Cairo University

Faculty of Engineering

Systems and Biomedical Department

Spring 2024

Computer Vision(SBE 3230)

Assignment 3
"Image matching and Feature extraction"

Abdulrahman Emad

Mariam Ahmed Saied

Mourad Magdy

Youssef Ashraf

Ziad Ahmed Meligy

Under the supervision of

Prof/Ahmed Badawy

Eng/Layla Abbas

Eng/Omar Hesham

Table of Contents

Table of Contents **2**
Feature Extraction **3**
Harris Operator 3
Steps of the algorithm: 3
Lambda Minus 4
Intro: 4
Results and time comparison: 5
Harris Operator VS Lambda Minus 6
Harris Detector: 6
Shi-Tomasi Detector: 6
SIFT **7**
Algorithm: 7
Challenges with implementation: 8
Program Output Sample 9
Image Matching **10**
The sum of squared differences (SSD): 10
Algorithm: 10
Challenges with the Algorithm: 10
Normalized Cross Correlation: 10
Normalized Cross-Correlation Algorithm Steps: 10
Problems with this algorithm: 11
Results and Time: 11
Matching SIFT keypoints 11
Algorithm: 11
Results: 12

Feature Extraction

Harris Operator

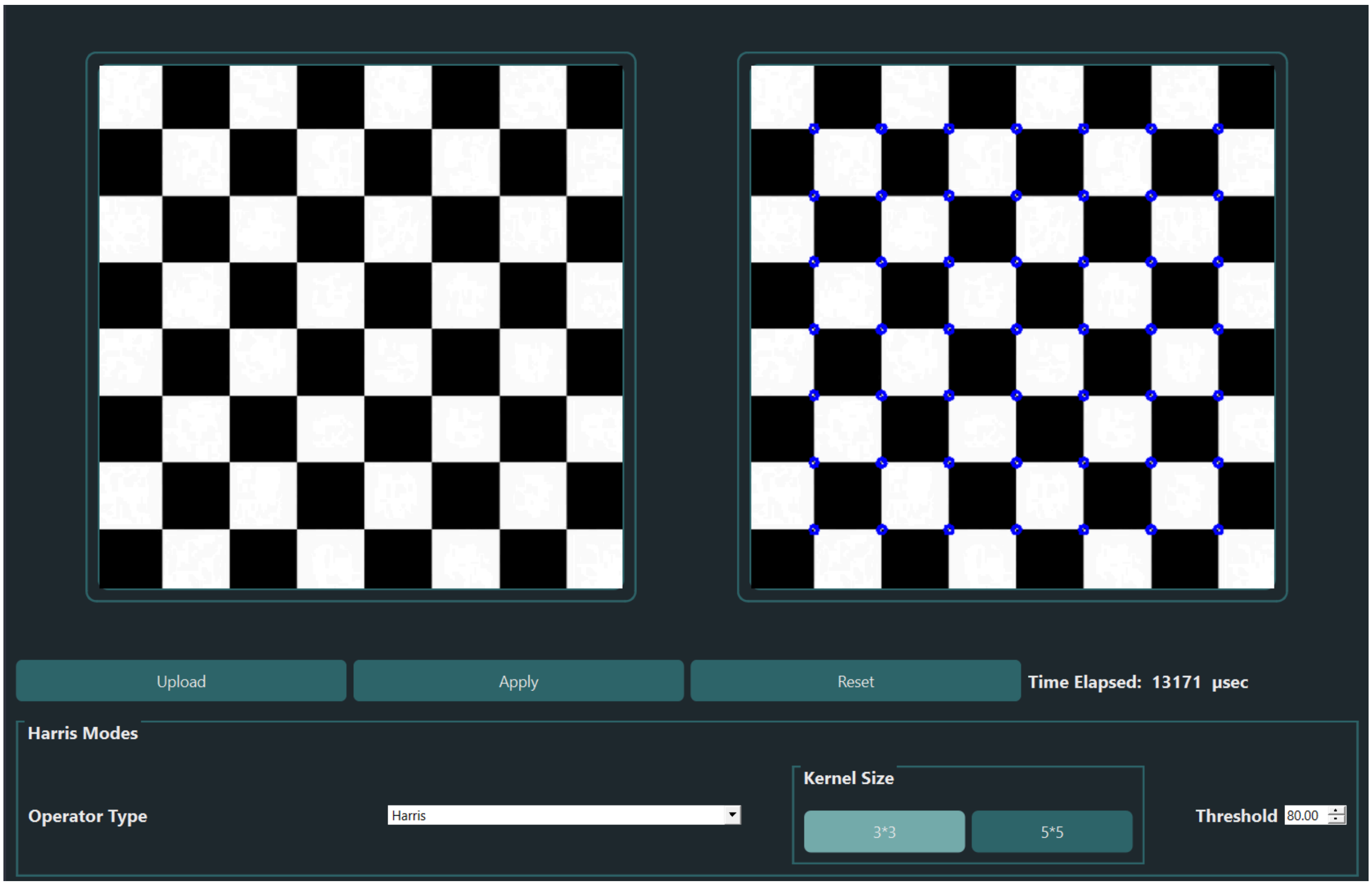
Steps of the algorithm:

- Apply a smoothing filter (**gaussian**) to smooth out any noise.
- Apply the Sobel operator to find the x and y gradients.
- For each pixel consider a 3x3 or 5x5 window around it and compute the corner strength function (Harris value).
- Find the pixels that exceed the threshold and are local maximum within a certain window then calculate the feature descriptor.

$$N = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (1)$$

R = Det (N) – K (Trace (N))2 ––-> (2)

- N is the Harris matrix used in the second equation (2)
- R is the measure of corner response.



