

Task 2

Computer Vision

Submitted to:

Eng. Ayman Anwar

Eng. Laila Abbas

Prepared by:

AlZahraa Eid

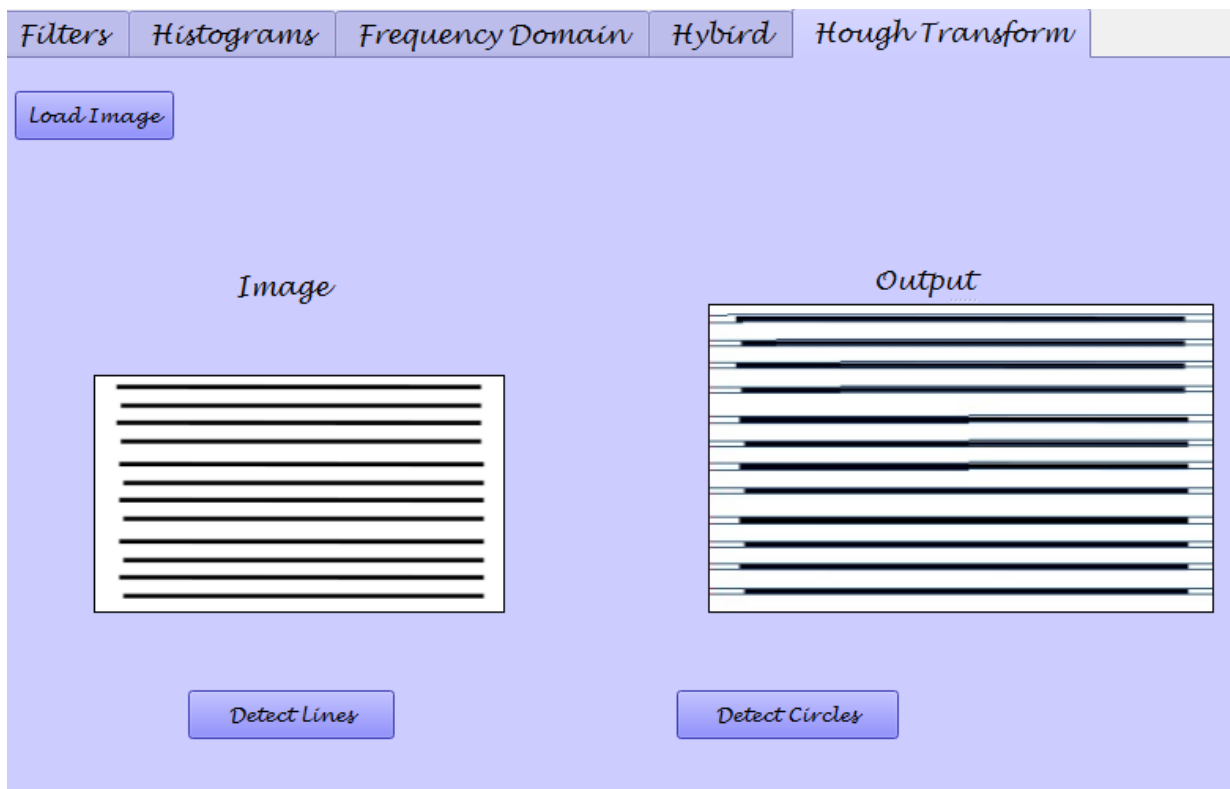
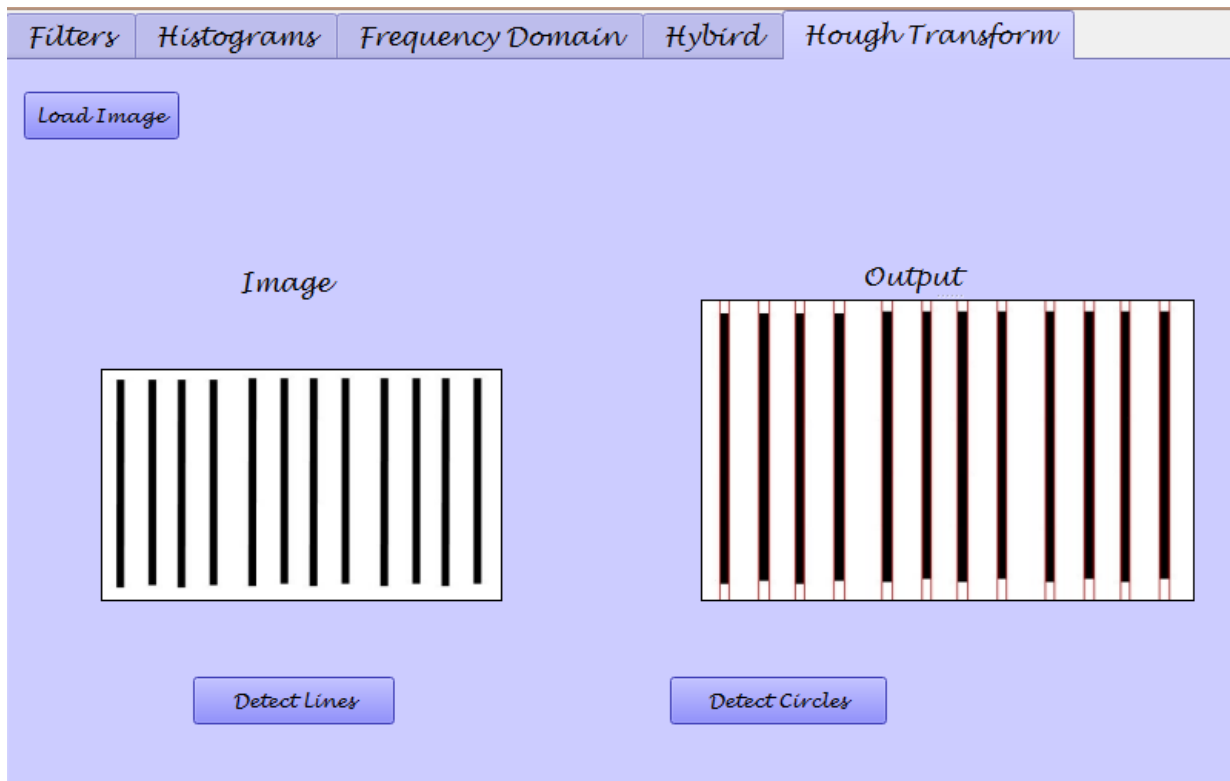
Amany Bahaa El-Din

