FINDING LANE LINES ON THE ROAD

The aim of this project is to find the lane lines on the road. It uses the following two known techniques to do so,

1. Canny Edge Detection
2. Hough Line Transforms

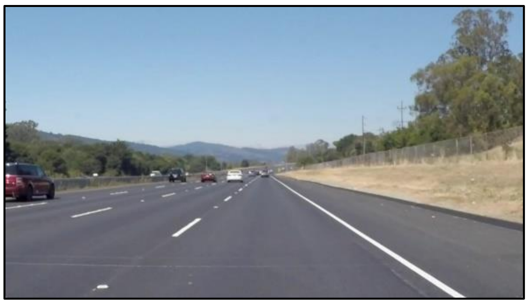
Fig 1: Captured Video Frame

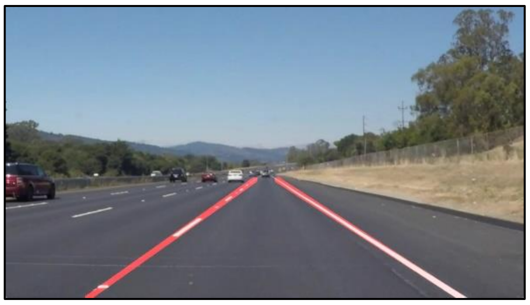
Fig2: Detected Lanes

This is a rather rudimentary way of finding the lane lines, or any lines on an image for that matter. The current robustness, followed by the likely shortcomings and lastly the potential improvements, have all been described below.

Pipeline:

The following describes the algorithm, right from accepting an image/frame, to detecting the lane lines and overlapping them with the same frame (*Fig 1* to *Fig 2*). This runs well with a particular image, but a video is nothing but a collection of images/frames, and hence it works well with a video as well.

1. Converting each coloured frame to a grayscale image containing only two channels (white and black). There's no need to capture the colour data since we don't use the RGB values in determining the lanes. This is one way of increasing the robustness by detecting the lanes irrespective of their colours, as seen in the subsequent steps. The resultant image is also smoothed using a kernel of **size 7**, in order to remove any unnecessary noise from the image.

Fig 1: Captured Frame in RGB

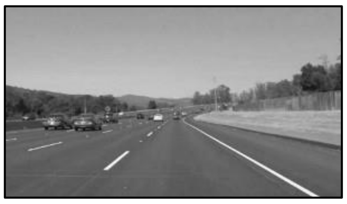
Fig 4: Captured Frame in Grayscale

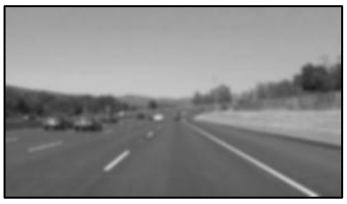
*Fig 5: Smoothed Frame (**k=7**)*

2. The edges (defined by sudden changes in image intensity, *Fig 6*) on the smoothed image are then detected using the Canny Edge Operator available in OpenCV. The current parameter values used for this function are as follows:
    a. Low Threshold: **20**
    b. High Threshold: **75**

Low and high gradient thresholds using which the edges are detected

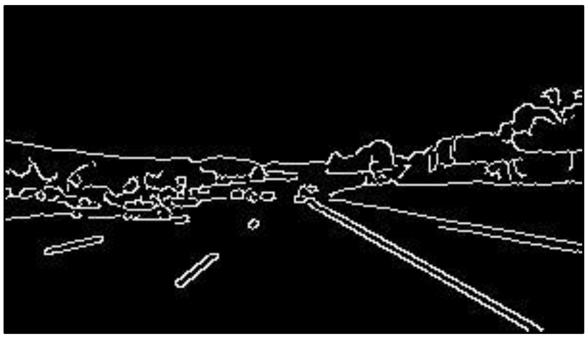

Fig 6: Canny Edges

3. The edges detected by the Canny Operator give the lane lines distinctly, but there's a lot of other unnecessary pixel data in this image as well. Hence, we filter out the region containing only the lane lines by defining a polygon in which only the lane lines lie. We black out rest of the image, and keep only the pixel data lying inside this polygon; and save this to a new image (*Fig 8*).

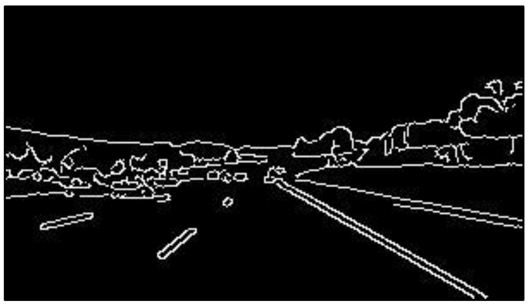

Fig 7: Canny Edges

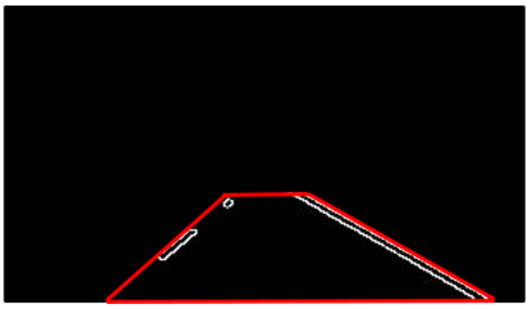

Fig 8: Canny Edges after masking

4. The edges obtained by the above operator after masking, do give us pixel data on the lane lines, but not enough data for us to clearly define the lane lines. Hence, we use the Hough transforms, where we use these pixels to plot lines over the image, thereby defining the lane 'lines' and not lane 'edges'. The 'HoughLines' operator available in OpenCV is used for this purpose and following are the parameters that I have used for this function.