Time Elapsed: 13171 µsec

Harris Modes

Operator Type: Harris

Kernel Size

Threshold 80.00

- This is the output of the Harris operator with a 3x3 kernel size.
- The time elapsed is about 12000 to 13000 microseconds.

Lambda Minus

Intro:

The Shi-Tomasi Detector, developed by J. Shi and C. Tomasi, is a modification of the Harris corner detector. While both detectors are based on similar principles, the Shi-Tomasi detector introduces a crucial difference in the selection criteria for identifying corners.

The core divergence lies in how the detectors evaluate the "score" or the measure of cornerness for each pixel. In the Harris detector, the score (often denoted as
R
R) is computed using both eigenvalues obtained from the structure tensor. This score is then compared to a threshold to determine if a pixel is considered a corner. The formula for computing the score in the Harris detector is:

$$R = \lambda 1 \text{ x } \lambda 2 - k(\lambda 1 + \lambda 2)\text{^}2$$

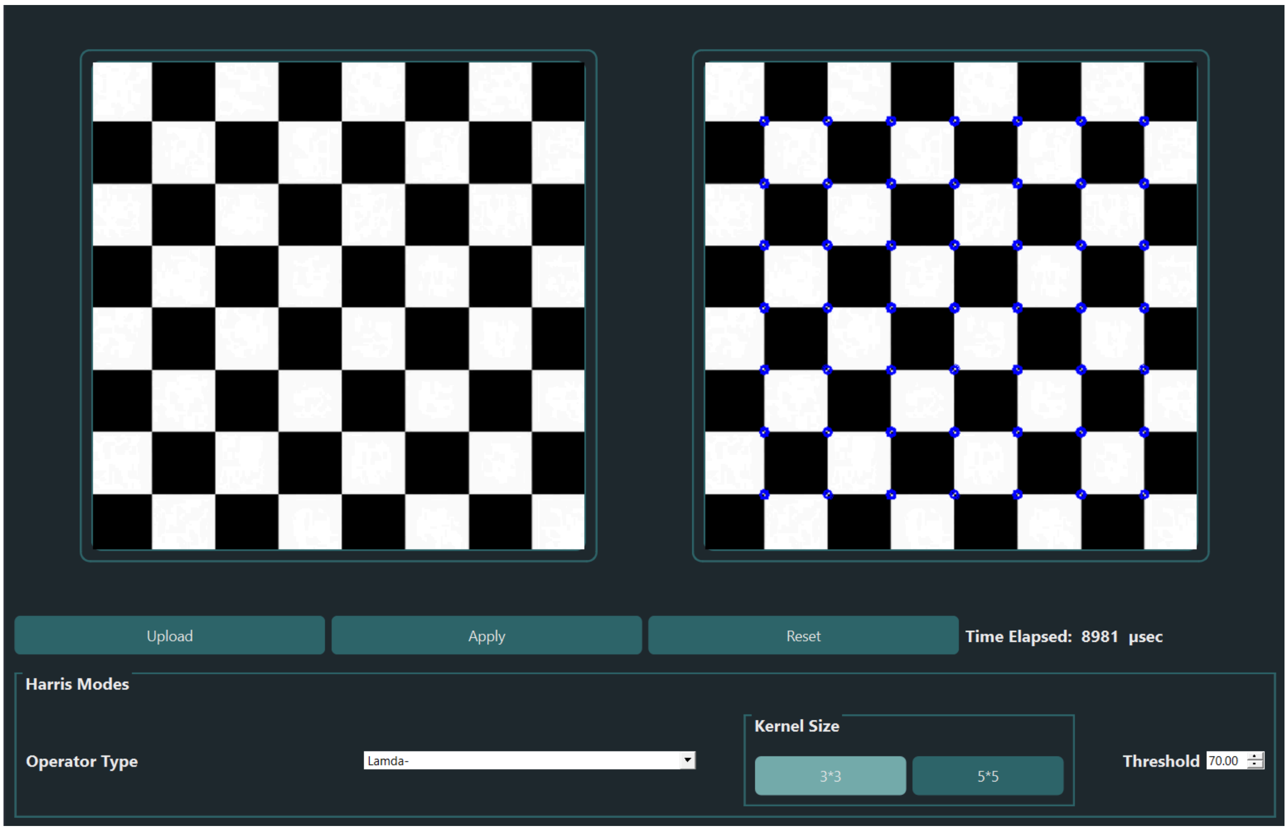
Here, $\lambda 1$ and $\lambda 2$ are the eigenvalues of the structure tensor, and k is a constant typically set to 0.04.
However, the Shi-Tomasi detector simplifies this process by discarding the complex function that combines both eigenvalues. Instead, it directly uses the minimum of the two eigenvalues to compute the score. The formula for computing the score in the Shi-Tomasi detector is:

$$R = min(\lambda 1 , \lambda 2)$$

This modification was empirically shown by Shi and Tomasi to be more effective in identifying corners, especially in cases where the Harris detector might fail. By directly considering the minimum eigenvalue, the Shi-Tomasi detector tends to select points in regions where there is high contrast along one direction, which often corresponds to corners or interesting features in the image.

