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.

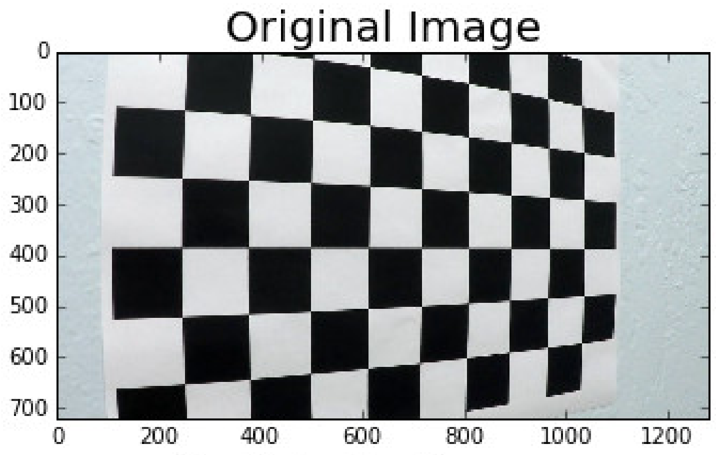
Camera Calibration

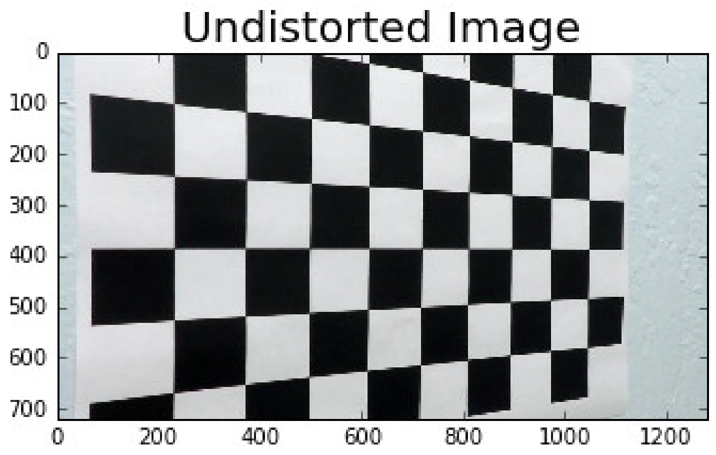
The code for this step is contained in the file called `camera_calibration_using_chessboard_images.py` .

1. Extract object points and image points for camera calibration.

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners which size is setted to 9x6. 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, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `camera_calibration_using_chessboard_images.py function` . I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

