ECE 661 Computer Vision

Homework 4

Yi-Hong Liao | liao163@purdue.edu | PUID 0031850073

1. Logic

Implement Harris corner detector and use NCC and SSD to find the correspondences between to two sets of interest points of the image pairs. Use the OpenCV implementation of SIFT and SuperGlue to find the correspondences between to two sets of interest points of the image pairs.

2. Step

a. Harris corner detector

The corner is any pixel in the vicinity of which the gray levels show significant variations in at least two different directions. Calculate d_x and d_y by applying the Haar filters at scale σ to the image. Then construct the matrix

$$C = \begin{bmatrix} \sum d_x^2 & \sum d_x d_y \\ \sum d_x d_y & \sum d_y^2 \end{bmatrix}.$$

Where the summations are over all the pixels in the $5\sigma \times 5\sigma$ neighborhood.

The Harris response can be calculated as below,

$$R = det(C) - ktr(M)^2,$$

where $k = 0.04$. R can be directly thresholded for corner detection. In this work, the interest points with top 500 largest R are selected.

b. Finding correspondence using NCC

I find the correspondences between the image pair by calculating the NCC value between the interest point in the first image and all the interest points in the second image using the following criteria

$$NCC = \frac{\sum\sum(f_1(i,j)-m_1)(f_2(i,j)-m_2)}{\sqrt{[\sum\sum(f_1(i,j)-m_1)^2][\sum\sum(f_2(i,j)-m_2)^2]}},$$

where f_1 is the first image, f_2 is the second image, m_1 is the mean of f_1 and m_2 is the mean of f_2. The interesting point in the second image with largest NCC is the corresponding point of the interest point in the first image.

c. Finding correspondence using SSD

I find the correspondences between the image pair by calculating the SSD value between the interest point in the first image and all the interest points in the second image using the following criteria

$$SSD = \sum\sum||\mathrm{f}_1(\mathrm{i},\mathrm{j}) - \mathrm{f}_2(\mathrm{i},\mathrm{j})|^2,$$

where f_1 is the first image and f_2 is the second image. The interesting point in the second image with smallest SSD is the corresponding point of the interest point in the first image.

d. Finding correspondence using SIFT
In this work, I use the OpenCV implementation of SIFT. The step to extract the SIFT feature is
   i. Find all the local extrema in the DoG pyramid. The extema are found by comparing with its 26 neighbors in a $3 \times 3 \times 3$ "volumetric" neighborhood.
   ii. When σ increases in the DoG, localize the extremum in the true location in the original image by $\vec{x} = -H^{-1}(\overrightarrow{x_0})J(X_0)$.
   iii. Weed out the weak extrema by thresholding the DoG values.
   iv. Find the dominant local orientation with the gradient vector of the Gaussian-smoothed image.
   v. Generate the 128 dimensional SIFT descriptor. An 8-bin orientation histogram is calculated at the 16 pixels in each cell surrounding the extremum. Then string together the 16 8-bin histograms to yield a 128-element descriptor.

3. Result

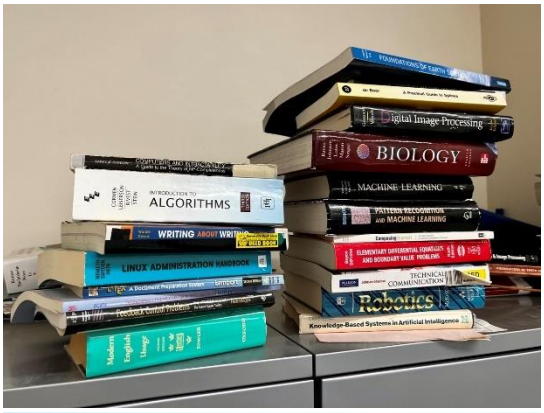
- ## Theory question

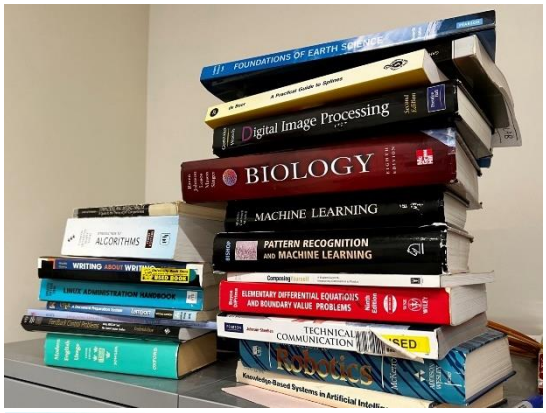
From the scale space theorem

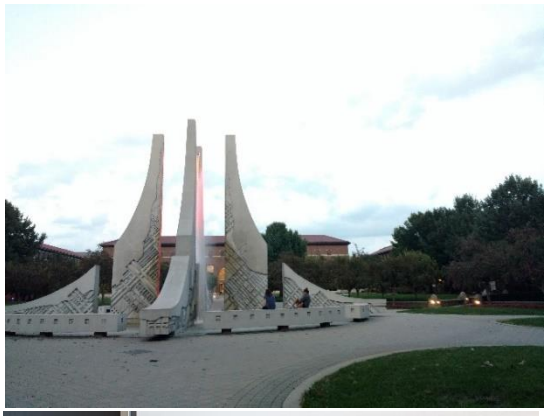
$$\sigma LoG\big(f(x,y)\big) = \frac{\partial}{\partial\sigma} ff(x,y,\sigma) \approxeq \frac{1}{\Delta\sigma}\big(ff(x,y,\sigma+\Delta\sigma) - ff(x,y,\sigma)\big)$$
$$\sigma\Delta\sigma LoG\big(f(x,y)\big) = \big(ff(x,y,\sigma+\Delta\sigma) - ff(x,y,\sigma)\big)$$

Since $\sigma\Delta\sigma$ is a constant that does not affect the local extrema, this implies that the LoG of an image $f(x,y)$ can be approximated by subtracting the $(\sigma+\Delta\sigma)$-smoothed version of $f(x,y)$ from the σ-smoothed version. The difference of two Gaussian-smoothed version of $f(x,y)$ is the Difference-of-Gaussian (DoG). Computing the LoG of an image as a DoG is computationally much more efficient is that the Gaussian kernel is separable. Therefore, instead of computing the 2D convolution, we can compute the 1D convolution in x and y direction and combine them.

