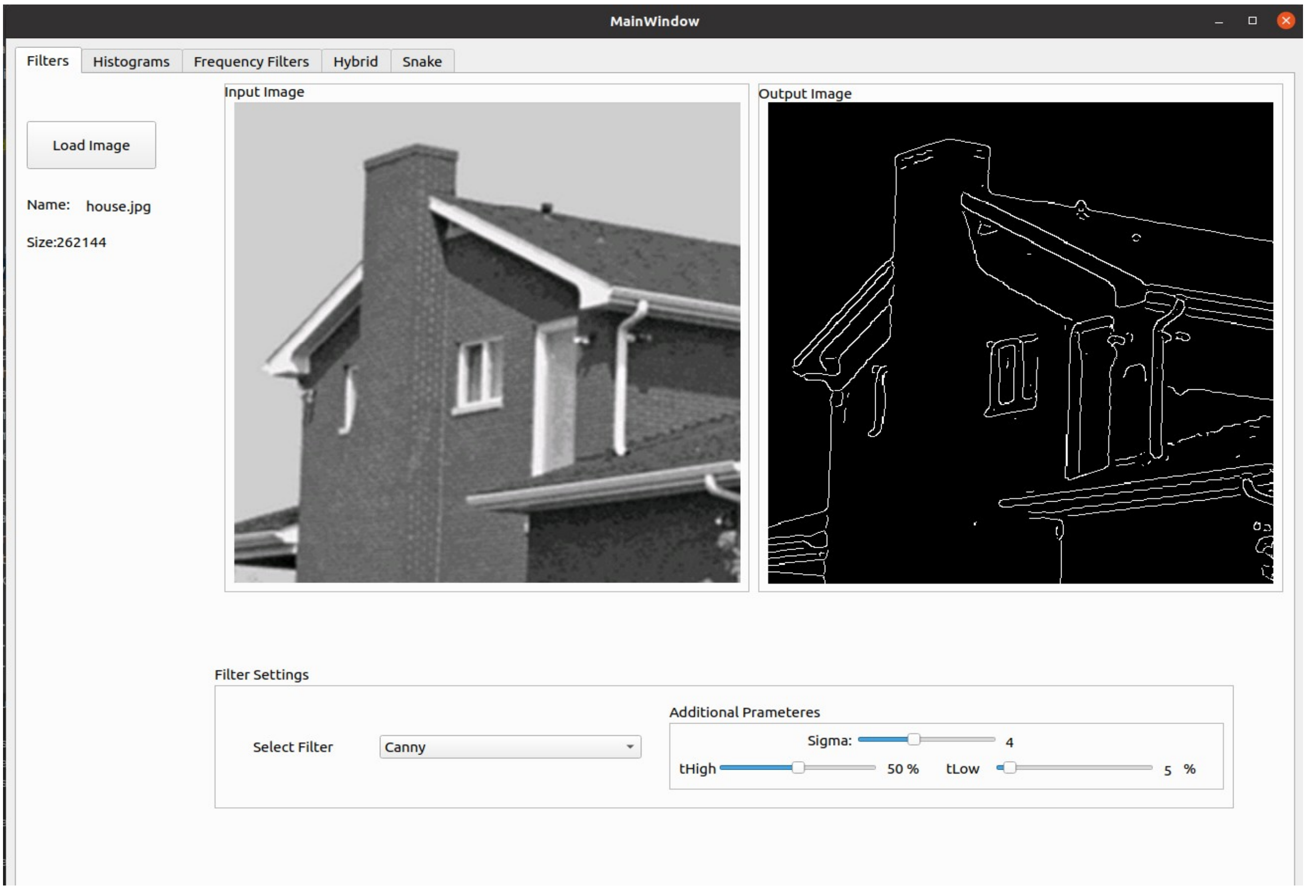
Task 2

➔ Canny edge detection

steps :

- Apply sobel filter in x & y direction to get magnitude and direction.
- Pass magnitude and direction to non-maximum suppression to get largest value in the gradient direction.
  [Function prototype] :
  Image cannyNonMaxSuppression(Image &mag, Image &dir);
  [Parameters] :
  mag → magnitude values of the image from sobel.
  dir → direction values of the image from sobel.
- Apply hysteresis threshold to convert image to binary image.
  [Function prototype] :
  Image cannyEdgeLink(int tHigh, int tLow, Image& image);
  [Parameters] :
  tHight → high threshold of the image.
  tLow → low threshold of the image.
  Image → image to apply threshold on.