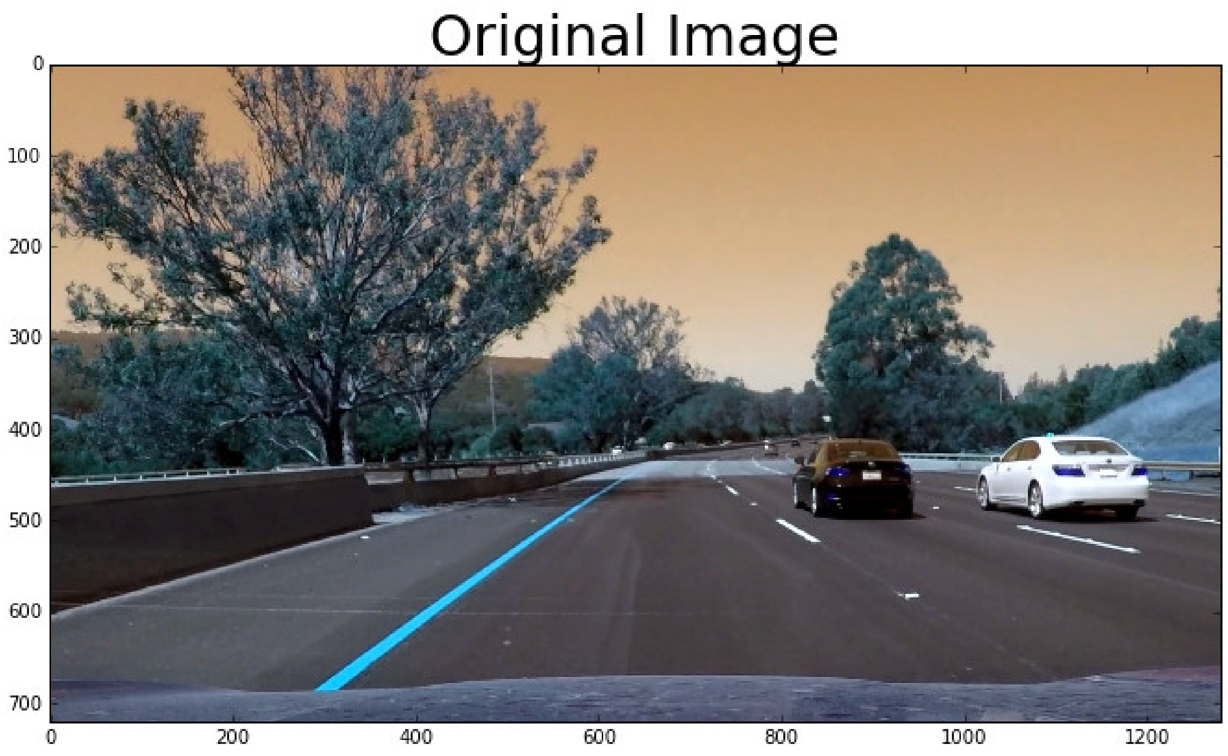


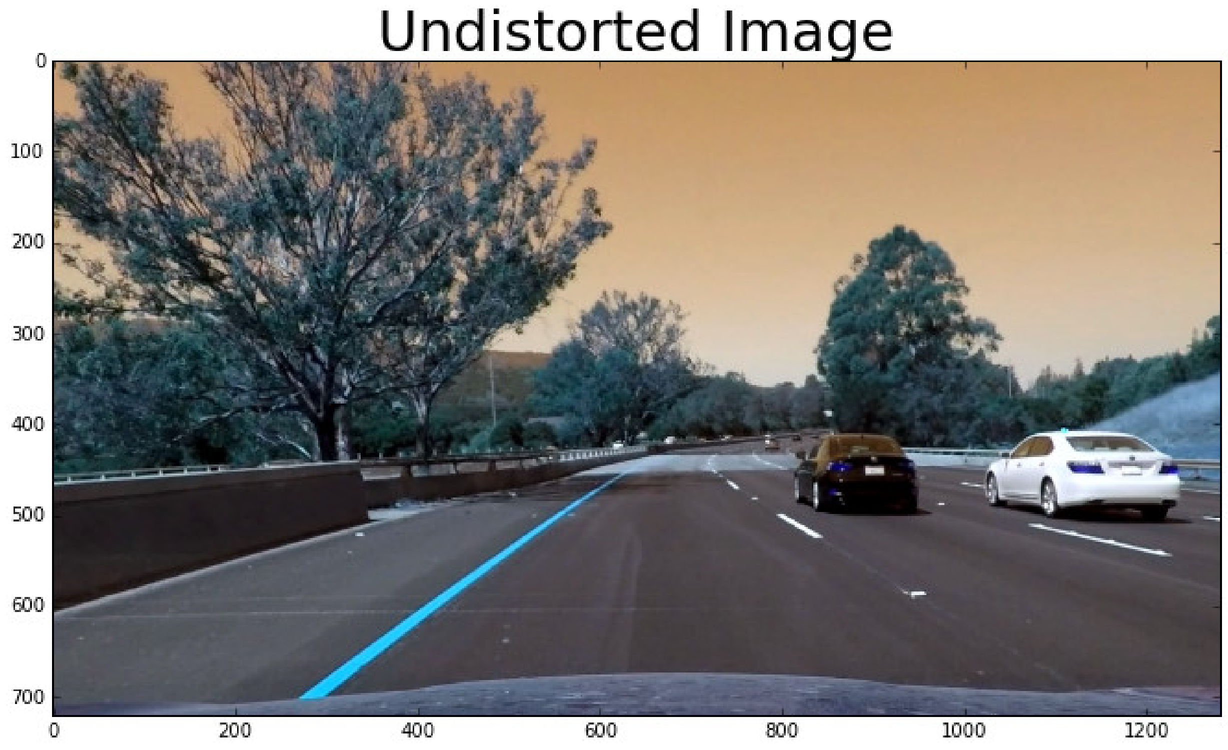


Pipeline (single images)

1. An example of a distortion-corrected image.

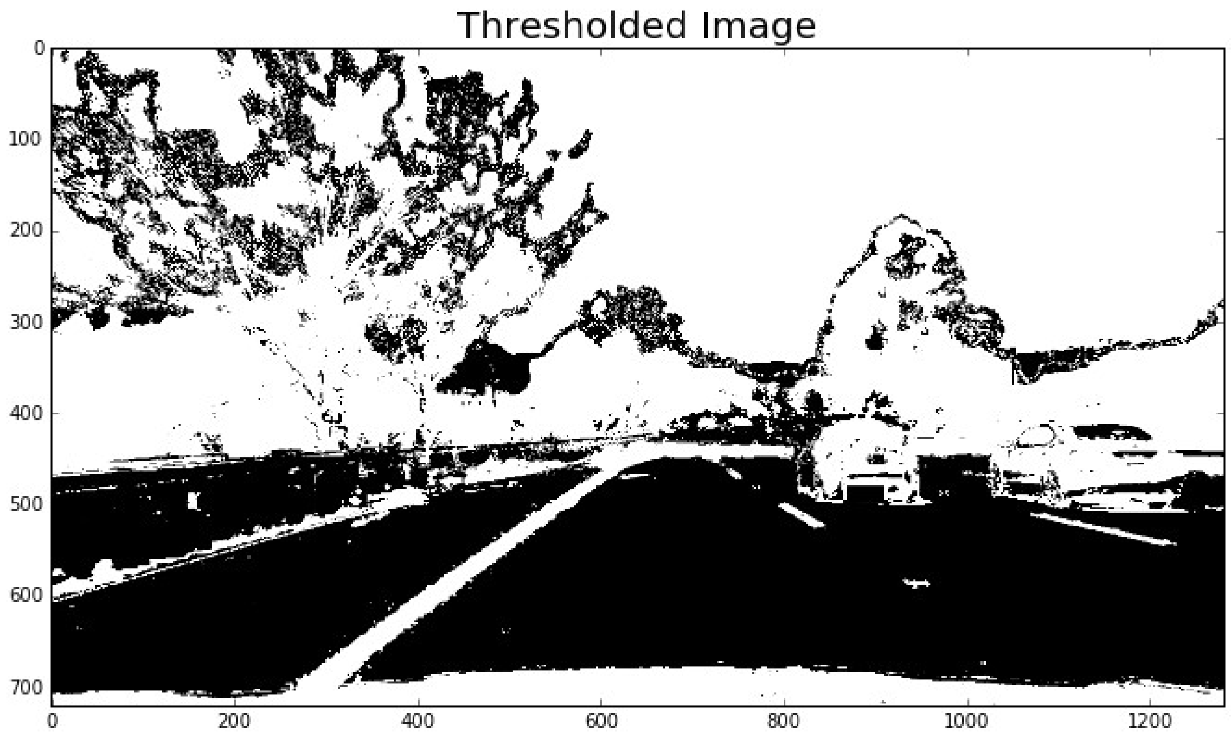
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:





2. Use color transforms, gradients or other methods to create a thresholded binary image.

I tried different combinations of color ,sobel operator, magnitude of the gradient and direction of the gradient thresholds to generate a binary image (thresholding steps at lines 141 through 149 in `instrument_function.py`). Below is an example of my output for this step.