Results and time comparison:

Kernel Size:3x3



Time Elapsed: 8981 µsec

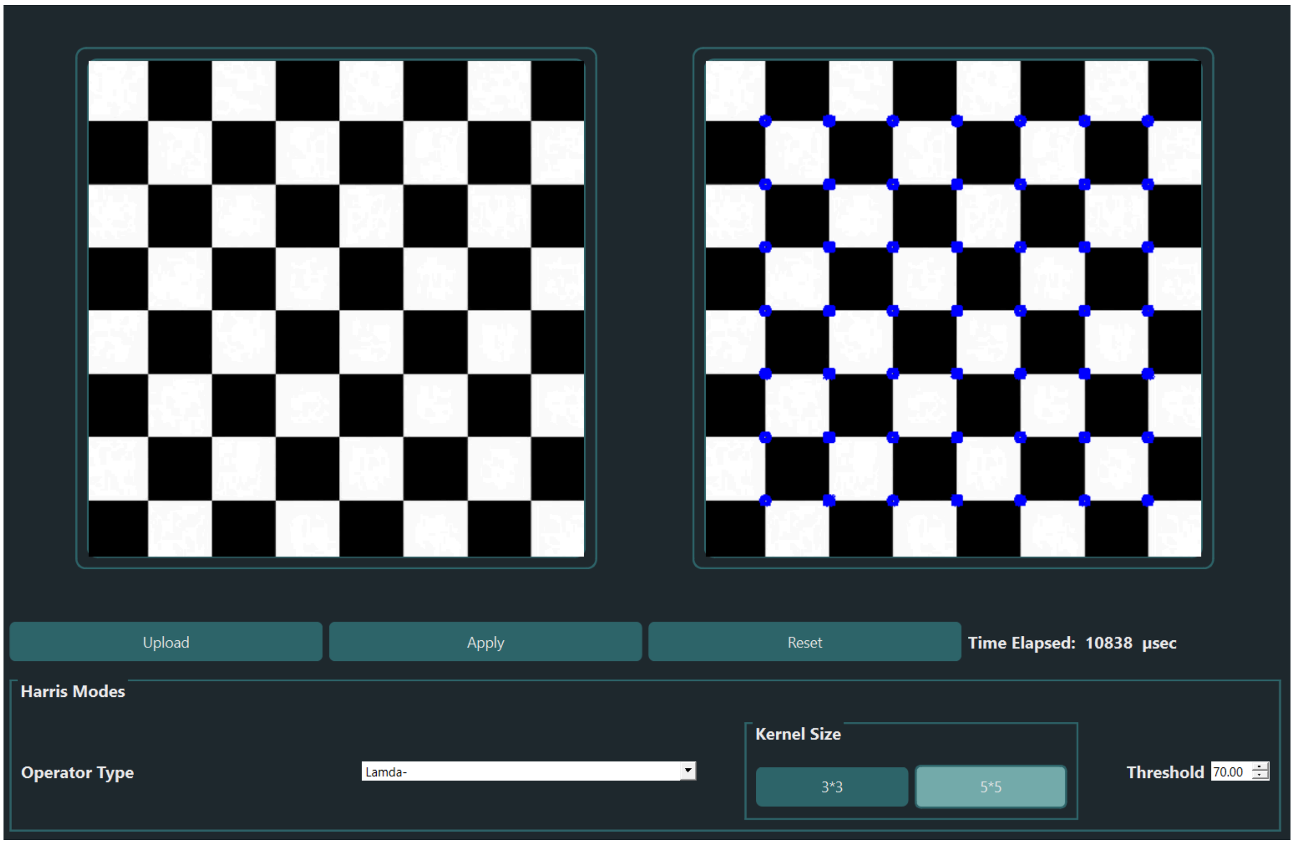
Harris Modes

Operator Type Lamda-

Kernel Size 3*3 5*5

Threshold 70.00

Kernel Size:5x5



Time Elapsed: 10838 µsec

Harris Modes

Operator Type Lamda-

Kernel Size 3*3 5*5

Threshold 70.00

Harris Operator VS Lambda Minus

Harris Detector:

The Harris detector is well-known for its ability to detect corners robustly, even in noisy images. It identifies image patches that show significant changes in intensity when shifted both horizontally and vertically. The core of the Harris detector lies in its auto-correlation function, which measures intensity variation when a window is shifted around a point. This function computes variations in intensity to effectively detect corners.

To compute the corner response, the Harris detector evaluates the eigenvalues of a matrix calculated from image gradients. These eigenvalues help categorize the window as flat, edge, or corner. A high corner response value indicates a corner, while a negative value suggests an edge. Non-maximum suppression is employed to select optimal corners.

Shi-Tomasi Detector:

The Shi-Tomasi detector is closely related to the Harris detector but differs in its corner selection criterion. While Harris computes a score for each pixel based on eigenvalues, Shi-Tomasi simplifies the scoring process by using only the minimum eigenvalue. This approach has been empirically proven to yield better results than Harris. By comparing this minimum eigenvalue against a threshold value, Shi-Tomasi identifies interest points. The region of influence for a pixel to be considered an interest point is determined accordingly.

SIFT

Humans can easily identify objects in images even if there are variations in the angles or scales, however, such task would be very difficult for a machine, as normal image comparison would go pixel by pixel comparison, which is very inefficient, and time consuming, as the illumination of the picture, or the size of the objects in the picture or the angle which the image was taken with or the rotation of the objects wouldn't be the same, thus a pixel by pixel comparison would fail to compare pictures properly and identify objects.

That's why a Scale Invariant Feature Transform (SIFT) algorithm is needed, so that machines could be trained to identify images at nearly human level, which is done by extracting features from the original image and extracting features from the image to be identified and compare features together rather than comparing pixels leading to a comparison method that isn't affected scales or rotations, other methods like MOPS do the same task but the features extracted are affine transformations invariant.

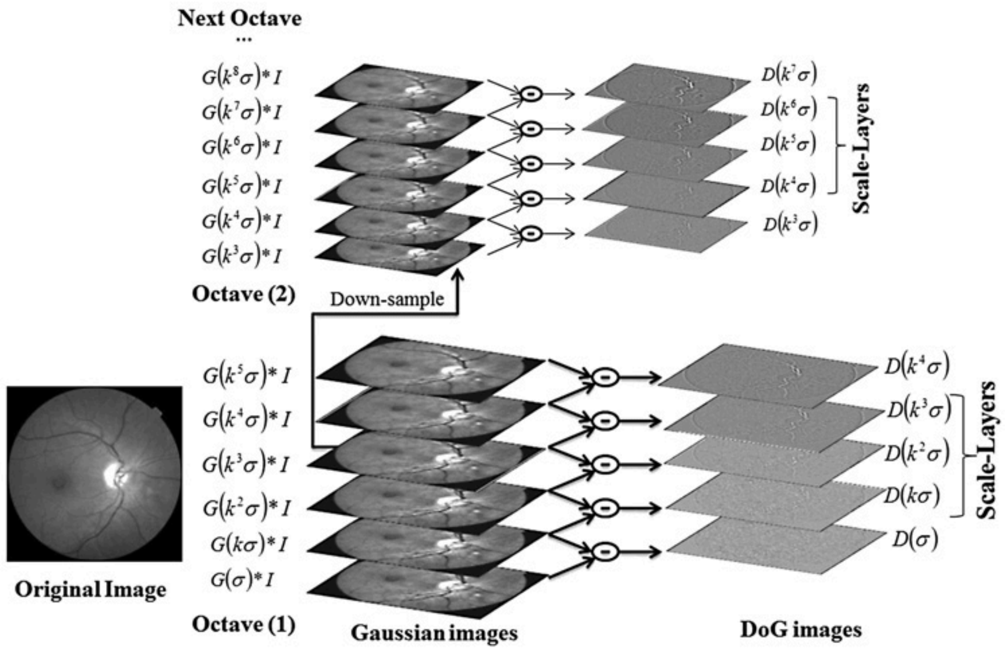
Algorithm:

First we need to generate our scale space so we do the following:

1. Convert the image to the greyscale and generate the scale space pyramid which has the number of images equal to the number of octaves.
2. Use Gaussian Filter to blur images at different scales and combine them in the Scale Space.

Then Compute the Difference of Gaussian (DoG) as an approximation to the Laplacian of Gaussian (LoG) as following:

1. For each octave in the scale space subtract the adjacent scales
2. Then Identify the potential keypoints



Then we need the Keypoint Detection:

After getting the LoG, we shall detect points that are either the local extrema or minima in the scale space.

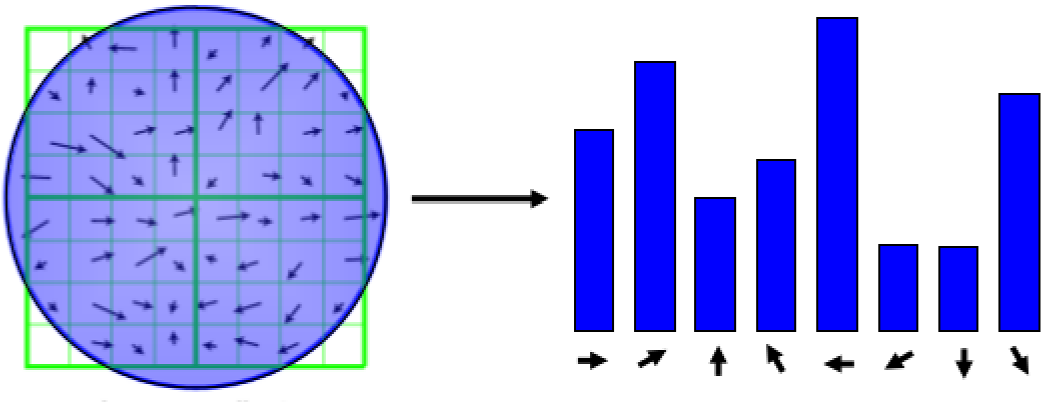
1. The steps go as follows:
2. Identify points that are a local extrema or minima in scale space
3. Refine the detected key points by eliminating any low-contrast key points or any points that are heavily influenced by edges or located on edges

4. Store these key points a the suitable data structure

The functions that apply these steps are: **get_keypoints**, **remove_low_contrast**, and **remove_edges functions.**

After that we need to assign the Orientation to the Keypoints as the following:

1. Calculate the dominant gradient direction in the neighborhood of each keypoint.
2. Assign this to the key point.



Then you need Descriptor Calculation:

1. You compute descriptors for each key point in a local image patch
2. For each key point, in each region you compute histograms, gradient magnitude and orientation.
3. Then you concatenate the histograms in order to get a key point descriptor.

