Cairo University

Faculty of Engineering

Systems & Biomedical Engineering

Computer Vision

(SBE 3230)

Under Supervision:

Professor, Ahmed M, Badawi

Eng. Laila Abbas

Eng. Omar Hesham

Team Members:

1. Fady Mohsen Magdy

2. Lama Zakaria

3. Lamees Mohee

4. Rana Ibrahim

5. Camelia Marwan

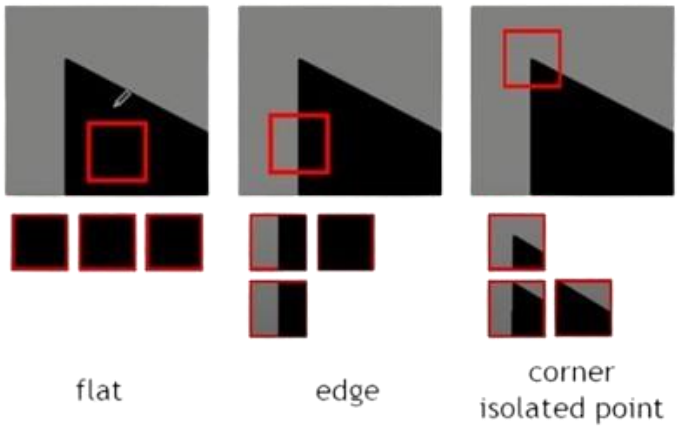
Table of Contents

Harris Corner Detection

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image.

Corner:

Corner is a distinctive feature of shapes and images that displays difference in intensity when shifted into different directions. They are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation, and illumination.



Corner Detection:

The idea is to consider a small window around each pixel p in an image. We want to identify all such pixel windows that are unique. Uniqueness can be measured by shifting each window by a small amount in a given direction and measuring the amount of change that occurs in the pixel values.

We take the sum squared difference (SSD) of the pixel values before and after the shift and identifying pixel windows where the SSD is large for shifts in all directions.

The Mathematics behind the Harris detector:

Let us define the change function E (u, v) as the sum of all the sum squared differences (SSD), where u, v are the x, y coordinates of every pixel in our 3 x 3 window and I is the intensity value of the pixel. The features in the image are all pixels that have large values of E (u, v) which is the minimum difference we take it as the cornerness response.

Change of intensity for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y) \right]^2$$

Window function — Shifted intensity — Intensity

So, we need to maximize this function E (u, v) for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps.

$$\sum [I(x+u, y+v) - I(x,y)]^2$$

$$\approx \sum [I(x,y) + uI_x + vI_y - I(x,y)]^2 \qquad \text{First order approx}$$

$$= \sum u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2$$

$$= \sum \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \qquad \text{Rewrite as matrix equation}$$

we get the final equation as:

$$E(u,v) \approx [u \quad v] \left(\sum \begin{bmatrix} {I_x}^2 & I_xI_y \\ I_xI_y & {I_y}^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

Now, we rename the auto-correlation matrix, and put it to be M:

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

Since we want the SSD to be large in shifts for all eight directions, By solving for the eigenvectors of M, we can obtain the directions for both the largest and smallest increases in SSD. The corresponding eigenvalues give us the actual value amount of these increases. A score, R, is calculated for each window:

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

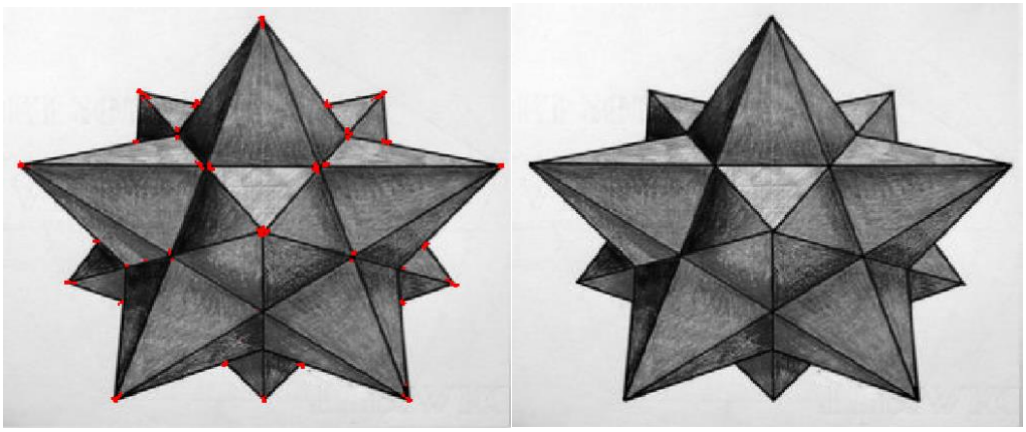
λ1 and λ2 are the eigenvalues of M. So, the values of these eigenvalues decide whether a region is a corner or not: When R is large, which happens when λ1 and λ2 are large and λ1∼λ2, the region is a corner.

