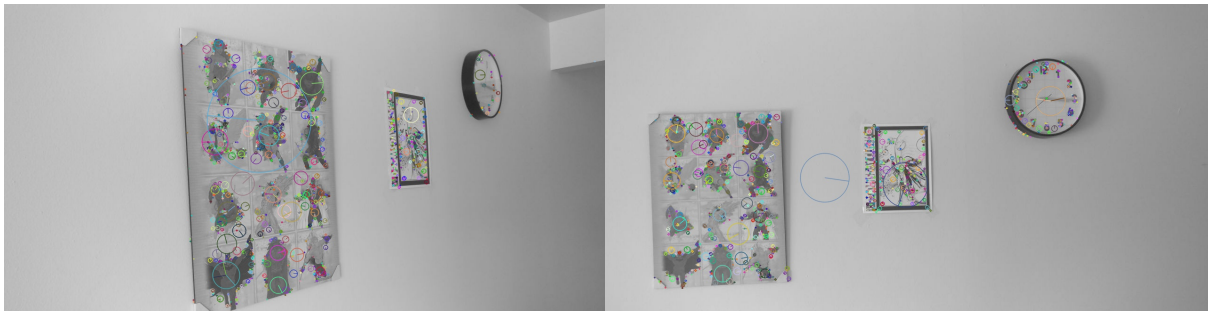

Figure 59: SIFT features identified for Image pair 4

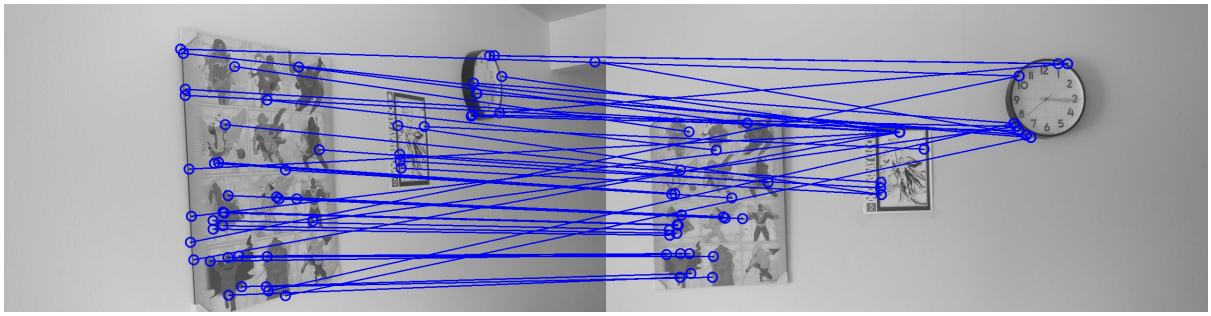

Figure 60: Correspondence-matching of SIFT features for Image pair 4

5 Observations and Conclusion

In all the pairs of images, it was observed that the effectiveness of Harris corner detection and correspondence matching depended extremely on the scale used. Even when the scale was good enough to detect a fairly large number of corners, the correspondence matching suffered in some cases as wrong correspondence pairs had higher strength. In addition, the success of Harris corner detection also depends on the window size selected for correspondence matching with SSD or NCC. In conclusion, even though Harris corner detector is a really good tool for interest point detection and correspondence matching, it is not robust to variations due to scale, illumination and noise. Its performance largely depends on the various tune-able parameters in the algorithm.

A comparison between the metrics SSD and NCC is worthwhile. Even though both the methods give fairly similar results for the image pairs 1, 2 and 3, NCC is observed to be marginally superior to SSD in image pair 4. But I highly doubt if the results add up to a justifiable sample space to draw any concrete inferences about the relative performance between SSD and NCC. Further investigation is required to state any meaningful remark regarding the same.

SIFT feature detection performs extremely well in comparison to Harris corner detection for all the image pairs. It gives a richer and more robust set of interest points. This is expected as SIFT makes effective use of the scale-space invariance criterion and dominant local orientation for interest point detection and matching. The robustness of the algorithm with respect to scale, orientation and illumination are unmatched by Harris corner detection. I recommend SIFT over Harris corner detection in situations where scales and illumination of the interest points are ambiguous.

6 Source code

The code is in python 2.7, with library_hw4.py having the common functions used, and task*.py calling the required functions from library_hw4.py. task*.py is the script for each of the tasks identified in the homework assignment. The names are self-explanatory.

