Hough Transform

Lab Assignment #5

Ralphilou Tatoy

April 25, 2022

Abstract:

In this lab, using the Hough transformation technique, identification of lines, circles, etc. is possible. Specifically, using a that uses polar coordinates to detect lines in an image. Like the use of slope intercept form equation technique, we can use polar coordinates by using the equation $\rho = xcos(\theta) + ysin(\theta)$. Each degrees outputs a unique line when paired with rho values. Thus, conceptually, we can discern each line by using theta and rho values in the parametric space. The output determines the vote in each cell in the accumulator matrix which represents the parametric space. Each index in the accumulator cell is the number of points in a line in the image space. The most votes in the accumulator will be the most prominent line in the image with specified theta_out and rho_out.

Technical Directions:

The process starts with extracting the size of the input image. For the real image, it was necessary to use the canny method as it is not binary unlike the other images such as the horizontal. The canny method will still work for the horizontal type images. The calculation of the diagonal line to be used as 'rho max' is the square root of the width and length of the image. The function also uses the maximum theta value that will be used which is 90 degrees in each sign. This is necessary because indexing will be using these variables and the initialization of the accumulator matrix. The for loop I implemented for inputting every theta value uses negative values therefore the initialization of the maximum for both theta and rho is necessary as both values may go negative, and you can't access the accumulator matrix if the value is negative. Once the initialization of the accumulator matrix is complete, the two for loops start the process of extracting data into line detection data. The two for loops account for the process of assessing every pixel value. The calculation must account for all theta values for each pixel value. The theta correlates to the uniqueness of each line. The equation the function uses is $\rho = xcos(\theta) + ysin(\theta)$. In short, for every pixel location in an image, we must go from -89 to 90. The output rho means that there is a line that indicates that the accumulator for a specific theta/rho must increment by one. The accumulator matrix provides information on how many points are in a line in the image plane. The rho must be rounded as it is used to index the accumulator matrix counter of lines along with theta value. The counter will relay the intersection representing the number of points in line in the image plane. The prominent line is saved as the output via rho_out and theta_out.

Results

Overall, the most difficult part of this process is trying to get some sense of the x ad y coordinates. I feel that the indexing was the most difficult as it was interchangeable during lecture. Since the images are mostly binary, I didn't have to use kernel matrices to detect edges therefore it was an easy process. For the real image, I had to use some sort of kernel such as the canny method for my method to accurately output the correct estimated theta and rho with the prominent line. Horizontal line seems like the accumulator visual have some sort of connecting point in the middle. A middle point where each side meet. For the vertical line, the accumulator visual converse at both side at the edge of the image.

