ECE 66100 Computer Vision **Name:** *Nikita Ravi (ravi30)*
Homework 4
09/21/2022

1 Theory Question

1. What is the theoretical reason for why the LoG of an image can be computed as a DoG?

   Answer: LoG stands for *Laplacian-of-Gaussian.* For LoG, the smoothing consists of convolving the image $f(x, y)$ with a Gaussian function $g(x, y)$

   $$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   which is dependent on the scale factor σ. The convolution of the image with the Gaussian function is as follows:

   $$f'(x, y, \sigma) = \iint f(x', y') g_\sigma(x - x', y - y') dx' dy'$$

   And, the derivative operation consists of applying the isotropic Laplacian operator

   $$\nabla^2 = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2}$$

   to the smoothed data. The edges correspond to the zero-crossings of the Laplacian operator output.

   By taking the partial derivative of $f'(x, y, \sigma)$ with respect to σ, we get the following result

   $$\frac{\delta}{\delta\sigma} f'(x, y, \sigma) = \sigma \nabla^2 f'(x, y, \sigma)$$

   Since $\nabla^2 f'(x, y, \sigma)$ is the LoG of $f(x, y, \sigma)$, this implies that the LoG of an image $f(x, y)$ can be approximated by subtracting the $(\delta + \delta\sigma)$-smoothed version of $f(x, y)$ from the σ-smoothed version. Finding this difference is referred to as the *Difference-of-Gaussian (DoG)*

2. Also explain in your own words why computing the LoG of an image as a DoG is computationally much more efficient for the same value of σ?

   Answer: It is much faster to calculate DoG with σ_1 and σ_2 for the two Gaussian smoothings than to calculate LoG for any given σ for two reasons. The first reason is because the Gaussian function $g(x, y)$ is separable in x and y so each of the 2D smoothings for DoG can be carried out by two applications of 1D smoothings, along x then along y, whereas the LoG operator is not separable so it must carry out computations as a 2D convolution. The second reason is, for the same σ, the DoG operator has a smaller kernel size than the LoG operator (for example if $\sigma = \sqrt{2}$, then the kernel size for the DoG operator is 9×9 whereas LoG would be 13×13) so therefore is more computationally efficient.

2 Programming Task

2.1 Theoretical Background

2.1.1 Harris Corner Detection

The Harris Corner Detection is an algorithm that predates the SIFT and SURF algorithms to identify interest points. A corner is represented by any pixel in the vicinity of which the gray levels show significant variations in at least two different variations. In other words, the algorithm identifies key points by using the x-and y-gradients, denoted as dx and dy. The dx and dy are the convolution outputs of the x- and y-oriented Haar filters at scale σ and the image $f(x, y)$ we are examining. The larger the σ, the larger the scale at which the gray levels must change in order to be detectable by the operators. The dimension of the Haar filter is $N \times N$ where N is the smallest even integer greater than 4σ. For example, if $\sigma = 1.8$, then $N = 8$, so the Haar filters would be:

$$h_x = \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix} \text{ and } h_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

So dx and dy are calculated as follows

$$dx = h_x * f(x, y) \text{ and } dy = h_y * f(x, y)$$

After this, we then construct a matrix C of dimension $5\sigma \times 5\sigma$ neighborhood of the pixel where we evaluate the presence or absence of a corner, as shown below

$$C = \begin{bmatrix} \sum dx^2 & \sum dxdy \\ \sum dxdy & \sum dy^2 \end{bmatrix}$$

When the eigenvalues of C, λ_1 and λ_2 are low, it represents a flat surface in a corner space. When one of them is high, then it represents an edge. When both of the eigenvalues are high, then it represents a corner. This is because a small change in any one direction will result in a large change when we are sliding the window in corner as shown in Figure 1 [1].

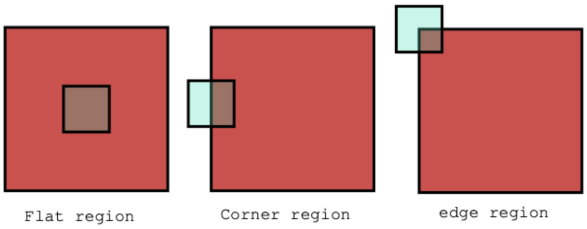
The ratio r which is $\frac{\lambda_2}{\lambda_1}$ can be computed efficiently without an explicit eigendecomposition of C by dividing the determinant of C by the square of trace of C. This ratio can be directly thresholded for corner detection.

The Harris Response function R is used to determine whether there is a corner on the current pixel we are examining,

$$R = det(C) - r[Tr(C)]^2$$

For this assignment, the threshold was set to preserve the top 10% of Rs. Non-maximum suppression was then conducted to remove excessive points since multiple pixels could be detected in the same corner.

Figure 1: Harris Corner and Edge Detection [1]



2.1.2 Establishing Correspondences using SSD and NCC Methods

When scale-space operators like Harris Corner Detection, SIFT, and SURF are used to extract key points from two images of a specific scene from different perspectives, the key points in the two images can be compared to each other by measuring the distance between their respective descriptive vectors in a neighborhood (chosen to be of dimension 21×21) around each pair of corner points. The two distance metrics used are Sum of Squared Differences (SSD)

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

and Normalized Cross Correlation (NCC)

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

where $f(x,y)$ is the image and $m_{\{1,2\}}$ is the mean of the image.

2.1.3 Overview of the SIFT Algorithm

SIFT stands for Scale Invariant Feature Transform. The steps involved in the SIFT algorithm are as follows

1. The first step is to construct the DoG Pyramid. This is accomplished by convolving the image against a Gaussian function with an initial scale σ_0. To go to the next octave we downsample the image and double the initial scale and we keep repeating this process. We then define a within-octave scale sampling interval $\Delta\sigma = \frac{\sigma_b}{2^s}$ where σ_b is the value of σ at the base of the octave and where s is a user-specified integer. We convolve the resolution at that octave against the Gaussian function with this within-octave sampling without changing the resolution at that octave. Finally, we find the difference between the stack of Gaussian smoothed images within the octave for each octave, and this is the DoG pyramid

2. Find all the local extrema in the DoG pyramid. Each point in the DoG pyramid is compared with the 8 points in the immediate 3×3 neighborhood at the same scale, the 9 points in the 3×3 neighborhood in the DoG that is in the next level up in the scale space, and finally another 9 points in the 3×3 neighborhood that is at the level below. To find at least two extrema per octave, we require two different values of scale factors which implies we need Gaussian smoothed images for at least five different scales in each octave.

