Advanced lane Finding

Author: Aneeq Mahmood

Email: Aneeq.sdc@gmail.com

The goal of this work has to enhance lane finding techniques developed at the start of the term and do advanced lane detections which can especially work well against shadows, tire-tread marks, missing lane lines etc.

The individual rubric points are addressed below

Files Submitted:

Following files are submitted for this work:

main.py : File used to create the video output.mp4; the pipeline is present inside this file. The pipeline requires project_video.mp4 which is **NOT** checked in with this code**.**

output.mp4: output video file

main_without_pipeline.py: This script is used just to view output images after different stages, on the contrary main.py file does not show any output images. The backbone of the code is similar in both scripts. However, there is no pipeline function to accept images from a video stream in this script. It runs on the images in the test_images/ directory and contains no sanity checks.

pipeline_helper.py: This file contains all the helper functions needed by the pipeline

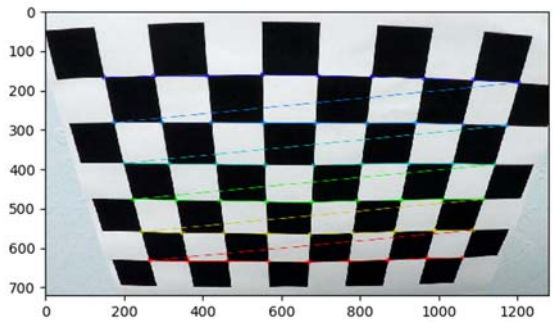
convol.py: Contains the documented code regarding convolution

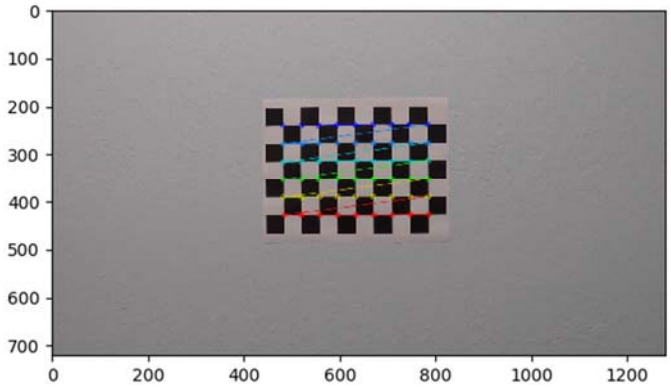
This write-up is also part of submitted work.

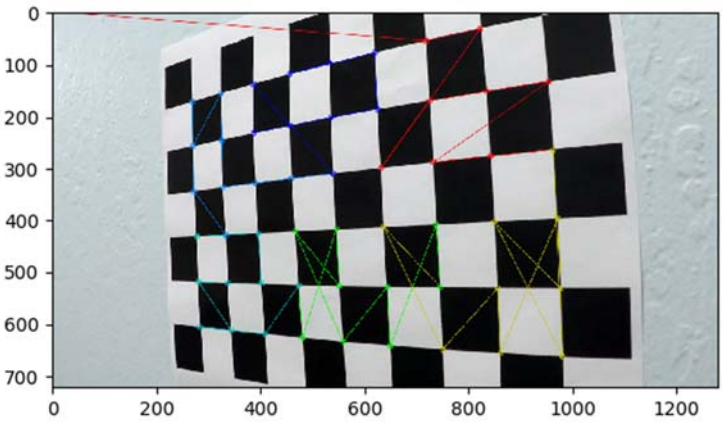
The individual Rubric points are addressed below

Camera Calibration.

