Advanced Lane Finding

Goals:

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 –

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

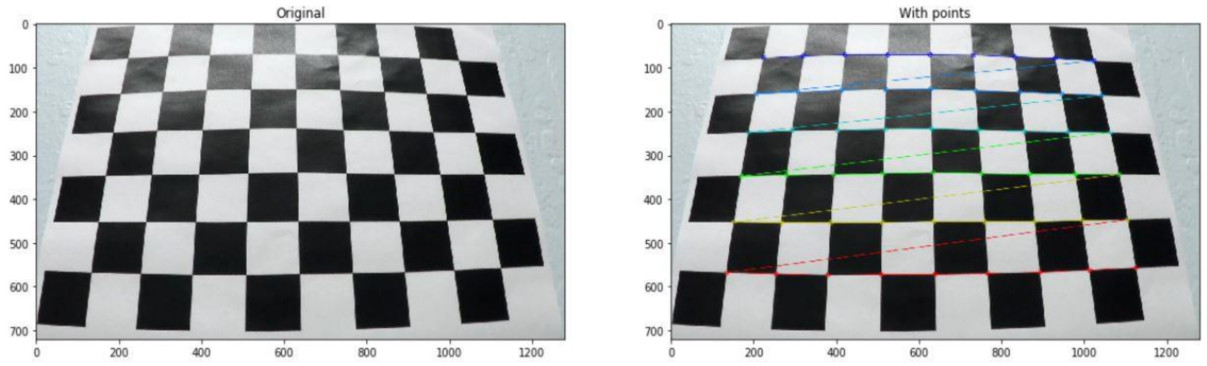
- You're reading it!

Camera Calibration

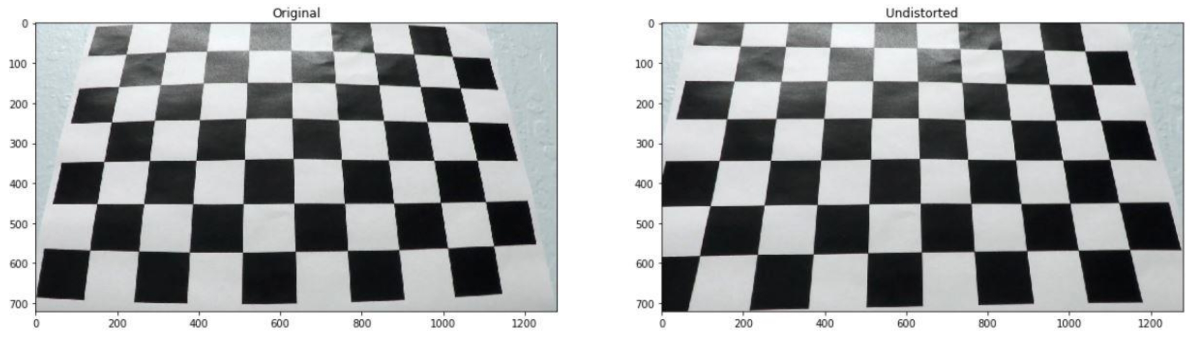
1. Briefly, state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

- Using `cv2.findChessboardCorners`, the corners points are stored in an array `imgpoints` for each calibration image where the chessboard could be found. The object points will always be the same as the known coordinates of the chessboard with zero as 'z' coordinate because the chessboard is flat. The object points are stored in an array called `objpoints`. Then the output objpoints and imgpoints are used to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera` function.

```
Out[61]: Text(0.5,1,'With points')
```

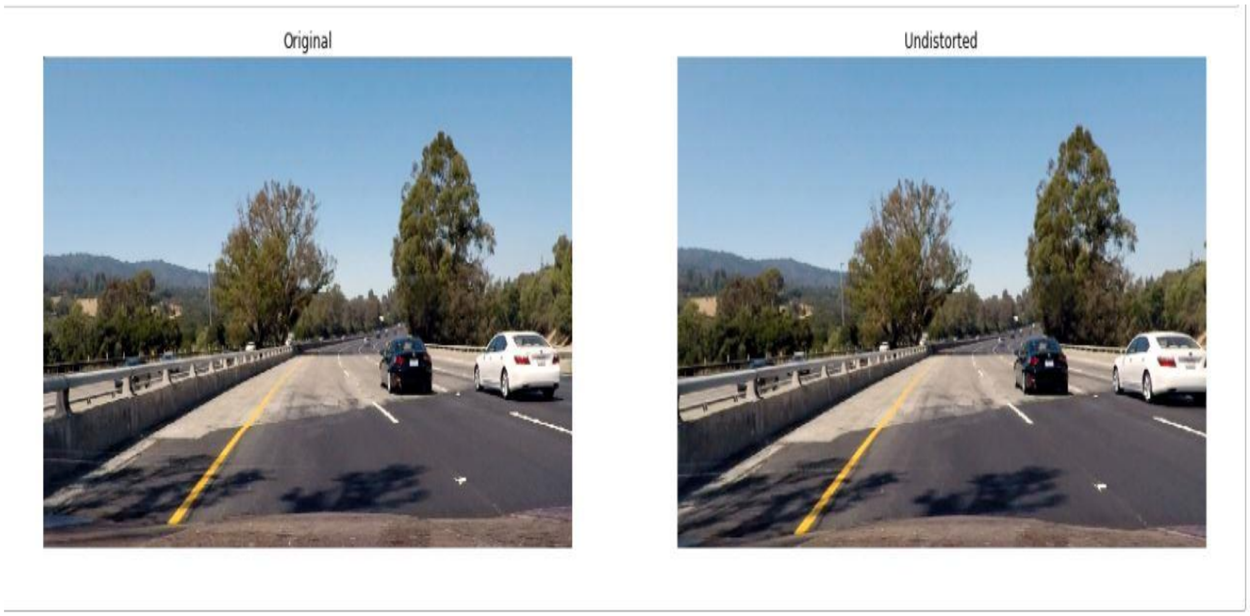


```
Out[64]: Text(0.5,1,'Undistorted')
```



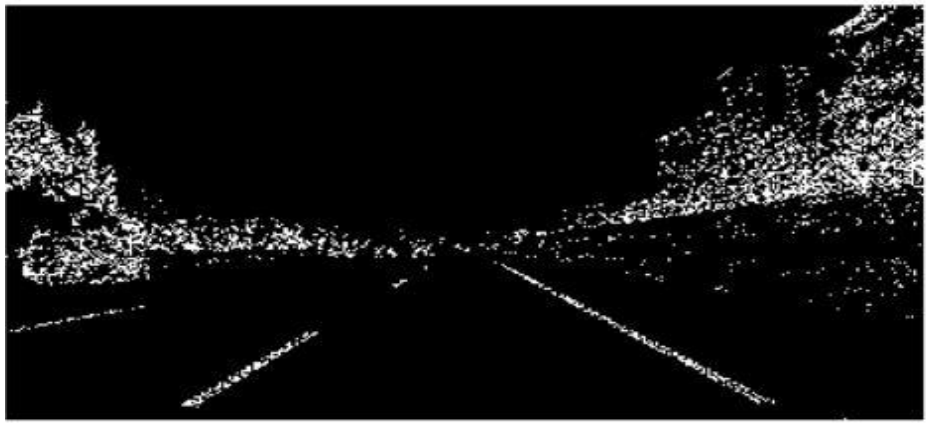
Pipeline

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



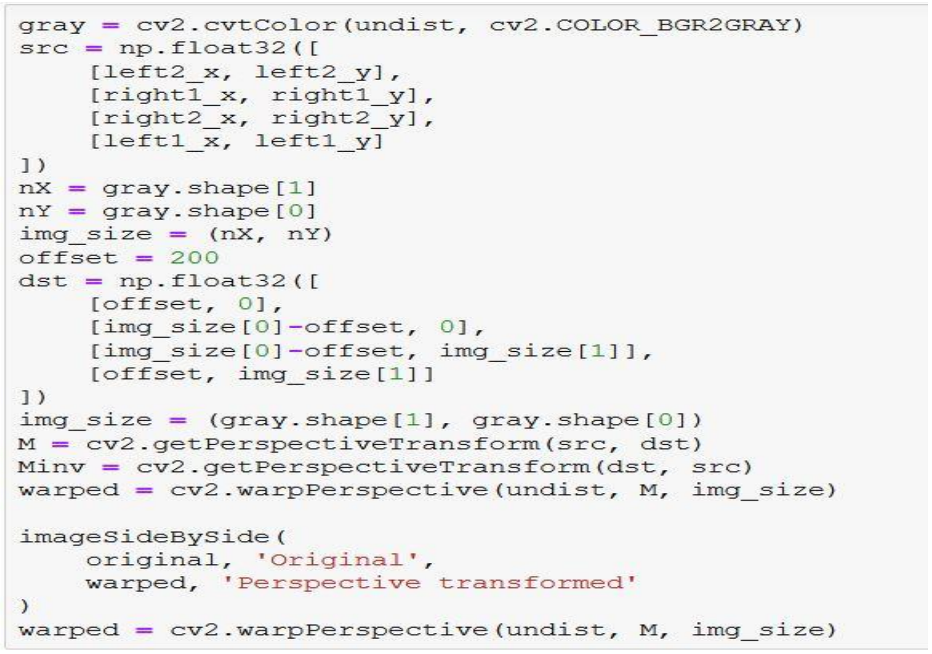
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

