Finding Lane Lines on the Road

Self Driving Car Engineer Nanodegree

Project 1

Alex Snyder

November 2019

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 image processing pipeline contains the following steps:

- **Convert to gray**
  Most image processing algorithms require or work better and faster with grayscale images instead of color images. The canny edge detection step requires a gray image.
- **Apply gaussian blur**
  Blur is applied to smooth out the image before performing canny edge detection. This reduces noise and the number of false edges. This step takes of a kernel size parameter. For my final result I used a kernel size of 3.
- **Canny Edge Detection**
  The canny edge detection algorithm detects edges by calculating image gradients. Fast changes in pixel brightness indicate the presence of an edge. The issue with this algorithm is that it doesn't know what the edges are – just that there are lines there – so if some other object enters the field of view, it will

detect it as an edge. Figure 1 shows the result of the canny algorithm. Note that many edges are detected including the trees on the horizon.

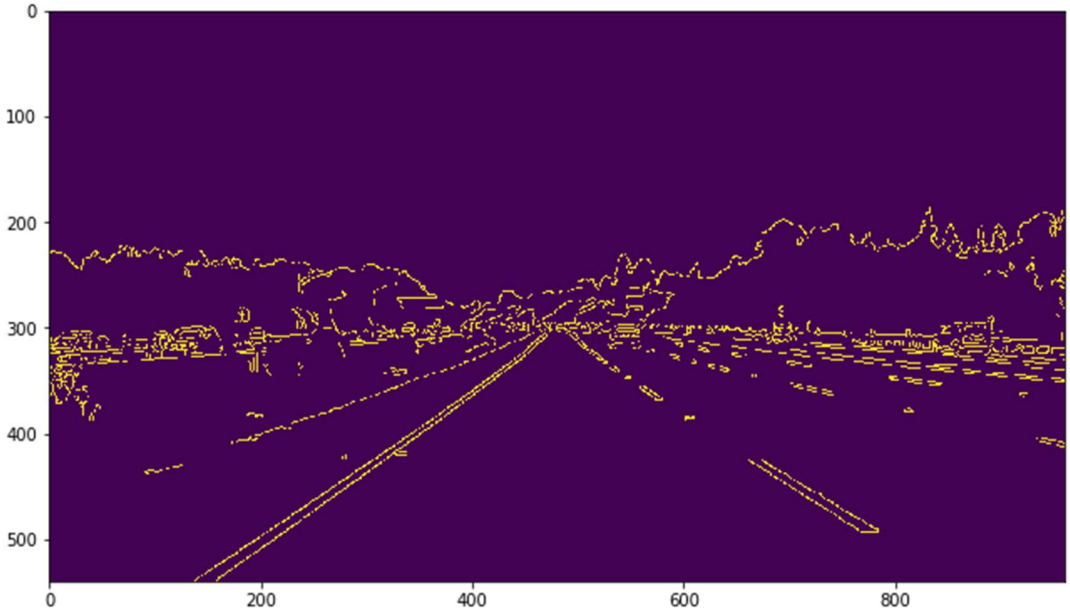

Figure 1 Canny Edges

- **Apply region of interest**
  To aid in the lane detection based on the canny results, a region of interest is applied. This restricts the processing to only a certain area of the frame. The results are shown below in Figure 2