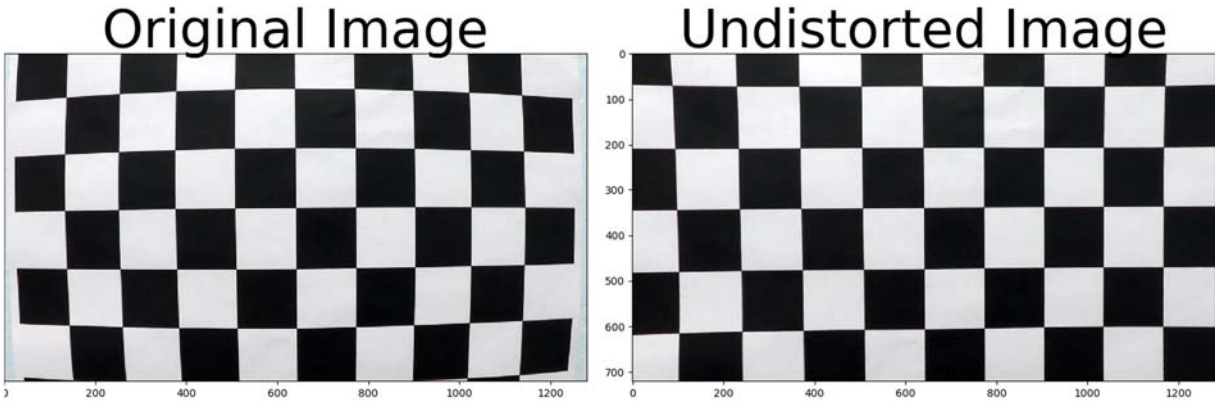
The camera calibration is using the calibration images inside the “camera_cal/” directory. The cameraCalibMatrices() function in the pipeline_herlper.py script uses these images as inputs, tries to find the corners, and if successful, stores the object and image points related to the image. Afterwards, cv2.calibrateCamera() is used to obtained a camera matrix and output vector of distortion coefficients. This whole function of cameraCalibMatrices() is based on the exercise which has been used in the classroom videos. Following images show the some cases of correct and incorrect chessboard corner detections. The object and image points for wrong corner detections are skipped. In total, out of 20 calibration images, only 3 images have been observed with wrong corners and they have been excluded in collecting image and object points.

