**Advanced Lane Finding Project**

The goals and steps of this project are the following:

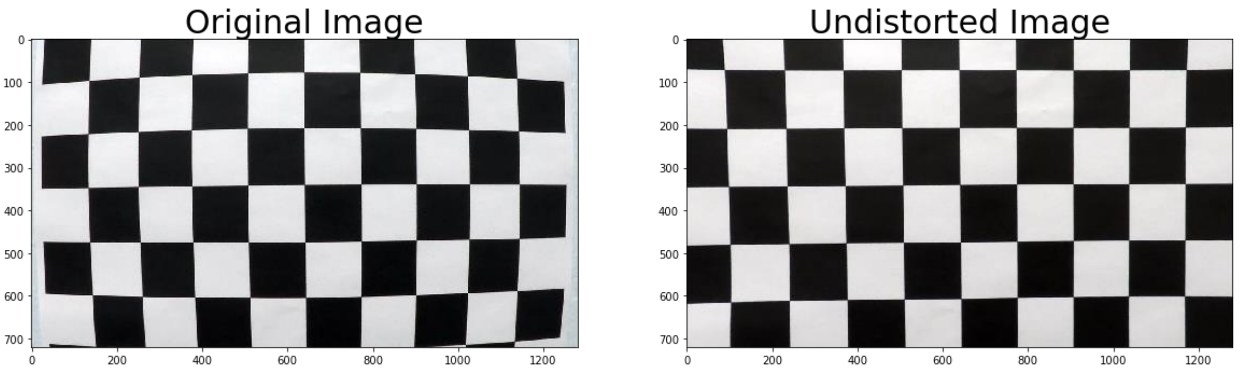
1. Compute the camera calibration matrix and distortion coefficient for the chessboard images
2. Undistort the input images
3. Apply color transforms and gradients to obtain thresholded binary images
4. Apply perspective transforms to obtain binary warped images
5. Detect lane pixels and fit to find land boundary using sliding window method
6. Compute the curvature of the lane and vehicle position with respect to center
7. Warp the detected lane boundaries back onto the original image
8. Generate a final output video with lane boundaries, vehicle positions and lane curvatures

Camera Calibration

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

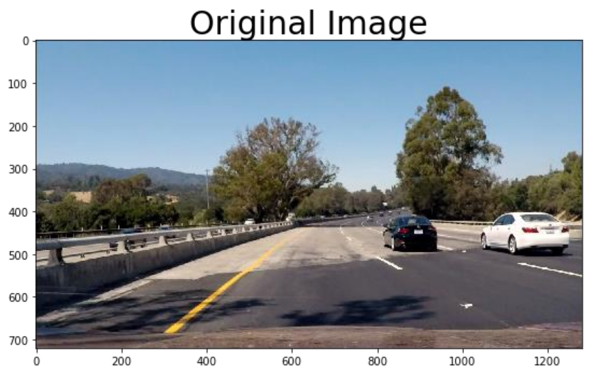
   I created two empty lists which are objpoints and imgpoints. Objpoints represents the 3D coordinates of the chessboard corners while imgpoints represents the 2D coordinates of the chessboard corners. First, I converted the image to grayscale. Then I used cv2.findChessboardCorners function to find chessboard corners and append object points and image points to objpoints list and imgpoints list accordingly. Then I used the object points and image points to compute camera matrix and distortion coefficients using cv2.calibrateCamera function.

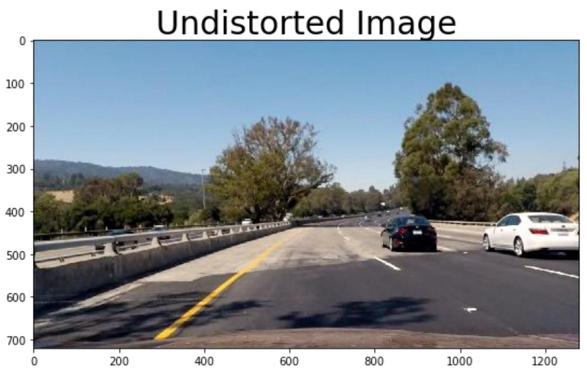
The original chessboard image and the undistorted chessboard image are shown below:



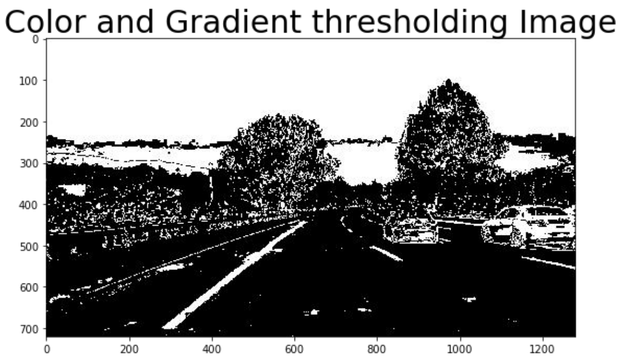
Pipeline

1. Provide an example of a distortion-corrected image.
   I saved the camera calibration result (camera matrix and distortion coefficient) from the previous part.  I applied the camera matrix and distortion coefficient to the undistortion function which I defined in 4th code cell to obtain a distortion-corrected road image.
   The original road image and the undistorted road image are shown below:

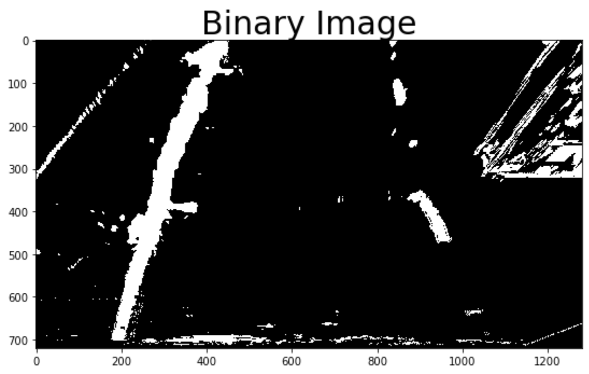


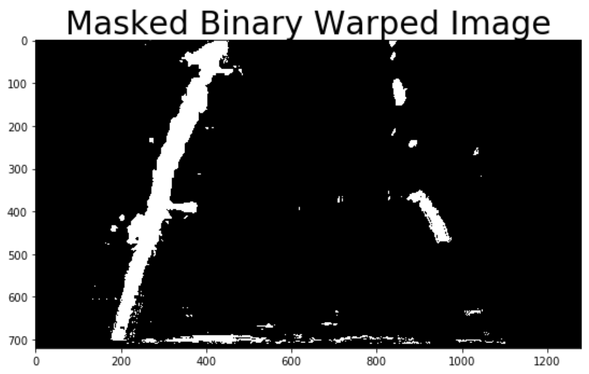


2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result**.**
   The code for this part is in 4th code cell.
   1. Gradient threshold:
      I first took the derivative of the image in the x direction. Then I calculated the magnitude of the gradient. I applied thresholds to both x direction gradient and gradient magnitude and combined them together,
   2. Color threshold:
      I converted the RGB color space to HLS color space and separated the s channel and l channel using color transform techniques. In addition, I converted the RGB color space to HSV color space to help identify white color and yellow color. I identified yellow color as the pixels whose intensities are between [10, 200, 0] and [255, 255, 255] and white color as the pixels whose intensities are between [15, 60, 130] and [150, 255, 255]. The I combined the white mask and yellow mask together. Finally, I combined the s channel and white and yellow color mask.
   3. Color and gradient threshold:
      Finally, I combined color threshold and gradient threshold together to obtain a final color and gradient thresholded binary image.
      The combined color and gradient thresholded binary image is shown below:



   4. Perspective transform and region of interest
      I applied perspective transform function to the binary image to get a binary warped image (see part 3). Then I applied the region of interest function (in 11th code cell) to the binary warped image to extract the lane lines.
      The masked binary warped image is shown below:





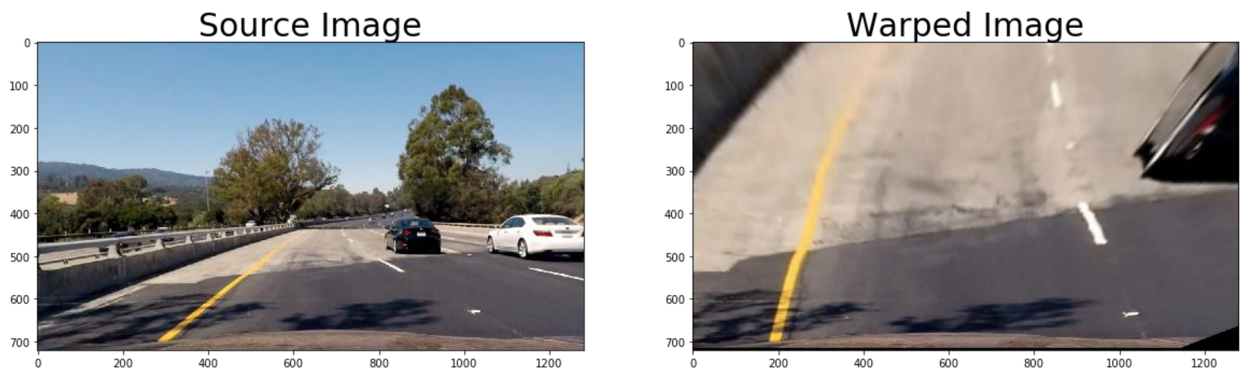
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

   The code for my perspective transform includes a function called warp () which locates in the 7th code cell. This function first defined four source coordinates and four destination coordinates and computed the perspective transform matrix M. Then the function warped the source image using the obtained perspective transform matrix, source coordinates and destination coordinates.

   The source and destination points are listed below:

| Source | Destination |
|---|---|
| [580, 450] | [300,700] |
| [760, 450] | [1150, 700] |
| [300, 0] | [200, 700] |
| [1000,0] | [1050, 700] |

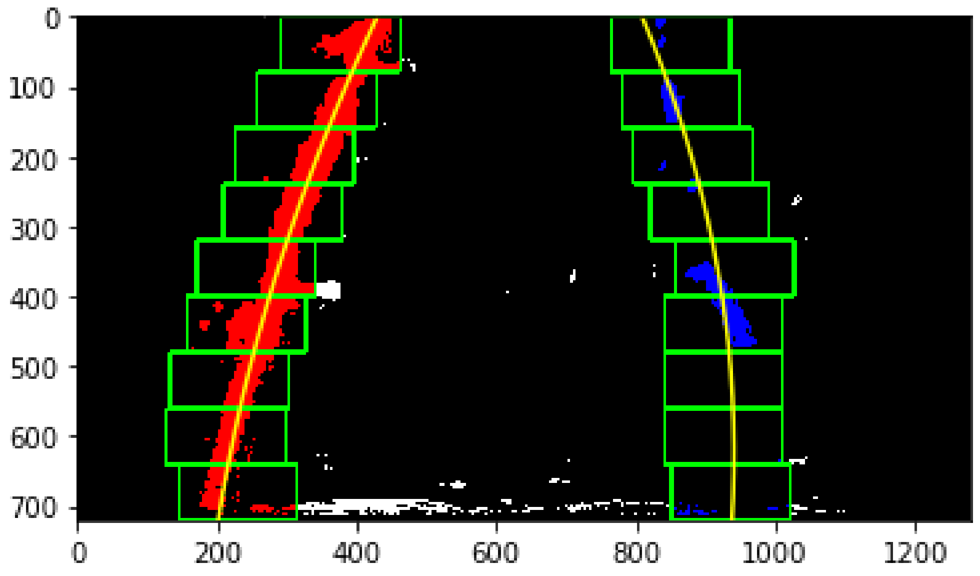
   The source image and warped image is shown below:



From the warped image shown above, it can be concluded that the warp function worked as expected.

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

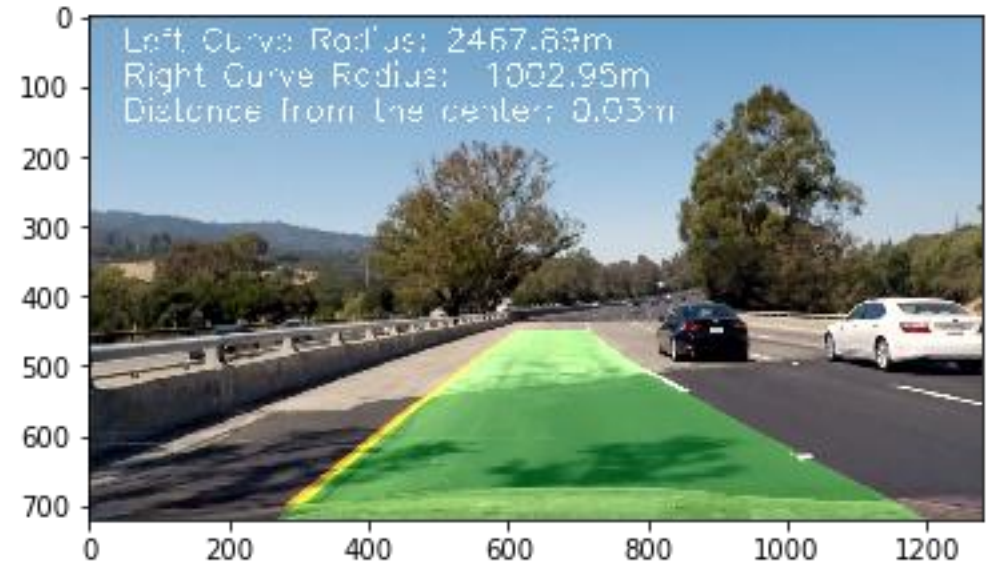
   The code for lane-line pixels identification is in 12th code cell. I took a histogram in the lower half of the image and identified peaks in the histogram. I implemented the sliding window search method and searched for lane lines. Finally, I identified the left and right line pixels and fitted with second order polynomials. After the left and right lane lines were detected, next time I can skip the blind search. I implemented a function ( in the 14th code cell) to find the lane lines in the next video frame based on the previous found lane lines.
   The result of sliding window search method is shown below:



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.
   The function for calculating the radius of curvature and the position of vehicle is in 12th and 14th code cell. I implemented this function within the find_lane_lines() function and the

find_new_lines_positions() function.   First, I defined conversions in x and y from pixels space to meters. Then I calculated the radius of curvature was calculated using the equation $R = \frac{(1+(2Ay+B)^2)^{\frac{3}{2}}}{|2A|}$ where A and B are coefficients of the second order polynomial. Finally, I calculated the position of car from the center by substracting the center of the image from the lane center.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.
   I implemented the function draw_lane_lines() which is in 14th code cell. This function combines the result with the original image.
   Here is an example of my result on a test image:


Left Curve Radius: 2467.89m
Right Curve Radius: 1002.95m
Distance from the center: 0.03m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Pipeline (video)

1. The final video output is included in the zip file

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?

1. Color thresholding

I first applied S channel and L channel color thresholding to the image, but it did not work well. Although S channel color thresholding seemed to work better than L channel color channel, using S channel color thresholding alone was not sufficient for lane detection. So, I decided to define a white and yellow color mask to extract the lane lines. The result was pretty good since the lane lines were identified and other irrelevant information was filtered out.

2. Region of interest function

After I obtained the binary warped image, I found there was still some irrelevant information such as the shadow of the car which was not helpful for detecting lane lines. I decided to mask the binary warped image to obtain the lane lines. As a result, this helped to increase the accuracy of the sliding window search method since it filtered out all irrelevant information.

3. Detecting outliers

At first, I did not perform a sanity check on the obtained lines. Therefore, my pipeline did not work well, and it failed to detect lane lines with shadows. Then I decided to implement the check_lines() function to detect and remove outliers. This function helped to check current lane lines against lane lines in the previous video frame. This function worked as it rejected lines with a significant change in line coefficients. Finally, I saw a great improvement in the output video.

4. Smoothing

At first, I kept only 10 lines in the line buffer and took average of them to obtain the line of best fit. After I increase the number of lines in the buffer to 20, I found output video became smoother and better.

5. Unexpected situations

In the project video, there is no sharp turns and the weather conditions are fine, so my algorithm performed well. If the weather is bad and there are some sharp turns on the road, my pipeline might not work. Therefore, I think I can improve my detecting outliers function which takes more factors into account such as bad weather conditions, lane curvature, detected lane pixels and the distance between left and right lane.