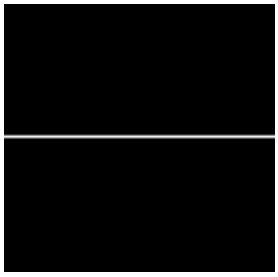

Figure 1: Horizontal Line

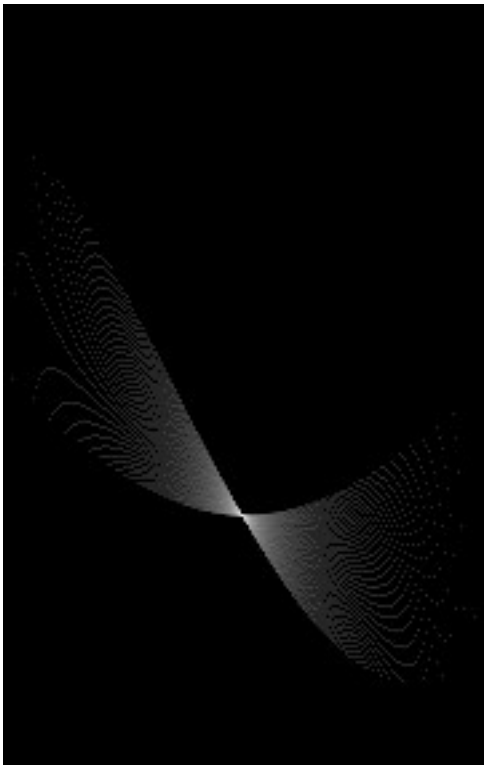

Figure 2: Horizontal Line Accumulator

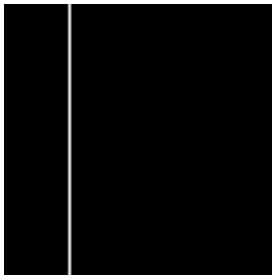

Figure 3: Vertical Line

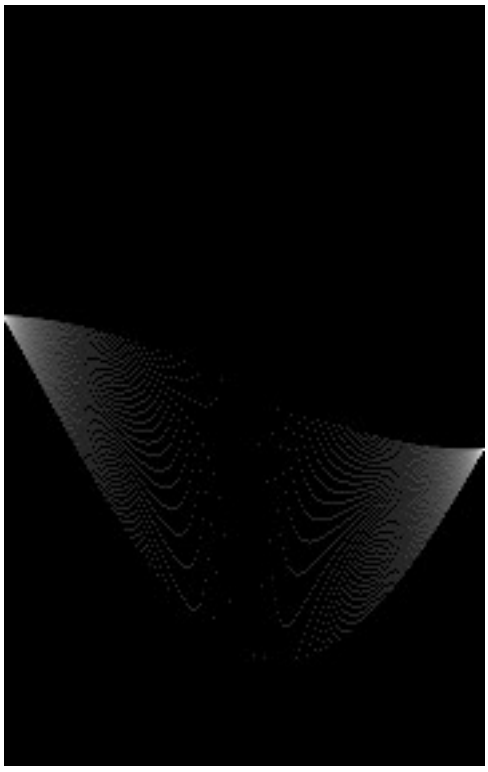

Figure 4: Vertical Line Accumulator

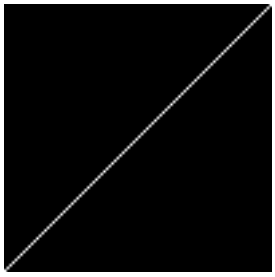

Figure 5: Positive Diagonal Line

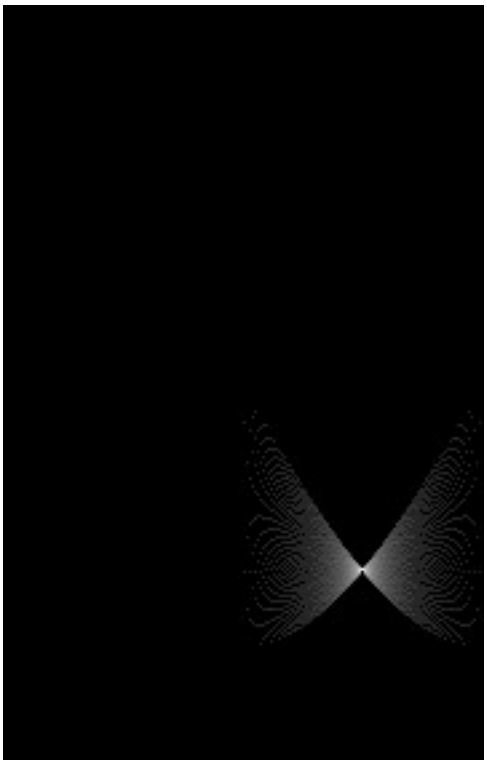

Figure 6: Positive Diagonal Line Accumulator

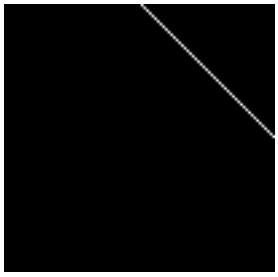

Figure 7: Negative Diagonal Line

Figure 8: Negative Diagonal Line

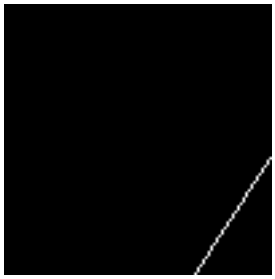

Figure 9: Random Line #1

Figure 10: Random Line #1 Accumulator

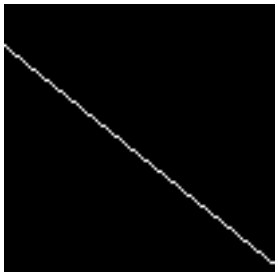

Figure 11: Random Line #2

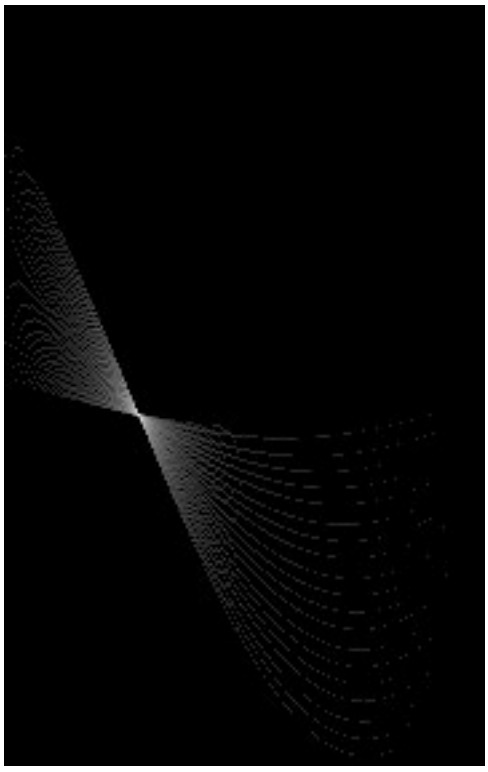

Figure 12: Random Line #2 Accumulator

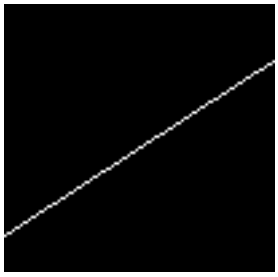

Figure 13: Random Line #3

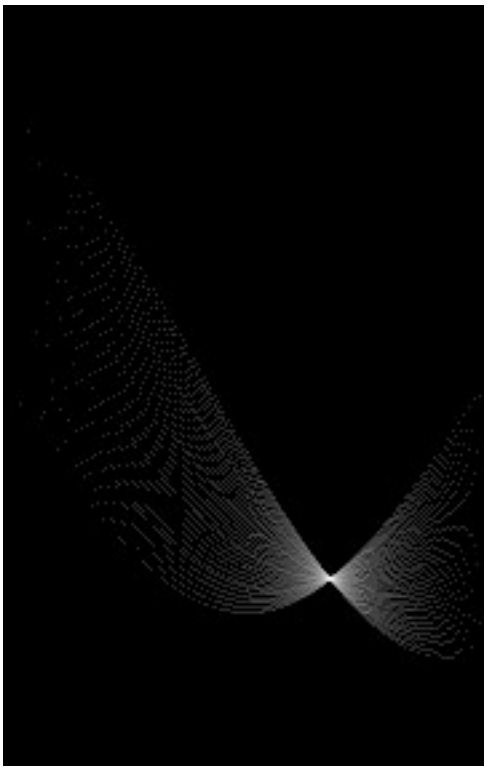

Figure 14: Random Line #3 Accumulator

Figure 15: Runway Image

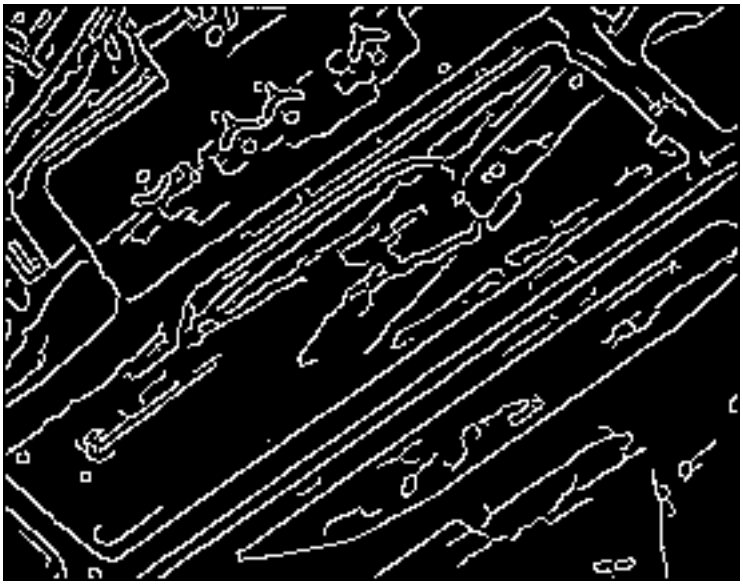

Figure 16: Runway Image Edge

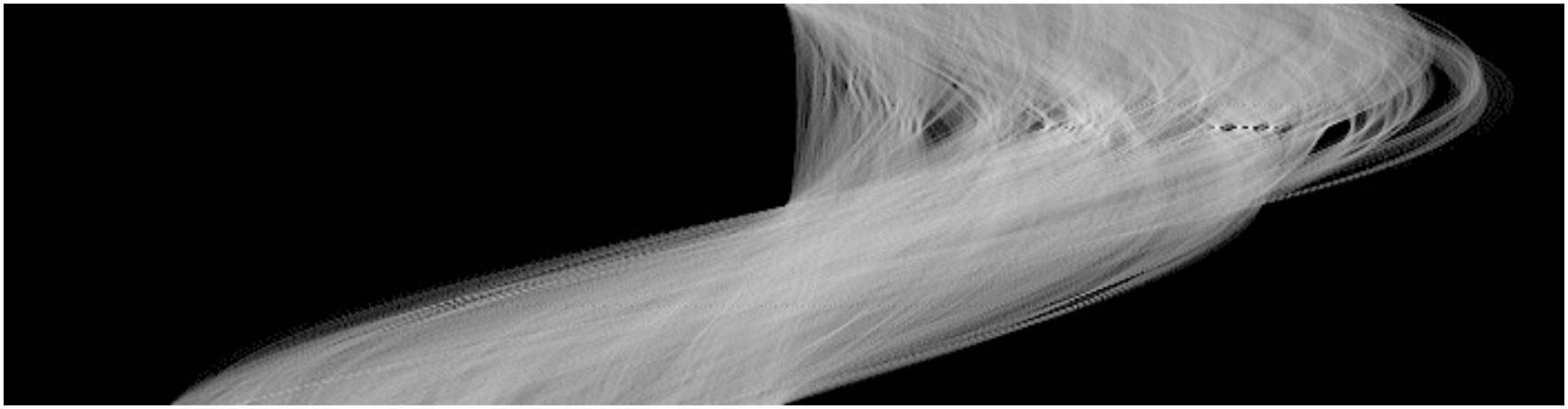

Figure 17: Runway Image Accumulator

Figure 18: Runway Image with Line without use of canny edge detection

```
>> test_horizontal_line
True theta = 0, true rho = 50
Estimated theta = 0, estimated rho = 50
>> test_vertical_line
True theta = 90, true rho = 25