- **Input images**

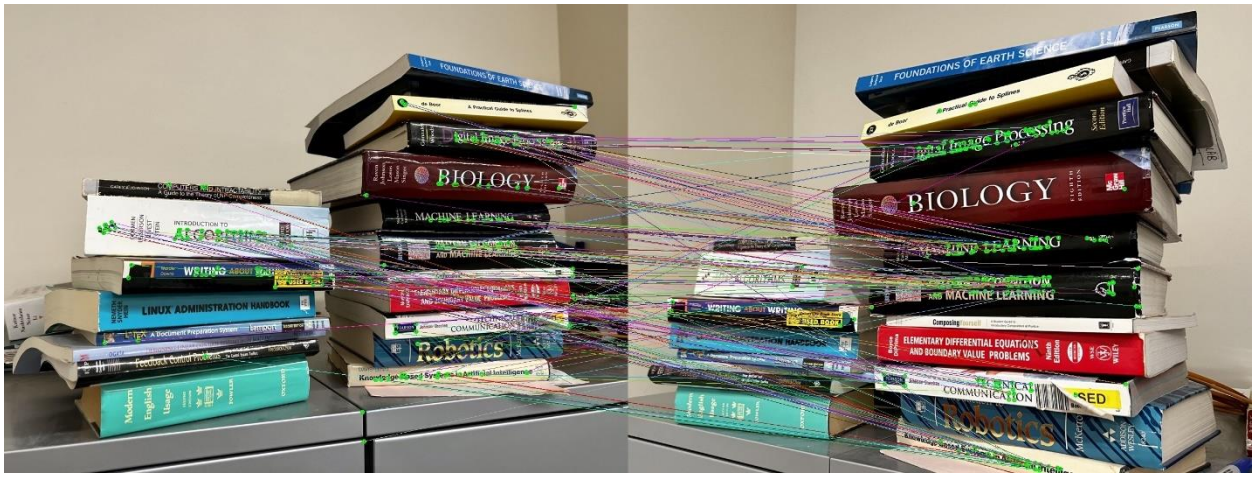


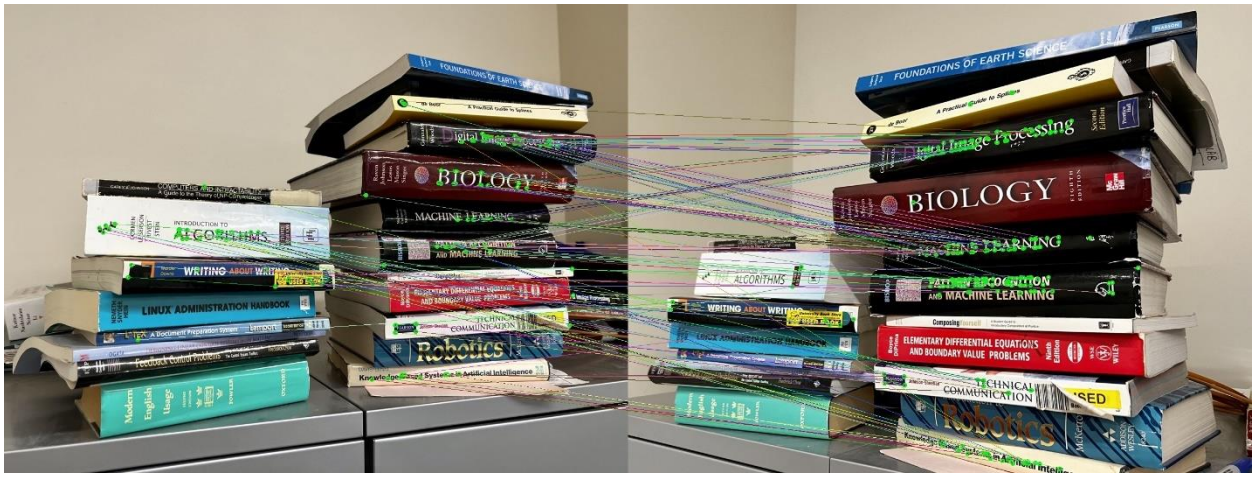


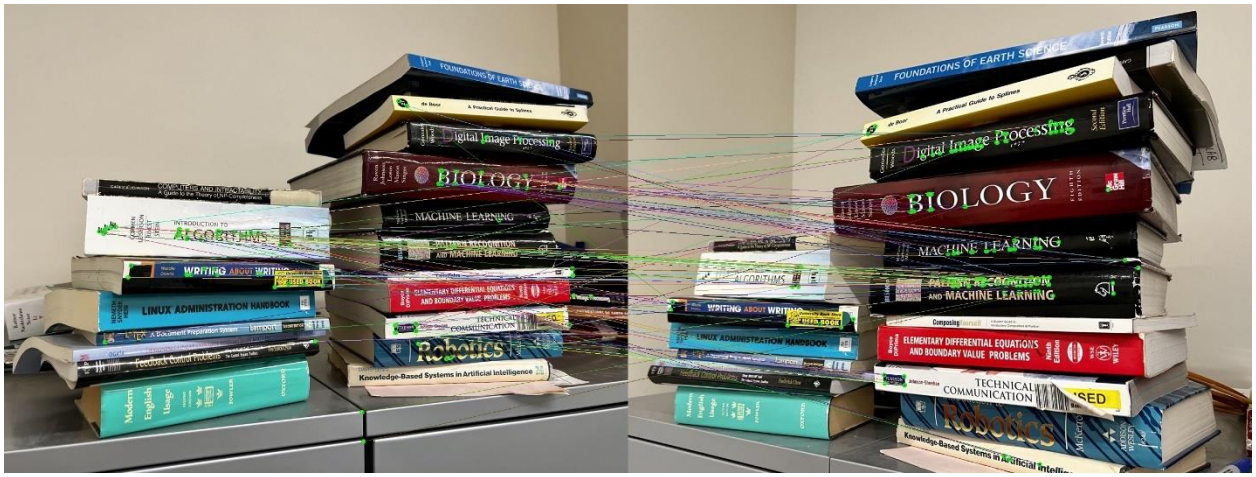


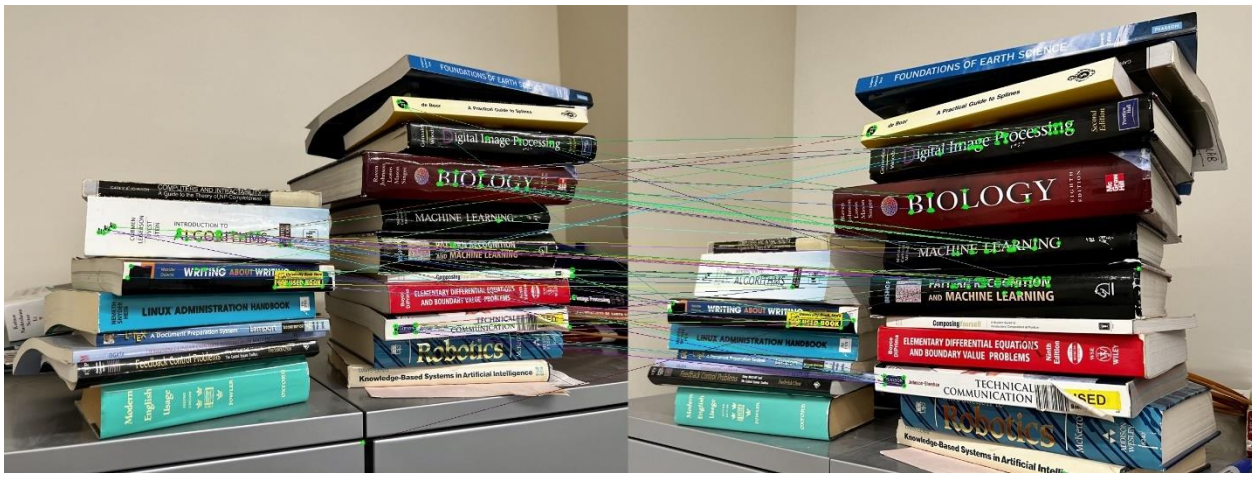
- **Harris corner detector**

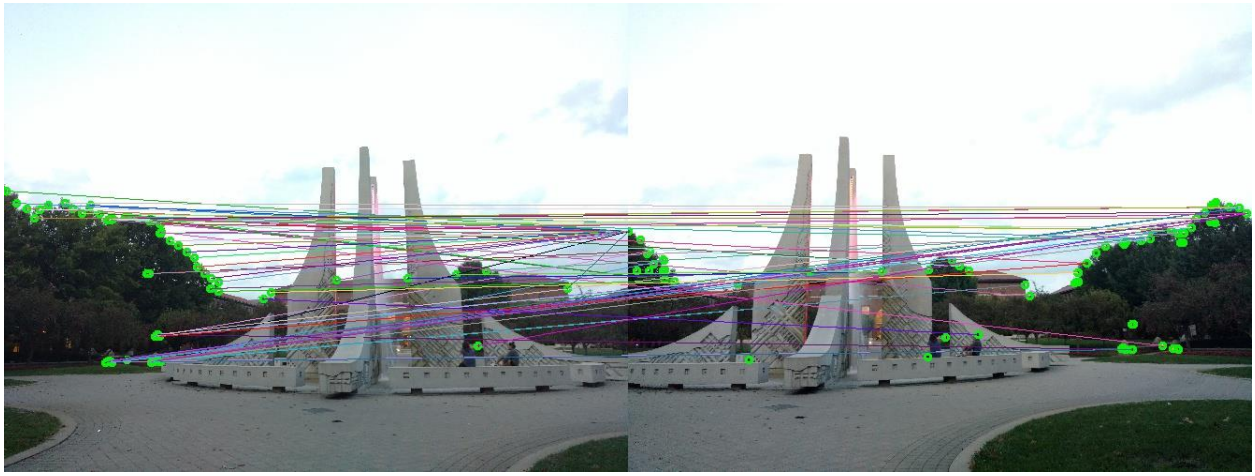
➢ NCC metric (window size: 7×7) with $\sigma = 0.8, 1.0, 1.2, 1.4$ from top to bottom for every image pairs

