FINDING LANES REPORT

By Vahin Reddy, B.Tech Mechatronics – VIth Sem

(1). Introduction

With this report, I will cover on how I set out on creating a pipeline that will source a given file format to detect and construct an understanding of the lanes on a road. The report will cover the two-part process that I went through to achieve the outcome.

Firstly, I covered the steps used to process a single frame/ image. This was achieved by using graying, extracting noise, filtering, masking, canny edge detection and Hough transform. Second and lastly, I went through an averaging procedure that would draw one uniform line on both sides of the road, as well as, using the same code to processes a video, frame-by-frame. This video would then be saved as an mp4 file.

At the end of the report, I concluded on the complete procedure used, as well as points to improve on for later.

(2). Source image

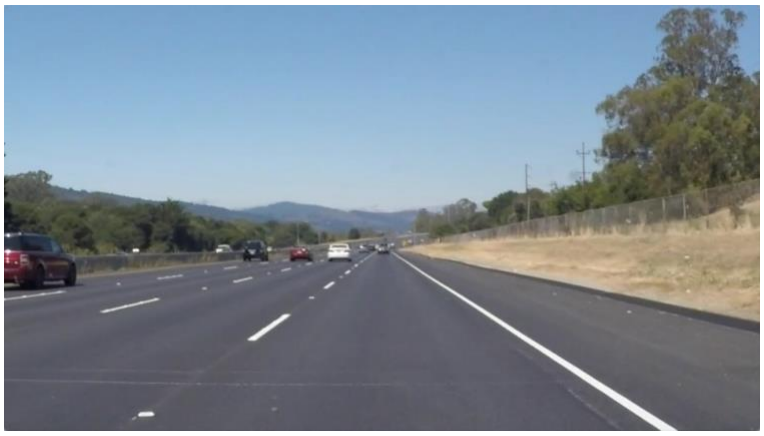

Fig. 1: Provided source image

The source image used constitutes of a vehicle travelling on an open highway road on a lane closest to the edge. The traffic density is low, and the road ahead is clear from obstructions. The vehicle must avoid crossing the right lane and can choose to continue or move to the left lane.

(3). Procedure

- Targeting image
- Grayscale conversion
- Denoising and Smoothing
- Canny image filtering
- Dilation
- Region Masking
- Mask over original image using Hough Transform
- Improving marking lines
- Processing on a video

Language: Python

Module used: NumPy, OpenCV

GitHub Link: https://github.com/CabinOnTheIsle/FindingLanes_VReddy.git

Part 1 – Image Processing

(4). Target Image

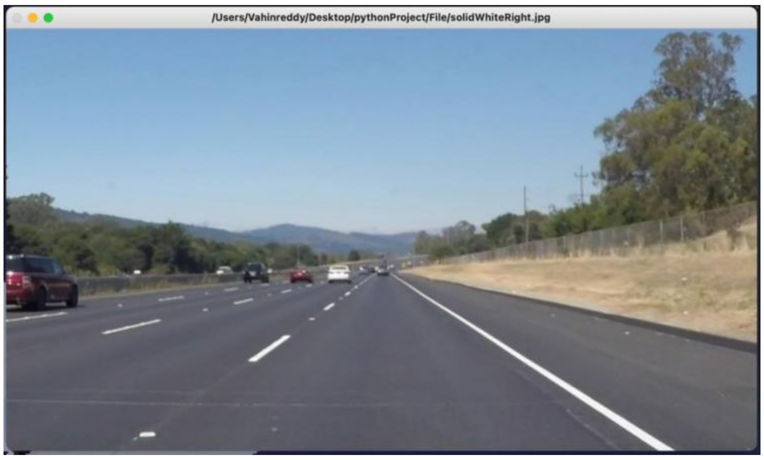
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 2: Input image

Using Open CV, we save the file path of the file to a variable. This can then be used to display the input image.

