Advanced Lane Detection

Done by: Wael Farag

1. Camera Calibration

The following steps are used to calibrate the Camera:

1. **Step 1 – finding the chessboard corners**: Using 20 chessboard images that have different sizes and orientations, I used the "findChessboardCorners()" function from the openCv3 library to locate the chessboard corners. The detected number of corners is 9x6 as shown in the 17 out of the 20 images that are depicted in Figure 1. The other 3 images only 9x5 have been detected. The corners are drawn using the "drawChessboardCorners()" function of openCv3. The code for this part can be found in "P3 AdvLane2.py" lines 65 => 116.

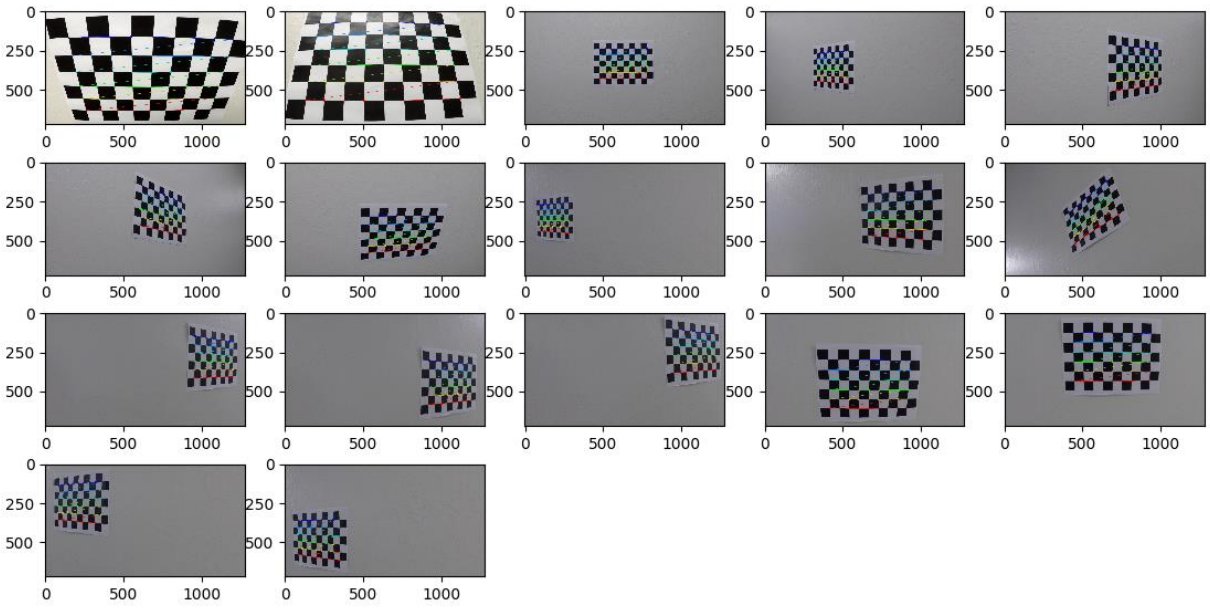

Figure 1. Chessboard images used for Calibration with corners drawn.

2. **Step 2 – get camera matrices**: a test chessboard image that has not been used in finding the corners is used; after being converted to gray scale; with the found corners in step one to find the camera matrices. I used "cv2.CalibrateCamera()" function to perform this step. To check the quality of the calibration, the gray test image together with the camera matrices to remove the distortion of this image as show in Figure 2. The code for this part can be found in "P3 AdvLane2.py" lines 118 => 137.

