Assignment #4 – Edges and Feature Descriptors

Faculty of Engineering Science
Dept. of Electrical and Computer Engineering
Introduction to Digital Image Processing, 361-1-4751

Maor Assayag 318550746

Refahel Shetrit 204654891

1 Edge Detection

In this section we will play with some famous edge detectors.

1.1 Reading the Image

Read the image named *tire.tif*. Normalize the image between [0; 1]. This image is built-in into MATLAB, you can simply read it by writing the line *imread('tire.tif ')*. For those of you who will not be able to do it, the image is added to the exercise file.

Figure 1.1.1

Explanation: We simply read the image and divided it by 255 (using double) to transform all the data to values in range of [0, 1].

1.2 Roberts Edge Detector

The Roberts edge detector is one of the first edge detectors invented. Its derivative kernels are defined as :

$$G_{Rx} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & |-1 \end{pmatrix}, G_{Ry} = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

1.2.1 Write your own function named *dip_roberts_edge(img,thresh)* that will apply the Roberts edge detector on *'img'* and output the edge image with the same size as the input image. The *'thresh'* parameter will determine the gradient magnitude cutoff threshold. Note: You don't have to deal with the image boundaries, you can use the *conv2()* function with the parameter *'same'.*

Explanation: [1]

The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. It thus highlights regions of high spatial frequency which often correspond to edges. In its most common usage, the input to the operator is a grayscale image, as is the output. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

The operator consists of a pair of 2×2 convolution kernels :

+1	0
0	-1

Gx

0	+1
-1	0

Gy

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy). These can then be combined to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$\nabla I(x,y) = G(x,y) = \sqrt{G_x^2 + G_y^2}$.

We used the function conv2() and this simple mathematical equation .

[1] Image processing learning resource HIPR - http://homepages.inf.ed.ac.uk/rbf/HIPRUSERS

1.2.2 Display 3 edge images generated using *dip_roberts_edge(img,thresh)* with 3 different thresholds.
Which of the edge images do you think represents the edges in the original image in the best manner? Why?

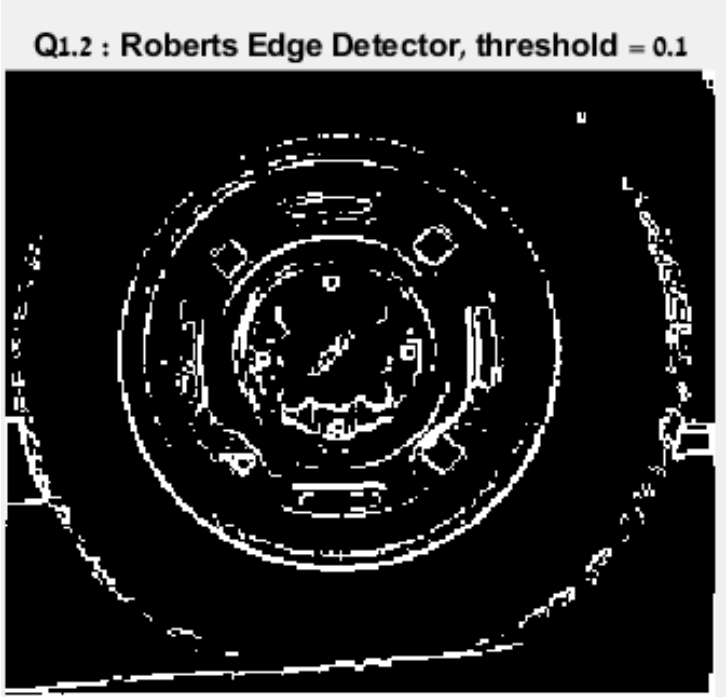


Figure 1.2.2.1

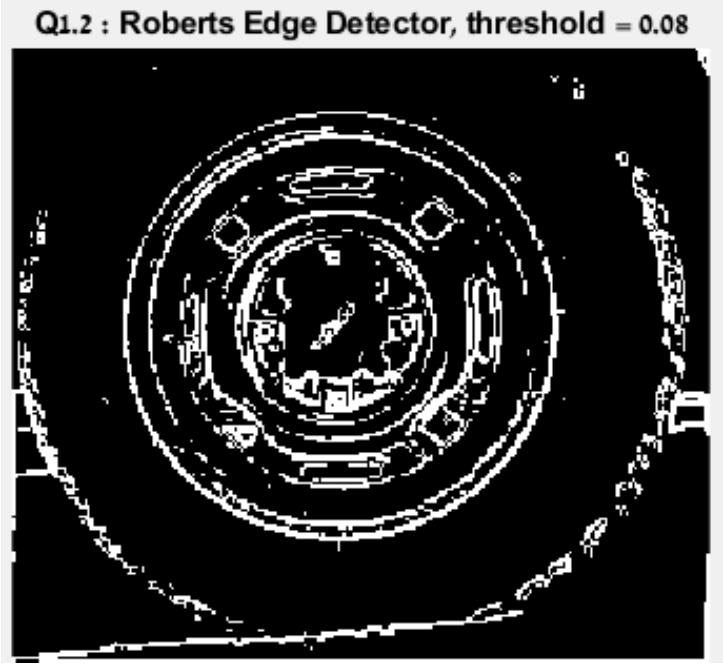


Figure 1.2.2.2

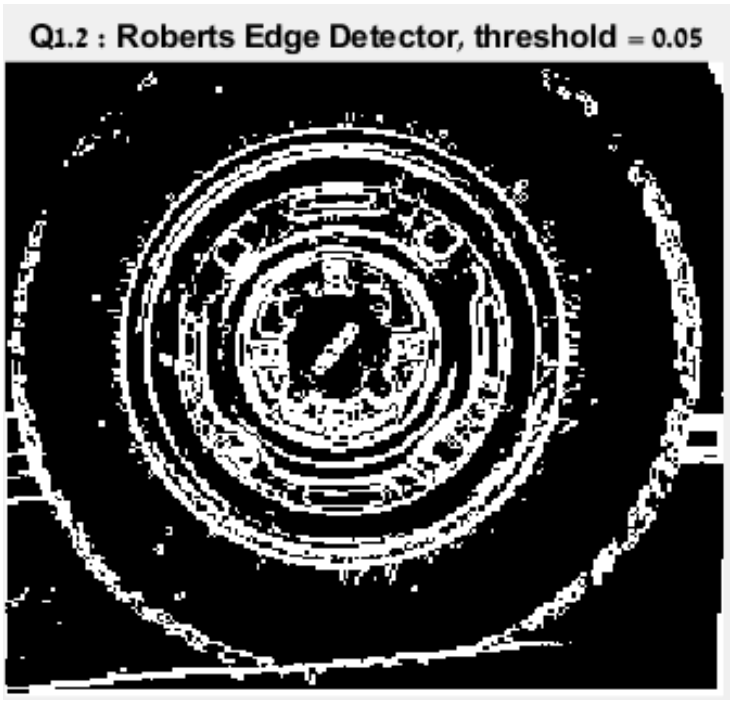


Figure 1.2.2.3

Explanation:
We choose 3 thresholds : 0.1, 0.08, 0.05. In our opinion 0.08 results a better representation of the image edges. This threshold results us an image which we can easily define the edges from it. Also, the other thresholds results have more noise (filled white areas – e.g. 0.05) and less edge details (0.1).

1.3 Prewitt Edge Detector

The Prewitt edge detector was developed to solve some of the problems of the Roberts detector. Its derivative kernels are defined as: GPx = 160

$$G_{Px} = \frac{1}{6}\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, G_{Py} = \frac{1}{6}\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

1.3.1 Write your own function named *dip_prewitt_edge(img,thresh)* that will apply the Prewitt edge detector on *'img'* and output the edge image with the same size as the input image. The 'thresh' parameter will determine the gradient magnitude cutoff threshold. **Note:** You don't have to deal with the image boundaries, you can use the conv2() function with the parameter 'same'.

Explanation:

Prewitt operator is used for edge detection in an image. It detects Horizontal edges and Vertical Edges, which are calculated by using difference between corresponding pixel intensities of the image.

The gradient of a two-variable function (here the image intensity function) is at each image point a 2D vector with the components given by the derivatives in the horizontal and vertical directions.

At each image point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction. The Algorithm follows the Prewitt operators :

$$p_1 = \frac{1}{6}\left[(a_{1,-1} + a_{1,0} + a_{1,1}) - (a_{-1,-1} + a_{-1,0} + a_{-1,1})\right]$$

$$p_2 = \frac{1}{6}\left[(a_{-1,1} + a_{0,1} + a_{1,1}) - (a_{-1,-1} + a_{0,-1} + a_{1,-1})\right]$$

$$E(i,j) = \begin{cases} 1 & \|(p_1, p_2)\| = \sqrt{(I * \Delta_1)^2 + (I * \Delta_2)^2} > \tau \\ 0 & otherwise \end{cases}$$

$$p_1 = I * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad p_2 = I * \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Δ_1 Δ_2

This implies that the result of the Prewitt operator at an image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.

1.3.2 Display 2 edge images generated using *dip_prewitt_edge(img,thresh)* with 2 different thresholds.

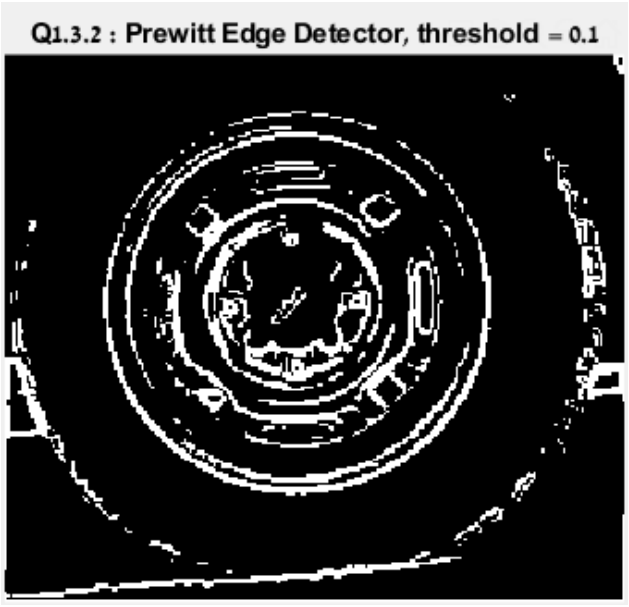


Figure 1.3.2.1

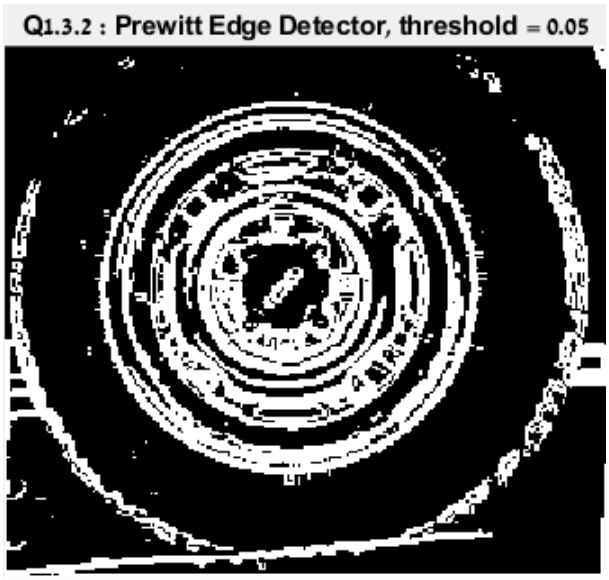


Figure 1.3.2.2

Explanation: We can see more details that implies edges, but the lower the threshold gets the less accurate the results gets.

1.3.3 Add 'gaussian' noise to the image using imnoise().

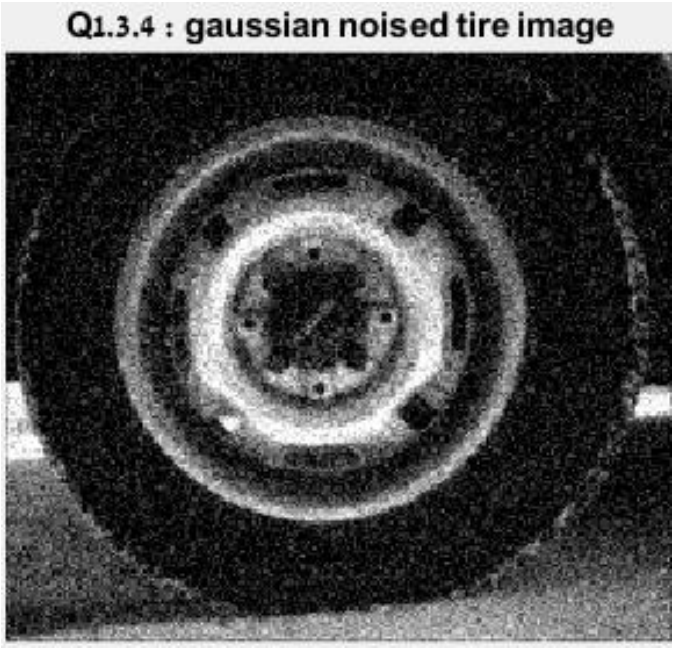


Figure 1.3.3.1

Explanation: We used the built-in MATLAB function *'imnoise()'*.

