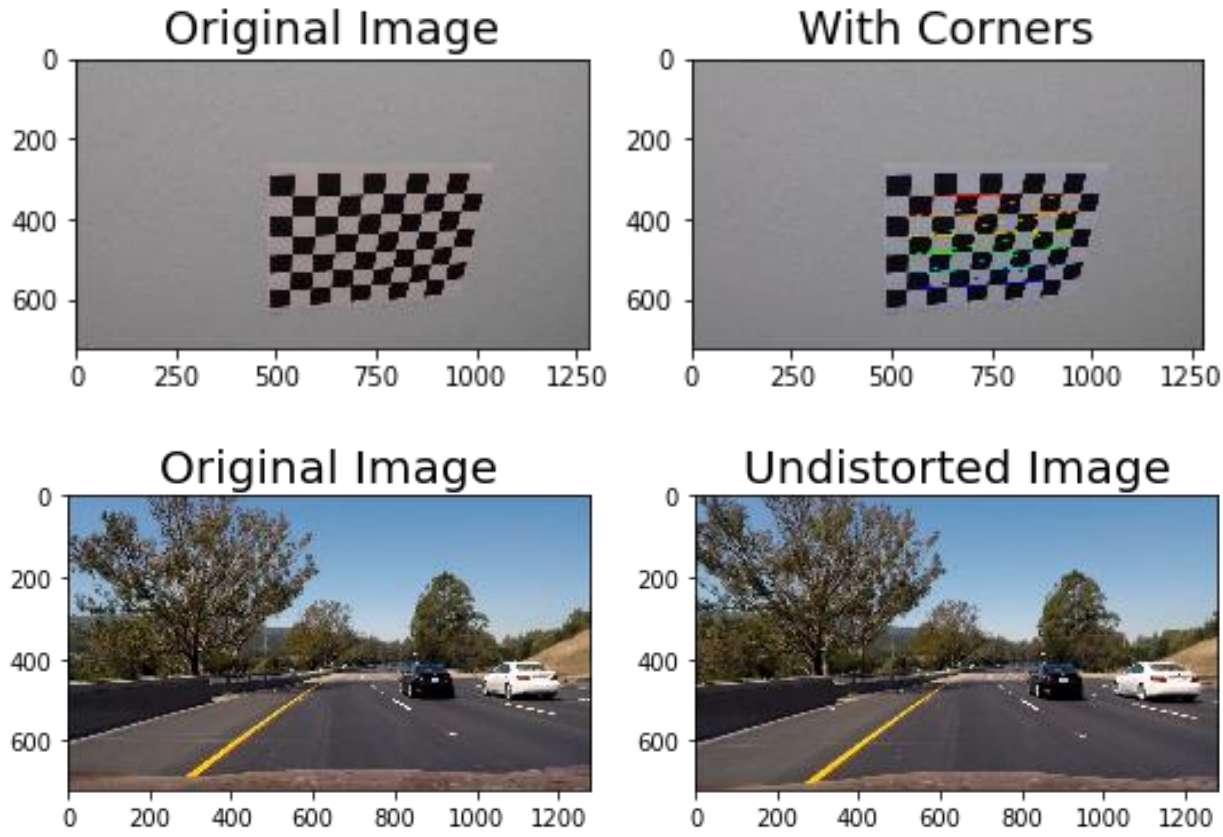
Advanced Lane Finding Project

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.

STEP 1: Camera Calibration And Distortion Correction

The first step in the project is to remove any distortion from the images by calculating the camera calibration matrix and distortion coefficients using a series of images of a chessboard.
I start by preparing 'object points' and 'ímage points'. "objpoints" are the 3D points In real world space and "imgpoints" are 2D points in image plane.
Next I will define a function `undistort()` which uses the calculate camera calibration matrix and distortion coefficients to remove distortion from an image and output the undistorted image.



