Task 3

Computer Vision

Team: 20

(Haris,SIFT,Feature Matching)

Name	Section	Bench Number
Abdullah Mohamed	1	39
Mohamed Gamal	2	12
Mohamed Ali	2	22

Harris:

the main function is the `MyHarris` is a Python implementation of the Harris Corner Detection algorithm.

- ## Implementation:

  - The function takes an input image and a factor parameter as arguments.
  - It first converts the input image to grayscale if it's in RGB format.
  - It computes the derivatives of the image using Sobel filters to calculate Ixx,Iyy and Ixy .
  - Using these components, it computes the corner response function H
  - It then applies a threshold to identify significant corners based on the computed corner response values.
  - Finally, it marks the detected corners on the input image by drawing red circles at their coordinates using OpenCV.

- ## Input and Output:

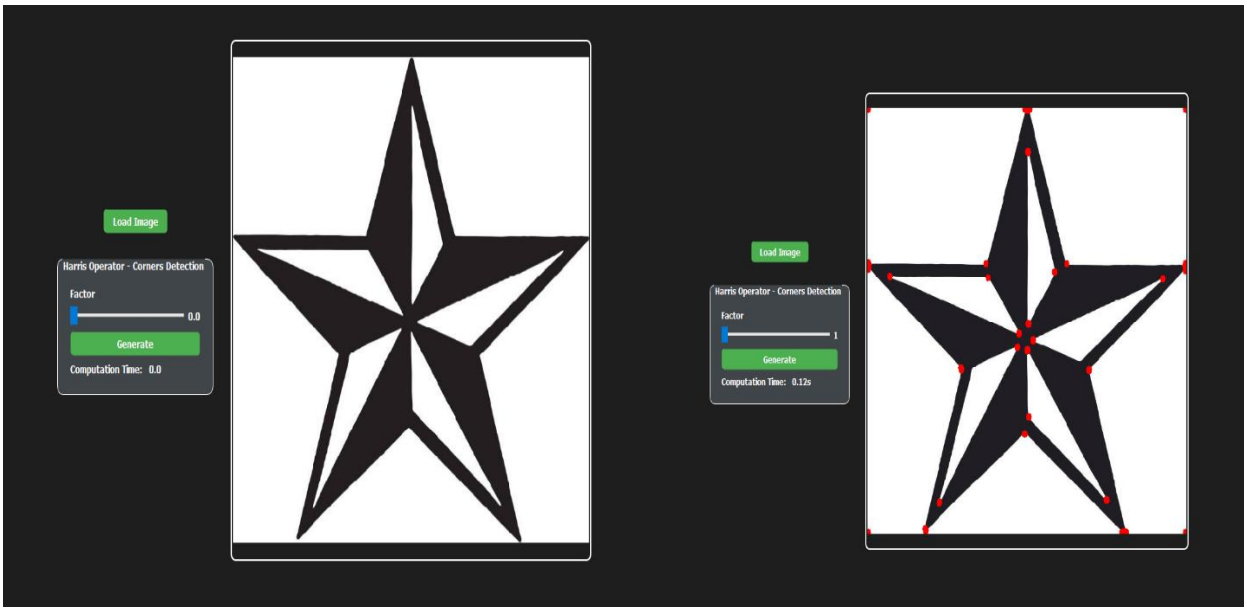
  - Input: The input to the function is an image (in RGB format) and a factor parameter that controls the sensitivity of corner detection.
  - Output: The function returns an image with detected corners marked as red circles, along with the computation time required for corner detection

- **Performance:**

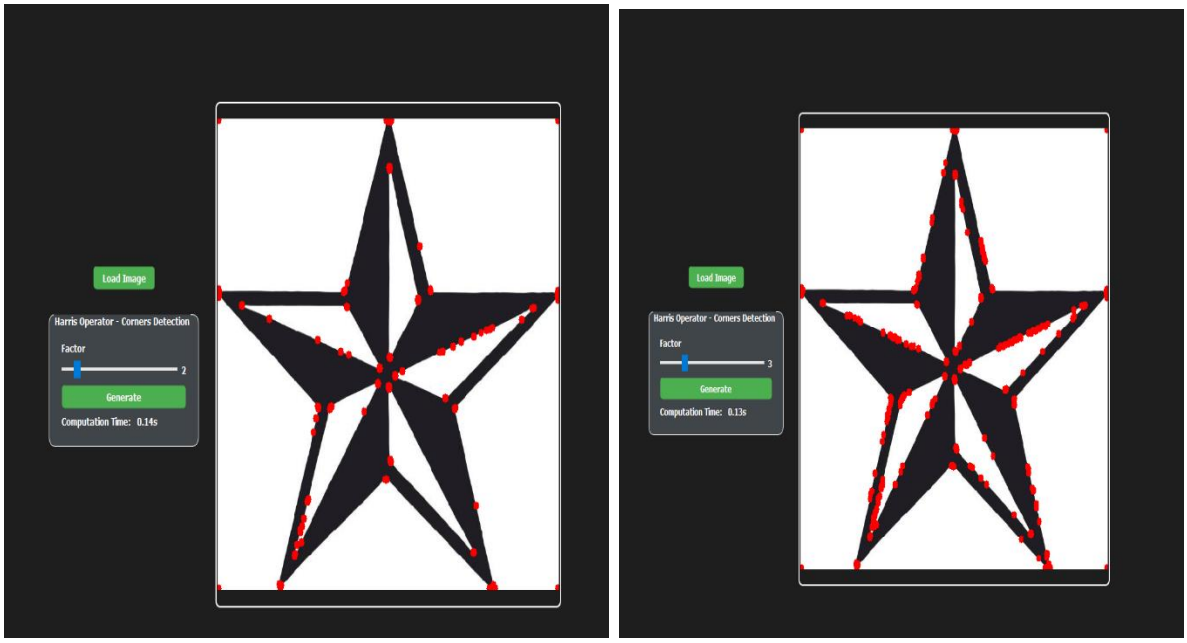
❖ The performance of the algorithm is measured in terms of computation time, which is the time taken to detect corners in the input image.

❖ The computation time may vary depending on the size of the input image and the complexity of the corner patterns

❖ Choosing factor from user play important rule in accuracy which control the threshold of corner here is some example for different threshold.



❖ The left image no corner detect as factor is =0 , and the right one contain small point marker in original image due to factor 1.



- The left image no corner detect as factor is =2 , and the right one contain small point marker in original image due to factor 3.
- From previous images we notice that when factor increase the detected coronal increase until particular limit.

Feature Matching:

Template matching is a technique in computer vision used to find a template image within a larger image. This report analyzes a Python implementation of a template matching algorithm using Scale-Invariant Feature Transform (SIFT) descriptors and different similarity measures.

- **Implementation:**
  - **I. calculateSimilarity:**
    - This function calculates the similarity between two descriptors using either Normalized Cross-Correlation (NCC) or Sum of Squared Differences (SSD).
    - **Depending on the selected similarity measure, it computes the similarity score accordingly.**

II. findTemplate:

- This function finds the template image within the original image using SIFT descriptors and the specified similarity measure.
- It utilizes OpenCV's SIFT detector to extract keypoints and descriptors from both images.
- Then, it matches descriptors from the template to those from the original image, computing the similarity scores using the calculateSimilarity function.
- Matches are filtered based on a user-defined threshold, and only good matches are retained.
- Finally, it draws the good matches on the original image and returns the result along with the computation time.