The whole source code is provided in the pages that follow.

6.1 library_hw4

```
# Author: Gopikrishnan Sasi Kumar (Krish)

import numpy as np
import cv2
import scipy.signal as scisig

# Variables used globally
scales = [0.6, 1.2, 1.8, 2.4]  # scales at which detection carried out
featureRadius = 10
featureThickness = 2
PINK = (147, 20, 255)
BLUE = (255, 0, 0)
RED = (0, 0, 255)
CYAN = (255, 255, 0)
GREEN = (0, 255, 0)
base_colors = [BLUE, RED, PINK, CYAN]
corresp_window = 10  # correspondence checked with 21x21 windows
# for Harris corner detector

line_thickness = 2

# Number of best matched corners to be displayed for Harris
no_featurePoints_harris = 150

# k-ratio used for computing harris response
k_ratio = 0.05

# Number of corners to be detected per image per scale
no_harris_corners_detect = 1000

# Number of SIFT features per image
no_sift_features = 7000

# Number of best matched SIFT features to display
no_sift_display = 50


def get_haar(sigma):
    """
    Function to generate the Haar wavelet for a given sigma
    :param sigma: standard deviation of gaussian/scale
    :return: [haar_dx, haar_dy] the Haar wavelets for
    d/dx and d/dy for the given sigma
    """
    # The Haar wavelets for sigma is of size
```

```
    # M x M, where M is the smallest even integer
    # greater than 4sigma
    M = np.ceil(4 * sigma)
    if M % 2 == 1:
        M = M + 1
    M = int(M)

    haar1 = -1.0 * np.ones([M, M/2])
    haar2 = np.ones([M, M / 2])

    haar_dx = np.c_[haar1, haar2]
    haar_dy = -1 * haar_dx.T

    return [haar_dx, haar_dy]


def get_harris_corners(image, sigmas):
    """
    The function to get the Harris corners in a gray-scale image
    :param image: input image
    :param sigmas: array of sigmas to carry out corner detection
    :param fraction: fraction of max of Harris response for threshold
    :return: the coordinates of the corners
    """
    # loop through each of the sigma values
    corner_pixels = []
    index = 0
    for sigma in sigmas:
        [haar_dx, haar_dy] = get_haar(sigma)  # get Haar-wavelets
        # Convolve with haar wavelets as kernels to get dx, dy
        dx = scisig.correlate2d(image, haar_dx, mode="same")
        dy = scisig.correlate2d(image, haar_dy, mode="same")
        # Find dx^2, dy^2 and dxdy for each of the pixels through
        # element-wise multiplication
        dx2 = np.multiply(dx, dx)
        dy2 = np.multiply(dy, dy)
        dxdy = np.multiply(dx, dy)
        # Generate a kernel for finding sums over 5sigma x 5sigma window
        ker_width = np.ceil(5 * sigma)
        if ker_width % 2 == 0:
            ker_width = ker_width + 1
        ker_width = int(ker_width)
        kernel = np.ones([ker_width, ker_width])
        # Now, convolve with kernel to get sums of dx2, dy2 and dxdy over
        # 5sigma x 5sigma windows
        dx2sum = scisig.correlate2d(dx2, kernel, mode="same")
        dy2sum = scisig.correlate2d(dy2, kernel, mode="same")
        dxdysum = scisig.correlate2d(dxdy, kernel, mode="same")
        # Find the determinant and trace of C matrix
        # (ref. Harris corner detection)
        C_det = np.subtract(np.multiply(dx2sum, dy2sum),
                            np.multiply(dxdysum, dxdysum))
        C_trace = np.add(dx2sum, dy2sum)

        # Find the square of trace of C
        C_trace2 = np.multiply(C_trace, C_trace)
```

```
            # Find the Harris response
            checker = np.subtract(C_det, k_ratio * C_trace2)

            # Find pixels where ratio is above threshold
            checkerflat = checker.flatten()
            sortedChecker = sorted(checkerflat)
            sortedChecker.reverse()
            R_thresh = sortedChecker[no_harris_corners_detect-1]
            # R_thresh = np.percentile(checker, R_threshold)
            # print(R_thresh)
            i, j = np.where(checker >= R_thresh) #checker.max()/fraction)

            # Updating corner pixels
            corner_pixels.append(zip(j, i))

        return corner_pixels


def mark_feature_pixels(image, pixel_locs, feature_color):
    """
    Function to return the input image with pixels marked
    :param image: Input image
    :param pixel_locs: locations of pixels to be marked
    :param feature_color: color for feature marking
    :return: image with pixels marked
    """
    for point in pixel_locs:
        cv2.circle(image, point, featureRadius, feature_color,
                                    featureThickness, cv2.LINE_AA)
    return image


def mark_feature_pixels_all_scales(image, corners):
    """
    Function to mark corners with all scales
    :param image: Input image
    :param corners: List of corners at different scales
    :return: image with all corners marked
    """
    index = 0
    for cornerset in corners:
        print(index)
        image = mark_feature_pixels(image, cornerset,
                                    base_colors[index % len(base_colors)])
        index = index + 1
    return image


def find_harris_corresp(image1, corners1, image2, corners2, ssd_or_ncc):
    """
    Function to find the correspondences between image1 with corners1
    and image2 with corners2
    :param image1: first image
    :param corners1: corners of 1st image
    :param image2: second image
```

```
:param corners2: corners of 2nd image
:param ssd_or_ncc: flag to define if ssd or ncc is used.
True: ssd, False: ncc
:return: correspondences: as pixel pairs, one list of pairs per scale
"""

class data:
    def __init__(self, point_pair, value):
        self.point_pair = point_pair
        self.value = value

# image1 = image1.astype("int32")
# image2 = image2.astype("int32")
index = 0
innerindex = 0
corresp_output = []
# Unpacking corners for all sigmas into single list
corners1_counter = [item for sublist in corners1 for item in sublist]
corners2_counter = [item for sublist in corners2 for item in sublist]
print("Determining Harris Corner Correspondences")
print("Corner {}/{}".format(index, len(corners1_counter)))



for corner_index in range(len(corners1)):
    corners1_this = corners1[corner_index]
    corners2_this = corners2[corner_index]
    correspondences = []
    ssd_ncc_trace = []
    f1_corpus = []
    f2_corpus = []
    m1_corpus = []
    m2_corpus = []

    for corner1 in corners1_this:
        f1 = get_corresp_window(image1, corner1, corresp_window)
        f1 = f1.astype("int64")
        f1_corpus.append(f1)
        m1_corpus.append(f1.sum() / np.prod(f1.shape))  # mean of f1
    for corner2 in corners2_this:
        f2 = get_corresp_window(image2, corner2, corresp_window)
        f2 = f2.astype("int64")
        f2_corpus.append(f2)
        m2_corpus.append(f2.sum() / np.prod(f2.shape))  # mean of f2

    for f1_counter, corner1 in enumerate(corners1_this):
        # get correspondence window for corner1
        f1 = f1_corpus[f1_counter]
        m1 = m1_corpus[f1_counter]
        f1_minus_m1 = f1 - m1
        ssd_best = np.inf  # best ssd initialisation
        ncc_best = -np.inf  # best ncc initialisation

        for f2_counter, corner2 in enumerate(corners2_this):
            # get correspondence window for corner2
            f2 = f2_corpus[f2_counter]
```

```
            # print(corner1, corner2)

            if ssd_or_ncc:  # if SSD is the metric chosen
                ssd = abs(f1 - f2)
                ssd = ssd.sum()  # get Sum of Squared Differences
                # minimize ssd
                if ssd < ssd_best:
                    ssd_best = ssd
                    new_pair = [corner1, corner2]
            else:   # if NCC is the metric chosen
                m2 = m2_corpus[f2_counter]
                f2_minus_m2 = f2 - m2

                num = np.multiply(f1_minus_m1, f2_minus_m2)
                num = num.sum()

                den1 = f1_minus_m1 ** 2
                den1 = den1.sum()

                den2 = f2_minus_m2 ** 2
                den2 = den2.sum()

                ncc = num / np.sqrt(den1 * den2)

                # maximize ncc
                if ncc > ncc_best:
                    ncc_best = ncc
                    new_pair = [corner1, corner2]

            innerindex = innerindex + 1

        # corners2.remove(pixel_to_remove)
        # append the new matched pair of pixels
        if ssd_or_ncc:
            correspondences.append(data(new_pair, ssd_best))
        else:
            correspondences.append(data(new_pair, ncc_best))
        index = index + 1
        if index % 100 == 0:
            print("Corner {}/{}".format(index, len(corners1_counter)))

    corr_new = []
    correspondences = sorted(correspondences, key = lambda x:x.value)
    if ssd_or_ncc:
        pass
    else:
        correspondences.reverse()

    for i in range(len(correspondences)):
        corr_new.append(correspondences[i].point_pair)

    corresp_output.append(corr_new[:no_featurePoints_harris])

return corresp_output
```

```
def get_corresp_window(image, point, window):
    """
    Function that returns 2M+1 by 2M+1 window centred at point in image
    :param image: input image
    :param point: centre point
    :param window: M
    :return: 2M+1 by 2M+1 window from image
    """
    imagecopy = cv2.copyMakeBorder(image, window, window,
                                   window, window,
                                   cv2.BORDER_REPLICATE)
    x = point[0]
    y = point[1]

    return imagecopy[y:y+2*window+1, x:x+2*window+1]


def get_matching_image(image1, image2, correspondences, color):
    """
    Function that returns the image matching the correspondences
    :param image1: 1st image
    :param image2: 2nd image
    :param correspondences: pairs of pixels with correspondences
    between image1 and image2
    :param color: Color for matching line
    :return: image with correspondences matched
    """
    shape1 = image1.shape
    shape2 = image2.shape
    height = max(shape1[0], shape2[0])
    img1 = np.zeros((height, shape1[1], 3), dtype='uint8')
    img1[0:height, 0:shape1[1]] = image1

    img2 = np.zeros((height, shape2[1], 3), dtype='uint8')
    img2[0:height, 0:shape2[1]] = image2