Estimated theta = 90, estimated rho = 25
>> test_pos_diagonal_line
True theta = 45, true rho = 71
Estimated theta = 45, estimated rho = 71
>> test_neg_diagonal_line
True theta = -45, true rho = -35
Estimated theta = -45, estimated rho = -36
>> test_random_line
True theta = 57, true rho = 115
Estimated theta = 58, estimated rho = 115
>> test_random_line
True theta = -39, true rho = 13
Estimated theta = -39, estimated rho = 12
>> test_random_line
True theta = 33, true rho = 73
Estimated theta = 33, estimated rho = 73
>> test_real_image
Estimated theta = 35, estimated rho = 220
>>
```

```
function [theta_out, rho_out, accumulator] = hough_transform(i_edge)
% Computers the Hough Transform of the input image. The hough_transform is
% designed to detect lines. This function uses the parametric
% representation of a line = rho = x* cosd(theta) + y* sind(theta). The
% function returns theta_out, rho_out, and accumulator
%
% Syntax:
% [theta_out, rho_out, accumulator] = hough_transform(i_edge)
%
% Input:
% i_edge - Binary image
%
% Output:
% accumulator - parametrer space matrix whoe rows and columns corresponds
% to rho and theta values
%
% theta_out - theta value with the prominent line
% rho_out - rho value with the prominent line
%
% History:
% Ralphilou Tatoy - 4/25/22 - Created function
% Ralphilou Tatoy - 4/25/22 - Completed function implementation
