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.

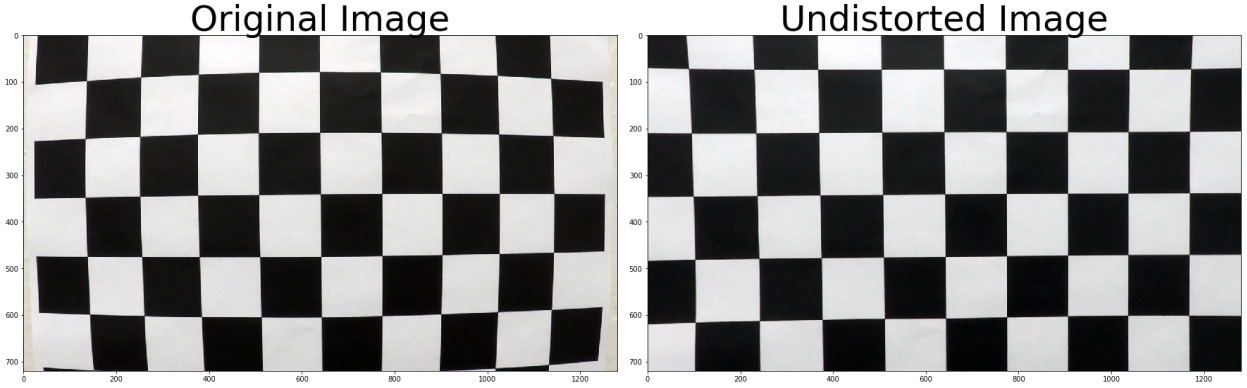
Camera Calibration

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 the second code cell of the IPython notebook located of the file called `some_file.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:



Pipeline (single images)
1. Provide an example of a distortion-corrected image.
The code for this step is contained in the thirteenth code cell of the IPython notebook located of the file called `some_file.py`.
Firstly, I masked the image to get the area I want. The code for this step is contained in the fourth code cell of the IPython notebook located of the file called `some_file.py`.
Next, I undistort the image using this function

cv2.undistort(image, cam_mtx, cam_dist, None, cam_mtx)

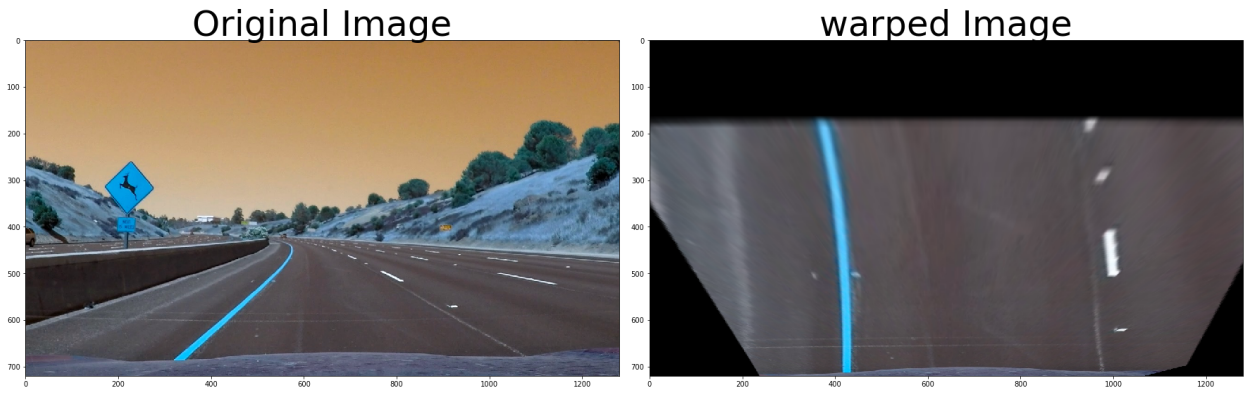
The output is below

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

Firstly, I gave the source and destination points which I want it. I followed this step used in the lecture notes. The code for this step is contained in the seventh code cell of the IPython notebook located of the file called `some_file.py`.
Then, I warped the image using warpPerspective function of the OpenCV. Here is the function

