Hough Transform

Sigi Lopez

11/13/18

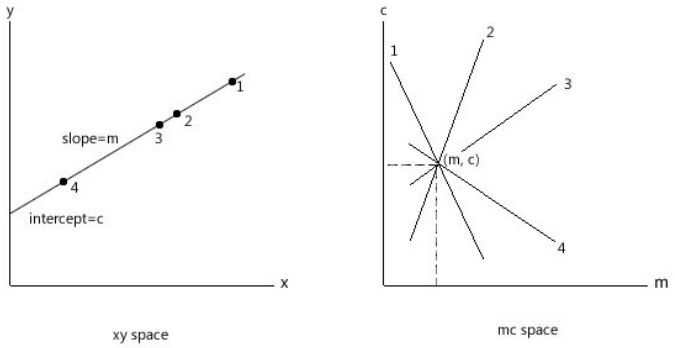
For this lab we implemented an algorithm called Hough Transform to detect the longest line in an image. This algorithm not only works with lines but also many other geometric objects such as circles. The implementation for this algorithm involved using an edge detection algorithm such as Canny or Sobel edge detection to find edges. The features of using this algorithm can be very helpful when you need to find specific objects in an image. For this lab we used a real-world example of detecting the longest road in a specific image.

Technical Discussion

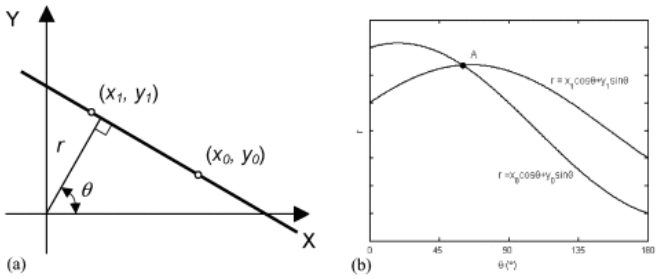
To find the Hough transform the first step was to find out a way to convert from y intercept points to a form that will find all. After that we needed to find all of the slopes for each single vertex since this problem has infinitely many solutions. So, to fix this problem we needed to generalize this information to find all crossings between these many edge points. We implemented this by converting these y intercept points to the normal form which directly gave use a relationship between the point space and the slope space. This correlation instead of showing the many cross intersections by, lines it gave us a sinusoid functions which horizontally it was the angle and vertically the rho. The space was easy to determine since the more prominent is a sinusoid line the more we know that the sinusoid is a possible intersection slope.

Equations

Slope Form to Normal Space



Normal Space Represented in Sinusoid Waves



Discussion of Results

The overall hardest part of implementing the Hough transform was to correctly convert the mathematical calculations on converting points to finding all possible sinusoids. For example, my indexing at first was incorrect therefore it was generating a false estimation for rho and theta. After correcting my math and calculations this problem was fixed. I did notice that while simulating the random lines the estimator had a harder time correlating with the intended value. Even though this was mostly a very tiny difference it goes on to tell us that rho

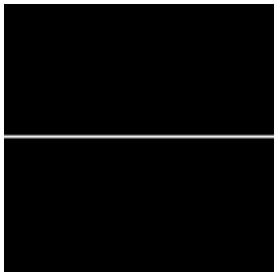
and theta might be different depending on how rounding and implying calculations takes a big part on finding the correct results.