3. As σ increases, the extrema identified could become inaccurate. As a result, to identify an extrema with good accuracy, let $\vec{x_0} = (x_0, y_0, z_0)^T$ be a detected point and the true extremum in its vicinity $\vec{x}$ is calculated as $\vec{x} = -H^{-1}(\vec{x_0})J(\vec{x_0})$ where $H(\vec{x_0})$ is the Hessian and $J(\vec{x_0})$ is the gradient estimated.

4. Apply a threshold to eliminate weak or inaccurate extremas. Typical threshold is 0.03

5. We create a 128-dimensional SIFT descriptor vector at each retained extremum in the DoG Pyramid

2.2 Observations

As the scale σ increased, the number of corners detected by the Harris algorithm decreased. This is because as σ increases, the larger the scale at which gray levels must change in order to detect a corner. As σ increased, the SSD correspondences out-performed that of the NCC correspondences in terms of accuracy for the Harris algorithm. The SIFT operator on the books input did not seem as accurate since some of the lines did not match up with the corners detected. However, the results for fountain input by the SIFT operator was promising. Keypoints that were not detected by the SIFT operator or by the Harris algorithm with different scales were detected by the Superglue pipeline. The Superglue pipeline seems to be more efficient than the classical approach as it doesn't manually conduct all the linear algebra computations for each input. The Inputs and Outputs of this assignment are found in the Appendix of this report.

2.3 Source Code

```
# Name: Nikita Ravi
# Class: ECE 66100
# Homework #4
# Deadline: 09/28/2022

# Import Modules
import cv2
import math
import numpy as np

def displayimage(image):
    cv2.imshow("window", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    quit()

def getimages(path):
    img = cv2.imread(path)

    return img

def getinputs():
    booksimage1 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/
   Semester 2/ECE 66100/hw04/Figures/books1.jpeg")
```

```
    fountainimage1 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/fountain1.jpg")
    booksimage2 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/books2.jpeg")
    fountainimage2 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/fountain2.jpg")
    krannertimage1 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/krannert1.jpg")
    ubsimage1 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/ubs1.jpg")
    krannertimage2 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/krannert2.jpg")
    ubsimage2 = getimages(r"/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/Figures/ubs2.jpg")

    return booksimage1, fountainimage1, booksimage2, fountainimage2, krannertimage1, ubsimage1, krannertimage2, ubsimage2

def getpath(subdir, sigma=None, filename=None):
    if(sigma and not filename):
        bookdirectory1 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/booksharris1.jpg"
        fountaindirectory1 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/fountainharris1.jpg"
        bookdirectory2 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/booksharris2.jpg"
        fountaindirectory2 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/fountainharris2.jpg"
        krannertdirectory1 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/krannert1harris.jpg"
        ubsdirectory1 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/ubs1harris.jpg"
        krannertdirectory2 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/krannert2harris.jpg"
        ubsdirectory2 = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+"/sigma" + str(sigma) + "/ubs2harris.jpg"
        return bookdirectory1, fountaindirectory1, bookdirectory2, fountaindirectory2, krannertdirectory1, krannertdirectory2, ubsdirectory1, ubsdirectory2

    elif(filename):
        directory = "/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/"+subdir+filename+str(sigma)+".jpg"
        return directory

def normalizeimage(image):
```

```
    image = image.astype(np.float64)
    image -= np.min(image)
    image /= np.max(image)

    return image

def generatehaarkernels(sigma):
    N = int(math.ceil(4 * sigma))
    N = N if not N%2 else N+1

    hx, hy = np.ones((N,N)), np.ones((N,N))
    hx[:, :int(N//2)] = -1
    hy[int(N//2):, :] = -1

    return hx, hy

def performthresholding(R):
    threshold = np.sort(R, axis=None)[-int(0.10 * len(R.flatten()))]
    return threshold

def performnonmaximumsuppression(radius, height, width, threshold, R):
    corners = []
    for y in range(radius, height - radius):
            for x in range(radius, width - radius):
                subwindow = R[y - radius: y + radius, x - radius: x + radius]
                if(R[y, x] == np.amax(subwindow) and np.amax(subwindow) ¿=
   threshold):
                    corners.append([x, y])

    return corners

def harriscorner(image, sigma, e=0.001):
    image = cv2.cvtColor(image, cv2.COLORBGR2GRAY) if len(image.shape) == 3
   else image #convert image to grayscale
    image = normalizeimage(image) # Normalize the image
    height, width = image.shape

    hx, hy = generatehaarkernels(sigma)
    dx = cv2.filter2D(src = image, ddepth = -1, kernel = hx) # convolve image
   with hx kernel
    dy = cv2.filter2D(src = image, ddepth = -1, kernel = hy) # convolve image
   with hy kernel

    dx2 = dx * dx
    dy2 = dy * dy
    dxdy = dx * dy

    N = 2 * int((5 * sigma) // 2) + 1
    kernelforc = np.ones((N,N))

    sumdx2 = cv2.filter2D(src = dx2, ddepth = -1, kernel = kernelforc)
    sumdy2 = cv2.filter2D(src = dy2, ddepth = -1, kernel = kernelforc)
    sumdxdy = cv2.filter2D(src = dxdy, ddepth = -1, kernel = kernelforc)