Esraa Sayed

Irini Adel

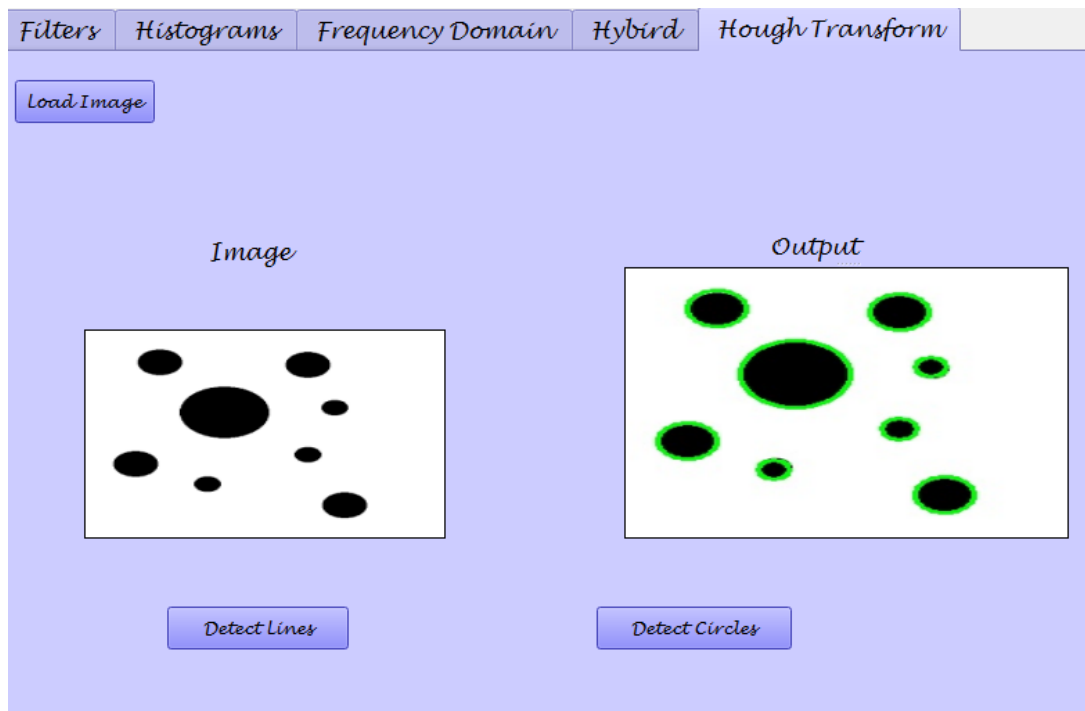
Hough Transform:

1. Line Detection with Hough Transform:

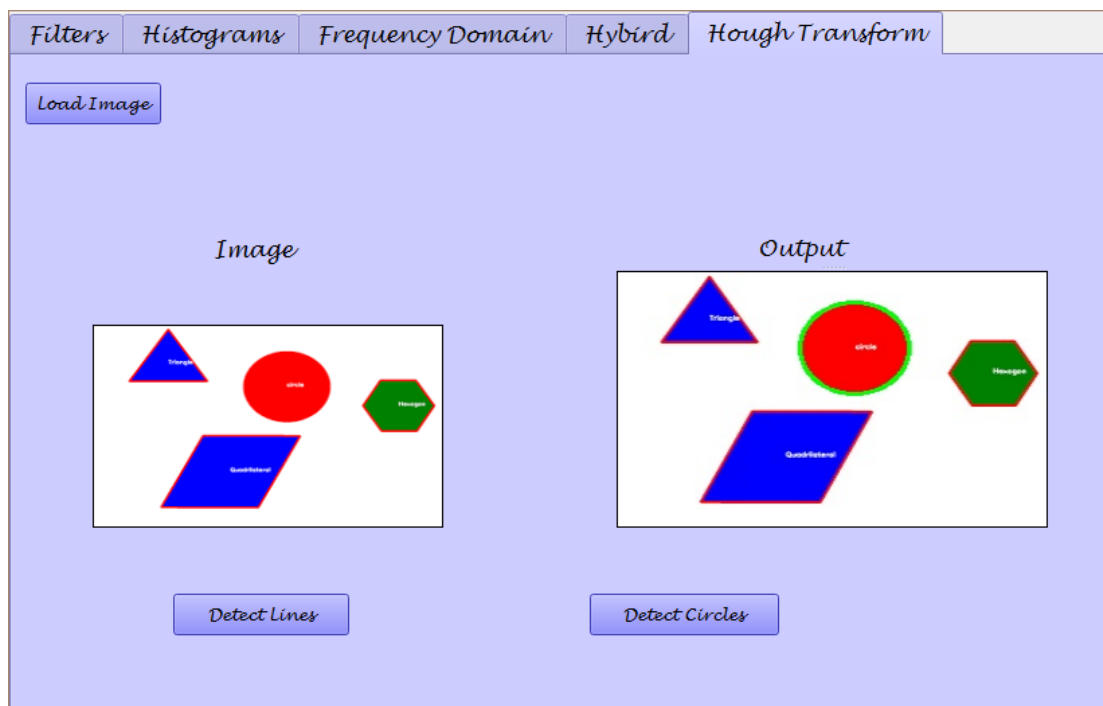


2. Circle Detection with Hough Transform:

- At minimum radius = 5



- At minimum radius = 10



How does line Detection with Hough transform work?

-The equation of the line in Cartesian coordinates is:

$$y = mx + b$$

Where **m** is the slope of the line, **b** is the intercept of the line with y axis.

But because the slope of any vertical line is ∞ and any horizontal line is 0, it would be better to use the polar form.

-The parametric/polar form:

$$\rho = x\cos\theta + y\sin\theta$$

Where **ρ** is the perpendicular distance from the origin to the line, **θ** is the angle formed by this perpendicular line and horizontal axis in counter clock-wise.

-How the algorithm works:

As we could represent any line in these 2 terms (ρ, θ) , which represent a point in the hough space.

1- The voting function

-We first create a 2D accumulator(array) initialized by zeros to hold the values of the parameters. the rows will hold **ρ** values and the columns will hold **θ** values.

-The accumulator would have accuracy of 1, so there would be 180 columns, and the number of rows would depend on max **ρ** which would be the maximum possible distance which is the diagonal length of the image.

-Using the edges given by the *Canny edge detector* and for each possible line, we loop over every pixel and calculate ρ and the radian angle, and add their votes in the accumulator, and add the values of the new votes to the existing values in the accumulator, this will end up with that the pixels which represent edges and lines will have maximum votes.

2-the non maximum suppression function:

we got the 8 nearest neighbor and find the maximum votes.

3-the inverse hough function:

- we get x, y index of hough space
- np.ravel Reduce the multidimensional array to 1 dimension
- np.argsort Sort the array elements from small to large and return the index
- reverse the order of the array from large to small using `[::-1]` for the first k values `[k]` .
- The value of k depends on the number of lines in the image. We used 24 for our used images as they have 12 lines.
- Finally we convert the image from hough space to image space.

How does Circle Detection with Hough transform work?

The parametric equation of a circle of **radius r** and **center (a, b)** .

$$\begin{cases} x = a + r\cos(t) \\ y = b + r\sin(t) \end{cases}$$

Where: $t \in [0, 2\pi)$

The set of all the possible circles is defined by all the possible values for a , b and r . And for each circle, the pixels that belong to the circle can be found by iterating over some possible values of t . To reduce the amount of circles to take into consideration, we will only consider values for r between r_{min} and r_{max} .