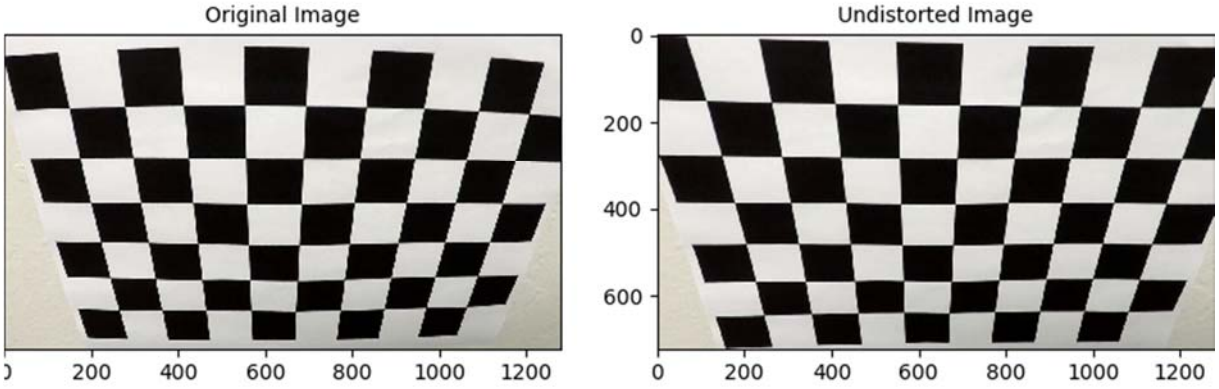






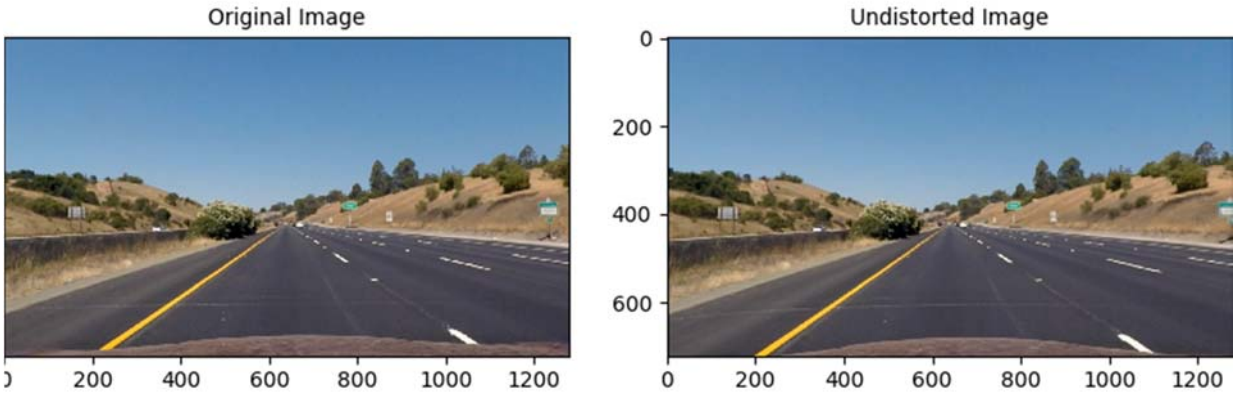
The result of distortion correction, using the distortion matrix and distortion coefficients, is shown below on two images which have also been used for calibration. The functionality is present in cameraDistremove() function inside pipeline_helper and is based on cv2 built-in functions.

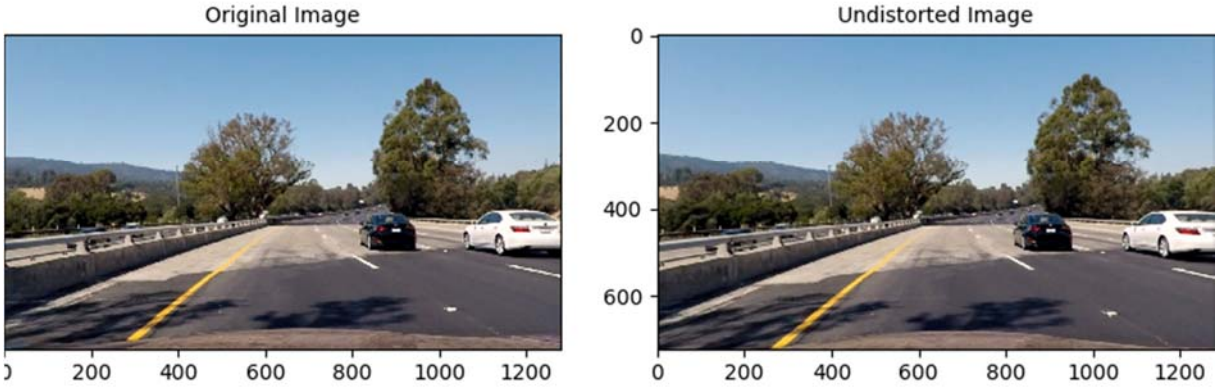




Pipeline Description:

The first step in the pipeline is to use the above mentioned functions to undistort the image passed as input. Below is the result of this image distortion shown for a couple of test images provided in the repository.

