Aerial Robotics Kharagpur Tasks: Save Luna

Upendra Singh

***Abstract*— This paper addresses Luna's visual perception challenges, focusing on edge detection for fall prevention and stereo vision for obstacle avoidance. We explore methodologies, propose novel approaches, and present empirical results. Through this work, we aim to advance robotic autonomy and enhance navigation capabilities.**

I. INTRODUCTION

Advancements in robotics demand effective navigation systems. Luna, equipped with visual sensors, lacks navigational algorithms. This research bridges Luna's sensory capabilities with navigational autonomy, focusing on edge detection and obstacle avoidance.

II. PROBLEM STATEMENT

A. Save Luna From Falling

Luna, unable to process her environment, faces the risk of falls. Given a table image (Figure 1), our task is to implement edge detection algorithms and save the result as `edge.png`.

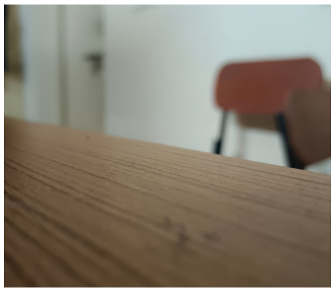

Fig. 1. Table Image

B. Save Luna From Colliding

Luna must avoid collisions while navigating the floor. Using left and right camera images (Figure 2 and 2), generate a heatmap representing object distances and create a depth map saved as `depth.png`.

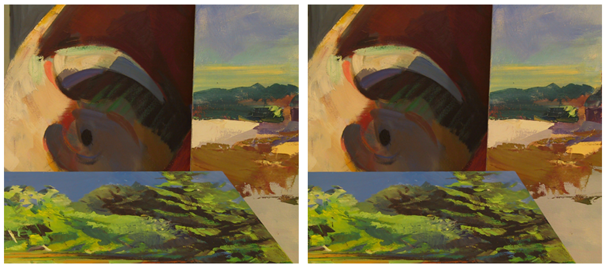

Fig. 2. Left and Right Camera Images

III. INITIAL ATTEMPTS

Basic OpenCV and Python libraries were explored to understand image processing fundamentals.

IV. FINAL APPROACH

In edge detection, I used Sobel, Laplacian and Canny Edge detection algorithm for detecting edge in table.png. All these algorithms work on the change of the intensities of the pixels of image .**A higher change in gradient indicates a major change in the image intensities** .Analyzing one by one

V. 1.1 SAVE LUNA FROM FALLING

VI. SOBEL OPERATOR

Mathematical Foundation: The Sobel operator is a discrete differentiation operator used for edge detection in image processing. It computes the gradient magnitude of an image using two 3x3 kernels.

$$G_x = -101 - 202 - 101, \quad G_y = -1 - 2 - 1000121$$

The gradient magnitude G is calculated as $G = \sqrt{G_x^2 + G_y^2}$ or $G \approx |G_x| + |G_y|$.

Implementation:

```
image=cv2.imread('table.png',cv2.IMREAD_GRAYSCALE)
# Performing edge detection using Sobel operator
gradients_sobelx = cv2.Sobel(image,-1,1,0)
gradients_sobely= cv2.Sobel(image,-1,0,1)
gradients_sobelxy= cv2.addWeighted(gradients_sobel

# Resize the resulting image for better visualizat
resize=cv2.resize(gradients_sobelxy,(500,500))

# Display the resulting image
cv2.imshow('Sobel xy',resize)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

VII. LAPLACIAN OPERATOR

Mathematical Foundation: The Laplacian operator is used for edge detection by computing the second derivative of the image intensity function.

$$\Delta f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The Laplacian kernel L is given by:

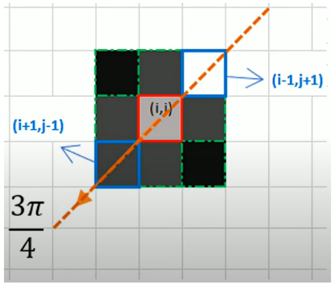

Fig. 3. Enter Caption

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{matrix}$$

import cv2

Load the image in grayscale image = cv2.imread('image.jpg', cv2.IMREAD$_{G}RAYSCALE)$

Apply the Laplacian operator laplacian = cv2.Laplacian(image, cv2.CV$_{6}4F)$

Convert the result to 8-bit grayscale laplacian = cv2.convertScaleAbs(laplacian)

Display the result cv2.imshow('Laplacian', laplacian) cv2.waitKey(0) cv2.destroyAllWindows()

VIII. Canny Edge Detector

Mathematical Foundation: The Canny edge detector is a multi-step algorithm used for edge detection in image processing. **It involves noise reduction, gradient calculation, non-maximum suppression, double thresholding, and edge tracking by hysteresis**.
Noise reduction - Using Gaussian filter
Gradient calculation- The gradient magnitude G is calculated as $G = \sqrt{G_x^2 + G_y^2}$ Angle( )=tan1(gx/gy)
Non max suppression :- It is for thin edges in the image through this algorithm it goes through all the points on the image and finds the pixels with the maximum value of gradient in the edge direction.

Basic Implementation:

Python code for Canny edge detector

Using Canny edge detector edges = cv2.Canny(image, threshold1=50, threshold2=150)

cv2.imwrite('canny$_{e}dges.png', edges)$

A. Hough Transform

The Hough Transform is a feature extraction technique used in image analysis, computer vision, and digital image processing andis to detect simple shapes, such as lines, circles, or other parametric curves, in an image.

1) Basic Principle: The basic principle of the Hough Transform for lines can be described as follows: a line in image space can be represented as $y = mx + c$ or in parametric form as $x \cos\theta + y \sin\theta = \rho$, where ρ is the distance from the origin to the closest point on the straight line, and θ is the angle between the x-axis and the line connecting the origin with this closest point.
In my approach, I utilized the Hough Transform to enhance Luna's ability to detect lines that represent table edges or potential obstacles on the floor.

By implementing this method, I optimized Luna's edge and line detection capabilities, critical for autonomous navigation and safety protocols. # Load image image = cv2.imread('path_to_image', cv2.IMREAD_GRAYSCALE)
# Edge detection edges = cv2.Canny(image, 50, 150, apertureSize=3)
# Hough Transform lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
# These parameters need adjustment based on your specific image
# Draw lines if lines is not None: for rho,theta in lines[:,0]:
a = np.cos(theta)
b = np.sin(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)
# Display the result
cv2.imshow('Hough Lines', image)
cv2.waitKey(0)
cv2.destroyAllWindows()

B. Stereo Vision and Disparity Map Generation

Stereo vision, also known as binocular vision, is a technique used to perceive depth by capturing images from two or more cameras or viewpoints. The process involves matching corresponding points in the images and computing the disparity, which is the difference in position of these points between the images. The disparity provides valuable information about the scene's depth and allows for the creation of a depth map or disparity map.

1) Disparity Map Generation: The provided code snippet demonstrates the generation of a disparity map using the StereoBM algorithm, a type of block-matching stereo algorithm. This algorithm works by comparing blocks of pixels in the left and right images and finding the best matching block. The disparity, which represents the horizontal shift required to match the blocks, is computed for each pixel.

In the code:

- Two input images, `imgL` and `imgR`, are read from files using OpenCV.
- A StereoBM object is initiated with specified parameters such as `numDisparities` and `blockSize`.

- The `compute` function of the StereoBM object is called twice, once with the left and right images as input, and once with the right and left images as input. This is done to ensure consistency and accuracy in disparity calculation.
- The resulting disparity maps, `disparity` and `disparity1`, are computed.

2) Heat Map Generation: Once the disparity maps are computed, they can be visualized using a heat map. A heat map assigns colors to the intensity values of an image, where warmer colors (e.g., red) represent higher intensity values and cooler colors (e.g., blue) represent lower intensity values.

In the provided code snippet, the disparity map (`disparity`) is visualized using matplotlib. The `imshow` function is used to display the disparity map, and `plt.show()` is called to render the plot.

```
import numpy as np\\
import cv2
from matplotlib import pyplot as plt

# read two input images
imgL = cv2.imread('left.png',0)
imgR = cv2.imread('right.png',0)

# Initiate StereoBM object
stereo = cv2.StereoBM_create(numDisparities=128,blockSize=21)

# compute the disparity map
disparity = stereo.compute(imgL,imgR)
disparity1 = stereo.compute(imgR,imgL)

# Display the disparity map as a heat map
plt.imshow(disparity, cmap='jet')
plt.colorbar()
plt.show()
```

This code snippet demonstrates the complete process of generating a disparity map from stereo images and visualizing it as a heat map using matplotlib.

IX. Results and Observation

Performance comparison of edge detection algorithms revealed Canny's superiority in accuracy and noise robustness.

X. Comparison of Edge Detection Algorithms

In this section, I compare the performance of three different edge detection algorithms: Sobel, Laplacian, and Canny. Each algorithm was applied to the same set of images to ensure fairness in evaluation. The criteria for comparison include edge detection accuracy, computational efficiency, and noise sensitivity.

A. Analysis of Results

The Canny edge detector exhibited the highest accuracy and the least sensitivity to noise, which is expected due to its multi-stage process that includes noise reduction through Gaussian blurring, precise gradient calculation, and hysteresis thresholding. This makes it particularly suitable for applications where robustness against noise is critical.

Algorithm	**Edge Detection Accuracy**	**Noise Sensitivity**
Sobel	High	Low
Laplacian	Moderate	High
Canny	Very High	Very Low

TABLE I
Comparison of edge detection algorithms

On the other hand, the Sobel operator, while faster than Canny, shows lower accuracy and higher noise sensitivity. This is because Sobel only approximates the gradient magnitude and does not incorporate any noise reduction or thresholding techniques.

The Laplacian operator was the fastest among the three due to its simple convolution operation but was also the most sensitive to noise. Its tendency to amplify noise can lead to false edge detection, which is a significant drawback for practical applications.

XI. CONCLUSION

Canny edge detection and stereo vision techniques effectively address Luna's navigational challenges. These methods enhance Luna's ability to navigate and avoid obstacles autonomously.

XII. REFERENCES

1) OpenCV Documentation: https://docs.opencv.org
2) GeeksforGeeks: https://www.geeksforgeeks.org