Photogrammetric Computer Vision Lab4 Report

Professor Sohn

By SeyedMostafa Ahmadi

November
2021

Introduction
This lab is about Hough Transform. We have to model straight lines fitting to our input image. First, we should get do corner detection on the input image. Using the result of corner detection, we can find the pixels that are candidates to fit our model. Prior to the line segmentation step, the process is pretty simple.

The algorithm in one glance

1. Creating theta ranges
2. Calculation of sin and cos of thetas
3. Extraction of (x, y) pairs of the edge points
4. Finding edge pixels
5. Calculation of rho values
6. Calculation of accumulator matrix
7. Cutting off the indices of the accumulator
8. Drawing the remaining lines
9. Line segmentation

I am going to explain each step of the algorithm separately:

Creating theta ranges
In this step, we simply create a range of theta values from 0 to 180 degrees (the right side is exclusive). For the optimization part, and after some explorations, I finally knew that it is best that we have each degree. What I am saying is that theta value will range in these values: [0, 1, 2, ..., 179]

Calculation of sin and cos of thetas
The Numpy package that I am using can calculate sin and cos of multiple values in parallel. So it is best to use parallel mode when these values do not need each other and can be used separately in separate threads.

Finding edge pixels
We have had done this through our previous exercises, and now we are gonna use it as a step. The input of the Canny edge detector is the input image in grayscale.
Then we will have (x, y) pairs of edge pixels. The final output of the whole process is to check if these pixels fit into a line model or not. So, we have (x, y) pairs for edge pixels.

Calculation of rho values
As the output of the Canny edge detector, we have (x, y) pairs of edge pixels. We do a normalization step and assume that (0, 0) is in the middle of the input image. Then, for

each resulting (x, y) pair and for all theta values that we had before, we calculate rho values.
rho = cos(theta) * x + sin(theta) * y
for all (x, y) pairs and all theta values.

Calculation of accumulator matrix

Now that we have rho and theta values we can calculate the accumulator matrix. Note that rho value can range from -image_diagonal/2 to +image_diagonal/2. It is high school geometry. For the optimization part, after some exploration, I knew that it would be best if I divide this range into 180 parts. (If you divide more parts, the line will be more accurate but you should lower your threshold!)
Based on rho and theta values, I make a 2d histogram. The first axis is theta in (0, 179) and the second axis is rho in (-image_diagonal/2, +image_diagonal/2).

Cutting off the indices of the accumulator

The output of the last step is something like voting. There should be some points voting to form a line model. We define a threshold for lines which if the vote is lower than that threshold, we give up and do not assign a line to that group of points anymore. After explorations, I found out that with mentioned ranges of theta and rho it would be best if I set the threshold to 220.

Drawing the remaining lines

Afterward, we can draw these lines. But there is a problem that these lines are infinite. So, we assume these lines as intermediate results and should go on to find the correct finite lines.
Drawing lines when we have rho and theta values is so easy.
This is the equation of the line:
rho = cos(theta) * x + sin(theta) * y

Line segmentation

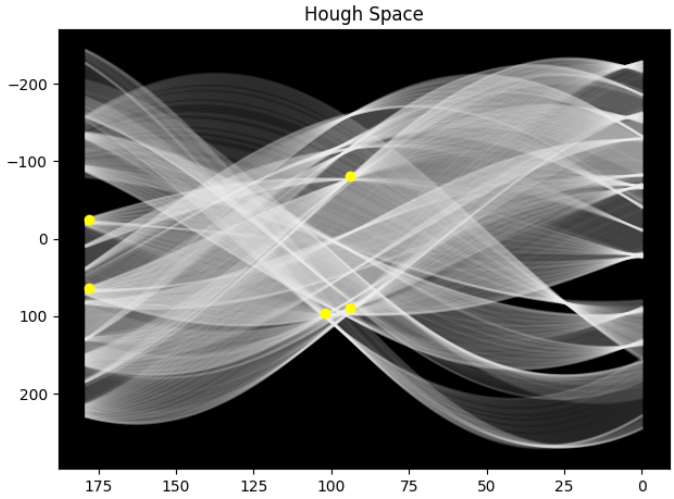
This step had a big challenge for me. Because I didn't do the "for" all over the pixels of the image as I said above, and Numpy did all the multiplications in parallel. So, what did I do? For each index of the accumulator that its value was above the threshold, I found the rho and theta values. Then, I found its (x,y) pair. Then in this set of pairs, I calculated two of them that were on the edge of the line.
Actually finding two edges was an interesting algorithmic problem. The problem is: "We have a set of points that are approximately on a line. How can we find two edges of the line with a low cost?"
The answer is:
We take a random point from the set. We calculate its distance to all the other points of the set. The one with the maximum distance is the first edge. And then we take the first edge point and calculate its distance to all the other points of the set. The one with the maximum distance is the second edge!
Then we have (x,y) pairs of both sides of a line.

