**Họ và tên:** Trần Trung Tín

**MSSV:** 19522351

**Lớp:** CS338.M21.KHCL

**CANNY EDGE & HOUGH TRANSFORM APPLICATION IN LINE DETECTION**

**\*HOUGH TRANSFORM\***

1. **Idea**

The main idea for this hough transform algorithm is:

* Base on the result of the Canny Edge Detection\_popular canny edge detection.
* On every **edge pixel**, we apply and try every equation line that goes through that point. The more lines we found the better result we get when we detect lines.
* When we’ve already scan through all edge pixel then we will filter out those lines base on a threadhold (already defined) on the statistical matrix after that we can keep lines that exist in the picture.
* When we already got those lines, our job is just to draw that them on the picture.

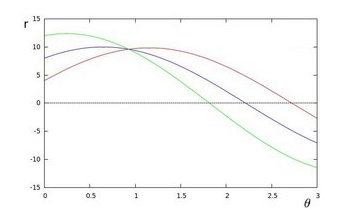
A straight line in a image space can be represented in two variables :

* Variables (m, b) in Descartes Coordinates
* Variables (r, Θ) polar coordinate

With this Hough Transform we will represented a line in polar coordinate. The formula can be written as below:

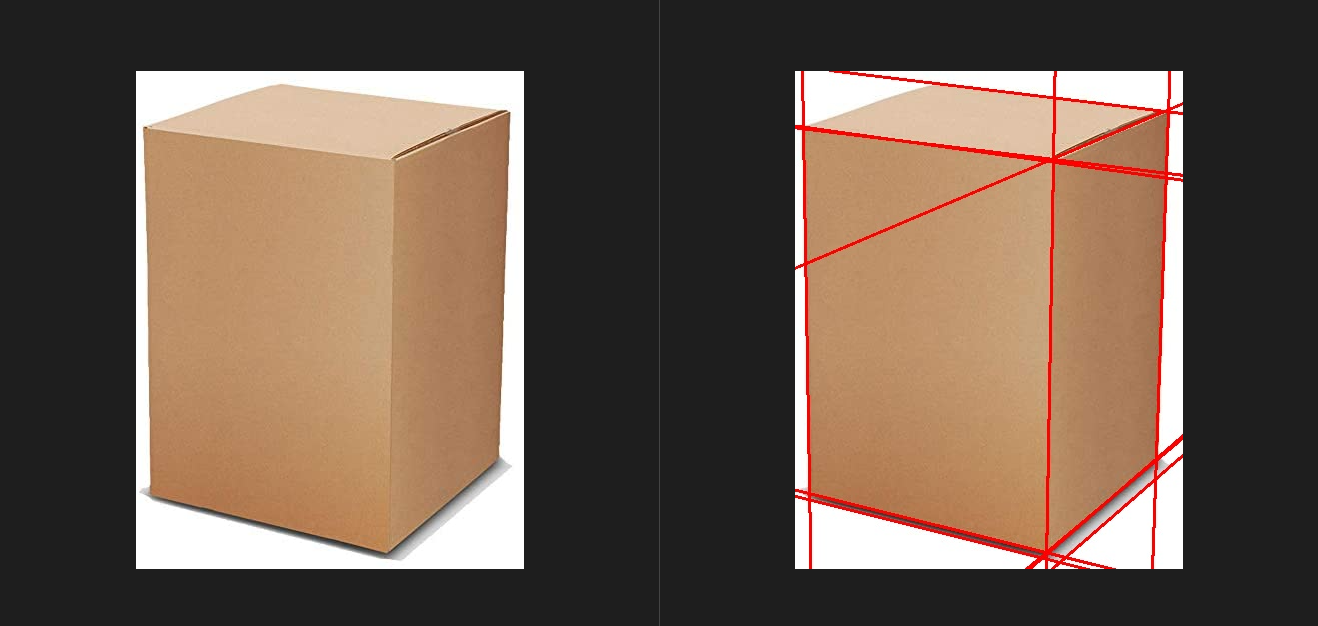


With Every (x0, y0) we can draw a straight line



A straight line can be detected by finding all the intersections between the curves. The more intersecting curves mean that the line represented by the intersections has more point\_which mean it’s clearer to be a straigth line than just some seperate points. We can set a thresdhold of the minimum number of intersections required to detect a straight line. If the number of the intersections exceeds a certain threshold, a straigh line can be determined.

