Finding Lane Lines on the Road

Author: Yusuf T. Tueysuez
Submitted: April 16, 2020

Finding Lane Lines on the Road

The goals / steps of this project are the following:

- Create a pipeline that finds lane lines on the road using images and video input
- Reflection of how the problem was approached and potential shortcomings

Example: Input image containing typical highway situation

Example: Output image showing detected lanes in image

Reflection

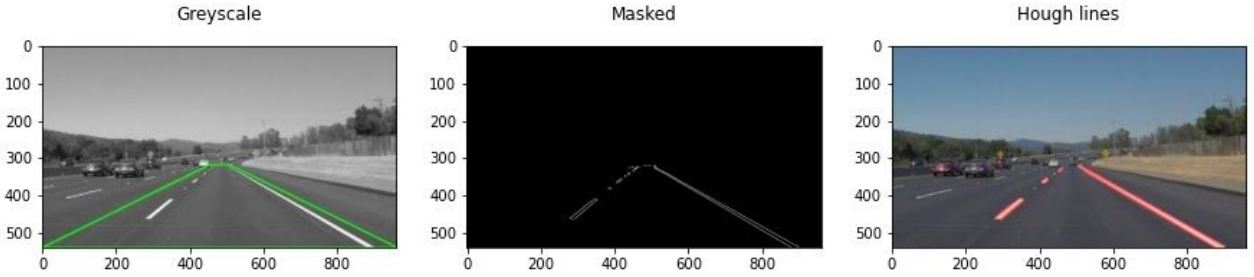
1. The lane detection pipeline

My pipeline consists of 5 steps, which are called by the main routine “lane_detect” in the source code:

1. Conversion of the image into grayscale
2. Gaussian smoothing
3. Canny edge detection and area of interest mask
4. Hough transform and line identification
5. Filtering of detected lines and extrapolation

Generally, I converted the images into grayscale (only one 0-255 value per pixel instead of three 0-255 values) to simplify the given problem, as we are specifically looking for high contrast edges to be detected. With the detected edges I mask out irrelevant regions in the image by defining a trapezoid that encloses the road incl. the lanes. Furthermore I employ Gaussian smoothing to smoothen out artifacts and unintentional edges. Next, I used hough transform on the Canny edge detected black/white image to detect lines using a set of parameters. These parameters were tuned so that it detects a significant share (subjectively >80%) of the lane lines in the image.

Before I started with merging and extrapolating the lines, I first tuned the parameters for (2) Gaussian smoothing, (3) canny edge detection incl. the mask as well as for (4) the hough transform. The output for a selected example is is shown below (optimized parameters are highlighted in the code):

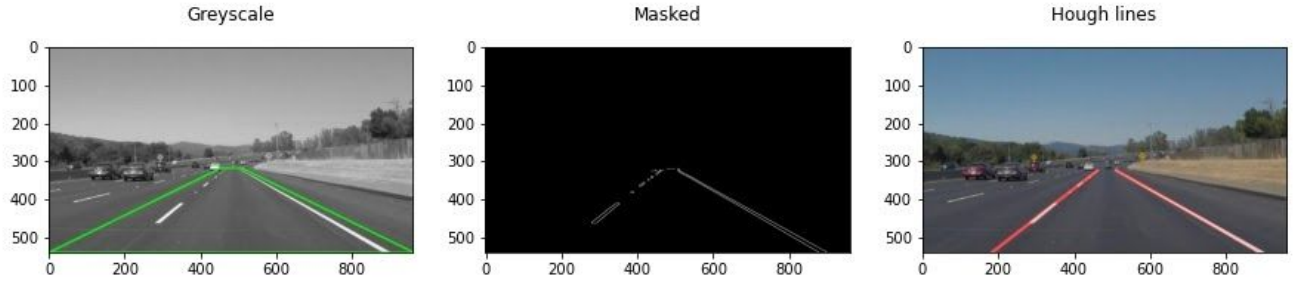


Example: Greyscale, masked Canny detected edges and lines drawn on the original image (code outputs this comparison for all included example images)

For the final step (5), the filtering of detected lines and extrapolation, the goal is to draw a single unambiguous line for the left and right lane. To do this I modified the given function draw_lines() by separating the detected lines by their slope. Lines we would attribute to the left lane show a positive slope (from bottom left to top right), while the right lane is characterized by a negative slope (top left to bottom right). However, to specifically detect lines that correspond to lane lines as we would expect them regarding the angle of the camera and the typical shape from all the example images, I further

filter lines by their slope. Specifically all lane lines that are close to horizontal or close to vertical are omitted, as they are unlikely to represent lane lines.

With a list of lines that are separated into left lane and right lane I average out their position and slope by adding up all the x and y endpoints of the lines and dividing them by their total number. (Notice: the openCV function HoughLinesP() outputs lines in “left to right” order, which means that we have to switch the positions of our (x1, y1) and (x2, y2) values in the list to correctly calculate their slope and average out their positions.) With the averaged lines I further continue with extrapolating them to the bottom of the image and a fixed position towards the top of the image along the y-axis. This results in the following output using the same input image and parameters for Canny filtering and the Hough transform:



Example: Comparison of greyscale, Canny detected image and extrapolated as well as averaged lines on the original image

With solid results for all the included images, this pipeline is then used for detecting lanes in videos containing typical highway situations. As a challenge a video is provided that shows a car driving in a curve and furthermore crosses a bridge with near-horizontal lines to challenge the robustness of the developed pipeline. Subjectively, the pipeline shows a decent performance on the challenge video and robustly detects lane lines with the given parameters and filtering.

2. Shortcomings

Despite the robust detection of lane lines in the images and videos, there are potential shortcomings I would like to address.

Firstly, the pipeline is optimized for the given set of images and videos. A broader robustness for various traffic situations with correct detection lane lines would require a much higher amount of input material. With a higher amount of input material, manual tuning of parameters for optimizing lane detection might become a cumbersome task. In this case, turning this task into a supervised machine learning problem could help increasing the number of training examples and making lane detection more robust.

Secondly, for simplifying the problem I converted the image into grayscale, while omitting a significant amount of information in the image. This might provide sufficient information to detect white and yellow lane lines for bright situations, however, at night, additional color information could be highly useful to further enhance sensitivity and robustness.

An additional shortcoming could be if the situation in which lane lines are to be detected includes a sharp curve. In this case, assuming that the lane lines on the left and right can be represented by a single straight line each appears far-fetched. In this case a separation of two or three stacked straight lines along the y-axis could help to (a) correctly identify the lane lines and (b) to gain information on the curve based on the relative slopes of the detected lines..

3. Areas for improvements and further ideas

Potential improvements to the pipeline would most likely address the shortcomings in the section above. These would encompass, but would not be limited to: (a) including a higher variety of images and videos to tune parameters, also using machine learning techniques to handle large quantities of images/videos; (b) including color information and processing it in the pipeline e.g., for enhancing robustness for currently ambiguous detection scenarios; (c) splitting the left and right line into two or three separate lines that could better detect curves and provide information on that for the downstream pipeline/process.

Another improvement could be to use hardware acceleration for image processing and encoding tasks. This is currently done by the cpu which is not fully optimized for this task and, hence, is slower.