% Ralphilou Tatoy - 4/25/22 - Completed commenting

%Size of input image
[sizeX,sizeY] = size(i_edge);
%Get rho maximum by doing two edge point of imageor diagonal of size of

%Finding D (Diagonal)
rho_max = floor(sqrt(sizeX^2+sizeY^2));

%Initialize variable
theta_max = 90;
theta_out = 0;
rho_out = 0;
counter = 0;

%i_edge = edge(i_edge,"canny");
%Initialize accumulator
accumulator = zeros(2*rho_max+1,180);
for row = 1:sizeX
    for col = 1:sizeY
        %Voting Process
        if i_edge(row,col) > 0
            i = row ;
            j = col ;
            % Must account for all theta for each point
            for theta = -89:theta_max
                %Hough calculation using polar coordinates
```

```
                rho = round((i*cosd(theta))+(j*sind(theta)));
                %index regulation
                rho_index = rho+rho_max;
                theta_index = theta + theta_max;
                %Counter
                accumulator(rho_index,theta_index) = accumulator(rho_index, ↙
theta_index)+1;

                % Prominent or which index has to most counted
                if(accumulator(rho_index,theta_index)>counter)
                        counter = accumulator(rho_index,theta_index);
                        rho_out = rho;
                        theta_out = theta;
                end
            end
        end
    end
end



end
```