ECEN 5283 Project 2: Camera Calibration

Alexander Rose

Oklahoma State University

3/2/2024

Objective

- For this project, two retina images were provided and edge detection methods were implemented to find the blood vessels on the images.
- A comparison of edge detection methods between Canny, Laplace of Gaussian (LoG), and Matched filtering with length filtering (LF) is shown.
- A variety of parameters must be tested to find the best edge detection results

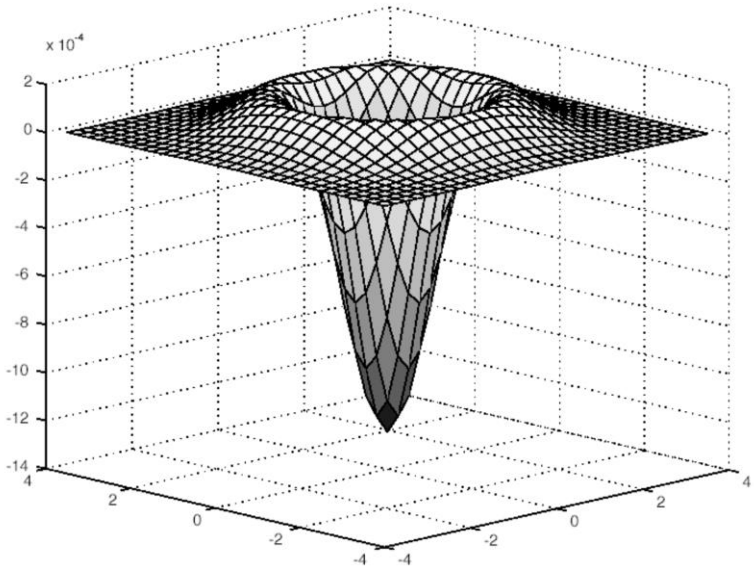
Technical Background

- Edge detection is important for finding features and borders in images
  - These can range from the retina blood vessels in this project, to animals or other objects
- The noise effect of an image must be reduced with pre-filtering
- Canny (1986) established critical criteria in selecting an edge detection filter:
  - Signal to Noise Ratio must be high, that is, stronger response to edge than noise
  - Edge Localization: The filter response should be strong at $x = 0$ and not elsewhere
  - Low False Positives: There must only be one maximum within the neighborhood
- From these criteria, three prominent edge detection filters emerged
  - Canny, LoG, and Matched Filtering

Technical Background

- The Laplace of Gaussian (LoG) is a Laplace function with a Gaussian pre-filter, shown below.

$$\frac{d^2 f}{dx^2} = f(x) - 2f(x-1) + f(x-2)$$



- The Canny method utilizes the gradient of two first order derivatives corresponding to the rows and columns of an image.
  - Compute the first order derivatives of the rows and columns
  - Compute the magnitude and direction of the gradient
  - Apply non-maximum suppression to the gradient magnitude, finding maximum along the gradient direction
  - Weak edges can be linked after implementing a connectivity analysis.

Technical Background – Canny continued

Relevant equations

- $M(x,y)$ and $\alpha(x,y)$ represent the magnitude and direction of the gradient respectively.