Step 2: Perspective Transform

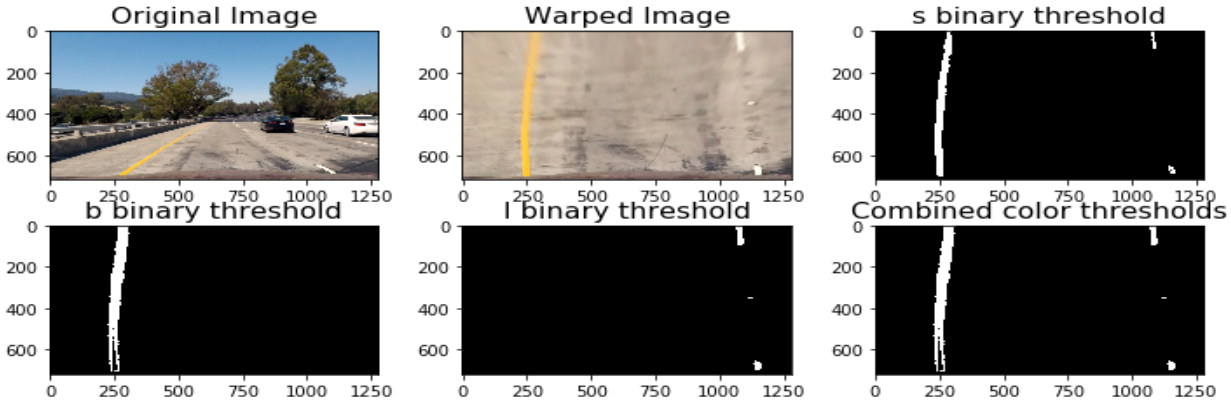
In this step I will define a function `birds_eye()` which transforms the undistorted image to a "birds eye view" of the road which focuses only on the lane lines and displays them in such a way that they appear to be relatively parallel to eachother. This will make it easier later on to fit polynomials to the lane lines and measure the curvature.

Step 3: Apply Binary Thresholds

In this step I attempted to convert the warped image to different color spaces and create binary thresholded images which highlight only the lane lines and ignore everything else. I found that the following color channels and thresholds did a good job of identifying the lane lines in the provided test images:

- The S Channel from the HLS color space, with a min threshold of 180 and a max threshold of 255, did a fairly good job of identifying both the white and yellow lane lines, but did not pick up 100% of the pixels in either one, and had a tendency to get distracted by shadows on the road.
- The L Channel from the LUV color space, with a min threshold of 225 and a max threshold of 255, did an almost perfect job of picking up the white lane lines, but completely ignored the yellow lines.
- The B channel from the Lab color space, with a min threshold of 155 and an upper threshold of 200, did a better job than the S channel in identifying the yellow lines, but completely ignored the white lines.

I chose to create a combined binary threshold based on the three above mentioned binary thresholds, to create one combination thresholded image which does a great job of highlighting almost all of the white and yellow lane lines.