➔ Hough line detection

steps :

- Check for image white pixels and loop through them to get possible lines passes by each point to form hough space and get votes for each line.

  [Function prototype] :

  ```
  HoughLineTransformData houghLineTransform(Image &bw, float
  thetaStep, float rohStep);
  ```

  [Parameters] :

  bw → output image after applying canny filter.
  thetaStep → theta resolution.
  rohStep → roh resolution.

  [Returns] :

  HoughLineTransformData which is struct contain votes, roh and theta arrays.

- Sort the votes array and gets specified number of peaks.

  [Function prototype] :

  ```
  std::vector<_Point> linePeaks(Image &houghImage, int peaksNum)
  ```

  [Parameters] :

  houghImage → votes array of hough space
  peaksNum → number of peaks to return

  [Returns] :

  vector of peaks

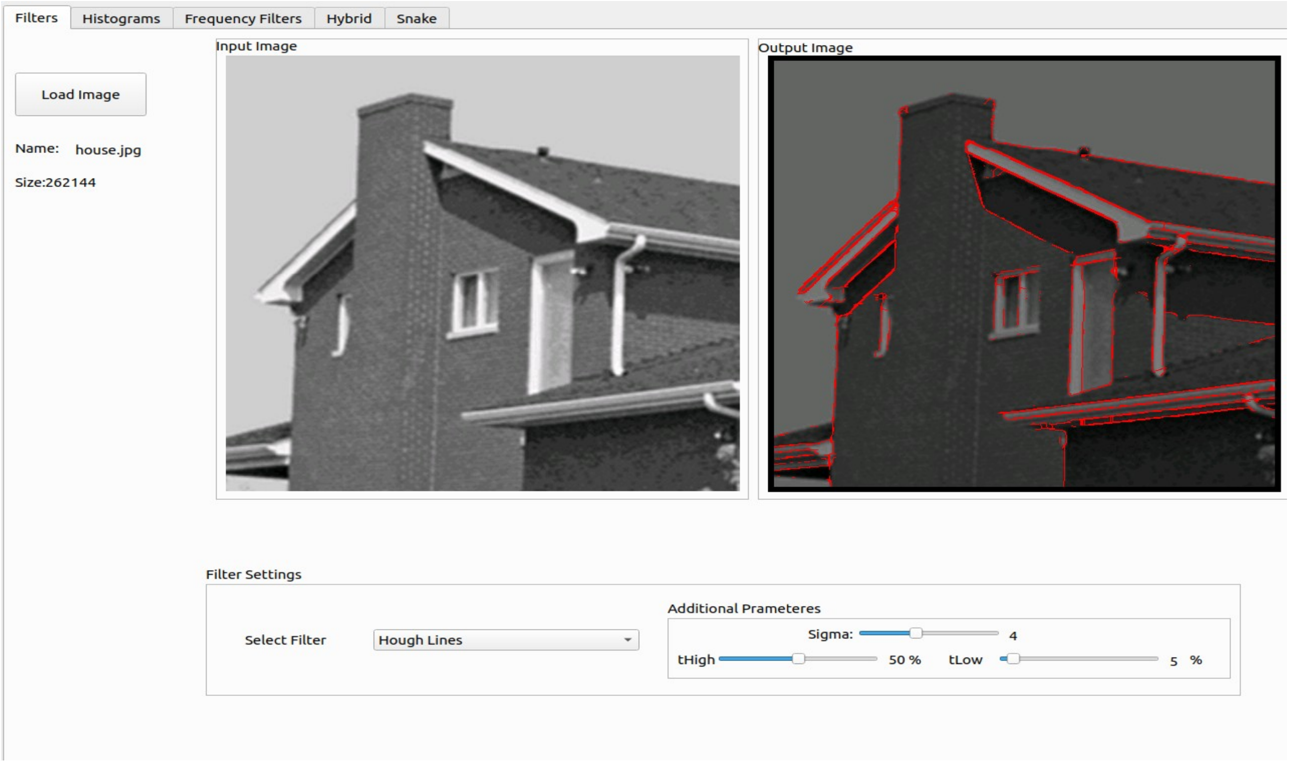
- Get lines from previous step and check for lines if true or not.

  [Function prototype] :

  ```
  std::vector<std::vector<_Point>> houghLines(Image &bw,
  std::vector<_Point> &peaks, std::vector<double> &thetaV,
  std::vector<double> &rohV, int maxGap);
  ```

  [Parameters] :

  bw → output image after applying canny filter.
  Peaks → peaks from the previous step.
  thetaV → array of theta of hough space.
  rohV → array of roh of hough space.
  maxGap → maximum distance allowed between two points on same line.



➔ Hough circle detection

steps :

- Check for image white pixels and loop through them to get possible circles passes by each point for each radius to form hough space and get votes for each circle.
- Then check votes which exceed threshold for reach radius.
- Then use binary image to get the true circles.

[Function prototype] :

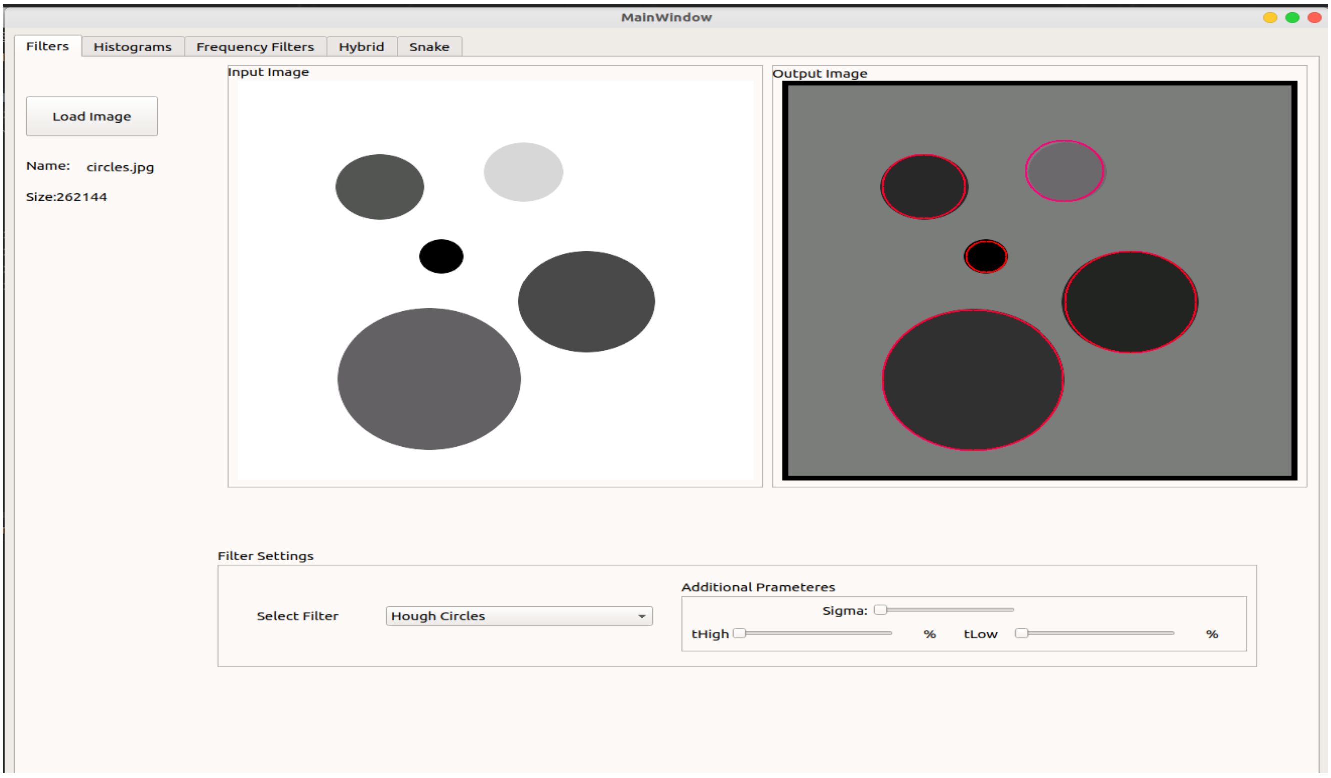
Std::vector<Circle> houghCircles(Image &bw, std::pair<int, int> rohRange, int threshold)

