Third Year – 2nd Term

Systems & Biomedical Eng. Department

Cairo University

COMPUTER VISION
TASK 3: CORNER DETECTION AND TEMPLATE MATCHING USING THE SIFT ALGORITHM REPORT

Team 5

Salma Ashraf

Sarah Mohamed

Habiba Salama

Hager Samir

Aya Eyad

April 18, 2024

Harris and λ- Operator

1. apply_harris_operator:

Parameters:

- img (np.ndarray): The input image to which the Harris operator will be applied.
- k (float): Sensitivity factor to separate corners from edges. Smaller values of k result in the detection of sharper corners.

Main Point:

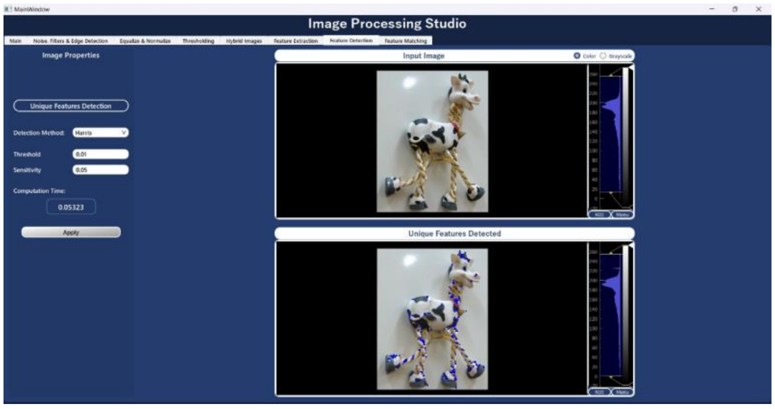
This function applies the Harris corner detection algorithm to the input image. It computes the Harris response for each pixel in the image based on the gradient of the image.

Observations:

- ➢ The function first converts the input image to grayscale.
- ➢ It then computes the gradients of the image using the Sobel operator.
- ➢ Gaussian blur is applied to the gradients to reduce noise.
- ➢ Finally, the Harris response is calculated using the determinant and trace of the gradient matrix.

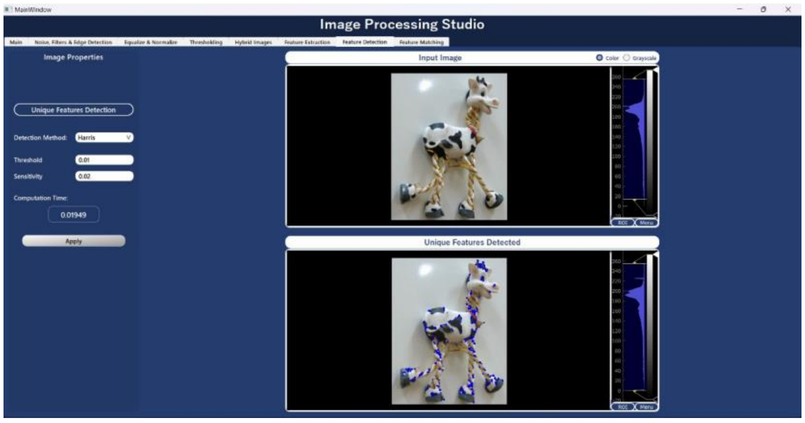
Experiments:

➢ **Apply Harris operator with TH= 0.01 & and K= 0.05 :**



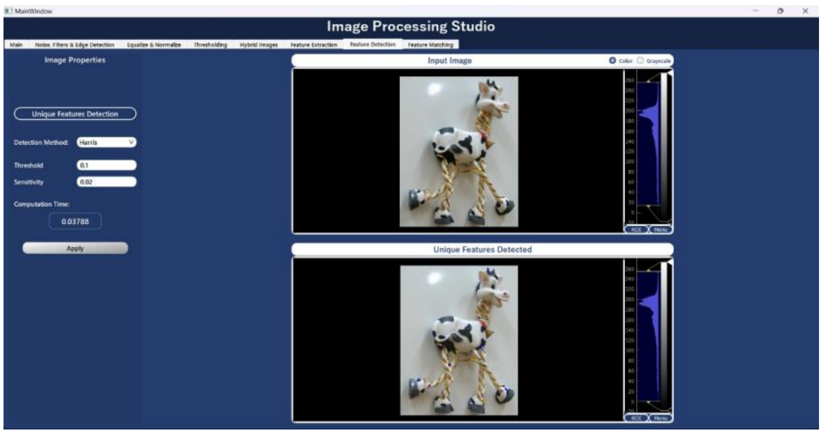
Detection Method: Harris
Threshold: 0.01
Sensitivity: 0.05
Computation Time: 0.05323