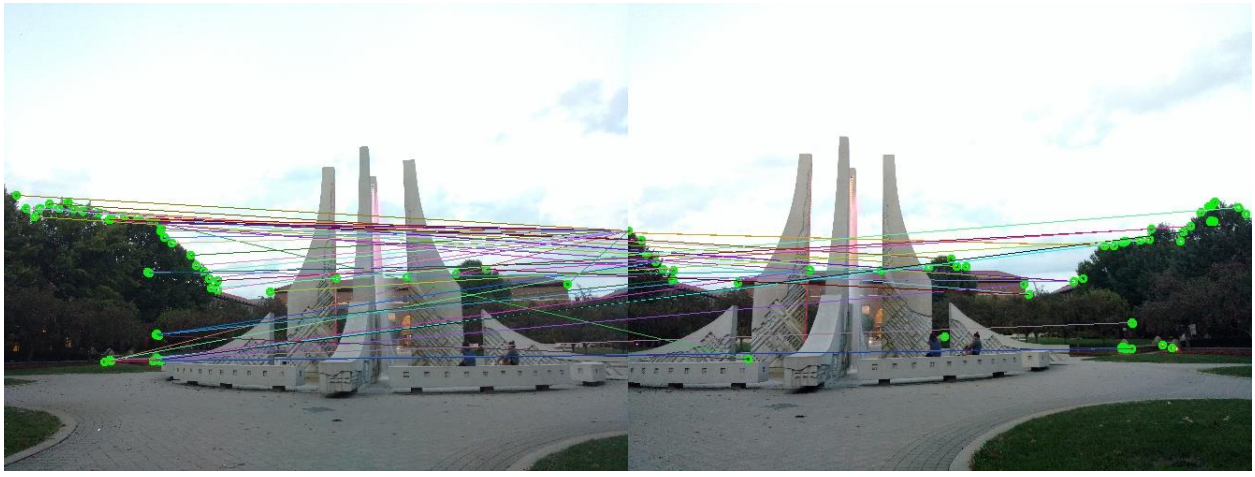


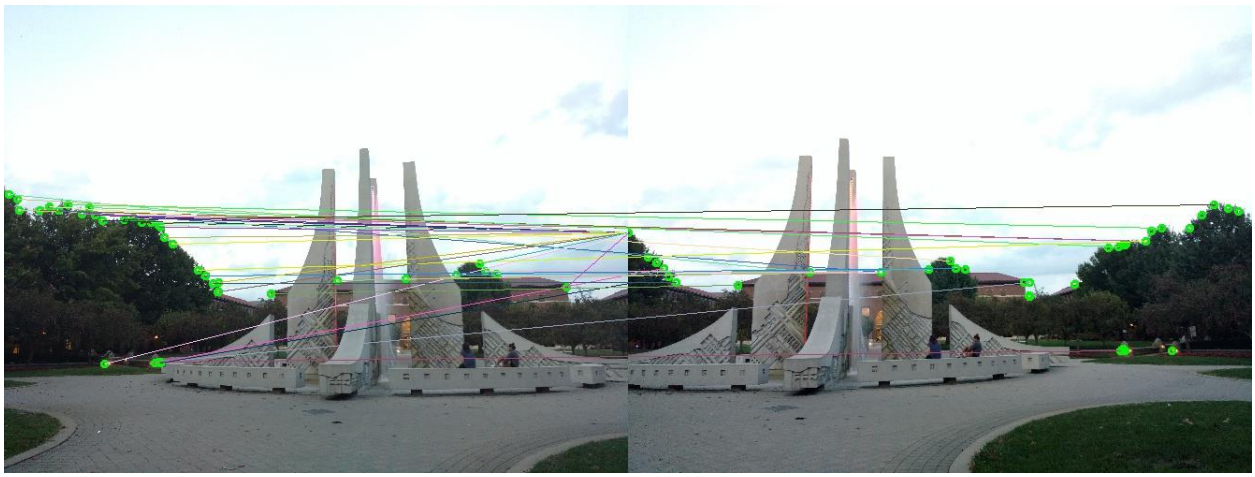


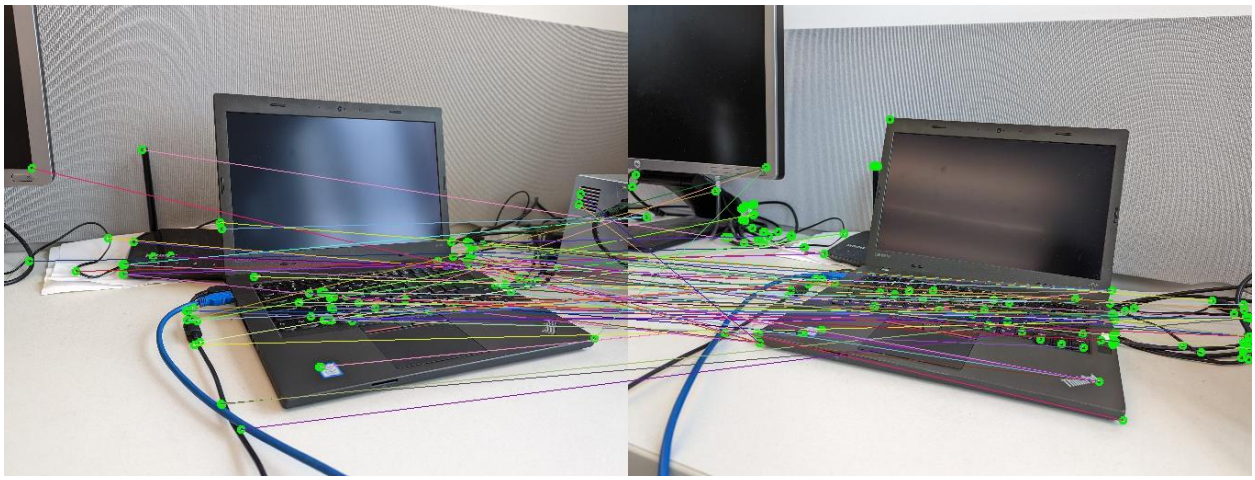


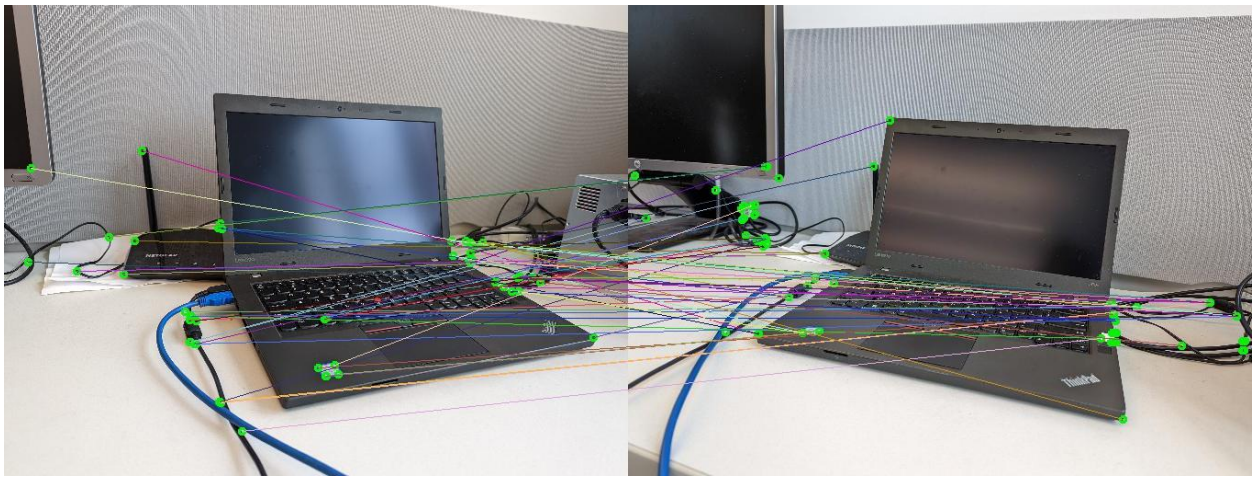

Digital Image Processing
BIOLOGY
MACHINE LEARNING
LINUX ADMINISTRATION HANDBOOK
Robotics
Knowledge-Based Systems in Artificial Intelligence
FOUNDATIONS OF EARTH SCIENCE
Digital Image Processing
BIOLOGY
MACHINE LEARNING
AND MACHINE LEARNING
ELEMENTARY DIFFERENTIAL EQUATIONS AND BOUNDARY VALUE PROBLEMS
TECHNICAL COMMUNICATION
Robotics

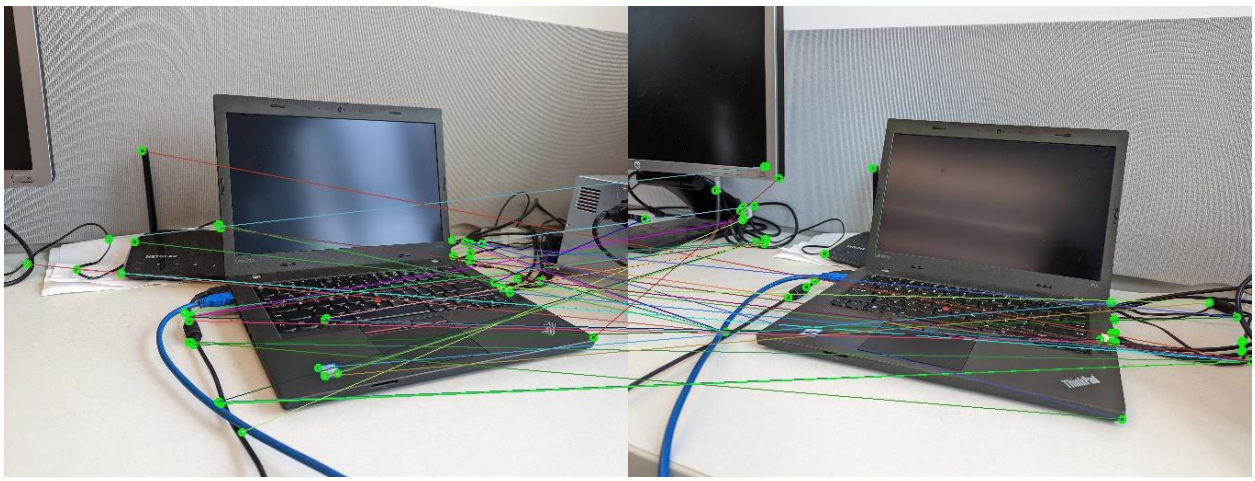


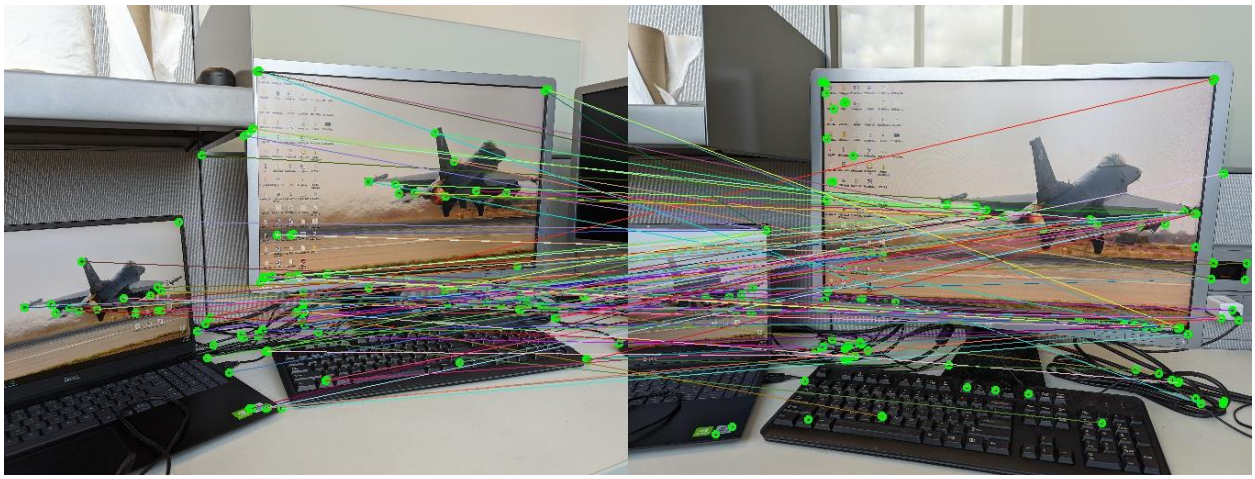


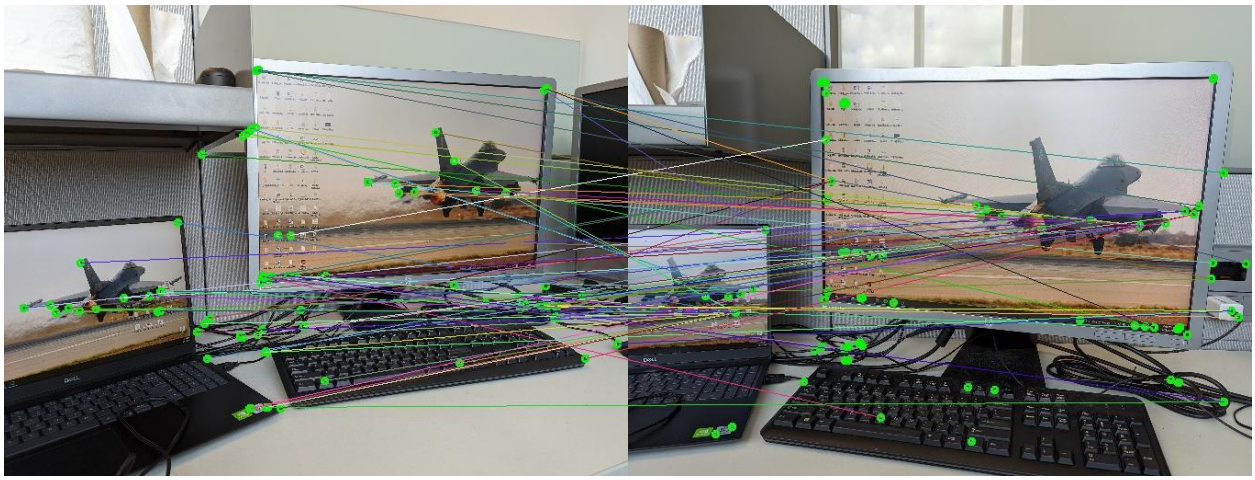


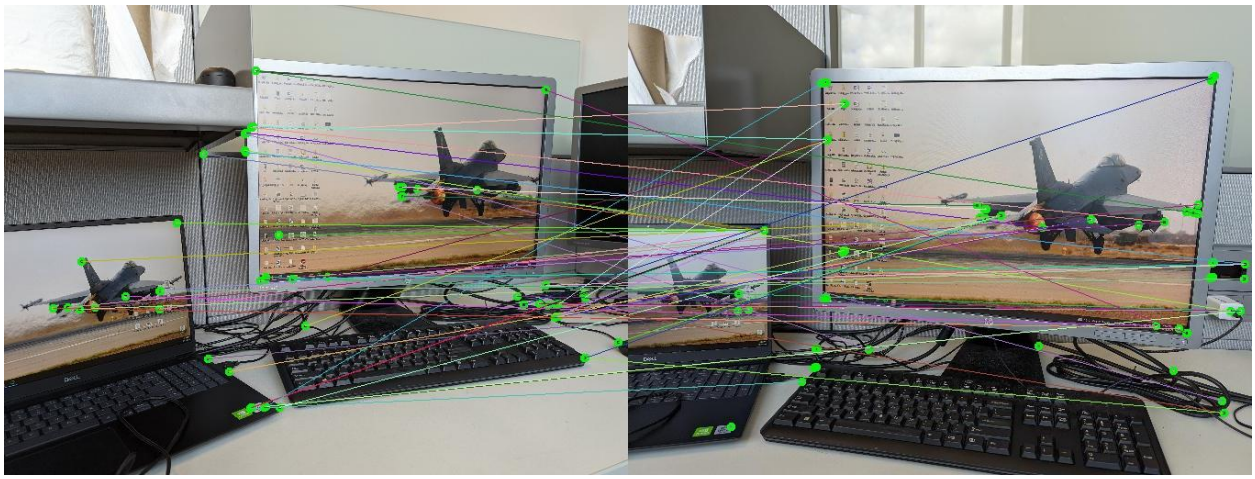






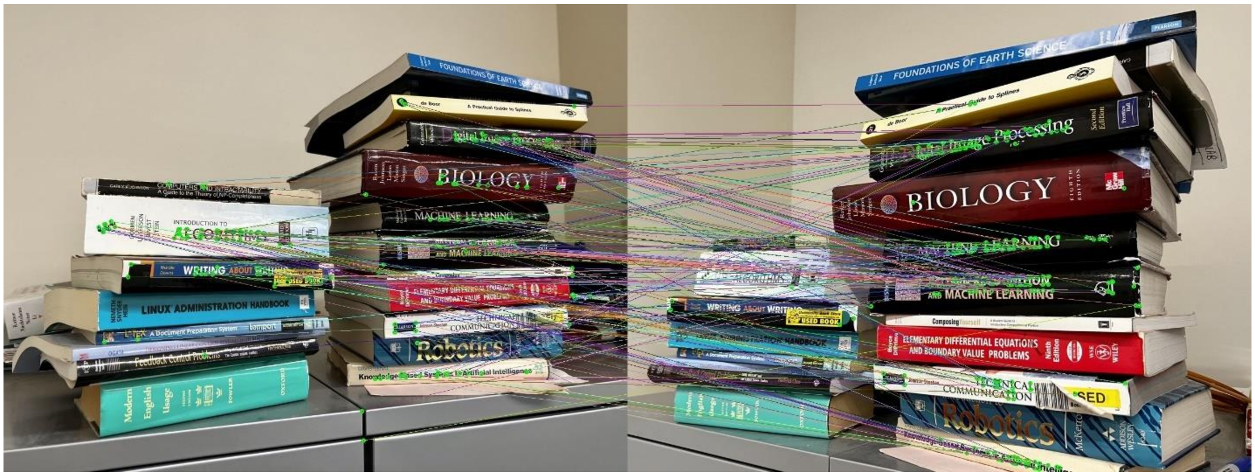


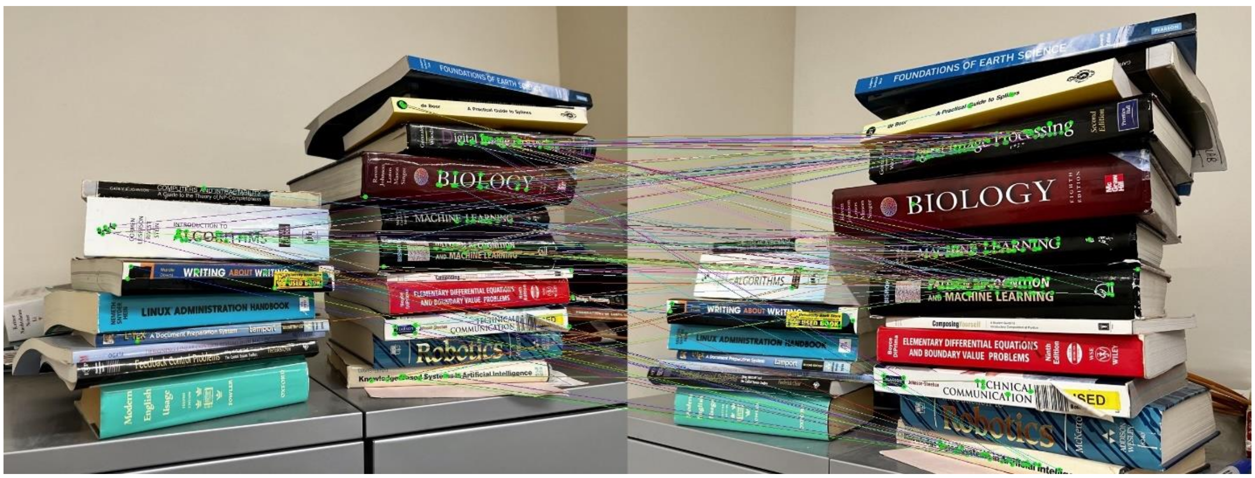


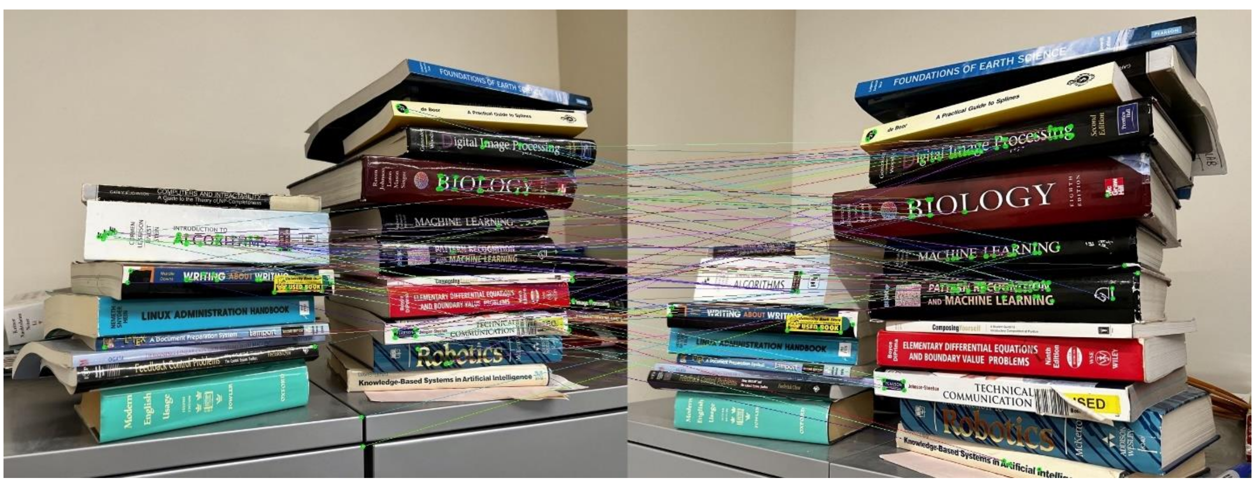


We can observe that when the σ value increases, the size of the corner that the Harris detector can detect decreased. This is because when σ increase the kernel size also increases.

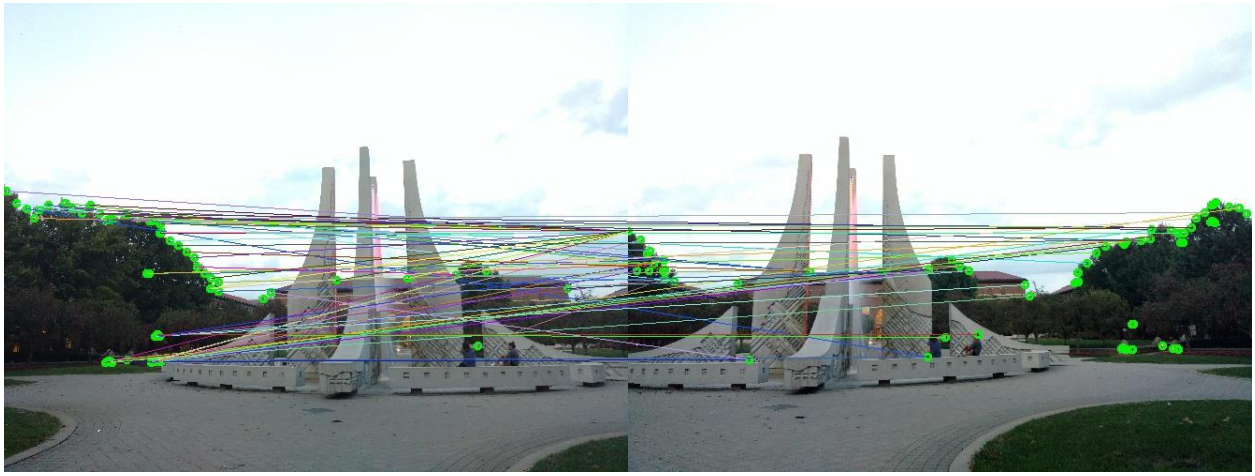
- SSD metric (window size: 7×7) with $\sigma = 0.8, 1.0, 1.2, 1.4$ from top to bottom for every image pairs

