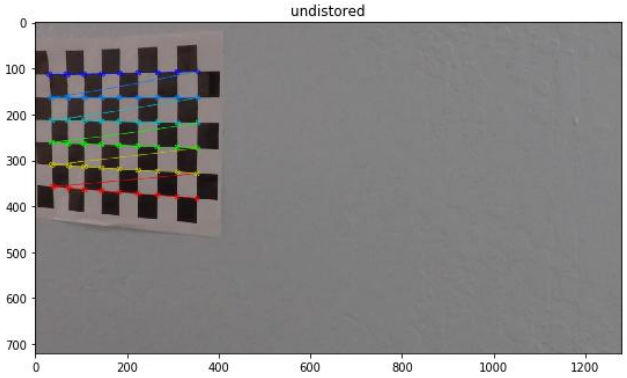
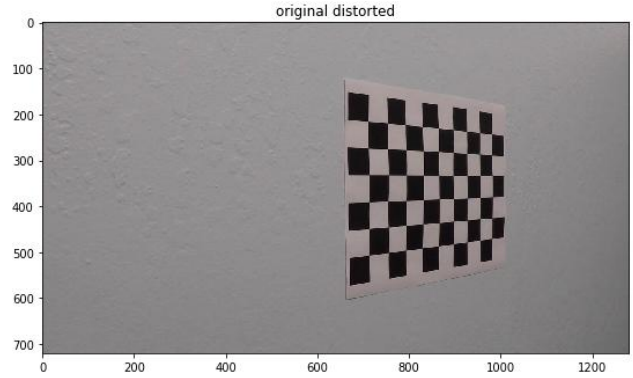
Rubric Points

Camera Calibration

Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.
The matrix and distortion coefficients are caluculated using OpenCV's calibrateCamera function. Specifically, I used cv2.findChessboardCorners() to get the chess board corners, and then manually created a list of points of where I think the chess board corners should be (i.e. equal distance apart from adjacent points). Then after getting camera matrix and distortion coefficients, I used cv2.undistort to undistort the image.

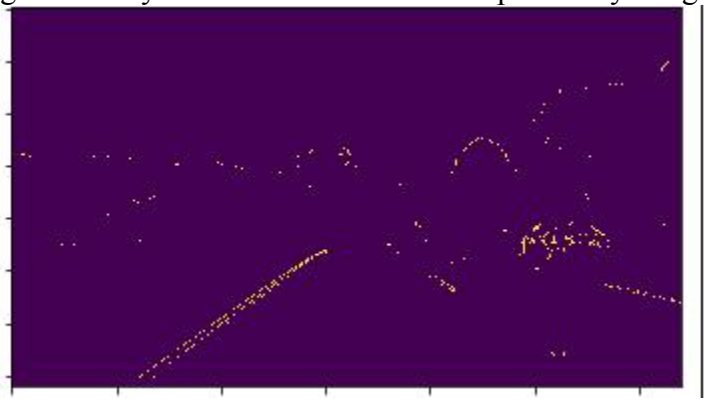




Pipeline (test images)

Provide an example of a distortion-corrected image.

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.
In the file find_lane_lines.py, from lines 28 to 54, I used multiple criteria such as Sobel operator and HLS color space masking to identify lane lines. Below is a sample binary image.

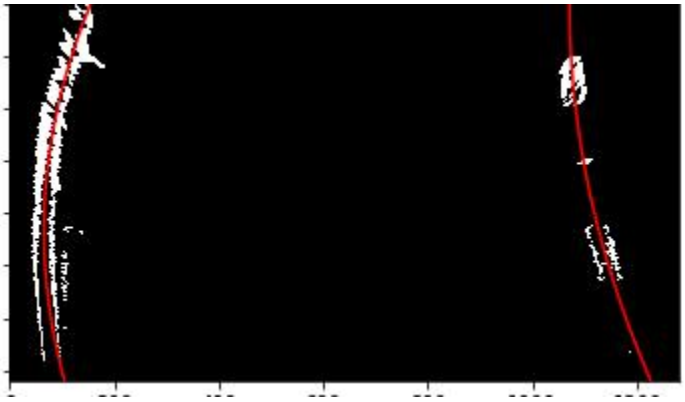


Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.
In find_lane_lines.py lines 60-61, I use cv2.warpPerspective() to transform to birds eye view. Specifically, I first calculated the perspective transform matrix and its inverse using cv2.getPerspectiveTransform(src, dst) and cv2.getPerspectiveTransform(dst, src).

Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?
In find_lane_lines.py lines 202 to 225, I fit a polynomial to the pixels on the line.

For the first search, I used a histogram to find the most likely location of the left and right lane lines, and slowly moved up the image in a "box" to capture other pixels that should be part of the lane line.

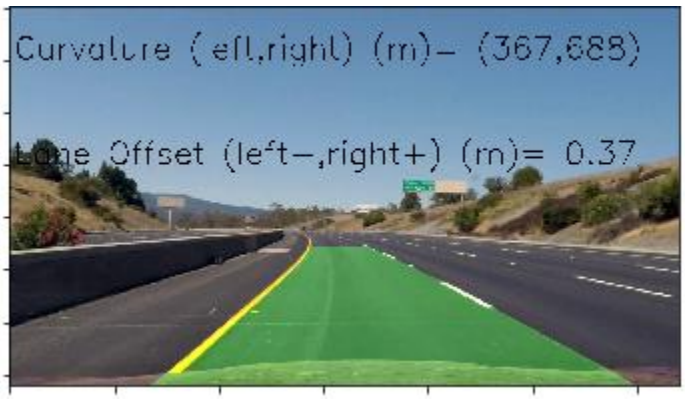
After the first search is done, subsequent lane line searchers only search in a horizontal margin around the previous fitted polynomial to save time.



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.

Lines 347 to 370 in find_lane_lines.py. Specifically how I did it was to convert the pixels into real distances, and then use a calculus formula to find the lane curvature.

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



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!)
See github repo "project_video_out.mp4"
https://github.com/brucecui97/CarND-Advanced-Lane-Lines/blob/master/project_video_out.mp4

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 hardest part was isolating the lane lines from the environment. Most likely my pipeline will fail in areas with shadows, or when the radius the car has to turn is large. Also, my pipeline does not differentiate lane lines with cracks on the ground very well. To make it more robust, I recommend adding more criteria to isolate the lane lines and/or use a deep learning approach to identify lane lines.