Coursework 2: Local Feature Detection and matching for object recognition

1. + 2. Harris Corner detector:

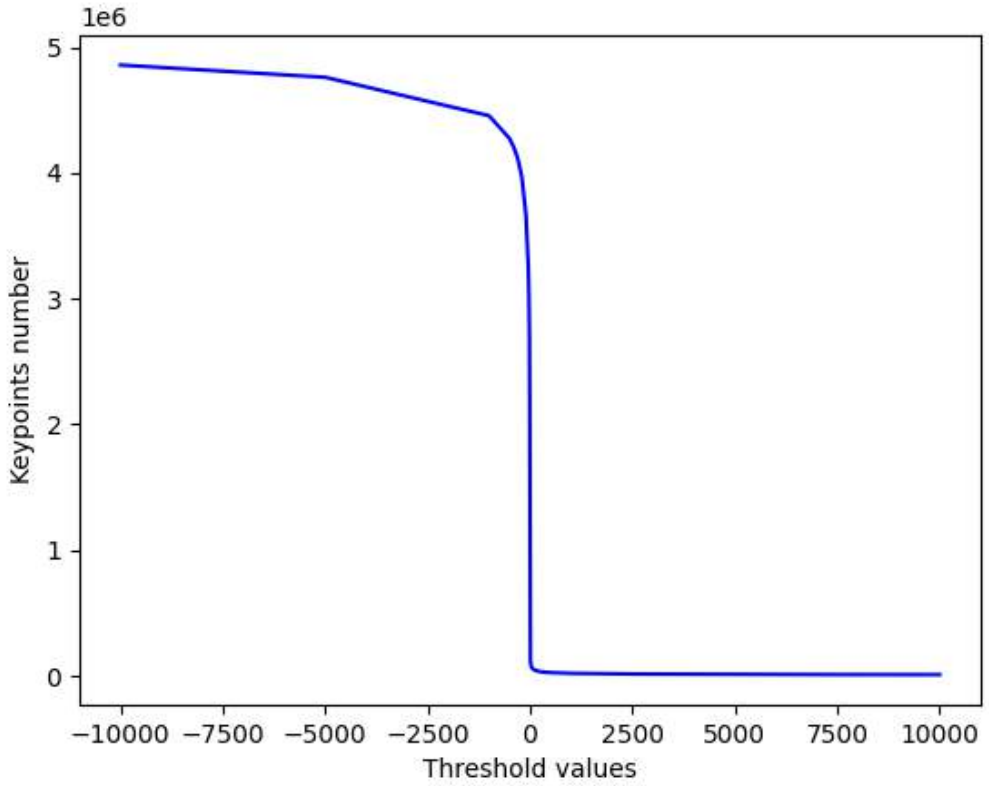
I have implemented two functions for the HarrisPointsDetector, one which calls another. The first one calculates the x and y derivatives with the help of the Sobel operator, applies the Gaussian mask, and makes use of the border reflect image padding in both those operations. First the Harris matrix M is computed and then the corner strength function R for every pixel in the input image, plus the orientation in degrees at every pixel is calculated.

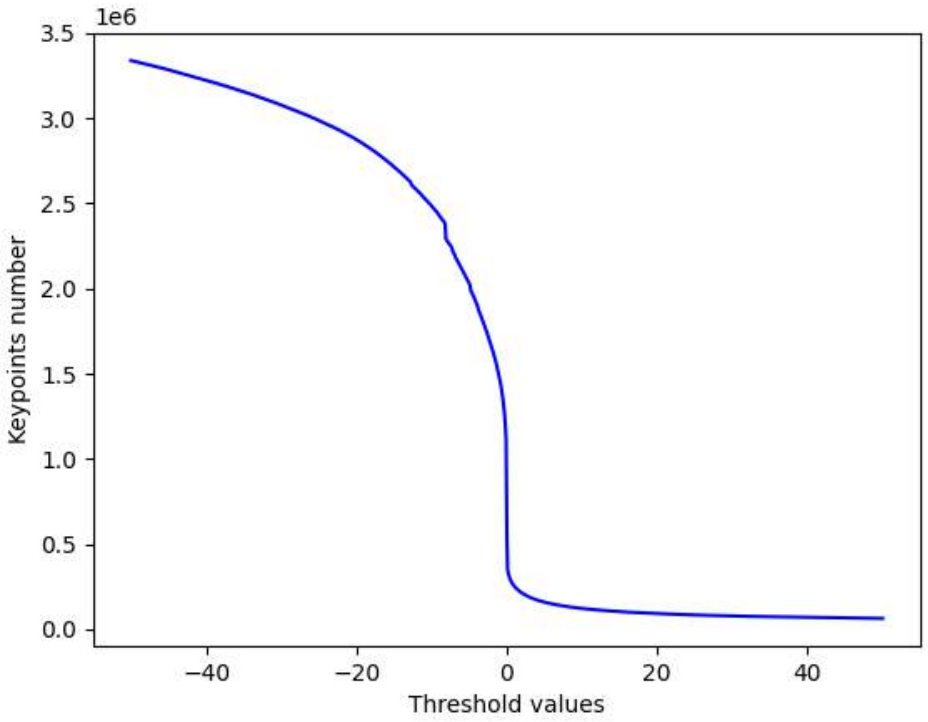
The second function grayscale the image, blurs it if we want, and selects the strongest interest points from the function R, in a 7x7 local maxima neighbourhood and above a selected threshold.

