Brianne Trollo
CS558: Computer Vision
26 September 2019
Assignment 1

This assignment was completed using Matlab. For all images, the boundary pixels were replicated as stated in the Assignment description.

To implement the gaussian filter, the size of the filter is calculated by multiplying the provided sigma by 6, to ensure that it spans three standard deviations on each side of the normal distribution, and subtracting one, so that the size remains odd (6*sigma-1). Then the filter is calculated using the formula provided in the slides for a spatially weighted average, which if it does not sum to 1, is normalized. The filter is applied to the image and the result is the image smoothed.

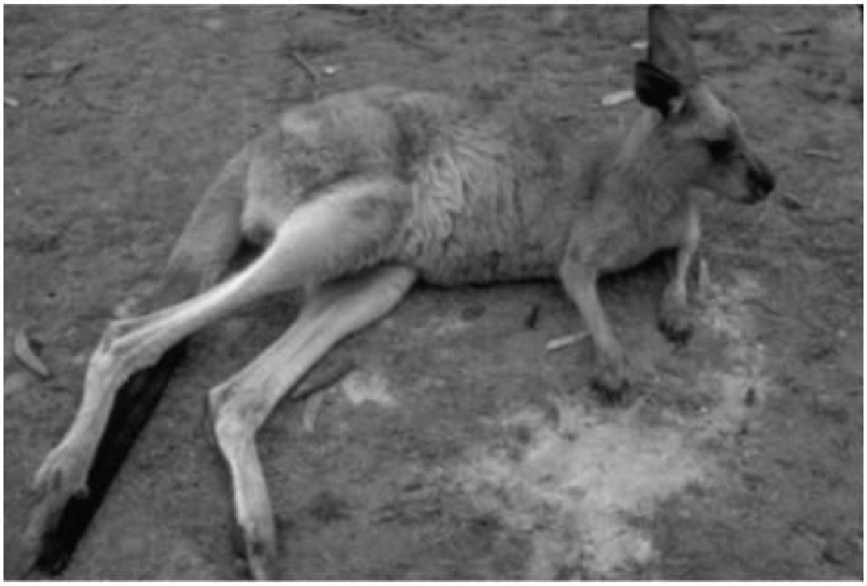
To compute the gradient, the vertical and horizontal sobel filters are applied to the smoothed image, individually, if the parameter for myfilter is “sobel-x” or “sobel-y”, or together, if the parameter for myfilter is “sobel”. Then, the threshold is applied to every pixel and all pixels that are not edges are set to black. The result is returned.

To implement the non-maximum suppression, first, the directional matrix of the gradient and the magnitude of the gradients is computed. Then 360 is added to each negative element of directional matrix to make all angles positive, and the angles are rounded to the nearest 0, 45, 90, or 135 degree angle. The magnitudes of each pixel is checked against those diagonal and orthogonal to it; if greater, then the pixel is set to the magnitude, otherwise it is set to zero. The result is returned.