```

```
    trace = sumdx2 + sumdy2
    det = (sumdx2 * sumdy2) - (sumdxdy)**2

    k = det / (trace**2 + e) # r / (1+r) 2
    k = np.sum(k) / (height * width) # Normalize

    R = det - k*(trace**2)
    Rthreshold = performthresholding(R)

    radius = N // 2
    return performnonmaximumsuppression(radius=radius, height=height, width=width, threshold=Rthreshold, R=R)

def addcirclesaroundcorners(corners, image, color):
    # image = cv2.cvtColor(image, cv2.COLORGRAY2RGB)
    for x, y in corners:
        image = cv2.circle(image, (x, y), radius=3, color=color)

    return image

def Harris(image, sigma, directory, color):
    corners = list(harriscorner(image=image, sigma=sigma))
    image = addcirclesaroundcorners(corners, image, color)
    cv2.imwrite(directory, image)

    return corners

def drawcorrespondences(image1, image2, width1, correspondences, path):
    combined = np.hstack((image1, image2))
    for pt1, pt2,   in correspondences:
        cv2.circle(combined, (pt1[0], pt1[1]), radius=3, color=(0,0,255))
        cv2.circle(combined, (pt2[0] + width1, pt2[1]), radius=3, color=(0,0,255))
        cv2.line(combined, pt1, (pt2[0] + width1, pt2[1]), color=(10,240,240), thickness=1)

    cv2.imwrite(path, combined)

def establishcorrespondences(image1, image2, corners1, corners2, mode="ssd", radius=21, path=""):
    height1, width1,   = image1.shape
    image2 = cv2.resize(image2, (width1, height1))
    height2, width2,   = image2.shape

    gray1 = cv2.cvtColor(image1, cv2.COLORBGR2GRAY) if(len(image1.shape) == 3) else image1
    gray2 = cv2.cvtColor(image2, cv2.COLORBGR2GRAY) if(len(image2.shape) == 3) else image2

    gray1 = normalizeimage(gray1)
    gray2 = normalizeimage(gray2)

    correspondences = []

```

```
    for c1 in corners1:
        bestMatch = None
        currBestDistance = float("inf")

        for c2 in corners2:
            distance = calculatedistance(gray1, gray2, c1, c2, mode, radius)
            if(distance ¡ currBestDistance):
                currBestDistance = distance
                bestMatch = c2

        correspondences.append((c1, bestMatch, currBestDistance))

    drawcorrespondences(image1, image2, width1, correspondences, path)

def calculatedistance(image1, image2, coord1, coord2, mode, radius=21):
    x1, y1 = coord1
    x2, y2 = coord2

    height1, width1 = image1.shape
    height2, width2 = image2.shape

    radius = min(x1, width1 - x1, y1, height1 - y1,
        x2, width2 - x2, y2, height2 - y2,
        radius)

    f1 = image1[y1 - radius: y1 + radius, x1 - radius: x1 + radius]
    f2 = image2[y2 - radius: y2 + radius, x2 - radius: x2 + radius]
    distance = None

    if(mode == "ssd"):
        distance = np.sum((f1 - f2)**2)
    elif(mode == "ncc"):
        m1, m2 = np.mean(f1), np.mean(f2)
        numerator = np.sum((f1-m1)*(f2-m2))
        denominator = np.sqrt(np.sum((f1-m1)**2) * np.sum((f2-m2)**2))
        distance = numerator / denominator
        distance = 1 - distance

    return distance

def sift(image1, image2, path=""):
    # Citation: https://docs.opencv.org/4.x/dc/dc3/tutorialpymatcher.html
    image1 = cv2.cvtColor(image1, cv2.COLORBGR2GRAY) if (len(image1.shape) ==
    3) else image1
    image2 = cv2.cvtColor(image2, cv2.COLORBGR2GRAY) if (len(image2.shape) ==
    3) else image2

    siftdetector = cv2.xfeatures2d.SIFTcreate() # Create SIFT Detector
    keyPoint1, descriptor1 = siftdetector.detectAndCompute(image1, None)
    keyPoint2, descriptor2 = siftdetector.detectAndCompute(image2, None)

    bruteForceMatcher = cv2.BFMatcher() # It takes the descriptor of one
   feature in first set and matches it with all other features in second set
   using some distance calculation - the closest distance is returned
```

```
    matches = bruteForceMatcher.knnMatch(descriptor1, descriptor2, k=2) # returns k best matches where k is specified by the user

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

    combined = np.hstack((image1, image2))
    result = cv2.drawMatchesKnn(image1, keyPoint1, image2, keyPoint2, goodmatches, combined, flags=2)
    cv2.imwrite("/Users/nikitaravi/Documents/Academics/Year 4/Semester 2/ECE 66100/hw04/SIFT/"+path, result)


if __name__ == "__main__":
    # Get Images
    booksimage1, fountainimage1, booksimage2, fountainimage2, krannertimage1, ubsimage1, krannertimage2, ubsimage2 = getinputs()

    # Task 1
    scale = [1.8,3.2,5.4,7.3]
    for sigma in scale:
        print("Sigma: ", sigma)
        bookscopy1, fountaincopy1, bookscopy2, fountaincopy2 = booksimage1.copy(), fountainimage1.copy(), booksimage2.copy(), fountainimage2.copy()
        krannertcopy1, krannertcopy2, ubscopy1, ubscopy2 = krannertimage1.copy(), krannertimage2.copy(), ubsimage1.copy(), ubsimage2.copy()
        bookdirectory1, fountaindirectory1, bookdirectory2, fountaindirectory2, krannertdirectory1, krannertdirectory2, ubsdirectory1, ubsdirectory2 = getpath("Harris", sigma)

        # 1a - Harris Corner Detection
        bookcorners1 = Harris(bookscopy1, sigma, bookdirectory1, (0,0,255))
        fountaincorners1 = Harris(fountaincopy1, sigma, fountaindirectory1, (0,255,0))

        bookcorners2 = Harris(bookscopy2, sigma, bookdirectory2, (0,0,255))
        fountaincorners2 = Harris(fountaincopy2, sigma, fountaindirectory2, (0,255,0))

        krannertcorners1 = Harris(krannertcopy1, sigma, krannertdirectory1, (0,0,255))
        ubscorners1 = Harris(ubscopy1, sigma, ubsdirectory1, (0,255,0))

        krannertcorners2 = Harris(krannertcopy2, sigma, krannertdirectory2, (0,0,255))
        ubscorners2 = Harris(ubscopy2, sigma, ubsdirectory2, (0,255,0))

        # 1b - Establish Correspondences
        bookscopy1, bookscopy2, fountaincopy1, fountaincopy2 = booksimage1.copy(), booksimage2.copy(), fountainimage1.copy(),
```

```
    fountainimage2.copy()
239     establishcorrespondences(bookscopy1, bookscopy2, bookcorners1,
    bookcorners2, mode="ssd", path=getpath("Harris/correspondences/", sigma=
    sigma, filename="bookscorrssd"))
240     establishcorrespondences(fountaincopy1, fountaincopy2,
    fountaincorners1, fountaincorners2, mode="ssd", path=getpath("Harris/
    correspondences/", sigma=sigma, filename="fountaincorrssd"))
241     establishcorrespondences(bookscopy1, bookscopy2, bookcorners1,
    bookcorners2, mode="ncc", path=getpath("Harris/correspondences/", sigma=
    sigma, filename="bookscorrncc"))
242     establishcorrespondences(fountaincopy1, fountaincopy2,
    fountaincorners1, fountaincorners2, mode="ncc", path=getpath("Harris/
    correspondences/", sigma=sigma, filename="fountaincorrncc"))
243
244     establishcorrespondences(krannertcopy1, krannertcopy2,
    krannertcorners1, krannertcorners2, mode="ssd", path=getpath("Harris/
    correspondences/", sigma=sigma, filename="krannertcorrssd"))
245     establishcorrespondences(ubscopy1, ubscopy2, ubscorners1,
    ubscorners2, mode="ssd", path=getpath("Harris/correspondences/", sigma=
    sigma, filename="ubscorrssd"))
246     establishcorrespondences(krannertcopy1, krannertcopy2,
    krannertcorners1, krannertcorners2, mode="ncc", path=getpath("Harris/
    correspondences/", sigma=sigma, filename="krannertcorrncc"))
247     establishcorrespondences(ubscopy1, ubscopy2, ubscorners1,
    ubscorners2, mode="ncc", path=getpath("Harris/correspondences/", sigma=
    sigma, filename="ubscorrncc"))
248
249
250 # Task 2 - SIFT
251 bookscopy1, bookscopy2, fountaincopy1, fountaincopy2 = booksimage1.
    copy(), booksimage2.copy(), fountainimage1.copy(), fountainimage2.copy
    ()
252 krannertcopy1, krannertcopy2, ubscopy1, ubscopy2 = krannertimage1.
    copy(), krannertimage2.copy(), ubsimage1.copy(), ubsimage2.copy()
253
254 sift(bookscopy1, bookscopy2, "books.jpg")
255 sift(fountaincopy1, fountaincopy2, "fountain.jpg")
256 sift(krannertcopy1, krannertcopy2, "krannert.jpg")
257 sift(ubscopy1, ubscopy2, "ubs.jpg")
```

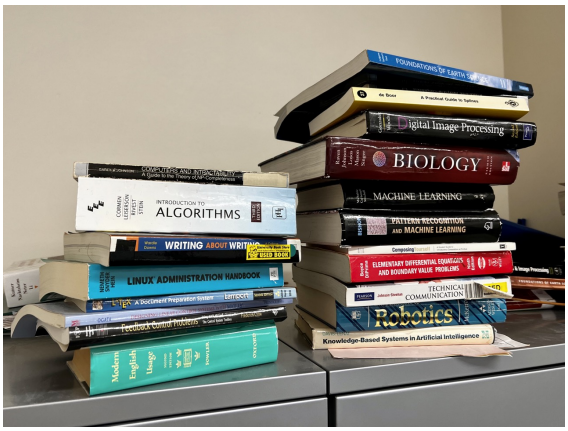
Listing 1: The Source Code

References

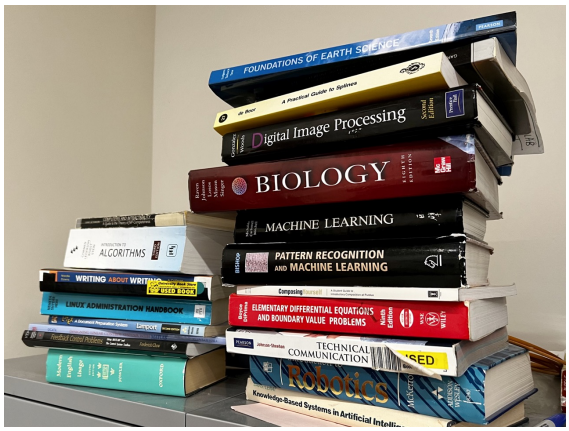
[1] J. B. Pal, "Detecting Features via Modified Harris Corners and Matching them through SIFT", , 2020, URL: https://papers.ssrn.com/sol3/papers.cfm?abstract$_id$ = 3619887.

3 Appendix

3.1 The Inputs and Outputs



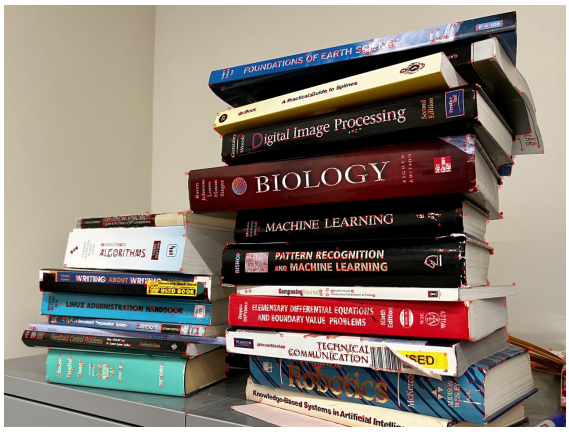
(a) First Viewpoint of Books



(b) Second Viewpoint of Books

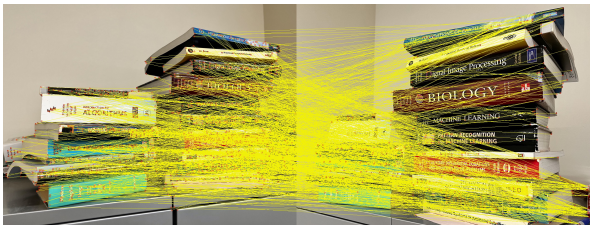
Figure 2: First Input Pair

(a) Detected Corners with $\sigma = 1.8$

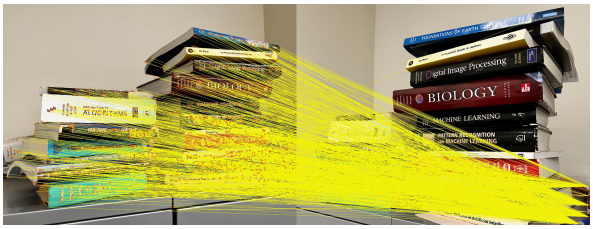


(b) Detected Corners with $\sigma = 1.8$

Figure 3: Harris Corner with $\sigma = 1.8$ for Books



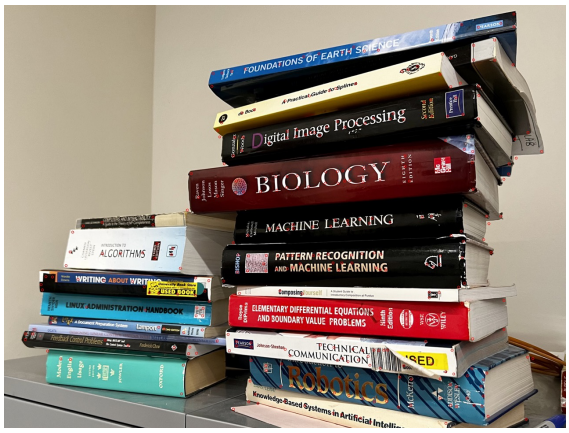
(a) NCC at $\sigma = 1.8$



(b) SSD at $\sigma = 1.8$

Figure 4: Establish Correspondences at $\sigma = 1.8$ for Books

(a) Detected Corners with $\sigma = 3.2$



(b) Detected Corners with $\sigma = 3.2$

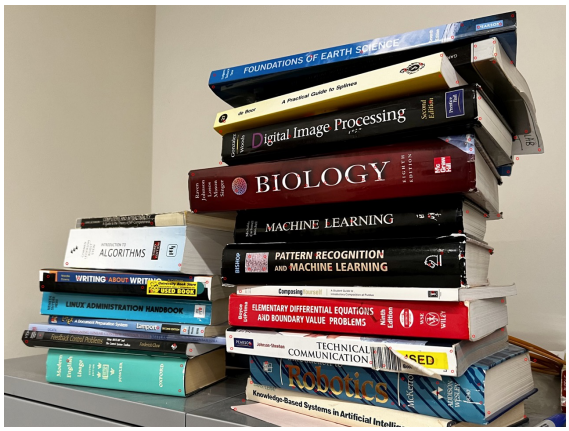
Figure 5: Harris Corner with $\sigma = 3.2$ for Books

(a) NCC at $\sigma = 3.2$

(b) SSD at $\sigma = 3.2$

Figure 6: Establish Correspondences at $\sigma = 3.2$ for Books

(a) Detected Corners with $\sigma = 5.4$



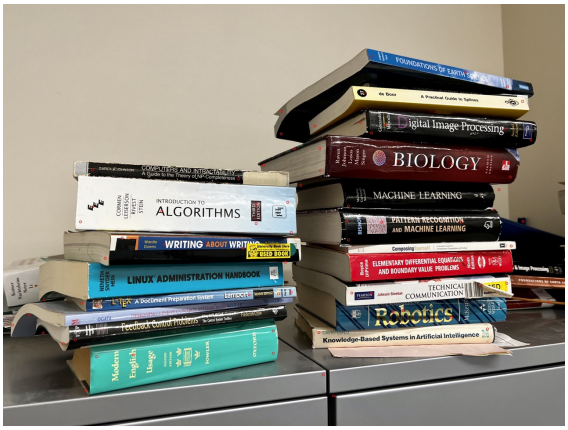
(b) Detected Corners with $\sigma = 5.4$

Figure 7: Harris Corner with $\sigma = 5.4$ for Books

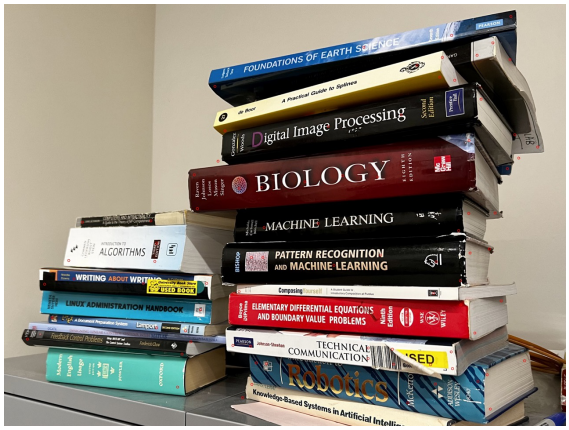
(a) NCC at $\sigma = 5.4$

(b) SSD at $\sigma = 5.4$

Figure 8: Establish Correspondences at $\sigma = 5.4$ for Books



(a) Detected Corners with $\sigma = 7.3$



(b) Detected Corners with $\sigma = 7.3$

Figure 9: Harris Corner with $\sigma = 7.3$ for Books

(a) NCC at $\sigma = 7.3$ (b) SSD at $\sigma = 7.3$

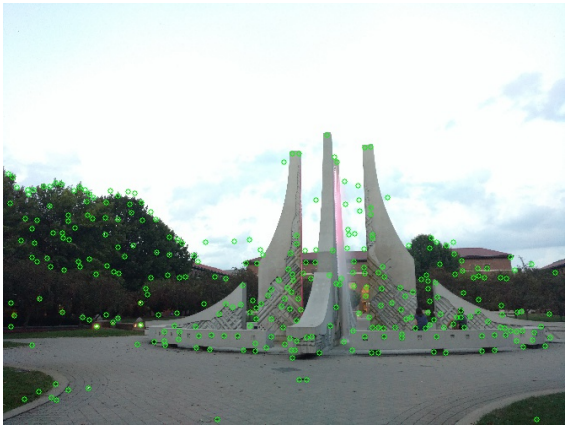
Figure 10: Establish Correspondences at $\sigma = 7.3$ for Books

(a) SIFT Output for the Books Input (b) Superglue Output for the Books Input

Figure 11: SIFT and Superglue for Books

(a) First Viewpoint of Fountain (b) Second Viewpoint of Fountain

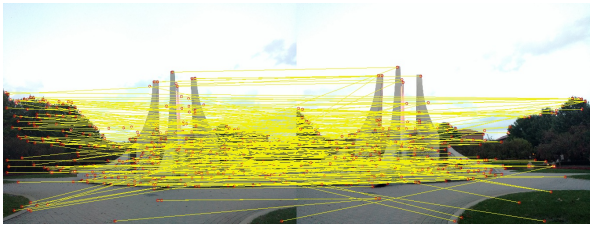
Figure 12: Second Input Pair



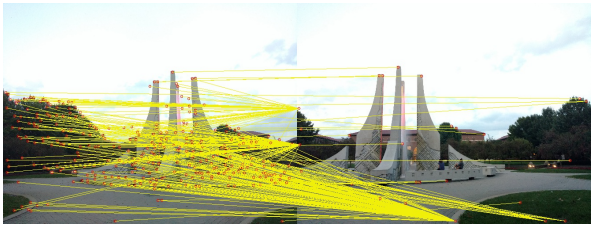
(a) Detected Corners with $\sigma = 1.8$

(b) Detected Corners with $\sigma = 1.8$

Figure 13: Harris Corner with $\sigma = 1.8$ for Fountain



(a) NCC at $\sigma = 1.8$



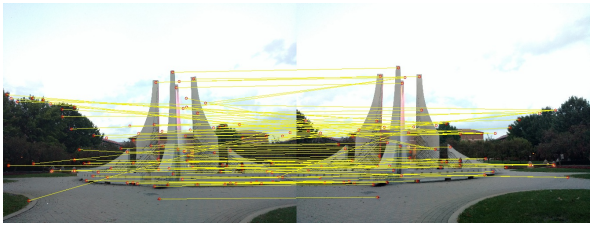
(b) SSD at $\sigma = 1.8$

Figure 14: Establish Correspondences at $\sigma = 1.8$ for Fountain

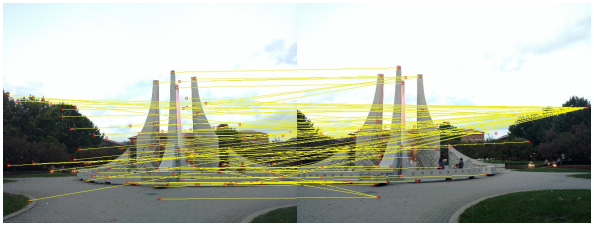
(a) Detected Corners with $\sigma = 3.2$

(b) Detected Corners with $\sigma = 3.2$

Figure 15: Harris Corner with $\sigma = 3.2$ for Fountain



(a) NCC at $\sigma = 3.2$



(b) SSD at $\sigma = 3.2$

Figure 16: Establish Correspondences at $\sigma = 3.2$ for Fountain

(a) Detected Corners with $\sigma = 5.4$

(b) Detected Corners with $\sigma = 5.4$

Figure 17: Harris Corner with $\sigma = 5.4$ for Fountain

(a) NCC at $\sigma = 5.4$

(b) SSD at $\sigma = 5.4$

Figure 18: Establish Correspondences at $\sigma = 5.4$ for Books

(a) Detected Corners with $\sigma = 7.3$

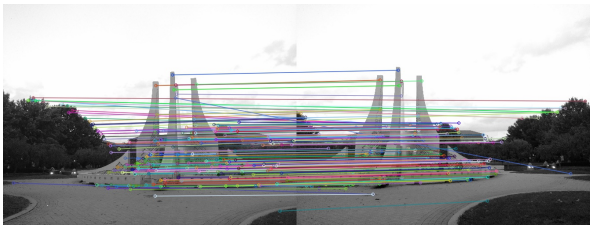
(b) Detected Corners with $\sigma = 7.3$

Figure 19: Harris Corner with $\sigma = 7.3$ for Fountain

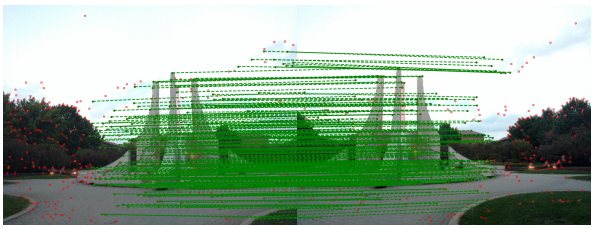
(a) NCC at $\sigma = 7.3$

(b) SSD at $\sigma = 7.3$

Figure 20: Establish Correspondences at $\sigma = 7.3$ for Fountain



(a) SIFT Output for the Fountains Input



(b) Superglue Output for the Fountain Input

Figure 21: SIFT and Superglue for Fountain

(a) First Viewpoint of Krannert

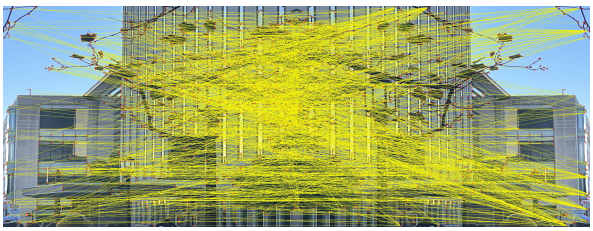
(b) Second Viewpoint of Krannert

Figure 22: Third Input Pair

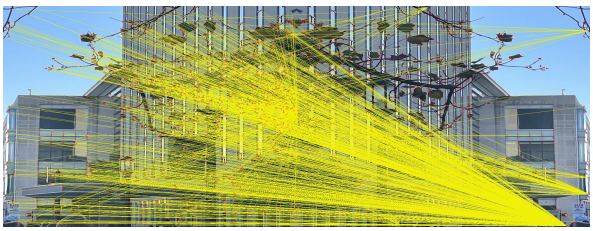
(a) Detected Corners with $\sigma = 1.8$

(b) Detected Corners with $\sigma = 1.8$

Figure 23: Harris Corner with $\sigma = 1.8$ for Krannert



(a) NCC at $\sigma = 1.8$



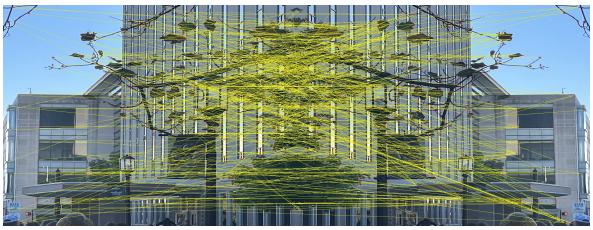
(b) SSD at $\sigma = 1.8$

Figure 24: Establish Correspondences at $\sigma = 1.8$ for Krannert

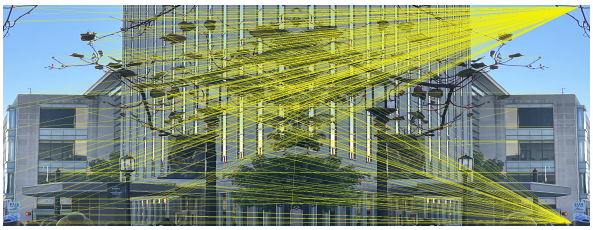
(a) Detected Corners with $\sigma = 3.2$

(b) Detected Corners with $\sigma = 3.2$

Figure 25: Harris Corner with $\sigma = 3.2$ for Krannert



(a) NCC at $\sigma = 3.2$



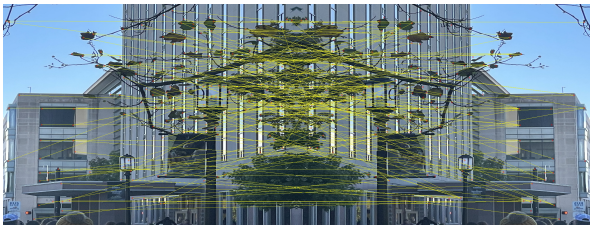
(b) SSD at $\sigma = 3.2$

Figure 26: Establish Correspondences at $\sigma = 3.2$ for Krannert

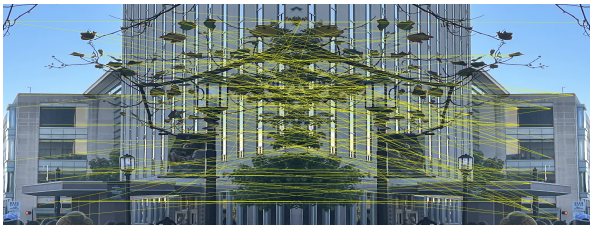
(a) Detected Corners with $\sigma = 5.4$

(b) Detected Corners with $\sigma = 5.4$

Figure 27: Harris Corner with $\sigma = 5.4$ for Krannert



(a) NCC at $\sigma = 5.4$



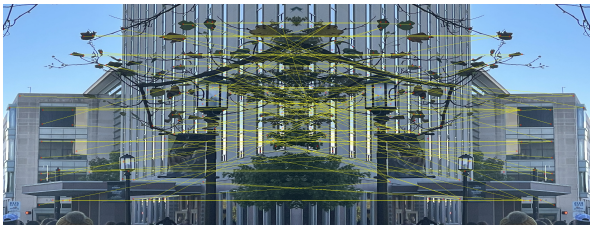
(b) SSD at $\sigma = 5.4$

Figure 28: Establish Correspondences at $\sigma = 5.4$ for Books

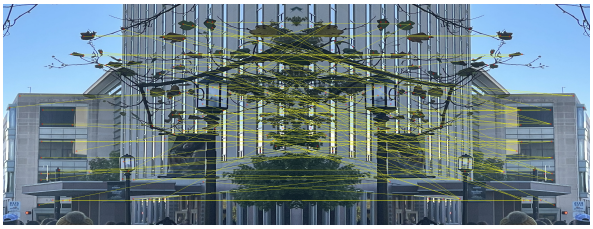
(a) Detected Corners with $\sigma = 7.3$

(b) Detected Corners with $\sigma = 7.3$

Figure 29: Harris Corner with $\sigma = 7.3$ for Krannert

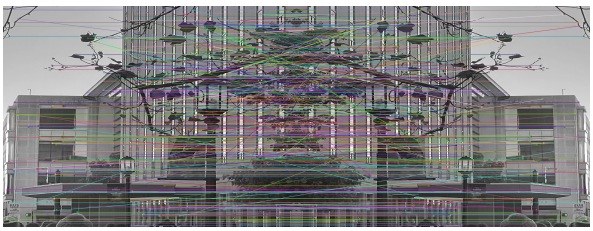


(a) NCC at $\sigma = 7.3$

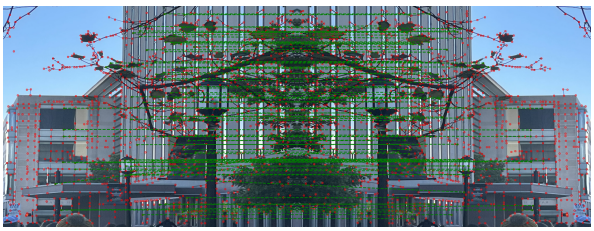


(b) SSD at $\sigma = 7.3$

Figure 30: Establish Correspondences at $\sigma = 7.3$ for Krannert



(a) SIFT Output for the Krannert Input



(b) Superglue Output for the Krannert Input

Figure 31: SIFT and Superglue for Krannert

(a) First Viewpoint of UBS

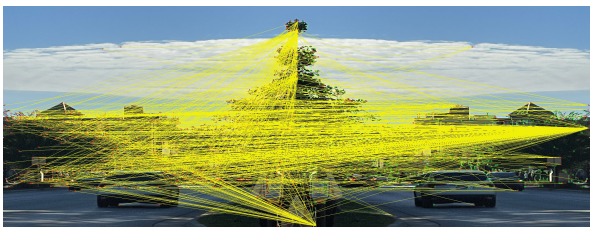
(b) Second Viewpoint of UBS

Figure 32: Fourth Input Pair

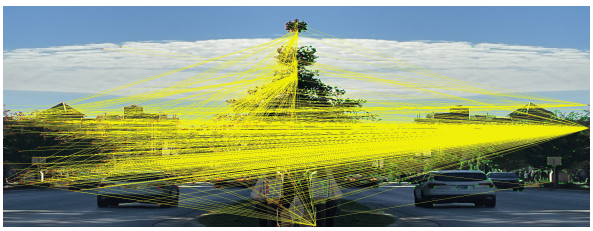
(a) Detected Corners with $\sigma = 1.8$

(b) Detected Corners with $\sigma = 1.8$

Figure 33: Harris Corner with $\sigma = 1.8$ for UBS



(a) NCC at $\sigma = 1.8$



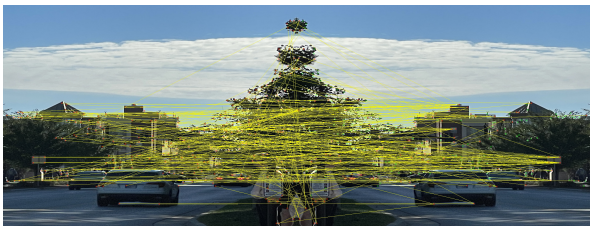
(b) SSD at $\sigma = 1.8$

Figure 34: Establish Correspondences at $\sigma = 1.8$ for UBS

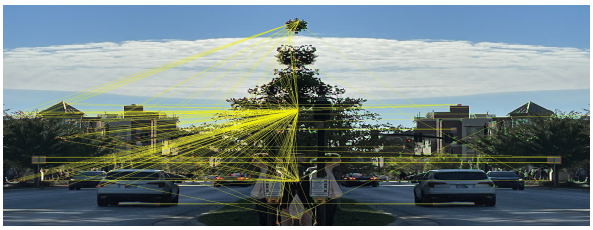
(a) Detected Corners with $\sigma = 3.2$

(b) Detected Corners with $\sigma = 3.2$

Figure 35: Harris Corner with $\sigma = 3.2$ for UBS



(a) NCC at $\sigma = 3.2$



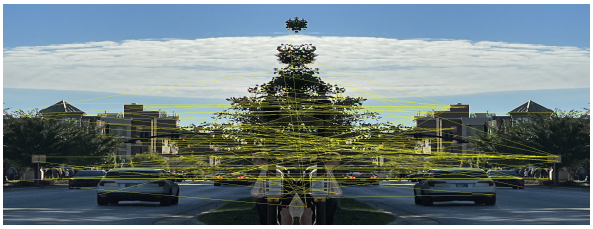
(b) SSD at $\sigma = 3.2$

Figure 36: Establish Correspondences at $\sigma = 3.2$ for UBS

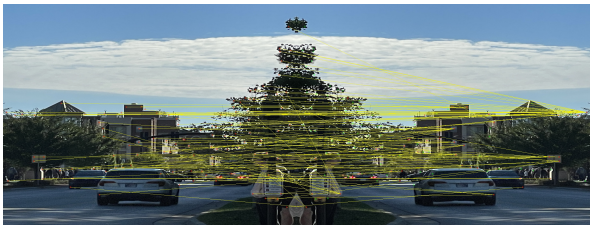
(a) Detected Corners with $\sigma = 5.4$

(b) Detected Corners with $\sigma = 5.4$

Figure 37: Harris Corner with $\sigma = 5.4$ for UBS



(a) NCC at $\sigma = 5.4$



(b) SSD at $\sigma = 5.4$

Figure 38: Establish Correspondences at $\sigma = 5.4$ for UBS

(a) Detected Corners with $\sigma = 7.3$

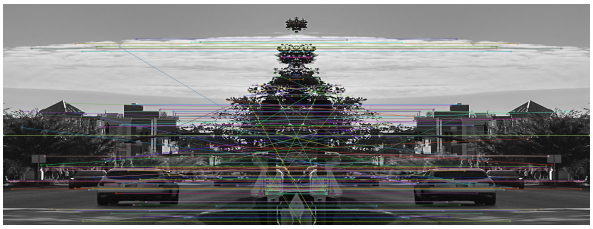
(b) Detected Corners with $\sigma = 7.3$

Figure 39: Harris Corner with $\sigma = 7.3$ for UBS

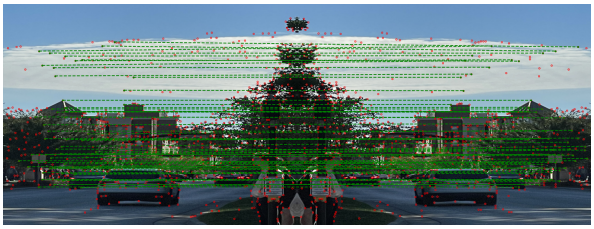
(a) NCC at $\sigma = 7.3$

(b) SSD at $\sigma = 7.3$

Figure 40: Establish Correspondences at $\sigma = 7.3$ for UBS



(a) SIFT Output for the UBS Input



(b) Superglue Output for the UBS Input

Figure 41: SIFT and Superglue for UBS