Computer Vision Report

Qi Xiang

March 22, 2024

1 Interest Point Detection

I plot the number of detected points against varying threshold values for all the images to select a threshold that finds the best interest points. The aim is to find a balance between the number of key points and the quality of the key points. After plotting the graph for all the images, the final result is shown in Figure 1. The conclusion is around 0.04 and 0.05; the number of key points detected in all images stabilises. It provides a balance as the number of key points is reduced compared to smaller threshold values. So, I chose 0.045 as the threshold value empirically.

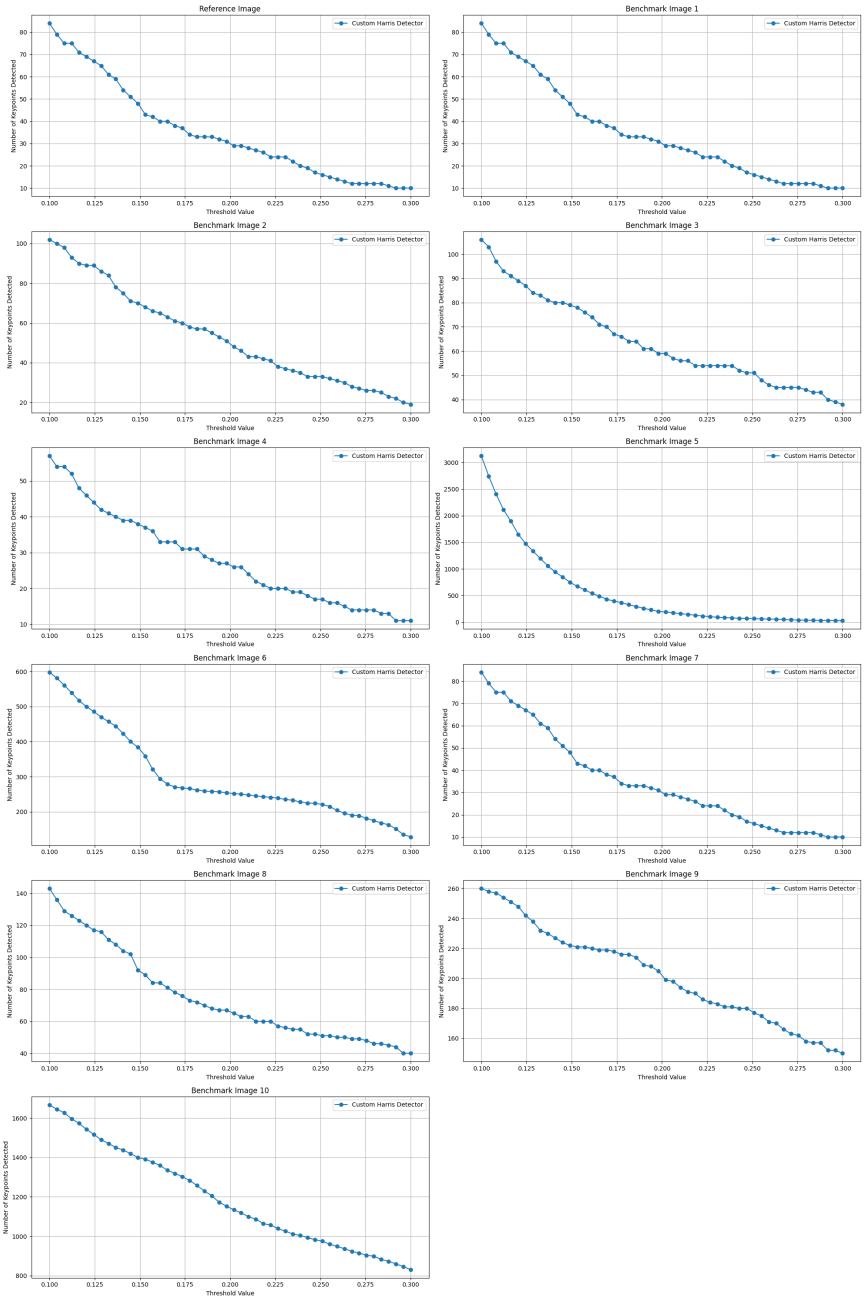

Figure 1: Interest points detected against varying threshold values

The same applies to darker bernie, where almost all the points on both images match.

