Ethan Che
CS558-A
Homework 2
10/22/2021
I pledge my honor that I have abided by the Stevens Honor System.

Instructions

- This script will either run RANSAC line detection or a Hough Transform to detect lines. RANSAC will take longer to run than the Hough Transform.
- The script should be ran as follows: hw2_eche ‘image.name’ ‘r|h’
- If RANSAC is ran
    - You must enter the threshold for the determinant of the Hessian. Recommended value is 5000.
    - You must enter the distance threshold. Recommended value is 4.
    - You must enter the required number of inliers for the line to be considered “good”. Recommended value is 220.
    -
```
>> hw2_eche 'road.png' 'r'
Enter the threshold for the determinant of the Hessian.
5000
Enter the distance threshold.
4
Enter the required number of inliers.
220
```
- If Hough Transform is ran
    - You must enter the threshold for the determinant of the Hessian. Recommended value is 75000.
    - You must enter the rho and theta dimensions for the bins. Values of 1 for each produce good results. Values must also be at least 1.
    -
```
>> hw2_eche 'road.png' 'h'
Enter the threshold for the determinant of the Hessian.
75000
Enter the rho dimension of the bins.
1
Enter the theta dimension of the bins.
1
```

RANSAC

Hough Transform

Source code

```
function main (filename, method)

    cla reset;
    img = imread(filename); %Reads in pgm image

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

    thresh = input('Enter the threshold for the determinant of the Hessian.\n'); %Rec 5000 for
ransac, 75000 for hough

    casted = preprocess(img, rowsize, colsize, thresh);

    if (method == 'r')
        dist = input('Enter the distance threshold.\n'); %Rec. 4
        inlier = input('Enter the required number of inliers.\n'); %Rec. 220

        [l1, l2, i_x, i_y] = ransac(casted, rowsize, colsize, dist, inlier);
        imshow(img);
        axis off;
        hold on;

        %Plot the lines
        for i=1:size(l1, 1)
            x = linspace(0,colsize);
            coefficients = polyfit([l1(i,1) l2(i,1)], [l1(i,2) l2(i,2)], 1);
            m = coefficients (1);
            b = coefficients (2);
            y=m*x+b;
            plot(x,y);
        end

        %Plot the inliers

        for i = 1:size(i_x, 2)
            plot(i_x(1,i), i_y(1,i), '-s', 'MarkerSize', 3, 'MarkerEdgeColor','red','MarkerFaceColor',[1 .6
.6]);
        end

    elseif (method == 'h')
        imshow(img);
        y=0;
        points = max(max(casted));
        dim_r = input('Enter the rho dimension of the bins.\n');
        dim_t = input('Enter the theta dimension of the bins.\n');
        axis off;
        hold on;
        [theta_list, rho_list] = hough(casted, rowsize, colsize, dim_r, dim_t);
```

```
        for i = 1:size(theta_list,2)
            theta = theta_list(i);
            rho = rho_list(i);
            x=linspace(0,colsize);
            y=(rho-x*cosd(theta))/(sind(theta));
            plot(x,y);
        end

    else
        disp('Invalid input');
    end



end

function casted = preprocess(image, im_r, im_c, thresh )

    %Gauss filter
    gauss_img = gauss_filter(im_r, im_c, image);
    [dx, dy] = sobel_filter(im_r, im_c, gauss_img);
    [dxx,dxy] = sobel_filter(im_r, im_c, dx);
    [dyx, dyy] = sobel_filter(im_r, im_c, dy); %dyx not needed

    det = dxx.*dyy-(dxy).^2; %Determinant

    %Threshold it

    for i = 1:size(det, 1)
        for j = 1:size(det, 2)
            if (det(i,j)<thresh)
                det(i,j) = 0;
            end
        end
    end

    %now apply nms in all directions

    for i = 2:size(det,1)-1
        for j = 2:size(det,2)-1
            curr = det(i,j);
            tl = det(i-1, j-1);
            tm = det(i-1,j);
            tr = det(i-1,j+1);
            left = det(i,j-1);
            right = det(i,j+1);
            bl = det(i+1, j-1);
            bm = det(i+1,j);
            br = det(i+1,j+1);

            arr = [curr tl tm tr left right bl bm br];
            m = max(arr);
```

```
        if (curr < m)
            det(i,j) = 0;
        end
    end
end

casted = cast(det, 'uint8');
imwrite(casted, 'output_det.jpg');




end

function [line_1, line_2, inlier_point_x, inlier_point_y] = ransac(image, im_r, im_c,
distance_thresh, in_thresh)

    %Step 0: Iterate over determinent image and add points to a list so we can
    %choose points at random
    points_i = [];
    points_j = [];
    inlier_point_x = [];
    inlier_point_y = [];

    num_lines = 0; %Number of good lines found
    line_1 = []; %First point of good line. Each item is itself a list [x y]. 2d
    line_2 = []; %Second point of good line
    %The good line will have the two points at identical indices in line_1
    %and 2

    point = max(max(image));

    while (num_lines < 4)
        for i = 1:im_r
            for j = 1:im_c
                if (image(i,j) == point)
                    points_i = [points_i i];
                    points_j = [points_j j];
                end
            end
        end

        len = size(points_i,2);

        %loop while number of good lines is less than 4

        %Step 1: Choose points at random
        ran = randperm(len,2);
```

```
i1 = points_i(ran(1));
j1 = points_j(ran(1));
i2 = points_i(ran(2));
j2 = points_j(ran(2));

%Find parameters of line. For each dot in hessian matrix, compute
%distance. remember, x is j and y is i. count number of inliers

slope = (i2-i1)/(j2-j1); %Slope of the line between the two points
b = i1 - slope*j1; %y-intercept of line
%Now we know the equation of the line. y = mx+b

%The location (0,0) is at top left corner.

%Now, iterate over image and count number of points that are inliers

count = 0; %Number of inliers


for i = 1:im_r
    for j = 1:im_c
        if (image(i,j) == point) %If there is a point
            t_x = j;    %Theoretical x, which is just the current x value
            t_y = slope*t_x + b; %Theoretical y, which is based off the equation
            distance = sqrt((t_x-j)^2 + (t_y-i)^2);
            if (distance < distance_thresh)
                count = count + 1;
            end
        end
    end
end
if (count >= in_thresh)
   num_lines = num_lines + 1;
   line_1 = [line_1; [j1 i1]];
   line_2 = [line_2; [j2 i2]];

   %Get rid of inliers only when you use the line
   for i = 1:im_r
       for j = 1:im_c
           if (image(i,j) == point) %If there is a point
               t_x = j;    %Theoretical x, which is just the current x value
               t_y = slope*t_x + b; %Theoretical y, which is based off the equation
               distance = sqrt((t_x-j)^2 + (t_y-i)^2);
               if (distance < distance_thresh)
                   inlier_point_x = [inlier_point_x j];
                   inlier_point_y = [inlier_point_y i];
                   %Get rid of inlier so other lines dont use them
                   image(i,j) = 0;
               end
           end
       end
    end
```

```
        end
    end

end

function [t_list, r_list] = hough(image, im_r, im_c, dim_r, dim_t)

    max_theta = round(180/dim_t)+1;
    max_rho = round((2*im_c+im_r)/dim_r)+1;
    bins = zeros(max_theta, max_rho); %H(theta, p)
    offset = round(im_c/dim_r);

    point = max(max(image));

    for i = 1:im_r
        for j = 1:im_c
            if (image(i,j) == point) %If we are at a point

                for t = 0:max_theta-1

                    p = (j*cosd(t)) + (i*sind(t)) + im_c; %im_c
                    p = round(p/dim_r); %Have to add offset to make indiex positive. make sure to
subtract out
                    bins(t+1,p) = bins(t+1,p)+1; %Have to add 1 to theta. make sure to subtract when
                    %finding equation of line.
                end

            end

        end
    end

    %Find local maxima

    localmax = islocalmax(bins);

    for i = 1:size(bins,1)
        for j = 1:size(bins,2)
            if (localmax(i,j) ~= 1)
                bins(i,j)=0;
            end
        end
    end




    t_list = [];
    r_list = [];


    for i=1:4
```

```
        [row, col] = find(ismember(bins, max(bins(:))));
        bins(row, col) = 0;
        th = (row(1)-1);
        th = th*dim_t;
        rh = (col(1)-offset);
        rh = rh*dim_r;
        t_list = [t_list th];
        r_list = [r_list rh];
    end


end







%---------------------------------------- Helpers ---------

function filtered_image = gauss_filter(im_r, im_c, image) %fsz is the filter size, 5 means 5x5, for
example
    %5x5 gauss filter with sigma=1
    mask = [0.003 0.013 0.022 0.013 0.003; 0.013 0.059 0.097 0.059 0.013; 0.022 0.097 0.159
0.097 0.022; 0.013 0.059 0.097 0.059 0.013; 0.003 0.013 0.022 0.013 0.003];
    offset = 2;

    filtered_image = image;

    for i = 1+offset:im_r-offset
        for j = 1+offset:im_c-offset
            %move filter over each pixel in extended image, where original
            %image is
            val = get_filter_val(image, i, j, mask, offset);
            filtered_image(i,j) = val;
        end
    end

end

function [dx, dy] = sobel_filter(im_r, im_c, image) %Gets sobel val of pixel
    %Also returns the matrix of gradient directions for each pixel, to be
    %used in NMS
    dx = zeros(im_r, im_c);
    dy = 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+offset:im_r-offset
        for j = 1+offset:im_c-offset
            %move filter over each pixel in extended image, where original
```

```
            %image is
            xval = get_filter_val(image,i,j,mask_x,offset);
            yval = get_filter_val(image,i,j,mask_y,offset);
            dx(i,j) = abs(double(xval));
            dy(i,j) = abs(double(yval));



        end
    end
end

function val = get_filter_val(image, r, c, filter, offset)
    val = 0;
    subimg = image((r-offset):(r+offset), (c-offset):(c+offset)); %Part of extended image that is
being filtered
    val = sum(dot(double(subimg),double(filter)));
end
```