Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

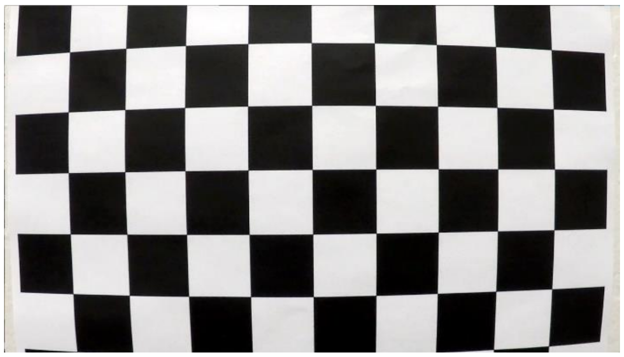
Camera Calibration

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

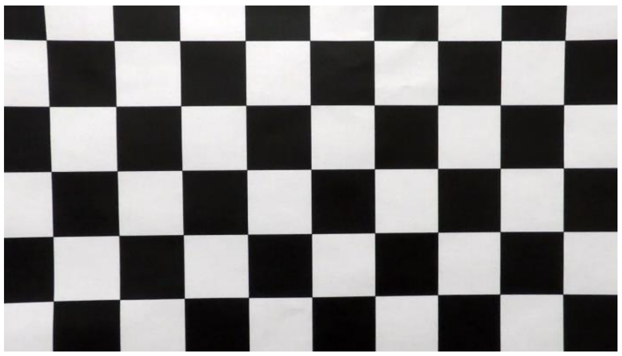
The file for camera calibration is cam_cal.py.

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. There are 6 rows and 9 columns of corners that I would like to identify. Then I used the cv2.findChessboardCorners function to identify the corners. If the corners were detected, those will be added into the imgpoints array. I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result listed below. There are three images that were not able to have corners identified due to cutting edges on some of the grids so only 17/20 have corners fully identified.

Original Image



Undistorted Image



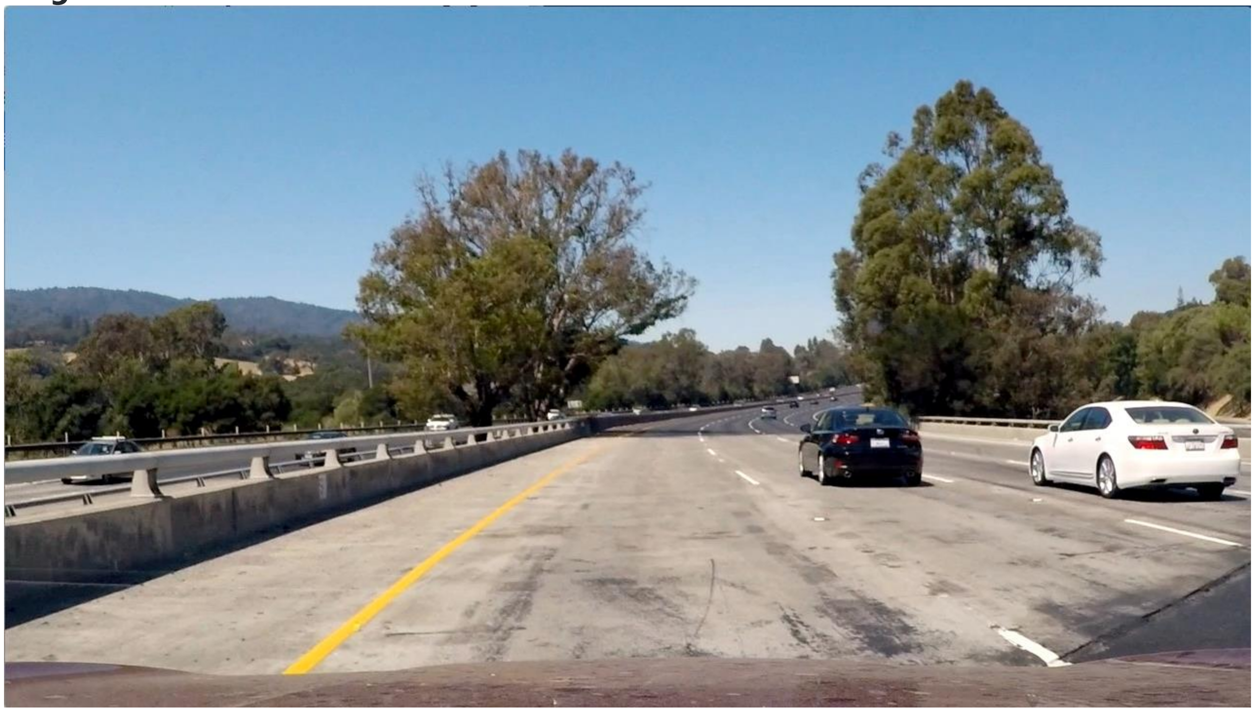
Pipeline (single images)

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

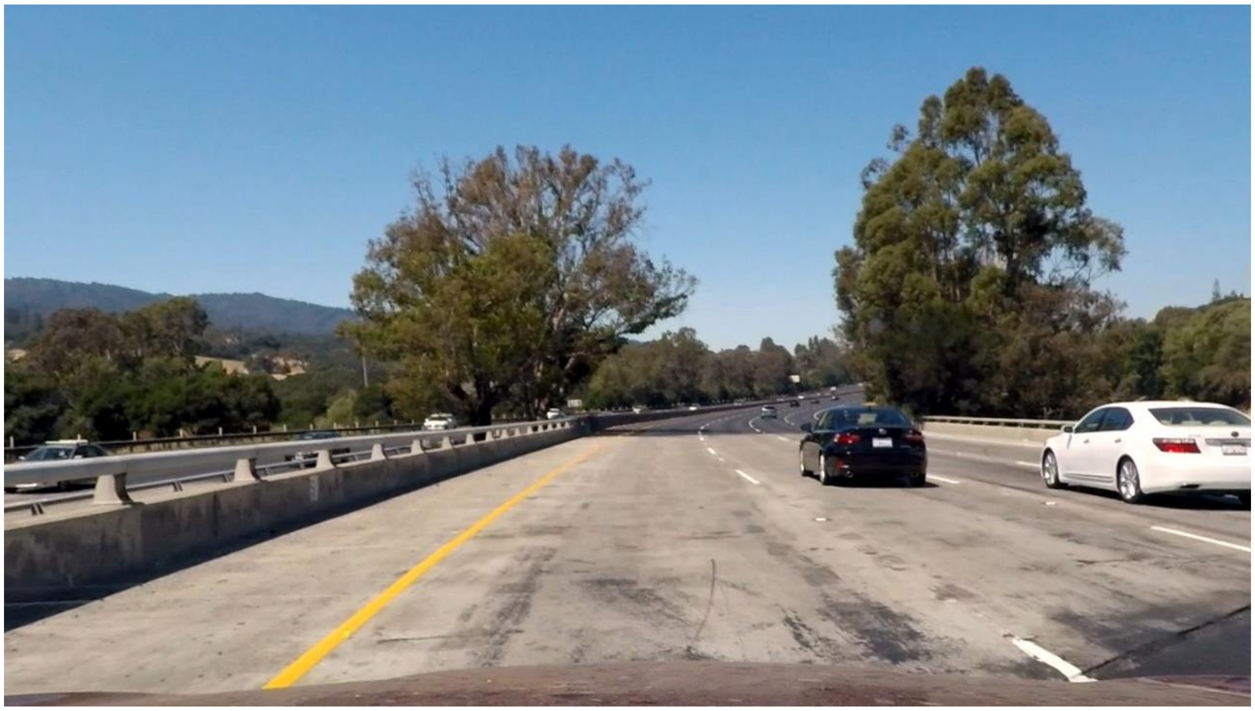
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

I took the distortion coefficient that was created through the camera calibration process and apply to the test images by using the cv2.undistort function. The undistorted and the original images difference can be clearly seen on the white car on the right side.

Original:



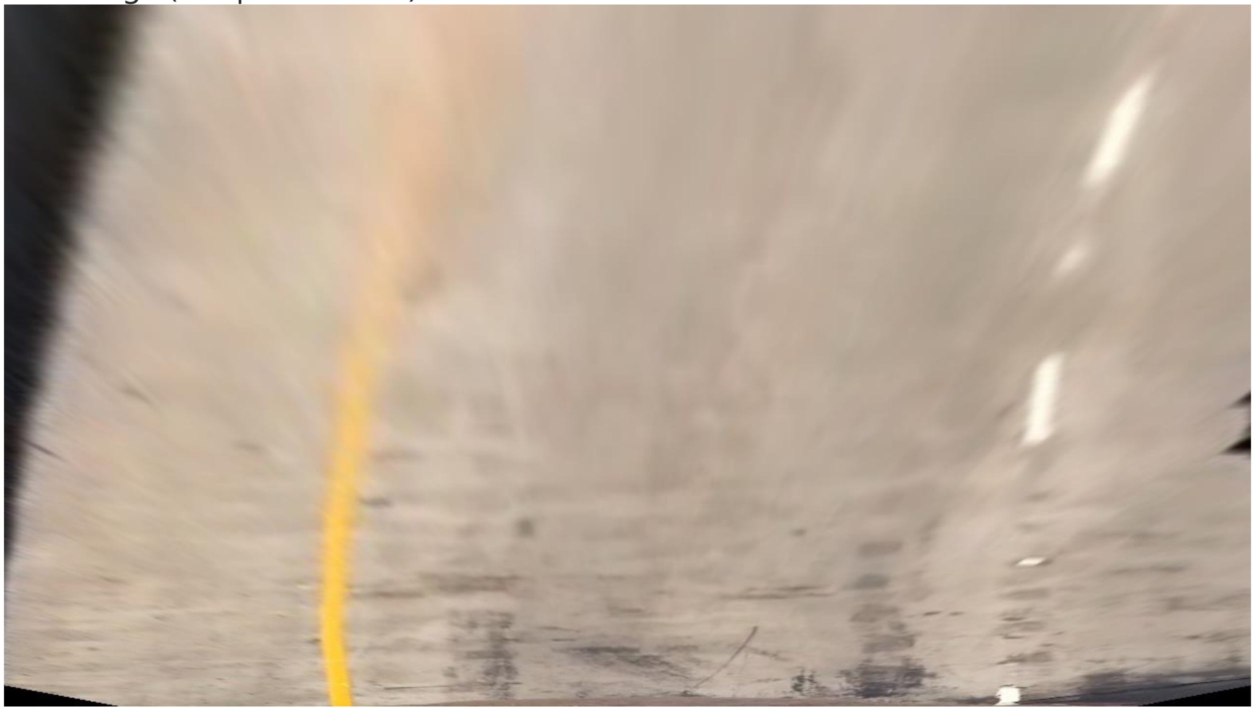
Undistorted:



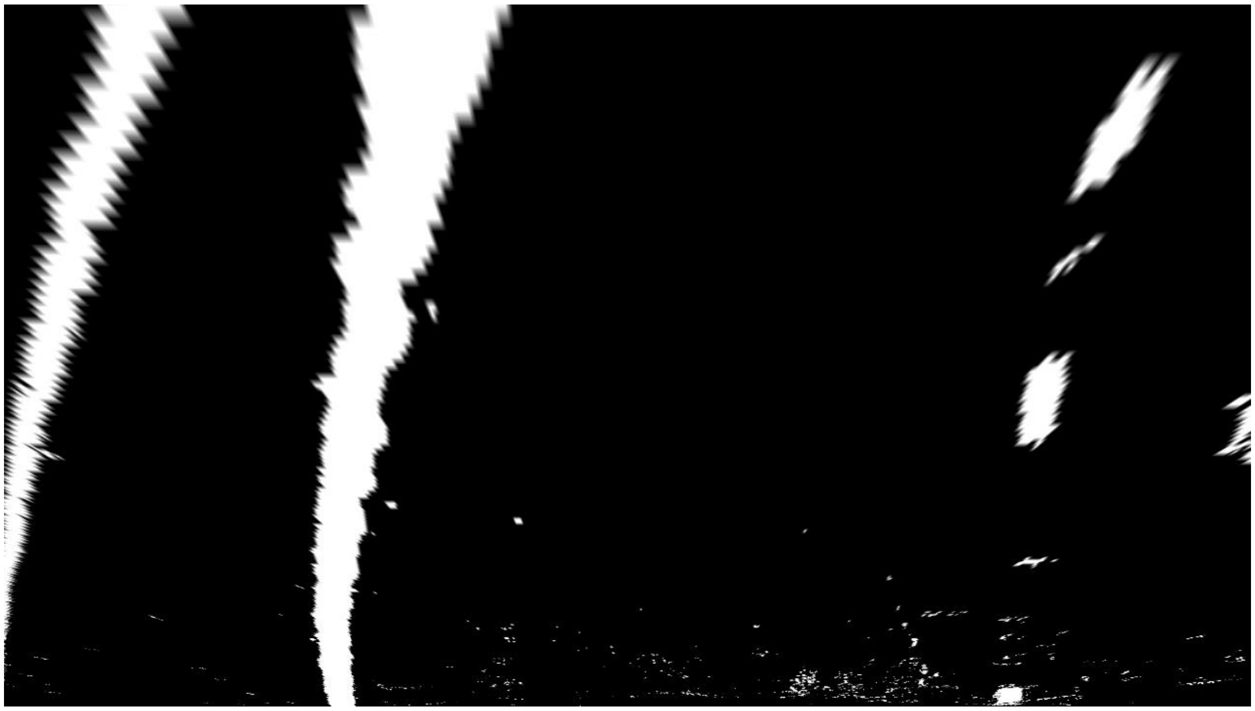
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

To get the maximum lane lines on the road for curvature identification, I utilize both the HLS and HSV color domain for lane identification. I combine the s-channel from the HLS domain and the v-channel from the HSV domain for color thresholding to create binary images. The images below are the original image and binary image comparison in perspective view.

Test Image (Perspective view)



Test Image(Binary)



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

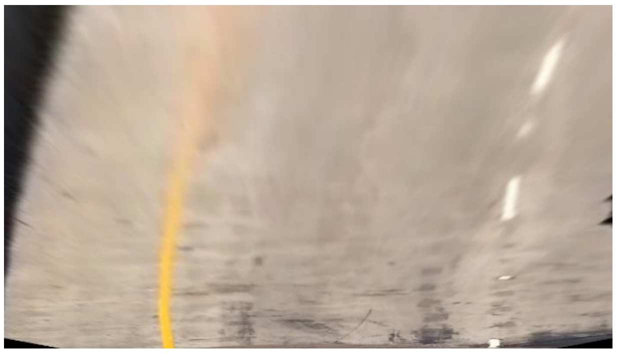
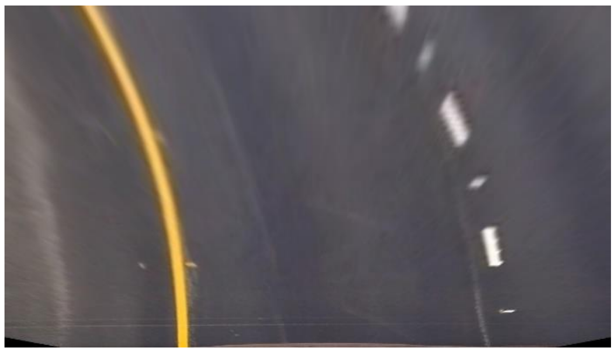
Perspective transform was applied to see the bird eye view. The following code shows the selected source and destination points

```
top_left_src = (563, 470)
bottom_left_src = (220, 700)
top_left_dst = (300,300)
bottom_left_dst = (300,720)

src = np.float32([[top_left_src[0],top_left_src[1]],
[bottom_left_src[0],bottom_left_src[1]],
[width - bottom_left_src[0],bottom_left_src[1]],
[width - top_left_src[0],top_left_src[1]]])

dst = np.float32([[top_left_dst[0],top_left_dst[1]],
[bottom_left_dst[0],bottom_left_dst[1]],
[width - bottom_left_dst[0],bottom_left_dst[1]],
[width - top_left_dst[0],top_left_dst[1]]])
```

This resulted in the following source and destination points:

Source	Destination
563, 470	300, 300
220, 700	300, 720
717, 700	980, 720
1060, 470	980, 300

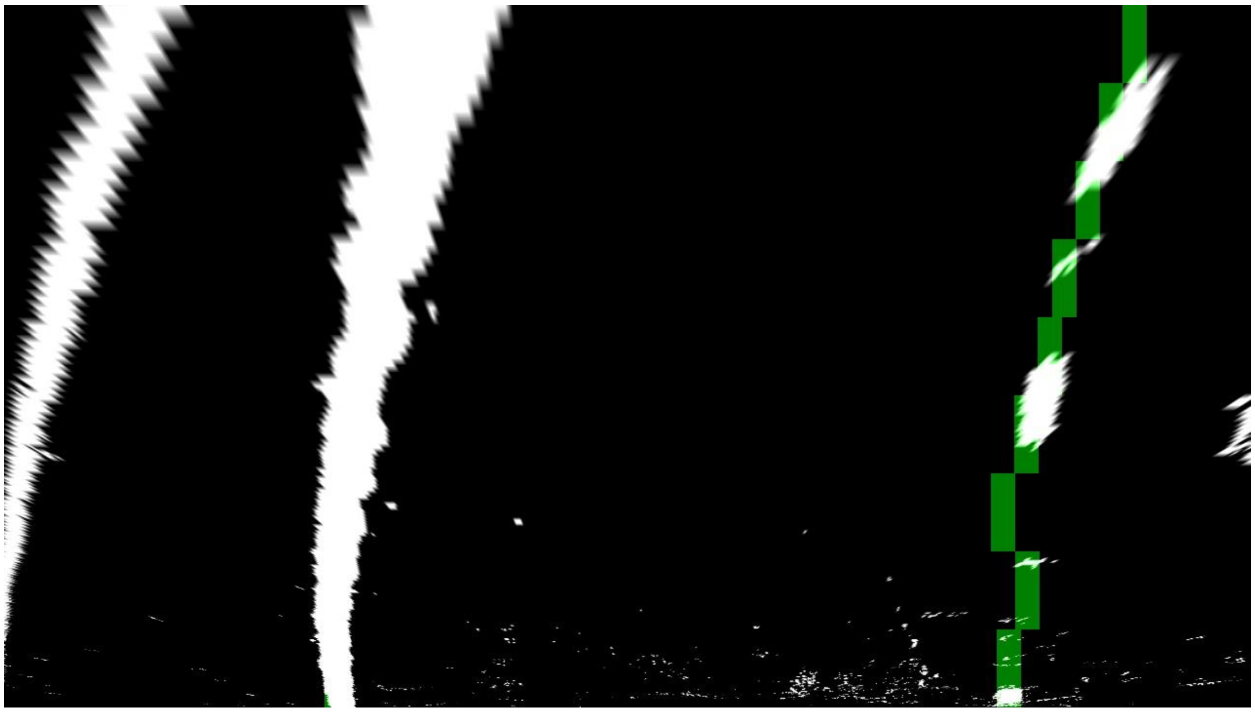
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

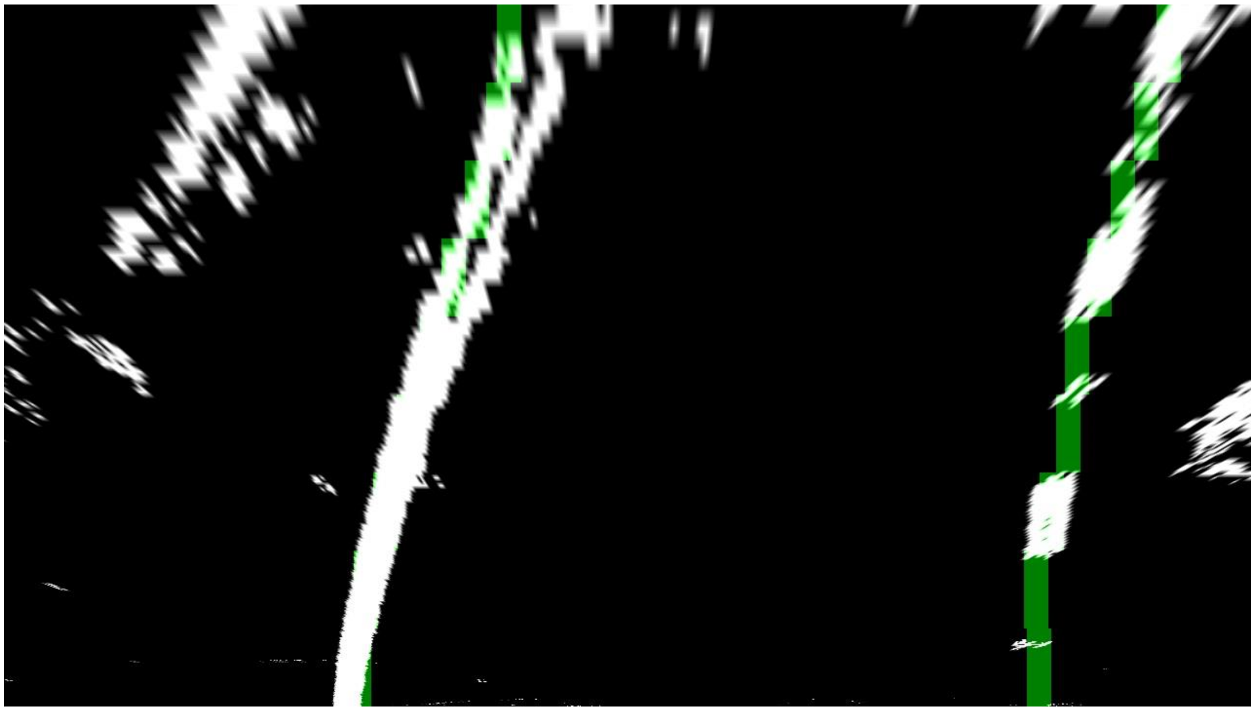



4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

To find the lane lines, instead of using the histogram search, I start with convolutional search since I found some discussion on forum mentioned the histogram search is not accurate enough. For the convolutional search method, I created a file tracker.py to have a tracker class in use of finding the window centroids. With the centroid, line 115 -146 in image_gen.py shows how the box on the centroid was draw. The polynomial fitting is done in line 152 – 175 in image_gen.py

Box draw on the curve

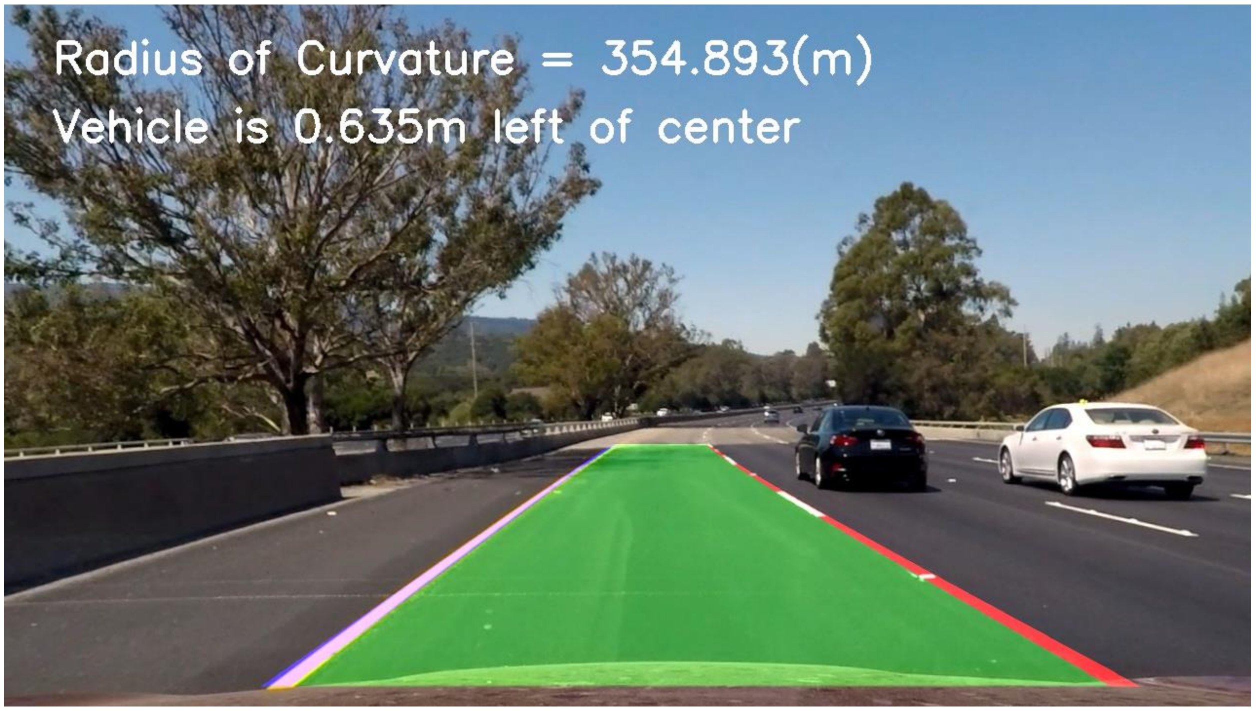




5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I calculate the radius of the curvature by calculating the second derivatives of the fitted polynomial over the bottom of the fitted line, and I assumed the warped pixel space has 10/720 meters per pixel height and 4/384 meters per pixel width. Because the curvature is calculated in pixel, we need to convert the curvature to meter. The curvature calculation is shown in line 182 - 185 in image_gen.py

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


Radius of Curvature = 354.893(m)
Vehicle is 0.635m left of center

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!).

Please see the file attached.

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?

I found my pipe has two main weakness. The first one is when there is shadow or the road concrete color is suddenly different, the lines might suddenly have some flickers. The possible Counter-measure to this might be adjusting the source and destination window a little bit more. Another one is during a curvier area; the top right and left would sway away from the lane line or not curve enough. The possible counter-measure to this might be having a third degrees polynomial to adapt more in-depth curve.