Resolution for d & Θ	**1 & 1°**
Maximum length of line	**50**
Minimum gap b/w two lines	**10**
#pixels that define the line width	**10**

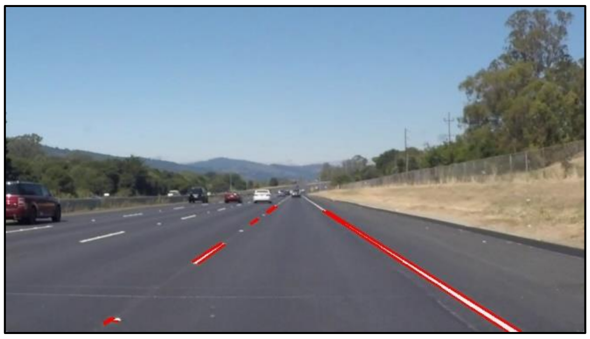
Fig 9: Hough Lines on the RGB Frame

5. The lines defined in the above step are not continuous and are present only in the regions where markings on the road are present. I have extrapolated the above data to construct continuous lines that run all along the lane line (Fig10). For this, instead of defining a single polygon for masking, I have used **two polygons – one for the left lanes and the other for the right lanes**. By doing so, I was able to segregate the line end points (obtained from the Hough operator) between the right and left ones. I then used these points to fit a first-degree polynomial over them; and this polynomial is nothing but the 'equation' for the lane line; which was then used to plot a continuous line.

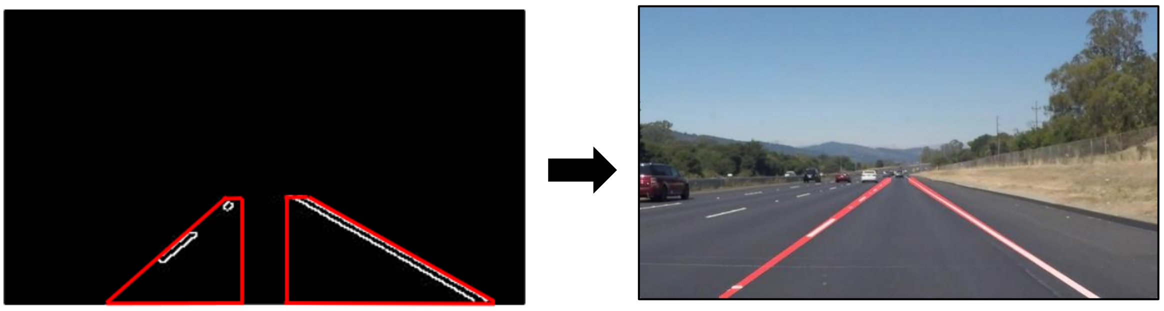
Fig 10: Two masking polygons

Fig 10: Continuous Lane Lines

<u>Shortcomings of this Pipeline:</u>

1. Since the masking polygons are defined using hard points entered directly in the code, this works well only for the case when the camera position on the vehicle is relatively similar to the one in the test videos. That's one of the reasons why this did not work well with the 'challenge' video.
2. The parameters that were used to run the Canny operator and the Hough line operator were found out by hit and trial. Again, entering hard points or values in the code is never a good idea; and would probably not work well in the different scenes and settings that the automotive can run in.
3. It will not run well with curves in the lane lines during cornering.

Possible Improvements:

1. If the position of the camera is made standard across all car manufacturers (for e.g. the fuel filler/charge port), the region within which the lane lines have to be found out can be narrowed down.
2. The parameter searching could be bettered in the following manner (provided we know the polygon region in which the edge search needs to be carried out):
    a. In case of the Canny operator, after the pixel values (for a generic value of low and high threshold) in the required region are obtained, we can carry out median filtering over the region. The unwanted pixel values, that are not part of any line or are part of smaller edges, can be treated as classic ‘salt & pepper noise’; and hence can be taken out using median filtering. The kernel size can be small enough such that continuous line edges are not filtered out, but random specs in this region are. Haven’t checked to see how robust this could be.
3. The curved line problem can be sorted by choosing the masking region small enough such that the lines are straight within the region (since any circle can also be thought to be comprised of smaller line segments!). But obviously this would not work well with corners with a shorter radius (hairpins and blind turns).