The function that applies this step is the **get_descriptors** function.

Descriptor Post-Processing:

1. Apply normalization to descriptors as to make the illumination invariance
2. Reduce large magnitudes to improve robustness

These steps are implemented in the **get_luminosity_invariance** function.

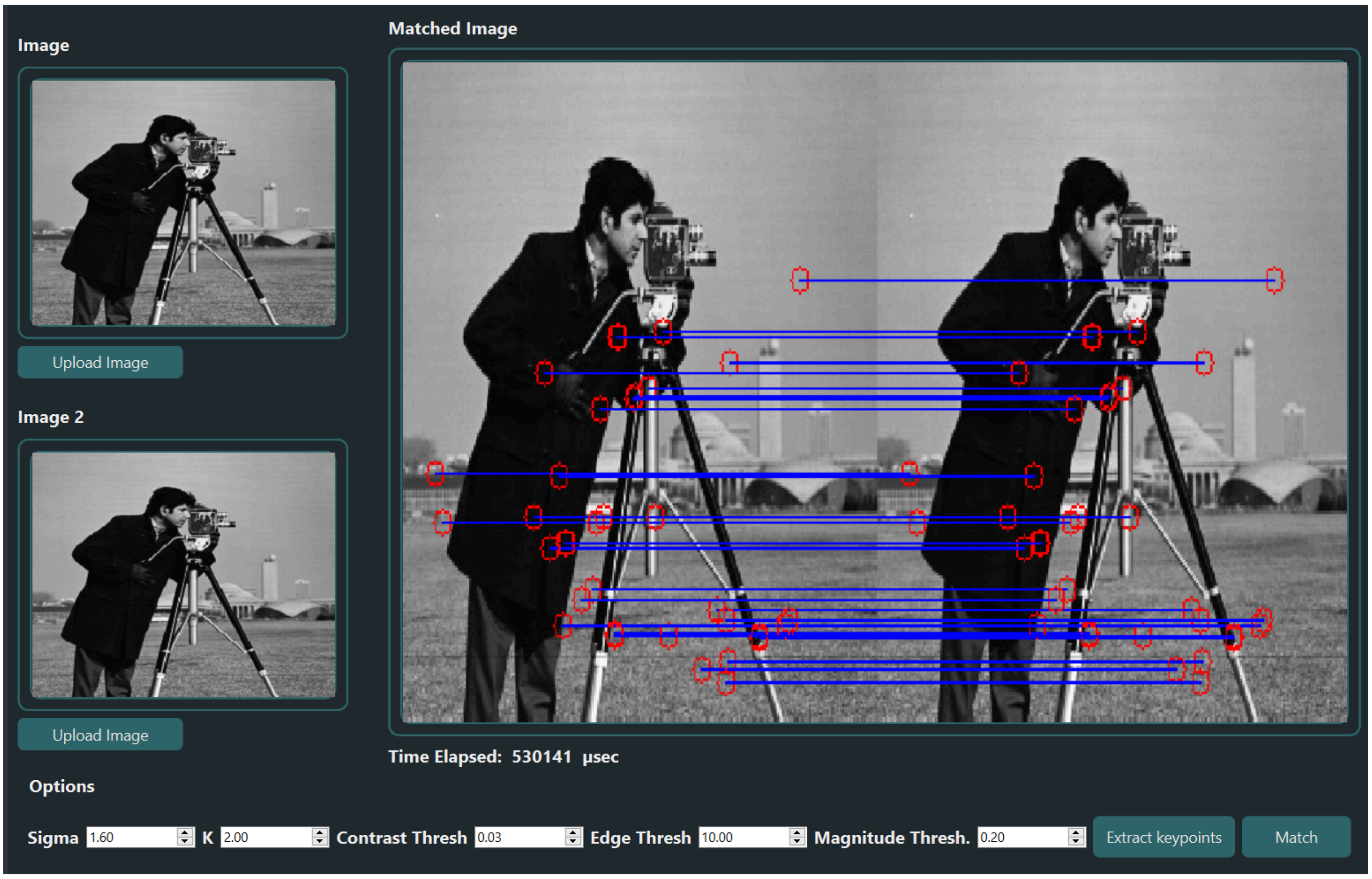
Finally extract your descriptors as following:

1. Orchestrate the entire SIFT descriptor extraction process by calling the aforementioned functions in sequence and providing necessary parameters.

Challenges with implementation:

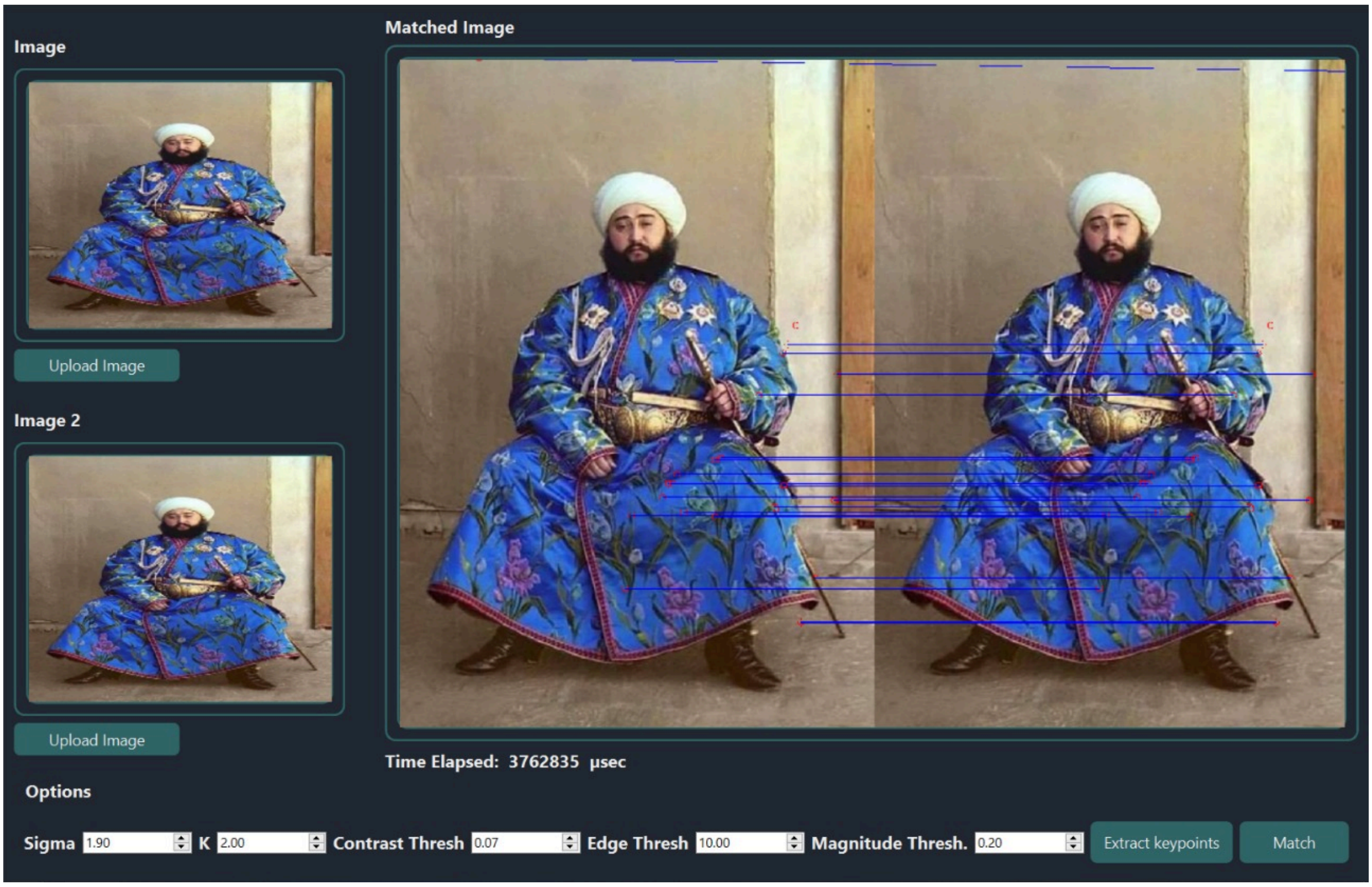
1. **Complexity:** Implementing SIFT involves a lot of steps like scale-space extrema detection and descriptor computation.
2. **Performance:** Optimizing for speed and memory usage, especially when using C++ optimization is easier due to being able to work on lower level more than Python
3. **Complex Data Types:** SIFT needs complex data types to work with (like vector of vector of matrices), in order to deal with such data types and understanding is challenging

Program Output Sample



Time Elapsed: 530141 μsec

Sigma 1.60 K 2.00 Contrast Thresh 0.03 Edge Thresh 10.00 Magnitude Thresh. 0.20



Time Elapsed: 3762835 μsec

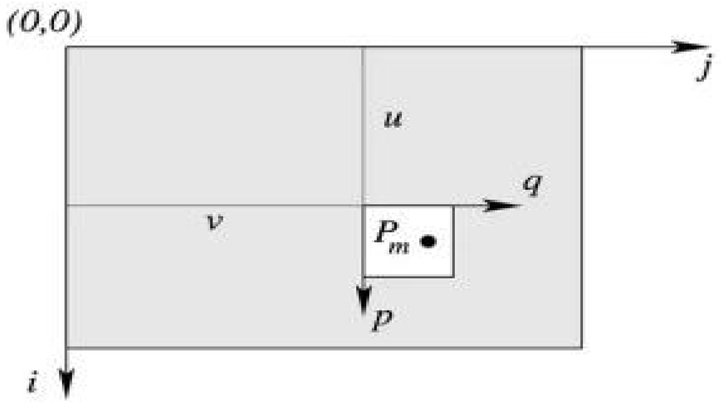
Sigma 1.90 K 2.00 Contrast Thresh 0.07 Edge Thresh 10.00 Magnitude Thresh. 0.20

Image Matching

The sum of squared differences (SSD):

Algorithm:

- **Intersection cutting**:- Identify the intersection between the template and the image
    - Cut this intersection from the image.
- **Template subtraction:** - subtract the template from the cut image
- **Squared difference calculation**:- square the difference between corresponding pixel values
    - Sum these squared differences to obtain a single number representing the similarity between the template and the image region
- **Template shifting**: shift the template by one pixel and repeat the previous steps
- **Minimum value identification**: -After calculating the similarity values for all shifts, find the min value.
    - Retrieve the index corresponding to this min value, this index represents the displacement [u,v] needed to align the template with the best match in the image.