Figure 2: Enter Caption

2 Feature Description

In this section, I compare the interest points detection result with that of my Harris detector to that of the built-in Harris detector. The comparison shows that the results are almost indistinguishable, except for some false negatives when detecting the chair. When detecting bernie, the three functions perform almost the same.

Figure 3: Interest points detected using custom Harris vs. built-in Harris and FAST

3 Feature matching

Here are the results of matching the features in the reference image against all the benchmark images.

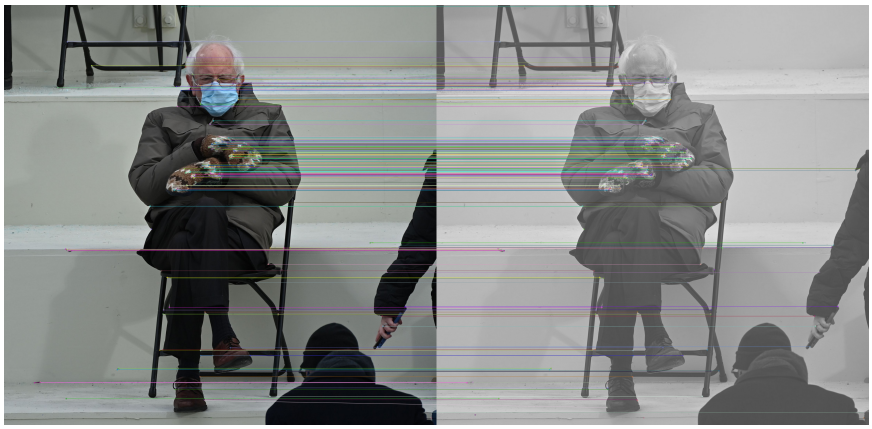

Figure 4: brighter bernie

When matching the reference image with bernieFriends, the algorithm struggles to find accurate correspondences between the two images. This could be due to several reasons: Bernie on the right-hand side image is at a small scale. Besides Bernie, the two images have limited shared visual content, making it difficult for the feature detection algorithm to find matches.

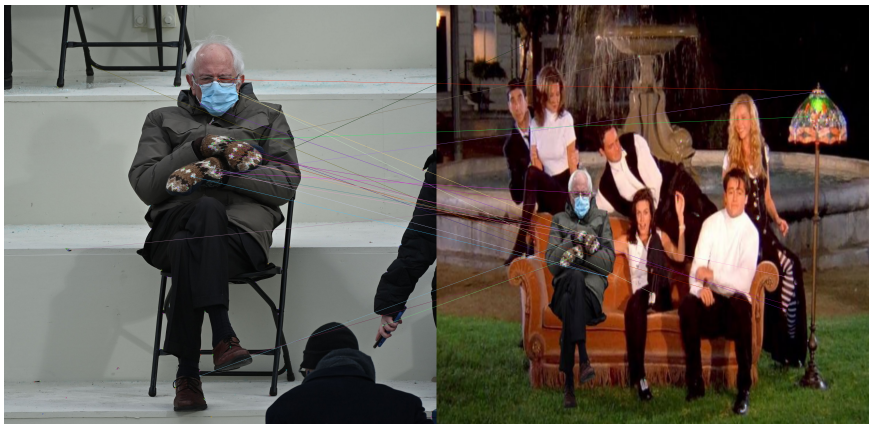

Figure 5: bernie friends

Below is the result of matching the 180-degree rotated image. The threshold for the ratio is 0.75. The ratio means that a good match should be significantly closer to the closest match than the second-closest match. And since my Harris detector is invariant to rotation, the interest points on both images perfectly match.

