Finding Lane Lines on the Road

Overview

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

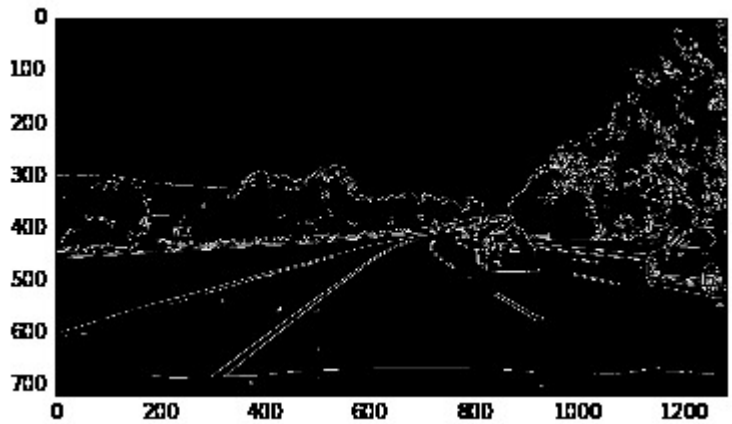
Reflection

1. My Pipeline for Lane Lines Detection

- Grayscale conversion
  First the color image is converted to grayscale for further processing.

- Gaussian smoothing
  The grayscale image is then smoothed using a gaussian filter, this is supposed to benefit the canny edge detection step.

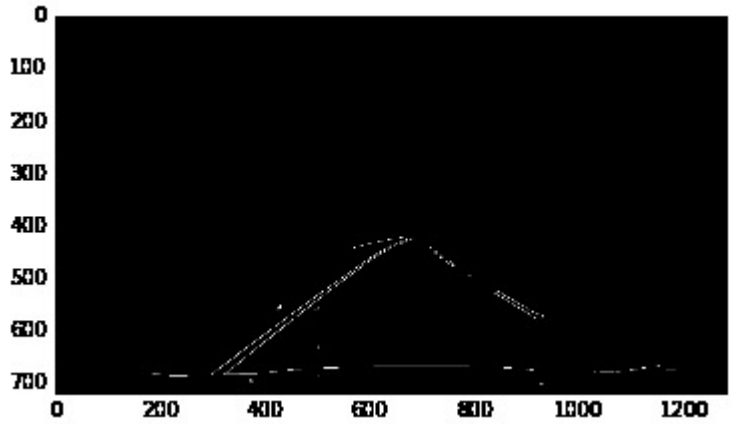
- Edge detection
  Then edges in the image are detected using the canny edge detector with thresholds that proofed to be robust for the lane detection task in previous lessons.



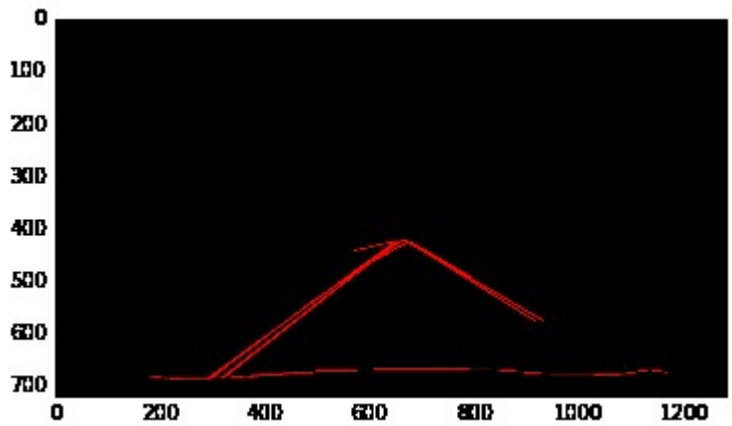
- ROI extraction
  Then a region of interest is cut out from the detected edges image. The points I used are

1. Left Bottom: The bottom of the image with a bit of offset from the left side.
2. Left Top: The x-Value is the center of the image with an offset to the left

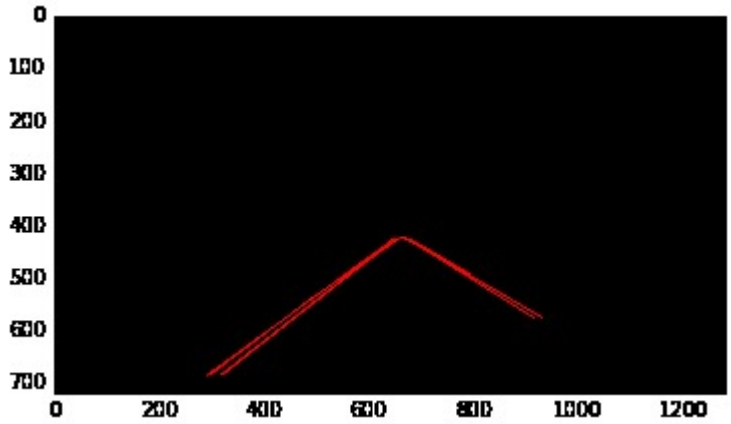
relative to the image shape, the y-Value is below the center of the image to avoid line crossings.
3. Right Top: Same as Left Top but with an offset to the right.
4. Right Bottom: Same as Left Bottom but with an offset from the right side.



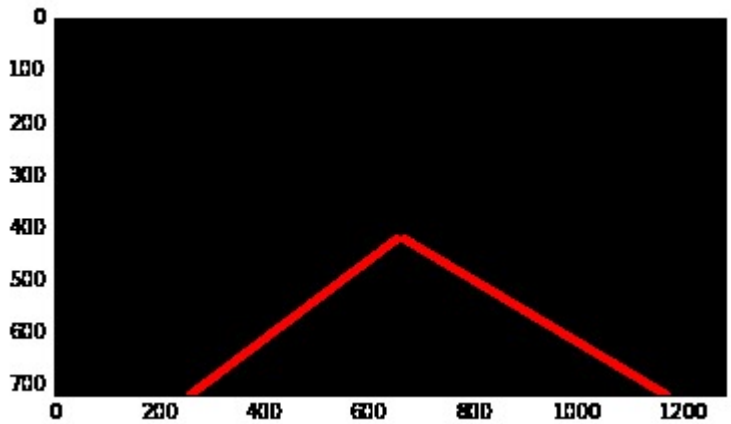
- Hough Transformation
  Then Hough Lines are generated with values that proofed to be robust to the task.



- Slope Rejection
  The hough transformation detects all lines in the image regardless their slope, so also lines which have a more horizontal shape. Those lines are likely outliers and not lines we need. These lines could lead to a worse performance when trying to estimate the single left and right line. Therefore the function reject_lines removes the lines which don't have a certain slope. A slope threshold of the image diagonal was chosen (positive for the left lane, negative for the right lane).



- Line Fitting
  The resulting lines are then used by the function fit_lines, which applies the opencv fitLine function that tries to fit a single line to a set of points. The output is the slope and height of the line functions, once for a line with positive slope and once for a line with negative slope. The start and end point of the lines are then generated using the same top left and right height as for the ROI.



- Robustness improvement for videos
  If no line was found, the last known line is used instead. This proofed useful and necessary for the "challenge" video with a curved track.

- Overlaying lines and image
  In the last step, the original input image is overlayed with the detected lines.

2. Potential Shortcomings

1. The Hough transformation for lines is not a good approach for curved lines, this is especially visible in the "challenge" video. Sometimes lines are not detected.

2. Sudden jumps of the lines are not taken care of at the moment.

3. The approach with taking the last valid line when no line is detected, could lead to a static and wrong line, when no line is detected for a longer time. The temporal aspect is not taken into account at the moment.

4. Lane crossing would lead to a loss of lines, because the ROI removes valuable information.

3. Possible Improvements

Possible improvements in relation to the shortcomings:

1. Try fitting a polynom instead of a line could benefit the detection in curved situations.

2. Average the lines between frames and detect and reject outliers to avoid line jumps.

3. If the last valid line is more than x frames old, avoid returning a line at all or adapt thresholds/ROIs in order to detect at least something.

4. Avoid using a ROI or adapt the ROI on the fly for each situation, e.g. try to keep the ROI centered between detected lines.