    # generate an image with image 1 and 2 on left and right
    image = np.concatenate((img1, img2), axis=1)

    index = 0

    for point_pair in correspondences:
        point1 = point_pair[0]
        point2 = point_pair[1]
        cv2.circle(image, tuple(point1), featureRadius,
                   color, featureThickness, cv2.LINE_AA)
        cv2.circle(image, tuple([point2[0] + shape1[1], point2[1]]), featureRadius,
                   color, featureThickness, cv2.LINE_AA)
        cv2.line(image, tuple(point1), tuple([point2[0] + shape1[1], point2[1]]),
                 color, line_thickness)
        index = index + 1

    return image


def task_harris(img1, img2, input_folder, output_folder):
```

```
    """
    Function that carries out Harris corner detection and correspondence
    matching for img1 and img2 in input_folder
    :param img1: image1
    :param img2: image2
    :param input_folder: input folder
    :param output_folder: output folder into which results are saved
    :return: saving of files into result folder
    """
    # detect corners in both images
    [corners1, image1_g] = harris_detect_save(img1, input_folder, output_folder,
                                              "Image1", False)
    [corners2, image2_g] = harris_detect_save(img2, input_folder, output_folder,
                                                "Image2", False)
    print("Corners detection complete")

    # determine correspondences between image features based on SSD
    correspondences_ssd = find_harris_corresp(image1_g.copy(), corners1, image2_g.copy(),
                                                corners2, True)
    print("Correspondences determined based on SSD")

    print("Generating output image")
    image1 = cv2.imread(input_folder + img1)
    image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    image1 = cv2.cvtColor(image1, cv2.COLOR_GRAY2BGR)

    image2 = cv2.imread(input_folder + img2)
    image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
    image2 = cv2.cvtColor(image2, cv2.COLOR_GRAY2BGR)

    for scale_index in range(len(correspondences_ssd)):
        image = get_matching_image(image1.copy(), image2.copy(), correspondences_ssd[scale_index],
                                   base_colors[scale_index % len(base_colors)])
        cv2.imwrite(output_folder + "match_SDD_scale{}.jpg".format(scale_index), image)
        open_named_window(image, "match_SDD_scale{}.jpg".format(scale_index))

    # determine correspondences between image features based on NCC
    correspondences_ncc = find_harris_corresp(image1_g.copy(), corners1, image2_g.copy(),
                                                corners2, False)
    print("Correspondences determined based on NCC")

    print("Generating output image")
    for scale_index in range(len(correspondences_ncc)):
        image = get_matching_image(image1.copy(), image2.copy(), correspondences_ncc[scale_index],
                                   base_colors[scale_index % len(base_colors)])
        cv2.imwrite(output_folder + "match_NCC_scale{}.jpg".format(scale_index), image)
        open_named_window(image, "match_NCC_scale{}.jpg".format(scale_index))

    print("Harris corner matching complete.\n"
          "Check results folder.")
    return 1


def harris_detect_save(img1, input_folder, output_folder, imagetext, display_switch):
    """
    Function to read an image, detect corners, and save the images
```

```
    with interest points marked
    :param img1: image location
    :param input_folder:
    :param output_folder:
    :param imagetext: text used to display and save outputs
    :return:
    """
    image1 = cv2.imread(input_folder + img1, cv2.IMREAD_GRAYSCALE)
    outputimage = image1.copy()

    # find corners with harris corner detector
    corners1 = get_harris_corners(image1, scales)
    temp = [item for sublist in corners1 for item in sublist]
    print("{} corners determined in {}".format(len(temp), imagetext))

    index = 0

    # Marking interest points on image 1 at different scales
    image1 = cv2.imread(input_folder + img1)
    image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    image1 = cv2.cvtColor(image1, cv2.COLOR_GRAY2BGR)
    image1_marked_scale0 = image1.copy()
    image1_marked_scale1 = image1.copy()
    image1_marked_scale2 = image1.copy()
    image1_marked_scale3 = image1.copy()
    image1_marked_scale0 = mark_feature_pixels(image1_marked_scale0, corners1[0],
                                               base_colors[index % len(base_colors)])
    index = index + 1
    image1_marked_scale1 = mark_feature_pixels(image1_marked_scale1, corners1[1],
                                               base_colors[index % len(base_colors)])
    index = index + 1
    image1_marked_scale2 = mark_feature_pixels(image1_marked_scale2, corners1[2],
                                               base_colors[index % len(base_colors)])
    index = index + 1
    image1_marked_scale3 = mark_feature_pixels(image1_marked_scale3, corners1[3],
