Camera Calibration

Criteria	Meets Specifications
Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.	OpenCV functions or other methods were used to calculate the correct camera matrix and distortion coefficients using the calibration chessboard images provided in the repository **(note these are 9x6 chessboard images, unlike the 8x6 images used in the lesson)**. The distortion matrix should be used to un-distort one of the calibration images provided as a demonstration that the calibration is correct. Example of undistorted calibration image is Included in the writeup (or saved to a folder).

I created a function called “FindCorners” to find the corners on the calibration images provided with the repository.

Within the function, I am looping over all the images, read the files one by one, grayscale them and call **cv2.findChessboardCorners**. The number of corners: 9 horizontally, 6 vertically which I store in variables.

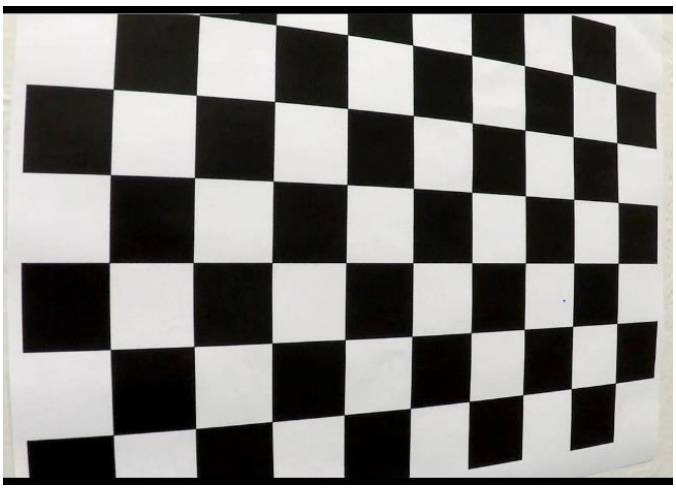
the result of **cv2.findChessboardCorners** is the detected coordinates of the corners on the chessboard image (calibration image corners) which we link to 3D real chessboard image corners (undistorted image corners). The format of the coordinates of the 3D image corners is (x,y,z), but z=0 always! Numpy.mgrid generates the coordinates for the 3D image.

I use the detected calibration image and undistorted image corners to calibrate the camera with function, **calibrateCamera of cv2**. The function return 5 values, but for calibration we need only the distortion coefficients and the camera matrix we need to transform 3D object point to 2D image points.

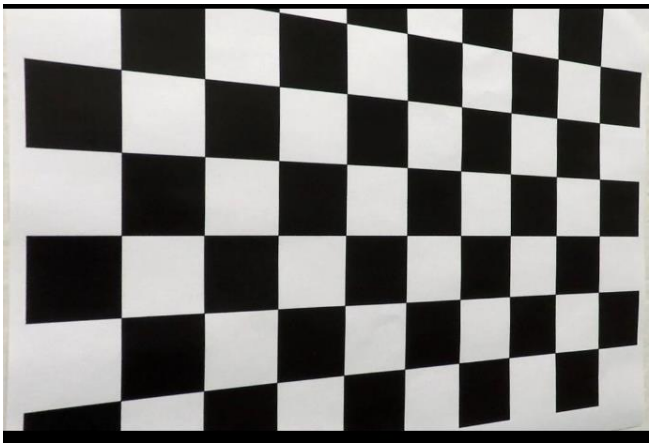
To undistort an image, I used **cv2** function **undistort**, which takes the calibration image, distortion coefficients and camera matrix as input and return the undistorted image.

This is how the distorted and undistorted images look like:

DISTORTED



DISTORTED

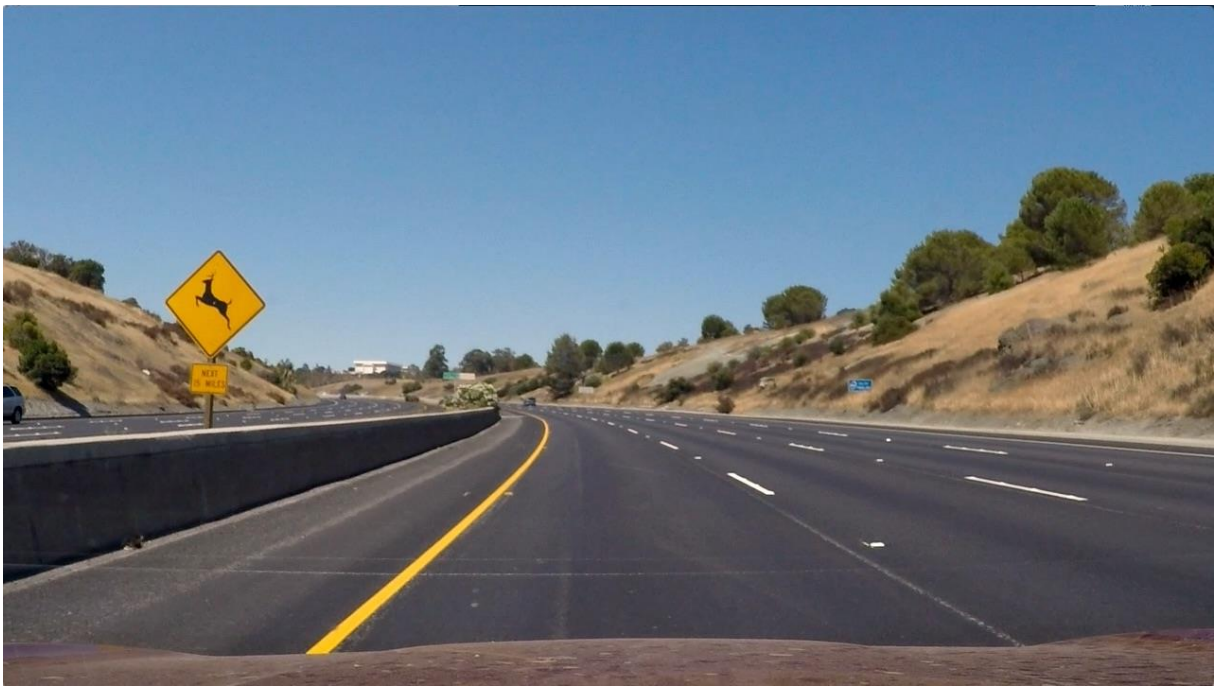


Pipeline (test images)

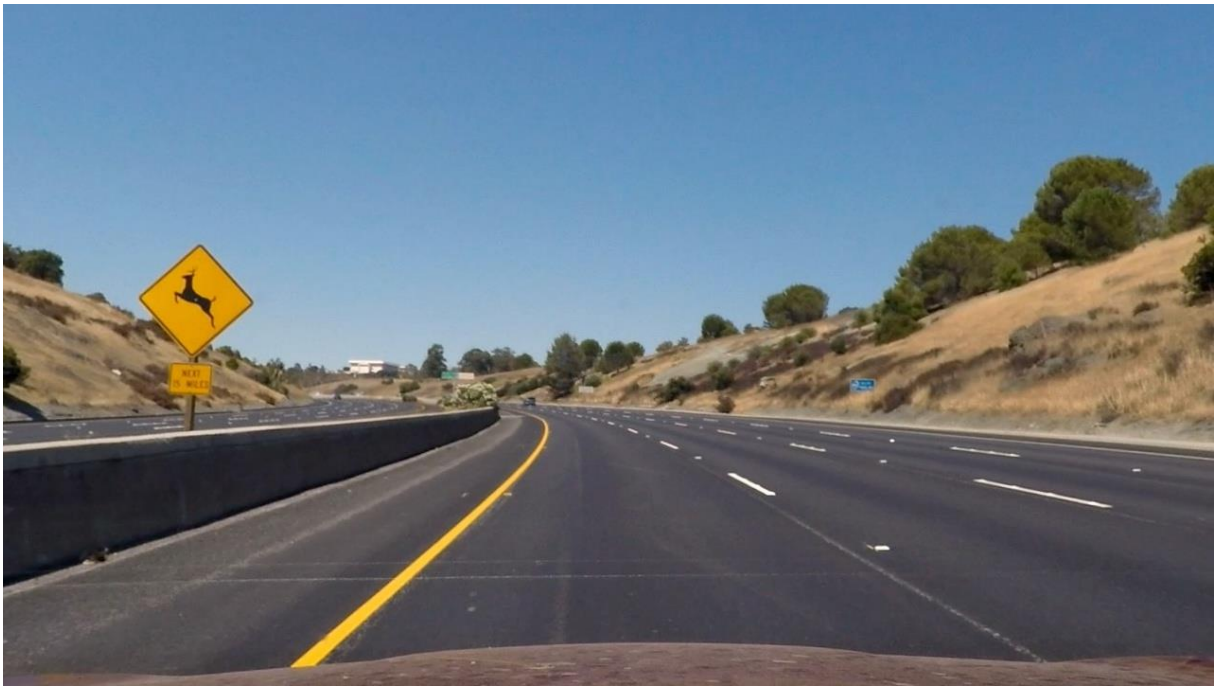
Provide an example of a distortion-corrected image.	Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.

