FEATURE EXTRACTION & MATCHING

Assignment 3

TEAM 1

- HABIBA FATAHALLA
- RAWAN MOHAMMED FEKRY
- SARA AYMAN EL-WATANY
- MARIAM WAEL

mariam meccawi

- Eng. Peter Emad
- Eng. Laila Abbaas

Introduction:

To implement the algorithms in the assignment from scratch we used C++ and Qt for GUI, to be able to obtain the same results as the algorithms in Open cv built in library.

To read and show the images we used open cv methods, and we also used it to test our output to see if it’s accurate or not compared to the built in algorithms.

Harris Corner Detection:

The Harris corner detector, also known as the Harris operator, is a popular algorithm for detecting corners or interest points.

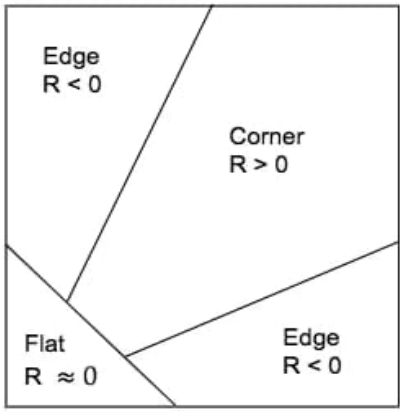
The algorithm uses the derivatives of the image to calculate the corner response function for each pixel, which measures the difference between the corner and the local neighborhood around the pixel. The algorithm is based on the assumption that corners have large variations in intensity in all directions.

The algorithm is implemented through 5 functions:

- **The nonMaxSuppression function:** This function is used to suppress non-maximum values in the corner response function. It iterates over each pixel and sets the pixel value to zero if it is not a local maximum. The function takes the response image and doesn't return any value.

- **The harrisCorner function:** This function defined to calculate the corner response function for the input grayscale image.
  The function first calculates the x and y derivatives of the image using the Sobel filter (Ix^2 & Iy^2) . Then, it calculates the products of these derivatives to get (IxIy).
  These products are filtered using a kernel, and then the corner response function is calculated for each pixel.
  Using the formula : R = det(M) - alpha*trace(M)^2.

  The value of R detects whether the point is a corner or not through these cases:



  The function returns the corner response function image.

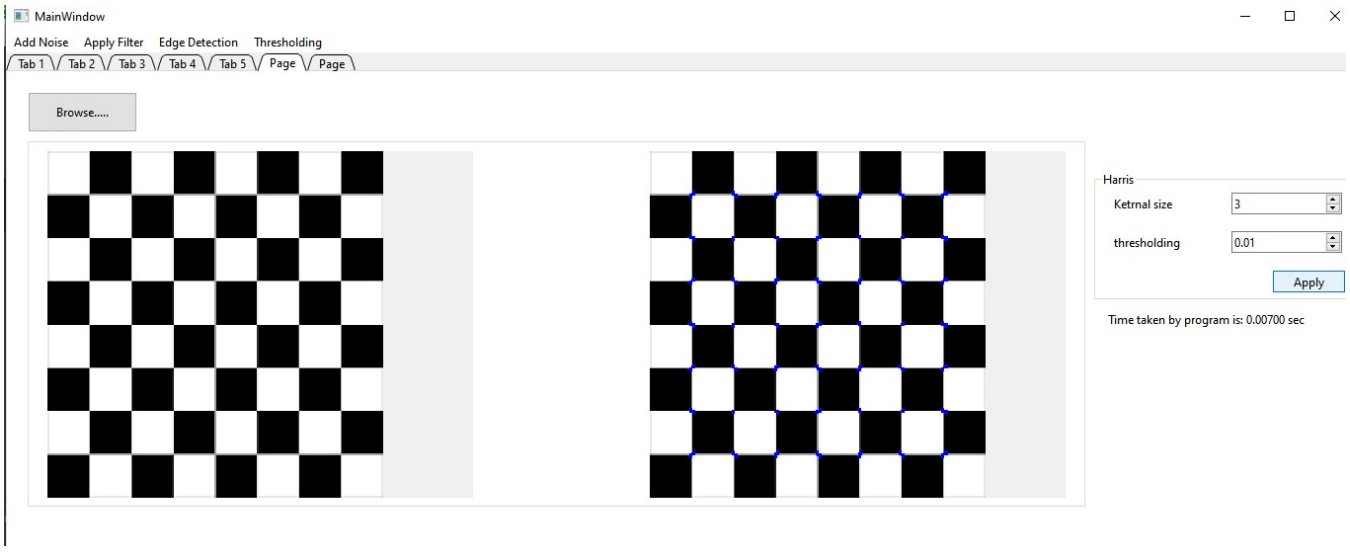
- **The setThreshold function:** This function is defined to set the threshold for the corner response function.
  It calculates the maximum value in the response function image and sets the threshold to be a fraction of that value.
  Then, the function sets all values below the threshold to zero.
  The threshold is important in selecting only the strongest corners in an image. The function doesn't return any value, it just sets the threshold value

