# ****Finding Lane Lines on the Road****

# Pipeline

My pipeline consists of the following steps:

1. First, the image is converted to greyscale using the helper function *grayscale(img)*



1. Then Gaussian smoothing is applied with kernel\_size 5 to filter out any noise – helper function *gaussian\_blur()*



1. Then, apply the canny edge detection algorithm that returns points that represent edges – helper function *canny(img, low\_threshold, high\_threshold)*



1. Construct a region of interest( ROI) using 4 vertices that form a trapezium. This only keeps the region of the image defined by the polygon formed from `vertices`. The rest of the image is set to black. Used helper function *region\_of\_interest(img, vertices).*

It was important to choose a region of interest with vertices approximated using a more generic approach, otherwise, it wouldn’t work on all the images and the videos.

vertices = np.array*([[(0.05\*imshape[1],ybottom),(imshape[1]\*0.45, ytop), (imshape[1]\*.55, ytop), (0.95\*imshape[1],ybottom)]*], dtype=np.int32)

Apply this mask to the edges returned earlier from canny so that only the points on lane lines are highlighted and everything else is black.



1. Run the Hough transform that returns an image with hough lines drawn– these would be dotted lines. And then draw lines on the vertices returned by hough. – Helper function hough\_lines

**Improvisation to draw\_lines() helper function**: In order to draw a single solid line over the left and right lanes, I separated the vertices from hough lines into left and right. Any line with positive slope would go towards the right line(origin is left top), and negative slope would go towards left lines. Also, I filtered out vertices from any hough lines that were kind of horizontal, so only took lines with absolute value of slope between 0.5 and 0.8.

Now that I got the vertices for the left and right separated, I wanted to find a line that best fits the points. For this purpose, I used np.polyfit() with degree 1 to fit a straight line and got the slope and intercept using it. And then got x coordinate for the top y and bottom y as per area of interest. And then plot a line using (x\_bottom, y\_bottom) and (x\_top, y\_top). Did this process separately for left and right lane to get 2 wholesome lines like below.



1. Superimpose this image with the original image using weighted\_img() helper function



# Test Image outputs

Here is the output of the pipeline on all the test images.

1. solidWhiteCurve



1. solidWhiteRight



1. solidYellowCurve



1. solidYellowCurve2



1. solidYellowLeft



1. whiteCarLaneSwitch



Running the pipeline on the videos was very rewarding experience. The challenge video was actually very interesting. Had to make some more tunings to the parameters to take care of the issues.

# 2. Potential shortcomings with current pipeline

1. Lanes clearly not marked or Steep curves –

The current approach and the pipeline is based on the assumption that lanes are generally straight. Sometimes lanes are not marked completely or there is too much spacing, so we would see dotted lanes. In that case, the pipeline may struggle as Hough lines won’t do too well. Similarly if the lanes curve steep, np.polyfit() might not be able to fit a straight line.



1. Shadows

If there are shadows or insufficient contrast in the image, edge detection will not work.

1. Car immediately ahead in the lane

If there are cars immediately ahead, it is possible that all of the lane markings won’t be clearly visible to the camera (atleast in the center), so it will be difficult to plot lanes. Also the algorithm would take into account edges of the car itself, which will be a problem



### 3. Suggest possible improvements to your pipeline

* Instead of np.polyfit with degree 1 to extrapolate the lane lines, we might have to allow for curves, but it might also introduce more issues. Use exponential smoothing
* Have some kind of filter for colors white and yellow to remove interference from foliage and the separator at the shoulder.
* If there are vehicles ahead, canny edge will detect its edges too. So improvise that to filter for edges that are straight and at some angle to the car
* Use deep learning to train and identify lanes based on other images?