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

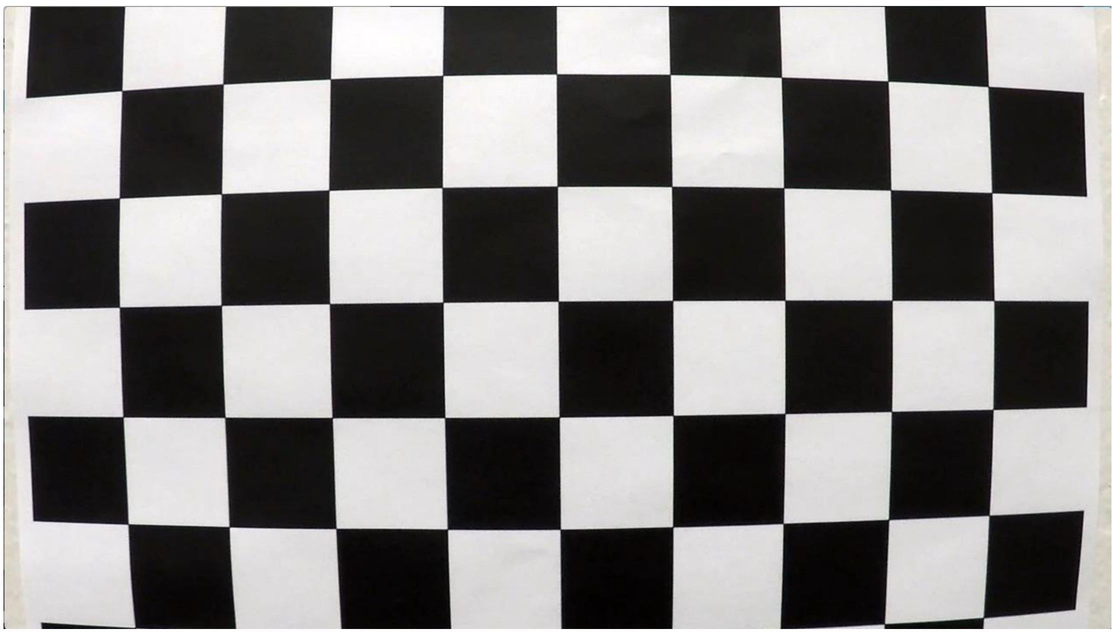
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

