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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Camera Calibration

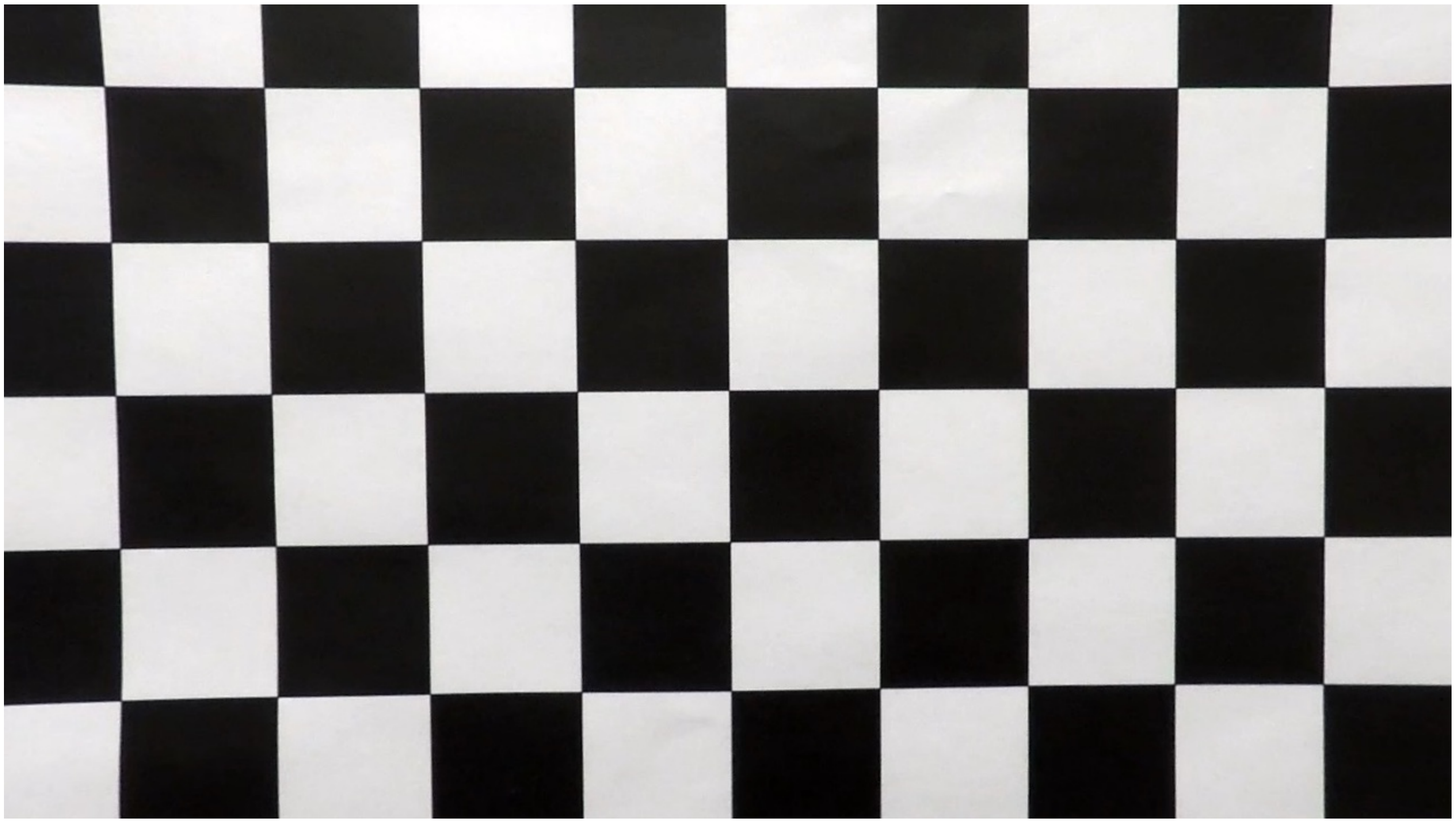
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the file `calibrate_camera.py` .