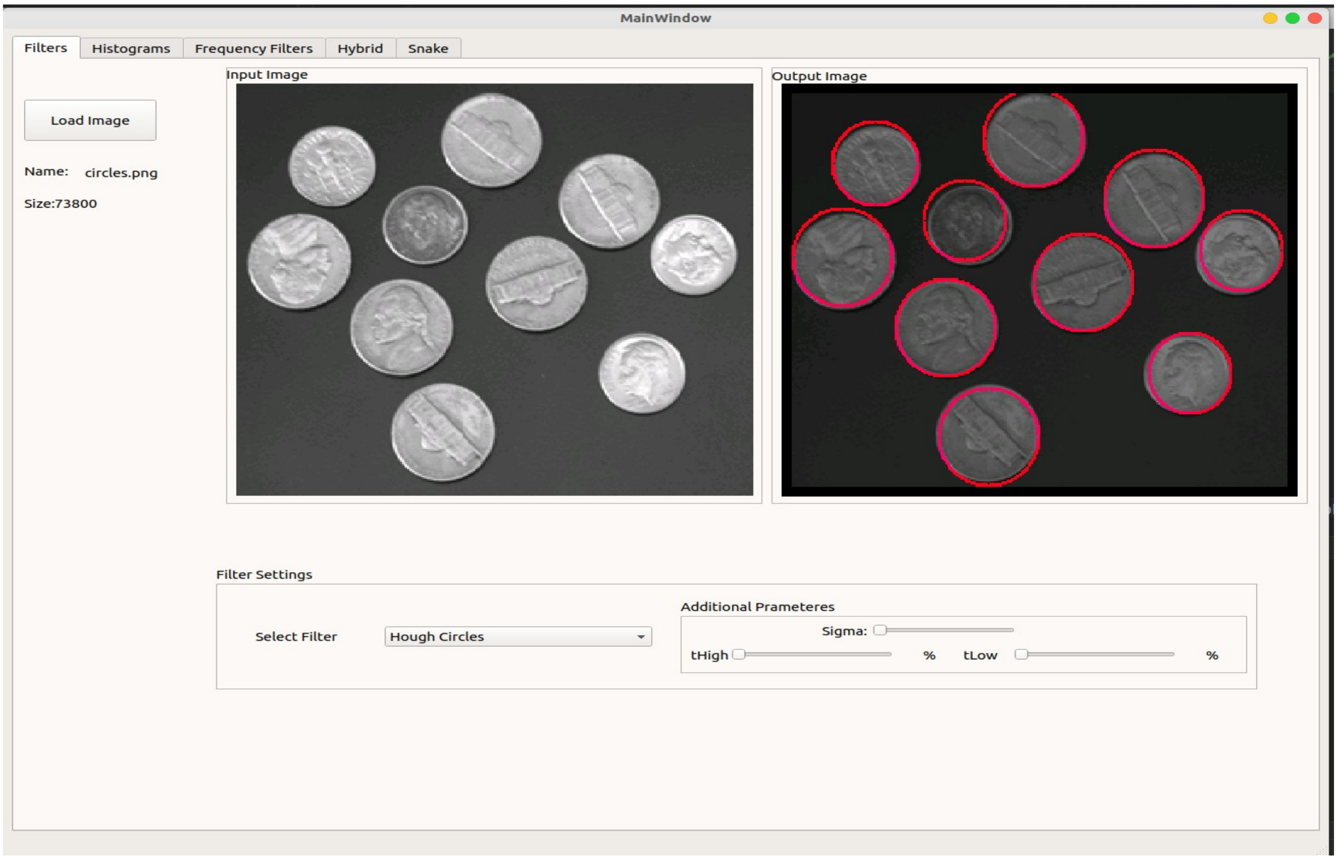
[Parameters] :

bw → binary image

rohRange → range of circle radius

threshold → minimum acceptable number of votes for a circle


MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
Load Image
Name: circles.jpg
Size:262144
Input Image
Output Image
Filter Settings
Select Filter
Hough Circles
Additional Prameteres
Sigma:
tHigh
%
tLow
%


MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
Load Image
Name: circles.png
Size:73800
Input Image
Output Image
Filter Settings
Select Filter
Hough Circles
Additional Prameteres
Sigma:
tHigh
%
tLow
%

Active Contour (Sanke) :

we use greedy algorithm snake energy calculated :

$E_{snake}(x, y) = \alpha(s_i)E_{ela}(x, y) + \beta(s_i)E_{curv}(x, y) + \gamma(s_i)E_{img}(x, y)$,

Where $E_{ela}(x, y)$ is the elasticity energy, $E_{curv}(x, y)$ is the curvature energy, $E_{img}(x, y)$ is the image energy and (x, y) are the indices to the points in the neighborhood and (s) is the parametric representation of curve

- **$E_{ela}(x, y)$:**

$E_{ela}(x, y) = d - \| v(s_i) – v(s_i -1)\|$,

Where where (d) is the average distance between all the points in the snake, and $v(s_i)$ is the point under study and

$v(s_i -1)$ is the previous point on snake.

- **$E_{curv}(x, y)$:**

$E_{curv}(x, y) = \|v(s_i +1) - 2v(s_i) + v(s_i -1)\|^2$.

- **$E_{img}(x, y)$:**

$E_{img} = \|\nabla[G\sigma (x, y) * I(x, y)]\|^2$

Where ∇ is the gradient , G σ (x, y) represent guassian filter , I(x,y) represent the image.

Functions used :

in file → greedy_snake.cpp & greedy_snake.h :

- **currentPrevNextPointIndex() :**

  get the index of previous and next points index

- **averageDistance() :**

  get the average distance between all snake points

- i**mageEnergy() :**

  get the image enaergy

- **greedySnake() :**

  function that represent the actual snake steps

Note :

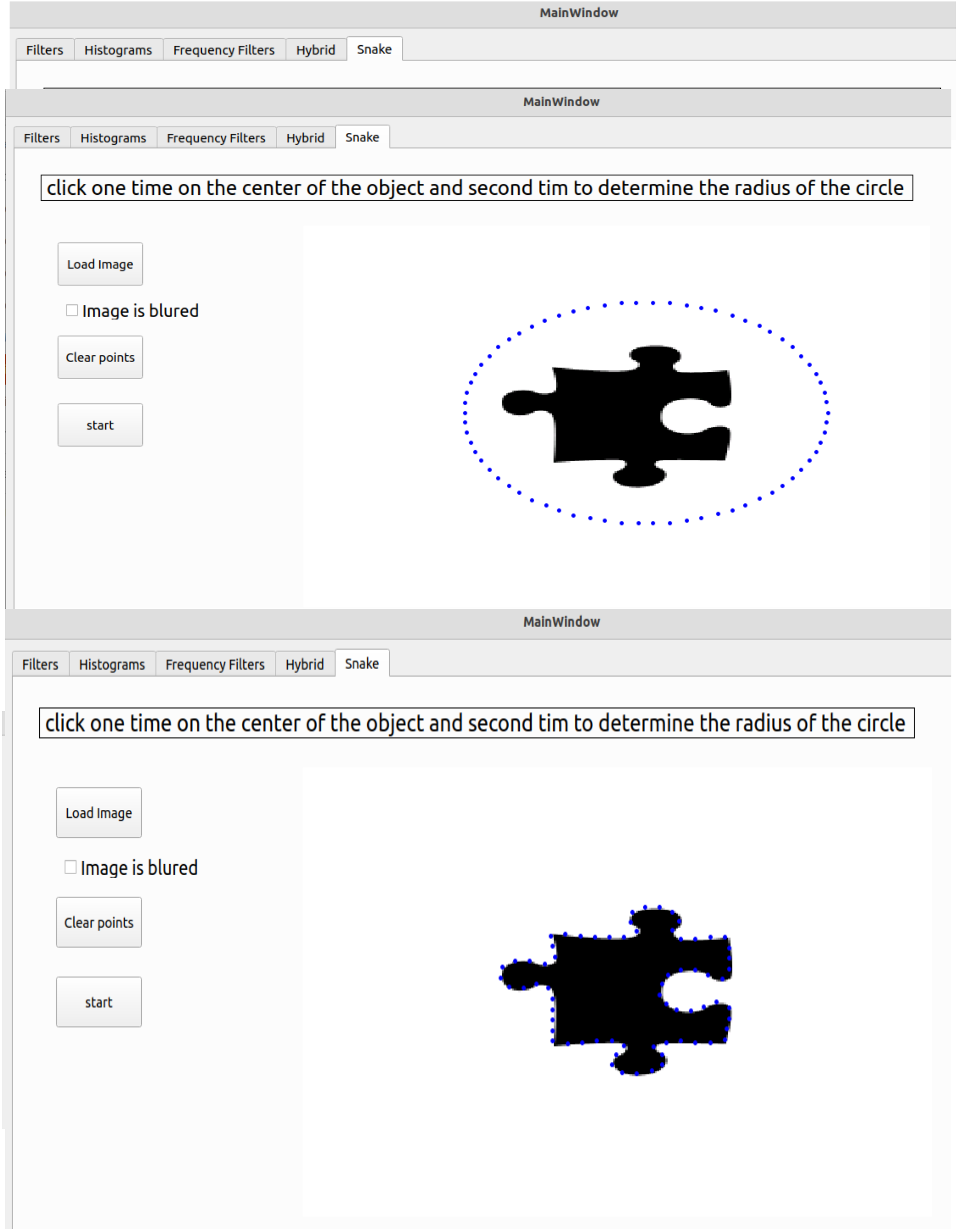
we use scale space continuation : first use large sigma for Gaussian filter then reduce it after that

