Self-Driving Car Engineer Nanodegree Program

Finding lane lines Project

François Masson
October 14th, 2018

I. Pipeline software

The pipeline used in this project covered the following steps:

- Importation of the images and conversion into gray scale
- Gaussian smoothing/blurring algorithm
- Canny edge algorithm
- Application of a mask edge
- Hough transform
- Combination of the original image and the filters algorithm
- Saving of the resulting images

Importation of the images and conversion into gray scale

The image is import by using imread() function and converted in gray scale with the grayscale() function.

Gaussian smoothing/blurring algorithm

A Gaussian Noise kernel is applied on the gray scale image with a value assigned to 5.

Canny edge algorithm

A Canny transform is proposed with a low threshold of 50 and a high threshold of 150.

Application of a mask edge

The regions of interest were located between the center of the image and the bottom of this one. So a polygon mask with the following parameters was used:
vertices = np.array ([[(0,imshape[0]),(400,350), (600,350), (imshape[1],imshape[0])]]

Hough transform

As a first step, the Hough algorithm was set with the following parameters:

rho = 2, theta = pi/180, threshold = 1, min_line_len = 50, max_line_gap = 30.

In order to determine which segments are part of the left vs right line, the slope ((y2-y1)/(x2-x1)) was calculated. If the slope was negative then it was assumed that we found the left lane. At the opposite if the slope was positive then we found the right lane. Knowing the slope and having points coordinates, it was possible to determine the equation of the line:

$$y = mx + b$$

Then, each of the lines was average then extrapolate to the middle and bottom of the lane.

Combination of the original image and the filters algorithm

The original image and the filters were combined thanks to the addWeighted() function. The combination uses the following equation for that:

$$FInal\ results = initial_{image} * \alpha + filters_{images} * \beta + \gamma$$

Saving of the resulting images

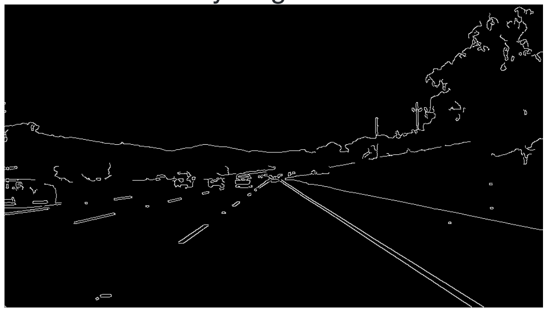
Because the imwrite() function was used to save the images, a cv2.COLOR_RGB2BGR was mandatory.

Below the results of every step on a random image:

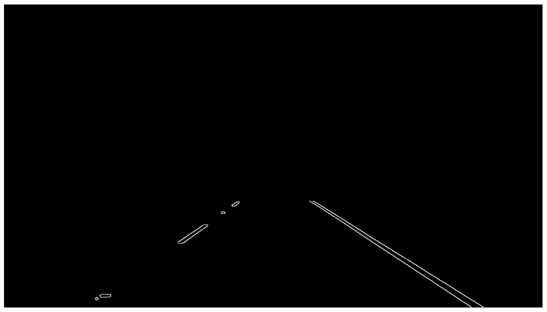
Original Image

Gray scale

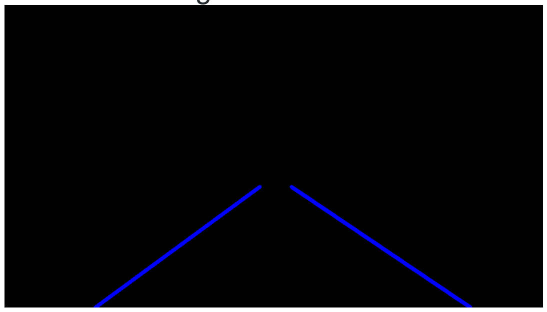
Canny Edge



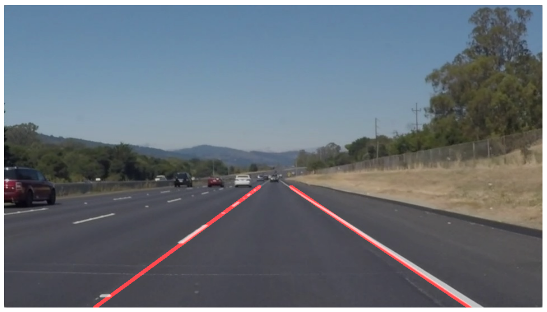
Masked area



Hough transform



Final result



Potential shortcomings

One potential shortcoming would be what would happen when the lines are not visible anymore such as (night, missing lines…)
Another shortcoming could be how the algorithm would react if the car is crossing the lines or the driver decides to change lanes…
And a last shortcoming could be what would happen if the road curves a lot or the car is arriving to a stop sign with a perpendicular line…

Possible improvements

A possible improvement would be to remove the camera distortion of the image and use a camera calibration to undistort images.
Another potential improvement could be to improve the edge detection for curves lines and obtained something similar to a bird eye's view.