Self-Driving Car Engineer

Nanodegree Program

Project: Finding Lane Lines

Robert Taylor

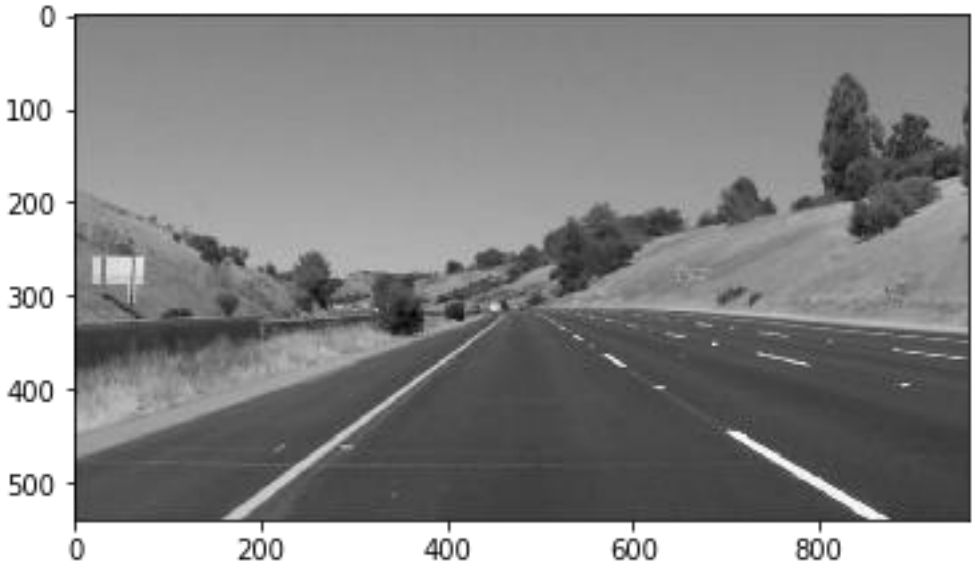
Pipeline Implementation

The object of this project is to take the image of a roadway and to annotate the lane lines from the perspective of the car from which the image came. Iniatially the image is a single frame. From there, a video image is annotated. The orignal single frame image is show below.

Original Image

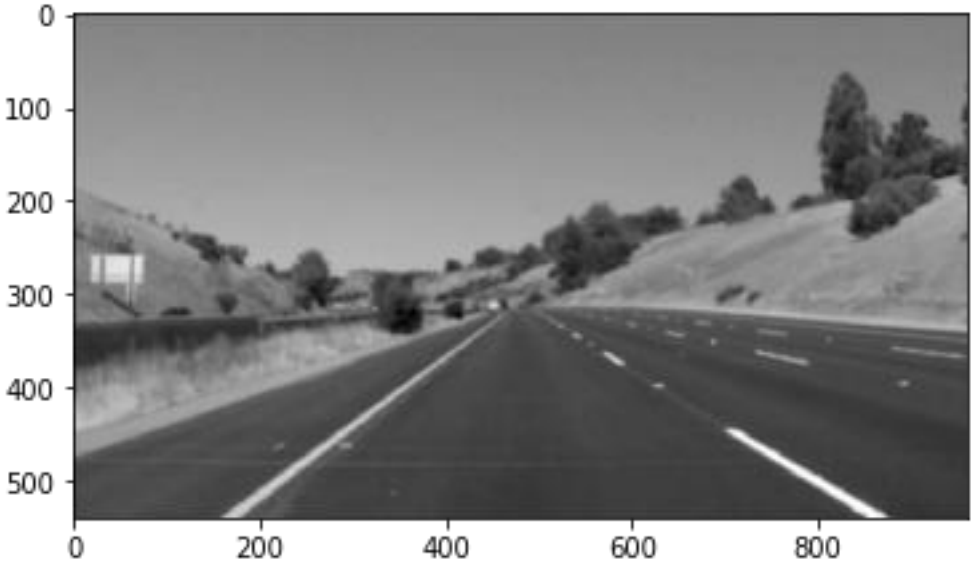
A “pipeline” of image processing techniques are used to achieve this result. My pipeline consists of the steps described below.

The first step is to read in the image and then convert it to grayscale. I used the cv2.cvtColor() function that is called in the helper function grayscale(). This function returns the image shown below, labeled “Grayscale Image.”



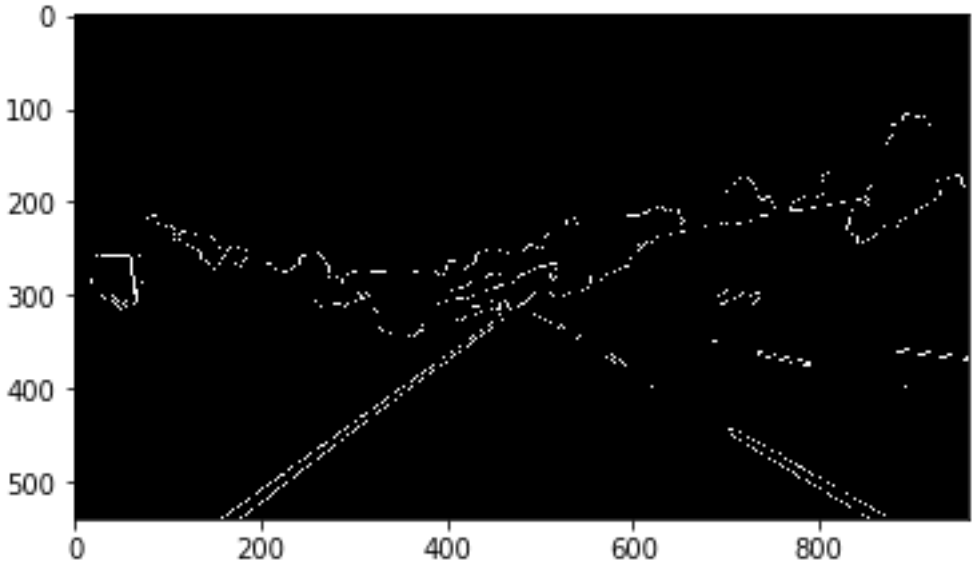
Grayscale Image

The next step is to apply Gaussian smoothing to the grayscale image. This is done using the cv2.Gaussian_Blur() function, which is called by the gaussian_blue() helper function. Gaussian blurring smooths the image to reduce noise, which are details that are not imortant to identifying lane line. The blurred grayscale image is shown below, labeled “Blurred Grayscale Image.”



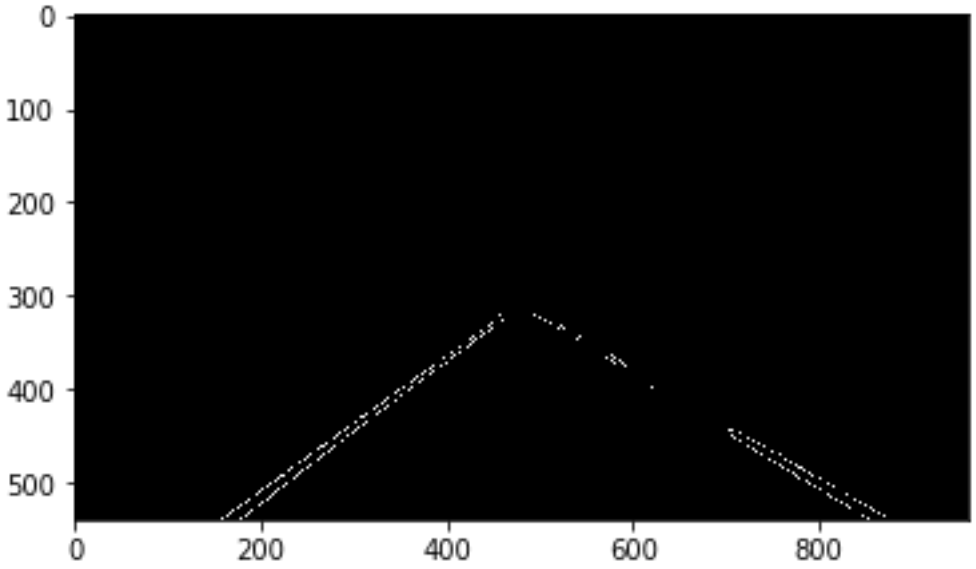
Blurred Grayscale Image

To determine the boundaries of the lane lanes, Canny edge detection is used. This function is called using the cv2.Canny() function, which is included in the helper function canny(). Canny edge detection identifies the pixels that are between high and low contrast areas of the image. The canny function returns an array of pixel locations showing the edges found in the image. The canny function will identify a number of different edges between high and low constrast areas in addition to the lane lines, as seen below in the output image of the canny function, labeled “Canny Image.”



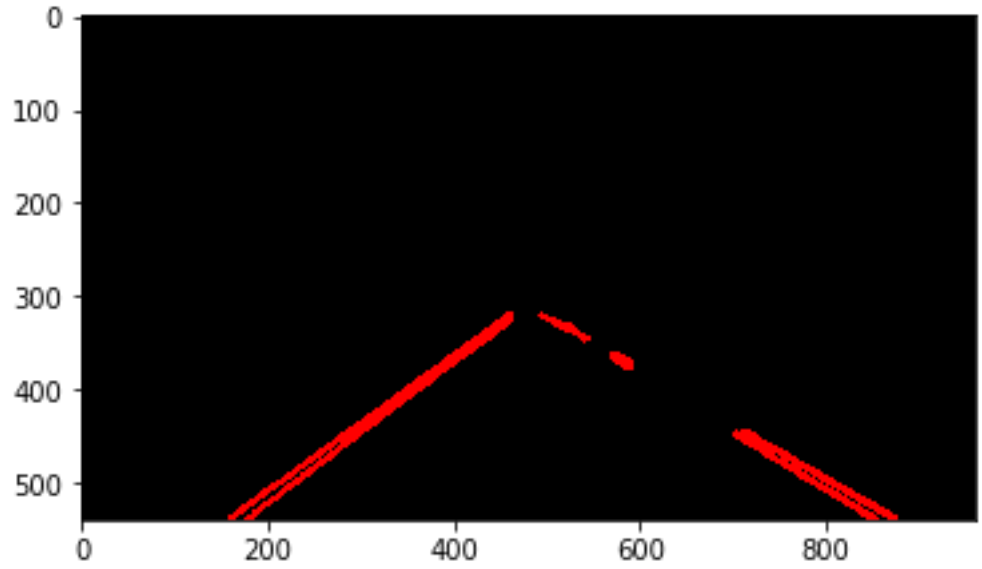
Canny Image

The next step in the pipe line is to black-out all the edges that are not lane lines. This is done by defining a polygon that includes the lane lines and excludes all other information. Pixels outside of the polygon are set to black and pixels inside, which includes only the lane lines, remain white. See the image labled “Masked Canny Image” below.



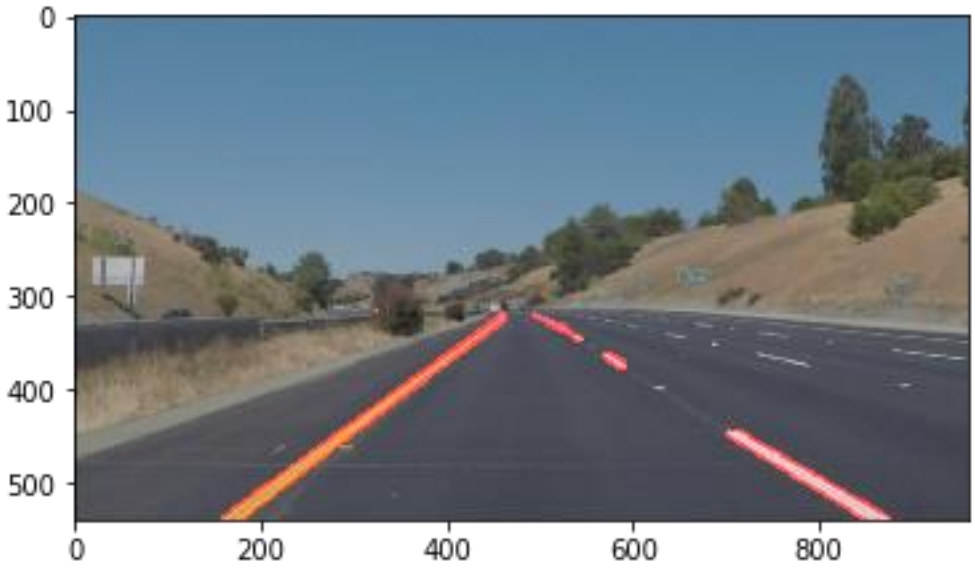
Masked Canny Image

The masked Canny image is an array of pixels identifying the edges of the lane lines. The next step in the pipeline is to convert the array of pixels into a set of lines. This is done using the Hough transform. The Hough transform is applied using the cv2.HoughLinesP() function, which is called in the hough_lines() helper function. This helper function calles the draw_lines() function, which returns an image with the Hough lines drawn on it. This image is shown below, labeled “Hough Lines.”



