Advanced Lane Finding Project #2

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.

Rubric Points

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

You're reading it!

Camera Calibration

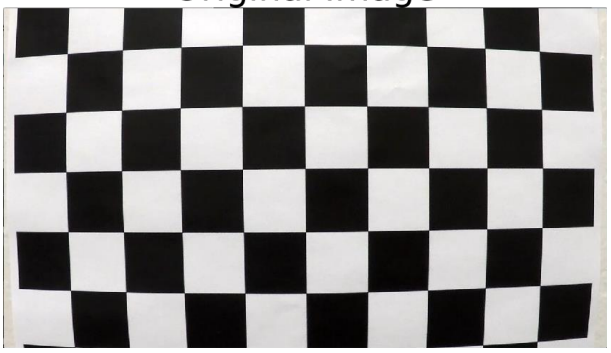
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the third code cell of the IPython notebook with output image located in "Output_images/Camera Calibration.ipynb".

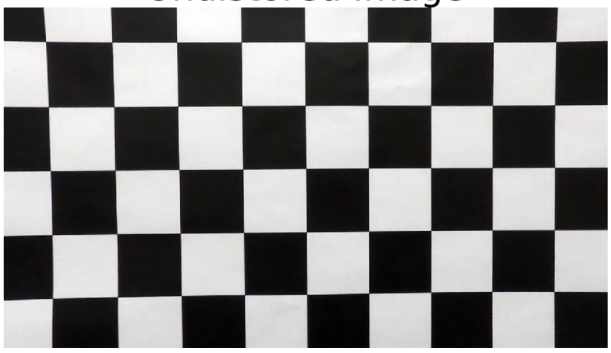
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:

Original Image



Undistored Image

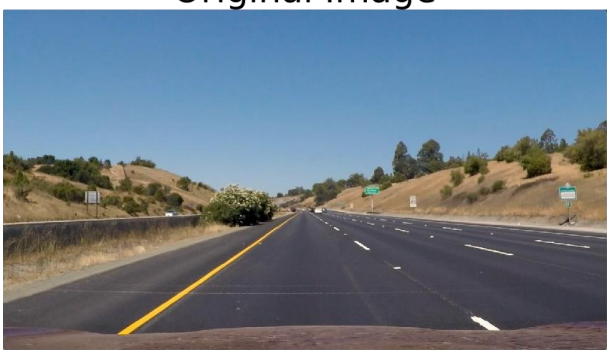


Pipeline (single images)

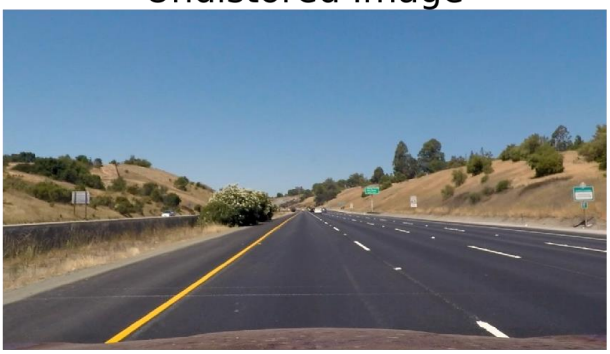
1. Provide an example of a distortion-corrected image.

I have used the distortion coefficients obtained from camera calibration images to prepare a undistorted images. The distortion correction to one of the test images like this one. Code for the output is present in the "***Output_images/ distortion-corrected.ipynb"***

Original Image



Undistored Image

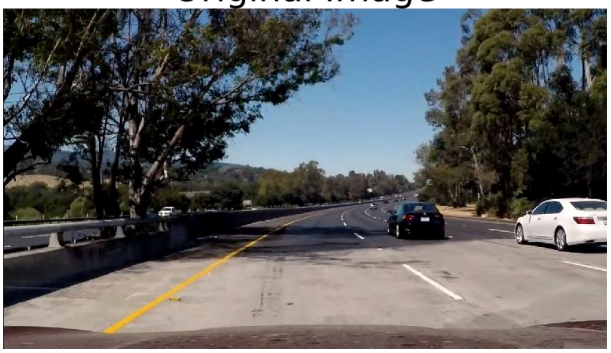


2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

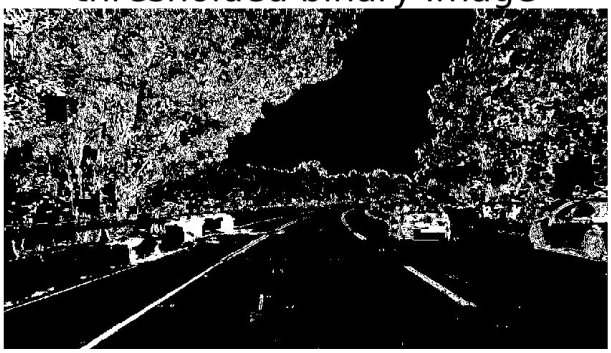
I used a combination of color and gradient thresholds to generate a binary image (thresholding steps are present at **cell1** function name **pipeline** at file **"*Output_images/***

binaryimage. Ipynb"). Here's an example of my output for this step.

Original Image

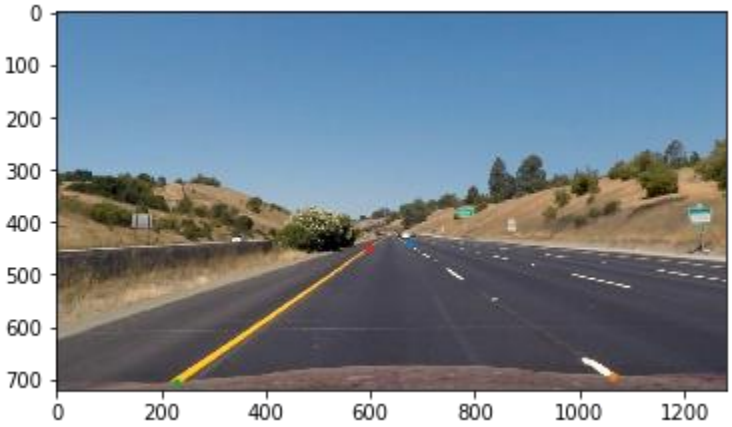


thresholded binary Image



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

First I found source points using the interactive window to get x and y coordinates. Code for this is present in the file "***Output_images/perspective transform.I*pynb")** at cell 28. Then I plotted the output In the undistorted image. (code is present in the cell 29 same file). The output is

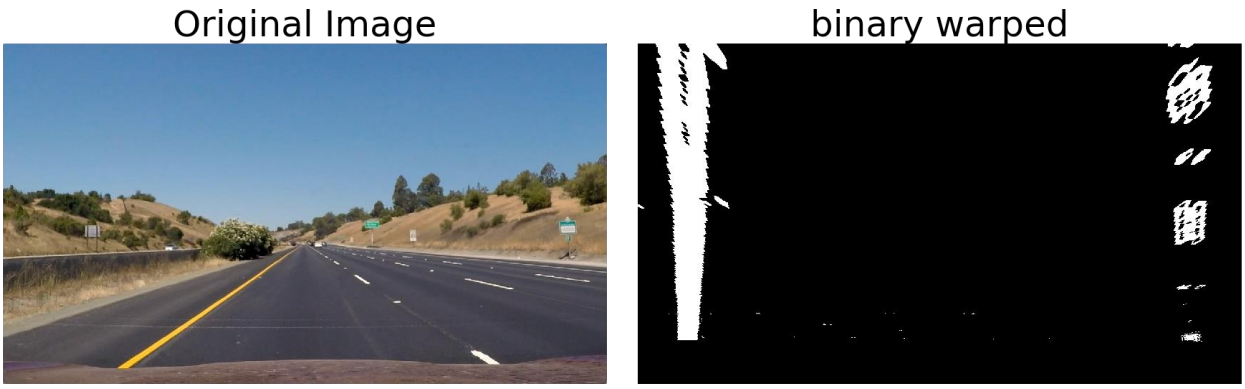


The `warper()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

```
img_size = (undistorted.shape[1], undistorted.shape[0])

 src = np.float32(

      [[707,463], #top right

       [1053,686], #bottom right

       [231,701], #bottom left
```

```
    [572,467]]) #top left
dst = np.float32(
    [[img_size[0]-offset, offset],
    [img_size[0]-offset, img_size[1]-offset],
    [offset, img_size[1]-offset],
    [offset, offset]])
```

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image. Output image is



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

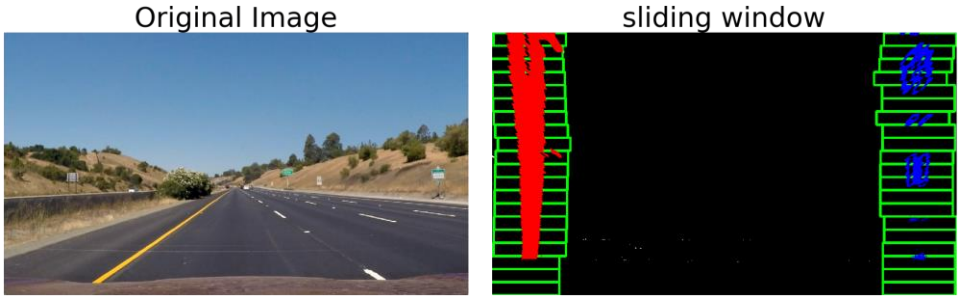
I used two methods to identify lane lines on the binary warped image. One is sliding window method and 2nd is search poly. Two explain in detail I have two methods code in the separate file.

"*Output_images/slidingpanel*.Ipynb" => output is present in the folder "example_slidingwindow_project_video.mp4"

Search poly method => output is present in the folder "example_slidingwindow_project_video"

I identified lane pixels by following below steps

1. **I took a nonzero pixels from binary warped image for x and y coordinates within the window.**
2. **I iterate through a prepared window by searing pixels near the window and recenter next window on their mean position**
3. **After iterating through out the window, extract the left and right lane pixels.**
4. **Fit a second order polynomial on the extracted left and right lane pixels**



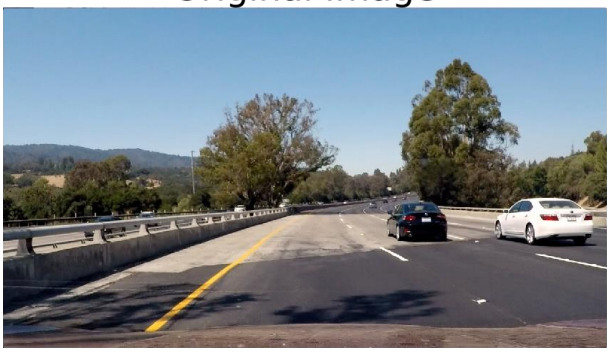
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

1. I first measured the width of the lane from the warped image
2. Then I converted pixel space to meters
3. I normally distributed lane line pixel values and calculated maximum y value corresponding to the bottom of the image
4. With the identified polynimial left and right fit values I applied the radius of curvature formula and found leftCurve and rightCurve values.
5. I stored the curvature values and display curvature value which is the mean of last 10 values. This I did to show smooth change in the value.
6. I defined the function **get_curvature** present in the file AdvanceLaneFind.ipynb(present in root directory) cell:1

6.Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I plotted the output I obtained like below

Original Image



draw lanes

Output video is present in the folder **"test_videos_output/ project_video_Final.mp4"**

Following are the output files I submit

1. Project video output => present in **"test_videos_output/ project_video_Final.mp4"**
2. Camera Calibration.ipynb
3. distortion-corrected.ipynb
4. perspective transform.ipynb
5. Searchpolymethod.ipynb
6. Slidingwindow.ipynb
7. AdvanceLaneFind.ipynb - main file with all functions combined present in the root directory