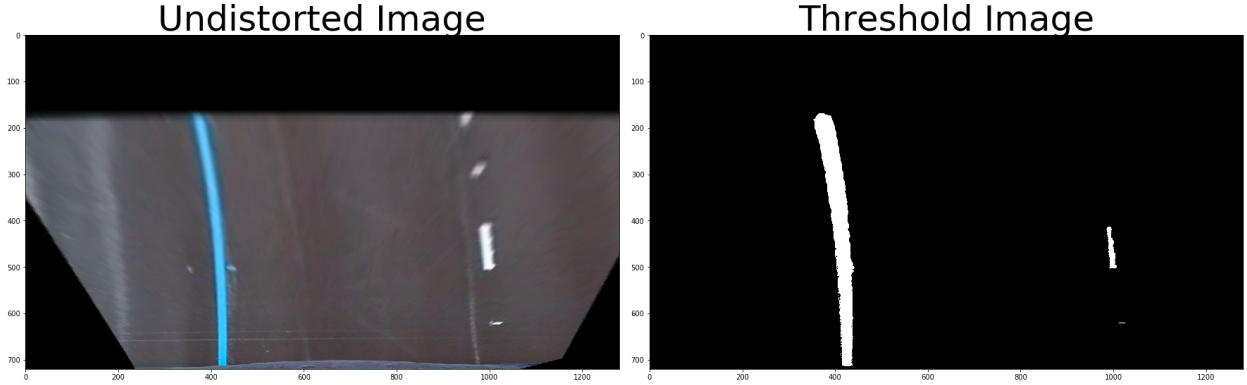
warped = cv2.warpPerspective(undistorted_dash, m, image_size, flags=cv2.INTER_LINEAR)

Below is the output of bird's eye view image with masked



####3. 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.

Firstly, I converted the original image into HLS and gray. Then, I increased the contrast in the gray image. From the HLS image, I took out s channel alone. Then, combined the image.The code for this step is contained in the sixth and fifteen code cell of the IPython notebook located of the file called `some_file.py`.

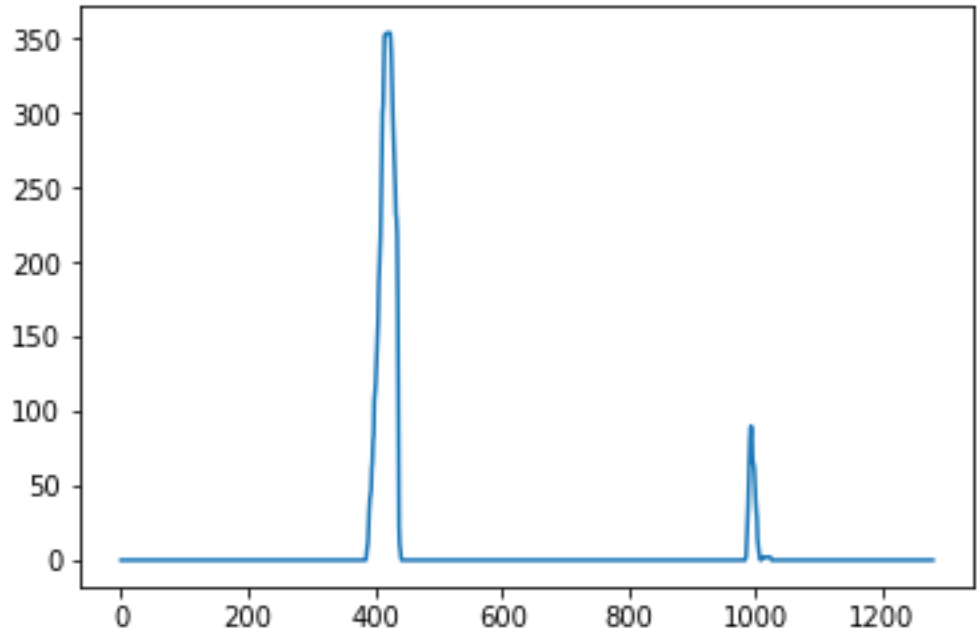


####4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

I used the histogram and sliding window method, as described in the lectures, to identify my lane lines and fit their position with a polynomial.

The code for this step is contained in the sixteenth code cell of the IPython notebook located of the file called `some_file.py`. Here is the function

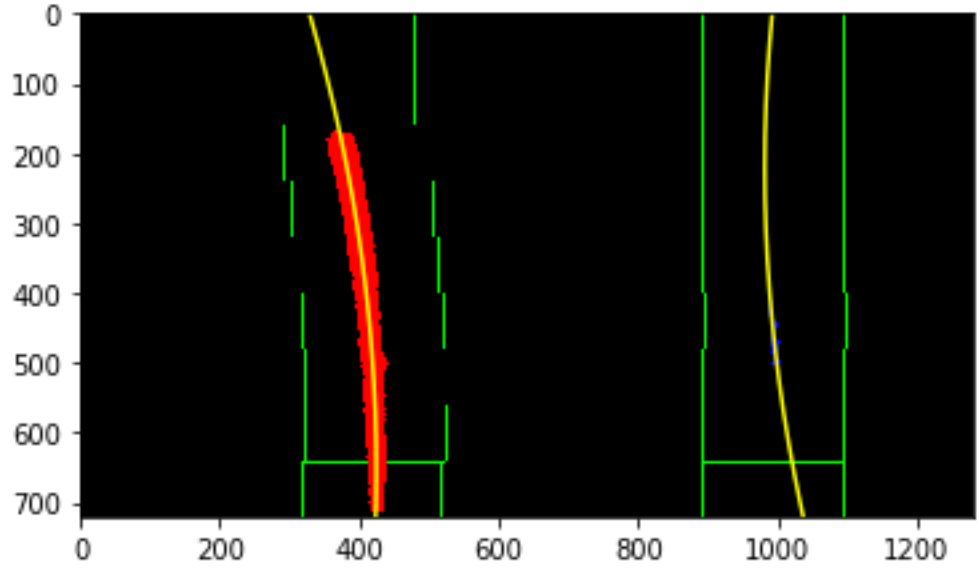
histogram = np.sum(threshold[threshold.shape[0] / 2:, :], axis=0)



After the baselines for the lane lines are identified, a sliding window technique is applied to discover the rest of the lane pixels. This technique divides the image into horizontal where baselines are derived and lane lines can be found and followed up to the top of the frame.

Once the left and right lane line pixels have been extracted, a second order polynomial is applied to each. The image below illustrates the sliding window and polynomial overlays on a binary lane line image

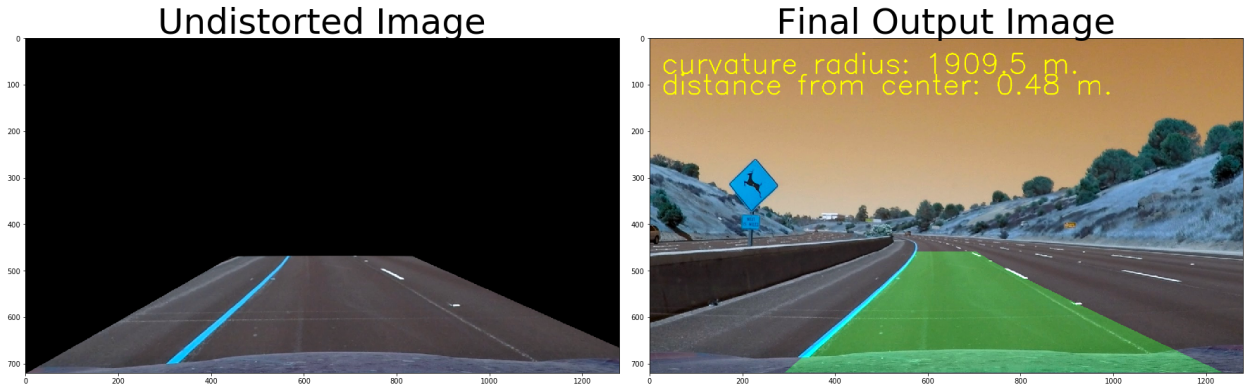
The code for this step is contained in the seventeenth code cell of the IPython notebook located of the file called `some_file.py`.



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.

The curvature of the lane lines were calculated with the algorithm provided in the class material. It plots a second order polynomial to pixel positions for each lane line, then calculates the radius.

The code for this step is contained in the eighteenth code cell of the IPython notebook located of the file called `some_file.py`. also here the pixels are converted into meters



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!).

Here's a [link to my video result](output_videos/project_ video_out6.mp4)

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?

To make the pipeline better, I tried masking technique learned in the finding lanes project but it didn’t come out good for me. Also, in the challenge video, the current pipeline fails in some part. I think because of lots of distortion like shadows and other things. If I manipulate parameters with distortion, I think the pipeline can be more robust than the current one. Also, masking technique can be better.