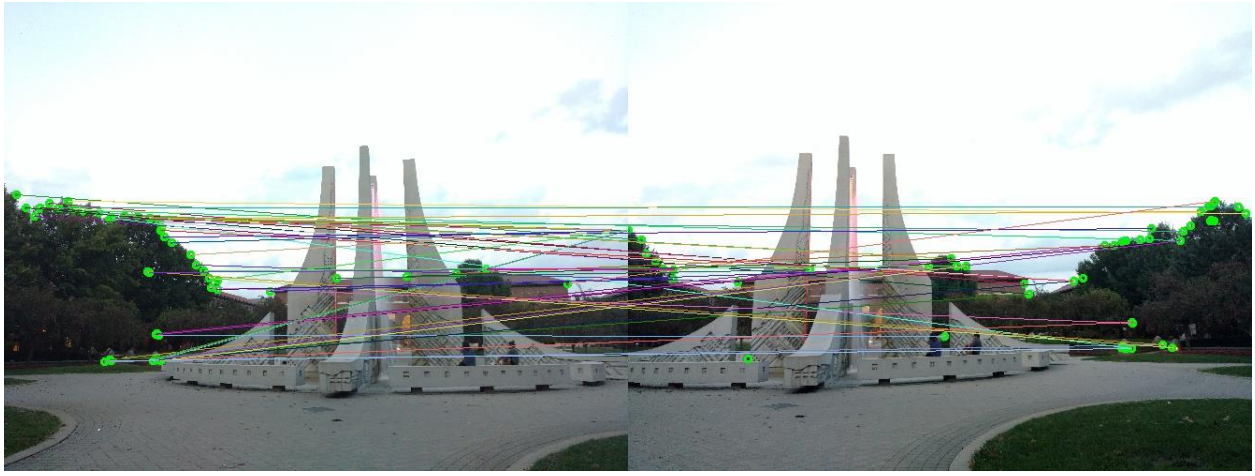


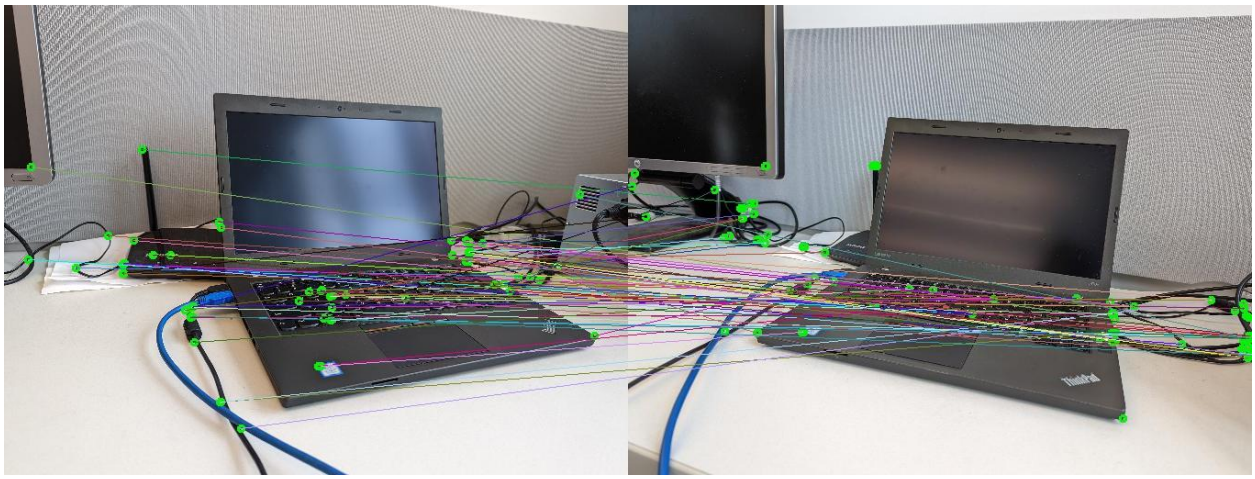


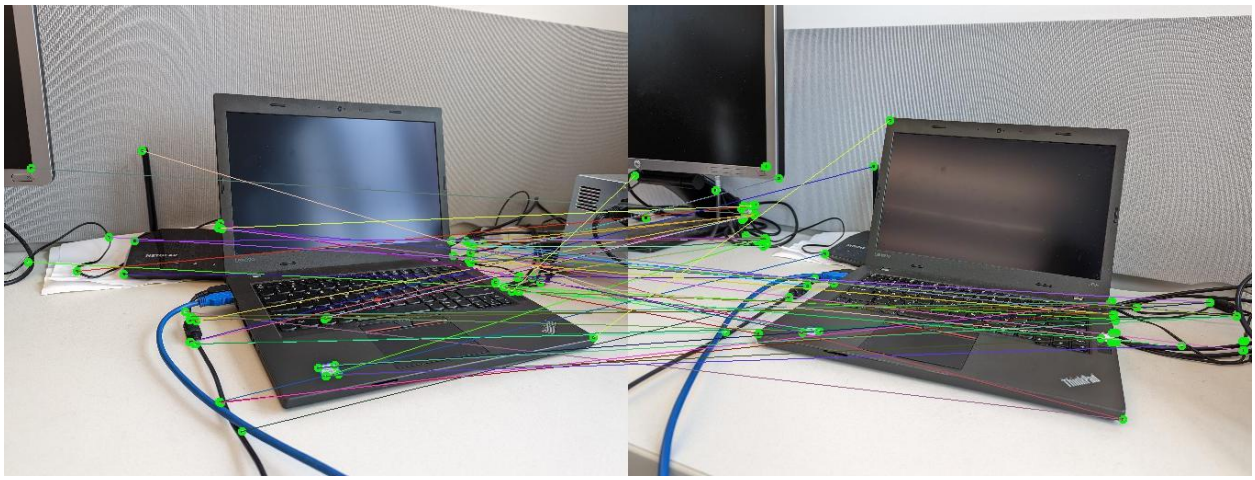


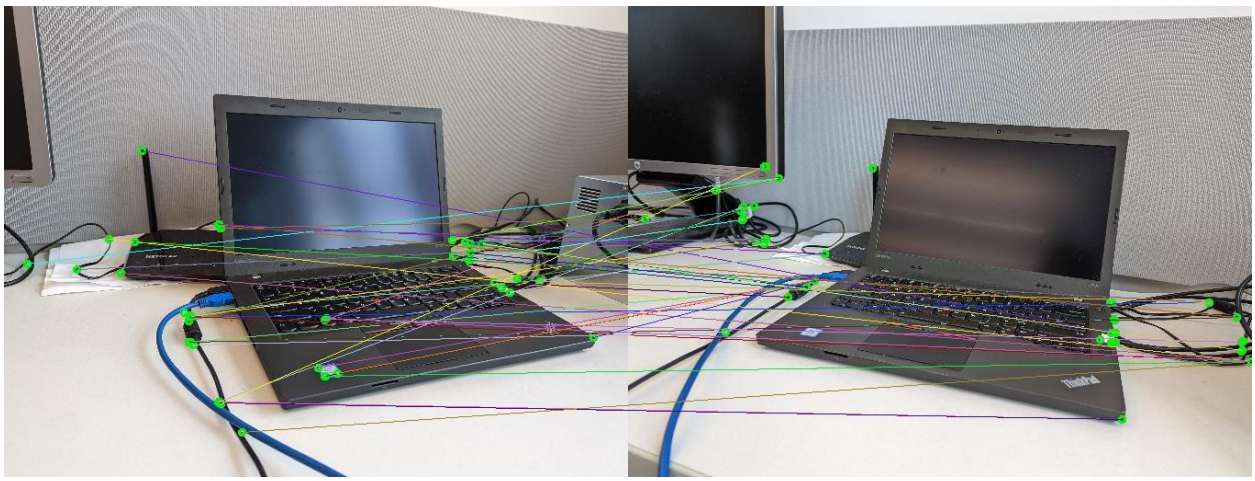

Digital Image Processing
BIOLOGY
MACHINE LEARNING
LINUX ADMINISTRATION HANDBOOK
Robotics
Knowledge-Based Systems in Artificial Intelligence
FOUNDATIONS OF EARTH SCIENCE
Digital Image Processing
BIOLOGY
MACHINE LEARNING
AND MACHINE LEARNING
WRITING ABOUT WRITING
LINUX ADMINISTRATION HANDBOOK
ELEMENTARY DIFFERENTIAL EQUATIONS AND BOUNDARY VALUE PROBLEMS
TECHNICAL COMMUNICATION
Robotics

