Finding Lane Lines on the Road

Writeup Report

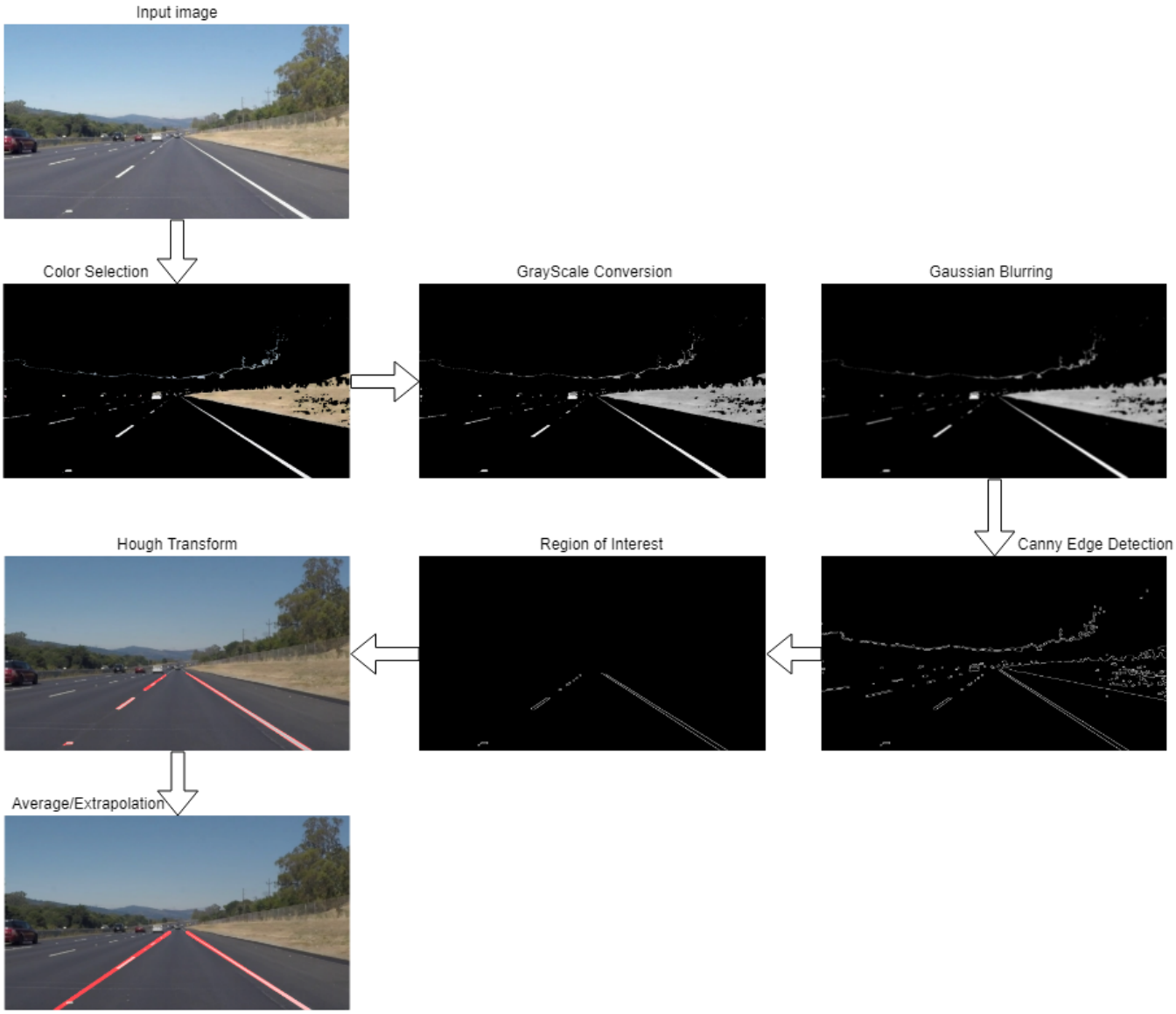
Project Overview

The goal of this project is to make pipeline, using computer vision techniques implemented by OpenCV, which gets an image as the input and finds the lane lines on the road. At the end of project, this pipeline is utilized to draw the lane lines on the video recording of real-world driving situation.

Reflection

1. Lane Line Detection Pipeline Overview

My pipeline consisted of six steps illustrated in the following image.



1. Color Selection

In this stage, since the lane lines are white and yellow in this project, the yellow and white colored pixels are filtered; this is done by defining two masks: one for white range of RGB and another for yellow range of the RGB. The final filter is obtained by performing the bitwise OR operation on these masks. Finally, by performing the bitwise AND operation between the mask and the initial image, the output image is created. The purpose of this stage is to increase the focus of the pipeline on the interesting areas of the image whick is the lane lines. Altough in this project the lane lines are all yellow and white, in the real-world they can be of any color. Therefore, I suppose this stage just suits this project images and videos not the real-life situations.

2. Gray-Scale Coversion

The image is converted to gray scale in this stage, so it can be used as the input to the canny-edge detection stage.

3. Gaussian Blurring

This is done to remove the noises on image which give us strong gradients at the canny edge detection stage. The guassian kernel size is the only parameter here, and the tuning method of this parameter is described in the next stage.

4. Canny Edge Detection

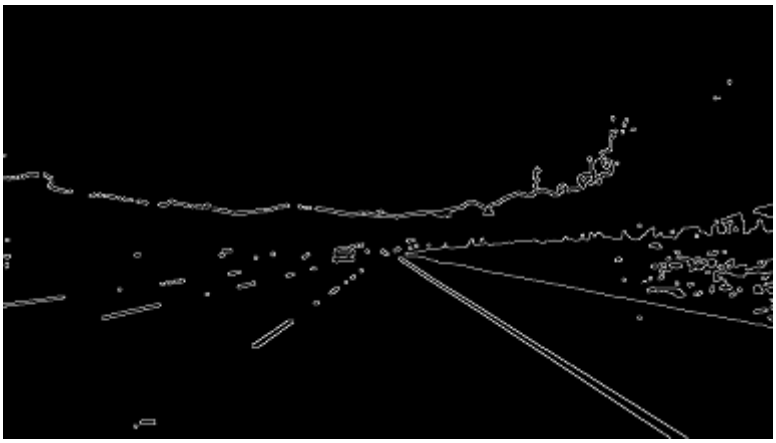
The Canny edge detection method, basically, performs the gradient over the gray-scale image to bring out where the pixels shift color. these edges can be used to distiguish between different objects in the image.

there are three parameters here:

- the low threshold
- high threshold
- the gaussian kernel size from the previous stage.

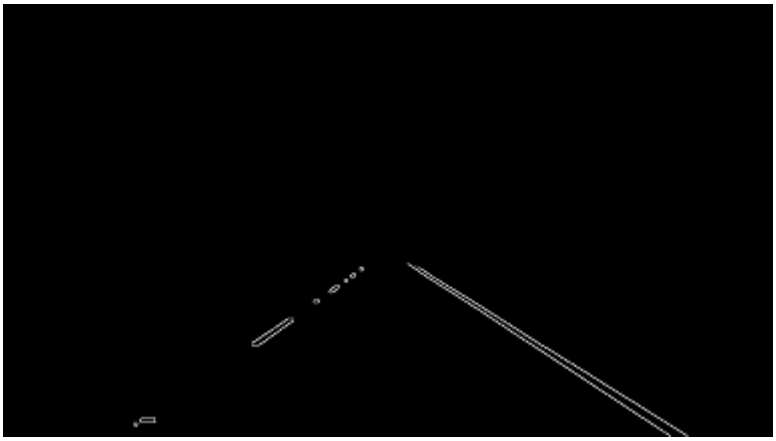
if a pixel after the gradient has a value below the low threshold it is not an edge, if the value is greater than the high threshold, it is an edge, and if the value is between the low and high threshold, it can be an edge if it neighbors an edge.

the GUI helper tool (https://github.com/maunesh/opencv-gui-helper-tool) is used to tune these parameter. I started with the kernel size 3, low threshold 10 , high threshold 50. first, I increased kernel size until all small edges were gone and lane lines were intact. then, i increased the low threshold until the lane lines started to disapear, then i increased the high threshold until i got lane lines clear on the output image.