1.3.4 Apply both *dip_roberts_edge(img,thresh)* and *dip_prewitt_edge(img,thresh)* on the noisy image (using thresh = 0.1) and show the results.

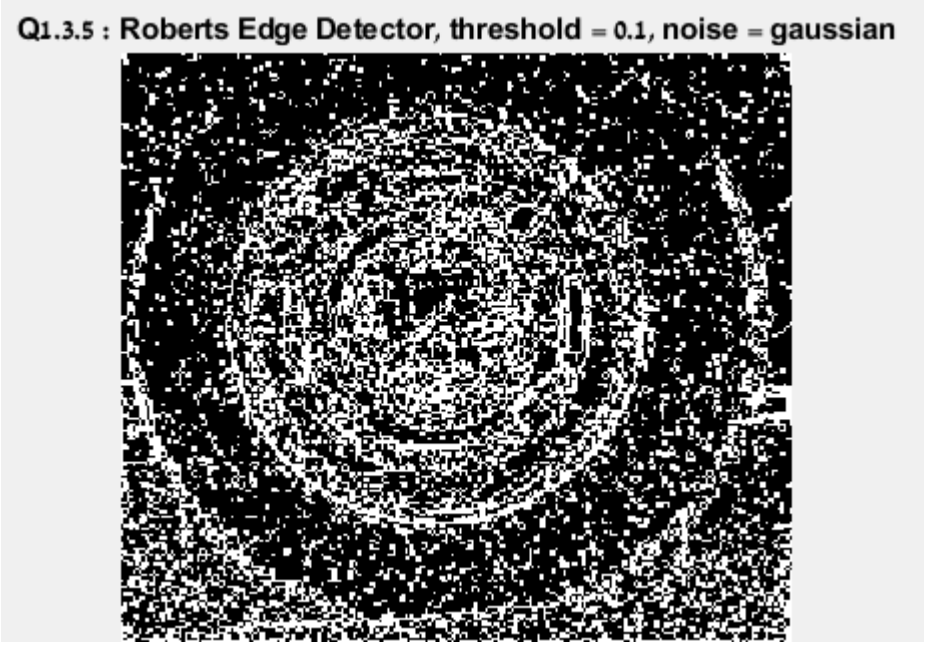


Figure 1.3.4.1

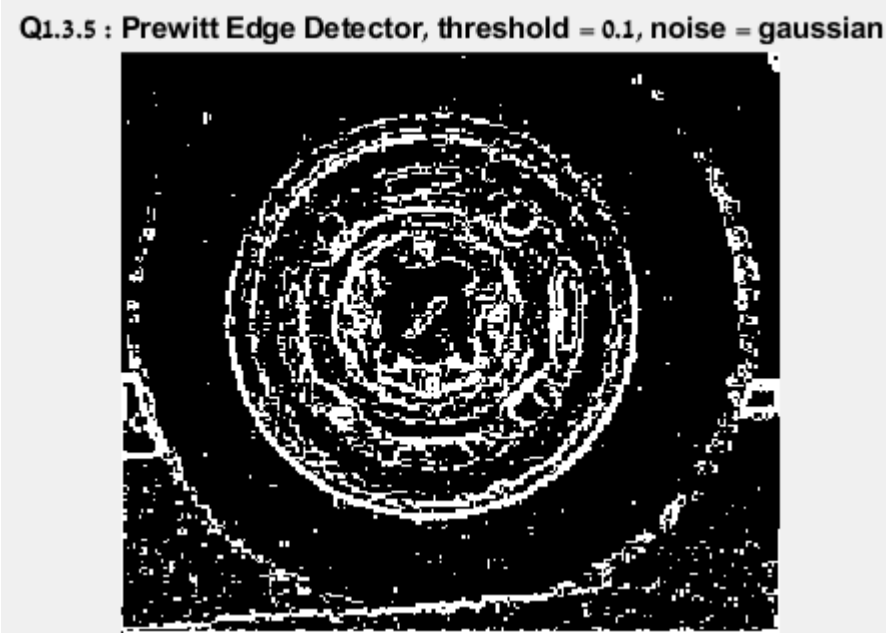


Figure 1.3.4.2

Explanation: in part 1.3.5.

1.3.5 What is the main disadvantage of the Roberts edge detector and how does the Prewitt edge detector solve it?

Explanation:
As we can see the Roberts edge detector is more sensitive to noise compare to Prewitt Edge detector. The main reason for using the Roberts Cross operator is that it is very quick to compute (Only four input pixels need to be examined to determine the value of each output pixel, and only subtractions and additions are used in the calculation). Its main disadvantages are that since it uses such a small kernel, it is very sensitive to noise. It also produces very weak responses to genuine edges unless they are very sharp. In this field Prewitt solves the problem using bigger kernel (3x3), but it will take more time to compute.[2]

1.4 Modified Prewitt Edge Detector

We dene a modified Prewitt edge detector. Its derivative kernels are defined as:

$$G_{MPx} = \frac{1}{9}\begin{pmatrix} -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \end{pmatrix}, G_{MPy} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

1.4.1 Write your own function named *dip_prewitt_edge_mod(img,thresh)* that will apply the modified Prewitt edge detector on *'img'* and output the edge image with the same size as the input image. The *'thresh'* parameter will determine the gradient magnitude cutoffthreshold.
Note: You don't have to deal with the image boundaries, you can use the conv2() function with the parameter 'same'.

Explanation: The only changed the kernels as requested.

[2] Image processing learning resource HIPR - http://homepages.inf.ed.ac.uk/rbf/HIPRUSERS

1.4.2 Compare the performance of the modified Prewitt detector to the original Prewitt detector on the noisy image from the previous section. Display the images and explain the reason for the difference.

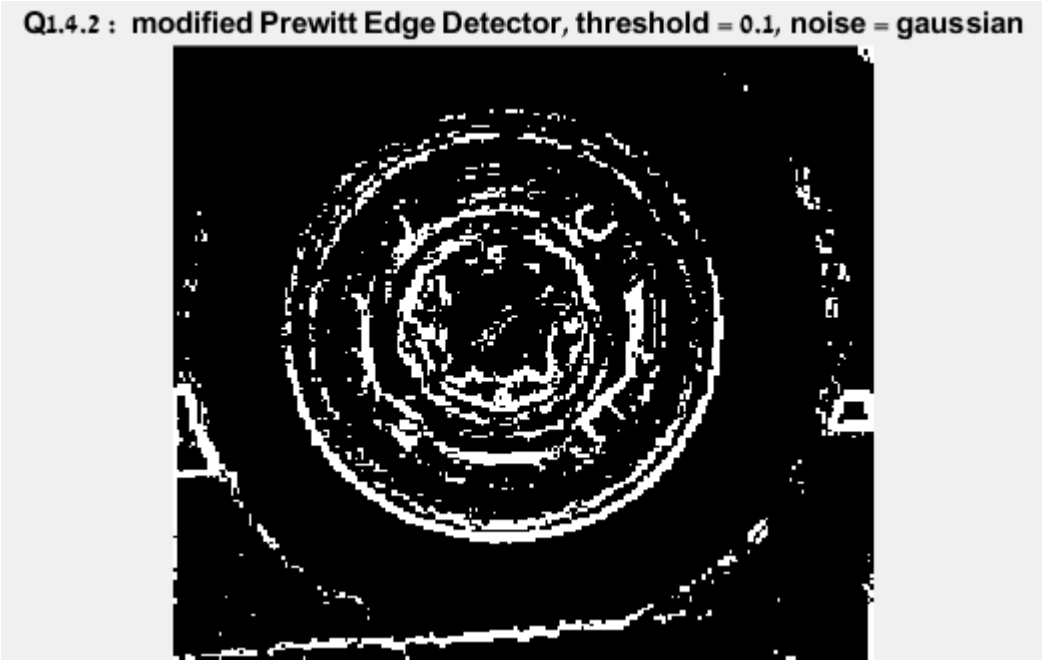


Figure 1.4.2.1

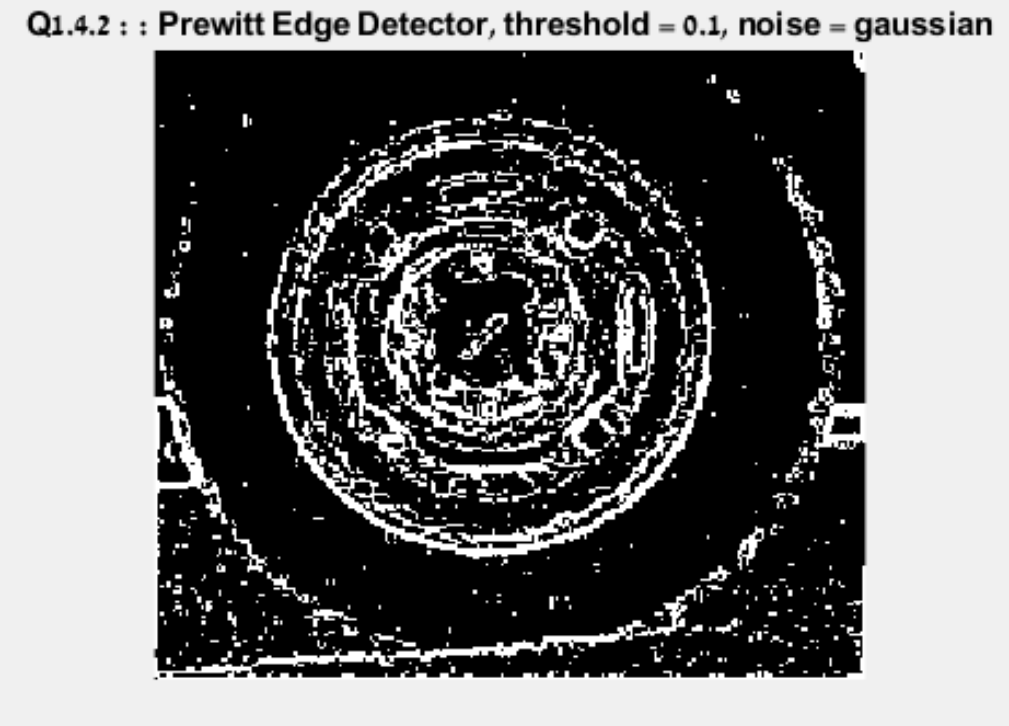


Figure 1.4.2.2

Explanation: We can see that the modified Prewitt Edge Detector performed better on the noised image and detected better the edges. Hence, the kernel size and the mathematical equation determine how well the edge detection will perform on noisy inputs. In the modified version, we are using larger kernels enable us to smooth the input image to greater extent and so makes the operator less sensitive to noise in the convolution.

1.4.3 What is the disadvantage of using the modified Prewitt edge detector?
Explanation: Since we are computing convolution with larger kernels usually it supposed to take more time to compute. The result of using this larger kernel will have thicker edges.

1.5 Canny Edge Detector

The Canny edge detector is the most commonly used edge detector.

1.5.1 Write a short summery about the Canny edge detector algorithm.

Explanation:
The canny edge detector is a multistage edge detection algorithm consist of :

1. **Preprocessing** – smoothing the image to get better results on noisy images.
2. **Calculating gradients** -gradient magnitudes and directions are calculated at every single point in the image. The magnitude of the gradient at a point determines if it possibly lies on an edge or not. A high gradient magnitude means the colors are changing rapidly - implying an edge. A low gradient implies no substantial changes. So, it's not am edge.
   The direction of the gradient shows how the edge is oriented. To calculate these, we can use one of the Edge Detector algorithms we learned above.
3. **Nonmaximum suppression** - Non-maximum suppression is applied to find "the largest" edge. Thus, non-maximum suppression can help to suppress all the gradient values except the local maxima, which indicate locations with the sharpest change of intensity value.
4. **Thresholding with hysteresis** - some edge pixels remain that are caused by noise and color variation. In order to account for these spurious responses, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value. Also, usually a weak edge pixel caused from true edges will be connected to a strong edge pixel while noise responses are unconnected. To track the edge connection, blob analysis is applied by looking at a weak edge pixel and its 8-connected neighborhood pixels. If there is one strong edge pixel that is involved in the blob, that weak edge point can be identified as one that should be preserved.

1.5.2 Use the MATLAB functions *edge(img,'Canny')* and *edge(img,'Prewitt')* on the image and show the results. Note: if a specific threshold is not mentioned, the MATLAB function will choose the threshold by itself.

