1. Briefly discuss and compare the implemented edge detector with the LoG method.

LoG

- LoG edge detector uses only one kernel.

$$\nabla^2 G_\sigma(x,y) = \left(\frac{1}{2\pi\sigma^4}\right)\left[\frac{x^2+y^2}{\sigma^2} - 2\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{(continuous LoG)}$$

$$K[i,j] = \left(\frac{1}{2\pi\sigma^4}\right)\left[\frac{(i-k-1)^2+(j-k-1)^2}{\sigma^2} - 2\right] e^{-\frac{(i-k-1)^2+(j-k-1)^2}{2\sigma^2}} \quad \text{(Discrete LoG)}$$

- The dimension of the kernel is $(2k+1)\times(2k+1)$. The variance σ^2 will determine the dimension of the kernel.
- π is usually ignored in computing the LoG coefficients.

Canny

- Smoothens the image with a Gaussian filter to reduce noise.
- Computes gradient using the gradient kernel operators.
- The orientation of gradients "theta = arctan(Gy/Gx)" is also computed from the vertical and horizontal gradients.

$$|\nabla f(x,y)| = \sqrt{f_x^2 + f_y^2} = \text{rate of change of } f(x,y)$$

$$\angle \nabla f(x,y) = \tan^{-1}\left(\frac{f_y}{f_x}\right) = \text{orientation of change of } f(x,y)$$

LoG

- LoG can be very sensitive to noise, and it is mostly affected by noise along the edges.
- A threshold parameter is used to find out the significant difference between the negative and positive values.
- The zero crossing is a significant parameter in LoG to find the edges of the objects.

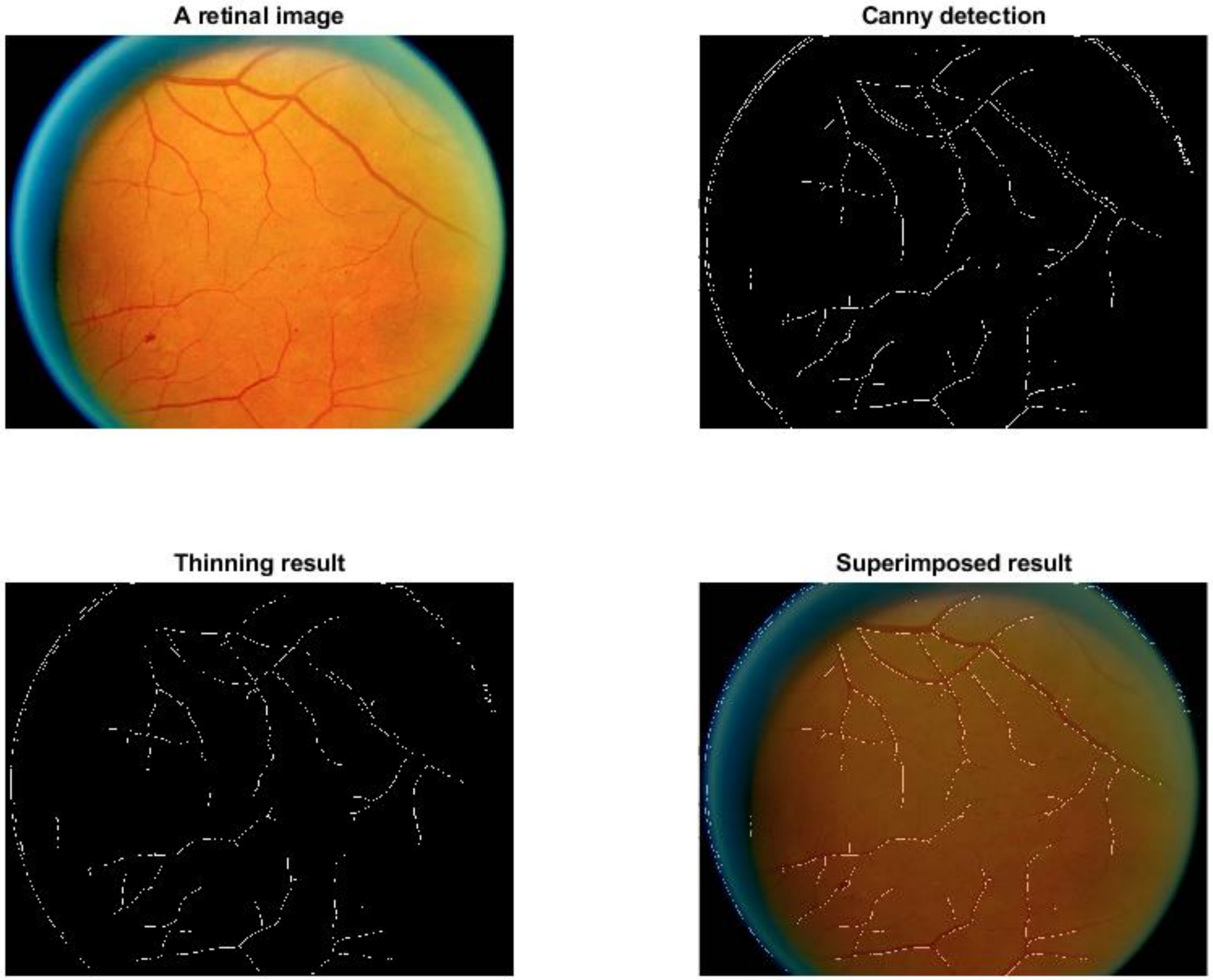
Canny

- π is used to find the gradient orientation.
- In canny, At each pixel location there are four possible directions. Perpendicular pixel values are compared with the value in the edge direction. If their value is lower than the pixel on the edge, then they are suppressed.
- High and low thresholds are used to fix the broken pixels in canny after the no maximum suppression.

