Writeup - 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

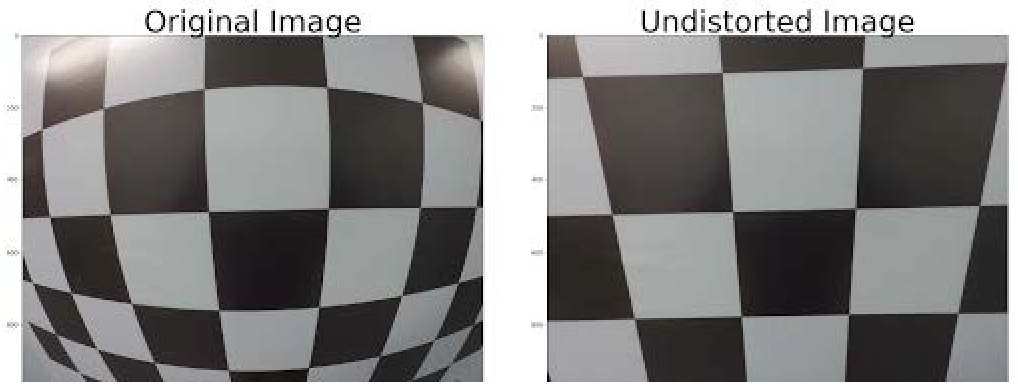
Camera Calibration

1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

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

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, `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 `cv2.calibrateCamera()` function. 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:



Pipeline (single images):

1. Has the distortion correction been correctly applied to each image?

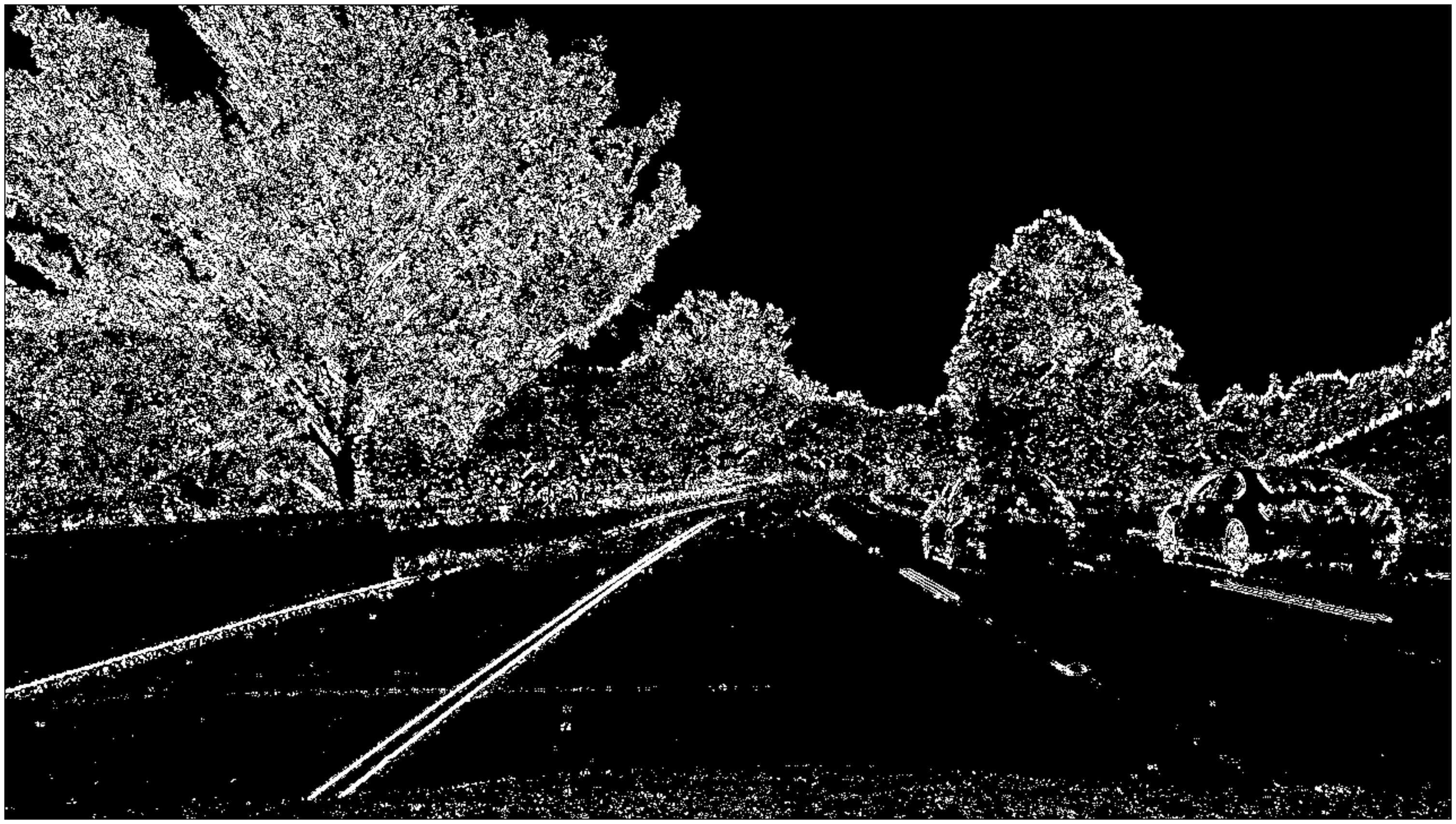
I used the camera matrix and the distortion coefficients obtained in the previous step and used the "cv2.undistort" function to undistort images in the video stream. I used pickle to save the values from the previous step and loaded them for use in the pipeline.