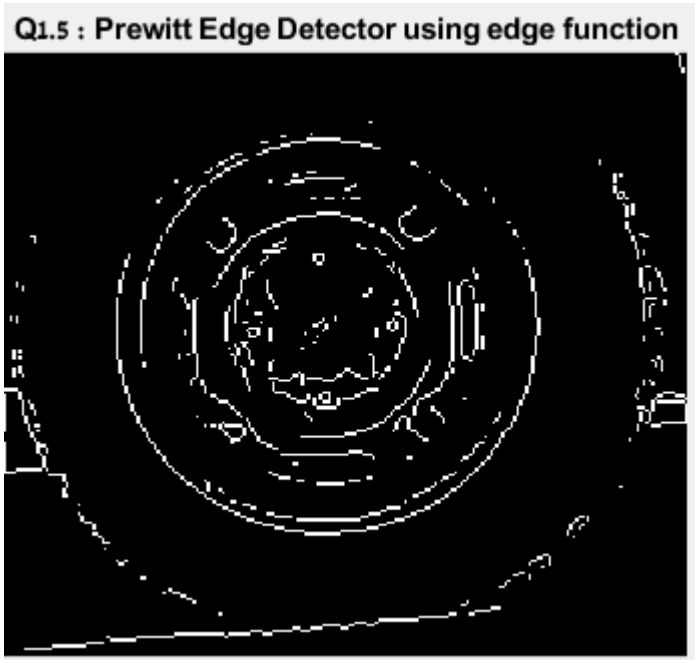


Figure 1.5.2.1

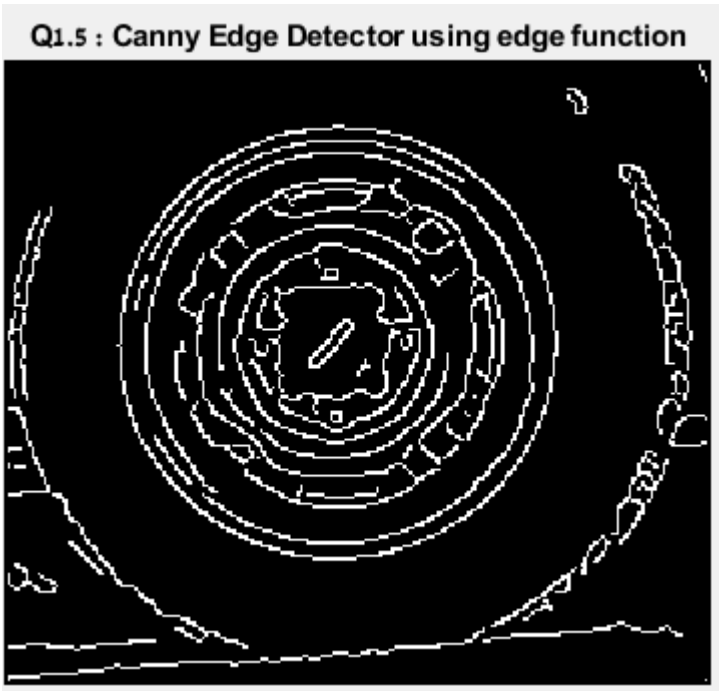


Figure 1.5.2.2

1.5.3 Explain the differences between the two edge detectors as it can be seen from the images. Which edge detector will you use?

Explanation: We can see that Canny Edge Detector perform better on this image. The results show us more edges and more defined edges compare to Prewitt Edge Detector. We will choose between the 2 algorithms depends on the situation. If we are looking for finding better defines edges (even on noisy images) we will use Canny Edge Detector, otherwise we will choose Prewitt Edge Detector for quicker results.

1.6 Diamonds are forever

In this subsection you will write a function named dip_edge_detect(img) that detects the outer bounds of the diamond in the attached diamond image, according to the following criteria. You may use built-in MATLAB functions. Special attention will be given to the criteria in your code, be very strict.

- The function receives an image of an unknown size, meaning you cannot assume a predefined size. The code will be checked with a different sized image.
- The function detects the diamond's edges.
- The function has two outputs: The first is a row vector with a length of 4, the second output is a one channeled logical edge image of the same size as the input image.
- The row vector represents the diamond's bounding box. A bounding box is the box with the smallest volume (surface in this case) that bounds the desired object. Thefirst and second coordinates of the vector are the bounding box's upper left corner's column and row number respectively. The third and fourth coordinates are the bounding box's width and height respectively. The vector must be of type uint16. Make sure the bounding box first in the image. It is recommended to use a built-in MATLAB function to find the bounding box.
- The edge image has to be of type logical, of the same size as the input image but must have one channel (i.e. even if the input has 3 channels because it is colored, the edge image still has to have one channel).
- Use the logical edge image to create the vector representing the bounding box.

- In case no edges are detected, the output vector is the zero vector of type uint16 and the output image is the zero matrix of type logical.
- It is okay if the bounding box misses a few pixels of the diamond.

1.6.1 Display the output of the function. Do not use figures, imshow, etc. within the function. No explanation is needed.

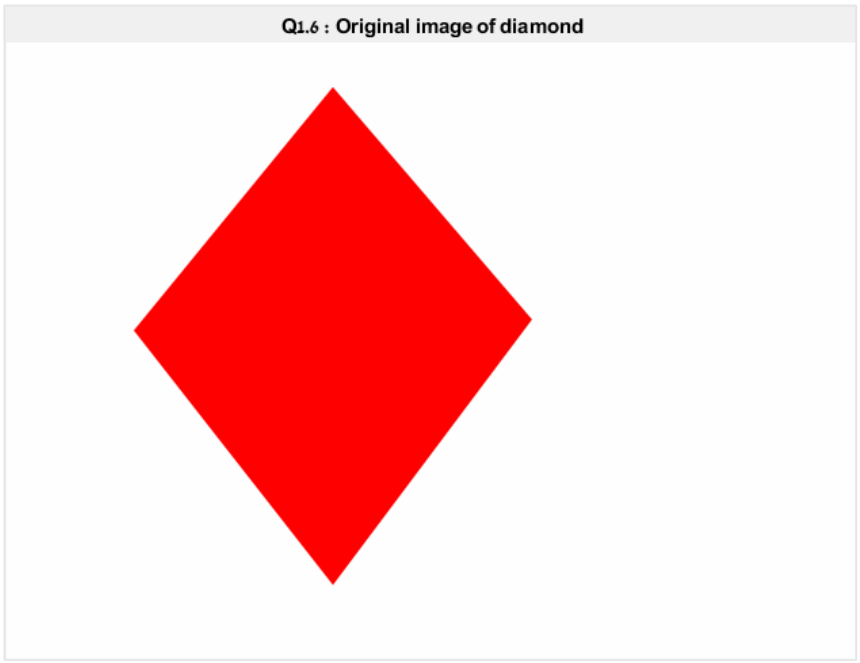


Figure 1.6.1.1 – Original image

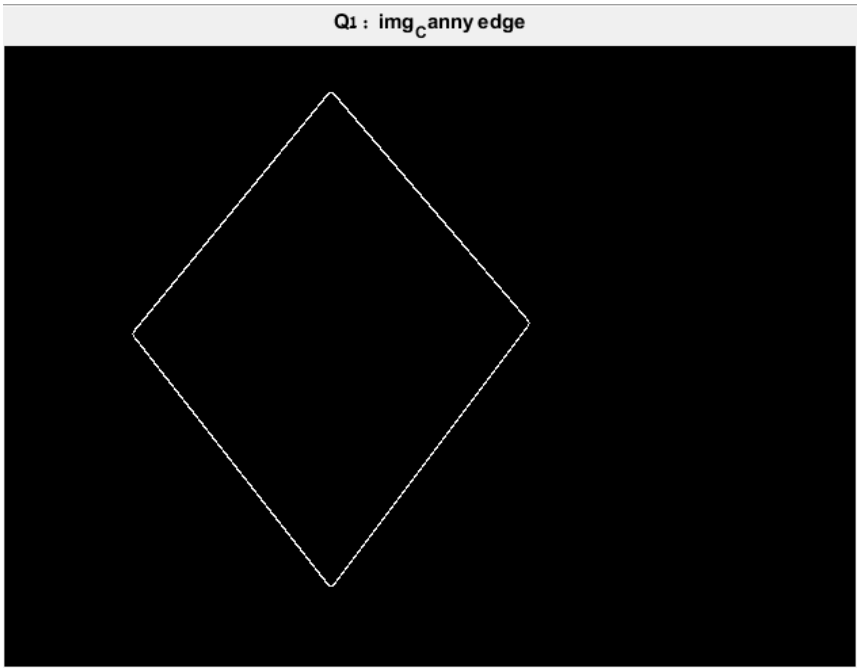


Figure 1.6.1.2 – results of inner use of Canny Edge Detector

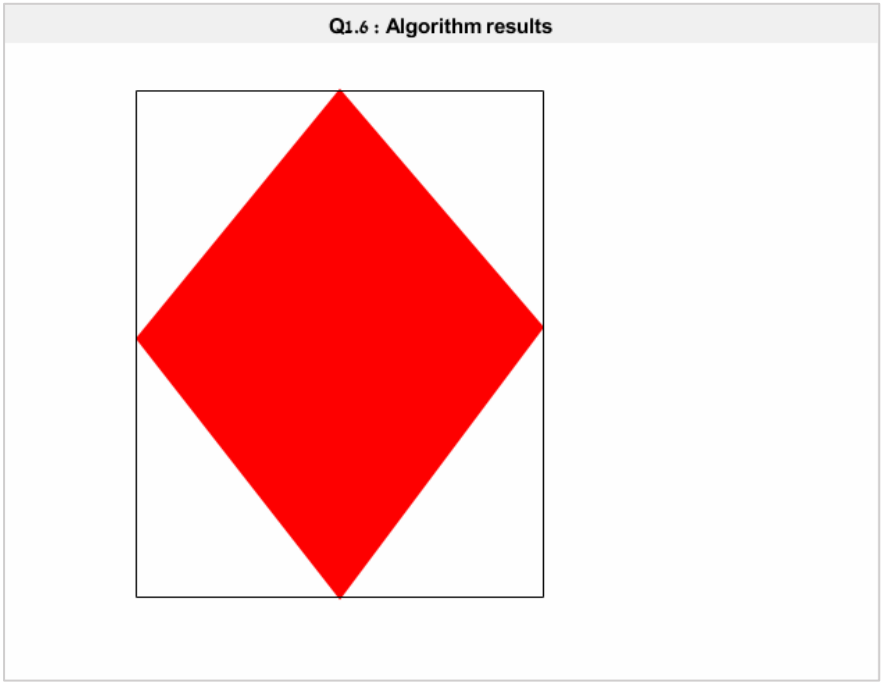
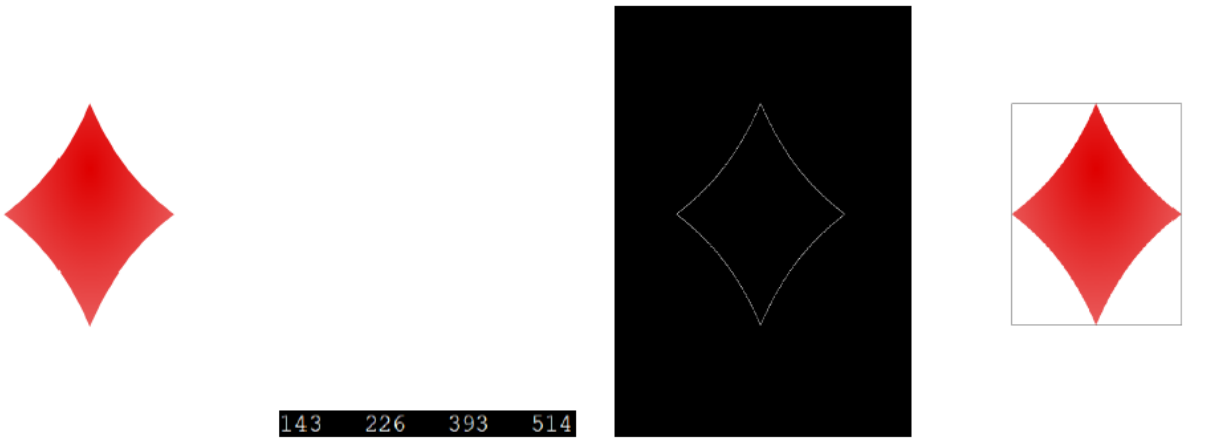


Figure 1.6.1.3 – Algorithm results

Explanation:

1.6.2 An example output is demonstrated in Figure 1 . You do not have to display the result image composed of the diamond together with the bounding box.



(a) example diamond (b) example vector (c) example edge (d) example result

Figure 1.6.2.1 - Example

2 Mona Surfaliza

In this exercise we will use SURF features in order to x the orientation of the image and detect objects in an image. You may use all available MATLAB functions but a special emphasis will be made to your written explanations.

2.1 Make Mona Straight Again

2.1.1 Explain the SIFT algorithm.

Figure 2.1.1.1

Explanation:

The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images

1. Constructing a scale space - This is the initial preparation. We create internal representations of the original image to ensure scale invariance. This is done by generating a "scale space".
2. LoG Approximation - The Laplacian of Gaussian is great for finding interesting points (or key points) in an image. But it's computationally expensive. So, we cheat and approximate it using the representation created earlier.
3. Finding keypoints - With the fast approximation, we now try to find key points. These are maxima and minima in the Difference of Gaussian image we calculate in step 2.
4. Get rid of bad key points - Edges and low contrast regions are bad key points. Eliminating these makes the algorithm efficient and robust. A technique like the Harris Corner Detector is used here.
5. Assigning an orientation to the key points - An orientation is calculated for each key point. Any further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant.
6. Generate SIFT features - Finally, with scale and rotation invariance in place, one more representation is generated. This helps uniquely identify features. Let's say we have 20,000 features. With this representation, we can easily identify the feature we are looking for (say, a eye, or a sign board).

7. After we got the SIFT features, we can Track images, detect and identify objects (which can be partly hidden as well) etc.

2.1.2 Shortly explain how the SURF algorithm improves the SIFT algorithm and what these improvements are.

Explanation:

This research[3] has evaluated those two feature detection methods for image registration. Based on the experimental results, it is found that the SIFT has detected a greater number of features compared to SURF, but it is suffered with speed.