Results

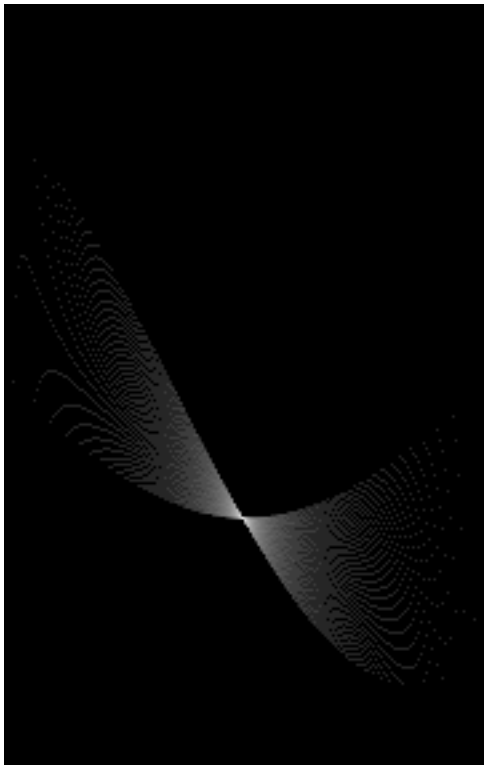
test_horizontal_line.m

True theta = 0, true rho = 50

Estimated theta = 0, estimated rho = 50

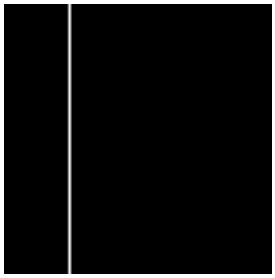


horizontal_line.tif

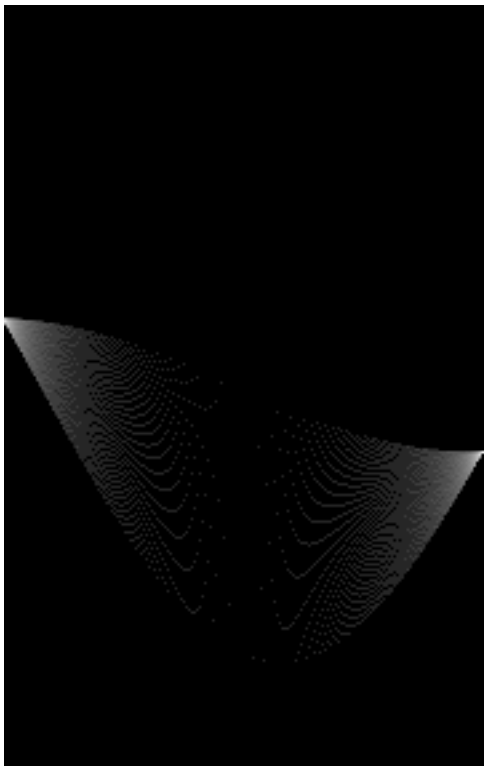


horizontal_line_accumulator.tif

test_vertical_line
True theta = 90, true rho = 25
Estimated theta = 90, estimated rho = 25

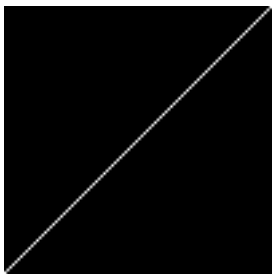


vertical_line.tif

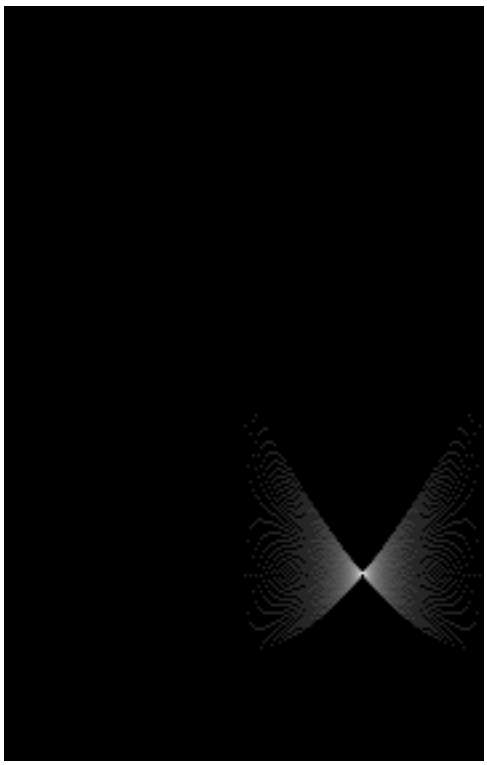


vertical_line_accumulator.tif

test_pos_diagonal_line.m
True theta = 45, true rho = 71
Estimated theta = 45, estimated rho = 71



