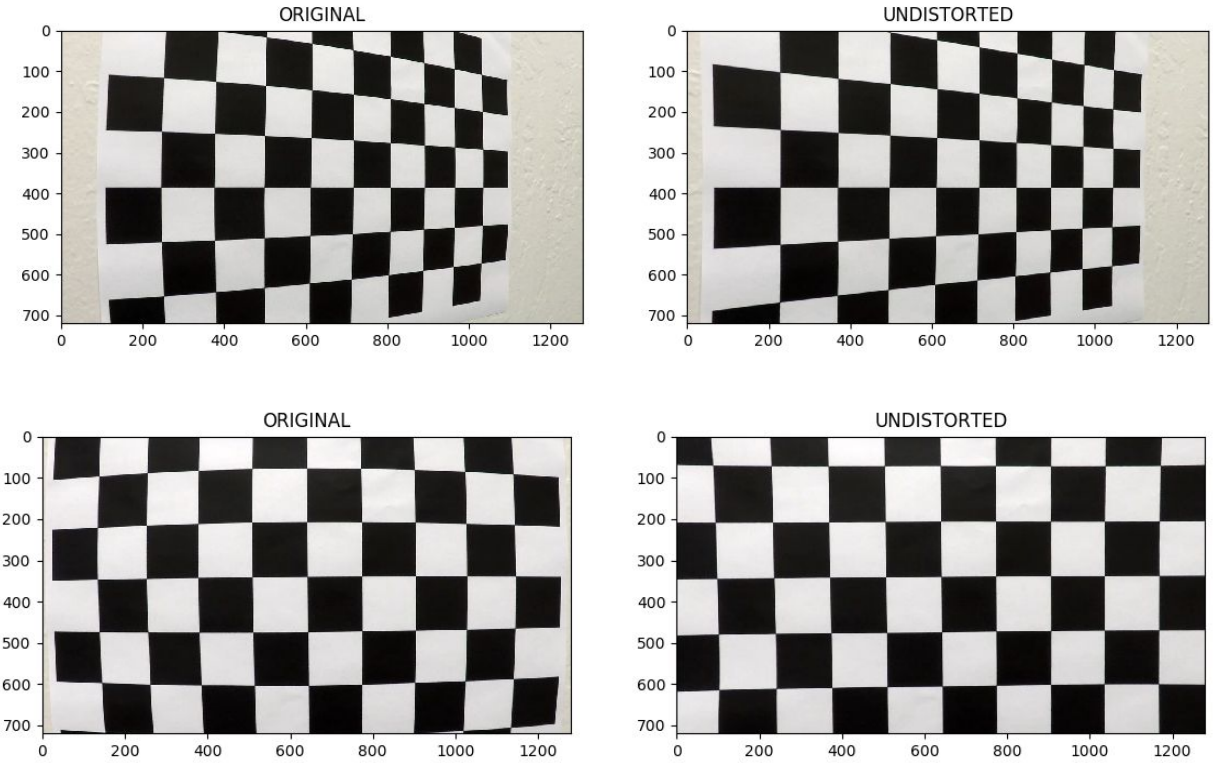
Advance Lane Line Write Up

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



As can be seen, the undistortion is doing a great job.

I save the undistortion matrix data so it can be used later in mt pipeline.
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.~~
This was an incorrect assumption in my earlier submission.
We need a the whole set of object points and image points from all the calibration images, which in its entirety helps to get the undistortion data.