To correct a distorted image, I applied the above mentioned cv2.undistort function to all the test images. The previously determined distortion coefficients and camera matrix were used as parameter.

This is a distorted image:



Undistorted image (the space between traffic sign and the edge of the image is less than on the original image):



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.	A method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.

To create a binary image, I applied the following steps (cell 5 in the code, functions are defined in cell 2):

1, applied Sobel operator in the “x” direction to detect lane line edges closer to vertical. Threshold that gave the best result for me: 20, 255

- converted undistorted image to grayscale
- calculated derivative to “x” direction
- took absolute value of derivative
- convert absolute value to int8
- used threshold

2, applied Sobel operator in the “y” direction to detect lane line edges closer to horizontal. Threshold that gave the best result for me: 20, 255

- converted undistorted image to grayscale
- calculated derivative to “y” direction

- took absolute value of derivative
- convert absolute value to int8
- used threshold

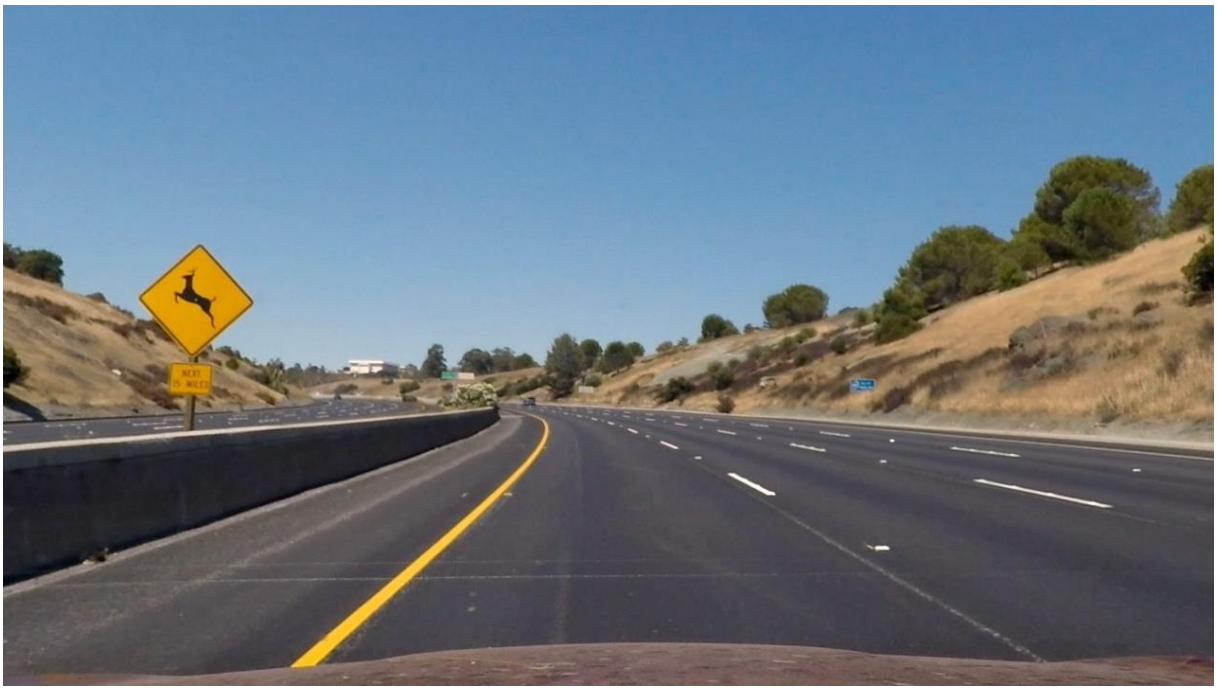
3, applied color threshold to the undistorted image. Steps:

- converted RGB image to HLS
- took S Channel
- used “S” threshold: 90, 255
- converted RGB image to HSV
- took V Channel
- used “V” threshold: 80, 255

4, combined “S” and “V” threshold

5, combined Sobel “X”, Sobel “X” and color threshold to get binary image

Undistorted image:



Binary image:

Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.	OpenCV function or other method has been used to correctly rectify each image to a "birds-eye view". Transformed images should be included in the writeup (or saved to a folder) and submitted with the project.

to do perspective transform I called function **cv2.getPerspectiveTransform(src, dst)** which computes perspective transform parameter "**M**" (cell 5 in the code, functions are defined in cell 2).
Input for the function are "src" and "dst".

"src" contains the coordinates of 4 points on the binary image. I defined rates that I used to calculate the coordinates of those 4 points (polynom. I ignored the bottom part of the image as it displays only a part of the car, so it is not important for us. I also ignored the upper part of the image (cut off the upper 30% of the image to ignore objects like trees). The bottom part of the rectangle is 83% of the image width, the top part of the rectangle is appr. 20% of the image width. I tried to follow the appearance of the lane line on the image.

"dst" defines the coordinates of the selected 4 "src" point where we want them to appear on the warped/transformed image. As "x" coordinates I use (image width*0.03) on the left side and (image width - image width*0.03) on the right side. "y" is 0 and image height (720).

I also computed the inverse perspective transform (Minv) for later use to convert image back from warped image to the real one with cv2.getPerspectiveTransform(dst, src). The difference is that parameters are passed to the function reversed (first dst, second src).

As final step, I used cv2.warpPerspective to transform the image. The function takes the binary image, "M" and image size as parameters to do the transformation.

Binary image:

Wrapped image:

Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?	Methods have been used to identify lane line pixels in the rectified binary image. The left and right line have been identified and fit with a curved functional form (e.g., spine or polynomial). Example images with line pixels identified and a fit overplotted should be included in the writeup (or saved to a folder) and submitted with the project.

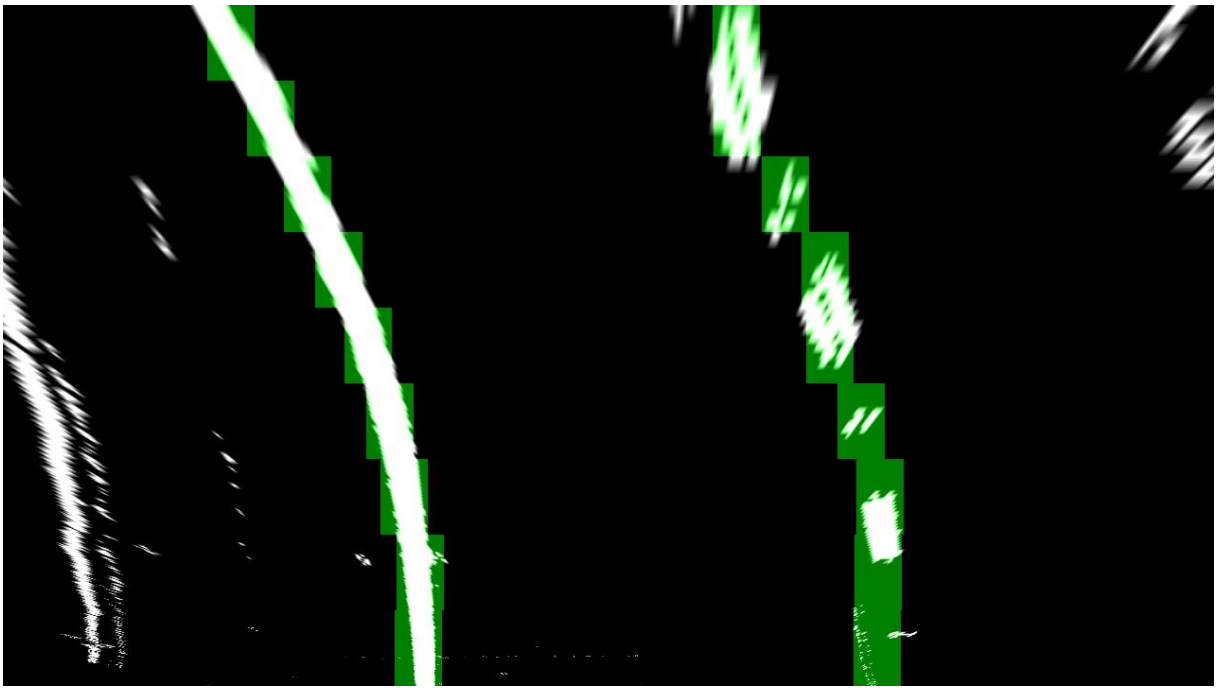
To identify lane lines on the warped image I used the sliding window method with convolution (cell 5 in the code, functions are defined in cell 2).