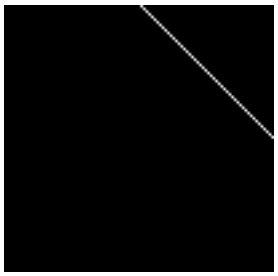
pos_diagonal_line.tif



pos_diagonal_line_accumulator.tif

test_neg_diagonal_line.m

True theta = -45, true rho = -35
Estimated theta = -45, estimated rho = -36



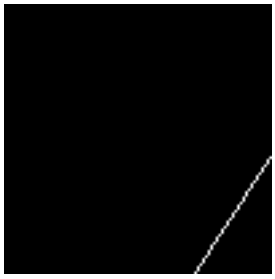
neg_diagonal_line.tif

neg_diagonal_line_accumulator.tif

test_random_line1.m

True theta = 57, true rho = 115

Estimated theta = 58, estimated rho = 115



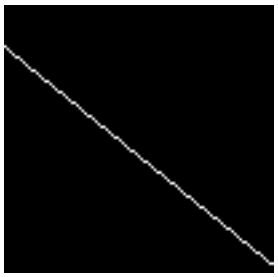
Random_line1.tiff

random_line1_accumulator1.tiff

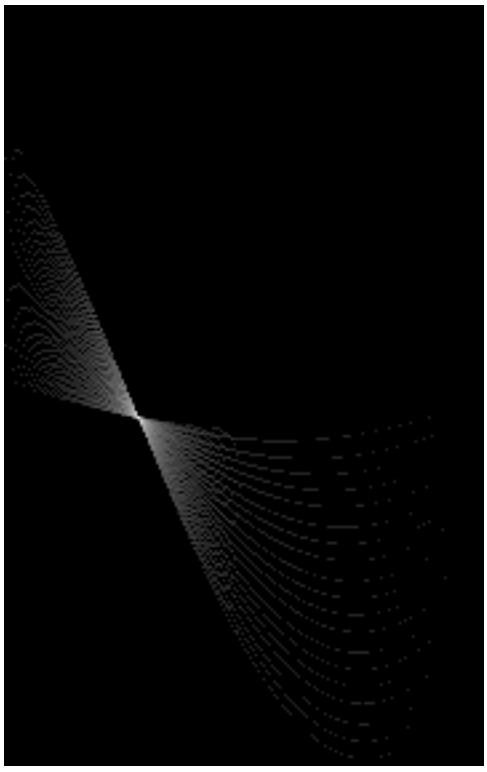
test_random_line2.m

True theta = -39, true rho = 13

Estimated theta = -39, estimated rho = 12



Random_line2.tiff



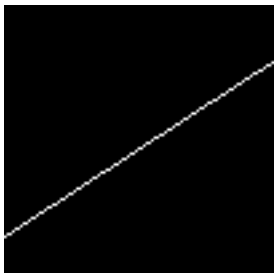
random_line3_accumulator.tiff

test_random_line3.m

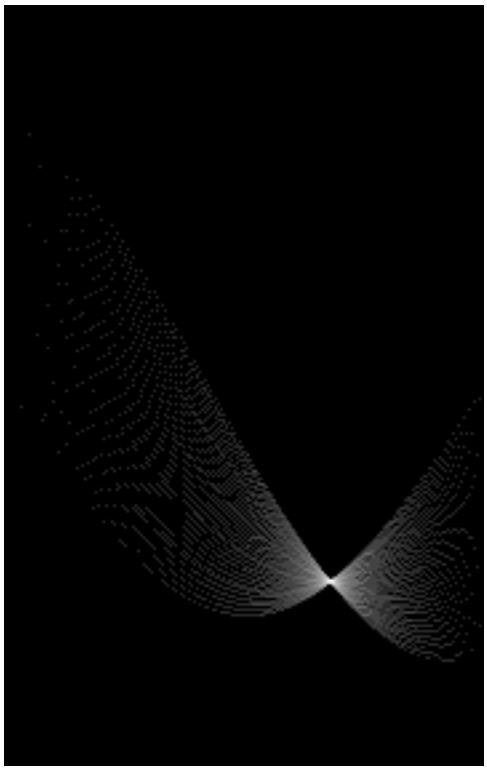
True theta = 33, true rho = 73

Estimated theta = 33, estimated rho = 73

Random_line3.tiff



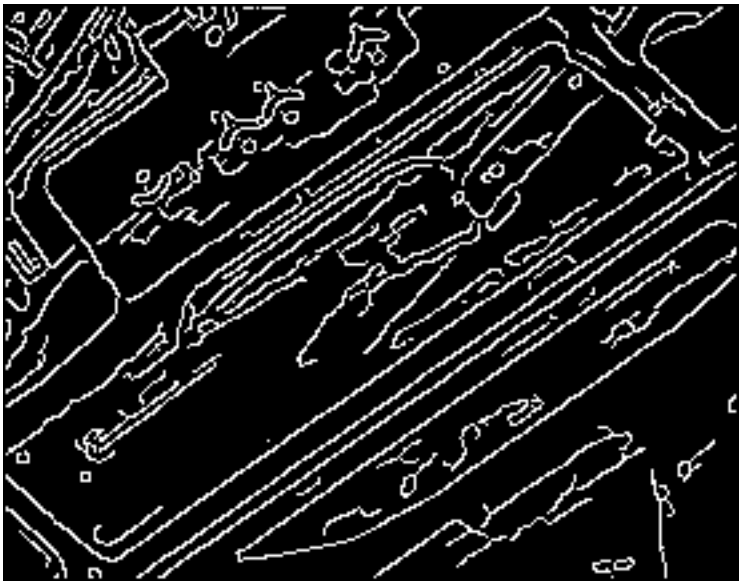
random_line3_accumulator.tiff



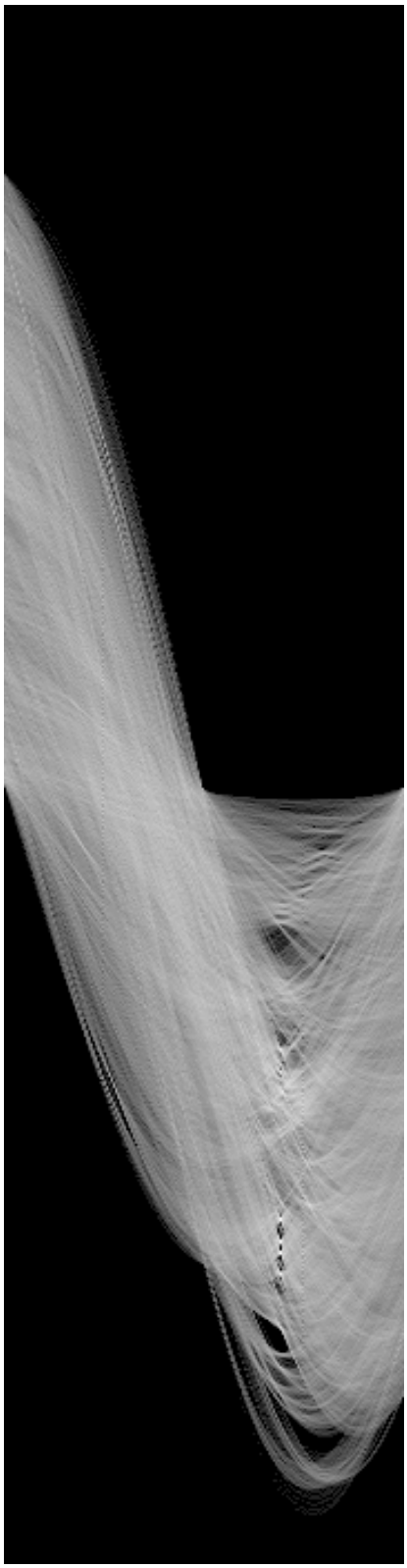