**Result:**



**\*CANNY EDGE DETECTION\***

Canny Edge is a algorithm initiated by John Canny of a MIT laboratory in 1986. Canny proposed a set on constraints that a edgee detection must meet. He presented an optimal method to implement those constraints. And this method called the Canny method. Now we will learn the come to detail about the operations and implementation of this algorithm.

Some basics step related to this algorithm:

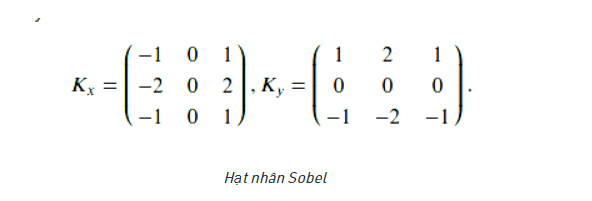
* Noise reduction with Gaussian Filter
* Calculate gradient along the horizontal and vertical axis
* No maximum cancellation of false edges
* Delay the threshold

1. **Reducing the noise using Gaussian Filter**

This step is extremely important in canny edge detection. Since edge detection is prone to noise in the image, the first step is to remove the noise in the image using a Gaussian filter.

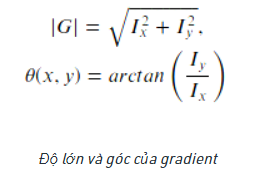
1. **Calculate the gradient along the horizontal and vertical axes**

When the image is smoothed, the Gx and Gy derivatives are calculated along the x and y axes. It can be done using convolution of Sobel kernels.



Use 2 kernels: 1 kernel for horizontal shift and one kernel to change vertically.

After applying this Kernel, we can use the magnitude of the gradient and the angle to continue processing this step. The magnitude and angle can be calculated as follows:

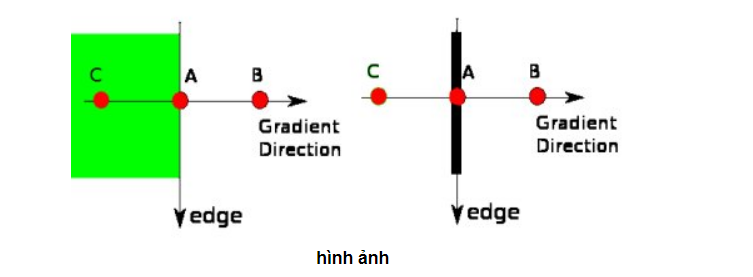


The magnitude of the gradient is used to measure how much the image correlation changes.

The angle of the gradient is used to determine the direction in which the magnitude of change is directed.

1. **Non-Maximum Suppresion**

After getting the magnitude and direction of the gradient, a full scan of the image is performed to remove any unwanted pixels that may not form an edge. For this, at each pixel is checked if it is a local maximum in its neighborhood in the gradient direction. If it is a neighboring maximum, we will keep that pixel. And if the pixel there is not a neighboring maximum, we will set its gradient magnitude to zero. We only compare the central pixel with 2 neighboring pixels in the **gradient direction.**



Point A lies on the edge (in the vertical direction). Gradient direction normal with respect to edge. Points B and C in the gradient direction. So point A is tested against points B and C to see if it forms a local maxima. If it is a local maximum it is considered for the next period, otherwise it is discarded (reduced to 0).

In short, the result we get is a binary image with “thin edges”.

1. **Delay Threshold**

Even after applying Non-Maximum Suppresion, we may need to remove areas of the image that are not technically edges, but still respond as edges after calculating gradient magnitude and applying Non-Maximum Supresion.

To ignore these regions of the image, we need to define two thresholds: Tupper and Trower.

Any gradient value G > Tupper is definitely an edge.