Steps:

- defined window size within the program looks for pixels (width = 50, height = 80) and margin to slide to left or right (100).
- First find the two starting positions for the left and right lane by using **numpy sum** to get the vertical image slice and then **numpy convolve** the vertical image slice with the window. Where we have peak values (**numpy argmax**) on the left and right side of the image there we start search lane lines, they are the center point of our search.
- then I go level by level (image height divided by the window height, 720/80 = 9) to find pixels and determine new window centers on the right and left side.
- by doing this I am trying to figure out which pixels belong to the left and right lane lines
- used "**numpy polifyt**" to find a line that best fit to the pixels determined on the left and right side in previous step.
- marked the left and right lane lines on the wrapped image

Wrapped image:

Marked Wrapped image:



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.	Here the idea is to take the measurements of where the lane lines are and estimate how much the road is curving and where the vehicle is located with respect to the center of the lane. The radius of curvature may be given in meters assuming the curve of the road follows a circle. For the position of the vehicle, you may assume the camera is mounted at the center of the car and the deviation of the midpoint of the lane from the center of the image is the offset you're looking for. As with the polynomial fitting, convert from pixels to meters.

to calculate line curvature and position of the car I did the following calculation in cell 5 in the code:

I located the lane line pixels, used their x and y pixel positions to fit a second order polynomial curve. It is better to calculate the curve for f(y) as the lane lines are almost vertical. „y" values were determined by numpy arange that returned evenly spaced values along the width of the image.

Curvature is calculated by the following mathematical formula:

R =((1+(2Ay+B)**2)****3/2)/|2A|

To convert the value to real world space I defined conversions in x and y from pixels space to meters and used in the curavture calculation:

- 30/720 : meters per pixel in y dimension
- 3.7/700 : meters per pixel in x dimension

To calculate the position of the vehicle with respect to the center I:

- first calculated the midpoint between the detected lane lines (polynomial fits ont he right and left side)
- subtracted the midpoint of the image from the calculated value center point
- in order to get real world distance I multiplid it by the previously deifned conversion for „x” dimesnion

I display the calculated curvature and vehicle position on the image:

Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.	The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and position from center should be included in the writeup (or saved to a folder) and submitted with the project.

Finally, I did the reverse perspective transformation with **cv2.warpPerspective**. I used inverse perspective transform parameter (Minv) that I calculated earlier (cell 5 in the code)

Unwrapped, undistorted image with marked lane lines with information in regards with the curvature and vehicle position:

Pipeline (video)

Criteria	Meets Specifications
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 image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.

The video marked with the lane lines can be found on github, filename “output.mp4”:

https://github.com/martonistvan/CarND-Advanced-Lane-Lines-master.git

In order to avoid blind search for each frame in the video I implemented “Line()” class to store some information about the actual findings.

I use the by polyfit calculated left X and right X values as a starting point for the upcoming images.

Discussion

Criteria	Meets Specifications
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?	Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

It was not easy to find the proper threshold values. Initially there were frames (images) in the video where the left side of the road is a bit lighter so when I was looking for lane lines on the binary image I detected them incorrectly as part of the line. With some adjustment of the threshold values I managed to overcome the issue.

For me it was the most difficult part, took time to try out different scenarios, combination of x and y Sobel together with color threshold.

What I was wondering and tried out if it is possible to do first a perspective transformation and then apply any color threshold. It gave good result as well.