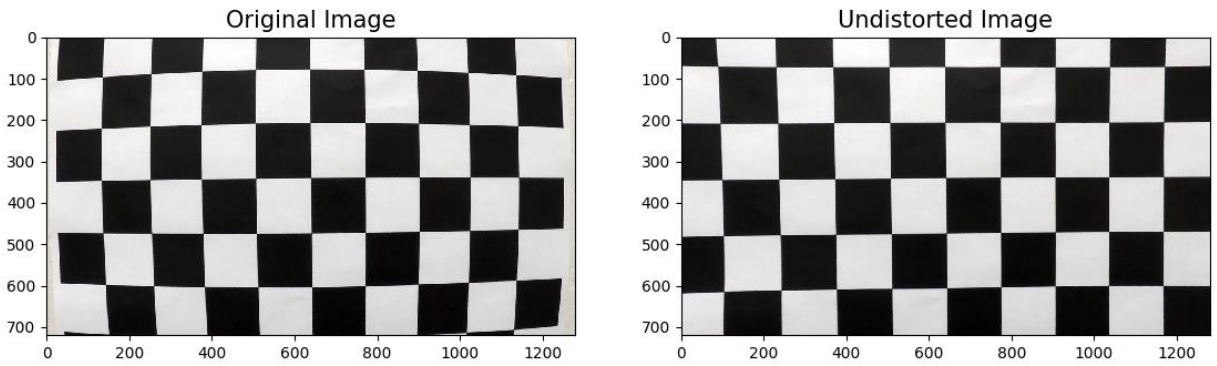

Figure 2. Test Chessboard image with distortion removal.

3. **Step 2 – saving camera matrices**: using Pickle library, the camera data: the camera matrix "mtx" as well as the distortion coefficients "dist" are saved in the pickle file "camera_calibration.p" for easy retrieval later. The code for this part can be found in "P3 AdvLane2.py" lines 139 => 145.

2. Image Processing Pipeline

The following steps describes the implemented image processing steps in order of execution:

1. **HLS conversion**: the images which are received in RGB color space are converted to HLS color space. Then the "S" (saturation) channel is then extracted using the function "HLS_select_S_Channel()" and stored in "hls_S_channel" image file. The threshold value used is '170' and the code for this function can be found in "P3 AdvLane2.py" lines 233 => 238.
2. **HSV conversion**: the images which are received in RGB color space are converted to HSV color space. Then the "V" (value) channel is then extracted using the function "HSV_select_V_Channel()". However, this function is being used during the trial and error while constructing the final pipeline. It finally skipped as it is found it is not adding much value to the final result. The code for this part can be found in "P3 AdvLane2.py" lines 240 => 245.
3. **Absolute Sobel Gradients**: the absolute Sobel gradient function "Abs_Sobel_Thresh()" is implemented in "P3 AdvLane2.py" lines 172 => 189. It is applied in the pipeline for both the x and y axis. The used minimum and maximum threshold for the $Sobel_x$ is '20' and '200' respectively. Moreover, the used minimum and maximum threshold for the $Sobel_y$ is '150' and '180' respectively.
4. **Magnitude Sobel Gradients**: the magnitude Sobel gradient function "Mag_Thresh()" in implemented in "P3 AdvLane2.py" lines 192 => 210. It is applied in the pipeline with kernel value of '9'. The used minimum and maximum threshold are '100' and '200' respectively.
5. **Direction Sobel Gradients**: the direction Sobel gradient function "Dir_Threshold()" in implemented in "P3 AdvLane2.py" lines 214 => 228. It is applied in the pipeline with kernel value of '3'. The used minimum and maximum threshold are '0.7' and '1.3' respectively.

6. **Combining All**: the above algorithm have been combined and produced an output represented by the image file “combined_sobel_s_channel”. The pipeline is implemented in “P3 AdvLane2.py” lines 251 => 268.

3. Lane Finding Pipeline

The following steps have been used to identify and locate lane lines, and presented in order of execution. I am using the “straight_lines1.jpg” image from the supplied testing images as a working example. The function “process_image()”; implemented in “P3 AdvLane2.py” lines 621 => 754; is the main procedure that executes all the necessary routines to successfully find the lane lines:

1. **Undistorting the image**: removing the distortion of the image using “cv2.undistort()” in “P3 AdvLane2.py” lines 627 => 632. Figure 3 on “straight_lines1.jpg” image shows the effect before and after the execution.