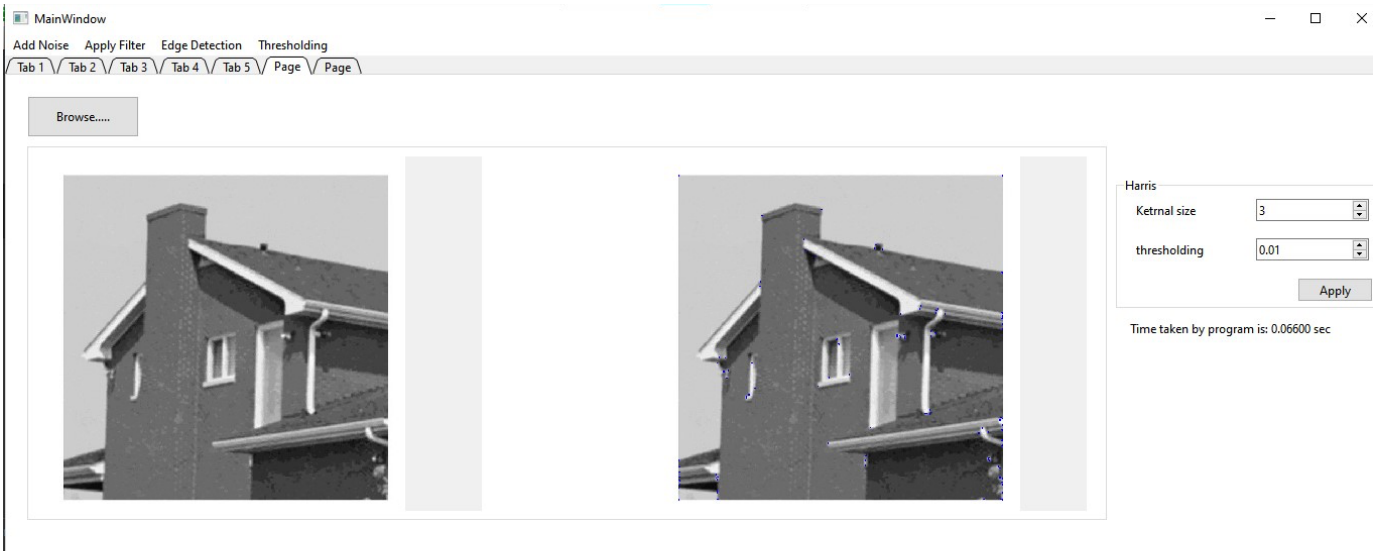
- **The getLocalMax function:** This function is defined to get the local maximum values in the corner response function image. The function first dilates the response function image to get the local maximum values.
  Then, it compares the original response function image to the dilated image to get the local maximum values.
  Finally, it subtracts 255 from the local maximum image and adds the dilated image to get the final local maximum values.

- **The drawPointsToImg function:** This function is defined to draw the detected corners on the input image. The function takes the input image and the local maximum values image as inputs. It then iterates over each pixel in the local maximum values image and draws a circle on the input image if the pixel value is nonzero.

Outputs:

Detecting Corners:

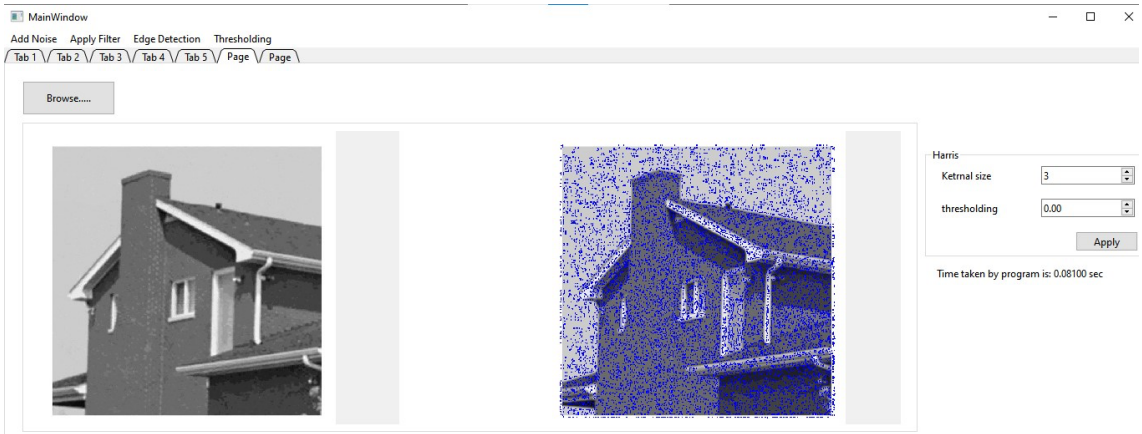




On changing Threshold values:

By reducing the threshold value. The detector will select more corners, but may also include more false positives. The detector may detect noise or other non-corner features as corners. This lead to decreasing the accuracy of detection.

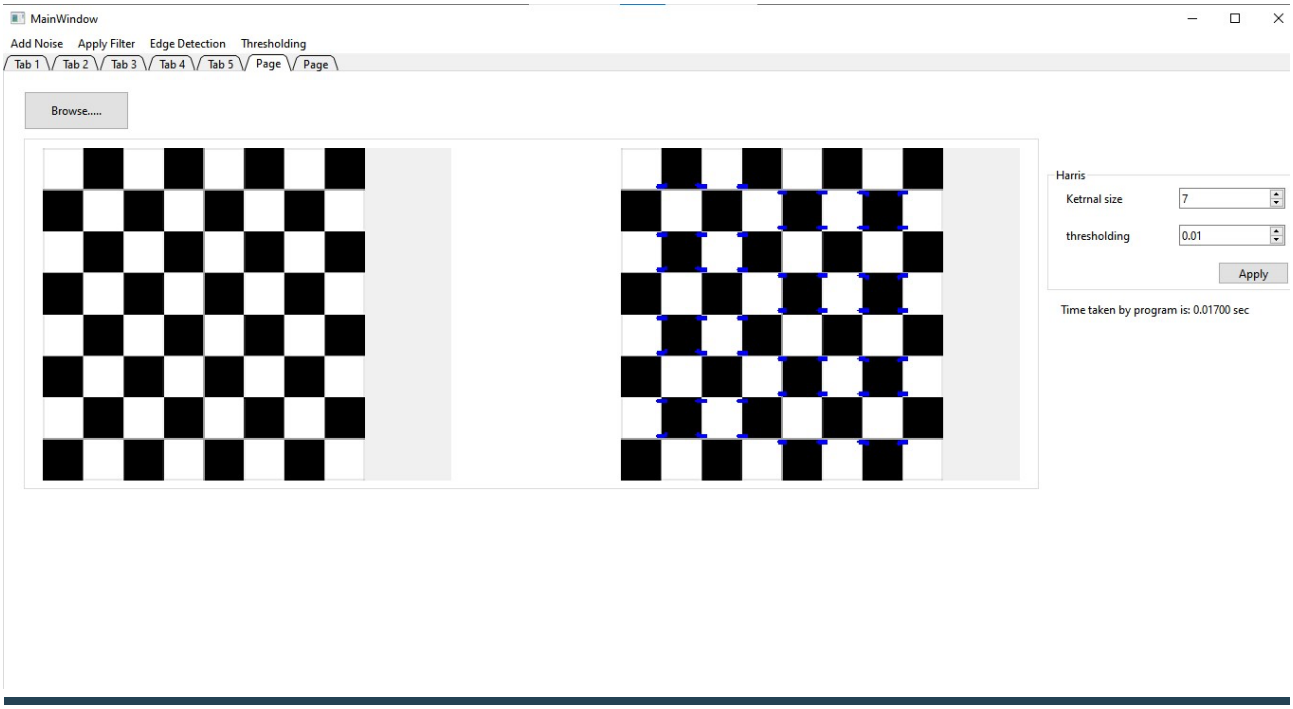
1. Set threshold value to zero



By increasing the value of threshold, the detector will select only the strongest corners, but may also miss some weaker corners. The detector may fail to detect corners that are important for the application. This cause missing important features.

2. Set threshold to its max value

Kernel value:



Scale Invariant Feature Transform (SIFT):

1. **Scale-space extrema detection:**
   is implemented by efficiently using a difference-of-Gaussian function to identify potential points of interest that remain invariant to changes in scale and orientation.

2. **Key point localization:**
   It is a detailed model is to determine the location and scale of a candidate, and select keypoints based on how stable they are.

3. **Orientation assignment:**
   One or more orientations are assigned to each key point location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

4. **Key point descriptor:**
   The selected scale in the region around each key point is used to measure the local image gradients, which are then transformed into a representation.

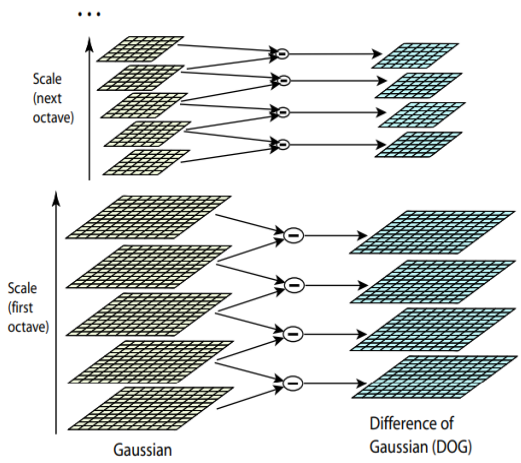
Our algorithm

1- Functions to build pyramids for Gaussian & Difference of gaussian (DOG) images, where each pyramid consists of octaves & within each octave number of layers(S+2) determined by the user (default value is 3 layers(S+2) as indicated in the paper presented by Lowe). (Refer to lines 27-142 in sift.cpp)

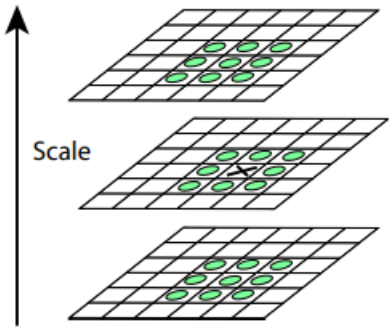
2- Functions to append layer (DOG) in each octave present in the pyramid.

3- Determine the key points present in each image that would describe the features present in it, this can be done by:

a. Looping through each pixel present in the given image and get the octaves, where each octave consists of certain scale of the image (each octave is half the dimensions of the previous one) and within this octave several layers(S+2) & each layer is a blurred image of the base image of that octave with different sigma value that can be computed by certain equation as indicated in the paper. (Refer to lines 151-254 in sift.cpp)



b. For each layer present in each octave get a window (4*4) & pass it all along the layer to get the minima or maxima pixels in that window with respect to the current layer & its neighbouring pixels from previous & next layer and store these values. (Refer to lines 151-254 in sift.cpp)



c. After storing the extrema points you can now start the elimination of unnecessary pixels, these can be determined by using certain contrast threshold value & determining whether they belong to an edge or not by using Hessian Matrix for that pixel and a set of mathematical equations present in the paper, where remove pixels belonging to edges & below certain Contrast Threshold value (0.03, as indicated by Lowe). (Refer to lines 265-385 in sift.cpp)

d. After storing these candidate key points, you need to draw window around each to get the orientations within the window & get their histogram with bins from 0 to 36 each bin with width of 10 (0-360 degree) and then use some threshold to get the dominant orientation & you can store it to be your pixel's orientation and then you continue for each key point present & store these orientations. (Refer to lines 395-512 in sift.cpp)

e. Get the feature descriptors for each image present in each layer within each octave, where this makes your algorithm invariant to rotation, where for each key point you draw a window with certain radius(4*4) then divide this window to four sub windows and get for each one of them the orientations present by using the histogram & then combine them you will end up having a descriptor of 128 values(4*4*8) & then you use certain threshold to eliminate some orientations to be able to normalize these values & then end up with a vector that is immune to rotations & contrast change (orientations won't be much affected by contrast change as much as magnitude). (Refer to lines 521-694 in sift.cpp)

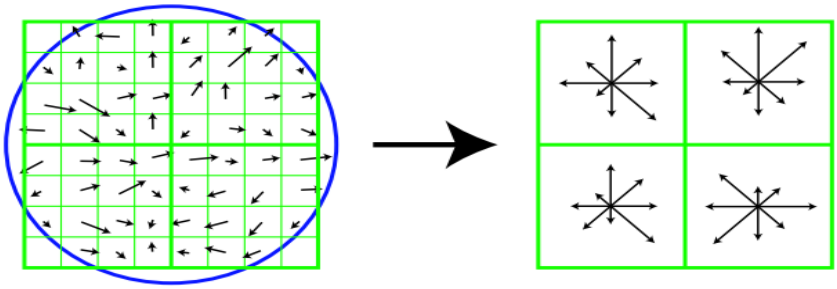


Figure 7: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas the experiments in this paper use 4x4 descriptors computed from a 16x16 sample array.

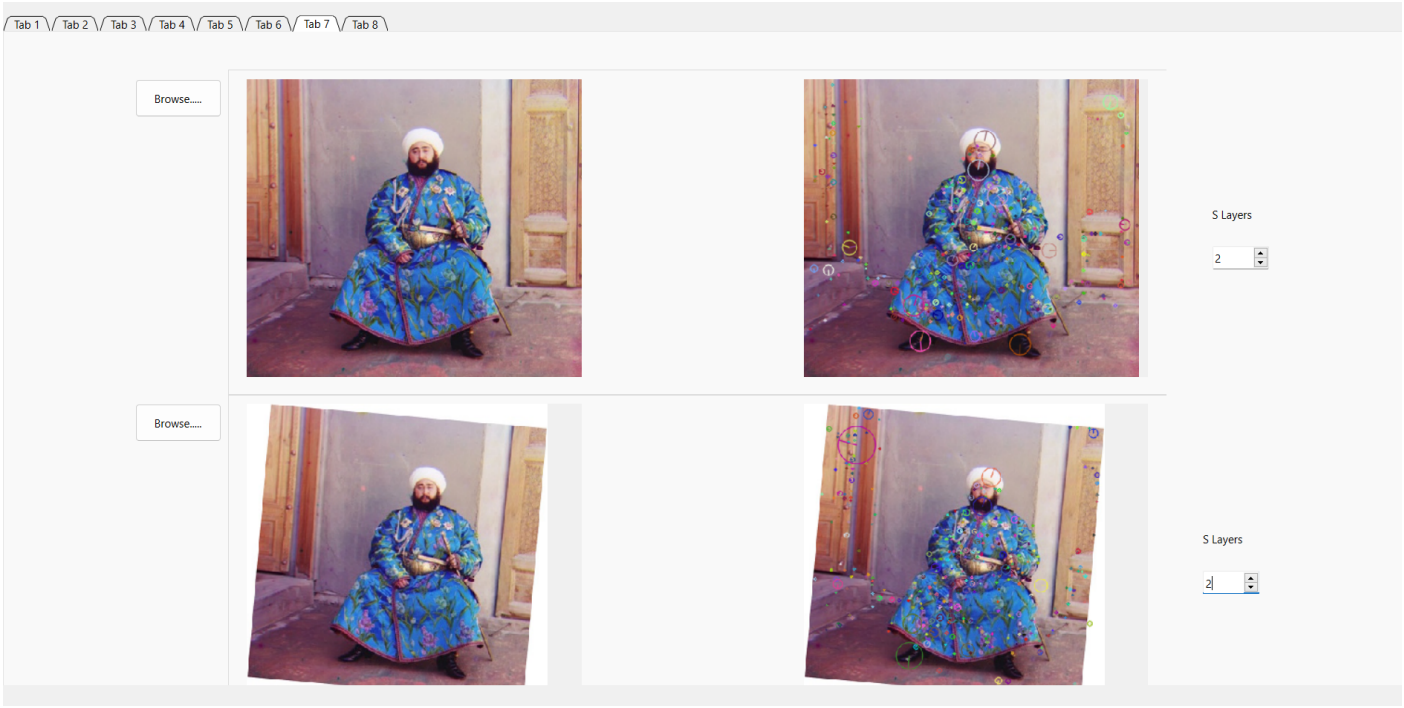
4- Function to Draw the detected key points on your image. (Refer to lines 700-724 in sift.cpp)

5- Function to Print the Computational time elapsed. (Refer to lines 732-739 in sift.cpp)

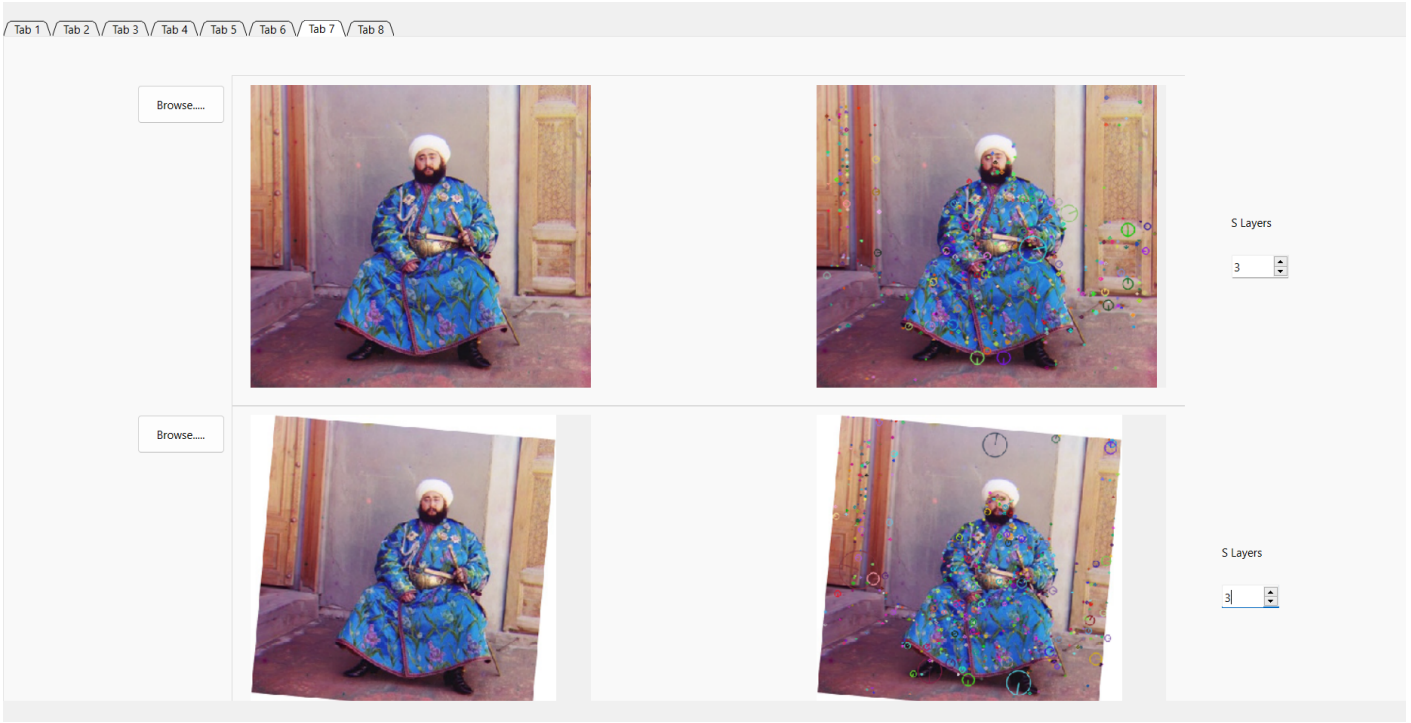
Output:

o Case 1:

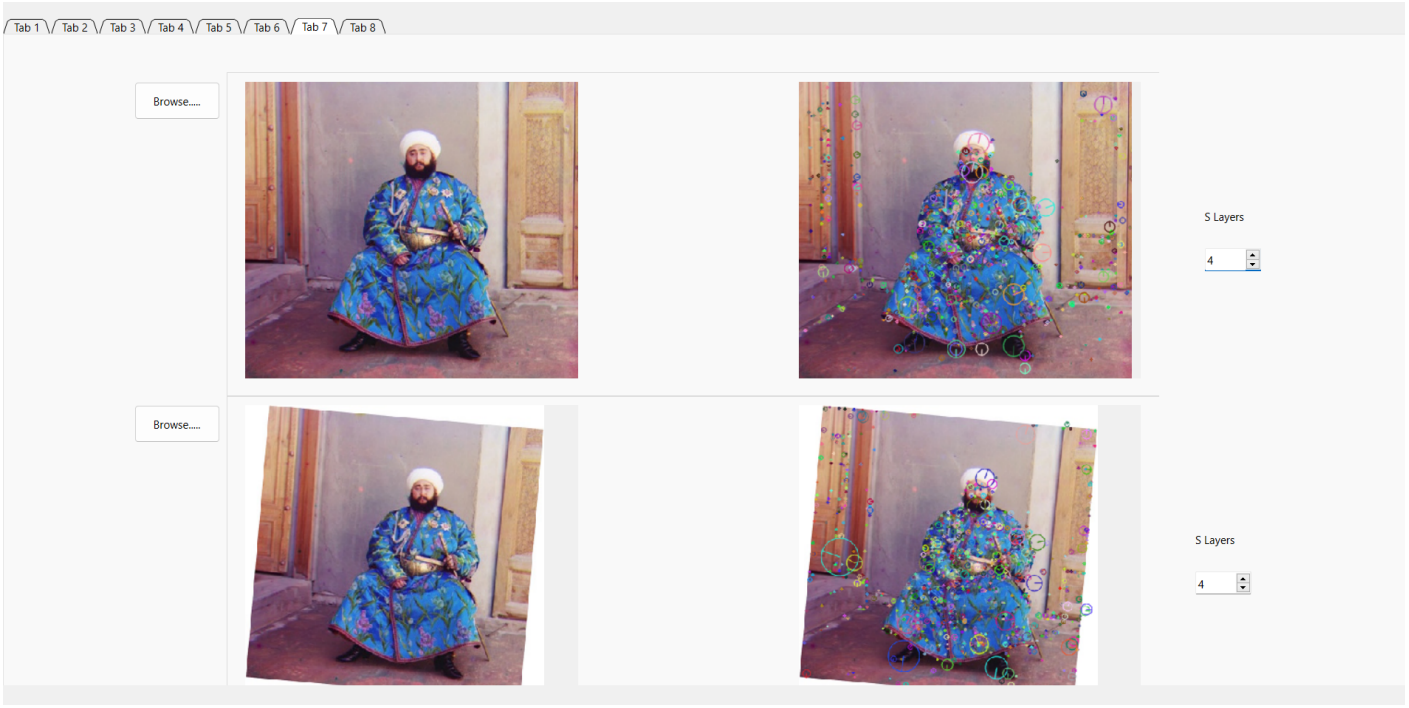
- S=2



- S=3

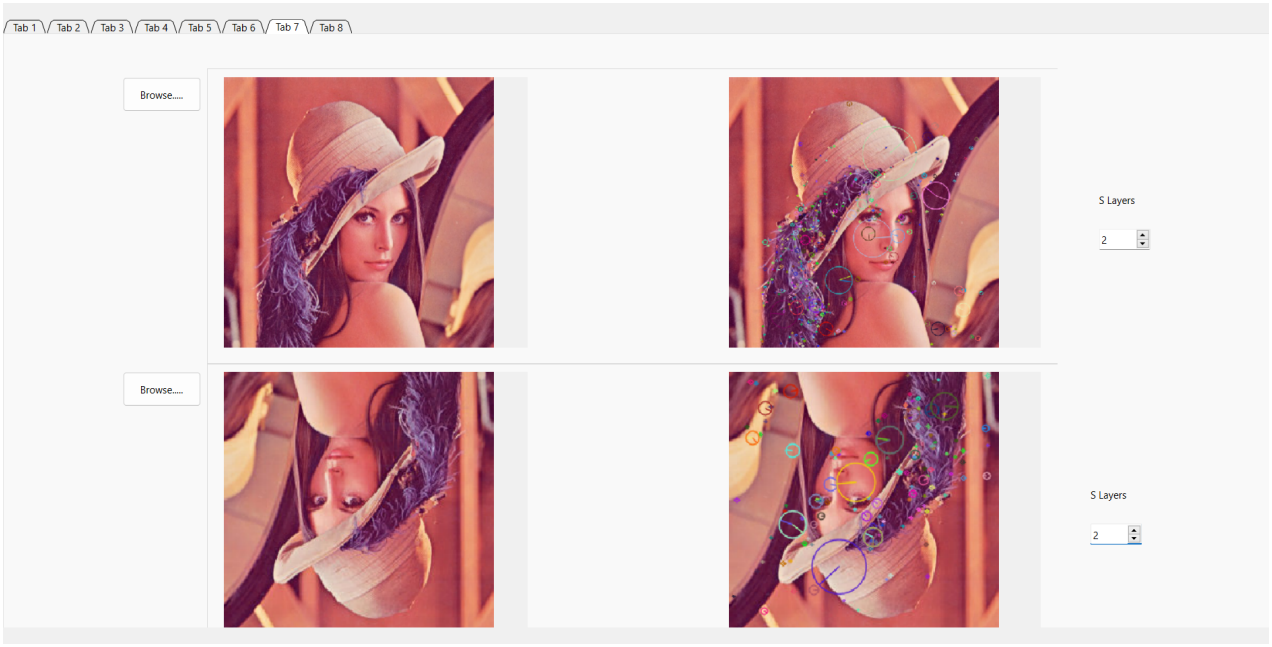


- S=4

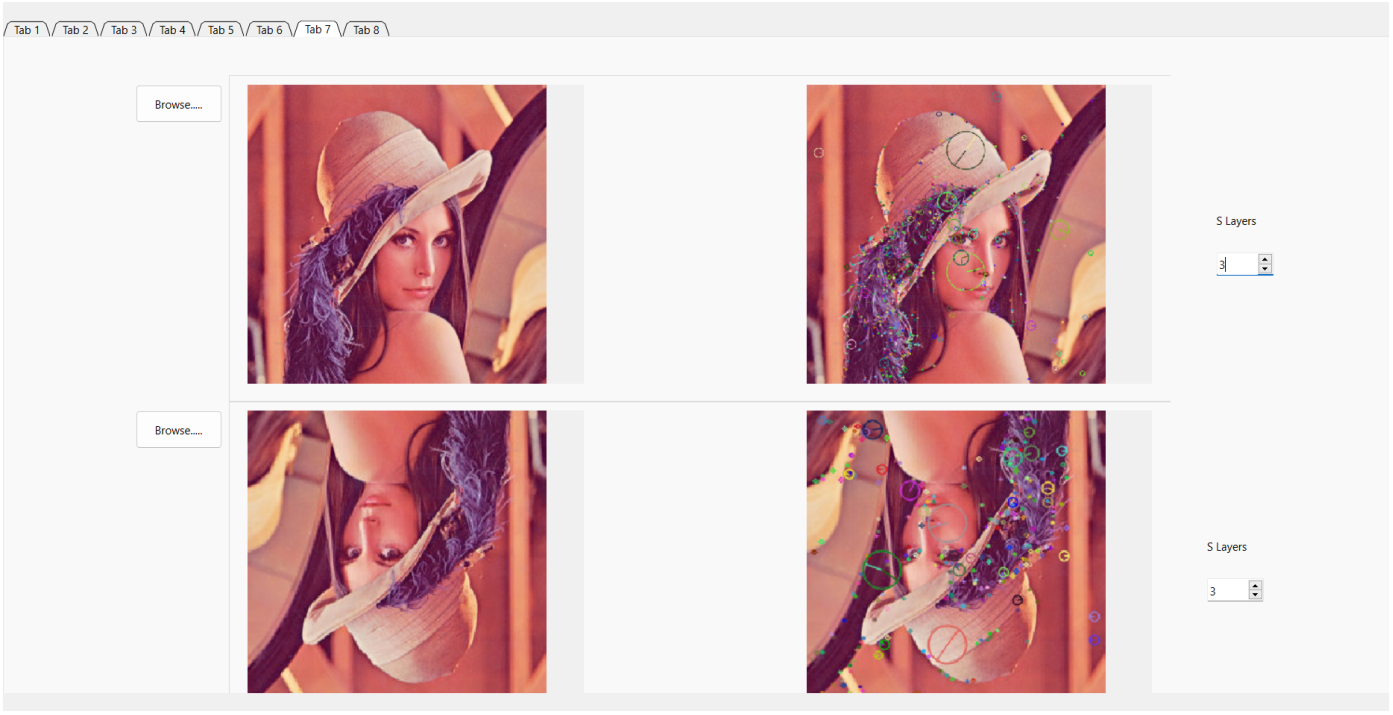

Tab 1
Tab 2
Tab 3
Tab 4
Tab 5
Tab 6
Tab 7
Tab 8
Browse.....
S Layers
4
Browse.....
S Layers
4

o Case 2:

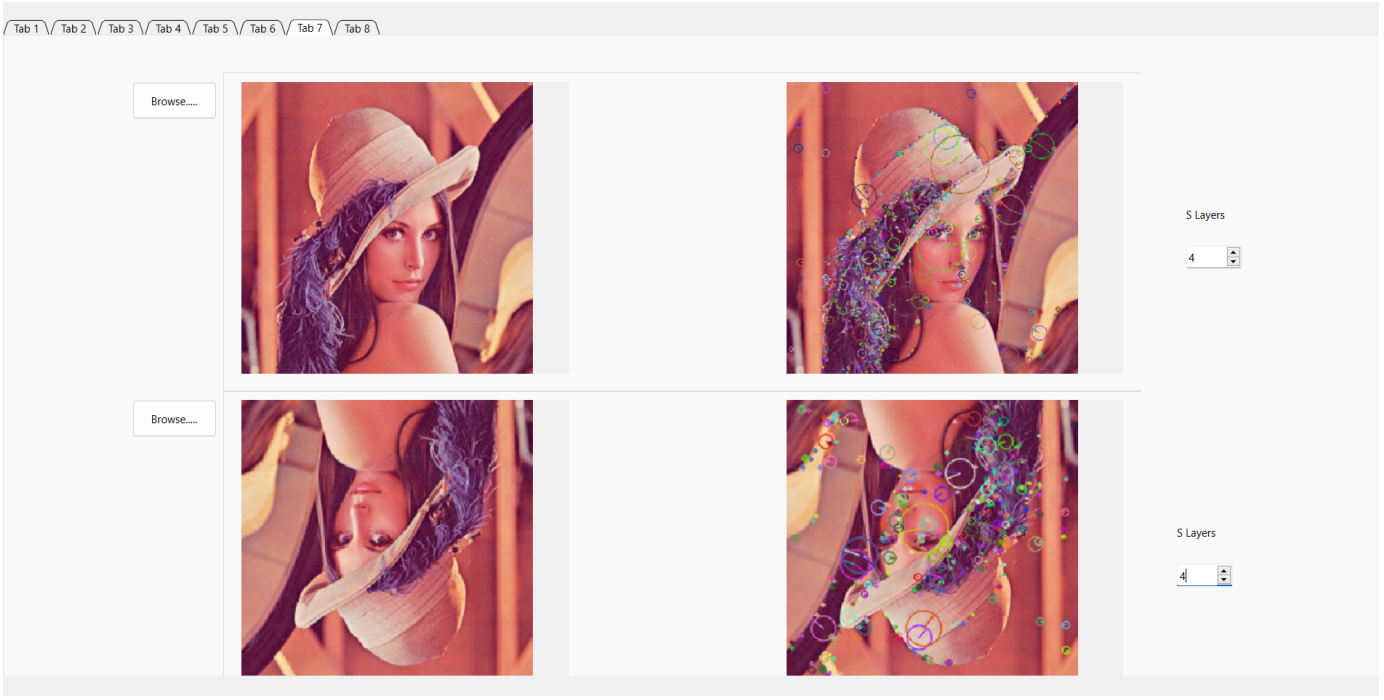
- S=2


Tab 1
Tab 2
Tab 3
Tab 4
Tab 5
Tab 6
Tab 7
Tab 8
Browse.....
S Layers
2
Browse.....
S Layers
2

- S=3


Tab 1
Tab 2
Tab 3
Tab 4
Tab 5
Tab 6
Tab 7
Tab 8
Browse.....
S Layers
3
Browse.....
S Layers
3

- S=4


Tab 1
Tab 2
Tab 3
Tab 4
Tab 5
Tab 6
Tab 7
Tab 8
Browse.....
S Layers
4
Browse.....
S Layers
4

- **Computational Time:**

- Estimated time at s= 3

```
Time taken in microseconds for first image: 389784
Time taken in microseconds for second image: 119795
```

- Estimated time at s=4

```
Time taken in microseconds for first image: 591718
Time taken in microseconds for second image: 179677
```

By increasing the value of S, the computational time increases.
So, the better choice is S=3, because by increasing the value of S the noise interferes with the selected features.

Feature Matching Algorithms:

- ## SSD Algorithm:

In feature matching, the Sum of Squared Differences (SSD) algorithm is a widely used method, especially in computer vision and image processing applications.

Finding correspondences between features in various photos or frames is the aim of feature matching. The sum of the squared differences between the respective pixel intensities or feature values of two feature descriptors is calculated as part of the SSD algorithm to compare the similarity between them.

The total of the squared differences between each matching feature value in A and B would be calculated by the SSD method, and is technically denoted as:

$$SSD = \sum(v1 - v2)2$$

The feature descriptor (result from feature extraction from SIFT algorithm) in one image with the shortest SSD can be compared to the feature descriptor in the second image using the SSD algorithm to match features. This is so that the smallest SSD will be found among feature descriptors that are most similar.

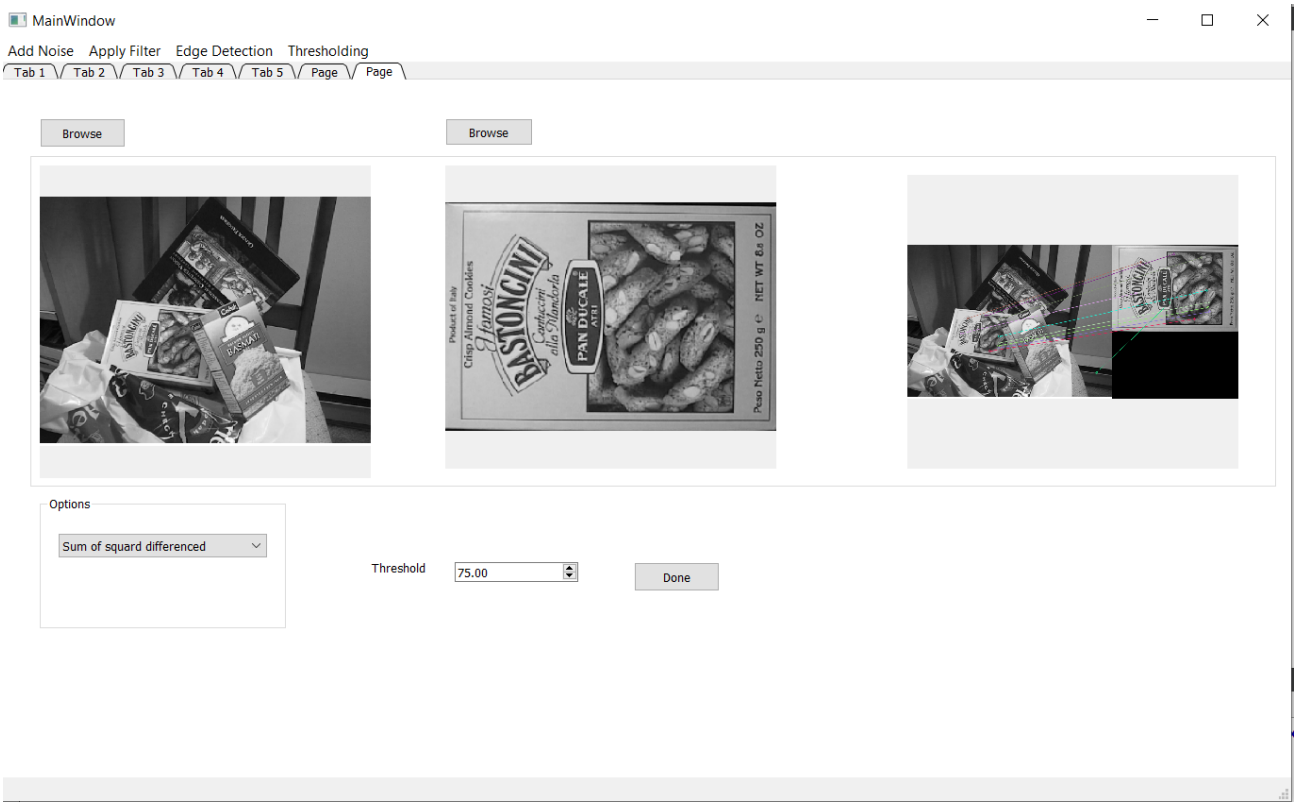
Although it can be susceptible to variations in lighting, rotation, and scale, the SSD method is not always the most reliable or accurate feature matching algorithm. In some circumstances, other algorithms, including the normalized cross-correlation (NCC) algorithm, may be better suitable.