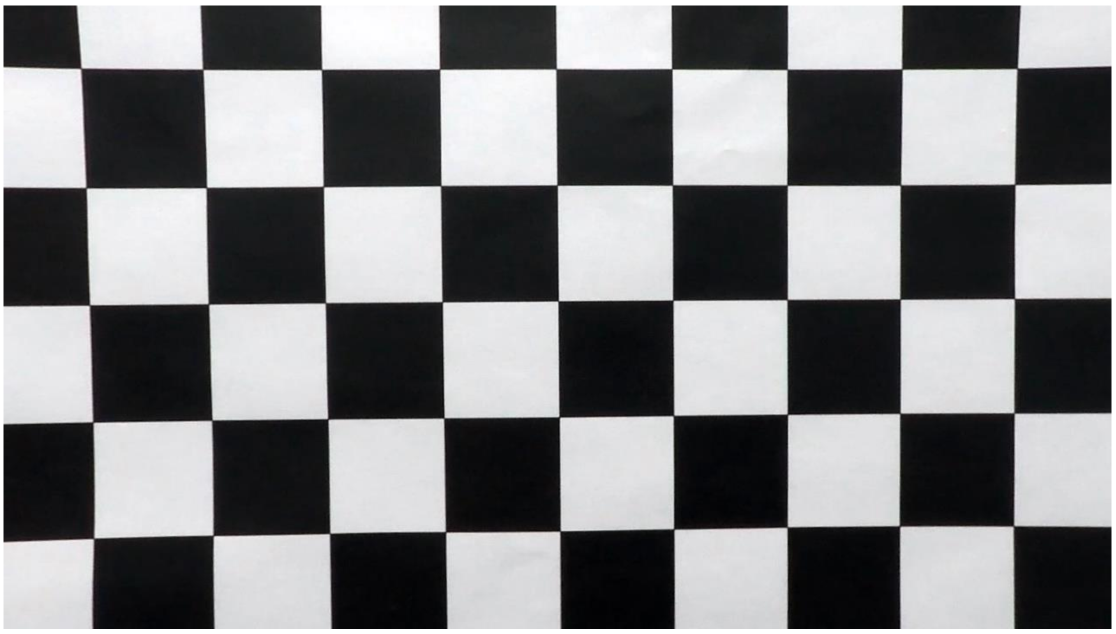
Camera Calibration

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 lines 10-52 of the file called `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 applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:
Original Image

