Implementing Hough Transforms for detecting lines

Mustafa Fitrat (mfitrat@masonlive.gmu.edu) and Kenny Peng (kpeng5@gmu.edu)

I. Introduction & Related Work

Note, though we had intended to implement Generalized Hough Transforms, we were unsuccessful in doing so due to time constraints. Thus, we were only able to implement the conventional form for detecting lines.

The Hough Transformation (HT) is an interestingly simple yet effective voting algorithm method of detecting lines within images, and it has garnered the attention of many researchers as it has many practical uses such as lane detection on roads [1]. It is very effective in detecting simple shapes and has been expanded upon with the Generalized Hough Transformation to be usable with shapes that cannot be defined as equations. However, issues exist preventing more widespread adoption which come mainly in the form of accuracy and performance concerns as this method becomes computationally complex as parameters are added increasing the transforms' parameter space dimensions, and this has spurred initiatives to improve performance through various means [2, 3, 4, 5]. Our goal was to implement a HT usable for line detection and to detail the process of doing so as well as any difficulties encountered.

Duda and Hart [5] present one of the first implementations of HT and introduce the novel rho-theta parametrization algorithm widely used today as the standard baseline for implementing and discussing HT. I addition, they also discuss the potential generalization of HT [5]. This method for line detection is the one we implement in this paper.

Casasent and Krishnapuram [3] note the previous implementations' inadequacies with regard to computational costs due to high parameter dimensionality, and they attempted to resolve this by applying inverse HT for curved object detection.

Li, Pao, and Jayakumar [4] observe that in the HT defined by Duda and Hart, there are inaccuracies that can come from using their because they do not account for contiguity of pixels leading to being unable to find length, distinguishing segments, and false positives. They propose a method of HT that takes into account pixel contiguity to solve this using a parallelization with systolic array and specialized hardware to improve accuracy and efficiency of HT. In their conclusion, they mention that the method of curved object detection with inverse HT made by Casasent and Krishnapuram [3] may have merit in reducing dimensionality of HT accumulators.

Shapiro [2] noted the limitations of the standard Hough Transform, the algorithm presented by Duda and Hart [5] and which we implemented in our paper. Shapiro concludes that this voting system fails to ”resolve a number of existing accuracy deficiencies” and ”considers a non-voting alternative called Hough-Green Transform”, even stating that the Hough-Green Transform performed better than the standard Hough Transform.

Hajjouji, Mars, Asrih, and Mourabit [5] explain the benefits of computing the angle and radius over the slope and y-intercept for Hough Transform. The algorithm they use is similar to the algorithm described in [2] by Shapiro.

II. Theory

Briefly, the implementation is taking images, detecting straight lines, then producing an output image with the detected lines marked. To achieve this, the RGB channels of the image are converted into a single gray scale channel, then edge detection is performed onto the grayed image. Then, certain Image space coordinates , treated as (x_i, y_i), are extracted from the edges, or the regions where the pixel value is above a certain intensity.

With the edge coordinates extracted, they are then converted into sinusoids in the Hough space, a parameter space with coordinates (p, θ). One important property to note is that when a point from Image space is converted to Hough space, it produces a sinusoid. If we convert from Hough space to Image space, a line is produced. So, with equation (1), we create sinusoids within the Hough space. For the Hough space, the range of theta is fixed such that $0 \leq \theta \leq \pi$ and that p is fixed by $-R \leq p \leq R$ [5]. In [5], "R is the size of the retina", which we set as the size of our input images. To reiterate, for every (x_i, y_i) coordinate and θ within $[0, \pi]$, we are producing a p that should be within the range of $[-R, R]$ then use these (p, θ) coordinates to produce sinusoids inside the Hough space.

$$p = x * cos(\theta) + y * sin(\theta) \tag{1}$$

These sinusoids will intersect at certain points producing these peaks or local maximas, which represent a probable line in the Image space. Therefore, once we convert all Image space coordinates into Hough space, these peaks will be extracted based off the threshold given, which is variable depending on the image inputted. Once all the important peaks, in the form of (p, θ), are collected, they are then converted back into lines within the Image space by using equations (2) and (3) to produce the line equation (4). Note, the x variable is a range from 0 to the width of the image, so just like equation (1), we're taking x from a range of 0 to the image width and producing y values to form a line in the image space. These equations were acquired from [1].

$$m = cot(\theta) \tag{2}$$

$$b = \frac{p}{sin(\theta)} \tag{3}$$

$$y = b - mx \tag{4}$$

The important aspect of the implementation to measure is whether the implementation correctly detects the number of lines that exist within the image. By observing the number of "correct" peaks produced in the Hough space $p_{expected}$ and comparing them with the number of peaks collected $p_{extracted}$, we produce equation (5). Note, $p_{incorrect}$, are any lines or peaks incorrectly extracted, mainly due to the limitations of our implementation, was determined by observing the images. We expected the lines to be related to or be on the road itself, else it was incorrect, see figure 5.

$$\frac{p_{extracted} - p_{incorrect}}{p_{expected}} * 100 \tag{5}$$

III. Data & Implementation

We downloaded images of roads from the Internet as input data. These images were then processed and used to produce output images that would mark the lines detected, using MatPlotLib.