3. Image Derivatives 4. Harris Corner Response and 5. Interest points / Keypoints:

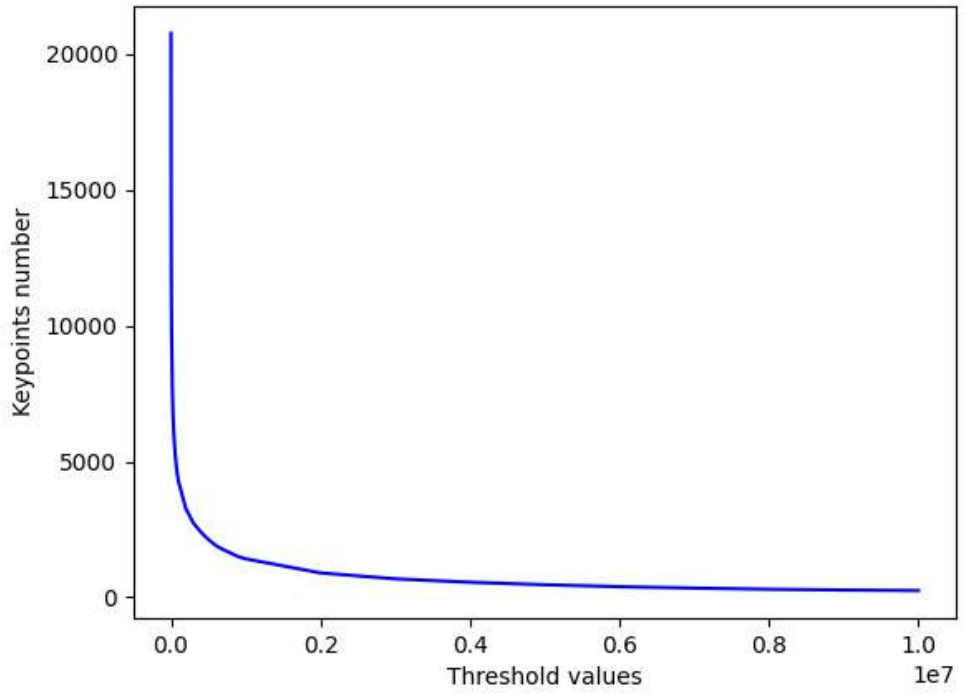
The derivatives are calculated in the HarrisStrengthFunction, together with the Harris corner response R function. While the strongest interest points are selected in the HarrisPointsDetector function, chosen with a hyperparameter, a threshold.

When plotting the interest points count with the threshold values, we notice that there is an abrupt shift at around 0, where we consider the detected corners actual keypoints, and from there there are only slight changes going up the threshold value.





While there are indeed seemingly small changes in the first plot, in the key points number from 50 upwards (keep in mind the keypoints number are at a scale of 1e6), I decided on using a higher threshold as it is noticeable in the drawn interest points on the image, getting rid of as many unnecessary and possibly incorrect key points for detecting just Bernie. The plot below shows the change from 1e3 to 1e7 in keypoints number, noticeable how there actually are pretty big changes even at this high of a threshold.



Below I have used the threshold 1e4, and we notice that even if it seemed alright from the plot, the threshold should be much higher as it is highlighted in the plot above.

For the next image I used the threshold 9e6:

Next, I tested a blurred input image, with threshold 2.5e4 and the results were as expected (if I blurred the original image, this number would have been significantly lower, and have a different outcome on the detected corners, lowering the noise impact, thus having less False Positives detected, but also less True Positives). This also is dependent on the threshold used.

6. ORB features plus comparison with OpenCV ORB built-in detector:

The function featureDescriptor takes the image and key points from my implemented HarrisPointsDetector and, using ORB Local Feature constructor from OpenCV, computes descriptors for them. The function orbDetect computes, with the help of the built-in score type features of the ORB constructor in OpenCV, Harris and FAST, both keypoints and descriptors for the images.

7. SSD and Ratio test:

SSDFeatureMatcher and RatioFeatureMatcher matching functions match two pairs of descriptors, one based on the sum of squared differences and the other one also having a ratio test distance to get rid of ambiguous and possibly wrong matches.

8. Experiments, Parameters and Comparisons:

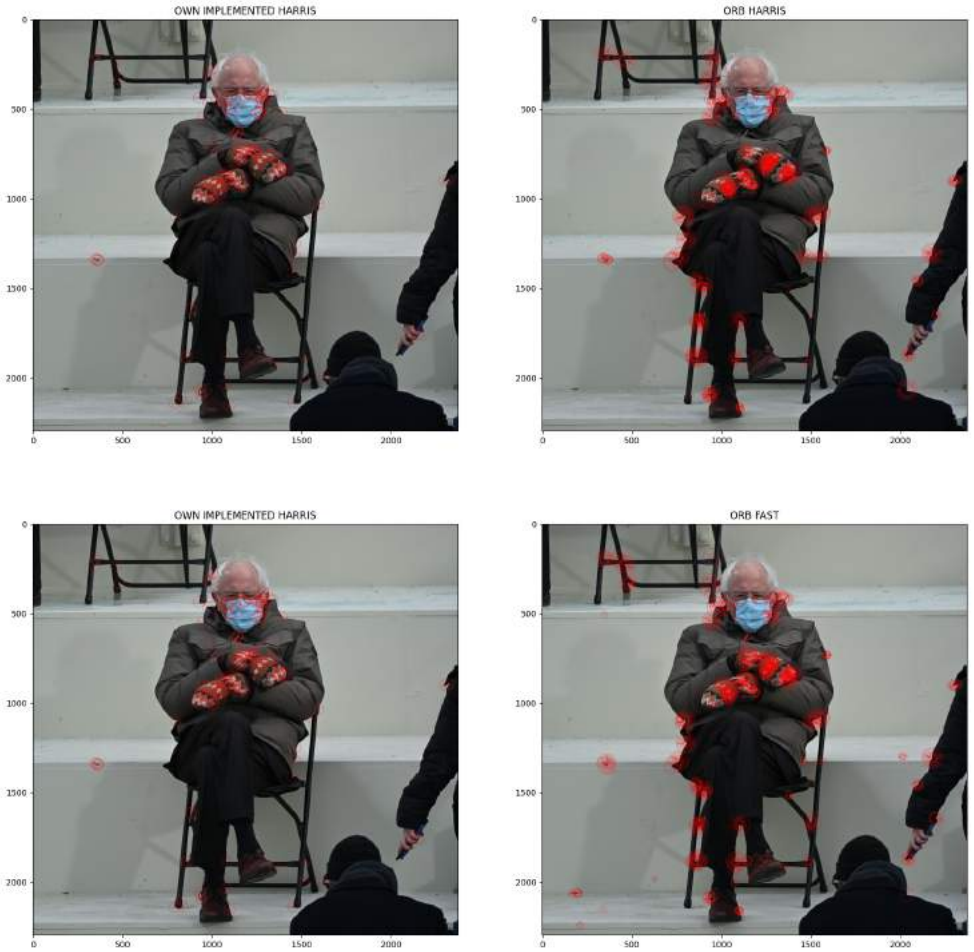
Threshold, ratio threshold, sobel and gaussian parameters:

As I said above, the threshold varies based on if we blur the original image or not. Blurring it makes us need a smaller threshold to still detect the True Positive edges while getting rid of the wrong ones caused by noise. If we do not blur the image then we need a higher threshold to try to get rid of false corners. The threshold for the ratio dictates how precise we want our matches to be. And the parameters for sobel and gaussian have an impact on the detected keypoints by the Harris detector.

Implemented HarrisPointsDetector vs Harris and FAST from OpenCV:

All three of the detectors mainly detected Bernie's mittens, face with the mask and parts of his outline, with differences being in what corners were detected in the background. I also do believe that the ORB built-in detectors do not filter out interest points that are very close to each other (as I did in my implementation local maximum in a 7x7 neighbourhood), and that is why there are a lot of them

overlapping. Next, I noticed that the built-in detectors do not detect as many false edges that are caused by noise so I presumed that they use something for noise removal, like gaussian blurring of the original image, so that is another reason for me deciding on Gaussian blurring the original image. After, I started getting better results that match with the Harris and FAST implementations from ORB. Threshold used was 2e4 (left is own implemented and right is HARRIS and FAST):



SSD vs Ratio:

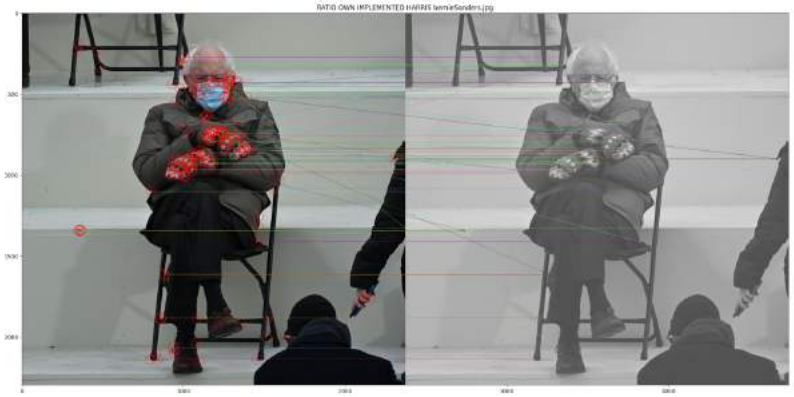
Both of them perform reasonably well, but with the SSD matching it is better to have a bigger threshold for the interest points, even if risk of losing some important of them, as it matches all of them to the second image, while with the Ratio matching we can afford to have a little lower threshold, as we match only the least vague keypoints. So, while the detections of the two matchers contain mostly the same True Positives, we get rid of a lot of ambiguous False Positives by using the Ratio Matcher, thus making it better.

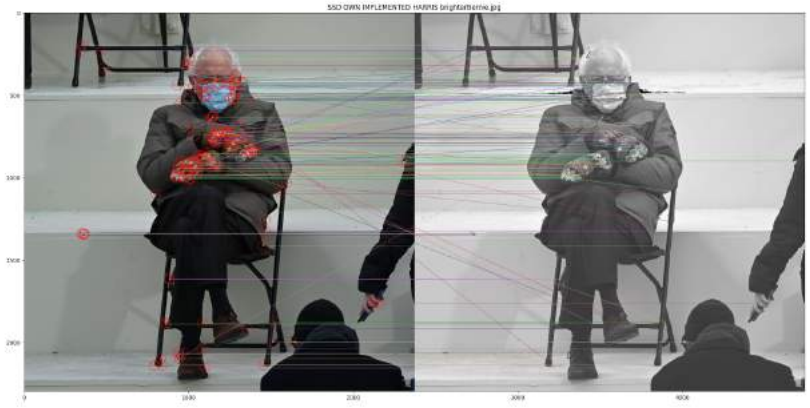
Below are outputs for all the provided testing images for matching, with threshold 2e4 (SSD then Ratio for each respectively):

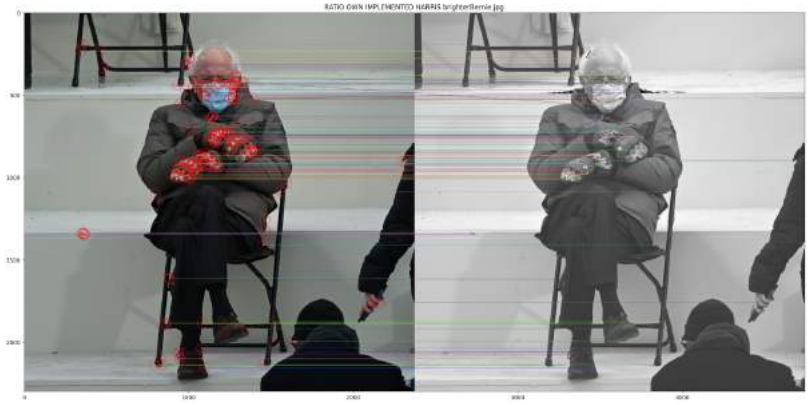

SSD OWN IMPLEMENTED HARRIS bernieSandersOriginal.jpg


RATIO OWN IMPLEMENTED HARRIS bernieSandersOriginal.jpg


SSD OWN IMPLEMENTED HARRIS bernieSanders.jpg

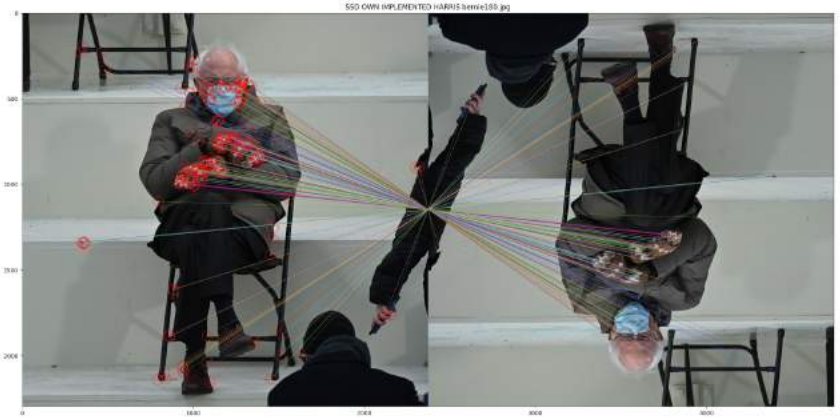

RATIO OWN IMPLEMENTED HARRIS bernieSanders.jpg

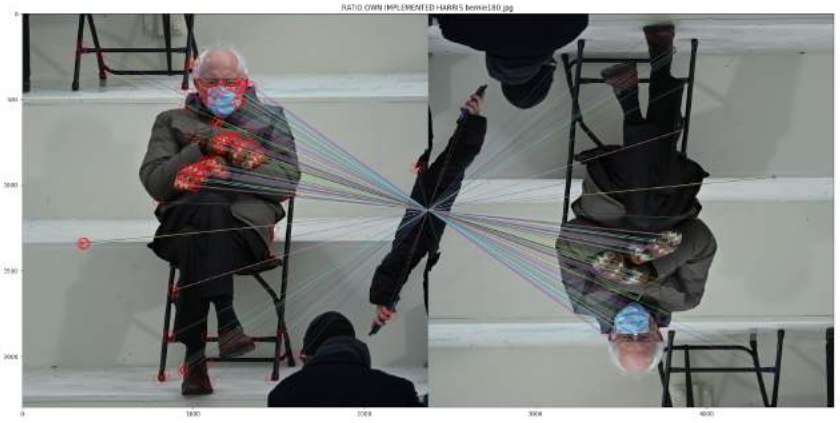

SSD OWN IMPLEMENTED HARRIS brightGetBernie.jpg

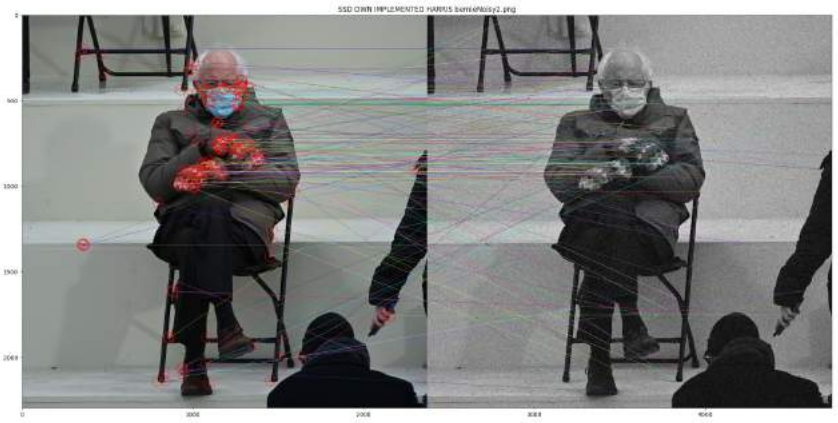

RATIO OWN IMPLEMENTED HARRIS brighterBernie.jpg


SSD OWN IMPLEMENTED HARRIS darkerBernie.jpg

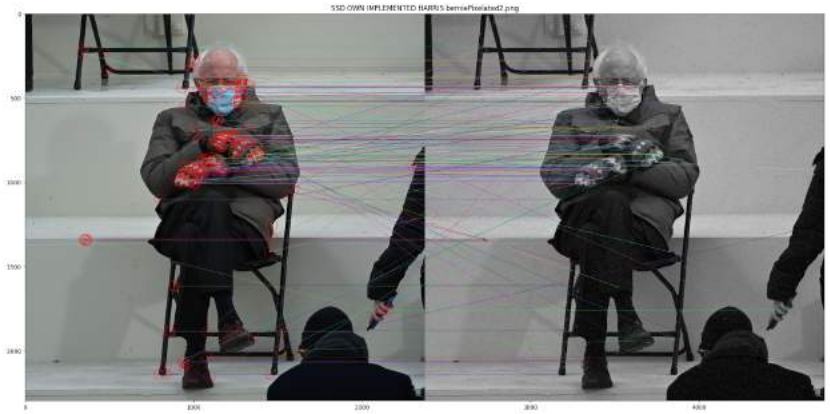

RATIO OWN IMPLEMENTED HARRIS darkerBernie.jpg


SSD OWN IMPLEMENTED HARRIS bernie180.jpg

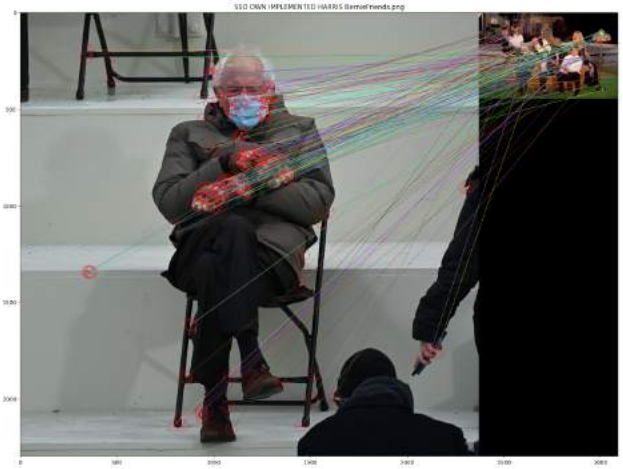

RATIO OWN IMPLEMENTED HARRIS bernie180.jpg

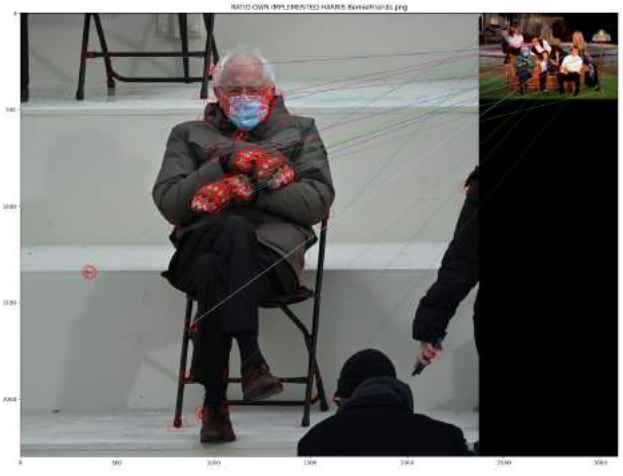

SSD OWN IMPLEMENTED HARRIS bernieNoisy2.png


RATIO OWN IMPLEMENTED HARRIS bernieNoisy2.png


SSD OWN IMPLEMENTED HARRIS bernieFloelated2.png

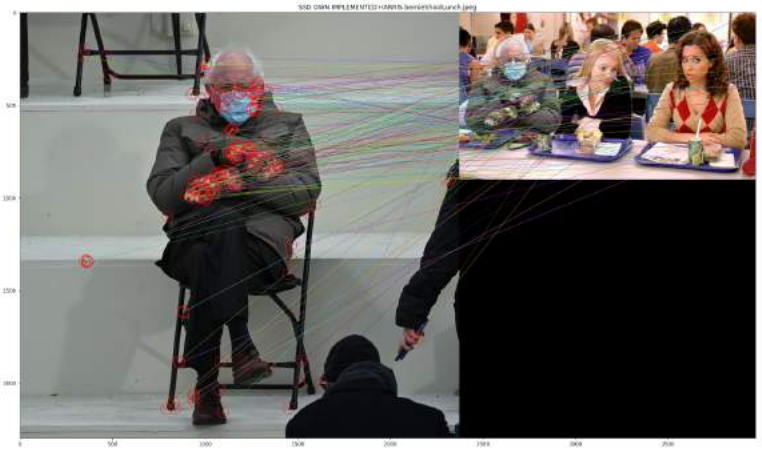

RATIO OWN IMPLEMENTED HARRIS bernieFloelated2.png


SSD OWN IMPLEMENTED HARRIS BernieFriends.png