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.

The code used to experiment with color, gradients, and thresholds could be found in the image_gen.py. For the limits, I chose a combination if Sobel-X, Sobel-Y, magnitude & directional gradient thresholds and S-channel HLS color transform gave the best results.

There is definitely scope for improvement here. I can see there are some areas where the color of the pavement is changed the lane lines were going beyond the boundary.

The binary image which was obtained for the test image for the selected gradient and color thresholds is below.



3. Has a perspective transform been applied to rectify the image?

The code for my perspective transform is includes a function called WarpImgConstants(), which appears in lines 115 in the file image_gen.py This function takes a test image and based on the selected source and destination points generated the transformation matrix M and its inverse.

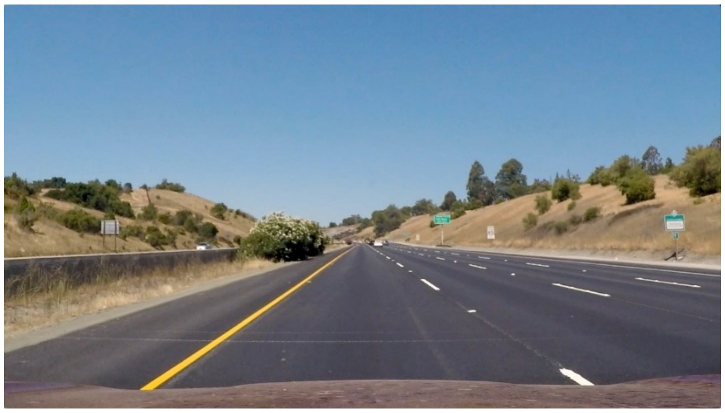
This selected source and destination points are:

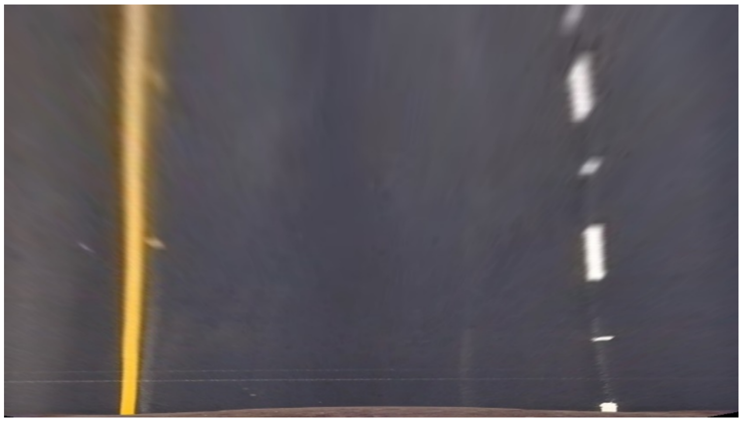
Source	Destination
585, 455	200, 0
705, 455	1080,0
1130, 720	1080, 720
190, 720	200, 720

This resulted in the following matrix values.

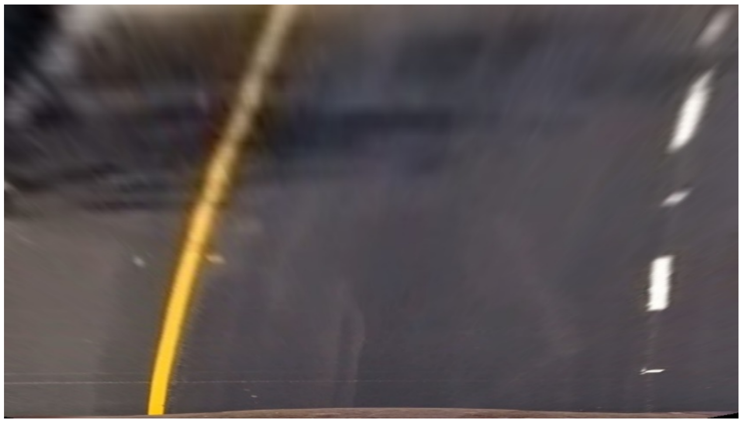
```
M:
[[-6.83269851e-01 -1.49897451e+00  1.06311163e+03]
 [-1.20729385e-15 -1.98300615e+00  9.02267800e+02]
 [-1.15194662e-18 -2.40257838e-03  1.00000000e+00]]

MInv:
[[ 1.36363636e-01 -7.78812057e-01  5.57727273e+02]
 [-5.26327952e-17 -5.04284870e-01  4.55000000e+02]
 [ 0.00000000e+00 -1.21158392e-03  1.00000000e+00]]
```

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.



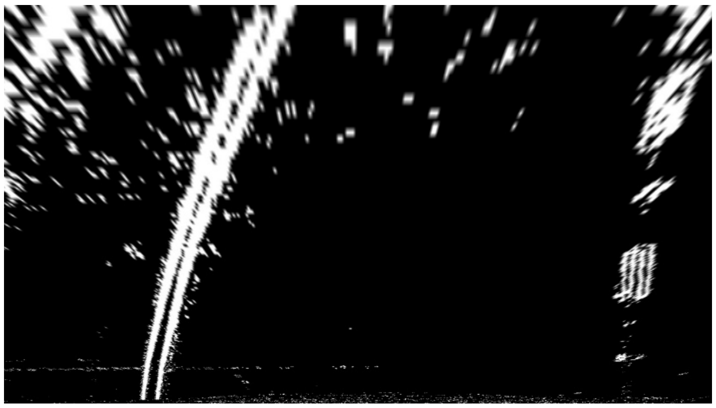


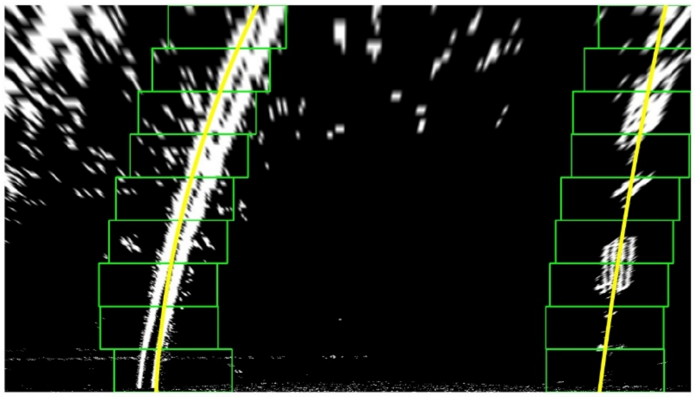
The below images show the transformation on the curved road.



4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

The algorithm calculates the histogram on the X axis. Finds the picks on the right and left side of the image, and collect the non-zero points contained on those windows. When all the points are collected, a polynomial fit is used (using np.polyfit) to find the line model. On the same code, another polynomial fit is done on the same points transforming pixels to meters to be used later on the curvature calculation. The following picture shows the points found on each window, the windows and the polynomials:





The source code for this section can be found from image_gen.py file from line 214 in the function findlines().

5. **Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?**

On the step 5 a polynomial was calculated on the meter's space to be used here to calculate the curvature. The formula is the following:

```
((1 + (2*fit[0]*yRange*ym_per_pix + fit[1])**2)**1.5) / np.absolute(2*fit[0])
```

where `fit` is the array containing the polynomial, `yRange` is the max Y value and `ym_per_pix` is the meter per pixel value.

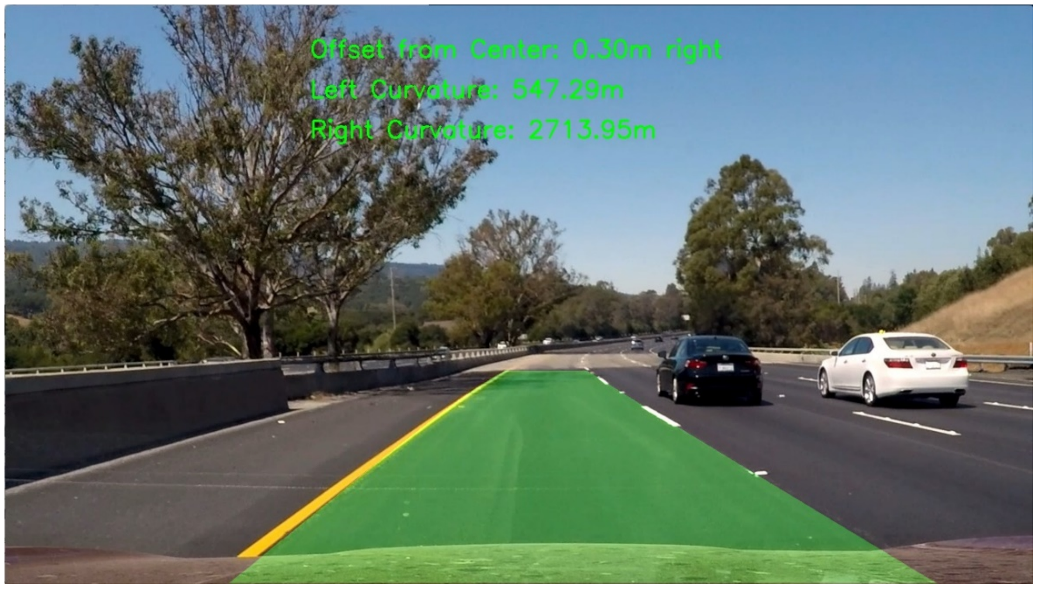
To find the vehicle position on the center:

- We calculate the lane center by evaluating the left and right polynomials at the maximum Y and find the middle point.
- The center is calculated by transforming the length from the center of the image to the lane center from pixels to meters
- The distance between the lane line and the vehicle center is calculated. If the distance is less than 0 or greater than 0 gives if the vehicle is on to the left or the right.

The code for the above can be found in the file lane_pipeline.jpg in the function calculateLanes().

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

To display the lane markings and the polynomials were evaluated to draw the boundaries and fill the polygon. The generated points were mapped back to image space using the inverse transformation matrix. The code used for this operation can be found at image_gen.py line 315, function drawLine(),

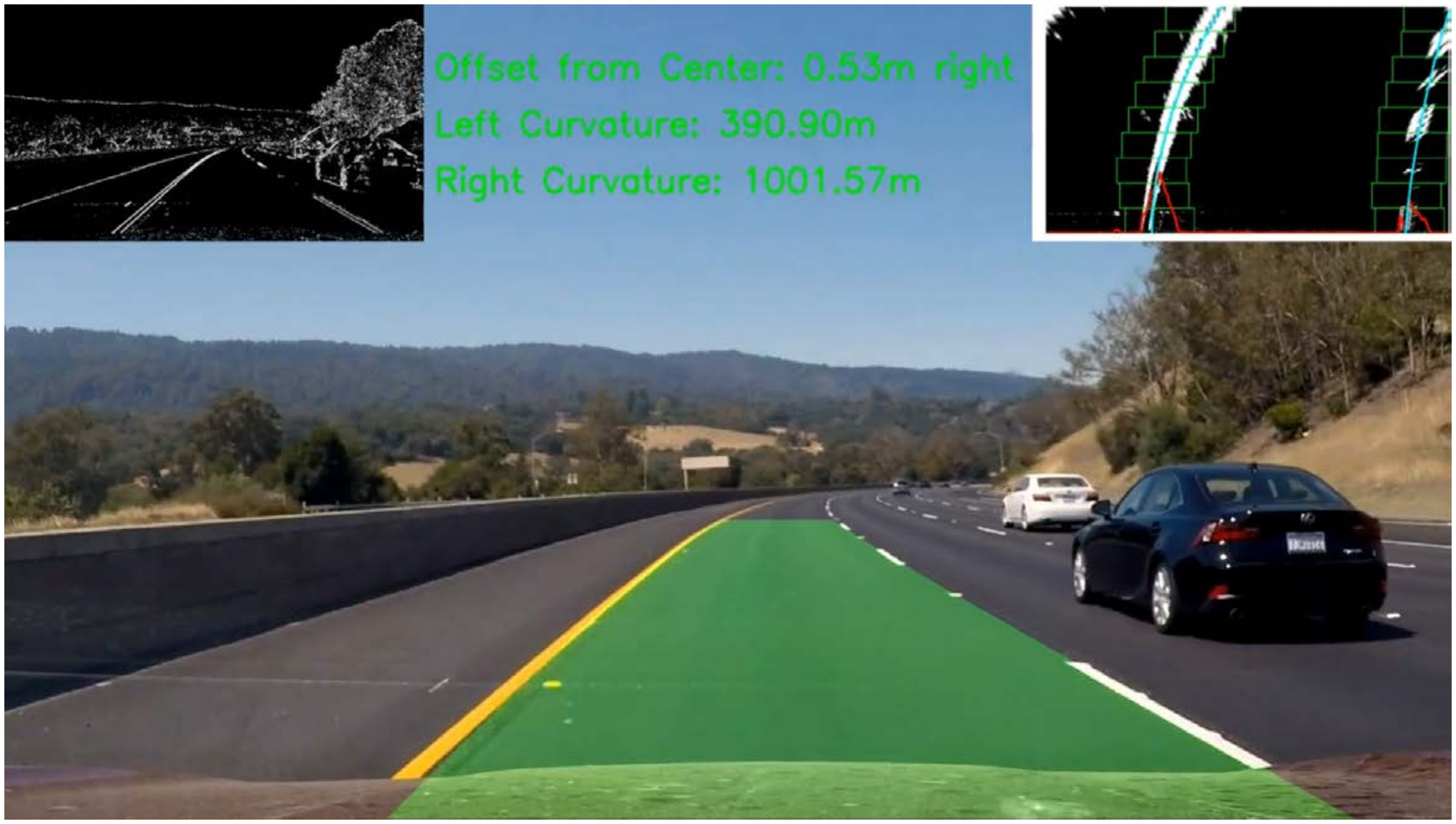


Pipeline (video)

1. Does the pipeline established with the test images work to process the video?

It sure does! Here's a link to my video result.

To give a clear picture of the inputs to the final frames I have plotted the thresholded image on one cornet and the transformed lane lines with a overlay of the histogram and the polynomials on the other corner.



The video pipeline can be found in the file lane_pipeline.py

Discussion

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

- As mentioned above tweaking of the thresholds for gradient and color can be performed.
- With this method, it’s very sensitive to light changes.
- This might not be working for driving in the night time with lots of lights from other cards
- Conditions like rain might not be handled

References:

https://github.com/JamesLuoau/Self-Driving-Car-Advanced-Lane-Lines

https://github.com/darienmt/CarND-Advanced-Lane-Lines-P4