Save Luna !

Sritoma Nandan

***Abstract*— In this task, the goal is to teach Luna, the robot , how to perceive and navigate her environment effectively. The task is divided into two subtasks, each aimed at preventing Luna's potential accident. Subpart 1 aims at detecting the edge of the table so that the robot does not fall off the table and cause damage and breakage thus reducing wreckage cost. The second subpart aims for obstacle detection using depth map and then color-mapping the same to provide clear information about which obstacle comes first. This is a crucial implementation to any automated robot be it aerial or ground , to help avoid obstacles in real time using perception techniques .**

I. INTRODUCTION

Luna is a robot who equipped with wheels and cameras to help it navigate through the environment. The images perceived by Luna are processed using OpenCV so that Luna does not fall off the table or collide with objects.

In the first subtask, Luna is placed on a table, and the goal is to detect the edges of the table to prevent Luna from falling off. The provided image **table.png** is processed using edge detection algorithm Sobel , to identify the table's edges.

In the second subtask, Luna is placed on the floor with two cameras (left and right), and the objective is to generate a depth map indicating the distance of objects from Luna using the images captured by the cameras (**left.png** and **right.png**) and color it accordingly (red for nearest object blue for farthest).

II. PROBLEM STATEMENT

SL_1

Luna is kept on a table with no obstacles placed on it. She can move freely there. Our task is to detect the table's edges to prevent Luna from falling off using Sobel or Laplacian-based Edge Detectors

Requirements

1. Write an algorithm and properly explain how it works
2. Save the result as edge.png in which edges are clearly visible.
3. Use the edge.png image to detect lines in the table.png
4. Save the best final result after tuning to the best thresholds found in which line is visible on the table's edge.

SL_2

Luna is on the floor and now our task is to save Luna from colliding with the objects. The image clicked by the left and right camera is given as left.png and right.png.

Requirements

1. Build a code (in Python preferably) that can generate a heatmap or colourmap of the environment with objects closer as red color and objects farther as blue color.
2. Explain the whole process
3. Save the final depth map depth.png inside SL 2 with all the necessary code and checkpoints.

III. INITIAL ATTEMPTS

SL_1:

Initially , I invested my time to understand the working of different kernels and about convolution. I came across particular 3x3 kernel used in Sobel edge detection but did not have a desired output . Next I implemented Canny edge detection to see what a feasible edge detection would be for the given problem and took ideas from it ,as I came to know that canny uses a slightly different calculation . I triend with different kernel size and values to get a desired output.

SL_2:

In this task,I initially tried to understand the working of stereo images and implement it by calculating difference between two images using cv2.absdiff() but the problem that aroused was that the disparity calculation was incorrect .The required equation to calculate disparity was:

$$disparity = x - x' = \frac{Bf}{z}$$

Where ,

- x ,x' - are the distance between points in image plane
- B is the distance between two cameras
- f is the focal length of camera

I also implemented the depth map program on other images like that provided on the OpenCV documentation namely tsukuba_l.png and tsukuba_r.png to check for its working .For color mapping of the greyscale I firstly implemented cv2.COLORMAP_RAINBOW but the output was just the inverse of what was desired .

IV. FINAL APPROACH

EDGE DETECTION:

The program has been divided into user defined functions explained individually as follows:

Sobel edge detection: It uses the Scharr() function kernel for edge detection as it calculates more accurate derivative of the image than the sobel kernel . Then the resultant x and y derivatives are added and thresholded accordingly so that it only retains the desired pixels thus returning 'edges'.

```
def sobel_edge_detection(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    sobel_x_kernel = np.array([[-3, 0, 3],[-10, 0, 10],[-3, 0, 3]])
    sobel_y_kernel = np.array([[-3, -10, -3],[ 0,  0,  0],[ 3,  10,  3]])

    sobel_x = cv2.filter2D(gray, -1, sobel_x_kernel)
    sobel_y = cv2.filter2D(gray, -1, sobel_y_kernel)
```

```
    gradient_magnitude= cv2.add(sobel_x,sobel_y)

    mask = cv2.inRange(gradient_magnitude,140, 190)
    result = cv2.bitwise_and(gradient_magnitude,
gradient_magnitude, mask=mask)

    _, edges = cv2.threshold(result, 128,255, cv2.THRESH_BINARY)
    return edges
```

Hough Transform: This function uses the Probabilistic Hough Line Transform to draw the edge that will help Luna from falling over.The cv2.houghLinesP() function returns rectangular coordinated of the line which are then used to filter out undesired edge detection focusing on the table and then finally displaying it .

```
def hough_transform(image):
    hough_image = np.copy(image)

lines=cv2.HoughLinesP(hough_image,1,np.pi/180,90,minLineLengt
h=10,maxLineGap=45)
    if lines is not None:
        for line in lines:
            x1,y1,x2,y2=line[0]
            if(x2-x1)!=0:
                slope=((y2-y1)/(x2-x1))
                if abs(slope)<1.732:
                    y3=(slope*(0+x2)*(-1))+y2
                    cv2.line(img,(0,int(y3)),(x2,y2),(36,33,187),10)
    else:
        pass
    return img
```

Main body: The given image table.png is read using cv2.imread() function and resized accordingly .The edge detection function is called and the returned value of the binary image is saved as edge.png , which is again read and passed for hough line transform implementation.The final image is then shown.

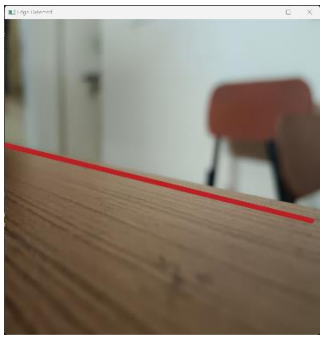
```
img=cv2.imread('table.png')
img=cv2.resize(img,(700,700))

edge=sobel_edge_detection(img)
cv2.imwrite('edge.png',edge)

edge_img=cv2.imread('edge.png',cv2.IMREAD_UNCHANGED)
hough_image = hough_transform(edge_img)
cv2.imshow('Edge Detected', hough_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 1:edge.png



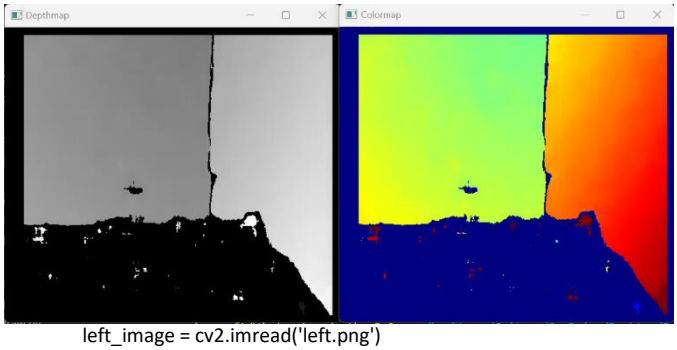
DEPTH MAP GENERATION:

The program functions are as follows :

Generate Depth Map: The function takes the two images left and right and converted to greyscale for proper implementation . We implement the StereoBM class to compute the disparity from the two images after adjusting the parameters numDisparities and blockSize. The resulting disparity is then normalized by the process of cv2.NORM_MINMAX and finally returned.

```
def generate_depth_map(left_img, right_img):
    gray_left = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
    gray_right = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)
    stereo = cv2.StereoBM_create(numDisparities=16, blockSize=21)
    disparity = stereo.compute(gray_left, gray_right)
    disp_normalized = cv2.normalize(disparity, None, alpha=0, beta=255,
norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    return disp_normalized
```

Main body: The left.png and right.png images given are read and then the required function is called . The returned value is given as output and also color mapped using cv2.COLORMAP_JET.




```
left_image = cv2.imread('left.png')
right_image = cv2.imread('right.png')
depth_map = generate_depth_map(left_image, right_image)
cv2.imshow('Depthmap',depth_map)
colormap = cv2.applyColorMap(depth_map, cv2.COLORMAP_JET)
cv2.imshow('Colormap', colormap)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

RESULTS AND OBSERVATION

By the end of the first project it was clear enough that Sobel edge detection was probably not the ideal process for the program . Had it been Canny edge detection , a far better result would have been generated while the code would also have been more brisk and precise .

Calculating absolute difference helped me observed that stereo image processing involves many facets as discussed above .

CONCLUSION

Overall, the task highlighted the importance of robust perception capabilities in robotic systems, with applications spanning various domains such as aerial robotics, autonomous vehicles, surveillance, and industrial automation. Through continuous

refinement and innovation in perception algorithms, we strive to enable Luna and similar robotic systems to navigate their environments safely and effectively, thereby contributing to advancements in robotics technology and enhancing human-machine interaction.

REFERENCES

[1] OpenCV documentation: Sobel Derivatives link
[2] OpenCV documentation: Hough Line Transform link
[3] OpenCV documentation: Canny Edge Detection link
[4] Wikipedia
[5] For thresholding:link
[6] OpenCV documentation:ColorMaps in OpenCV link
[7] OpenCV documentation: Depth Map form Stereo Images link
[8] GeeksforGeeks