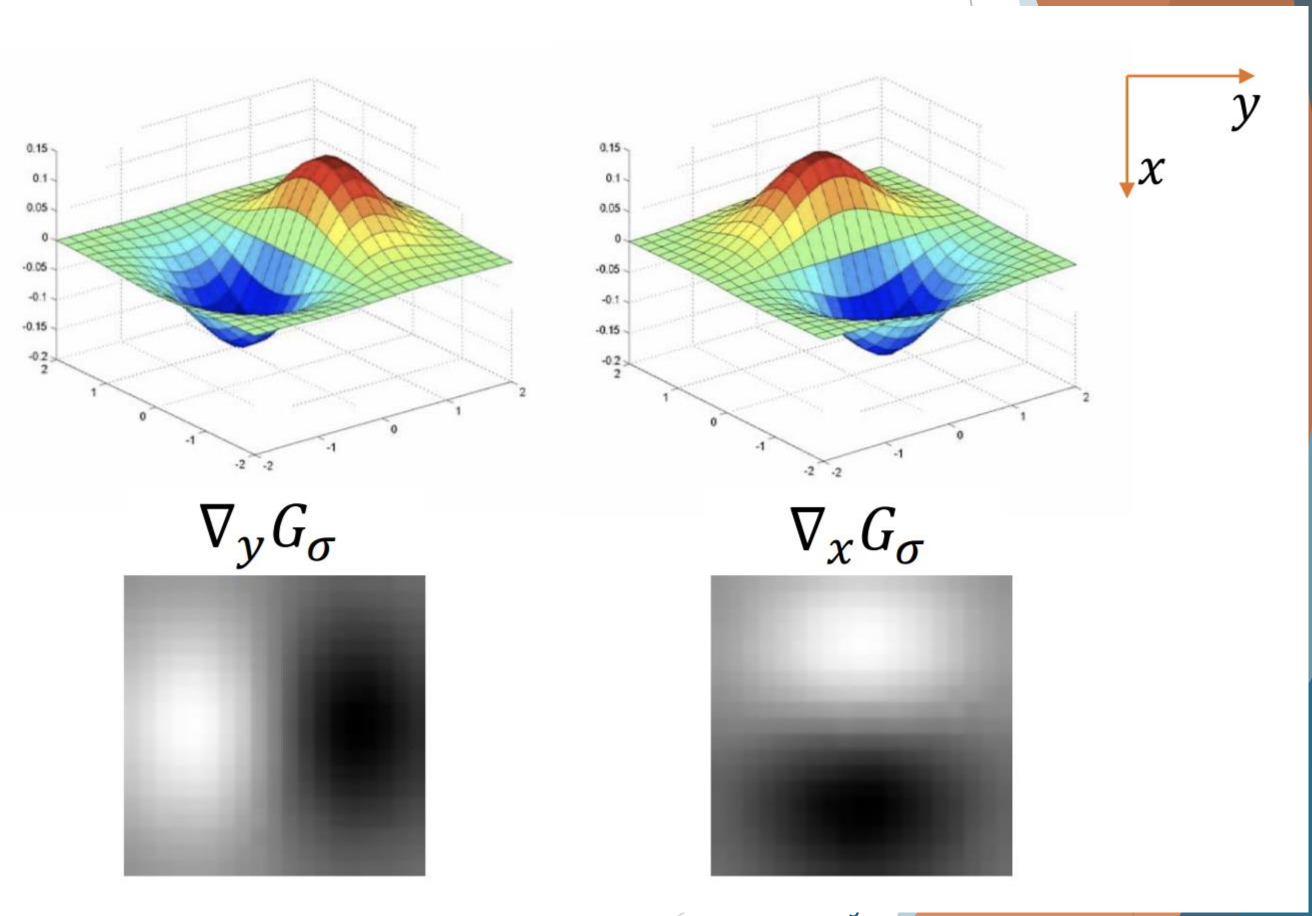
$$f_x = \nabla_x G_\sigma * f(x,y)$$

$$f_y = \nabla_y G_\sigma * f(x,y)$$

$$M(x,y) = \sqrt{f_x^2 + f_y^2}$$

$$\alpha(x,y) = \tan^{-1}\left(\frac{f_y}{f_x}\right)$$

Canny Gradient



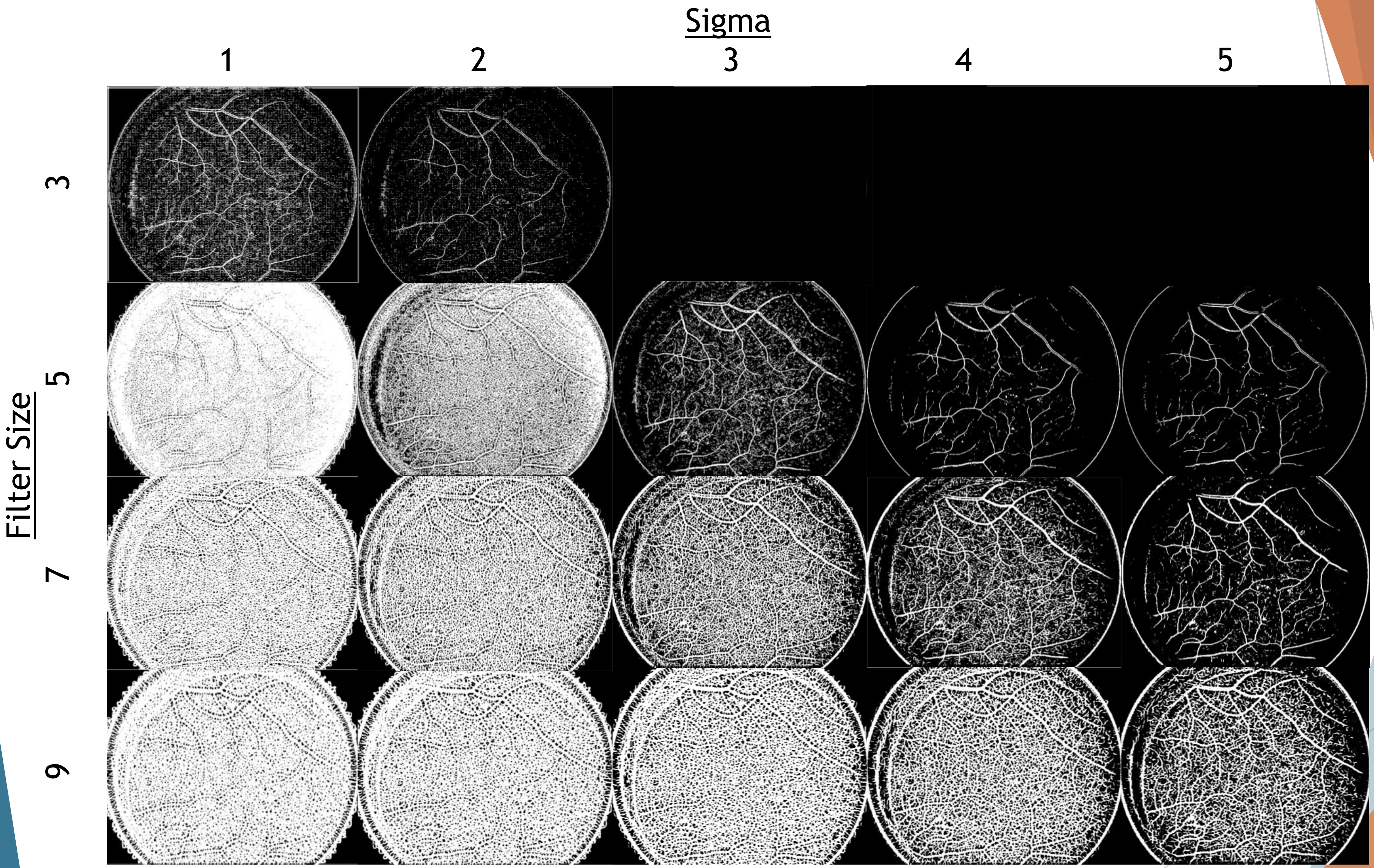
Technical Background – Matched Filter

- The matched filter implements a gaussian filter along one direction, rotating it, then applying the rotated filter. This is repeated until a bank of filtered images is completed.
- The filtered image bank is then checked pixel by pixel to find the strongest response to filtering with a new image formed from these responses.
- After a threshold is chosen, pixel gray values below the threshold are set to 0 (black) and above are set to 1 (white).
- Lastly, to eliminate small, weak edges which are disconnected from stronger edges, a length filtering technique is applied to the image. This involves labeling each cluster of connected pixels with value = 1, then removing the clusters below a certain threshold.

Experimental Results

General process

- I first experimented with Retina 1 to get a feel for how the different parameters affected the image quality after filtering. In particular, I varied sigma, the filter size, and the number of filters.
  - The filter is the gaussian function in matrix form with the size determinine the rows/columns of the nxn matrix.
  - The number of filters, as implied, changes the number of filters used on the image. I set the rotation angle between each filter to be equal to 180°/(number of filters). That is, if number of filters is 10, then there is an 18° difference between each filter with the first filter at 0°, then 10°and stopping at 162°.
  - After finding optimal matched filtering parameters, I experimented with the length filtering to determine how many pixels to remove for a good image.

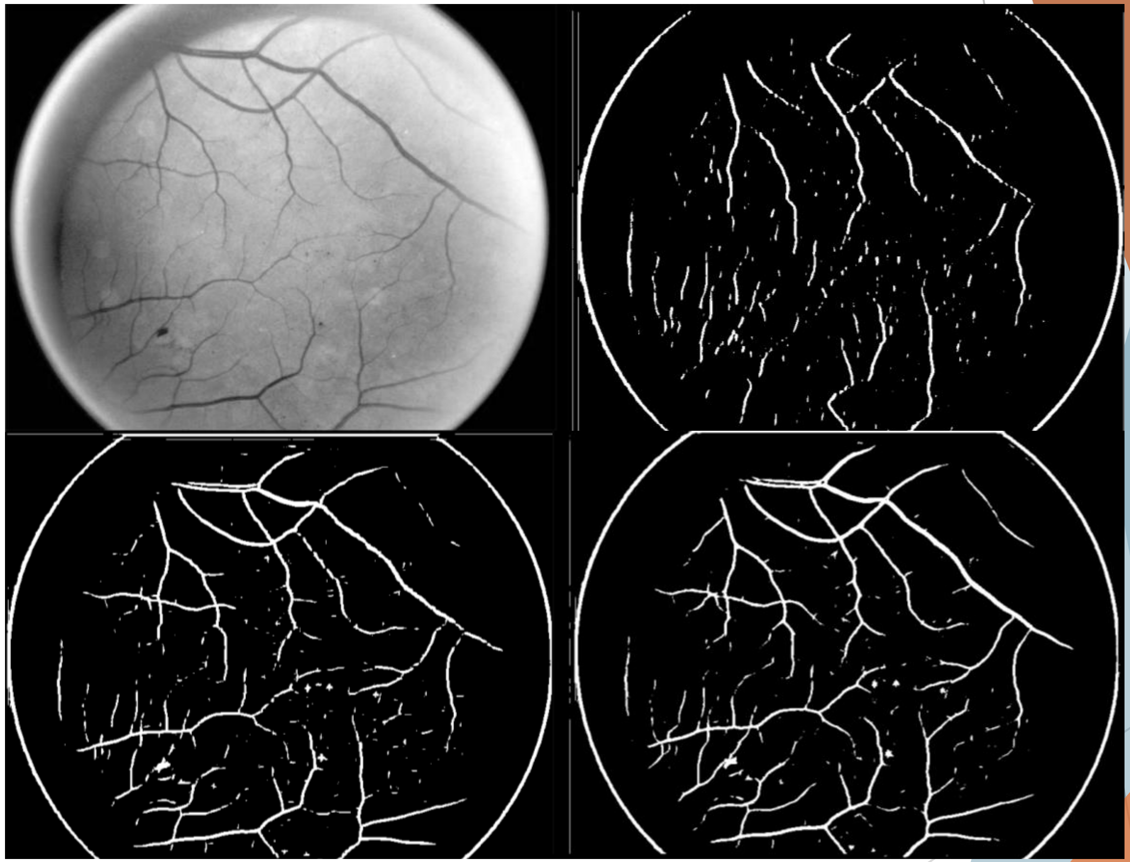

Sigma
1
2
3
4
5
Filter Size
3
5
7
9

Observations

- Sigma should roughly equal to the filter size but both should be relatively large
- This is likely since sigma translates to the intensity of the gaussian function; when the sigma is large and the filter small, only very prominent features over a small distance are detected. If a feature is not prominent enough over a short distance, it isn't detected (see sigma 3, 4, 5 for filter size 3).
  - For this same reason if sigma is small with a large filter size, small features and noise over a large area are detected when they should not be. This is why some images appear almost completely white.
- After tweaking more, I chose a sigma of 8, filter size of 9, shown on the right.