3. Performed a perspective transform to image.

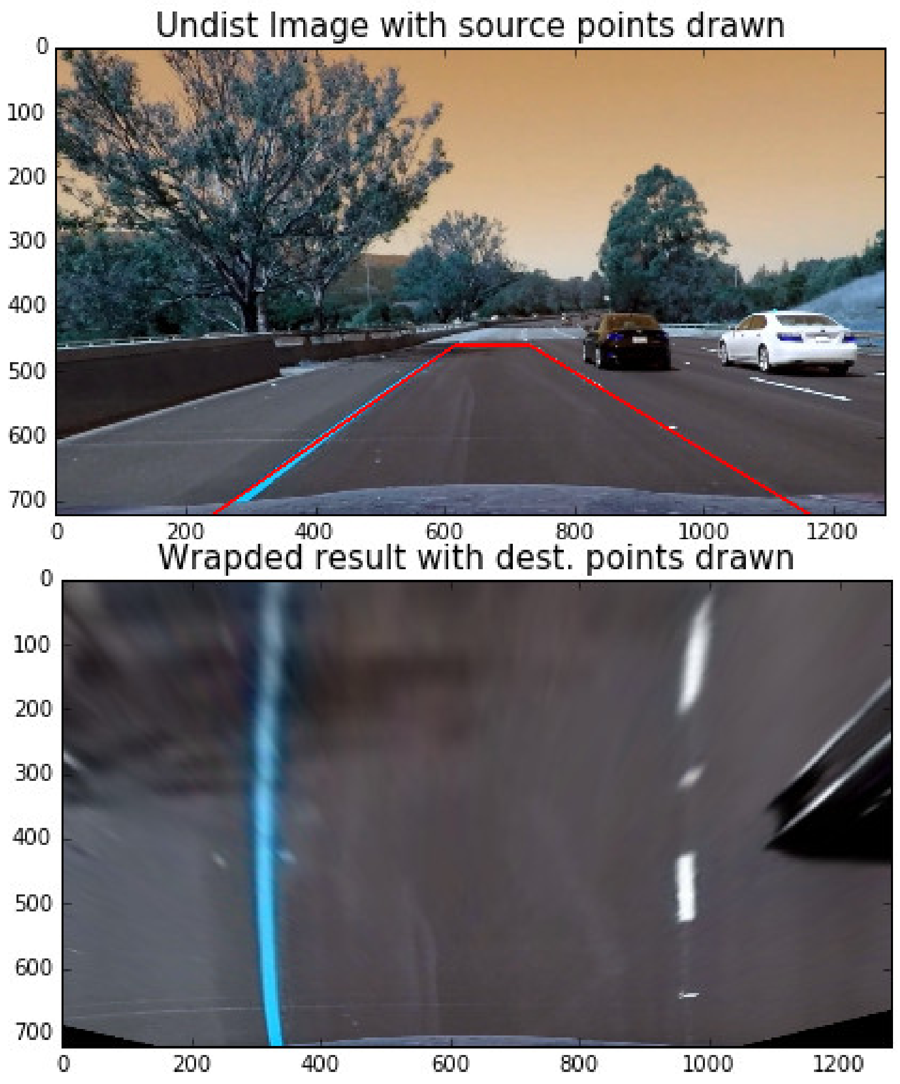
After distorted, the code for my perspective transform also includes in file `instrument_function.py` , which appears in lines 161 through 175. The code in `instrument_function.py` takes as inputs an image (`./calibration_wide/test_undist.jpg`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

```
src =
np.float32([[(img_size[0] / 2) - 25, img_size[1] / 2 + 100],\
            [((img_size[0] / 6) + 30), img_size[1]],\
            [(img_size[0] * 5 / 6) + 95, img_size[1]],\
            [(img_size[0] / 2 + 90), img_size[1] / 2 + 100]])
dst =
np.float32([[(img_size[0] / 4), 0],\
            [(img_size[0] / 4), img_size[1]],\
            [(img_size[0] * 3 / 4), img_size[1]],\
            [(img_size[0] * 3 / 4), 0]])
```