For the implementation, we wrote code that would extract the edges from an image, quantize a Hough space, convert (x, y) Image space coordinates into (p, θ) Hough space coordinates, simulate the voting system by creating an accumulator 2D array used by a Hough transform, and convert Hough space coordinates to Image space coordinates.

A function that would extract local maximas from the accumulator was taken from Dr. Gregory Stein, who taught Computer Vision for the Spring semester. Pillow and Numpy were used to load in images, convert RGB channels into gray scale, and have images represented as an 2 dimensional array. Scipy was used to perform edge detection on images. MatPlotLib was used to mark the lines detected and save our output images.

IV. Results

Based off the table and figures below, it appears as though our implementation was nearly successful. By taking the average of the values in the Correctness column, the overall correctness of our implementation for these eleven images is 76.82%, with most of the prominent lines having been found. However, our implementation appears to collect too many possible edges and this in turn makes the accumulator become very noisy and difficult to threshold correct peaks from incorrect peaks. Because of this problem, if we use a threshold value too high, we will not be able to detect every lines we expect, and if it is too low, we mark extra lines that are not important. Therefore, if there were more time to improve on this implementation, we would attempt to reduce the number of votes in order to produce more accurate results.

TABLE I
The correctness of our implementation of Houghs transforms

Image	Lines Expected	Lines Extracted	Lines Incorrect	Correctnes
box	8	8	0	100%
lines1	53	48	0	91%
lines2	78	78	4	95%
lines3	52	52	0	100%
road1	8	6	0	75%
road2	6	3	0	50%
road3	6	6	0	100%
road4	6	4	0	67%
road5	3	3	1	67%
road6	3	6	4	67%
road7	3	1	0	33%

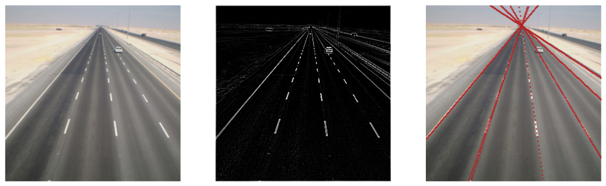

Fig. 1. Using Houghs Transforms to detect lines on road1.png. In order from left to right, original image, edge detection performed on image, and final result. Threshold value was 40.

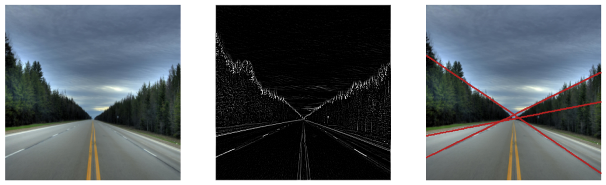

Fig. 2. Using Houghs Transforms to detect lines on road2.png. In order from left to right, original image, edge detection performed on image, and final result. Threshold value was 37.

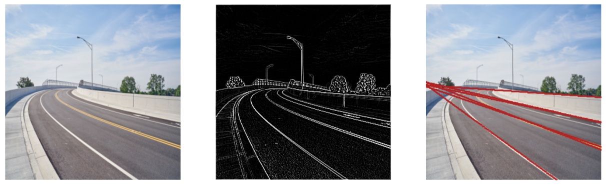

Fig. 3. Using Houghs Transforms to detect lines on road3.png. In order from left to right, original image, edge detection performed on image, and final result. Threshold value was 80.

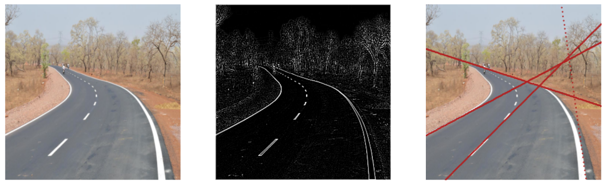

Fig. 4. Using Houghs Transforms to detect lines on road4.png. In order from left to right, original image, edge detection performed on image, and final result. Threshold value was 50.

References

[1] I. El Hajjouji, S. Mars, Z. Asrih, and A. El Mourabit, "A novel FPGA implementation of Hough Transform for straight lane detection," *Engineering Science and Technology, an International Journal*, vol. 23, no. 2, pp. 274–280, Apr. 2020, doi: 10.1016/j.jestch.2019.05.008.s

[2] V. Shapiro, "Accuracy of the straight line Hough transform: the non-voting approach," *Comput. Vis. Image Underst.*, vol. 103, no. 1, pp. 1–21, Jul. 2006, doi: 10.1016/j.cviu.2006.02.001.

[3] D. Casasent and R. Krishnapuram, "Curved object location by Hough transformations and inversions," *Pattern Recogn.*, vol. 20, no. 2, pp. 181–188, Mar. 1987, doi: 10.1016/0031-3203(87)90052-5.

[4] H. F. Li, D. Pao, and R. Jayakumar, "Improvements and systolic implementation of the hough transformation for straight line detection," *Pattern*

Recognition, vol. 22, no. 6, pp. 697–706, Jan. 1989, doi: 10.1016/0031-3203(89)90006-X.

[5] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, doi: 10.1145/361237.361242.