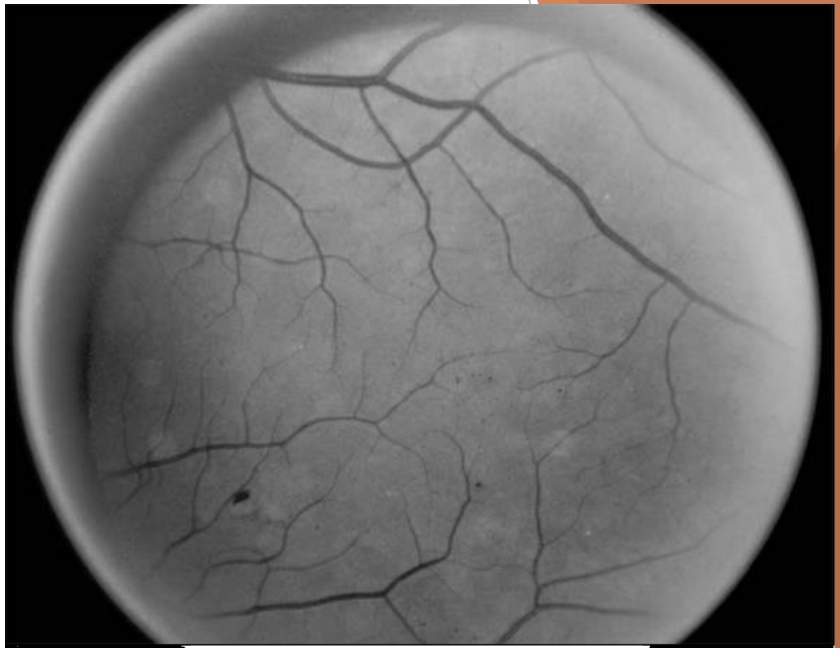
Altering Number of Filters

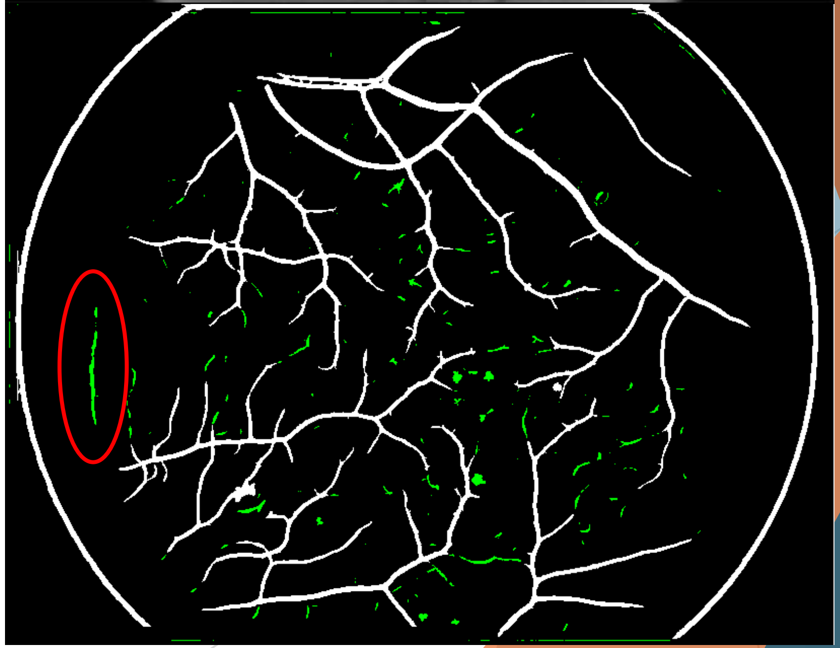
- Another observation is that the number of filters doesn’t need to be very large to get a good image.
- Shown on the right is a comparison of the retina for sigma = 8, filter size = 9 and a varying filter number.
  - Top Left: Retina
  - Top Right: filter number = 1
  - Bottom Left: filter number = 2
  - Bottom Right: filter number = 3
- As a reminder, the angle between filters will be 180° / (filter number) with the first filter at 0°



Length Filtering Retina 1

- I determined that a filter with sigma = 8, filter size = 9 (i.e. 9x9 matrix), and filter number = 10 was a good balance of the parameters.
- For length filtering, I turned up the cluster size to be filtered out until all edges too small and all edges that aren't veins were eliminated. Shown below, the green pixels indicate edges that were eliminated.
  - The any clusters below a size of 300 pixels were eliminated only to eliminate the green edge circled in red (this isn't a vein). Otherwise, a size of 120 is sufficient.



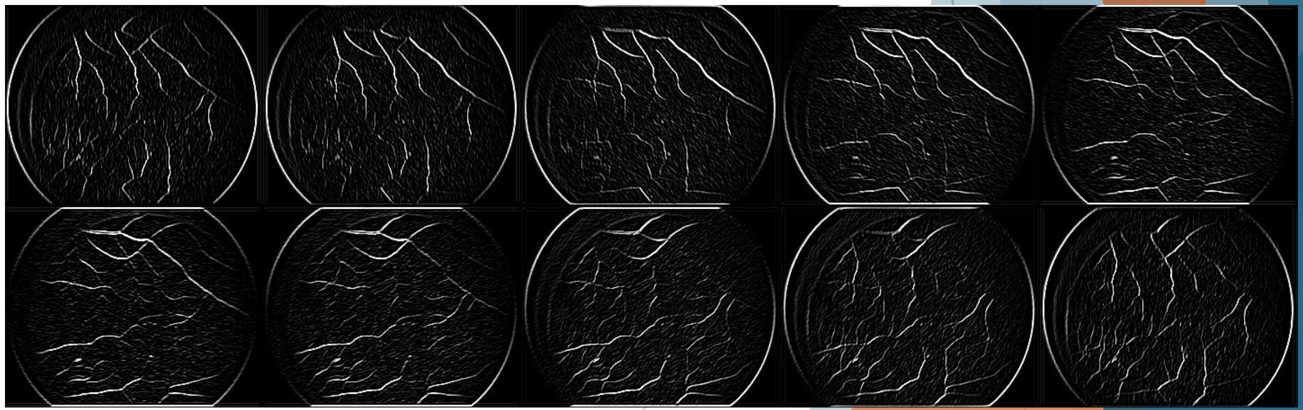


Retina 1 Final Results

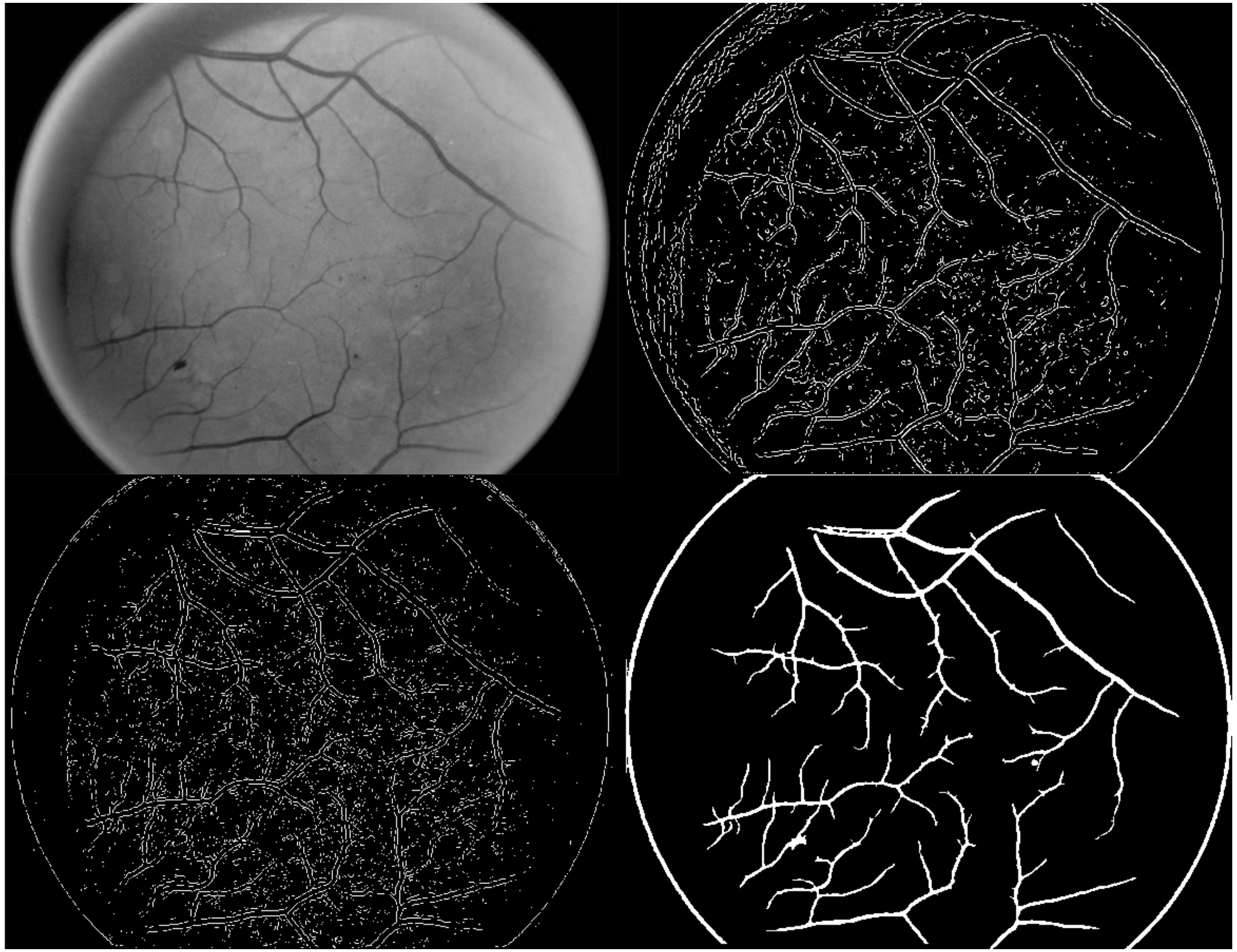
Above: The Retina at each stage: Original, Green Channel only, image fusion, thresholding, and length filtering