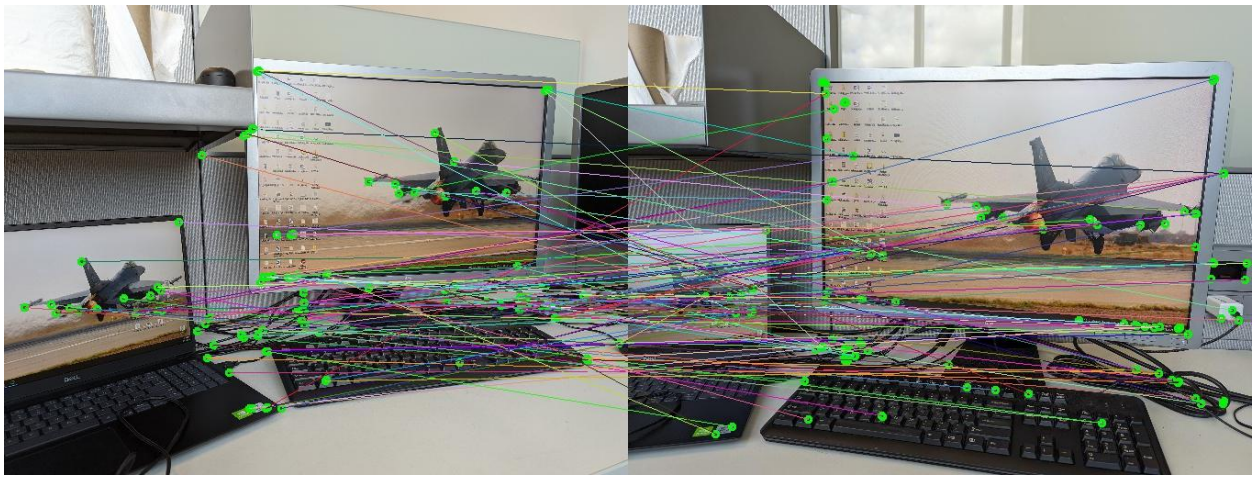


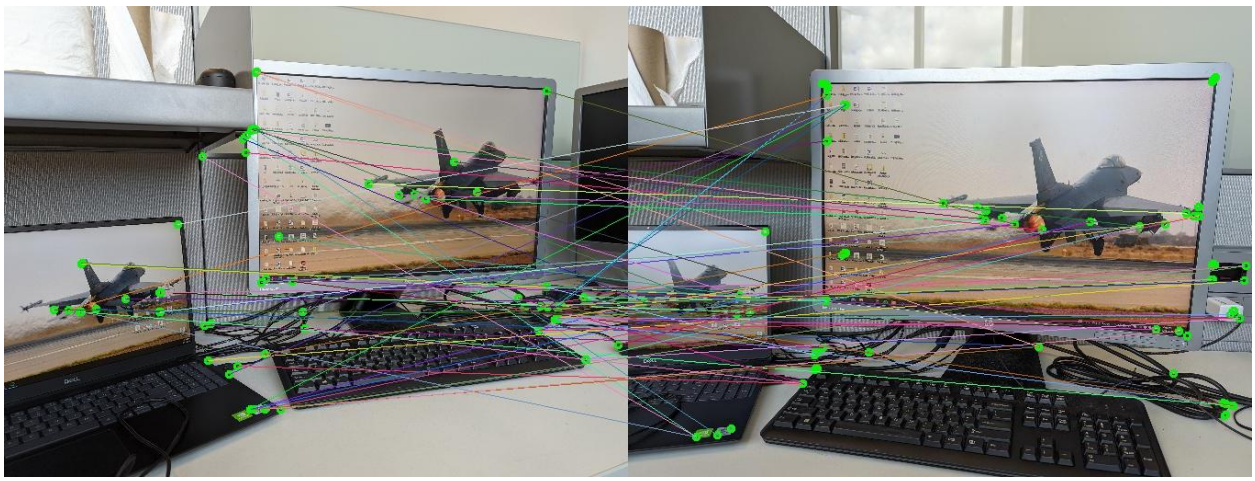


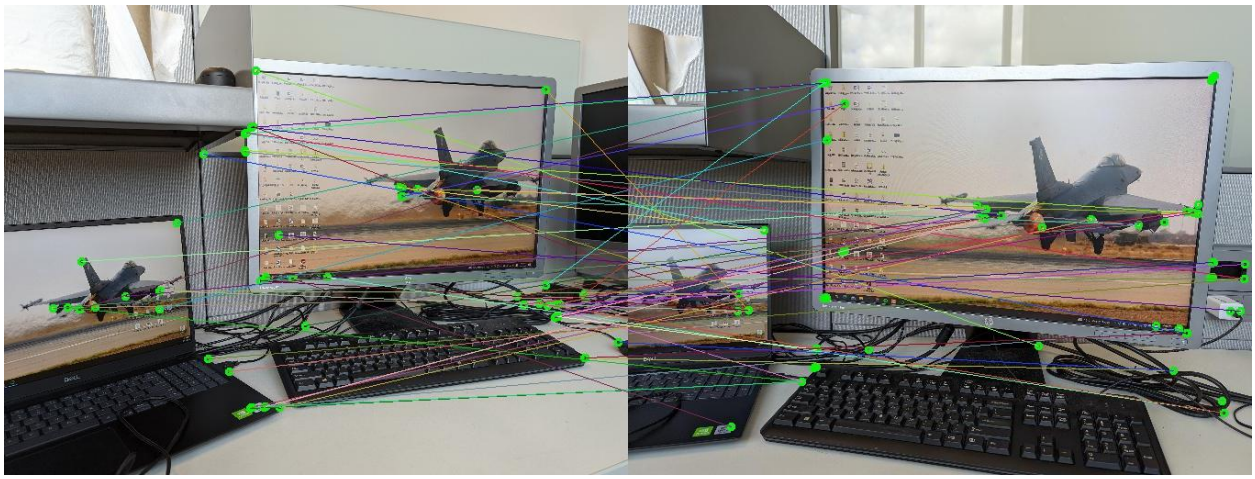






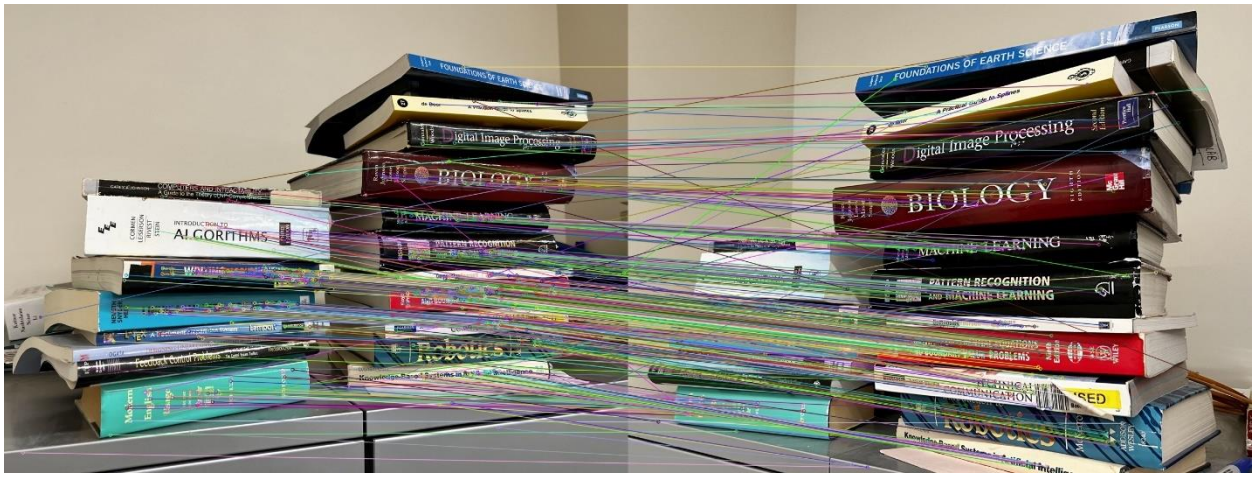


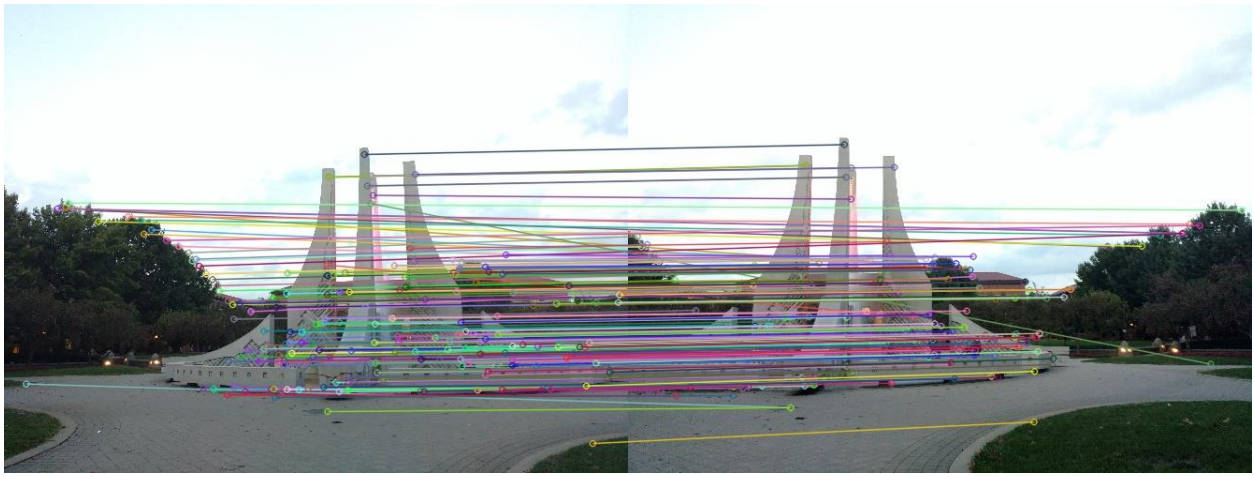


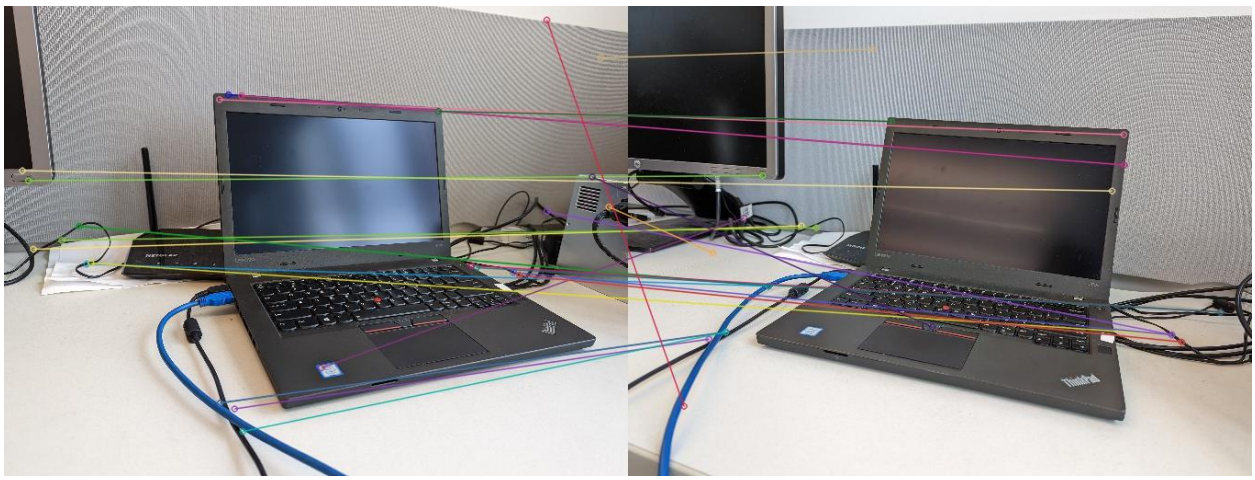


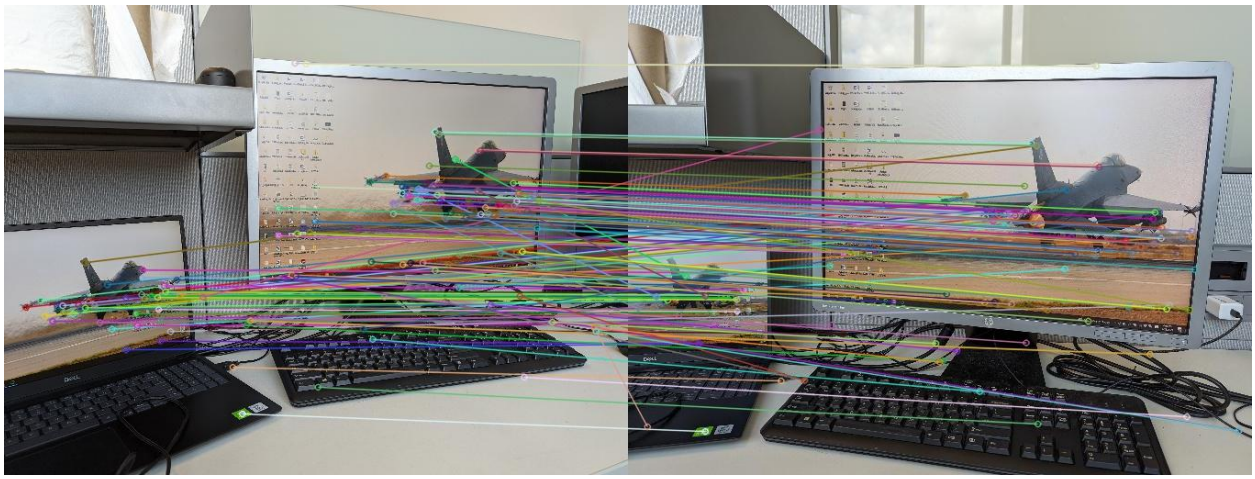
Same as we observed in the NCC case, when the σ value increases, the size of the corner that the Harris detector can detect decreased. Also, the SSD metric is more vulnerable to illumination change because it does not subtract the pixel value with the mean of the pixel in the matching window. The parameter I chose for giving the best result is NCC metric (window size: 7×7) with $\sigma = 1.2$.

