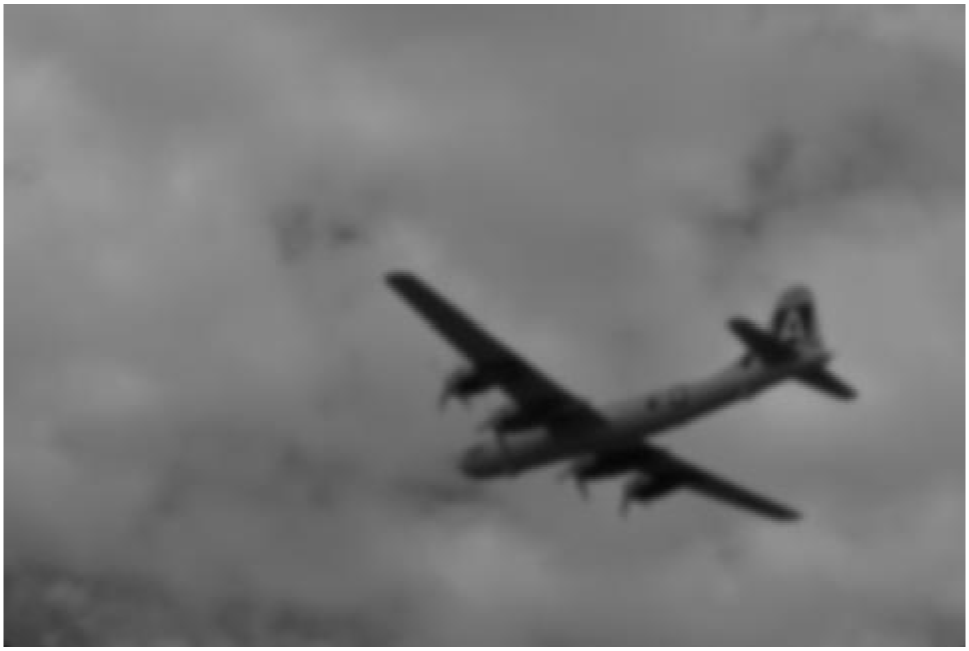
Ethan Che
CS558-A
Homework 1
09/30/2021
I pledge my honor that I have abided by the Stevens Honor System.

Source code on last pages

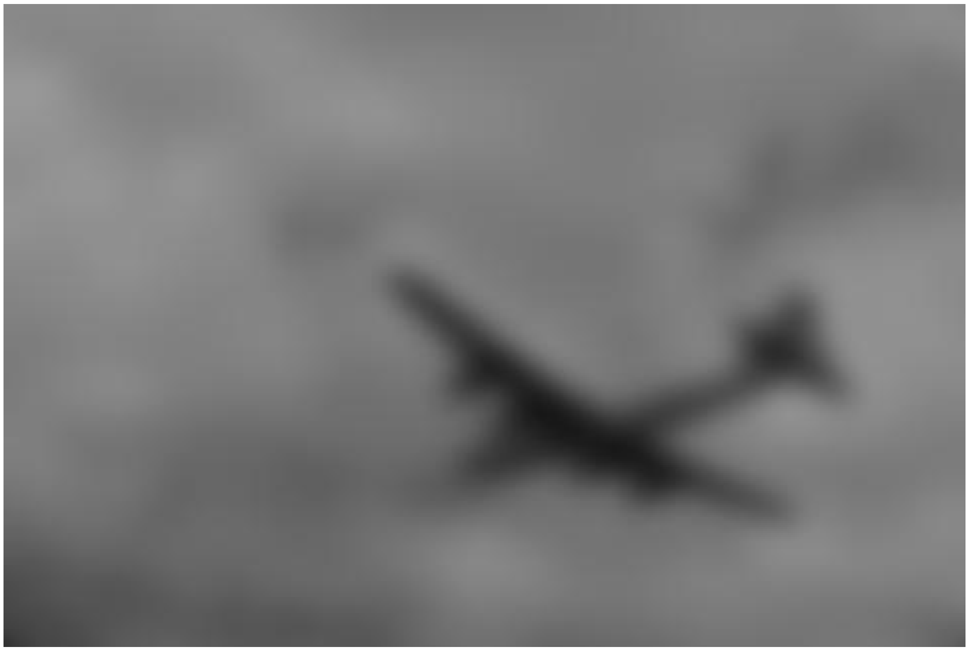
- I coded this assignment in Matlab, so the script should be run inside of Matlab.
- You can either call the script with no input arguments (in this case, it will run with default settings) or with two arguments (‘filepath’ ‘sigma’)
    - ex: hw1_eche
    - ex: hw1_eche ‘red.pgm’ ‘3’
- If for some reason Matlab crashes during execution, reload the program.
- My program displays the images side by side upon completion and also saves the new images as jpg files in the same directory as the script.

