CO4204

Computer Vision

Lab 03

Road Lane Detection

NAME : ILLANGARATHNA A.N.L.

INDEX NUMBER : 19/ENG/033

REGISTRATION NUMBER : EN93917

SUBMISSION DATE : 12/23/2023

Introduction

Lane detection is very important in autonomous vehicles and driver assistance systems that use AI . It is about tracking the lanes on a road, enabling the vehicle to stay within its lane and stay safely on the go. In this report, the lane detection algorithm implemented using Python using OpenCV and NumPy is discussed.

Methodology

In this current code, The process is started by loading a sequence of images captured from a moving vehicle camera feed. The images are loaded using the OpenCV library from Python, and a series of preprocessing are applied to enhance the lane detection process and it is as follows.

1. Original Images

Figure 01: Original Images

2. Grayscale Conversion

The color images are converted to grayscale to simplify processing. So the image is represented using 0 to 255 intensity range. So working on these images after this is relatively easier.

Figure 02: Grey Scaled Images

3. **Sobel Edge Detection**

The Sobel operation is used to detect edges in grayscale images. It is an edge detection operator. It calculates the gradient of image intensity change and detects spikes, highlighting edges.

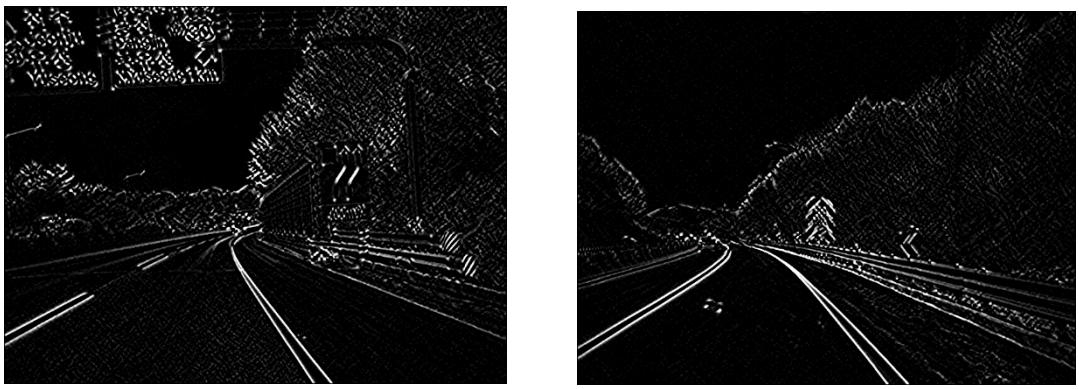

Figure 03: Sobel Edge Detection Images

4. **Region of Interest Masking**

It is used to define which area of the image we are interested in considering. It can is to make an accurate process and reduce the computational load.

5. **Edge Pixel Detection**

To locate the pixels corresponding to the edges of the lanes, this custom algorithm is used. Starting from the center of the image, which is defined as a hardcoded calculation, the algorithm searches for edge pixels to the left and right and detects and adds to an array when intensity is changed drastically, creating separate lists of left and right edges.

Figure 04: Visualizing Detected edges Images

6. **Hough Transform**

The Hough transform is then applied to identify lines in the edge-detected image. This technique converts the image space to the parameter space, allowing the identification of lines even in the presence of noise and gaps. (Figure 05 below)

7. **Lane Averaging and Visualization**

After obtaining lines from the Hough transform, the algorithm performs slope averaging to distinguish between left and right lanes. The lanes are then visualized by drawing lines on the original image.

Figure 04: Lanes detected using Hough transformation and intercept point

8. **Video generation**

After creating all the lane lines on those frames, a script is used to combine all in the correct order to generate the video again.

Code Walkthrough

This is what each function does in the code.

slope_averaging_custom(image, lines)

It carefully examines each detected line by Hough transformation, calculates its slope and intercept, and then categorizes them into left and right lanes. Finally, it averages the slopes for each side.

coordinates_maker(image, line_parameters)

Given the slope and intercept, it efficiently calculates the coordinates to draw a line on the image.

find_edge_pixels(edge_detected_image, center_x_position)

This function starts from the middle, looks left and right, and collects the pixel of where those lane edges might be.

sobel_edge_detection(image)

This function wields the Sobel operator to reveal the edges in the gray scaled image. It is about detecting drastic intensity changes.

5. sobel_edge_visualization(image)

This function is about visualization. It does the sobel operation for the visualization purposes.

6. find_and_visualize_edge_pixels(image, center_x_position)

This function not only finds the edge pixels derived using Sobel, but also paints them in different colors for left and right lanes. (figure 04)

7. **line_intersection_coordinate(lines)**

This function finds that exact point where detected lane lines intercept each other.

8. generate_lines(image, lines)

Given the intersection point, it draws lines on our image connecting that point to the detected lanes. It is about visualization.

9. process_area(image)

It decides which part of the image is important and masks out the rest. This way, the code focuses only on the region where lanes are likely to be found.

10. hough_transform(img)

It consider the edges found using **find_and_visualize_edge_pixels** function. Then it finds where lines are using the Hough transform custom implementation method.

11. find_lines(hough_space, thetas, diag_len, threshold=100)

It filters out the lines that aren't important, leaving only those that have a significant impact on our lane detection function.

12. detect_lanes(img)

This function conducts the entire lane detection operation. It orchestrates Sobel edge detection, interested area processing, Hough transform, and line filtering to create the visual representation of the detected lanes.

13. video_generator(input_folder, output_folder, name)

This function combines all the images into a video. Then the detected lanes comes to life.

Important.

Line 69 has a this code like follows,

```
for y in range(height//2, height):  # Start from the middle of the image and go to the bottom
```

It is related to the Region Of interest. To define where the script should consider when each image is processed. In it the the bottom half of the image is considered. Below I have compared with this bottom half only and without the bottom half only constraint related outputs generated.

- Without bottom half constraint.

Figure 05: Unnecessary parts are considered in edge detection

- With bottom half constraint.

Figure 06: Only the necessary parts are considered in edge detection