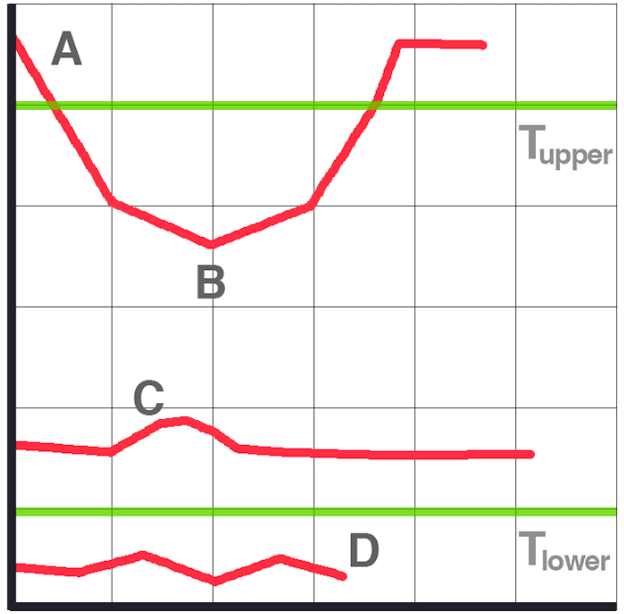
Any gradient value G < Trower is not an edge, so discard immediately.

And any gradient value in the range Tlower < G < Tupper needs to be considered that:

1. If the specific gradient value is connected to a strong edge (ie G > Tupper), then mark the pixel as an edge.

2. If the gradient pixel is not connected to a strong edge, discard it.

The delay threshold is actually better visually explained:



• At the top of the graph, we can see that **A** is a solid edge, because **A** > Tupper.

• **B** is also an edge, although B < Tupper since it is connected with a strong edge.

• C is not an adjacent edge from C < Tupper and must not be connected to a strong edge.

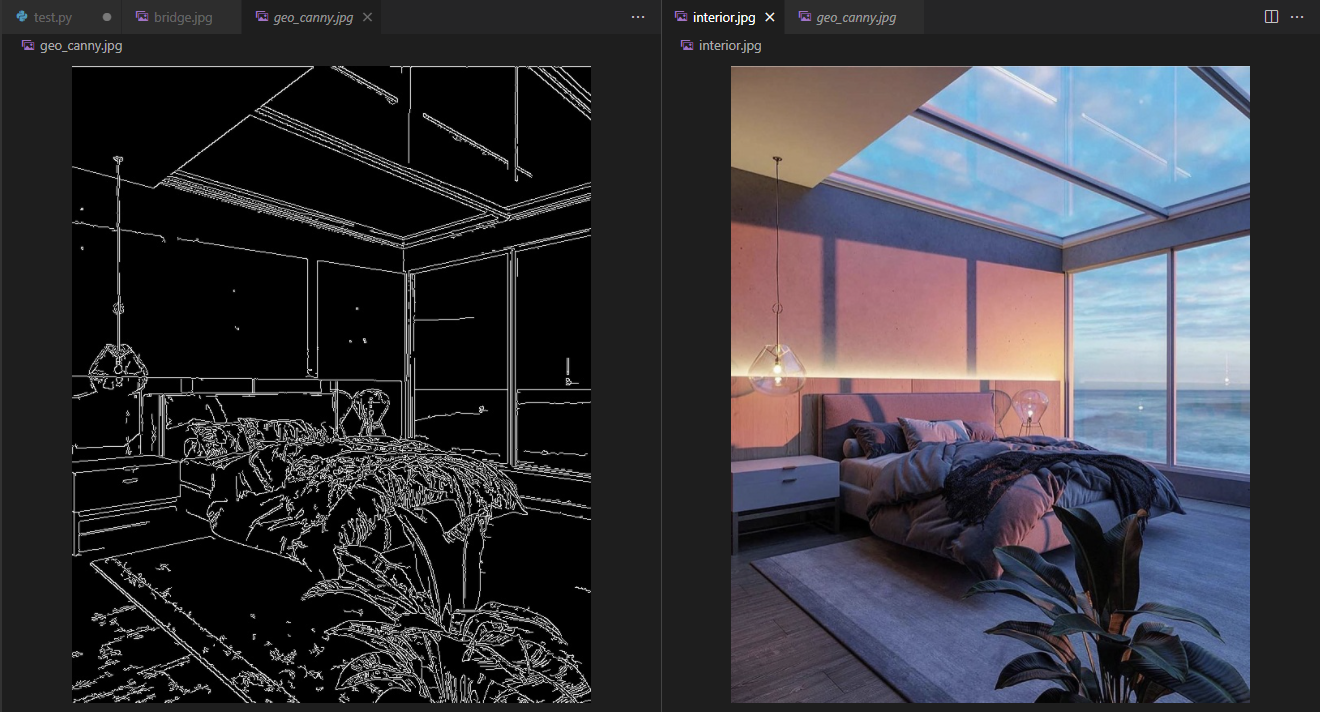
• Finally, **D** is not an edge because D < Trower and is automatically discarded.