Gauss Filter

sigma = 2

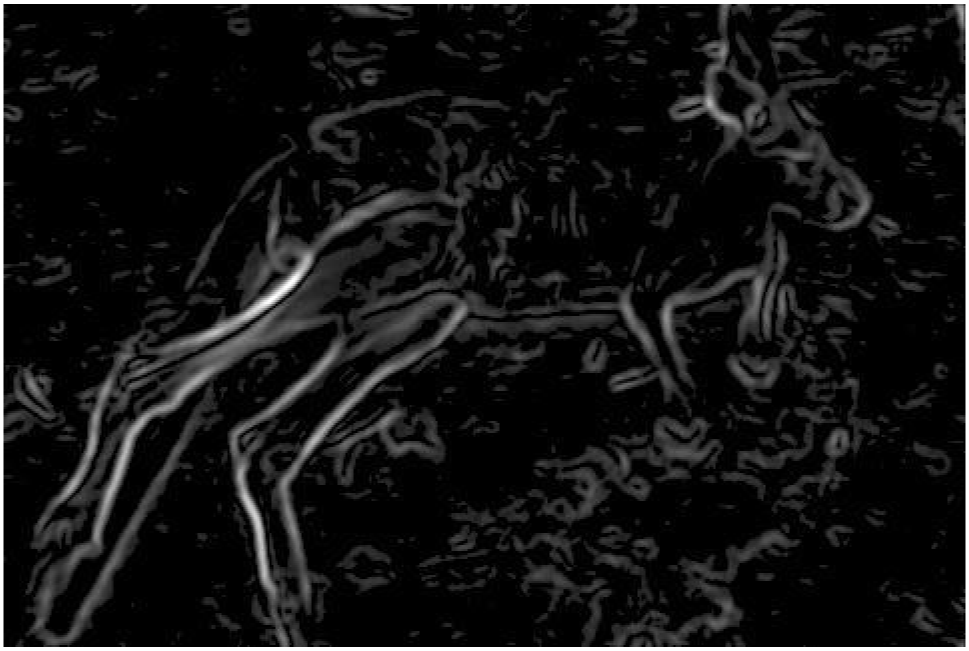


sigma = 8



Sobel Filter

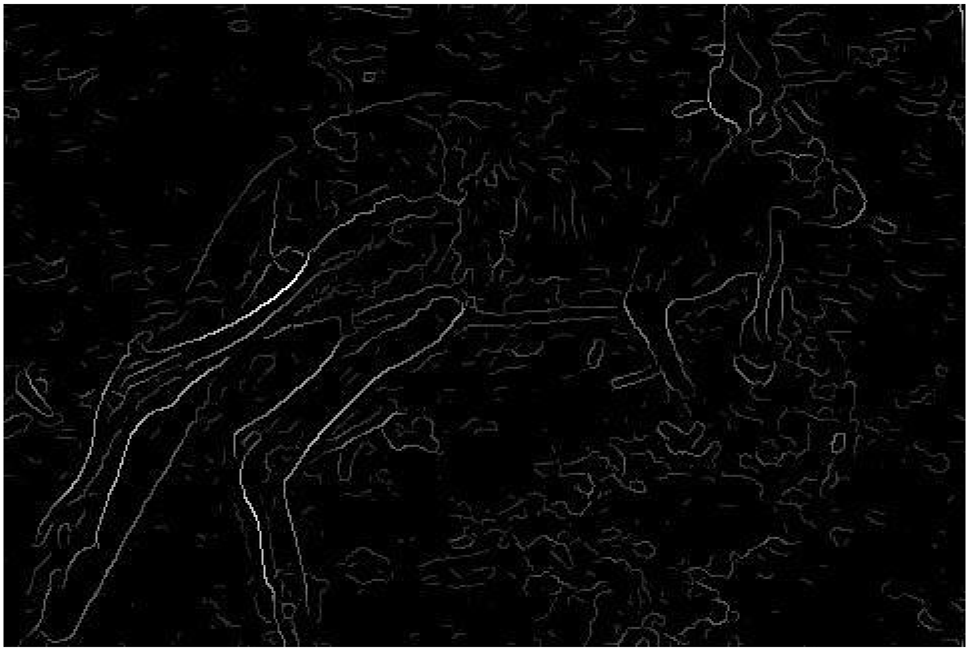
Sigma = 2 for each image below. The images were run through the Gauss filter before running them through the Sobel filter.