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. Warping the images:

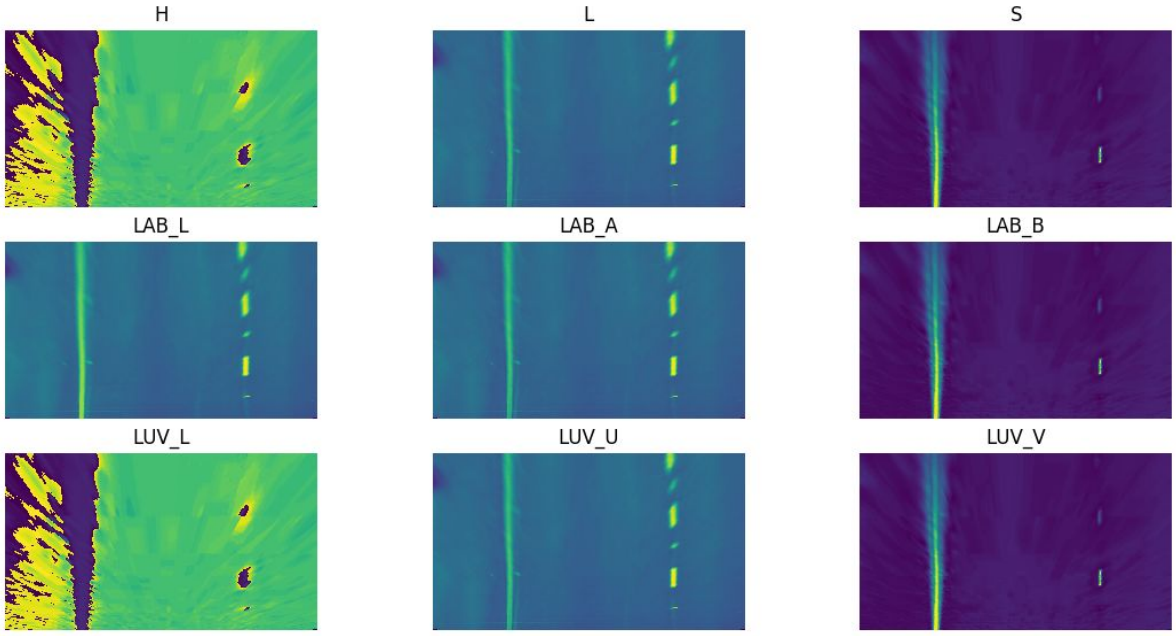
I had to calibrate the quadrilateral on the straight road image to get the right trapezium to do the warping.

The warped images have parallel lines.

The dots in the images show the points where the warping vertices were

3. Image Thresholding:

Color Space exploration:
I tried different color spaces to get better results.
I tried: HLS, LAB, LUV
I found that A in LAB, U in LUV and L in H:S were rather shadow resistent