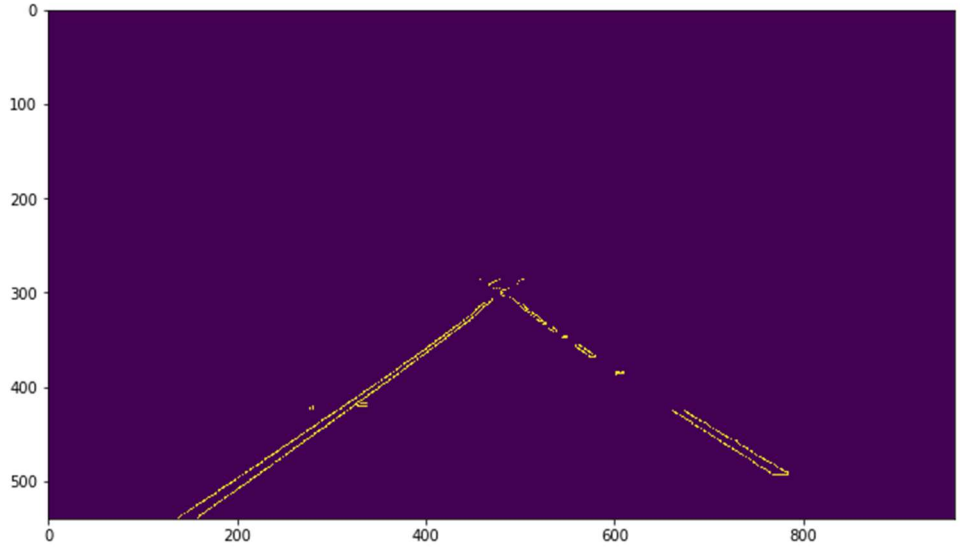

Figure 2 Canny with ROI

- **Hough Transform**

The hough transform takes the pixels in the canny image, and detects lines. The HoughLinesP method from OpenCV is used. It returns a set of lines in x,y coordinates. The results are shown below. Parameters in the hough algorithm include minimum line length, and maximum gap. In the example here, you can see that the lines on the line on the right which was broken up has been joined together.

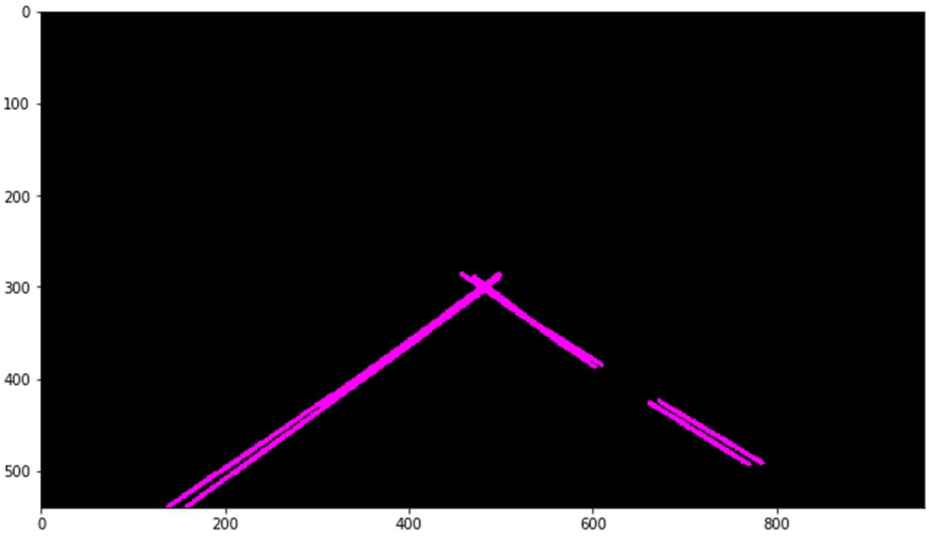

Figure 3 Hough Lines

- **Line Filtering and Regression**

The goal is to produce one lane line on the left and one lane line on the right. The hough transform detects a set of several lines on each side, including the multiple parts of a dashed line, but it also detects both edges of a line, producing two lines per lane. To process the lines, first a filter was applied to reject any lines that are too flat, as the lines are expected to be near vertical. Next, the angle of each line was calculated, and the lines were separated into a set of left lines and right lines, based on the slope angle.

A linear regression algorithm was applied to each batch of lines, using the start and end coordinates of those lines. The output of the regression was the slope and intercept of a line equation. The line equation was then queried at full frame height (y=540) and at the y value of the top of the region of interest.

