Advanced Lane Finding Project

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration

Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

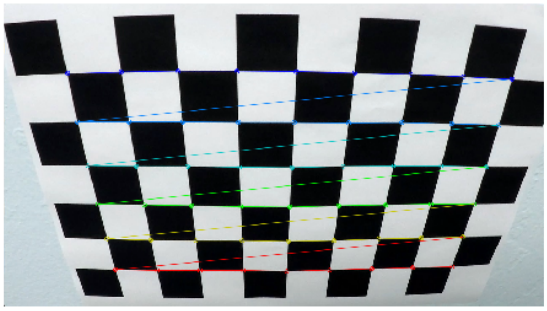
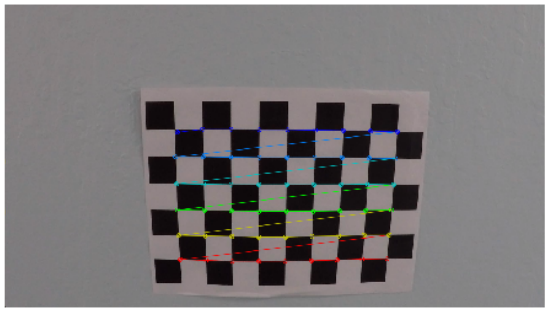
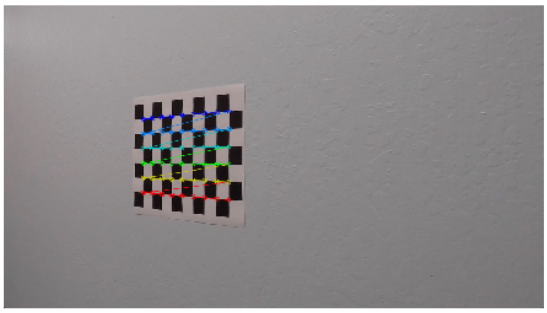
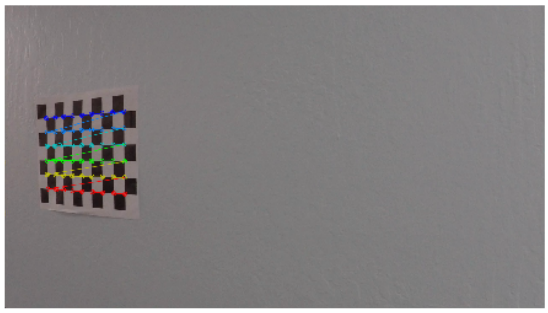
- The code for this step is contained in the first section of the IPython notebook located in "CarND-Advanced-Lane-Lines/P2_Borysenko.ipynb"

Goal: calibrate a camera to get rid of distortion on the image.
Approach: Take pictures of the known shapes → this will help us identify the distorted areas of the picture. Any regular+hight contrast pattern could be used - we will use checkboard as one of the most common ones. Create a transform that maps distorted points to undistorted points

Step-0: Load images used for calibration:

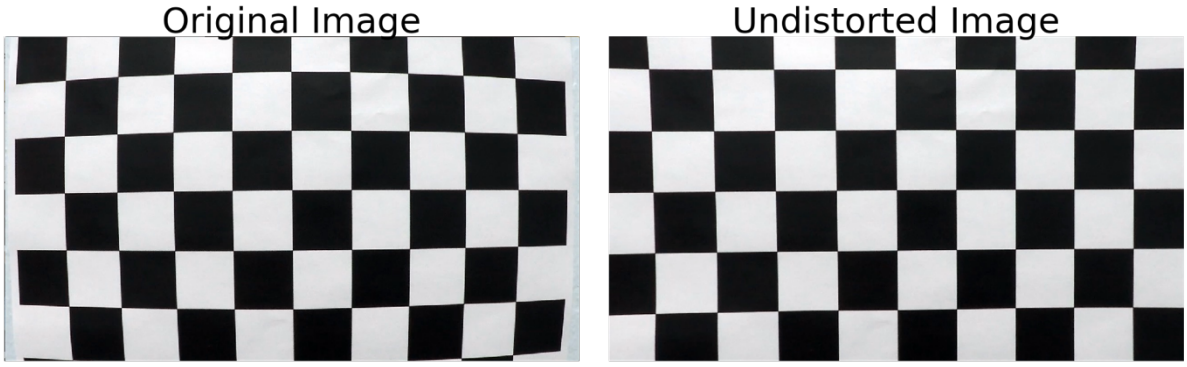
Step-1:
Map the coordinates of the corners of the 2D-calibration image [Image Points] to the 3D-coordinates of the real undistorted chessboard corners [Object Points]
Prepare 2 arrays to store `Image-Points` and `Object-Points`

