Lab 3 - Hough Transform

Computational Vision & Imaging

Student ID: XXXXXXX
Random Student, School of Computer Science,
University of Birmingham, UK, B15 2TT

***Abstract*—This document is a formal solution for the Lab 3 Formative Assignment based on Hough Transform. It analyses the algorithm of detecting lines in the image using Hough Transform. It checks the impact of changing the number of desired lines to detects and experiments with different edge filters required to produce an input for the Hough Transform.**

***Index Terms*—Lab 3, formative assignment, computer vision, imaging, Hough Transform, filters, kernels, Roberts, Prewitt, Sobel, Laplacian, Gaussian, Canny, MATLAB.**

I. Introduction

BUILDING on top of the previous topics *Edge Detection* and *Noise Removal*, we were introduced to another important task in imaging - *line detection*. After having the edges of the image, they can be used to determine and match with the straight lines. The method this paper analyses is **Hough Transform**. Detecting line segments is an important feature in machine vision.

II. Setup

We continue the setup of the first 2 labs because the methods for edge detection, such as *Sobel* or *Laplacian* operators will be reused here. One additional operator, not covered before but that is going to be used in the experiments is **Prewitt**. It works the same way as the *Sobel* operator and its horizontal and vertical filters can be seen in Equation 1.

$$G_X = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}; \; G_Y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{1}$$

This time we're given a different image "cluttera2.jpg" which can be seen in Figure 1.

Fig. 1: Cluttera

This is a solution for the Formative Assignment by Hamid Dehghani
The report was submitted to Canvas on Mar 04 and reviewed by a TA

III. Methodology

Before section IV, it is necessary to explain certain algorithms met while completing the tasks - **Canny Edge Detection** and **Hough Transform**.

A. Canny Edge Detection

It is a relatively straightforward algorithm considering Canny's observation that the first derivative of the *Gaussian* provides an operator that optimizes signal-to-noise ratio and localisation [1]. The algorithm works like this:

1) Compute the image gradients $\nabla_x f = f * \nabla_x G$ and $\nabla_y f = f * \nabla_y G$ where G is the *Gaussian* function of which the kernels can be found by:
   - $\nabla_x G(x, y) = -\frac{x}{\sigma^2} G(x, y)$
   - $\nabla_y G(x, y) = -\frac{y}{\sigma^2} G(x, y)$
2) Compute image gradient *magnitude* and *direction*
3) Apply **non-maxima** suppression
4) Apply **hysteresis** thresholding

Non-maxima suppression - checking if gradient magnitude at a pixel location along the gradient direction (*edge normal*) is local maximum and setting to ' if it is not

Hysteresis thresholding - a method that uses 2 threshold values, one for certain edges and one for certain non-edges (usually $t_h = 2t_l$). Any value that falls in between is considered as an edge if neighboring pixels are a strong edge.

B. Hough Transform

We know that a point in *Cartesian space* can represented in *Polar space* as a point from the origin at some angle as can be seen in Equation 2.

$$\rho = x \cos\theta + y \sin\theta \tag{2}$$

When talking about **Hough Transform** we are concerned with parameters (explained below) and by using this *Polar* representation, we impose limits for the parameters - $\theta \in [-90, 90)$ and $\rho \in [0, \text{image diagonal}]$ - and so they can't go to infinity. We want to find the parameters that describe a line and for this we introduce **Hough Space**.

Hough Space - a plane defined by ρ and θ which takes points (x, y) in image space and represents them as sinusoids in the

new space. Each point in such space (ρ, θ) is parameters for a line in the image space.

To perform **Hough Transform**, we take the mostly intersected points of sinusoids in the **Hough Space** and map them back to the lines in the image space. Such lines pass through those points which as sinusoids were part of the intersection point. The process can be seen in algorithm 1.

Algorithm 1 Hough Transform

Initialize vectors $\boldsymbol{\rho}$ and $\boldsymbol{\theta}$ for all possible lines
Initialize matrix A of zeros indexed by ρ_i and θ_j
for each point (x, y) **do**
  for each angle θ **do**
    $\rho \leftarrow x \cos\theta + y \sin\theta$
    $A[\rho, \theta] \leftarrow A[\rho, \theta] + 1$
  end for
end for
where $A >$ Threshold return a line

We also still need to suppress non-local maxima. Note that there are generalised versions for ellipses, circles etc. But we are concerned with a line equation, i.e., MATLAB works with **Standard Hough Transform**.

IV. Tasks

A. Task 1

Task 1 asks to walk through the script file and explain how each function works, from edge detection, *Hough Transform* to line detection. The answer is presented in the following paragraphs.

Edge Detection

Firstly, the algorithm takes an image, converts it to greyscale and applies *Canny Edge Detection* technique to identify borders. It applies *Gaussian filter* derivative to compute image gradients, computes their magnitude, applies *Non-maxima Suppression* and *Hysteresis Thresholding*. The output is a binary image matrix with pixels with values of 1 indicating edges. The output edge map of the given image can be seen in Figure 2.

Fig. 2: Cluttera edge map

The explanation corresponds to MATLAB documentation: *"The Canny method finds edges by looking for local maxima of the gradient of I. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds, to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be "fooled" by noise, and more likely to detect true weak edges."*

Note that the threshold values are determined automatically and the default sigma value is $\sqrt{2}$

Hough Transform

Secondly, the algorithm applies *Standard Hough Transform*, where it takes a binary image matrix and transforms image space to *Hough* space (maps points to sinusoids). It digitalizes each line to a parameter matrix where each entry is incremented if a discrete line passes through that matrix cell. The output is the matrix itself with rows representing possible magnitudes and columns representing all possible angles. It also returns an array of considered angles and an array of considered magnitudes.

The explanation corresponds to MATLAB documentation: *"hough(BW) computes the Standard Hough Transform (SHT) of the binary image BW. The hough function is designed to detect lines. The function uses the parametric representation of a line: rho = x*cos(theta) + y*sin(theta)."*

Thirdly, the algorithm finds *Hough Peaks* - it takes the *Hough Transform* matrix and finds up to specified number of peaks (in the provided case, up to 10), whithin a specific minimum threshold (in the provided case, 0.3 times the maximum value of the *Hough* matrix). The output is a coordinate matrix with rows representing coordinates of the peaks in the parameter matrix. Visualized *Hough Transform* with identified peaks of the provided image can be seen in 3.

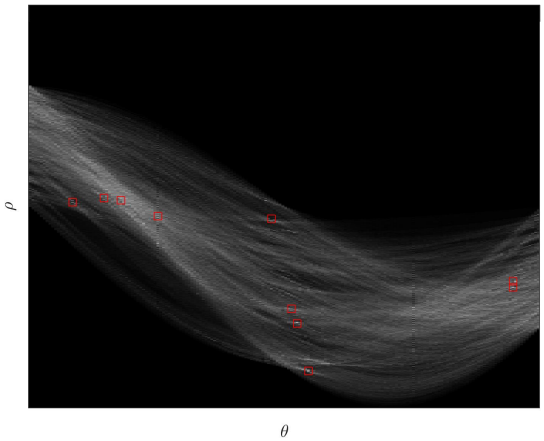


Fig. 3: Cluttera Hough Transform

The explanation corresponds to MATLAB documentation: *"houghpeaks(H,numpeaks) locates peaks in the Hough transform matrix, H, generated by the hough function. numpeaks specifies the maximum number of peaks to identify."*

Line Detection

Finally, the line segments are extracted via 'houghlines' method. It takes in the *Hough* matrix, theta and rho values as well as calculated peaks and computes the line segments in the image. A threshold to merge multiple lines to one (in the provided case, 5) and a minimum line length (in the provided case, 7) can also be specified. The start and the finish of the line is determined by checking the original edge values to check whether there actually should be a line. Idea of edge connectivity is used through named parameter 'FillGap' which is a distance between 2 line segments of the same bin. *"When the distance between the line segments is less than the value specified, the houghlines function merges the line segments into a single line segment"*. The output with line segments of the provided image can be seen in Figure 4.

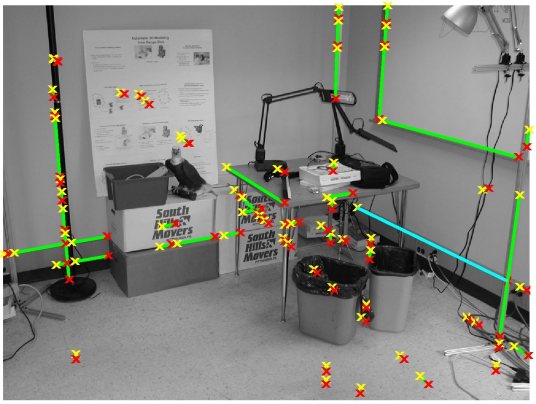


Fig. 4: Cluttera line segments

The explanation corresponds to MATLAB documentation: *"houghlines(BW,theta,rho,peaks) extracts line segments in the image BW associated with particular bins in a Hough transform. theta and rho are vectors returned by function hough. peaks is a matrix returned by the houghpeaks function that contains the row and column coordinates of the Hough transform bins to use in searching for line segments."*

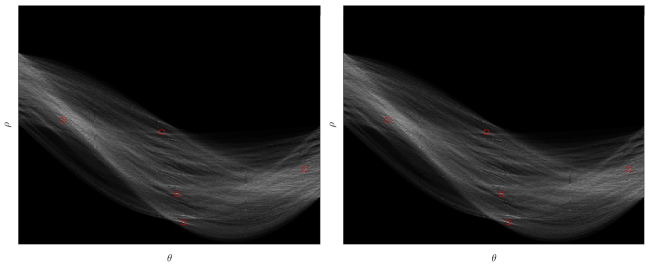
B. Task 2

Task 2 requires to experiment with one parameter and explain the results.

Quesion: what is the effect of increasing/decreasing the required number of peaks in 'houghpeaks'?

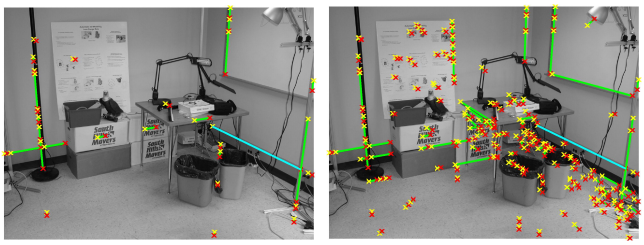
Answer: by decresing the maximum number of peaks, only the brightest ones are marked in the *Hough Map*. After extracting lines, few clear long lines can be seen (in the chosen case, 5) with few to no very short line segments. This is because at very bright peaks many sinusoids overlap, meaning many points on the same line, thus longer edge regions. By increasing the maximum number of peaks, the opposite can be observed - less bright peaks are marked in the *Hough Map*, meaning there are lines made up of not as many edge points and also such lines consist of short line segments rather than being solid long lines.

The results of the experiment on the *Hough Transform* graph can be seen in Figure 5 and on the line segmentation in Figure 6



(a) houghpeaks = 5 (b) houghpeaks = 20

Fig. 5: Hough Transform graphs with 5 and 20 identified peaks



(a) houghpeaks = 5 (b) houghpeaks = 20

Fig. 6: Images with 5 and 20 detected major lines

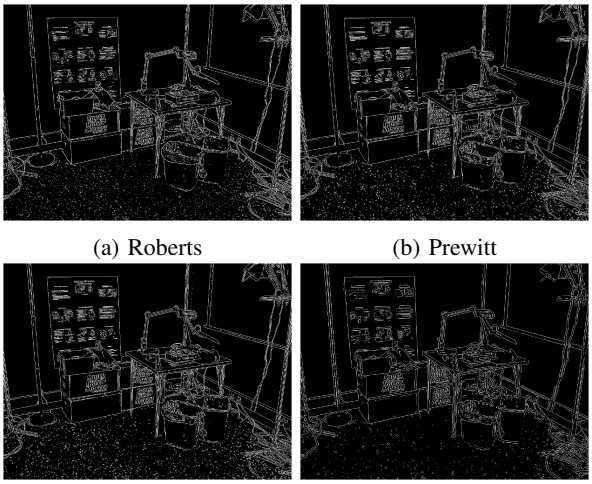
C. Task 3

Task 3 requires to experiment with different edge detection filters to see the impact on the detected line segments.

Question: replace the Canny Edge detector with other algorithms. Which one do you think performs best and why?

Answer: assuming the thresholds for each edge detection method are chosen such that they produce similar edge maps and assuming all parameters for *Hough Transform* are kept the same, it seems that *Canny* technique provides the best results due to its edge smoothness in the edge map, even though other filters seem to identify the specified number of lines (10 in the given case) at the same or very similar positions (note: not line segments but the general lines themselves).

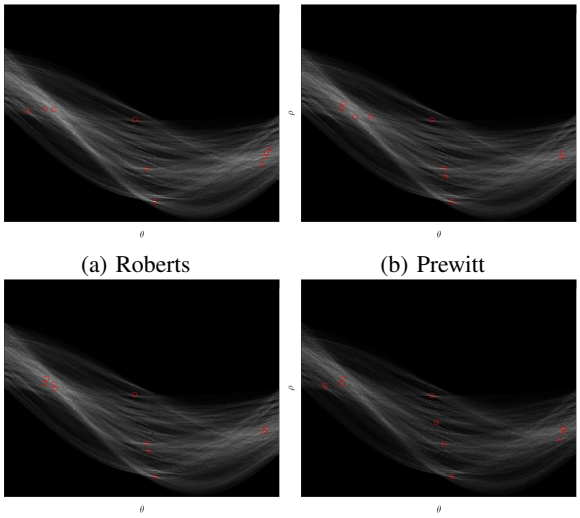
The 4 different edge maps, *Hough Transform* graphs and line detection maps acquired through applying different edge detection operators can be seen in figures 7, 8 and 9 respectively. The threshold chosen for *Roberts* operator is 0.03 and for both *Prewitt* and *Sobel* operators it is 0.025 since filters are larger and smoothen the noise better. For *Log of Gaussian* operator, *zerocrossing* is applied since thresholding is not needed for 2nd order gradient map. Note that the results of the *Canny* operator are not involved as they were shown previously.



(a) Roberts (b) Prewitt

(c) Sobel (d) Log of Gaussian

Fig. 7: Edge maps by 4 different edge operators



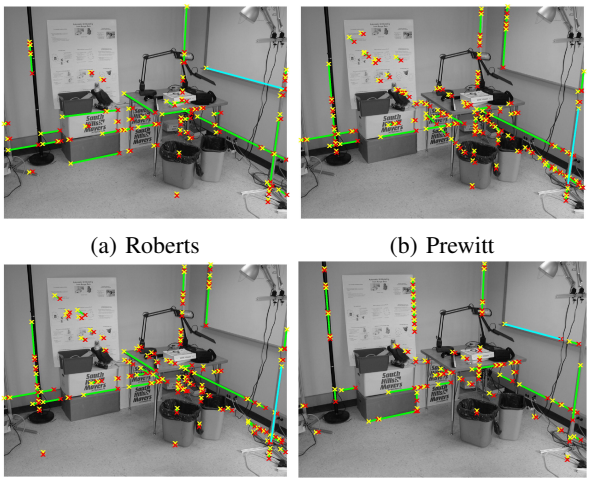
(a) Roberts (b) Prewitt

(c) Sobel (d) Log of Gaussian

Fig. 8: Hough Transform graphs based on 4 different edge operators

Roberts - due to a slightly higher threshold than for *Prewitt* and *Sobel* filters it shows less noise in the line map, i.e., it has less short line segments. Overall it captures the lines quite well, likely because it captures diagonal edges which the image possesses (boundaries are slightly tilted). It still lacks completeness for some lines, such as for the black vertical ”pipe” on the left, showing that edges should be smoother for them to correspond with the line endpoints.

Prewitt - because this filter is larger, a slightly lower threshold compared to *Roberts* filter is enough because this filter is smoother. It still, however induces a bit more noise (due to lower threshold) meaning there are more short line segments in the line map. It is better at recognizing vertical lines than *Roberts* map and generally the longest lines are longer than the longest lines when using the *Roberts* filter because of more certainty in the edges.



(a) Roberts (b) Prewitt

(c) Sobel (d) Log of Gaussian

Fig. 9: Line segment maps based on 4 different edge operators

Sobel - produces very similar results to *Prewitt*, however it is better because there is less noise due to larger weights in the filter on the center pixels meaning the edge map is more accurate. It has also less false line segments (segments which do not correspond to the actual edges) again because of less noise - the line detection method does not connect false positive lines simply because noise does not introduce them.

Laplassian of Gaussian - this method produces great results with minimal noise. This is because the Gaussian part of the method averages the noise out and the *Laplassian* part detects the edges more precisely due to the use of the 2nd order derivative. The breaking points of line segments are even more precise than when using *Canny* filter. Likely, *zero-crossing* points align better than thresholded points, thus giving more precise line segments. It still lacks some major lines, such as the side of the whiteboard, compared to when using *Canny* - it does not have such great connectivity for weak edges (*Canny* tackles that with *Hysteresis thresholding*). It could potentially be a better method than *Canny* if played well with *Hough* line detection parameters.

REFERENCES

[1] Canny, John, ”A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 6, 1986, pp. 679-698

[2] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, ”Digital Image Processing Using MATLAB”, 2nd ed., Gatesmark Publishing, 2009

[3] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, ”Digital Image Processing Using MATLAB”, Prentice Hall, 2004

[4] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, ”Digital Image Processing Using MATLAB”, Prentice Hall, 2003