CSE 107

Lab 5: Hough Transform

Darren Correa

05/09/2022

Abstract

Hough Transform is line detection technique that transforms edge points in an image's (x, y) plane, into curves in the normal space. (ρ, θ). These curves in the normal space help let us know if a straight line is present in the original image. The first part of the Hough Transformation process is identifying the edge points in the original image. With those edge points, we can compute the angle theta (θ), which is the orientation of rho (ρ) with respect to the x – axis, and rho which is the distance from the origin to the line at the edge point. This is possible by using the slope intercept line equation, $y = ax + b$, and the normal line equation, $\rho = xcos(\theta) + ysin(\theta)$. With the information from these two equitation's, we can accumulate votes to determine if a straight line exists at the edge point.

Technical Discussion

The goal of the Hough Transform function is to accurately compute the distance from the origin (ρ), the orientation of the line (θ), and create the matrix image of the accumulator votes which shows the sinusoidal curves to determine if a straight line is present. The function takes in a black and white image of a single line. This image only contains the pixel values 0 or 255. The possible angles of orientation only lie from -90 degrees to 90 degrees. Since MATLAB loops do not iterate from negative values to positive values, subtracting 89 degrees from 180 will yield us our desired degree value for theta. To find the edge points in the image is simple. While iterating through the edge point image, check if the pixel value has the intensity value 255. If it does, we know we have found an edge point at $x = i$ and $y = j$. Using these values, we can iterate through possible values of theta to determine what angle reflects a straight line in the original image. The built-in find function returns the linear indices of a non-zero elements. I used the find function to determine the indices of the point where our ρ value lies. At the index location (ρ, θ) we can vote that here lies a straight line. As we iterate through different edge points and angles, we generate more votes to determine if we have found a straight line. These votes can be visualized as the sinusoidal curve images shown in the results section. The final

output of the function is our distance from the origin our line is at, the orientation of the line, and the accumulator cell matrix.

Discussion of Results

I am more than happy with the results from my function. Each test with the different scripts produces great results. Shown below are the images for the three random line tests. The theta values are the same as the true theta values and the rho values are within an increment of one from the true rho value. After testing my function with the real image script, I knew the function was implemented correctly cause the results were the same as the results from the lab instructions.

The most difficult part of the implementation was gathering the indices for the computed rho values. When we use the normal line function, our rho value could be negative or a non-integer value. This was an issue when collecting votes for the accumulator cells. That's why collecting the index values with the built-in find function made that process much easier.

Results

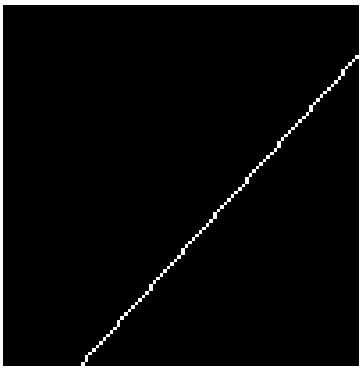

Image 1: Random Line 1

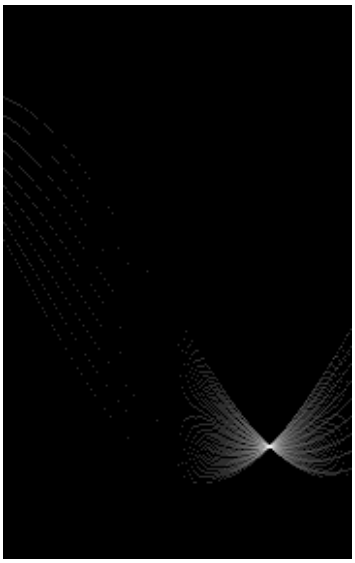

Image 2: Random Line Accumulator Image 1

True Theta: 48	**True Rho: 84**
Estimated Theta: 48	**Estimated Rho: 85**

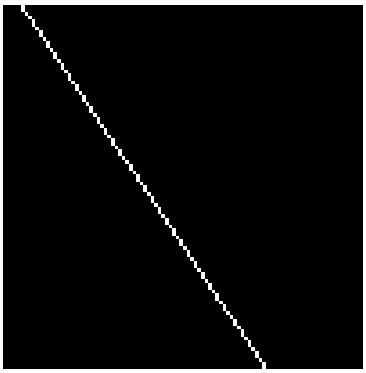

Image 3: Random Line 2

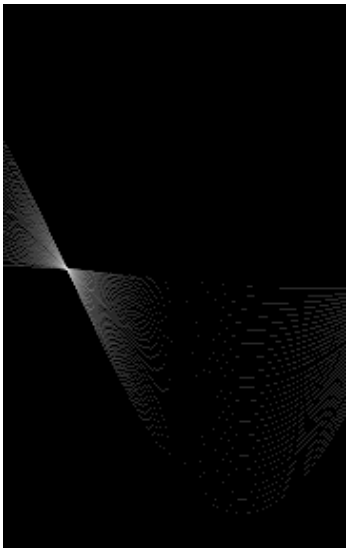

Image 4: Random Line Accumulator Image 2

True Theta: -56	**True Rho: -3**
Estimated Theta: -56	**Estimated Rho: -3**

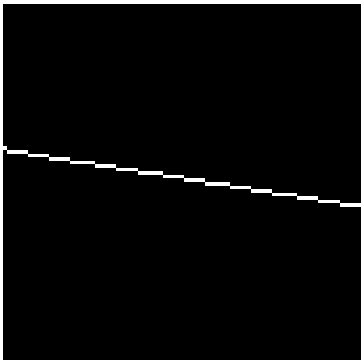

Image 5: Random Line 3

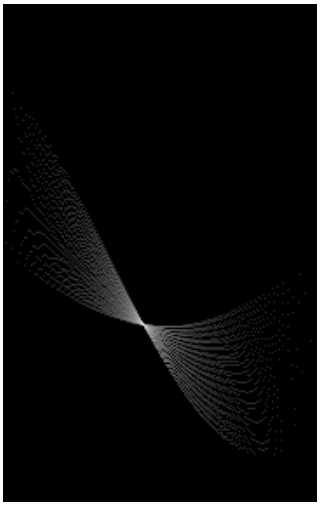

Image 6: Random Line Accumulator Image 3

True Theta: -9	**True Rho: 41**
Estimated Theta = -9	**Estimated Rho: 42**

Code

Hough_transform.m

```
function [theta_out, rho_out, accumulator] = hough_transform(i_edge)
% hough_transform     Takes in a black and white edge line image and
%                     returns the distance from origin of the line,
%                     its angle of orientation, and an accumulator matrix
%                     representing the detection of the line at an
%                     orientation.
%
% Syntax:
%   [theta_out, rho_out, accumulator] = hough_transform(i_edge)
%
% Input:
%   i_edge = Edge image consisting only of pixel intensities 0 and 255.
%
% Output:
%   theta_out = Estimated angle orientation of the detected line.
%
%   rho_out = Estimated distance from origin to the detected line.
%
%   accumulator = Accumulator matrix containing the votes at each index of
%                 rho and theta that determines if edge point is part of the
%                 line.
%
% History:
%           D. Correa   Created     5/7/2022
%           D. Correa   Completed   5/9/2022


[size_x, size_y] = size(i_edge);

%Compute diagonal size of image i_edge
size_d = ceil(sqrt(size_x^2 + size_y^2));

accumulator = zeros(floor(2*size_d), 180);

%Since we cant perform -89:90 in a loop we will set our inital degree now
%account for this issue in our loop declaration. (init_degree + k - 1)
init_degree = -89;

%Rho gives non integer values so we have to use some index to keep track of
%votes in accumulator
for i = 1:floor((2*size_d+1));
    index(i) = round(-size_d + i -1);
end

for i = 1:size_x
    for j = 1:size_y
```

```
        %Check for edge point
        if(i_edge(i,j) == 255)

            for k = 1:180

                %Compute rho using line normal form equation
                rho = i * cosd(init_degree + k -1) + j * sind(init_degree + k
-1);

                ind = find(index == round(rho));

                %Accumulate votes
                accumulator(ind,k) = accumulator(ind,k) + 1;
            end
        end
    end
end

%Accumulator with most votes
[m,indx] = max(accumulator(:));

%Return index values
[rho_out, theta_out] = ind2sub(size(accumulator),indx);

%Adjust for range of rows and theta
rho_out = rho_out - size_d;
theta_out = theta_out - 90;

end
```