Detecting Lane Lines on the Road

Udacity Self-Driving Car Term 1 - Project 1

Keeping the car in the correct lane is one of the key adaptive features of the Advanced Driver Assistance Systems (ADAS).

Detecting Lane Lines on the Road

The **goals** of this project are the following:

1. Build a image/video processing pipeline that finds lane lines on the road
2. The pipeline for line identification takes road images from a video as input, and return an annotated video stream as output
3. The left and right lane lines are accurately annotated by solid lines throughout most of the video
4. All required code and files are uploaded onto GitHub for submission
5. A written report summarizes my reflection, weakness and potential improvement

The **steps** of this project are as follows:

1. Transform RGB to grayscale using cv2.cvtColor()
2. Apply Gaussian smoothing using cv2.GaussianBlur()
3. Perform Canny Edge Detection using cv2.Canny()
4. Create a masked edges image with defined region of interest using cv2.fillPoly() & cv2.bitwise_and()
5. Retrieve Hough lines using cv2.HoughLinesP() which will draw lines to the original image using cv2.line() based on the input to extrapolate them or not
6. Draw the lines on the edge image using cv2.addWeighted()

Reflection

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

My pipeline is very simple, it is consisted of 6 steps which are shown as above. It is applied from what I've learned from the classes before Project 1.

Here is what my approach on how I implemented the draw_lines() function which is the core of this project:

First of all, per the requirement of the project to draw segmented lines and solid lines based on the lane lines detection, I added an argument "extrapolate" as an input to the draw_lines() function, so that developers are clear about the usage of this API.

Second, I divided the lane lines into two parts based on their slopes, with negative slopes for left lines and positive slopes for right lines. To improve the accuracy of finding the targeted lines, I introduced two thresholds to limit the range of the targeted slopes based on manual calculation with a buffer of slope fluctuation to compensate different orientation of the car while driving on the road. By averaging out the collected slopes and intercepts and the maximal and minimal of the vertical axis for the two lines, I located the four points required to draw the two lines.

Lastly, what's worth mentioning is that when tackling the challenge video, I realized that my draw_lines() function is encountering Division by Zero error. Through debugging and trials and errors with different Hough Transform parameters, I figured that was likely caused by the shadow of tree shined onto the yellow line which caused different color variation, hence not detected by the cv2.HoughLinesP(). By weighing the cost and risk of delaying the project, I decided to add a very small 10^(-6) to the average slope in the denominator to prevent the error of divided-by-zero without sacrificing too much accuracy, and added a bug on my Task List to improve this algorithm later.

In all, I really enjoy working on my first project, it gives me a deeper understanding of the OpenCV framework with Matplotlib and NumPy using Python 3, and refreshes my knowledge of Canny Detection and Gaussian smoothing from digital image processing, as well as Hough Transform from linear algebra.

2. Identify potential shortcomings with your current pipeline

Here are a list of potential shortcomings would be what would happen:

1. All the existing parameters to make the current example images & videos work may not work with different roads or driving direction, e.g. more curvy roads, roads with sharp turns, roads without lane marks, road with false lanes, or reversing the car;
2. Environmental changes, like weather, light, color of the lane lines, road debris or obstacles;
3. Change of physical devices: orientation of the car and other cars on the road, the angle and orientation of the mounted cameras, the sudden failure of any sensors or cameras caused the computer vision system malfunctions, etc.

3. Suggest possible improvements to your pipeline

Here are some possible improvement to uplift the adaptability of my pipeline:

1. Use cv2.inRange() function for color selection, since I did not use this OpenCV function given as a hint, this could be a potential alternative or the HSL/HLS approach mentioned on the forum;
2. Implement a feedback loop system to compensate the weakness of my existing algorithm with historic data from previous frame so that it impersonates like a real human in driving, as human would not stop driving if s/he finds a part of lane line is missing;
3. Combine Deep Learning or other advanced Machine Learning methods with Computer Vision to improve the intelligence of the Self-Driving Car to detect not only the lane lines but also the drive-able area with the consideration of other objects or conditions.