Algorithm for Harris:

1. Compute the gradient at each point in the image.
2. Compute products of derivatives at every pixel.
3. Compute the sum of products of derivatives at every pixel by applying gaussian filter.
4. Create the matrix M form the entries in the gradient.
5. Compute the eigne values (which both should be strong) to calculate the corner response function R.
6. Apply threshold on value of R.
7. Compute non-maximum suppression to find local maxima of response function.

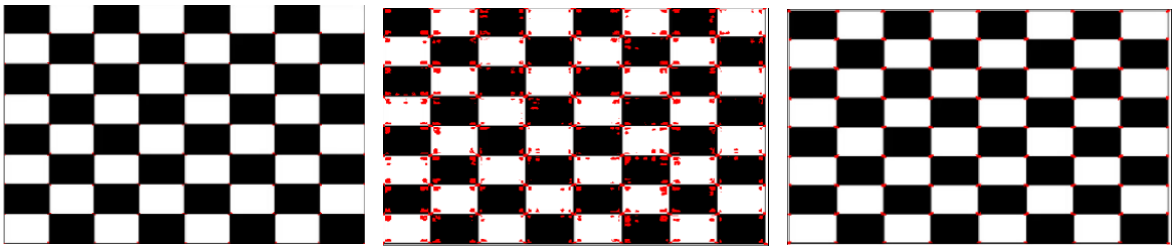
Observations and Results:

The local maxima points of every neighborhood are detected as corners and are displayed in the image as bright red regions.



Before Harris corner detection After Harris corner detection

A slider is added to control the value of the threshold to be able to change in the number of corners detected. When the threshold is **too low** it results in many weak corners to be detected which make the algorithm more prone to false positives. When the threshold is **too high** it may result in only strong corners being detected, potentially missing out on important but slightly weaker corner features and the algorithm may miss corners that are present in the image but do not meet the high threshold criteria.



