Finding Lane Lines on the Road (P1)

Goals

The goals / steps of this project are the following:
Make a pipeline that finds lane lines on the road
Reflect on your work in a written report

Reflection

1. Description of the pipeline

The pipeline consists of 6 processing steps which are called are called in the processing pipeline *process_lane_detection_pipeline()* function. Instead of modifying the *draw_lines()* function a new function called *determine_left_right_lane_lines*() has been introduced. The list below gives an overview about all processing steps. All relevant steps are explained in detail in the following chapters.

1. Convert image to grayscale
2. Add Gaussian blur to the image with a kernel size of 5
3. Apply canny edge detector with a low threshold of 70 and high threshold of 100
4. Mask everything in the canny edge image out which is out of the region of interest (ROI)
5. Apply Hough transformation to ROI in canny edge image with threshold of 40, a min line length of 5 and a max line length gap of 10
6. Determine left and right lane line based on the slopes of Hough lines

1.1 Canny Edge Filter (step 3)

The third step applies the canny algorithm to detect lane marker edges in the blurred grayscale image from step 2. The filter is parametrized with a low hysteresis threshold of 70 and a high hysteresis threshold of 100. The following image shows the output of the third processing step.

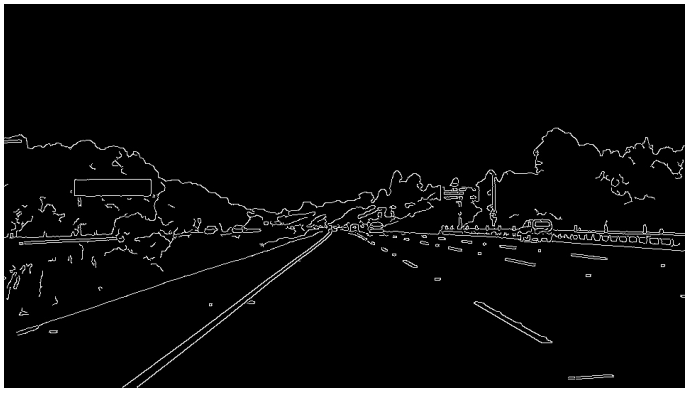

Image: Canny edge image (low threshold = 70 and high threshold = 100)

1.2 Hough Transformation (step 5)

In step 5 the Hough transformation is applied on the edge image masked by the region of interest (ROI) in order to find all lines segments. The Hough algorithm is parametrized with 1 pixel distance resolution, 1° angle resolution, an accumulator threshold of 40 (lines with less votes are rejected), a min line length of 10 pixels and a max line gap of 10 pixels. The image below shows the result of this processing step as overlay on the canny edge image.

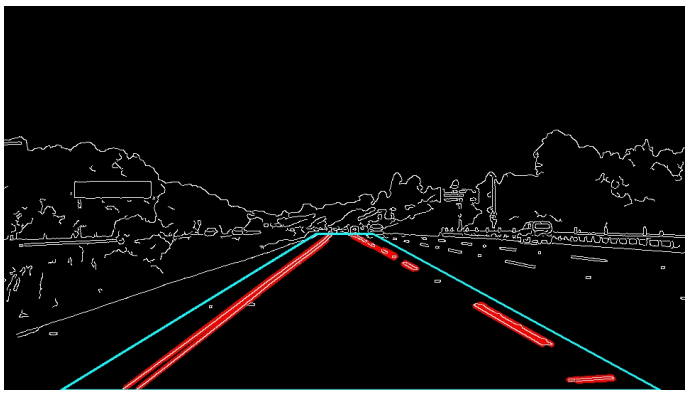

Image: Determined Hough lines (red) and used ROI (cyan) drawn in the canny edge image

1.3 Determination of left and right lane line (step 6)

The last step determines the left and right lane lines based on the Hough lines of the previous processing step. In order to filter the relevant Hough lines and to separate them into left and right lines, the slope for each line is calculated (depicted in the top left diagram in the image below).

Calculation of the slope: $m = \frac{y_1 - y_2}{x_1 - x_2}$

Hough lines with slopes <= -0.5 are belonging to the left lane line and Hough lines with slopes >= 0.5 are belonging to right lane line. All other Hough lines are clutter and rejected. The bottom plot in the figure below shows the filtered start and end points of the Hough lines as red and blue crosses. Red crosses are belonging to the left lane line and blue ones to the right lane line.

In order to extrapolate the lane lines, a 1 degree polynomial fit through all Hough line start and end points was implemented. The polynomial fit returns the coefficients m and b of the linear equation.

Linear equation: $y = mx + b$

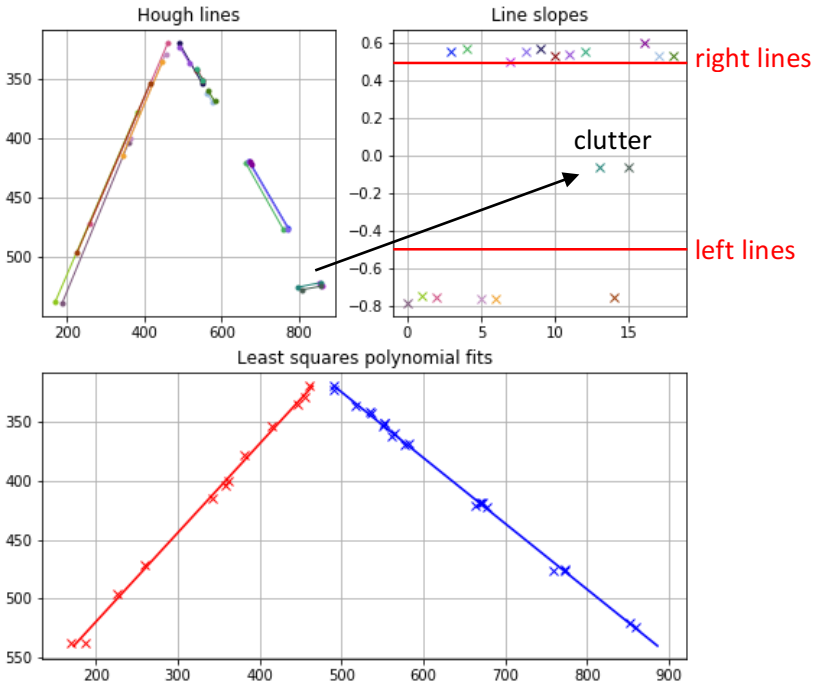

Image: top left: Hough lines, top right: slopes of the Hough lines, bottom: determined left/right lane line

Results

The image below shows the extrapolated lane lines as overlay in the original image. To debug the pipeline additionally some internal data like Hough start/end points and the applied ROI is drawn.

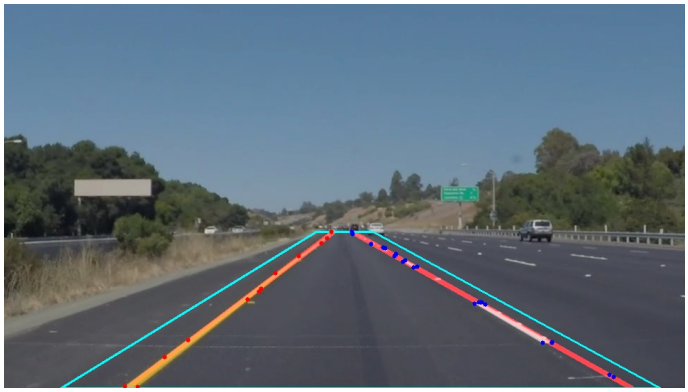

Image: Processing pipeline results (red: lane lines, cyan: ROI, dots: filtered Hough start/end points)

2. Potential shortcomings in the pipeline

One potential shortcoming would be that the line model works well on straight lanes but does not model curves accurately. Furthermore, the pipeline is prone to shadows on the road because they generate quite a lot of edges in the canny edge image leading to problems in the Hough transformation.

3. Suggest possible improvements

A possible improvement would be to eliminate the “edge clutter” by a better image preprocessing. One approach would be to exchange the canny edge detector by a row scanning algorithm which scans for a positive and negative gradient around a lane marker (color change dark to bright and bright to dark).
Another potential improvement could be to apply the canny edge filter on all RGB channels to improve the detection rate of yellow lines especially on bright concrete.
In order to improve the general lane model stability and accuracy in curves, the implementation of a Kalman filter in combination with a 3rd order polynomial fit would be useful.