Udacity Self Driving Cars Nanodegree- Advanced Lane Finding Project

Objective:

The objective is to write a Python program to identify the lane boundaries in a video from a front-facing camera on a car. The camera calibration images, test road images, and project videos are provided.

Steps:

1. Removing Distortion
2. Isolate Lanes
3. Warp
4. Curve Fit
5. Final Image

1. Removing distortion:

Cameras use lens which distort image. OpenCV provides function to correct these distortions and to calibrate the camera.

A set of 20 chess board images were used to calibrate camera.

First, a set of chess board image are passed to findChessboardCorners function to get the corner points, thru the following code:

Ret, corners=cv2.findChessboardCorners (gray, (9, 6), None)

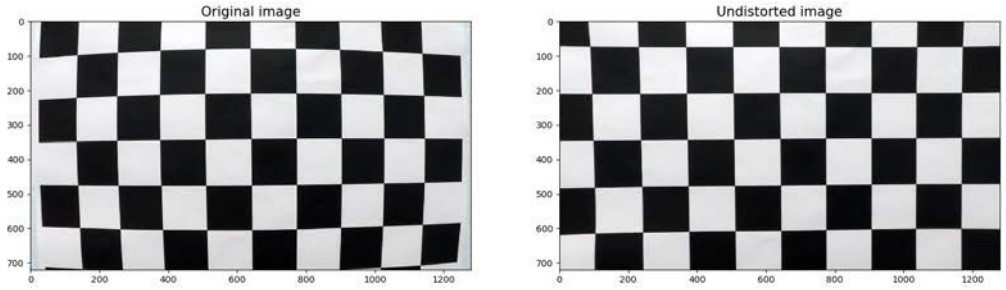
Then, the corner points discovered are passed to the calibrate Camera function, thru the following code:

Ret,mtx,dist,rvecs,tvecs=cv2.calibrateCamera(objpoints,imgpoints,gray.shape[::-1],None,None)

This function returns the camera matrix and the distortion coefficients which are then passed to undistort function to correct the distortion of the images, thru the following code:

undist=cv2.undistort(img,mtx,dist,None,mtx)

The following image shows the Original image and the undistorted image side by side:



2. Isolate Lanes:

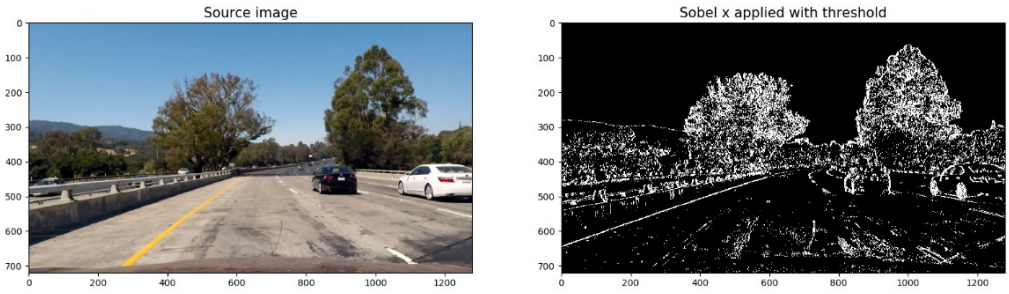
Then, color and direction gradients are applied to isolate lanes. Color spaces (RGB, HSV, HLS, LAB, etc.) are explored for the images to see which channels of these color spaces may be used to distinguish lane markings based on colors. In our example, S channel of HLS color space and V channel of the HSV color space was used.

Then, direction gradient (Sobel filter) is used to detect lanes based on change of intensity (edge detection) where lane markings exists. Both color channels and Sobel filters are applied with the appropriate thresholds to isolate lane lines.

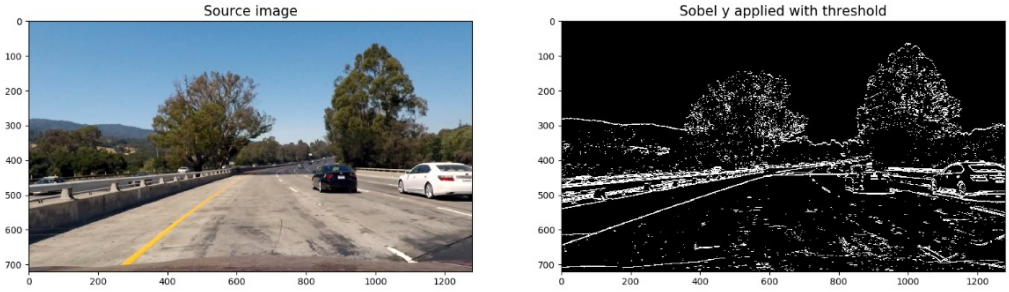
OpenCV provides cvtColor function to convert the image to different color spaces, like the following conversion to HLS and HSV color spaces:

hls=cv2.cvtColor (RGB, cv2.COLOR_RGB2HLS) hsv=cv2.cvtColor(img,cv2.COLOR_RGB2HSV)