5. Masking the Region of Interest

To direct the focus to where lane lines are more likely are located, this stage masks the image with a quadrilateral. the quadrilateral's height and top and bottom edges length and position are the parameters here, and they should be expressed with respect to the image size. these are chosen to get the most of the lane lines visible in the images.



6. Probabilistic Hough Transform

Edges points from the Canny edge detection are fed into the Probabilistic Hough transform stage. the Hough transform converts each point in the image into a sin curve and each line in the image into a point in the Hough plane.

$$(x_0, y_0) => x_0 cos(\theta) + y_0 sin(\theta) = \rho$$
$$x cos(\theta_0) + y sin(\theta_0) = \rho_0 => (\rho_0, \theta_0)$$

with this transform, if the adjacent points form a line, the sin curves in the hough space cross one point. if the random samples of the edge are chosen to detect the line, the transform is probabilistic. this technique reduces the time and processing power. The parameters for Hough transform are

- ρ_0: ρ unit in the hough space, low value causes the line detection to be more sensetive, and with a high value, the line would be lost, i found out 2 is good value
- θ_0: θ unit in the hough space, similar to the ρ_0, i chose the minimum value for this parameter, which is one gradian
- threshold: number of the votes for the line. the value of 15 was chosen. higher value will cause more accuracy but loss of data.
- minimun line length: how long the the line should be to be consider as a line. i chose 10.
- maximum gap in the line: maximum gap in a line. i chose 20. a higher value allows the algorithm to fill the gaps and results in longer lines but less accuracy since it is possible that the algorithm finds a line where there is none.

Modifying the draw_line function

In order to draw a single line on the left and right lanes, I modified the draw_lines() function by first, determining each line belongs to which side. this is done by calculating the slope of each line and comparing its endpoints to the middle line. for example, lines with the negative slope (wrt to the origin at the top left) can belong to left lane if the x of both endpoints are less than the x of the middle.

also, to add more constraints, i consider a minimum slope for the lines, meaning if the line segments are too horizental they are not part of the lane lines. this is a fare assumption since lane lines are almost vertical. some line segments are rejected at this stage, meaning they have the wrong slope or they are on the wrong side of the image.

after this stage, since all the endpoints of the lines which fit the condition of the lane lines are determined, a great line can be fit to all the lines since all of them have to be on one giant line. I used linear least-square regression method implemented by scipy. this function gets all the endpoints as the input and returns the slope(m), intercept (b), and the error of the regression.

using the estimated m and b now we can draw the detected lane lines on the image. the endpoints of lane lines are needed to draw them on the image. since the y of the bottom and top of the region of interest are known, we can calculate the x like this:

$$x_{top/bottom} = (y_{top/bottom} - b_{est})/m_{est}$$

2. Identify potential shortcomings with your current pipeline

One potential shortcoming would be what would happen when there are stains with the same color and orientation of the lane lines on the road. in this case, those stains are also taken into account in the linear least-square regression, making the lane line incorrectly drawn on the screen.

Another shortcoming could be the region of interest, color filters, canny and hough parameters are hard coded in the pipeline. this is the problem when this pipeline is applied on a different set of images and videos.

Moreover, the drawn lines on the videos are a bit flickery and unsteady perhaps due to the average methos used which calculates the lane lines based on all line segments.

3. Suggest possible improvements to your pipeline

A possible improvement would be to add memory to the function which processes the video frames. this would help in the situations like the challenge video where the image of the road is too noisy to detect any lane lines. with this technique, i suppose we can focuse on the region where the lane lines had been detected before and ignore the noise and other changes on the road.

Another potential improvement could be to eliminate the line segements which are too far away from the others. this eliminates the noise in the lane line detection (which causes the lane lines to be unsteady) due to

In [ ]: