Computer Vision Coursework2 Report

Haojing Tong

April 2024

1 Results

1.1 Threshold Finding

According to the instruction in Feature Detection, I wrote a function called harris_corner_detector. This is a function which would return a list of keypoints and keypoints_orientations. When I input the bernieSanders.jpg as my input image, I could get the number of keypoints and each keypoint's position information. Based on each keypoint's position. The function draw_keypoints return the image with keypoints. Here is an example image with many keypoints.

Figure 1: Here is the bernieSanders image with keypoints.

After I get the basic image with keypoints. Because the Harris Corner Detector function has many parameters. One of the parameters we need to research is threshold value. Thus, I wrote a fucntion called plot_keypoints_threshold_variation. It could show the number of keypoints under different threshold values. Here, I selected the threshold value start at 0.001 to 0.2 and divided into 20 steps. Here is the plot table.

For the decreasing rate shown in the table above, we could obviously find while the threshold value range from 0.075 to 0.175, the number of keypoints detected decreasing slowly. Such that, I focus more on this range. Therefore, I

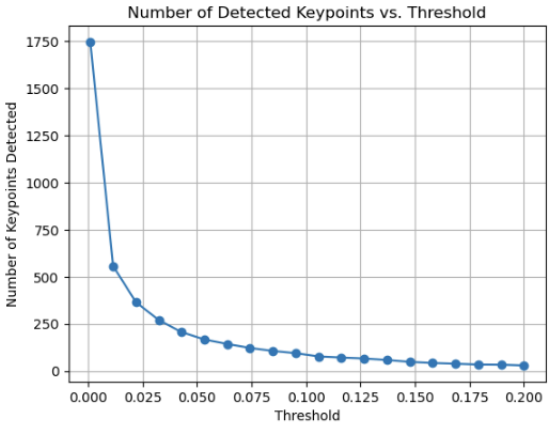


Figure 2: Here is the image shows Number of Detected Keypoints vs. Threshold.

wrote the function called plot_image_with_keypoints, which could genrate many images with keypoints. I set the threshold start at 0.075, end at 0.175, and divided into 6 steps. It could intuitive observe which threshold value is most suitable. Here is the image of six images of keypoints with different threshold values. (Figure 3)

After my careful examination, I selected threshold value is equal to 0.095. Thus, 0.095 is my threshold value to find my strongest interest points.

1.2 Feature Matching

Before I entering the feature matching step, I need to complete feature description firstly. Feature Description has one function called compute_orb_descriptors. This function would return keypoints_cv2 and descriptors, which would be used in later feature matching step.

In feature matching, we have two methods. One is basic match_feature function with ratio_thresh = 0.7, another is ssd_match_features based on Sum of Squared Differences(SSD) between descriptors.

In the end, draw_matches function is used to connecting two matching keypoints with a line. Now, I would show a series of image feature matching picture below. The upper one is using basic matching method, the one below is using the method of Sum of Squared Differences.

Figure 3: Here is the image shows the six images with keypoint with different threshold values.

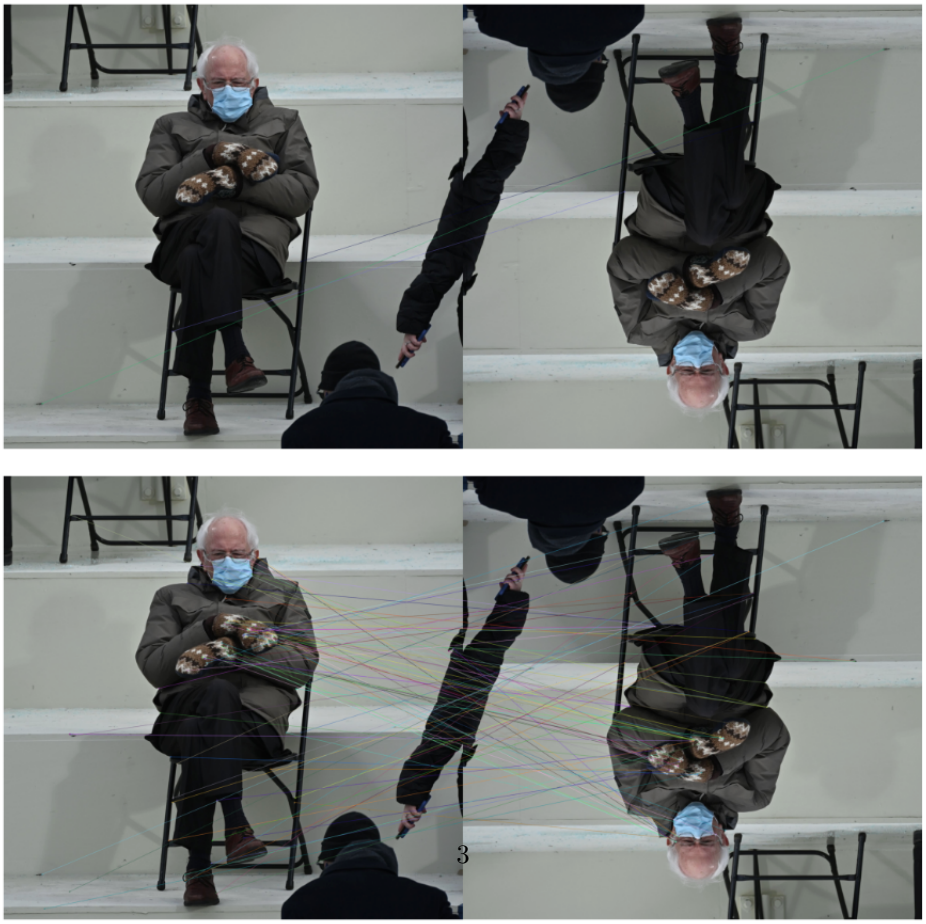


Figure 4: Here is the image matching bernie and bernie180. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

According to the ratio matching image and Sum of Squared Differences Matching image. The Sum of Squared Differences method has better effectiveness. I found the keypoints matching almost correctly. However, here are several keypoints do not match correctly.

