Finding Lane Lines on the Road

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. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consisted of 6 steps.

1. I converted the images to grayscale. This step reduces the dimensionality of the image and make the following transformations much easier.

2. With the Gaussian Blur the gradients of the image are smoothened. This prepares the image for the next step. This step reduces the “noise” in the image. The size of the kernel represents the smoothness of the outcome.

3. The Canny transformation of the gray scale image detects edges. The result is, that the gradients of an image are shown, which represents the shape of an object. With the help of the gaussian blur, single pixels are smoothened out, so that more likely the edges are detected. The lower and upper threshold determines the strength of the gradient to be detected.

4. The previous steps are applied to the whole image. Since only the lane in front of the vehicle should be detected, a region of interest had to be set. This region makes sure, that only the relevant part of the image is processed.

5. The Hough filter get lines out of the previous image. The Canny transformation returns dots, which will be transformed to the Hough space. With different parameters, the relevant lines from the Hough space are sorted out and returned as lines.

6. The last step combines the image with the lane information with the original image. This is the result of our effort and enables manual checks, how good the lane detection works. There is one line on the left and one on the right.

I modified the draw_lines() function in two different ways. The first steps are the same for both versions. I calculate the slope for each line in the np.array(). Since it's quite hard to extend an existing array withe the slope:

- I created an empty array and added both points and the slope to it. Then I differentiate between left and right lane by the calculated slope. A positive slope describes the left lane line and a negative slope describes the right lane line.

For version 1, calculate the line be the **average slope between coordinates**:

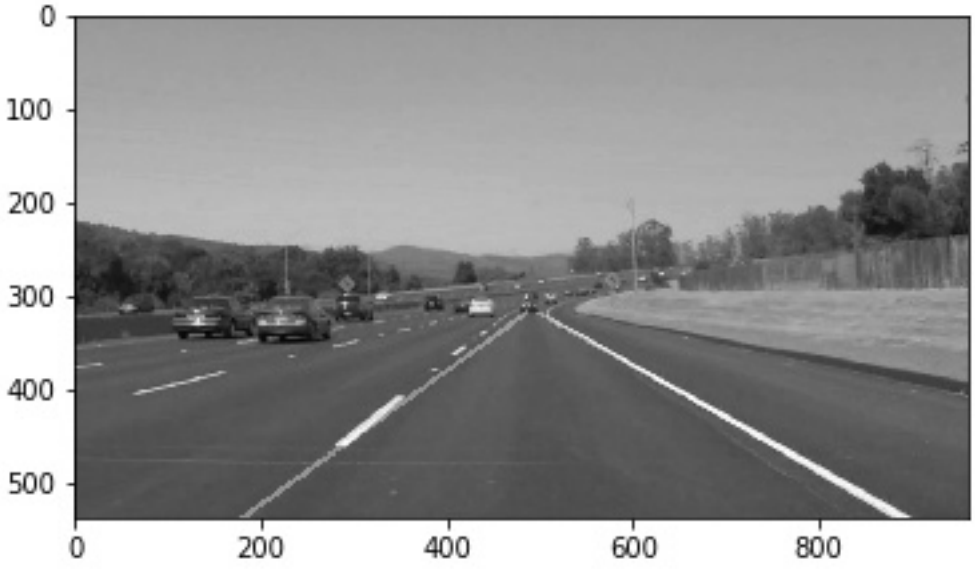
- I calculate the average slope for each lane
- Determine the minimum and maximum x,y-coordinates for each lane (y_min and y_max is defined by the image and the region of interest)
- Calculate the unknown x values for y_min and y_max
- Return the two points, which define the left and right lane line

For version 2, **fitting a linear regression** for the points:

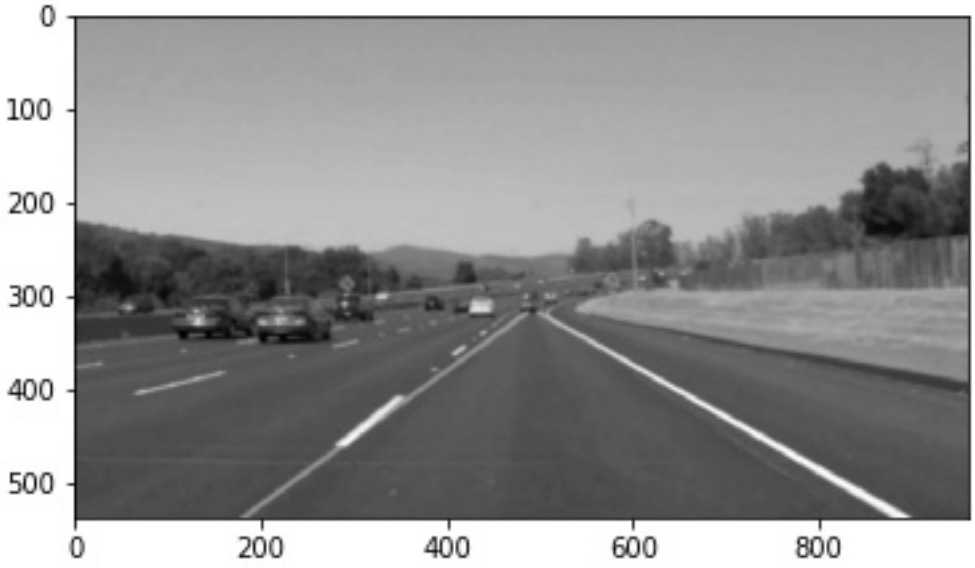
- Calculate the slope and intercept for the left and right lane by a linear regression
- Calculate the unknown x values for y_min and y_max
- Return the two points, which define the left and right lane line

I included the image after every step:

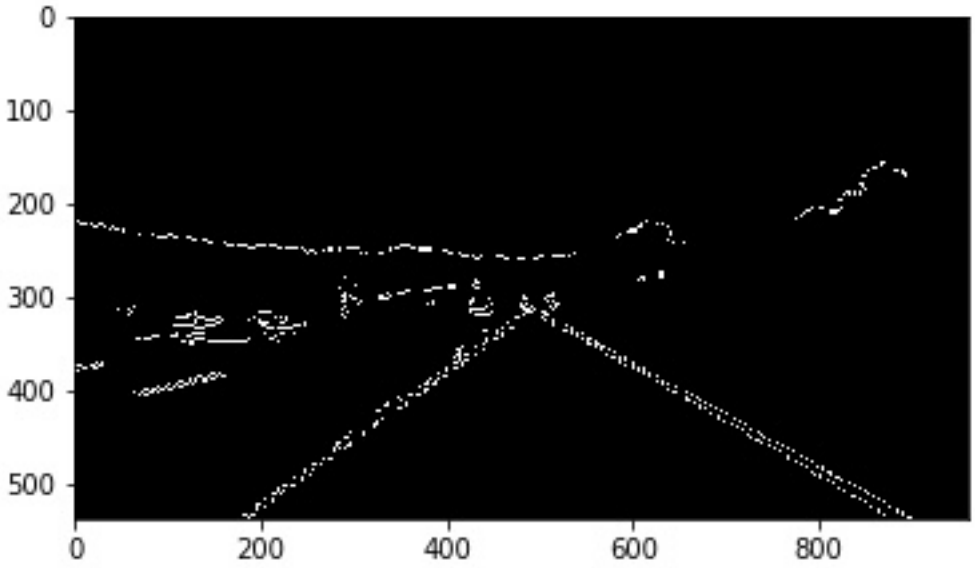
Step 1: Conversion to gray scale



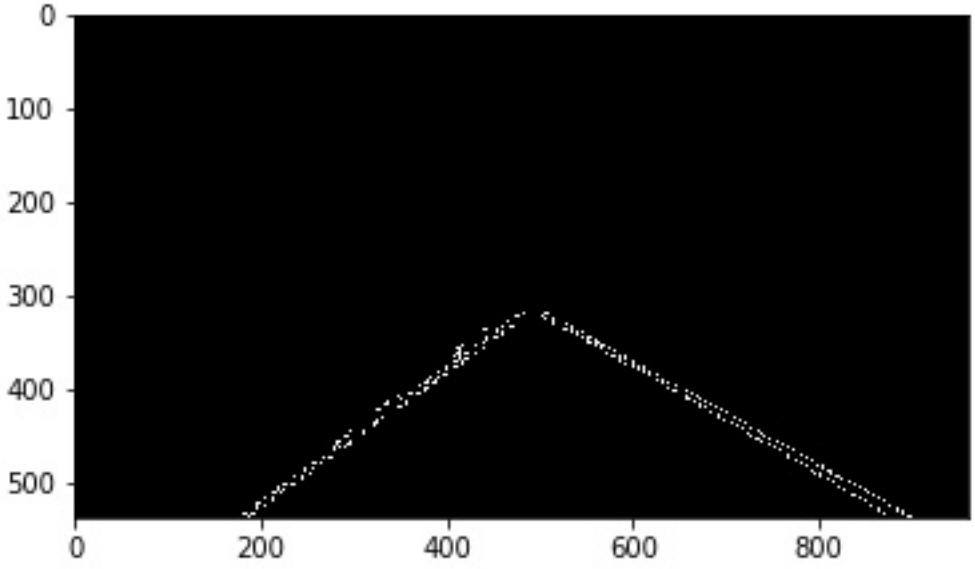
Step 2: Adding gaussian blur



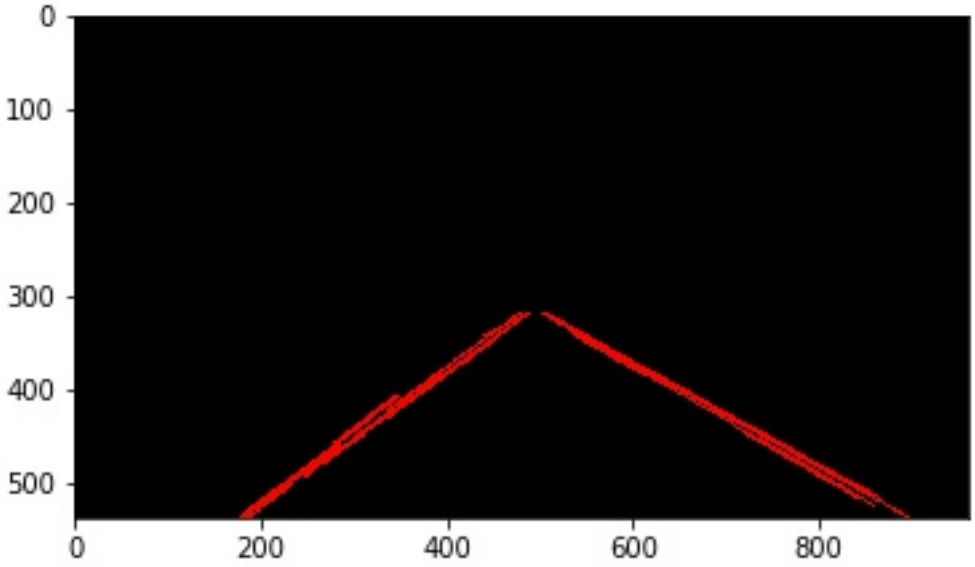
Step 3: Canny transformation



Step 4: Region of interest



Step 5: Hough transformation



Step 6: Combination of initial image and Hough transformation

2. Identify potential shortcomings with your current pipeline

One potential shortcoming is that the region of interest, which I use for the lane detection is fixed. This means, that the lane detection fails for curves and during the change of lanes. Also, this fixed region doesn't allow to detect other lanes.

This approach only works, when there are lanes like on a highway. A typical street inside a city, which is defined by the curbs, this approach can't work, since the gradient between curb and street is too low (won't be detected or there will be too much noise, if the pipeline is applied). Also, if the weather is rainy or cloudy, the gradients wouldn't be strong enough for the detection.

In general, this approach is very static. If something unforeseen event occur, it will definitely fail. This static approach needs probably adjusted for each car and operation area.

3. Suggest possible improvements to your pipeline

A possible improvement would be a polynomial curve fitting, which enables the detection of curves. If I will stay with this approach, I had to get finer adjustments of the parameters for each step inside the pipeline.

Probably the big improvement would be a more dynamical approach, which doesn't need this strict adjustment. I don't know much about it, but I will learn the explicit usecase for deep learning in this case.