Advanced Lane Finding Project

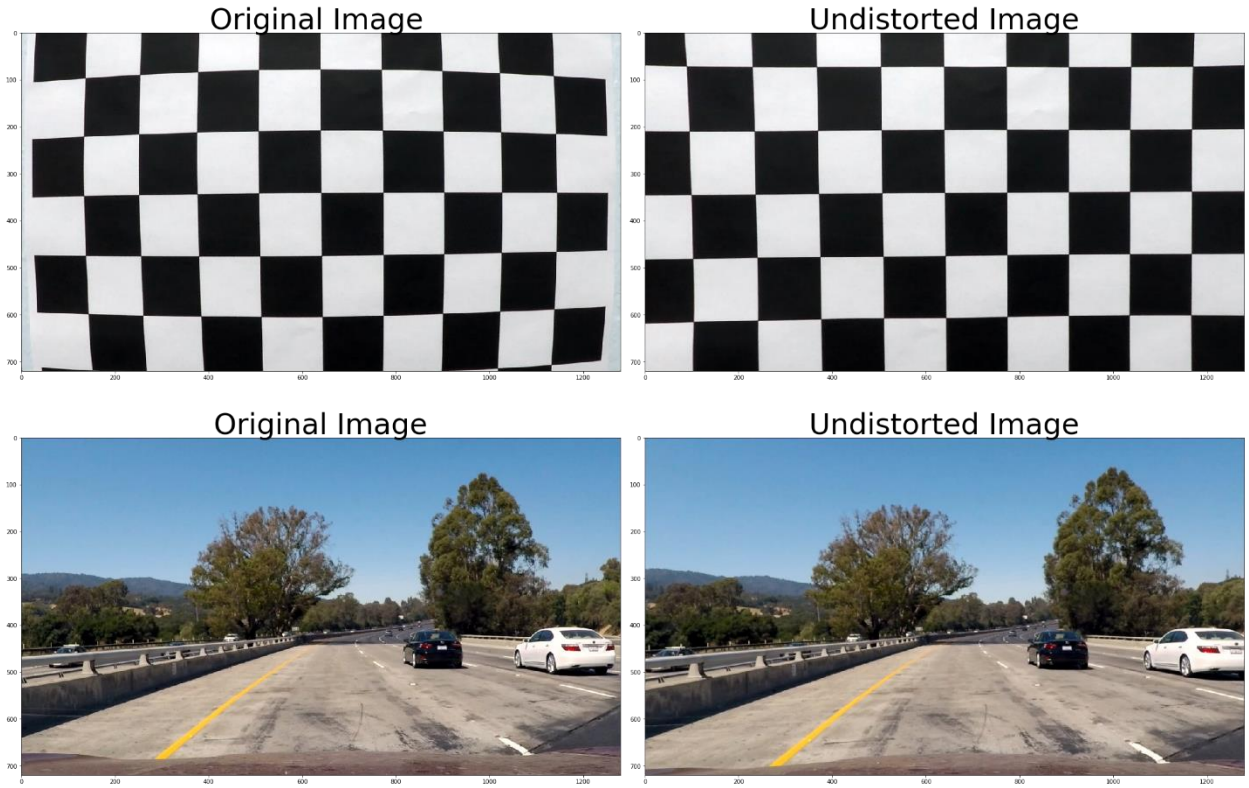
The goals / steps of this project are the following:

- 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.

Step 1: Camera Calibration and distortion correction

The camera was calibrated using chessboard images that was taken. We can calibrate the image taken by finding the corners of the chessboard (image points) ,using cv2.findChessboardCorners(), and comparing it to the standard coordinates of chessboard corners (object points).

The distortion coefficient (dst) and camera matrix (mtx) can then be computed using cv2.calibrateCamera() with the inputs of the image points and object points. We can now undistort any images that is taken by this camera using cv2.undistort. Examples is shown below:

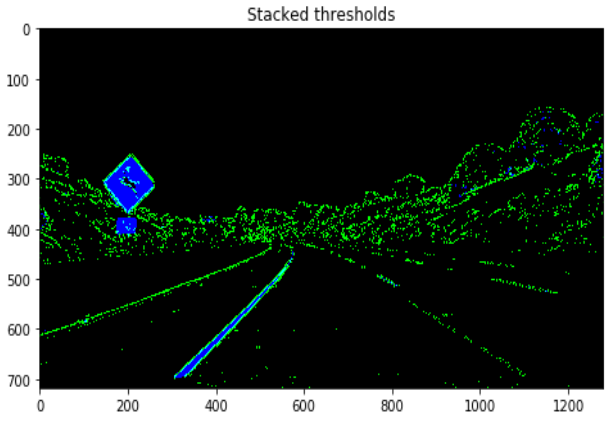


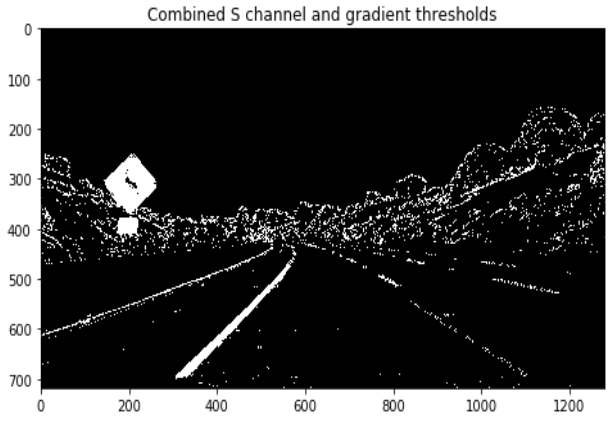
Step 2: Color and Gradient Thresholding

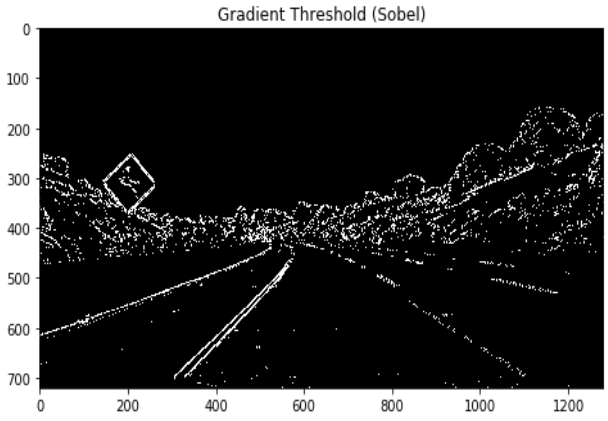
The color space was converted to HLS and Saturation was selected as the specific color layer that we are thresholding (filtering). The selection was done based off plotting the image on a few different color spaces and various thresholds. The best selection was determined by the prominence of the lane lines in the image. Color thresholding is very important as we can filter away unwanted noise such as shadows.

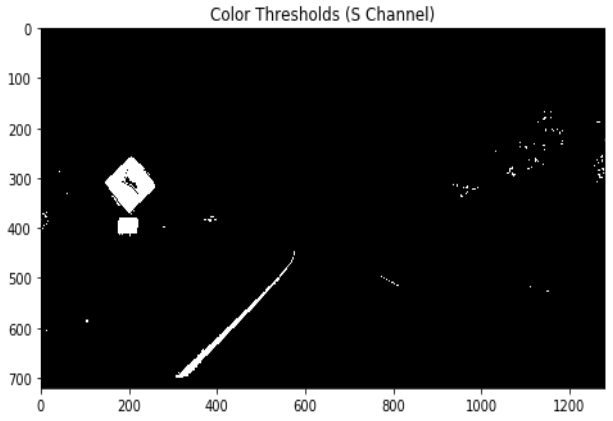
The gradient thresholding was done using Sobel operators. Both color and gradient thresholds returns a binary image which was then combined. The individual plots for gradient threshold and color threshold can be seen below. The combined thresholds are of satisfactory standard as the lane lines are clearly defined. Gradient thresholding can be seen in the code under the function combined_grad while color thresholding is in the function color_threshold. (Cell 5)

After project feedback, R channel was filtered to get rid of shadow.