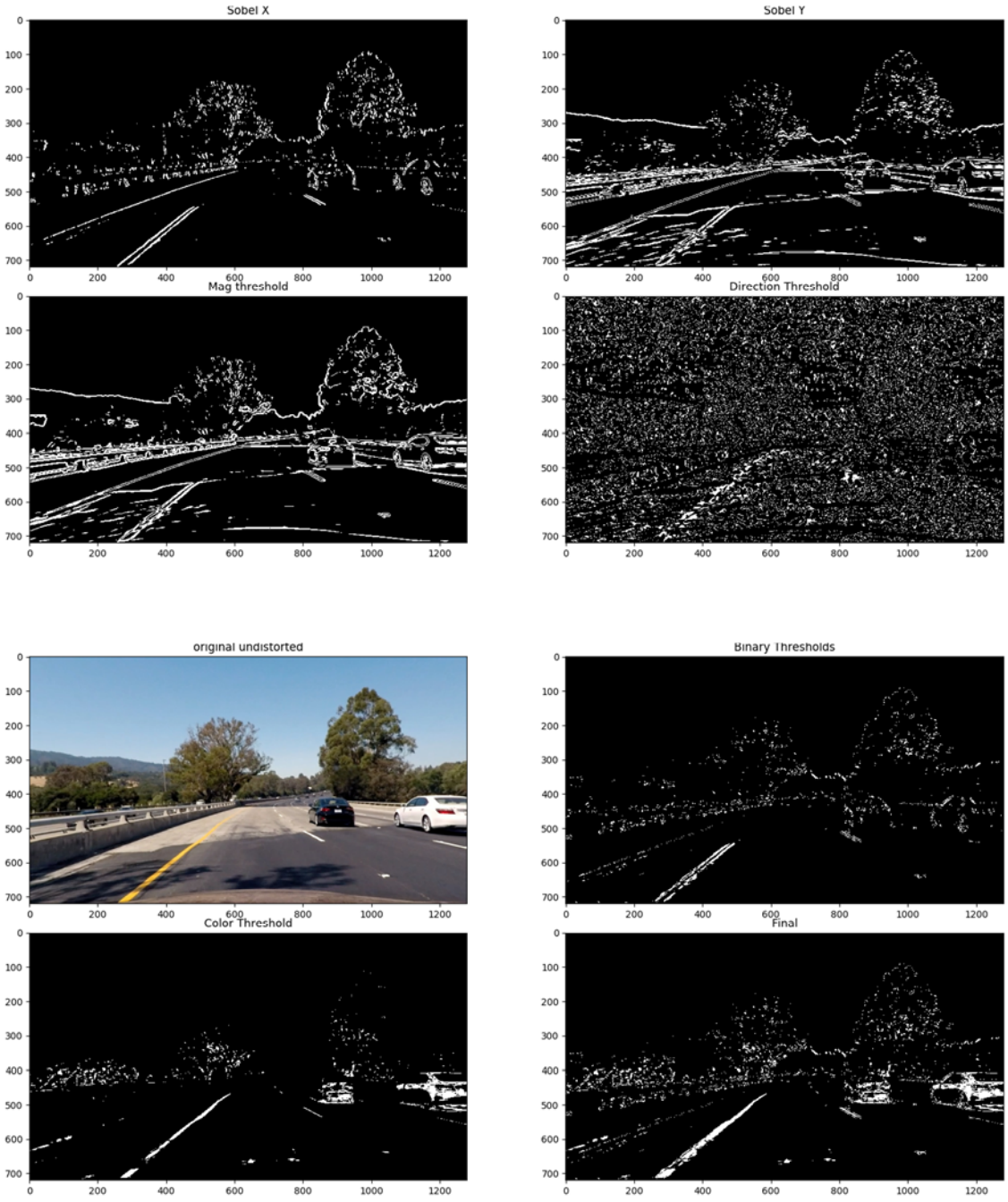




Binary Image Creation

The binary image to remove edges around the lane and to highlight lanes further is done in two steps. First calcGradients() function in line 88 of main.py is called to detect edges. This is done by calculating derivate in x and y direction using Sobel operator, combining the magnitude of the derivate, and doing directional threshold detection. The thresholds and Sobel kernel sizes are obtained through experimentation, and are described at line 39 of main.py. The four techniques are combined in the same way as the video lesson to create a single image. For all these four operations, instead of gray picture, the R channel is used, as it has proved to be more robust to detect lanes.

For color threshold removal, S channel is used as it also removes noise from the image but preserve lane lines. Line 80 in the main.py file has the function colorThr_s() which performs threshold on the S channel of the image. Finally, the binary images form color detection and gradient technique sis combined using the getBinaryImg() function in line 83 of main.py. The results of individual processing are given below. The first picture below shows four gradient based techniques to get the binary images. The second shows results of all operations on the input image.



In this second picture, top left is the original image, top right is the combination of all gradient based techniques, bottom left is the result of color threshold-ing, and last one is the final image.

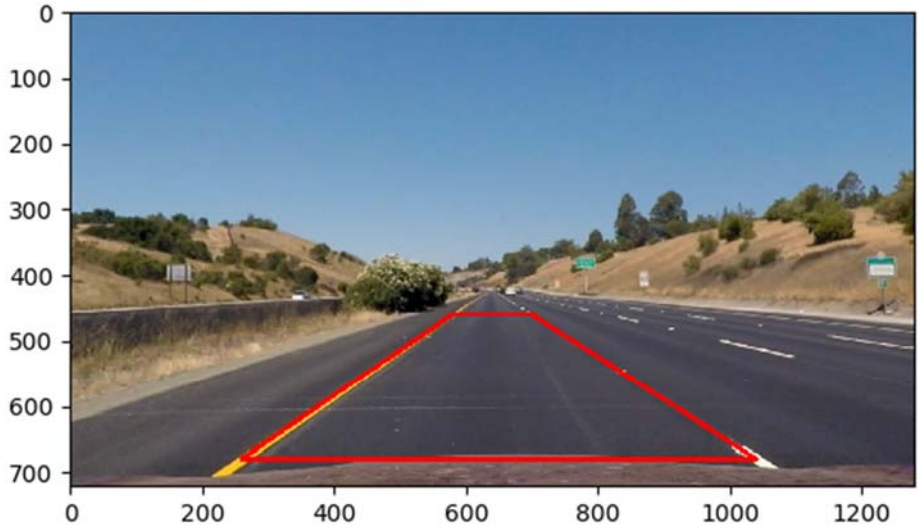
Perspective Transformation

This process has been inside the main.py script at line 86; this function itself calls the getPerspectiveTransform() to get warp matrix 'M' and warpPerspective() from cv2 to do perspective transform on the image. For this work, the warping is done on the lane line using the following x and y coordinates vectors, defined in the pipeline_helper.py file.

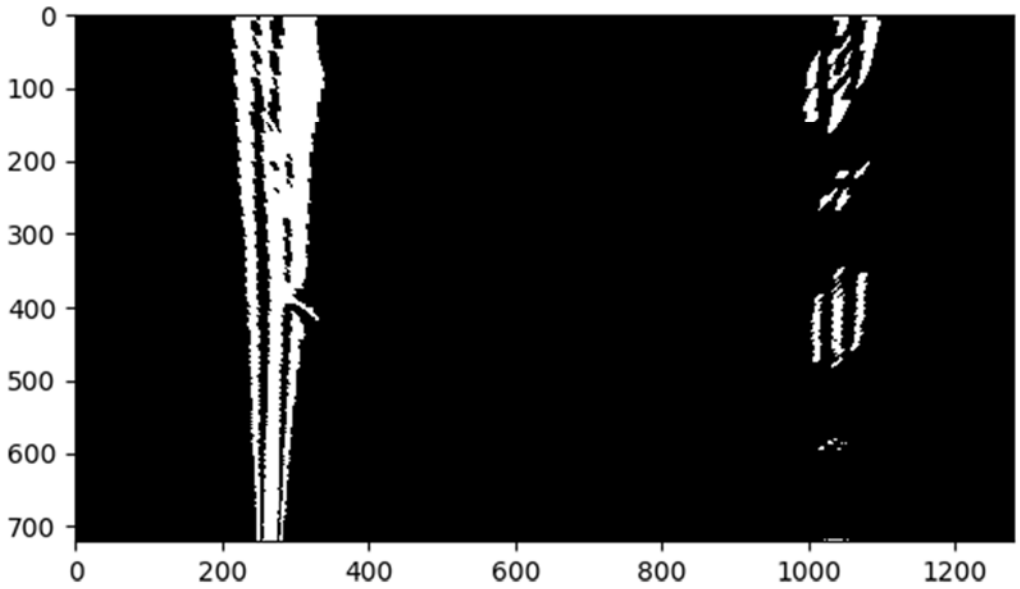
x = [580,700,1040,260]

y = [460,460,680,680]

