Brianne Trollo
CS558: Computer Vision
10 October 2019

Assignment 2: Line Detection

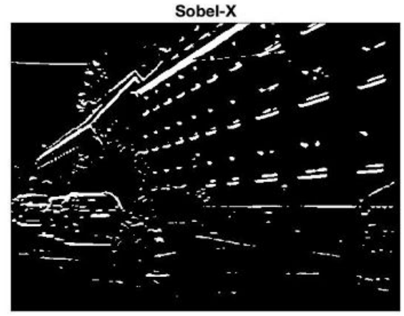
Problem 1: Preprocessing

Preprocessing for the image was done by completing the Hessian filter. The Hessian filter used the Gaussian filter to reduce noise in the image and Sobel filters as the derivative operators. These were completed without adding additional boundary pixels to the image. Any pixel that was too close to the edge of the image to fit the filter were ignored. The determinant of the Hessian was thresholded and the non-maximum suppression filter was used in 3x3 neighborhoods.

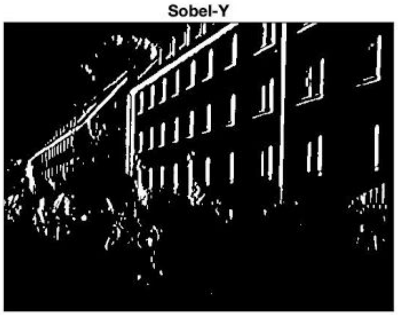
Original

Gaussian

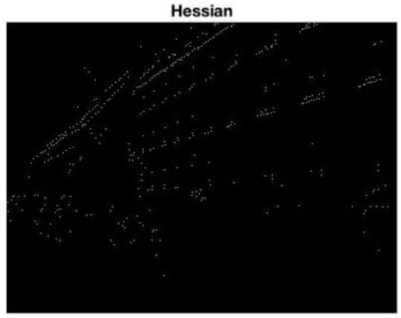
Sobel-X



Sobel-Y



Hessian



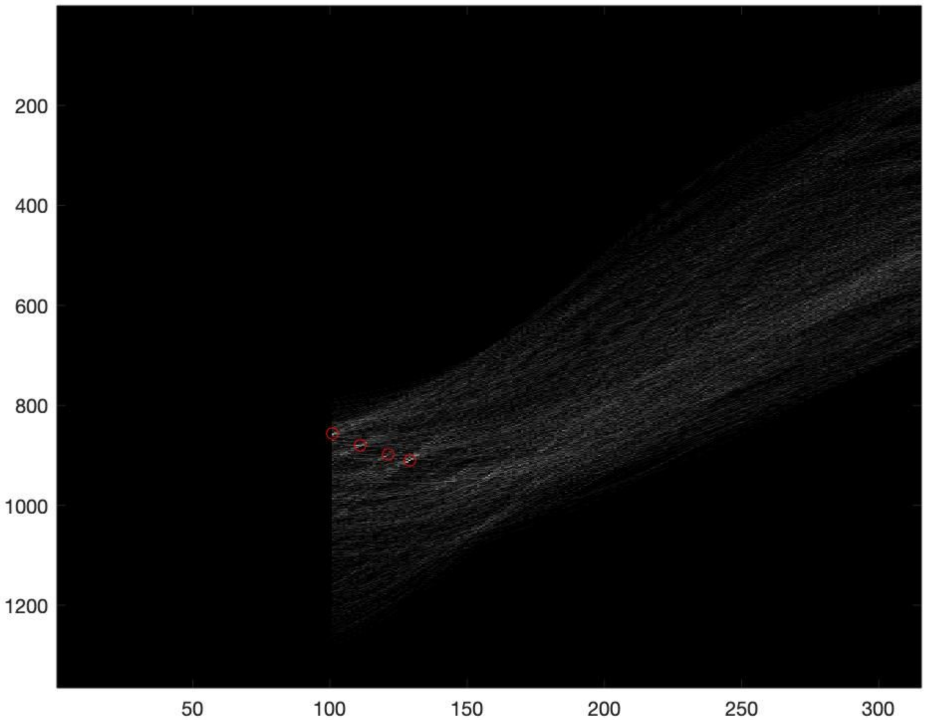
Problem 2: RANSAC

From the feature points provided by the results of the Hessian filter, two points are chosen at random. From those points a linear model is devised, ax + by = d, and then the distance from all other feature points to the line is calculated to determine the percentage of points, inliers, that support the model. The best four lines are plotted on the image along with the supporting inliers of those lines.

The image with the detected lines has displayed four lines and the inliers of those lines in yellow boxes. The lines are grouped into two groups of two, which makes one of them more difficult to see. The lower lines are purple and red. The higher lines are blue and a lighter blue.

Problem 3: Hough

The Hough transformation for detecting lines each of the feature points vote for a linear model. The accumulator dimensions is determined by the input values theta and rho. To mark each vote in the accumulator, H, the calculated rho, rho = x*cos(theta) + y*sin(theta) for each theta 0 to pi, and theta are rounded. The accumulated votes can be seen below with the lines of most support and the highest number of inliers circled in red. The second image below shows the four lines with the greatest support superimposed on the original image.


200
400
600
800
1000
1200
50
100
150
200
250
300

Code:

```
% Brianne Trollo
% CS 558: Computer Vision
% 10 October 2019
% Assignment 2: Line Detection

function main()
    close all;
    clear variables;

    sigma = 1;
    threshold = 20;
    edg = "none";
    thres_h = 5;

    img = imread("road.png");
    figure(1);
    subplot(2, 3, 1);
    imshow(img);
    title("Original");

    % Problem 1: Pre-processing
    % Step 1: Apply Gaussian
    img_g = myfilter(img, sigma, threshold, "gaussian", edg);
    subplot(2, 3, 2);
    imshow(img_g, []);
    title("Gaussian");
    % Step 2: Get Sobel fitlers
    % Sobel - X
    img_sx = myfilter(img_g, sigma, threshold, "sobel-x", edg);
    subplot(2, 3, 3);
    imshow(img_sx);
    title("Sobel-X");
    % Sobel - Y
    img_sy = myfilter(img_g, sigma, threshold, "sobel-y", edg);
    subplot(2, 3, 4);
    imshow(img_sy);
    title("Sobel-Y");
    % Step 3: Threshold the determinant of the Hessian & Step 4: Apply Non-maximum
suppression in 3x3 neighbors
    img_hes = hessian(img, sigma, threshold, thres_h);
    subplot(2, 3, 5);
    imshow(img_hes);
```

```
    title("Hessian");
    figure(6);
    imshow(img_hes);

    % Problem 2: RANSAC
    n_lines = 4;  % Four lines
    t = 0.5;      % distance threshold
    s = 2;      % points to find line
    p = 0.95;    % probability for inlier

    myransac(img, img_hes, t, s, p, n_lines);

    % Problem 3: Hough Transform
    rho = 1;
    theta = 0.01;
    myhough(img, img_hes, theta, rho, n_lines);

end

function bordered = myborder(edg, img, sz)
    % edg = edge type (replicate, clip, none)
    % img = original image
    % sz = size of border to add
% Replicate boundary pixels
    img = double(img);
    [x, y] = size(img);
    if strcmp(edg, "replicate")
        bordered = meshgrid(x+sz*2, y+sz*2);
        imgx = 1;
        imgy = 1;
        for i=1:(sz*2)+x
            for j=1:(sz*2)+y
                if i <= sz && j <= sz
                    bordered(i,j) = img(1,1);
                elseif i <= sz && j > y+sz
                    bordered(i,j) = img(1,y);
                elseif i > x+sz && j <= sz
                    bordered(i,j) = img(x, 1);
                elseif i> x+sz && j>y+sz
                    bordered(i,j) = img(x,y);
                elseif j<=sz && i > sz
                    bordered(i,j) = img(imgx, 1);
                elseif j > y+sz && i > sz
```

```
                bordered(i,j) = img(imgx, y);
            elseif j > sz && i <= sz
                bordered(i,j) = img(1, imgy);
                imgy = imgy + 1;
            elseif i > x+sz && j > sz
                bordered(i,j) = img(x, imgy);
                imgy = imgy+1;
            else
                bordered(i,j)=img(imgx, imgy);
                imgy = imgy+1;
            end
        end
        imgy = 1;
        if i > sz
            imgx = imgx+1;
        end
    end
%         Add border of zeros
  elseif strcmp(edg, "clip")