Hough Lines

The next step in the pipeline is to combine the Hough lines with the original image to create an annotated image, shown be low as “Annotated Original Image.” This is done using the cv2.addWeighted() function, which is called in the helper function weighted_img().



Annotated Original Image

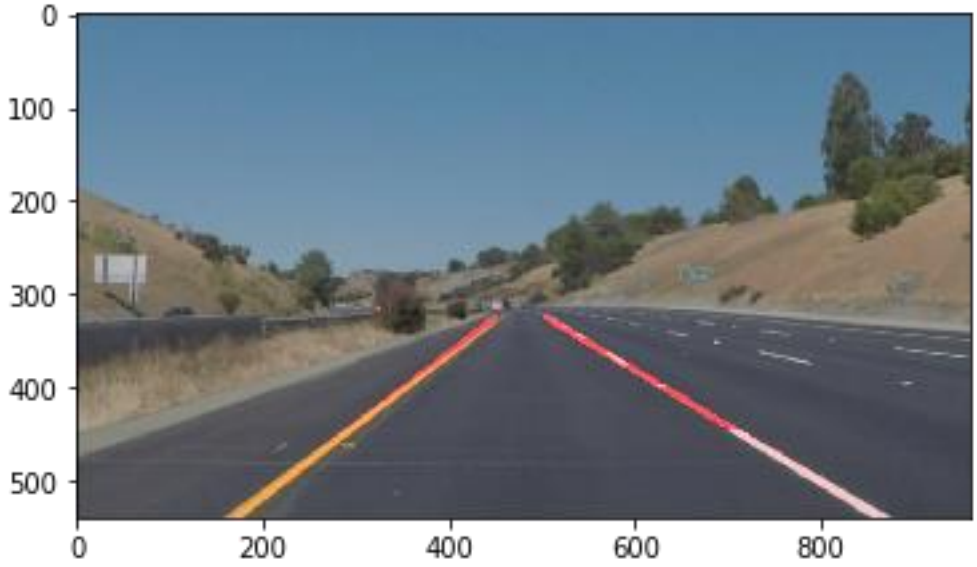
The pipeline described does not provide for continuous lane line markers that extend from the bottom to the top of the region of interest generated above. As can be seen in the image “Houge Lines” above, there are gaps in the lines shown. To generate continuous lane line annotations, it is necessary to modify the draw_lines() function to average and extrapolate the lines to fill in the gaps.

Each Hough lines image is comprised of a set of lines that includes discrete combinations of pixels found for both left and right lane lines. The characteristics of these lines depends on the parametes set for the Hough transformation.

The approach to creating a right side and left side continuous line is to calculate two points for each side that defines a line. The first step in this process is to separate the left side and right side Hough lines. This is done by calculating the slope of each Hough line. The negative slopes are left side lines and the positive slopes are right side lines.

The next step is to average the slopes of the lines for each side. Lines with extreme values are excluded from the average. The average slopes are then used to find the averaged x values for the top and bottom of the region of interest. The bottom and top y values are hard coded to the y limits of the region of interest. The result is two points for each lane line which

are drawn on the original image using the weight_img() helper function. A still image of the result is shown below, labeled “Continuous Lane Line Markers.”



Continuous Lane Line Markers

The modified functions applied to the video streams show continuous lines updating as the video advances.

Reflections

One of the shortcomings of the implemented lane finder is that there is some jitter in the continuous line. This is possibly due to extreme cases where the current frame does not have enough information for the algorithm to calculate an x value and thus uses the previous x value instead. A more sophisticated algorithm may solve this problem.

Also, one of the limitations is for video where the car is rounding a curve. The current implementation does not handle this situation, as demonstrated in the challenge video. A better method of identifying the region of interest may be required and a way to annotate a curved rather than a straight line would be necessary required.