(5). Grayscale conversion

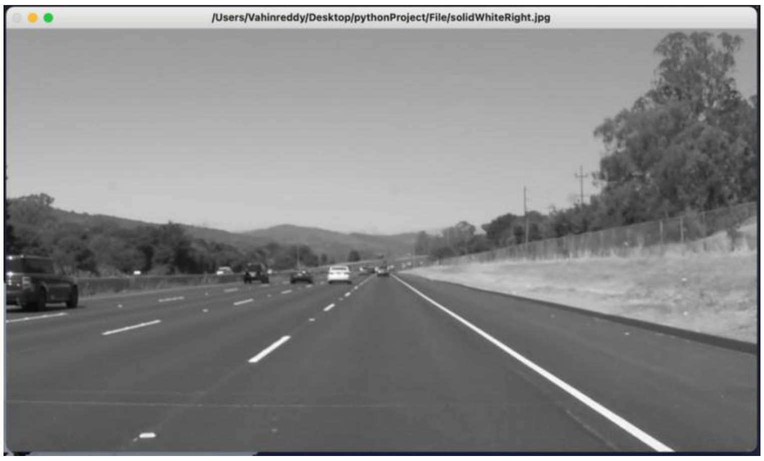
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 3: Grayscale image

To begin with, we convert the target image to a grayscale image using .cvtColor() and .COLOR_BGR2GRAY(). By graying we can minimize the processing time which would be significantly noticeable when using the same system in processing a live video feed. We can also utilize this step as stage 1 of filtering the image for further image processing.

(6). Denoising and Smoothing

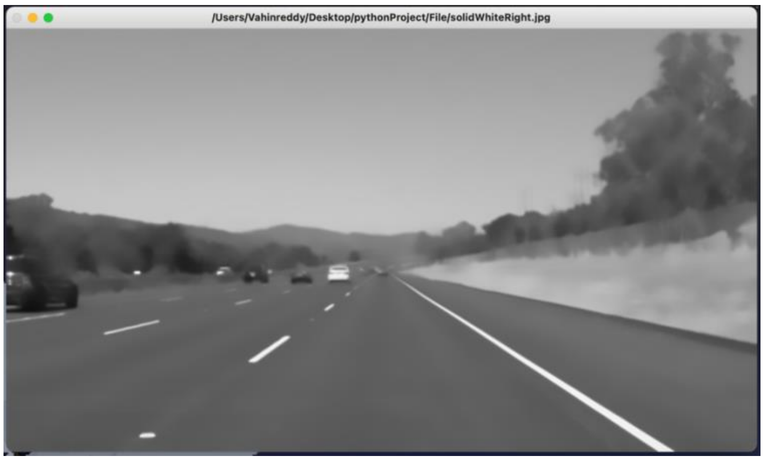
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 4: Denoised and Smoothened image

To remove the noise and smoothen the image, we use .fastNlMeansDenoising(). The image now has a blur and distant objects are not as clear as the original image. The lanes necessary are still visible which will be used to in later steps. Through trial and error, I received better results using a threshold of 25.

(7). Canny image filtering

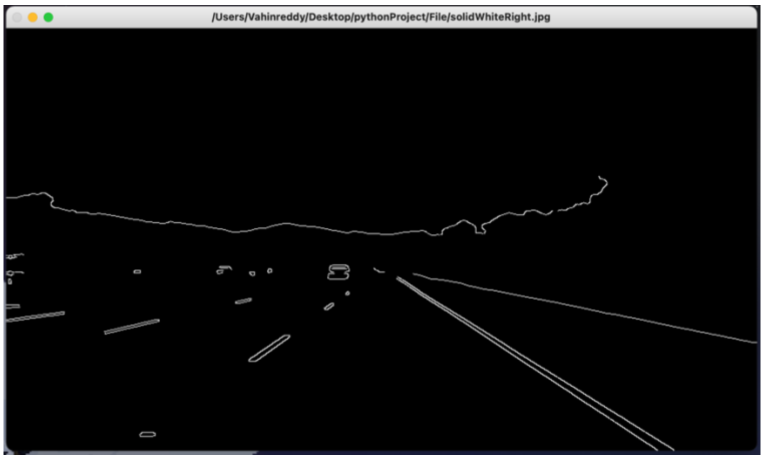
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 5: Canny image

We procced to canny filtering the image using .Canny(). As we can observe regions where the image showed a greater shift from dark to light - for example, the area between the road and the white lane marking – these sports were bordered with a white border. The remaining space is black. If denoising and blurring were not performed prior to this step, the Canny image filter may pull out noise in the image. The thresholds after testing were 100 and 200 respectively.

(8). Dilation

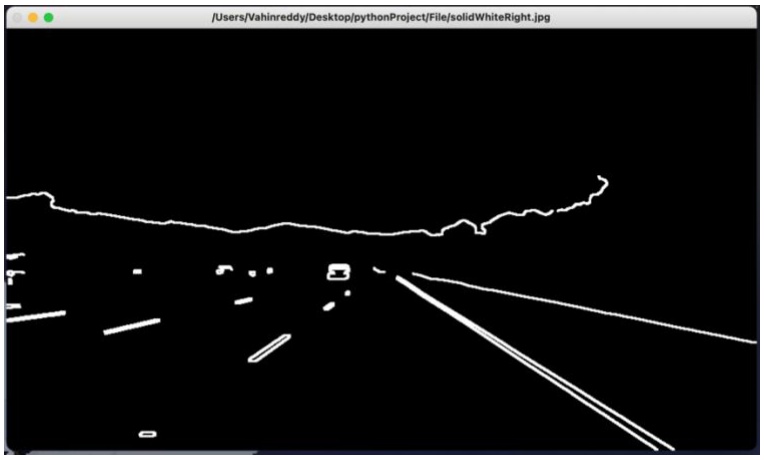
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 6: Dilated canny image

Dilation was used to increase the thickness of the white borders in the image to make the lanes more visible.

(9). Region Masking

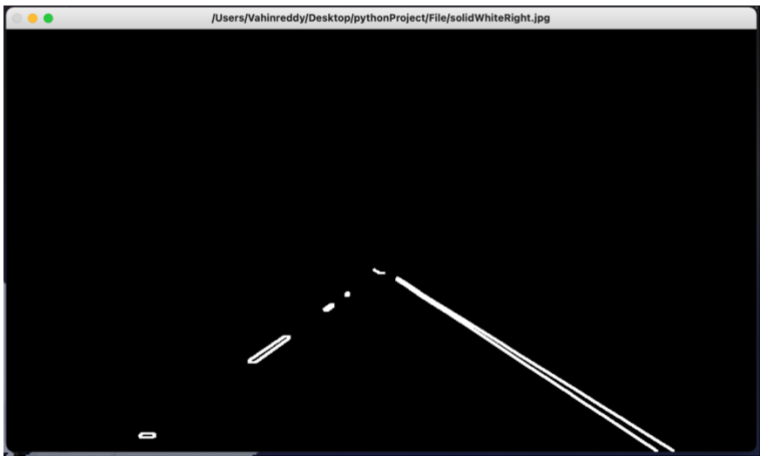
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 7: Masked canny image

To get the final part of the lane detection, we mask off the required area. Using the method presented by youtuber ProgrammingKnowledge from his video, we procced by creating an array of points of interests that will create the triangle masked region. We then create a function that will receive 2 inputs: image and region of interest. By using the 2 inputs, we start of by creating a blank array of zeros that is in the same shape as the image. The code then using the match mask color variable, we can reveal out the section within the region of interest while leaving the rest of the image dark. This image is then processed as the output of the function.

It is important to perform the canny image filtering before masking as this avoids creating a border around the region of interest.