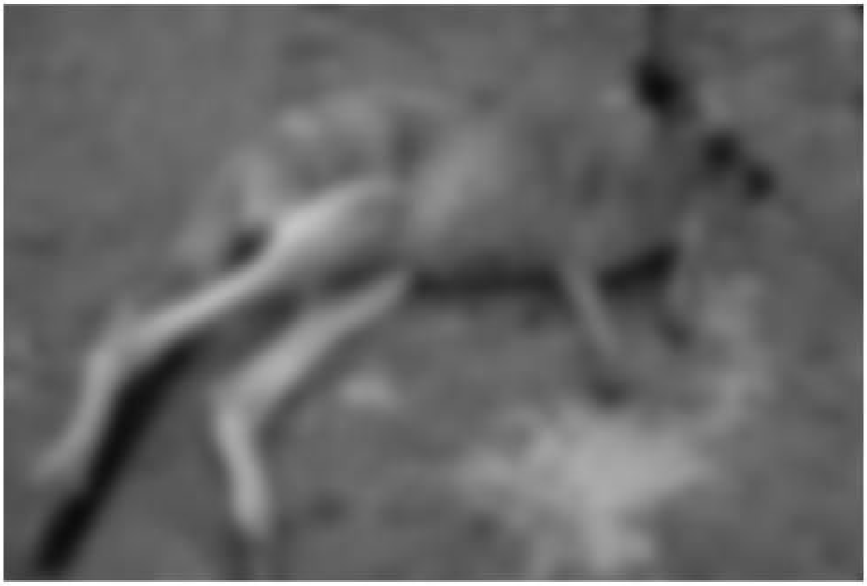
PART 1: Gaussian Filtering

Kangaroo

$\sigma = 1$

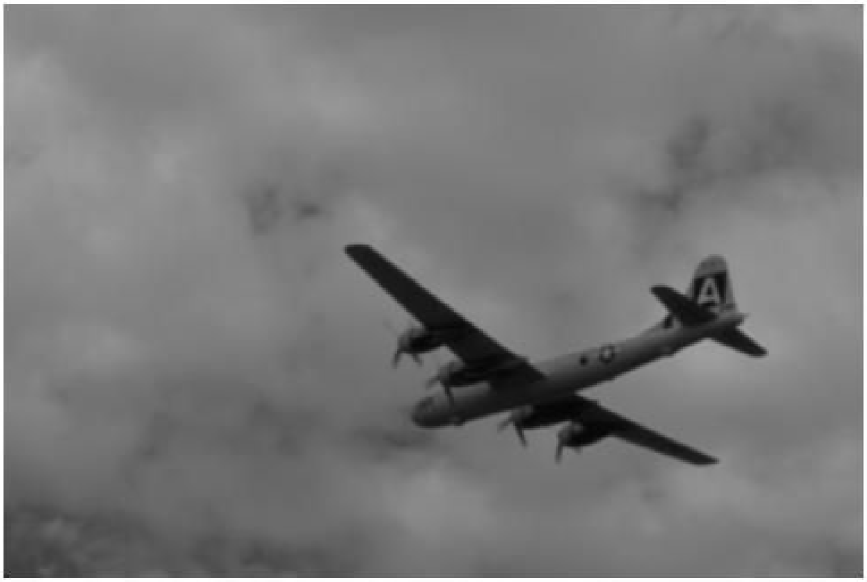


$\sigma = 5$

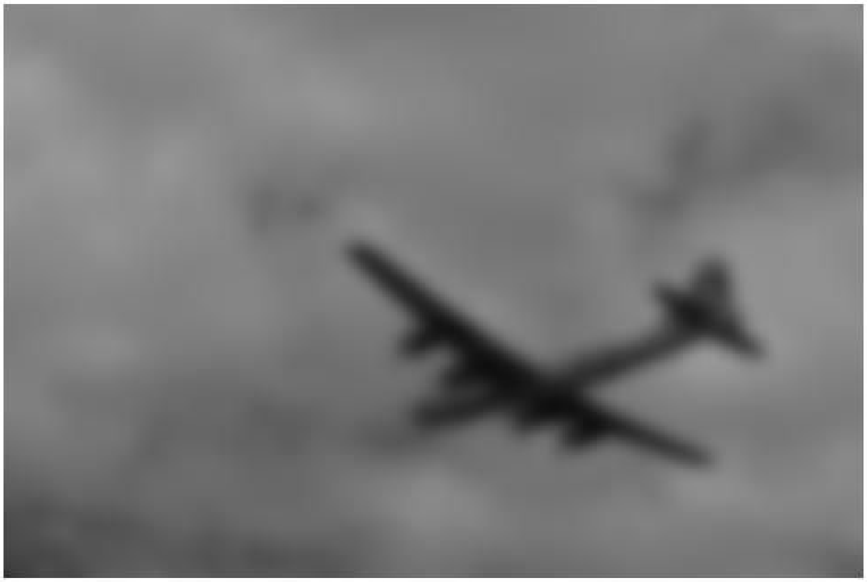


Plane

σ = 1

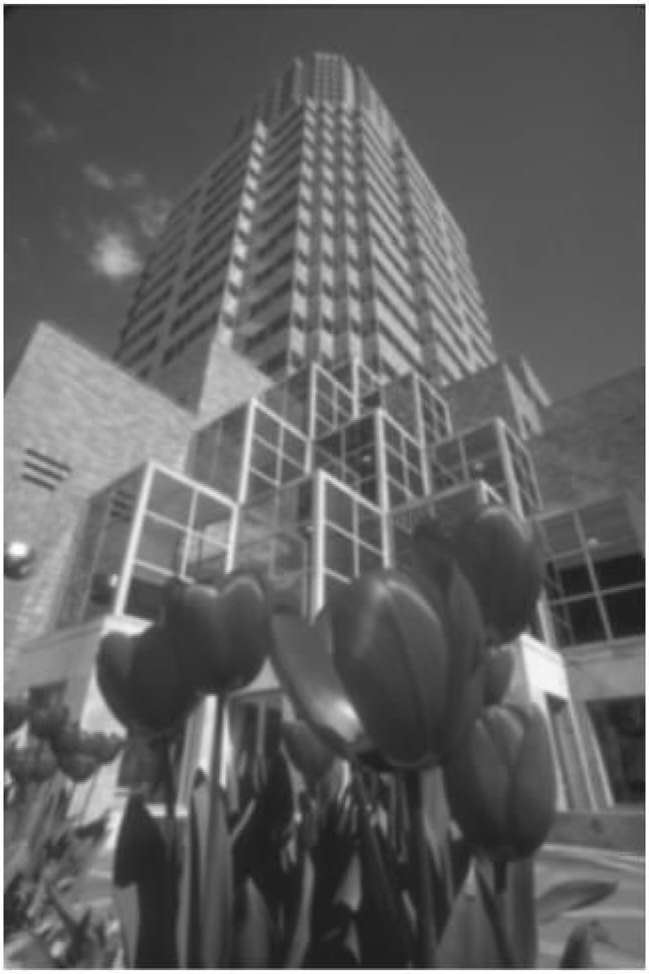


σ = 5

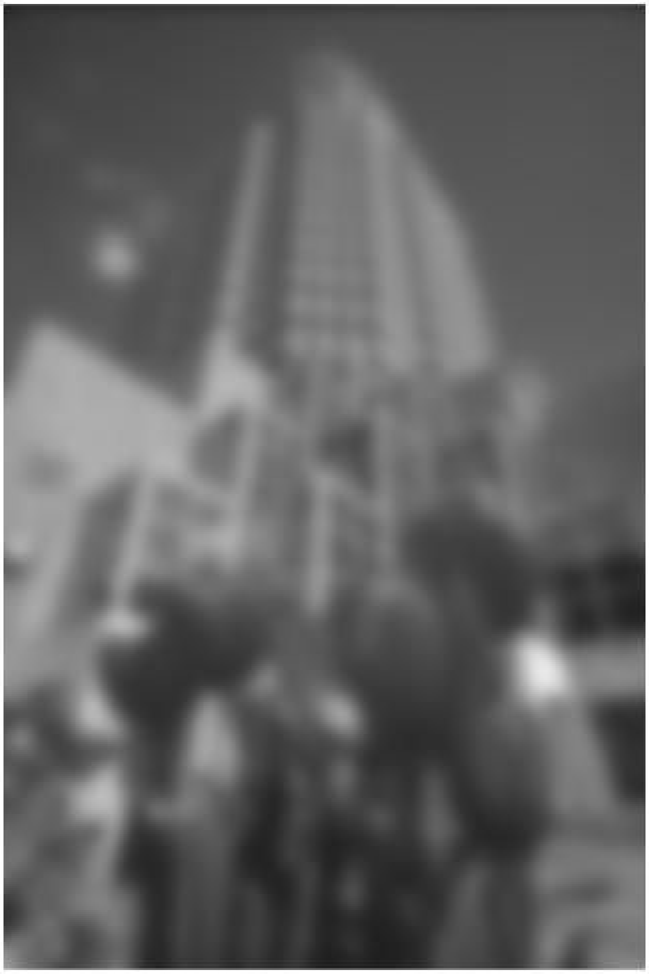


Red

$\sigma = 1$



σ = 5

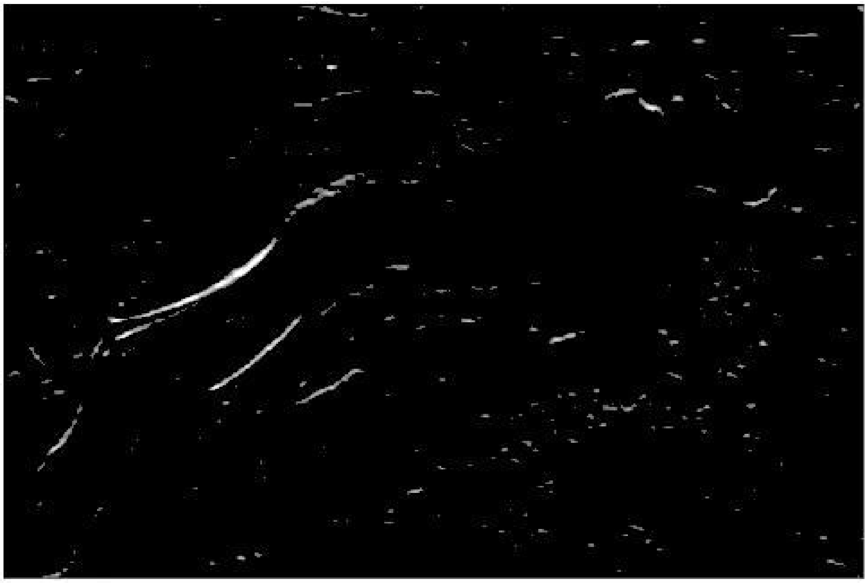


PART 2: Sobel Filters

Kangaroo

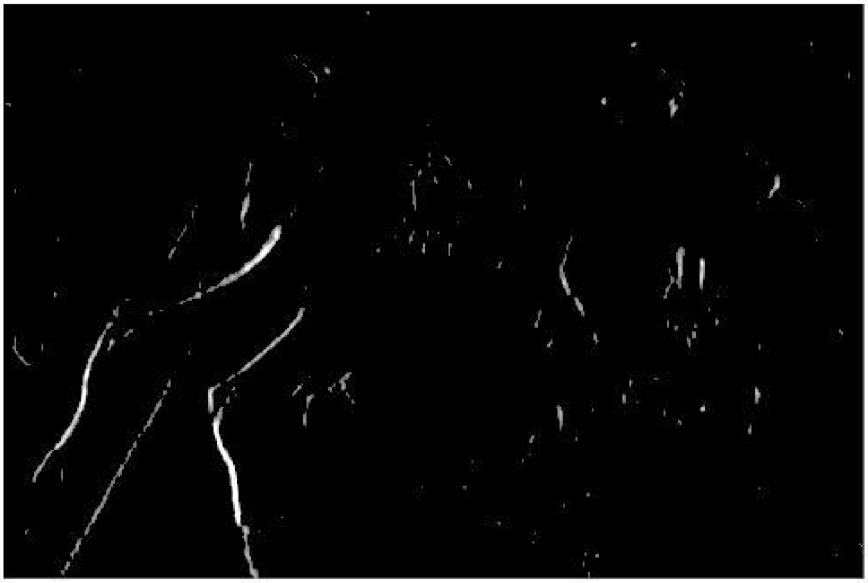
Horizontal Sobel Filter

$\sigma = 1,\ Threshold = 95$



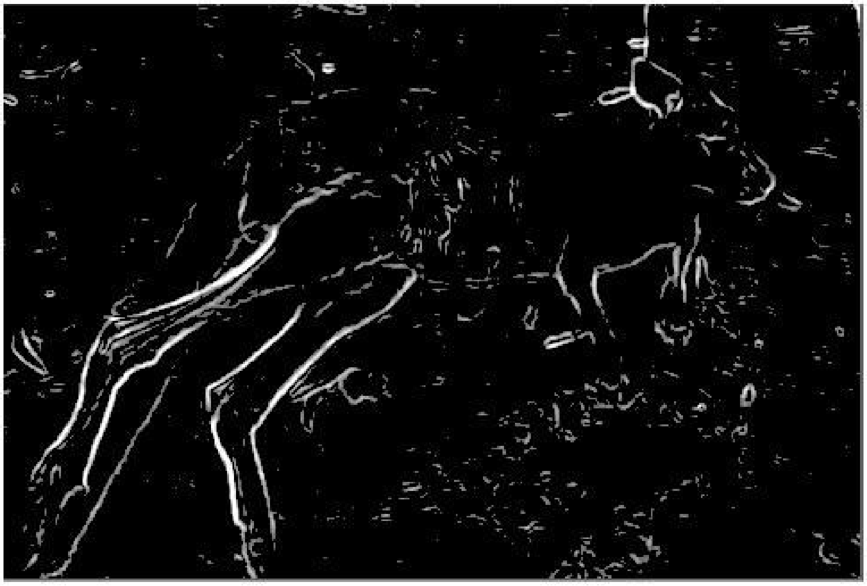
Vertical Sobel Filter

$\sigma = 1,\ Threshold = 95$



Combined Horizontal and Vertical Sobel Filters

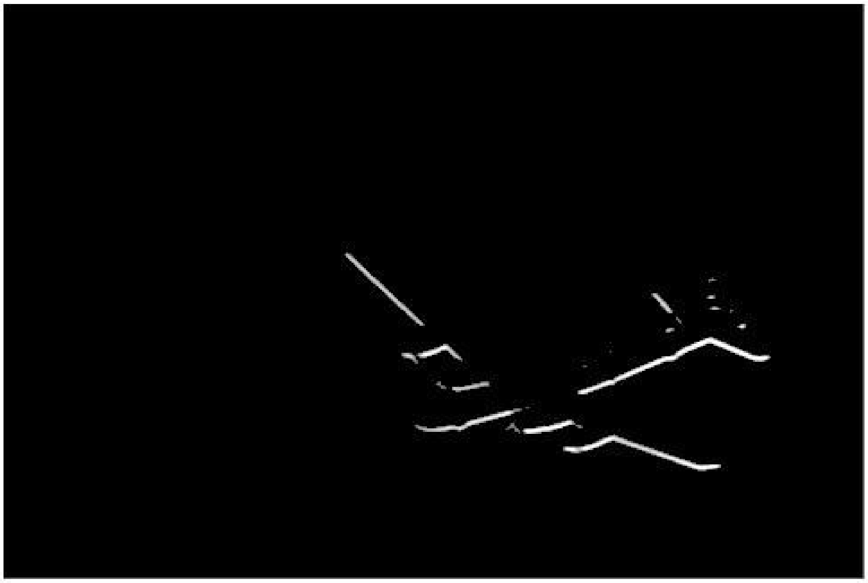
$\sigma = 1,\ Threshold = 95$



Plane

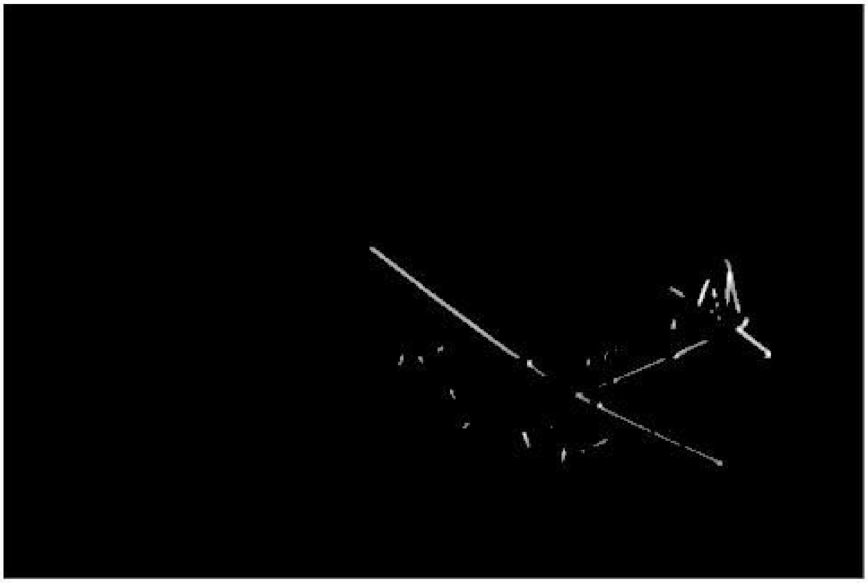
Horizontal Sobel Filter

$\sigma = 1,\ Threshold = 95$



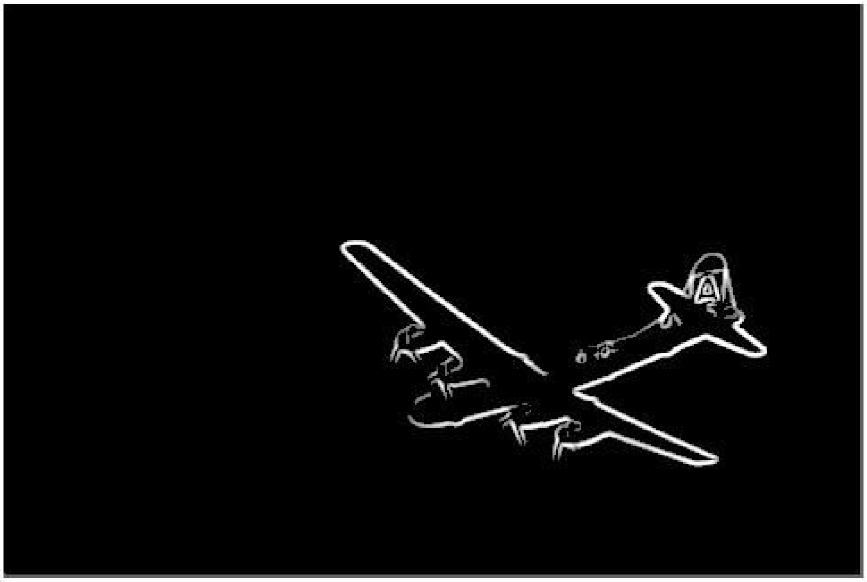
Vertical Sobel Filter

$\sigma = 1,\ \mathit{Threshold} = 95$



Combined Horizontal and Vertical Sobel Filters

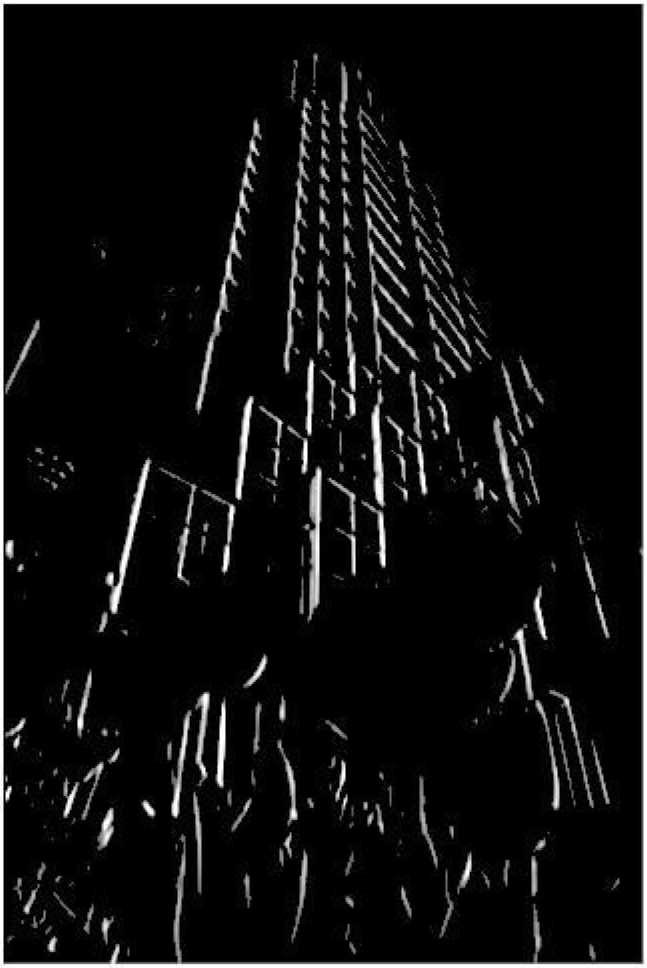
$\sigma = 1,\ \mathit{Threshold} = 95$



Red

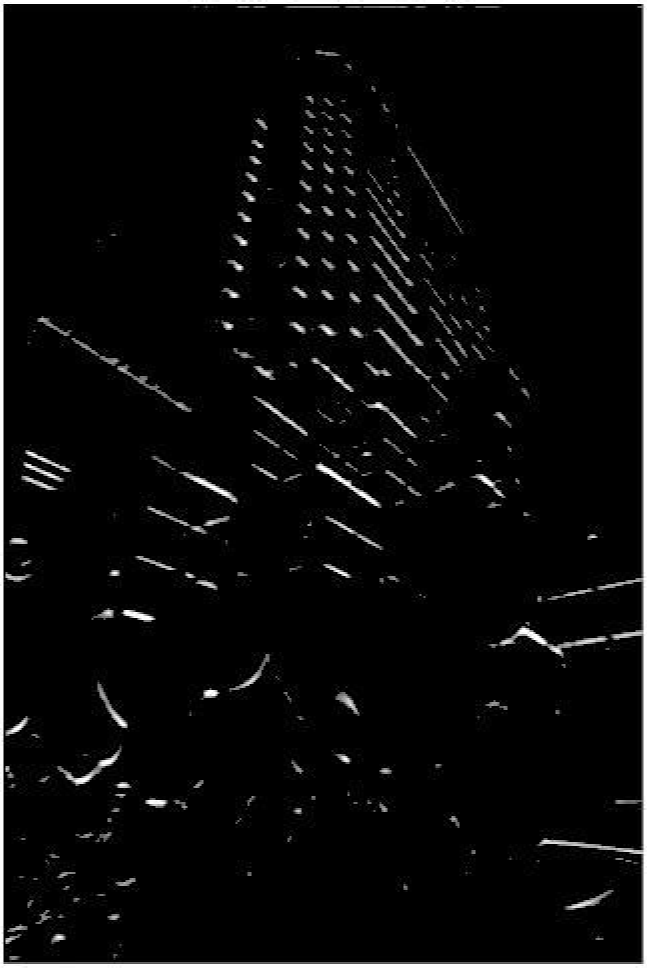
Horizontal Sobel Filter

$\sigma = 1, \ Threshold = 95$



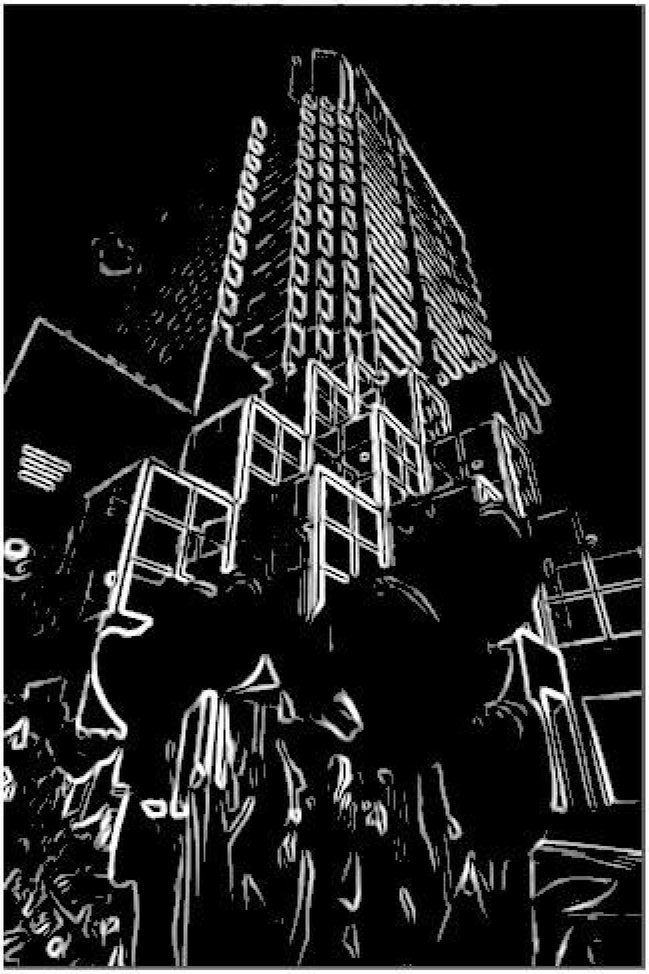
Vertical Sobel Filter

$\sigma = 1,\ Threshold = 95$



Combined Horizontal and Vertical Sobel Filters