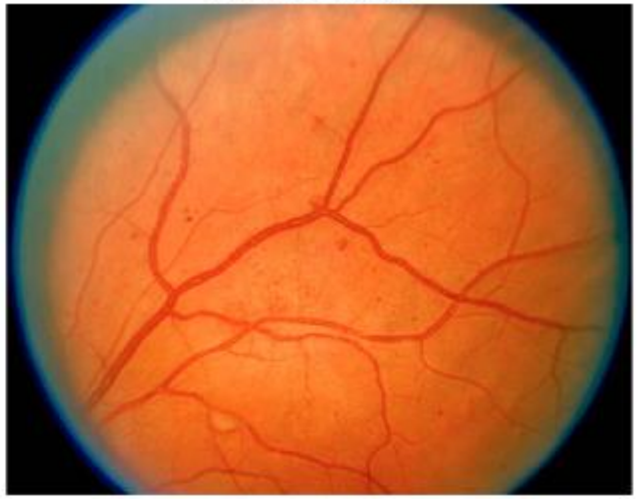
2. The source codes for the method implemented should be attached.

- The source code is attached in the zip folder.

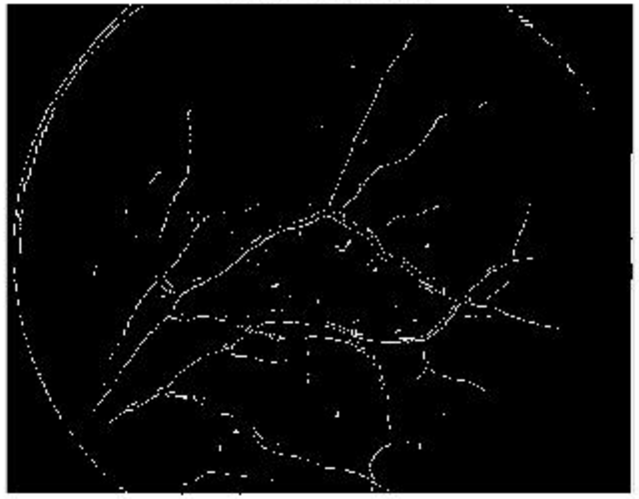
3. Demonstrate and discuss your experimental results for all four retinal images. Try to optimize the parameters to produce good edge detection results. Some examples are shown below.



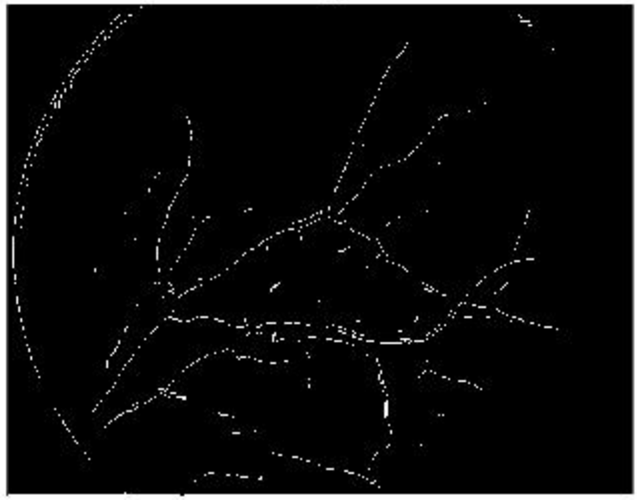
A retinal image



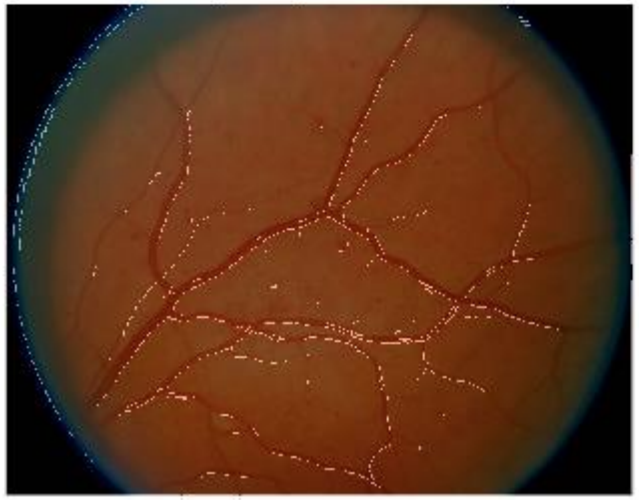
Canny detection



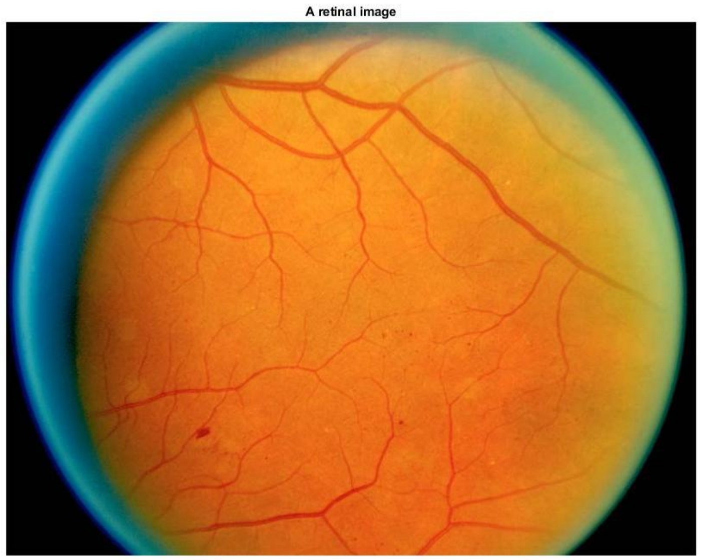
Thinning result

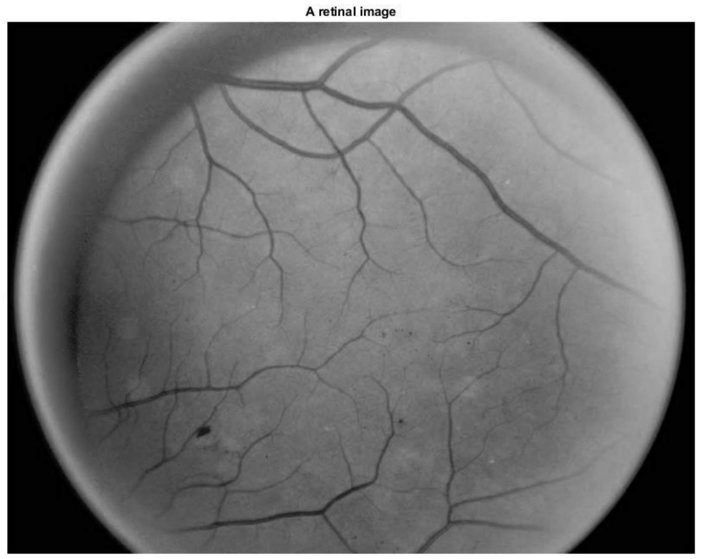


Superimposed result

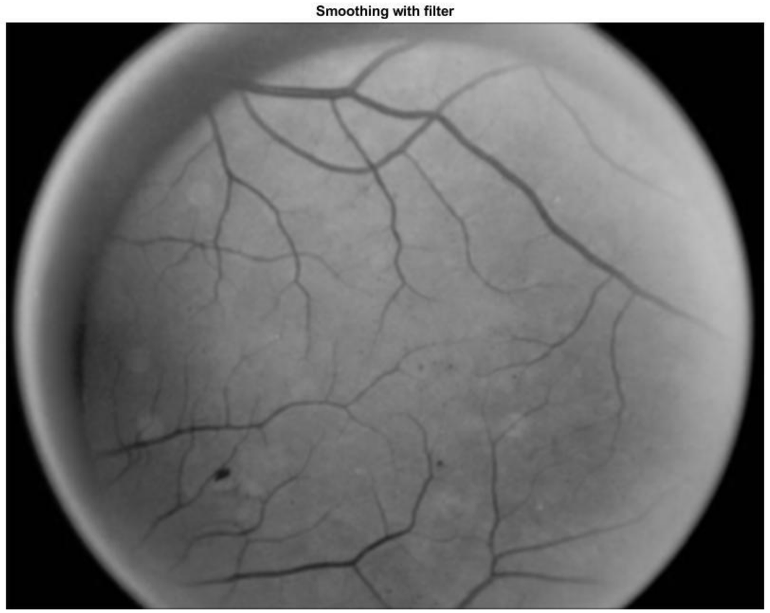


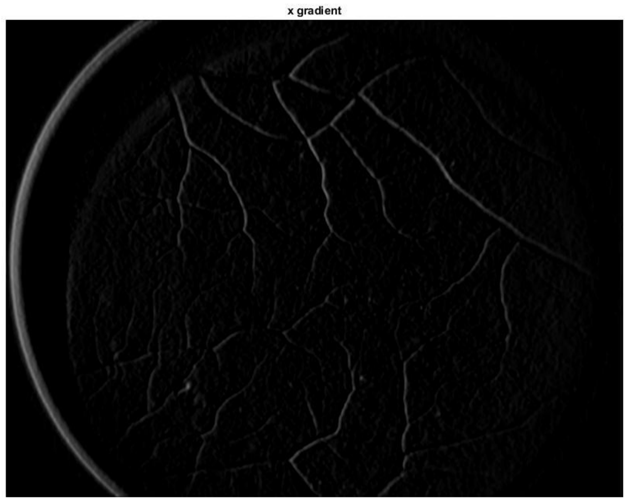
- For discussion let's consider the first retinal image as the outputs follow the same pattern for different images.
- First, we need to load the image and take the grayscale image for computation. Since the image is in RGB, we need to convert it to binary. Taking the 2nd channel aka the green channel also works.

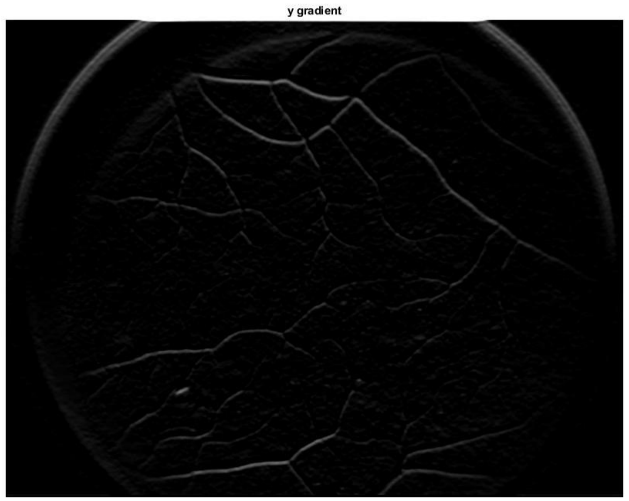




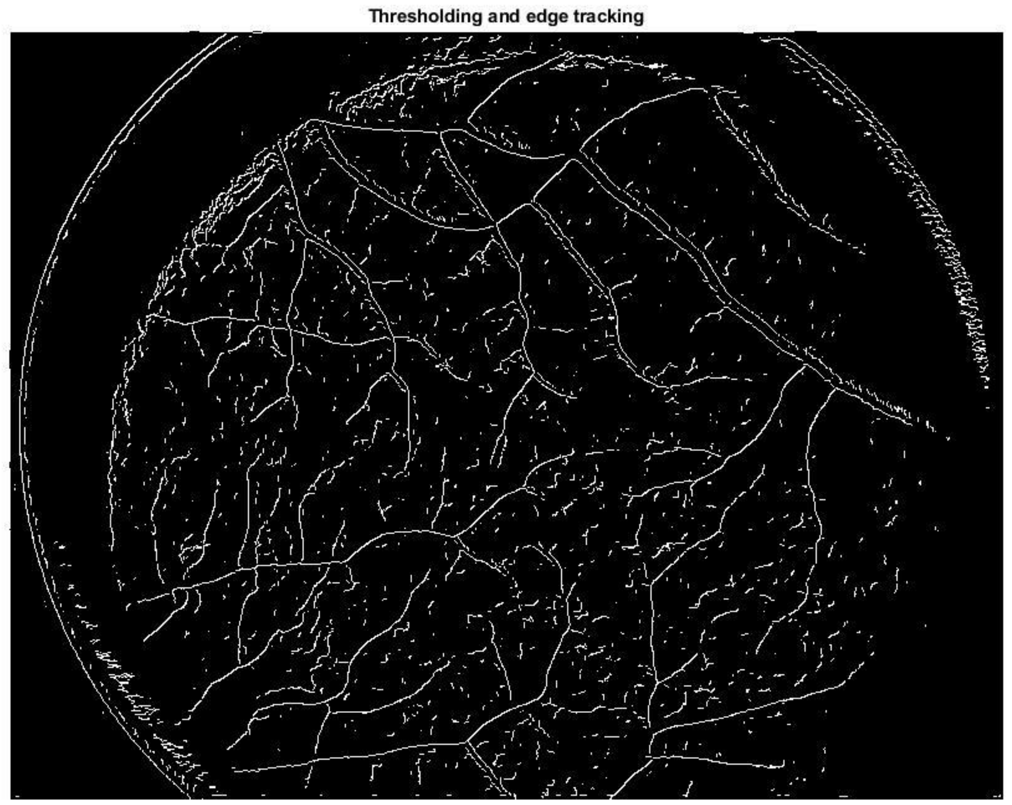
- Next, we apply gaussian filter (aka Sobel filter) to smoothen the image.
- On the smoothen image we apply vertical and horizontal kernels to find the x and y gradients.



