University at Buffalo

Department of Computer Science and and Engineering

CSE 573 - Computer Vision and Image Processing

Fall 2022

Final Project Report

Lane detection using classical image processing

Jason DCosta 50468245 jasondco@buffalo.edu
Shreya Dhareshwar 50471594 shreyadh@buffalo.edu

1] Problem:

We aim to solve the problem of lane detection using classical image processing techniques, which do not require large amounts of training data, are easier to implement on commodity hardware and are computationally lightweight in comparison to deep learning techniques.

1.1] Inputs: Series of images taken from datasets
1.2] Outputs: Series of images with lanes identified in each instance with reasonable accuracy.

2] Datasets:

We used 2 datasets to test the robustness of our lane detection pipeline in the intermediate stages. We started with a simple dataset, which was taken from the Udacity Self Driving NanoDegree Github repository [1], since the images provided are free of distortion, have a well defined lane and do not have many extraneous features to confuse the detection pipeline.

However, the aim of the final lane detection pipeline is to perform lane detection on more complex images. Thus, we observe the results obtained on the CULane Dataset [2], since this data is more indicative of real world cases, containing images taken across multiple lighting conditions, shadows and other objects on the road. It is a challenging dataset used to evaluate performance of deep learning methods of lane detection.

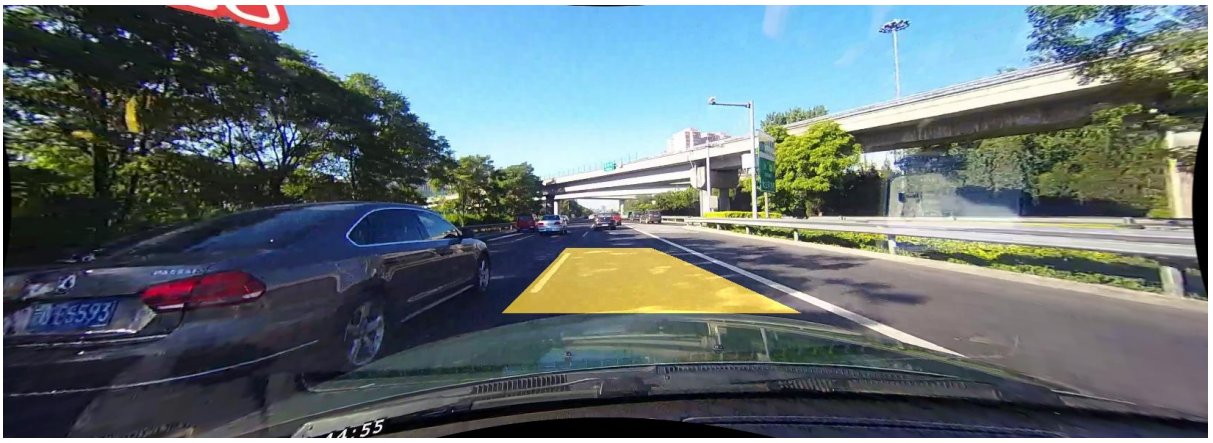
3] **Algorithm:**

1. Convert the original image which is in the BGR color space to the HSV colorspace.
2. Find the image gradient along the X axis using the sobel operator and taking the absolute value of the gradient obtained.
3. Isolate the parts of the image which have color similar to the color of expected lanes.
4. Perform image bitwise OR on the above two intermediate results (gradient and color components) to obtain binary threshold image.
5. Isolate the Region of Interest (ROI) in the image. ROI would be a polygon inscribed by the edges we are interested in i.e lane lines.
6. Perform a perspective transform on the region of interest to get a bird eye's view of the ROI.
7. Compute the histogram for the ROI. There will be two peaks, each one corresponding to the left and right lane.
8. Use the sliding window algorithm to find the points which are part of the lane, for both the left and right lanes.
9. Fit a polynomial to each set of points, the polynomials represent the left and right lanes respectively.
10. Perform an inverse perspective transform and draw the polynomials on the original image.

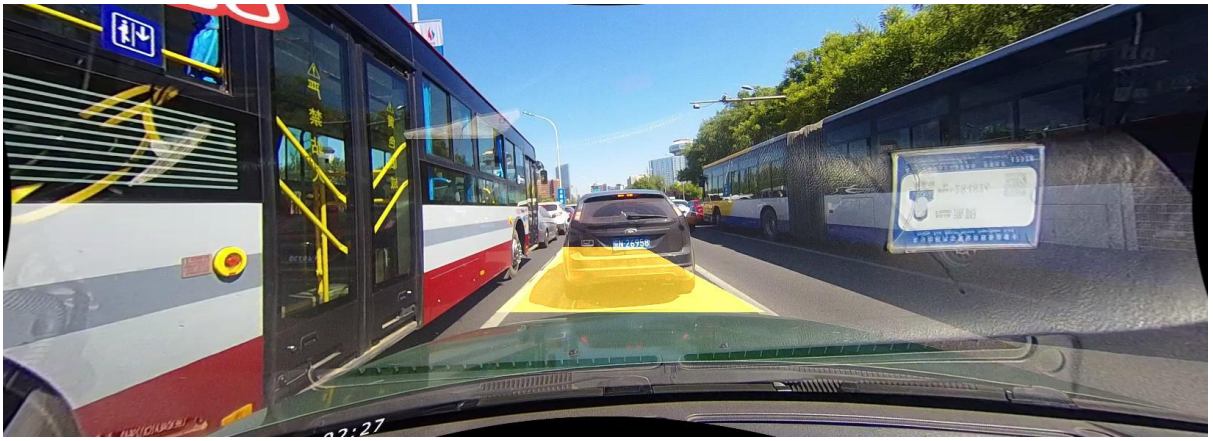
The entire lane detection pipeline is implemented on our own, with help of OpenCV's built-in methods for Sobel Operator, perspective transform, polynomial fitting and methods to read, write and perform color conversion on images.

4] **Results:**

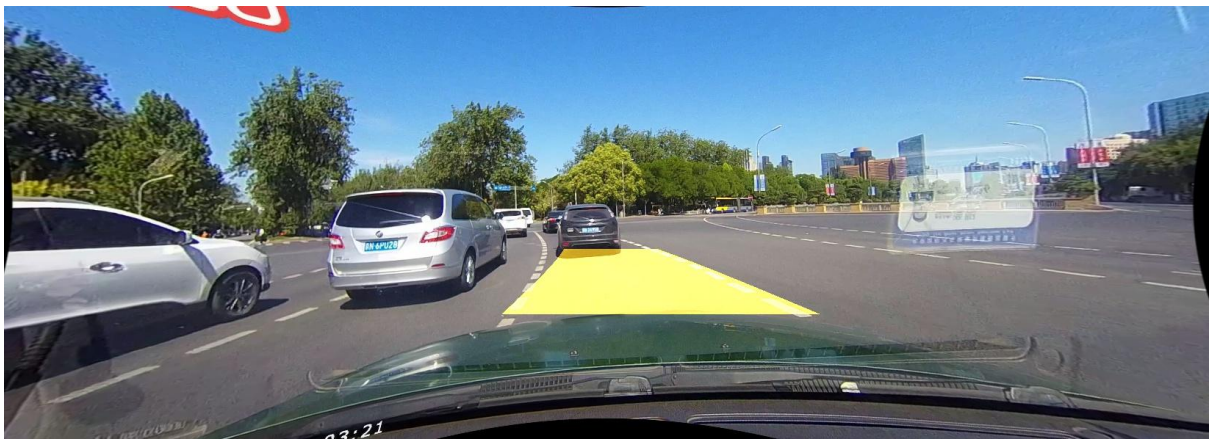
We would like to demonstrate 3 classes of results: a case of lane detection which has challenging lighting conditions, a case in which the lane is occluded and a third case in which the lane is curved.



The above image corresponds to the first case, we see that the scene has challenging lighting conditions due to shadows and the left and right lane boundary having different lighting conditions. The detection pipeline highlights the lane present in this scene and demonstrates the ability of the pipeline to handle varying lighting conditions.



The above image corresponds to the second case, in which the lane is occluded due to the presence of a vehicle in the same region in which the lane is present. This case demonstrates the robustness of the pipeline to handle similar cases in which objects interfere with lane detection.



The above image corresponds to the third case, in which the lane is not straight. This case demonstrates the ability of the pipeline to handle curved lanes as well as straight lanes.

5] Analysis:

1. Since we have used a two degree polynomial to approximate the detected lanes, the pipeline detects curved lanes.
2. The results demonstrate that the final lane detection pipeline successfully handles different lighting conditions, time of day, shadows and dynamic range.
3. The results also demonstrate the ability of the lane detection pipeline to handle objects present in the same region of interest as the lanes.
4. Thus, the final lane detection pipeline improves on the shortcomings of the intermediate lane detection pipeline and gives robust lane detection performance.

6] References:

[1] Udacity. (n.d.). Udacity/Carnd-LaneLines-P1: Lane Finding Project for self-driving car nd. GitHub. Retrieved November 21, 2022, from https://github.com/udacity/CarND-LaneLines-P1

[2] Xingang Pan, & Xiaoou Tang (2018). Spatial As Deep: Spatial CNN for Traffic Scene Understanding. In AAAI Conference on Artificial Intelligence (AAAI).

[3] OpenCV Documentation. OpenCV. (n.d.). Retrieved December 8, 2022, from https://docs.opencv.org/4.x/