Step-2:
Convert image to grayscale and apply `cv.findChessboardCorners()` function, that returns the corners coordinate from a calibration image. Assign this corners coordinates to `Image Points`
Repeat this for full collection of calibration images to improve calibration process accuracy

Step-3:

Calibrate camera using precalculated `Image Points` and `Object Points`
`ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera()` will return us 2 required parameters:
`mtx` - camera matrix and **`dist`** distortion coefficientes
Use **`cv2.undistort()`** to remove distortion from the image



Step-4:

Demonstrate Distortion removal on the Road conditions raw image:

Perspective transform

Goal: map the points in a given image to different, desired, image points with a new perspective.
Approach: make a "bird's-eye" view transform that will let us view a lane from above;

To make a perspective transform we are doing following assumptions:

1. Road is always relatively flat and there is no significant altitude change along the route
2. Camera position is fixed on the roof and located in the middle of the car width

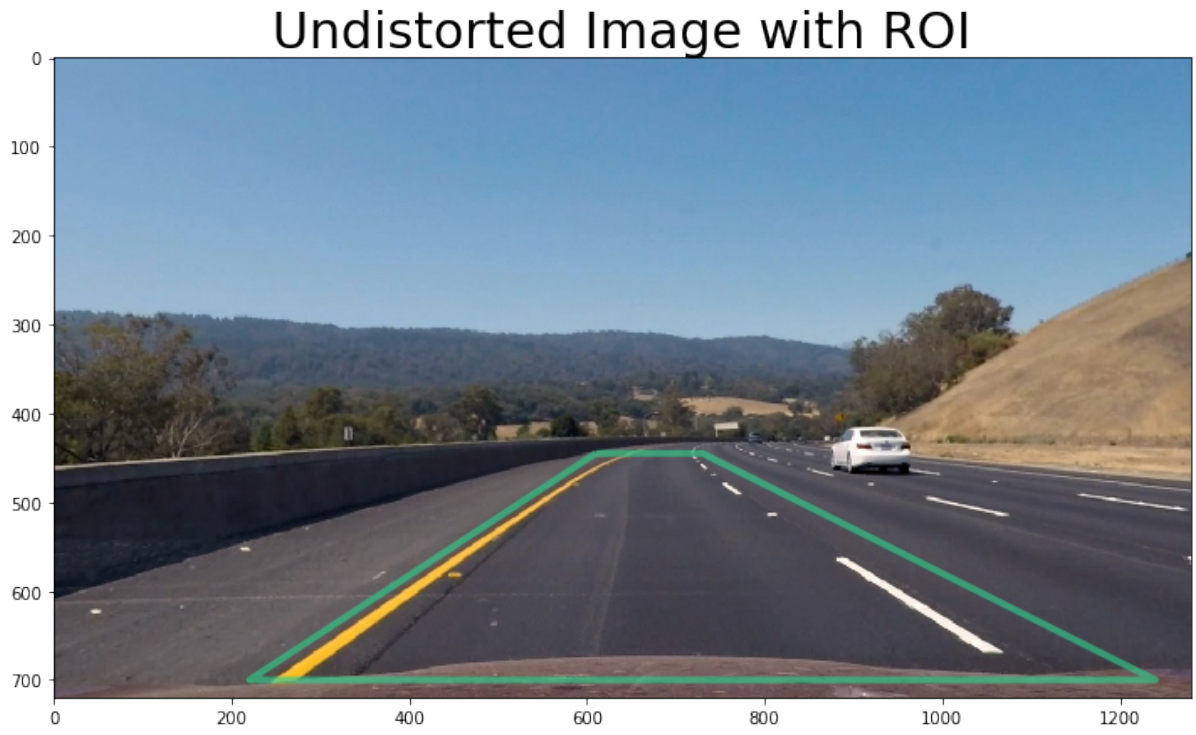
Steps to follow:

Step-0:

Define `warp()` function that will perform a `Perspective transform` for a desired image: using `cv2.warpPerspective()` function

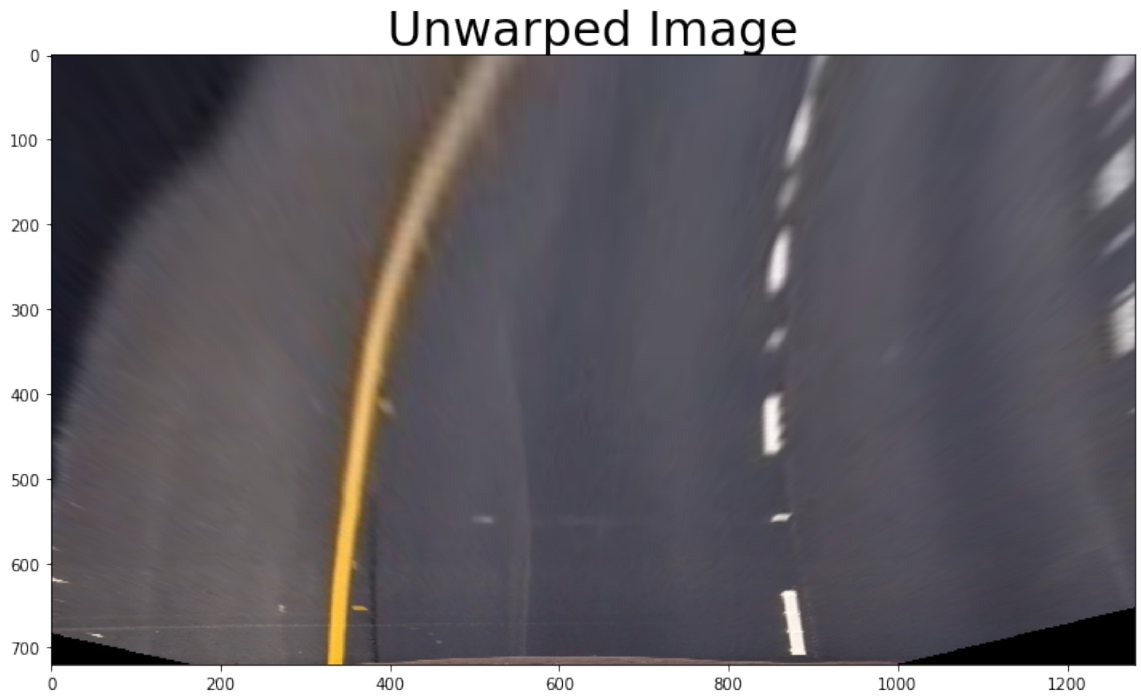
Step-1:

Define a Region of Interest `src[]`– a Trapezoid with 4 points **(In [9]: section of .ipynb notebook)**



Step-2:

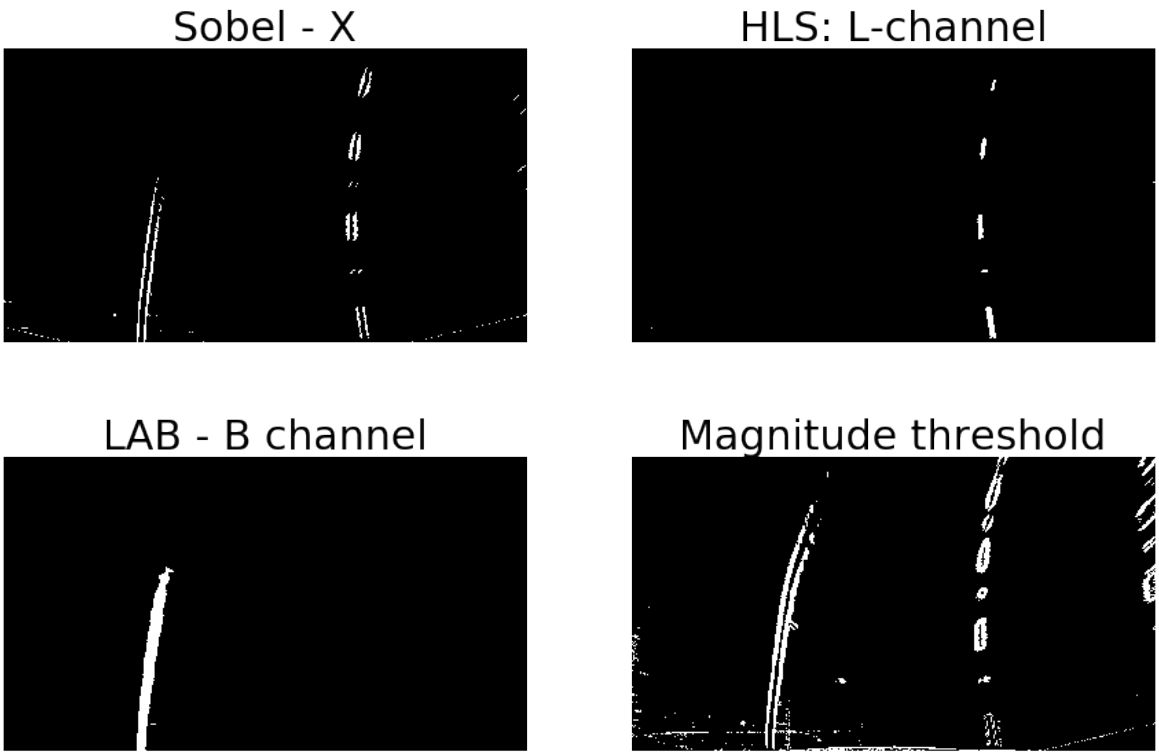
Select appropriate distanation points `dst []` in order to see good bird eye perspective of the image. And performe a warp transform:



Color and Directional Gradient transform

Goal: use gradients in a smarter way to detect steep edges that are more likely to be lanes - take individual derivatives/gradient along 1 direction X or Y.

Following 4 different thresholds were most promising to identify lines in given conditions



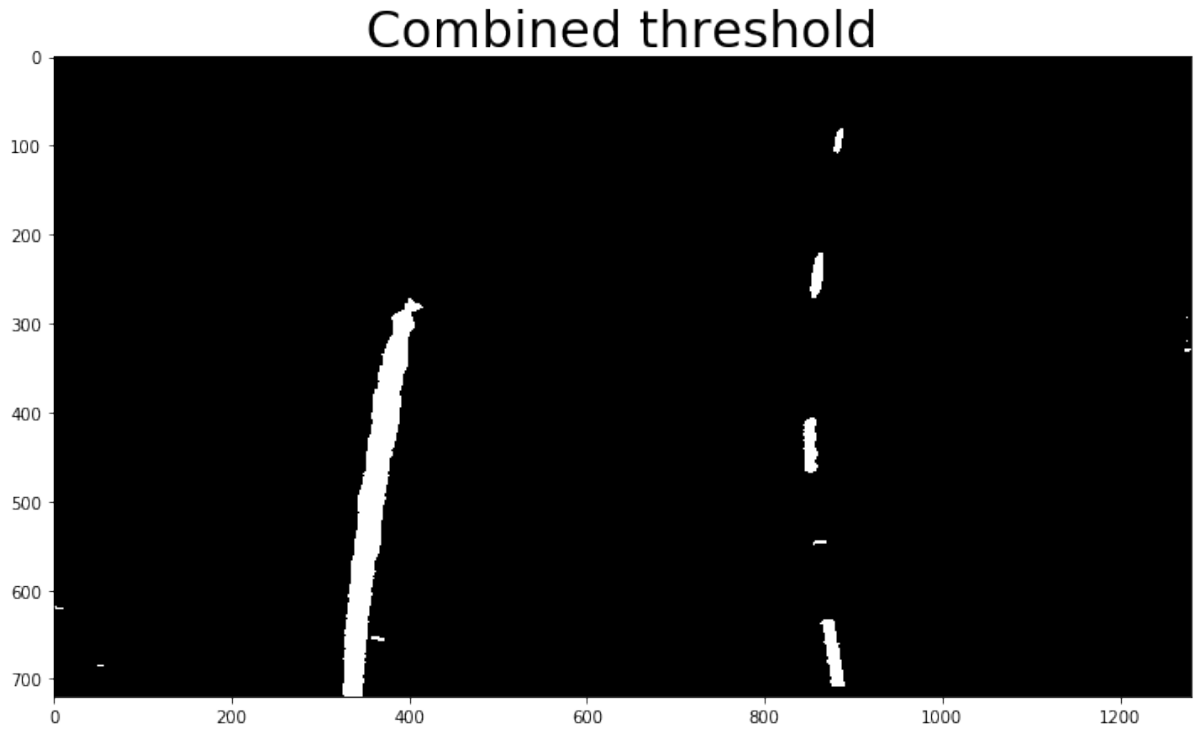
To generate a final Combination_threshold image a combination of the following 2 channels has shown to be most efficient:

1. LAB: B channel - naturally is working well identifyin yellow solid line
2. HLS: L cnannel – work nearly perfectly for white lines identification

This color transforming was one of the most chalanging parts of the project, I experienced following complications:

1. For given test images, after warping of the ROI often there is a neighbor car on one of the lanes that produces a lot of "ghost pixels" for Sobel-X directional gradient of Magnitude threschold
2. One has to work deeply on ROI selection and image warping before Greadient transforming to ensure a nice lane identification

Final Result for one of the test images looks like this:

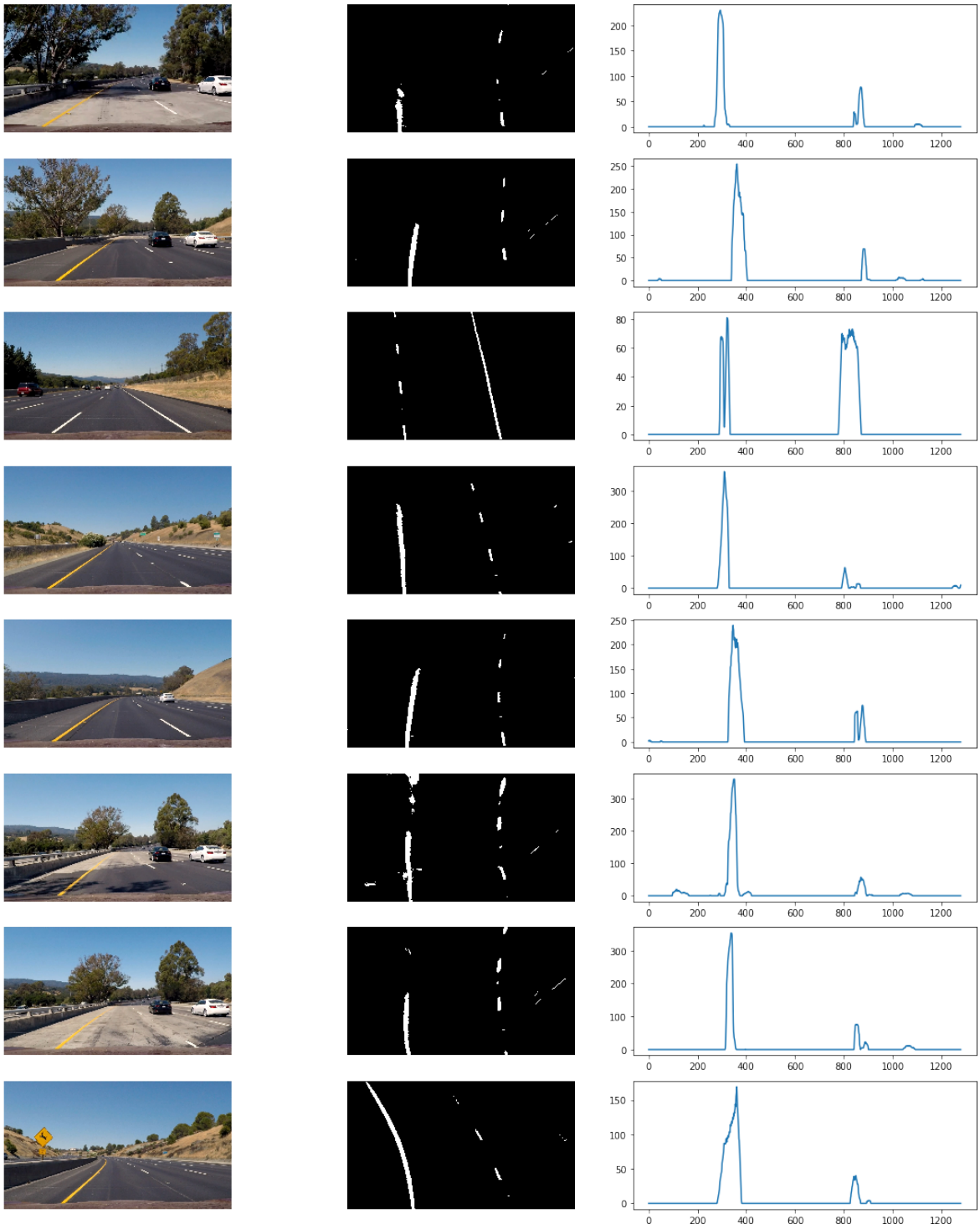


Identify Lane-Pixels using histogram

Applying a Histogram function along X direction (bins) of the binary image, we are able to identify 2 lanes and distinguish between them. Later these 2 peaks will be used to assign pixels to left or right late correspondingly:

```
In [18]: def Hist_calc(img):
             return np.sum(img[img.shape[0]//2:,:], axis=0)
```

The summary of how Histogram is applied to test images is shown below.



Implement sliding windows and Fit Polynomial

Now we use the two highest peaks from our histogram as a starting point for determining where the lane lines are, and then use sliding windows moving upward in the image (further along the road) to determine where the lane lines go.

Here we just acting according to the lesson suggested algorithm, ensuring each step provides required results:

Step-0:
The first step we'll take is to split the histogram into two sides, one for each lane line. This 2 peaks will be used as a starting positions fort he sliding windows

Step-1:
Identify the boundaries of the boxes and set up macroparameters (dimension oft he box, minimum amount of pixels to shift window)
For current position oft he windows identify:
Check what pixels „fall into“ the boundaries of specified boxes:
`good_left_inds` and `good_right_inds`

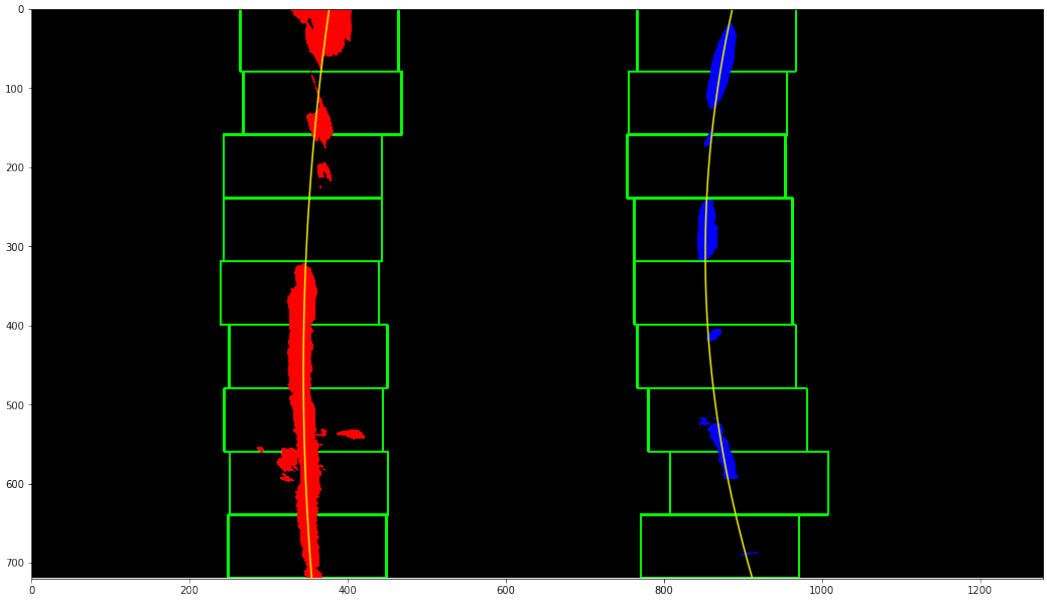
*(First iteration is on the bottom of the picture with a first windows for left and right lanes

Step-2:
Check for each lane if amount of points in the box is above threschold, if YES: update the value of `leftx_current` and `rightx_current` to reflect the changes oft he X position of the box – turning lane line
Add indexes oft he pixels that be lowng tot he left/right box tot he left/right lane correspondingly (`left_lane_inds` and `right_lane_inds`)

Iterate through steps 1 and 2 until reach the top oft he image. By the end of iteration, we should have: `left_lane_inds` and `right_lane_inds` that represent the full lanes.

Step-Final:
Fit second order polynomial using `np.polyfit()`

```
# Fit a second order polynomial to each
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)
```



Radius of curvature and distance from center calculation

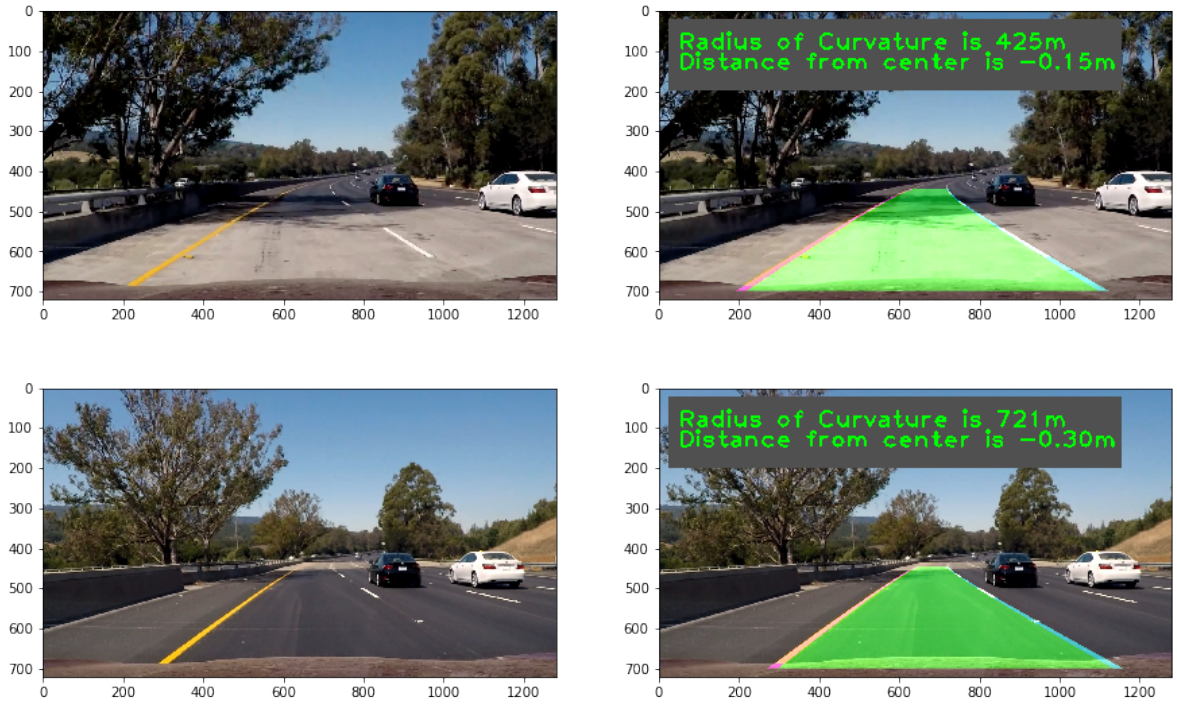
Implemented in `def calc_ROC()` function :

```
In [27]: def calc_ROC(binary_warped,left_fit,right_fit):
```

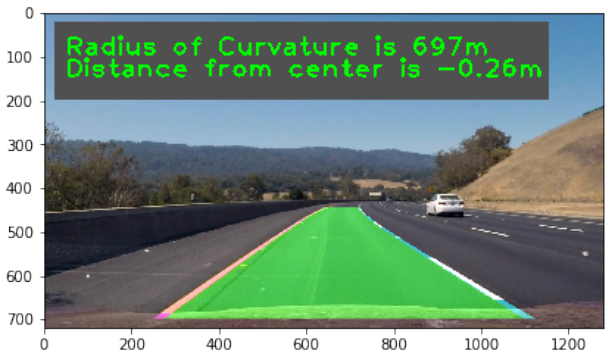
1. Based on lesson provided information we make an assumption regarding the pixel-to-meter transform
   *We could do this in detail by measuring out the physical lane in the field of view of the camera, but for this project, you can assume that if you're projecting a section of lane similar to the images above, the lane is about 30 meters long and 3.7 meters wide.

2. Assuming the car position is in the middle of the picture
3. Using conversion coefficients from (1) we can calculate the polynom function for left and right lane in the "Real World" coordinate space
4. Knowing polynom coefficients we can estimate separately for each lane a real radius of curvature.
5. To calculate distance from center we need to once again use polynom coefficients from left and right lanes – this time just an "intercept" coefficient of both lines. Find mean of these values and subtract it from middle of the image taking into account assumption (2)
6. Mean of the radius of curvature for left and right lane assumed to be a final radius of curvature

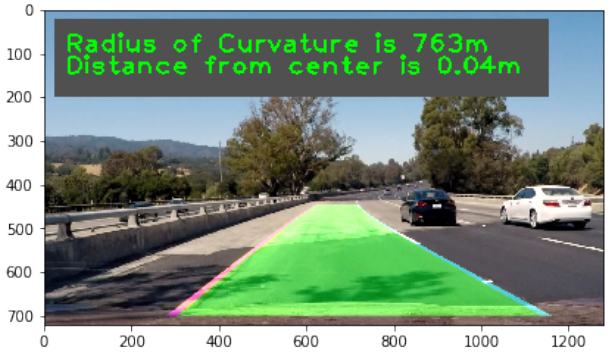
Test of the complete Algorithm on the test images:



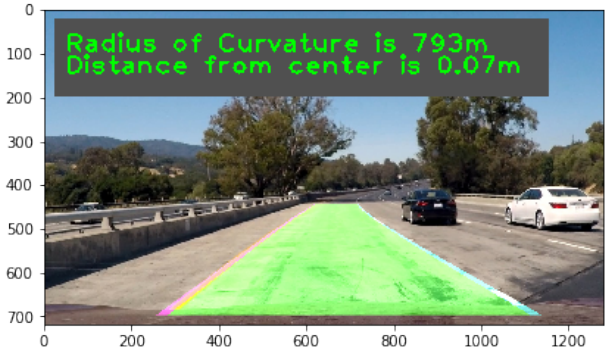

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature is 697m
Distance from center is -0.26m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

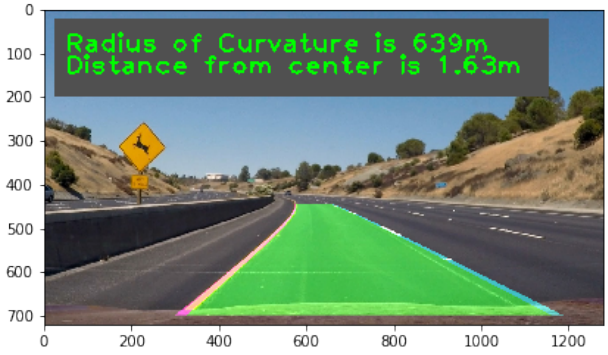

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature is 763m
Distance from center is 0.04m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature is 793m
Distance from center is 0.07m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature is 639m
Distance from center is 1.63m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Project video evaluation

Evaluated video is located in the project folder with filename: **project_video_output.mp4**