III. Output:

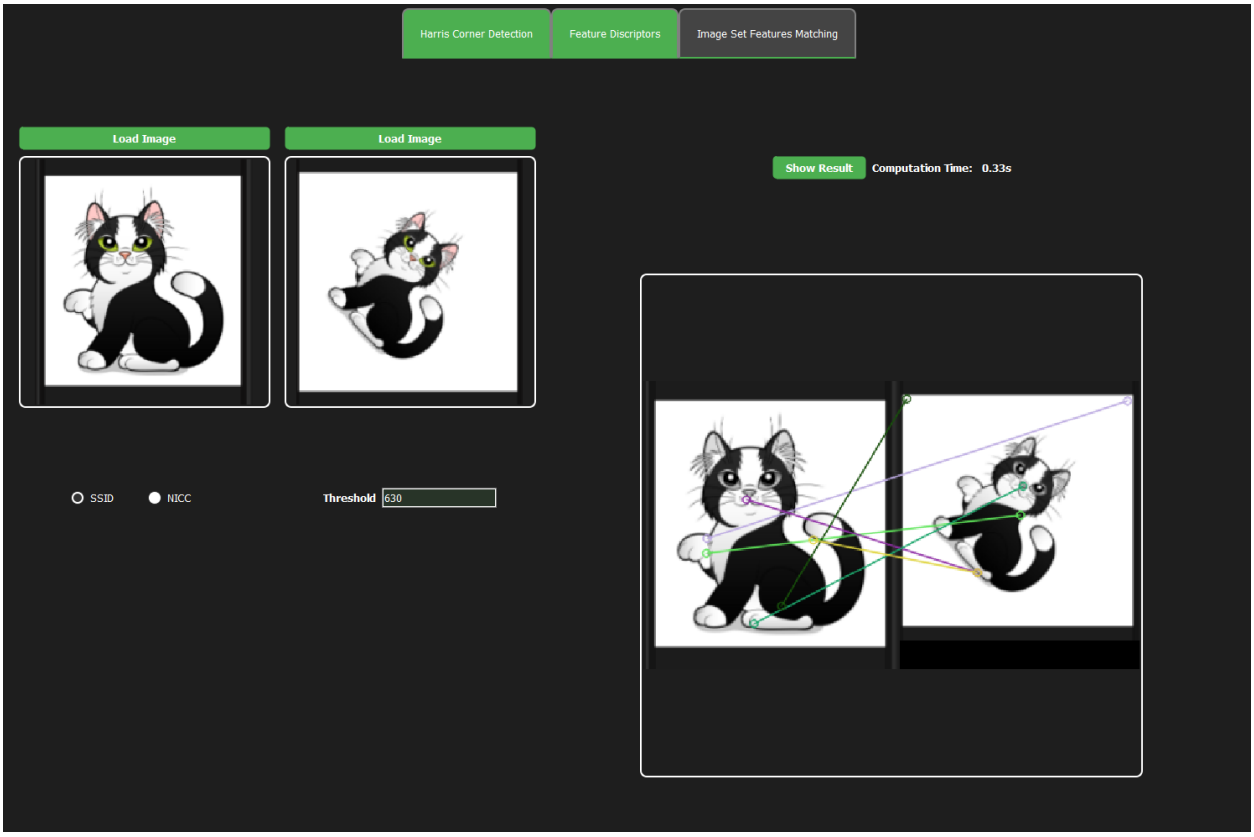
- The method returns an image with visualizations of the matched keypoints and descriptors, along with the computation time required for template matching

• Performance:

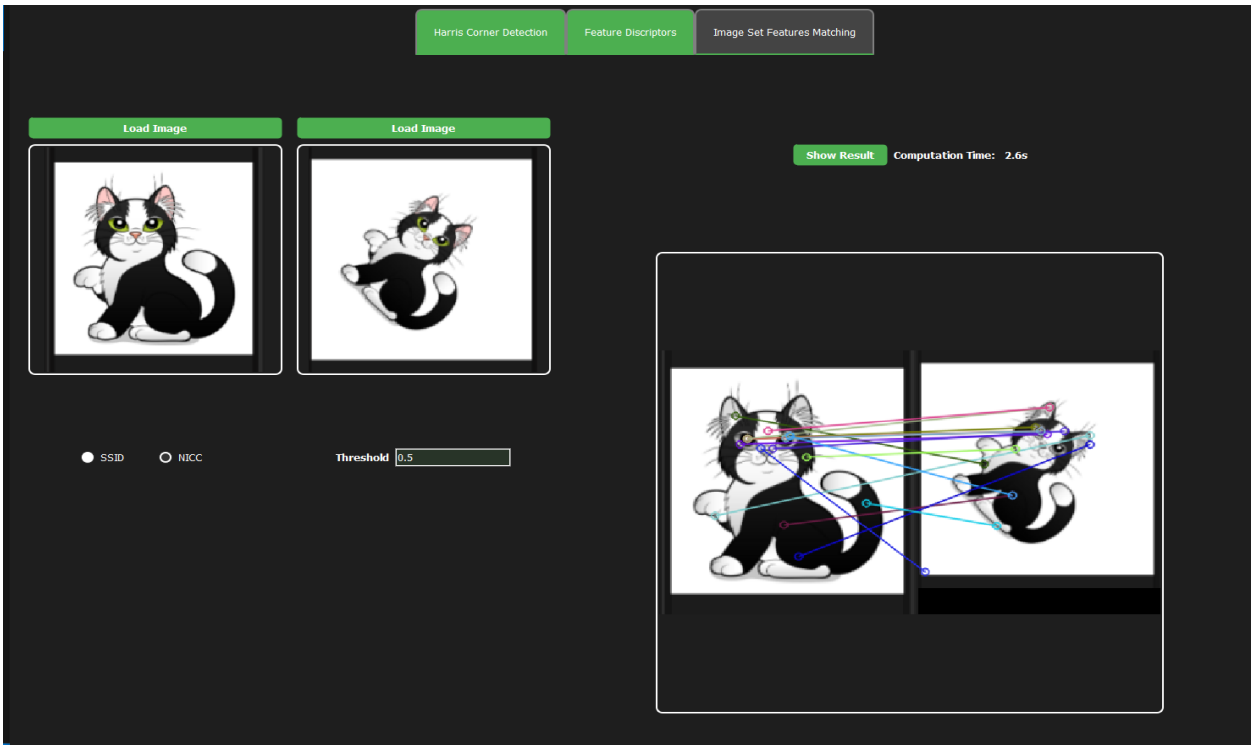
- **The performance of the algorithm is evaluated in terms of computation time, which is the time taken to find the best match and perform SIFT descriptor matching.**
- **The computation time may vary depending on the size of the original image, the complexity of the template, and the number of keypoints and descriptors detected,whtere user choose SSD for calculation of similarty or NCC which affect in time and also choosing the threshold can control the number of connected line between the tow images .**

Here is some examples:

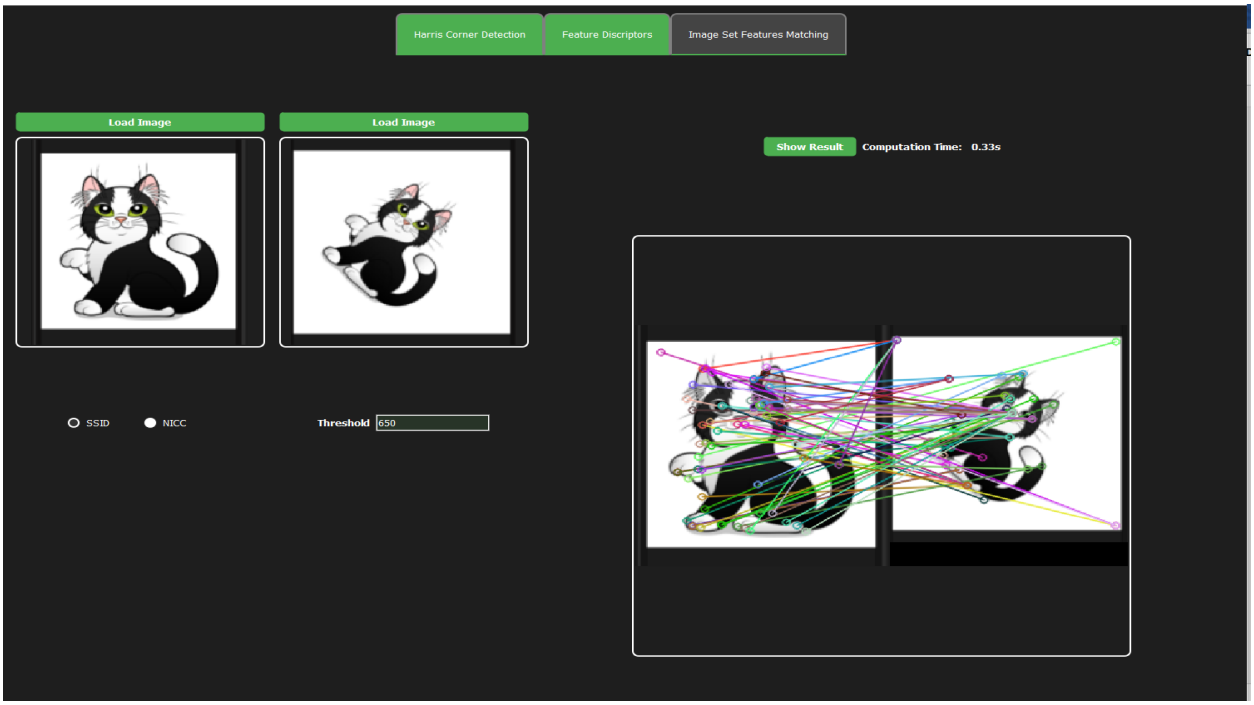
Using SSD with threshold 630



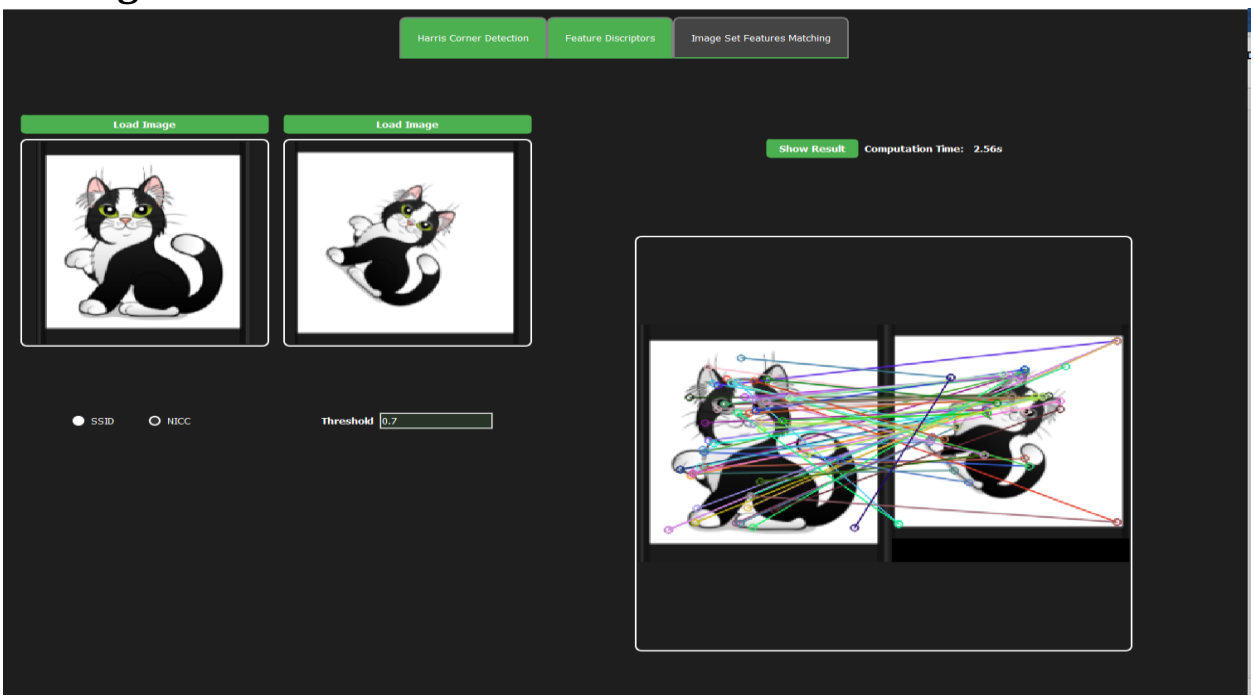
Using NCC with threshold 0.5



Using SSD with threshold 650



Using NCC with threshold 0.7



From previous image we notice that SSD take short time from NCC due to large computation from the rule of NCC

And also we notice when we increase threshold the number of line increase and vis versa .

Distance Metric

- Sum of Squared Difference (SSD)

$$\sum_i (I_1(\mathbf{x}_i) - I_0(\mathbf{x}_i))^2$$

- Normalized Cross-Correlation (NCC)

$$\frac{1}{n-1}\sum_i \frac{(f(x_i)-\bar{f})(g(x_i)-\bar{g})}{\sigma_f \sigma_g}$$

.

.

SIFT Keypoint Detection Algorithm:

Introduction: The Scale-Invariant Feature Transform (SIFT) algorithm is widely used in computer vision for tasks like object recognition, image stitching, and 3D reconstruction. It detects and describes local features in images, which are invariant to scale, rotation, and illumination changes.

The steps of the algorithm:

1. **Gaussian Kernel Generation (gaussian_kernel):**
    - **The function generates a Gaussian kernel used for image blurring. It's parameterized by size, scale factor (k), and standard deviation (sigma).**
2. **2D Convolution (convolve):**
    - **This function performs 2D convolution between an image and a kernel with support for zero-padding and stride.**
3. **Keypoint Detection (detect_keypoints):**
    - **Utilizes the Difference of Gaussians (DoG) approach to identify keypoints across different scales and octaves in the image.**
    - **Thresholding based on local extrema in the DoG space helps in selecting robust keypoints.**
4. **Gradient Magnitude and Direction Computation (compute_magnitude_direction):**

- **Calculates gradient magnitude and direction using image gradients.**

5. **Orientation Assignment (assign_orientation):**
    - **Assigns orientations to keypoints based on gradient directions in their local neighborhoods.**
    - **Quantizes gradient directions and selects the dominant orientation for each keypoint.**
6. **Descriptor Extraction (extract_descriptors):**
    - **Computes descriptors for keypoints using gradient histograms in their local regions.**
    - **Descriptors are normalized and quantized to improve robustness against variations.**
7. **Image Matching (generate_image):**
    - **Matches keypoints between two images and visualizes the matches.**

Discussion:

- **Algorithm Implementation: the code effectively implements key components of the SIFT algorithm, making it suitable for feature-based image matching tasks.**
- **Efficiency: there might be opportunities for optimization. For example, optimizing nested loops and using numpy's vectorized operations can improve performance, especially for large images.**
- **Parameter Sensitivity: The algorithm's performance can be sensitive to parameters like the number of octaves, scales, sigma, and contrast threshold. Fine-tuning these parameters**

based on the specific application or image characteristics is crucial.

Discussion of parameters:

1- number of octaves.
2- number of scales to generate gaussian pyramid.
3- sigma, k for generating gaussian kernel.
4- contrast threshold for removing some keypoints.

Here are examples for specific cases that we have considered.

1- The first case, we selected five images per octave, one scale, sigma are kept constant 1.6 for all cases according to the paper , k = sqrt(2), and threshold of 4 to filter some keypoints.