- A color transformation to HLS was done `In [15]`
- The gradients were calculated and all gradients were joined in `In [23]`



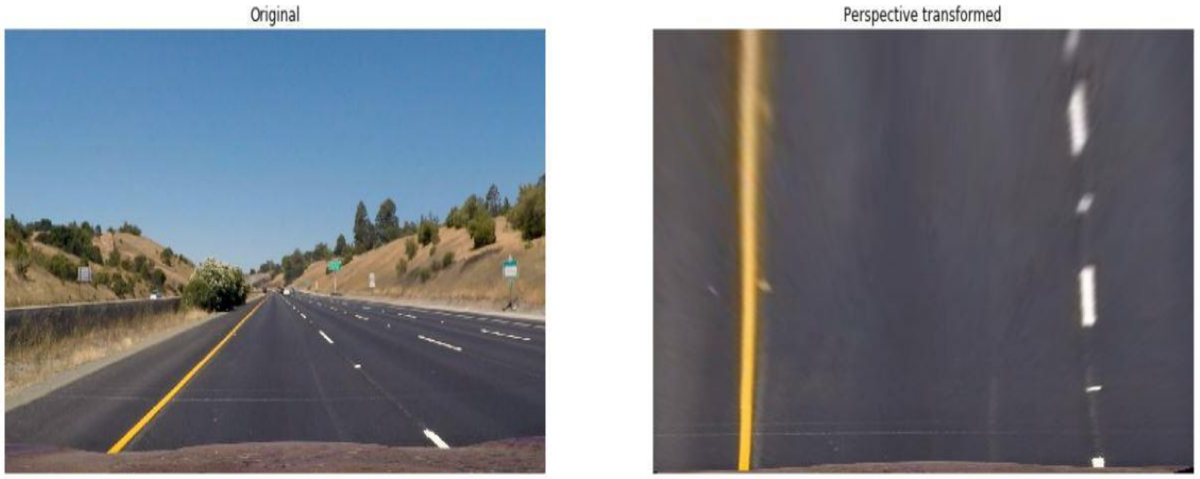
3. Describe how (and identify where in your code) you performed a perspective transform and provided an example of a transformed image.

- The code for perspective transform is given bellow and it can be found in `In [25]` –



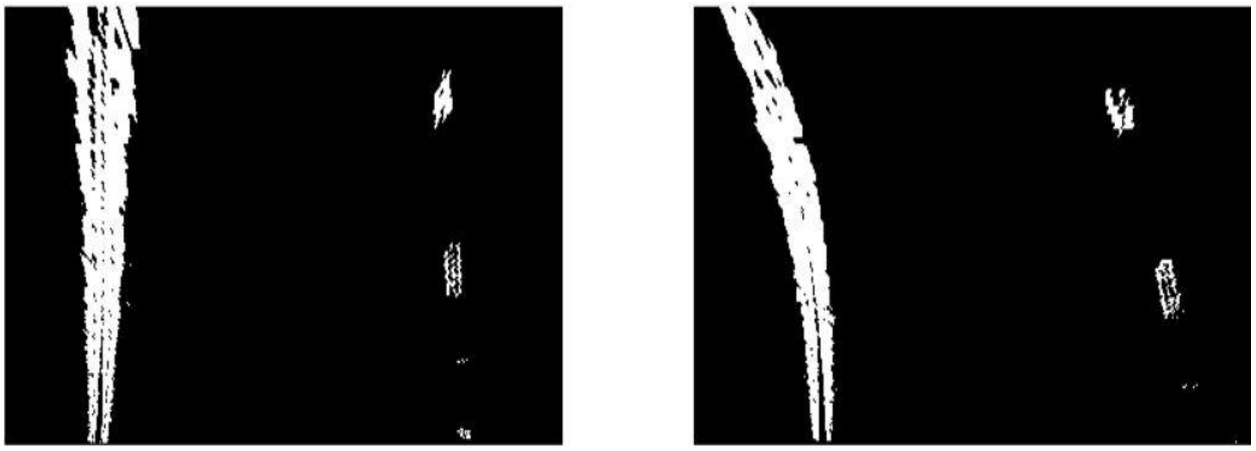
```
gray = cv2.cvtColor(undist, cv2.COLOR_BGR2GRAY)
src = np.float32([
    [left2_x, left2_y],
    [right1_x, right1_y],
    [right2_x, right2_y],
    [left1_x, left1_y]
])
nX = gray.shape[1]
nY = gray.shape[0]
img_size = (nX, nY)
offset = 200
dst = np.float32([
    [offset, 0],
    [img_size[0]-offset, 0],
    [img_size[0]-offset, img_size[1]],
    [offset, img_size[1]]
])
img_size = (gray.shape[1], gray.shape[0])
M = cv2.getPerspectiveTransform(src, dst)
Minv = cv2.getPerspectiveTransform(dst, src)
warped = cv2.warpPerspective(undist, M, img_size)

imageSideBySide(
    original, 'Original',
    warped, 'Perspective transformed'
)
warped = cv2.warpPerspective(undist, M, img_size)
```

And the output of this code is



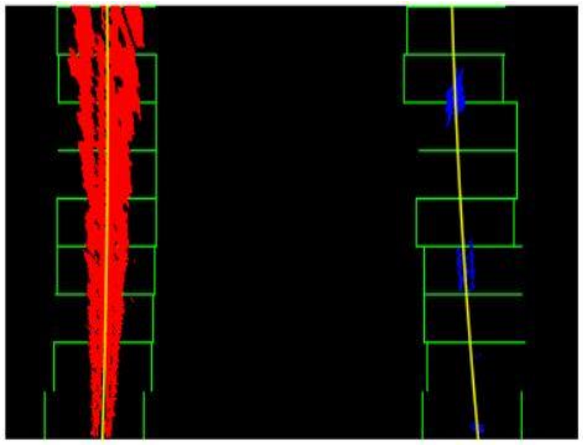
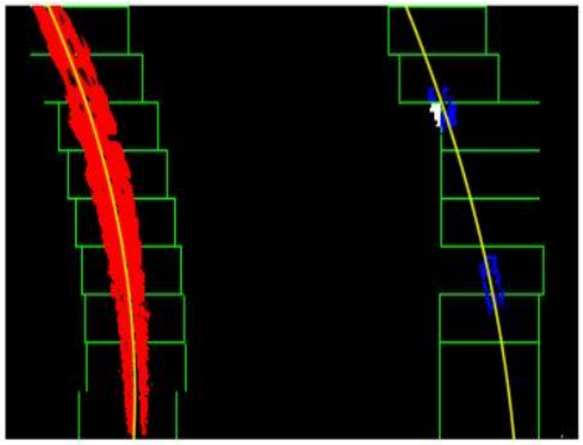
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial.

- The line detection code could be found at `In [27]`
- The algorithm calculates the histogram on the X axis. Finds the picks on the right and left side of the image, and collect the non-zero points contained on those windows. When all the points are collected, a polynomial fit is used (using `np.polyfit`) to find the line model. On the same code, another polynomial fit is done on the same points transforming pixels to meters to be used later on the curvature calculation.

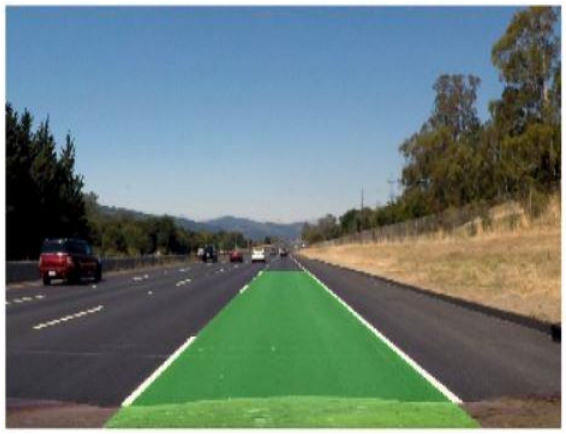
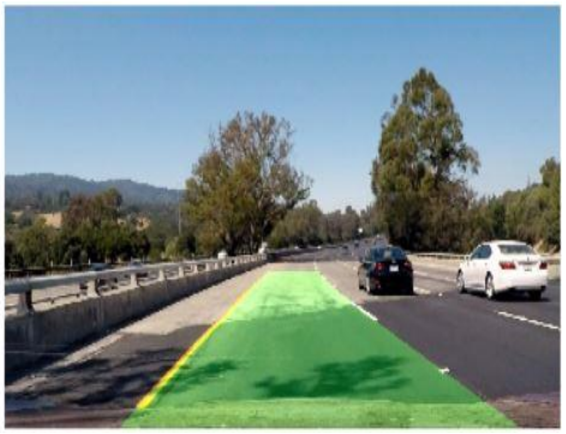


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle on the center.

- The radius of curvature of the lane and the position of the vehicle on the center was calculated on the `In [32]`

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

Pipeline (video)

1. 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 file is uploaded with this file. Please check output_video.mp4

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?

- The pipeline might fall if a white car is too close to the left or right lanes, and also when a white car is in front of our car and quite close to us.
- There are a few improvements that could be done on the performance of the process due to repetitive calculations.
    a. Keep track of the last several detections of the lane lines were and what the curvature was, so it can properly be treated new detections.
    b. Perform some sanity checks to confirm that the detected lane lines are real.
    c. Other gradients could be used to improve the line detection.
    d. More information could be use from frame to frame to improve the robustness of the process.