Challenges with the Algorithm:

1. Sensitivity to Brightness and Contrast:
The algorithm may fail to perform accurately in the presence of changes in image brightness or contrast. These alterations can affect the pixel values and consequently influence the dissimilarity calculation.

2. Inability to Handle Template Rotations:
The algorithm does not account for template rotations. It only considers translational shifts of the template within the image. Rotations of the template may result in poor matching accuracy.

Normalized Cross Correlation:

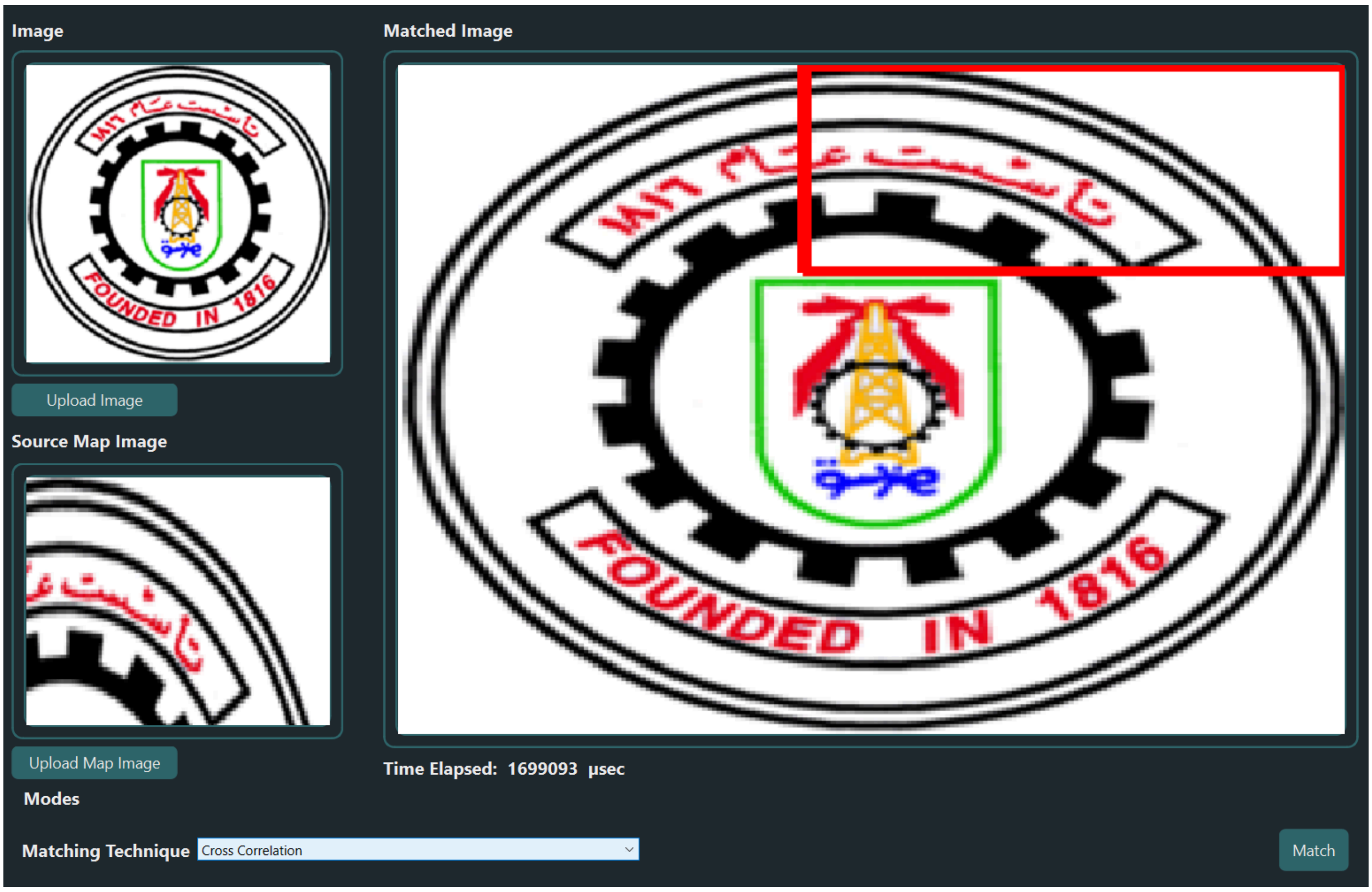
Normalized Cross-Correlation Algorithm Steps:

1. Cut the intersection between the template and the image.
2. Calculate the variance of the template image.
3. Calculate the variance of the cut image.
4. Compute the variance between these two images.
5. Repeat steps 1-4 for each pair of coordinates (u,v) in the equation.
6. Find the maximum value among the computed variances.

Problems with this algorithm:

1. It is very expensive to calculate the variance each time, but this can be made faster using pyramids for calculation.
2. The algorithm cannot deal with rotations, but this limitation can be improved upon.

Results and Time:



Time Elapsed: 1699093 µsec

Matching Technique: Cross Correlation

Matching SIFT keypoints

The matching of descriptors extracted from images is a fundamental task in computer vision, essential for various applications such as object recognition, image stitching, and 3D reconstruction. The implementation of a custom matching algorithm designed specifically for matching descriptors generated by the Scale-Invariant Feature Transform (SIFT).

Algorithm:

1. **Cross-Correlation Matrix Computation:**

- The procedure initializes a matrix, to retain cross-correlation coefficients between descriptors from two sets, descriptors1 (from image 1) and descriptors2 (from image 2).
- It sequentially examines each descriptor in descriptors1 and descriptors2, calculating the cross-correlation coefficient between them.

- For every descriptor in descriptors1, we compute its mean and standard deviation.
- Subsequently, for each descriptor in descriptors2, we determine its mean.
- Through nested loops, we should compute the cross-correlation coefficient for every descriptor pair, utilizing means and standard deviations.

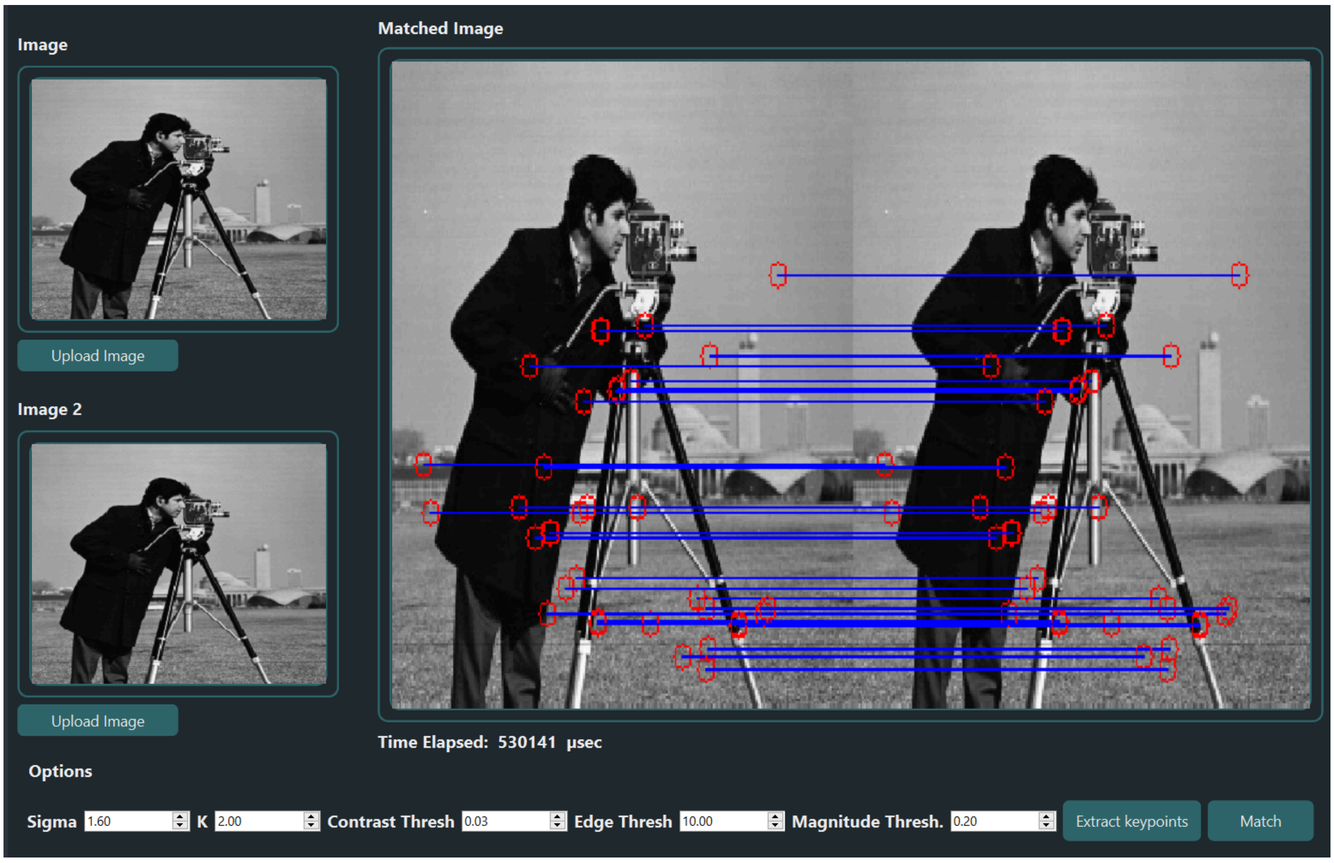
2. **Optimal Match Detection:**

- Following the cross-correlation coefficient computation, the algorithm iterates over each descriptor in descriptors1 once more.
- Within this iteration, it scans through each descriptor in descriptors2 to identify the descriptor with the highest correlation to the present descriptor in des1.
- For each descriptor in descriptors2, it identifies the index of the descriptor in des2 exhibiting the highest correlation coefficient.
- Then we retain the index of the optimal match and its corresponding correlation coefficient for each descriptor..

3. **Match Storage:**

- Ultimately, the function stores the indices of the optimal matches alongside their correlation coefficients in a vector named matches.
- The matches vector represents a Vec3d structure comprising three elements: the index of the descriptor in descriptors1, the index of its optimal match in descriptors2, and the correlation coefficient between them. This ultimately defines each descriptor with its corresponding descriptor from the other image using indexing and stating their relevance with correlation coefficients.

Results:



Time Elapsed: 530141 μsec

Options

Sigma 1.60 K 2.00 Contrast Thresh 0.03 Edge Thresh 10.00 Magnitude Thresh. 0.20