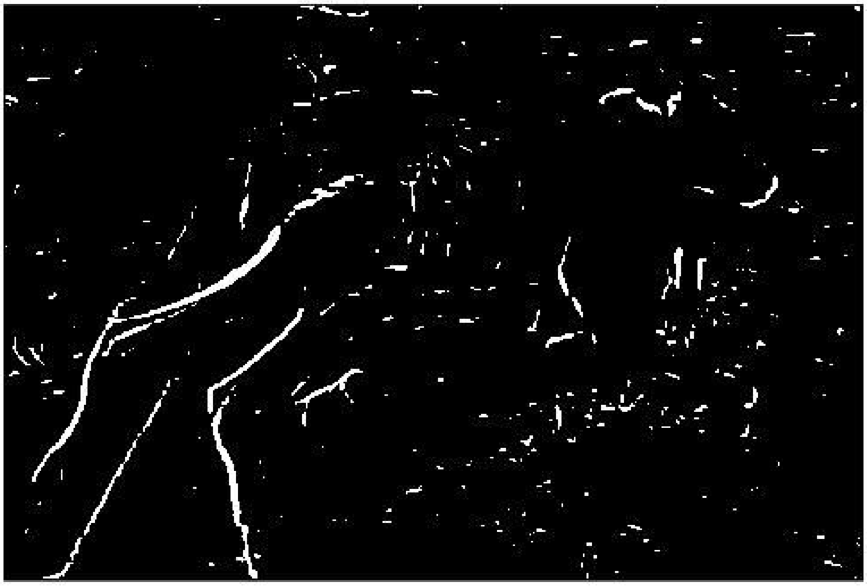
$\sigma = 1,\ Threshold = 95$



PART 3: Non-maximum Suppression

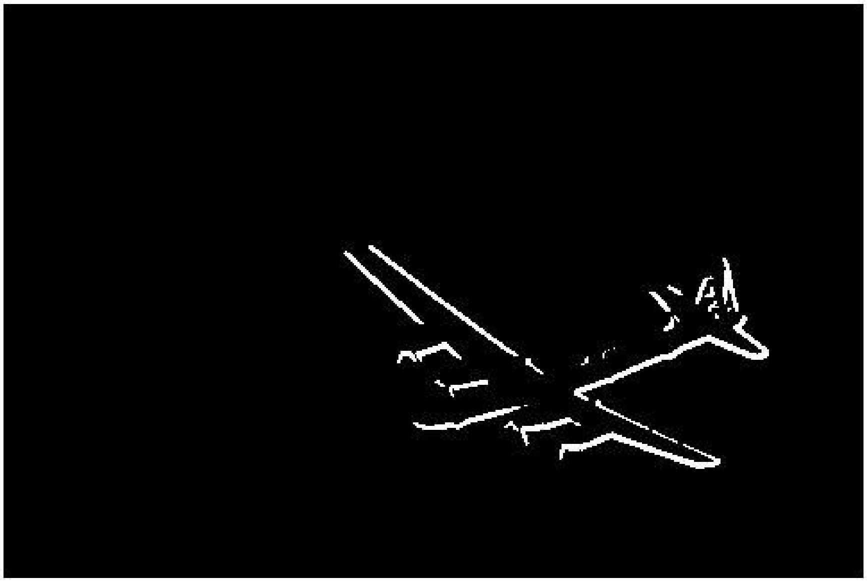
Kangaroo

$\sigma = 1, \ Threshold = 95$



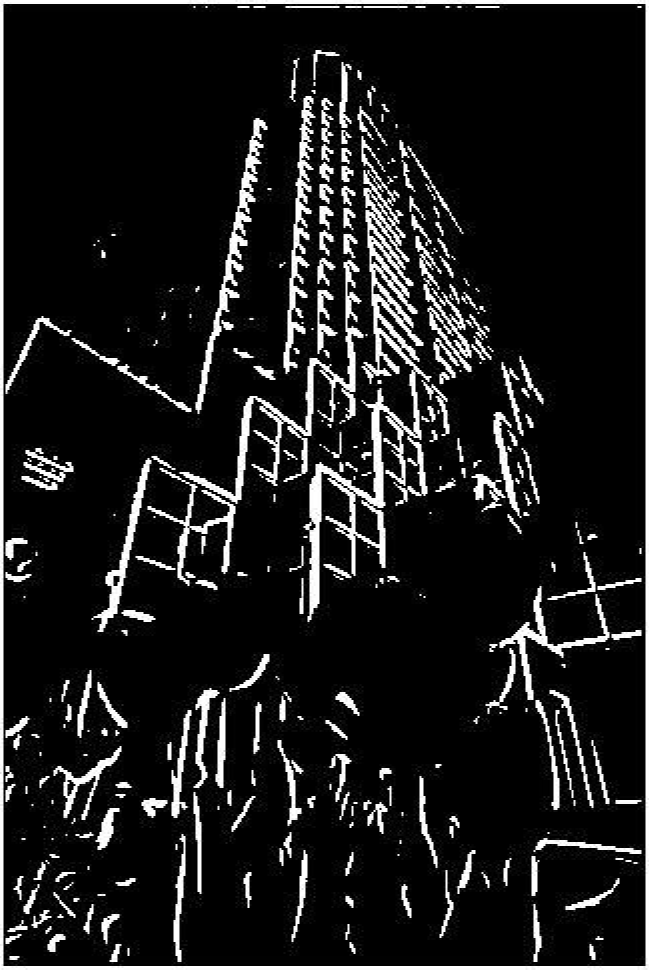
Plane

$\sigma = 1, \ Threshold = 95$



Red

$\sigma = 1,\ Threshold = 95$



```
% Brianne Trollo
% CS 558: Computer Vision
% 26 September 2019
% Assignment 1

function main()
    close all;
    clear variables;

    sigma = 1;
    threshold = 95;

    img = imread("plane.pgm");
    figure(1);
    subplot(2, 3, 1);
    imshow(img);
    title("Original");
% %     Gaussian Filter
    filter_g = myfilter(img, sigma, threshold, "gaussian", "replicate");
    subplot(2, 3, 2);
    imshow(filter_g);
    title("Guassian");
% % %     Sobel Filter
    filter_s = myfilter(filter_g, sigma, threshold, "sobel", "replicate");
    subplot(2, 3, 3);
    imshow(filter_s);
    title("Sobel");
    %     Horizontal Sobel Filter try threshold > 200
    filter_sx = myfilter(filter_g, sigma, threshold, "sobel-x", "replicate");
    subplot(2, 3, 4);
    imshow(filter_sx);
    title("Sobel-X");
%     %     Vertical Sobel Filter
    filter_sy = myfilter(filter_g, sigma, threshold, "sobel-y", "replicate");
    subplot(2, 3, 5);
    imshow(filter_sy);
    title("Sobel-Y");
%     Non-maximum suppression
    filter_nms = mynms(filter_g, filter_sx, filter_sy);
    subplot(2, 3, 6);
    imshow(filter_nms);
    title("NMS");
```

```
end

function bordered = myborder(edg, img, sz)
% Replicate boundary pixels
    if strcmp(edg, "replicate")
        bordered = padarray(img, [sz sz], 'replicate');
%           Add border of zeros
    elseif strcmp(edg, "clip")
        bordered = padarray(img, [sz sz]);
    end
end

function result = myfilter(img, sigma, threshold, filt, edg)
    c_img = double(img);
%       Gaussian Filter
    if strcmp(filt, "gaussian")
%           Window size - must be odd
        wind_size = 6*sigma-1;
%           Gaussian filter
        [x,y] = meshgrid(-wind_size:wind_size);
        G = (exp(-(x.^2 + y.^2)/(2*sigma^2)))/(2*pi*sigma^2);

%           Get sum of filter coefficients
        co_sum = round(sum(sum(G)));

%          if sum not equal to 1, normalize
        if co_sum ~= 1
            G = G./(wind_size^2);
        end

%          Initialize result image
        result = zeros(size(c_img));

%           pad image with edge type edg
        c_img = myborder(edg, c_img, wind_size);

%           Apply Gaussian filter
        X = size(x, 1) -1;
        Y = size(y, 1) -1;
        for i = 1:size(c_img, 1) - X
            for j = 1:size(c_img, 2) - Y
                tmp = c_img(i:i+X, j:j+Y).*G;
                result(i, j) = sum(tmp(:));
            end
        end
        result = uint8(result);
    end
%        Combined Sobel Filter
    if strcmp(filt, "sobel")
        Gx = [-1 -2 -1; 0 0 0; 1 2 1];
        Gy = [-1 0 1; -2 0 2; -1 0 1];

%       Initialize result image
        result = zeros(size(c_img));

%        Apply sobel x and y filters
        for i = 1:size(c_img, 1) - 2
            for j = 1:size(c_img, 2) - 2
                tmpx = c_img(i:i+2, j:j+2).*Gx;
                tmpy = c_img(i:i+2, j:j+2).*Gy;
                result(i, j) = sqrt(sum(tmpx(:)).^2 + sum(tmpy(:)).^2);
            end
        end

%        Apply threshold
        result = max(result, threshold);
        for i = 1:size(result, 1)-2
            for j = 1:size(c_img, 2)-2
                if result(i, j) == round(threshold)
                    result(i, j) = 0;
                end
            end
        end


        result = uint8(result);
    end
%        Horizontal Sobel Filter
    if strcmp(filt, "sobel-x")
        Gx = [-1 -2 -1; 0 0 0; 1 2 1];

%        Apply sobel x filter
        for i = 1:size(c_img, 1) - 2
            for j = 1:size(c_img, 2) - 2
                tmpx = sum(sum(c_img(i:i+2, j:j+2).*Gx));
```

```