Using the edges given by the **Canny edge detector** and for each possible circle, we count the number of edges that are part of each circle. We iterate over the coordinates of edge pixels (x, y) and compute the coordinates of the center of all the circles that pass by that point. This is done using the equation above by setting r and t . For each of these circles, we increment an accumulator (a, b, r) . (In the code, it is in the form of a dictionary: Keys are the coordinates, and values are the counts)

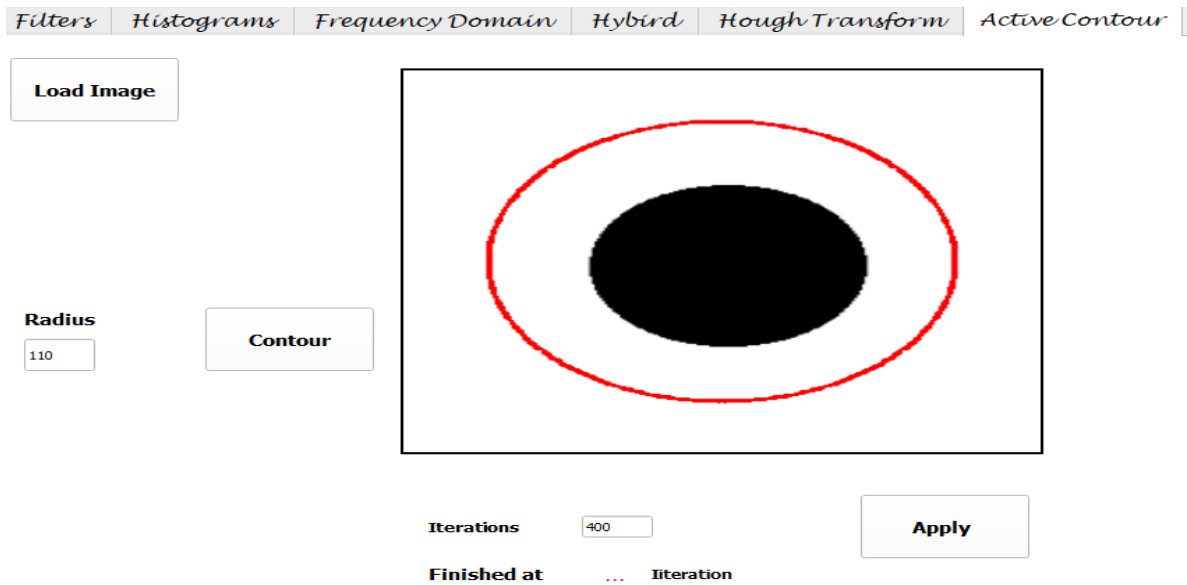
In order to select which circles are good enough, we use two criteria:

1. A threshold (at least 30% of the pixels of a circle must be detected)
NOTE: The threshold differs from an image to another (adjusted by try and error on 2 images)
2. We exclude circles that are too close of each other (Once a circle has been selected, we reject all the circles whose center is inside that circle).

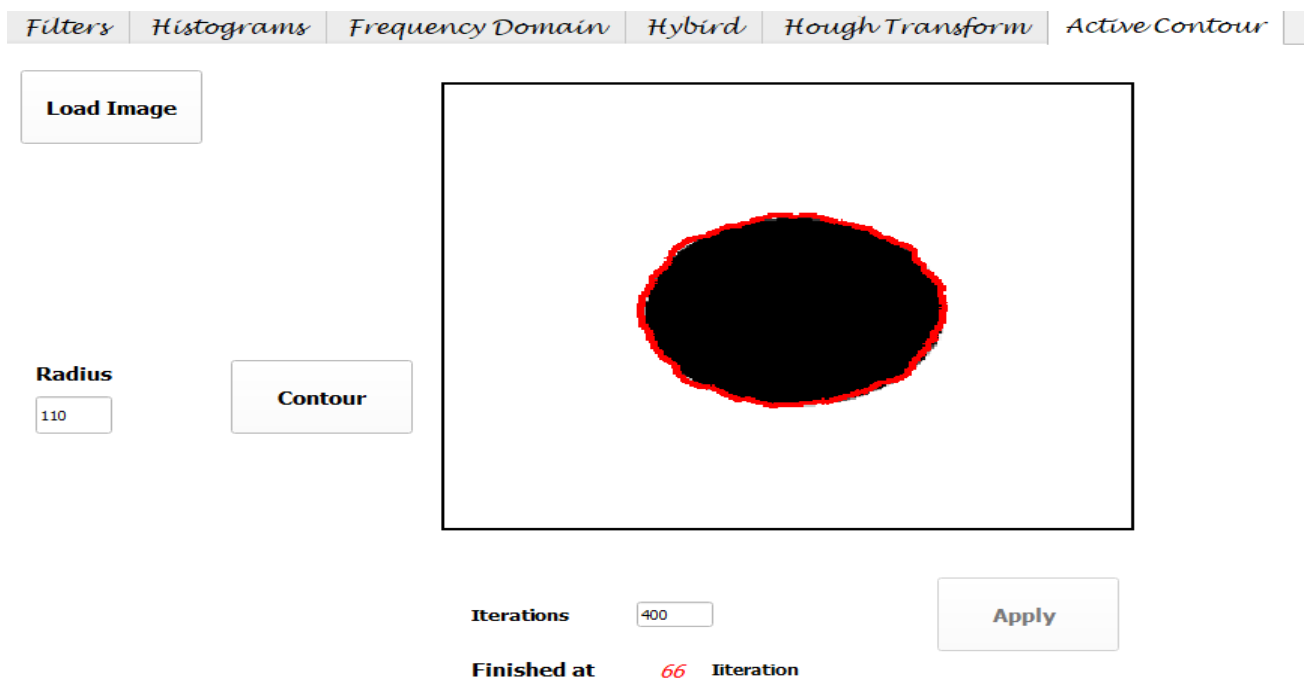
Active contour:

Result:

1) applying contour



Result



2)applying contour

Filters

Histograms

Frequency Domain

Hybird

Hough Transform

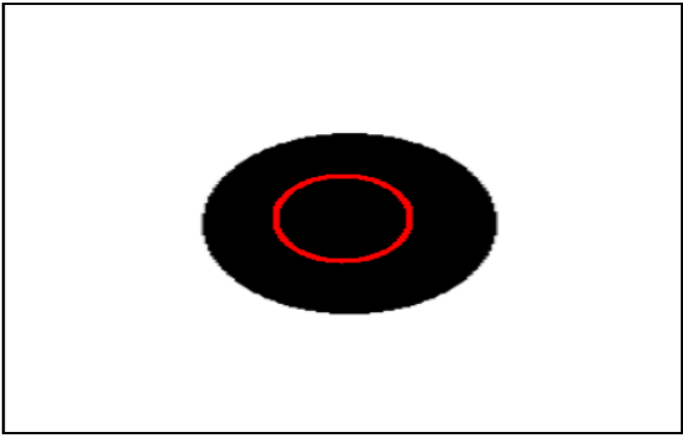
Active Contour

Load Image

Radius

30

Contour



Iterations

400

Finished at

...

Iteration

Apply

Result

Filters

Histograms

Frequency Domain

Hybird

Hough Transform

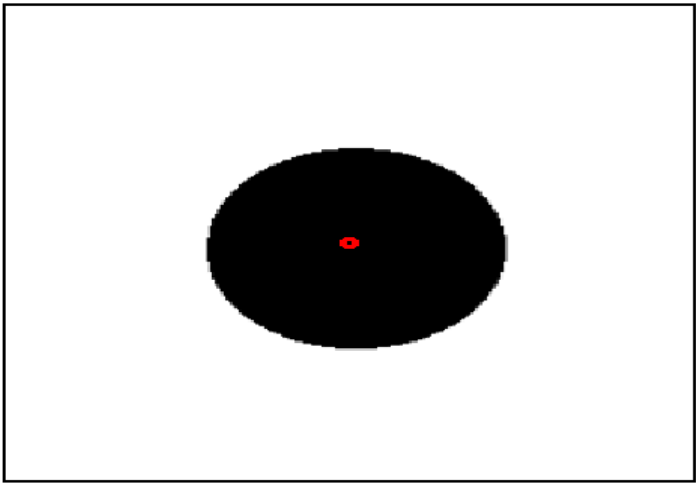
Active Contour

Load Image

Radius

30

Contour



Iterations

400

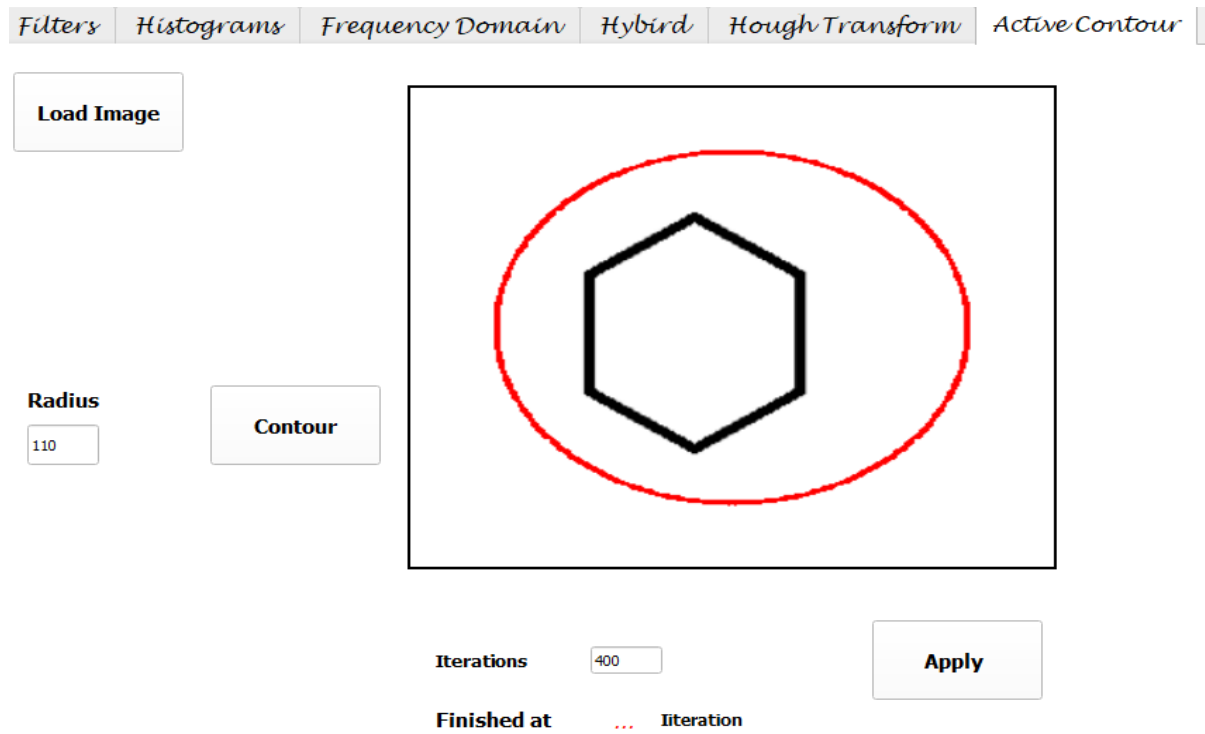
Finished at

99

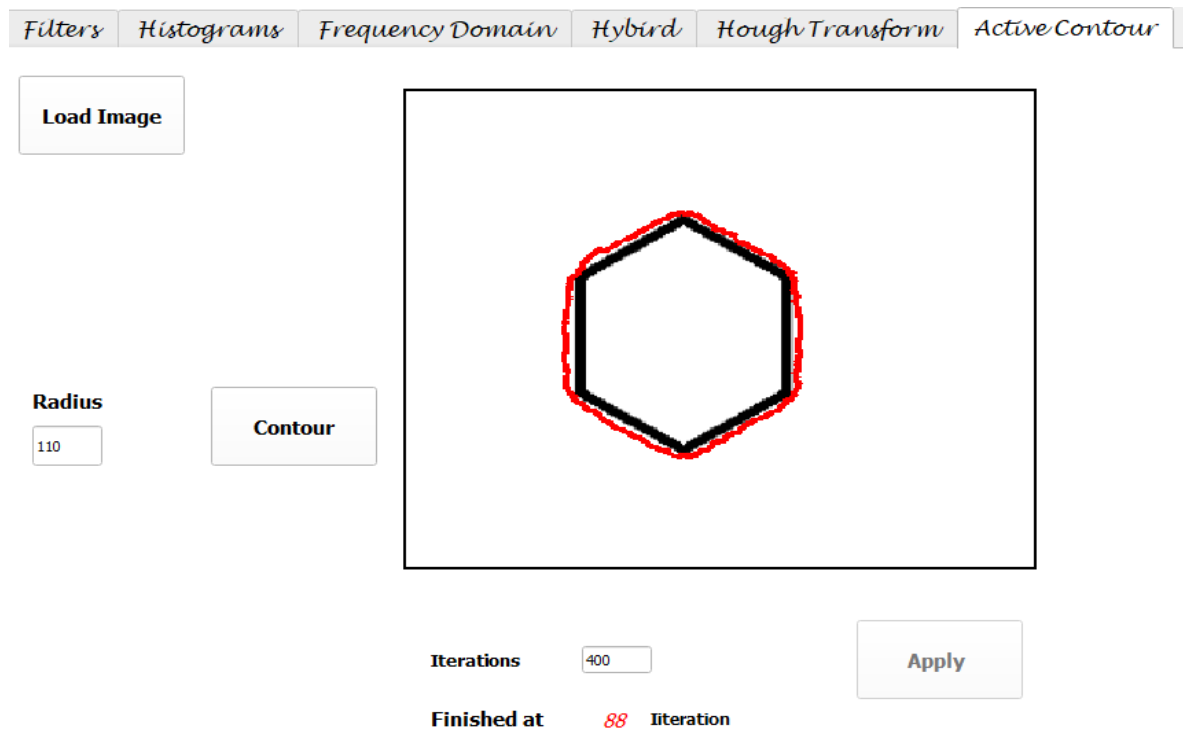
Iteration

Apply

3)applying contour



Result



4)applying contour

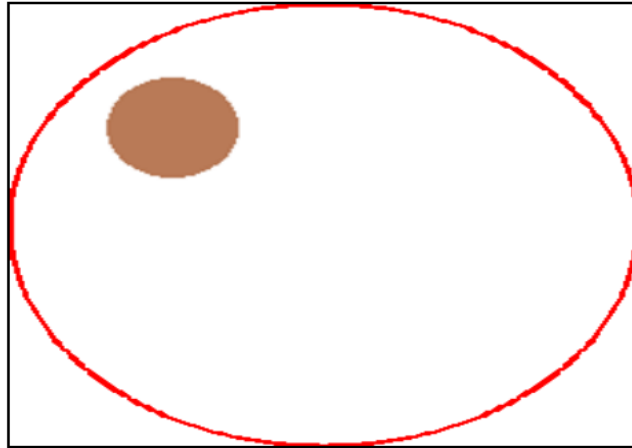
Filters *Histograms* *Frequency Domain* *Hybird* *Hough Transform* **Active Contour**

Load Image

Radius

150

Contour



Iterations

400

Apply

Finished at

...

Iteration

Result

Filters *Histograms* *Frequency Domain* *Hybird* *Hough Transform* **Active Contour**

Load Image

Radius

150

Contour



Iterations

400

Apply

Finished at

220

Iteration

5)applying contour

Filters

Histograms

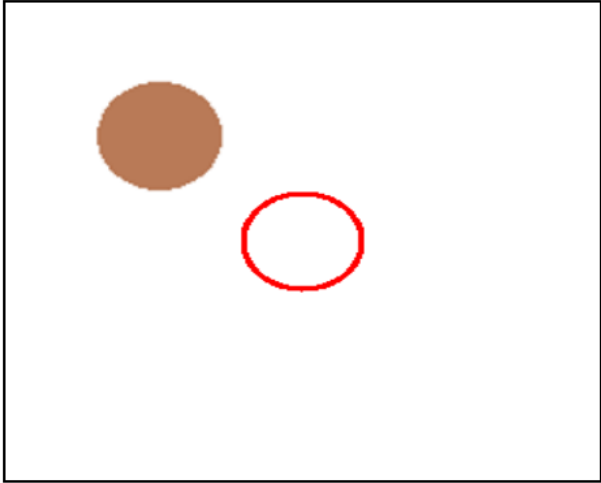
Frequency Domain

Hybird

Hough Transform

Active Contour

Load Image



Radius

30

Contour

Iterations

400

Finished at ... Iteration

Apply

Result

Filters

Histograms

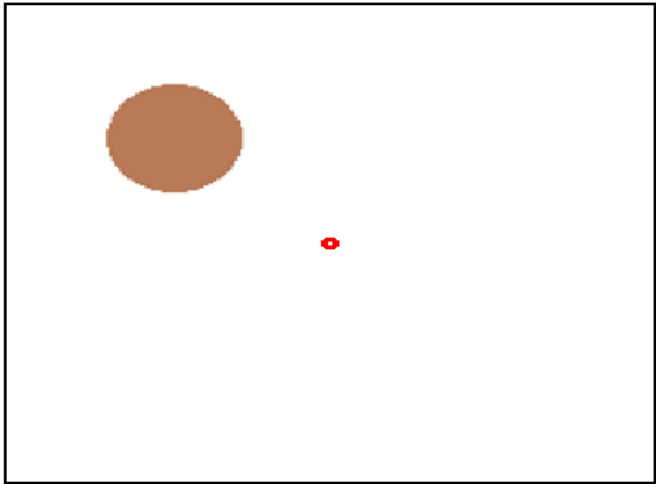
Frequency Domain

Hybird

Hough Transform

Active Contour

Load Image



Radius

30

Contour

Iterations

400

Finished at 99 Iteration

Apply

6)applying contour

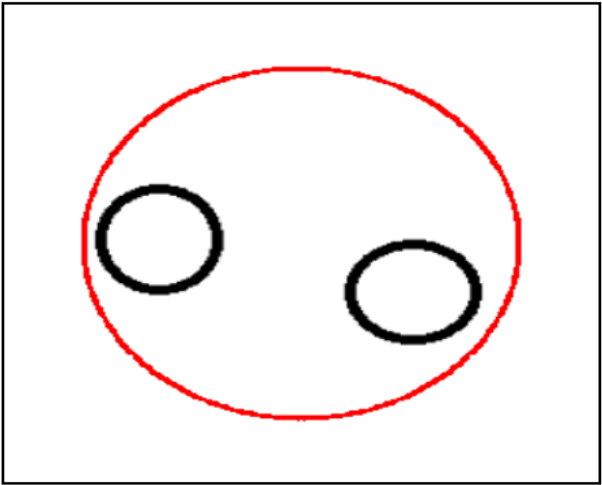
Filters Histograms Frequency Domain Hybrid Hough Transform Active Contour

Load Image

Radius

110

Contour



Iterations 400

Finished at 121 Iteration

Apply

Result

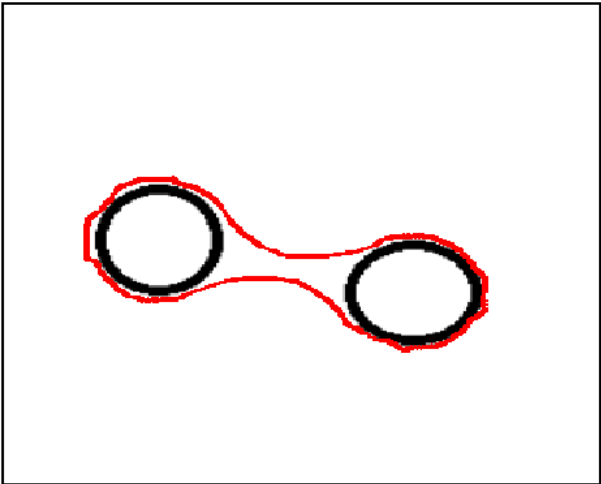
Filters Histograms Frequency Domain Hybrid Hough Transform Active Contour

Load Image

Radius

110

Contour



Iterations 400

Finished at 121 Iteration

Apply

7)applying contour

Filters

Histograms

Frequency Domain

Hybird

Hough Transform

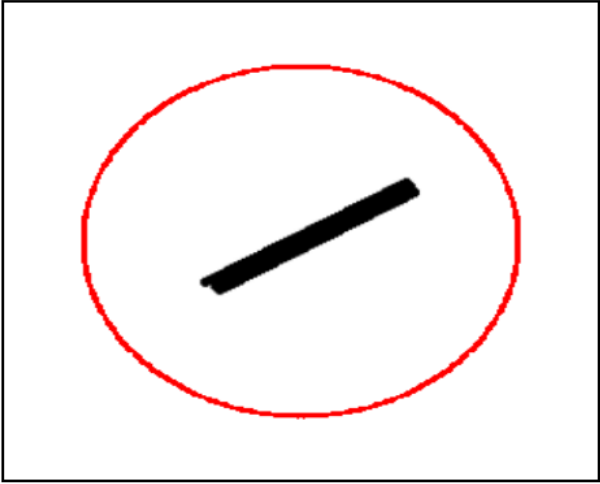
Active Contour

Load Image

Radius

110

Contour



Iterations

400

Finished at

...

Iteration

Apply

Result

Filters

Histograms

Frequency Domain

Hybird

Hough Transform

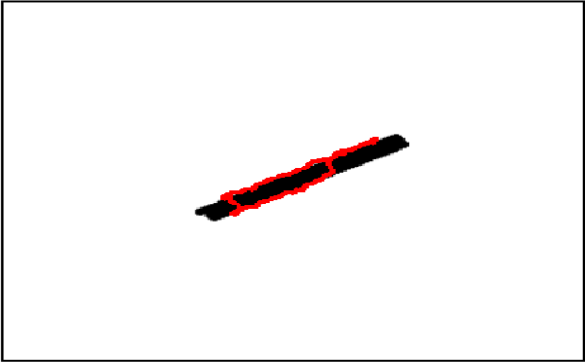
Active Contour

Load Image

Radius

110

Contour



Iterations

400

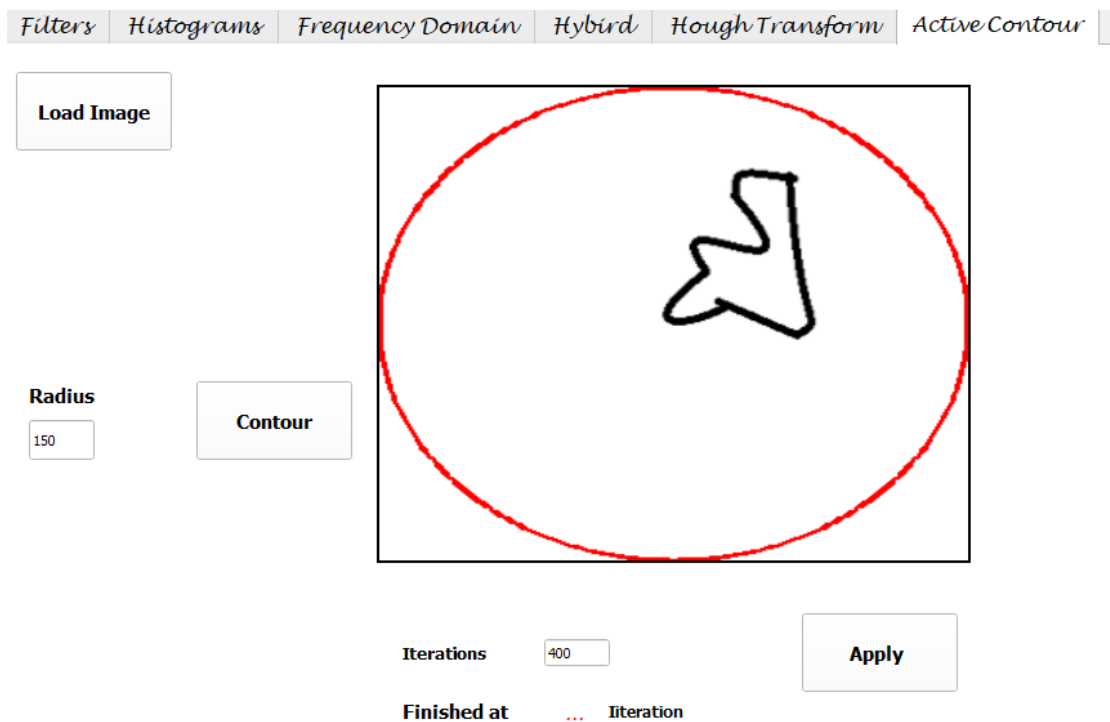
Finished at

400

Iteration

Apply

8)applying contour



Result