Threshold value of 0.3 Threshold value of 0.01 Threshold value of 0.0

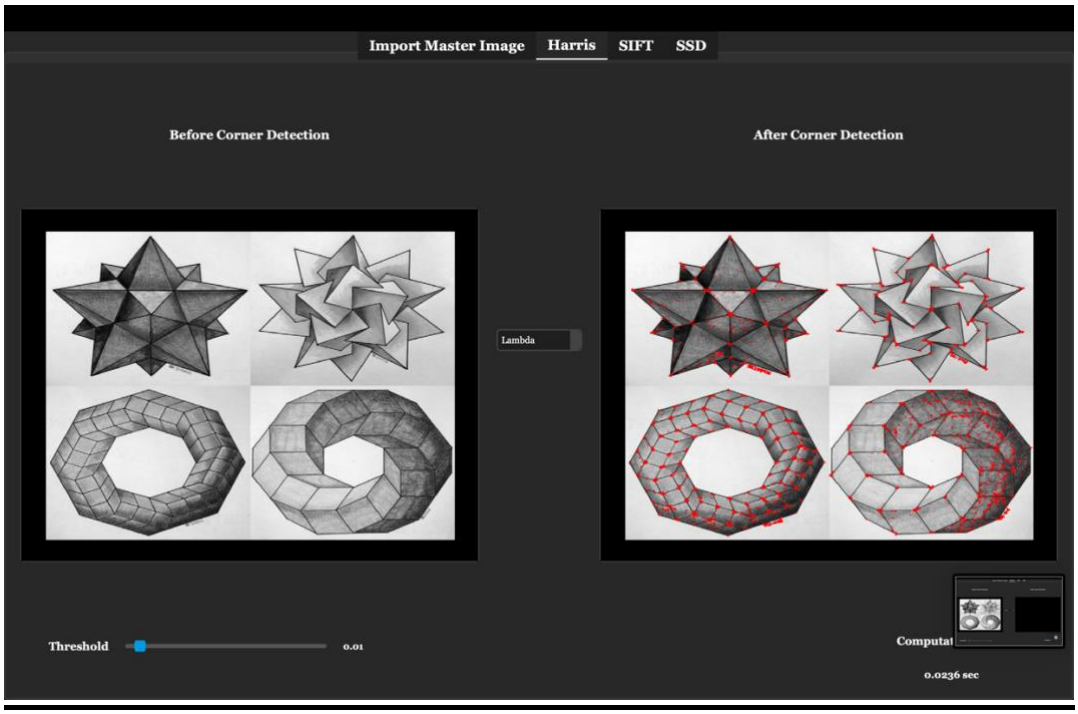
The computation time for Harris corner detection using OpenCV standard library was calculated and measured to be approximately equal **0.0034 seconds.**

The computation time for Harris corner detection implemented from scratch following the algorithm steps was calculated and measured to be approximately equal **0.0280 seconds.**

Lambda Minus Technique

The Lambda Minus method is a technique in computer vision primarily used for edge and corner detection. It operates by minimizing the smaller eigenvalue of the autocorrelation matrix, like how the Harris corner detector works but focuses on a different aspect of the matrix. While not as commonly referenced as Harris or SIFT, Lambda Minus provides an interesting approach for detecting features that are both robust and distinctive.

1. **Gradient Computation:** Calculate the horizontal and vertical gradients of the image, typically using operators like Sobel or Scharr.
2. **Autocorrelation Matrix Formation:** Form the autocorrelation matrix for each pixel using these gradients.
3. **Windowing Function:** Apply a Gaussian window to weight the contributions of nearby pixels to the autocorrelation matrix, emphasizing the locality of the matrix.
    a. Eigenvalue Analysis: Once the matrix is formed for each pixel, the eigenvalues are computed. The key step in Lambda Minus is to focus on the smaller eigenvalue of this matrix:
    b. **Eigenvalue Calculation:** Compute the two eigenvalues.
4. **Threshold:** is applied to determine whether a given pixel can be considered a feature point (corner or edge).



SIFT (Scale Invariant Feature Transform)

SIFT stands for Scale-Invariant Feature Transform and was first presented in 2004, by **D. Lowe**, University of British Columbia. SIFT is invariant to image scale and rotation.

There are mainly four steps involved in the SIFT algorithm.

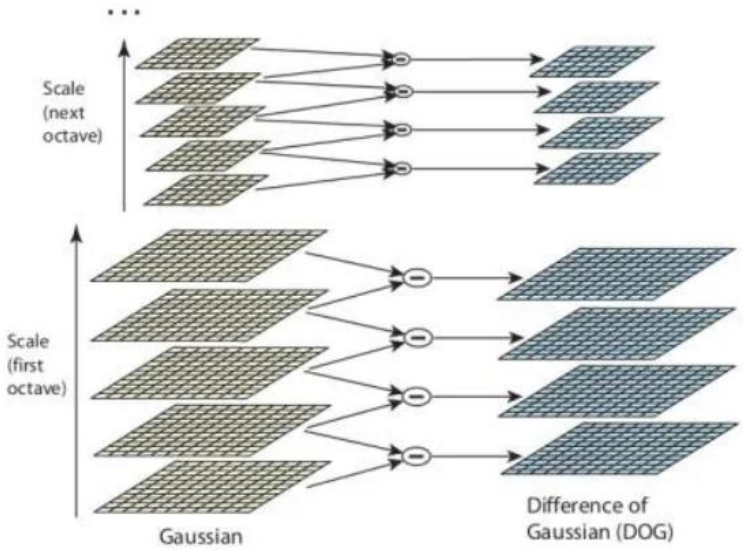
1- **Scale-space Peak Selection:** Potential location for finding features.
2- **Keypoint Localization:** Accurately locating the feature keypoints.
3- **Orientation Assignment:** Assigning orientation to keypoints.
4- **Keypoint descriptor:** Describing the keypoints as a high dimensional vector.
5- **Keypoint Matching**

