Computer Vision
Task 3

Team 06

Team Members:

Name	Section	BN
Romaisaa Shrief	1	36
Kamel Mohamed	2	11
Youssef Shaban Mohamed	2	56

Part 1- Harris operator and λ-

Extraction Steps for both:

1. Compute image gradients over small region:

Differentiate to x, y depending on kernel size(given from user 1-3-5) not pixel by pixel using sobel.

2. Compute the elements of the Harris matrix:

Calculator Ixx,Ixy,Iyy by multiplication from step 1

3. Compute the sum of the elements in matrix:

Using our implemented box filter and implemented convolution.

4. Compute the determinant and trace of the Harris matrix:

det= Ixx*Iyy-Ixy*Ixy

trace = Ixx+Iyy

5. Function check if extraction by λ- or not:

If λ-use eq: R=determinant-0.04*trace_squared

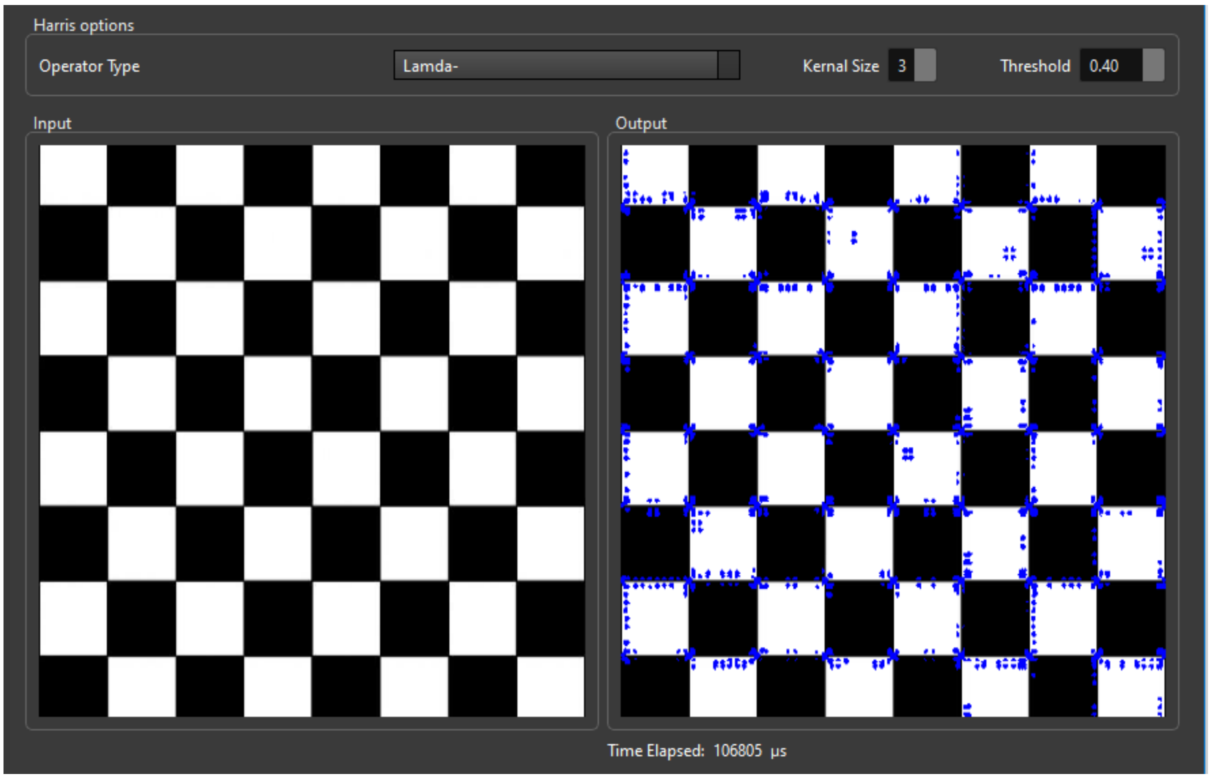
If harris use eq: R=determinant /trace

6. Plot corners with plotCorners functions:

Loop over response and plot points where R>threshold (given from user)