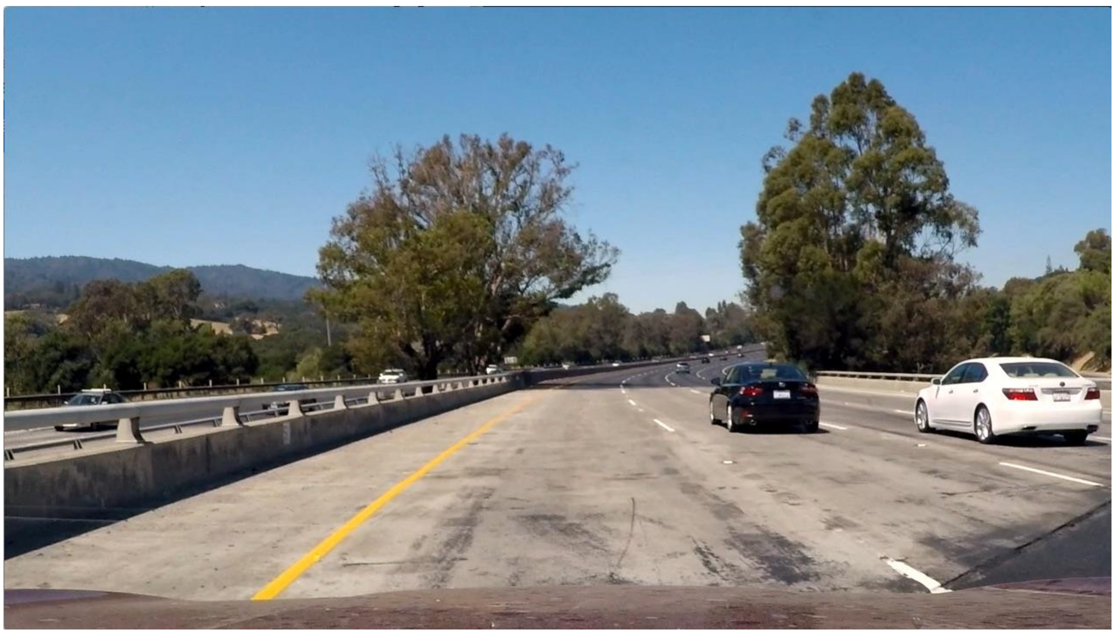


Undistorted Image

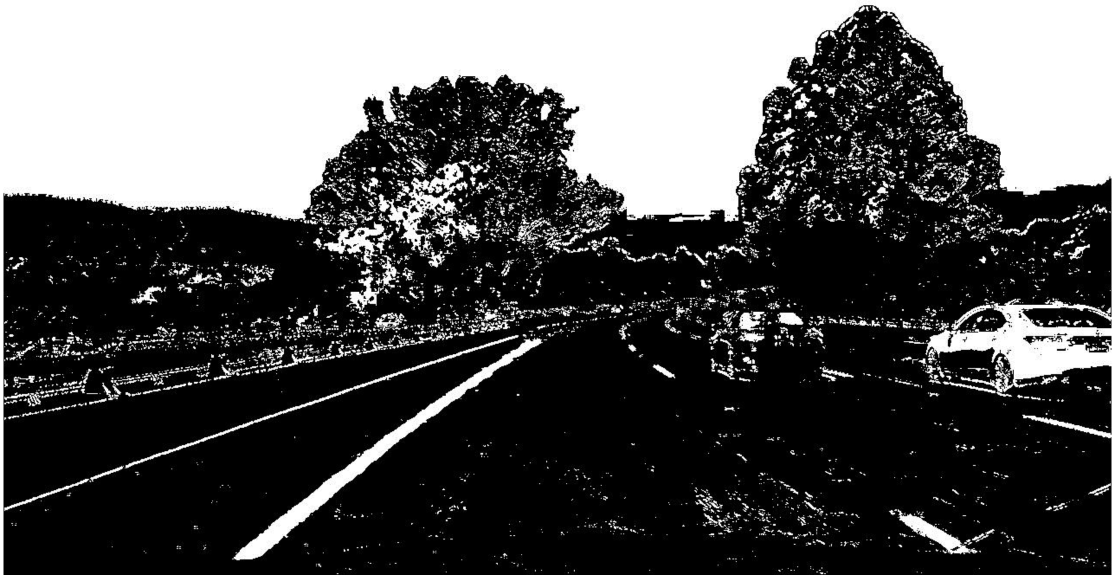


Pipeline (single images)

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:



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 (thresholding steps at lines 75-79 in `image_gen.py`). Here's an example of my output for this step.



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 appears in lines 85 through 97 in the file `image_gen.py`. I make the source and destination points in the following manner:

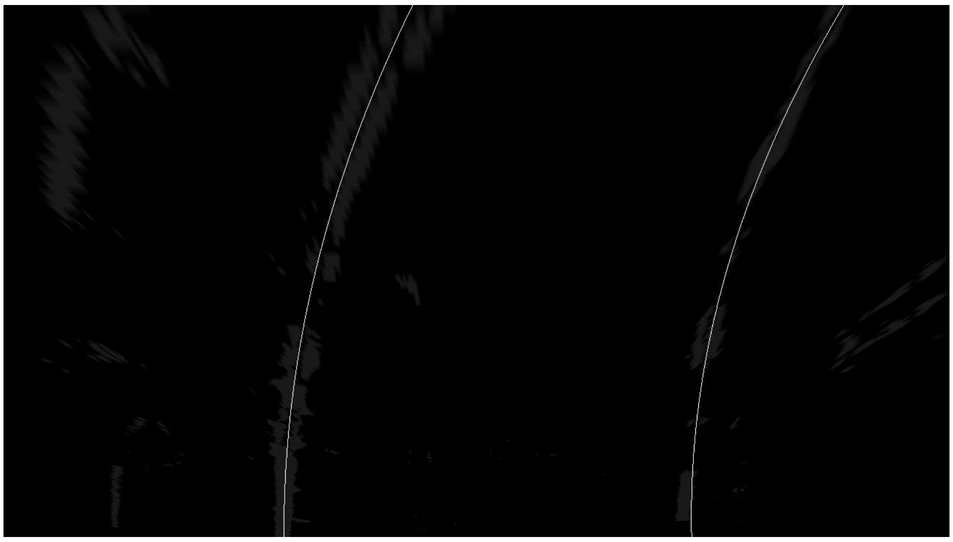
```
img_size = (img.shape[1], img.shape[0])
bot_width = .76 # percent of bottom trapizoid height
mid_width = .08 # percent of middle trapizoid height
height_pct = .62 # percent of trapizoid height
bottom_trim = .935 # percent from top to bottom to avoid car hood

src = np.float32([[img.shape[1]*(.5-mid_width/2), img.shape[0]*height_pct],
[img.shape[1]*(.5+mid_width/2), img.shape[0]*height_pct],
[img.shape[1]*(.5+bot_width/2), img.shape[0]*bottom_trim],
[img.shape[1]*(.5-bot_width/2), img.shape[0]*bottom_trim]])
offset = img_size[0]*.25
dst = np.float32([[offset, 0], [img_size[0]-offset, 0], [img_size[0]-offset,
img_size[1]], [offset, img_size[1]]])
```

I verified that my perspective transform was working as expected.

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

I identified lane-line pixels in lines 99 through 125 in my code in `image_gen.py`. The window_width is 25 and the window_height is 80. I made a new class called tracker() and function find_window_centroids(). I learned this method from the video of our teacher.

Then I fit my lane lines in lines 139 through 149 in my code in `image_gen.py`. I used a 2nd order polynomial to fit lane lines like this:

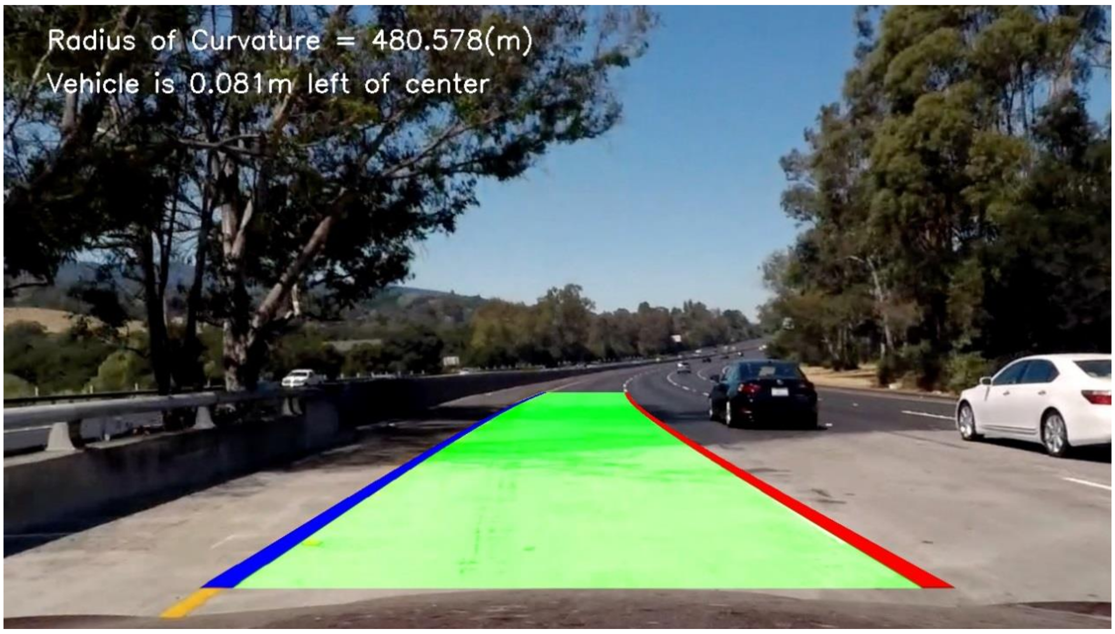


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I did this in lines 180 through 184 in my code in `image_gen.py`.
I used the function that I learned from the course 'Measuring Curvature'.

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 162 through 178 in my code in `image_gen.py` . Here is an example of my result on a test image:


Radius of Curvature = 480.578(m)
Vehicle is 0.081m 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!).

I submitted the video output1_tracked.mp4 with writeup_report.

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?

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.

The biggest problem is how to make thresholds of s in HLS and v in HSV. And I learned them from the tutorial video on youtube. And the second biggest problem is how to convolve windows to find lane lines. I find the best left/right centroid by using past left/right center as a reference.

My pipeline will likely fail when the the positions of lane lines are changed relatively to the camera. Actually it is the position of lane lines in the images we get from the video.

I will change some parameters to make my pipeline robust and the ways to find lane lines with window_centroids.

In this project, I used two methods to find lane lines. One is learned from youtube Q&A session of Udacity(image_gen.py & video_gen.py), and the other one is learned from the classroom(image_gen1.py & video_gen1.py).

I get two results. Output11_tracked.mp4 from video_gen1.py is better.