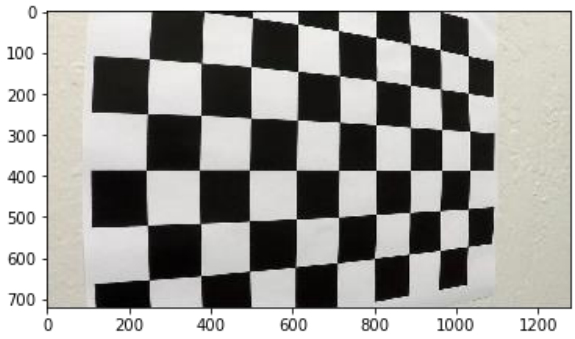
Advanced Lane Finding Project

1. Camera Calibration

The code for this step is included in the third code cell of the IPython notebook. 20 calibration images used for calibration. The object points of the corners has coordinates (x,y,z) where z is always zero as the board is in the flat image plane. The diagonally opposite corner of top left corner of the chessboard is taken as the origin.

For the image points, we need to find the chessboard corners. Cell 3 of the notebook uses the function to find the corners. The object points are same for all images but we get different image points of the corners. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

The output of the objpoints and the imgpoints is used to compute the distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained the result shown below:



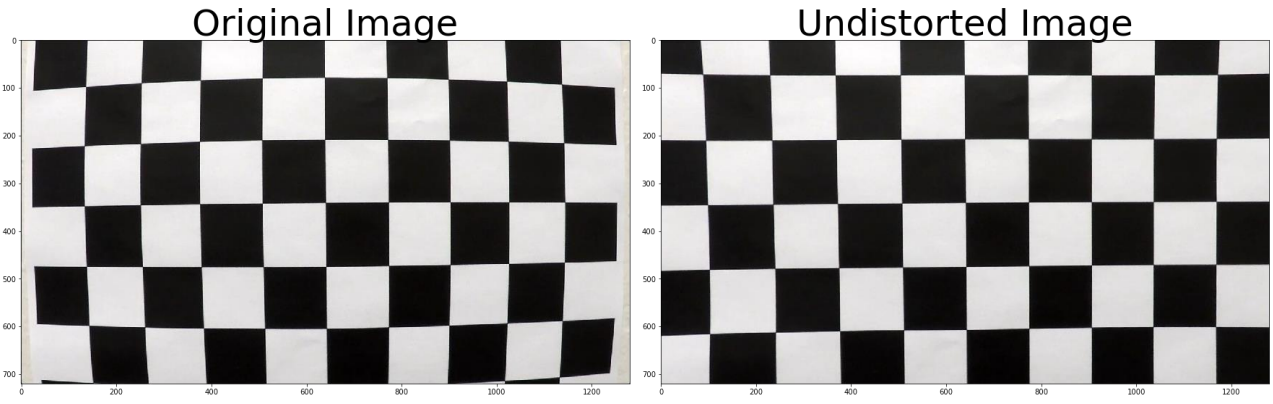


Figure 1 Undistorted Image

2. Example of a distortion-corrected image.

The image below shows the undistorted image. The distortion correction step is included in cell 10 of the Ipython notebook

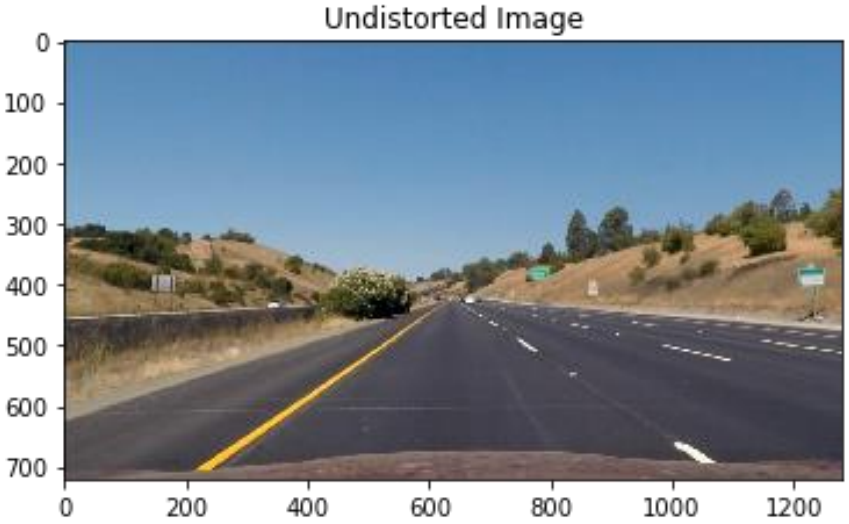


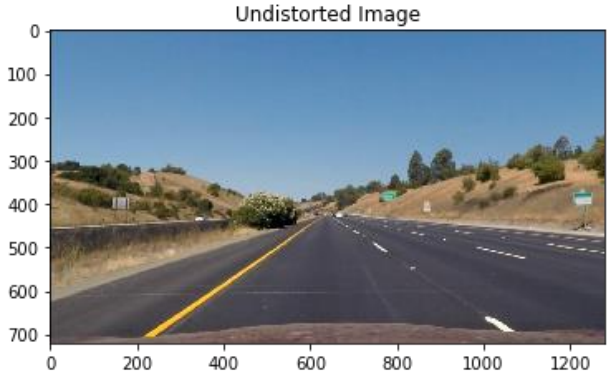
Figure 2 Undistorted Image

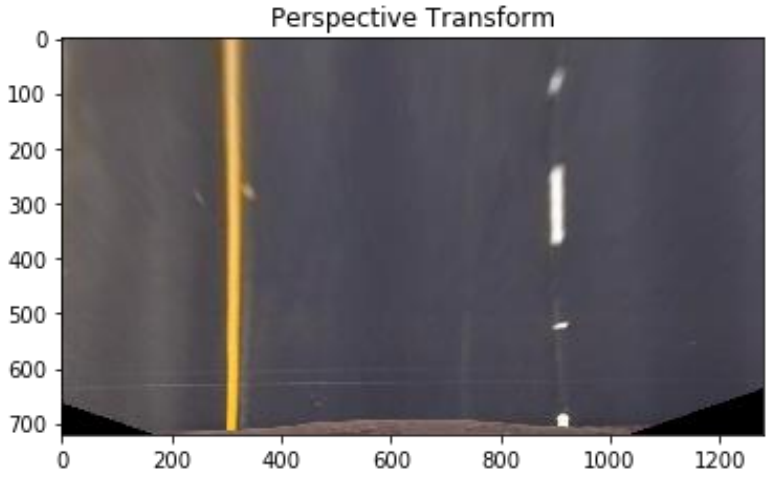
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

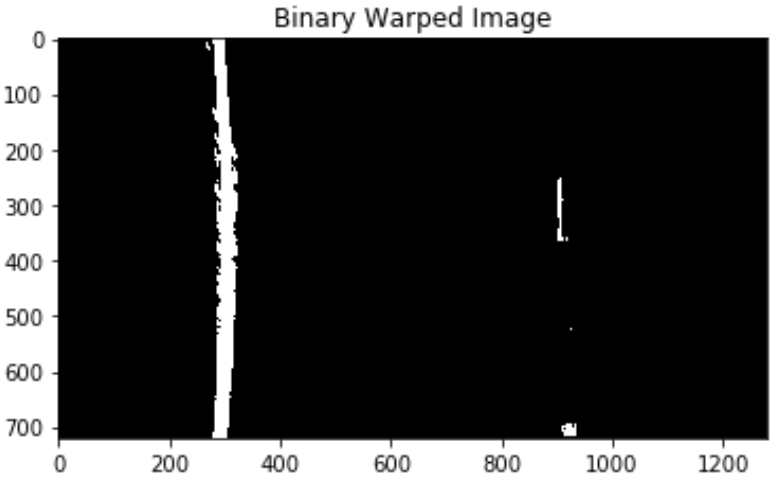
The code of the perspective transform is defined in the function called warp() in cell 45. The source and destination points are shown below. The point were chosen and finalized after a number of iterations.

Sno.	Location	Source Points	Destination Points
1	Bottom Left	220,720	320,720
2	Bottom right	1,110,720	920,720
3	Top Right	722,470	920,1
4	Top Left	570,470	320,1

The example of the transformed image is shown below:



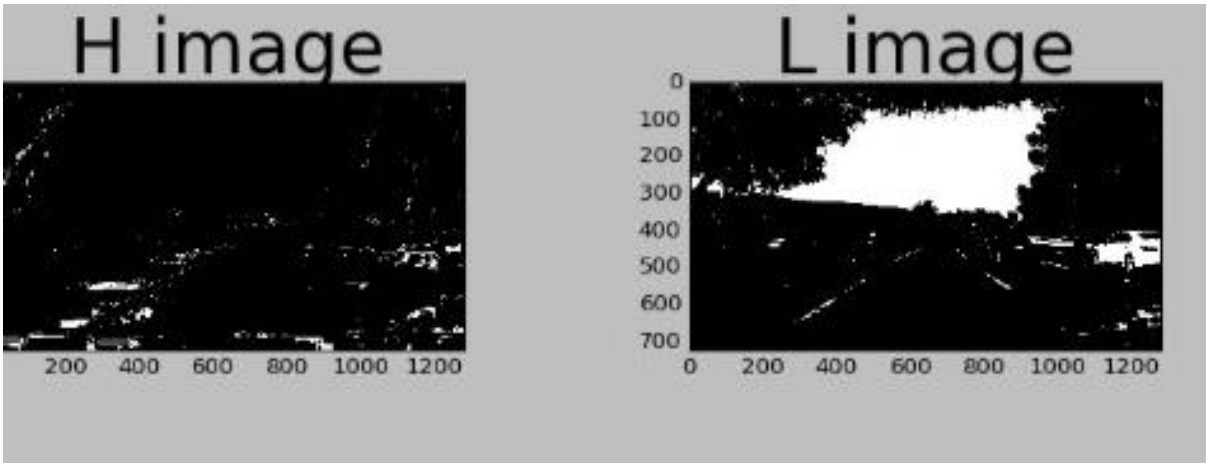


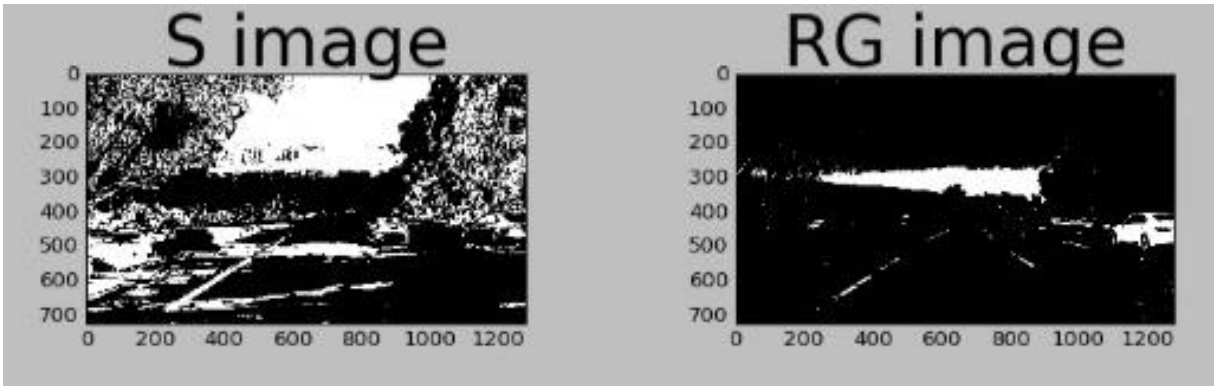


4. **Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?**

Step 1: The first step in this section is the exploration of the color channels for the image shown below as this is one of the challenging image given that it has shadows

The exploration of the color channels is shown below (Cell Number 42,43)



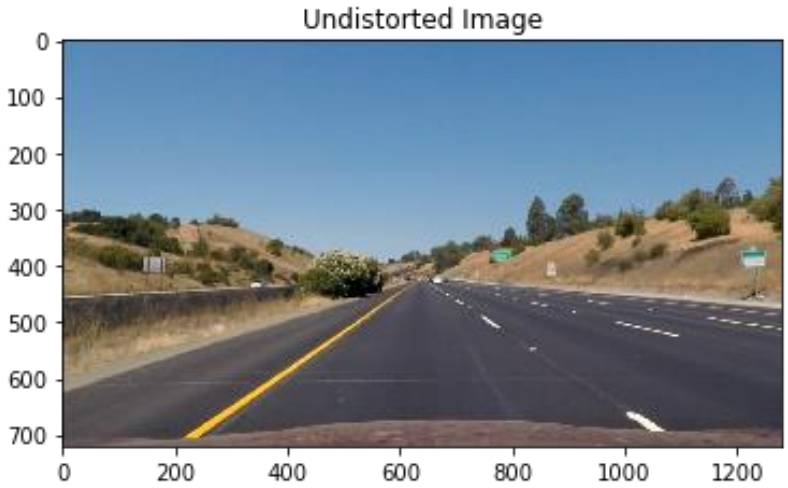


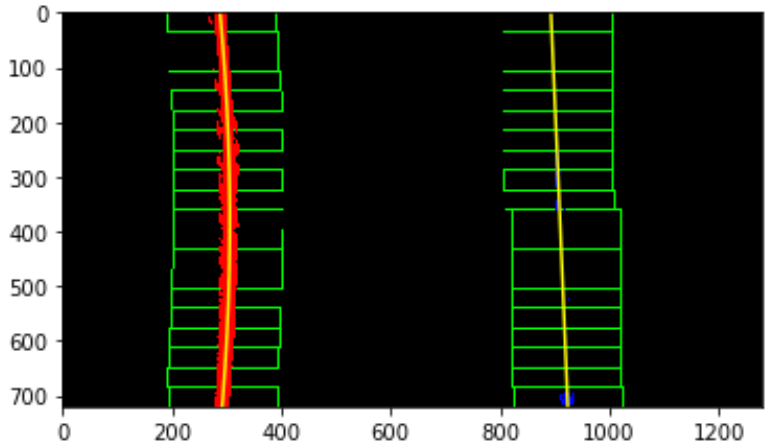
Step 2: After a lot of iterations, the channel threshold were tuned and the tuned parameters are shown below:

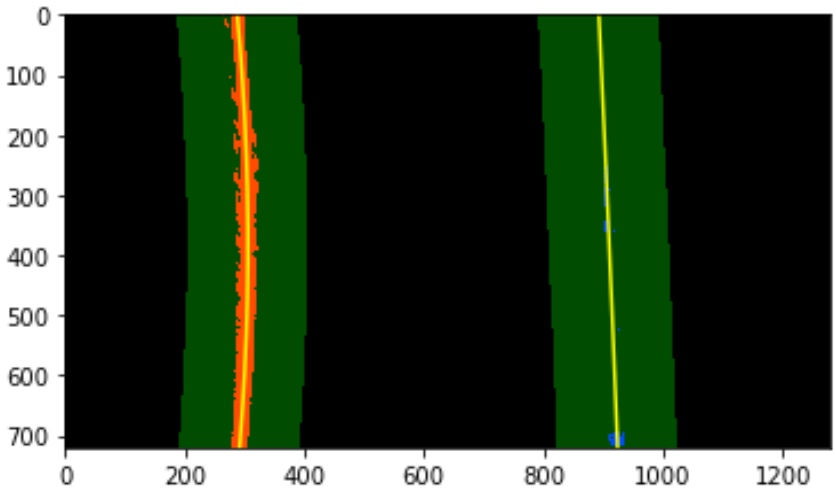
The function process image() in cell 41 explains the process to extract the lane features. The V ,S and L channel are used along with the gradient in the x- direction.

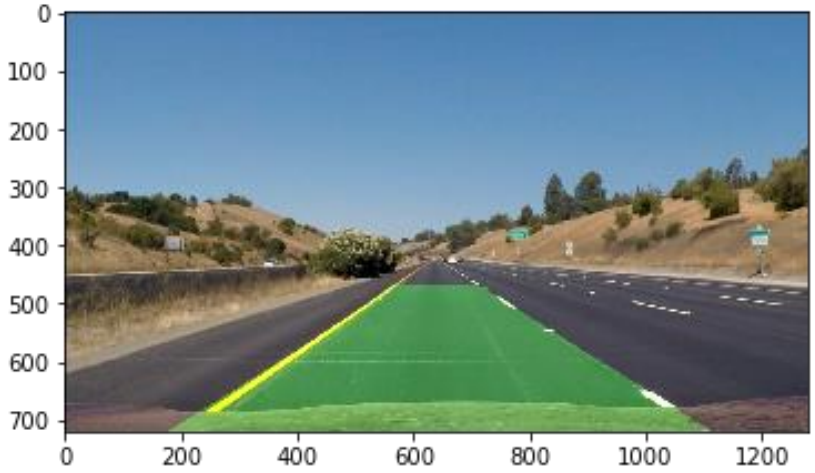
Sno	Parameter	Minimum Threshold	Maximum threshold
1	S channel	110	255
2	V channel	45	255
3	L channel	80	255
4	Sobel x Gradient	100	255

This was tested against the test images









For the video feed the function lane_pipeline coded used (cell 38). The functions used are

1. Warp()- Perspective transform image
2. Sliding_Window- Sliding window search function
3. Line_polyfit – returns the polyfit for the left and right lanes
4. Lane_storage- Used for averaging of lanes using 20 previous frames to ensure stability of the lines
5. Average Radius- cell(33) . Calculates the radius of the curvature
6. Lane_pipeline – uses all the functions and returns the final result. (cell 38)

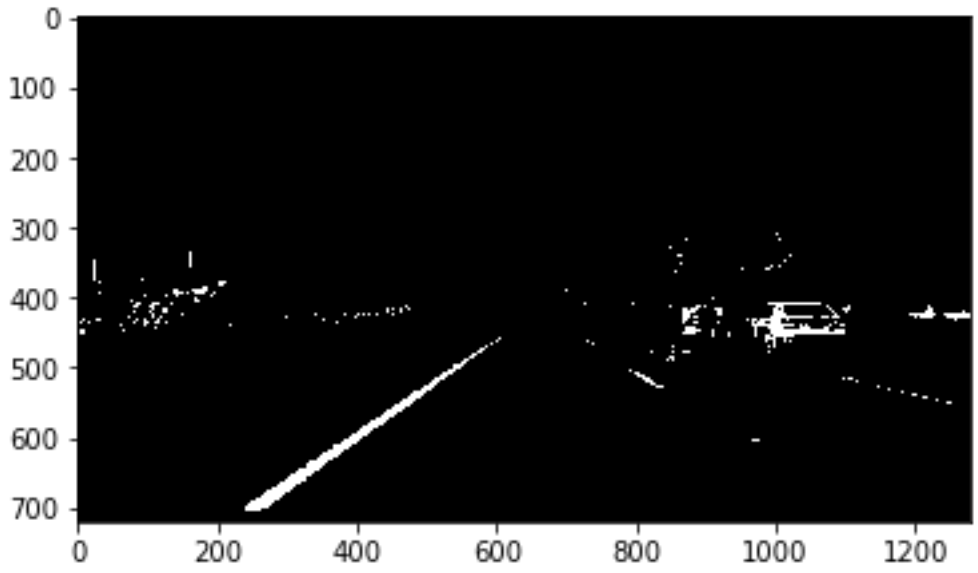
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

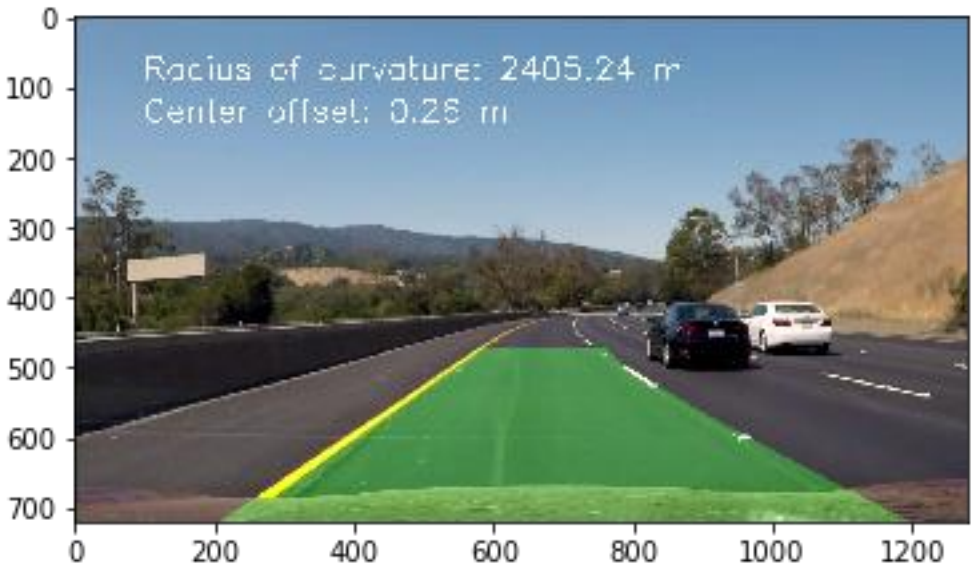
To calculate the radius of curvature, the meters per pixel in the x and y directions to repolyfit the lines. The code piece is shown below. The radius of the curvature is calculated as the average of the left and right line curvature

```
def average_radius(image,left_line, right_line):
    ym_per_pix = 30/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension
    ploty=np.linspace(0, image.shape[0]-1, image.shape[0])
    y_eval=np.max(ploty)
    left_fit_world = np.polyfit(ploty*ym_per_pix, left_line*xm_per_pix, 2)
    right_fit_world = np.polyfit(ploty*ym_per_pix, right_line*xm_per_pix, 2)
    # Calculate the new radii of curvature
    left_curverad = ((1 + (2*left_fit_world[0]*y_eval*ym_per_pix + right_fit_world[1])**2)**1.5) /
    np.absolute(2*left_fit_world[0])
    right_curverad = ((1 + (2*right_fit_world[0]*y_eval*ym_per_pix + right_fit_world[1])**2)**1.5) /
    np.absolute(2*right_fit_world[0])
    average=(left_curverad+right_curverad)/2
    lane_center = (left_line+right_line)/2
    offset= abs(image.shape[1]/2 - lane_center)
    offset_world=  offset*xm_per_pix
    return average
```

The position of the vehicle of with respect to the center is based on the offset calculation. Its calculated at the start of the line. The offset is the difference between half width of the image and the lane center. The lane center is the difference between the right and left line at index 720. Finally the offset is multiplied by the meters per pixel in the x direction. Refer to the cell 37 for details

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





7. Discussion

This project was tedious in terms of tuning the color channel thresholds and to find the right combination of channels and the gradients to eliminate noise and extract only the lane features. Another problem that I faced was to implement the sanity check, I finally used running averages but was trying to implement another algorithm, The pipeline can fail in very abrupt and steep turns but it handles very bright images as V channel captures them well and darker regions with shadows as well with the help of the L channel. The S channel is another filter used.

To make the algorithm more robust, I would try to further tune and explore the combination of color channels. Furthermore, code refining is important as latency has to be minimal. To further improve my algorithm, I have decided to collect some lane data using cameras during the night and use it test my algorithm and fine tune parameters which would be applicable for both day and night driving