Advanced Lane Finding

Writeup / README

CRITERIA	MEETS SPECIFICATIONS
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.	The project submission includes all the required files • The jupyter notebook with code (CarND-Advanced-Lane-Lines.ipynb) • A writeup report (Writeup_CarND_Advanced_Lane_Lines.pdf) • In addition an html (CarND-Advanced-Lane-Lines.html) and pdf (CarND-Advanced-Lane-Lines.pdf) copies of the notebook are attached. • In the html file, the processed video can be played. A video link will also be provided separately.

Camera Calibration

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.	OpenCV functions or other methods were used to calculate the correct camera matrix and distortion coefficients using the calibration chessboard images provided in the repository. **(note these are 9x6 chessboard images, unlike the 8x6 images used in the lesson)**. The distortion matrix should be used to un-distort one of the calibration images provided as a demonstration that the calibration is correct. Example of undistorted calibration image is Included in the writeup (or saved to a folder).	First the object points and image points are calculated using the function getImageAndObjectPoints() and in turn used the object and images points are used to calculate the matrix and distortion coefficients. These can be found in cell, 5-9 in the notebook/html/pdf provided. The undistorted cells are present in cells 11, 17.

Pipeline (test images)

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Provide an example of a distortion-corrected image.	Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.	It is present in cells 11 and 17 in the notebook/html/pdf
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 method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.	I have tested with multiple color, gradient and used thresholds that better work on the test images. The sample tests done are present in cells from 22-45. Again please refer notebook / pdf / html. The binaryies are saved with combined* prefix in the test_images folder. Also displayed in cell 56.
Describe how (and identify where in your code) you performed a perspective transform	OpenCV function or other method has been used to correctly rectify each image to a "birds-eye view". Transformed images should be included in the writeup (or saved to a folder) and submitted	Perspective transform is coded in function `transform_and_warp()` in cell 18, an example is tested an displayed in cell 21.

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
and provide an example of a transformed image.	with the project.	
Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?	Methods have been used to identify lane line pixels in the rectified binary image. The left and right line have been identified and fit with a curved functional form (e.g., spine or polynomial). Example images with line pixels identified and a fit overplotted should be included in the writeup (or saved to a folder) and submitted with the project.	This is coded in function findLanes() in cell 60. The lane pixels and their fit positions are done in two stages. If the lane is never discovered, the code will take the if condition and if the lane is already discovered it goes to the else condition. Test images is displayed in cell 62 and 63.
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.	Here the idea is to take the measurements of where the lane lines are and estimate how much the road is curving and where the vehicle is located with respect to the center of the lane. The radius of curvature may be given in meters assuming the curve of the road follows a circle. For the position of the vehicle, you may assume the camera is mounted at the center of the car and the deviation of the midpoint of the lane from the center of the image is the offset you're	This is coded in cell 65 and 67 in functions rocLane() and vehicle_position() functions. The radius of curvature is calculated both in pixels and meters.

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
	looking for. As with the polynomial fitting, convert from pixels to meters.	
Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.	The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and position from center should be included in the writeup (or saved to a folder) and submitted with the project.	This is coded up in cell 69 as drawLane() function. And image showing the result is in cell 71.

Pipeline (video)

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS

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 image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.	The whole pipeline is tested in cell 75, 76. A function called process_image is written using the pipeline. This function is in turn called to process the video.
		
		The results can be found in cells 75-83.
		
		The project videos are under
		
		`project_videos_output/project_video_after_process.mp4`
		
		`project_videos_output/harder_challenge_video_after_process.mp4`
		
		in the zip file.

Discussion

CRITERIA	MEETS SPECIFICATIONS
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 project is working well with the project video but fails in scenarios of the challenge video. Multiple things that are contributing the failure are: - shades - curves - brightness - closeness of lanes to the edges. We need to device a better color thresholding matrix to find the levels in all these scenarios and create a combined filter and then the project should work on challenge video too.

Suggestions to Make Your Project Stand Out!

For a standout submission, you should follow the suggestion in the lesson to not just search blindly for the lane lines in each frame of video, but rather, once you have a high-confidence detection, use that to inform the search for the position of the lines in subsequent frames of video. For example, if a polynomial fit was found to be robust in the previous frame, then rather than search the entire next frame for the lines, just a window around the previous detection could be searched. This will improve speed and provide a more robust method for rejecting outliers.

The above is addressed in the project to use the previously detected lanes. But the below two are not addressed

For an additional improvement you should implement outlier rejection and use a low-pass filter to smooth the lane detection over frames, meaning add each new detection to a weighted mean of the position of the lines to avoid jitter.

If you really want to go above and beyond, implement these methods on the challenge videos as well, or on your own videos you've recorded yourself.