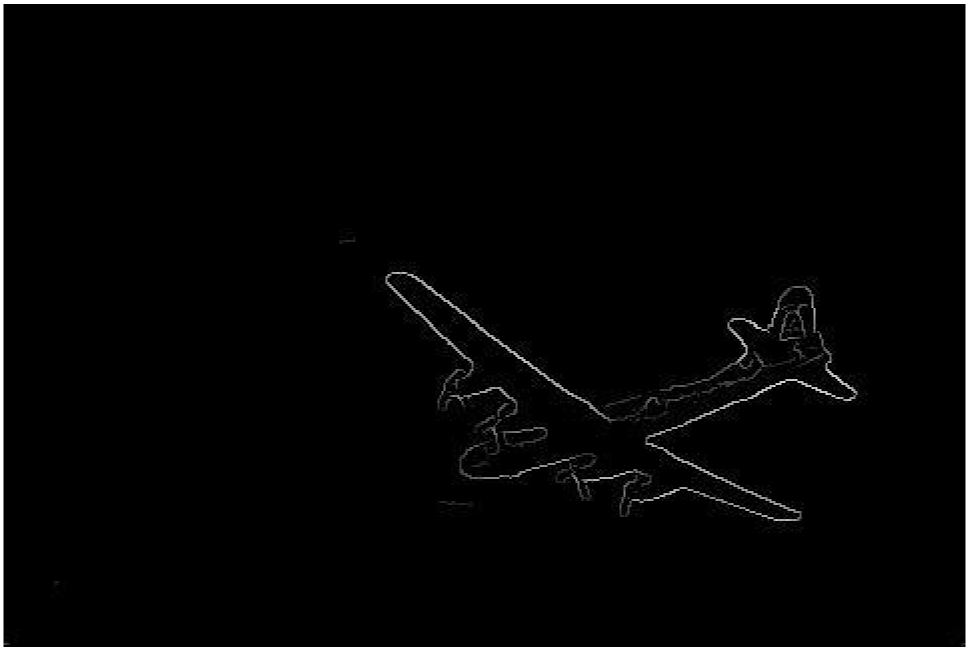


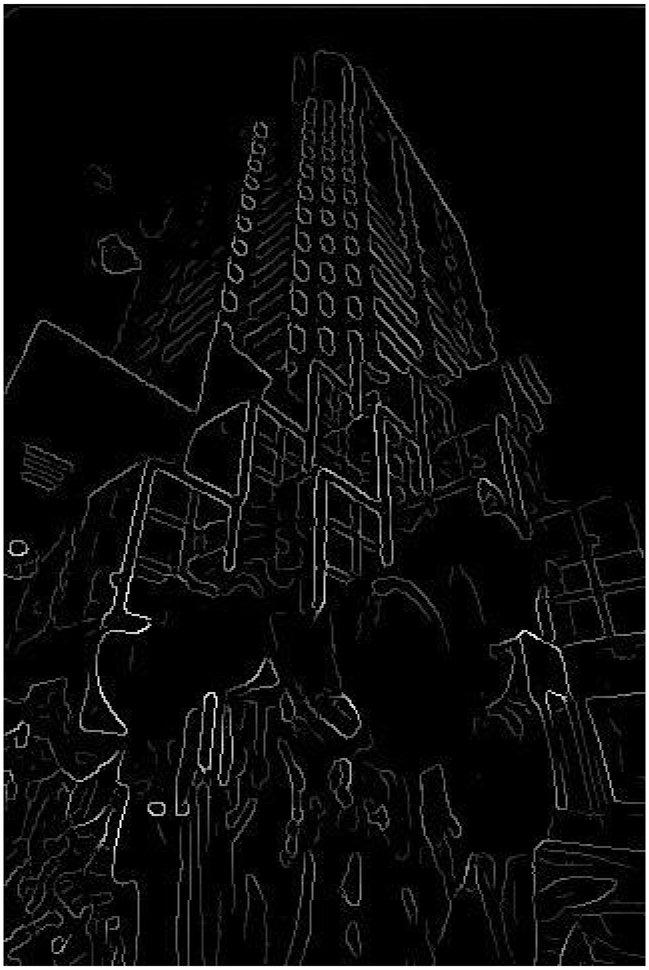
A

NMS

The NMS filter was applied to each of the three images from the section above







Source code starts on next page

```
function main(filename, stdv)

    if nargin == 0
        filename = 'plane.pgm';
        stdv = '2';
    end

    img = imread(filename,'pgm'); %Reads in pgm image

    rowsize = size(img, 1);
    colsize = size(img, 2);

    stdv = str2num(stdv);

    temp = 2*floor(stdv/2)+1; %Standard deviation, rounded to next odd number
    width = 6*temp;
    width = 2*floor(width/2)+1;

    add = floor(width/2); %How many pixels have to add aropund edge. Change the first number.

    extended = extend_edges(img, add);

    filtered_image = cast(gauss_filter(rowsize, colsize, extended, stdv, width), 'uint8');
    imwrite(filtered_image, 'output_gauss.jpg');

    subplot(2,2,1);
    imshow(img);
    title('Input Image');

    subplot(2,2,2);
    imshow('output_gauss.jpg');
    title('Gauss Filtered Image');

    gauss_extend = extend_edges(filtered_image, 1);
    [sobel, grad] = sobel_filter(rowsize, colsize, gauss_extend);
    imwrite(cast(sobel, 'uint8'),'output_sobel.jpg');
    subplot(2,2,3);
    imshow('output_sobel.jpg');
    title('Sobel Filtered Image');

    nms = cast(nms_filter(sobel, grad),'uint8');
    imwrite(nms,'output_nms.jpg');
    subplot(2,2,4);
    imshow('output_nms.jpg');
    title('NMS Applied to Sobel');


end

function filtered_image = gauss_filter(im_r, im_c, extended_image, stdv, fsz) %fsz is the filter
size, 5 means 5x5, for example
    filtered_image = zeros(im_r, im_c); %Size of regular image
    mask = zeros(fsz, fsz);
    mid = ceil(fsz/2); %Center of matrix. fsv should be odd
    total = 0;
    for i = 1:fsz
        for j = 1:fsz
            temp = gauss_eq(stdv, i-mid, j-mid);
            mask(i,j) = temp;
            total = total+temp;
        end
    end
    mask = mask./total;
    %disp(sum(sum(mask)));
    offset = floor(fsz/2); %Half of the filter width rounded down
    for i = 1:im_r
        for j = 1:im_c
            %move filter over each pixel in extended image, where original
            %image is
            filtered_image(i, j) = get_filter_val(extended_image,i+offset,j+offset,mask,offset);
        end
    end
end

function [edg_img, grad_matrix] = sobel_filter(im_r, im_c, extended_image) %Gets sobel val of
pixel
    %Also returns the matrix of gradient directions for each pixel, to be
    %used in NMS
    edg_img = zeros(im_r, im_c); %Size of regular image
    grad_matrix = zeros(im_r, im_c);
    mask_x = [-1 0 1; -2 0 2; -1 0 1];
    mask_y = [1 2 1; 0 0 0; -1 -2 -1];
    offset = 1; %Sobel filter is 3x3, so offset is only 1
    for i = 1:im_r
        for j = 1:im_c
            %move filter over each pixel in extended image, where original