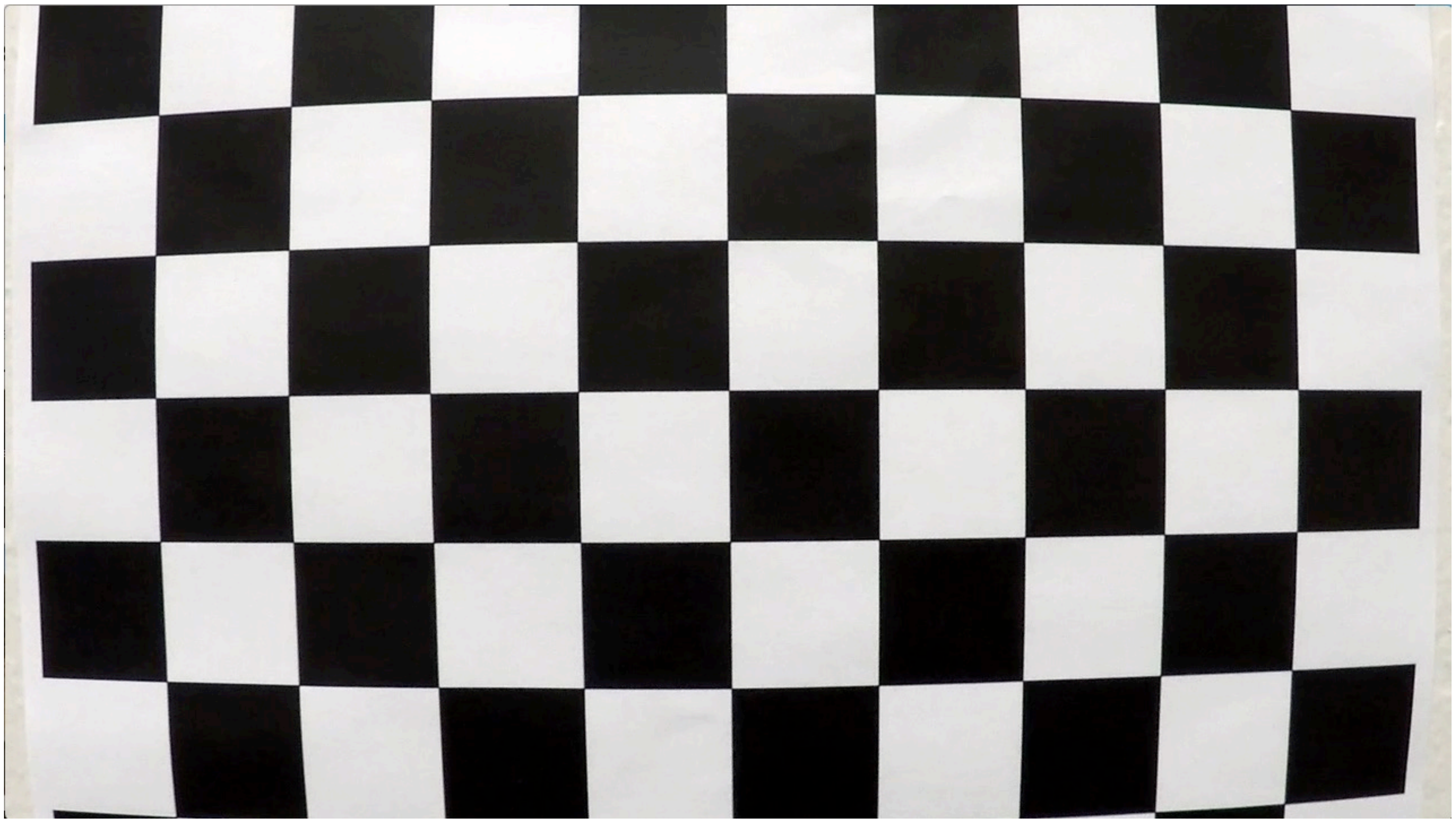
In the constructor of the CameraCalibration class, I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `self.objp` is just a replicated array of coordinates, and `self.obj_points` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image (see line 43). `self.img_points` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection (found using the `cv2.findChessboardCorners` function).

I then used the `self.obj_points` and `self.img_points` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. The camera matrix and the distortion coefficients thus computed are stored as object member variables and stored in a pickle file to be used later on the project test images/video frames.

I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



for the first calibration image given below:

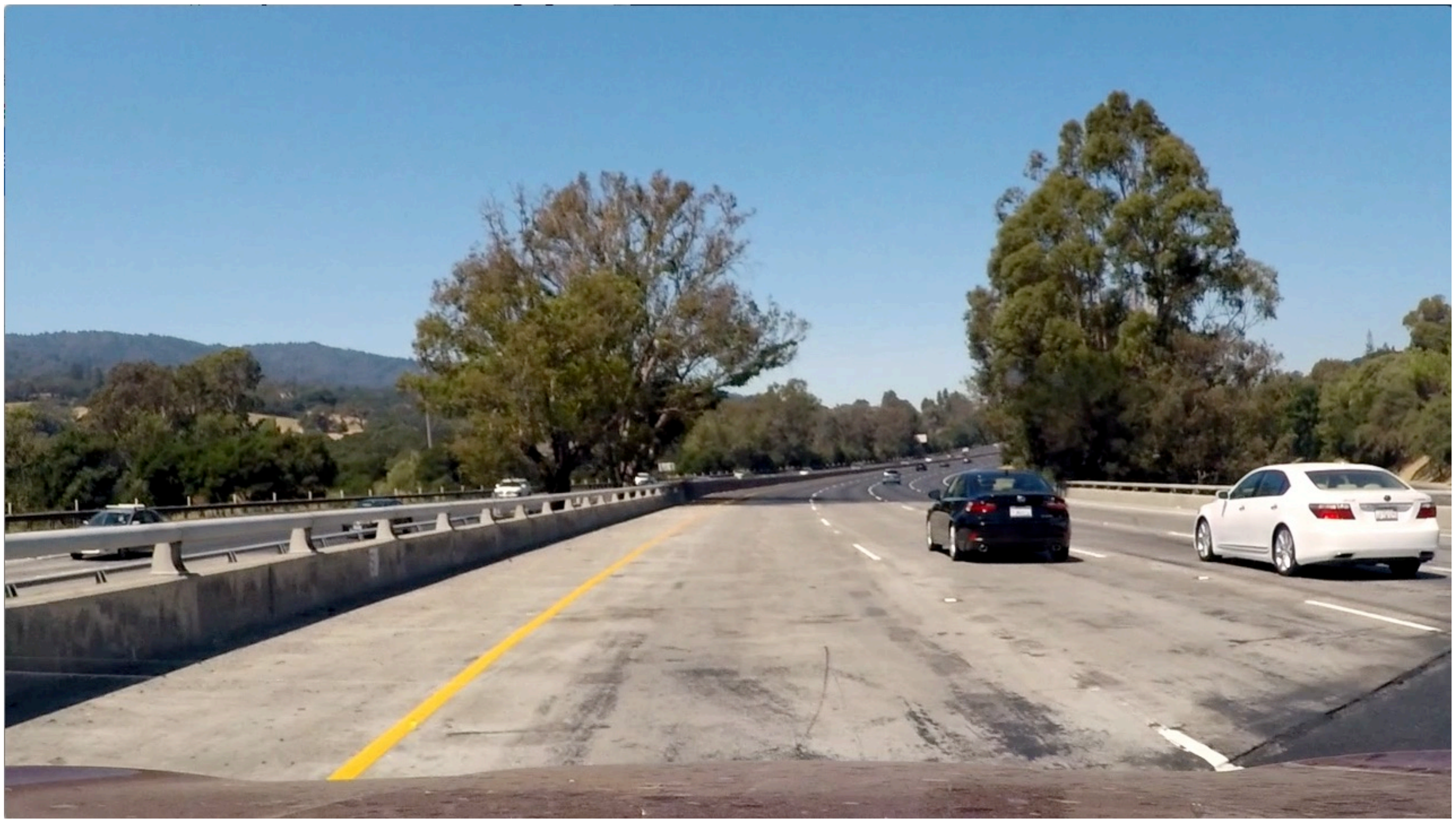


Please run `calibrate_camera.py` for reproducing these results.

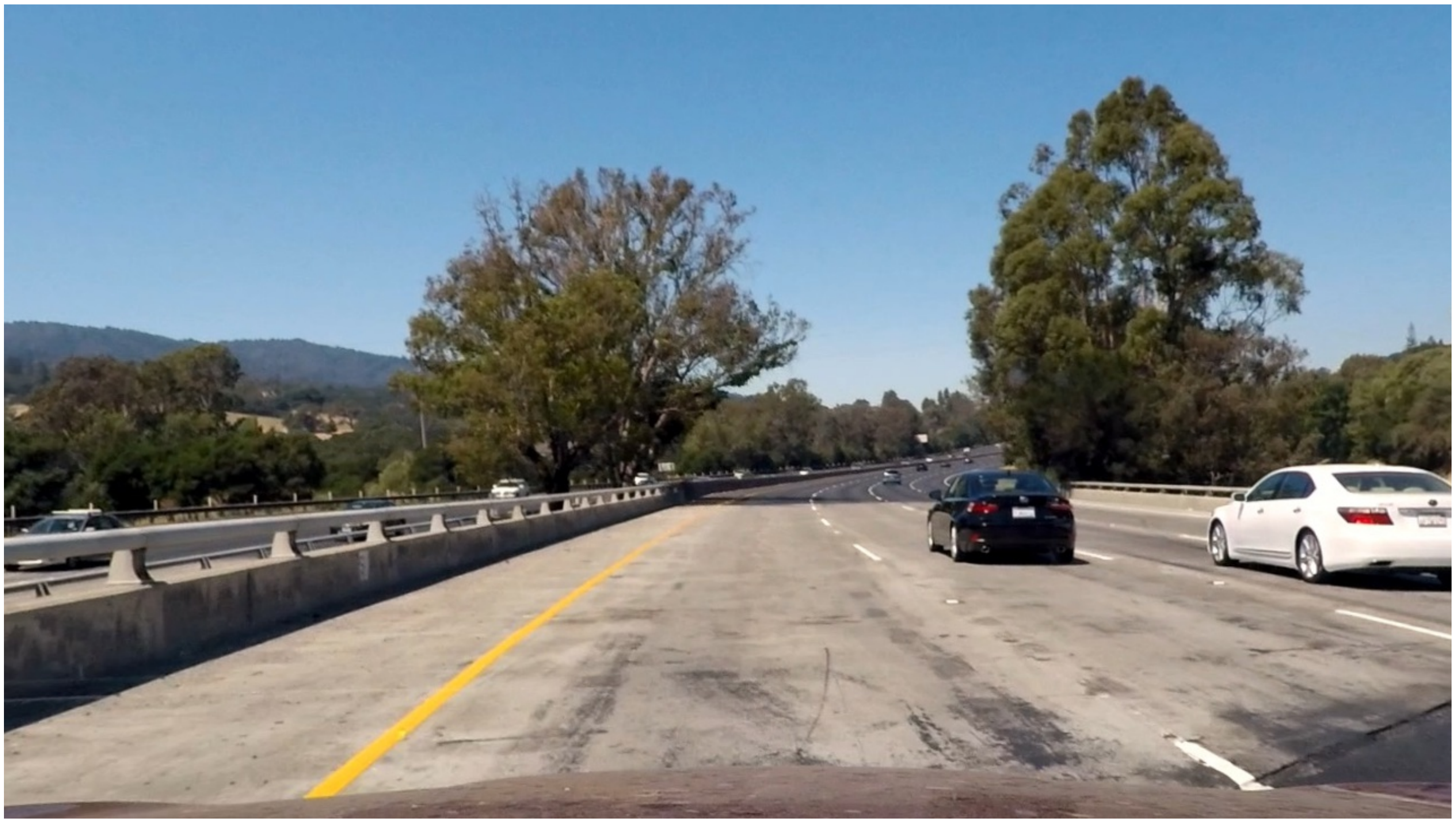
Pipeline (single images)

1. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:



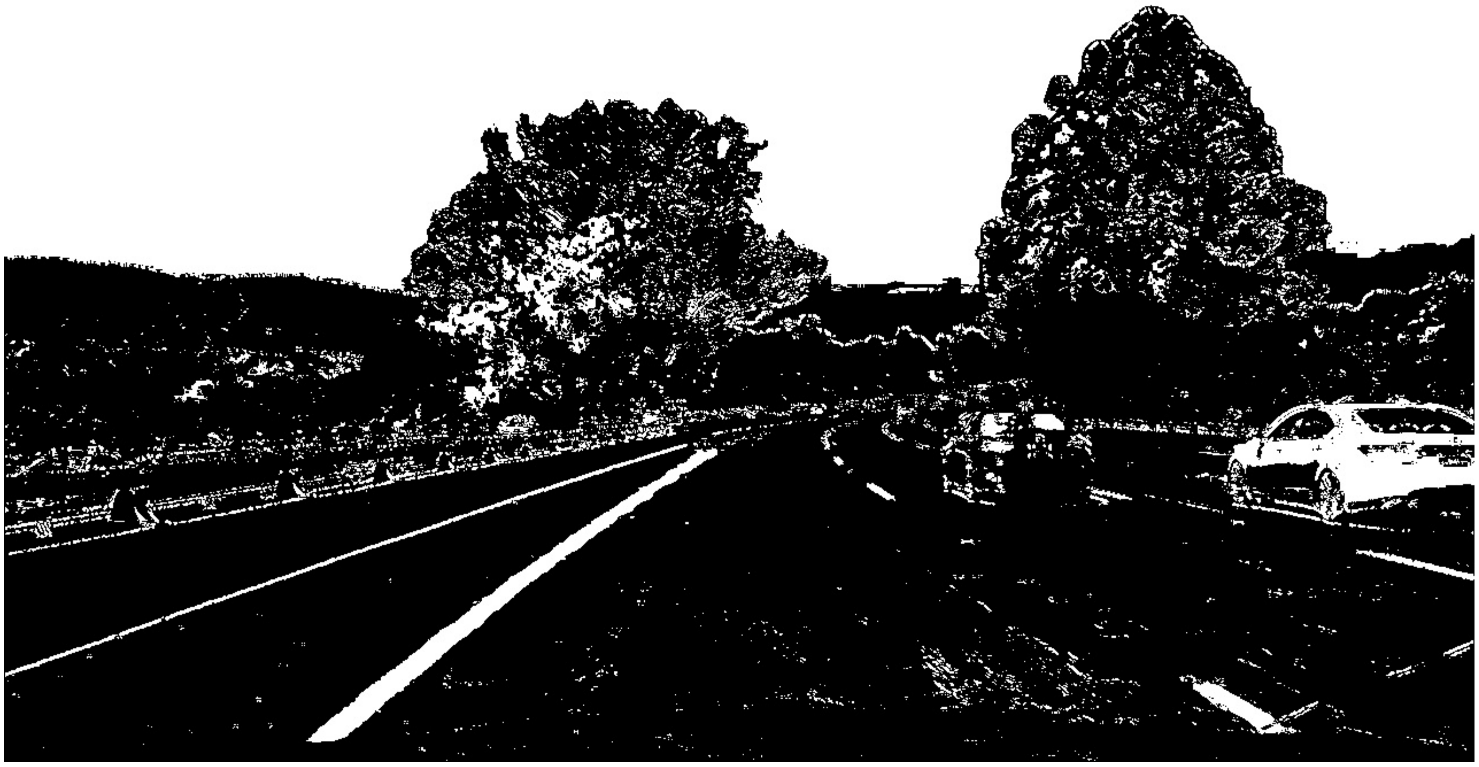
The corresponding undistorted image is as shown below:



2. Describe how (and identify where in your code) you used color transforms, gradients or

other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image. The gradient thresholding is implemented by `abs_sobel_thresh()` function while color thresholding is performed by `hls_thresh()` for both HSV and HLS images on the C and S channels, respectively. These function's are defined in `utils.py`. Here's an example of my output for this step (For the same test image as shown above).



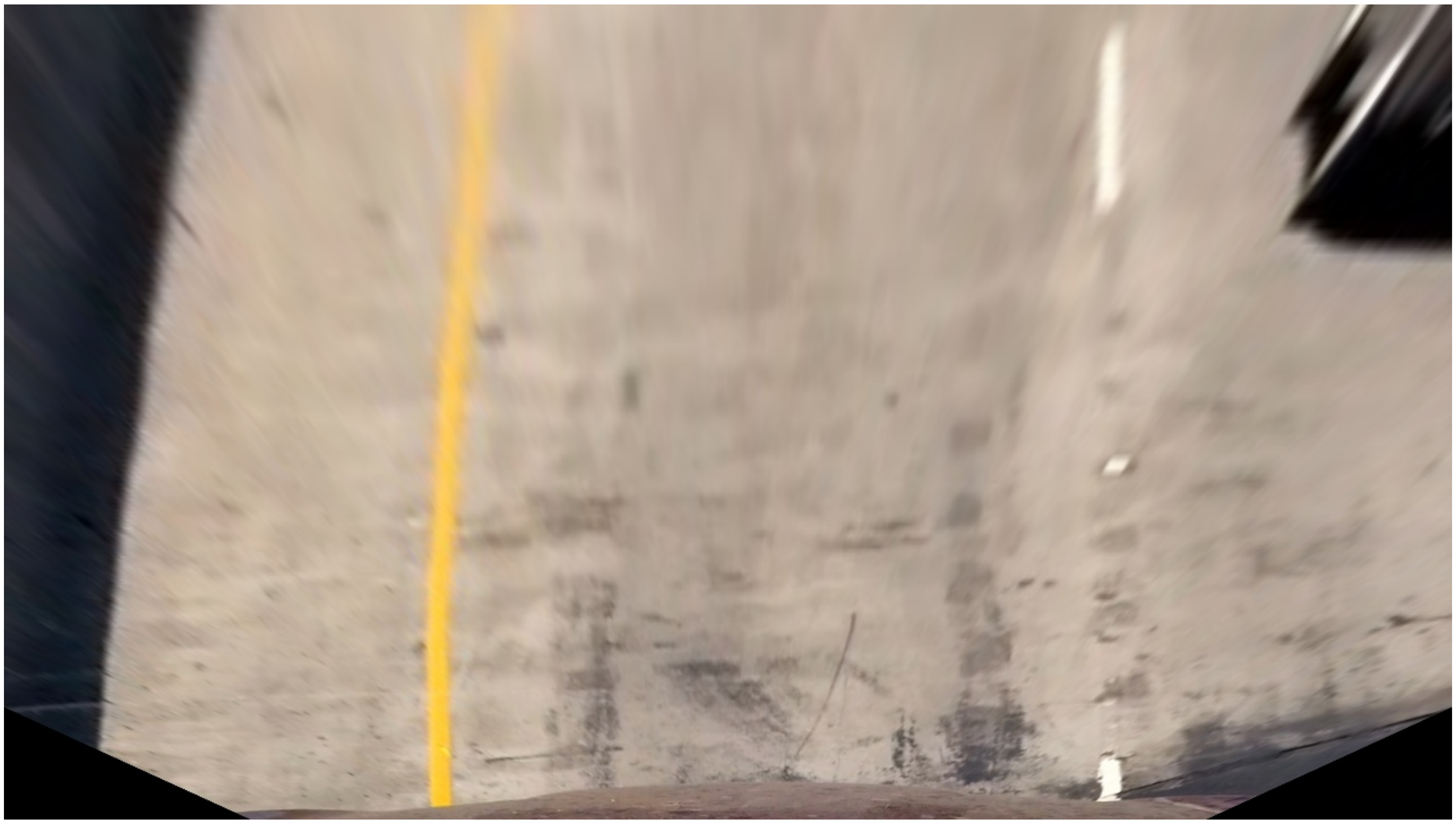
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `warp_image()`, which appears in lines 45 through 66 in the file `utils.py`. The `warp_image()` function takes as inputs an image (`img`), and hardcodes the source (`src`) and destination (`dst`) points. Please see `utils.py` for values of the hardcoded src and dst points.

This resulted in the following source and destination points:

Source	Destination
(588, 446)	(320, 0)
(691, 446)	(960, 0)
(1126, 719)	(960, 720)
(153, 719)	(320, 720)

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image. The following shows the warped image produced from the test image example as shown in the previous three figures.



4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The `LaneFinder` and `Tracker` classes in `find_lanes.py` and `tracker.py` identifies the lane line pixels and fits a second order polynomial for the left and right lane pixels.

The `Tracker` class is responsible for identifying the lane line pixels and sliding a rectangular window so its centroid coincides with the respective lane centers. First, the lane line positions are identified at the bottom of the image by thresholding the pixels in the binary warped image separately in the left and right halfs of the image (for the bottom 1/4 of the image to make it more robust to curves in the lanes) and then finding the x position where the maximum value of the histogram occurs. Thereafter, two separate sliding windows (for left and right lanes) are used to maintain the `x` positions of the R/L lanes as we move in the `y` direction. The pixels in the window are convolved with a one dimensional array of length equal to the width of the window to find the peak in the `local histograms`. The window height `window_h` dictates how many such windows we use in the vertical direction. A smoothing factor is used to smoothen the lane line curves across the successive frames of a video.

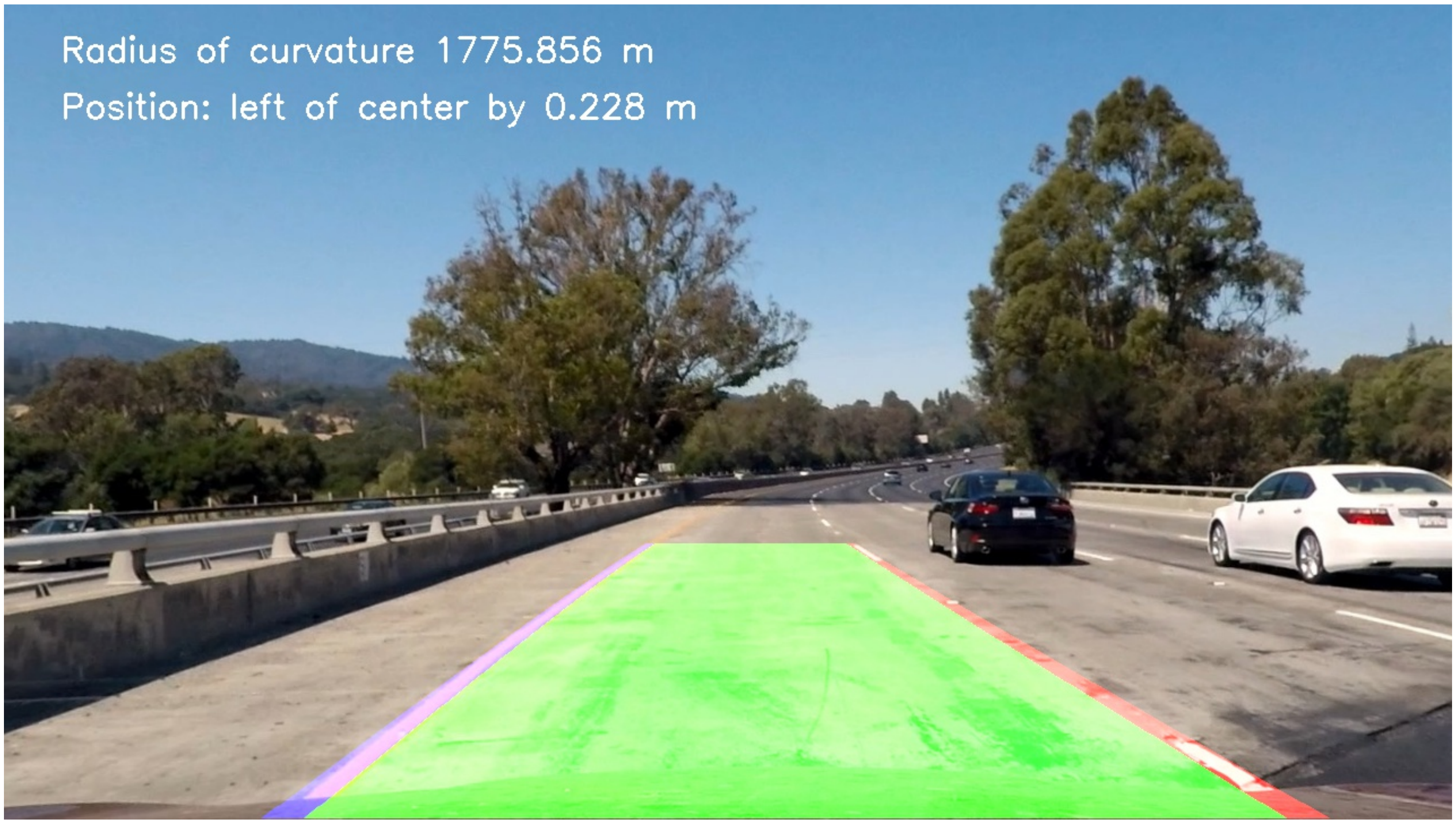
The polynomial fit is then computed using `numpy.polyfit` function in `find_lanes.py` lines 54 through 61.

5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I did this in lines # through # in my code in `my_other_file.py`

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in lines 83 through 90 in my code in `find_lanes.py` in the function `process_image()`. Here is an example of my result on the same test image above:


Radius of curvature 1775.856 m
Position: left of center by 0.228 m

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Here's a link to my video result It is computed by file `video.py`.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

As I observe the results of my pipeline on the project video, I observed that the results are a bit shaky when the frame contains shadows of trees and other vehicles passing nearby. I would try to improve these results further by playing with the smoothing factor between frames (currently at 15) and see if I get better results. Another idea I have is to somehow use the info that lane lines (on highways at least) are at a standard distance from each other, that should reduce the number of spurious pixels detected as lane lines. It would also be interesting to explore a deep learning based approach to this problem - although the problem formulation may be challenging in that case.

Also, I would like to improve the results on the challenge video, where there is a lot of glare from the sun in the window and the roads are prettying winding. The histogram approach to find the intial positions of the lanes may be improved by reducing the `y` range where we take the histogram in the bottom of the warped image. Also different thresholding values and color channels to reduce the effect of the glare are worth exploring.