This resulted in the following source and destination points:

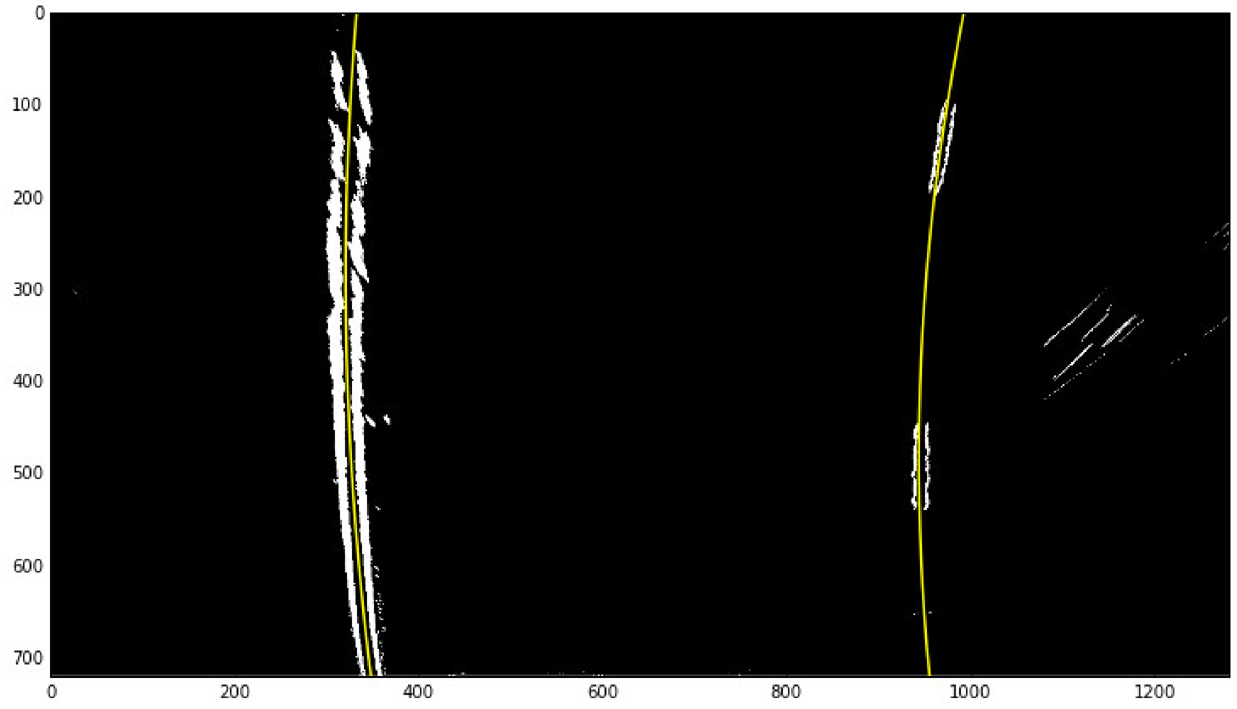
Source	Destination
585, 460.	320, 0
203.33, 720	320, 700
1121.66, 720	960, 700
700, 460	960, 0

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.


Undist Image with source points drawn
Wrapded result with dest. points drawn
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

4. Identified lane-line pixels and fit their positions with a polynomial

Then I implemented sliding windows to find which “hot” pixels are associated with the lane lines in fuction called `find_lines()` in file `Finding_the_Lines.py` . Then I fit my lane lines with a 2nd order polynomial kinda like this:



$$f_{left} = A_{left}y^2 + B_{left}y + C$$

$$f_{right} = A_{right}y^2 + B_{right}y + C$$

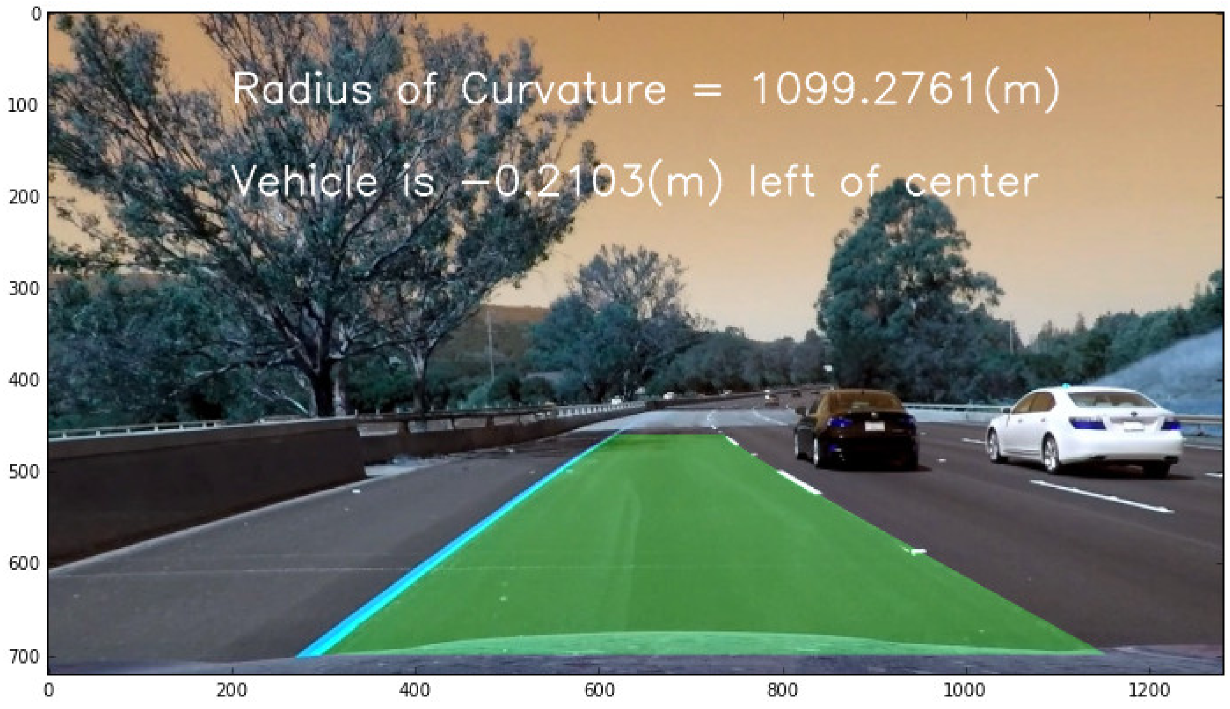
5.Calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I did this in lines 203 through 208 in my code in `instrument_function.py` , and called a function `meas_cur()` ,the result is as follow:

```
left_curverad : 1099.27608597 (m) ,right_curverad : 806.926849431
(m)
Vehicle is -0.210319161451 (m) left of center
```

6. Processing result of image plotted back down onto the road

I implemented this step in lines 309 through 339 in my code in `Tracking.py` . Here is an example of my result on a test image:


Radius of Curvature = 1099.2761(m)
Vehicle is -0.2103(m) left of center
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

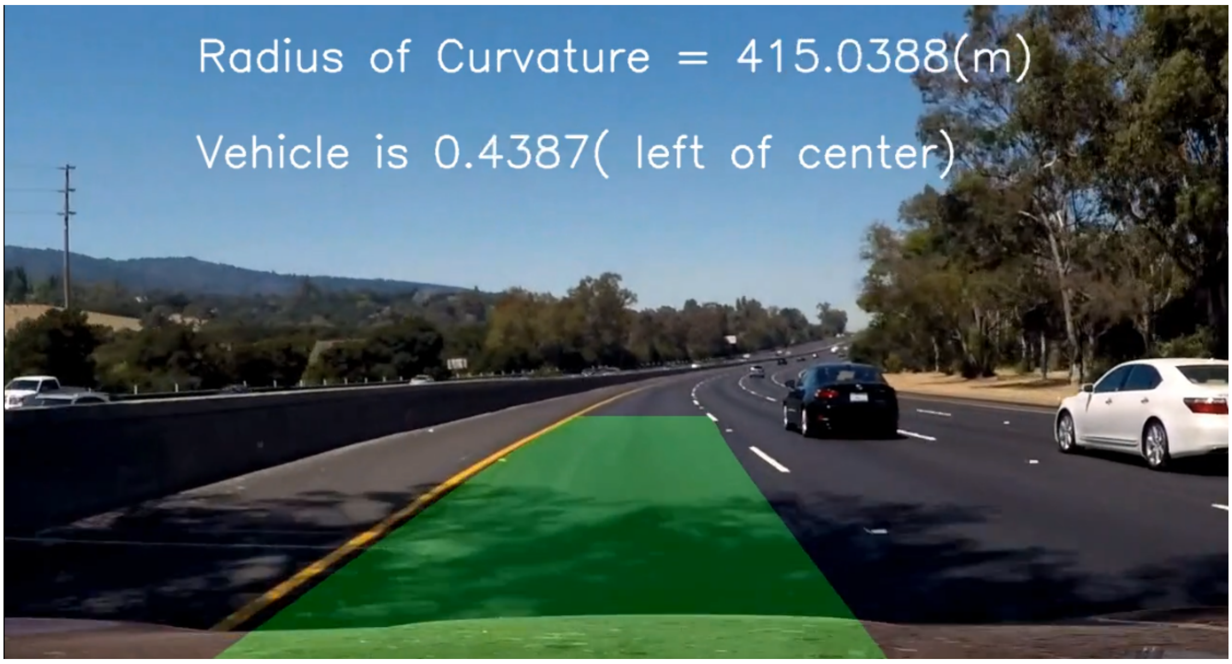
Pipeline (video)

I use `Tracking.py` to do the lane finding and here's a link to my video result.

Discussion

This project was difficult in that it took a while to organize everything into a functioning system ,and tuning of several parameters was a very tedious process. This project let me to appreciate deep learning based approaches even more.

In my premier implementation of my code, I find wobbly region appeared when the car dive in intensity sunlight or shadow where the lane line is unclear,something like this:


Radius of Curvature = 415.0388(m)
Vehicle is 0.4387(left of center)

To avoid the impact of sunlight and shadow , I tried some combination of thresholds method to generate a binary image as clear as possible. At last I used with combination of information from two color spaces: HSL and HSV. Only the L and S channels are used from HSL color space, while only V channel is used from HSV color space. The code is in lines 67 through 112 in `instrument_function.py` . The L channel and S channel are used for gradient extraction and filtering along x, while the V channel with some thresholding is used to mask shadows area in binarized image. Then, I creat a class Line to receive and process the characteristics of each line detection. I defined some functions to improve robust:

`sanity_check()` checking the detection result

- Checking that they have similar curvature or roughly parallel
- Checking that they are separated by approximately the right distance horizontally
  I chose the threshold of parallel refer to Horizontal Curvature of Highways without Superelevation,and width threshold of the road is set to >3.2 and < 4.5 considering some error in computation and estimation.

`smoothing()` smooth over the last n frames of video to obtain a cleaner result
If detection from some frame did not passed check of `sanity_check()` because of some reasons. I average the n past fit results to estimate the current fit coefficients and if successive detection lost exceed n times, reset and start searching from scratch using a histogram and sliding window to re-establish measurement.

`look_ahead_Filter`
If the lane lines in one frame of video has been dectected and passed check, search within a window around the previous detection.