- **SIFT**


FOUNDATIONS OF EARTH SCIENCE
Digital Image Processing
BIOLOGY
MACHINE LEARNING
PATTERN RECOGNITION
AND MACHINE LEARNING
INTRODUCTION TO
ALGORITHMS

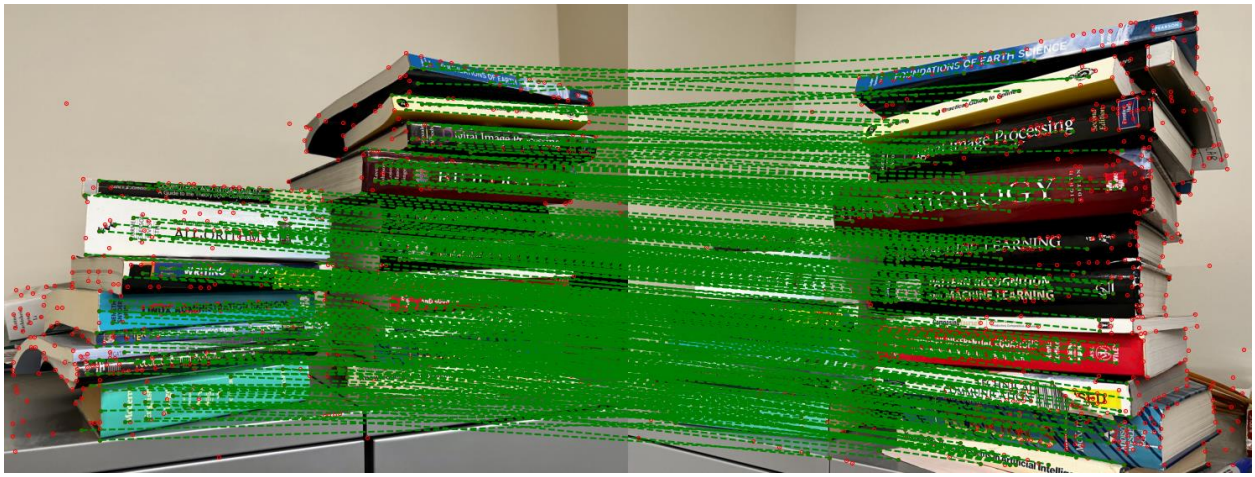


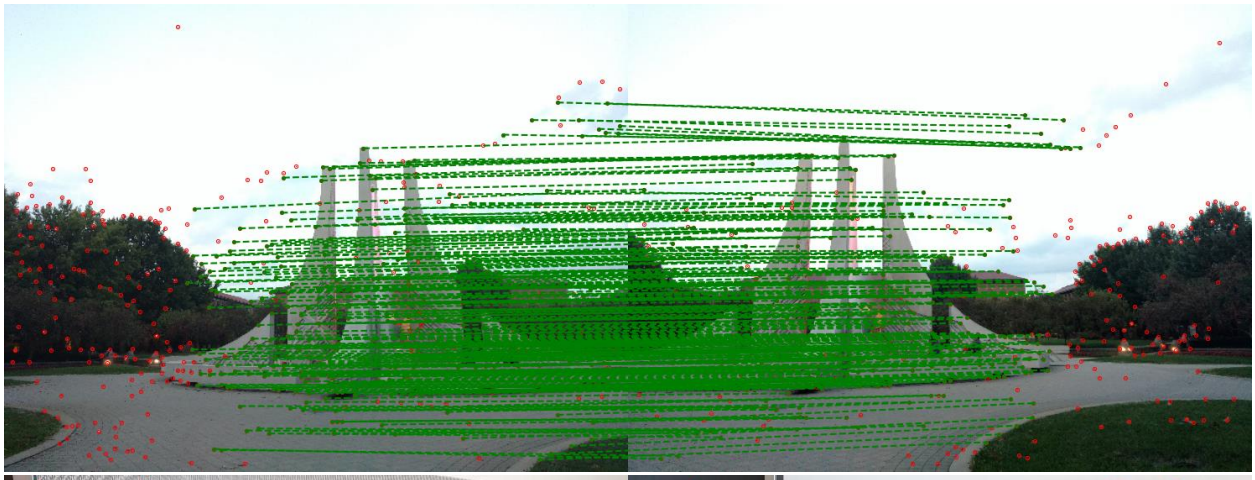


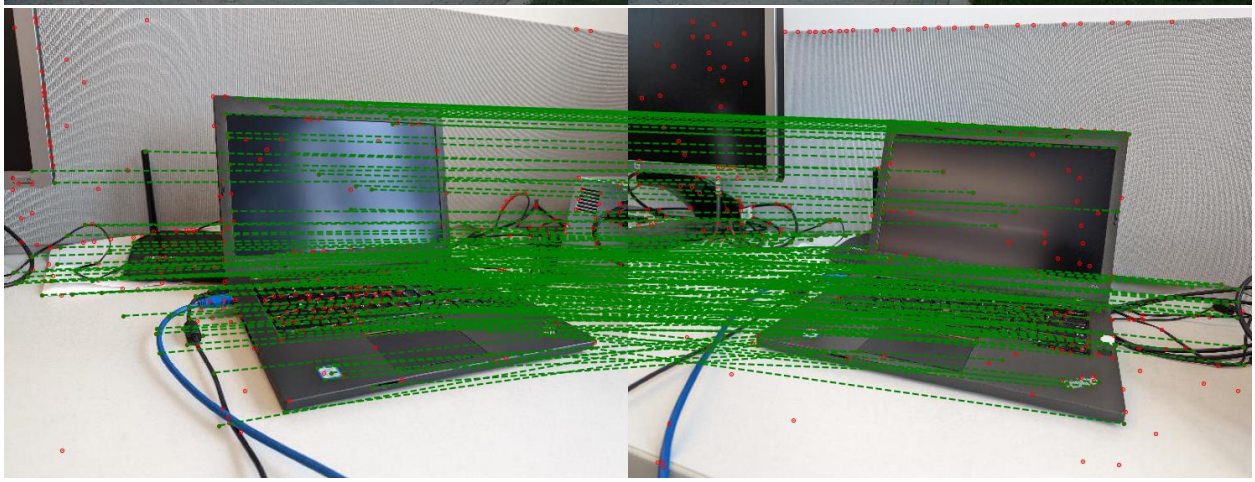


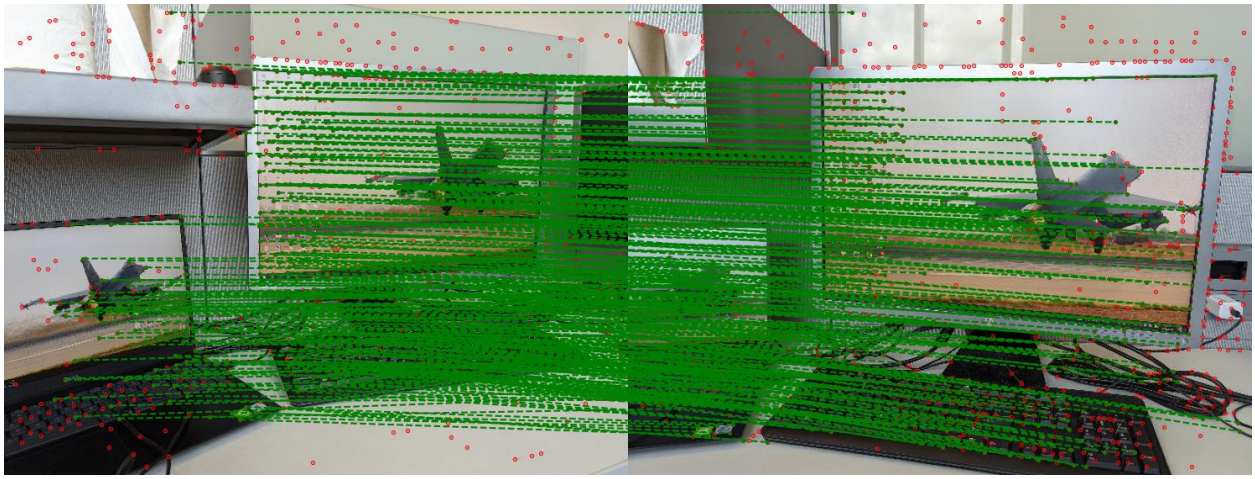
The SIFT method outperform the Harris detector in both interesting point finding and correspondence establishing.

- **SuperPoint and SuperGlue**









The SuperGlue has the best result for finding interest points and the correspondence among all the method (Harris and SIFT). The correct rate for the correspondence is higher than SIFT and much higher than Harris + NCC/SSD.