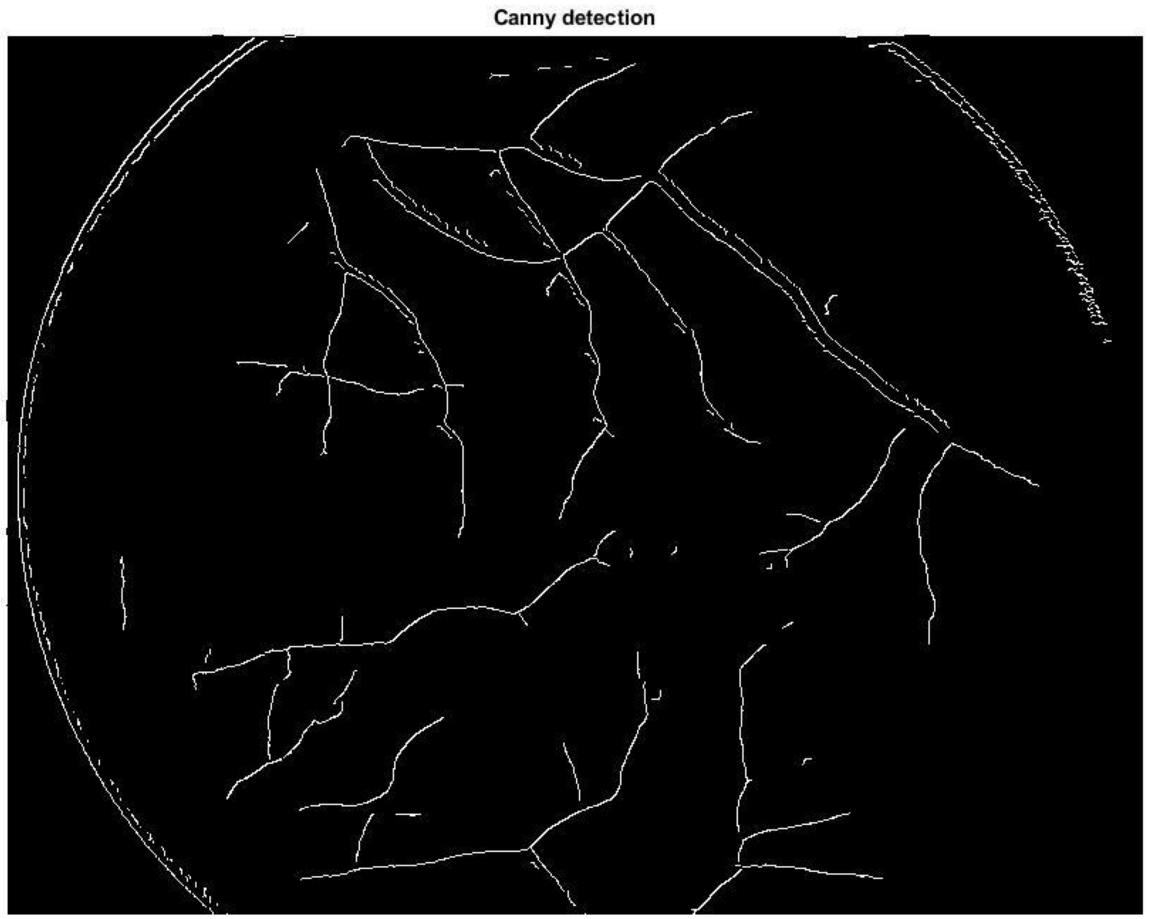




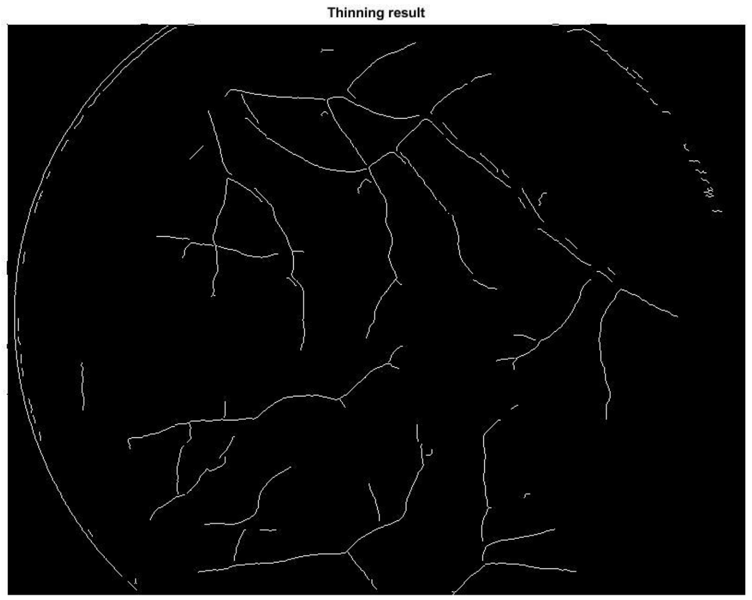
- In the next step we find the magnitude and angle to determine the non maximum suppression.
- Later we define the high and low threshold for the edge tracking.
- If its less than the low threshold we set that to 0 and if its less than high threshold we set that to the low threshold label. This technique can draw a boundary line on the edges and remove unnecessary backgrounds.

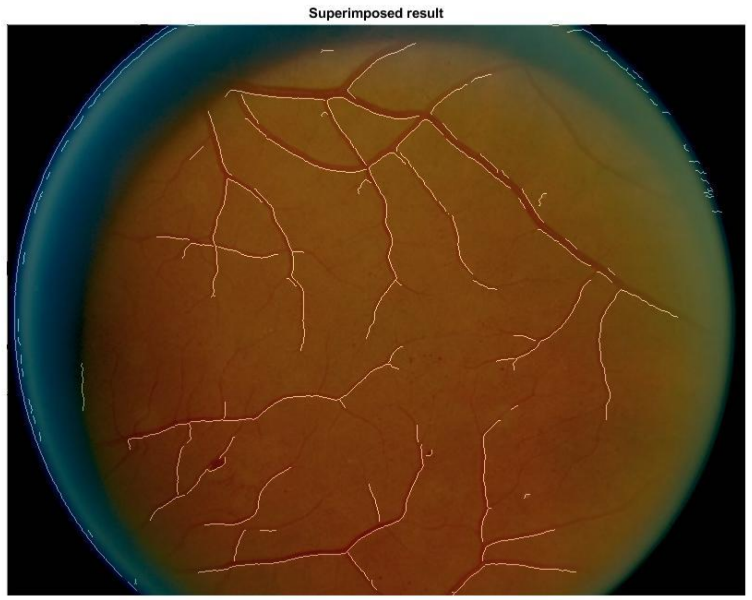


- However, there are still so many clutters and background noises in the images that needs to be removed in order to achieve canny filter.
- So we set all the low threshold values to high threshold values and ignore the rest. This enhances the edge lines and remove unnecessary clutters.



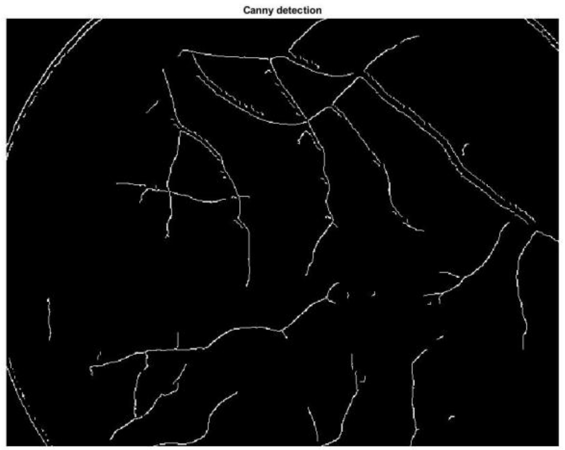
- There are still some background pixels in the image which needs to be removed. Also, the edges are uniform in thickness that needs to be thinned.
- So, the image is converted to binary label and connected pixels are calculated. From there we remove the small number of pixels.
- Lastly, we overlap the images in a superimposed manner to see the nonvisible blood vessels in the original image.