                                               base_colors[index % len(base_colors)])

    # saving image with interest points marked
    cv2.imwrite(output_folder + imagetext + "_marked_scale0.jpg", image1_marked_scale0)
    cv2.imwrite(output_folder + imagetext + "_marked_scale1.jpg", image1_marked_scale1)
    cv2.imwrite(output_folder + imagetext + "_marked_scale2.jpg", image1_marked_scale2)
    cv2.imwrite(output_folder + imagetext + "_marked_scale3.jpg", image1_marked_scale3)
    print(imagetext + " with interest points saved to files")

    if display_switch:
        # displaying image with interest points marked
        open_named_window(image1_marked_scale0, imagetext + ': interest points scale 0')
        open_named_window(image1_marked_scale1, imagetext + ': interest points scale 1')
        open_named_window(image1_marked_scale2, imagetext + ': interest points scale 2')
        open_named_window(image1_marked_scale3, imagetext + ': interest points scale 3')
        cv2.destroyAllWindows()

    return [corners1, outputimage]


def open_named_window(image, windowname, switch = False):
```

```
    """
    Function to open cv2 named window with image and wait for keystroke
    :param image:
    :param windowname:
    :param switch: window required or not? Optional input.
    :return:
    """
    if switch == False:
        return 1
    cv2.namedWindow(windowname, cv2.WINDOW_NORMAL)
    cv2.imshow(windowname, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return 1


def task_sift(img1, img2, input_folder, output_folder):
    """
    Function that carries out SIFT feature detection and
    correspondence matching for img1 and img2 in input_folder
    :param img1: image1
    :param img2: image2
    :param input_folder: input folder
    :param output_folder: output folder into which results are saved
    :return: saving of files into result folder
    """
    # Extract features from first image
    image1 = cv2.imread(input_folder + img1)
    gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    sift1 = cv2.xfeatures2d.SIFT_create(no_sift_features)
    print("Features determined in Image 1")
    kp1, des1 = sift1.detectAndCompute(gray1, None)
    image_op1 = cv2.drawKeypoints(gray1, kp1, outImage=np.array([]),
                                  flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    cv2.imwrite(output_folder + "sift_features_" + img1, image_op1)

    # Extract features from second image
    image2 = cv2.imread(input_folder + img2)
    gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
    sift2 = cv2.xfeatures2d.SIFT_create(no_sift_features)
    print("Features determined in Image 2")
    kp2, des2 = sift2.detectAndCompute(gray2, None)
    image_op2 = cv2.drawKeypoints(gray2, kp2, outImage=np.array([]),
                                  flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    cv2.imwrite(output_folder + "sift_features_" + img2, image_op2)

    # Find correspondences between features in image1 and image2
    correspondences = find_sift_correspondences(kp1, des1, kp2, des2, image_op1, image_op2)
    image = get_matching_image(cv2.cvtColor(gray1, cv2.COLOR_GRAY2BGR),
                               cv2.cvtColor(gray2, cv2.COLOR_GRAY2BGR),
                               correspondences, BLUE)
    cv2.imwrite(output_folder + "sift_match.jpg", image)
    print("Files saved. Check Results folder.")
    return 1
```

```
def find_sift_correspondences(kp1, des1, kp2, des2, img1, img2):
    """
    Function to find the correspondences between feature points in image1
    to those in image2
    :param kp1: keypoints in image 1
    :param des1: feature descriptors for keypoints in kp1
    :param kp2: keypoints in image 2
    :param des2: feature descriptors for keypoints in kp2
    :param img1: image 1
    :param img2: image 2
    """
    class data:  # class structure used for sorting point pairs
        def __init__(self, point_pair, value):
            self.point_pair = point_pair
            self.value = value

    des1_order = des1.shape
    des2_order = des2.shape

    correspondences = []
    corr_new = []

    # Find euclidean distance between all pairs of keypoints
    # Add them to correspondences list based on closeness
    for i in range(des1_order[0]):
        dist_max = np.Inf
        point1 = kp1[i].pt
        point1 = (int(round(point1[0])), int(round(point1[1])))
        for j in range(des2_order[0]):
            point2 = kp2[j].pt
            point2 = (int(round(point2[0])), int(round(point2[1])))
            dist = np.linalg.norm(des1[i, :] - des2[j, :])
            if dist < dist_max:
                dist_max = dist
                new_pair = [point1, point2]
        correspondences.append(data(new_pair, dist_max))
        # Sort point pairs based on euclidean distance
        correspondences = sorted(correspondences, key = lambda x:x.value)

    # Choose only the point pairs as output
    for i in range(len(correspondences)):
        corr_new.append(correspondences[i].point_pair)
    return corr_new[:no_sift_display]
```