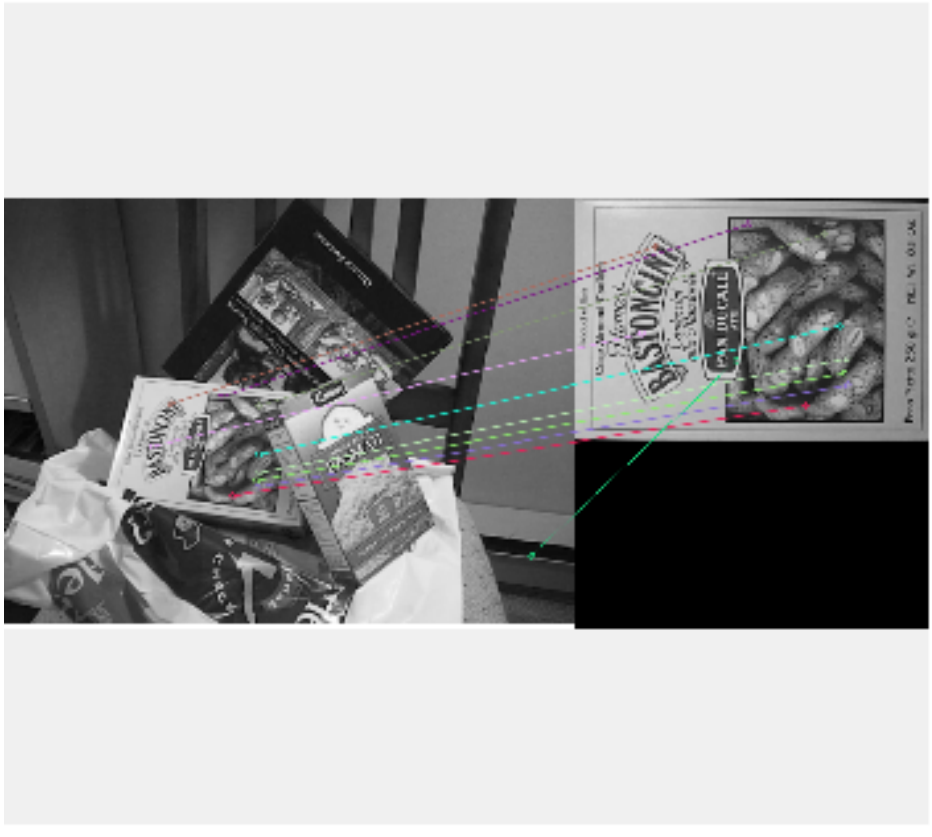
The SSD algorithm's threshold is used to determine the least level of similarity between two features that is considered to be acceptable. A match is only regarded as a good match if its SSD value falls below a particular threshold. The algorithm keeps looking for a better match if the SSD value for a particular match is higher than the threshold.

Based on the application and the features of the photos being matched, the threshold value can be selected. Sometimes while fewer matches will be discovered with a higher threshold, they will be of high quality. More matches will

be discovered at a lower threshold, but some of those matches might not be accurate or dependable.

- **Our Output**





ComputationTime:

```
Time taken by the algorithm: 12.2512 seconds
```

• NCC Algorithm:

The NCC stands for "Normalized Cross-Correlation".

This algorithm works by comparing two images, pixel by pixel, using a mathematical formula called cross-correlation. Cross-correlation measures how similar two images are.

The formula for NCC is :

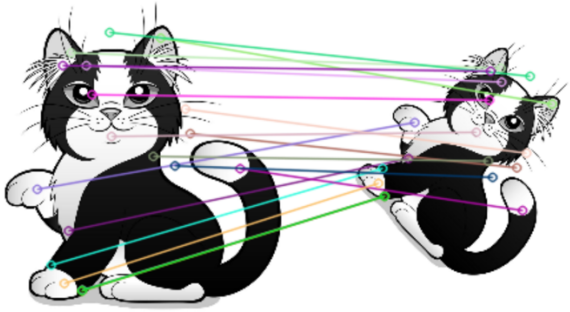
$$NCC(x,y) = (1/N) * \sum[I(x,y) - \mu_I] * [T(x,y) - \mu_T] / (\sigma_I * \sigma_T)$$

Where N is the number of pixels in the image, I(x,y) & T(x,y) are the intensities of pixel (x,y) in the first and second images respectively, μ_I and μ_T are the mean intensities of the two images, and σ_I and σ_T are the standard deviations of the two images.

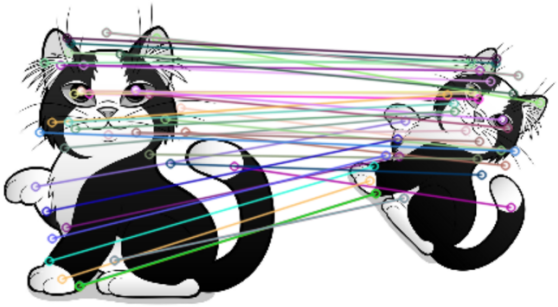
The algorithm's steps:

- Extract feature descriptors from both images using SIFT feature extraction method.
- Extract matching features by computing the NCC value for every pixel in both descriptors.
- Get the best NCC value for each row in second descriptor compared to each row in first descriptor.
- Storing all matches together in a vector.
- Sort the vector of matches descending to keep the best matches at the beginning of the vector.
- Choosing different number of features to display.

20 features :



40 features:



Our Project output using 30 features: