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.

Please note this is resubmission as some frames were going beyond yellow lane marker. The problem frames are below:

Turned tracking on for this document so that I can reflect what changes were made to result in optimal video with no errors.

###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.

This is the writeup document ☺

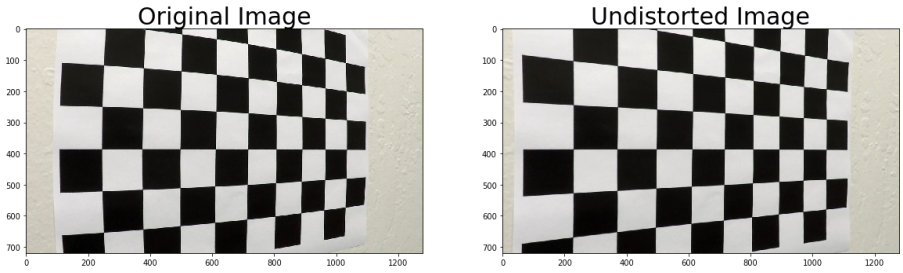
###Camera Calibration

####1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

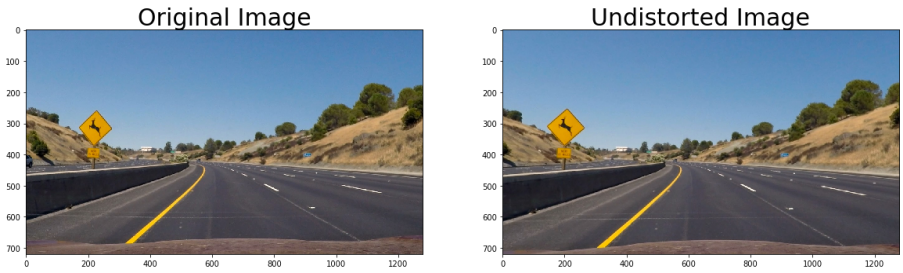
First I understood that the size of the chess board was 9x6. Based on this I read in the calibration images from the /camera_cal folder and used the cv2 function findChessboardCorners. On finding the corners I appended it on object and corners array.

The code for this step is contained in the first code cell of the IPython notebook located in cell#1 of "AdvancedLaneLines.ipynb". Chess board size is 9x6 for this project
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:



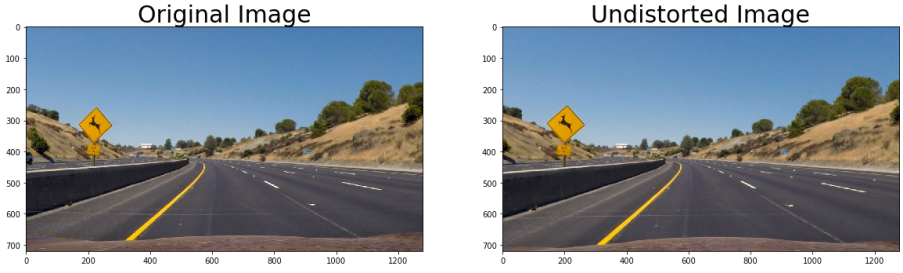
I also fed a test image (test_images/test2.jpg) and the result looks like below. Please note all the images in this document can be found in the jupyter notebook attached as part of the project submission. (examples are also output to the output_images folder)



###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:

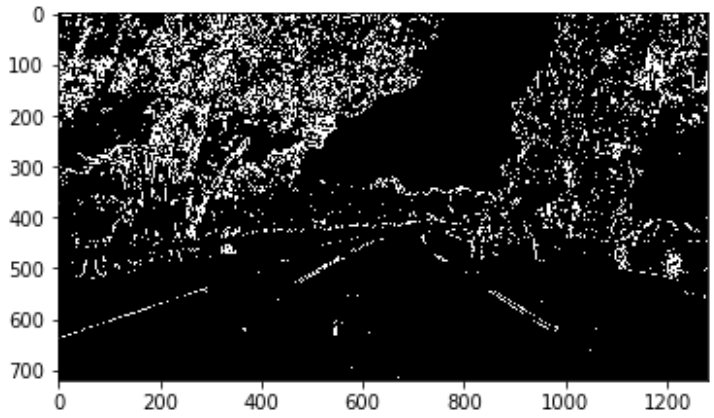
This was done in cell #3 and the sample output is below. Basically the image read in and use the user defined function (undistort_lens) to undistort the image. This user defined function in turn calls the cv2 functions **calibrateCamera** and followed by **undistort**.



####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 # through # in `another_file.py`). Here's an example of my output for this step.

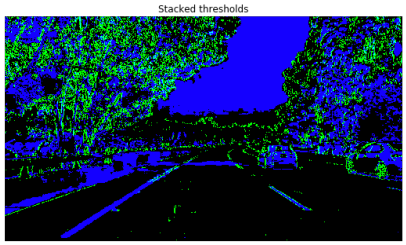
In cell#14 I have used the Sobel operator and applied a threshold of min=20/max=100.

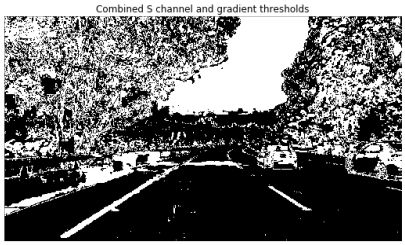
The test output is below:



Then based on the feedback from previous lesson I found that the S channel is still doing a fairly robust job of picking up the lines under very different color and contrast conditions, while the other selections look messy. I tweaked the thresholds for mix=90 and max=255.

The stacked results are below:





####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 includes a function called `get_src_dest_warp_points ()`, which appears in cell#5.
The `get_src_dest_warp_points ()` function takes as inputs an image (`image`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points. I tried to use a logic to derive the points but ended up tweaking it manually using a photo editor.

This resulted in the following source and destination points:

source

```
[[  230.   690.]
 [  585.   450.]
 [  700.   455.]
 [ 1060.   690.]]
```

destination

```
[[  280.   690.]
 [  280.     0.]
 [ 1010.     0.]
 [ 1010.   690.]]
```

Once I changed the perspective transform source and destination points the video produced was great. The errors associated with overflow frame disappeared.

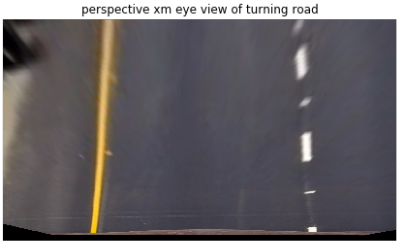
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

test with actual image

Formatted: Strikethrough

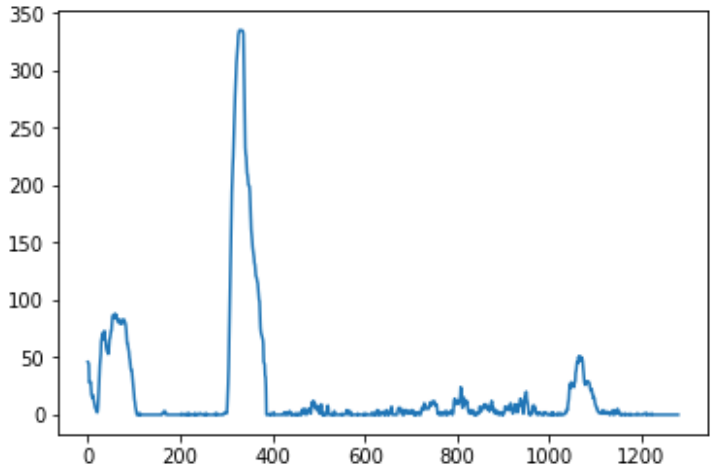
straight road

perspective xm eye view of turning road

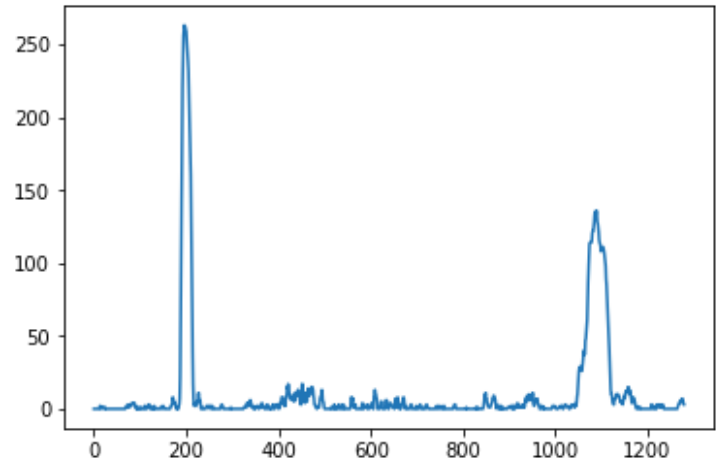


####4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

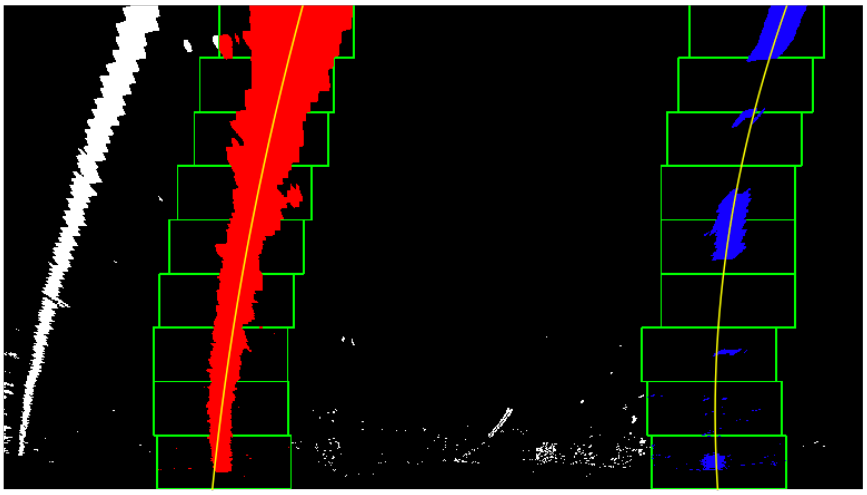
Basically, the first step is to take a histogram to get the peaks within a band in the middle of the frame.



new histogram after changes



functions find_hotpixels_for_lanes() (cell#57) and measure_curvature() (cell#56)are used to find hot pixels for lanes and to fit a polynomial.



####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 cell#57 towards the bottom of the code line.

####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 in cell#33

###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!).

The output video is saved in the output_images folder called project_video_output.mp4 and attached to the project submission with the same name.

Challenge video is called project_video_challenge_output.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?

The pipeline would fail if there are walls or deep lines near the left lane. To make it more robust I would have to rework on the thresholding and maybe crop the images.

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 approach I took is exactly as laid out in the module “Advanced Lane Finding”. This was one of the best taught modules and hope other modules go as deep as this one ☺

1. I used calibration techniques using chess board to calibrate camera
2. Then I undistorted the image using cv2 libraries
3. Then I took into account the lane curvature and fit a second degree polynomial to the hot pixels
4. Also I took a perspective transform of the image from the top
5. After this is mainly picture enhancements to detect the lane lines better. Use of Sobel, gradient thresholds and HLS color space select filter
6. After this I accounted for the curvature of earth

The pipeline fails mainly due to deep lines which can appear next to the left lane line. This can be corrected by using a better cropping technique and also smoothening based on previous lines. If anomaly appears then I basically ignore the lines and transpose the previous line.

After submission reviewer requested I tweak my program so that I don’t have any over flow images beyond the left yellow line.

Problem:

Solution:

Changed perspective transform points and also tweaked HLS s_threah_min = 180 (from 90).

Output frame corrected


Left curve: 2171.61, Right curve: 3190.61
Vehicle is -0.01 m left of center