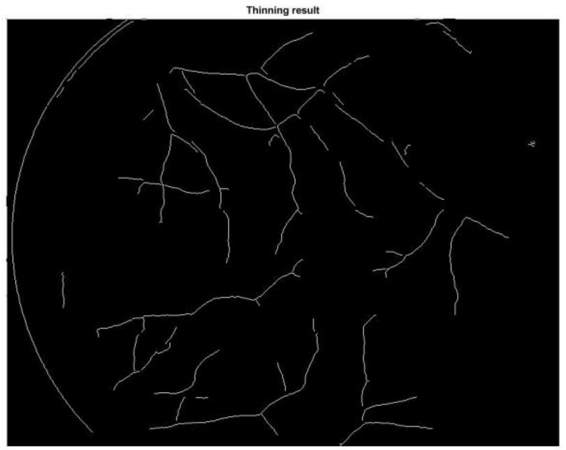




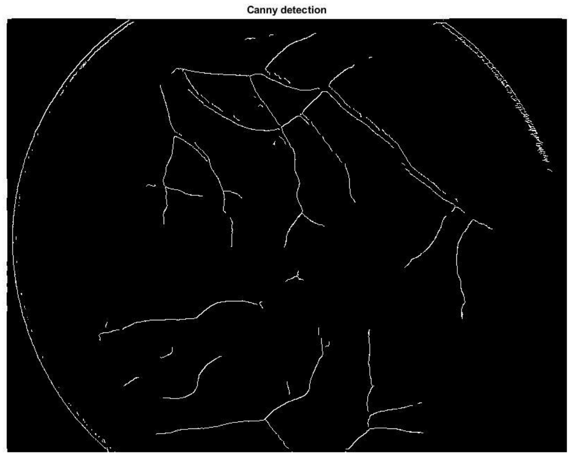
Tuning the parameters:

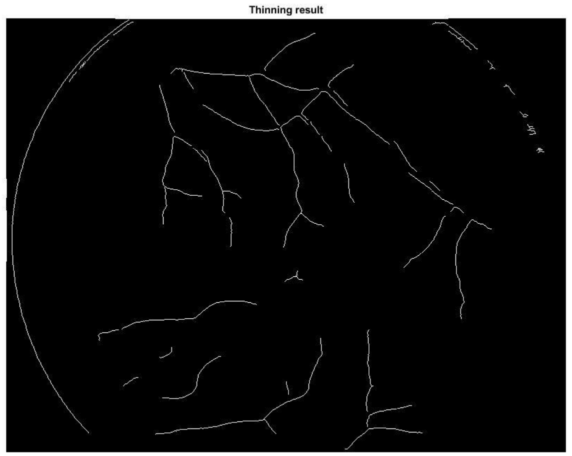
High threshold = 0.1, Low threshold = 0.5, Kernel size = 5, pixel threshold = 10.





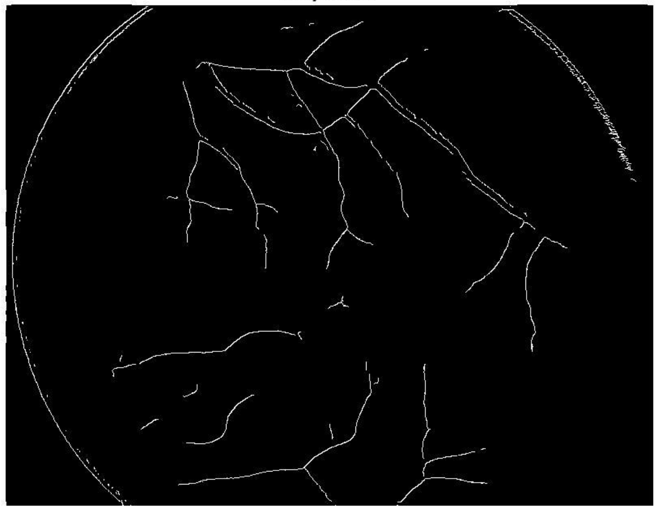
High threshold = 0.1, Low threshold = 0.5, Kernel size = 5, pixel threshold = 20.



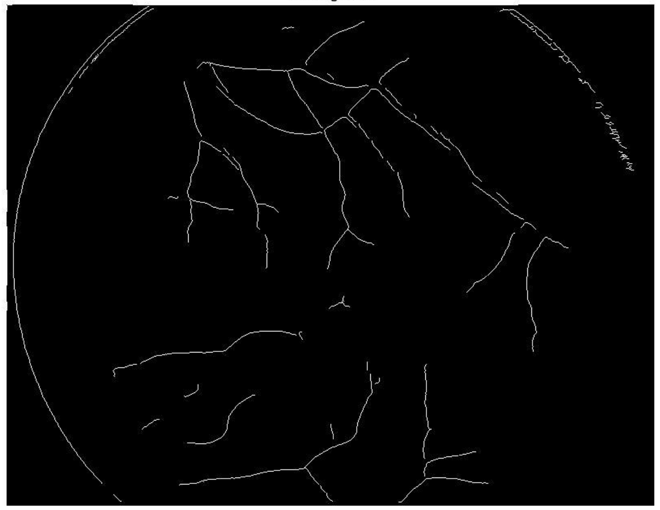


High threshold = 0.1, Low threshold = 0.5, Kernel size = 7, pixel threshold = 10.

