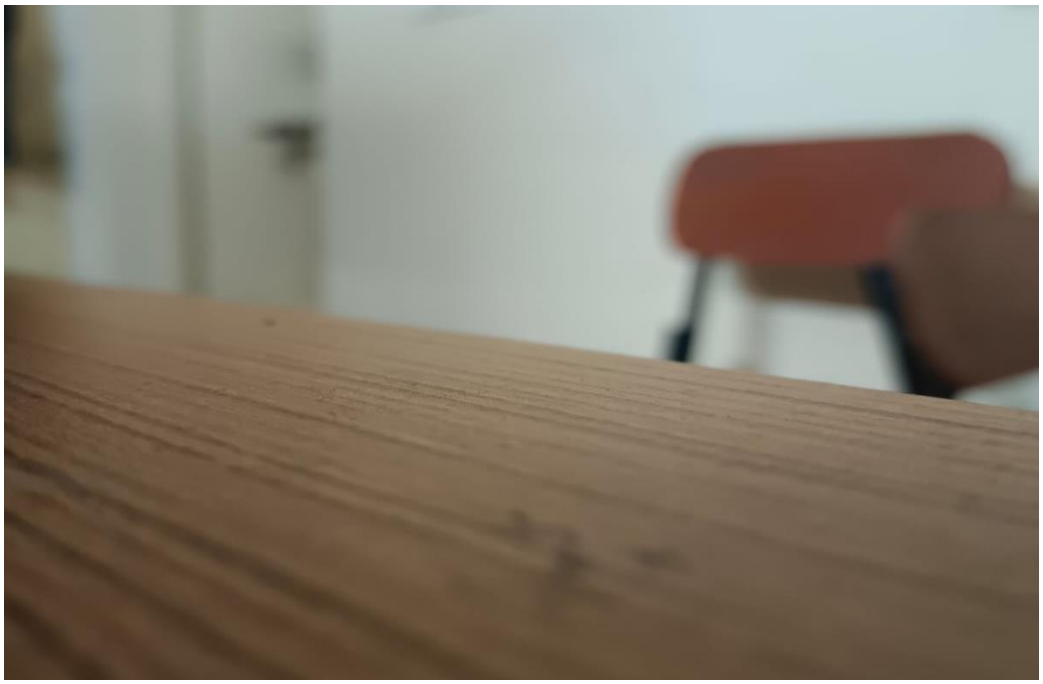
# ARK TASK-1 REPORT

## INTRODUCTION AND PROBLEM STATEMENT:

### Problem statement_1: Save Luna From Falling

- Luna is a robot that is kept on a table with no obstacles placed on it. Your task is to detect the table's edges to prevent Luna from falling off. Now use Hough Transform to find out the lines in the image in which you detected the edges.

## TABLE IMAGE WHICH WAS GIVEN FOR EDGE DETECTION:



### Problem statement_2: Save Luna From Colliding

- Now, Luna is on the floor, and she sees several objects in front of her, but she is unsure whether to move forward or not. Save Luna from colliding with the objects. Build a code that can generate a heatmap or colormap of the

environment with objects closer to red color and objects farther to blue color.

## GIVEN LEFT AND RIGHT IMAGES FOR DEPTH MAP CALCULATION:

**LEFT IMAGE** **RIGHT IMAGE**

## MY APPROACH:

### FOR EDGE DETECTION:

I used the **Sobel** edge detection technique for calculating the edges in the table image. Initially, I resized the image and converted the colored image having RGB channels to a grayscale image based on the luminosity of human eyes. (0.21 * r + 0.72 * g + 0.07 * b). Then I applied **Gaussian blur** on the image to reduce the noises in the image. After reducing the noise I used **kernels in the x and y directions** to convolve it with the image and calculated the gradient in x as well as y directions.

These are the Sobel kernels used for convolution:

**sobel_x** = np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]])

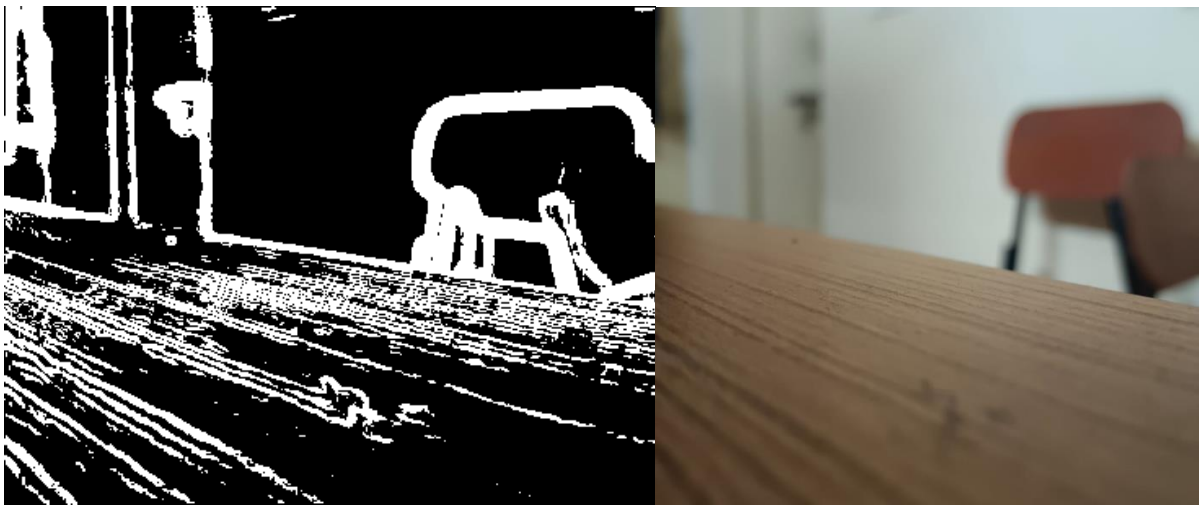
**sobel_y** = np.array([[-1, -2, -1],

[0, 0, 0],

[1, 2, 1]])

Then I calculated the gradient magnitude for each pixel and normalized the gradient magnitude in the range [0,255]. For better visualization of the edges, I added a factor of **alpha=2.2** (gets multiplied by each pixel) and **beta=1**(gets added to each pixel value) for better/enhanced visualization of the edges. Finally, I applied thresholding to my image and set a **threshold value of 21**.

## MY RESULTS FOR EDGE DETECTION:

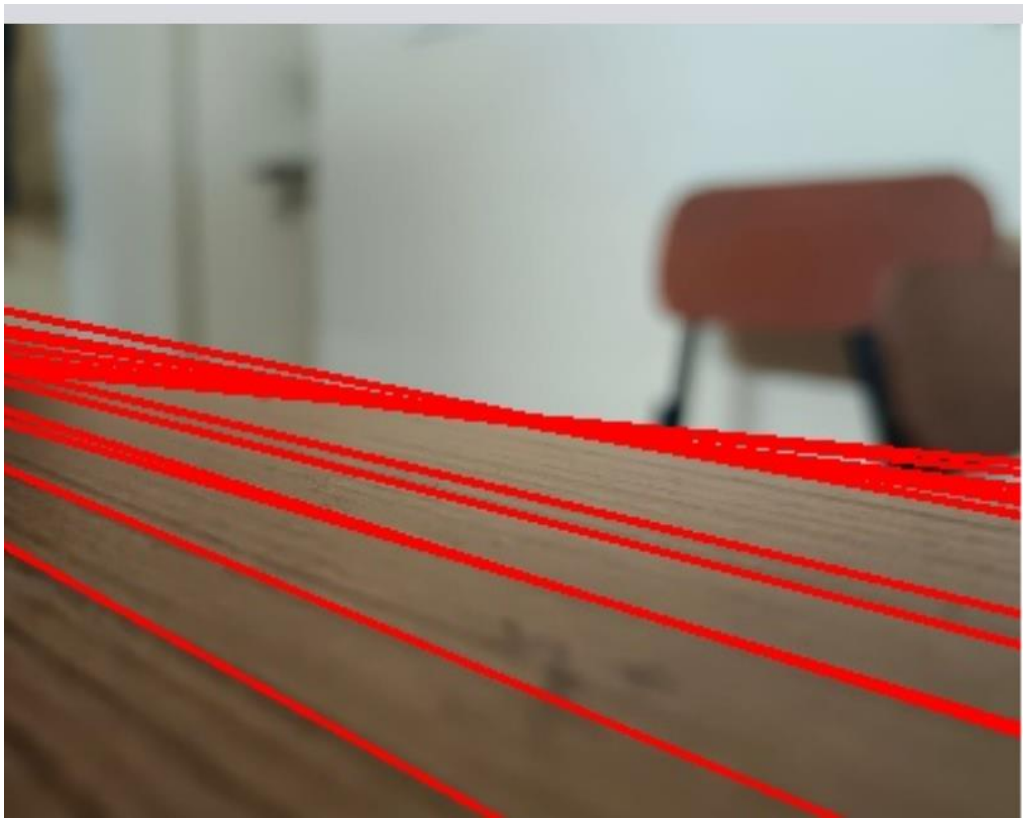


## FOR HOUGH TRANSFORMATION:

Firstly, for the hough transformation, I used **Python's built-in function** for calculating the hough lines on the table using my edge image mentioned above.

**Hough_lines = cv2.HoughLines(edge_image, rho=1, theta=np.pi/180, threshold=292)**

## RESULTS:



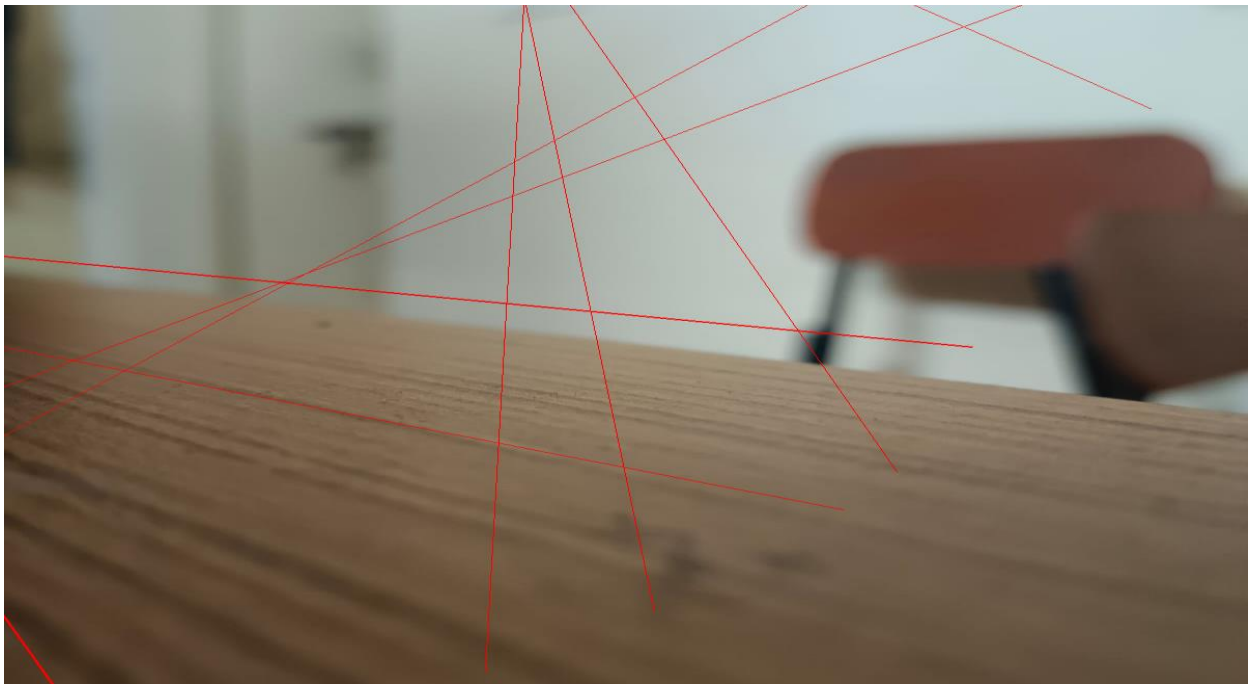
Secondly, I implemented the **Hough transformation from scratch.**

1. **Hough Transform Function (hough_transform):**
    - This function takes an edge image and a threshold as input.
    - It sets up the Hough space in (rho, theta) to represent lines.
    - An accumulator array is initialized to store votes for each possible line.
    - Edge points are obtained from the input image.
    - For each edge point, the accumulator array is updated by calculating the corresponding rho value for each theta.
    - Lines with votes above the specified threshold are identified and returned.
2. **Main Code:**
    - Reads the edge image from the file (**'edge.png'**).
    - Calls the **hough_transform** function to detect lines in the edge image.
    - Reads the original image (**'table.png'**) for visualization.
    - Iterates through the detected lines and extends them to the edges of the image.

- Draws the detected lines on the original image in red color with a thickness of 1.
- Saves the resulting image with detected lines as **'Hough_lines_from_scratch.png'.**

## RESULTS:

**This is the best from my side that I could detect using hough transform from scratch**.

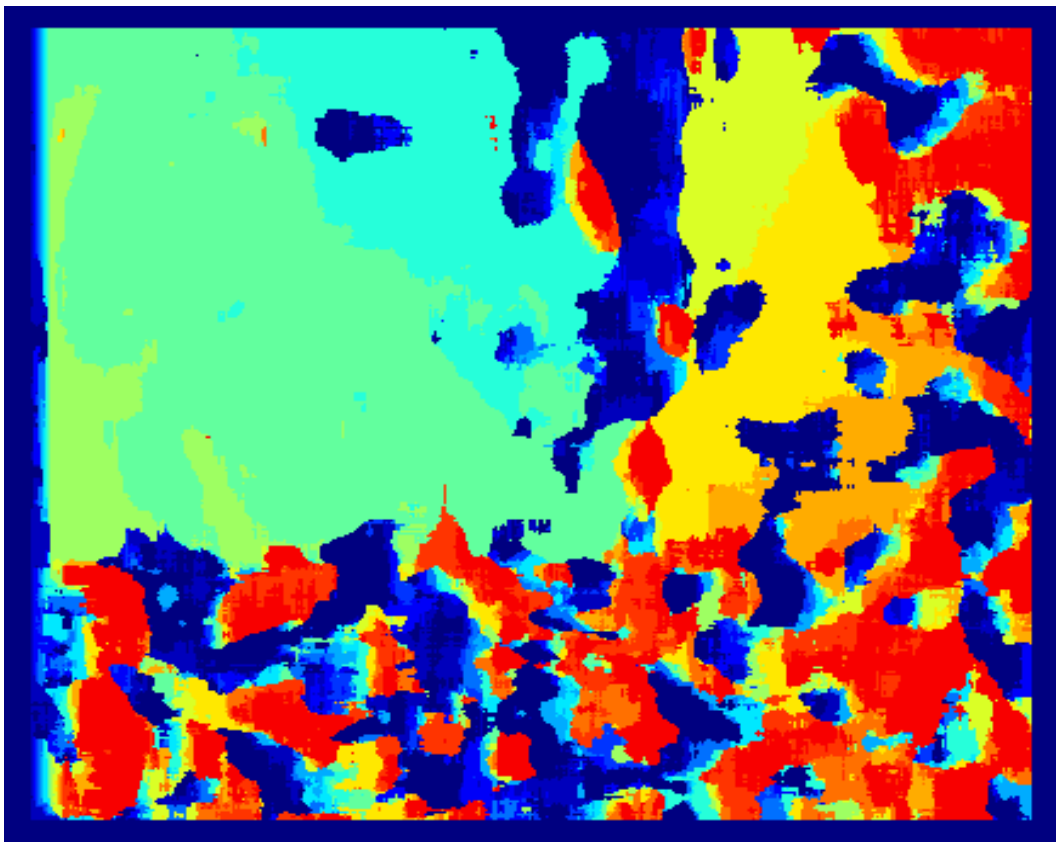


## FOR DEPTH MAP:

For calculating the depth map from scratch I did **stereo matching** in the left and the right images using a **disparity range of 16 and a block of size 20**. What I did was iterate over the pixels in the left image using a nested loop and then construct a square box around that particular pixel. Then afterwards using horizontal displacement values(disparity

values in the disparity range) I constructed different square boxes with shifted horizontal positions on the right image and calculated the difference in the pixel values one by one. The value of the disparity where I would get the minimum difference would be selected and would be mapped to that particular pixel.

## RESULT:



## CONCLUSION:

Overall the problem was fascinating and a little bit challenging. I faced some difficulties while implementing the Hough transform from scratch as even after various attempts

I could not get the desired results. Still, after several attempts and trying many threshold values I have given you my best result for the line detection.

## REFERENCES AND LINKS:

**https://github.com/adamiao/sobel-filter-tutorial**

**SUBMITTED BY-**
**AYUSH GOEL**

**ROLL NO-23EX10008**