Save Luna!!

Samarth Barve

***Abstract*— OpenCV provides diverse applications of image processing. In this paper, we try and guide Luna, a robot which has wheels and two cameras, over a table. We implement computer vision based models in python using the ”opencv” library for detecting and modifying the images.**

We have used Sobel and Laplacian based edge detectors and Semi Global Matching algorithms for depth detection.

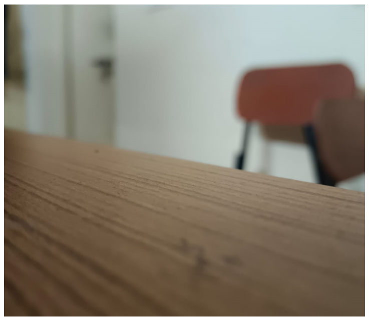
I. Introduction

The task revolves around enhancing Luna, a robot equipped with wheels and cameras, to effectively navigate and perceive its environment. Divided into two stages, the first focuses on preventing Luna from falling off a table by detecting its edges using edge detection algorithms like Sobel or Laplacian-based detectors. Subsequently, the Hough Transform algorithm is implemented to identify lines on the table’s edge. The second stage involves saving Luna from colliding with objects on the floor by generating a depth map using images from its left and right cameras. The depth map visualizes object proximity, facilitating safe navigation.

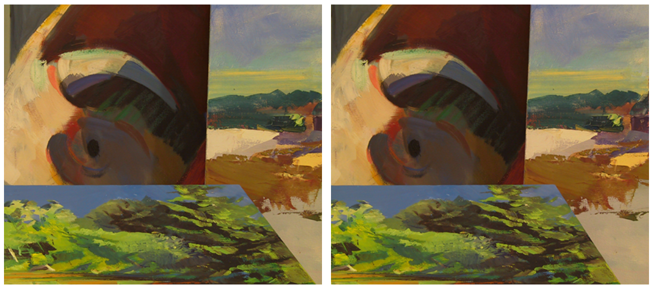
II. Problem Statement

The problem deals with guiding Luna, a robot with wheels and two cameras.

In the first part of the problem, she is kept on a table and we must come up with a code to detect the edges of the table based on the images that her cameras capture. Hence preventing Luna from falling off the table.



In the second part of the problem, based on the (following two) images generated by the right and left cameras of the robot, we must come up with a depth map of the surroundings. Hence identifying the environment around her.



III. Related Work

In the first part, we have worked with Sobel filter, Laplcian filter, Hough transform. The Sobel operators combine Gaussian smoothing and differentiation, so the result is more or less resistant to the noise.

1) **Hough Transform:** The Hough Transform is a feature extraction technique used primarily in image processing and computer vision for detecting geometric shapes, such as lines, circles, or ellipses, in digital images. It works by converting the image space into a parameter space, where each pixel in the image casts a vote for possible shapes it might belong to. By accumulating these votes, the algorithm identifies the most likely shapes present in the image. The Hough Transform is particularly useful for tasks like line detection, where traditional methods like gradient-based edge detection may not be sufficient.
2) **Sobel Operator:** The Sobel Operator is a popular edge detection algorithm used to highlight edges in digital images. It works by convolving the image with a pair of kernels, one for detecting horizontal changes in intensity and the other for vertical changes. These kernels calculate the gradient approximation of the image, which represents the rate of change of intensity at each pixel. By combining the horizontal and vertical gradients, the Sobel Operator detects edges by identifying significant changes in intensity, producing a map of edge strengths in the image.
3) **Laplacian Operator:** The Laplacian Operator, also known as the Laplacian of Gaussian (LoG) operator, is another edge detection algorithm used to highlight edges in digital images. It computes the second derivative of the image intensity to detect regions of rapid intensity change, which typically correspond to edges. The Laplacian Operator is more sensitive to noise compared to the Sobel Operator but can produce sharper edge maps by capturing abrupt changes in

intensity. It is often used in conjunction with Gaussian smoothing to reduce noise before edge detection.

In the second part we work with Stereo Rectification and SGM based models which are used to spot the differences between the left and the right images of the robot cameras and based these imagaes make a depth map of the surrondings.

1) **Stereo Rectification:** Stereo rectification is a preprocessing step in stereo vision that ensures corresponding points in the left and right images lie on the same horizontal scan lines. This simplifies the stereo matching process by effectively transforming the stereo image pair into a simplified setup where disparities can be computed more efficiently. Stereo rectification is achieved by computing rectification transformations based on the intrinsic and extrinsic parameters of the cameras. These transformations align the epipolar lines in the stereo image pair, making it easier to find corresponding points and compute disparities accurately.
2) **Semi-Global Matching (SGM):** Semi-Global Matching (SGM) is a popular algorithm used for stereo correspondence, which is the process of finding corresponding points between the left and right images in a stereo pair. SGM works by computing a cost volume representing the similarity between pixels in the left and right images at different disparity levels. It then aggregates these costs along multiple directions (hence "semi-global") to find the disparity map that minimizes the overall cost. SGM is known for its robustness and efficiency in handling occlusions, textureless regions, and large disparity variations in stereo images. It is widely used in applications such as 3D reconstruction, depth sensing, and autonomous navigation.

IV. INITIAL ATTEMPTS

Initially, we began by learning about 'Canny operator', how it is used to implement 'Hough transformations'. To gain hands-on experience, we opt to implement one of these algorithms from scratch using Python and OpenCV. Our implementation involves convolving the image with the Sobel or Laplacian kernel to compute the gradient magnitude, which highlights regions of significant intensity changes corresponding to edges. We carefully explain the workings of our chosen algorithm in the report, detailing the convolution process and how it identifies edges in the image.

As beginners delving into writing code in OpenCV and Python for stereo rectification and stereo matching using Semi-Global Matching (SGM), our initial steps and approaches are aimed at gaining a comprehensive understanding of the process. We start by familiarizing ourselves with the fundamental concepts of stereo vision, including stereo rectification and SGM. Through research and tutorials, we grasp the significance of stereo rectification in aligning corresponding points in stereo image pairs, simplifying subsequent stereo matching tasks. With this understanding, we delve into OpenCV's documentation and resources to explore the available functions and methods for stereo rectification, such as cv2.stereoRectify. Additionally, we acquaint ourselves with SGM, a powerful algorithm for stereo correspondence, which allows us to find corresponding points between rectified stereo images. By studying examples and tutorials, we gain insights into using OpenCV's cv2.StereoSGBM_create function to perform SGM-based stereo matching. Through hands-on experimentation and iteration, we refine our code, adjusting parameters and techniques to optimize performance and accuracy. Throughout this learning journey, we maintain a curious and exploratory mindset, seeking to deepen our understanding and proficiency in stereo vision techniques with OpenCV and Python.

V. FINAL APPROACH

In our final attempt, we begin by implementing the Sobel edge detection algorithm to identify the edges of the table in the input image. The sobel_edge_detection function takes the input image, converts it to grayscale, applies the Sobel operator in both the x and y directions, computes the gradient magnitude, and normalizes it to obtain the edge image.

Next, we implement the Hough line detection algorithm in the hough_line_detection function. We first binarize the edge image using a binary threshold to obtain a binary edge image. Then, we apply the Hough Transform to detect lines in the binary edge image. Detected lines are drawn on the original image using OpenCV's cv2.line function.

Finally, we read the input image, perform edge detection using Sobel, followed by Hough line detection, and save the resulting image as result.png.

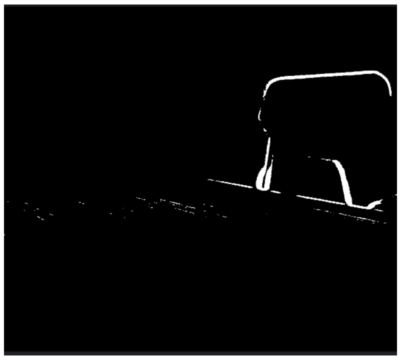
This detailed final attempt combines edge detection and line detection techniques to accurately detect the edges of the table, ensuring Luna's safety.

Building Sobel from scratch

```
def sobel_edge_detection(image):

gray_image =
cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

sobel_x = cv2.Sobel
(gray_image, cv2.CV_64F, 1, 0, ksize=5)

sobel_y = cv2.Sobel
(gray_image, cv2.CV_64F, 0, 1, ksize=3)

gradient_magnitude =
np.sqrt(sobel_x**2 + sobel_y**2)

normalized_gradient =
cv2.normalize(gradient_magnitude, None,
0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

return normalized_gradient
```

Final code snippet

```
import cv2
import numpy as np

image = cv2.imread('table.png')

gray =
cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


gradient_x =
cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=11)
gradient_y =
cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=11)


gradient_magnitude =
np.sqrt(gradient_x**2 + gradient_y**2)


threshold = 2350000
edges =
(gradient_magnitude > threshold) * 255


edges = edges.astype(np.uint8)


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



For the second part of the problem statement, I was not able to get to the final solution but here is part of code that I was able to learn and think about.

ORB

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img =
cv.imread('simple.jpg', cv.IMREAD_GRAYSCALE)

# Initiate ORB detector
orb = cv.ORB_create()

# find the keypoints with ORB
kp = orb.detect(img,None)

# compute the descriptors with ORB
kp, des = orb.compute(img, kp)

# draw only keypoints
location,not size and orientation
img2 = cv.drawKeypoints
(img, kp, None, color=(0,255,0), flags=0)

plt.imshow(img2), plt.show()
```

VI. Results and Observation

The code works fine for the part of the table that is nearer to the viewing plane, but for identifying the edge for the part of the table that is away from the viewer, we have to further increase the kernel size in the sobel operator.

CONCLUSION

Write overall about what the problem was, how you solved it, difficulties faced and the net output with it's usefulness to ARK or in general.

References

[1] Hough Transform, OpenCV.org
[2] Sobel Operator, OpenCV.org
[3] Morphological Image processing techniques
[4] ORB OpenCV.org