These coordinates were mentioned in a discussion on the online forum, and as the author's points for not giving satisfactory result, they have instead been employed. The lane area marked by the above coordinates is shown below



The warped image is shown below

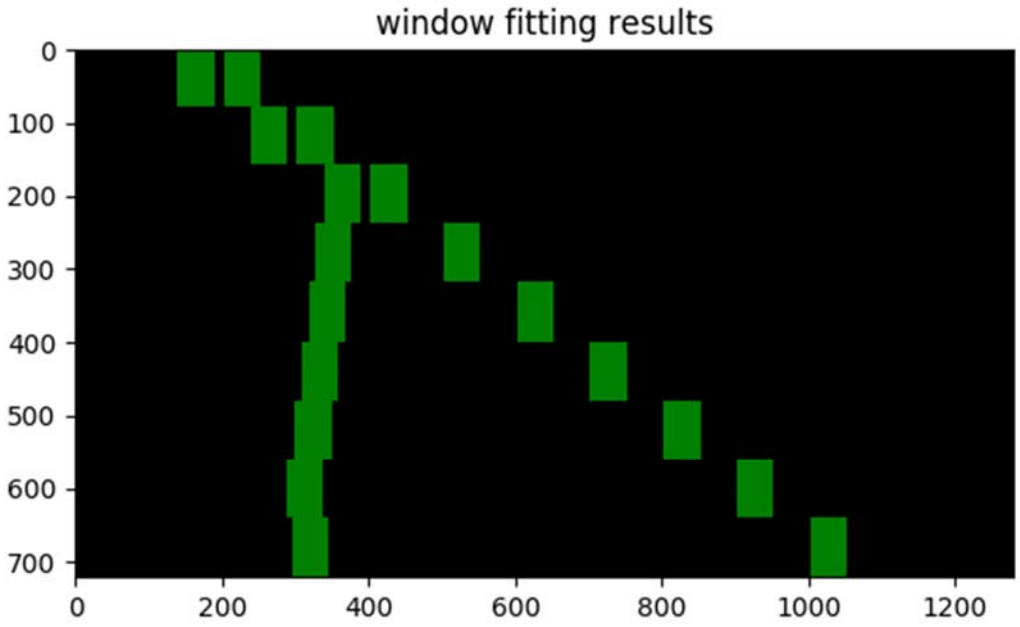


Lane Line Detection:

Lane lines detection on these binary warped images is done via convolution using the find_window_centroids() and find_all_centroids() functions in line 89 and 92 of main.py. These functions are defined in the convol.py and the basic code is similar to what is presented in the classroom lectures. A couple of changes are made to the code though; firstly initial left and right centroid are sought in last 1/8 of the image. Moreover, as these centroids only give the x-coordinates of possible lane marking, a y coordinate is also obtained by taking this y coordinate as the center of the area being convolved (in this case, center of the last $1/8^{th}$ of the image).

Thus at the end of find_all_centroids(), not only x- but y-coordinates of the lane marking are available. Thus we have points on both left and right traces of the lane, which are used to obtain proper lane markings using line fitting technique from numpy called polyfit().

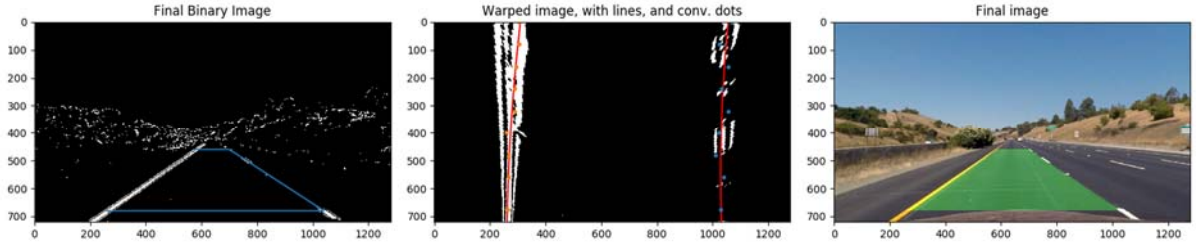
Here at this point, sanity checking is introduced inside the function checkLaneWidth() inline 518 of the pipeline_herlper file, as convolution does lead to wrong answers. Following image shows the case by showing wrong window fitting after convolution.