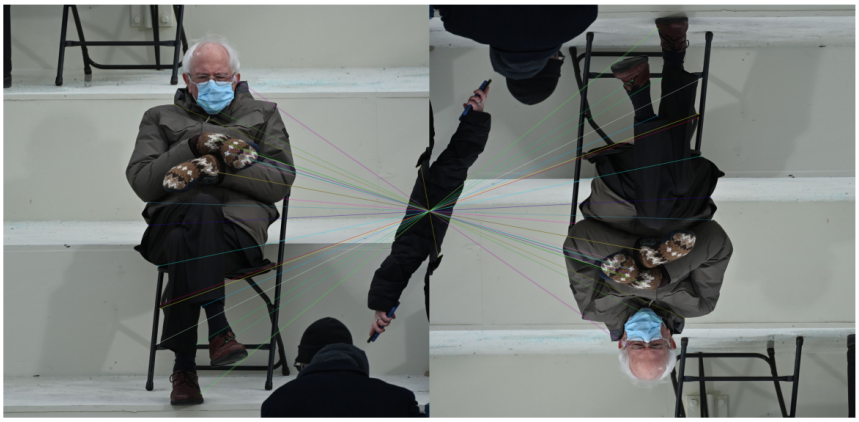

Figure 6: bernie 180

However, my Harris detector doesn't work when it comes to scaled images since it is not made to be invariant to scale.

Figure 7: bernie scaled

Figure 8: bernie scaled (2)

When it comes to noisy images, the detector also doesn't work. Noisy images produce false gradients and can create various local maxima.

Figure 9: noisy bernie

Figure 10: noisy bernie (2)

4 Appendix

```
import cv2
import numpy as np
from scipy.ndimage import gaussian_filter, maximum_filter, sobel, convolve
import matplotlib.pyplot as plt
import os

def HarrisPointsDetector(image, sigma=0.5, k=0.05, thresh=0.045):
    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = np.float64(gray)

    # Compute x and y derivatives using Sobel operator
    dx = sobel(gray, axis=1, mode='reflect')
    dy = sobel(gray, axis=0, mode='reflect')

    # Compute products of derivatives
    Ixx = dx**2
    Ixy = dx*dy
    Iyy = dy**2

    # Apply Gaussian filter to the products of derivatives
    Ixx = gaussian_filter(Ixx, sigma, mode='reflect')
    Ixy = gaussian_filter(Ixy, sigma, mode='reflect')
```

```
    Iyy = gaussian_filter(Iyy, sigma, mode='reflect')

    # Compute the Harris matrix M and corner response function R
    detM = Ixx * Iyy - Ixy ** 2
    traceM = Ixx + Iyy
    R = detM - k * (traceM ** 2)

    # Apply non-maximum suppression to find local maxima
    local_max = maximum_filter(R, size=(7,7), mode='reflect')
    R = np.where((R == local_max) & (R > thresh * R.max()), R, 0)

    # Find coordinates of local maxima
    keypoints = np.argwhere(R)

    orientation = np.degrees(np.arctan2(dy, dx))

    # Convert to OpenCV keypoint format and compute orientation
    keypoints_cv = [cv2.KeyPoint(float(x[1]), float(x[0]), 20, angle=float(orientati
                    for x in keypoints]

    return keypoints_cv

def featureDescriptor(image, keypoints):
  orb = cv2.ORB_create()

  keypoints, descriptors = orb.compute(image, keypoints)

  return descriptors

ref_img_path = 'reference_image/bernieSanders.jpg'
ref_img = cv2.imread(ref_img_path)

benchmark_dir = 'COMP37212'
benchmark_image_files = [f for f in os.listdir(benchmark_dir) if os.path.isfile(os.p

threshold_values = np.linspace(0.1, 0.3, 50)

num_images = len(benchmark_image_files) + 1
num_cols = 2
num_rows = (num_images + 1) // num_cols
```

```
plt.figure(figsize=(20, 5 * num_rows))

# Compute keypoint detection for the reference image
ref_num_keypoints_custom = []

for threshold in threshold_values:
    ref_keypoints_custom = HarrisPointsDetector(ref_img, sigma=0.5, thresh=threshold
    ref_num_keypoints_custom.append(len(ref_keypoints_custom))

plt.subplot(num_rows, num_cols, 1)
plt.plot(threshold_values, ref_num_keypoints_custom, marker='o', label='Custom Harri
plt.xlabel('Threshold Value')
plt.ylabel('Number of Keypoints Detected')
plt.title('Reference Image')
plt.legend()
plt.grid(True)

# Compute keypoint detection for each benchmark image
for num_image, benchmark_image_file in enumerate(benchmark_image_files):
    benchmark_image_path = os.path.join(benchmark_dir, benchmark_image_file)
    benchmark_img = cv2.imread(benchmark_image_path)

    benchmark_num_keypoints_custom = []

    for threshold in threshold_values:
        benchmark_keypoints_custom = HarrisPointsDetector(benchmark_img, sigma=0.5,
        benchmark_num_keypoints_custom.append(len(benchmark_keypoints_custom))

    plt.subplot(num_rows, num_cols, num_image + 2)
    plt.plot(threshold_values, benchmark_num_keypoints_custom, marker='o', label='Cu
    plt.xlabel('Threshold Value')
    plt.ylabel('Number of Keypoints Detected')
    plt.title(f'Benchmark Image {num_image+1}')
    plt.legend()
    plt.grid(True)

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