            %image is
            xval = get_filter_val(extended_image,i+offset,j+offset,mask_x,offset);
            yval = get_filter_val(extended_image,i+offset,j+offset,mask_y,offset);
            total = sqrt((xval*xval) + (yval*yval));
            grad_matrix(i,j) = atand(yval/xval);
            if (total < 30)
                edg_img(i,j) = 0;
            else
                edg_img(i,j) = total;
            end
        end
    end
end

function gauss = gauss_eq(stdv, x,y) %Calculates the value of the gaussian distribution, with x
and y as offsets from center
    a = 1/(2*pi*(stdv*stdv));
    pow = (x*x)+(y*y);
    pow = pow/(-2*stdv*stdv);
    gauss = a*exp(pow);
end

function suppressed = nms_filter(edges, grad_matrix)

    im_r = size(edges, 1);
    im_c = size(edges, 2);
    suppressed = zeros(im_r, im_c);

    %extend edges so we can check each pixel surrounding a pixel
    add = 1;
    extended_edges = extend_edges(edges, add);
    for i = 1+add:im_r+add
        for j = 1+add:im_c+add
            grad_dir = grad_matrix(i-add, j-add);
            if  (grad_dir < 22.5 && grad_dir > -22.5) %If the gradient direction is horizontal
                %Check pixel to left and right
                if ((extended_edges(i,j) >= extended_edges(i, j-add)) && (extended_edges(i,j) >=
extended_edges(i, j+add)))

                   suppressed(i-add, j-add) = extended_edges(i,j);
                   %suppressed(i-add, j-add) = 255;
                   %disp(extended_edges(i,j));
                else
                    suppressed(i-add, j-add) = 0;
                end
            elseif (grad_dir < 67.5 && grad_dir >= 22.5) %If the gradient is right diagonal
            %Check pixels in TL and BR corners

            if ((extended_edges(i,j) >= extended_edges(i+add, j-add)) && (extended_edges(i,j) >=
extended_edges(i-add, j+add)))

               suppressed(i-add, j-add) = extended_edges(i,j);
              %disp(extended_edges(i,j));
            else
               suppressed(i-add, j-add) = 0;
            end
         elseif (grad_dir <= -22.5 && grad_dir > -67.5) %If the gradient is left diagonal
            %Check pixels in TR and BL corner
            if ((extended_edges(i,j) >= extended_edges(i-add, j-add)) && (extended_edges(i,j) >=
extended_edges(i+add, j+add)))
              suppressed(i-add, j-add) = extended_edges(i,j);
              %disp(extended_edges(i,j));
            else
               suppressed(i-add, j-add) = 0;
            end
         else %If the gradient is vertical
            %Check pixels above and below
            if ((extended_edges(i,j) >= extended_edges(i-add, j)) && (extended_edges(i,j) >=
extended_edges(i+add, j)))
              suppressed(i-add, j-add) = extended_edges(i,j);
              %disp(extended_edges(i,j));
            else
               suppressed(i-add, j-add) = 0;
            end
         end
      end
   end

end

%Worls for gauss and sobel
function val = get_filter_val(extended_image, r, c, filter, offset)
   val = 0;
   subimg = extended_image((r-offset):(r+offset), (c-offset):(c+offset)); %Part of extended image
that is being filtered
   for i = 1:size(subimg, 1)
      for j = 1:size(subimg, 2)
         filter_val = filter(i,j); %Value from the filter
         curr_val = subimg(i,j); %Value of the actual pixel
         val = val + (filter_val * curr_val);
```

```
        end
    end
end



function extended = extend_edges(img, add) %Extends image so we can apply filters

    rowsize = size(img, 1);
    colsize = size(img, 2);
    top = zeros(add,colsize); %Top pixels
    bottom = zeros(add,colsize); %Bottom pixels
    left = zeros(rowsize, add); %Left pixels
    right = zeros(rowsize, add); %Right pixels

    %Expand top and bottom row of pixels
    for c = 1:colsize
        for r = 1:add
            top(r,c) = img(1,c);
            bottom(r,c) = img(rowsize,c);
        end
    end

    for c = 1:add
        for r = 1:rowsize
            left(r,c) = img(r,1);
            right(r,c) = img(r,colsize);
        end
    end

    %Now, need to add the four corners where the pixels weren't extended
    tl = top(1,1); %Top left corner value
    tr = top(1,colsize); %Top right corner value
    bl = bottom(1,1); %Bottom left corner value
    br = bottom(1,colsize); %Bottom right corner value

    %Arrays that hold the corner pixels
    atl = zeros(add,add);
    atr = zeros(add,add);
    abl = zeros(add,add);
    abr = zeros(add,add);
    for i = 1:add
        for j = 1:add
            atl(i,j) = tl;
            atr(i,j) = tr;
            abl(i,j) = bl;
            abr(i,j) = br;
        end
    end
    extended = ones(rowsize+add+add, colsize+add+add); %Image extended out
    extended = extended.*-1; %Initialize each element to -1 so we know which elements have
been changed

    top = [atl top atr];
    bottom = [abl bottom abr];
    %disp(bottom);
    %Add top row
    for i = 1:add
        for j = 1:size(extended,2)
            extended(i,j) = top(i,j);
        end
    end
    %Add bottom row
    for i = (size(extended,1)-add+1):size(extended,1)
        r = 1;
        for j = 1:size(extended,2)
            extended(i,j) = bottom(r,j);
        end
        r = r+1;
    end
    %Add left and right rows
    for i = 1+add:size(extended,1)-add
        for j = 1:add
            extended(i,j) = left(i-add,j);
        end
        c = 1;
        for j = (size(extended,2)-add+1):size(extended,2)
            extended(i,j) = right(i-add,c);
        end
        c = c+1;
    end
    %Add back in original image
    for i = 1:rowsize
        for j = 1:colsize
            extended(i+add,j+add) = img(i,j);
        end
    end
```

end