There are a couple reasons to why SURF is faster. It approximates the difference of Gaussians in the scale space with box filters (Instead of Gaussian averaging the image, squares are used for approximation since the convolution with square is much faster if the integral image is used.), Also this can be done in parallel for different scales.

In addition, SURF uses a BLOB detector which is based on the Hessian matrix to find the points of interest. For orientation assignment and feature description, it uses wavelet responses in both horizontal and vertical directions by applying adequate Gaussian weights. For also SURF uses the wavelet responses.

The sign of Laplacian which is already computed in the detection is used for underlying interest points. The sign of the Laplacian distinguishes bright blobs on dark backgrounds from the reverse case. In case of matching the features are compared only if they have same type of contrast (based on sign) which allows faster matching.

[3] http://www.ijircce.com/upload/2013/april/21_V1204057_A%20Comparison_H.pdf - A Comparison of SIFT and SURF

2.1.3 Read the images 'mona_org.jpg' and 'crooked_mona.jpg '.

Figure 2.1.2.1

Figure 2.1.2.2

Explanation: We read the images. Later, we will use the function rgb2gray() for converting them to grayscale. Note : we could use our own function dip_rgb2gray() from assignment 2 which uses the exact weights for each R, G, B channel according to MATLAB documentation.

2.1.4 Extract the SURF feature points of each of the images and display 10 of them for each of the images correspondingly.

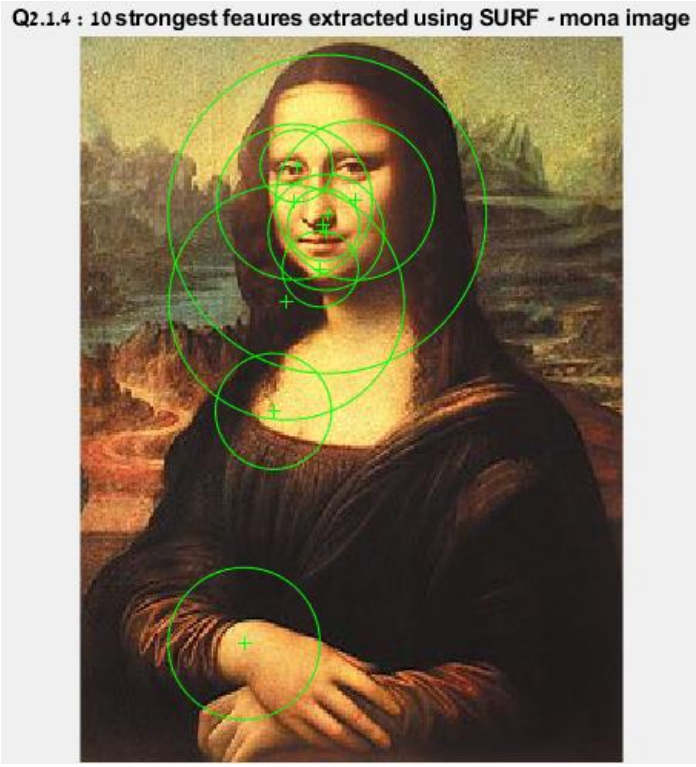


Figure 2.1.4.1

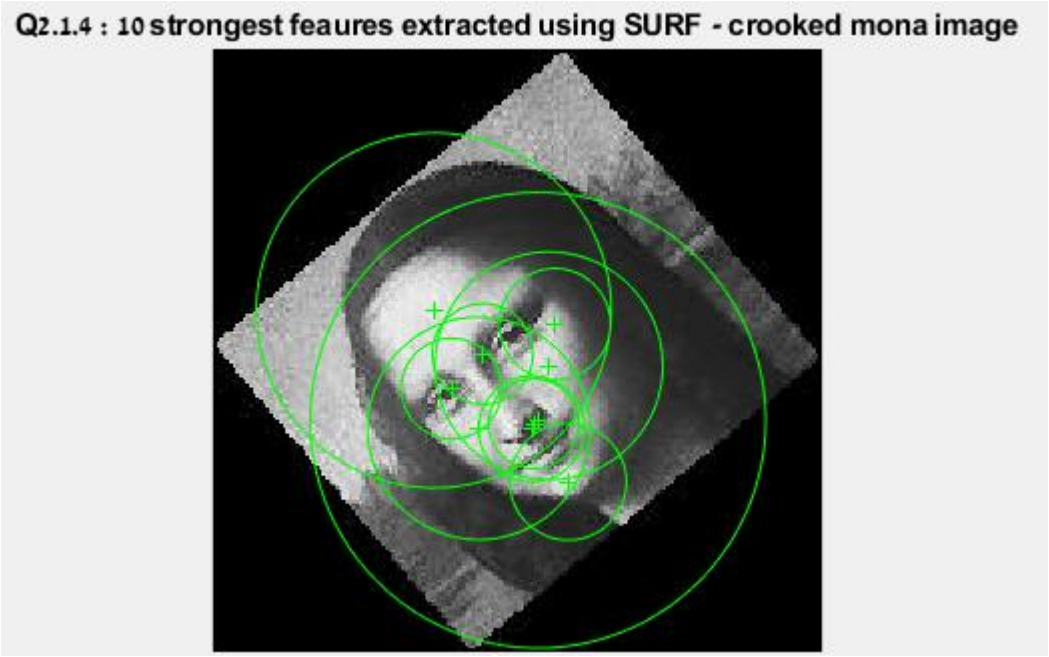


Figure 2.1.4.2

Explanation: We can see some resembling in most of the features, which concentrate on the face area.

2.1.5 Use the extracted feature points to straighten the 'crooked_mona.jpg ' image so that Mona's face will be straight. Display the images. Explain your algorithm using a block diagram and elaborate on each of the steps.

Figure 2.1.4.1

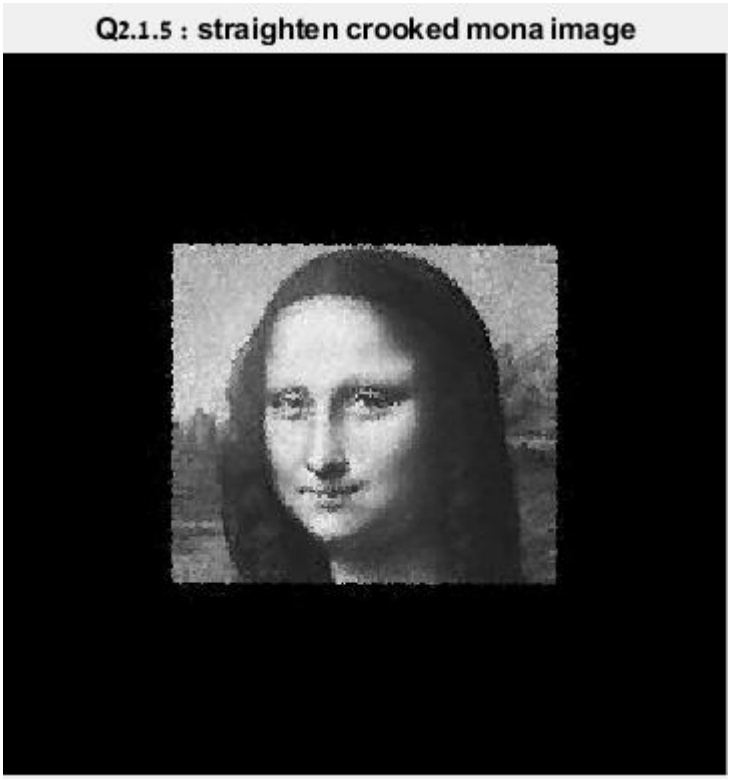


Figure 2.1.4.2

Explanation: We used our own function 'dip_ straight (img1,img2,features1 ,features2, valid_corners1 ,valid_corners2)' which gets 2 images and the required information extracted previously from SURF. As we can see our algorithm output valid result.

Our Straight algorithm :

1. Preparation for inputs - Extract the strongest features of the 2 images using SURF or SIFT algorithms. Extracted feature vectors, also known as descriptors, and their corresponding locations.
2. Using *'matchFeatures(features1,features2)'* built-in MATLAB function we get indices of the matching features in the two input feature sets - features1 and features2.
3. After that, we are extracting the location of each matched feature from valid_corners1 and valid_corners2 using the index pairs values returning from Step 2.
4. Now we will look at the pairs (x, y) of each correspond features, using 'Location' property of the returning values from SURF.

5. For each correspond feature, we will compute the slope between the location of this feature on image A and image B, then we will extract the angle difference using

$$m = \left(\frac{\Delta y}{\Delta x}\right) = \tan(\theta).$$

6. After getting all the angle differences (each angle represents the angle differences of specific feature) we sorted the results, then output the median which should represent the angle differences of the entire images with the highest probability (hence we sorted and choose the median, the error values should be on the margins of the array).
7. Finally, we will rotate the image using the built-in MATLAB function '*imrotate(Image, Angle)*' with the result angle teta.

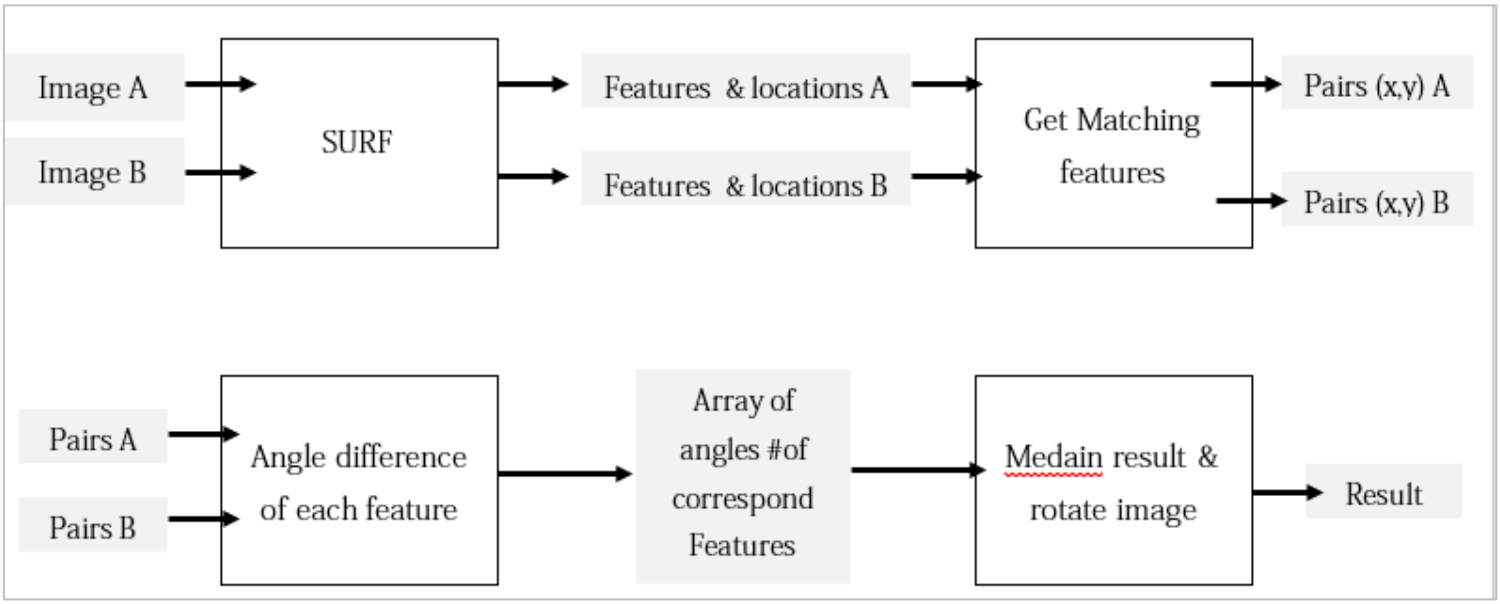


Figure 2.1.4.3 – Our algorithm flow

2.2 Fake or Real

2.2.1 Read the image 'straight_mona.jpg' and use the SURF features to automatically detect images in which the real Mona Liza face exists.

1. The 10 test images are in the zip file named 'Monas'.
2. The last letter in the image name 'Y' or 'N' suggests if we consider that Mona's face is in the image or not.
3. The code should print out the names of the images you detected Mona's face inside.
4. The algorithm should be totally automatic and run on all the images in the same manner.
5. Try taking under consideration the trade-off between finding a lot of correct images and falsely detecting wrong images.
6. Try to do your (and your algorithms) best, if you can't detect an image, just try your best and explain why it was impossible.