    bordered = meshgrid(x+sz*2, y+sz*2);
    imgx = 1;
    imgy = 1;
    for i=1:(sz*2)+x
        for j=1:(sz*2)+y
            if i <= sz || j <= sz || i > x+sz || j > y+sz
                bordered(i,j)=0;
            else
                bordered(i,j)=img(imgx, imgy);
                imgy = imgy+1;
            end
        end
        if i > sz
            imgx = imgx+1;
            imgy = 1;
        end
    end
 % No boundary
  elseif strcmp(edg, "none")
    bordered = img;
  end
end
```

```
function result = myfilter(img, sigma, threshold, filt, edg)
    % img = original image
    % sigma = gaussian sigma filter
    % threshold = threshold for sobel filter
    % edg = what to do with boundary pixels
    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./co_sum;
        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

    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


    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

  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

  end
end

% Non-maximum Suppression
function result = mynms(img, sobel_x, sobel_y)
  % img = original image
  % sobel_x = horizontal sobel filtered result
  % sobel_y = vertical sobel filtered result
  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

function result = mynms2(img)
    % img = preprocessed image

    % Non-maximum suppression applied in 3x3 neighbors
    [x,y] = size(img);
    result = zeros(x, y);
    for i=2:x-1
        for j=2:y-1
            neighbors = img(i-1:i+1, j-1:j+1);
            % if img(i,j) is max - is added to resulting image
            if max(neighbors(:)) == img(i,j)
                result(i,j) = img(i, j);
            end
        end
    end
end
```

```
function result = hessian(img, sigma, threshold, thres_h)
   % img = image
   % sigma = for gaussian filter
   % threshold = for sobel filter
   % thres_h = for hessian filter

   edg = "none";

   % Gaussian smoothing
   G = myfilter(img, sigma, threshold, "gaussian", edg);

   % First Derivative
   Gx = myfilter(G, sigma, threshold, "sobel-x", edg);
   Gy = myfilter(G, sigma, threshold, "sobel-y", edg);
   % Second Derivative
   Gxx = myfilter(Gx, sigma, threshold, "sobel-x", edg);
   Gxy = myfilter(Gy, sigma, threshold, "sobel-x", edg);
   Gyy = myfilter(Gy, sigma, threshold, "sobel-y", edg);

   % Determinant of Hessian
   determinant = (Gxx.*Gyy)-((Gxy).^2);

   % dimensions of img
   [x,y] = size(img);

   % dimensions of determinant
   [w, h] = size(determinant);

   % threshold the determinant
   for i=1:w
      for j=1:h
         if determinant(i,j) < thres_h
            determinant(i,j) = 0;
         end
      end
   end

   % Apply Non-maximum suppression
   result = mynms2(determinant);
end

function b = distToLine(p, l)
```

```
%   l = line [a b d]
%   p = point [x y]
   b = abs(l(1)*p(1)+l(2)*p(2)-l(3)) / sqrt(l(1)^2 + l(2)^2);
end


function myransac(img, hes, t, s, pc, num_lines)
%     img = original image
%     hes = img after hessian applied
%     t = distance threshold
%     s = points to fine shape
%     pc = confidence
%     num_lines = number of lines to calculate

   % Find feature points in hessian
   [y, x] = find(hes > 0);

   % Initialize figure
   f = figure; imshow(img), hold on;

   f_points = [x y];
   total_points = length(f_points);

   % Run ransac to create num_lines
   for n_lines=1:num_lines
      count = 0;
      N = Inf;
      best_inliers_count = 0;
      best_inliers_idx = [];
      best_line = [];

      while N > count
         p1_i = 0;
         p2_i = 0;
         % Step 1: Randomly select minimal subset of points
         % randomly pick 2 different points from f_points array
         while (p1_i == p2_i || p1_i == 0 || p2_i == 0)
            p1_i = ceil(rand*total_points);
            p2_i = ceil(rand*total_points);