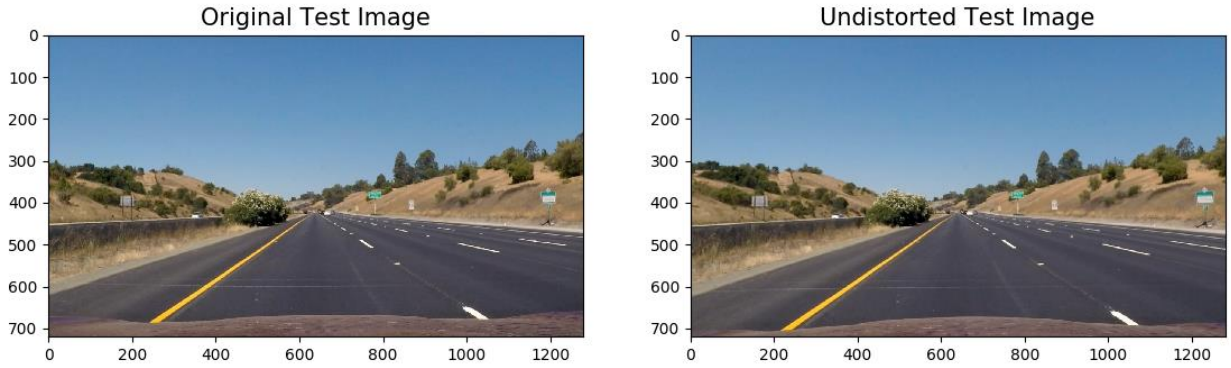

Figure 3. “straight_lines1.jpg” image with distortion removal.

2. **Applying the image processing pipeline:** the image processing pipeline function “” is being called, which results in the extraction of the S channel in the HLS color space and combination of the Sobel gradients algorithms (Absolute (x and y), Magnitude & Direction) with the S channel as shown in Figure 4. The call is implemented in “P3 AdvLane2.py” lines 633 => 638.

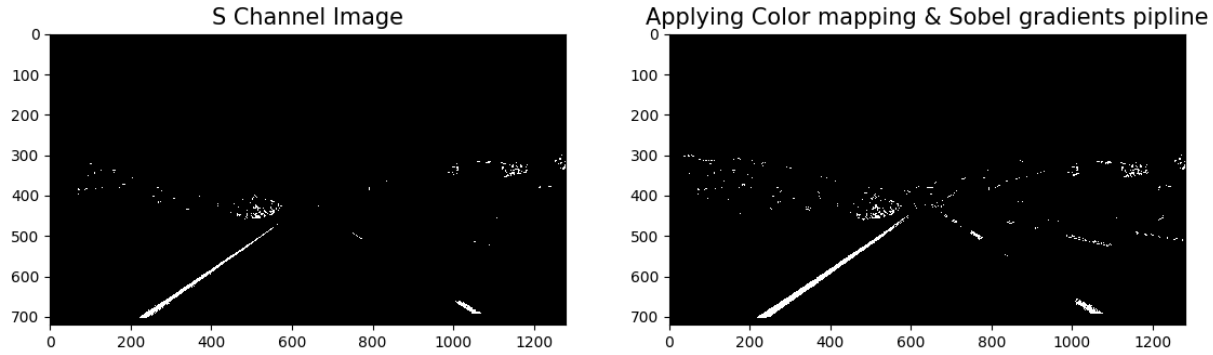

Figure 4. S Channel and Sobel gradients combined.

3. **Identifying the region of Interest (ROI)**: after exhaustive trials and errors the region of interest has been identified and the applied on the resulting image of step 2 as shown in Figure 5. The identification is implemented in “P3 AdvLane2.py” lines 639 => 661.