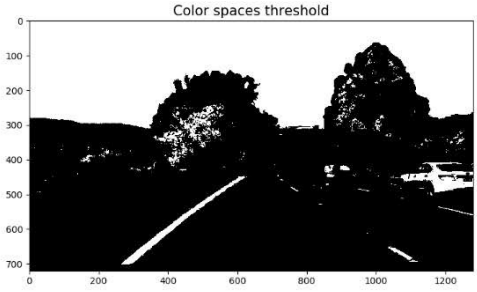
The following image shows the image after passed to Sobel x operator (with the appropriate threshold):



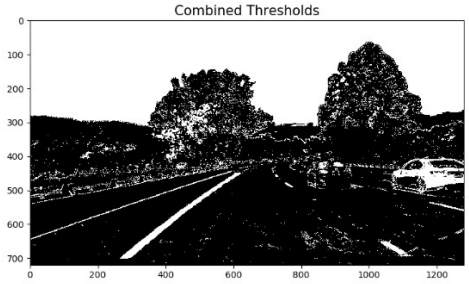
The following image shows the image after passed to Sobel y operator (with the appropriate threshold):



The following image shows the image after passed thru the Color spaces (S channel of HLS and V channel of HSV):



The following image shows the image after passed thru the combination of Sobel and Color spaces channels thresholds:



3. **Warp:**

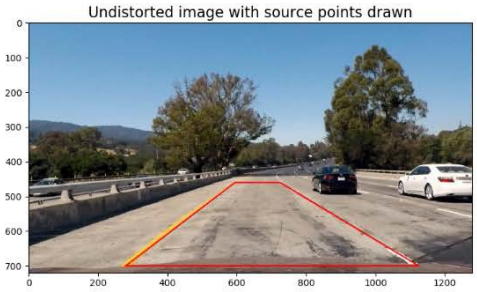
Source and destination points are passed to the getPerspective function, thru the following code:

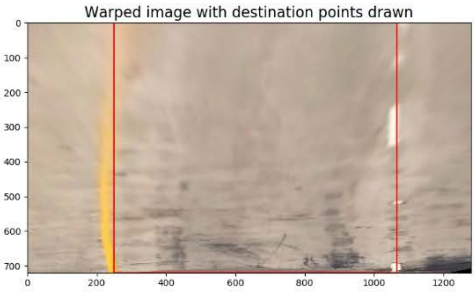
M=cv2.getPerspectiveTransform (src, dst)

This results in a Transformation Matrix M, which along with the original image is passed to the warpPerspective function to get a bird's eye view (warped perspective) of the image. Following code is used:

Warped=cv2.warpPerspective(img,M,img_size)

The following image shows Undistorted image with Source points drawn along with the Warped image with destination points drawn:





4. **Curve Fit:**

Find the left and the right lines and then apply curve fit to those lines.

We take a histogram of the bottom half of the image and we determine where the lane lines start based on wherever the histogram peaks are. Y value is sum of the pixels at that x location and the 2 maximums are where the lines start.

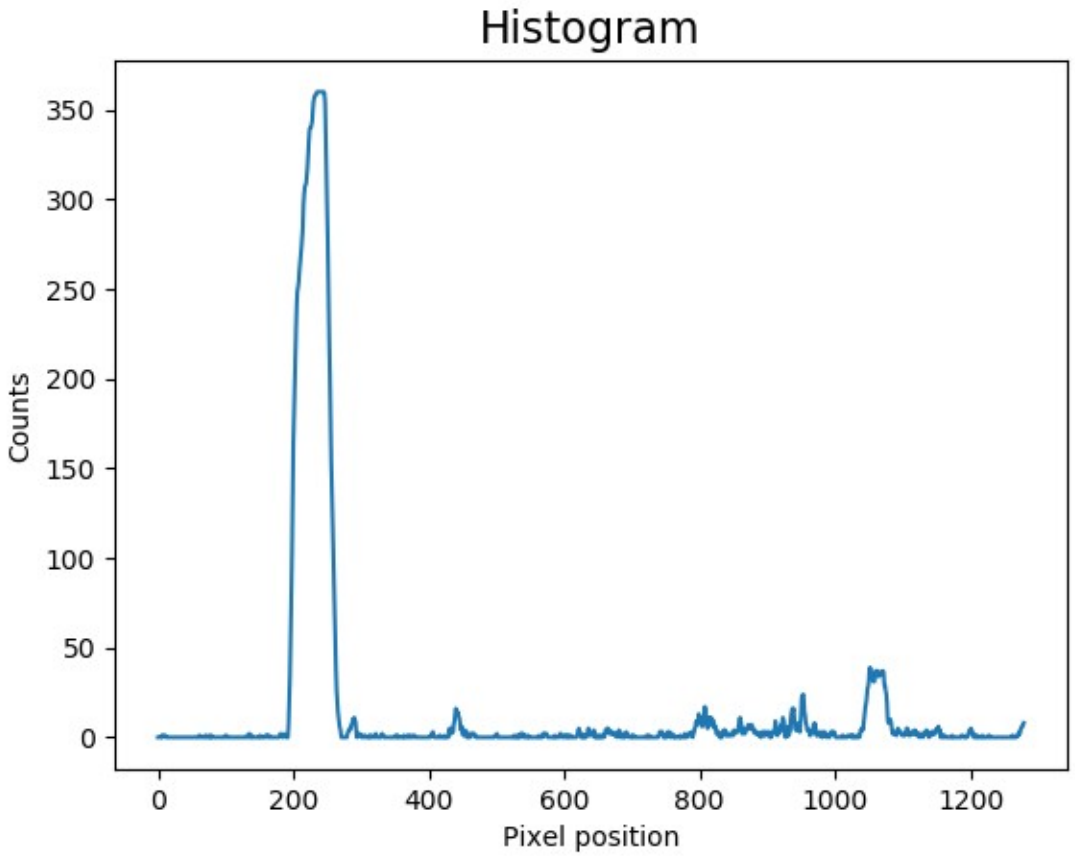
Following code is used:

Histogram=np.sum(img[img.shape[0]//2:,:],axis=0]

Leftx_base=np.argmax(histogram[:midpoint])

Rightx_base=np.argmax(histogram[midpoint:]+midpoint Following

image shows histogram output:



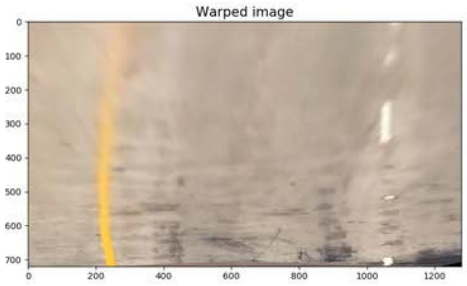
Then, we use a sliding windows technique described as follows: we draw a box around the lane starting points and any of the white pixels from the binary image that fall into that box; we put them in a list of x and y values. Then, we add a box on top of that and do the same thing and as we move up the lane line; we will move the boxes to the left or right based on the average of the previous line. At the end of this; we will have lists of x and y values of the pixels that are in the line and then we run the 2nd order numpy polyfit to those pixels to get the curve fit in the pixel space. Following code is used:

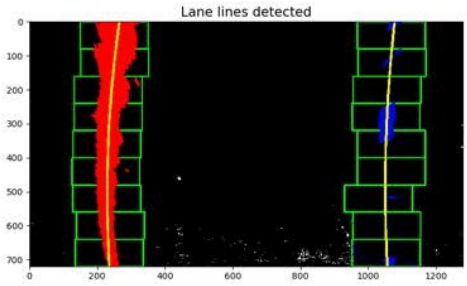
Left_fit=np.polyfit(lefty,leftx,2) Right_fit=np.polyfit(righty,rightx,2)

We then do the pixel to meters conversion based on US Highway standards (lanes are 12 feet wide).

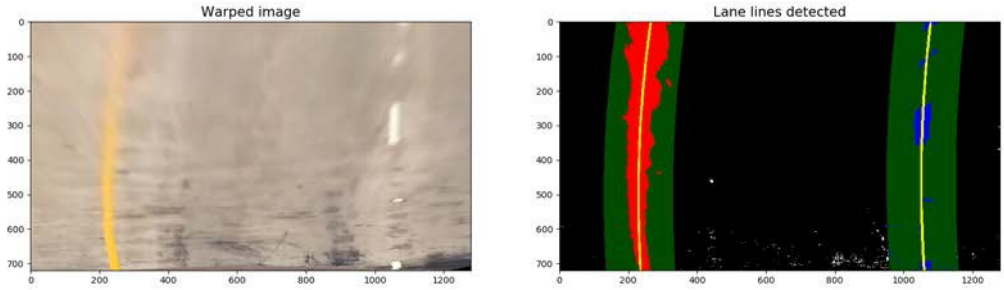
After the first window frame; instead of the sliding windows technique; we take the previous fit and move it to left and right by 100 pixels and any pixels (minimum 50) found in that area are added to the list of x and y values and then we run the polyfit function to get the lane lines.

The following image shows curve fit with sliding windows technique.





The following image shows curve fit where the program looks for the next lane point within 100 pixels of the previous fit.



5. Final Image:

Then, we create the final image and put text (using putText function) on the image for the Radius of curvature and the Distance to lane center; which are calculated using the following formulae:

Radius of curvature=([1+(dy/dx)^2]^3/2)/(d^2y/dx^2)

laneCenter=(rigthPos-leftPos)/2+leftPOs

distancetoCenter=laneCenter-imageCenter

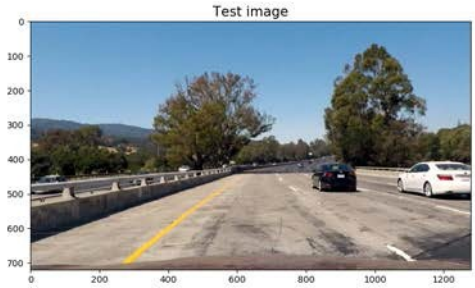
Then, we plot the curve fits and green color the drive area onto the image using fillPoly and polylines function as shown below:

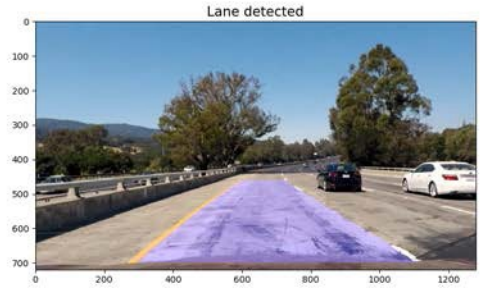
Cv2.fillPoly(color_warp,np.int_([pts]),(0,255,0))

We want to overlay on the original image and not the warped image. So we need to unwarp the image (using undistort function reversing the source and destination points) and then add to the original (using addWeighted function) as below.

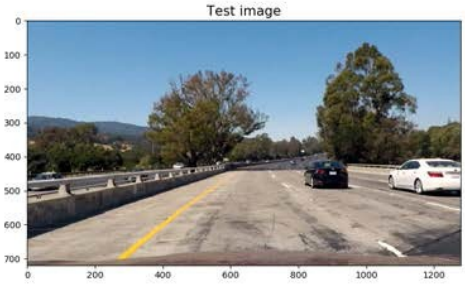
Result=cv2.addWeighted(img,1,newwarp,0.5,0)

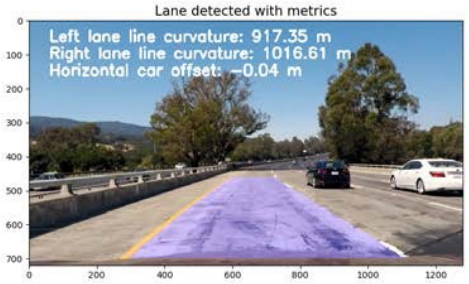
The following image shows the final drawn image with color filled between the lane lines.





The following image shows the final drawn image with color filled between the lane lines; and text illustrating radius of curvature and distance from center.





Improvement Points:

There is lot of manual effort in the following areas; so the program will not scale well for all videos and images:

1. Selection of best channels from the color spaces, which are used for lanes detection
2. Selection of thresholds (minimum and maximum) and parameters (example, kernel size) for Sobel filter and for color spaces
3. Selection of source and destination points for perspective transform

A deep learning/machine learning approach will be more appropriate for the above points; so the program can learn those parameters itself; rather than we manually specifying or selecting those parameters.