Advanced Lane Finding Project

The goals / steps of this project are the following:

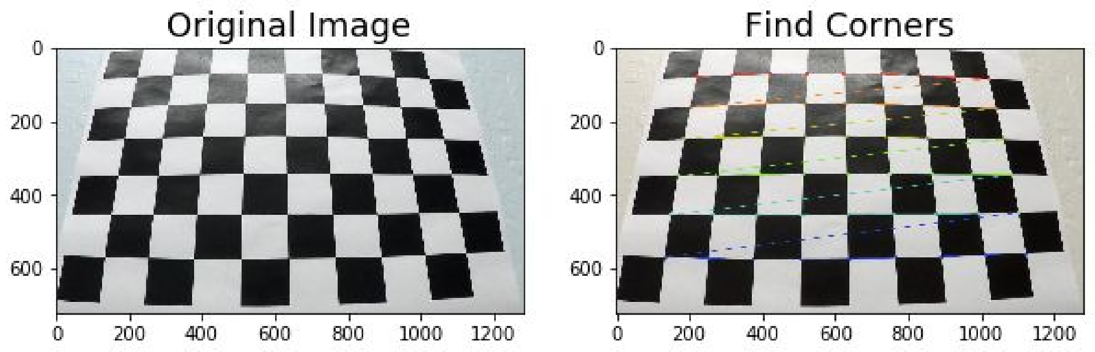
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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.

Camera Calibration

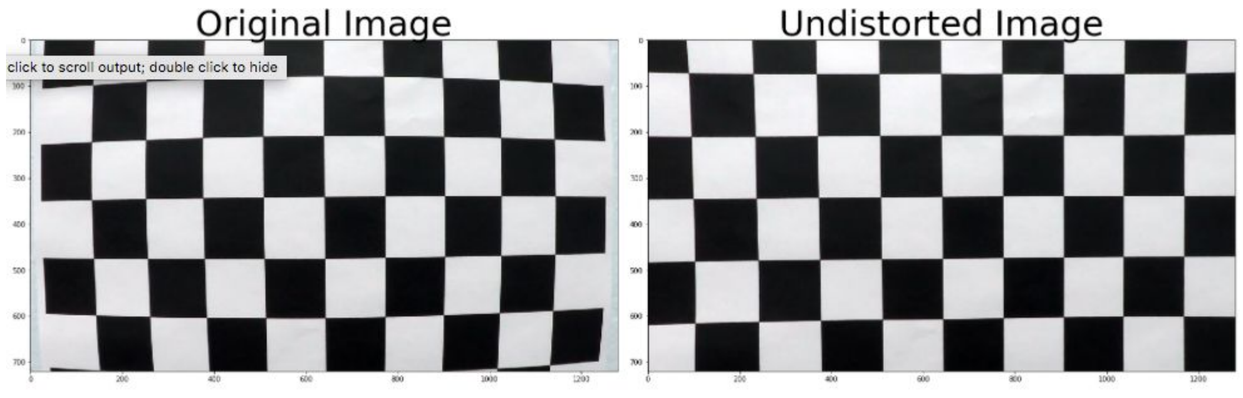
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

(1) 20 images of one chessboard from different angles and distances have been taken into consideration about camera distortion coefficients calculation.

(2) OpenCV function **findChessBoardCorners()** able to find the chessboard corners in each image and **drawChessBoardCorners()** mark them up.

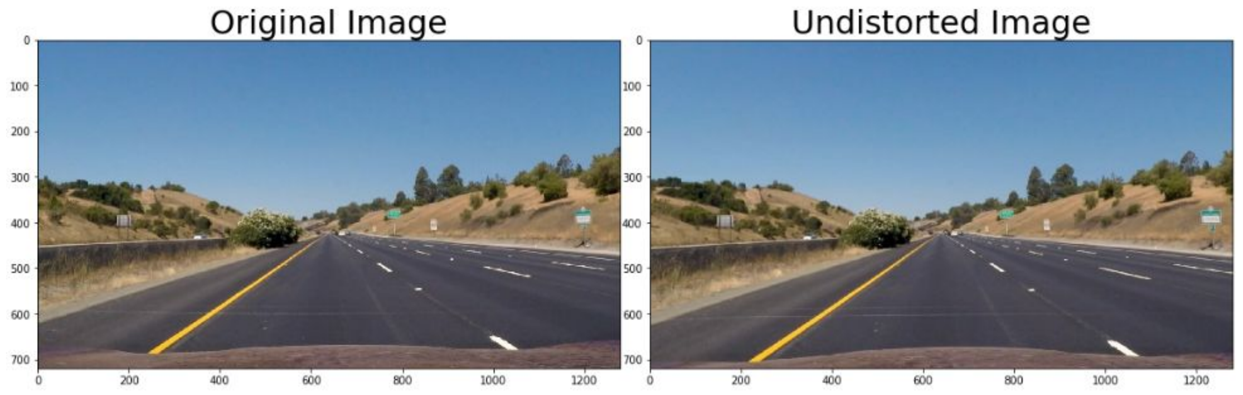


(3) OpenCV function **calibrateCamera()** able to calculate camera calibration matrix and distortion coefficient based on chessboard corners location.
(4) OpenCV function **undistort()** able to remove distortion from image.



Pipeline(test images)

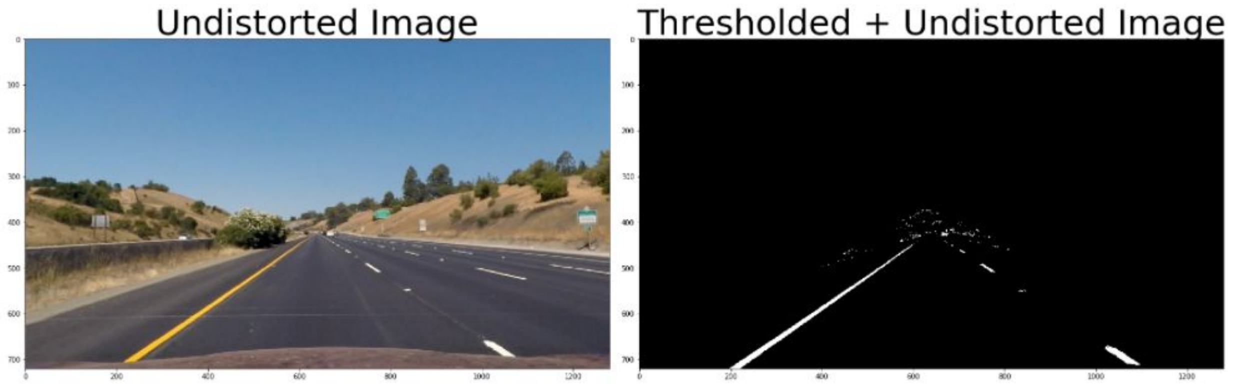
2.Provide an example of a distortion-corrected image.



3.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.

Gradient: Along the X axis and Directional Threshold of pi/6 to pi/2. Function **abs_sobel()** and function **dir_threshold()**.

Color transform using R&G in RGB, L and S in HLS. Function **thresholded_image().** R & G channel thresholds so that yellow lanes are detected well. L channel threshold so that we don't take into account edges generated due to shadows. S channel threshold since it does a good job of separating out white & yellow lanes.

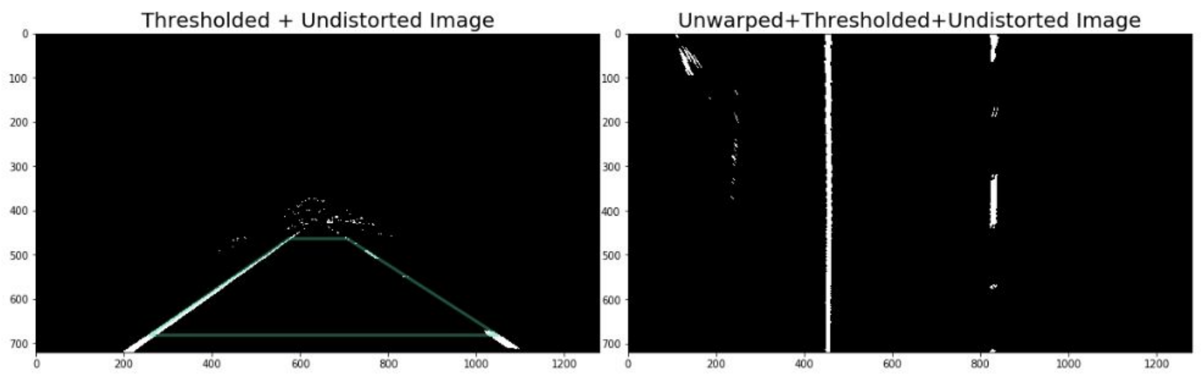


4.Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

Find four points as start points (src)in the previous image and four positions(dst) as destination for the same four points.

```
src = np.float32([(575,464),
                (707,464),
                (258,682),
                (1049,682)])
dst = np.float32([(450,0),
                (w-450,0),
                (450,h),
                (w-450,h)])   w as width and h as height of image.
```

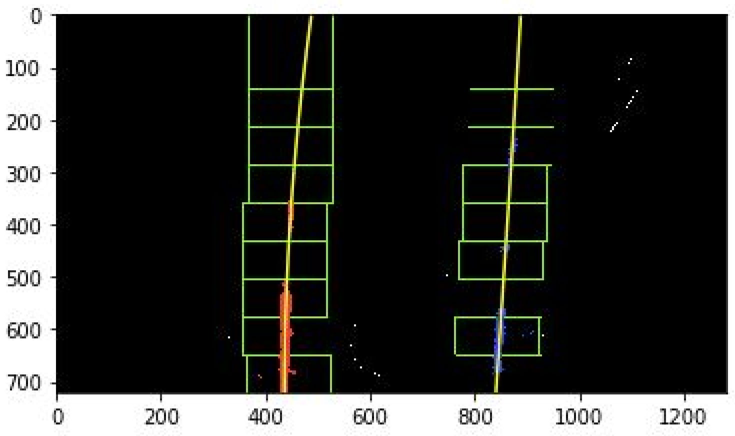
OpenCV function **getPerspectiveTransform()** able to finish the unwarp perspective transform in **unwarp()** function.



