Advanced Lane Finding

This project involves application of several image processing techniques to detect lanes under different road conditions.

The goals of this project are as following:

- Computing the camera calibration matrix and distortion coefficients using a set of chessboard images.
- Applying a distortion correction to raw images.
- Using sobel tranform to compute image gradient magnitude and direction.
- Applying threshold on gradient magnitude and direction to form thresholded binary image.
- Using HLS color transform to get hue and saturation of image pixels.
- Thresholding on the basis of hue and saturation to form another thresholded binary image.
- Combining the two thresholded binary images to form a final binary image.
- Applying a perspective transform to get the bird-eye view of the binary image.
- Applying a mask to remove unnecessary parts from the image.
- Detecting lane pixels and fit polynomial to find the lane boundary.
- Using the information from previous image to narrow down the lane search window.
- Determining the curvature of the lane and vehicle position with respect to center.
- Warping the detected lane region back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

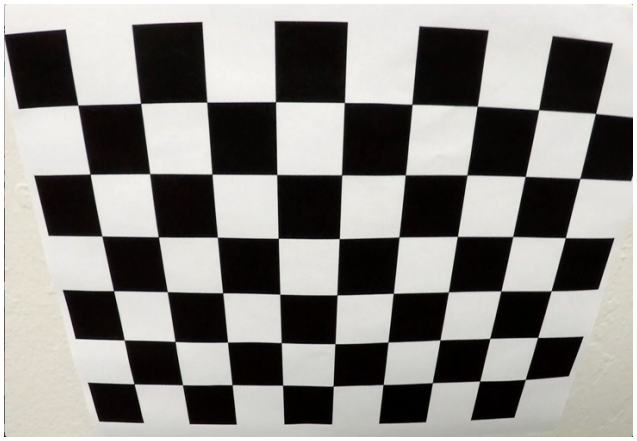
Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

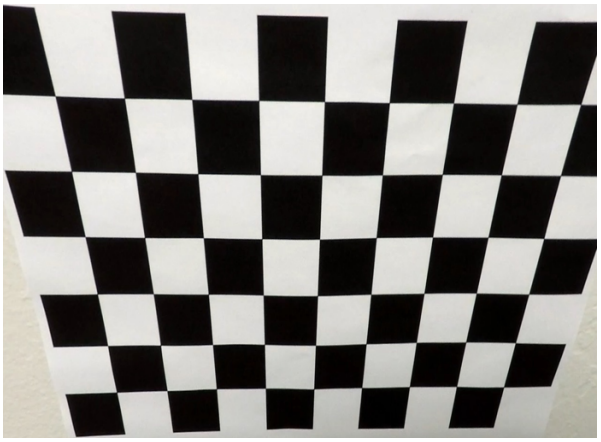
Camera Calibration:

1. **Computation of distortion coefficients:**

   First I created a matrix representing the ideal position of each corner of the chessboard in 3D space and stored it in variable ‘objp’. After that, I used ‘findChessboardCorners’ function of the cv2 library to find the actual location of the chessboard corners in the image and stored the location matric in variable ‘corners’. If the corners were successfully found in an image, I appended the ‘corners’ matrix to the ‘corner_points’ matrix and a copy of ‘objp’ to ‘object_points’ matrix. After going through all the images, I used ‘calibrateCamera’ function of cv2 library to compute the distortion matrix.



Chessboard Original Image



Undistorted Chessboard Image

Image Pipeline:

1. **Example of distortion-corrected image:**

   Below is the example of distortion-correction applied to an image

Original Image

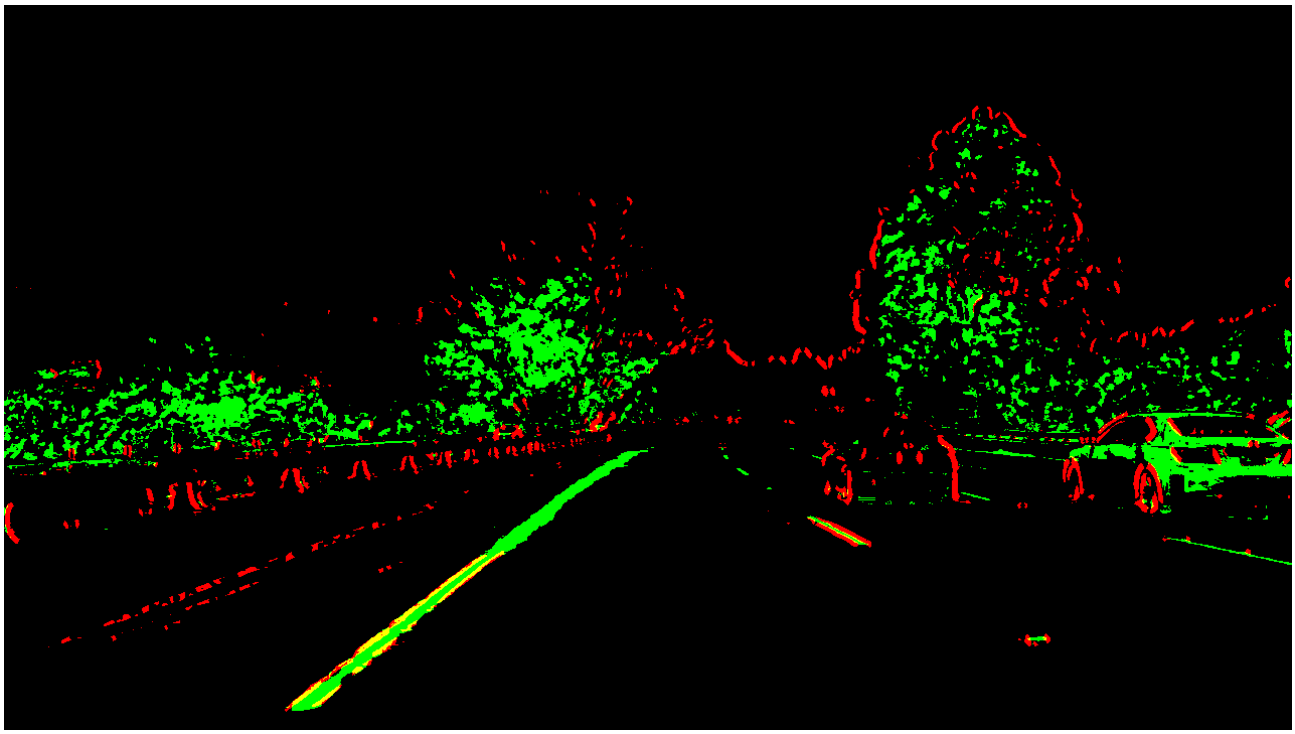
Distortion Corrected Image

2. **Methods used to create thresholded binary image:**

   I used various methods to create the final binary image, those methods are as follows:

   1. *sobel_mag_thresh()*: thresholding on the basis of gradient magnitude.
   2. *dir_threshold()*: thresholding on the basis of gradient direction.
   3. *sat_threshold()*: thresholding on the basis of saturation component in the HLS color space.
   4. *hue_threshold()*: thresholding on the basis of hue component in the HLS color space.

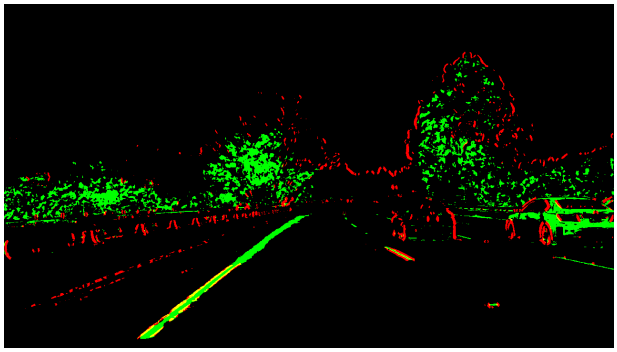
   After this I used 'logical AND' operation to get gradient thresholded image from the images obtained from functions (1) and (2) and similarly to get color-space thresholded image from the images obtained from functions (3) and (4). Finally I used 'logical OR' operation on the gradient thresholded image and color-space thresholded image to form the final thresholded image.



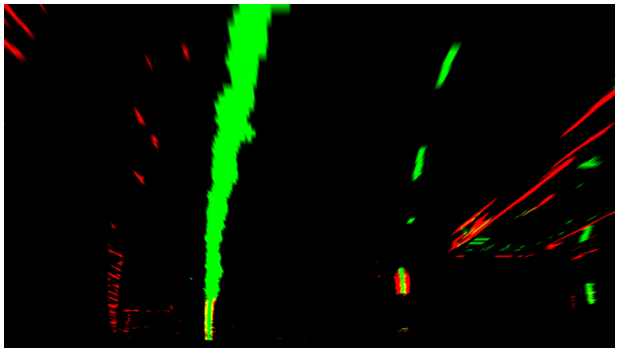
Green: component from color-space thresholding, Red: component from gradient thresholding

3. **Perspective Transform:**

For perspective transform I chose 4 points on the road, 2 at the beginning and 2 near horizon and then I chose the 4 points such that after the transform we'll get the bird-eye view of the complete road. I selected a straight road to compute the parameters for perspective transform, and then same parameter for all images.



Thresholded binary image



Bird-Eye view of thresholded binary image

4. **Identifying lane line pixels and curve fitting:**

I have used two functions to identify lane pixels-

compute_lane_prior(): it firstly guess the initial location of lane and then form squares one over other depending on the location of lane pixels, in the end all the pixels inside the boxes are considered lane pixels. This is more robust but computationally expensive method. Therefore, it is used in the beginning or if the main lane finding function struggles.

Find_lane(): it uses the lane information from the previous frame to find lane in the new image. It takes the computes curve from previous run and creates a region around it, all the pixels that fall in this region are considered lane pixels.

For fitting the curve to lane pixels, I found the x and y coordinates of the active lane pixels and used polyfit function to polynomial curve of degree 2.

5. **Calculation of radius of curvature and vehicle position:**

For finding the curvature I have created a function, *compute_real_curvature()*, which uses the formula given in lectures to compute the curvature of polynomial curve, near the car. To get the curvature in meters instead of pixels, I'm using pixel to meter conversion scale which I have defined in the class constructor.

For computing deviation of car position from the center, I'm calculating the center-point of the lane and finding its difference from expected center value. After this, I'm using x-axis pixel to meter scale to covert deviation to meter.

6. **Example image of the result:**

Final image after identifying lane and calculating lane curvature and deviation of vehicle from the center is as follows.

Pipeline (Video):

1. **Here is the link to final output video:**
   https://view5639f7e7.udacity-student-workspaces.com/view/CarND-Advanced-Lane-Lines/Advanced%20Lane%20Detection%20Output%20Video.mp4

Discussion:

1. **Features parallel to lane:**

   Currently if the features that are parallel to the lane will be close enough, they could be identified as lanes as well. This is because gradient filter will consider the involved pixels as lane pixels and final thresholded image consists of all the pixels present in gradient binary thresholded image. This could be solved by applying gradient thresholds on different components of color spaces rather than on gray scale image.

2. **Low contrast between lane markings and road:**

   Currently the pipeline heavily depends on saturation of HLS color space to obtain the lane markings, hue mainly acts as a rejecting factor. Therefore, pipeline is not robust enough if the contrast between road and marking color is low. To overcome this shortcoming, different components of different color spaces can be used to bolster the pipeline, for example, value component of HSV color space could be helpful in such cases.