Setting these threshold ranges is not always a trivial process.

If the threshold range is too wide, we will get many false edges instead of simply finding the structure and boundary of an object in the image.

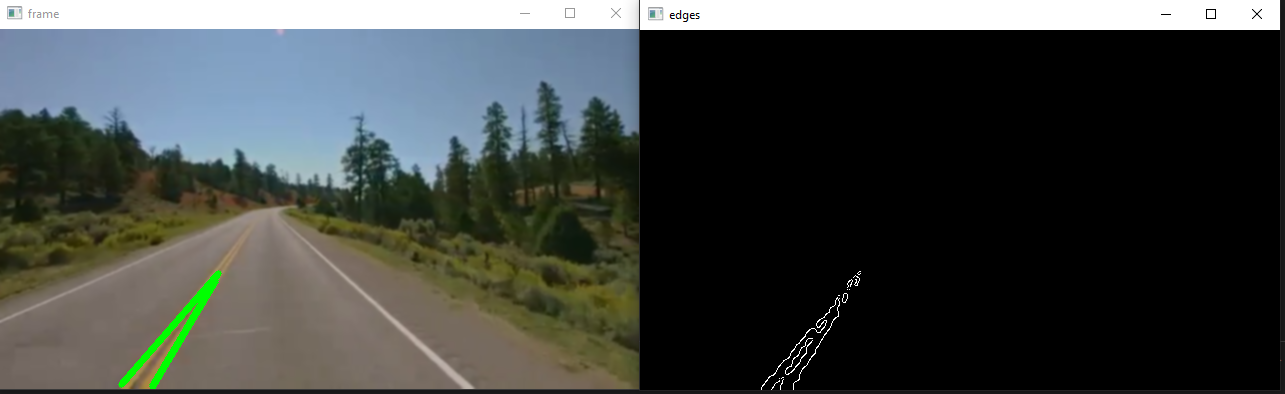
Similarly, if the threshold range is too tight, we won't find many edges and may run the risk of missing the object's structure/contour entirely.

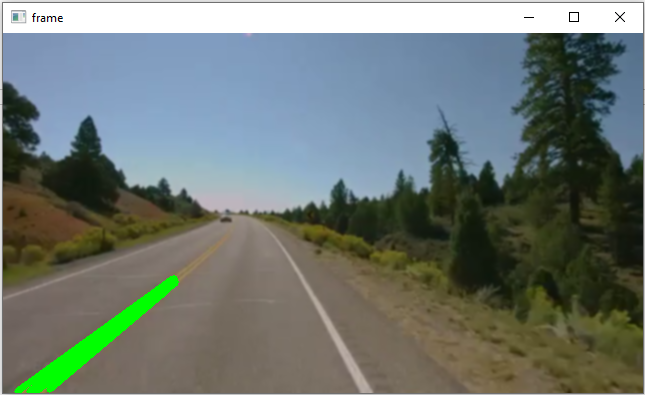
**Result :**

****

**\*LINE DETECTION\***

Combine these two algorithm into a video we got this result:







* The detections is pretty good but there’s some limit maybe because of the light conditions in this video.

Link drive: <https://drive.google.com/drive/folders/1IMtEJXERvJjIR5H6TEdyXLgv7avhsFSz?usp=sharing>