Drawing contour (circle) in two steps :

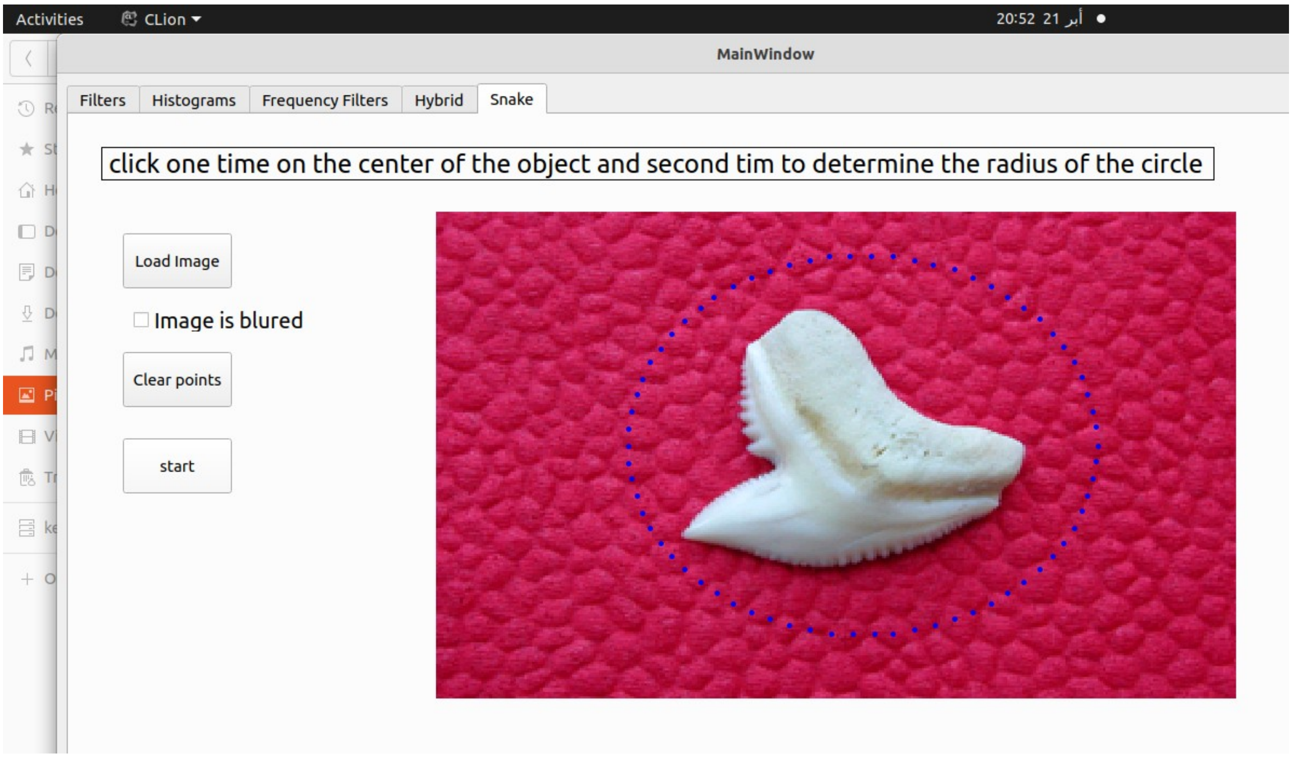
- click with mouse to determine the location of the center :
  preferred to click on center of object
- click with mouse to represent the radius of circle :
  outside the object

Output :

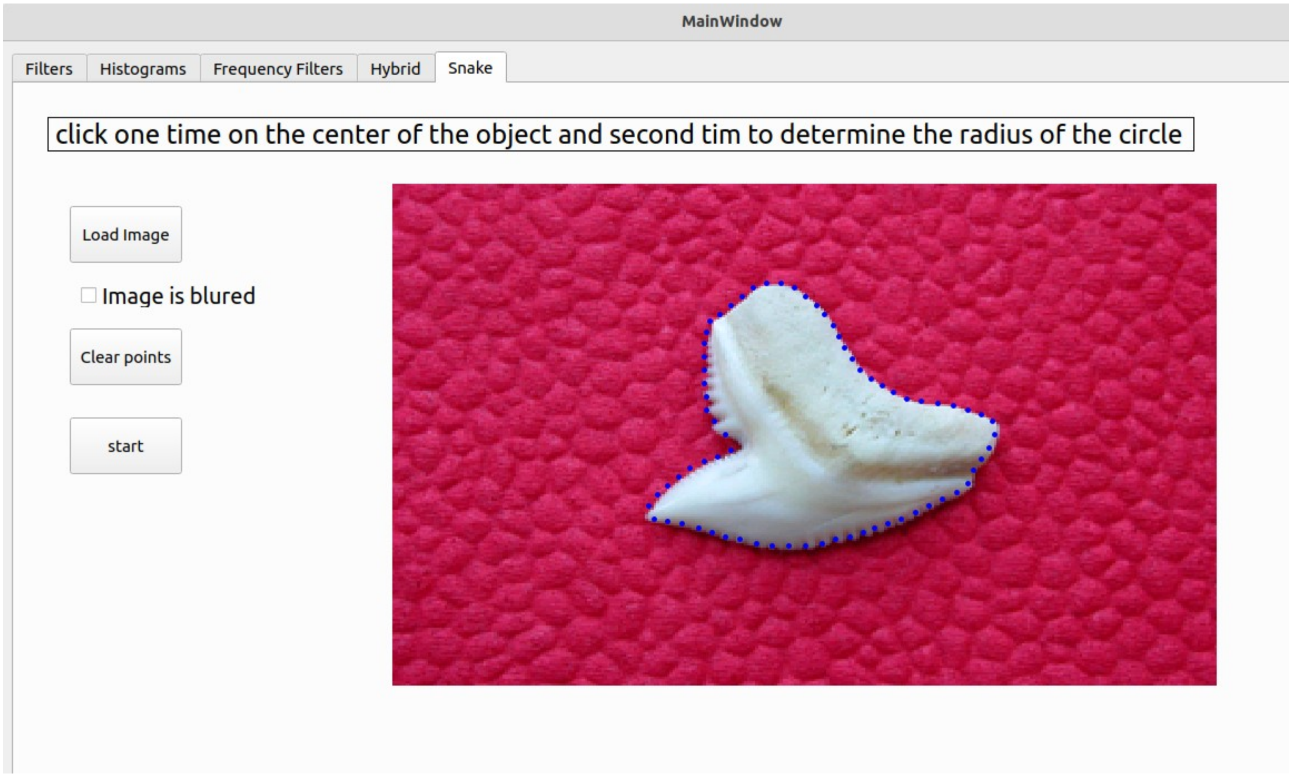
first test :


MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
click one time on the center of the object and second tim to determine the radius of the circle
Load Image
Image is blured
Clear points
start
MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
click one time on the center of the object and second tim to determine the radius of the circle
Load Image
Image is blured
Clear points
start

second test:



Activities
CLion
20:52 21 أبر
MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
click one time on the center of the object and second tim to determine the radius of the circle
Load Image
Image is blured
Clear points
start



MainWindow
Filters
Histograms
Frequency Filters
Hybrid
Snake
click one time on the center of the object and second tim to determine the radius of the circle
Load Image
Image is blured
Clear points
start