##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.

[**Rubric**](https://review.udacity.com/#!/rubrics/571/view)**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](https://github.com/udacity/CarND-Advanced-Lane-Lines/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

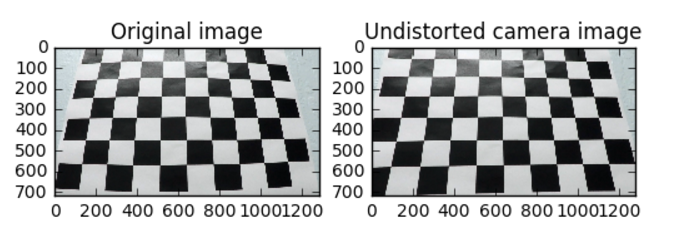
###Camera Calibration

####**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 cells 2,3.4 and 6 of the IPython notebook AdvancedLaneFinding\_V1.ipynb.

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.

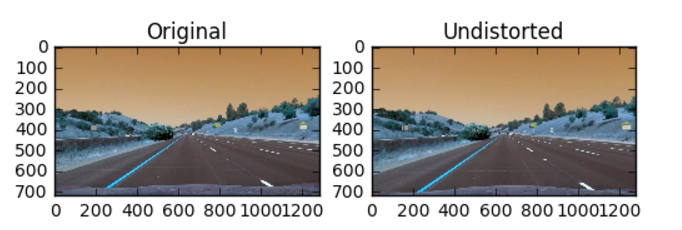
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:



**##Pipeline (single images)**

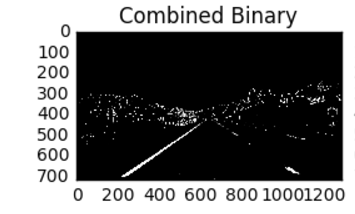
**####1. Provide an example of a distortion-corrected image. To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:**

I pickled the camera calibration coefficients mtx and dst (AdvancedLaneFinding\_V1.ipynb cell 3) and used these in cv2.undistort (AdvancedLaneFinding\_V1.ipynb cell 4) to remove distortion in the test images. See example below of a distortion corrected image



**####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 in function “apply\_threshold” in cell 4 in the file AdvancedLaneFinding\_V1.ipynb and applied to image straightlines1.jpg in cell 13). 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 for image StraightLines1.jpg is in cell 13 in the iPython notebook AdvancedLaneFinding\_V1.ipynb. I chose the hardcode the source and destination points:

src\_points = np.float32([[270,674],

[579,460],

[702,460],

[1035,674]])

dst\_points = np.float32([[270,674],

[270,0],

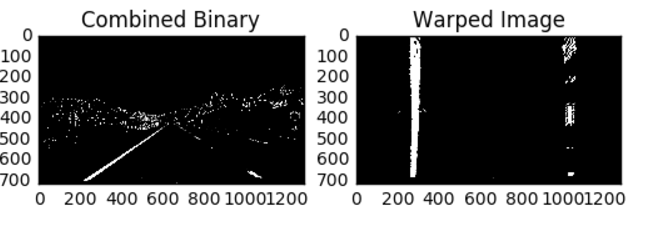
[1035,0],

[1035,674]])

This resulted in the following source and destination points:

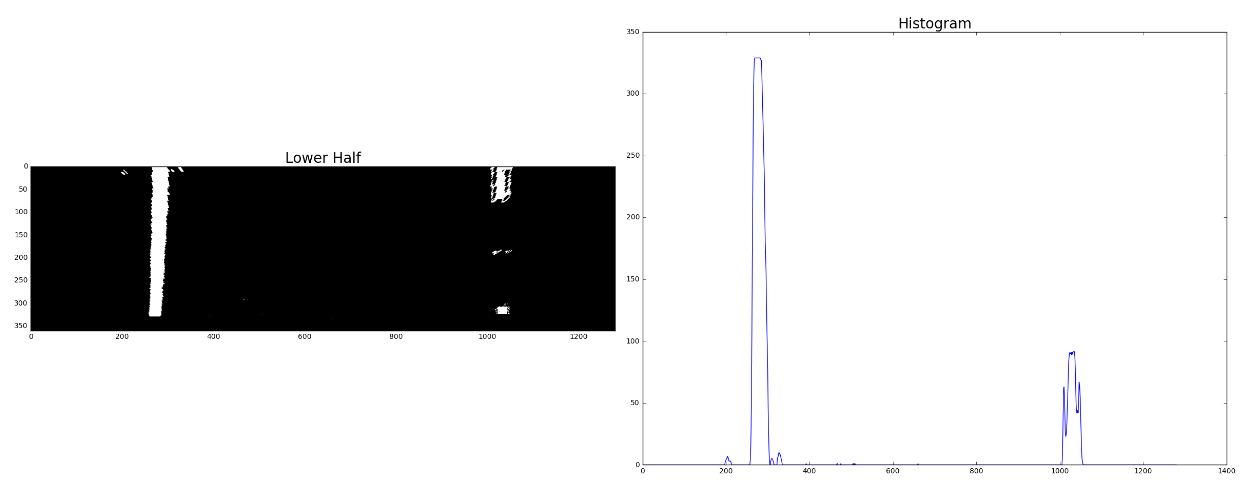
| **Source** | **Destination** |
| --- | --- |
| 270,674 | 270,674 |
| 579,460 | 270,0 |
| 702,460 | 1035,0 |
| 1035,674 | 1035,674 |

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.

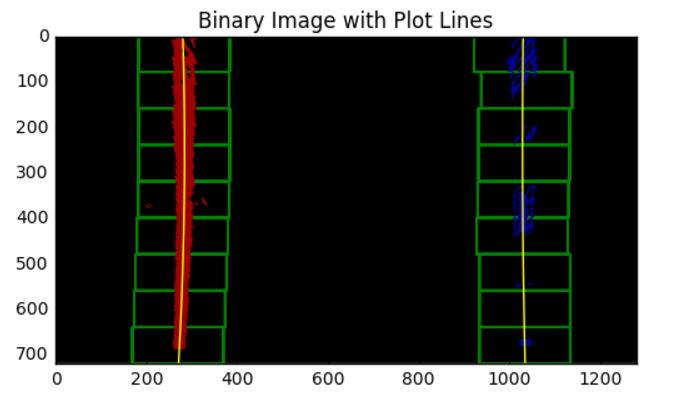


**####4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?**

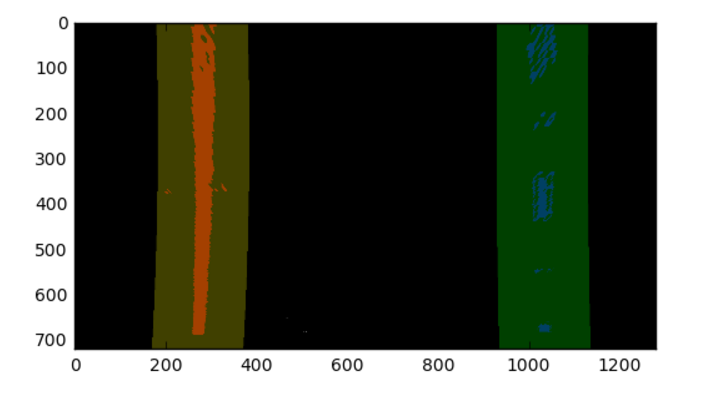
In order to better estimate where the lane is, I used a histogram of the bottom half of image to identify potential left and right lane markings (code cell 14 in the iPython notebook AdvancedLaneFinding\_V1.ipynb).



The two most prominent peaks in this histogram are good indicators of the x-position of the base of the lane lines. I used that as a starting point for where to search for the lines and used a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame (cells 15, 16 and 17 in the iPython notebook AdvancedLaneFinding\_V1.ipynb).



I did some more processing, fit my lane lines with a 2nd order polynomial, created an image to draw on and drew the lines back on to that image. I used code similar to what was used in the lectures. This was the generated image:



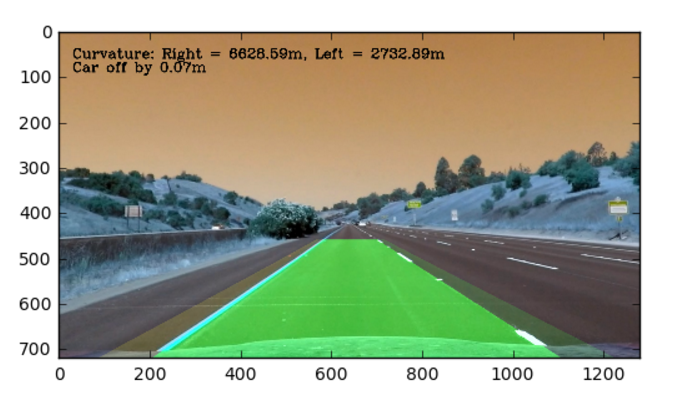
The code for the above steps is in cells is in cells 14 to 19 in the iPython notebook AdvancedLaneFinding\_V1.ipynb.

**####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.**

Radius of curvature was calculated in cell 20 of iPython Notebook AdvancedLaneFinding\_V1.ipynb. I used the code similar to what was in the lectures. I picked the maximum y value corresponding to the bottom of the image and used a formula to find the left and right curve radius. Then after that I repeated the calculation to convert the x and y values to “real world” space. I defined conversions in x and y from pixel space to meters, fit new polynomials to x, y in real world space and calculated the new radii of curvature in “meters per pixel”. The code for the converted radius of curvature is in cell 21 in iPython notebook AdvancedLaneFinding\_V1.ipynb.

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

Then I warped the detected lane boundaries back to the original image. Here is the resulting image showing the lane curvature. The code for this is in cell 23 in iPython notebook AdvancedLaneFinding\_V1.ipynb.



**####6. Creating a pipeline:**

After processing one image, I now decided to create an image pipeline for images. I created functions for fitting the lines (“fit\_lines” and “fit\_continuous”), doing a sanity check (“sanity\_check”) and calculating curvature (“curvature”). I created another function for the image pipeline (“image\_pipeline”). I tested the pipeline on another test image “test4.jpg” and it worked. In the AdvancedLaneFinding\_V1.ipynb file, I have included the pictures for the various steps in processing this image.

I also created another function for the image pipeline for the video (“image\_pipenine\_video”) because for generating the video, we do not need to read the image (input from the movie editor is a read file) and I found after getting an error that the function should return only one variable. The video is included with the documents for this project – test\_video\_final.mp4. The lane lines were identified well in the video and I was happy with it.

**###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?**

One of the problems I had was that when I was running the code to generate the video. The video code (clip1 = VideoFileClip('project\_video.mp4'), out\_clip = clip1.fl\_image(image\_pipeline\_video)) was generating an error when I was calling the function “image\_pipeline”. I searched the forums and realized that for the video I should not be reading the image file because the input from the movie editor is a read file. Also, I was returning more than one variable from the “image\_pipeline” function and that was also causing the error. So, I created another function for the video file pipeline “image\_pipeline\_video” in which I did not read the image and returned only one variable.

For this project, I relied heavily on the code provided in the lectures. I can only become an expert in this if I learn to generate my own code for any picture or video.

Another concern I have is that I used hardcoded parameters for creating the warped image (“src” and “dst” points) and in general this approach relies on tuning the parameters that are camera and track specific. The better approach will be to generalize the parameters for different situations. I really enjoyed working on this project and I was truly amazed when the lane was recognized in the video. This was another fun project and I learned a lot.

**Resources used:**

1. Udacity lectures – extremely helpful!
2. <https://github.com/>
3. <http://stackoverflow.com/>
4. <https://carnd.slack.com/messages/C3CKXKWRK/search/'tuple'%20object%20has%20no%20attribute%20'shape'/>
5. Udacity forums