2.2.2 Display the images you found.

Figure 2.2.2.1

Figure 2.2.2.2

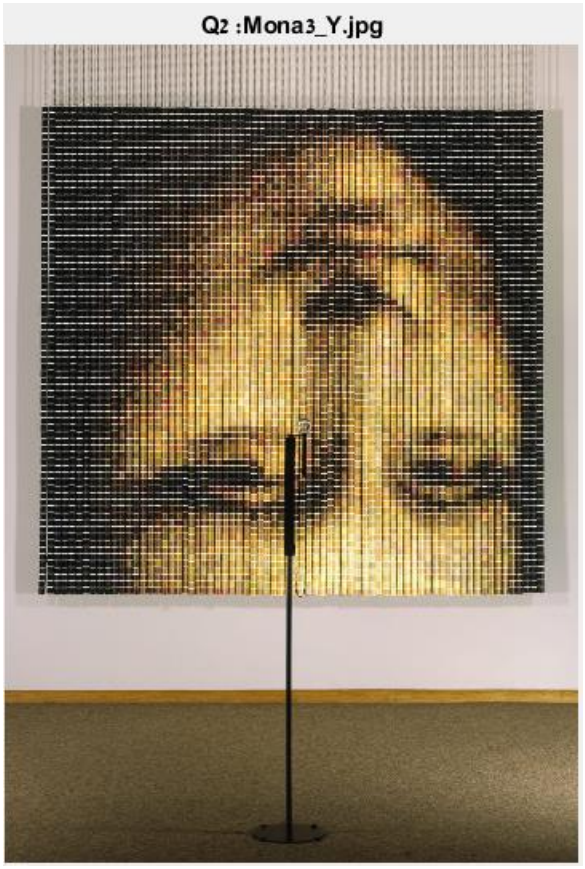


Figure 2.2.2.3

Figure 2.2.2.4

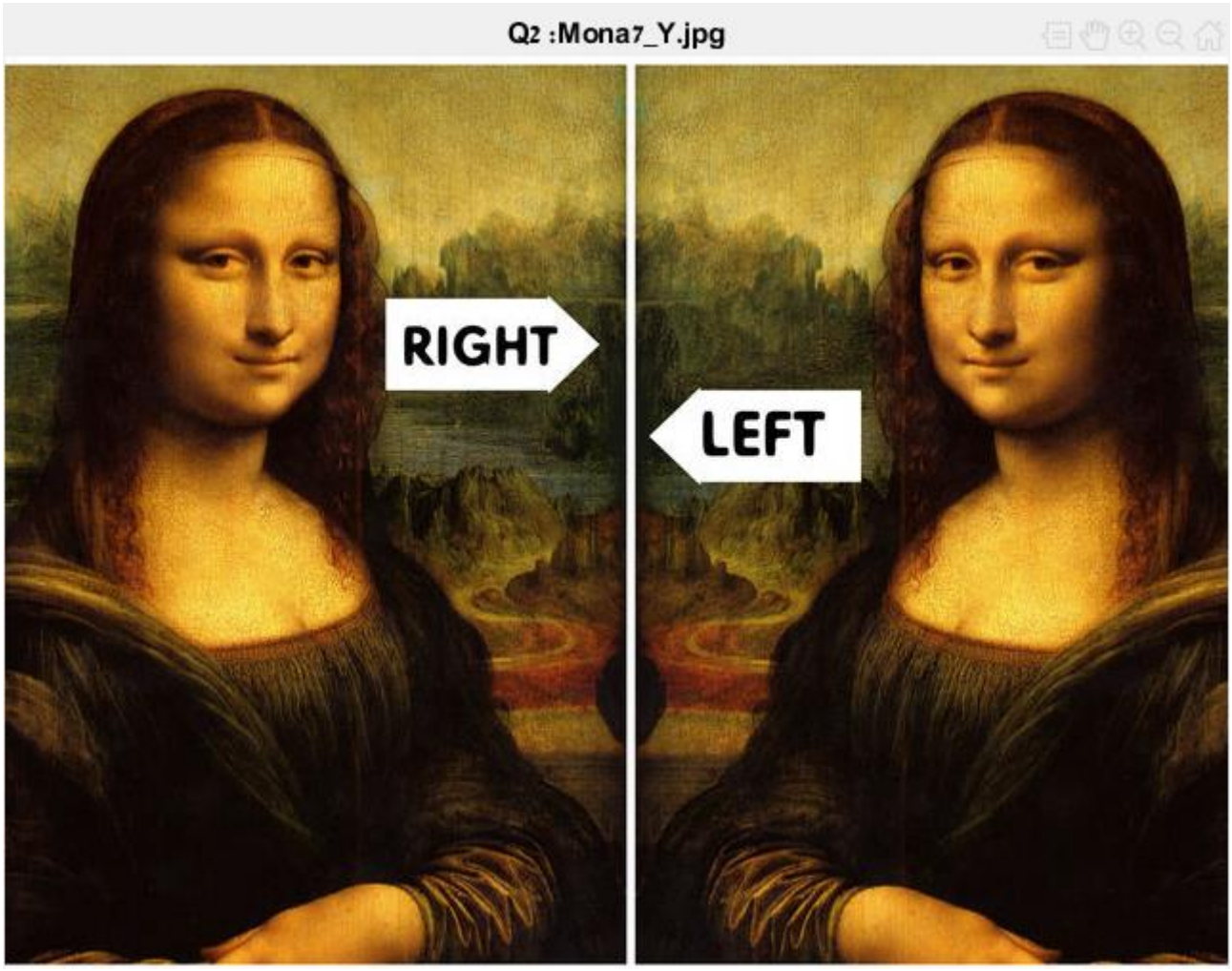


Figure 2.2.2.5

Figure 2.2.2.6

Explanation: As we can see, our algorithm has been able to find 6/8 of the image contains mona-liza face labeled "Y" and has been successful to exclude 2/2 images labeled "N".

2.2.3 Explain your algorithm using a block diagram and elaborate on each of the steps.

Explanation:
Image detection SURF based Algorithm :

1. We unzipped the ' Monas.zip' file and send it as input to this algorithm call.
2. We read the original mona-liza image and convert it to grayscale. This image can be replaced for any other reference image representing the features that we wish to look for in another images.
3. We extracted the original image features with SURF algorithm using the built-in MATLAB function *'detectSURFFeatures'* using 8 scale levels to compute per octave (Increase This number to detect more blobs at finer scale increments) which returns the features object descriptor and their location.

4. For each image in the ZIP file provided, we execute the following :
    a. We read the image from the input object using an index var I and converted it to grayscale.
    b. We compute the image features using Step 3.
    c. Using *'matchFeatures(features1,features2)'* built-in MATLAB function we get indices of the matching features in the two input feature sets – features from the original image and features from the current image we are checking.
    d. If there are more than 1 corresponding feature we are deciding that this picture have mona-liza face in it.