Results for image of size 600x570:

1. λ- computation time about 10,000 Microseconds

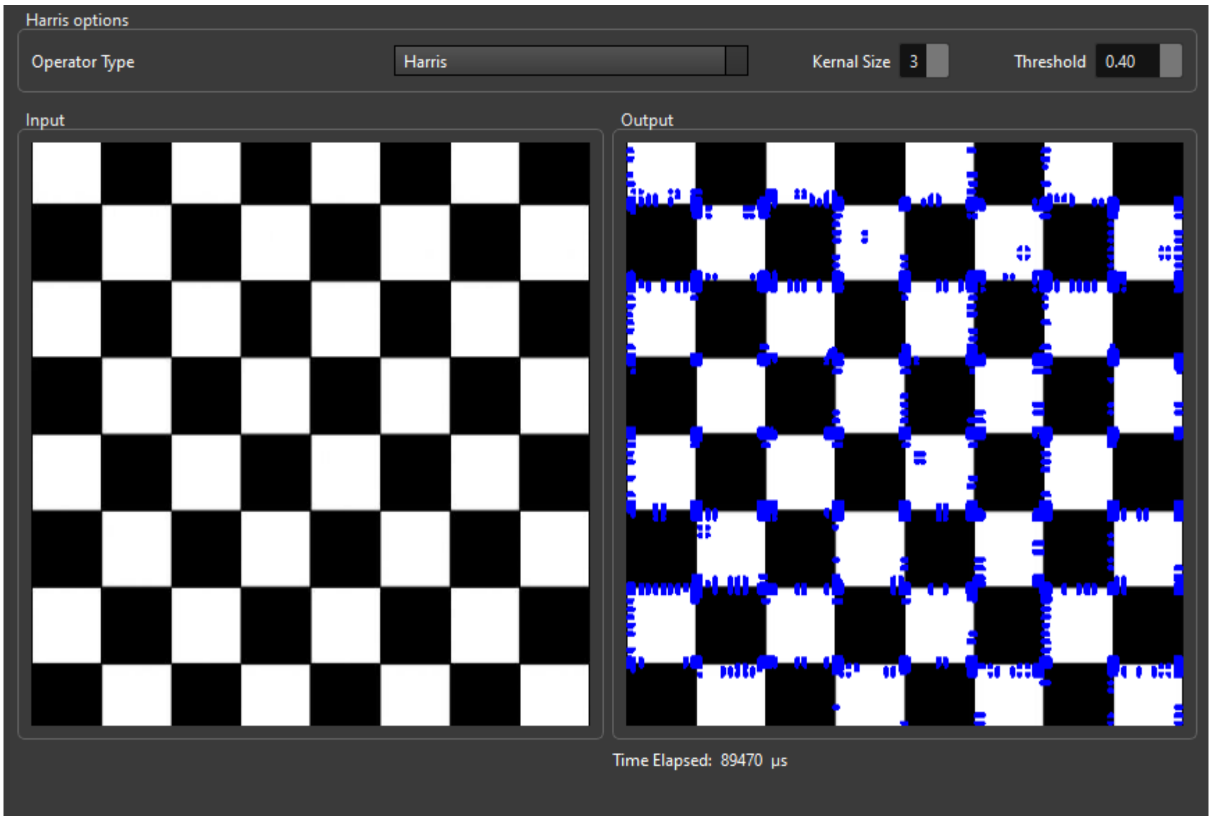


Harris options

Operator Type: Lamda-   Kernal Size: 3   Threshold: 0.40

Time Elapsed: 106805 μs

2. Harris computation time about 10,000 Microseconds



Harris options

Operator Type: Harris   Kernal Size: 3   Threshold: 0.40

Time Elapsed: 89470 μs

Part 2- SIFT feature descriptors

Algorithm steps:

A. Computing Keypoints:

1. Scale-space extrema detection and DoG:

Kernel for scale space computed by Guassian function

So, scale spaced image is:

L(x, y,) = G(x, y,) ∗ I(x, y) , where G gaussian function giving sigma from user

DoG: difference of gaussian by k factor given from user

By get_scale_space, *get_DoG functions*.

2. Local Maxima detection:

Check for 8 neighbours pixels of the image the max in the DoG matrix

By *get_keypoint*s function

3. **Contrast Improvement:**

Remove low contrast keypoints depending on threshold from user (contrast_threshold) using r*emove_low_contrast* function

4. Edge Improvement by hessian matrix

Same as one used in harris operator one.

But we compare edge threshold(given from user)

By the formula of:

trace^2/det < (threshold+1)^2/threshold

This done by r*emove_edges* function

5. *Draw_keyoints* on image function

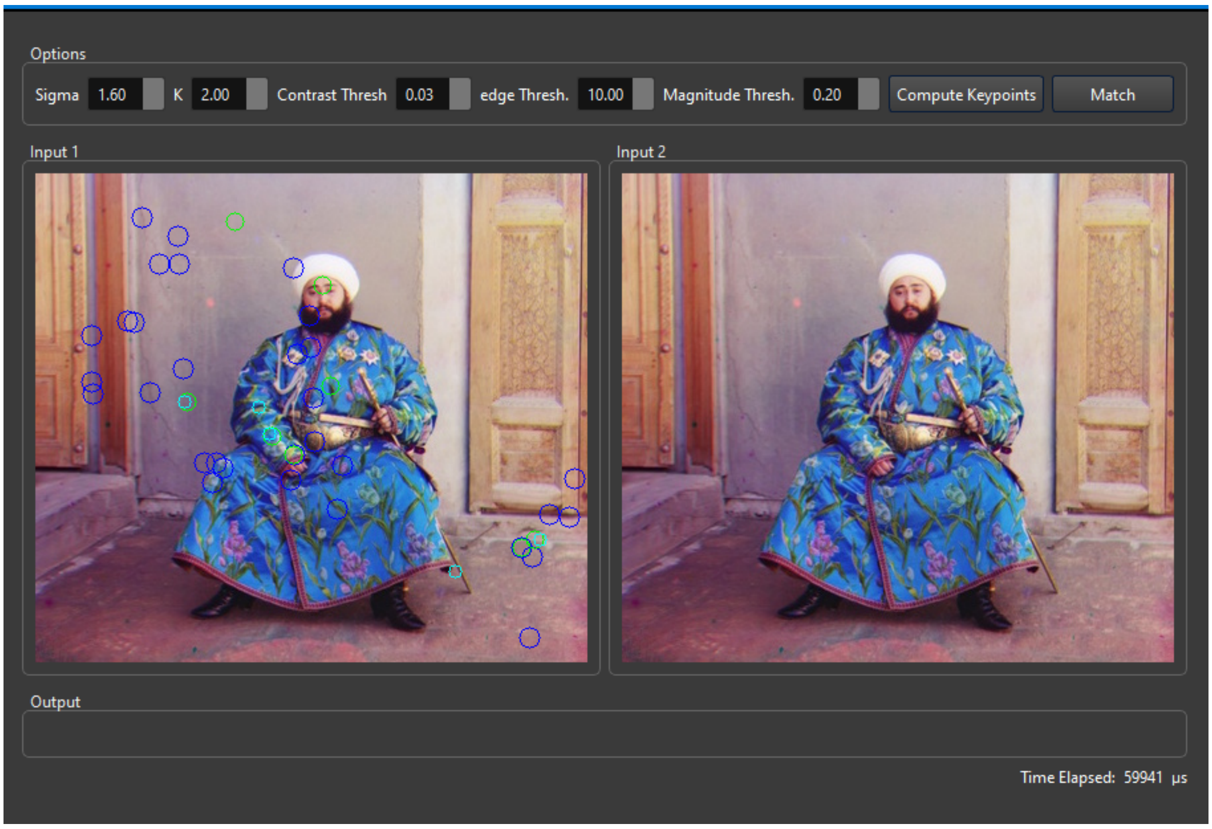
All these 5 steps wrapped in *sift_keypoints* function

Note: implemented e*xtract_keypoints* functions is just for data type conversions

Results for keypoints generation:

Image size-> 415x370

Computation Time about 60,000 Microseconds



Options

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

Time Elapsed: 59941 μs

B. Computing Descriptors:

1. *Get_magnitude, get_orientation*:

Calculate mag and angle of gradient in the scale space
Helper functions used with get_orientation-> *get_angle* from dx and dy

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

2.*get_keypoints_orientations*:

We compute an orientation histogram formed within neighbours around the keypoint.
It has 36 bins covering the 360 degree range of orientations.
Each sample added to the histogram is weighted by its gradient magnitude by a Gaussian-weighted circular window with a that is 1.5 times that of the scale of the keypoint**.**

After calculating histogram, we take the peak of the keypoint-> dominant direction.

Helper function used :

get_gaussian_weight-> return weights from a gaussian distribution

get_gaussian_circle-> convolve circular filter on th image

get_pos_histogram1-> **r**eturn bin in the histogram

3. ***get_descriptors*:**

Final descriptors combining all values of orientation and magnitude

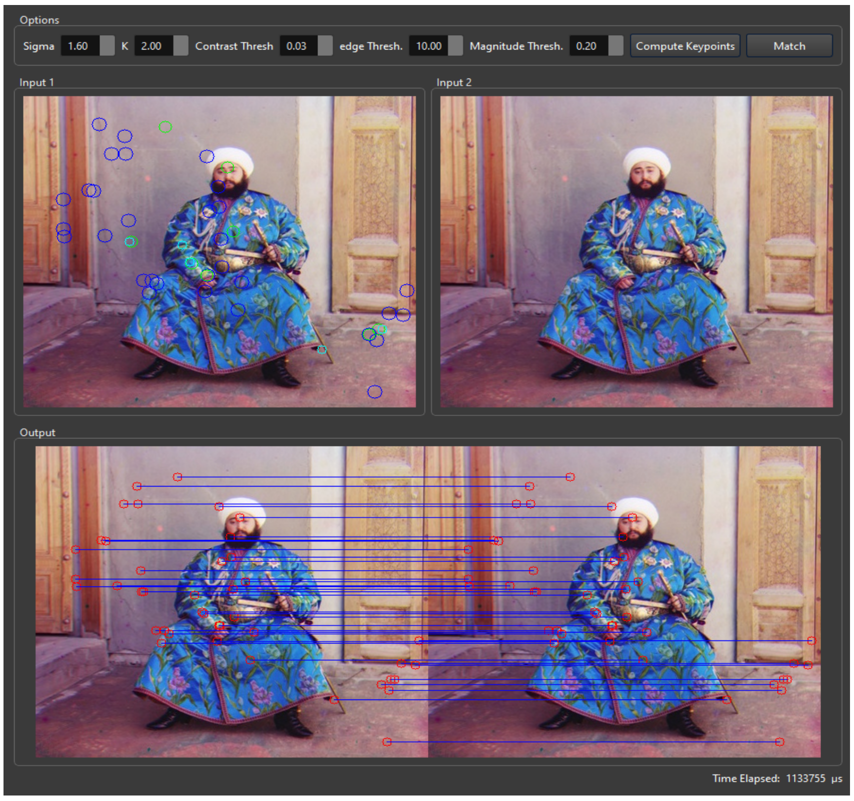
4. Large Magnitudes improvement:

Normalize descriptor then neglect large magnitudes based on magnitude threshold given from user Normalization by->*normalize_vector* function Filter magnitudes by-> *reduce_large_magnitudes* Two function wrapped in-> *get_luminosity_invariance*

5. *SIFT_Desrciptors->* *wrapper for all functions of descriptor computing*

6. Using Correlation Match and, get key points of correlation, then connect points by each other by draw_lines function in paage8.cpp (on_pushButtonclicked)

Results for SIFT feature Matching (not built-in):
Time-> 1133755 (Time for extracting and matching from both images)



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

Time Elapsed: 1133755 µs

Part 3- Image Matching

1. Squared Sum of Differences (SSD):

Algorithm steps:

A- cut the intersection between template and image.

