



Cairo University

Computer Vision (SBE3230)

Task 3

Name	Sec	BN
Dina Hussam Assem	1	28
Omar Ahmed Anwar	2	2
Omar Saad Mohamed	2	3
Mohamed Ahmed Ismail	2	16
Neveen Mohamed Ayman	2	49

Supervised By

Dr. Ahmed Badwy

TA: eng. Peter Emad & eng. Lila Abbas

Table of Contents

Introduction.....	3
Algorithms Implemented and How They Work.....	3
Harris Operator	3
SIFT	5
Matching Images	5
Compare To Library	8
Harris Operator	8
SIFT and Matching	8
Application Screenshots	9
Harris Operator Tab	9
SIFT & Matching Images Tab	9

Introduction:

It's an image processing implementation functions project implemented in C++ with a desktop application (Qt).

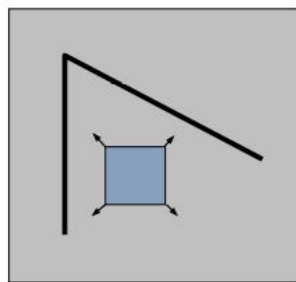
Algorithms Implemented And How They Work:

1) Harris Operator:

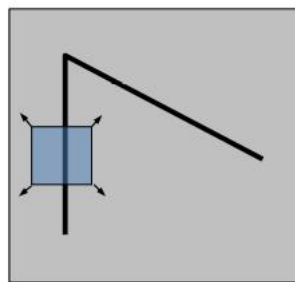
The Harris operator, also known as the Harris corner detector or Harris-Stephens corner detector, is a commonly used algorithm for detecting and extracting features from images. It was developed by Chris Harris and Mike Stephens in 1988.

The Harris operator is used to identify interest points or corners in an image by looking for areas where the intensity of the image changes significantly in all directions. The operator calculates the sum of squared differences between the image intensity values in a small window and their average, for a given displacement in all directions. This is done using the Harris matrix, which is a 2×2 matrix that represents the local gradient of the image.

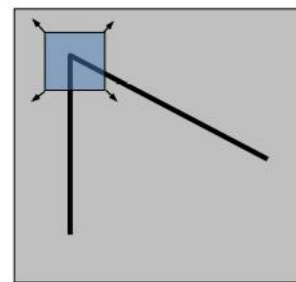
The Harris operator then uses the eigenvalues of the Harris matrix to determine whether a pixel corresponds to a corner or not. If the eigenvalues are both large, then the pixel is considered a corner. If one eigenvalue is much larger than the other, then the pixel corresponds to an edge. If both eigenvalues are small, then the pixel corresponds to a flat region.



"flat" region:
no change in all
directions



"edge":
no change along the
edge direction

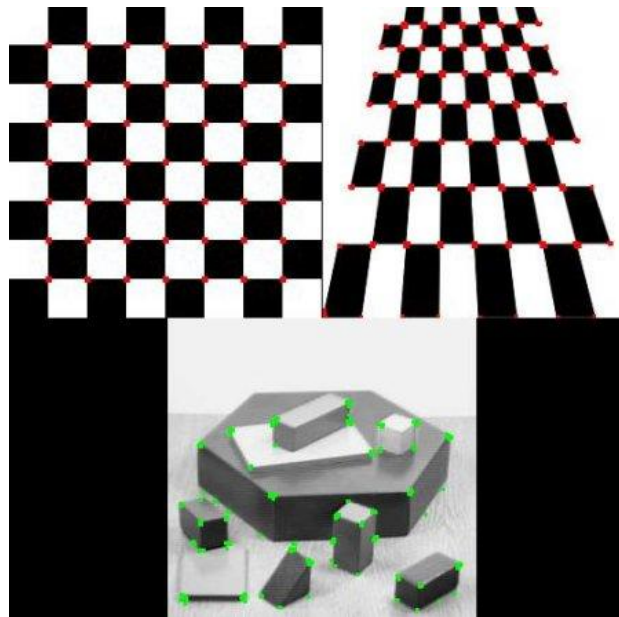


"corner":
significant change in
all directions

The Harris operator is widely used in computer vision applications, such as object recognition, image alignment, and 3D reconstruction. It is also used in robotics, where it can be used to identify landmarks for navigation purposes.

Algorithm:

1. We define our main function and load an input image "input_image.jpg" using OpenCV's imread function. The image is stored as a Mat object named "img".
2. We convert the input image to grayscale using OpenCV's cvtColor function. The grayscale image is stored as a Mat object named "gray".
3. We compute the image derivatives using the Sobel operator. We first define two Mat objects "dx" and "dy" to store the horizontal and vertical derivatives respectively. We then apply the Sobel operator to the grayscale image "gray" using OpenCV's Sobel function. The first Sobel function call computes the horizontal derivative using the x-direction Sobel kernel (1,0) and stores the result in "dx". The second Sobel function call computes the vertical derivative using the y-direction Sobel kernel (0,1) and stores the result in "dy".
4. We compute the elements of the structure tensor using the image derivatives. We define three Mat objects "lxx", "lxy", and "lyy" to store the elements lxx, lxy, and lyy of the structure tensor respectively. We compute these elements by multiplying the horizontal and vertical derivatives in different combinations using OpenCV's element-wise multiplication function, "mul".
5. We compute the sum of structure tensor elements in a local neighborhood using a box filter. We define three Mat objects "Sxx", "Sxy", and "Syy" to store the sums of the elements lxx, lxy, and lyy in the local neighborhood respectively. We apply a box filter to each of these Mat objects using OpenCV's boxFilter function. The box filter size is specified by the "blockSize" variable, which is set to 3 in this example.
6. We compute the Harris response using the elements of the structure tensor. We define three Mat objects "det", "trace", and "response" to store the determinant, trace, and Harris response respectively.



2) SIFT:

Algorithm:

1. Scale-space extrema detection: In this step, we construct a scale space representation of the input image by repeatedly smoothing the image with Gaussian filters at different scales. We then search for local extrema in the scale space by comparing each pixel to its 26 neighbors at adjacent scales.
2. Keypoint localization: Once we have detected potential keypoints in the scale space, we refine their locations and scales by fitting a 3D quadratic function to the scale space values at each extrema. We discard keypoints with low contrast or poorly localized on the scale space.
3. Orientation assignment: For each keypoint, we compute its dominant orientation by calculating the gradient magnitude and orientation of the pixels within a circular window around the keypoint. We then assign the keypoint to multiple orientations based on the gradient orientation histograms within the window.
4. Keypoint descriptor calculation: We compute a 128-dimensional descriptor for each keypoint by constructing a histogram of gradient orientations in a 16x16 pixel grid centered on the keypoint and normalizing the histogram.
5. Keypoint matching: Finally, we match keypoints between different images by comparing their descriptors using a distance metric such as Euclidean distance or cosine similarity. We can use various techniques to filter out false matches and estimate the geometric transformation between the matched keypoints.

3) Matching Images:

About

Matching is a common problem in computer vision and image processing, where the goal is to find correspondences between two images or between an image and a template. Two widely used techniques for image matching are Sum of Squared Differences (SSD) and Normalized Cross Correlation (NCC).

Sum of Squared Differences (SSD) is a measure of similarity between two images based on the sum of the squared differences between corresponding pixel intensities. To compute SSD, the two images are first aligned, usually by translating one image relative to the other. Then, for each pixel in the aligned images, the difference between the pixel intensities is computed and squared. Finally, the sum of these squared differences is computed over all pixels in the images. The resulting value is a measure of the dissimilarity between the two images, with a smaller value indicating a higher degree of similarity.

Normalized Cross Correlation (NCC) is another measure of similarity between two images that is based on the correlation between their pixel intensities. NCC is similar to SSD in that it also involves aligning the images and comparing the intensities of corresponding pixels. However, instead of computing the sum of squared differences, NCC computes the normalized cross correlation between the two images. NCC is defined as the dot product of the normalized intensity values of the two images, where the normalization ensures that the resulting value is between -1 and 1. A value of 1 indicates perfect similarity, while a value of -1 indicates perfect dissimilarity.

Both SSD and NCC have their advantages and disadvantages. SSD is computationally simple and easy to implement, but it can be sensitive to changes in lighting and contrast. NCC, on the other hand, is more robust to lighting and contrast changes, but it can be more computationally demanding and requires normalization to ensure that the correlation value is meaningful.

In summary, SSD and NCC are two commonly used techniques for image matching in computer vision and image processing. SSD is based on the sum of squared differences between corresponding pixel intensities, while NCC is based on the normalized cross correlation between the intensities of the two images. Both techniques have their advantages and disadvantages, and the choice between them depends on the specific application and requirements of the task at hand.

$$h[m,n] = \frac{\sum_{k,l} (g[k,l] - \bar{g})(f[m-k,n-l] - \bar{f}_{m,n})}{\left(\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m-k,n-l] - \bar{f}_{m,n})^2 \right)^{0.5}}$$

mean template
↓
mean image patch
↓

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2$$

Matching with SIFT

Once the features have been extracted, the next step is to match corresponding features in two images. This is where techniques such as Sum of Squared Differences (SSD) and Normalized Cross Correlation (NCC) can be used.

To use SSD for feature matching, the first step is to compute the distance between the descriptors of each pair of key points. This is done by computing the sum of the squared differences between the corresponding elements of the two descriptors. The key point pairs with the smallest distances are then considered to be correspondences between the two images.

Similarly, to use NCC for feature matching, the first step is to compute the normalized cross correlation between the descriptors of each pair of key points. This is done by computing the dot product of the normalized descriptors. The key point pairs with the highest correlation values are then considered to be correspondences between the two images.

In practice, a combination of both techniques can be used for feature matching. For example, SSD can be used as a fast initial filter to reduce the number of potential matches, followed by NCC to refine the matches and improve their robustness.

Algorithm:

1. The code then defines two functions: **Sum_of_Squared_Differences** and **Normalized_Cross_Correlation**, which compute the matching score between two feature descriptor vectors. The **Sum_of_Squared_Differences** function calculates the sum of the squared differences between each pair of points in the two vectors, while the **Normalized_Cross_Correlation** function computes the normalized cross-correlation between the two vectors.
2. The **match-features** function takes two sets of feature descriptor vectors as input along with a boolean value indicating whether to use SSD or NCC as the matching score. It loops through each feature in the first set and compares it with every feature in the second set, calculating the matching score between the two vectors. If the score is above a certain threshold, the two features are considered a match, and their indices are stored in a vector of pairs.
3. The **matrix_to_vectors** function is used to convert a matrix of feature descriptor vectors into a vector of vectors.
4. Detects features and computes their descriptors using the SIFT algorithm, converts the descriptors to vectors, and matches the features using SSD. The matches are then sorted based on distance and visualized using Open CV.

Matching Computation Time :

The computation time of SSD and NCC depends on various factors such as the size and dimensional of the feature descriptor vectors, the number of features to be matched, and the computational resources available. In general, SSD is faster than NCC, but the difference in computation time depends on the specific implementation and the characteristics of the feature descriptors being used.

SSD

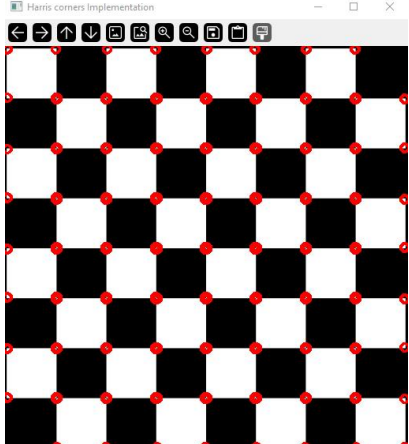
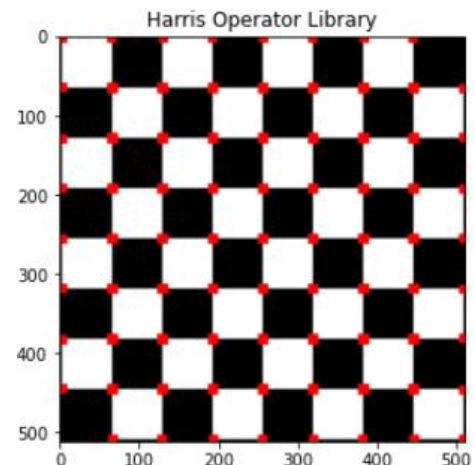
```
Computation time: 5544941 microseconds  
Number of matches : 2303
```

NCC


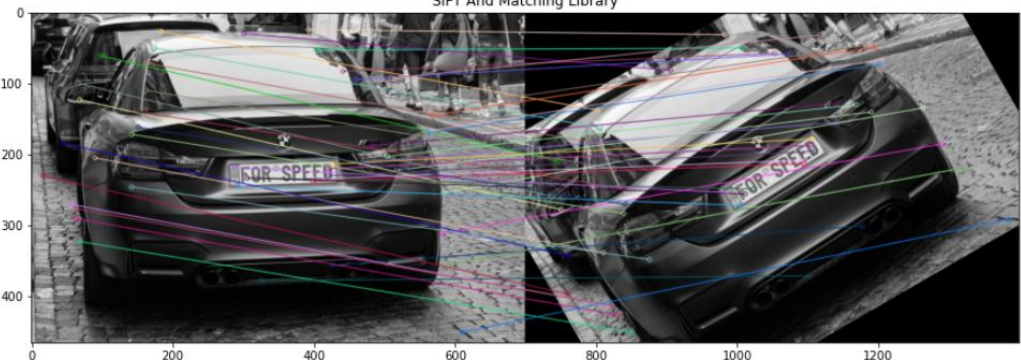
```
Computation time: 12237893 microseconds  
Number of matches : 107
```


Compare To Library:

1) Harris Operator:

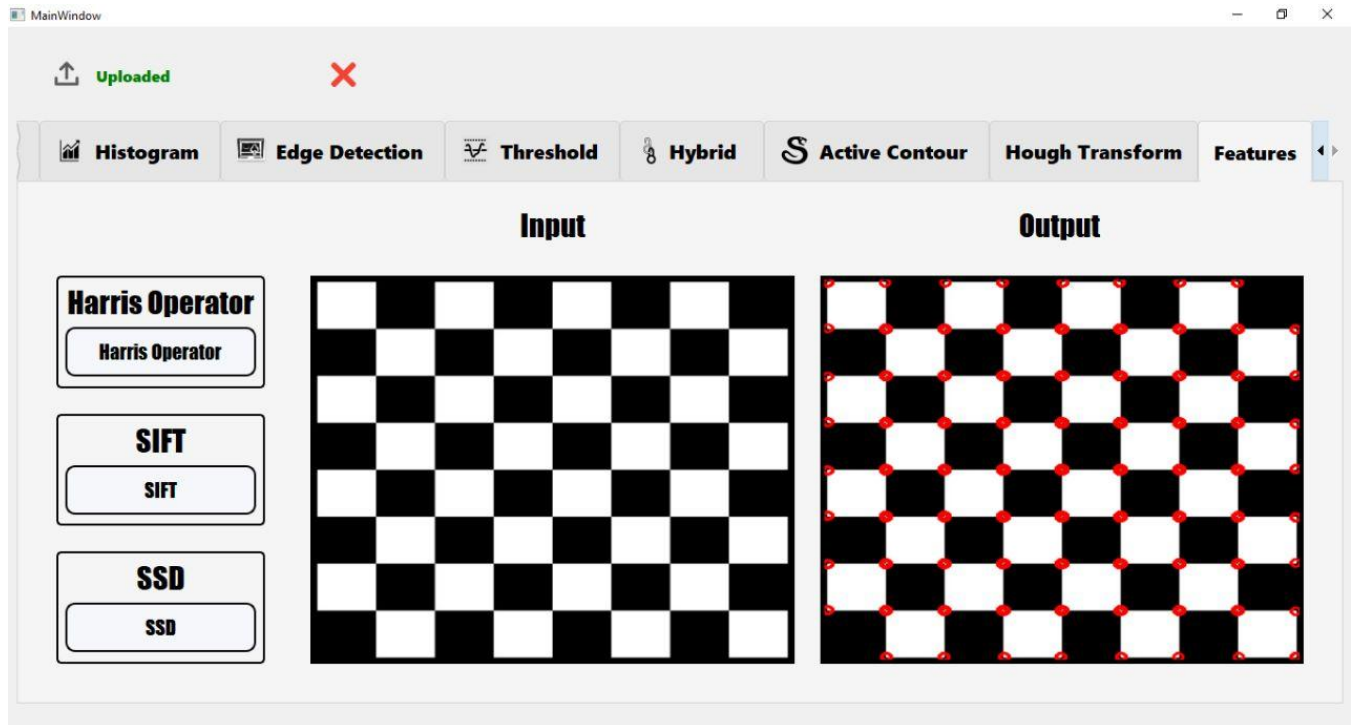
	C++ Implementation	Library
Harris Operator		

2) SIFT And Matching:

	C++ Implementation	Library
SIFT And Matching		

Application Screenshots:

1) Harris Operator Tab:



2) SIFT Tab & Matching Images Tab :