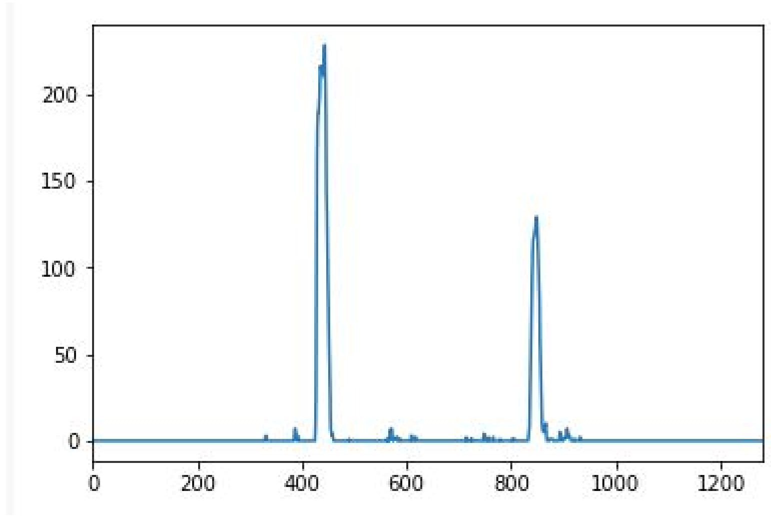
5.Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

A sliding window have been applied in function **slinding_window_polyfit()**. Calculate a histogram of the bottom half of the image and finds the x position of the two biggest part as left and right lane lines is the first step. A great idea online said only taking a quarter of the wide instead of half helps to reduce the effect of adjacent lines.
Ten windows, each one centered on the midpoint of the pixels help to identify the lane line and find the next position based on previous position. Pixels belonging to each lane line are identified and Numpy **polyfit()** method fits a second order polynomial to each set of pixels.

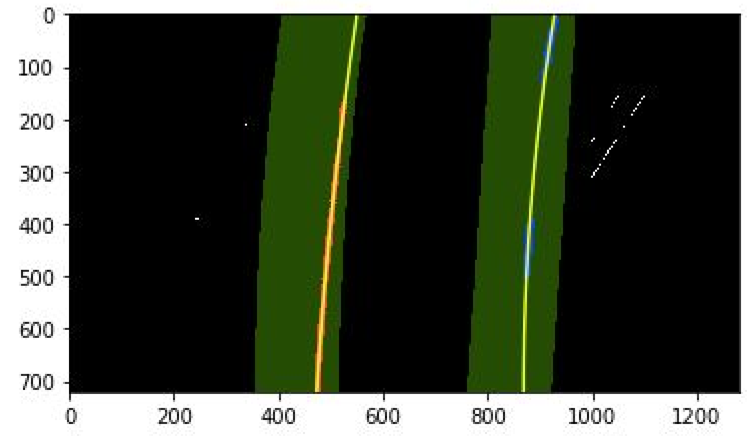
Ten windows



Histogram



Function **polyfit_using_prev_fit()** make it easier to calculate the next position based on previous position.



6.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.

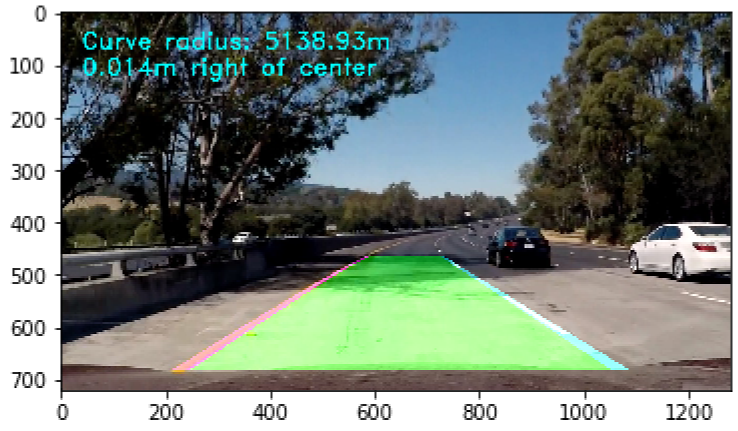
In function calc_curv_rad_and_center_dist().
With the polynomial fit (f(y) = fit[0]*y**2 + fit[1]*y + C) , radius of curvature could be computed, following:

```
curve_radius = ((1 + (2*fit[0]*y_0*y_meters_per_pixel + fit[1])**2)**1.5) /
np.absolute(2*fit[0])
```

The position of the vehicle with respect to center is :

```
lane_center_position = (r_fit_x_int + l_fit_x_int) /2
center_dist = (car_position - lane_center_position) * x_meters_per_pix
```

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



8.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!)

Please see attachment “project_video_output.mp4”

9.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 biggest problem in this project for me is to figure out how to make the polynomial fit. This video pipeline able to handle the “project_video.mp4” well. But I have saw clearly mislead in shadow area. The reason I believe is there are too many noise in thresholded binary image.
So I believe a better way for thresholding processing is a good next step for me.