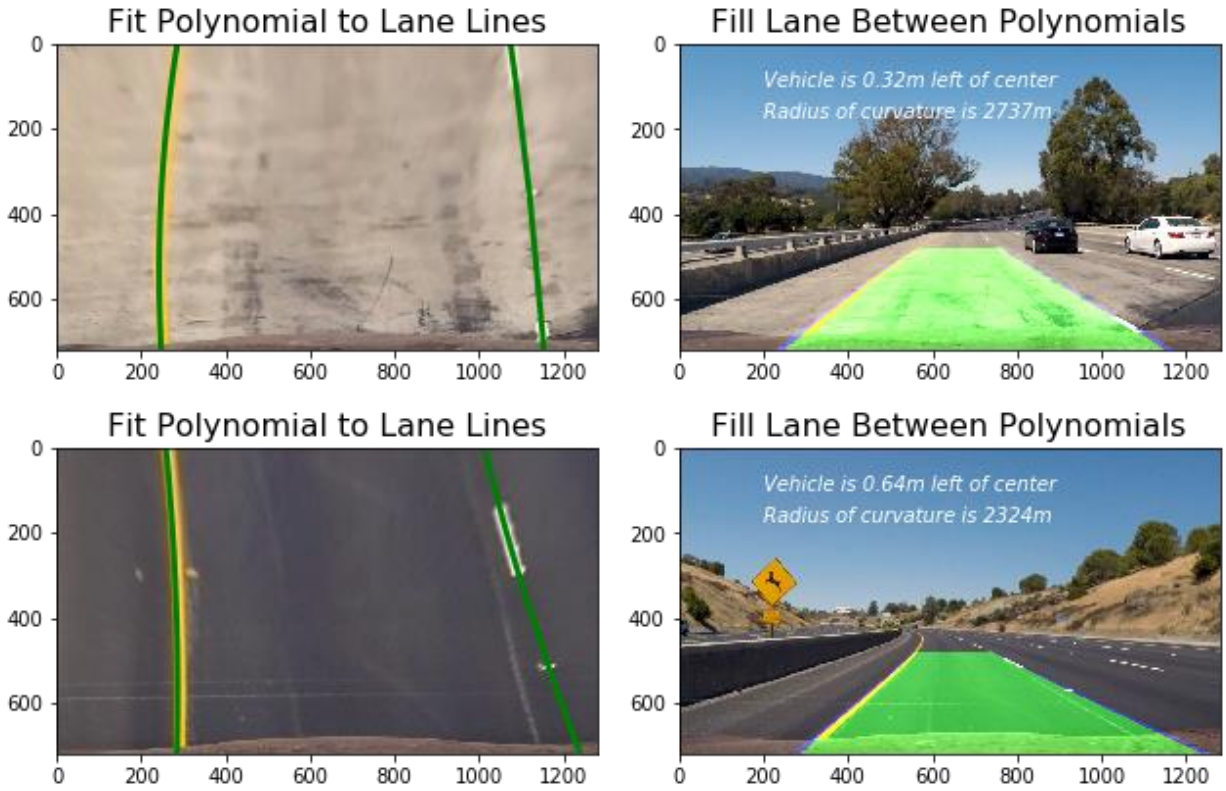


Steps 4, 5 and 6: Fitting a polynomial to the lane lines, calculating vehicle position and radius of curvature:

At this point I was able to use the combined binary image to isolate lane line pixels and fit a polynomial to each of the lane lines. The space in between the identified lane lines is filled in to highlight the driveable area in the lane. The position of the vehicle was measured by taking the average of the x intercepts of each line.

In the function `fill_lane()` below, lane lines are detected by identifying peaks in a histogram of the image and detecting nonzero pixels in close proximity to the peaks.

Step 7: Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position



Video Processing Pipeline:

After establishing a pipeline to process still images, the final step was to expand the pipeline to process videos frame-by-frame, to simulate what it would be like to process an image stream in real time on an actual vehicle.

My goal in developing a video processing pipeline was to create as smooth of an output as possible. To achieve this, I created a class for each of the left and right lane lines and stored features of each lane for averaging across frames.

The video pipeline first checks whether or not the lane was detected in the previous frame. If it was, then it only checks for lane pixels in close proximity to the polynomial calculated in the previous frame. This way, the pipeline does not need to scan the entire image, and the pixels detected have a high

confidence of belonging to the lane line because they are based on the location of the lane in the previous frame.

If at any time, the pipeline fails to detect lane pixels based on the the previous frame, it will go back in to blind search mode and scan the entire binary image for nonzero pixels to represent the lanes.

In order to make the output smooth I chose to average the coefficients of the polynomials for each lane line over a span of 10 frames. The gif below is the result of my pipeline running on the test video provided for the project, as well as an optional challenge video which presented additional challenges to the lane detection pipeline.

PROJECT VIDEO:

DISCUSSIONS

The video pipeline developed in this project did a fairly robust job of detecting the lane lines in the test video provided for the project, which shows a road in basically ideal conditions, with fairly distinct lane lines, and on a clear day. It also did a decent job with the challenge video, although it did lose the lane lines momentarily when there was heavy shadow over the road from an overpass.

What I have learned from this project is that it is relatively easy to finetune a software pipeline to work well for consistent road and weather conditions, but what is challenging is finding a single combination which produces the same quality result in any condition. I have not yet tested the pipeline on additional video streams which could challenge the pipeline with varying lighting and weather conditions, road quality, faded lane lines, and different types of driving like lane shifts, passing, and exiting a highway. For further research I plan to record some additional video streams of my own driving in various conditions and continue to refine my pipeline to work in more varied environments.