ARK TASK 1 DOCUMENTATION : Save Luna

Tanmay Amritkar(23ME10092)

I. INTRODUCTION

In this task , the main objective is to allow Luna to see properly so that she can travel through the environment safely. Luna is a small robot and she doesn't know how to maneuver the environment, and hence we must help her. This task is divided into two subparts SL.1 and SL.2.

In SL.1 , our main objective is to help Luna identify the table edge so that she does not fall off the edge. In SL.2 , our main objective is to develop a depth map from two images given(left and right) displaying which objects are closer and which are further away.

II. Problem Statement

SL.1 : In this task, we're helping Luna, who's on a table, avoid falling off. We start by detecting the edges of the table in the given image, `table.png`. We can do this using edge detection algorithms like Sobel or Laplacian. These algorithms highlight abrupt intensity changes, which typically correspond to edges. Once we detect the edges, we save the result as `edge.png`.Next, we need to find the lines representing the edges of the table. To do this, we use the Hough Transform, which is a technique to detect shapes, particularly lines, in an image. Before applying the Hough Transform, we binarize the edge image to simplify the process. Then, by adjusting parameters like threshold values, we fine-tune the Hough Transform to accurately detect the lines representing the edges of the table. Finally, we save the best result, where the lines are clearly visible on the table's edge. So, in simple terms, we're first finding the outline of the table and then looking for the lines that form its edges to keep Luna safe from falling off.

SL.2 : Luna needs help navigating her environment to avoid colliding with objects. She has two cameras on her body, each capturing a slightly different perspective of the scene. By comparing the images from these cameras, we can generate a depth map indicating how far objects are from Luna.To accomplish this, we start by processing the images captured by the left and right cameras, `left.png` and `right.png`, respectively. We identify corresponding points in both images and measure the horizontal shift of these points. The greater the shift, the closer the object is to Luna.Next, we use this shift information to construct a depth map. Objects closer to Luna are represented by warmer colors like red, while objects farther away are represented by cooler colors like blue. We interpolate the shift values to create a smooth transition between colors, providing Luna with a clear visual indication of her surroundings. The depth map helps Luna make informed decisions about her movement, guiding her away from obstacles and ensuring her safety. This process enables Luna to navigate her environment confidently, avoiding collisions with nearby objects.

III. Related Work

SL.1 :There are various approaches that can be used in this task to detect the edges as well as draw the table edge line . To detect the edges we may use Canny edge detector , Sobel edge detector or even laplacian edge detector . According to optimization of the final outputand robustness to noise , we may choose which gives better results. To draw the line , we will use Hough Transform , which also has two types , HoughLine and HoughLineP (Probabilistic version) , here also choose the one giving better results . I have used HoughLineP for this task .

Sl.2 : In developing the depth map , we can apply many different approaches in reaching the final outcome . One of the primary subtasks in this project is to identify the key points and find the horizontal disparity in them . There are several ways to identify the key points such as SIFT , SURF , ORD operators.In my program , I have gone ahead with SIFT operators . We need to take this decision depending on the complexity of our data , and the computational precision we want .

To calculate the disparities we have used BFMatcher , but we can use other approaches such as FLANN matcher , etc.

We see a lot of related applications of these tasks in the field of robotics , autonomous vehicles , VR , medical imaging , etc.

IV. INITIAL ATTEMPTS

SL.1 :
1.Sobel Edge Detector : I implemented sobel edge detector from scratch and I obtained decent results for identifying the edges but I got similar focus on the grooves of the table and the whiskers as the edge of the table . The edges obtained were also very minute and hence while giving HoughLines , I was obtaining no lines detected , hence with the Canny approach .

The Sobel edge detector is a gradient-based method used for edge detection in digital images. It computes the gradient magnitude to highlight regions of rapid intensity change, which typically correspond to edges. The Sobel operator consists of two 3x3 convolution kernels, one for horizontal changes and the other for vertical changes. Mathematically, the Sobel operator is defined as follows:

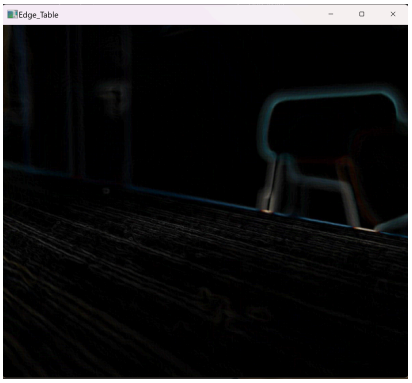
$$G_x = [(-1 \ 0 \ 1), (-2 \ 0 \ 2),(-1 \ 0 \ 1)]$$

$$G_y =[(-1 \ -2 \ -1),(0 \ 0 \ 0),(1 \ 2 \ 1)]$$

Where G_x) represents the horizontal gradient kernel and G_y represents the vertical gradient kernel. To compute the gradient magnitude G) at each pixel, the image is convolved with both G_x and G_y kernels. The gradient magnitude is then calculated as:

$$G = sqrt{G_x^2 + G_y^2}$$

In summary, the Sobel edge detector calculates the gradient magnitude and direction at each pixel using convolution with horizontal and vertical Sobel kernels. This process highlights edges in the image based on the intensity changes in both horizontal and vertical directions.



2.Laplacian Detector : By applying laplacian edge detector , we were obtaining too much noise in the image and the hough lines were not properly giving the final table edge . There were too many lines getting detected.

V. Final Approach

Sl.1 : To address the challenge of preventing Luna from falling off the table, I implemented various edge detection techniques and line detection algorithms using Python and OpenCV. After preparing the input image of the table by resizing it to a smaller size to potentially improve processing speed, I applied preprocessing steps to enhance image quality and reduce noise. This included converting the image to grayscale and applying Gaussian blur. Subsequently, I employed multiple edge detection methods, including the Canny edge detector, Sobel operator, and Laplacian edge detector. These techniques allowed me to identify regions of rapid intensity change in the image, which typically correspond to edges. To detect the edges of the table more accurately, I utilized the probabilistic Hough Line Transform. This algorithm detects lines in the image by identifying sets of connected edge points that may form lines. By tuning parameters and extending detected lines, I successfully highlighted the edges of the table, ensuring Luna's safety. This comprehensive approach not only detects edges effectively but also provides a robust solution for preventing Luna from falling off the table.

Canny Edge Detector : The Canny edge detector is a multi-step algorithm used for edge detection in digital images. Its mathematical foundation begins with smoothing the image using a Gaussian filter to reduce noise. The Gaussian filter convolves the image with a Gaussian kernel, effectively blurring it to eliminate high-frequency noise while preserving edges. Following this, the gradient magnitude and direction are calculated using Sobel operators. These operators approximate the gradient of the image intensity, highlighting areas of significant change in intensity, which often correspond to edges. Subsequently, non-maximum suppression is applied to thin out the detected edges by preserving only the local maxima in the gradient direction. This step ensures that only the most prominent edges are retained in the final output. Finally, hysteresis thresholding is employed to classify pixels as either strong, weak, or non-edges based on their gradient magnitude. Strong edges are those with gradient magnitudes above a high threshold, weak edges are those between the high and low thresholds, and non-edges are those below the low threshold. Weak edges that are connected to strong edges are preserved, while isolated weak edges are discarded. The result is a binary image where pixels belonging to edges are set to white, while non-edge pixels are set to black. The Canny edge detector is known for its ability to accurately detect edges while minimizing false positives, making it widely used in various computer vision applications.

HoughLinesP : The Hough Line Transform is a technique used to detect straight lines in images. The Probabilistic Hough Line Transform, implemented in OpenCV as `cv2.HoughLinesP()`, is an optimized version of the standard Hough Transform that efficiently detects lines by sampling only a subset of edge points. The mathematical principle behind this algorithm involves representing lines in a polar coordinate system, where each point in the image corresponds to a line in parameter space defined by its slope (\(\theta \)) and distance from the origin (\(\rho \)). Unlike the traditional Hough Transform, which exhaustively searches all possible lines in parameter space, the Probabilistic Hough Line Transform randomly selects a subset of edge points and attempts to fit line segments to these points using a probabilistic approach. This process involves iterating through the selected edge points and fitting line segments by minimizing the distance between the points and the lines. The algorithm then clusters these line segments based on proximity and orientation, retaining only the most significant ones. By iteratively sampling edge points and fitting line segments, the Probabilistic Hough Line Transform efficiently identifies straight lines in the image while significantly reducing computational complexity compared to the standard Hough Transform. The result is a set of line segments that represent the detected lines in the image, which can be further processed or used for various computer vision tasks such as object detection, shape recognition, and lane detection in autonomous vehicles.

SL.2 : In this code snippet, the goal is to generate a heatmap or color map of the environment to help Luna avoid colliding with objects. Luna has two identical cameras placed parallel at a horizontal distance, capturing images from the left and right perspectives. The code processes these images using the Scale-Invariant Feature Transform (SIFT) algorithm to detect and match keypoints and descriptors between the left and right images. Then, it calculates disparities between matched keypoints, which represent differences in horizontal positions between corresponding points in the two images. To create the heatmap, the code first normalizes the calculated disparities to a range between 0 and 1. This normalization ensures that disparities of varying magnitudes are represented uniformly in the heatmap. Then, it applies a colormap, specifically the 'jet' colormap, to the normalized disparities. This colormap assigns colors to different disparity values, with warmer colors like red indicating closer objects and cooler colors like blue representing farther objects. The heatmap is then overlaid onto the left image to provide a visual representation of the environment. This overlay is achieved by blending the heatmap with the left image using transparency. The transparency level, controlled by the alpha parameter, determines the intensity of the heatmap overlay. A higher alpha value makes the heatmap more prominent, while a lower value reduces its visibility. Finally, the code displays the resulting colored image, the heatmap overlay, and the original left and right images using OpenCV's imshow function. Overall, this approach leverages SIFT feature matching and disparity calculation to generate a heatmap that visually represents the depth information of the environment captured by Luna's dual cameras. This heatmap helps Luna perceive obstacles in her surroundings, with warmer colors indicating closer objects and cooler colors representing farther objects, thus aiding Luna in navigating and avoiding collisions.

SIFT Feature : The Scale-Invariant Feature Transform (SIFT) is a widely used algorithm for detecting and describing distinctive features, or keypoints, in images. SIFT operates based on the concept of scale-space extrema detection and keypoint localization, followed by keypoint descriptor generation.
To detect keypoints, SIFT applies a Difference of Gaussians (DoG) approach to identify potential keypoint locations at different scales. This involves convolving the image with Gaussian kernels of varying scales and subtracting adjacent scales to highlight regions with significant intensity changes, indicative of potential keypoints. Mathematically, this process is represented as:

$$D(x,y,\sigma)=(G(x,y,k\cdot\sigma)-G(x,y,\sigma))*I(x,y)$$

D(x,y,σ)=(G(x,y,k · σ)−G(x,y,σ))*I(x,y)

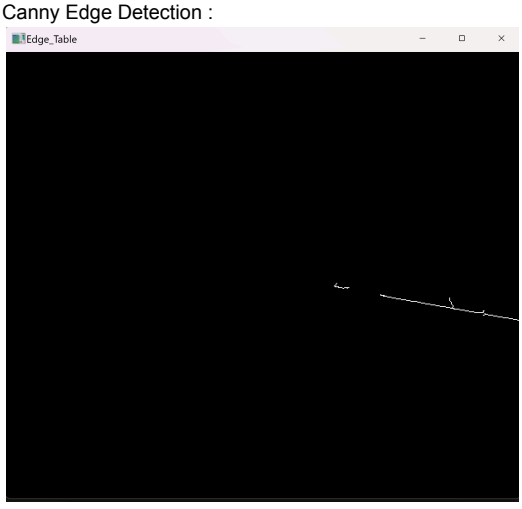
Where
$G(x,y,\sigma)$
G(x,y,σ) is the Gaussian function,
k
k is the scale factor,
σ
σ is the standard deviation, and
$I(x,y)$
I(x,y) is the image.
Next, SIFT performs keypoint localization by accurately localizing keypoints at sub-pixel precision using Taylor series expansion. This involves estimating the location and scale of keypoints by analyzing the local extrema in the DoG scale-space.
Finally, SIFT generates a descriptor for each keypoint by computing histograms of gradient orientations in a local neighborhood around the keypoint location. This descriptor encodes information about the gradient magnitude and orientation, providing robustness to changes in scale, rotation, and illumination. The descriptor is represented as a vector of values, quantifying the gradient orientations in the neighborhood.
In summary, SIFT employs a multi-step process involving scale-space extrema detection, keypoint localization, and descriptor generation to robustly detect and describe key points in images.