```
ref_img_path = 'COMP37212/bernieSanders.jpg'
ref_img = cv2.imread(ref_img_path)

ref_keypoints = HarrisPointsDetector(ref_img, thresh=0.045)
ref_img_with_keypoints = cv2.drawKeypoints(ref_img, ref_keypoints, None, color=(255,
ref_img_with_keypoints_rgb = cv2.cvtColor(ref_img_with_keypoints, cv2.COLOR_BGR2RGB)

# Create an instance of ORB detector with FAST keypoints
orb_fast = cv2.ORB_create()

# Create another instance of ORB detector with Harris corners
orb_harris = cv2.ORB_create(scoreType=cv2.ORB_HARRIS_SCORE)

kp_fast = orb_fast.detect(ref_img, None)
kp_harris = orb_harris.detect(ref_img, None)

_, descriptors_FAST = orb_fast.compute(ref_img, kp_fast)
_, descriptors_Harris = orb_harris.compute(ref_img, kp_harris)

image_orb_harris = cv2.drawKeypoints(ref_img, kp_harris, None, color=(255, 255, 0))
image_orb_fast = cv2.drawKeypoints(ref_img, kp_fast, None, color=(255, 255, 0))

# Convert to RGB for Matplotlib
image_orb_harris_rgb = cv2.cvtColor(image_orb_harris, cv2.COLOR_BGR2RGB)
image_orb_fast_rgb = cv2.cvtColor(image_orb_fast, cv2.COLOR_BGR2RGB)

# Display the images
plt.figure(figsize=(24, 12))

plt.subplot(1, 3, 1)
plt.imshow(ref_img_with_keypoints_rgb)
plt.title('Custom Harris')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(image_orb_harris_rgb)
plt.title('ORB Harris')
plt.axis('off')

plt.subplot(1, 3, 3)
```

```
plt.imshow(image_orb_fast_rgb)
plt.title('ORB FAST')
plt.axis('off')

plt.tight_layout()
plt.show()

from scipy.spatial.distance import cdist

def calculate_ssd(des1, des2):
  ssd_matrix = cdist(des1, des2, 'sqeuclidean')  # 'sqeuclidean' computes the squar
  return ssd_matrix

def ratio_test(ssd_matrix, threshold):
  matches = []
  for i, ssds in enumerate(ssd_matrix):
    sorted_indices = np.argsort(ssds)
    closest, second_closest = sorted_indices[:2]

    if ssds[closest] <= 100.0 and ssds[closest] / ssds[second_closest] < threshold:
      # Append a cv2.DMatch object to the list of matches
      matches.append(cv2.DMatch(_queryIdx=i, _trainIdx=closest, _distance=ssds[close

  return matches

orb = cv2.ORB_create()

benchmark_image_path = 'COMP37212/berniePixelated2.png'
benchmark_img = cv2.imread(benchmark_image_path)

ref_img = cv2.imread('reference_image/bernieSanders.jpg')

kp_ref = HarrisPointsDetector(ref_img, thresh=0.08)
des_ref = featureDescriptor(ref_img, kp_ref)

kp_benchmark = HarrisPointsDetector(benchmark_img, thresh=0.08)
des_benchmark = featureDescriptor(benchmark_img, kp_benchmark)

ssd_matrix = calculate_ssd(des_ref, des_benchmark)
matches = ratio_test(ssd_matrix, threshold=0.75)
```

```
matched_img = cv2.drawMatches(ref_img, kp_ref, benchmark_img, kp_benchmark, matches,
matched_img_rgb = cv2.cvtColor(matched_img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15, 10))  # You can adjust the figure size as needed
plt.imshow(matched_img_rgb)
plt.axis('off')  # Hide the axis
plt.show()

print(f"Image: {benchmark_image_path}")
print(f"Number of matches: {len(matches)}")
```