

CSE 397/497 Intro to Mobile Robotics

Laboratory Assignment: Hough Transformation
Report Due Date: 27 Oct 05

A. Objective: In this assignment, we will implement a Hough transformation of an edge image to extract the strongest lines from the image.

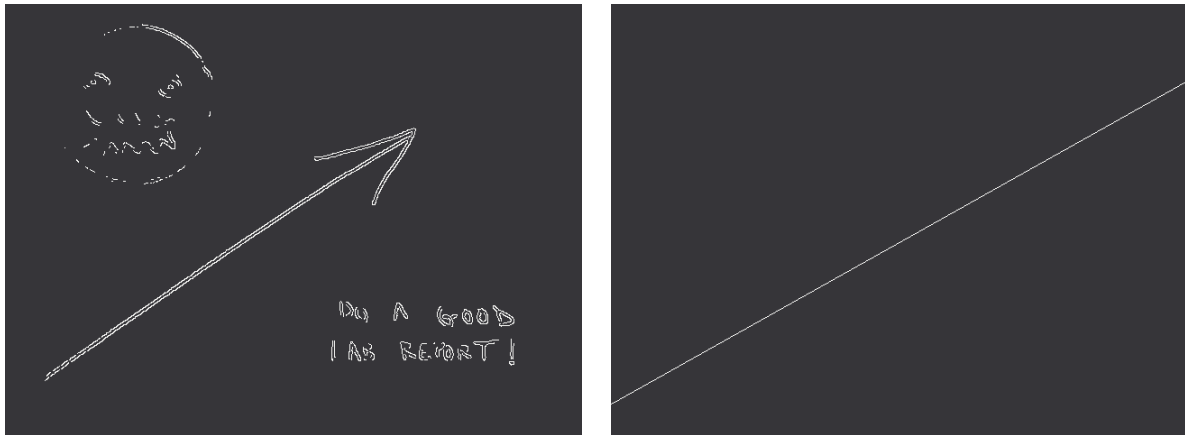


Figure 1: Canny (left) and corresponding Hough line image.

B. The Implementation: Implementation will be consistent with the procedures discussed in class, and will encompass three phases (see the end for algorithm details):

- 1) Generating an edge image
- 2) Generating the Hough Transform of the image
- 3) Plotting the line(s) with the largest number of votes.

To accomplish this, you are to implement the following functions:

```
void myHough(Image* pIm, int pAcc[][360/dTheta], int height, int width);
```

```
void plotLines(Image* pIm, int pAcc[][360/dTheta], int threshold);
```

`myHough` will take the source EDGE image and an accumulator array as input, as well as the height and width of the source image.

The accumulator array will contain the number of votes for each (ρ, θ) candidate. The dimensions of the accumulator array should be $[\text{maxDistance}/dp][360/d\theta]$ where $(dp, d\theta)$ correspond to the resolution of the discretization resolution.

TIP: Be certain to zero out the accumulator array at the beginning of each iteration.

`plotLines` will take a blank image, the accumulator, and a threshold as arguments. Any lines receiving more votes than the threshold value will be drawn to the blank image buffer.

C. Possibly Useful functions:

```
cvCanny(pMyImage->pIplHeader, pMyImage2->pIplHeader, t1, t2);
```

Is the OpenCV function that performs Canny edge detection. You can use this as input to Hough if you like (you can also try a Sobel version instead. It will be significantly faster).

```
pMyImage3->Clear();
```

Sets the image to black (zero).

```
pMyImage3->DrawLine(0, 480-b, 639, 480-tan(theta)*639+b, cvScalar(255));
```

Allows you to draw lines on the blank image buffer.

```
pMyImage2->Save("canny.tiff");
```

Allows you to save images.

Your end result will be to display the original edge image as well as the corresponding line image as shown in Figure 1.


READ THE REPORT REQUIREMENTS BELOW FOR ADDITIONAL LAB REQUIREMENTS THAT WILL NEED TO BE ADDRESSED.

D. Your Report: You and your partner are to submit a lab report with the results of your implementation. It must include

- Your source code
- Appropriate images as described below
- At a minimum, responses to the following questions
 1. Adjust your settings for $(dp, d\theta)$. How will changes to either of these affect algorithm complexity?
 2. Besides adjusting $(dp, d\theta)$, what else affects Hough run-time performance? Can we control this?
 3. Save at least two pairs of images from the SAME SCENE but using different discretization values. What is the impact on the line image?


4. Why do we ignore negative radius values when we are doing the Hough transform? How could we exploit this to improve algorithm performance?

FINAL NOTE: SHOW ME YOUR END RESULT BEFORE LEAVING THE LAB TO VERIFY THAT YOUR IMPLEMENTATION IS CORRECT!



Hough Transform Algorithm

1. Take as input an edge image $E(i,j)$
2. Define a resolution $d\theta$ and $d\rho$ for θ and ρ , respectively
3. Construct an accumulator array $A(m,n)=0$ where $m=1/d\rho*\rho_{max}$ and $n=2\pi/d\theta$

$$\rho_{max} \in [0, \sqrt{im_h^2 + im_w^2}]$$


Hough Transform Algorithm

4. For each pixel $E(i,j) \neq 255$ do
 - For $\theta = d\theta:d\theta:2*\pi$
 1. $\rho = i*\cos(\theta) + j*\sin(\theta)$
 2. if $\rho < 0$, continue
 3. Round ρ to the nearest $d\rho$ value
 4. $A(\rho,\theta)++$
5. Threshold $A(\rho,\theta)$ to find all relevant lines