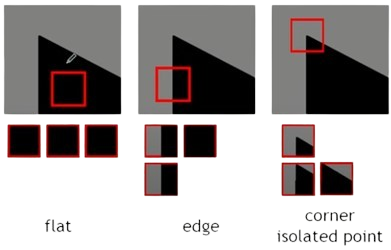
**Harris Corner Detection**

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.



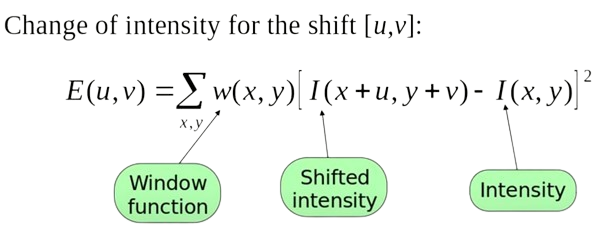
**Corner Detection:**

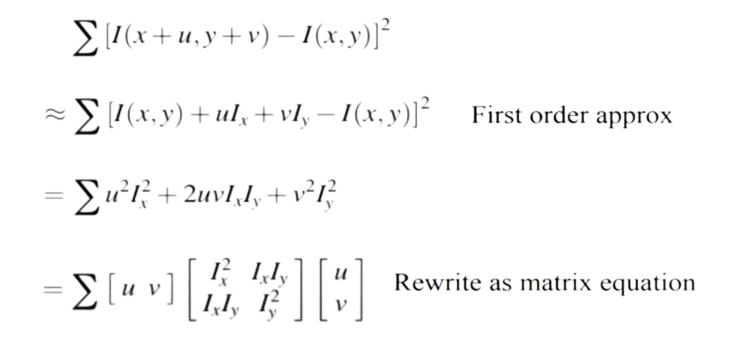
The idea is to consider a small window around each pixel p in an image. We want to identify all such pixel windows that are unique. Uniqueness can be measured by shifting each window by a small amount in a given direction and measuring the amount of change that occurs in the pixel values.

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.

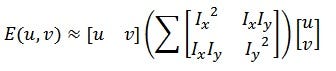
**The Mathematics behind the Harris detector:**

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.

****

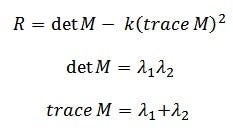
So, we need to maximize this function E (u, v) for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps.

we get the final equation as:



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

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:



λ1 and λ2 are the eigenvalues of M. So, the values of these eigenvalues decide whether a region is a corner or not:

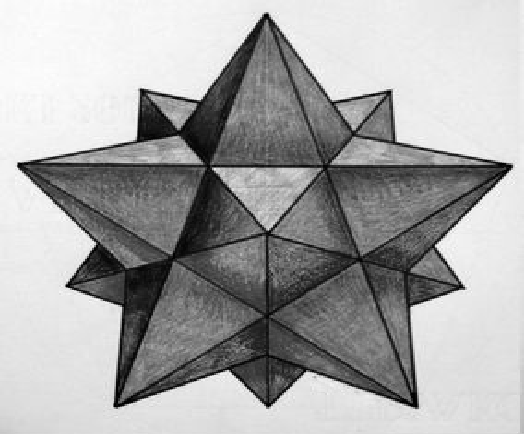
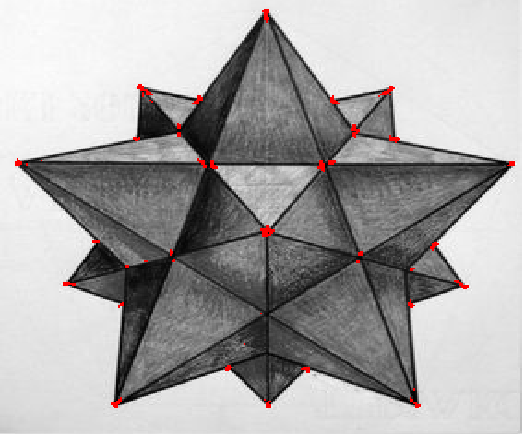
When R is large, which happens when λ1 and λ2 are large and λ1∼λ2, the region is a corner.

**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.

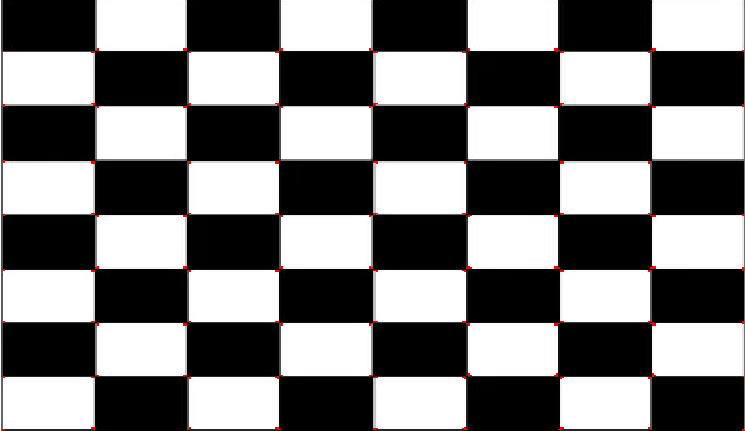
**Observations and Results:**

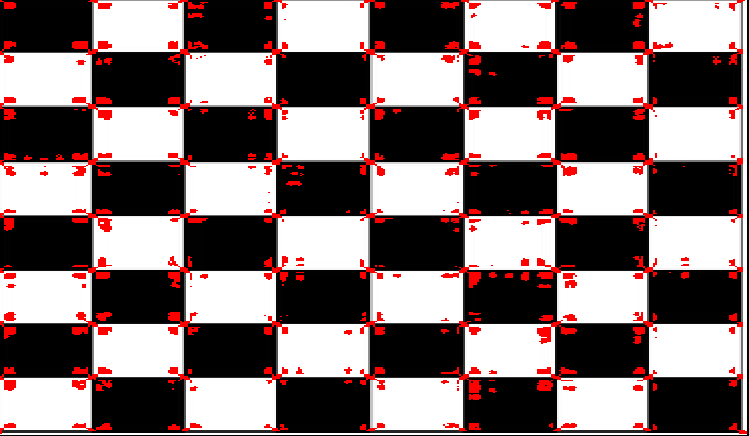
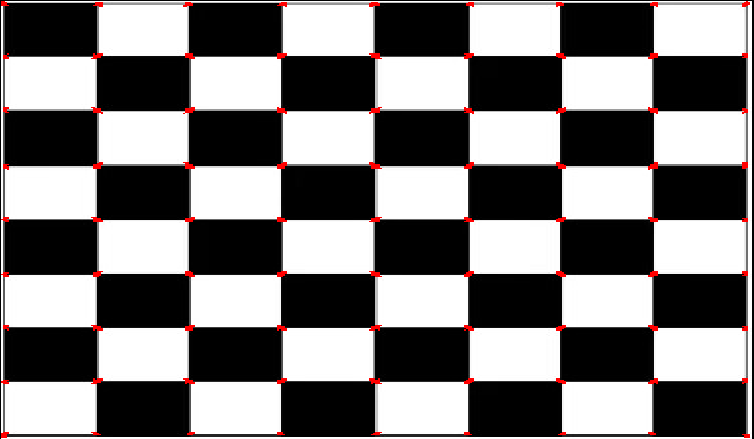
The local maxima points of every neighborhood are detected as corners and are displayed in the image as bright red regions.



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 0.0*

*Threshold value of 0.3 Threshold value of 0.01*

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.**

**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**
6. **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:

A diagram of a diagram of a person

Description automatically generated with medium confidence

**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.

A diagram of a game

Description automatically generated

1. **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.

**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.

**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 ,where **H** is the Hessian matrix and is the gradient vector.

**Application of Extremum Update:**

The computed extremum update vector 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.

