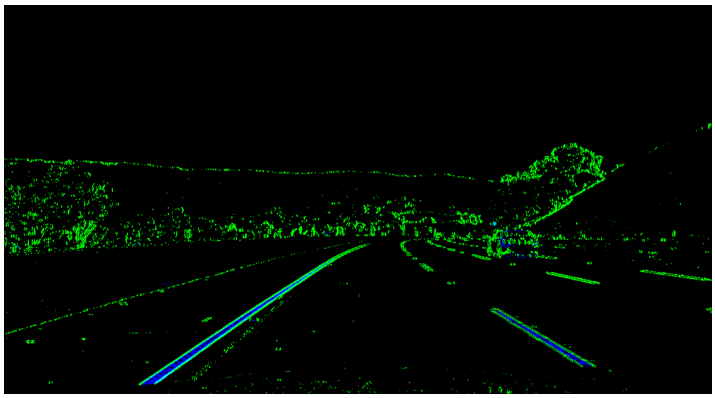
1.)
The first step in detecting lanes is to calibrate the camera because if the camera is distorted the results won't be effective. So, I was given 20 camera image of chessboard using the car's camera from different angles. Using OpenCV we calculate obj_points which is basically poinst of chessboard on x,y,z axis where z axis is 0. So, it starts from (0,0)-(6,9). Then we have img_points which are the points calculated by cv2.findChessboardCorners(). Using these points in list we feed them to cv2.calibrateCamera() which provides us with details like the positon of camera , camera matrix, distortion coefficient.
After getting all this information about the camera we use cv2.undistort() to undistort any images which we get from the camera.
The right image is the undistorted image. If we look at left hand corner of each image we can see the white spot tend to be different which sugges the images have changed.

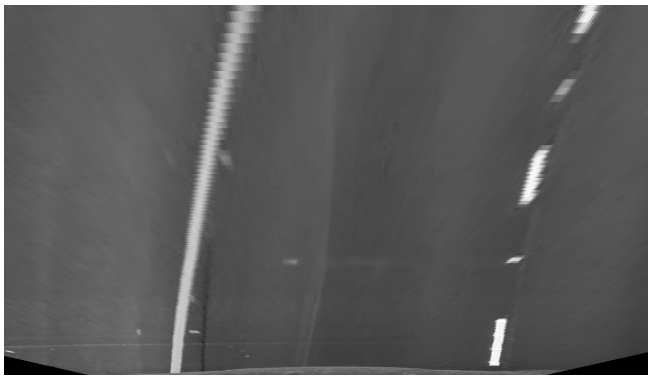

2.)This second part exists because when we try to convert an image to gray scale we tend to have some information loss. So, if there is too much sunlight or shadow over the road and we take the grayscale gradient of it, it cause information loss. So, we start using HLS and s channel of it in combination with gray gradient threshold which helps us preserve most information.



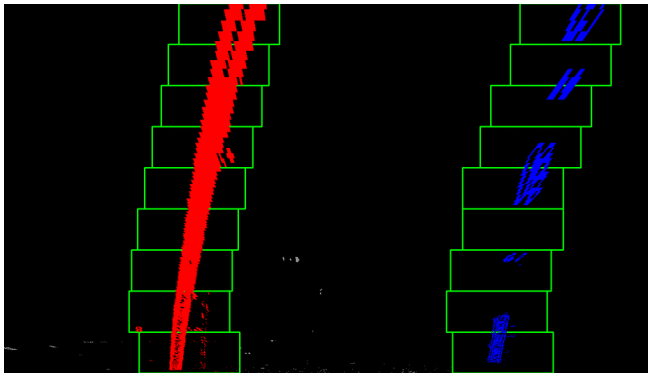
3.) Perspective Transform helps us to see the bird's eye view of the road because when we see the road from ground level we tend to see it straight but from above we can see if there is a curve in lane.
We use cv2.getPerspectiveTransform(src, dst).
Where src is a polygon we use in normal image and dst is the same polgon in our birds view image.

```
img_size=(img.shape[1],img.shape[0])
src = np.float32(
    [[(img_size[0] / 2) - 55, img_size[1] / 2 + 100],
    [((img_size[0] / 6) - 10), img_size[1]],
    [(img_size[0] * 5 / 6) + 60, img_size[1]],
```

```
    [(img_size[0] / 2 + 55), img_size[1] / 2 + 100]])
dst = np.float32(
    [[(img_size[0] / 4), 0],
    [(img_size[0] / 4), img_size[1]],
    [(img_size[0] * 3 / 4), img_size[1]],
    [(img_size[0] * 3 / 4), 0]])
```



4.)Finding the lane lines using sliding window technique.
In order to identify the lanes which were produced from bird's view from perspective we deploy sliding window technique.
We use a histogram to find the lane marking for the bottom of the image and we also choose a windows. These windows are used to find activated pixels in the that region of the image.
After all the pixels for the lanes are detected, we make a polynomial which perfectly satisfies going through the pixels we selected. We form two polynomial equations for two lines.



5.) Calculating the radius of curvature of lane can we done using the varibales which we got from fitting the polynomial equation in last part.
$Rcurve = |2A|(1+(2Ay+B)2)3/2$

6.)Lane area can be indentified using everything we did until now from using the warped image from perspective transformation, to the equations from polynomial. We take the points from left lanes and right lane and fill all the area between them and superimpose it over a black image of same dimension. Then we then do an inverse perspective to make the image normal again and apply it over the undistorted image.

PIPLINE:
The video is attached in the github repo

Discussion:
The major problem which I ran through was that I was using cv2.imread and cv2.imwrite and using these the images which I got had inverse colors which made my output incorrect at multiple places. I had to start again and change and use mpimg.imread and plt.imsave() to overcome the problem of inverse color. My pipleline is likely to fail if the lane lines are too lines and faded away. This will cause problem in detection.