RATIO OWN IMPLEMENTED HARRIS BernieFriends.png


SSD OWN IMPLEMENTED HARRIS bernieBeneFitBeautySalon.jpeg


RATIO OWN IMPLEMENTED HARRIS bernieBeneFitBeautySalon.jpeg


SSD OWN IMPLEMENTED HARRIS bernieWalkLunch.jpeg


RATIO OWN IMPLEMENTED HARRIS bernieWalkLunch.jpeg

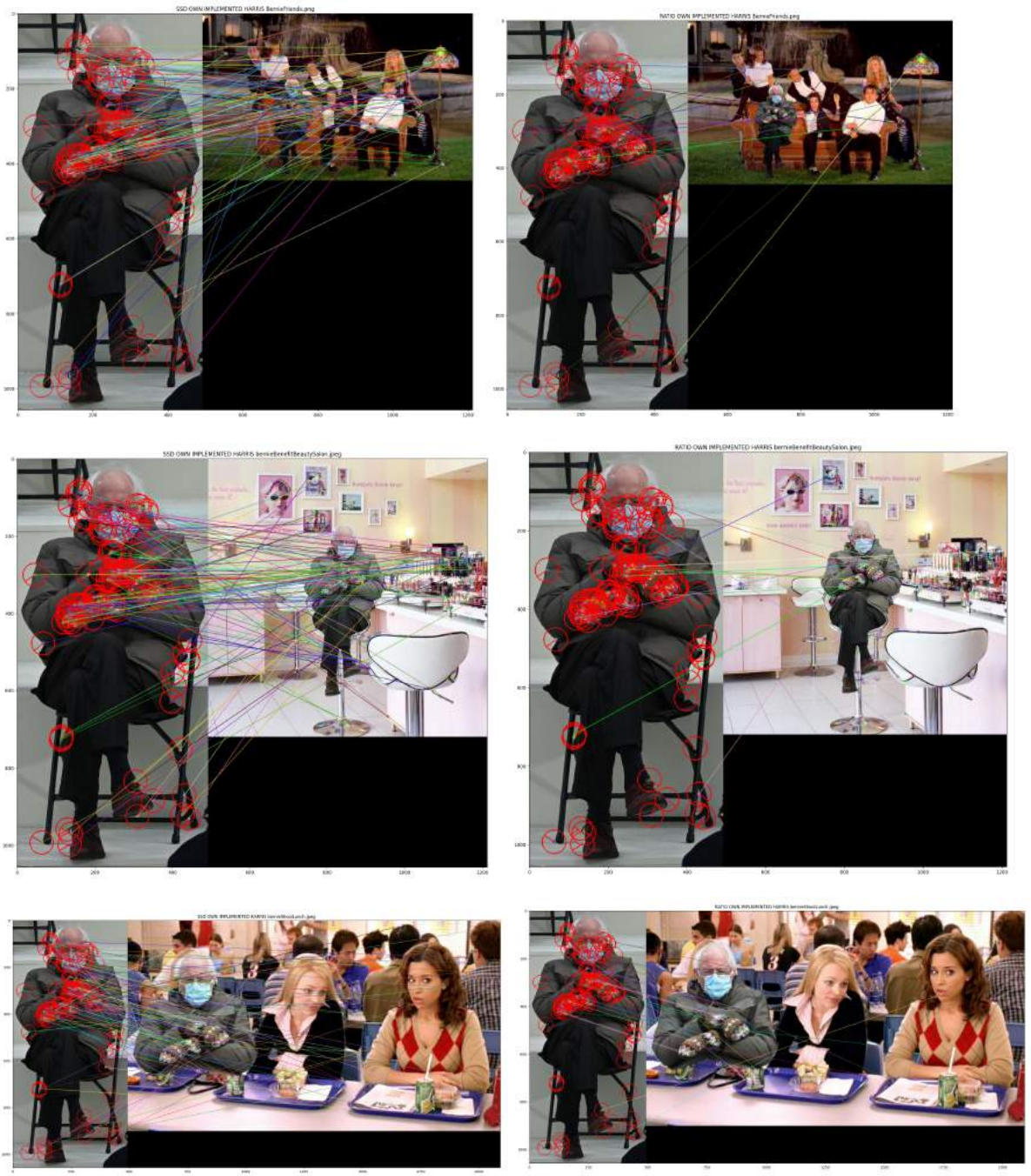
Here we notice that most in the pictures where it is the same photo but augmented, there are a number of correct edges detected, with the NoisyBernie being the hardest ones to detect, and a decently good performance on the Bernie180. We can see that the Ratio matching looks more clean as it gets rid of a lot of ambiguous and vague detections from the Euclidean distance, thus removing most of the False Positives from the matching.

As for the last 3 pictures, the BeautySalon, Friends and SchoolLunch Bernie have a fall on Precision, they still manage to detect some edges, mainly the mask and the mittens.

Extra Experiment:

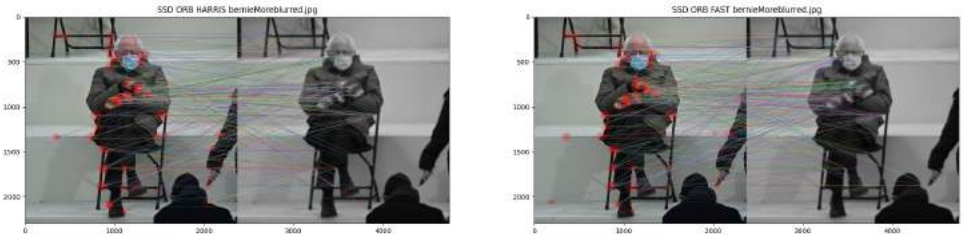
Because of the poorer performance of the last 3 photos, and also that there is a significant number of interest points found in the background of the photo (around Bernie in our case), which would not necessarily help in detecting Bernie in photos. I decided on testing with an input photo of just a cropped Bernie from the original image.

After testing, the results were not much better (or at least visibly better), in comparison with the first ones, possibly because of the resolution difference of the images, for which we would need to implement more preprocessing of the images. But overall, this would be a better way to look just for Bernie in random images.



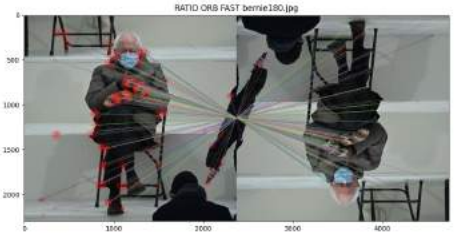
9. Conclusion:

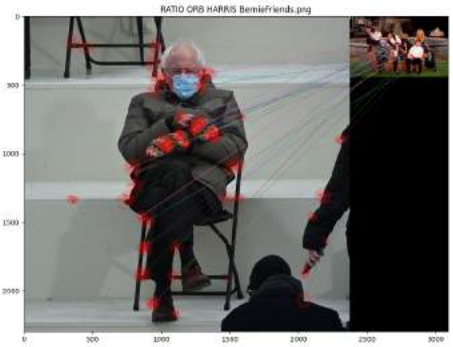
Overall, the interest points were detected fairly well, and the descriptors and matching worked well too. I had a problem with reading the image bernieMoreblurred.jpg as it resulted in a ValueError. I believe this was caused because the HarrisPointsDetector could not detect any key points in the image and thus there were no keypoints and descriptors to work with. Although ORB built-in functions managed to detect keypoints.



There are 3 main parameters that majorly change the results. The threshold for the key points, the gaussian blur of the original image, and the ratio threshold of the Ratio matching function. The results are similar to the ones from the built-in ORB detector, and the matching works decently but more preprocessing could be added. Additionally, the sobel and gaussian mask for calculating the Harris corner function R could be fine-tuned to possibly get better results.

Lastly, between Harris and FAST detectors, I could not perceive a significant difference in the matching of the key points by descriptors. The only difference I could notice is that in the original Bernie image, FAST detects more interest points caused by noise and clutter in the background, compared to Harris (as it can be seen in the down left corner and near the arm of the man on the right). So you could possibly argue that Harris did a better job at finding Bernie. I put only the Ratio matcher photos from them because that is where the differences can be spotted.





10. Code:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
import scipy.ndimage
import scipy.spatial
import cv2

def HarrisStrengthFunction(img):
    """ Compute Harris corner strength function """
    x = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=3, borderType=cv2.BORDER_REFLECT)
    y = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=3, borderType=cv2.BORDER_REFLECT)
    xy = x * y
```

```
    x = cv2.GaussianBlur(x, (5,5), 0.5, borderType=cv2.BORDER_REFLECT)
    y = cv2.GaussianBlur(y, (5,5), 0.5, borderType=cv2.BORDER_REFLECT)
    xy = cv2.GaussianBlur(xy, (5,5), 0.5, borderType=cv2.BORDER_REFLECT)
    xx = x**2
    yy = y**2

    detM = (xx * yy) - (xy ** 2)
    traceM = xx + yy
    R = detM - 0.05 * (traceM ** 2)

    orientation =np.arctan2(y, x)
    orientation = orientation * 180 / np.pi

    return R, orientation

def HarrisPointsDetector(img, blur_img=False, th=0):
    """ Detect Harris points in image """
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    if blur_img:
        img_gray = cv2.GaussianBlur(img_gray, (7,7), 7,
borderType=cv2.BORDER_REFLECT)

    R, orientation = HarrisStrengthFunction(img_gray)

    max_filter = scipy.ndimage.maximum_filter(R, size=7)
    local_maxima = (R >= th) & (R == max_filter)
    output = local_maxima * 255

    kp = []
    for j in range(output.shape[0]):
        for i in range(output.shape[1]):
            if output[j, i] > 0:
                kp.append(cv2.KeyPoint(i, j, 60, orientation[j, i]))

    return kp, R

def featureDescriptor(img, kp):
    """ Compute feature descriptors for keypoints """
    orb = cv2.ORB_create()
    kp, des = orb.compute(img, kp)
    return des, kp

def orbDetect(img, method='harris'):
    """ Detect keypoints in image and get descriptors for them """
    if method == 'harris':
        orb = cv2.ORB_create(scoreType=cv2.ORB_HARRIS_SCORE)
    elif method == 'fast':
        orb = cv2.ORB_create(scoreType=cv2.ORB_FAST_SCORE)
    else:
        return None
    kp = orb.detect(img, None)
    kp, des = orb.compute(img, kp)
    return kp, des
```

```
def SSDFeatureMatcher(des1, des2):
    """ Match features using SSD distance """
    dist = scipy.spatial.distance.cdist(des1, des2, 'euclidean')
    matches = []

    for i in range(len(des1)):
        m = np.argmin(dist[i])
        matches.append(cv2.DMatch(i, m, dist[i, m]))

    return matches

def RatioFeatureMatcher(des1, des2, ratio_threshold):
    """ Match features using ratio test """
    dist = scipy.spatial.distance.cdist(des1, des2, 'euclidean')
    matches = []

    for i in range(len(des1)):
        ordered = np.argsort(dist[i])
        first, second = (dist[i][ordered[0]], dist[i][ordered[1]])

        ratio = float(first) / float(second)
        if ratio < ratio_threshold:
            matches.append(cv2.DMatch(i, ordered[0], dist[i, ordered[0]]))
    return matches

def getAllFileNames(path):
    """ Get all file names in a directory """
    import os
    files = []
    for f in os.listdir(path):
        if os.path.isfile(os.path.join(path, f)):
            files.append(f)
    return files

## Parameters
threshold = 2e4
do_blur = True
# threshold = 9e6
# do_blur = False
ratio_threshold = 0.85

## Get images
image = cv2.imread("bernieSanders.jpg", cv2.IMREAD_COLOR)
image_copy = image.copy()
image_harris = image.copy()
image_fast = image.copy()

## Detect Harris points
kp, R = HarrisPointsDetector(image, do_blur, threshold)
des, kp = featureDescriptor(image, kp)

## Plot various threshold values with keypoint numbers
# threshold_values = [-10000, -5000, -1000, -500, -400, -300, -200, -100]
# # threshold_values = []
# threshold_values.extend([x*0.1 for x in range(-500, 501)])
```

```
# threshold_values.extend([60, 70, 80, 90, 100, 200, 300, 400, 500, 1000, 2500,
5000, 7500, 10000])
# # threshold_values.extend([1e3, 2e3, 3e3, 4e3, 5e3, 6e3, 7e3, 8e3, 9e3, 1e4, 2e4,
3e4, 4e4, 5e4, 6e4, 7e4, 8e4, 9e4, 1e5, 2e5, 3e5, 4e5, 5e5, 6e5, 7e5, 8e5, 9e5,
1e6, 2e6, 3e6, 4e6, 5e6, 6e6, 7e6, 8e6, 9e6, 1e7])
# axis_x = []
# axis_y = []
# for t in threshold_values:
#     axis_x.append(t)
#     axis_y.append((R >= t).sum())

# plt.plot(axis_x, axis_y, color='b', alpha=1)
# plt.xlabel("Threshold values")
# plt.ylabel("Keypoints number")
# plt.show()