4. Source Code

```
#!/usr/bin/env python
# coding: utf-8

# In[45]:


import numpy as np
import matplotlib.pyplot as plt
import cv2


# In[58]:


def create_haar_kernel(scale):
    M = np.ceil(4*scale)
    M += np.mod(M, 2)
    M = int(M)
    Kdx = np.ones((M, M))
    Kdy = np.ones((M, M))
    Kdx[:, 0:int(M/2)] = -1
    Kdy[int(M/2):, :] = -1

    return Kdx, Kdy

def harris_detector(img, scale):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_out = img.copy()
    img_norm = img_gray/255

    M = np.ceil(4*scale)
```

```
M += np.mod(M, 2)

Kdx, Kdy = create_haar_kernel(scale)
dx = cv2.filter2D(img_norm, -1, kernel = Kdx)
dy = cv2.filter2D(img_norm, -1, kernel = Kdy)

dx_sqr = dx * dx
dxdy = dx * dy
dy_sqr = dy * dy

Ksum = np.ones((int(np.ceil(5*scale)), int(np.ceil(5*scale))))

dx_sqr_sum = cv2.filter2D(dx_sqr, -1, kernel = Ksum)
dxdy_sum = cv2.filter2D(dxdy, -1, kernel = Ksum)
dy_sqr_sum = cv2.filter2D(dy_sqr, -1, kernel = Ksum)

trace = dx_sqr_sum + dy_sqr_sum
det = dx_sqr_sum*dy_sqr_sum - dxdy_sum*dxdy_sum

k=0.04

R = det - k*trace**2
R_threshold = np.sort(R.flatten())[-500]

corners = []
dist = 10
prev_cor = [-dist, -dist]

for i in range(0, img.shape[1]):
    for j in range(0, img.shape[0]):
        if R[j, i] > R_threshold:
            if len(corners) != 0:
```

```
                prev_cor = corners[-1]
            if ((i-prev_cor[0])**2+(j-prev_cor[1])**2) >= dist**2:
                corners.append([i, j])
                cv2.circle(img_out,(i, j), 3, (10,240,10), 2)

    return corners, img_out


def calculate_NCC(point1, img1, point2, img2, win_size):
    h = img1.shape[0]
    w = img1.shape[1]
    hwin_size = int(np.floor(win_size/2))


    img1p = cv2.copyMakeBorder(img1, hwin_size, hwin_size, hwin_size, hwin_size,
cv2.BORDER_CONSTANT, 0);
    img2p = cv2.copyMakeBorder(img2, hwin_size, hwin_size, hwin_size, hwin_size,
cv2.BORDER_CONSTANT, 0);


    sec1 = img1p[point1[1]:point1[1]+win_size, point1[0]:point1[0]+win_size]
    sec2 = img2p[point2[1]:point2[1]+win_size, point2[0]:point2[0]+win_size]
    m1 = np.mean(sec1)
    m2 = np.mean(sec2)


    sec1m = sec1 - m1
    sec2m = sec2 - m2


    NCC = np.sum(sec1m*sec2m)/np.sqrt(np.sum(sec1m**2)*np.sum(sec2m**2))


    return NCC


def calculate_SSD(point1, img1, point2, img2, win_size):
    h = img1.shape[0]
    w = img1.shape[1]
    hwin_size = int(np.floor(win_size/2))
```

```
    img1p = cv2.copyMakeBorder(img1, hwin_size, hwin_size, hwin_size, hwin_size,
cv2.BORDER_CONSTANT, 0);
    img2p = cv2.copyMakeBorder(img2, hwin_size, hwin_size, hwin_size, hwin_size,
cv2.BORDER_CONSTANT, 0);


    sec1 = img1p[point1[1]:point1[1]+win_size, point1[0]:point1[0]+win_size]
    sec2 = img2p[point2[1]:point2[1]+win_size, point2[0]:point2[0]+win_size]


    SSD = np.sum((sec1-sec2)**2)


    return SSD



def find_NCC_correspondence(corners1, img1, corners2, img2, win_size):
    img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    corr_corners = []
    for corner1 in corners1:
        max_NCC = -1000
        for corner2 in corners2:
            NCC = calculate_NCC(corner1, img1_gray, corner2, img2_gray, win_size)
            if NCC > max_NCC:
                max_NCC = NCC
                corr_corner = corner2
        corr_corners.append(corr_corner)


    return corr_corners


def find_SSD_correspondence(corners1, img1, corners2, img2, win_size):
    img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    corr_corners = []
```

```
    for corner1 in corners1:
        min_SSD = (255**2)*(win_size**2)+1
        for corner2 in corners2:
            SSD = calculate_SSD(corner1, img1_gray, corner2, img2_gray, win_size)
            if SSD < min_SSD:
                min_SSD = SSD
                corr_corner = corner2
        corr_corners.append(corr_corner)

    return corr_corners

def draw_correspondence(corners1, img1, corners2, img2):
    w = img1.shape[1]
    img = np.concatenate((img1, img2), axis=1)
    for i in range(0, len(corners1)):
        color = np.random.randint(0, 255, size=(3, ))
        #convert data types int64 to int
        color = (int(color[0]), int(color[1]), int(color[2]))
        cv2.line(img, tuple(corners1[i]), (corners2[i][0]+w, corners2[i][1]), color, 1)
    return img

def SIFT_detector(img1, img2):
    # This code is modified from OpenCV document:
    # https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

    img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Initiate SIFT detector
    sift = cv2.SIFT_create()

    # find the keypoints and descriptors with SIFT
```

```
    kp1, des1 = sift.detectAndCompute(img1_gray, None)
    kp2, des2 = sift.detectAndCompute(img2_gray, None)

    # BFMatcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)

    # Apply ratio test
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append([m])

    # cv.drawMatchesKnn expects list of lists as matches.
    img3 =
cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv2.DrawMatchesFlags_NOT_
DRAW_SINGLE_POINTS)
    return img3




# In[52]:



if __name__ == '__main__':
    path = r'C:\Users\yhosc\Desktop\ECE661\HW4\HW4-Images\Figures\\'
    outputPath = r'C:\Users\yhosc\Desktop\ECE661\HW4\\'

    # Reading images in default mode
    img_books1 = cv2.imread(path+'books_1.jpeg')
    img_books2 = cv2.imread(path+'books_2.jpeg')
    img_fountain1 = cv2.imread(path+'fountain_1.jpg')
    img_fountain2 = cv2.imread(path+'fountain_2.jpg')
```

```
    img_laptop1 = cv2.imread(path+'laptop_1.jpg')
    img_laptop2 = cv2.imread(path+'laptop_2.jpg')
    img_monitor1 = cv2.imread(path+'monitor_1.jpg')
    img_monitor2 = cv2.imread(path+'monitor_2.jpg')


    scales = [0.8, 1.0, 1.2, 1.4]



# In[48]:



for scale in scales:
    corners_books1, img_books1_c = harris_detector(img_books1, scale)
    corners_books2, img_books2_c = harris_detector(img_books2, scale)


    corners_books1_corr_NCC = find_NCC_correspondence(corners_books1, img_books1,
corners_books2, img_books2, 7)
    img_books_c_NCC = draw_correspondence(corners_books1, img_books1_c,
corners_books1_corr_NCC, img_books2_c)
    cv2.imwrite(outputPath+f'books_NCC_{scale}.jpg', img_books_c_NCC)


    corners_books1_corr_SSD = find_SSD_correspondence(corners_books1, img_books1,
corners_books2, img_books2, 7)
    img_books_c_SSD = draw_correspondence(corners_books1, img_books1_c,
corners_books1_corr_SSD, img_books2_c)
    cv2.imwrite(outputPath+f'books_SSD_{scale}.jpg', img_books_c_SSD)


img_books_c_SIFT = SIFT_detector(img_books1, img_books2)
cv2.imwrite(outputPath+'books_SIFT.jpg', img_books_c_SIFT)



# In[50]:
```

```
for scale in scales:
    corners_fountain1, img_fountain1_c = harris_detector(img_fountain1, scale)
    corners_fountain2, img_fountain2_c = harris_detector(img_fountain2, scale)

    corners_fountain1_corr_NCC = find_NCC_correspondence(corners_fountain1, img_fountain1, corners_fountain2, img_fountain2_c, 7)
    img_fountain_c_NCC = draw_correspondence(corners_fountain1, img_fountain1_c, corners_fountain1_corr_NCC, img_fountain2_c)
    cv2.imwrite(outputPath+f'fountain_NCC_{scale}.jpg', img_fountain_c_NCC)

    corners_fountain1_corr_SSD = find_SSD_correspondence(corners_fountain1, img_fountain1, corners_fountain2, img_fountain2_c, 7)
    img_fountain_c_SSD = draw_correspondence(corners_fountain1, img_fountain1_c, corners_fountain1_corr_SSD, img_fountain2_c)
    cv2.imwrite(outputPath+f'fountain_SSD_{scale}.jpg', img_fountain_c_SSD)

img_fountain_c_SIFT = SIFT_detector(img_fountain1, img_fountain2)
cv2.imwrite(outputPath+'fountain_SIFT.jpg', img_fountain_c_SIFT)

# In[54]:

for scale in scales:
    corners_laptop1, img_laptop1_c = harris_detector(img_laptop1, scale)
    corners_laptop2, img_laptop2_c = harris_detector(img_laptop2, scale)

    corners_laptop1_corr_NCC = find_NCC_correspondence(corners_laptop1, img_laptop1, corners_laptop2, img_laptop2_c, 7)
    img_laptop_c_NCC = draw_correspondence(corners_laptop1, img_laptop1_c, corners_laptop1_corr_NCC, img_laptop2_c)
    cv2.imwrite(outputPath+f'laptop_NCC_{scale}.jpg', img_laptop_c_NCC)

    corners_laptop1_corr_SSD = find_SSD_correspondence(corners_laptop1, img_laptop1, corners_laptop2, img_laptop2_c, 7)
```

```
    img_laptop_c_SSD = draw_correspondence(corners_laptop1, img_laptop1_c, corners_laptop1_corr_SSD, img_laptop2_c)
    cv2.imwrite(outputPath+f'laptop_SSD_{scale}.jpg', img_laptop_c_SSD)

img_laptop_c_SIFT = SIFT_detector(img_laptop1, img_laptop2)
cv2.imwrite(outputPath+'laptop_SIFT.jpg', img_laptop_c_SIFT)

# In[55]:

for scale in scales:
    corners_monitor1, img_monitor1_c = harris_detector(img_monitor1, scale)
    corners_monitor2, img_monitor2_c = harris_detector(img_monitor2, scale)

    corners_monitor1_corr_NCC = find_NCC_correspondence(corners_monitor1, img_monitor1, corners_monitor2, img_monitor2_c, 7)
    img_monitor_c_NCC = draw_correspondence(corners_monitor1, img_monitor1_c, corners_monitor1_corr_NCC, img_monitor2_c)
    cv2.imwrite(outputPath+f'monitor_NCC_{scale}.jpg', img_monitor_c_NCC)

    corners_monitor1_corr_SSD = find_SSD_correspondence(corners_monitor1, img_monitor1, corners_monitor2, img_monitor2_c, 7)
    img_monitor_c_SSD = draw_correspondence(corners_monitor1, img_monitor1_c, corners_monitor1_corr_SSD, img_monitor2_c)
    cv2.imwrite(outputPath+f'monitor_SSD_{scale}.jpg', img_monitor_c_SSD)

img_monitor_c_SIFT = SIFT_detector(img_monitor1, img_monitor2)
cv2.imwrite(outputPath+'monitor_SIFT.jpg', img_monitor_c_SIFT)
```