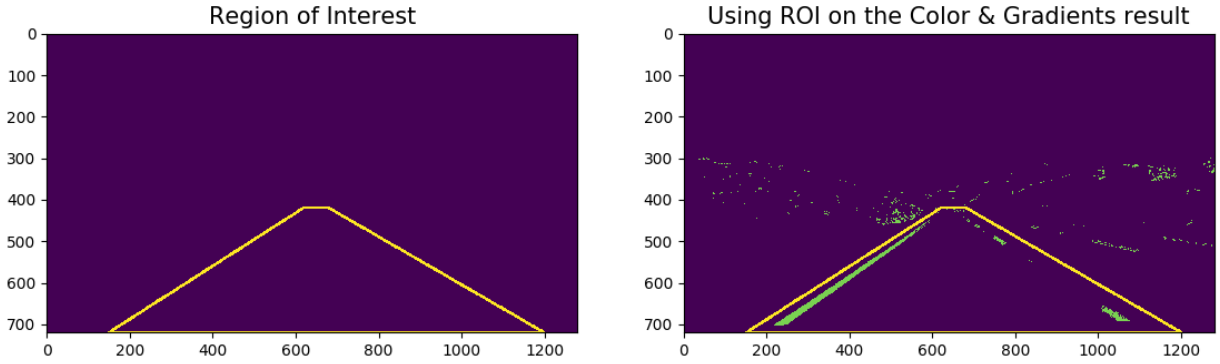

Figure 5. Identification and application of the Region of Interest.

4. **Mask the undesired image details:** The regions other than the region of interest are then masked (as shown in Figure 6) to give the algorithm more focus. The masking is implemented in “P3 AdvLane2.py” lines 663 => 672.

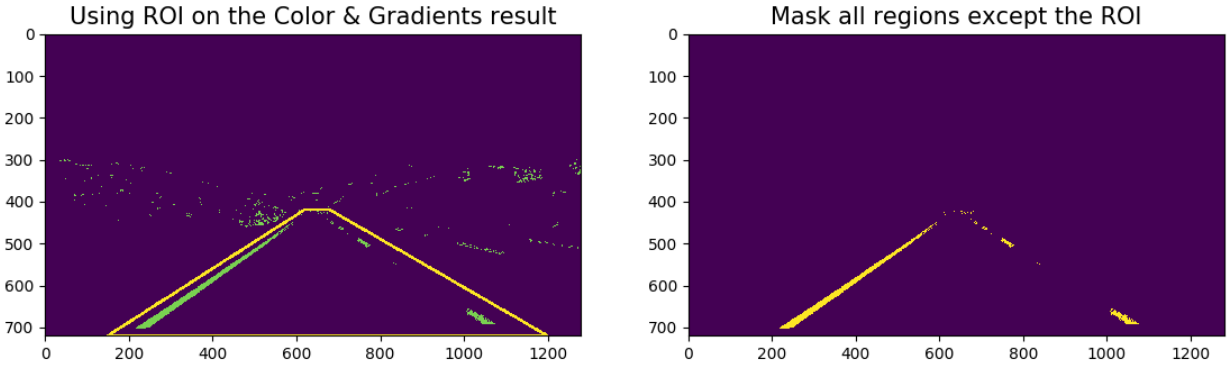

Figure 6. Masking other than the Region of Interest.

5. **Identifying and applying the Perspective Region of Interest (PROI)**: after exhaustive trials and errors the perspective region of interest has been identified and the applied on (warping) the resulting image of step 1 (the undistorted image) as shown in Figure 7. The identification is implemented in “P3 AdvLane2.py” lines 673 => 703.

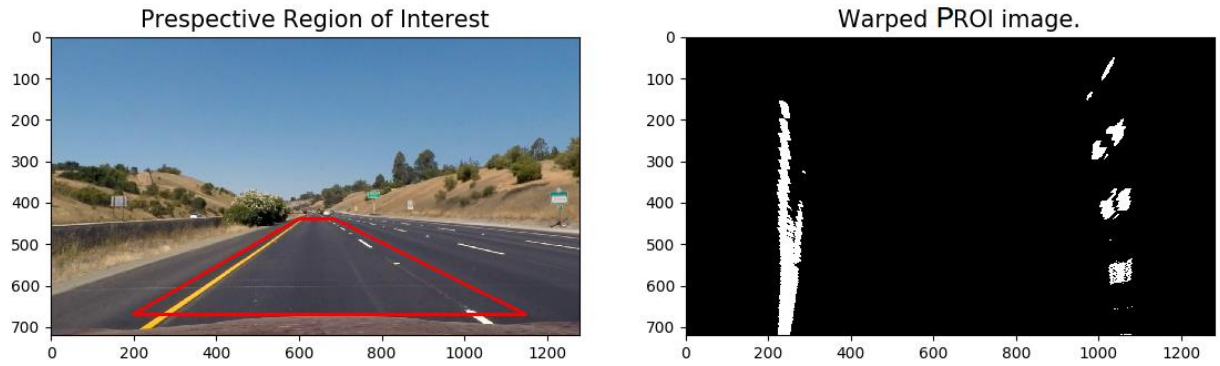

Figure 7. Identification and application of the Perspective Region of Interest.

6. **Applying Sliding Windows Search and Fit Polynomials:** the sliding windows search function "Locate_Lane_Lines_Fit_Polynomial()" has been is implemented in "P3 AdvLane2.py" lines 306 => 416. The function takes the "Warped PROI image" (shown in Figure 7) and produces the histogram and the left and right fit polynomials shown in Figure 8. The purpose of the function is the initial identification of lane lines points.

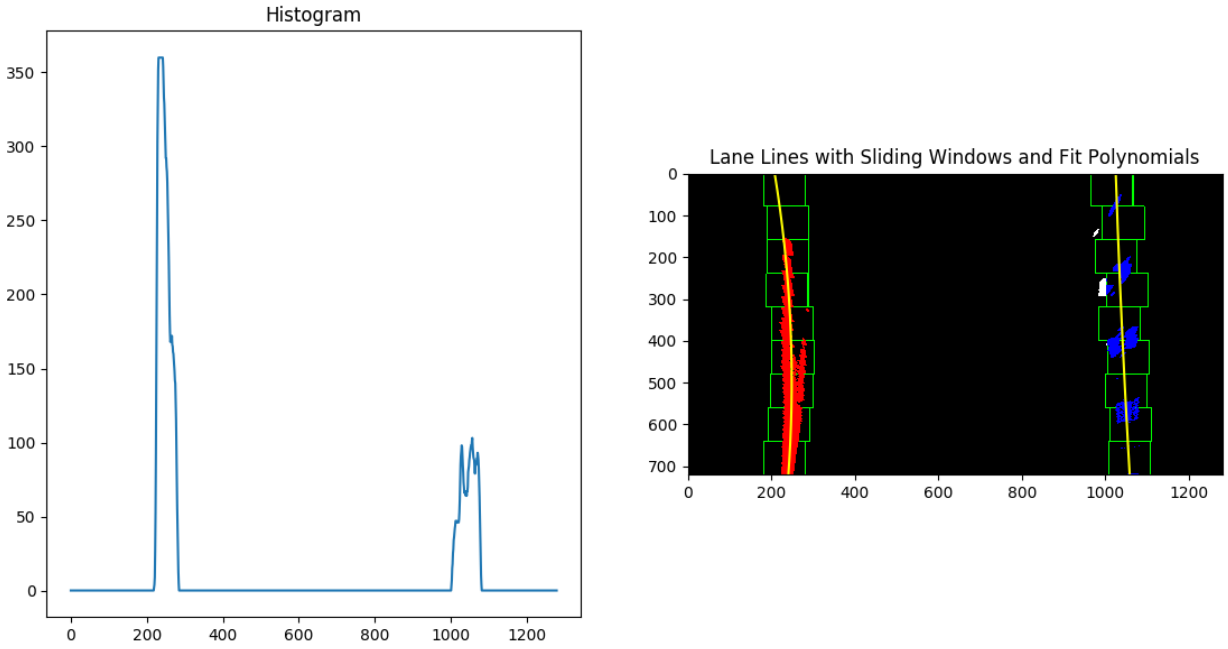

Figure 8. Histogram and sliding window search and the fitting left and right lane polynomials.

7. **Applying Recursive Search:** the recursive fine search function has been implemented as "Recursive_Search_Lane_Lines()" in "P3 AdvLane2.py" lines 419 => 545. The function takes the "Warped PROI image" (shown in Figure 7) and initial left and right polynomial coefficients from step 6 or itself from previous iteration (recursive) and produces the refined left and right fit polynomials shown in Figure 9. The purpose of the function is the recursive identification of lane lines points.

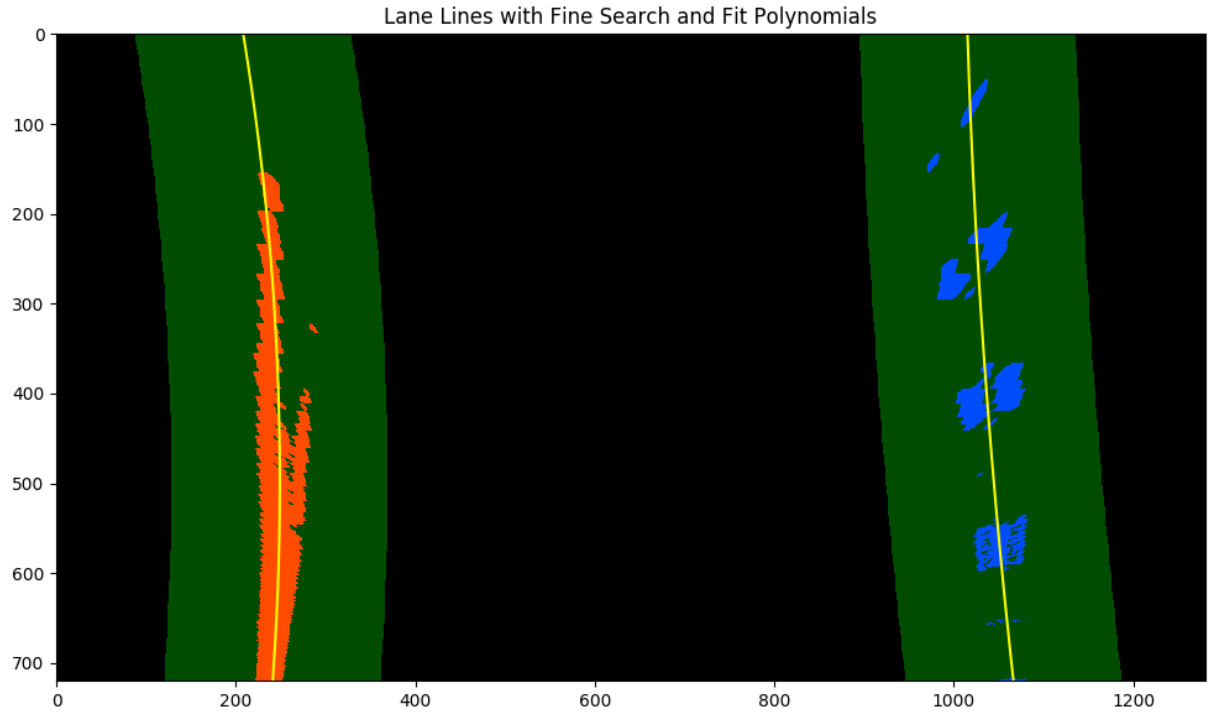

Figure 9. Recursive fine search and the fitting left and right lane polynomials.

8. **Measuring lane curvature and center:** the curvature of both the left and right lanes are then calculated as well as the position of the car with respect of the center of the lane. These calculations have been implemented as "Measure_Curvature_and_Center_Dist()" function in "P3 AdvLane2.py" lines 546 => 588.

9. **Unwarping the image:** the resulting image from step 7 (Figure 9) is being unwrapped using the "cv2.warpPerspective()" function and the calculated inverse warp Matrix "M_{inv}" from step 5. The implementation is in "P3 AdvLane2.py" lines 610 => 611.

10. **Highlight the identified lane**: as the final step in the finding lanes pipeline, the unwrapped lane lines are drawn back on the undistorted image (Figure 3) with the area between the identified lane lines been highlighted in green as shown in Figure 10. The values of the lane curvatures and the distance of the car from the lane center is printed at the top of the image. The implementation is done in "HighLight_Lane()" function "P3 AdvLane2.py" lines 591 => 620.

Figure 10. Identified lane lines highlighted in green and the measured lane curvatures.

4. Solution Approach

The following points shed some light on some technical tricks and aspects that have been tried or implemented in the described pipelines:

1. **HSV color space:** the function that converts the RGB images to HSV ones and extracts the designated channels H or S or V, has been implemented. The three channels have been tried to investigate the worthiness of adding any of them to the final overall combination. However, we find out that the S channel is exactly the same like the one in the HSL color space, so no need for duplication. The H and V channels, when we combine them separately or together with the overall combination, do not add value to the final result or sometime they make worse. As a result, we decided not integrate the HSV color space in the final combination.

2. **Perspective region of interest (PROI):** we tried to make the lane region of interest of the image (ROI) the same like the perspective region of interest (PROI), we couldn't make successful, and decided to define them separately for accurate perspective transformation.

3. **Sanity Checks:** several sanity checks have been used throughout the pipeline in order to try to remove pad lines to reach the final image. The following are a list of them:

a) **Order of the fitted left and right lines:** in order to ensure that the fitted left and right polynomials have intercepts with the x axis in the right order, we include this check as in "P3 AdvLane2.py" lines 491 => 494.
b) **Update of the left and right lines:** during video testing, both the right and left lane polynomial coefficients are getting updated each frame. Logically, the change of these coefficients should be small and if it is found big, this indicated that this line fit is not good enough and should be rejected. This check is done for the left and right lane separately in "P3 AdvLane2.py" lines 455 => 461 and lines 476 => 482 respectively.
c) **Check on the right and left lanes curvatures:** logically, the right and left lanes radius of curvatures should be almost identical all the time. However, doing this checks to separate good and bad line fits proved very tricky especially when the actual lane lines are vertical with no curves. The use of this method is not robust and can result in the rejection of many good lines. Therefore, it is not used.

4. **Applying Resets:** the initial lane line fits is being determined using the sliding windows search function "Locate_Lane_Lines_Fit_Polynomial()" mentioned in step 6 of the pipeline. Then, the next frames the recursive fine search function "Recursive_Search_Lane_Lines()"; that is mentioned in step 7 of the pipeline; kicks in to calculate the line fits for the next frames recursively. However, after several frames the estimation errors accumulate and the found line diverge and become unrealistic. For this reason, the sliding windows function has be used again to determine the lane lines from the raw undistorted images. In order to avoid this problem and make this procedure more systematic, reset procedure has been adopted. For each number of frames (determined by the constant "reset_span" which has been set to "8" in the current code in line 705), the sliding windows function kicks in and then followed by the recursive search function for several frames. In our current code, reset_span = 8, which the sliding windows function will work for the first frame and then followed by recursive search function for the next 7, and so on … etc. This technique is implemented in "P3 AdvLane2.py" lines 705 => 718, and works very well to avoid divergence and reduces the video processing time.

5. **Smoothing using FIR filtering:** after calculating the left or the right fit polynomials on a certain frame, instead of using it directly, we average the results over the last 3 samples to smooth out the determined values and reduces noise and unexpected jumps in directions. This step is done for both the left and right lanes in "P3 AdvLane2.py" lines 450 => 453 and lines 471 => 474 respectively.

6. **Lanes data acquisition:** using the Line() class implemented in "P3 AdvLane2.py" lines 34 => 64, we were able to record several useful information about the fitted left and right lanes. The collected information helped a lot in many tasks of the pipeline like "smoothing and FIR filtering", "sanity checks" … etc.

5. Shortcoming of the implemented approach

The following list summaries the identified shortcomings:

1. The current pipeline is still not performing at curves with the same level, as at straight lines, especially at harsh curvatures.
2. The performance of the implemented “Warp” function or the perspective transform is not consistent over several images with different lane orientations.

3. There many parameters to tune, you have a lot of trials and errors. There is no consistent way to make sure that you have selected the right value for a certain parameter.

4. The way the radius of curvatures are measured has a very big margin of error.

6. Suggested Improvements

The following list summaries the suggested improvements:

1. Use higher order of the FIR to smooth-out the transition of line fit values. The current used on is of order 3, I believe we can try-out even to order 20.

2. Need to design the perspective transform “Source” and “Destination” matrices based several images input not only one or two images. If there is a way to make it adaptive based on the input image, this will be a very good area of research.

3. Need to add “an FIR filtering” to the way we measure the curvature of the lanes as well as some sanity checks. We can even use the calculated value of the right lane to evaluate the left lane and vice versa. I think there is a big room for improvement in this part specifically, and can be taken as a future work.

4. The Line() class for the lane data acquisition can be enhanced a lot, several statistical function can be added for sanity check and help in better finding the lanes.

5. The slope properties of the fitted polynomials are used as sanity checks or as a way to improve the calculated polynomials coefficients. This part is a promising area of investigation.