                result(i+1, j+1) = tmpx;
            end
        end

%           Apply threshold
        result = max(result, threshold);
        for i = 1:size(c_img, 1) - 2
            for j = 1:size(c_img, 2) - 2
                if result(i, j) == threshold
                    result(i, j) = 0;
                end
            end
        end

        result = uint8(result);
    end
%           Vertical Sobel Filter
    if strcmp(filt, "sobel-y")
        Gy = [-1 0 1; -2 0 2; -1 0 1];

%           Apply sobel y filter
        for i = 1:size(c_img, 1) - 2
            for j = 1:size(c_img, 2) - 2
                tmpy = sum(sum(c_img(i:i+2, j:j+2).*Gy));
                result(i+1, j+1) = tmpy;
            end
        end

%           Apply threshold
        result = max(result, threshold);
        for i = 1:size(c_img, 1) - 2
            for j = 1:size(c_img, 2) - 2
                if result(i, j) == threshold
                    result(i, j) = 0;
                end
            end
        end


        result = uint8(result);

    end
end
```

```
% Non-maximum Suppression
function result = mynms(img, sobel_x, sobel_y)
    c_img = double(img);

    angle_matrix = atan2(double(sobel_y), double(sobel_x))*180/pi;

    magn = sqrt(double(sobel_x.^2 + sobel_y.^2));

    X = size(angle_matrix, 1);
    Y = size(angle_matrix, 2);

%       Make all angles positive
%       Adjust angles to 0, 45, 90, or 135
    for i=1:X
        for j=1:Y
            if angle_matrix(i, j) < 0
                angle_matrix(i,j) = 360 + angle_matrix(i,j);
            end
            if ((angle_matrix(i,j) >= 0) && (angle_matrix(i,j) < 22.5) || ...
                (angle_matrix(i,j) >= 337.5) && (angle_matrix(i,j) <= 360)  || ...
                (angle_matrix(i,j) < 157.5) && (angle_matrix(i,j) < 202.5))
                 % Round anything around 0, 180, or 360 to 0
                angle_matrix(i, j) = 0;
            elseif ((angle_matrix(i,j) >= 22.5) && (angle_matrix(i,j) < 67.5) || ...
                        (angle_matrix(i,j) >= 202.5) && (angle_matrix(i,j) < 247.5))
                    % Round anything around 45, or 225 to 45
                    angle_matrix(i,j) = 45;
            elseif ((angle_matrix(i,j) >= 67.5) && (angle_matrix(i,j) < 112.5) || ...
                    (angle_matrix(i,j) >= 247.5) && (angle_matrix(i,j) < 292.5))
                    % Round anything around 90 or 270 to 90
                    angle_matrix(i,j) = 90;
            elseif ((angle_matrix(i,j) >= 112.5) && (angle_matrix(i,j) < 157.5) || ...
                    (angle_matrix(i,j) >= 292.5) && (angle_matrix(i,j) < 337.5))
                    % Round anything around 135 or 315 to 135
                    angle_matrix(i,j) = 135;
            end
        end
    end

    [X, Y] = size(c_img);
    % Initial result
    result = zeros(X,Y);
```

```
%     Compare if magnitude of pixel is greater than surrounding pixels
% if not set to zero
   for i=2:X-2
      for j=2:Y-2
         if angle_matrix == 0
            if (magn(i,j) >= magn(i,j+1)) && (magn(i,j) >= magn(i,j-1))
               result(i,j) = magn(i,j);
            else
               result(i,j)=0;
            end
         elseif angle_matrix == 45
            if (magn(i,j) >= magn(i+1,j+1)) && (magn(i,j) >= magn(i-1,j-1))
               result(i,j) = magn(i,j);
            else
               result(i,j)=0;
            end
         elseif angle_matrix == 90
            if (magn(i,j) >= magn(i+1,j)) && (magn(i,j) >= magn(i,j-1))
               result(i,j) = magn(i,j);
            else
               result(i,j)=0;
            end
         elseif angle_matrix == 135
            if (magn(i,j) >= magn(i+1,j-1)) && (magn(i,j) >= magn(i-1,j+1))
               result(i,j) = magn(i,j);
            else
               result(i,j)=0;
            end
         end
      end
   end

%     Normalize results
   result = result/max(result(:));
end
```