         end

         % get point from f_point using random index
         p1 = f_points(p1_i, :);
```

```
    p2 = f_points(p2_i, :);

    % Step 2: Hypothesis a model : ax + by = d
    % y = mx + c
    % Slope
    m = (p1(2)-p2(2))/(p1(1)-p1(1));
    % y-intercept
    c = p1(2) - m*p1(1);

    % ax + by = d
    a = p1(2)-p2(2);
    b = p2(1)-p1(1);
    d = p1(2)*p2(1)-p1(1)*p2(2);

    % Step 3: Compute error function
    % && Step 4: Select points consistent with model - distance
    % threshold

    % Calculate the distance from each point to the line
    dist = Inf(size(f_points,1), 1);
    for pt=1:length(f_points)
        cur_point = f_points(pt, :);
        dist(pt) = distToLine(cur_point, [a b d]);
    end
    % Find inliers - points within the distance threshold of the
    % line
    inliers = find(dist <= t);

    % Keep track of greatest number of inliers
    inliers_count = size(inliers, 1);
    if inliers_count > best_inliers_count
        best_inliers_count = inliers_count;
        best_inliers_idx = inliers;
        best_line = [a b d];
    end

    % Step 5: Repeat hypothesize-and-verify loop
    % Repeat N times
    e = 1 - (inliers_count/total_points);
    N = log(1-pc)/log(1-power((1-e),s));
    count = count + 1;
end
```

```
        % Get inlier points from indexes found
        i_points = f_points(best_inliers_idx, :);
        % Draw line on photo
        [~, idx1] = min(i_points(:, 1));
        [~, idx2] = max(i_points(:, 1));
        figure(f), hold on;
        plot(i_points([idx1 idx2],1), i_points([idx1 idx2],2), "LineWidth", 1), hold on;

        lx = i_points(idx1,1):i_points(idx2,1);
        ly = (best_line(3)-best_line(1).*lx)./best_line(2);
        plot(lx, ly, "LineWidth", 1), hold on;

        for i=1:best_inliers_count
            px = i_points(i,1);
            py = i_points(i,2);
            [xx, yy] = meshgrid(px-1:px+1, py-1:py+1);
            hold on;
            scatter(xx(:), yy(:), 'square', 'y');
        end
        hold off;
    end
end


function myhough(img, hes, theta, rho, num_lines)
% img = original image
% hes = image after hessian filter
% theta = dimension of bin of accumulator theta
% rho = dimension of bin of accumulator rho
% num_lines = number of lines to display on image
    [X, Y] = size(img);
     % Find feature points in hessian
    [y, x] = find(hes > 0);

    % Feature pooints
    f_points = [x y];

    % Number of feature points
    total_points = length(f_points);

    maximum_rho = floor(sqrt(X^2 + Y^2));
    rho = -maximum_rho:rho:maximum_rho;
    xtheta = 0:theta:pi;
```

```
% Accumulator
Hheight = numel(rho);
Hwidth = numel(xtheta);
H = zeros(Hheight, Hwidth);

% Voting
for i=1:total_points
    % For each feature point (x,y) in image
    x = f_points(i,1);
    y = f_points(i,2);
    % For theta = 0 to 180 (pi)
    for t=1:theta:pi
        r = x*cos(t) + y*sin(t);
        it = round(t*100);
        ir = round(r + Hheight/2);
        H(ir, it) = H(ir, it)+1;
    end
end



% Peak Votes plot
f2 = figure; imagesc(H), colormap gray, hold on;

tempH = H;
% Circle highest supported num_lines lines
for n_lines=1:num_lines
    [~, idx] = max(tempH(:));

    ir = mod(idx, Hheight);
    it = (idx-ir)/Hheight+1;

    figure(f2); scatter(it, ir, 'r');

    tempH(ir-1:ir+1, it-1:it+1)=0;
end
hold off;
drawnow;


% Plot num_lines lines on image
f5 = figure; imshow(img), hold on;
```

```
    tempH = H;
    for n_lines=1:num_lines
        [~, idx] = max(tempH(:));

        ir = mod(idx, Hheight);
        it = (idx-ir)/Hheight+1;
        r = ir-Hheight/2;
        t = (it)/100.0;

        x=1:Y;
        y=(r-x.*cos(t))/sin(t);

        figure(f5); plot(x, y, 'LineWidth', 1.33), hold on;

        tempH(ir-1:ir+1, it-1:it+1)=0;
    end
    hold off;
    drawnow;
end
```