Step 3: Perspective Transform

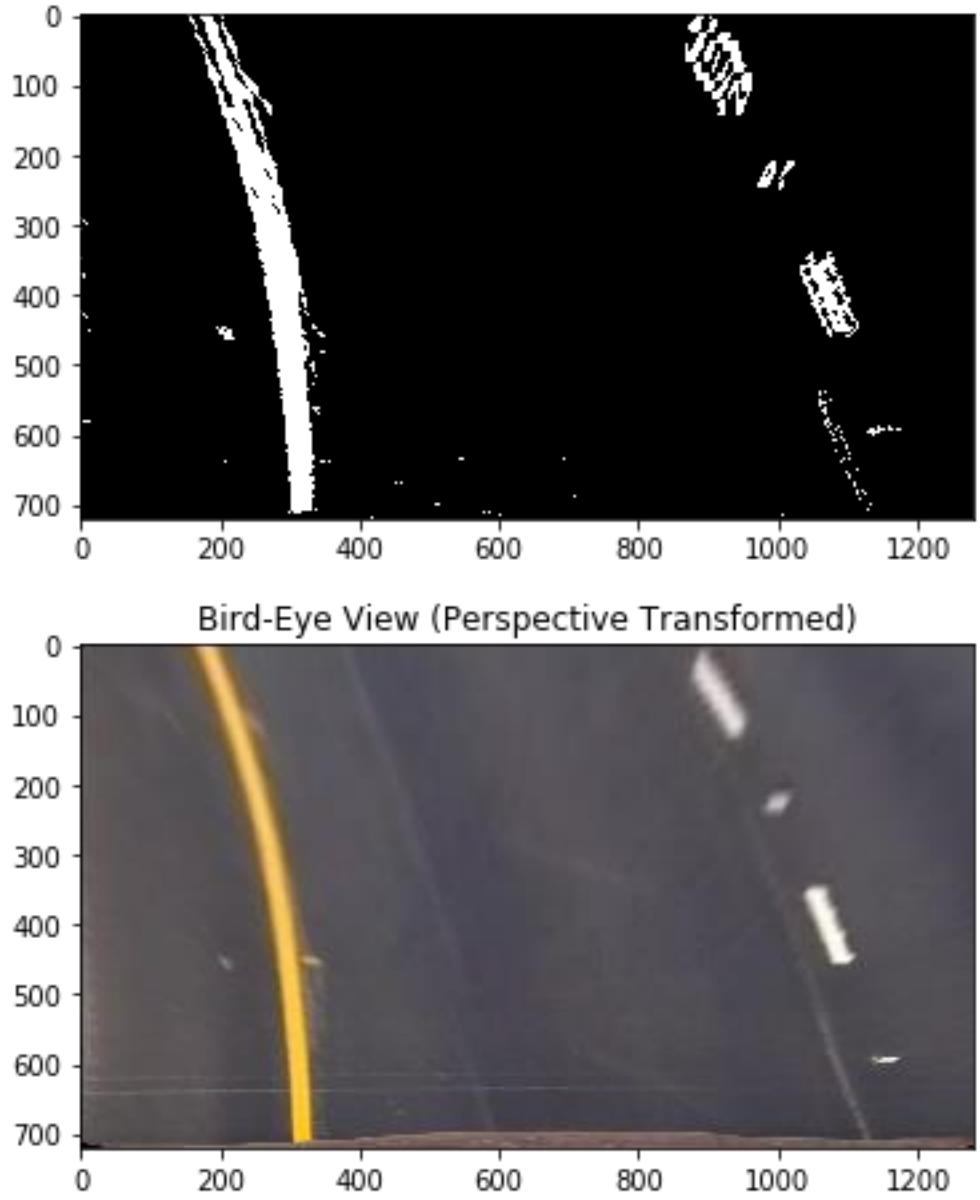
The binary image from the threshold filtering was then warped to get a bird-eye view of the lane lines. This is necessary to better detect curve lines and compute the polynomial equation of the curve lane line. Firstly, we can looking to find 4 coordinates points (source coordinates) to form the a trapezoidal shaped region of interest around the lane lines. The images below show how we can gauge the coordinates by manipulating the position of the dots.

Source	Destination
580,460	230,0
200,719	230,720
1160,719	1150,720

740,460	1150,0

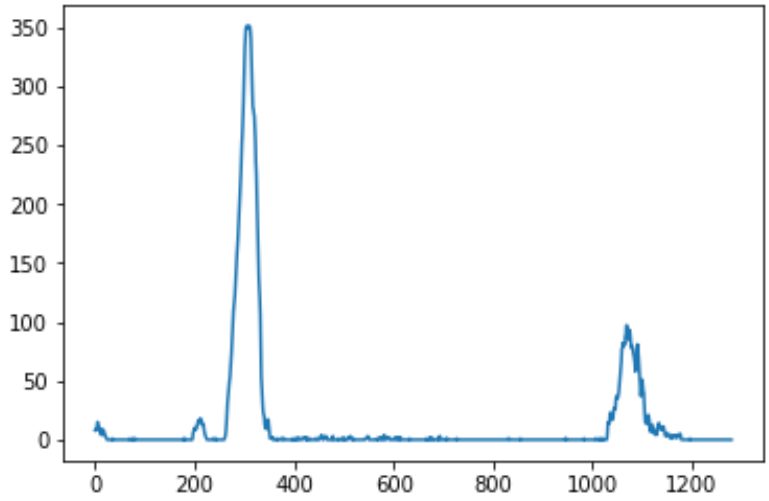
Once we are happy with the source coordinates, we can warp it to the destination coordinates. The destination coordinates are determined by trial and error to get a bird-eye view image with the lane lines clearly defined and also removing the noise such as the divider at the side.

With the source and destination coordinates, we can use the functions cv2.getPerspectiveTransform to get the transformation matrix required for warping. The function cv2.warpPerspective is then use by inputting the transformation matrix found to produce the warped image. The images below shows the warped image of the road.

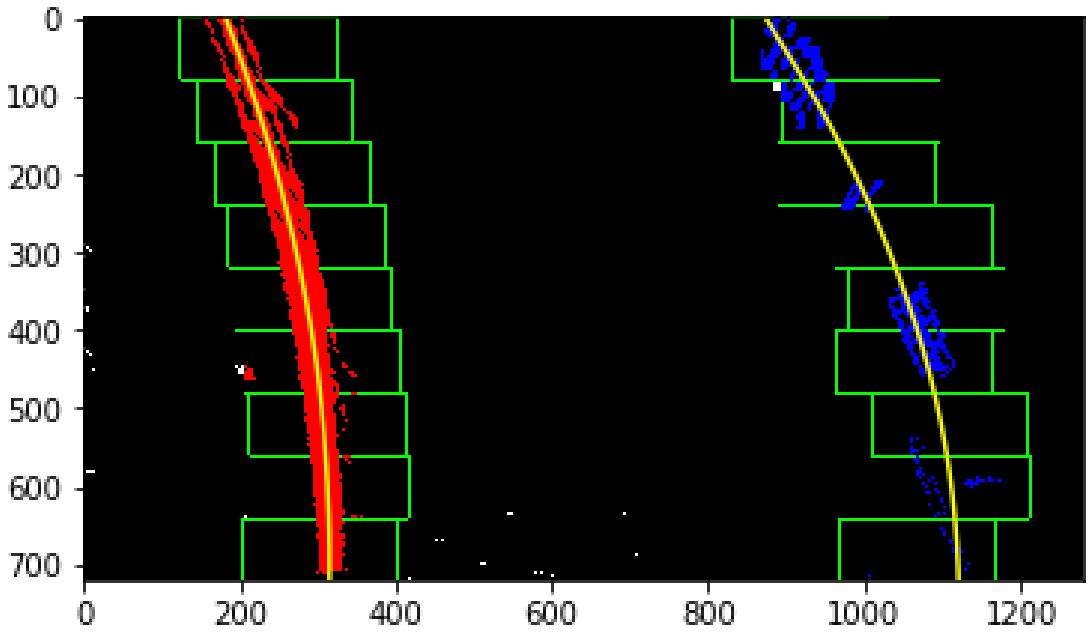


Step 4: Identifying lane-line pixels and fitting polynomial

Using the warped binary image, we can then apply the sliding windows technique to find the center of the pixels (lines) along the y-axis. The starting point of the sliding window was determined by finding the points where the pixel intensity is the highest. This can be visualized using by plotting the graph:



It can be observed that there 2 obvious peaks where most pixels are located at which gives us the starting point for the sliding window. We can determine the center line of the lanes through the sliding windows. With the coordinates of the pixel positions, the function np.polyfit can be used to produce the coefficient of the quadratic equation. The results was visualized as seen below:



Step 5: Determining radius curvature of lane lines and the position of vehicle with respect to the center

As we now know where the lane lines are, we can estimate how much the road is curving and where is the vehicle located with respect to the center of the lane. Note that the pixels in the image are converted to meters for calculations. These are radius of curvature is calculated in the function find_lane (Cell 10) and the center offset of the vehicle is found in the function drawLine (Cell 11). The center offset is determined by finding the difference between the center of the image to the center of the lanes.

The end expected result of this project for an image:

Pipeline (Video):

Go to the file -> test.mp4

Link: https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/5d1efbaa-27d0-4ad5-a67a-48729ccebd9c/lessons/7cb63828-36aa-4cea-9239-700b5ea41f0b/concepts/0a96d23f-6c22-4053-a7f6-83e12ce5a6ec

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 main challenges faced in this project is figuring out the most suitable thresholds for the Sobel operators and specific color spaces to output a binary image that has clearly defined lane lines. I plotted the results from each threshold individually to get some idea on what the best combination for will be overlapping the various binary images. One of the issues were that the left curvature values is sometimes wrong due to the lane lines breaking up which causes the polynomial fit to be incorrect.

The pipeline will most probably fail under conditions where the curve is of shorter curvature. The implementation of the pipeline will not be able to detect as the region of interest is rigidly fixed. Therefore, the warped image under this condition will be erroneous. Besides that, I believe the pipeline will always fail under certain road conditions such as when it is raining (camera noise) or when there are a lot of shadows.

The pipeline is now is very slow in computing. I cannot manage to figure out how to incorporate previous information of the sliding windows and polynomial found to speed up the next calculation.