Files: lane.ipynb; result.mp4, challenge_video.mp4; my test code lecturenotes.ipynb.

The goals / steps of this project are the following:

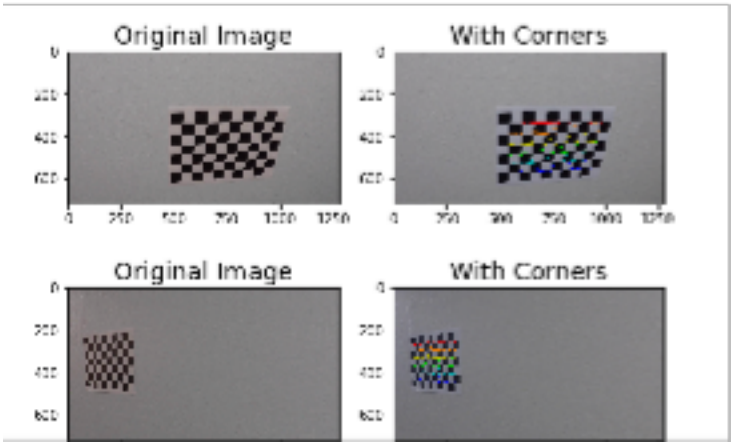
• Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

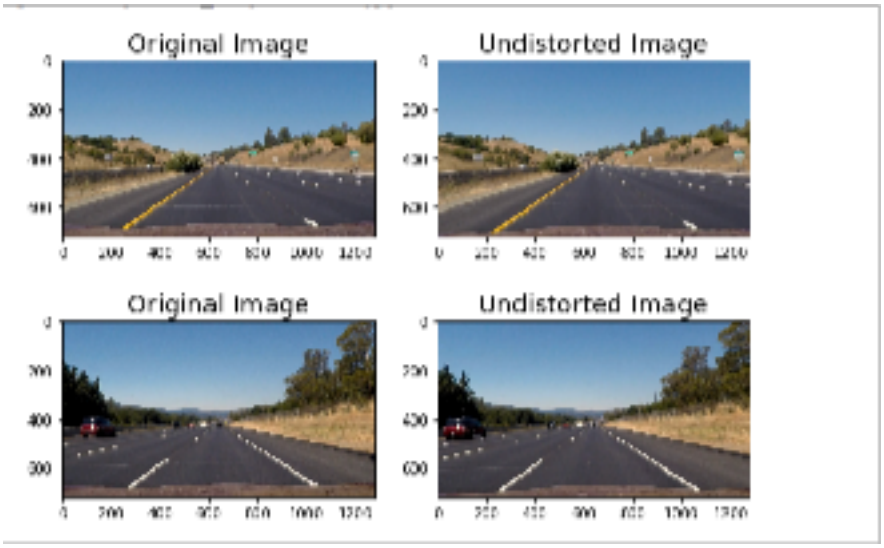
Camera calibration uses the provided checkerboard calibration images using findCheckerboardCorners and matches them with a square grid. This calculates a perspective transform matrix which can be used to correct for radial distortion. There are 20 images of which 2 are shown below. The rest are shown in the ipython notebook.

• Apply a distortion correction to raw images.

The distortion corrections are computed using cv2.calibrateCamera with datapoints computed from the camera calibration step above.

The distortion correction is are applied to 6 test images of which 2 are shown below. The rest are shown in the ipython notebook.