**Features Matching**

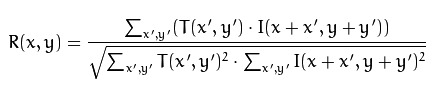
Template matching is a common technique used in various applications such as object recognition, image alignment, and motion tracking. It involves comparing a template image against a larger image to find instances of the template.

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.

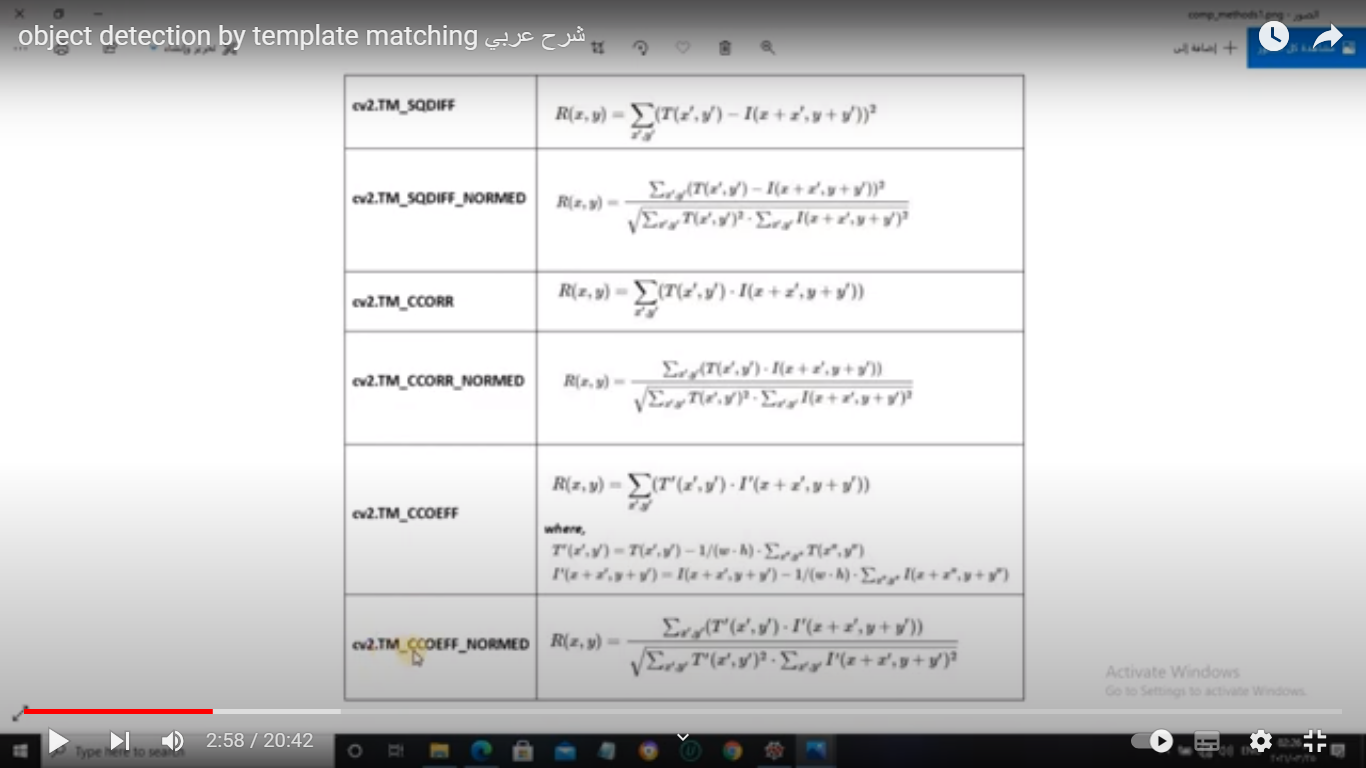
**1. Methodology:**

**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.



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



* **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.

**1.2. Implementation Details:**

The algorithm is implemented in Python using the OpenCV and PyQt5 libraries. The GUI allows users to select images and matching methods interactively.