Canny detection

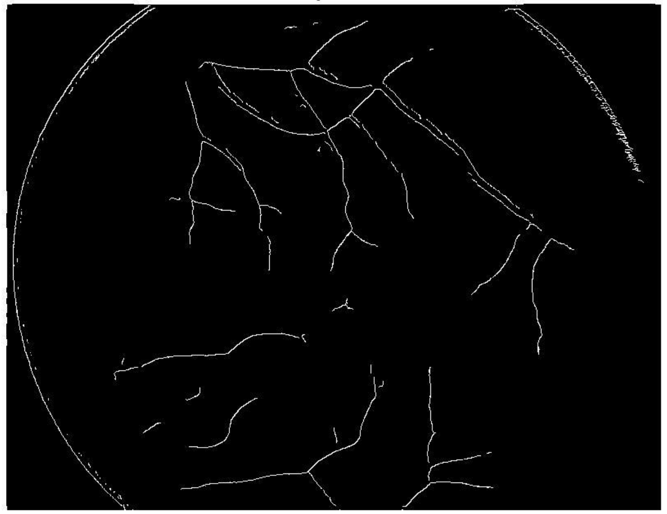


Thinning result

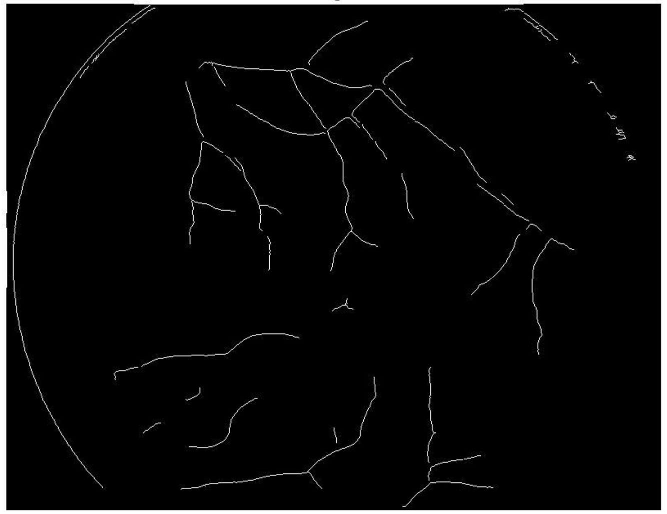


High threshold = 0.1, Low threshold = 0.5, Kernel size = 7, pixel threshold = 20.

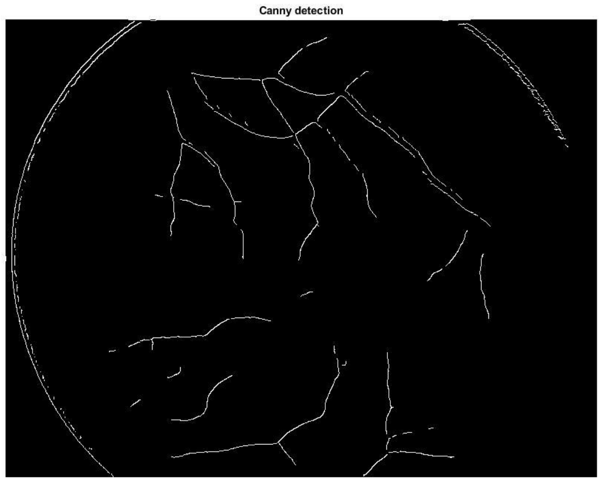
Canny detection

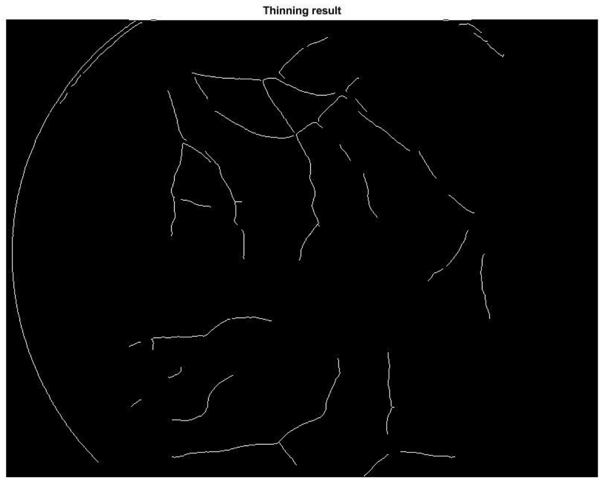


Thinning result

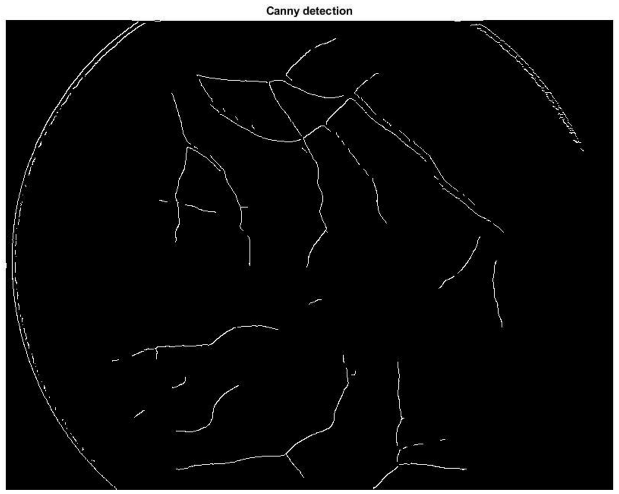


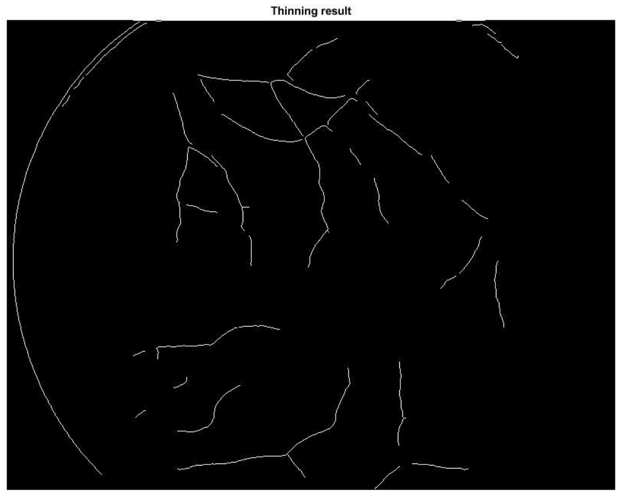
High threshold = 0.2, Low threshold = 0.1, Kernel size = 5, pixel threshold = 10.



