Finding Lane Lines on the Road

The goals of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on my work to identify shortcomings in my current pipeline and suggest possible improvements

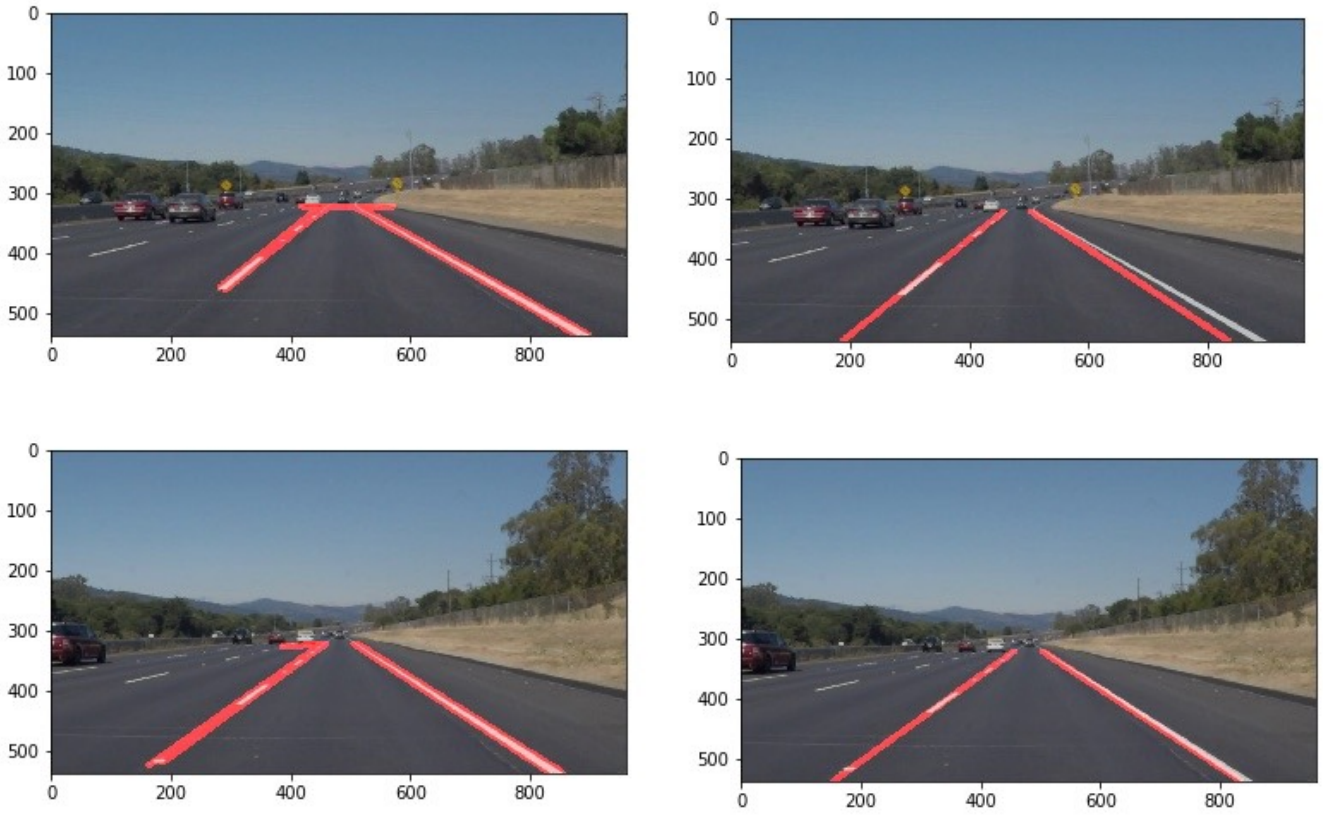
Hence, this report consists of two parts addressing these issues.

I. Lane Finding Pipeline

The lane finding **pipeline** consists of following steps as covered in this course:

1. Turn the RGB image into a grayscale image
2. Smooth the grayscale image via Gaussian blur
3. Detect edges in the grayscale image via Canny transform
4. Select the region of interest from the image generated in step 3
5. Detect straight line via Hough Transform
6. Calculate the **average** lines for left and right lane lines
7. Draw the average lines on the original image

Result. Before we introduce some nice techniques used in this pipeline, let's first take a look at the results of our processing pipelines on images. **On the left are the processed images without step 6. On the right are the processed images of the whole pipeline.**

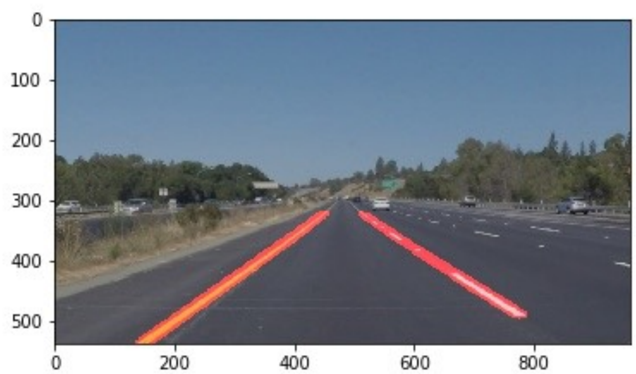



0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800


0
100
200
300
400
500
0
200
400
600
800

Here, we only discuss the **mechanisms used in this project that are not covered during the course** (we won't reiterate things like the parameter selection and the function of each algorithm).

- **Region of Interest Selection.** Since the camera is fixed on the car, the lane lines in the image are most often at the fixed position. We capture the region via a 4-point polygon defined by 4 vertices: (0.05*width, height), (0.4*width, 0.6*height), (0.6*width, 0.6*height), (0.95*width, height)[1].

- **Extrapolating Lane Lines**. To achieve this, we follow steps below:

    1. Express all straight line segment in a **slope-intercept form**
    2. Group line segments **into two groups**: one group with positive slope, and the other with negative slope, each group can contribute to our extrapolation of one lane line.
    3. We calculate the weighted **average slope and intercept** in each group — weight of each line segment is the length of the segment. Then we have two slope-intercept forms, with each being the estimation for one lane line
    4. We calculate the **intercepted points** of each lane line with y = 0.6*height and y = height.
    5. We draw the two lane lines on the image

- **Fill missing lane lines in videos.** Sometimes, we don't have line segments detected for some frames in videos. In this case, **we just use the lane lines detected in previous frame for the current frame.**

Summary, our lane line detection pipeline generally detect the lane lines well for the given tested input images and videos.

II. Reflection

Shortcomings of this project include the following:

- **It doesn't work with the the curved lane lines.**
- **It doesn't work at the moment of the car changing lanes,** considering our region of interest selection might not cover the lane lines at the edge of the image.
- **It doesn't work if the car's direction is perpendicular to the lane lines.**
- **It may not find the best possible parameter combinations**. We just select the parameters based on experience.

Some **suggestions** might help**:**

- **Selecting yellow and white pixels might help improve.**
- **More advanced line fitting algorithm is indeed for cured lanes**
- **Different algorithm/strategies need to be selected at runtime in accordance to the vehicle's state, like whether the vehicle is change lane lines.**
- **Certain parameter optimization is in need.**

---

[1] Note notation width and height constitute the dimension of the image.