**2. Results:**

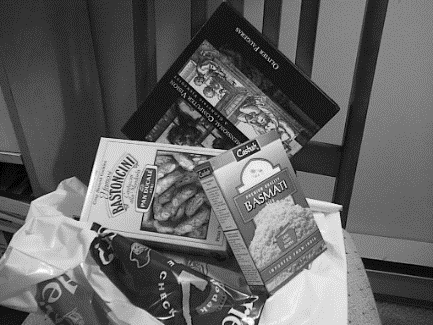
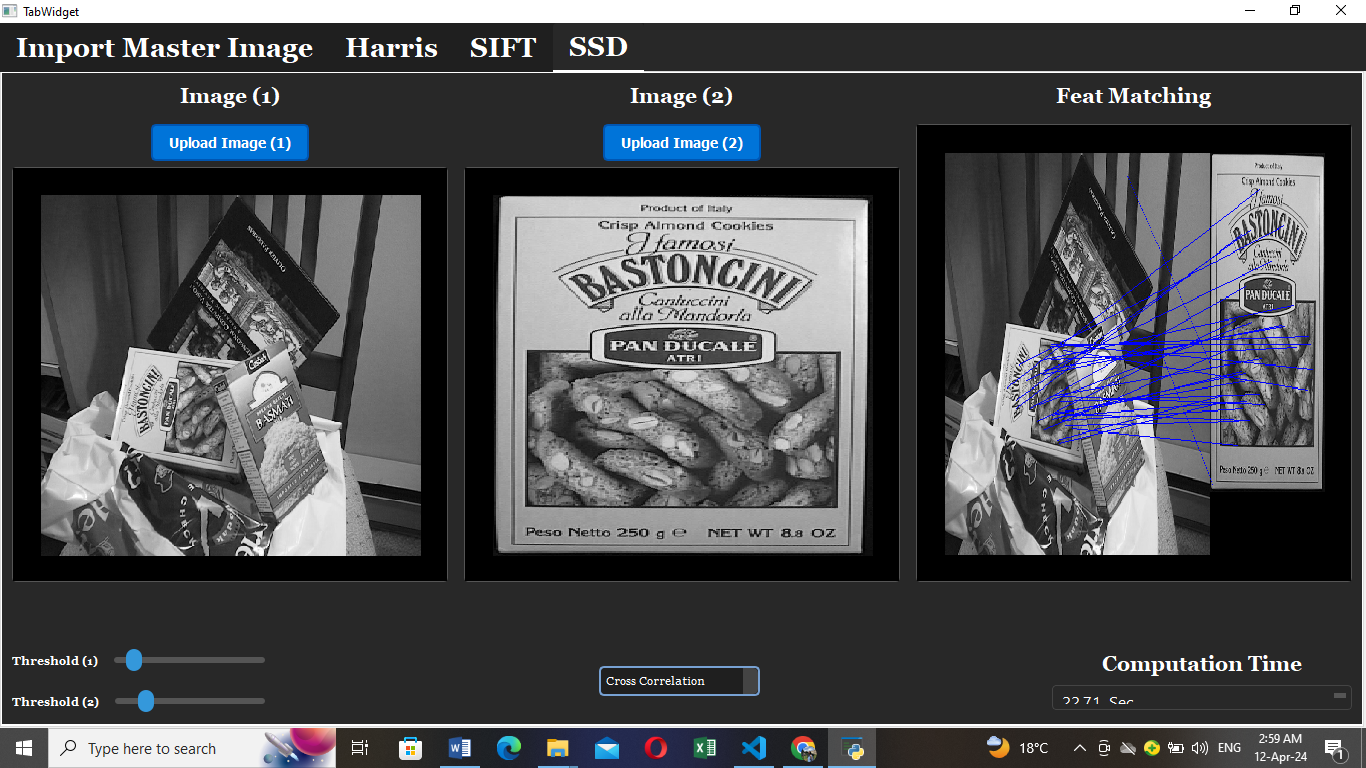
** **

Image 1 before Image 2 before

**Applying Cross Correlation:**



**Applying Sum Square Difference:**