➢ **Apply Harris operator with TH= 0.01 & and K= 0.02 (Small value of K detect sharp corners) :**



Detection Method: Harris
Threshold: 0.01
Sensitivity: 0.02
Computation Time: 0.01949

- **Apply Harris operator with TH= 0.1 & and K= 0.02 (large value of TH detect less corners) :**



Detection Method: Harris
Threshold 0.1
Sensitivity 0.02
Computation Time: 0.03788

2. apply_lamda_minus_operator:

Parameters:

- img (np.ndarray): The input image to which the λ- operator will be applied.

Main Point:

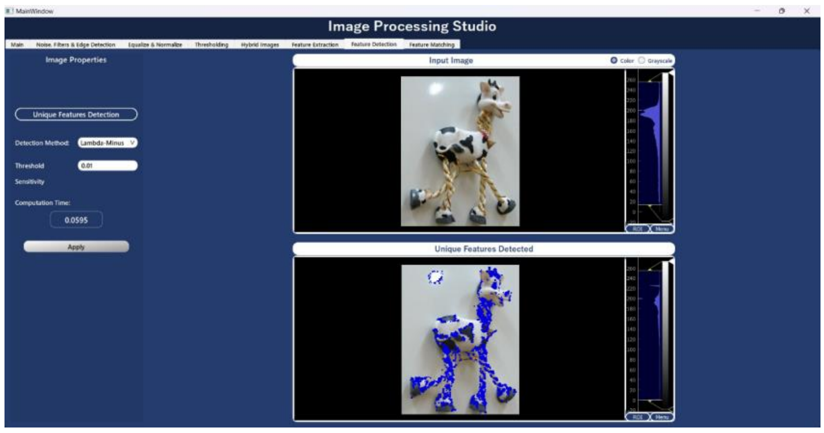
This function applies the λ- corner detection operator to the input image. It computes the λ-response for each pixel in the image based on the gradients of the image.

Observations:

- Similar to the Harris operator, it starts by converting the input image to grayscale and computing gradients.
- Instead of directly calculating the Harris response, it computes the sum of squared gradients within a window around each pixel to find the corner response.
- Eigenvalues of the gradient matrix are calculated to obtain the λ- response.

Experiment:

- **Apply Lamda Minus operator with TH= 0.01 (take more time than Harris) :**



Detection Method: Lambda-Minus
Threshold 0.01
Sensitivity
Computation Time: 0.0595

3. get_operator_indices:

Parameters:

- operator_response (np.ndarray): An array representing the Operator response of an image.
- threshold (float, optional): Threshold value used for computing local maxima.

Main Point:

This function identifies corners, edges, and flat areas in the image based on the Operator response.

Observations:

- ➢ It dilates the Operator response to make the points clearer.
- ➢ Corner indices are identified where the Operator response is greater than a certain threshold.
- ➢ Edge indices are identified where the Operator response is negative.
- ➢ Flat indices are identified where the Operator response is zero.

4. map_indices_to_image:

Parameters:

- img (np.ndarray): The input image on which dots will be drawn.
- indices (np.ndarray): An array of 0's and 1's representing indices of interest.
- color (list): Color to draw the dots with, specified as a list

Main Point:

This function draws dots on the input image based on provided indices.

Observations:

- ➢ It creates a copy of the input image to avoid modifying the original.
- ➢ Dots are drawn on the image where the indices are equal to 1, using the specified color.
- ➢ The function returns a new image with dots drawn based on the provided indices.

SIFT

- **The SIFT algorithm can be decomposed into four steps:**
    1. Scale space formation
    2. Keypoint detection.
    3. Orientation assignment.
    4. Feature descriptor generation.

Step 1: Scale space formation Time taken for this step: 0.4591035842895508 seconds

1. construct_scale_space:

Parameters:

- image (np.ndarray): The input image to construct the different scales and octaves.
- sigma_values: The standard deviation of gaussian kernels forming the scales and it equals (k^i)*sigma, where sigma = 1.6 and k = sqrt(2). These numbers were recommended by the paper.

Returns:

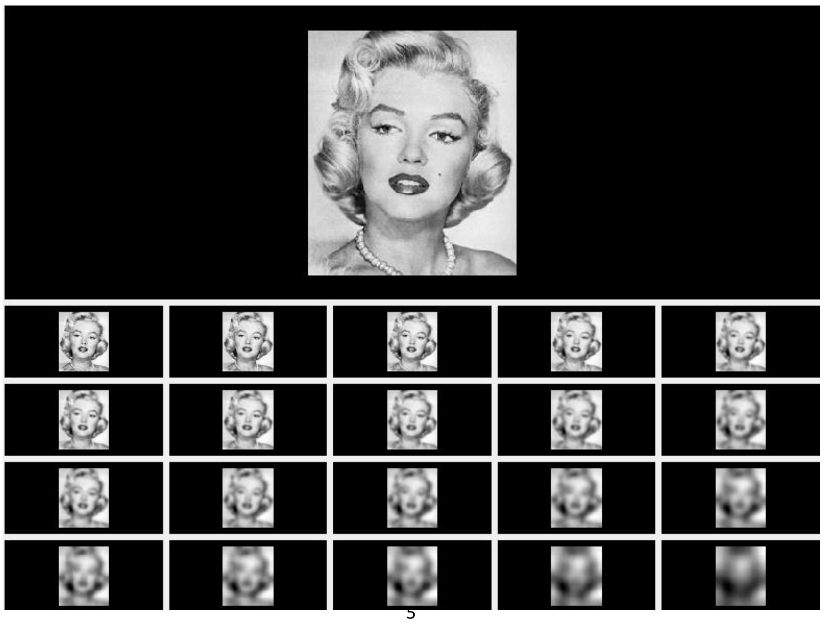
- diff_of_gaussians (2d list): A list of lists carrying the DOGs of all octaves (4x4).
- octaves (list): The list of 4 octaves of different resolutions.

Steps:

1. Apply gaussian kernel with the list of sigma values recommended by the paper to get the octave's starting image.
2. Subtract every two successive scales to get the DOGs.
3. For the upcoming octave, decrease the size (resolution) by half.

Experiment:

- **Constructing 4 octaves, each of 5 scales:**



Step 2: Key point detection

1. compare_with_neighbors:

Parameters:

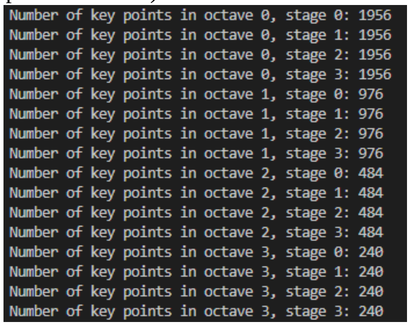
- stage_list (np.ndarray): A list of stages for one octave.

Returns:

- stage_list (np.ndarray): The list after modification (extraction of key points)

Steps:

1. For every point in the image not equal to zero (key point candidate):
    1. Compare it to the points in the 3x3 kernel above and below it and in the same stage
    2. Unless it is their maximum or minimum value, we zero it and it is discarded from being a key point
2. points in the upper and lowermost octaves are compared to points in the same stage and the stage above/below it. Then points are thresholded at 0.03 (recommended by the paper) to eliminate points of slight differences between scales.



```
Number of key points in octave 0, stage 0: 1956
Number of key points in octave 0, stage 1: 1956
Number of key points in octave 0, stage 2: 1956
Number of key points in octave 0, stage 3: 1956
Number of key points in octave 1, stage 0: 976
Number of key points in octave 1, stage 1: 976
Number of key points in octave 1, stage 2: 976
Number of key points in octave 1, stage 3: 976
Number of key points in octave 2, stage 0: 484
Number of key points in octave 2, stage 1: 484
Number of key points in octave 2, stage 2: 484
Number of key points in octave 2, stage 3: 484
Number of key points in octave 3, stage 0: 240
Number of key points in octave 3, stage 1: 240
Number of key points in octave 3, stage 2: 240
Number of key points in octave 3, stage 3: 240
```

Experiment:

➢ The number of key points produced for every octave and stage

Step 3: Descriptors Calculation

1. orientation_assignment:

Parameters:

- keypoints_list (list): A list of tuples containing information about keypoints, each tuple consists of (octave_index, sigma_value, keypoint_coordinates).
- list_octaves (list): List of Gaussian octaves generated from the input image.

Description:

Assigns orientation to key points based on gradient information.

Steps:

1. Iterate over each octave index and each keypoint in the keypoints_list.
2. Retrieve the octave and scale information for the keypoint.
3. Calculate the Gaussian kernel parameters based on the scale.
4. Compute the gradient magnitude and orientation for the keypoint's scale.
5. Iterate over each keypoint coordinate.
6. Compute the orientation bin index for the gradient orientation.
7. Define the window around the keypoint for gradient calculation.
8. Extract the gradient magnitude and orientation information within the window.
9. Weight the gradient magnitude with the Gaussian kernel.
10. Compute a histogram of weighted magnitudes based on orientation bins.
11. Determine the dominant orientation as the bin with the maximum magnitude.
12. Append the keypoint and its dominant orientation to the dominant_dir_list.

2. compute_gradient_magnitude_orientation:

Parameters:

- image (numpy.ndarray): The input image for which gradient magnitude and orientation are computed.

Main Points:

- Computes the gradient magnitude and orientation of an image using Sobel operators.
- Utilizes convolution operations to calculate the gradients along the x and y axes.
- Computes magnitude as the square root of the sum of squared gradients and orientation as the arctangent of the y-gradient over the x-gradient.

Observation:

This function efficiently computes gradient information, enabling further analysis such as edge detection or feature extraction in image processing tasks.

Commented [AE1]: h

Commented [AE2R1]:

3. extract_region:

Parameters:

- image (numpy.ndarray): The input image from which a region is extracted.
- window (list): A list representing the window coordinates [start_x, end_x, start_y, end_y] from which the region is extracted.

Main Points:

- Computes the dimensions of the output region based on the specified window coordinates and the shape of the input image.
- Determines the source (src) and destination (dst) regions within the input and output arrays, respectively.
- Copies the specified region from the input image to the output array.
- Returns the extracted region as a numpy array.

Observation:

This function efficiently extracts a region of interest from an image, facilitating localized operations or analysis on specific parts of the image.

Step 4: Feature descriptor generation

1. compute_ descriptors:

Parameters:

- keypoints (list): A list containing tuples of keypoints, each tuple containing the octave index, sigma value, and a list of (x, y) coordinates.
- list_octaves (list): A list containing octave information used in computing descriptors.

Main Points:

- Iterates through keypoints and computes descriptors based on gradient magnitude and orientation.
- Extracts a region of 16x16 around the keypoint and computes gradient information using Sobel operators and smoothing gradient magnitude using a Gaussian kernels.
- Subtracts the gradient direction of the extracted region from the dominant orientation of the keypoint.

- Constructs histograms of oriented gradients (HOG) for each of the 16 sub-regions of size 4x4 around the keypoint.
- Normalizes and clips feature vectors to ensure numerical stability.
- Returns list of descriptors for all keypoints.

Observation:

This function efficiently computes descriptors using HOG features, which are valuable for tasks like object recognition and image matching in computer vision applications

Feature Matching

1. calculate_SSD: **Time taken to match by SSD:** 20.61273717880249 **seconds**

Parameters:

- descriptor1: The first of the 2 descriptors to calculate their error using SSD.
- descriptor2: The second of the 2 descriptors to calculate their error using SSD.

Returns:

- ssd_value (float): The error of the 2 descriptors.

$$SSD(D_1, D_2)\sqrt{\Sigma_k \left(B_1(k) - B_2(k)\right)^2}$$

$$where, D_1, D_2 are\ the\ n_{th}\ descriptors\ of\ the\ 1st, 2nd\ images\ keypoints$$

Observation:

- SSD values range from 0 to positive infinity.
- Lower SSD values indicate better matches or higher similarity between the images.
- An SSD value of 0 indicates perfect similarity, meaning that the two images are identical.

2. Calculate_NCC: **Time taken to match by NCC:** 26.718332529067993 seconds

Parameters:

- descriptor1: The first of the 2 descriptors to calculate their error using normalized cross correlation.
- descriptor2: The second of the 2 descriptors to calculate their error using normalized cross correlation.

Returns:

- ncc_value (float): The error of the 2 descriptors ranges.

$$NCC = \frac{1}{len(D_1(k))} \sum_k \frac{[(D_1(k) - \overline{D_1})(D_2(k) - \overline{D_2})]}{\sigma_1^2\ \sigma_2^2}$$

Observation:

- SSD is faster and has less complexity than the NCC, although NCC is more robust to intensity variations and we can see in the experiments that it produced a better result.
- NCC values range from -1 to 1.
- NCC = 1 indicates perfect positive correlation or similarity.
- NCC = -1 indicates perfect negative correlation (i.e., completely dissimilar images).
- NCC values close to 1 indicate high similarity, while values close to -1 indicate high dissimilarity.
- NCC = 0 means no correlation or similarity between the images.

3. Find_best_match: finds the best match for a given SIFT descriptor among a set of descriptors

Parameters:

- descriptor (list): A single SIFT descriptor.
- descriptors (list): List of SIFT descriptors to search for matches.

Returns:

- int or None: The index of the best matching descriptor in descriptors, or None if no match is found.

4. Match_descriptors:

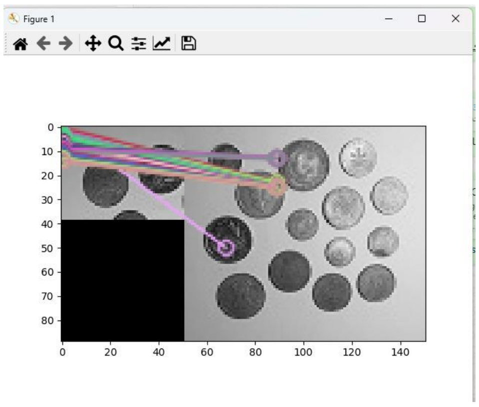
Parameters:

- descriptors1 (list): List of SIFT descriptors from the first image.
- descriptors2 (list): List of SIFT descriptors from the second image.

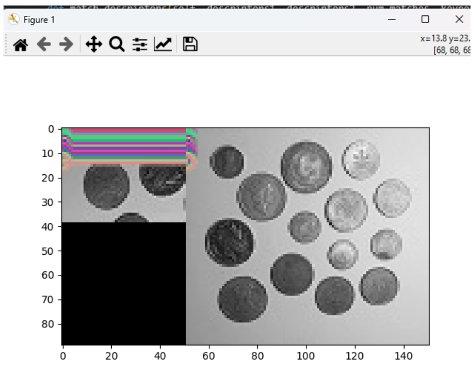
Returns:

- matches (list): A list of tuples containing indices of matched descriptors. Each tuple consists of the index of the descriptor in descriptors1 and the index of its best match in descriptors2.

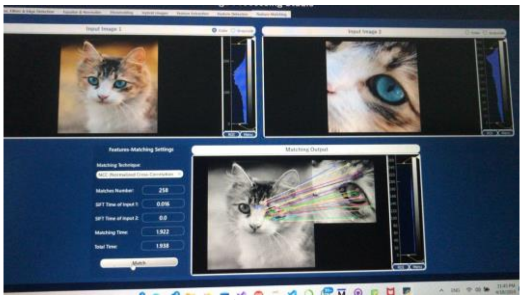
Experiment:



Using NCC



Using SSD



Matching Technique: NCC (Normalized Cross Correlation)
Matches Number: 258
SIFT Time of Input 1: 0.016
SIFT Time of Input 2: 0.0
Matching Time: 1.922
Total Time: 1.938

Using built-in sift and our matcher