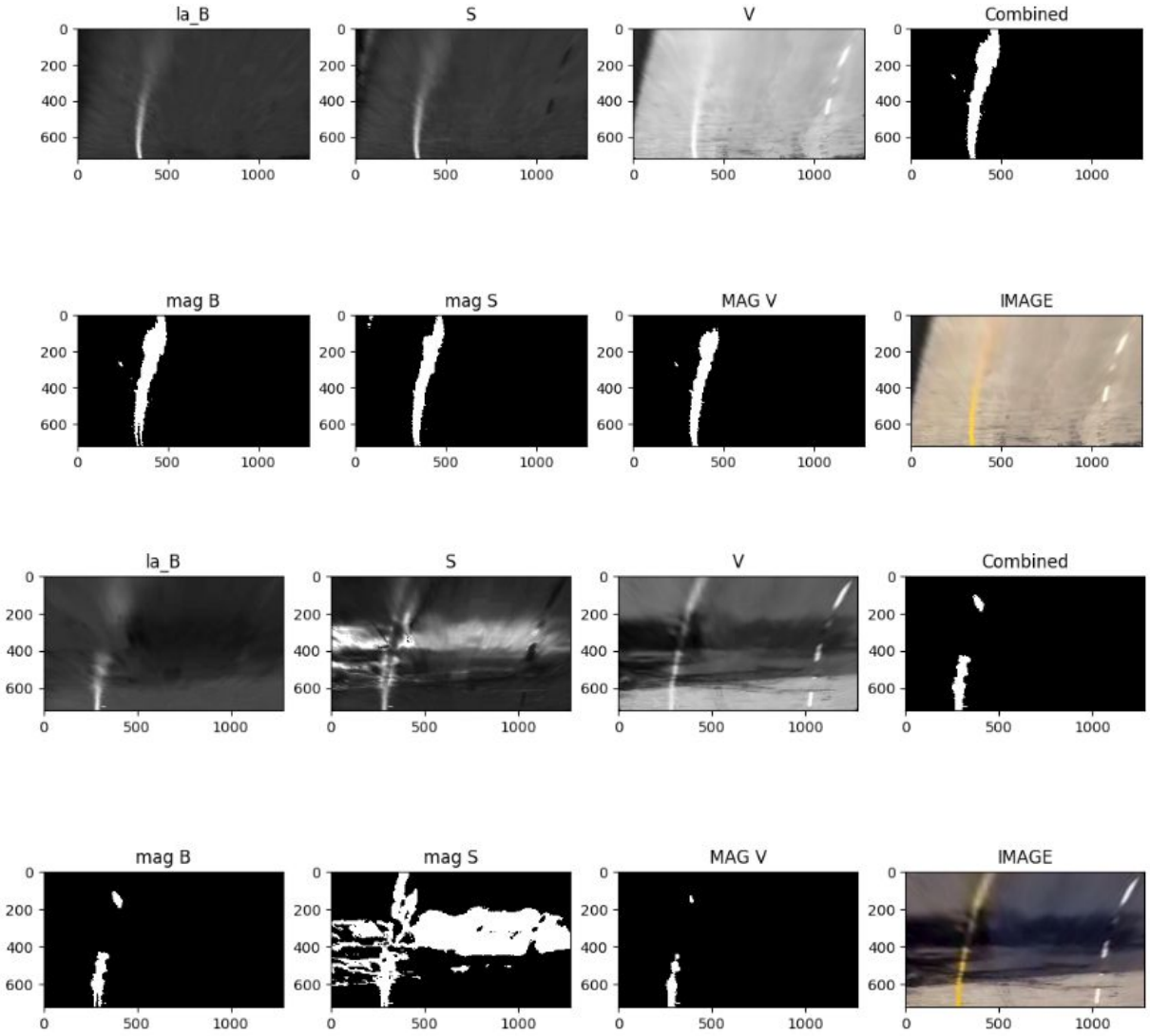
A certain amount of experimenting was required in order to make the thresholding work. To experiment with the different possibilities, I plotted them together which brought out the differences and picked the combination which gave the best results.

I decided to use magnitude as a way for separating the lane lines as the color spaces were able to demarcate them very well.

I extracted the left and right lanes separately and combined them later.

Left Lane:

The main challenge was when the yellow left lane was very faint.



I used a combination of magnitudes on:

1. B channel of LAB
2. S channel of HSV
3. V of LUV

This gave good results even for the very faint cases

The combination was :

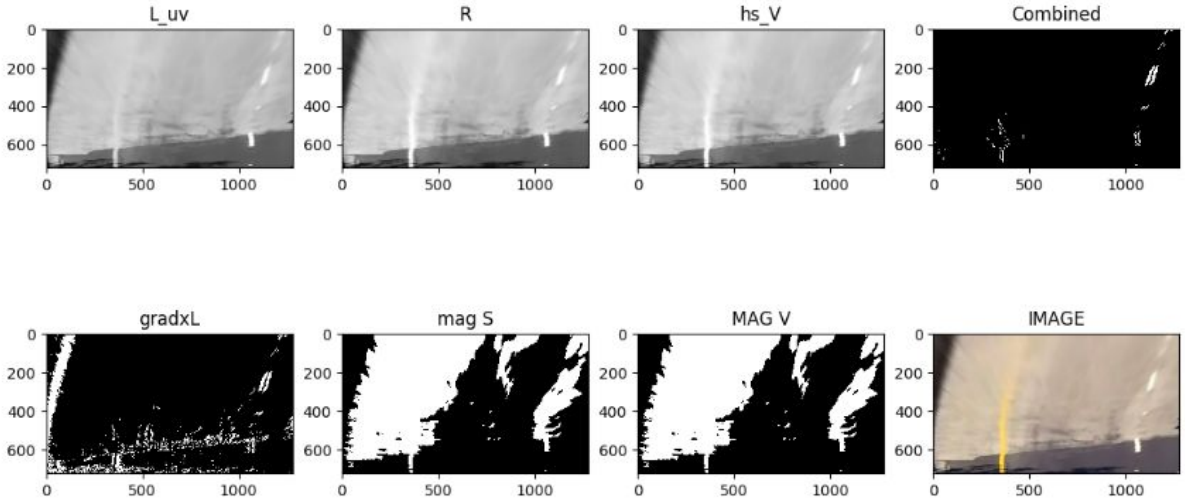
```
(mag_v==1)|(mag_b==1)) | ((mag_b==1) & (mag_s==1 )
```

Right Lane:

For this, I used a combination of:

1. X Gradient on L of LUV
2. Magnitude threshold on L of LUV

3. Magnitude threshold on R of RGB
4. Magnitude threshold on V of HSV



The x gradient on L of LUV was necessary to make it resilient to lighting changes.
The other 3 were able to handle most of the other cases.

```
combined_left[((mag_v==1)|(mag_b==1)) | ((mag_b==1) & (mag_s==1 ))] = 1
```

Both Lanes:
The right and left lanes were then combined using OR

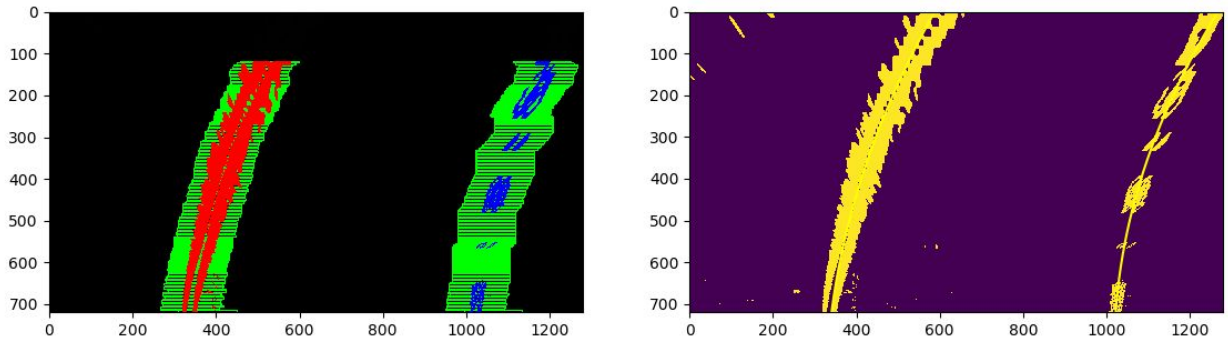
Once, I had experimented with the threshold values and the combinations, I put the final selection in the function: `threshold` in `image_pipeline.py`

The resulting images were mostly great.

All the images can be seen in `threshold_images/`

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 within 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`

Discussions:

Currently, the pipeline will struggle if:

1. The elevation changes

The warping and unwarping is dependent on the assumption that the elevation will not change. This can lead to issues on slopy or bumpy areas.

2. When an object obstructs the lane:

Like in the challenge video, a biker crosses the lane line.
This could potentially obstruct or deform the pipeline's idea of the lanes.

3. False Positives:

The pipeline, being image dependent can be easily be fooled intentionally or unintentionally. For example, if the back of a car has the image of a road, the pipeline might get confused.

4. Deformed Lanes:

Another possibility is that, if the lane might reflect in another car's glossy paint and misinform the idea of the lanes.

Improvements:

- Even better thresholding techniques can definitely benefit the system as most of what the system sees is after the thresholds are applied.
- Augmenting the system with LIDAR like distance data will also reduce the dependence on image like data.