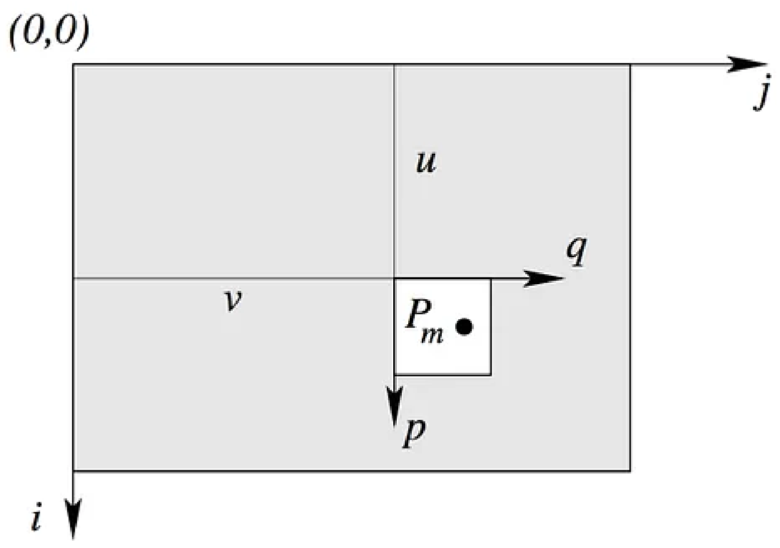
B- subtract template from cutted image.

C- square the difference and then sum it to a single number.

D- shift template with one pixel and repeat steps A-C.

E- after calculating all numbers, we get minimum value and takes its index.

This indexes is [u,v], that we need to calculate.

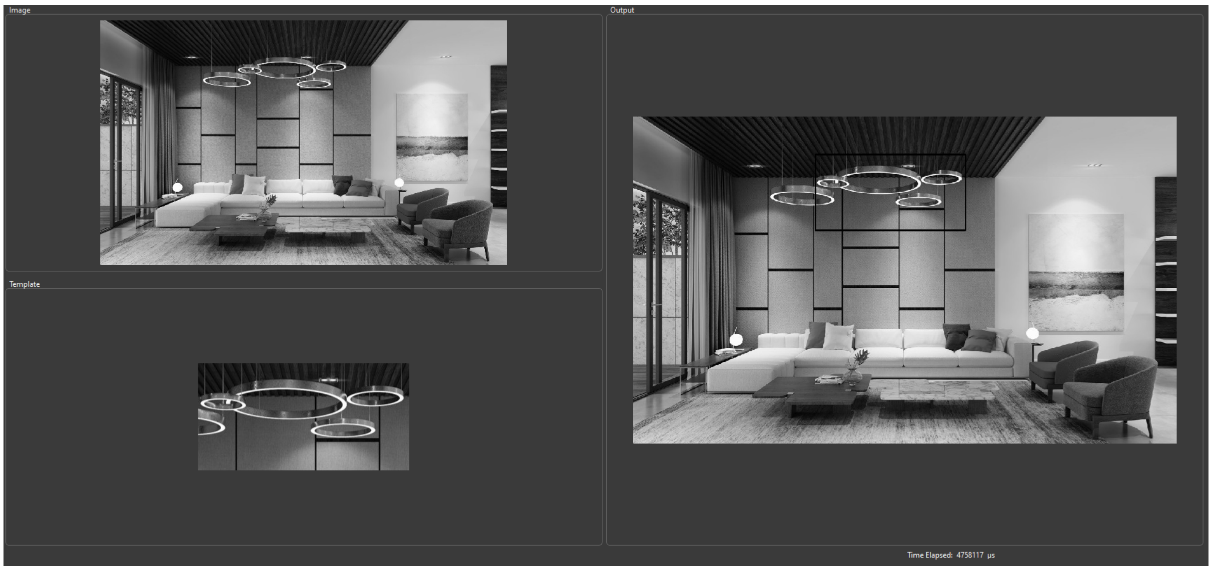


The problems with this algorithm is:

1- can’t work probably with change in image brightness or contrast.

2- can deal with rotations of template.

And this is an output from our program:



Time Elapsed: 4758117 µs

2. Normalized Cross Correlation:

Algorithm steps:

A- cut the intersection between template and image.

B- calculate variance of template image.

C- calculate variance of cutted image.

D- calculate variance between these two images.

$$\rho_{12}(u,v) = \frac{\sigma_{g_1 g_2}(u,v)}{\sigma_{g_1}(u,v)\sigma_{g_2}}$$

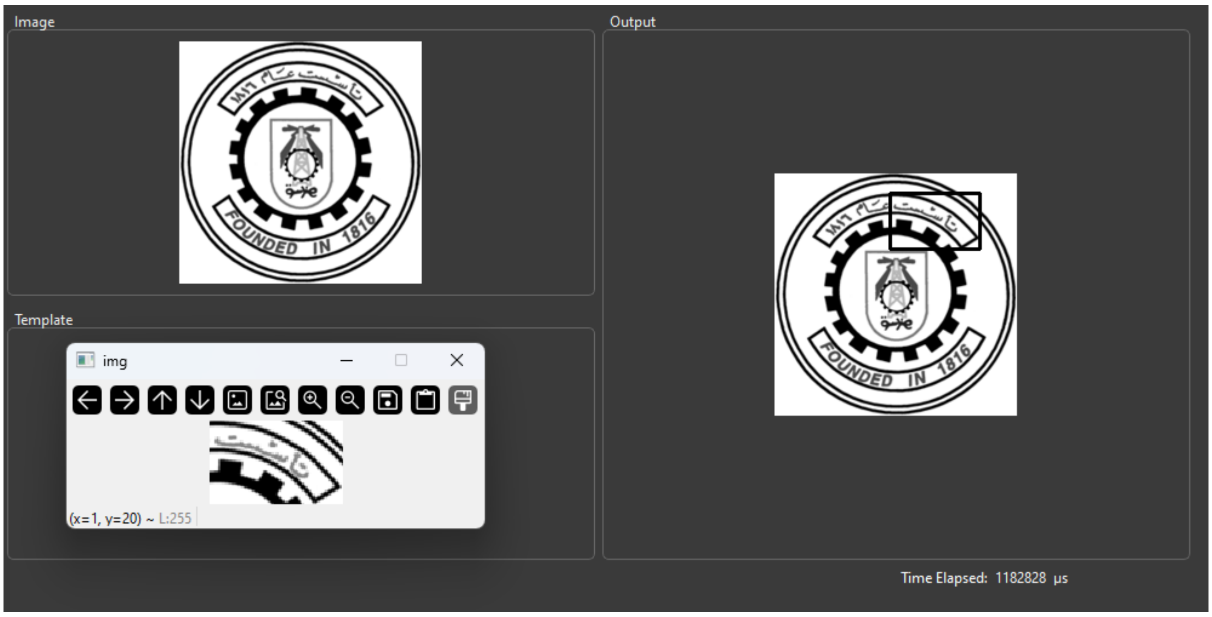
E- repeat for each u,v in this equation, and then get maximum value.

The problems with this algorithm is:

1- very expensive to calculate variance each time, but can be faster using pyramids to calculate.

2- can't deal with rotations, but can be improved.

And this is an output from our program:



Time Elapsed: 1182828 µs