Project 4 Advanced Lane Detection

1. Introduction

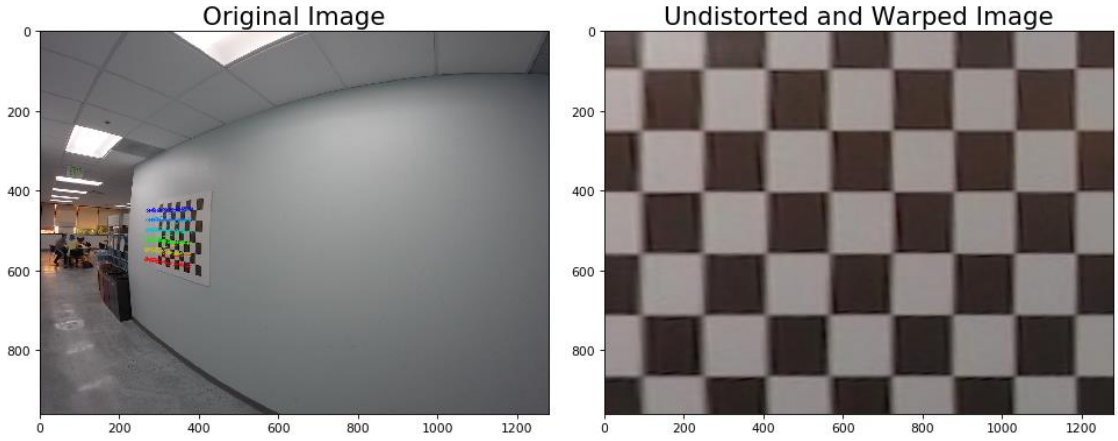
This is an extension to the preojct1 lane detection. First camera calibration is performed to correct image distortion caused by camera lens. Next the front view camera image is transformed into a bird eye view to focus on the driving lane. Then, a various color/gradient filters are applied to make the line marks easier for the line detection pipeline to find the lines. Finically, apply the polynomial fit to the line marks found previously and transfer the annotated image back to its original view perspective.

2. Image Process Pipeline

All the code in this part are in the utilites.py file.

2.1 Camera Calibration

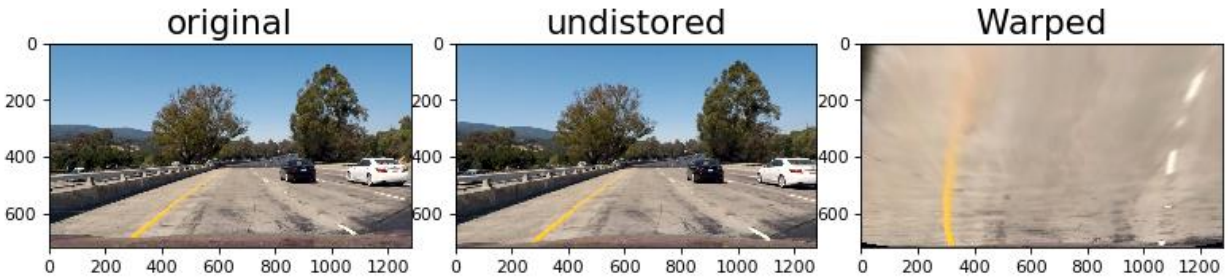
For the camera that used to record the project videos, chessboard images were given to calibrate the camera. After the calibration, the camera(mtx) and distortion coefficient (dist) were saved for image process later.



2.2 Perspective Transfer

Apply perspective transformation to the undistorted images. I found it later that the line detection, poly fitting is sensitive to the perspective transformation. Here is what I used in the end and it works well for both the project and challenge video.

Source	Destination
[210, 720]	[280, 720]
[1250, 720]	[1200, 720]
[580, 460]	[280, 0]
[760, 460]	[1200, 0]



2.3 Image filtering

I wasn't sure which filter was best to find the line marks, so I tried several of them, namely: HLS, HVS, LAB, RGB, x_grad, y_grad, mag_grad, dir_grad. The code is in img_filtering.py. Here is an example of the binary output.

I noticed that the B channel from the LAB is most effective finding the yellow lines. For the white lines, there are several of them: L channel from both HLS and LAB, S channel from HLS. R channel from RBG is kind of works both yellow and white. It's interesting that none of the gradient filter seems working well. I didn't spend too much time digging into it.

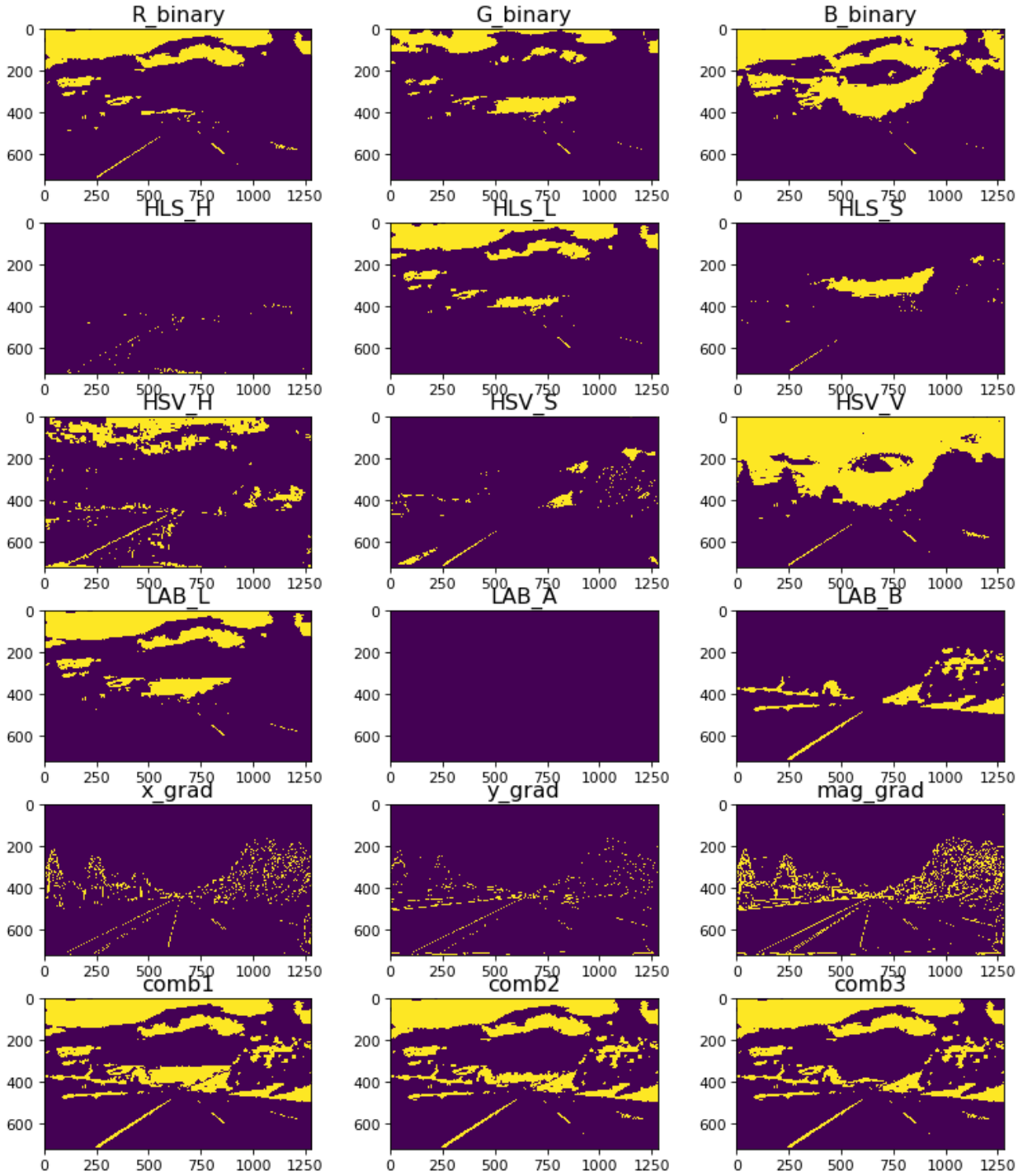
I found two combinations that both work well:

comb1[(LAB_L_binary == 1) | (LAB_B_binary == 1) | (HLS_L_binary == 1)] = 1

comb2[(LAB_B_binary == 1) | (RGB_R_binary == 1)] = 1

comb1 was used for project. Here are the details of each filter:

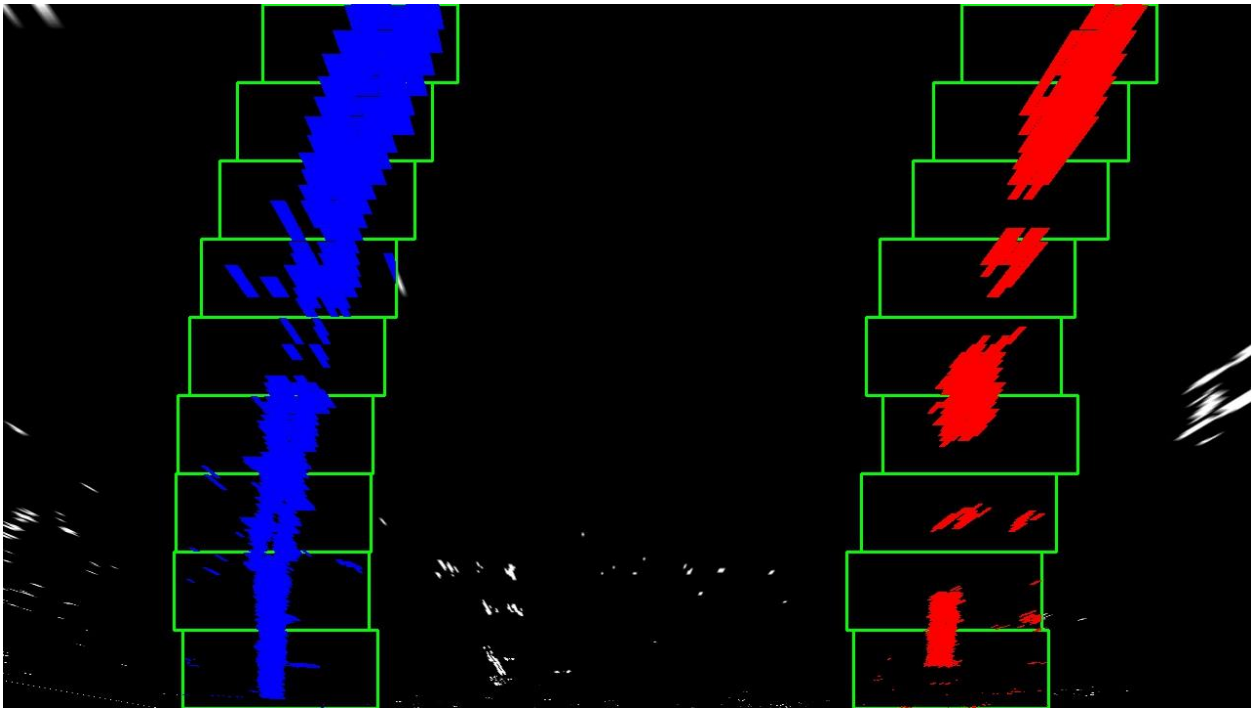
```
HLS_L_binary = color_thresh(HLS_L, thresh = (190, 255)) #  good with white

LAB_L_binary = color_thresh(LAB_L, thresh = (190, 255)) # good with white

LAB_B_binary = color_thresh(LAB_B, thresh = (140, 255)) # best yellow

RGB_R_binary = color_thresh(RGB_R, thresh = (170, 255))  # both
```



3. Line Detection

I just used two method that given in the project, namely: window sliding, and convolution. I didn't see much difference between the two. The window sliding was used for the project. Here is an example of the code finding the lines.

The code is in the file: line_detection.py



3.1 Curvature Calculation

Just use the method introduced in the material. I used the slope's sign to tell if its left bending or right bending curve.

3.2 Vehicle Offset

Just take the average of the left and right line position at the bottom of the image, and compare it to the middle point of the image (1280/2)

3.3 The Final Image

4. Video Process Pipeline

The pipeline is simple, first apply the camera correction to the raw image, and then image process (perspective transfer, filtering). Next line detection. As recommended, I added a Line class to store the previous lines. There are two usage of the stored lines: one for line stabilization, the other is in case of a failed line detection (a line fitting that doesn't make sense or no line was found). Lastly, a polygon was filled based on the detected lines then transfer back to the original camera view.

5. Final Thought

Finding the lines robustly based on only image processing is challenging, (did I tell you the code failed on the third video?), especially with different light condition, bad lane marks, etc. I wish I could find the best filter combination that can detect the lines in most conditions, does it exist? I wonder if deep neural nets perform this task, but that's for later.