Figure 5: Here is the image matching bernie and bernieBenefitBeautySalon. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

Figure 6: Here is the image matching bernie and BernieFriends. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

For the above two image matching outcomes, bernie vs bernieBenefitBeautySalon and bernie vs bernieFriends and bernie vs bernieSchoolLunch. From these two outcomes, I drew similar conclusions. This two images copy the character in original image into a new background. Thus, most of the keypoints matching correctly and a few keypoints matching to the background.

Figure 7: Here is the image matching bernie and bernieNoisy2. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

According to the results of bernie vs bernieNoisy2. The keypoints matching is not good. Here are two main reasons: 1.Impact of Noise: Noise increases the differences in pixel values between the images, resulting in a higher SSD value when comparing a noisy image with the original. It would lead to despite the content of noisy image is same as original image, the noise make them looks dissimilar. 2.Comparison of Keypoints: When comparing keypoints, if the areas around the keypoints are significantly affected by noise, the comparison will show greater differences. This could lead to keypoints that should match failing.

Figure 8: Here is the image matching bernie and berniePixelated2. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

According to the results of bernie vs berniePixelated2. The keypoints matching is not good. Here are two main reasons: 1.Overall Image Difference: Unlike the random pixel value changes introduced by noise, pixelation causes structural changes. The SSD value between the original image and its pixelated version might significantly increase since pixelation alters many pixel values in the image, especially in areas of edges and details. 2.Comparison of Keypoints: Pixelation can blur keypoints in an image, especially when the pixelation level is high. This would result in keypoints from the original image not finding accurate matches in the pixelated image because pixelation reduces the detail information available for keypoint identification and matching.

Figure 9: Here is the image matching bernie and bernieSchoolLunch. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

Figure 10: Here is the image matching bernie and brighterBernie. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

Figure 11: Here is the image matching bernie and darkerBernie. The upper one is using ratio matching method, the one below is using the method of Sum of Squared Differences..

The results between bernie vs brighterBernie is comparing the original image with image after grayscaled and bernie vs darkerBernie(with adding a threshold and change the color to pure white and black). Because the grayscaled image would not change their edges and corners information. Their basic structure would not change. Therefore, the keypoints matching should not be heavily influenced. However, in my matching picture, the matching effectiveness is bad.

2 Appendix

In [1]:
```
# import some necessary libraries

import cv2
import numpy as np
from scipy.ndimage import gaussian_filter, sobel, maximum_filter
```

Step 1: Feature Detection

In [2]:
```
# Harris corner detector function

def harris_corner_detector(image, sobel_size=3, gaussian_size=5, sigma=0.5, k=0.05, threshold=0.095):
    """
    Parameters:
    - image: Input image
    - sobel_size: Size of the sobel kernel
    - gaussian_size: Size of the Gaussian kernel
    - sigma: Standard deviation of the Gaussian kernel
    - k: Harris detector free parameter
    - threshold: Threshold for detecting corners


    Returns:
    - keypoints: Detected keypoints (corners)
    - orientations: Orientation of the keypoints
    """

    # Convert image to grayscale
    if len(image.shape) > 2:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray = image

    gray = np.float32(gray)

    # Calculate gradients using Sobel operator
    Ix = sobel(gray, axis=0, mode='reflect')
    Iy = sobel(gray, axis=1, mode='reflect')

    # Calculate the products of derivatives
    Ixx = gaussian_filter(Ix**2, sigma=sigma, mode='reflect')
    Ixy = gaussian_filter(Ix*Iy, sigma=sigma, mode='reflect')
    Iyy = gaussian_filter(Iy**2, sigma=sigma, mode='reflect')

    # Compute Harris matrix M for each pixel
    det_M = Ixx * Iyy - Ixy ** 2
    trace_M = Ixx + Iyy
    response = det_M - k * (trace_M ** 2)

    #Thresholding
    keypoints = np.argwhere(response > threshold * response.max())

    # Compute orientations
    orientations = np.rad2deg(np.arctan2(Iy, Ix))

    # Non-maximum suppression using a maximum filter
    nms_response = maximum_filter(response, size=7, mode='reflect')

    # Find the local maxima that are also above the threshold
    local_maxima = (response == nms_response)
    thresholded_local_maxima = local_maxima & (response > threshold * response.max())

    # Keep keypoints that are local maxima
    keypoints = np.argwhere(thresholded_local_maxima)
    keypoints_orientations = orientations[keypoints[:, 0], keypoints[:, 1]]

    return keypoints, keypoints_orientations
```

In [3]:
```
# Load the image

image = cv2.imread('image/bernieSanders.jpg')
```

In [4]:
```
# Apply Harris corner detector

keypoints, orientations = harris_corner_detector(image)

# Print the number of keypoints detected
print(f"Number of keypoints detected: {len(keypoints)}")
print(keypoints)
```

```
Number of keypoints detected: 94
[[   8  975]
 [ 201  347]
 [ 402 1271]
 [ 423 1313]
 [ 426 1181]
 [ 428 1255]
 [ 440 1063]
 [ 447 1076]
 [ 461 1015]
 [ 468  971]
 [ 508 1283]
 [ 516 1284]
 [ 534 1075]
 [ 554  950]
 [ 633 1148]
 [ 725 1230]
 [ 747 1189]
 [ 750 1159]
 [ 757 1149]
 [ 757 1154]
 [ 760 1138]
 [ 764 1141]
 [ 774 1307]
 [ 775 1302]
 [ 788 1275]
 [ 790 1137]
 [ 794 1259]
 [ 797 1281]
 [ 798 1277]
 [ 800 1263]
 [ 806 1363]
 [ 810 1360]
 [ 816 1255]
 [ 818 1242]
 [ 825 1237]
 [ 829 1259]
 [ 829 1394]
 [ 832 1252]
 [ 839 1393]
 [ 845 1364]
 [ 856 1370]
 [ 857 1366]
 [ 863 1126]
 [ 875 1338]
 [ 876 1375]
 [ 882 1128]
 [ 893 2322]
 [ 905 2297]
 [ 907 1141]
 [ 909 1153]
 [ 921 1155]
 [ 924 1180]
 [ 926 1160]
 [ 927 1050]
 [ 929 1184]
 [ 930 1177]
 [ 938 1025]
 [ 944 1039]
 [ 944 1193]
 [ 968 1049]
 [1006  949]
 [1079 1558]
 [1099  801]
 [1102 1510]
 [1225  808]
 [1299 2022]
 [1300 2007]
 [1328 1459]
 [1328 1534]
 [1335  342]
 [1336  348]
 [1343  371]
 [1453  863]
 [1461 2099]
 [1489  894]
 [1517 1378]
 [1666 1506]
 [1667  874]
 [1678 1101]
 [1689  871]
 [1689 1503]
 [1873 1241]
 [1884 1484]
 [1887  853]
 [1888 1077]
 [2004 1072]
 [2062  199]
 [2063  937]
 [2093  939]
 [2101  939]
 [2151  341]
```

```
[2158 1097]
[2162 1102]
[2240  212]]
```

In [5]:
```python
# The function of draw the image with keypoints

def draw_keypoints(image, keypoints):
    """
    Parameters:
    - image: The original image.
    - keypoints: The array of keypoints detected by the Harris corner detector.

    Returns:
    - image_with_keypoints: The original image with keypoints drawn on it.
    """

    # Copy the original image
    image_with_keypoints = image.copy()
    # Draw each keypoints as a little circle(like a point)
    for keypoint in keypoints:
        y, x = keypoint
        cv2.circle(image_with_keypoints, (x, y), radius=5, color=(0,0,255), thickness=1)
    return image_with_keypoints
```

In [6]:
```python
# Show the image with keypoints

import matplotlib.pyplot as plt

image_with_keypoints = draw_keypoints(image, keypoints)
plt.imshow(image_with_keypoints[:, :, ::-1])
plt.axis('off')
plt.show()
```

Step 2: Feature Description

In [7]:
```python
# Function to compute ORB descriptors for keypoints
def compute_orb_descriptors(image, keypoints):
    """
    Parameters:
    - image: The original image
    - keypoints: The array of keypoints for which descriptors are to be computed.

    Returns:
    - keypoints: Keypoints with size and angle added.
    - descriptors: Computed ORB descriptors for the keypoints.
    """

    # Initialize the ORB detector
    orb = cv2.ORB_create()

    # Convert keypoints to cv2.KeyPoint objects
    keypoints_cv2 = [cv2.KeyPoint(x=float(kp[1]), y=float(kp[0]), size=1, angle=0) for kp in keypoints]

    # Compute ORB descriptors
    keypoints_cv2, descriptors = orb.compute(image, keypoints_cv2)

    return keypoints_cv2, descriptors
```

In [8]:
```python
# Compute ORB descriptors for the keypoints
keypoints_cv2, descriptors = compute_orb_descriptors(image, keypoints)

print(len(keypoints_cv2))
```

```
print(len(descriptors))
print(descriptors)
```

```
93
93
[[198  54  24 ...  85 141 199]
 [151 127 107 ... 179 102 241]
 [ 74  33  16 ...   6   7 226]
 ...
 [130 181 187 ...  76 147 243]
 [128 163 147 ... 232   1 161]
 [ 46  15 224 ... 127 121 196]]
```

Step 3: Feature Matching

In [9]:
```
# Function of match features

from scipy.spatial import distance

def match_features(descriptor1, descriptor2, ratio_thresh=0.7):
    """
    Parameters:
    - descriptor1: Descriptors from the first image.
    - descriptor2: Descriptors from the second image.
    - ratio_thresh: The ratio threshold for Lowe's ratio test.

    Returns:
    - good_matches: A list of good matches passing the ratio test.
    """

    # Create a BFMatcher object
    # It will find all matches in both images using KNN.
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
    matches = bf.knnMatch(descriptor1, descriptor2, k=2)

    # Apply Lowe's ratio test to find good matches
    good_matches = []
    for m, n in matches:
        if m.distance < ratio_thresh * n.distance:
            good_matches.append(m)

    return good_matches
```

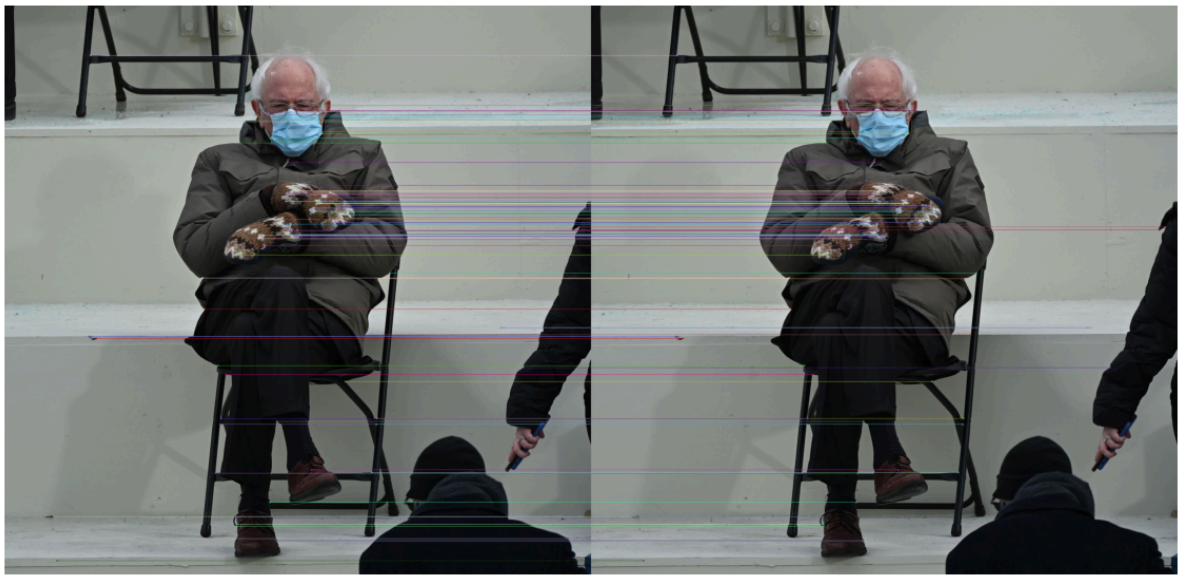
In [10]:
```
# Function of matching features based on the Sum of Squared Differences(SSD) between descriptors.

def ssd_match_features(descriptor1, descriptor2):
    """
    Parameters:
    - descriptor1: Descriptors from the first image.
    - descriptor2: Descriptors from the second image.

    Returns:
    - matches: A list of matches based on SSD.
    """

    # Compute the SSD distance matrix between all descriptors
    ssd_distances = distance.cdist(descriptor1, descriptor2, 'euclidean') ** 2

    # Find the smallest distance for each descriptor in the first image
    indices1 = np.arange(descriptor1.shape[0])
    indices2 = ssd_distances.argmin(axis=1)

    # Create matches based on SSD, ignoring the actual distance value for now
    matches = [cv2.DMatch(_queryIdx=i, _trainIdx=j, _distance=ssd_distances[i, j]) for i, j in zip(indices1, indice

    return matches
```

In [11]:
```
# Perform feature matching using Lowe's ratio test
good_matches_ratio_test = match_features(descriptors, descriptors)

# Perform feature matching using SSD
matches_ssd = ssd_match_features(descriptors, descriptors)
```

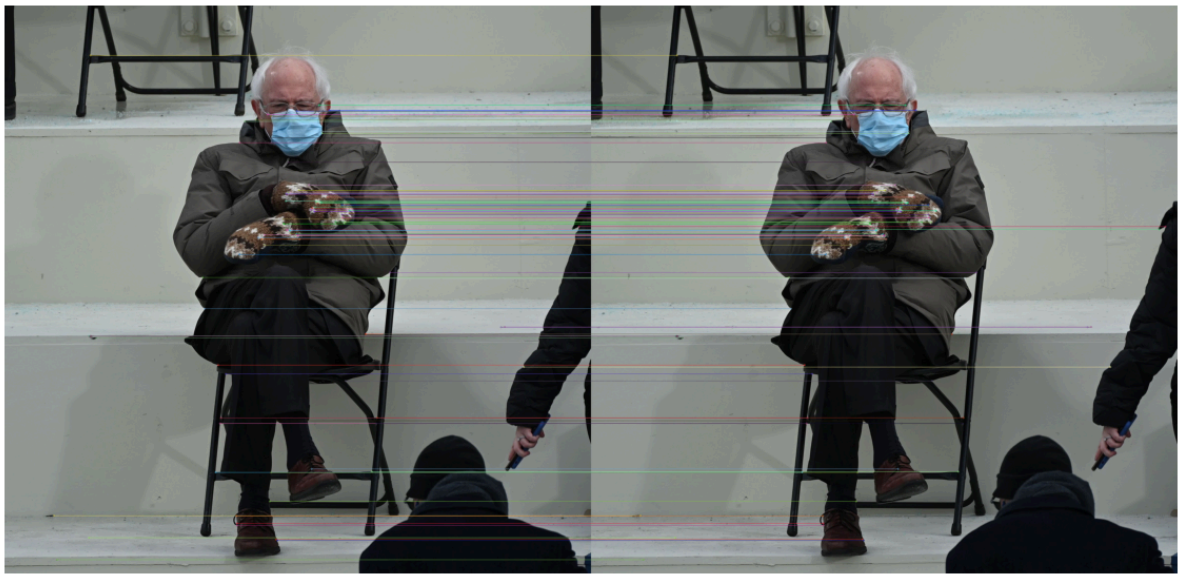
In [12]:
```
# Function of draw matched image

# def draw_matches(img1, kp1, img2, kp2, good_matches):
#     # Draw the matches using OpenCV
#     matched_img = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, flags=2)

#     # Convert BGR to RGB for matplotlib display
#     matched_img = cv2.cvtColor(matched_img, cv2.COLOR_BGR2RGB)

#     # Create a figure to display the matches
#     plt.figure(figsize=(15, 7))
#     plt.imshow(matched_img)
#     plt.axis('off')
#     plt.show()

def draw_matches(img1, kp1, img2, kp2, good_matches):
```

```
    # Calculate the scaling factor to make the height of the two images the same
    height1, width1 = img1.shape[:2]
    height2, width2 = img2.shape[:2]

    # Determine the target height and calculate scaling factors
    target_height = min(height1, height2)
    scale1 = target_height / height1
    scale2 = target_height / height2

    # Resize images to have the same height
    img1_resized = cv2.resize(img1, (int(width1 * scale1), target_height))
    img2_resized = cv2.resize(img2, (int(width2 * scale2), target_height))

    # Update keypoints' positions based on the scaling
    kp1_scaled = [cv2.KeyPoint(k.pt[0] * scale1, k.pt[1] * scale1, k.size, k.angle, k.response, k.octave, k.class_i
    kp2_scaled = [cv2.KeyPoint(k.pt[0] * scale2, k.pt[1] * scale2, k.size, k.angle, k.response, k.octave, k.class_i

    # Draw the matches using OpenCV
    matched_img = cv2.drawMatches(img1_resized, kp1_scaled, img2_resized, kp2_scaled, good_matches, None, flags=2)

    # Convert BGR to RGB for matplotlib display
    matched_img = cv2.cvtColor(matched_img, cv2.COLOR_BGR2RGB)

    # Create a figure to display the matches
    plt.figure(figsize=(15, 7))
    plt.imshow(matched_img)
    plt.axis('off')
    plt.show()
```

In [13]:
```
draw_matches(image, keypoints_cv2, image, keypoints_cv2, good_matches_ratio_test)
```



In [14]:
```
draw_matches(image, keypoints_cv2, image, keypoints_cv2, matches_ssd)
```

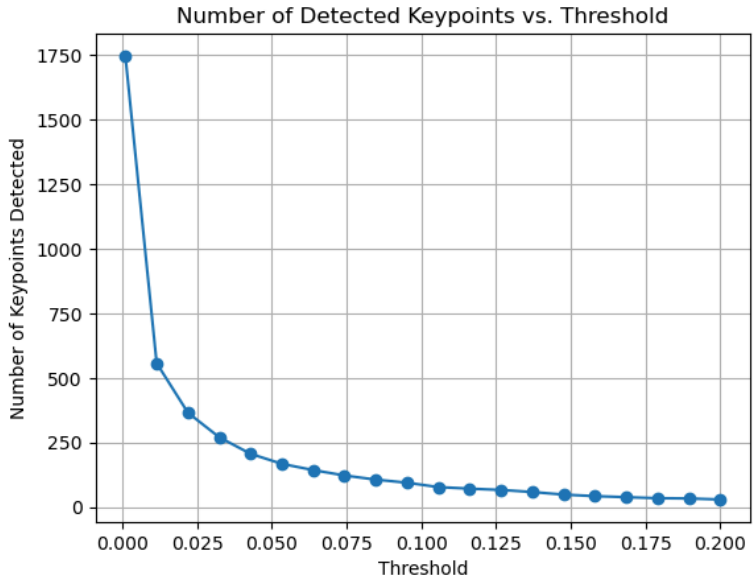


Step 4: Visualize how to choose the threshold value

In [15]:
```
# Plot the number of detected keypoints as a function of the threshold value.


def plot_keypoints_threshold_variation(image, threshold_start, threshold_end, num_steps):
    """
    Parameters:
    - image: The input image on which to perform Harris corner detection.
    - threshold_start: The start value of the threshold range.
    - threshold_end: The end value of the threshold range.
    - num_steps: The number of steps in the threshold steps.
    """

    thresholds = np.linspace(threshold_start, threshold_end, num=num_steps)
    num_keypoints = []

    for thresh in thresholds:
        keypoints, keypoints_orientations = harris_corner_detector(image, sobel_size=3, gaussian_size=5, sigma=0.5,
        num_keypoints.append(len(keypoints))

    print(num_keypoints)
    plt.plot(thresholds, num_keypoints, marker='o')
    plt.xlabel('Threshold')
    plt.ylabel('Number of Keypoints Detected')
    plt.title('Number of Detected Keypoints vs. Threshold')
    plt.grid(True)
    plt.show()
```

In [16]:
```
# Call the function to plot the number of keypoints vs. threshold values
plot_keypoints_threshold_variation(image, threshold_start=0.001, threshold_end=0.2, num_steps=20)
```

```
[1747, 557, 365, 270, 206, 167, 143, 122, 106, 94, 77, 71, 66, 58, 48, 42, 38, 34, 33, 29]
```



In [17]:
```
# Plots images with keypoints detected at various threshold levels.

def plot_images_with_keypoints(image, threshold_start, threshold_end, num_steps):
    """
    Parameters:
    - image: The input image on which to perform Harris corner detection.
    - threshold_start: The start value of the threshold range.
    - threshold_end: The end value of the threshold range.
    - num_steps: The number of steps in the threshold steps.
    """

    thresholds = np.linspace(threshold_start, threshold_end, num=num_steps)

    plt.figure(figsize=(20, 10))

    for i, thresh in enumerate(thresholds, 1):
        keypoints, keypoints_orientations = harris_corner_detector(image, threshold=thresh)

        # Draw keypoints on the image
        img_with_keypoints = np.copy(image)
        for y, x in keypoints:
            cv2.circle(img_with_keypoints, (x, y), radius=5, color=(0, 255, 0), thickness=-1)

        # Convert image to RGB if it's grayscale for plotting
        if len(img_with_keypoints.shape) == 2:
            img_with_keypoints = cv2.cvtColor(img_with_keypoints, cv2.COLOR_GRAY2BGR)
```

```
        # Plotting
        plt.subplot(2, 3, i)
        plt.imshow(cv2.cvtColor(img_with_keypoints, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
        plt.title(f'Threshold: {thresh:.4f}')
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

In [18]:
```
# Show many images with different threshold value

plot_images_with_keypoints(image, threshold_start=0.075, threshold_end=0.175, num_steps=6)
```

Step 5: Feature Matching and Compute

In [19]:
```
# Load the image should be matched

image = cv2.imread('image/bernieSanders.jpg')
image_bernie180 = cv2.imread('image/bernie180.jpg')
image_bernieBenefitBeautySalon = cv2.imread('image/bernieBenefitBeautySalon.jpeg')
image_BernieFriends = cv2.imread('image/BernieFriends.png')
image_bernieMoreblurred = cv2.imread('image/bernieMoreblurred.jpg')
image_bernieNoisy2 = cv2.imread('image/bernieNoisy2.png')
image_berniePixelated2 = cv2.imread('image/berniePixelated2.png')
image_bernieShoolLunch = cv2.imread('image/bernieShoolLunch.jpeg')
image_brighterBernie = cv2.imread('image/brighterBernie.jpg')
image_darkerBernie = cv2.imread('image/darkerBernie.jpg')
```

In [20]:
```
# Get keypoints for all image, using Harris Corner Detector function

keypoints, orientations = harris_corner_detector(image)
keypoints_bernie180, orientations_bernie180 = harris_corner_detector(image_bernie180)
keypoints_bernieBenefitBeautySalon, orientations_bernieBenefitBeautySalon = harris_corner_detector(image_bernieBene
keypoints_BernieFriends, orientations_BernieFriends = harris_corner_detector(image_BernieFriends)
keypoints_bernieMoreblurred, orientations_bernieMoreblurred = harris_corner_detector(image_bernieMoreblurred)
keypoints_bernieNoisy2, orientations_bernieNoisy2 = harris_corner_detector(image_bernieNoisy2)
keypoints_berniePixelated2, orientations_berniePixelated2 = harris_corner_detector(image_berniePixelated2)
keypoints_bernieShoolLunch, orientations_bernieShoolLunch = harris_corner_detector(image_bernieShoolLunch)
keypoints_brighterBernie, orientations_brighterBernie = harris_corner_detector(image_brighterBernie)
keypoints_darkerBernie, orientations_darkerBernie = harris_corner_detector(image_darkerBernie)

print(f"Number of keypoints detected: {len(keypoints)}")
print(f"Number of keypoints_bernie180 detected: {len(keypoints_bernie180)}")
print(f"Number of keypoints_bernieBenefitBeautySalon detected: {len(keypoints_bernieBenefitBeautySalon)}")
print(f"Number of keypoints_BernieFriends detected: {len(keypoints_BernieFriends)}")
print(f"Number of keypoints_bernieMoreblurred detected: {len(keypoints_bernieMoreblurred)}")
print(f"Number of keypoints_bernieNoisy2 detected: {len(keypoints_bernieNoisy2)}")
print(f"Number of keypoints_berniePixelated2 detected: {len(keypoints_berniePixelated2)}")
print(f"Number of keypoints_bernieShoolLunch detected: {len(keypoints_bernieShoolLunch)}")
print(f"Number of keypoints_brighterBernie detected: {len(keypoints_brighterBernie)}")
print(f"Number of keypoints_darkerBernie detected: {len(keypoints_darkerBernie)}")
```

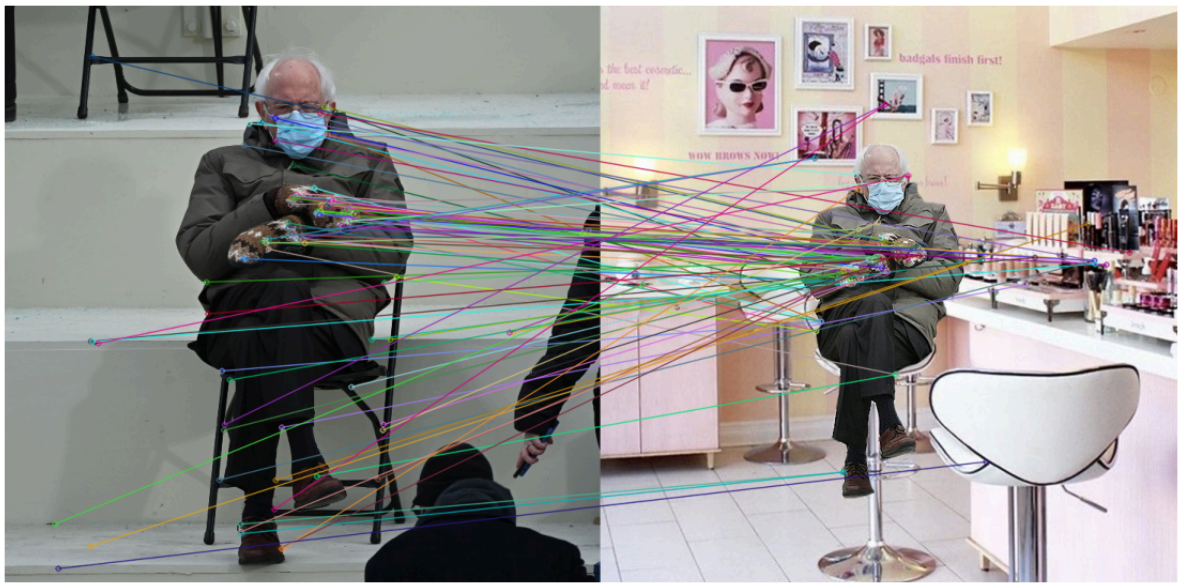
```
Number of keypoints detected: 94
Number of keypoints_bernie180 detected: 94
Number of keypoints_bernieBenefitBeautySalon detected: 110
Number of keypoints_BernieFriends detected: 113
Number of keypoints_bernieMoreblurred detected: 58
Number of keypoints_bernieNoisy2 detected: 3685
Number of keypoints_berniePixelated2 detected: 625
Number of keypoints_bernieShoolLunch detected: 153
Number of keypoints_brighterBernie detected: 265
Number of keypoints_darkerBernie detected: 1696
```

In [21]:
```
# Show some images with keypoints

image_bernieNoisy2_with_keypoints = draw_keypoints(image_bernieNoisy2, keypoints_bernieNoisy2)
plt.imshow(image_bernieNoisy2_with_keypoints[:, :, ::-1])
plt.axis('off')
plt.show()
```

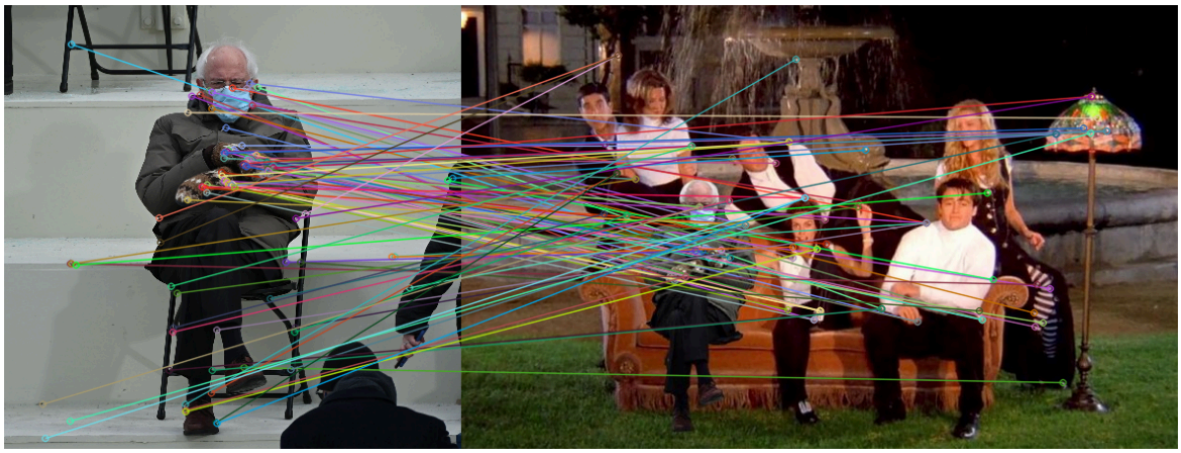
In [22]:
```
# Compute ORB descriptors for the keypoints

keypoints_cv2_bernie180, descriptors_bernie180 = compute_orb_descriptors(image_bernie180, keypoints_bernie180)
keypoints_cv2_bernieBenefitBeautySalon, descriptors_bernieBenefitBeautySalon = compute_orb_descriptors(image_bernie
keypoints_cv2_BernieFriends, descriptors_BernieFriends = compute_orb_descriptors(image_BernieFriends, keypoints_Ber
keypoints_cv2_bernieMoreblurred, descriptors_bernieMoreblurred = compute_orb_descriptors(image_bernieMoreblurred, k
keypoints_cv2_bernieNoisy2, descriptors_bernieNoisy2 = compute_orb_descriptors(image_bernieNoisy2, keypoints_bernie
keypoints_cv2_berniePixelated2, descriptors_berniePixelated2 = compute_orb_descriptors(image_berniePixelated2, keyp
keypoints_cv2_bernieShoolLunch, descriptors_bernieShoolLunch = compute_orb_descriptors(image_bernieShoolLunch, keyp
keypoints_cv2_brighterBernie, descriptors_brighterBernie = compute_orb_descriptors(image_brighterBernie, keypoints_
keypoints_cv2_darkerBernie, descriptors_darkerBernie = compute_orb_descriptors(image_darkerBernie, keypoints_darker
```

In [23]:
```
# Show the matching image

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_bernie180 = match_features(descriptors, descriptors_bernie180)

# Perform feature matching using SSD
matches_ssd_bernie180 = ssd_match_features(descriptors, descriptors_bernie180)

# Original vs bernie180
draw_matches(image, keypoints_cv2, image_bernie180, keypoints_cv2_bernie180, good_matches_ratio_test_bernie180)
draw_matches(image, keypoints_cv2, image_bernie180, keypoints_cv2_bernie180, matches_ssd_bernie180)
```

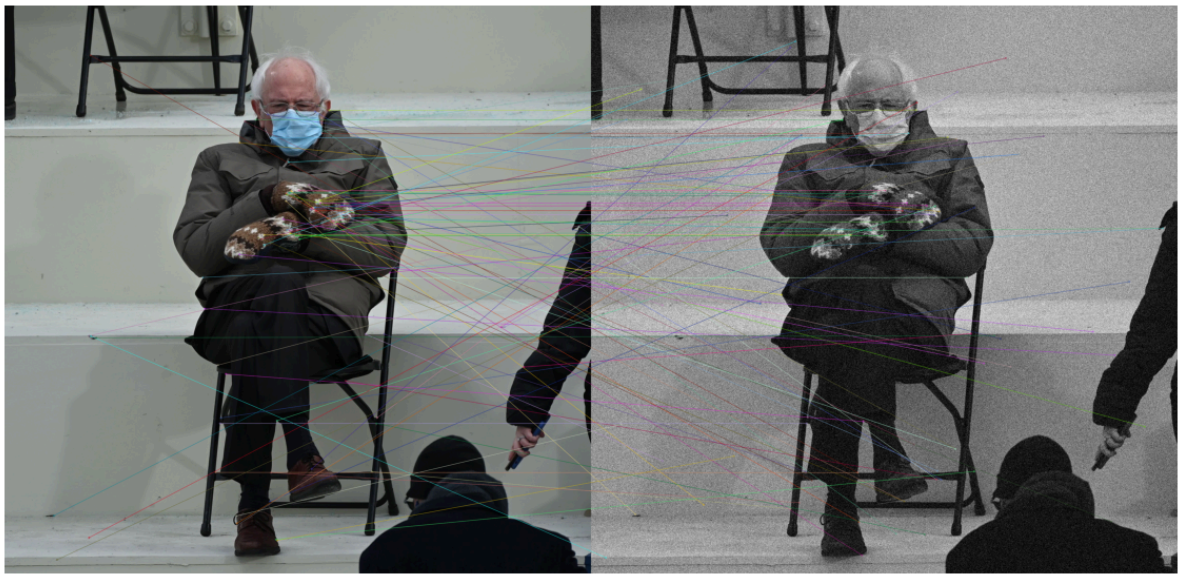
In [24]:
```
# Original vs bernieBenefitBeautySalon

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_bernieBenefitBeautySalon = match_features(descriptors, descriptors_bernieBenefitBeautySalon

# Perform feature matching using SSD
matches_ssd_bernieBenefitBeautySalon = ssd_match_features(descriptors, descriptors_bernieBenefitBeautySalon)


draw_matches(image, keypoints_cv2, image_bernieBenefitBeautySalon, keypoints_cv2_bernieBenefitBeautySalon, good_mat
draw_matches(image, keypoints_cv2, image_bernieBenefitBeautySalon, keypoints_cv2_bernieBenefitBeautySalon, matches_
```



In [25]:
```
# Original vs BernieFriends

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_BernieFriends = match_features(descriptors, descriptors_BernieFriends)

# Perform feature matching using SSD
matches_ssd_BernieFriends = ssd_match_features(descriptors, descriptors_BernieFriends)

draw_matches(image, keypoints_cv2, image_BernieFriends, keypoints_cv2_BernieFriends, good_matches_ratio_test_Bernie
draw_matches(image, keypoints_cv2, image_BernieFriends, keypoints_cv2_BernieFriends, matches_ssd_BernieFriends)
```



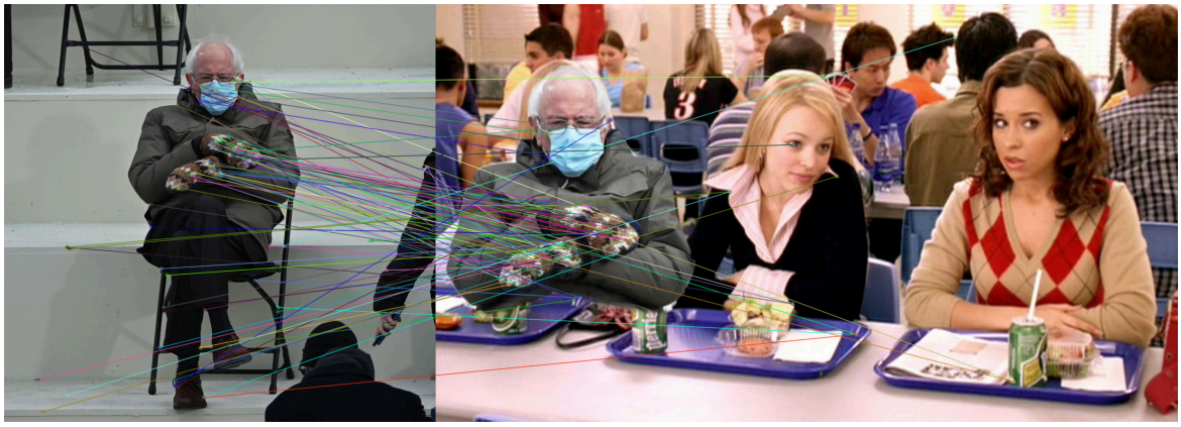
In [26]:
```
# Original vs _bernieNoisy2

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_bernieNoisy2 = match_features(descriptors, descriptors_bernieNoisy2)

# Perform feature matching using SSD
matches_ssd_bernieNoisy2 = ssd_match_features(descriptors, descriptors_bernieNoisy2)

draw_matches(image, keypoints_cv2, image_bernieNoisy2, keypoints_cv2_bernieNoisy2, good_matches_ratio_test_bernieNo
draw_matches(image, keypoints_cv2, image_bernieNoisy2, keypoints_cv2_bernieNoisy2, matches_ssd_bernieNoisy2)
```



In [27]:
```
# Original vs _berniePixelated2
```

```
# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_berniePixelated2 = match_features(descriptors, descriptors_berniePixelated2)

# Perform feature matching using SSD
matches_ssd_berniePixelated2 = ssd_match_features(descriptors, descriptors_berniePixelated2)

draw_matches(image, keypoints_cv2, image_berniePixelated2, keypoints_cv2_berniePixelated2, good_matches_ratio_test_
draw_matches(image, keypoints_cv2, image_berniePixelated2, keypoints_cv2_berniePixelated2, matches_ssd_berniePixela
```

In [28]:
```
# Original vs _bernieShoolLunch

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_bernieShoolLunch = match_features(descriptors, descriptors_bernieShoolLunch)

# Perform feature matching using SSD
matches_ssd_bernieShoolLunch = ssd_match_features(descriptors, descriptors_bernieShoolLunch)

draw_matches(image, keypoints_cv2, image_bernieShoolLunch, keypoints_cv2_bernieShoolLunch, good_matches_ratio_test_
draw_matches(image, keypoints_cv2, image_bernieShoolLunch, keypoints_cv2_bernieShoolLunch, matches_ssd_bernieShoolL
```



In [29]:
```
# Original vs _brighterBernie

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_brighterBernie = match_features(descriptors, descriptors_brighterBernie)

# Perform feature matching using SSD
matches_ssd_brighterBernie = ssd_match_features(descriptors, descriptors_brighterBernie)

draw_matches(image, keypoints_cv2, image_brighterBernie, keypoints_cv2_brighterBernie, good_matches_ratio_test_brig
draw_matches(image, keypoints_cv2, image_brighterBernie, keypoints_cv2_brighterBernie, matches_ssd_brighterBernie)
```

In [30]:
```
# Original vs _darkerBernie

# Perform feature matching using Lowe's ratio test
good_matches_ratio_test_darkerBernie = match_features(descriptors, descriptors_darkerBernie)

# Perform feature matching using SSD
matches_ssd_darkerBernie = ssd_match_features(descriptors, descriptors_darkerBernie)

draw_matches(image, keypoints_cv2, image_darkerBernie, keypoints_cv2_darkerBernie, good_matches_ratio_test_darkerBe
draw_matches(image, keypoints_cv2, image_darkerBernie, keypoints_cv2_darkerBernie, matches_ssd_darkerBernie)
```