6.2 task_harris_image_pair1

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair1/"
img1 = "1.jpg"
img2 = "2.jpg"

output_folder = "Results/pair1/"
```

```
task_harris(img1, img2, input_folder, output_folder)
```

6.3 task_harris_image_pair2

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair2/"
img1 = "truck1.jpg"
img2 = "truck2.jpg"

output_folder = "Results/pair2/"

task_harris(img1, img2, input_folder, output_folder)
```

6.4 task_harris_image_pair3

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair3/"
img1 = "1.jpg"
img2 = "2.jpg"

output_folder = "Results/pair3/"

task_harris(img1, img2, input_folder, output_folder)
```

6.5 task_harris_image_pair4

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair4/"
img1 = "1.jpg"
img2 = "2.jpg"

output_folder = "Results/pair4/"

task_harris(img1, img2, input_folder, output_folder)
```

6.6 task_sift_image_pair1.py

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair1/"
```

```
img1 = "1.jpg"
img2 = "2.jpg"

output_folder = "Results/pair1/"

task_sift(img1, img2, input_folder, output_folder)
```

6.7 task_sift_image_pair2.py

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair2/"
img1 = "truck1.jpg"
img2 = "truck2.jpg"

output_folder = "Results/pair2/"

task_sift(img1, img2, input_folder, output_folder)
```

6.8 task_sift_image_pair3.py

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair3/"
img1 = "1.jpg"
img2 = "2.jpg"

output_folder = "Results/pair3/"

task_sift(img1, img2, input_folder, output_folder)
```

6.9 task_sift_image_pair4.py

```
# Author: Gopikrishnan Sasi Kumar (Krish)

from library_hw4 import *

input_folder = "HW4Pics/pair4/"
img1 = "1.jpg"
img2 = "2.jpg"

output_folder = "Results/pair4/"

task_sift(img1, img2, input_folder, output_folder)
```