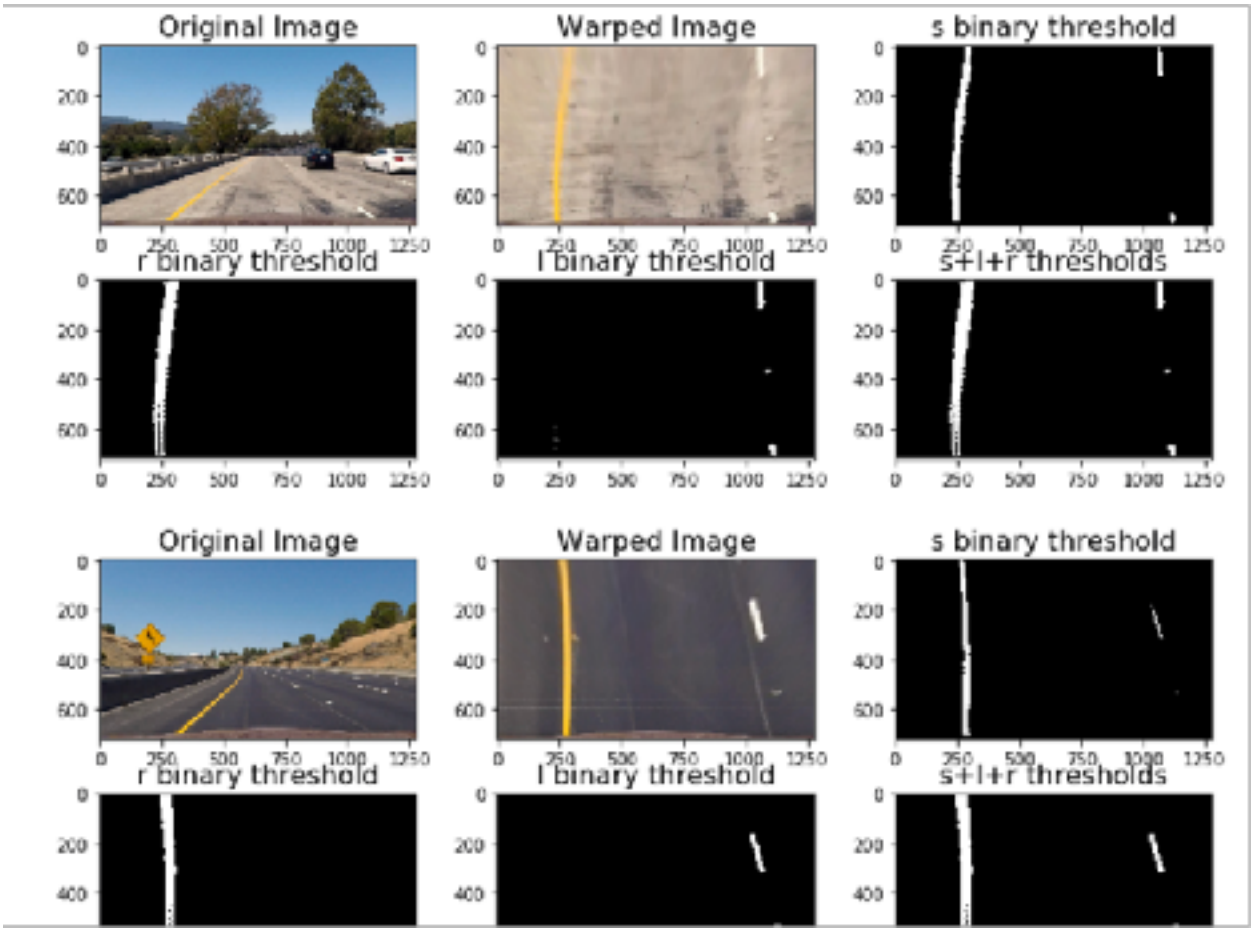




- **Use color transforms, gradients, etc., to create a thresholded binary image.**

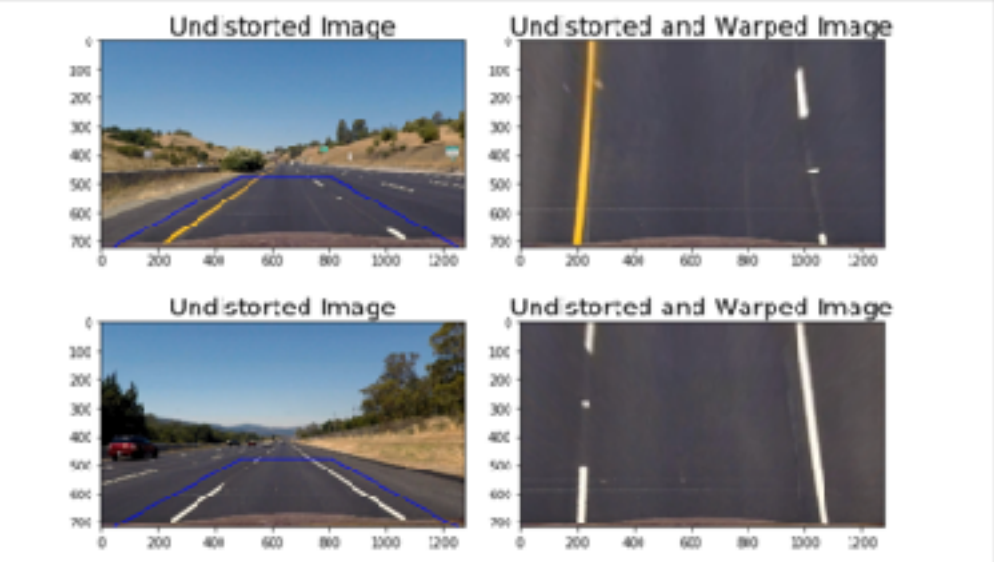
The lecture shows a combination of gradients + color thresholding for detecting edges. A combination of x and y gradients would work for detecting edges and the color thresholding to recognize the lane coloring. I found the gradients to not be necessary. The gradient code is in a separate notebook; lecturenotes. Gradient code would be more useful for sharper curves where the magnitude would be more important than a horizontal or vertical gradient. Blending R and L channels was enough using the test images as a metric. This assumes the 6 test images are representative of the video input. The output of 2 test images are shown below; the rest are in the notebook. The key to determining which are sufficient is to compare visually which combination of thresholded images matches the original without

removing data which would cause the curvature calculation to be incorrect.



- **Apply a perspective transform to rectify binary image ("birds-eye view").**

The birds eye view is calculated using the ROI shown in blue in the test images below. The ROI is supposed to bound the lane lines as in the first assignment on lane lines. The unwarping is done by assigning points from the ROI to a square which is smaller than the display window. The BEV is shown in the images below; the rest are in the notebook.

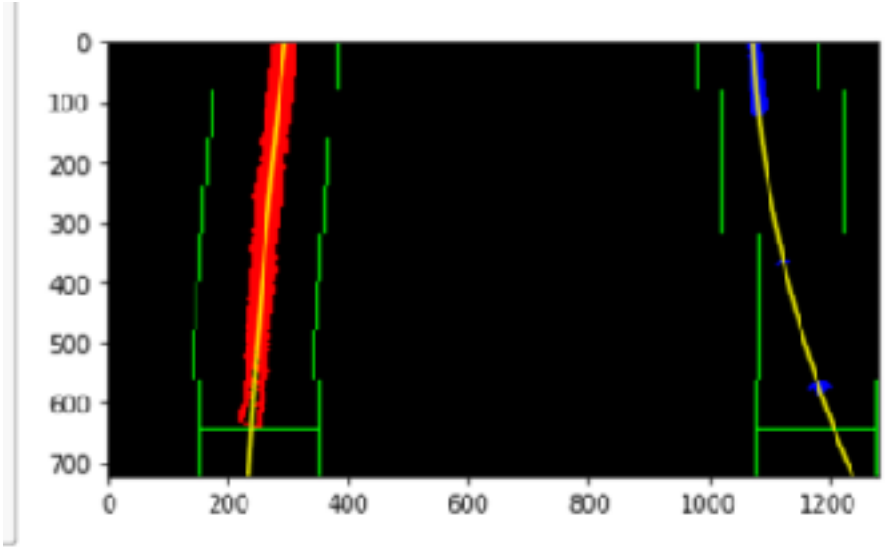


Detect lane pixels and fit to find the lane boundary.

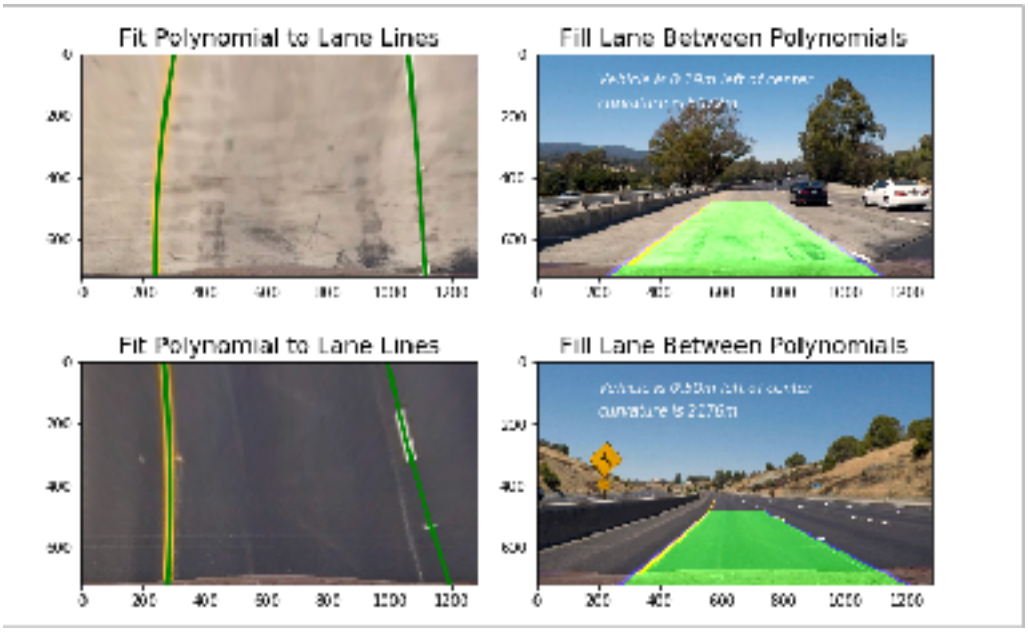
Detecting lane pixels follows the lecture code; where we

1. Convert the transformed image using np.nonzero() to binary and consider bottom half of image. Calculate the midpoint of the image and find the max to the right and left of the midpoint using a histogram to find the left and right lanes using np.sum on the right and left vertical halves of the image. np.sum is a continuous histogram vs np.histogram which generates bins.. These left and right max values are the starting position for the left and right lanes.

2. use a sliding window to find pixels in the left and right sides of the image. Set the margin to 100, define the x coordinates of the right and left sliding windows to have a +/- 100px margin from the center of the left/right lanes. This is the nonzero x pixel value from the left and right values from the histogram. Return the indexes where there are pixels for the left and right lane markings in the horizonal window and vertical window size for each of the 9 windows. This method assumes the window size is small enough to deal with sharp curvatures and the lanes are inside the windows defined by these hard coded parameters.



3. Uses lane centering to move the search window to the middle. The risk with this approach is it assumes the lanes are within the hard coded margins. This works great for the sample demo but for cases like U-turns or lane changing scenarios a different algorithm may be required. Test cases shown below; rest in notebook. Code copied from lecture examples included the window search code as find_lecture which returns leftx, lefty, rightx,righty pixel positions.



Determine the curvature of the lane and vehicle position with respect to center.

The curvature is calculated from the polynomial fit. once we have the pixels from the left and right lanes in the step above. The coefficients from the polyfit are used to draw the curve onto the warped space image. The radius of curvature is taken from lecture code with a given constant for pixels in warped space to real world meter dimensions. Another polyfit is applied to get values of left and right cuvatures. The vehicle position is determined by defining a center which is the average of the x_right and x_left lanes and figuring out if this average is to teh right or left of the center of the frame; splitting lhe x coordinate in half from the left/right x coordinate. The results of the curvature and vehicle position left/right from center are show above. The rest of the test images are in the notebook.

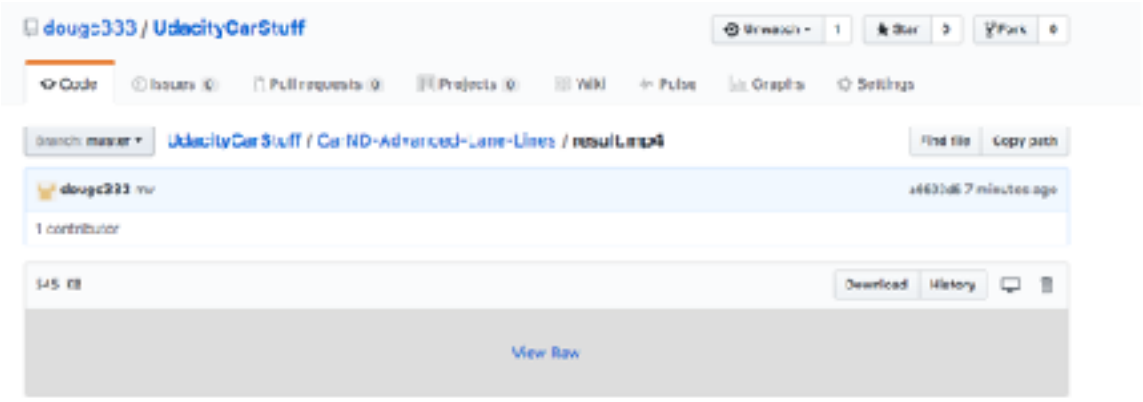
Warp the detected lane boundaries back onto the original image.

The distortion transform matrix from the step after the camera calibration is used to reverse the transform. Use another polyfit in the undistorted space to draw the lane lines.

Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

The video is calculated and the link is under result.mp4 and challenge_result.mp4.

Note: when downloading the video files from github, result.mp4; cannot download as raw you have to click on the download button on the page with the video as shown below:



Doing a save link as will download a mp4 file but the OS cannot read the file.