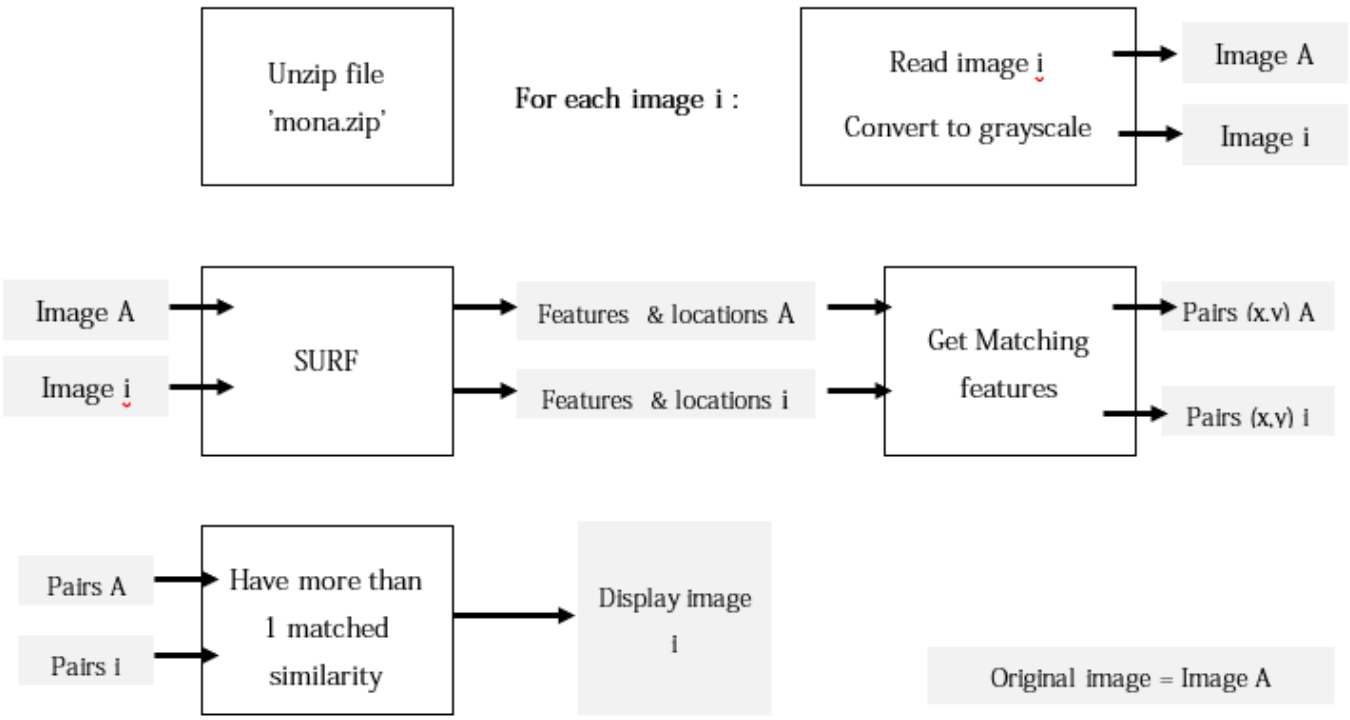


Figure 2.2.3.1 – Algorithm flow

2.2.4 For each image show the matching features.

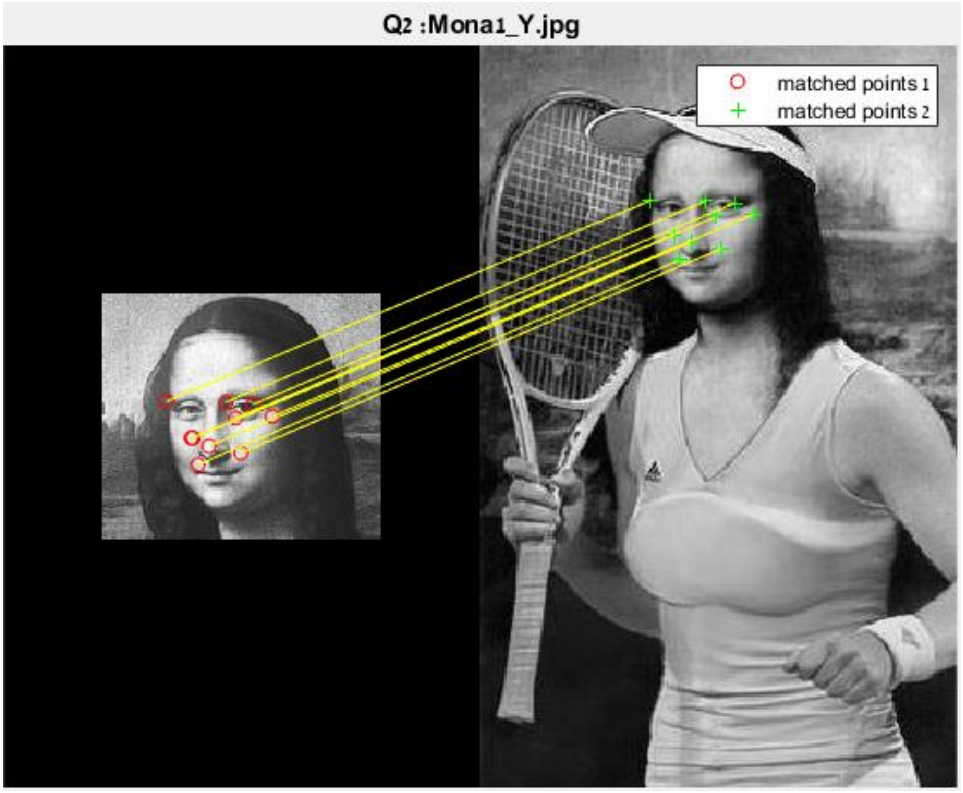


Figure 2.2.4.1

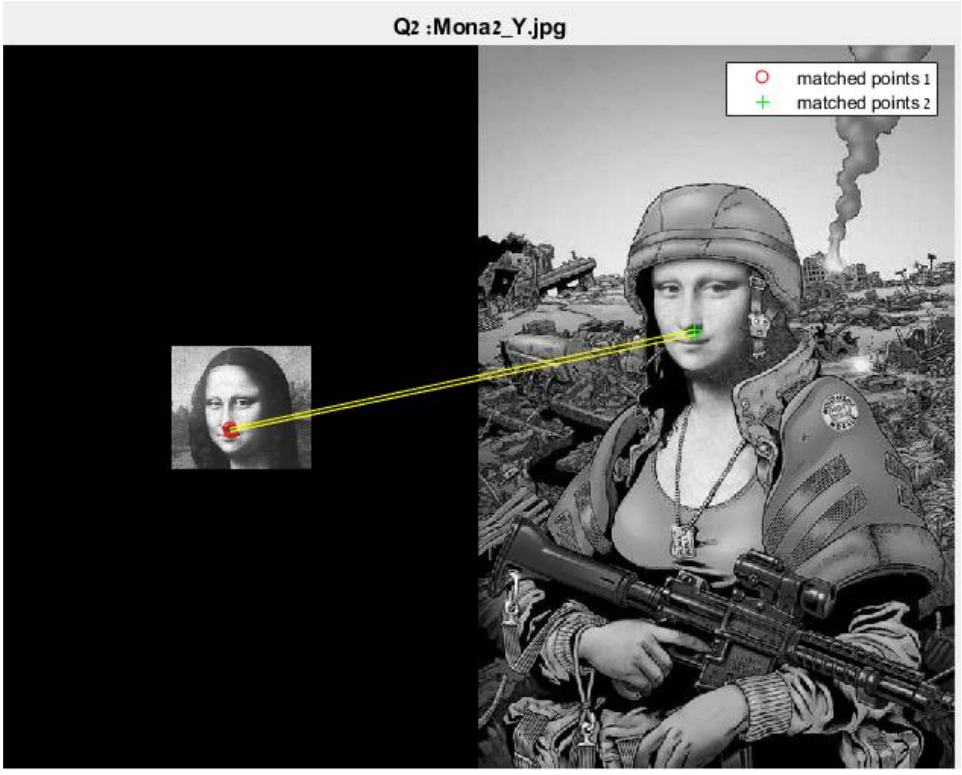


Figure 2.2.4.2

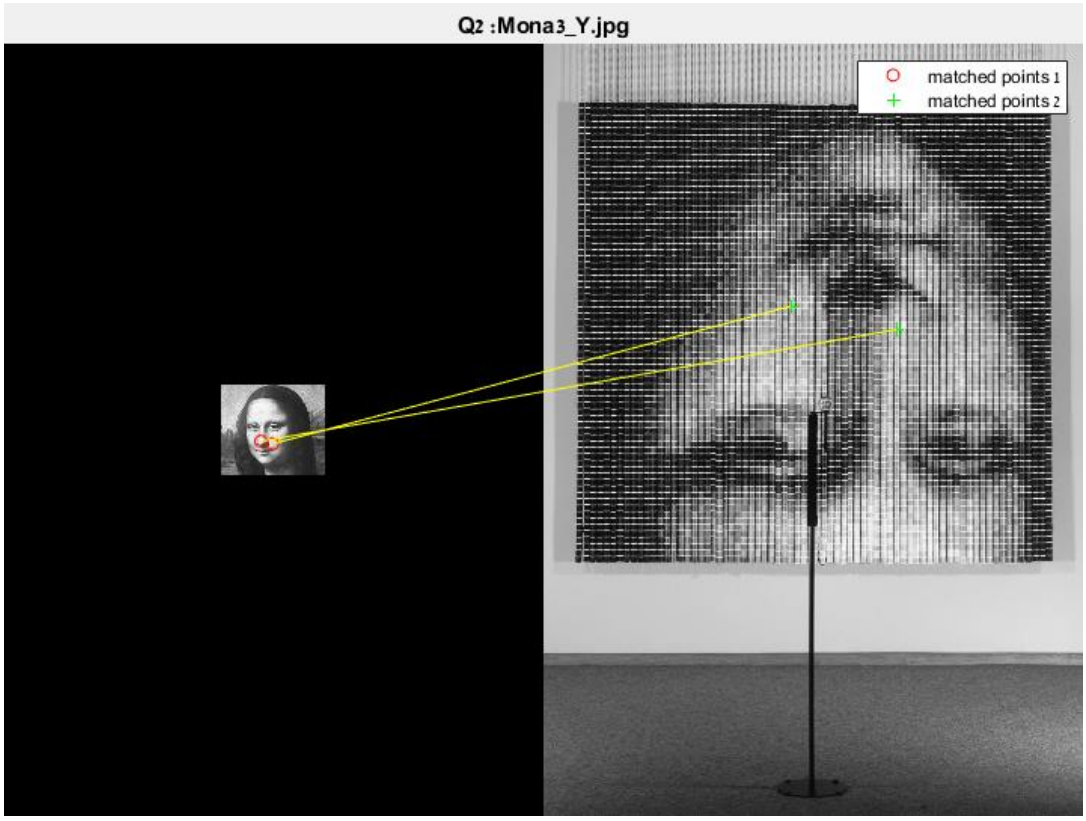


Figure 2.2.4.3.1

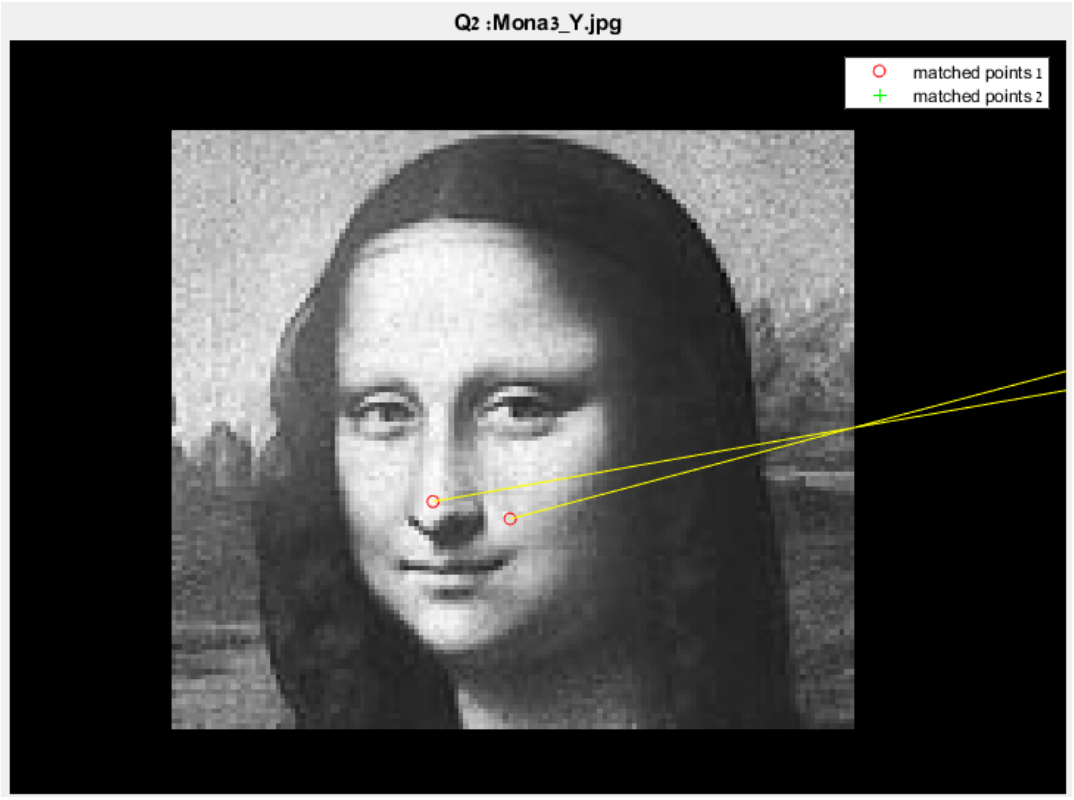


Figure 2.2.4.3.2 – zoom in of Mona3_Y

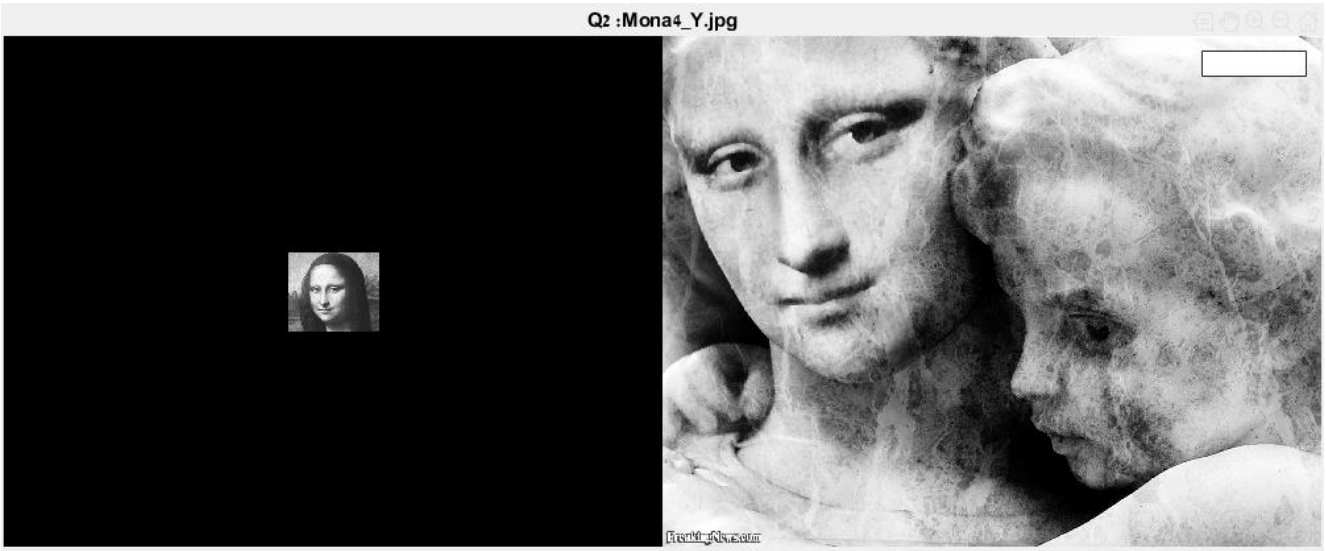


Figure 2.2.4.4

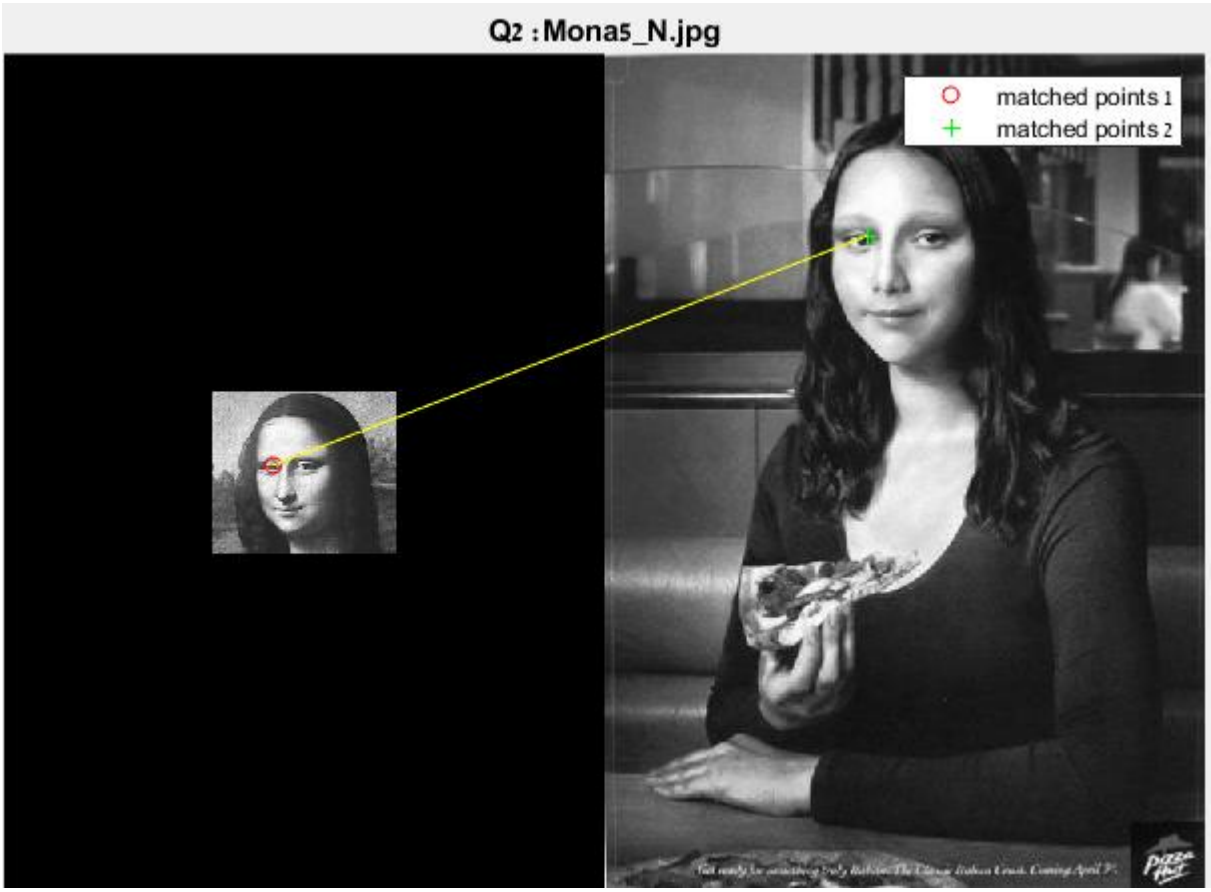


Figure 2.2.4.5.1

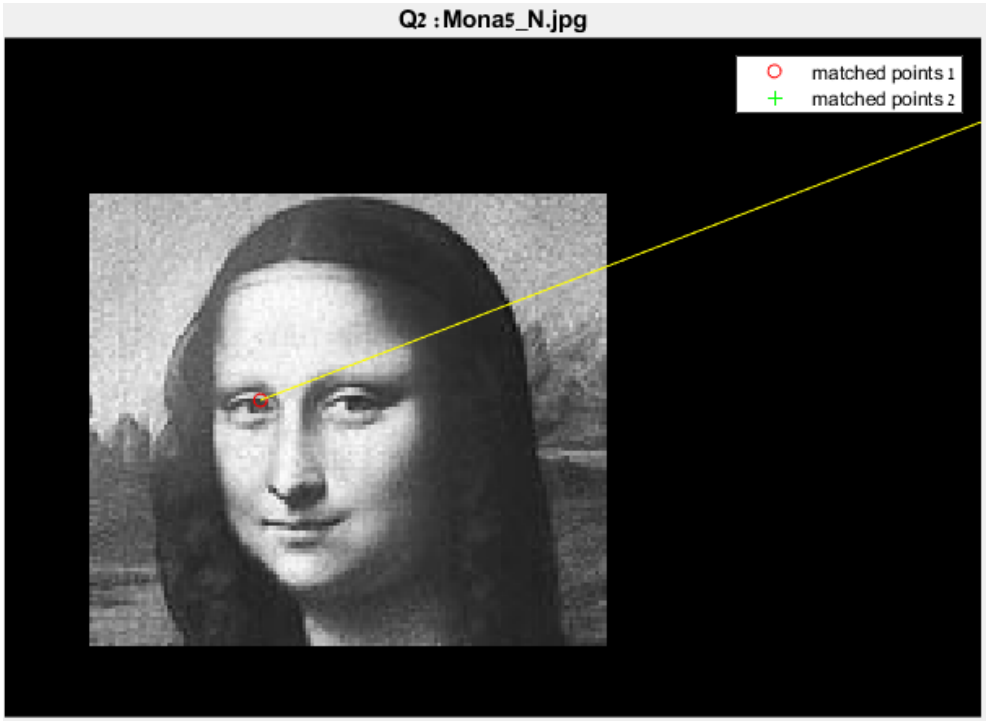


Figure 2.2.4.5.2 – Zoom in of Mona5_N

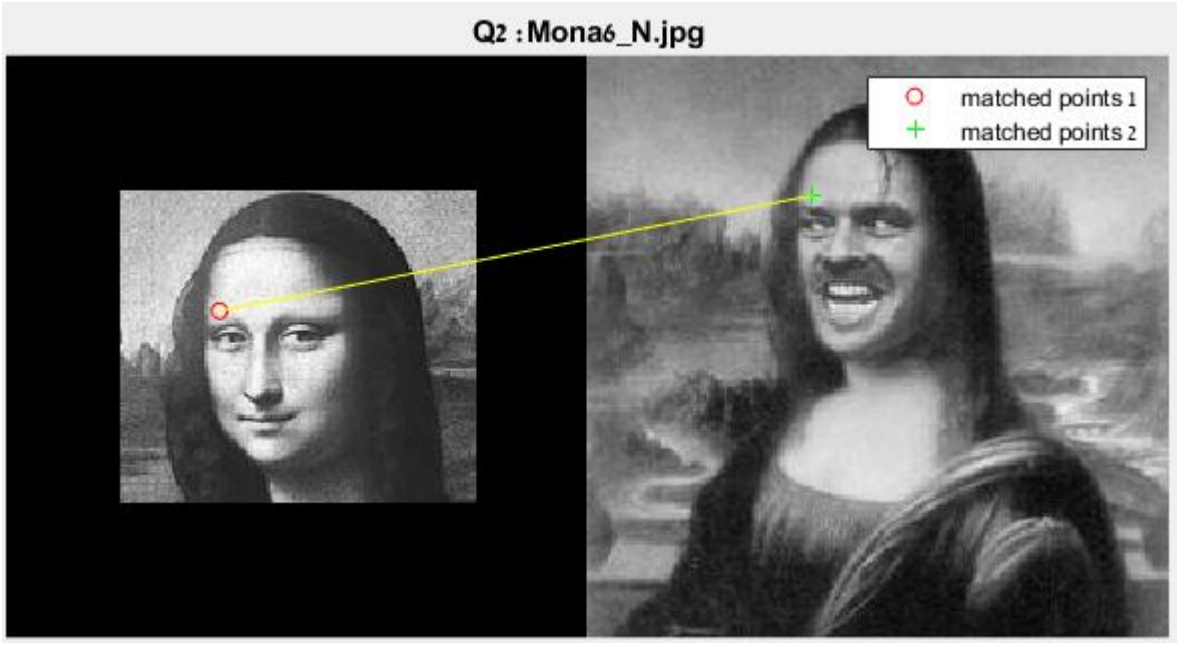


Figure 2.2.4.6

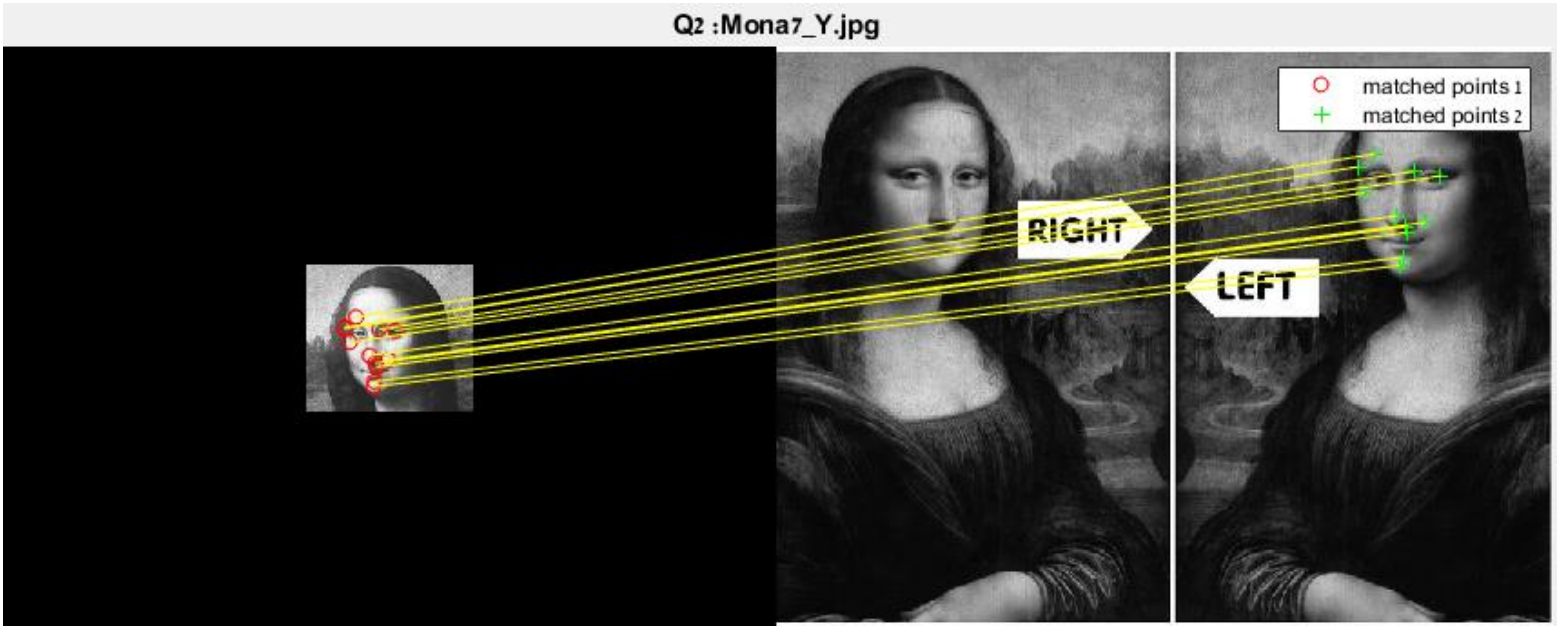


Figure 2.2.4.7.1

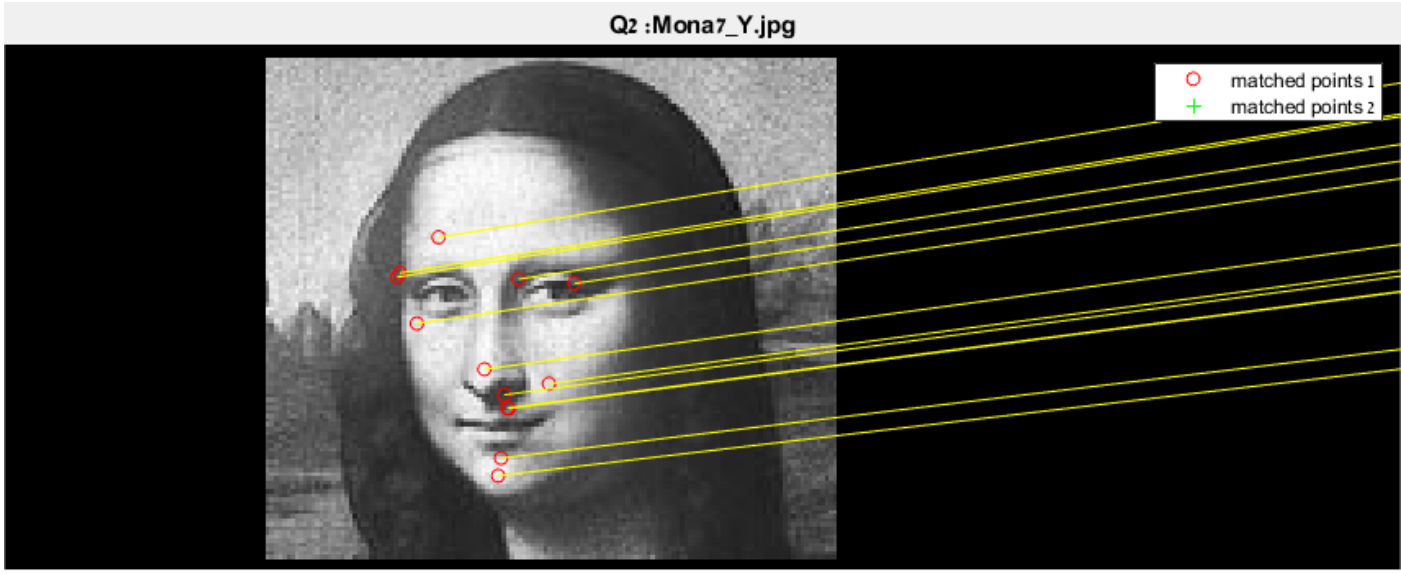


Figure 2.2.4.7.2 – zoom in on Mona7_Y

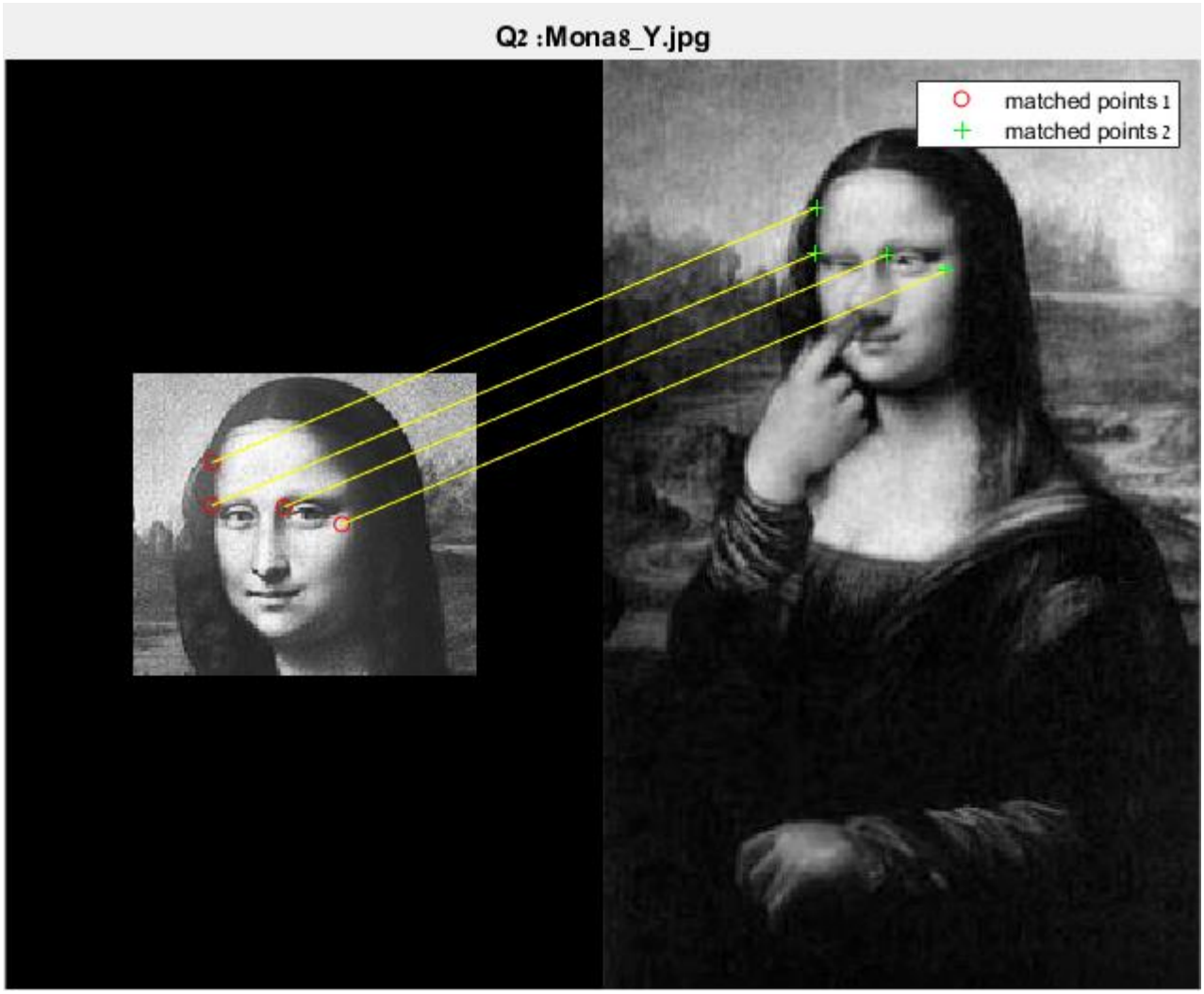


Figure 2.2.4.8

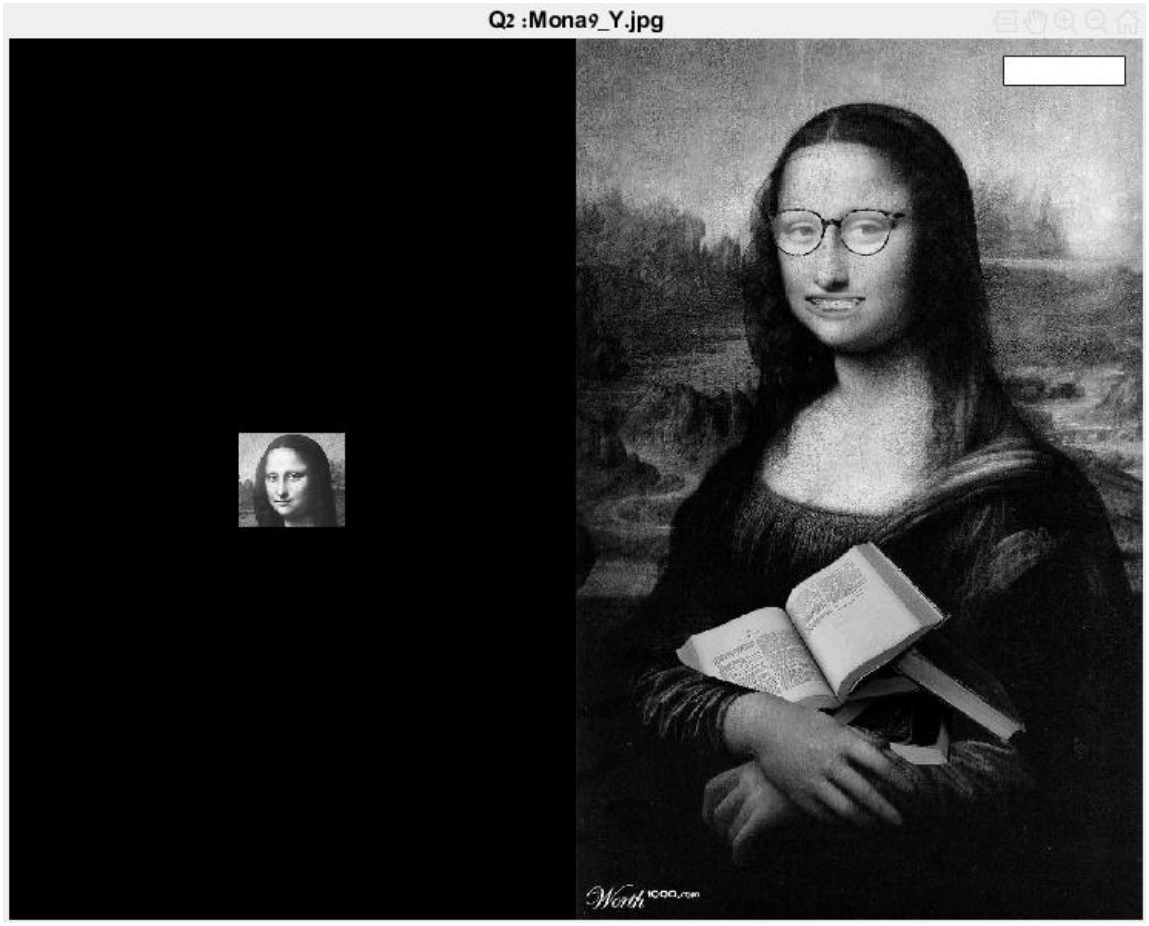