1- Scale-space Peak Selection

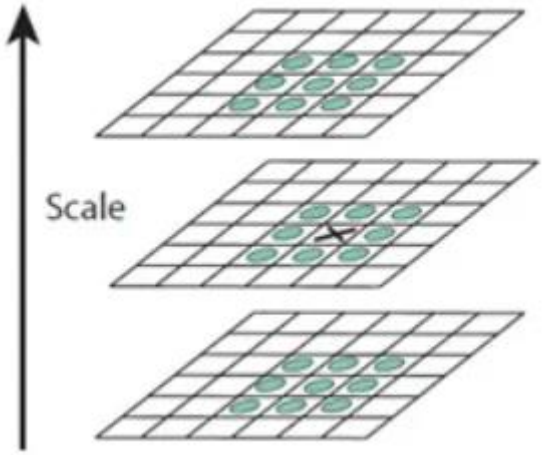
The scale space of an image is a function L(x,y,σ) that is produced from the convolution of a Gaussian kernel (Blurring) at different scales with the input image. Scale-space is separated into **octaves** and the number of **octaves** and **scale** depends on the size of the original image. So, we generate several octaves of the original image. Each octave's image size is **half** the previous one.

Image Blurring
Within an octave, images are progressively blurred using the Gaussian Blur operator. Mathematically, "blurring" is referred to as the convolution of the Gaussian operator and the image. Gaussian blur has a particular expression or "operator" that is applied to each pixel. What results is the blurred image.

DOG (Difference of Gaussian)
Now we use those blurred images to generate another set of images, the Difference of Gaussians (DoG). These DoG images are great for finding out interesting keypoints in the image. The difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and kσ. This process is done for different octaves of the image in the Gaussian Pyramid. It is represented in below image:



Finding Keypoints
One pixel in an image is compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales. This way, a total of 26 checks are made. If it is a local extremum, it is a potential keypoint. It basically means that keypoint is best represented in that scale.



2- Keypoint Localization

Keypoints generated in the previous step produce a lot of keypoints. Some of them lie along an edge, or they don't have enough contrast. In both cases, they are not as useful as features. So, we get rid of them by iteratively refining keypoints based on **quadratic fitting** and the **Hessian matrix**, the algorithm can remove keypoints located in flat regions and along edges, focusing on identifying keypoints located at stable features like corners.

Gradient Calculation:

Gradient Calculation helps in finding the direction of steepest ascent or descent at the keypoint, which helps determine the direction to refine the keypoint location.

The gradient at the center pixel of a 3x3x3 cube is computed using central differences.

$$\frac{\partial f}{\partial x} = \frac{f(x+h,y,s) - f(x-h,y,s)}{2h}$$

$$\frac{\partial f}{\partial y} = \frac{f(x,y+h,s) - f(x,y-h,s)}{2h}$$

$$\frac{\partial f}{\partial s} = \frac{f(x,y,s+h) - f(x-h,y,s-h)}{2h}$$

Hessian Matrix Calculation:

- Hessian Matrix calculations approximate the curvature of the function at the keypoint, which helps determine the rate of change of the gradient and further refine the keypoint location.
- The Hessian matrix at the center pixel is computed to approximate second-order derivatives also using central differences.

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial s} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial s} \\ \frac{\partial^2 f}{\partial s \partial x} & \frac{\partial^2 f}{\partial s \partial y} & \frac{\partial^2 f}{\partial s^2} \end{bmatrix}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x+h,y,s) - 2f(x,y,s) + f(x-h,y,s)}{h^2}$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{f(x,y+h,s) - 2f(x,y,s) + f(x,y-h,s)}{h^2}$$

$$\frac{\partial^2 f}{\partial s^2} = \frac{f(x,y,s+h) - 2f(x,y,s) + f(x,y,s-h)}{h^2}$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{f(x+h,y+h,s) - 2f(x+h,y-h,s) - f(x-h,y+h,s) + f(x-h,y,s)}{4h^2}$$

$$\frac{\partial^2 f}{\partial x \partial s} = \frac{f(x+h,y,s+h) - 2f(x+h,y,s-h) - f(x-h,y,s+h) + f(x-h,y,s-h)}{4h^2}$$

$$\frac{\partial^2 f}{\partial y \partial s} = \frac{f(x,y+h,s+h) - 2f(x,y+h,s-h) - f(x,y-h,s+h) + f(x,y-h,s-h)}{4h^2}$$

Extremum Update Calculation:

The purpose of this is to find the optimal direction and magnitude to update the keypoint location, ensuring that it converges to the true extremum.

The extremum update vector is computed by solving the linear system $\mathbf{H} \cdot \Delta x = -\nabla f$,where **H** is the Hessian matrix and ∇f is the gradient vector.

Application of Extremum Update:

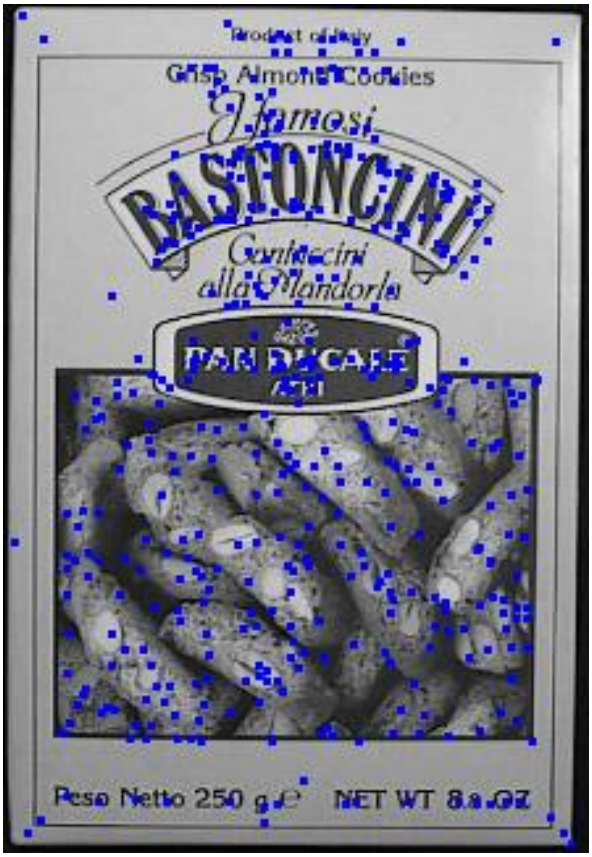
The computed extremum update vector Δx is applied to the initial keypoint coordinates to refine its location.

Iterative Refinement:

The refinement process is iterated until convergence, ensuring keypoints are accurately located at subpixel levels.

Results

First Image



First Image Keypoints

Second Image

Second Image Keypoints

3- Keypoints Orientations

Keypoints Orientations Calculation helps enhance the robustness of the SIFT algorithm by assigning multiple orientations to keypoints, making them more invariant to image transformations like rotation.

Scale Calculation:
The scale of the keypoint is calculated based on the keypoint size and the octave index. This scale is used to determine the radius of the region around the keypoint for orientation computation.

Orientation Histogram:
For each pixel in the region around the keypoint, the gradient magnitude and orientation are computed. These values are used to build a histogram of gradient orientations, weighted by the gradient magnitude and a Gaussian weight based on the distance from the keypoint.

Smoothing:
The raw histogram is smoothed to reduce noise. This is done by averaging each bin with its neighboring bins.

Peak Detection:
Peaks in the smoothed histogram are identified as potential orientations. Peaks are detected where a bin has a higher value than its neighbors.

Peak Interpolation:
Quadratic interpolation is used to refine the peak locations for better accuracy. This interpolation finds the precise peak location by fitting a quadratic curve to the three neighboring histogram bins around the peak.

Orientation Assignment:
Orientations are assigned to keypoints based on the refined peak locations. If a peak value is sufficiently higher than the others, an orientation is assigned to the keypoint at that peak angle. The orientation is converted to a value between 0 and 360 degrees.

Keypoint Creation:
For each assigned orientation, a new keypoint is created with the same location, size, and response as the original keypoint, but with the new orientation. These keypoints with orientations are added to the list of keypoints with orientations for further processing.

4- Descriptors Generation

Descriptor Generation:

A 3D histogram tensor is constructed to represent the keypoint's local gradient distribution. Trilinear interpolation is used to smooth the histogram, and the descriptor vector is computed by flattening the smoothed histogram tensor.

Descriptor Normalization:
The descriptor vector is normalized by dividing it by the maximum norm value of the vector. This step ensures that the descriptor is invariant to changes in illumination and contrast.

Descriptor Quantization:
The normalized descriptor values are multiplied by 512, rounded, and saturated between 0 and 255 to convert them from float32 to unsigned char, following the OpenCV convention for descriptors. This quantization step reduces the memory required to store descriptors and makes them suitable for further processing and matching.

Features Matching

Template matching is a common technique used in various applications such as object recognition, image alignment, and motion tracking. It involves comparing a template image against a larger image to find instances of the template.

In this report, we present an implementation of a template matching algorithm utilizing SIFT descriptors. SIFT descriptors are robust to changes in scale, rotation, and illumination, making them suitable for matching keypoints across images.

1. Methodology:

1.1. Key Components:

- **SIFT Descriptor Extraction:** Keypoints and descriptors are extracted from both the input image and the template image using the Scale-Invariant Feature Transform (SIFT) algorithm.

- **Matching Methods:** Two matching methods are implemented:
  - Normalized Cross Correlation: Measures the similarity between patches using normalized cross-correlation.

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

  - Normalized Sum of Squared Differences: Measures the dissimilarity between patches using normalized sum of squared differences.

$$R(x,y) = \frac{\sum_{x'y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x'y'} T(x',y')^2 . \sum_{x'y'} I(x+x',y+y')^2}}$$

- **Matching Process:** The keypoints and descriptors from the input image are compared with those of the template image using the selected matching method.

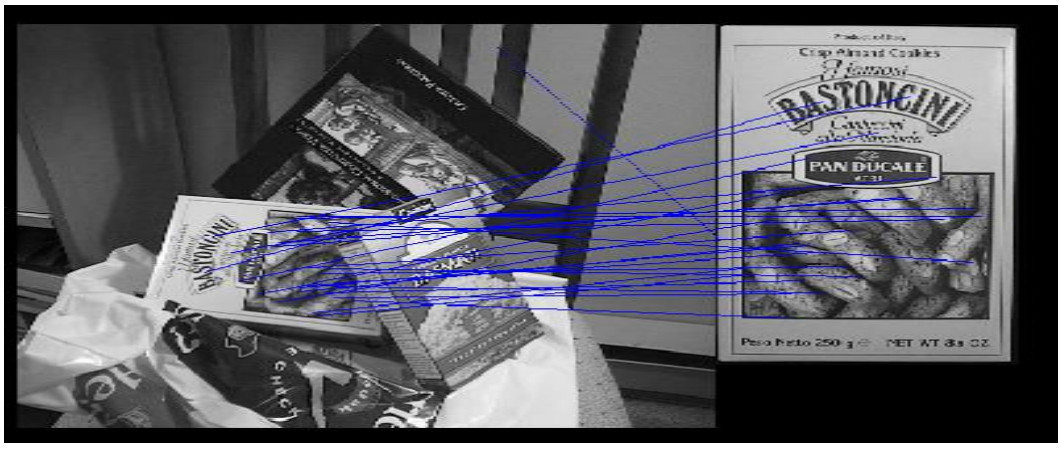
- **Filtering Repeated Points:** Repeated keypoints are filtered, and only the most significant matches are retained.

2. Results:

Image 1

Image 2

Applying Normalized Cross Correlation (NCC)



Applying Sum Square Difference (SSD)