Udacity Advanced Lane Finding Project Report:

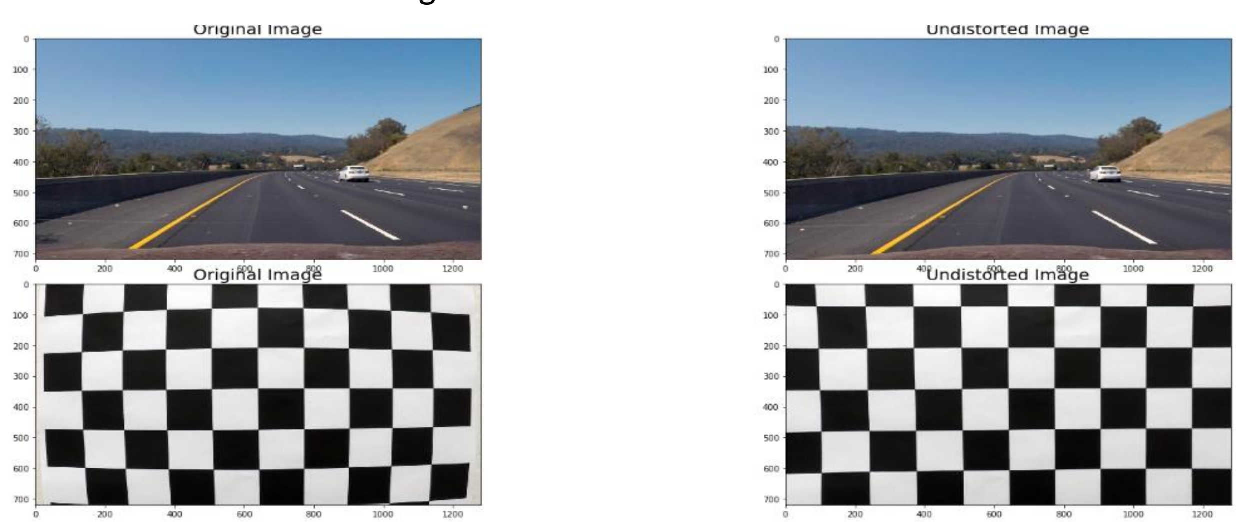
Advanced Lane Finding Project The goals / steps of this project are the following:

1- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
2- Apply a distortion correction to raw images.
3- Use color transforms, gradients, etc., to create a thresholded binary image.
4- Apply a perspective transform to rectify binary image ("birds-eye view").
5- Detect lane pixels and fit to find the lane boundary.
6- Determine the curvature of the lane and vehicle position with respect to center.
7- Warp the detected lane boundaries back onto the original image.
8- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration:

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. 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.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result on one calibration and one test image:



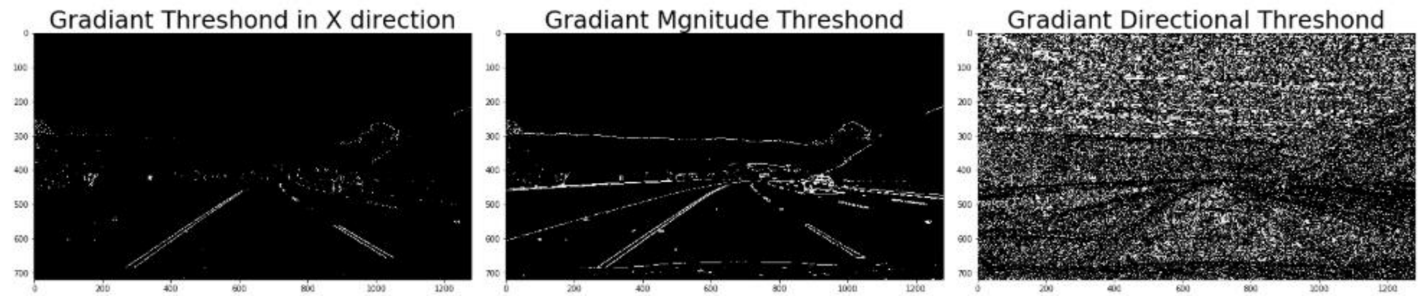
Gradient Threshold:

I've tried different gradient thresholds getting this result by using :

1 - abs_sobel_thresh to calculate binary threshold in x or in y directions.

2 - mag_thresh to calculate binary threshold of the magnitude of sobel_x and sobe_y.

3 - dir_threshold to calculate binary threshold of the directional gradiant of sobel_x and sobe_y.

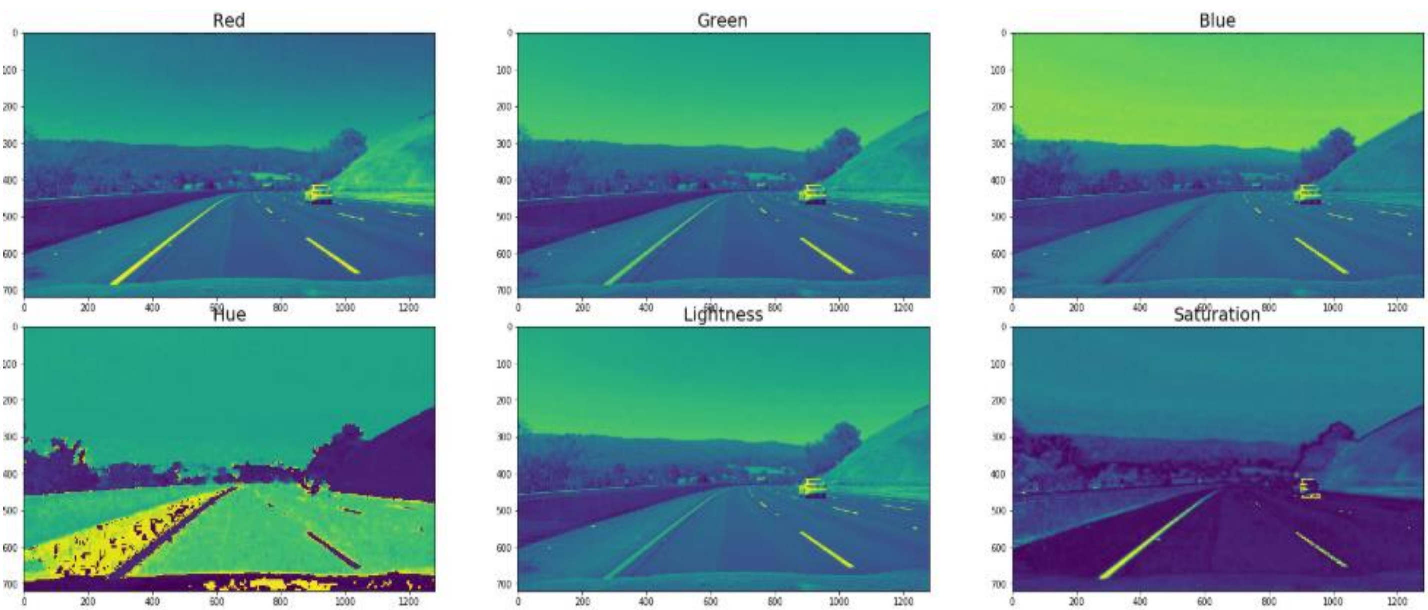


So I think gradient magnitude gives us the best result .

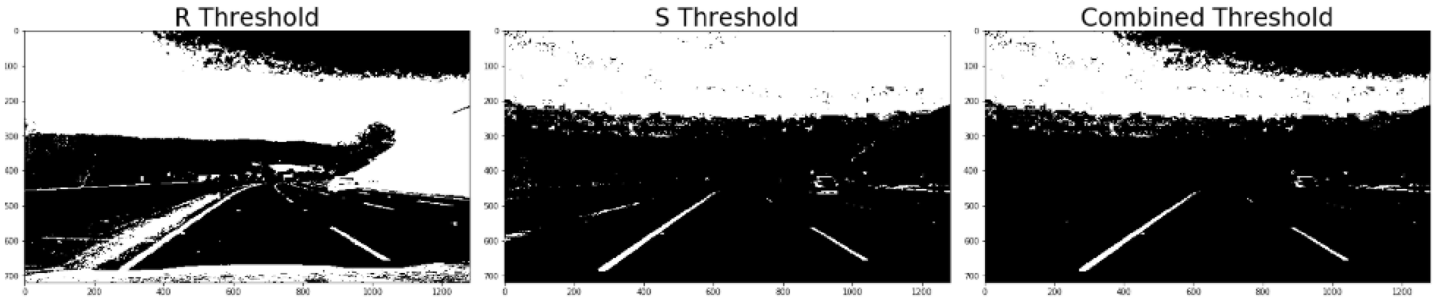
Color Threshold:

I've tried different color thresholds in different color spaces and different color channels:

1 – Visualization of RGB channel and HLS channel.



2- Through trial and error I found that applying a threshold to the S channel in the HLS color space and applying a threshold to the R channel in the RGB color combined with gradient magnitude threshold gives a good indication of where the lane is located:

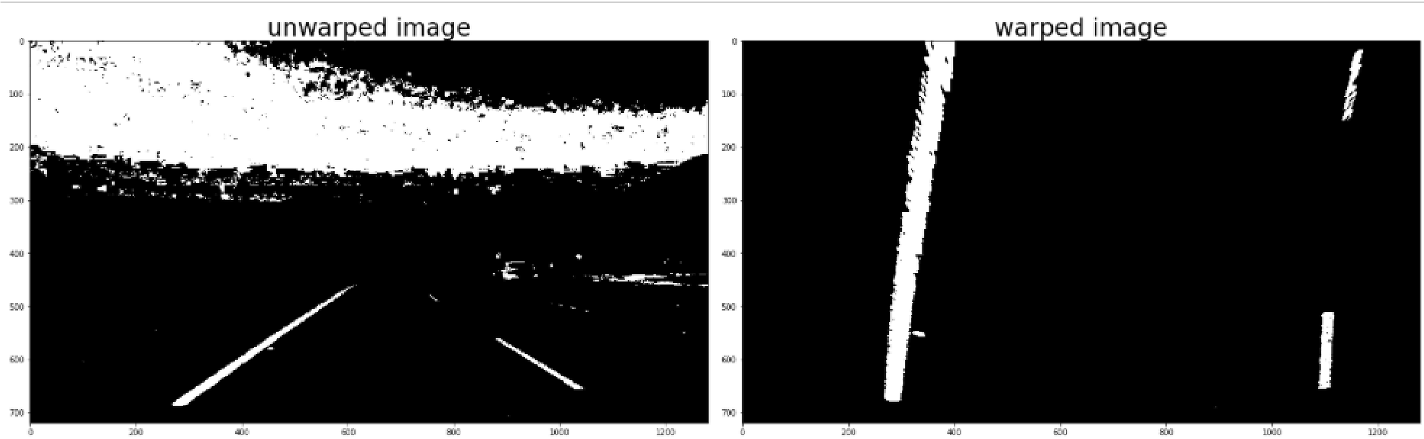


Perspective Transform:

1- Identify 4 points in source and destination images.

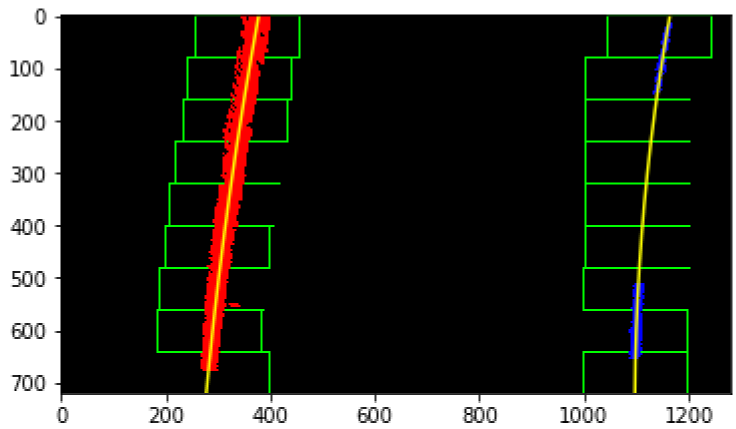
```
# Four source coordinates
src = np.float32([[760,490],
                  [1100,690],
                  [280,690],
                  [550,490]])
# Four desired coordinates
dst = np.float32([[1100,155],
                  [1100,680],
                  [290,680],
                  [300,155]])
```

2- Apply cv2.getPerspectiveTransform to get the transformation matrix and then apply it to transform the whole image using cv2.warpPerspective



Line Fitting using sliding window:

Using sliding window algorithm to obtain the indices of the lines candidate pixels and then fit those pixels into quadratic function.



Measure curvature:

After getting the lane fitted with the quadratic polynomial, we can calculate the curvature using this formula:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

The offset from the center is simply the difference (in real world meters) between the center of the video stream and the midpoint of the two detected lane lines.

Final result:

After applying the preprocessing and fitting the lanes and calculating the curvature, we undo the perspective transform using unwarp function getting that result: