Aerial Robotics Kharagpur Task 1.1

Lakshay Gupta

***Abstract*— The task is to detect edges in the image table.png so that Luna is prevented from falling off the table's edge**

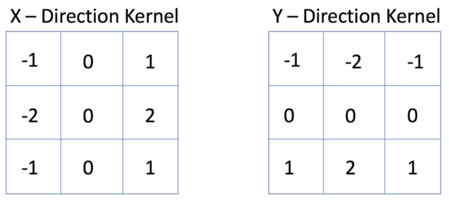
I have used a sobel edge operator from scratch to detect edges in the image. The Sobel edge detector uses a pair of 3 x 3 convolution masks, one estimating gradient in the x-direction and the other estimating gradient in y–direction. The Sobel detector is incredibly sensitive to noise in pic- tures, it effectively highlight them as edges. For the detection of edge lines I have used the hough transform. It converts shapes such as lines and circles into mathematical representations in a parameter space, making it easier to identify them even if they're broken or obscured. Some applications include object tracking and recognition, biometrics including hand detection models etc.

I. INTRODUCTION

The problem statement requires to make an algorithm that can detect edges in the image table.png and which prevents Luna from falling off the table.To perform this task we can use many edge detection tools such as the sobel, canny or laplacian based edge detectors. I have used sobel as it is relatively simpler to operate. To construct edge lines from edge.png, I have used the hough transform. I have tried various threshold levels and also while binarizing the image i have set a threshold level but still my code detects an extra edge line which is a false edge.

II. PROBLEM STATEMENT

The problem statement required 2 things to be done. The first was to detect edges in the image table.png and then detect edge lines from the subsequent edge.png. For the former task I have used the sobel operator which uses a pair of 3 x 3 kernels, one for the x-direction and one for the y directions, the kernels are as follows:



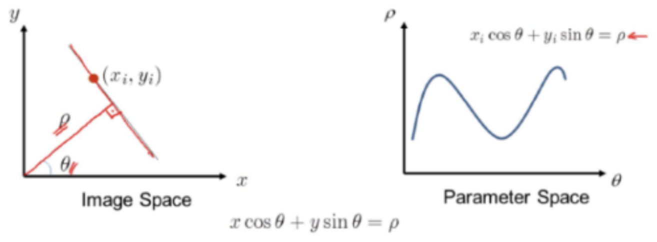
X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

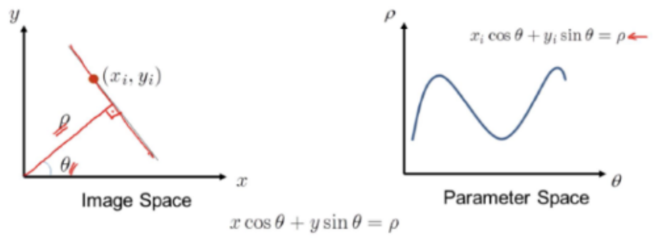
Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

When using Sobel Edge Detection, the image is processed in the X and Y directions separately first, and then combined together to form a new image which represents the sum of the X and Y edges of the image.


y
(x_i, y_i)
ρ
θ
x
Image Space
ρ
$x_i \cos\theta + y_i \sin\theta = \rho$
θ
Parameter Space
$x\cos\theta + y\sin\theta = \rho$

Next we use the Hough transform to convert well differentiated edge lines from all the detected edges. It uses a voting mechanism to identify bad examples of objects inside a given class of forms. This voting mechanism is carried out in parameter space. Object candidates are produced as local maxima in an accumulator space using the HT algorithm.


y
(x_i, y_i)
ρ
θ
x
Image Space
ρ
$x_i \cos\theta + y_i \sin\theta = \rho$
θ
Parameter Space
$x\cos\theta + y\sin\theta = \rho$

III. INITIAL ATTEMPTS

Initial attempts include using of the opencv library to apply the sobel operator and hough transform.

IV. FINAL APPROACH

The Final code for sobel edge detection: import numpy as np import cv2 import matplotlib.pyplot as plt
def convol(image, kernel): dimensions of the image and kernel image$_h, image_w = image.shape kernel_size = kernel.shape[0] print(kernel.shape[0]) kernel_radius = kernel_size//2$
Initializing the result matrix result = np.zeros$_like(image)$
Padding the image to handle borders padded$_image = np.pad(image, ((kernel_radius, kernel_radius), (kernel_radius, kernel_r$
$constant')$
convolution for y in range(kernel$_radius, image_h + kernel_radius) : for x in range(kernel_radius, image_w + kernel_radius) : Extracting the region of interest (ROI) roi = padded_image[y - kernel_radius : y+kernel_radius+1, x-kernel_radius : x+kernel_radius+1] Computing the convolution sum result[y - kernel_radius, x - kernel_radius] = np.sum(roi * kernel)$
return result

def sobel(image, threshold=100): Converting image to grayscale gray$_scale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)gray_scale = cv2.GaussianBlur(gray_scale, (3, 3), 0)$

Defining Sobel kernels for horizontal and vertical edge detection

sobel$_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])$

sobel$_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])$

Convolving the image with Sobel kernels

grad$_x = convol(gray_scale, sobel_x)grad_y = convol(gray_scale, sobel_y)$

Computing the gradient magnitude grad$_mag = np.sqrt(np.square(grad_x) + np.square(grad_y))$

Normalizing gradient magnitude to [0, 255]

grad$_mag* = 255.0/grad_mag.max()grad_mag[grad_mag < threshold] = 0$

return grad$_mag$

Loading image image = cv2.imread('Photos/table.png')

Resizing the image to improve processing speed image = cv2.resize(image, (500, 500))

ima$_blur = cv2.GaussianBlur(image, (3, 3), 0)$

Performing edge detection edges = sobel(ima$_blur, threshold = 150)$

Saving the detected edges as a PNG image

cv2.imwrite('edge.png', edges)

Displaying the original and edge-detected images using Matplotlib plt.figure(figsize=(10, 5)) plt.subplot(1, 2, 1) plt.imshow(cv2.cvtColor(image, cv2.COLOR$_BGR2RGB))plt.title('OriginalImage')plt.axis('off')$ plt.subplot(1, 2, 2) plt.imshow(edges, cmap='gray') plt.title('Edges Detected') plt.axis('off') plt.show()

Final code for Hough transform:

import numpy as np import cv2

def binarise(image, threshold=128): Converting image to grayscale gray$_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)$

Binarising image using the given threshold $_, binary_image = cv2.threshold(gray_image, threshold, 255, cv2.THRESH_BI$

return binary$_image$

def hough(image, theta$_res = 1, rho_res = 1, threshold = 100) : Definingtherangeoftheta(0to180degrees)andrho(-max_distancetomax_distance)theta_range = np.deg2rad(np.arange(0, 180, theta_res))max_distance = int(np.sqrt(image.shape[0] **2 + image.shape[1] **2))rho_range = np.arange(-max_distance, max_distance + 1, rho_res)$

Initializing the Hough accumulator accumulator = np.zeros((len(rho$_range), len(theta_range)), dtype = np.uint64)$

Finding edge pixels in the image

edge$_pixels = np.argwhere(image > 0)$

Loop over each edge pixel for y, x in edge$_pixels : Loopovereachthetavaluefortheta_idx, thetainenumerate(theta_range) : rho = int(x * np.cos(theta) + y * np.sin(theta))rho_idx = np.argmin(np.abs(rho_range - rho))accumulator[rho_idx, theta_idx] += 1$

Finding indices of accumulator cells above the threshold rho$_idxs, theta_idxs = np.where(accumulator >= threshold)$

Extracting rho and theta values for detected lines rhos = rho$_range[rho_idxs]thetas = np.rad2deg(theta_range[theta_idxs])$

return rhos, thetas

def lines(image, rhos, thetas): for rho, theta in zip(rhos, thetas): a = np.cos(np.deg2rad(theta)) b = np.sin(np.deg2rad(theta)) x0 = a * rho y0 = b * rho x1 = int(x0 + 1000 * (-b)) y1 = int(y0 + 1000 * (a)) x2 = int(x0 - 1000 * (-b)) y2 = int(y0 - 1000 * (a)) cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

Reading the edge image

edge$_image = cv2.imread('edge.png')$

Binarising the edge image

binary$_edge_image = binarise(edge_image, threshold = 231)Adjustablethreshold$

Applying Hough Transform rhos, thetas = hough(binary$_edge_image, theta_res = 1, rho_res = 1, threshold = 101)Adjustablethreshold$

Reading original image original$_image = cv2.imread('Photos/table.png')original_image = cv2.resize(original_image, (500, 500))$

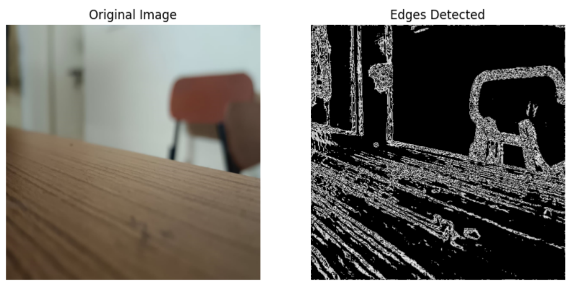
Drawing detected lines on the original image lines$_image = original_image.copy()lines(lines_image, rhos, thetas)$

Saving the final result

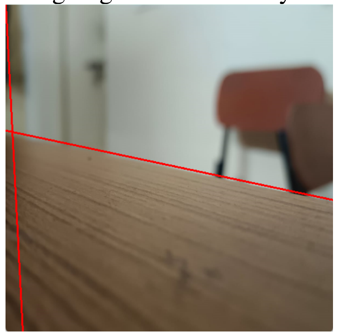
cv2.imwrite('detected$_lines.png', lines_image)$

Displaying the final result cv2.imshow('Detected Lines', lines$_image)cv2.waitKey(0)cv2.destroyAllWindows()$

V. Results and Observation



As evident from the above images, the sobel operator has detected various edges where it can find intensity difference going from the x or y direction.



The hough transform has detected edge lines as indicated by red lines.

VI. Future Work

One major problem with the the sobel operator is it is very sensitive to noise and intesity changes, as a result it can detect many false edges which can hamper the ouput. To tackle this problem gaussian blur is used to minimise noise in the image and thresholding is used to filter out intensity changes. Still the algorithm does not seem very efficient as the resultant output gives out 2 edge lines as opposed to only 1 that is necessary. Thresholding is used at several places including when binarising the image.

CONCLUSION

The sobel operator and the hough transform are some of the most widely used algorithms for edge detection algorithms. For this task it is particularly useful as the robot Luna can detect the table edge and either stop its movement or deflect its direction to avoid falling.

References

[1] "Sobel Operator," www.tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/dip/sobel$_{o}perator.htm.$

[2] https://www.projectrhea.org/rhea/index.php/An$_{I}mplementation_{o}f_{S}obel_{E}dge_{D}etection/$

[3] https://www.analyticsvidhya.com/blog/2022/06/a-complete-guide-on-hough-transform/: :text=Hough