ADVANCED LANE DETECTION

ZEESHAN ANJUM

UDACITY

Introduction:

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.

Main file is main.py

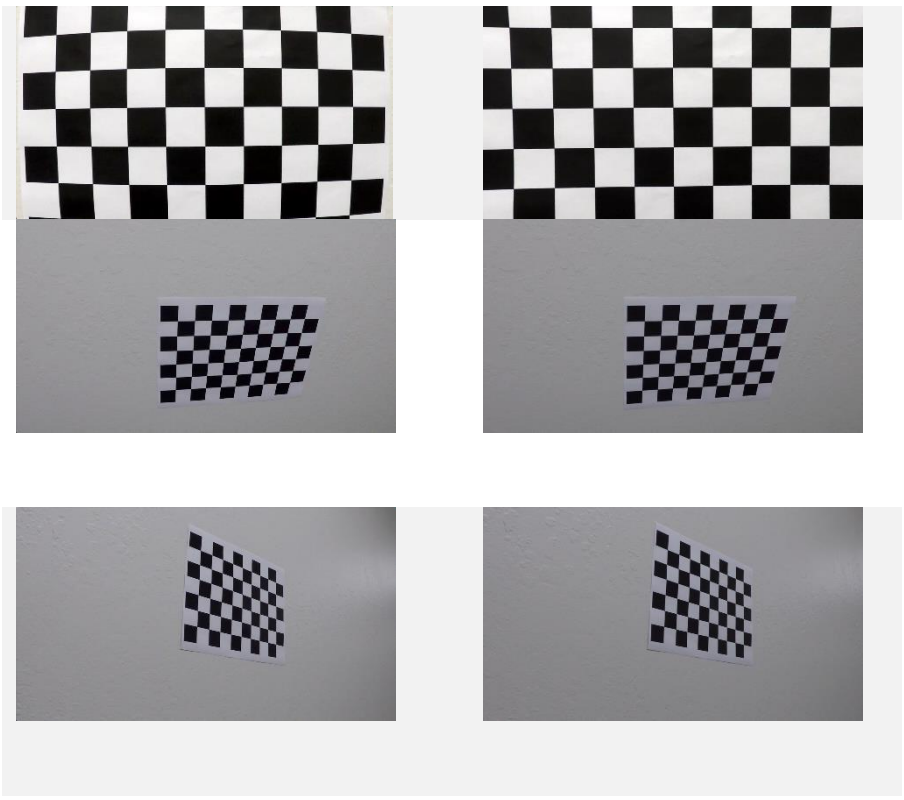
Camera Calibration:

I calibrate lens distortion of the camera using 9x6 chess board images. All images were provided by the Udacity.

You can find code in calibration.py. Here I assume that chessboard image is placed on 2D plane. Function `calibrate_using_checkerd_images` takes an image, count of cell corners in chessboard, and optional parameter draw_marker to display the detected points. This function return image points and object points.

Example on test image:

Uncalibrated Image **Calibrated Image**



Example of distortion correction on Image:

In main file, there is a function named camera_calib I calculated and saved calibration coefficients for later use.

In processing pipeline, I read camera coefficients once and saved it into global variables.

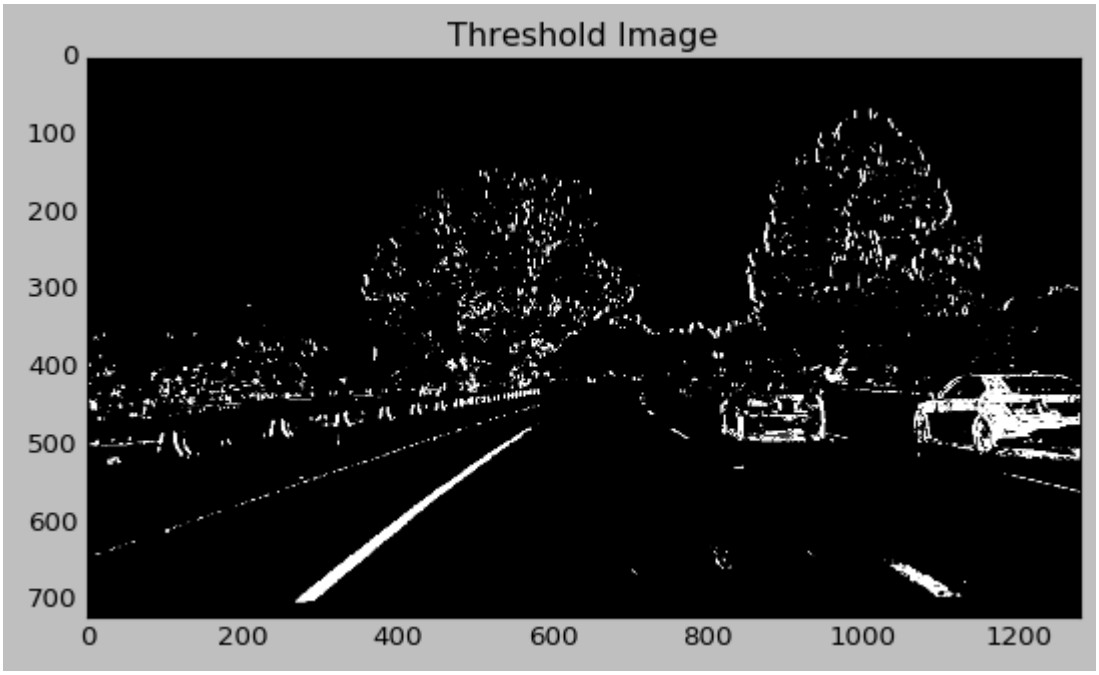
Pipeline

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

Distortion corrected image:

As already described above the calculation of distortion correction coefficients.

I used a combination of color and gradient thresholds to generate a binary image. In file image_processing.py, on line number 145, I create a function 'apply_thresholds', which tak image, kernel size and detailed image display as input and generate a binary image output. Here's an example of my output for this step.



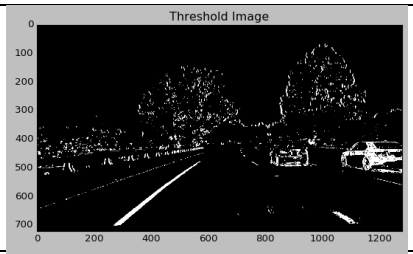
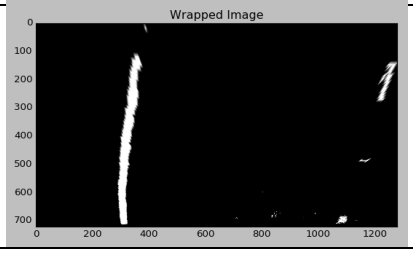
Perspective Transform:

I consider road as a plane surface (which is not in real life) and apply perspective transform to see the top view of the road. I manually find the points in the test images and approximate for all others.

Following are the points used as source and destination for the image.

Source	`0, 720`	`550, 470`	`700, 470`	`1000, 720`
Destination	`200,720`	`200,0`	`1000,0`	`1000,720`

Example of perspective distortion correction.

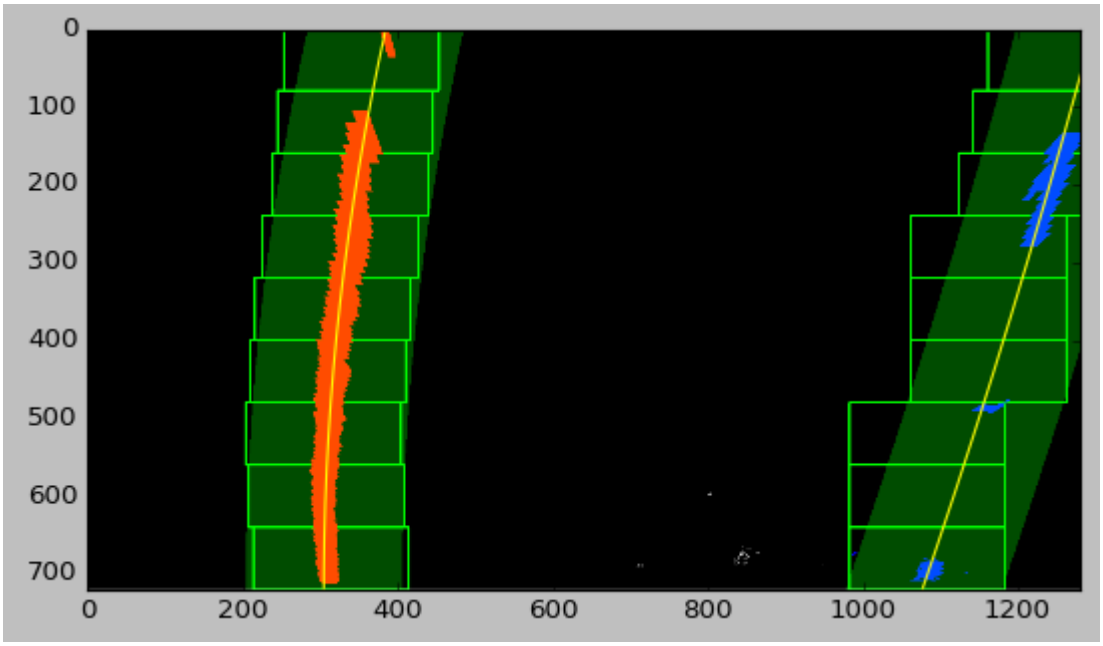
Source Image	
Destination Image	

Lane Line Pixels Detection:

I find high peaks in image and consider it as lane line pixels, then I use 2nd order polynomial to find the curve line.

- Calculate a histogram of the bottom half of the image
- Partition the image into 9 horizontal slices
- Starting from the bottom slice, enclose a 100 pixel wide window around the left peak and right peak of the histogram (split the histogram in half vertically)
- Go up to the horizontal window slices to find pixels that are likely to be part of the left and right lanes, and tries to center the window with line.
- Given 2 groups of pixels (left and right lane line candidate pixels), fit a 2nd order polynomial to each group, which represents the estimated left and right lane lines

The code perform above steps is in the histogram_pixels_v3 function of image_processing.py file.



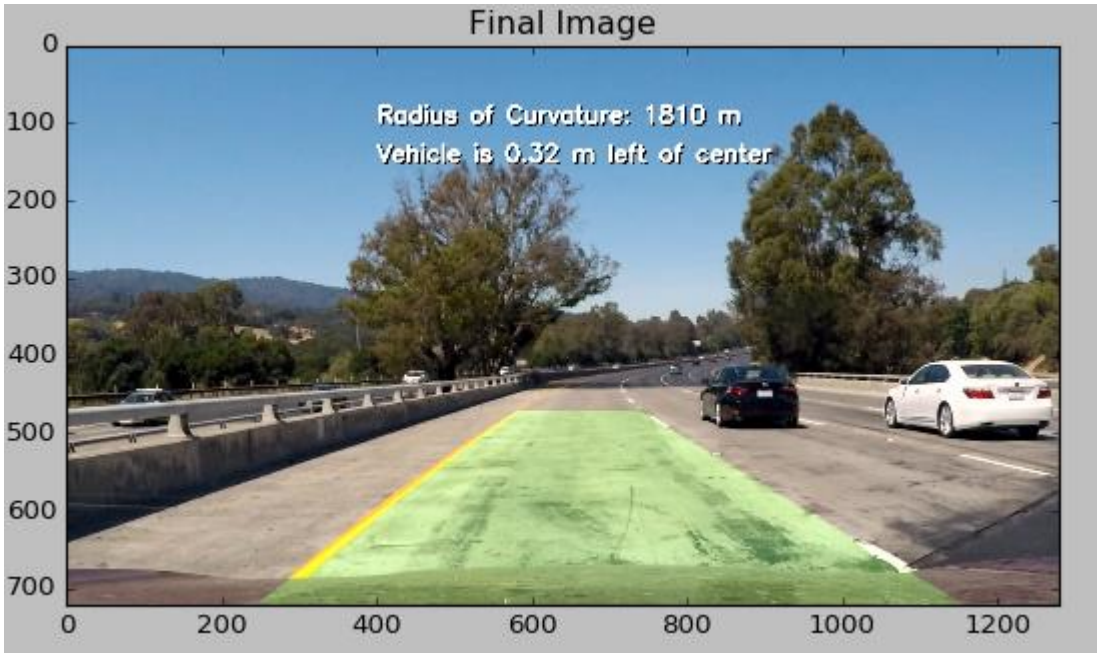
Calculating Radius of curvature and center of vehicle respect to the lane:

I defined two functions 1st find_curvature at line no 168, and find_position at line no 181 in file 'line.py'. I calculate the radius of curvature using following code:

```
fit_cr = np.polyfit(yvals * ym_per_pix, fitx * xm_per_pix, 2)
curve = ((1 + (2 * fit_cr[0] * y_eval + fit_cr[1]) ** 2) ** 1.5) / np.absolute(2 * fit_cr[0])
```

and I assume that camera is mounted in the center of the car, so I find vehicle center position by getting image center then threshold the left and right lane pixels with center position and 700 then subtract the center from position and multiply by pixels to meter constant.

Final output:



Video link:

Here is the link of video output for advanced lane detection: https://youtu.be/jsYS6xHdH-o

Discussion:

This algorithm is good for finding clear curved lanes. But this is not very advanced to be applied on real life application. Because it can be affected by illumination invariants such as shadows, cracks, missing lane lines etc. Other objects and vehicle in front of the road can also trick the algorithm to miss/hit the pixels for lane detection. We can use semantic segmentation approach to deal with this type of problem like used in Ankit Laddha's paper, Road Detection and Semantic Segmentation without Strong Human Supervision.