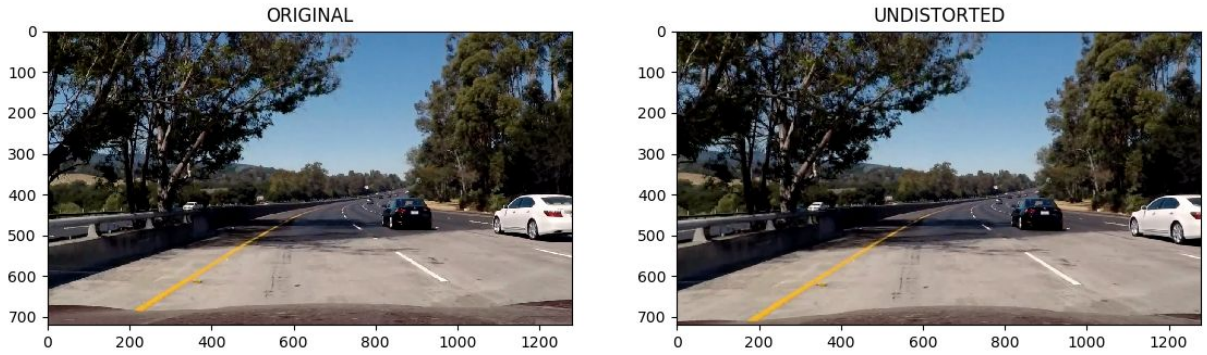
Advance Lane Line Write Up

Camera Calibration

I wrote a script : `undistort_images.py` which goes through the images in camera_cal and learns the calibration matrices and vectors.
The script further displays the distorted and distorted images side by side



Note:
However, I realized that there is no point in going through all the calibration images as only one of the transformation vectors will be ultimately be used.
So, I go through only one of the chessboard images, and save it's undistortion matrices as a pickle file so it can be used later.

Pipeline:

All the code relevant to the pipeline can be found in `image_pipeline.py`
Here I define a function for each stage of the pipeline and then call it while processing the video

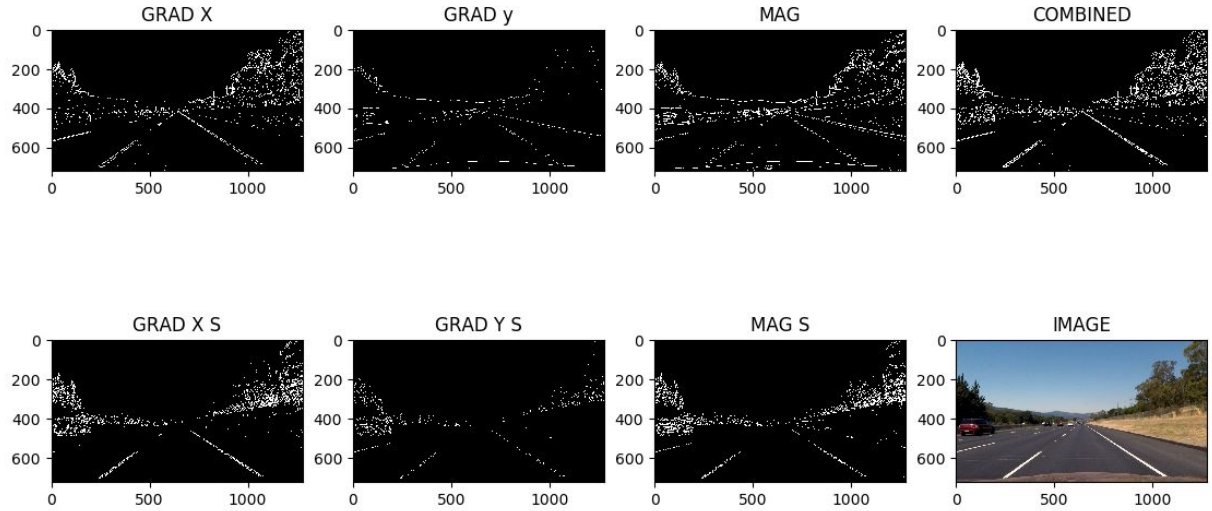
1. Undistortion:

This uses the `undistort` function to undistort the image.

This is how the undistorted image looks.
The function takes the calibration parameters which were pickled and uses them to undistort the image.

2. Image Thresholding:

A certain amount of experimenting was required in order to make the thresholding feasible. To experiment with the different possibilities, I plotted them together which brought out the differences and picked the combination which gave the best results.



I found that the following combination gave the best result:

```
(gradx == 1) | (gradx_s == 1) & (mag_binary_s == 1)
```

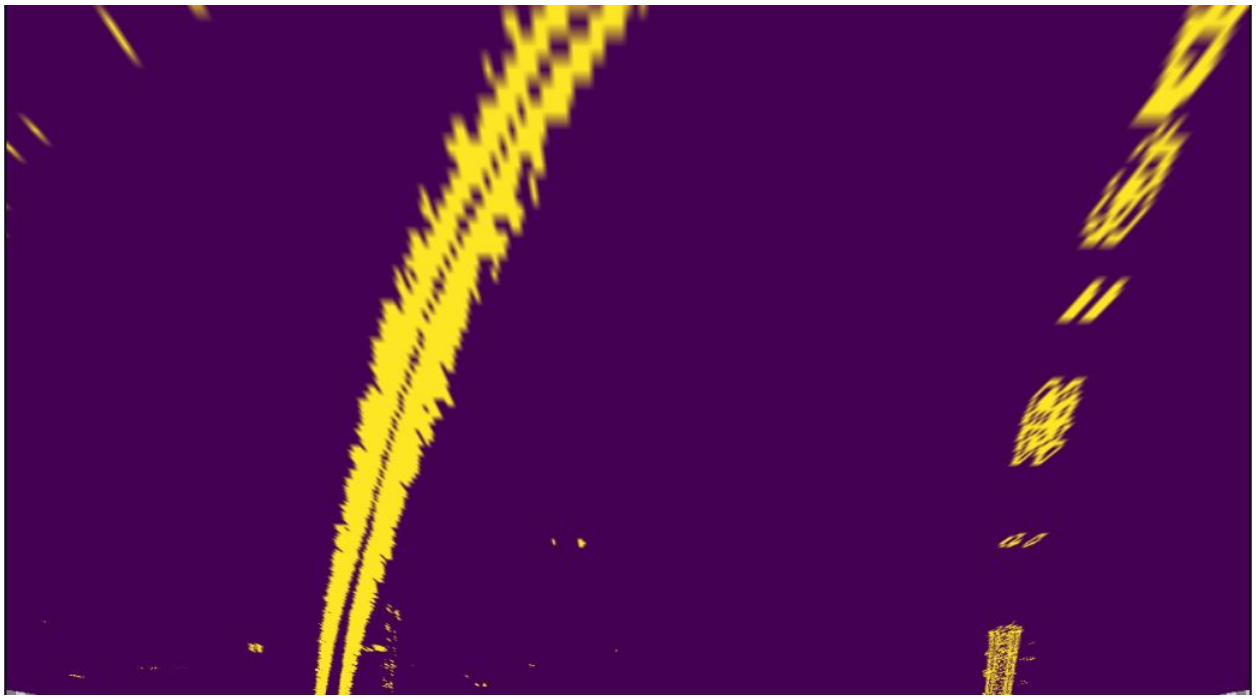
Ie

```
(Gradient on X OR Gradient on X of the S channel) AND Magnitude Threshold on the S channel
```

Once, I had experimented with the threshold values and the combinations, I put the final selection in the function: `threshold` in `image_pipeline.py`

3. Warping the images:

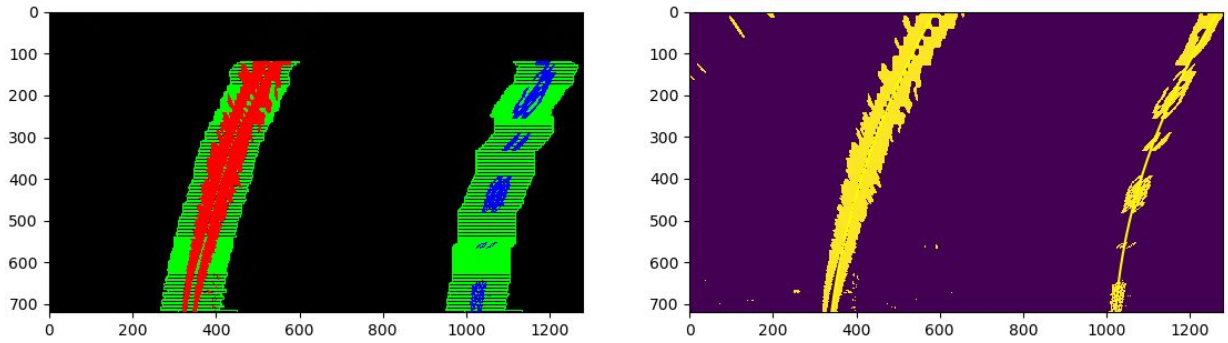
I had to calibrate the quadrilateral on the straight road image to get the right trapezium to do the warping.



The warped images have parallel lines.

4. Finding Lane lines

I tried both the convolutional and histogram methods.
I found that the histogram method was less prone to noise terms which were further away from the lane lines in general.
So, I adopted that method.
I tweaked different values of number of windows till I found a value that worked : 150
This mostly got the lanes right. I plotted the output.



After finding the lane lines in the first sweep, I look for regions of interest only. For this I make use of a global variable `history` which keeps track of the previous found line and searches for margins withing it.

5. Getting the Radius of Curvature

Once, the lane line pixels were established with the left and right lane lines individually, I used polyfit to get the equations.

Then, using the equation for Radius of Curvature, I got it's value for each lane and added it to the image.

Here, the values are: left radius, right radius, offset from center

The final processed image was saved and can be found in `output_video/final.mp4`