John Cinquegrana
CS – 558, Computer Vision

# Homework 1 – Edge Detection

The following is the resulting code from the RANSAC and Hough Transform assignement of Homework1, as well as short description and summations of each method and step.

```
clear;

%Reading in and showing the image
original = im2double( imread( 'road', 'png' ) );
rgb_original = cat(3, original, original, original);
imshow( original )
```

```
%Creating and using the Gaussian filter
sigma = 0.7;
dim = 2*ceil(2*sigma)+1;

gauss = fspecial('gaussian', dim, sigma);

smooth_image = my_filter( original, gauss );

imshow( smooth_image )
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

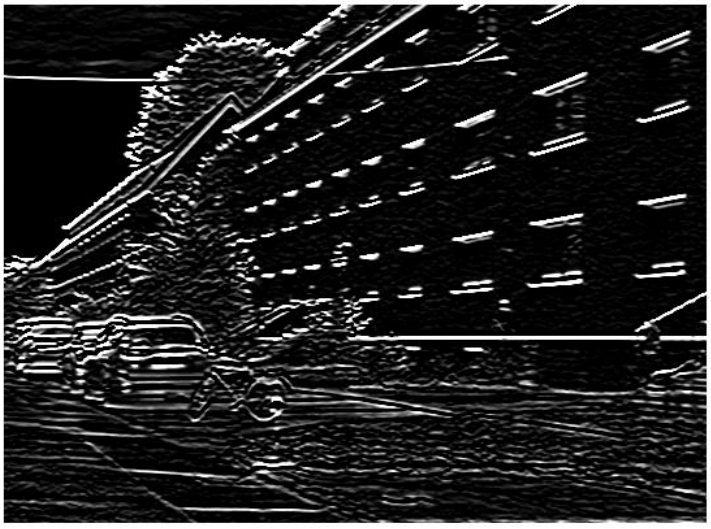
We smooth the image using the Gaussian Filter. This reduces noise and has a smoothing effect over the image.

```
%Cleaning environment
clear gauss sigma dim

%Running the sobel derivation
sobel_x = [-1 0 1; -2 0 2; -1 0 1];
sobel_y = [1 2 1; 0 0 0; -1 -2 -1];

I_y = my_filter( smooth_image, sobel_y );
I_x = my_filter( smooth_image, sobel_x );

%Second-Order derivatives
I_yy = my_filter( I_y, sobel_y );
I_xx = my_filter( I_x, sobel_x );
I_xy = my_filter( I_x, sobel_y );
imshow( I_xx )
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

```
imshow( I_yy )
```



```
imshow( I_xy )
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

We develop the partial derivatives of the image. We do this by running either horizontal or vertical Sobel filters over the image, depending on which directional derivative we want. We need these partial derivatives in order to compute the Hessian of the image.

```
%Retrieving the Hessian
Hess = (I_yy .* I_xx) - (I_xy .^ 2);

%Thresholding the Hessian
thresh = 5;
Hess( Hess < thresh ) = 0;
imshow( Hess )
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

The Hessian is the name given to the determinant of a specific matrix (as the code shows). After simply computing the Hessian for each pixel, we threshold it. In this any value we see under our threshold, we replace with a zero. As the edges in this picture were quite hard, our threshold was quite high.

```
%Non-Maximum suppresion
determinant = atan( I_y ./ I_x );
determinant = determinant( 1:size(Hess, 1), 1:size(Hess, 2) );
Hessian_thresholded = non_max_suppression( Hess, determinant );
imshow( Hessian_thresholded )
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

After thresholding the Hessian, we run the Non-Maximum suppression algorithm. We take a pixel and view its neighbors that are collinear with its gradient, of those three pixels, we keep only the highest, and set the rest to zero. This reduces clusters of pixels around most strong edges.

```
%Cleaning environment
clear I_x I_3y sobel_x sobel_y thresh

%Changing from an image object to a point-array
points = to_point_list( Hessian_thresholded );

%Superimpose the Hessian coords
Hessian_imposed = impose_circles( rgb_original, points );
imshow( Hessian_imposed )
```

The “to_points_list” function takes any picture and makes a matrix containing all coordinates where pixels do not have a zero value. Instead of using large images containing only a few worthwhile pixels, this allows us to work with a much smaller matrix that contains only the values we care about. This image are those values (The post-suppression Hessian values) superimposed as squares onto the original image.

```
%RANSAC the points
d = 5;
N = 200;
I = 20;
line1 = RANSAC( points, N, d, I);
new_points = remove_inliers( line1, points, d);
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

```
line2 = RANSAC( new_points, N, d, I );
new_points = remove_inliers( line2, new_points, d);
line3 = RANSAC( new_points, N, d, I );
new_points = remove_inliers( line3, new_points, d);
line4 = RANSAC( new_points, N, d, I );
RANSAC_lines = cat(1, line1, line2, line3, line4 )
```

RANSAC_lines = 4×4

| | | | |
|---|---|---|---|
| 116 | 65 | 55 | 267 |
| 48 | 174 | 240 | 16 |
| 161 | 24 | 410 | 68 |
| 193 | 103 | 59 | 280 |

Here, I apply the RANSAC algorithm to find four different lines. By using RANSAC to only find one individual line, then removing that lines outliers from our list of points, we completely remove the chance that we re-count any outliers or attempt to draw the same line twice. Any line we find with 20 or more inliers, is kept and refined into a proper model. Each model is represented as the two most-extreme points in the subset of inliers $[(x_1, y_1)\,(x_2, y_2)]$.

```
with_inliers = print_inliers( rgb_original, RANSAC_lines, points, d );
with_lines = draw_line_segment( with_inliers, RANSAC_lines );
imshow( with_lines )
```

These the RANSAC generated lines, and their inliers, superimposed onto the original image.

```
clear new_points line1 line2 line3 line4 N I with_inliers

%HOUGH transform
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

```
 hough = my_hough( points, size(Hessian_thresholded, 1), size(Hessian_thresholded,
2), 1 );
 imshow( hough );
```

This picture is the result of running the Hough Transform on the original points generated from the Hessian. The horizonal axis represents Theta,

I pledge my honor that I have abided by the Stevens Honor System.

and the vertical axis represents ro. The curved lines are generated because of the polar representation of lines that we use in the Hough Transform. The black portions of the image represent a line that would not have a single Hessian point as an inlier. The whiter a pixel is, the more inliers it's representative line would have.

```
hough_lines = get_lines( hough, size(Hessian_thresholded, 1),
size(Hessian_thresholded, 2) )
```

hough_lines = 4×2

```
   -0.8098  257.4205
   -0.7536  270.8473
   -0.9004  240.6107
   -1.0000  224.8600
```

These lines are the results of the four strongest lines in the Hough Transform. They are shown here in [m b] form to suggest the line equation of $y = mx + b$.

```
with_inliers = plot_hough_inliners( rgb_original, hough_lines, points, d );
with_hough_lines = draw_line( with_inliers, hough_lines);
imshow( with_hough_lines )
```

This is the result of plotting the four strongest lines form the Hough transform onto the original image. Again, the inliers for each line are plotted in yellow, and the lines themselves are plotted in blue. Unlike the RANSAC approach, these lines are not line segments, and therefore they stretch across the entirety of the image.

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

# Helper Functions

### Hough Transform

This is the “my_hough” function. This function receives the list of points from a Hessian algorithm, a maximum x and y value, as well as a resolution, and create a Hough Transform of an image. It creates buckets for ro and theta, and iterates through each point’s coordinates adding their corresponding ro and theta values into the final Hough Transform.

```
function img = my_hough( points, max_y, max_x, res )

    max_r = floor(norm([max_x-1,max_y-1]));
    r = -(max_r+1):res:(max_r+1);
    length_r = length(r);
    theta_length = 180 / res + 1;
    img = zeros(length_r,theta_length);
    for row = 1:size(points, 1)
        x = points( row, 1);
        y = points( row, 2);
        for j = 1:theta_length
            th = (-90 + res*j) * pi/180;
            r = x * cos(th) + y * sin (th);
            i = floor(r + (length_r-1)/2);
            img(i,j) = img(i,j) + 0.05;
        end
    end
end
```

### RANSAC

The RANSAC function receives a list of points very similarly to the Hough function, and refers to them as features. RANSAC randomly chooses two points from the set of features and tests a line segment model for an edge against those two points. It ranks these models by the number of inliers that is has and keeps those that it finds to be successful in this area.

```
function arg = RANSAC( features, N, d, I )
    arg = zeros( 5, 1 );
    % Line = x1, y1, x2, y2, #of_inliers
    loop_num = 0;

    while( loop_num < N )
        %recieve two points to base the model off of
        point1 = randi( [1 size(features,1)] );
        point2 = randi( [1 size(features,1)] );
        if ( point1 == point2 )
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

```
            continue;
        end
        line = zeros(1, 5);
        x1 = features( point1, 1 );
        y1 = features( point1, 2 );
        x2 = features( point2, 1 );
        y2 = features( point2, 2 );

        line(1) = x1;
        line(2) = y1;
        line(3) = x2;
        line(4) = y2;
        line(5) = inliers( line, features, d);
        if compare( line, arg ) >= 0
            arg = line;
        end

        loop_num = loop_num + 1;
    end
    arg = arg( 1, 1:4 );
    %Return arg

    %Determines order of lines
    function num = compare( line1, line2 )
        num = line1(5) - line2(5);
    end

    %calculates the number of inliers for any given line based off of a minimum
    %distance
    function num = inliers(line, points, d)
        num = 0;
        for iter = 1:size(points, 1)
            x = points( iter, 1 );
            y = points( iter, 2 );
            if d > distance( x, y, line(1), line(2), line(3), line(4) )
                num = num + 1;
            end
        end
        %return num
    end
    %end inliers function
end
```

## get_lines

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

This is one of the helper functions I used for the Hough transformation, and the function I had the most difficulty writing. This function takes, as input, the result of a Hough transform and outputs the specific lines that correspond to its four most prominent models. Translating from polar to cartesian coordinates proved problematic, as did exchanging from pixels in buckets, to the true theta and ro values that were necessary. It was achieved through simple imperative programming in the end; looping over each pixel to find the ones with the strongest models, and then looping over those models to clean them up into a useable form.

```
function arg = get_lines(hough)
%line in form = m b rating
    nro = size( hough, 1 );
    arg = zeros( 4, 3 );
    for some = 1:4
        for i = 1:size(hough, 2) %theta
            for j = 1:size(hough,1) %ro
                line = zeros(1, 3);
                line(1) = j;
                line(2) = i;
                line(3) = hough(j, i);
                if ( line(3) >= arg(some, 3) )
                    arg( some, 1:3) = line;
                end
            end
        end
        %Blank out the pixels around the line we picked, prevents us from
        %picking very similar lines
        hough( arg(some, 1)-1:arg(some, 1)+1, arg(some, 2)-1:arg(some, 2)+1) =
zeros(3, 3);
    end

    %arg = arg(1:4, 1:2);
    %Change all of the args into their actual theta and ro values
    for row = 1:4
        ro = arg(row, 1) - (nro-1)/2;
        th = arg(row, 2) * pi / 180;
        m = -(cos(th)/sin(th));
        b = sin(th)*ro - m*cos(th)*ro;
        arg(row, 1) = -1/m;
        arg(row, 2) = b;
    end
    arg = arg( 1:4, 1:2 );
    %return args
end
```

I pledge my honor that I have abided by the Stevens Honor System.

John Cinquegrana
CS – 558, Computer Vision

This is normal text

## Filter

The filter function is absolutely necessary to any computer vision assignment, no matter how trivial it may be. My filter function simply slides a window over each square of pixels and puts the resulting weight average into a pixel in a new image. The filter function reduces the size of the original image linearly with the size of the filter.

```
%the my_filter function
function A = my_filter(I, filter)
    %get height and weidth of image
    org_height=size(I,1);
    org_width=size(I,2);
    %get height and width of filter
    fil_height=size(filter,1);
    fil_width=size(filter,2);
    %create new image to hold output
    new_height = org_height - fil_height + 1;
    new_width = org_width - fil_width + 1;
    A = zeros( new_height , new_width);

    %Apply filter to the new image
    for y = 1:new_height
        for x = 1:new_width
            %For each individual point in the image
            result = 0;
            for yadj = 0:fil_height-1
                for xadj = 0:fil_width-1
                    %For each point in the filter
                    result = result + I( y+yadj, x+xadj ) * filter( yadj+1,
xadj+1 );
                end
            end
            A(y, x) = result;
        end
    end
    %Return the new image
end
```

I pledge my honor that I have abided by the Stevens Honor System.