(10). Mask over original image using Hough Transform

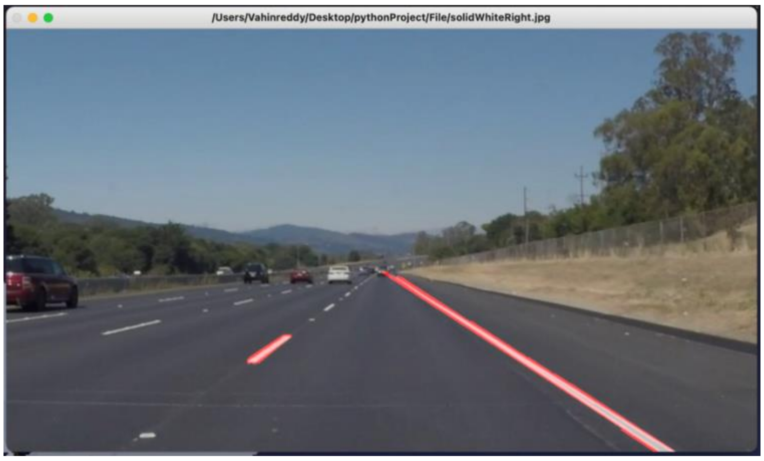
/Users/Vahinreddy/Desktop/pythonProject/File/solidWhiteRight.jpg

Fig. 8: Result image

Lastly, we region of interest array to construct the red lines from the masked imaged, on to the original image. This method follows similar characteristics to that needed in a Hough transform function to create straight lines that follow along the lanes of the road. As we can see the Hough transform shows the solid white line with a straight red line. This can tell the system that the right lane is inaccessible to overtaking when the vehicle approaches another vehicle ahead. The left lane, however, does show a breakup in the lane. This can tell the system that the left lane is available to perform a pass when required. With modification to the code, we can perform lane detection from a live feed camera on to each frame.

Part 2 – Improved Lane Detection and video processing

(11). Source Video

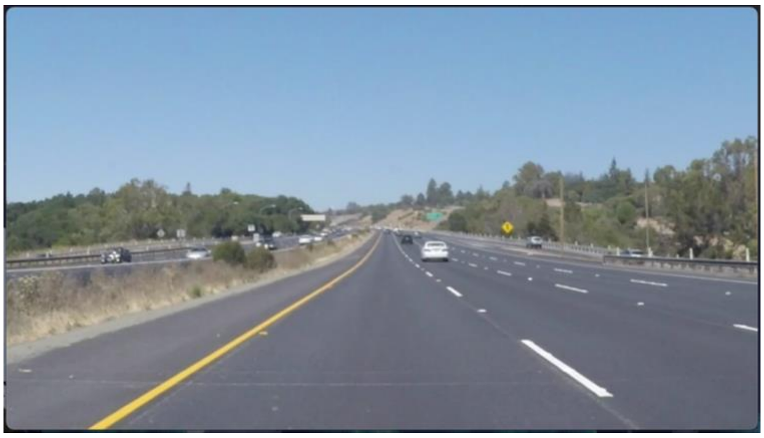

Fig. 9: Provided source video

The source video used constitutes of a vehicle travelling on a straight open highway road on the fast lane. The traffic density is low, and the road ahead is clear from obstructions. The vehicle must avoid crossing the left lane and can choose to continue or move to the right lane. This video is 27 seconds long.
For the improve lane detection, we will use one frame of this video to work on.

(12). Averaging the slopes and intercepts

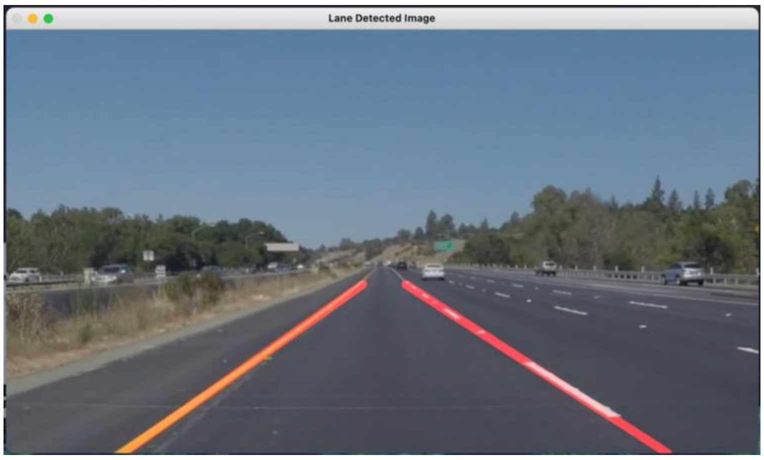

Fig. 10: Averaged lane detected image

Before beginning, the previous procedures were defined in a function to accept the necessary inputs and return the required output.

As seen in the previous step, the lanes presented by the program were scattered and presented a range of straight lines parallel to the lanes. To improve on this, two functions were defined: average_slope_intercept and make_coords. Average_slope_intercept creates two empty arrays for the left fit and right fit. The for loop within sifts through all the data points and appends the data to the correct fit after sorting, using the slope variable. The average of the sorted fits is saved in a new variable. Using the make_coords, we pass the image and the average of the fit. This will return out the first xy and the second xy for the respective lines in a 1D NumPy array. Average_slope_intercept will finally output the 1D NumPy array with the left_line and right_line coordinates.

Finally, this will then be passed onto the draw_lines function to create 2 solid lines as seen in the image above.

(13). Video processing

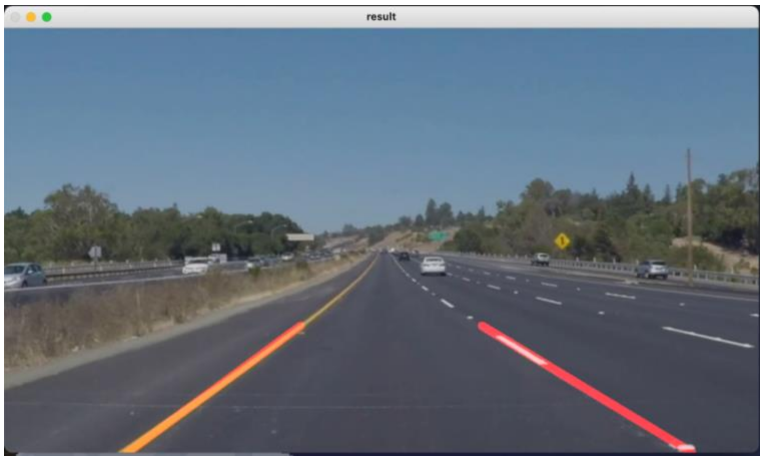

Fig. 11: Averaged lane detected video

Using the same functions, we can process each frame of the video to display lane lines on the road as the vehicle drives through traffic on the highway. Using the cv2.VideoCapture() and While loop to keep the video open, we save the frames in a variable, frame. The same procedure as the previous image processing is done and the output is displayed in the “result” window. The process will go through until the video is over or the user interrupts by using ‘q’ keyboard interrupt.

(14). Result

At the end of the procedures used, we were able to perform lane detection on a given image as well as a given video. We utilized the same code to perform various image processing and masking to acquire the desired output.

The image displayed showed 2 parallel lines placed exactly above the lane markings that the vehicle was driving on. These lines were drawn from the bottom of the image, till a set distance below the center of the image.

The video displayed the same output as the image processing; however, this processing was done to each frame. It can be noted that when the frame showed a large space between road markings, the respective line drawn would not appear for that frame. The video processed would then be saved to another .mp4 file that was created and written into.

(15). Limitations

The lane detection model employed here may not perform well in every circumstance. These may include:

- Curved roads
- Intersections and roundabouts
- Worn or un-marked road markings
- Road direction and speed indicators
- Bridges
- Close range vehicles
- Various time and weather conditions

(16). Improvements

Considering the downfalls of the model, the model can be improved by:

- Continuous edge tracking
- Dynamic region of interest detection
- Image correction for various lighting and road conditions
- Lane color differentiation
- Fixing bug on large gaps between lanes

(17). References:

Image processing codes:

https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gga7be549266bad7b2e6a04db49827f9f32a2b9f6b6fb168b4d1d5277caebfe7b73d

https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c

https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

Referred tutorials:

https://www.youtube.com/watch?v=yvfI4p6Wyvk

https://www.youtube.com/watch?v=G0cHyaP9HaQ

https://www.youtube.com/watch?v=eLTLtUVuuy4

Error correction and saving video:

https://stackoverflow.com/questions/54273077/cannot-unpack-non-iterable-numpy-float64-object-python3-opencv

http://www.learningaboutelectronics.com/Articles/How-to-save-a-video-Python-OpenCV.php