test_real_image.m

Estimated theta = 35, estimated rho = 220

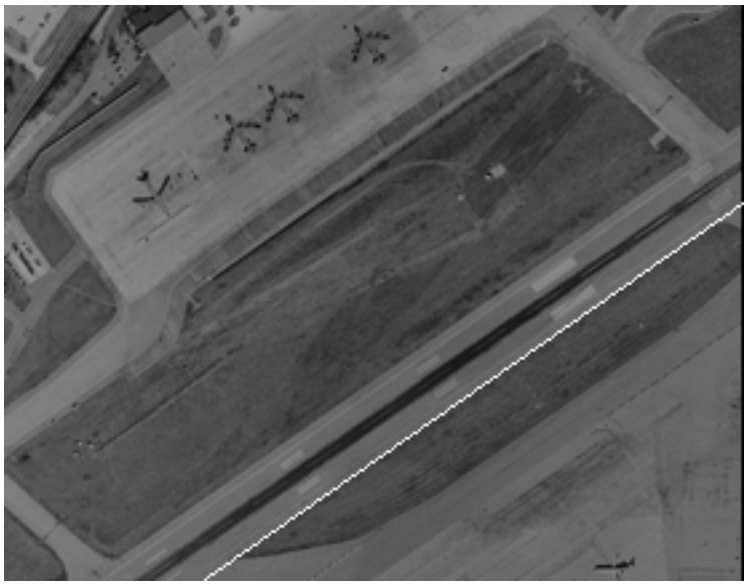
runway_image.tiff



runway_image_edge.tif



runway_image_accumulator.tif



runway_image_with_line.tif

Code

```
function [theta_out, rho_out, accumulator] = hough_transform(i_edge)
%hough_transform
%   Determines whether edge points lie on specific boundaries of objects.
%   This function is specifically only for lines
%
%INPUT: Edge Image (Sobel, Canny)
%
%Output: theta = normal angle most repeated
%        rho = nornal ditance most repeated
%        accumulator = matrix that portrays all possible line intersections
%

%gets size of image
imageSize = size(i_edge);
imageRow = imageSize(1);
imageColumn = imageSize(2);

points = ones(imageRow+imageColumn, 2);
%points are negative if they do not exist
points = points*-1;

%find points that are edges
points_count = 1;
for i = 1:imageRow
    for j = 1:imageColumn
        if(i_edge(i,j) == 255)
            points(points_count,1) = i;
            points(points_count,2) = j;
            points_count = points_count + 1;
        end
    end
end

%get diagonal distance of image
x = imageRow;
y = imageColumn;
D = round(sqrt(x^2+y^2));

%create empty accumulator matrix
accumulator_matrix = zeros(2*D+1,180);

    %find all possible sinusoid curves
    for i = 1 : points_count-1
        for theta = -89:90
             rho = points(i,1) * cosd(theta) + points(i,2) * sind(theta);
             accumulator_matrix(round(rho)+D+1,theta+90) =
             accumulator_matrix(round(rho)+D+1,theta+90) + 1;
        end
    end

    %find which sinusoid intersects the most
    temp = 0;
    for j = 1 : 2*D+1
        for k = 1:180
```

```
            if(accumulator_matrix(j,k) > temp)

                temp = accumulator_matrix(j,k);
                theta_out = k - 90;
                rho_out = j - D - 1;

            end
        end
    end

    accumulator = accumulator_matrix;
end
```