The results

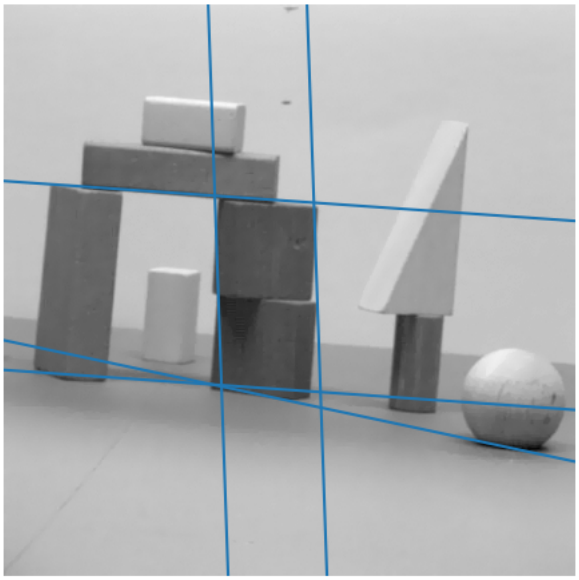
These results are achieved after optimization.
So, I write down the final values, and then show the results.

Arch

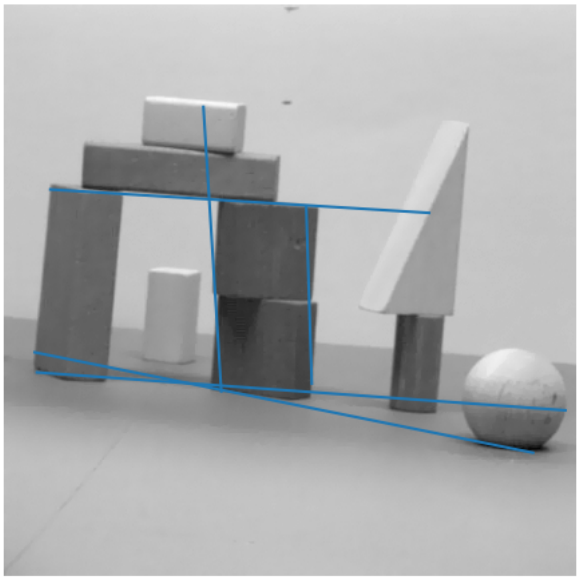
Sigma=3
Threshold=220



Detected Infinite Lines

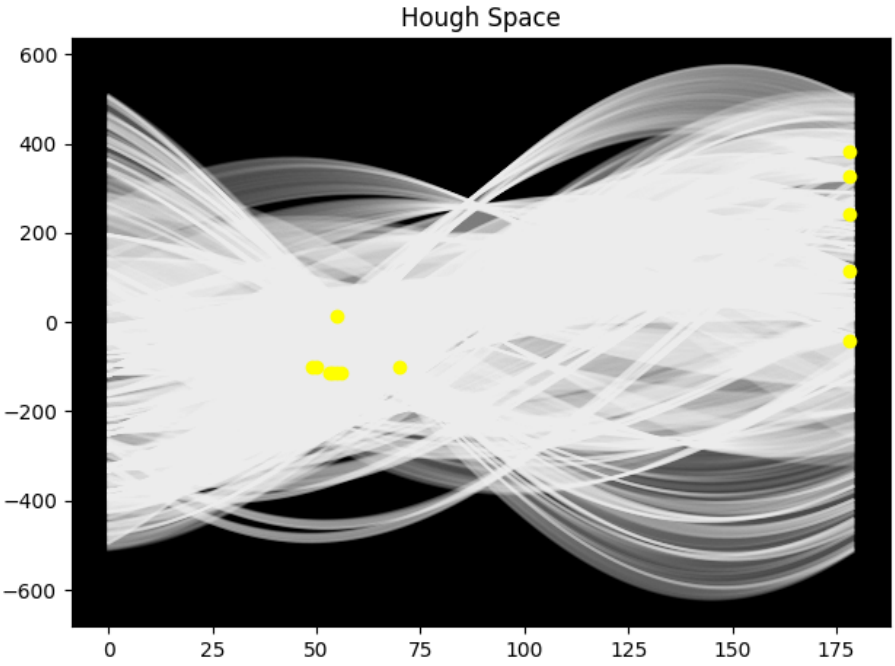


Detected Segmented Lines

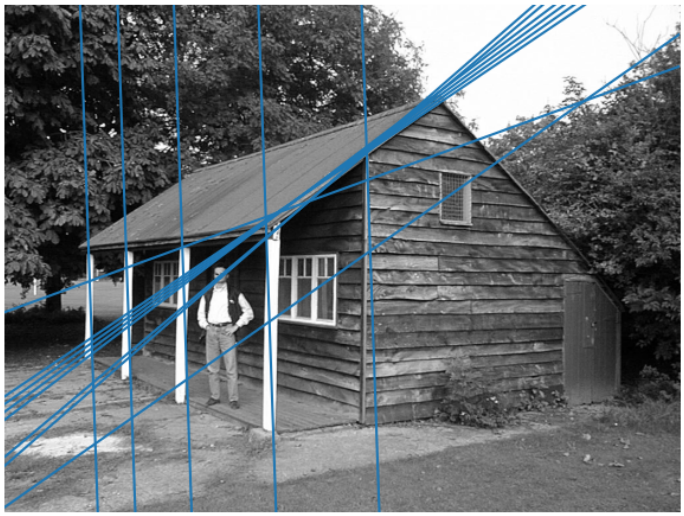


House

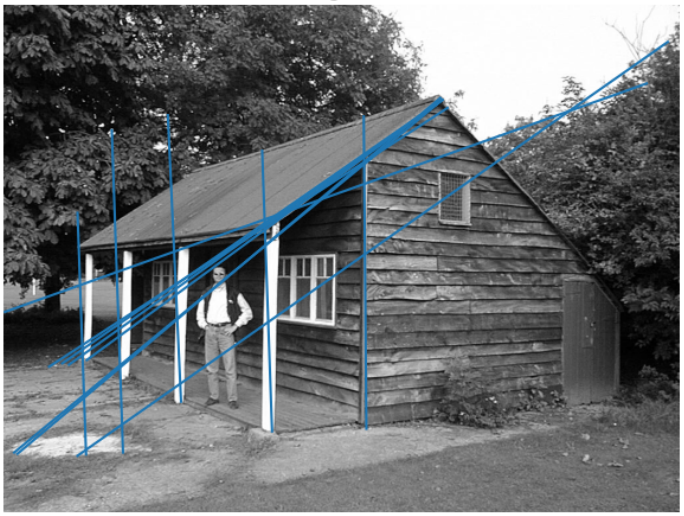
Sigma=5
Threshold=450



Detected Infinite Lines

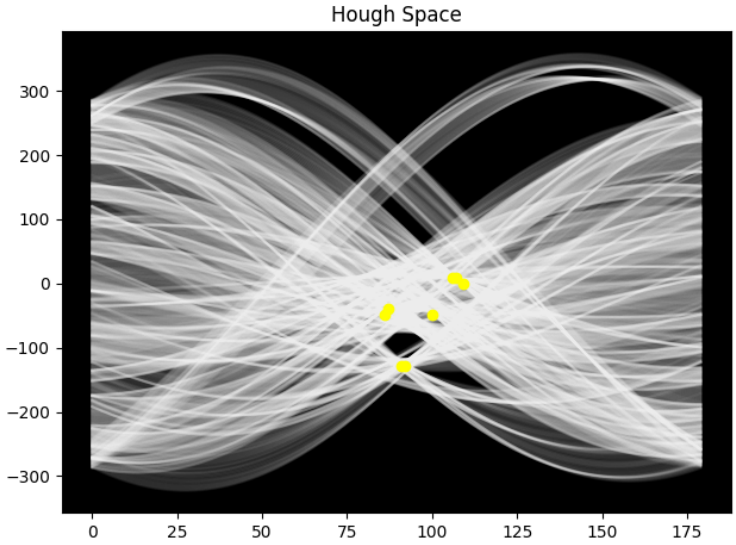


Detected Segmented Lines



Merton

Sigma=5
Threshold=300

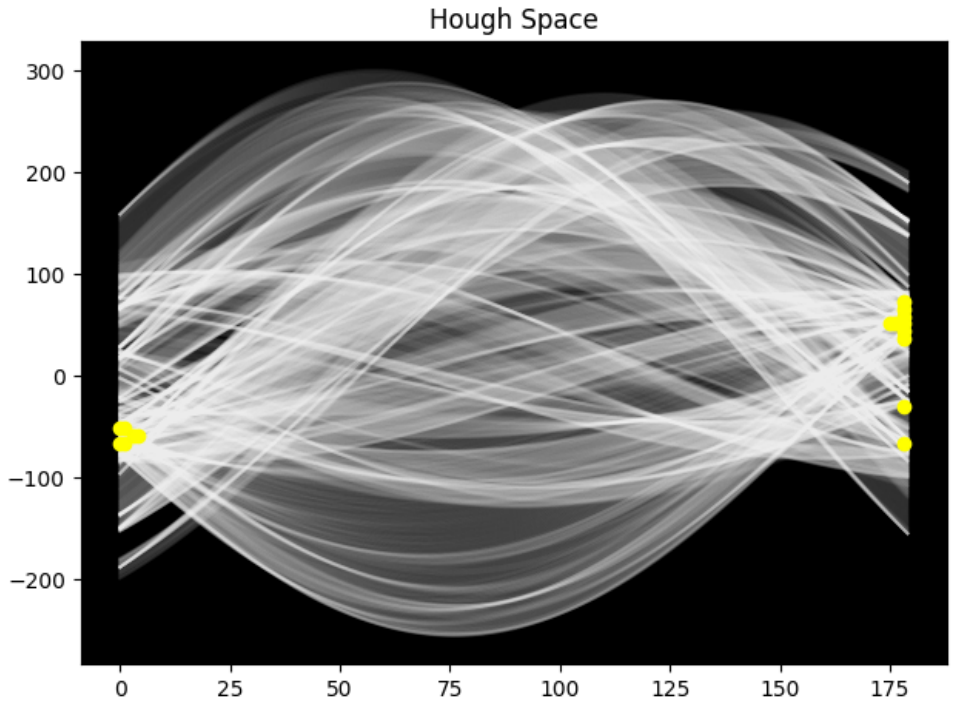


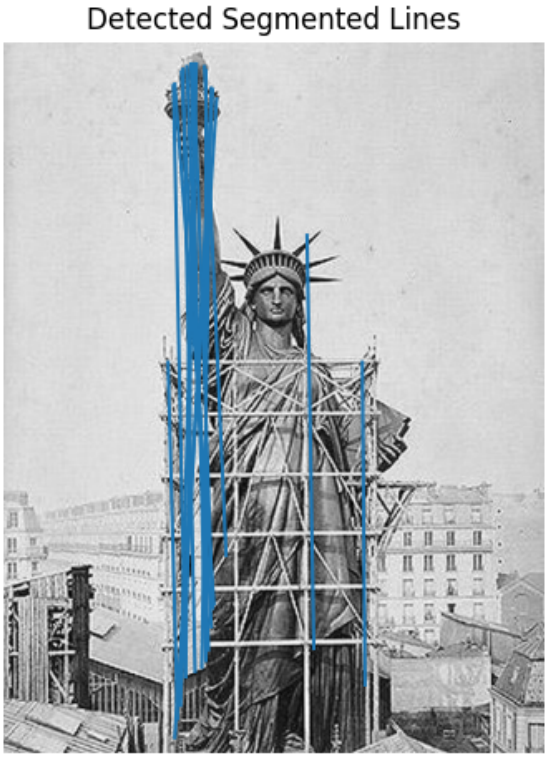
Detected Infinite Lines

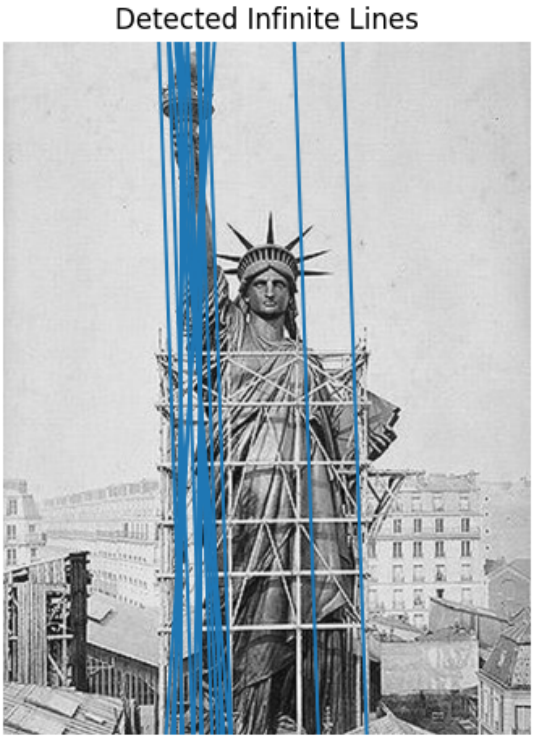
Detected Segmented Lines

Liberty

Sigma=5
Threshold=250







As you can see the results, it is not the best!
But we can trust them that work.
An important point is, as the image resolution increases, we should increase the threshold value. Because more pictures are assigned to the same edges and as a result we should increase the minimum votes for a line model.

I hope it is clear!
Thank you for your attention!