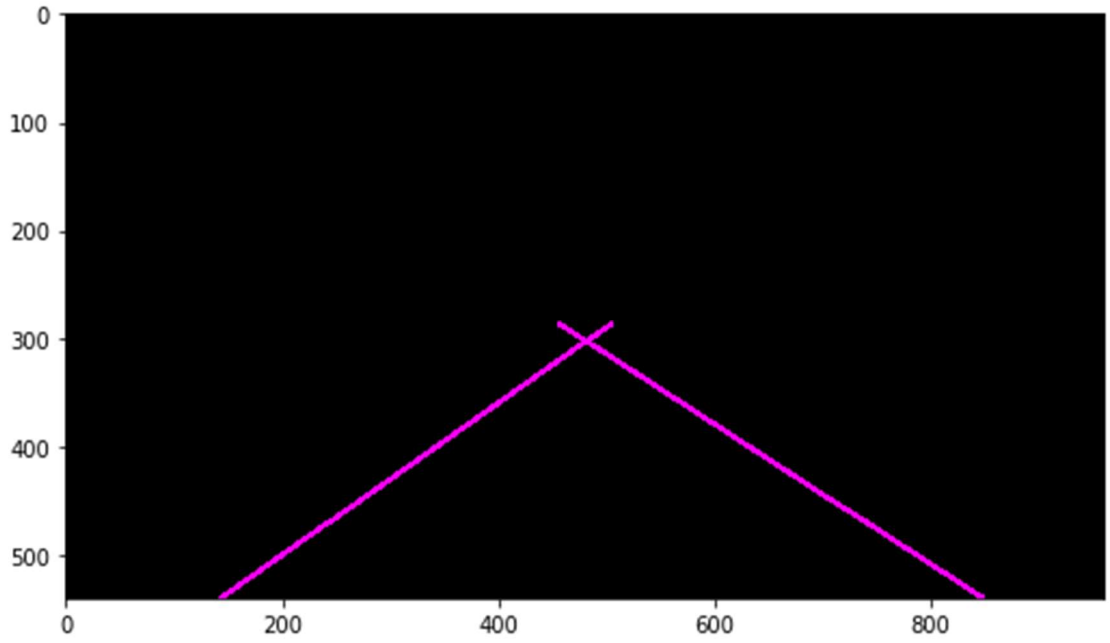

Figure 4 Filtering and Regression Applied

2. Identify potential shortcomings with your current pipeline

Shortcomings with this style of processing include the reliance of tuned parameters. I created a dictionary of parameters in order to easily access and tune them. In a real world situation, weather, lighting, road conditions will all come into play to require different parameters, which are hard to anticipate.

If the car enters an area with wider lane lines, or with other markings on the road, this algorithm will not work.

3. Suggest possible improvements to your pipeline

In the current pipeline, a trapezoidal region of interest is used, so that only lines detected within the bottom half, towards the middle of the frame are detected. If a line is erroneously detected closer to the car, between the main lines, this will throw off the result. A stricter region of interest could be applied which would only allow processing within the area where lines are expected, and nowhere else. Figure 5 shows this, where only the region in black would be considered, but the region in red is masked out.

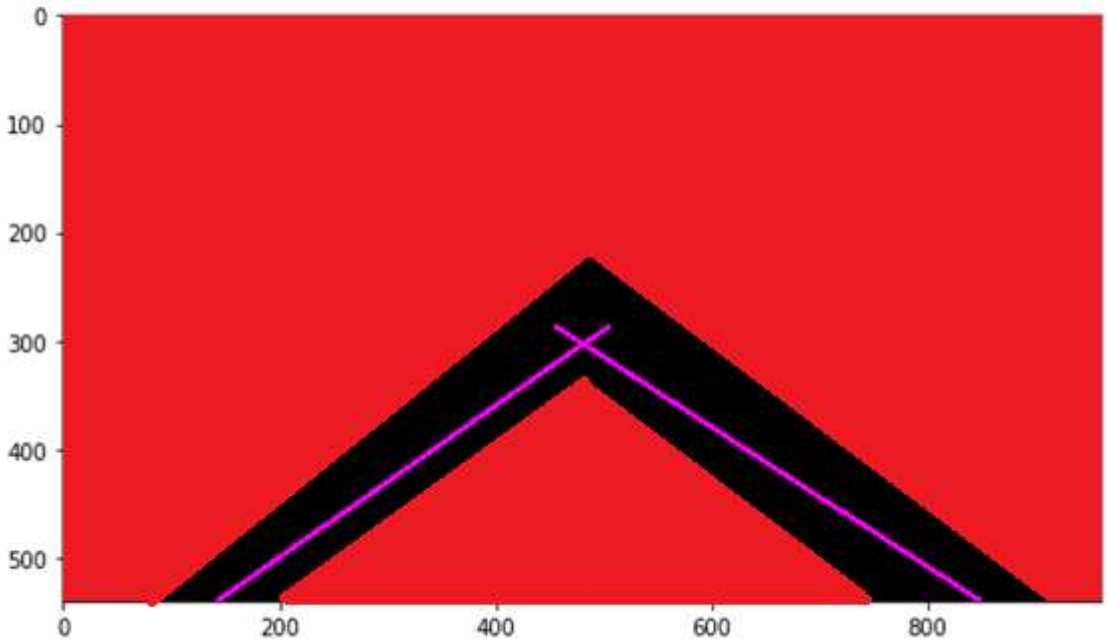

Figure 5 Stricter ROI

An additional improvement to the pipeline would be to consider line colors. Lane lines in the provided video, and in most regions of the world are either yellow or white. To detect only yellow and white, we would convert the color image from the RGB color space to the HSV (hue, saturation, value) color space, and then apply a filter based on the hue – this may work well to detect the yellow lines. To detect white lines, we could look at the value, or lightness if using the HSL color space.