## Detect Harris points using ORB built-in functions
kp_harris, des_harris = orbDetect(image, 'harris')
kp_fast, des_fast = orbDetect(image, 'fast')

## Draw keypoints on image
cv2.drawKeypoints(image_copy, kp, image_copy, color=(0, 0, 255),
flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)
cv2.drawKeypoints(image_harris, kp_harris, image_harris, color=(0, 0, 255),
flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)
cv2.drawKeypoints(image_fast, kp_fast, image_fast, color=(0, 0, 255),
flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

## Plot images with keypoints
cv2.cvtColor(image, cv2.COLOR_BGR2RGB, image)
cv2.cvtColor(image_copy, cv2.COLOR_BGR2RGB, image_copy)
cv2.cvtColor(image_harris, cv2.COLOR_BGR2RGB, image_harris)
cv2.cvtColor(image_fast, cv2.COLOR_BGR2RGB, image_fast)
fig, ax = plt.subplots(1, 2, figsize=(20, 16))
ax[0].imshow(image_copy)
ax[0].set_title("OWN IMPLEMENTED HARRIS")
ax[1].imshow(image_harris)
ax[1].set_title("ORB HARRIS")
plt.savefig('IMPLEMENTED|HARRIS.png', dpi=100)
#plt.show()
fig, ax = plt.subplots(1, 2, figsize=(20, 16))
ax[0].imshow(image_copy)
ax[0].set_title("OWN IMPLEMENTED HARRIS")
ax[1].imshow(image_fast)
ax[1].set_title("ORB FAST")
plt.savefig('IMPLEMENTED|FAST.png', dpi=100)
#plt.show()
cv2.cvtColor(image_copy, cv2.COLOR_RGB2BGR, image_copy)
cv2.cvtColor(image_harris, cv2.COLOR_RGB2BGR, image_harris)
cv2.cvtColor(image_fast, cv2.COLOR_RGB2BGR, image_fast)

## Repeat above steps for each image plus do matching with original image
files = getAllFileNames('images')
for f in files:
    try:
```

```
        img = cv2.imread("images/" + f, cv2.IMREAD_COLOR)
        kp_img, _ = HarrisPointsDetector(img, do_blur, threshold)
        des_img, kp_img = featureDescriptor(img, kp_img)
        kp_img1, des_img1 = orbDetect(img, 'harris')
        kp_img2, des_img2 = orbDetect(img, 'fast')

        matches = SSDFeatureMatcher(des, des_img)
        ratios = RatioFeatureMatcher(des, des_img, ratio_threshold)
        matches_harris = SSDFeatureMatcher(des_harris, des_img1)
        ratio_harris = RatioFeatureMatcher(des_harris, des_img1, ratio_threshold)
        matches_fast = SSDFeatureMatcher(des_fast, des_img2)
        ratio_fast = RatioFeatureMatcher(des_fast, des_img2, ratio_threshold)

        img_copy = cv2.drawMatches(image_copy, kp, img, kp_img, matches, None)
        img_harris = cv2.drawMatches(image_harris, kp_harris, img, kp_img1,
matches_harris, None)
        img_fast = cv2.drawMatches(image_fast, kp_fast, img, kp_img2, matches_fast,
None)

        img_copy_ratio = cv2.drawMatches(image_copy, kp, img, kp_img, ratios, None)
        img_harris_ratio = cv2.drawMatches(image_harris, kp_harris, img, kp_img1,
ratio_harris, None)
        img_fast_ratio = cv2.drawMatches(image_fast, kp_fast, img, kp_img2,
ratio_fast, None)

        cv2.drawKeypoints(image_copy, kp, image_copy, color=(0, 0, 255),
flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)
        cv2.cvtColor(img_copy, cv2.COLOR_BGR2RGB, img_copy)
        cv2.cvtColor(img_copy_ratio, cv2.COLOR_BGR2RGB, img_copy_ratio)

        cv2.cvtColor(img_harris, cv2.COLOR_BGR2RGB, img_harris)
        cv2.cvtColor(img_harris_ratio, cv2.COLOR_BGR2RGB, img_harris_ratio)
        cv2.cvtColor(img_fast, cv2.COLOR_BGR2RGB, img_fast)
        cv2.cvtColor(img_fast_ratio, cv2.COLOR_BGR2RGB, img_fast_ratio)
        cv2.cvtColor(img, cv2.COLOR_BGR2RGB, img)
        fig, ax = plt.subplots(1, 2, figsize=(25, 16))
        # ax.imshow(img_copy)
        # ax.set_title("SSD OWN IMPLEMENTED HARRIS " + f)
        ax[0].imshow(img_harris)
        ax[0].set_title("SSD ORB HARRIS " + f)
        ax[1].imshow(img_fast)
        ax[1].set_title("SSD ORB FAST " + f)
        plt.savefig('matches/' + f + '_SSD.png', dpi=100)
        plt.close()
        #plt.show()

        fig, ax = plt.subplots(1, 2, figsize=(25, 16))
        # ax.imshow(img_copy_ratio)
        # ax.set_title("RATIO OWN IMPLEMENTED HARRIS " + f)
        ax[0].imshow(img_harris_ratio)
        ax[0].set_title("RATIO ORB HARRIS " + f)
        ax[1].imshow(img_fast_ratio)
        ax[1].set_title("RATIO ORB FAST " + f)
        plt.savefig('matches/' + f + '_RATIO.png', dpi=100)
        plt.close()
```

```
        #plt.show()
    except ValueError:
        ## bernieMoreblurred.jpg image not readable due to a ValueError, because no
keypoints were found
        print(sys.exc_info()[0], " at file ", f)
```