Below: Bank of filter kernels used

Below: Images after filtering

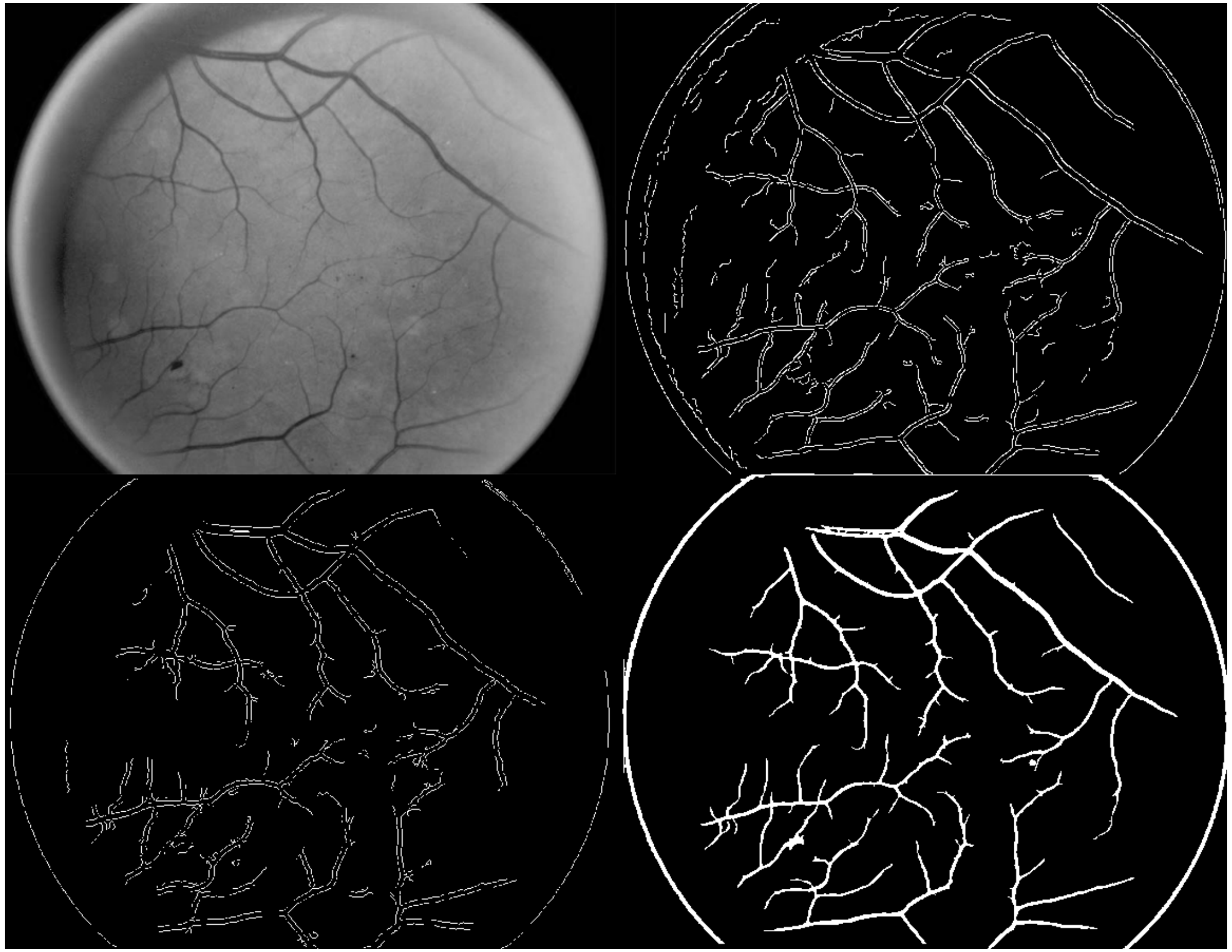


Retina 1 Final Results



Top Left: Retina;
Top Right: Canny;
Bottom Left: LoG;
Bottom Right: Matched Filter

Retina 1 Final Results

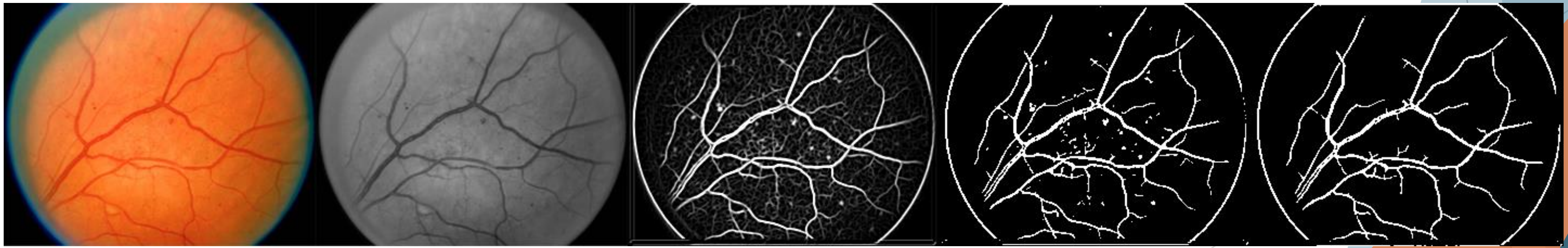


The same images after length filtering (LF) applied to Canny and LoG.

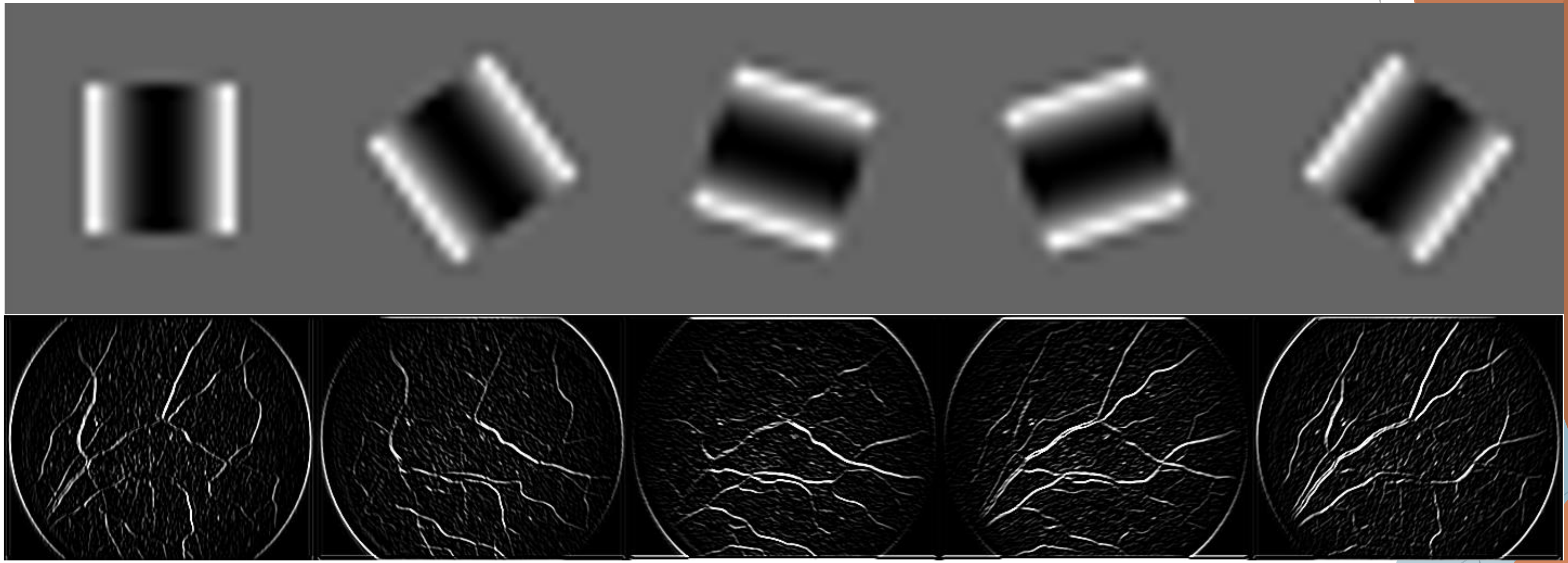
Retina 2 Final Results

- With the insights gained from testing retina under various parameter values, I optimized for retina 2. Unlike with retina 1, the following slides contain the final result rather than the full process.

Below: The Retina at each stage: Original, Green Channel only, image fusion, thresholding, and length filtering



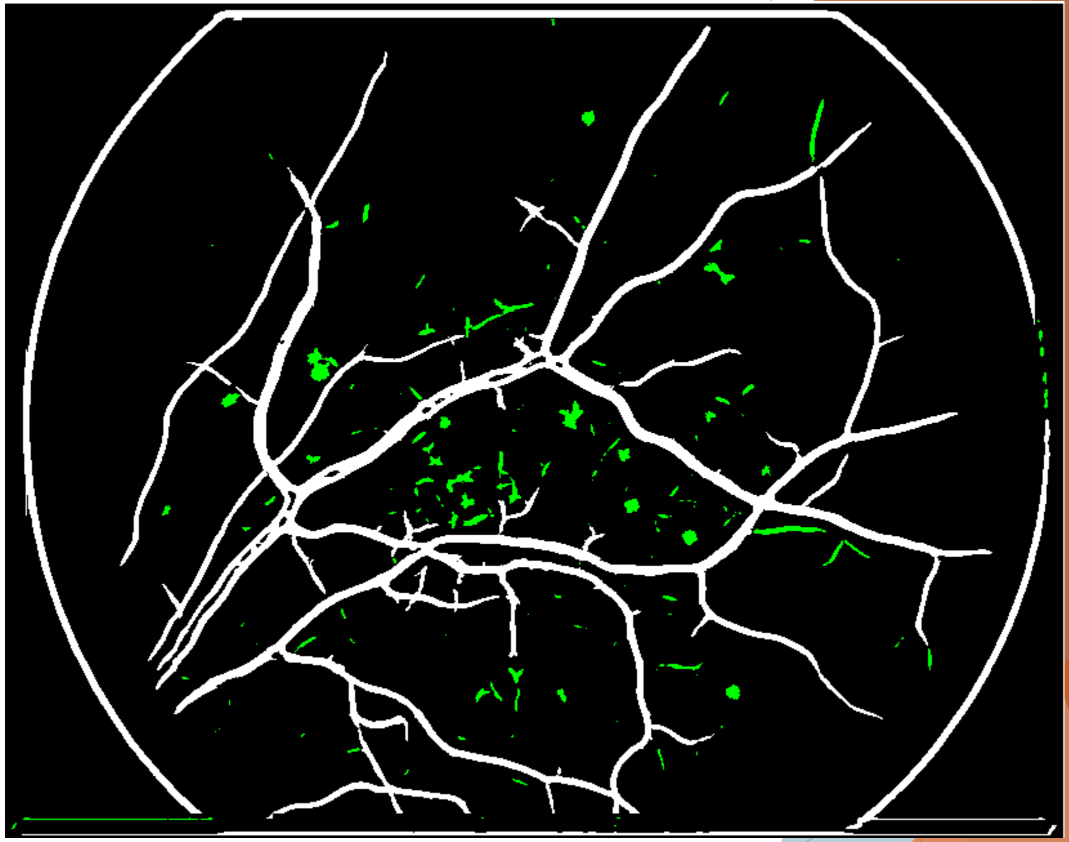
Retina 2 Final Results



Above: Filter bank; Below: Filtered images

Retina 2 Technique Comparison

Top left: Original; Top Right: Canny with LF
Bottom Left: LoG with LF; Bottom Right: Matched Filtering with LF



Retina 2 after length filtering

Final Parameter Values

Retina 1

- Matched Filter
  - σ = 8
  - Filter size = 9
  - # of filters = 10
  - LF = 300
- Canny
  - LF = 25
- LoG
  - LF = 25

Retina 2

- Matched Filter
  - σ = 11
  - Filter size = 11
  - # of filters = 5
  - LF = 300
- Canny
  - LF = 90
- LoG
  - LF = 25

Discussion and Conclusion

- The matched filtering looks better to me than the Canny and the LoG, but maybe I’m biased because I made it and I’m proud of it.
- I tried to optimize the threshold of the Canny filtering on both retina 1 and retina 2, but it only helped a little. Length filtering was more effective than manually changing the thresholding.

Appendix – Code – Main file

```
clear;

%% Read in the image
%  per the document, only need to read in the green channel.
I1=imread('retina1.jpg');
I2=imread('retina2.jpg');
J1(:,:)=I1(:,:,2); % I1 in height x width x RGB value
J2(:,:)=I2(:,:,2);


%% Run the Matched Filtering
[BW1, I_bank1, Filter_Bank1, Ker_pad1] = Matched_Filter(J1, 8, 9, 10);
[BW2, I_bank2, Filter_Bank2, Ker_pad2] = Matched_Filter(J2, 11, 11, 5);


%% Length Filtering
[LF_1] = Length_Filter(BW1, 8, 300);
[LF_2] = Length_Filter(BW2, 8, 300);

% Trim the borders of the image
LF_1(627:633,:) = [];
LF_1(1:7,:) = [];
LF_1(:,1:3) = [];


%% Pick images to be displayed
Show_I1 = true; % show image 1
Show_I2 = true; % show image 2

% this variable counts which figure I'm at so I don't have to manually
% adjust figures each time I add a new one.
fig_num = 0;
```

Appendix – Code – Main file

```
fig_num = 0;


%% Display Image 1 (J1)
if(Show_I1)
% default thresh = [0.0375, 0.0938]
% [I_Canny1_default, threshOut_C1] = edge(J1, "canny");
[I_Canny1] = edge(J1, "canny", [0.093549, 0.09355]); % Canny
[I_LoG1, threshOut_L1] = edge(J1, "log"); % LoG

%Image before filtering vs canny vs LoG vs matchfiltered/lengthfiltered
fig_num = fig_num+1;
figure(fig_num);
I_list1 = {J1, I_Canny1, I_LoG1, LF_1};
montage(I_list1);

%Filters used for image 1
fig_num = fig_num+1;
figure(fig_num);
montage(mat2gray(Filter_Bank1), Size=[2 5]);

%images after filtering
fig_num = fig_num+1;
figure(fig_num);
montage(I_bank1, Size=[2 5]);

%Show effect of length filtering on image
fig_num = fig_num+1;
figure(fig_num);
% display 2 images: "montage" for side-by-side; "falsecolor" for layered image;
imshowpair(J1, LF_1, "montage");
```

Appendix – Code – Main file

```

%try length filtering on LoG and Canny
[LF_Canny] = Length_Filter(I_Canny1, 8, 25);
[LF_LoG] = Length_Filter(I_LoG1, 8, 25);
fig_num = fig_num+1;
figure(fig_num);
montage({J1, LF_Canny, LF_LoG, LF_1});

end % End of Display Image 1


%% Display Image 2 (J2)
if(Show_I2)
%  thresh seems to work best.
% default thresh = [0.025, 0.0625]
[I_Canny2] = edge(J2, "canny", [0.07 0.075]);
[I_LoG2, threshOut_L2] = edge(J2, "log");

%Image before filtering vs canny vs LoG vs matchfiltered/lengthfiltered
fig_num = fig_num+1;
figure(fig_num);
I_list2 = {J2, I_Canny2, I_LoG2, LF_2};
montage(I_list2);

%Filters used for image 2
fig_num = fig_num+1;
figure(fig_num);
montage(mat2gray(Filter_Bank2), Size=[1 5]);

%images after filtering
fig_num = fig_num+1;
```

Appendix – Code – Main file

```
fig_num = fig_num+1;
figure(fig_num);
montage(I_bank2, Size=[1 5]);

%Show effect of length filtering on image
fig_num = fig_num+1;
figure(fig_num);
% display 2 images: "montage" for side-by-side; "falsecolor" for layered image;
imshowpair(BW2, LF_2, "montage");

%try length filtering on LoG and Canny
[LF_Canny] = Length_Filter(I_Canny2, 8, 90);
[LF_LoG] = Length_Filter(I_LoG2, 8, 25);
fig_num = fig_num+1;
figure(fig_num);
montage({J2, LF_Canny, LF_LoG, LF_2});

end % End of Display Image 2

```

Appendix – Code – Matched Filter

```
function [BW, I_bank, Filter_Bank, Ker_pad] = Matched_Filter(image, sigma, filter_size, filter_num)
% This function filters the image,

% This variable is used frequently for various operations.
% It represents the half of the filter_size (odd) rounded to zero.
x = double(idivide(int64(filter_size), int64(2), 'fix'));

%% Create the Kernel
% preallocate matrix size with zeros
Ker = zeros(1, filter_size);
for j = -x:x % corresponding to the column
        % breaking the 1D Gaussian function down for readability
        G_c = -1/(sqrt(2*pi*(sigma^2)));
        e_c = exp(-(j^2)/(2*(sigma^2)));
        G = G_c * e_c;
        Ker(j+x+1) = G;
end

% Find the mean of the distribution.
m0 = mean(Ker);
% Subtract the distribution by the mean (bringing mean to 0).
Ker = Ker - m0;


% Form matrix rows which are all uniform.
% Use Ker_temp for appending rows. If Ker is used instead, the matrix row
% filter_size will double every iteration rather than linearly increase by +1
Ker_temp = Ker;
for i = 1:filter_size-1
    Ker = [Ker; Ker_temp];
end
```

Appendix – Code – Matched Filter

```
%% Create a bank of filters at different rotations
% Pad filter kernel with zeros before rotation (the +1 keeps the matrix odd)
Ker_pad = zeros(filter_size*2+1);
for j = 1:filter_size
    for i = 1:filter_size
        % Fill padded matrix with Gaussian matrix and align centers
        % by increasing index by a value of x+1
        Ker_pad(i+x+1,j+x+1) = Ker(i,j);
    end
end

% Create Filter Bank
theta = 0;
Filter_Bank = [];
for i = 1:filter_num
    B = imrotate(Ker_pad, theta, 'bicubic', 'crop');
    Filter_Bank(:,:,i) = B;
    theta = theta + 180/filter_num; % increase rotation by 180/number of filters
end


%% Apply Kernel to the image (i.e. filter)

% create a bank of filtered images
for i = 1:filter_num
    I = conv2(Filter_Bank(:,:,i), image);
    I_bank(:,:,i) = I; % I_bank stores the images
end
```

Appendix – Code – Matched Filter

```
%% Fuse all filtered images

%  assign the pixel value to be the maximum one across all filtered images
s = size(image);

m = s(:,1);
n = s(:,2);
for i = 1:filter_num
    for y = 1:n
        for x = 1:m
            I(x,y) = max(I_bank(x,y,i), I(x,y));
        end
    end
end


%% Find the appropriate threshold and binarize the image data
%  (MATLAB "GRAYTHRESH")
T = graythresh(I);
BW = imbinarize(I, T);

end
```

Appendix - Code - Length Filtering

```
function [Image_out] = Length_Filter(BW, connectivity, pixel_filter)
%LENGTH_FILTER For Project 2 in Computer Vision
% This function filters out small, disconnected pixel clusters under
% a certain size determined by pixel_filter.

% connectivity must be 4 or 8
L = bwlabel(BW, connectivity);

% Go through and find all groups and assign to different groups.
% Then filter out the groups smaller than pixel_filter
for i = 1:max(max(L))
    % Find all pixels with label i; store coordinates in r and c
    [r, c] = find(L == i);
    for j = 1:size(r)
        if(size(r) < pixel_filter)
            x = r(j);
            y = c(j);
            Image_out(x,y) = false; % place a 0 at x,y
        else
            x = r(j);
            y = c(j);
            Image_out(x,y) = true; % place a 1 at x,y
        end
    end
end
end

```