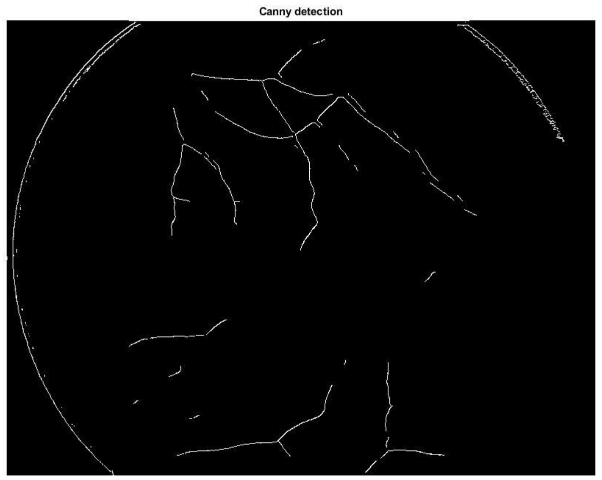


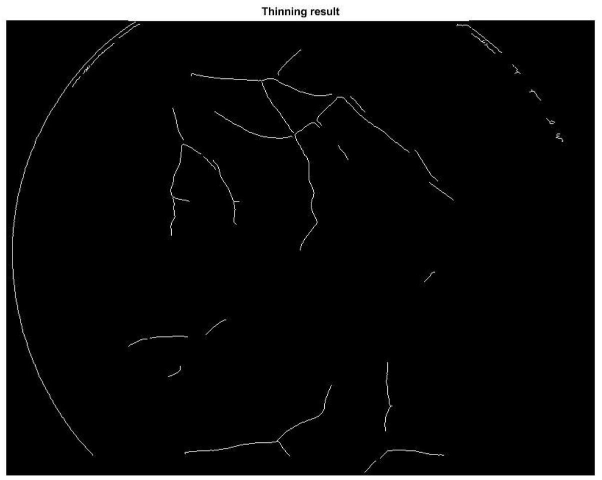
High threshold = 0.2, Low threshold = 0.1, Kernel size = 5, pixel threshold = 20.



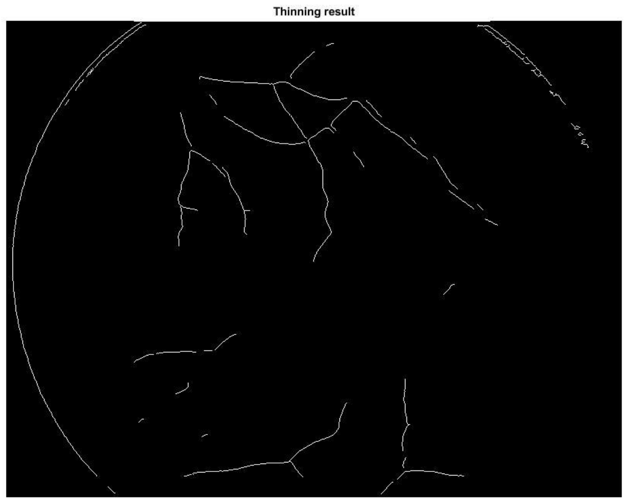


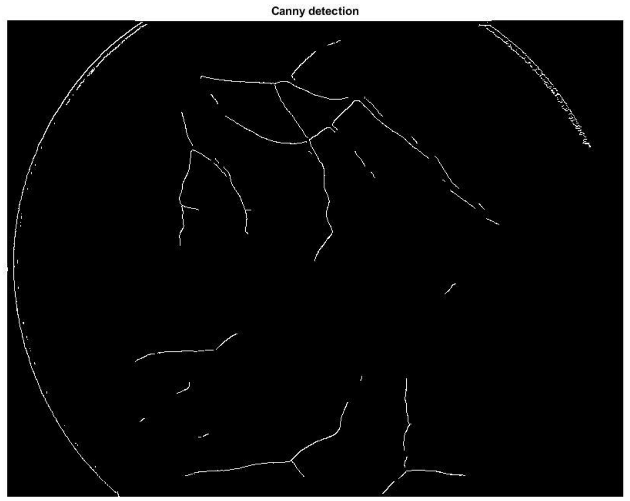
High threshold = 0.2, Low threshold = 0.1, Kernel size = 7, pixel threshold = 10.





High threshold = 0.2, Low threshold = 0.1, Kernel size = 7, pixel threshold = 20.





Findings from the parameters:

- Keeping high threshold will bring out more edges but along with more noise.
- A low threshold will even filter out useful pixels.
- Low threshold is important to filter out small cluttered pixels.
- If pixel threshold is set high, more edges are thinned which creates broken lines. Also, if it’s low, less noise is filtered out.
- The kernel size hardly makes any difference.