To avoid such scenarios firstly the left and right centroids should stay in their respective halves of the image, the difference (right-left) of centroids should never be negative, and any left centroid should not be negative. Lastly, as the lane width is 700 pixels, the difference of right and left centroids should not differ by a tolerance value of 100 pixels (0.5 m) around this value of 700. If these criteria are met, then centroids are considered 'sane'; they are used to obtain coefficients for left and right lane marking using numpy polyfit() function. These coefficients are saved in a deque() ring buffer of size 30. The buffer size is taken as it is not too big and still can give basis for reasonable estimates. The 'sane' centroids are also saved as the most recent reasonable lane points.

For obtain the right lane lines (lines 100 till 110 of main.py), first of all it is checked if the deque() is full of 'sane' values. If yes, then the average of coefficients of left and right lane lines are used to draw lines using fitVector() function (line 116). If the buffer is not full, then the current 'sane' values are used. If the current values were not reasonable, then most recent 'sane' values for centroids are obtained; if no reasonable old values of centroids are present (for example start of the video), then this frame is dropped. Once the left and right lane fits are obtained eventually using fitVector() function, radius of curvature is obtained for the left and right lane lines using getRadiusCurve() on line 119, and their average is used to obtain a final value. The above function is based on the code shown in the online classroom.The getCameraOffset() function is used to see how much a the central camera (the car itself) is off from the centre. Lastly, the lane markings are then used inside the warpToColor() function on line 128 to map the lane lines on a unwarped image. This function uses the code provided in the lessons to perform this action, and shows the marked area on the initial colored image.

The results of the whole process are summarized in the figure below:



The first image on the left is the binary undistorted image with a blue box showing the area to be warped. The middle image is the warped section of the road showing the bird eyeview of the road ahead. The blue and orange dots are the centroids obtained after convolution. The red line on the white lane marks are obtained from the polynomial-line fitting. The third image shows the mapping of the warped area back onto the road.

The pipeline is used on the project video to create output.mp4. The initial lane markings have some jitter as the ring deque buffer is still being filled at this point. However, once it is full, the lines become smooth. The lane markings do not fail in the presence of shadows or with the changing lane colors due to sanity checking and smoothing effects. Also, the radius of curvature remains reasonable and does not exceed above 1.5m. However, when the lane is straight, then the radius of curvature tends to reach very high values because the function tries to calculate the derivate of a straight vertical line. For such cases, there is a fixed limit on radius of curvature being portrayed on the image of 3 km, and values above this are replaced by previous value, which was below the threshold of 3km.

Possible Improvements

There are number of ways this work can be improved.

1) As above mentioned, radius of curvature does not give reasonable values when the lane is straight, a better work around rather than displaying the last acceptable value should be devised.
2) Initially, it was thought that radius of curvature can be an important metric to see verify correct operation. However, as mentioned above, it does not work all the time, and offset from centre seemed apparently a better choice. At the end, average lane size appeared a better choice to do sanity checks. A combination of these two plus additional other metrics such as difference between radii of curvature of the right and the left lane line can be used for sanity checks.

3) During the project, the convolution does tend to fail under shadows and where lane lines are not clear. In the project, the ring deque buffer was full by the time it occurred, and older values of centroids were used for those images. This worked fine for the project video. However the challenge videos has strong curves and some shadowy sections right at the start where convolution failed, and a lot of frames are rejected at the start. Hence, initial lane markings are off the road and are not acceptable for those challenge videos. Thus, some more work is required especially to improve the performance of convolution operation.