Figure 2.2.4.9

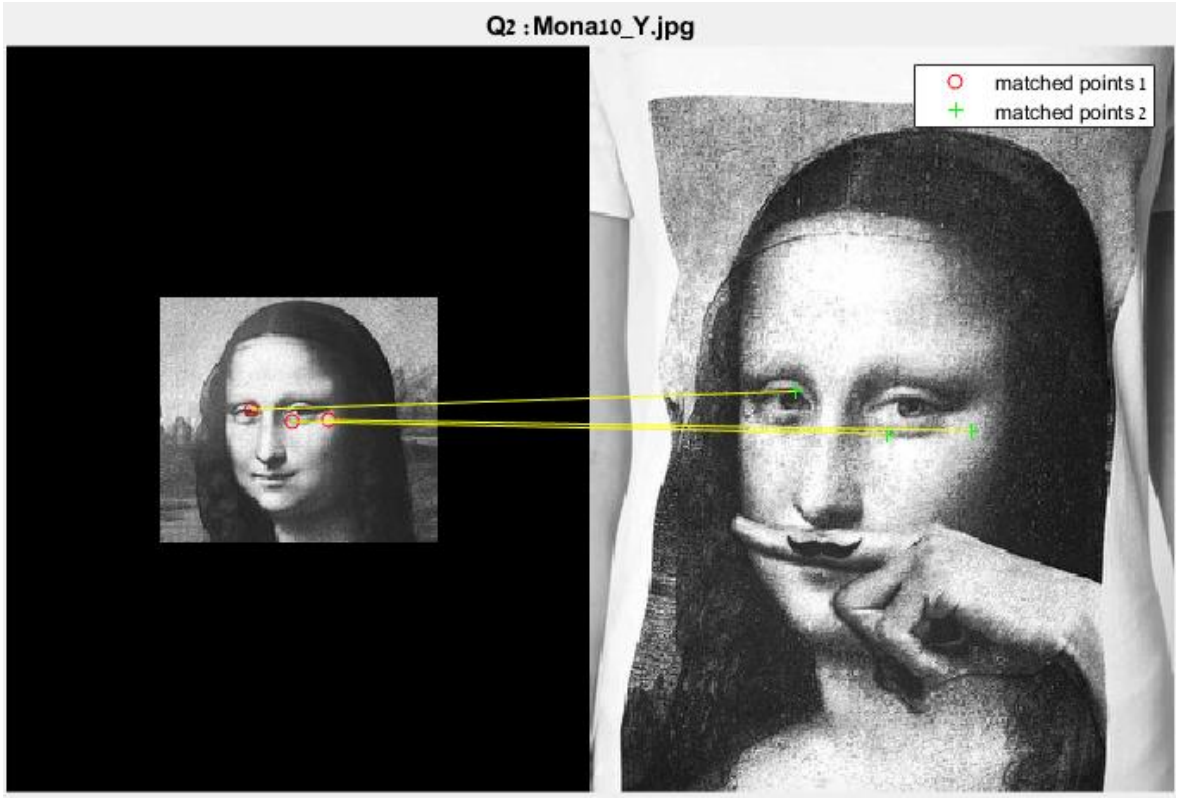


Figure 2.2.4.10

2.2.5 Analyze the results. Why did you succeed in the image you did? Why did you get it wrong in the images you did? Where did the SURF features have good performance and where did they not?

Explanation:

As we can see, our algorithm has been able to find 6/8 of the image contains mona-liza face labeled "Y" and has been successful to exclude 2/2 images labeled "N".
The results we got depends on the parameters we used on *'detectSURFFeatures'* function, such as
'NumOctaves' - Number of octaves to use. Increase this value to detect larger blobs. Recommended values are between 1 and 4. Default: 3

'NumScaleLevels' - Integer scalar, NumScaleLevels >= 3. Number of scale levels to compute per octave. Increase this number to detect more blobs at finer scale increments. Recommended values are between 3 and 6. Default: 4

We used default number of Octaves and NumScaleLevels of 8. We tried other combination but couldn't success where this combination failed to recognize images 9 and 4.

From our analysis, images that contains mona-liza face with minimal disruption (to the main areas where was the strongest features from SURF, e.g. the eyes) was found with at least 2 features by the SURF algorithm, while images that their disruption of the original image was too impactful that the features of that image were no longer has common features, such as image 4 – Mona4, As shown :

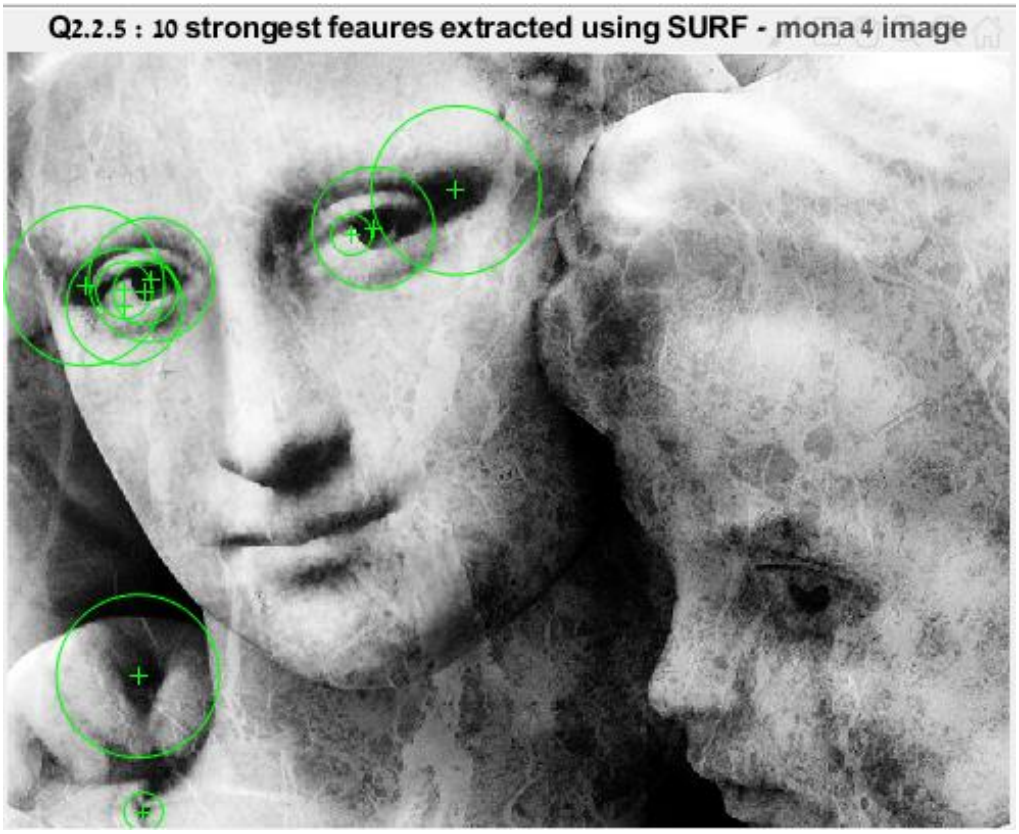


Figure 2.2.5.1 – zoom in

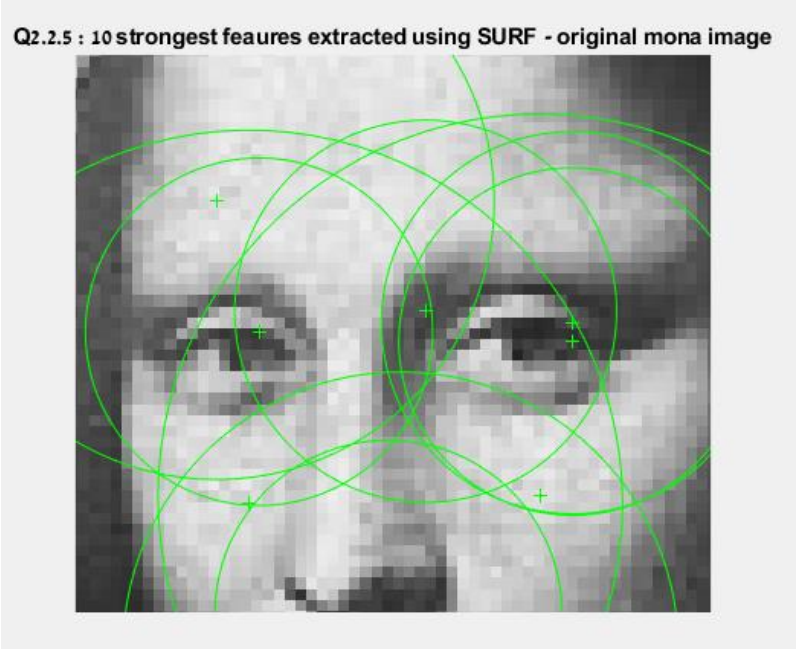


Figure 2.2.5.2 – zoom in

As we can see, those images have a different feature selected in the common area (mona-liza face), which yields that the algorithm will not find any common features.