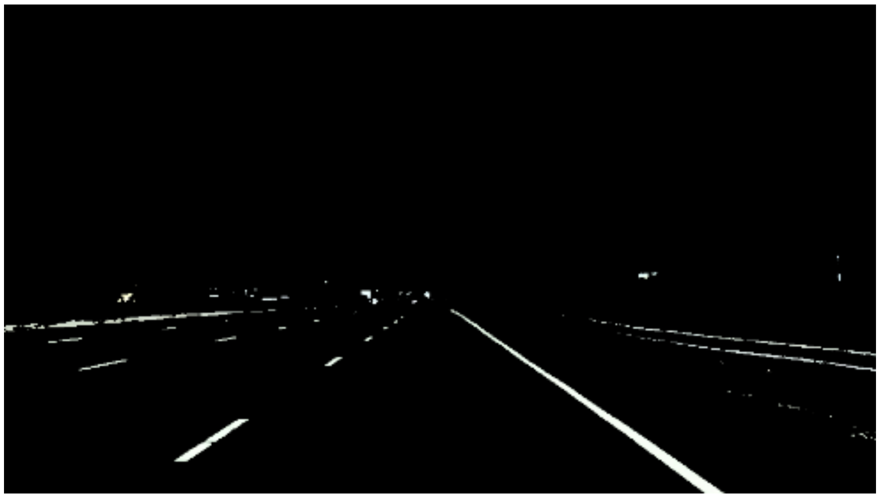
Self-Driving Car Engineer Nanodegree
Project 1: Finding Lane Lines on the Road
Lakshay Khatter

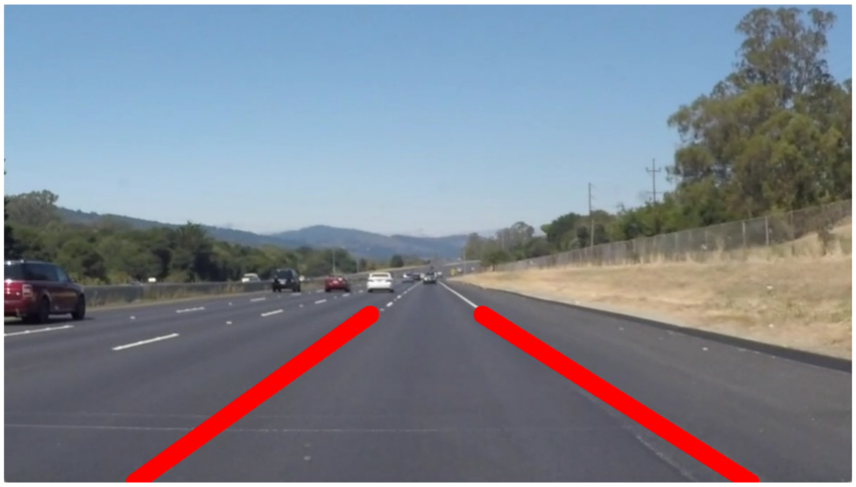
Self driving cars have many different parts that are required to be fully functional. One of these aspects are detecting lane lines. Using different computer vision approaches we are able to find, mask, and extrapolate lane lines onto an image. This paper will go through the journey.

Computer vision problems are interesting for human beings because they in hindsight seem simple. It's easy to label a lane line; you can look for a. How do we translate that into something a computer can identify? Well we can hard code these labels for an image but that doesn't work well across a wide array of unseen results. Taking this approach would be finding all the white pixels in an image and marking the lane lines as seen below (with additional transformations on the image). This is very tedious and does not lead to successful results.



A better approach is to find an image agnostic way:
1) Take an existing image and turn it into a grayscale copy
2) Get the edges using canny edge detection
3) Create a masked image on top of that to isolate the lane lines of interest
4) From that region of interest, extract the hough lines
5) Average out the hough lines and extrapolate the lines.

Which leads to results as seen below.



Self-Driving Car Engineer Nanodegree
Project 1: Finding Lane Lines on the Road
Lakshay Khatter

While most of the interesting things were done using the computer vision library openCV. There was another component of expanding upon the draw lines function. I set up the function as such:

1) Iterate of the lane lines
2) Calculate there slope
3) Check if they fall within our range of absolute value of 0.4 and 0.9
4) Check if they are a left or right lane line and append it the correct list
5) After all the lane lines are segmented, Find the median line and calculate two points as seen in the code. Do this for both left and right lane lines
6) Draw out the lane lines to the page.

The results of this approach are average and perform well on this set of images however they would not perform the best on a different set of images, or even if the lane lines were overly curved. A way to overcome this is to shorten the lane line extrapolation, but even than does not solve the problem. There are more improvements to be made. I think one of the things I wanted to factor in was the function cv2.inRange(). This function allows us to mask images based on color. By doing that to detect lane lines we have a small set of possible colors to choose from and can hard code this. I feel a combined set of techniques across both edge detection and lane lines would allow for superior results.