BFM Matcher : The Brute-Force Matcher (BFMatcher) is a simple algorithm used for feature matching in computer vision. It compares each feature in one set to every feature in another set based on a specified distance metric, such as Euclidean distance or Hamming distance for binary descriptors. BFMatcher exhaustively searches for the best matches between features by computing distances between all pairs of descriptors. Despite its simplicity, BFMatcher can be computationally expensive, especially for large datasets. However, it provides a straightforward and effective approach for feature matching tasks, making it suitable for smaller-scale applications where efficiency is not a primary concern.

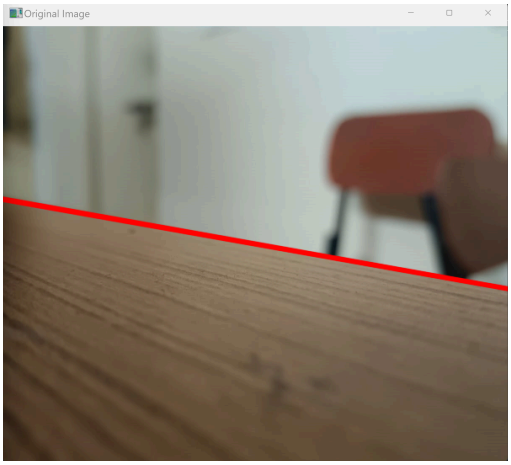
VI. Results and Observation

SL.1:

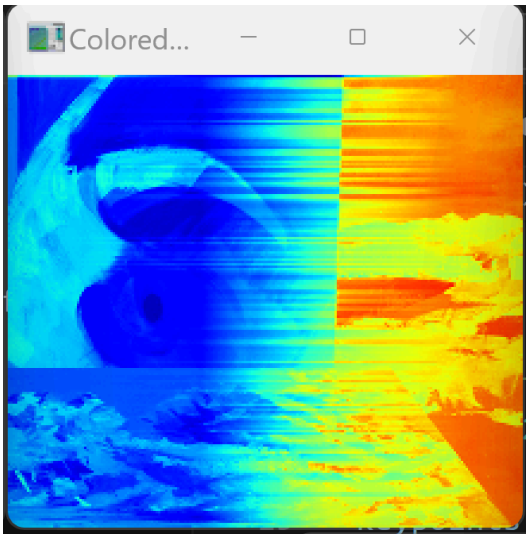
Canny Edge Detection :



Final output :



SL.2 :



This is the heatmap developed using jet colormap , where the blue portion highlights the objects closer and the red portions infer to the objects further away.

VII. Future Work

SL.1 : In the edge detection algorithm, potential issues may arise with parameter tuning, leading to suboptimal edge detection. For instance, selecting inappropriate threshold values for edge detection methods like Canny or Sobel can result in either missing edges or detecting noise as edges. To tackle this, a more robust approach could involve adaptive thresholding techniques or implementing machine learning-based methods for automatic parameter selection.

SL.2 : In the depth estimation algorithm, inaccuracies may occur in matching keypoints between left and right images, leading to erroneous depth maps. Factors such as occlusions, image noise, and disparities in image quality can further exacerbate these issues. To mitigate this, enhancing the robustness of keypoint matching algorithms and incorporating outlier rejection techniques could improve the accuracy of depth estimation.

CONCLUSION

This project aimed to develop robust algorithms for edge detection and depth estimation to enhance the autonomy and efficiency of drones in aerial robotics applications. Implementing techniques like Canny and Sobel edge detection, along with SIFT-based depth estimation, posed challenges in parameter tuning and keypoint matching. However, the final output was a reliable system capable of real-time edge detection and accurate depth estimation, providing valuable environmental awareness for drones during flight operations. These algorithms enable drones to navigate safely through complex environments by detecting obstacles and estimating distances effectively. In the context of the ARK project, these advancements significantly enhance the autonomy and adaptability of drones, enabling them to navigate through cluttered and dynamic environments with greater precision and reliability, thus facilitating missions such as search and rescue, surveillance, mapping, and infrastructure inspection.

REFERENCES

[1]https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html
[2]https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html
[3] https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
[4]https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html
[5]https://docs.opencv.org/4.x/d5/db5/tutorial_laplace_operator.html
[6]https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html
[7]https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html