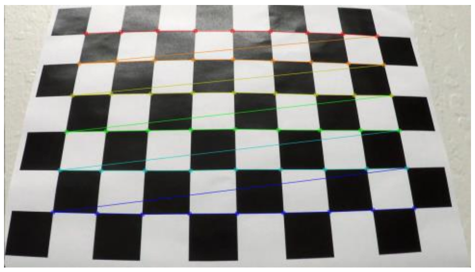
Advanced Lane-finding project write-up

The code created for this project is in my P4 github repository as a Jupyter notebook under P4.ipnyb. The code borrowed heavily from the code discussed in the video lecture, benefited from the Udacity discussion forums, and in many ways is very problematic. While it processes the test images and the project video well, it leaves much to be desired with the two challenge videos (but if the self-driving car followed the path laid out by my code, it would drive similar to me!).
Output images for each stage discussed below are in the output_images folder in the repository, and further organized by folder for the relevant stage. The three output video files are also in the repository: final_projectout, challengeout and hard_challengeout.

Below I will discuss each of the code blocks in the Project4.ipynb notebook; please refer to that file in the repository. **TL;DR—see the red bolded parts.**

Camera Calibration

Using the 20 chessboard images provided, I largely **used the code provided to derive the camera matrix and distortion coefficients using the cv2.calibrateCamera function**. The cv2.findChessboardCorners function was used to detect the corners of the 6(rows) x 9(columns) chessboard pattern. 17 of the 20 images were successfully processed, as 3 of the images did not have all the chessboard corners clearly detected. All **17 images are in the output_images folder**. Below is an example.



Camera distortion correction

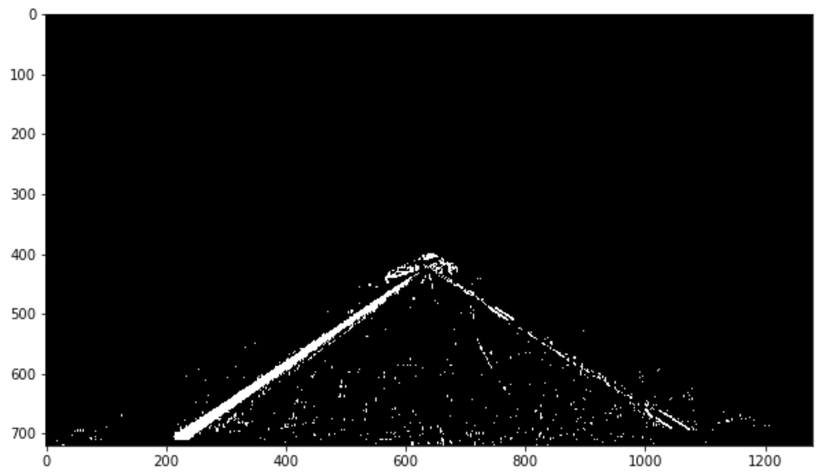
The parameters calculated in the previous step were used to undistort the images taken by the camera using the cv2.undistort function. The **camera matrix, 'mtx' and five distortion coefficients, 'dist' calculated by the cv2.calibrateCamera function were used for the undistortion**. Sample undistorted images are in the /output_images/Undistorted_test_imgs folder, and one sample is also shown in the notebook file. (Note that some of the images have incorrect colors, due to cv2 image process/ mpimg image save step, an issue discovered somewhat belatedly).

Color and Gradient thresholding

This portion of the code is under step 3 of the Project4.ipynb notebook. I explored numerous color thresholding techniques and Sobel gradient transforms; most worked acceptably with the still images, but failed for the video pipeline. I even took a number of still images from the harder challenge video in an attempt to decipher ***the*** thresholding process that would work universally; this was mostly futile and consumed a lot of time.

I settled on **using the LAB color space, along with CLAHE** to make lane line discovery less immune to varying lighting conditions. I scaled back on using Sobel for edge detection for delineating the lane lines and ended up using only Sobel gradient in the x-direction, applied on the Luminance channel (after a CLAHE image normalization). I was a little concerned that the CLAHE process may add to the computation load, but since it was done on only one of the channels(L), it turned out to be not too excessive.

The combined **color threshold binary was bitwise OR combined with the SobelX** binary, and filtered through a triangular region-of-interest mask. Below is an example of the output.

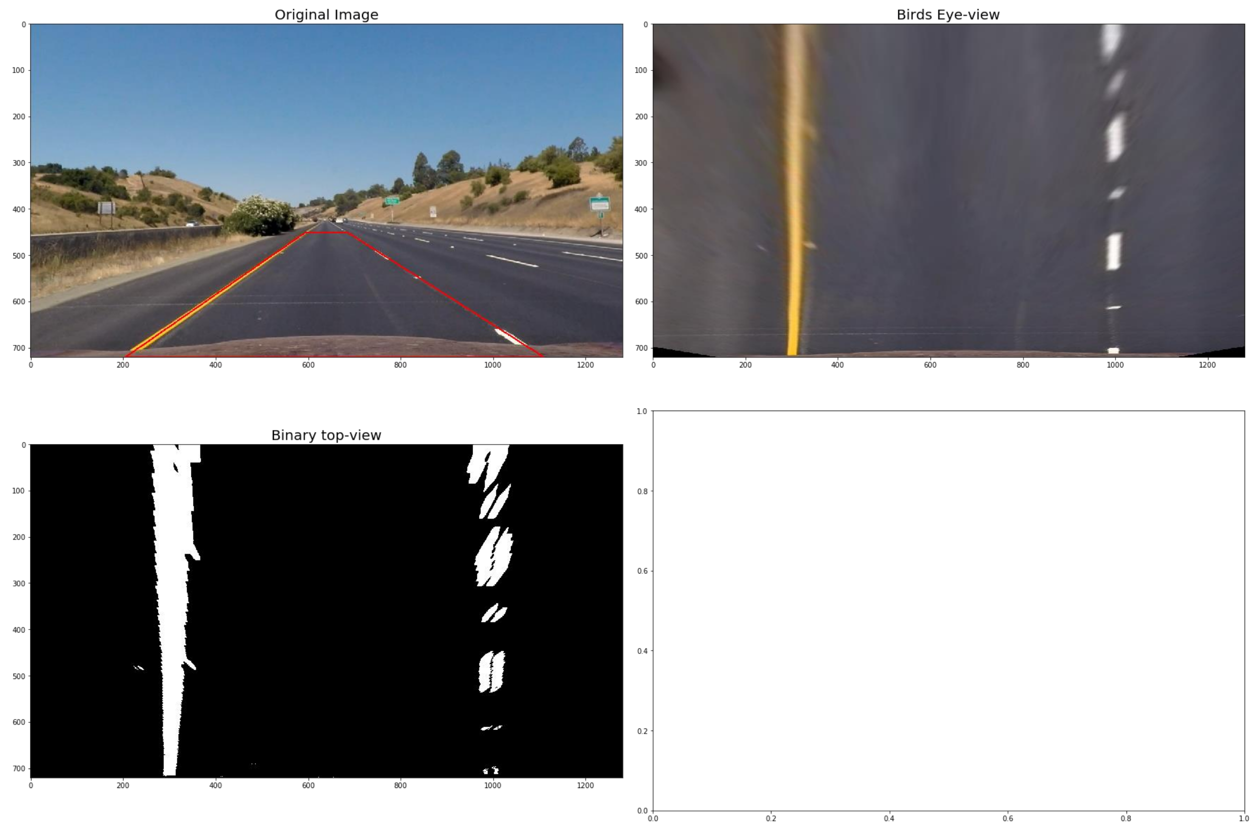


Perspective Transform and Bird's eye-view

The code block for this is in section 4 of the Project4 notebook file. The **source and destination coordinates were hard-coded** in after some trial and error. I am very unsatisfied with the manual intervention required here, and it is **not entirely clear to me that there is a "universal" set of source and destination points**, especially considering the sharp turns in the harder challenge video.

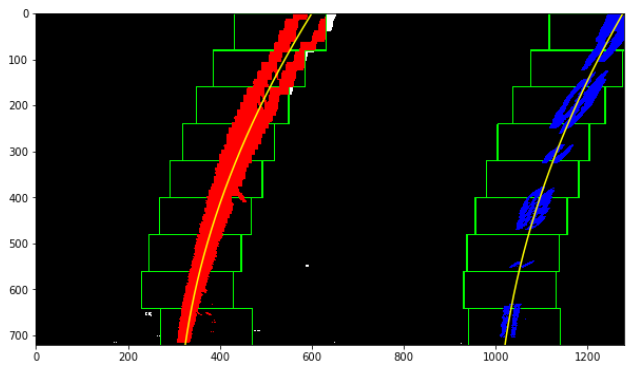
The perspective transform parameter, 'M', and the reverse transform parameter, 'Minv" were calculated using the cv2.PerspectiveTransform function, and then applied onto the undistorted images and the thresholded binary . Examples of bird's eye-view versions and binary bird's eye-view versions of some test images are in the /output_images/Perspective_transformed folder.

Below are a few examples of transformed images and a transformed binary output.

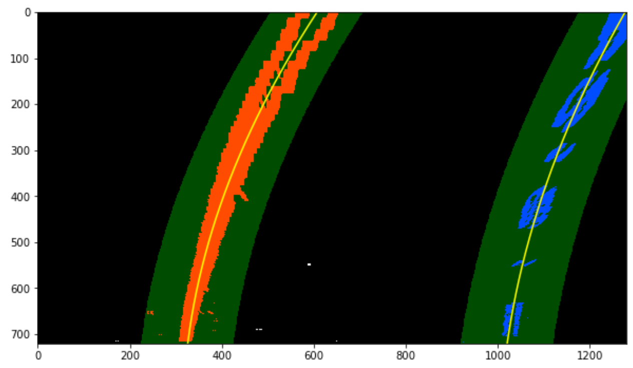


Lane line identification

The code for identifying lane lines using the sliding-windows technique is in section 5 of the project notebook, and is largely unchanged from the code in class. An example of the output is shown below.



In section 6 of the notebook, is the code for the more efficient lane line finding after the polynomial line fit parameters (left_fit and right_fit) are calculated in section 5. For the processing of consecutive frame-stills from a video file, where the lane line positions should not shift significantly between frames, looking in the vicinity of the last good known position of the line should be much quicker. The algorithm looks for a lane line within 100 pixels to the left and right of the last known good position. An example of the output for the same test image as the previous picture is shown below. Other examples are in the /output_images/Lane_Search folder of my repository.

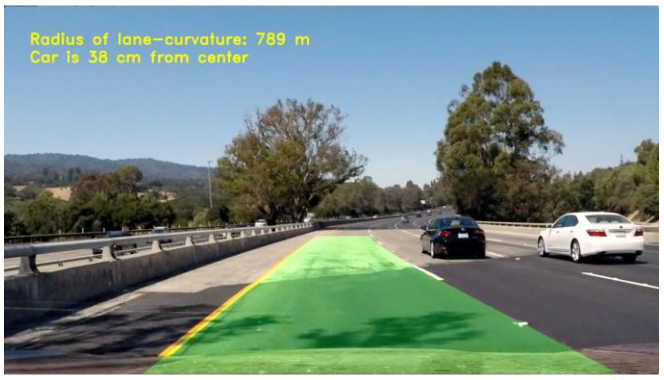


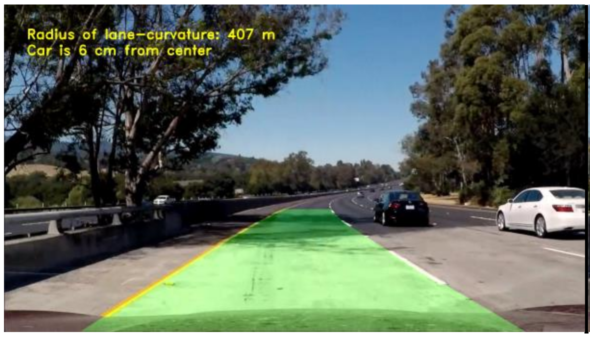
Lane line curvature and vehicle position estimation

The code for this is in section 7 of the notebook, and **uses the polynomial lane line fit derived in the last section to determine curvature** of the left and right lane lines. The position of the car is estimated from the x-intercept of the two lane lines, and the position of the center of the image frame relative to the difference of the two lane x-intercepts.

Lane highlighting

The final step is the highlighting of the path for the car to travel on, and this is done by "projecting" the calculated ends of the left lane-line and the right lane-line onto the regular-view of the road surface using the cv2.warpPerspective function with the 'Minv' parameter calculated in step 4. Below are three sample outputs; additional images are in the /output_images/Final_output_images folder of my repository.





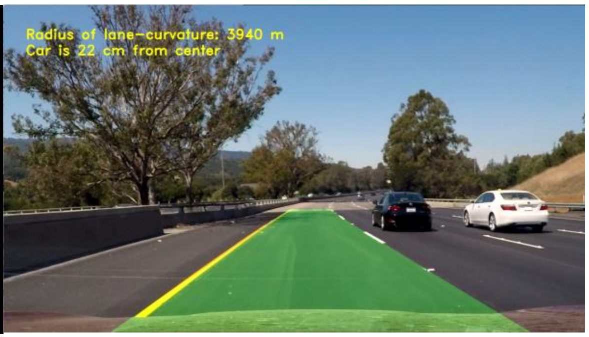


Image processing pipeline

In order to easily run the operations listed above on individual images and the stills from the video file, the code was reformulated as functions, and the final process_image function is listed in section 10 of the notebook. For individual images, or the first still frame from a video file, the function invokes a sliding windows search for the lane lines. For subsequent stills from the video, the more efficient faster_line_predictor function is used. A line **smoothing function that averages up to 15 previous still frames** is used to reduce jitter in the lane prediction. The video processing code is in section 12 of the notebook.

Discussion of results

The **heart of this project was the image thresholding** in section 3 of the notebook. This was incredibly time consuming, and I think the **process of choosing appropriate parameters was rather inefficient**. The challenge was in finding parameters that would work for all the test images and also all of the test videos. I could find parameters that worked well on the test images but would then fail for the video. I even took some still images from the hard_challenge_video, but eventually gave up in hopes that the frame-to-frame averaging function would save me.

I tried to use an adaptive lane detection approach using CLAHE, which I was told helped in the traffic signal classifier project. It may have helped, but the challenge video and hard-challenge video output is still not perfect. **There has got to be a simpler approach to lane delineation than this**. Clearly the Nvidia approach https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/ used a neural net based approach to driving without explicitly laying out a path, but that requires a lot of data.

I would like to improve further the image thresholding process to make this algorithm more robust. Also I'm not happy that I had to hard-code the source and destination coordinates for the perspective transforms. I would like to derive an automated scaled derivation of these points that will work in a wider set of conditions.

My code will fail when sharp turns are involved, road lighting is poor (especially at night), and when there is glare on the camera. Unfortunately, I felt the learning was not as commensurate with the time I spent on this project