**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**

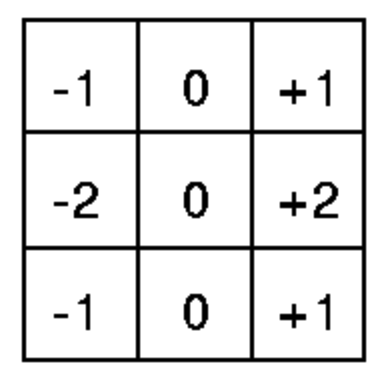
The first step for creating the edge detector was to find out how filtering worked. For our problem we needed a filter that told us were in the image pixel values changed drastically. This is the reason we picked the Sobel filters. The 0 row ignores the values that are in between while giving more preference to the places where 1’s or 2’s. Therefore, in a simple example if the value doesn’t change frequently the values would cancel out giving you a filter value of 0. In the equation below it essentially shows how the index mapping should be implemented for the filter. In this case our filter is marked as “w” and our image is “f” in the function below. The both iterations “s” and “t” are indexes that check for all eight adjacent blocks from the current pixel you are trying to apply the filter to. Another thing to point out is that in this lab we dealt with edge problems by implementing zero padding which essentially means that you add extra rows of zeros on the sides to avoid negative indexes. In theory zero padding should not affect the image that much since the values should cancel out.

**Equations**

Spatial Filtering

****

Sobel Filters

Horizontal Vertical

**Discussion of Results**

The results for applying edge detection using the Sobel filters turned out to be mostly successful with the only problem being the noise detected. Since Sobel filters check for changes in pixel values sometimes noise created by objects or light could come up as an edge. Applying a threshold sort of fixes this problem by reducing the amount of values but it also gets rid of details on edges. I noticed that the higher the threshold the thinner the edges became therefore eliminating noise to a certain extend. The problem can be further improved by implementing a gaussian filter that smoothens the image thus gets rid of unnecessary edges. This is something that another type of filter does which is called the Canny edge detection. I have displayed both results of the Sobel filter and Canny edge detection. Overall Canny edge detection surpasses the quality of detection due to its complexity and ability to improve the image before applying the filter.

**Results**

****

**Original Greyscale Image**

****

**Sobel Edge Detector with Threshold of 200**

****

**Canny Edge Detector (MATLAB Function)**

**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

A