Finding Lane Lines on the Road

I. Introduction

A. Background on the problem of lane detection in autonomous driving

The traditional image processing approach for lane detection involves a series of steps to extract lane lines from an image. These steps include color filtering to extract white and yellow pixels, region of interest selection, edge detection, and Hough transform to detect lines. The output of the pipeline is a set of lane lines that can be overlaid on the original image.

B. Objective of the paper

The objective is to recreate a traditional image processing pipeline for lane detection and identify any shortcomings in the method. The pipeline consists of several steps including color filtering, region of interest selection, edge detection, and Hough transform for line detection. By recreating this pipeline, we can analyze its performance and identify areas where it may fall short in accurately detecting lanes.

II. Methodology

1. Description of the pipeline

Tools:

* Python 3.8
* Jupyter Notebook
* MoviePy library for working with video files

Libraries:

* matplotlib
* numpy
* cv2 (OpenCV)
* os
* math

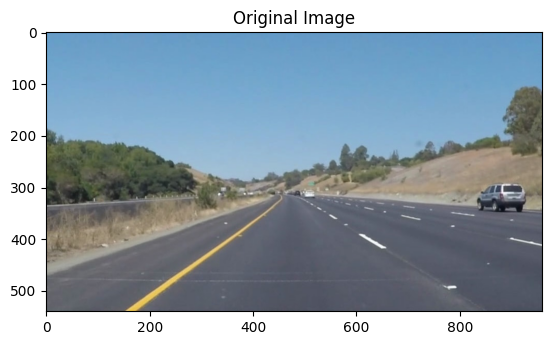
Diagram

Description automatically generated

Figure 1: Pipeline

1. Image preprocessing:

The image is read in the first step of the lane detection pipeline to obtain key parameters such as average image brightness and image shape. Typically, the image is a photograph or a video frame captured by a camera mounted on a vehicle. The image dimensions, which are required for subsequent processing steps, and the average image brightness, which is used to adjust the color thresholds of the lane lines to make them visible in the image, are among the key parameters we extract from the image.



The average image brightness is calculated by adding the RGB values of each pixel in the image and then dividing the total number of pixels in the image by the total number of pixels in the image. A higher average brightness suggests that the image contains lighter colored lane lines, whereas a lower average brightness suggests that the image contains darker colored lane lines. This information is used to optimize the detection of lane lines in the image by adjusting the color thresholds of the lane line detection algorithm.

2. Color filtering

In the lane detection pipeline, color filters are used to isolate the white and yellow pixels in the image. The color filters are defined using lower and upper color thresholds for white and yellow colors. These color thresholds are defined as numpy arrays of RGB values, where the lower color threshold is the minimum RGB values for the color of interest and the upper color threshold is the maximum RGB values for the color of interest.

Once the color thresholds are defined, they are used to create binary masks of the same size as the original image. The binary masks have value 1 for pixels that fall within the color range and 0 for pixels that fall outside of the color range. By using these binary masks, the pipeline can isolate only the white and yellow pixels from the image, which helps to better detect the lane lines.

Chart

Description automatically generatedChart

Description automatically generated

Finally, the binary masks are combined using a logical OR operation to create a single color filter that includes both white and yellow pixels. Chart

Description automatically generated

3. Region of interest segmentation

The ROI is defined as a polygon that is fitted to the area of the image where the lanes are expected to be present. The polygon is defined by a set of vertices that are chosen based on the position of the camera and the expected location of the lanes on the road. Typically, the polygon is defined as a trapezoid, with the narrow end at the top of the image and the wide end at the bottom of the image.

Once the ROI has been defined, it is applied to the original image as a mask. Outside the ROI, the pixels are set to zero, effectively removing them from the image. This generates a new image that only contains the portion of the original image that is relevant for lane detection.

Chart

Description automatically generated

4. Canny edge detection

To reduce computational load, the Canny algorithm first converts the image to grayscale. To reduce noise and unwanted details, the grayscale image is smoothed with a Gaussian filter. The Sobel operator is then used to compute the gradient magnitude and orientation. The edges are then thinned to one pixel width using non-maximum suppression. Finally, a threshold is applied to the edges, with two values: high and low thresholds. Edges with magnitudes greater than the high threshold are considered strong, while those between the low and high thresholds are considered weak. Edges that fall below the low threshold are ignored. The Canny algorithm produces a binary image with white pixels representing the edges. Chart

Description automatically generated with low confidence

Chart

Description automatically generated

5. Hough transform

The Hough transform algorithm uses the edge-detected image from the previous step to identify lines through a voting process. The transform works by mapping the image's edge pixels to the Hough space, which is a parameter space defined by a line equation. In the Hough space, each edge pixel votes for all the lines that pass through it. The votes are accumulated in the Hough space, with the peaks representing the lines in the image. The draw lines() function then takes the Hough transform output, which is a set of lines, and draws them onto a black image with the same dimensions as the original input image.

First, it divides the Hough transform lines into two groups: those with a positive slope (corresponding to the right lane) and those with a negative slope (corresponding to the left lane). The average slope and intercept for each group are then calculated by averaging the values for all lines in the group.

Next, it determines the y-coordinates corresponding to the bottom of the image and the top of the region of interest. Using the slope and intercept for each group, it calculates the x-coordinates corresponding to these y-coordinates, resulting in the two endpoints of each lane line.

Finally, it draws the two-lane lines onto the black image by connecting the endpoints using the cv2.line() function.

Chart, line chart

Description automatically generated

6. Output visualization

Once the lines have been drawn, we blend this image with the original image using the cv2.addWeighted() function. This creates a new image where the lane lines are overlaid on top of the original image. We can then save this image to disk or display it on screen using matplotlib.

A car driving on a road

Description automatically generated with medium confidence

III. Results

A. Performance metrics

The performance of this Pipeline can be evaluated through a few ways.

Video processing time: Performing lowlight adaptive thresholding reduced the processing speed from over 50 iterations per second to just over 30 iterations per second. While tuning parameters and setting detection thresholds, it was observed that achieving higher accuracy in challenging scenarios often required a tradeoff between processing time.

Lane line detection robustness: By running the pipeline on a set of diverse images and comparing the output with ground truth annotations, we can assess the performance of the pipeline and identify any potential weaknesses or limitations. It is important to ensure that the evaluation dataset includes examples of challenging scenarios, such as those with varying lighting conditions, occlusions, and road markings. By analyzing the results and identifying areas for improvement, we can iteratively refine the pipeline and increase its robustness over time.

Low light setting: In low light conditions, lane detection can be a challenge due to the lack of contrast between the lane lines and the road surface. This can result in the detection of false positives or false negatives, leading to incorrect lane markings.

Shadows: The presence of shadows on the road can cause problems in lane detection, as the shadows may affect the color and brightness of the road surface, leading to inaccuracies in the color filter and edge detection steps. This can result in a failure to detect lane lines, or the detection of false lane lines. Additionally, shadows cast by overhead structures such as trees or buildings can create dark areas on the road that may be interpreted as lane lines by the algorithm, leading to further inaccuracies.

Tailing cars: Possible problems in lane line detection when tailing a car include reduced visibility of lane markings due to the car in front obstructing the view, and difficulty in distinguishing the lane markings of the car in front from the driver's own lane markings due to perspective distortion. These issues can affect the accuracy of lane detection and increase the risk of accidents.

C. Comparison with state-of-the-art methods

Our current implementation is basic and serves the purpose of creating an interpretable lane detection algorithm. However, compared to current state-of-the-art methods, there is still a sizable performance gap. Modern deep learning methods achieve high performance in lane detection, but it is still difficult to accurately detect lanes under various conditions such as shadows, occlusions, and curved roads. State-of-the-art algorithms use CNNs for both tasks, with significant consumption of computing resources.

IV. Discussion

A. Interpretation of results

We have 4 videos in total that we use to test our pipeline.

* White lane lines
* Yellow lane lines
* Mixed color along with curved road and tree shadows. Also, a portion of the road looks to have spilled paint.
* Dark setting on Taiwan highway.

For the first two videos our pipeline consistently performs well. However the third and forth videos have vastly different settings and thus, tuning the pipeline with one video cause the pipeline to perform less accurately on the other. The result is a pipeline that performs acceptably on both, but has some obvious limitations.

B. Limitations of the pipeline

While the Low-light adaptive thresholding technique has improved the pipeline's robustness in low-light environments, it also has several limitations that may impact its performance. One significant drawback is that it can reduce the processing speed, which may affect the system's ability to operate in real-time. This is because the technique requires calculating the average pixel intensity on a frame-by-frame basis, which can be computationally intensive.

Moreover, the adaptive thresholding technique may not perform well with frames that have high contrast, such as those with shadows. In these situations, the algorithm's reliance on average pixel intensity can result in inaccurate line detection, as it cannot accurately compensate for variations caused by shadows.

Another limitation of the pipeline is that it lacks continuity between frames, which can cause inconsistencies in line detection. As the Pipeline is applied on a frame-by-frame basis, there is no guarantee that the detected lines will be consistent across frames.

Furthermore, the pipeline's adaptability to different scenarios is limited. This is because the lines are detected mainly by color and doesn’t consider other factors that may affect line detection accuracy.

Overall, while the pipeline's Low-light adaptive thresholding technique has improved its performance in certain conditions, it is not without limitations that must be considered when using the system.

C. Future directions for improvement

There are several possible avenues for future improvements to the lane detection pipeline. One possible improvement would be to explore more advanced computer vision techniques, such as deep learning-based approaches, which have shown promise in recent years for a variety of vision tasks, including lane detection.

Ensuring continuity between frames is crucial for maintaining path prediction stability. If there is a momentary obstruction in the vision, or if the pipeline produces an obviously incorrect line, the vehicle may swerve off course. To address this issue, one potential solution is to calculate an average of the lane line positions from previous frames and check whether the current position is potentially incorrect.

Another area for improvement would be to investigate more sophisticated methods for handling challenging scenarios such as shadows and low light conditions. This could involve in-frame low light adaptive thresholding that will account for different portions within the frame. This could also involve the use of more complex image processing techniques or the incorporation of additional sensors such as lidar or radar.

In addition, there may be opportunities to optimize the existing pipeline by exploring more efficient algorithms or parallel processing techniques, which could help to improve the processing speed and overall performance of the system.

V. Conclusion

A. Summary of findings

The traditional image processing pipeline for lane detection involves various steps such as color filtering, region of interest selection, Canny edge detection, Hough transform, and output visualization. However, this approach has limitations in detecting lane lines under certain conditions such as low light, shadows, and tailing cars. To improve the robustness of the pipeline, lowlight adaptive thresholding can be used, and continuity between frames can be ensured through the averaging of lane line positions.

VI. References

M. A. Alshehri and M. A. Alghamdi, "Overview of Current State of the Art Lane Detection Models and Methods and Testing and Training a SCNN on Google Colaboratory," 2021 International Conference on Computer Vision and Image Analysis Applications (ICCVIA), Riyadh, Saudi Arabia, 2021, pp. 1-6, doi: 10.1109/ICCVIA52656.2021.9467553.

M. Lee, J. Lee, D. Lee, W. Kim, S. Hwang, and S. Lee, “Robust Lane detection via expanded self attention,” 2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2022.