Self-Driving Car Engineer Nanodegree Program

Advanced Lane Finding

Second Attempt

John Reilly

Due date June 2018

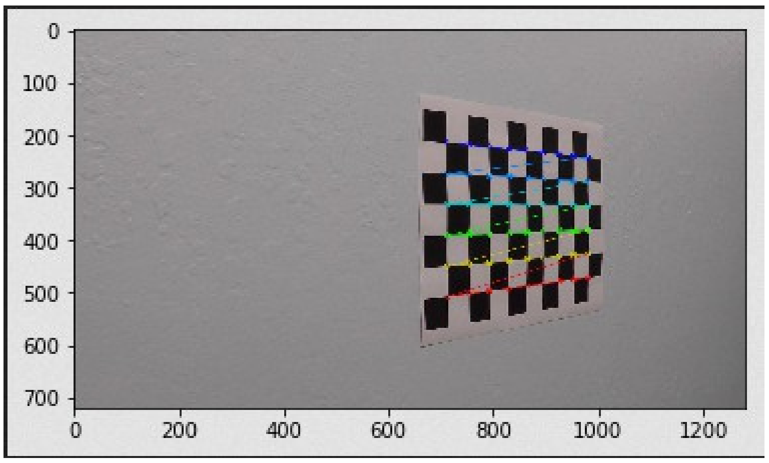
Table of Contents:

Camera Calibration

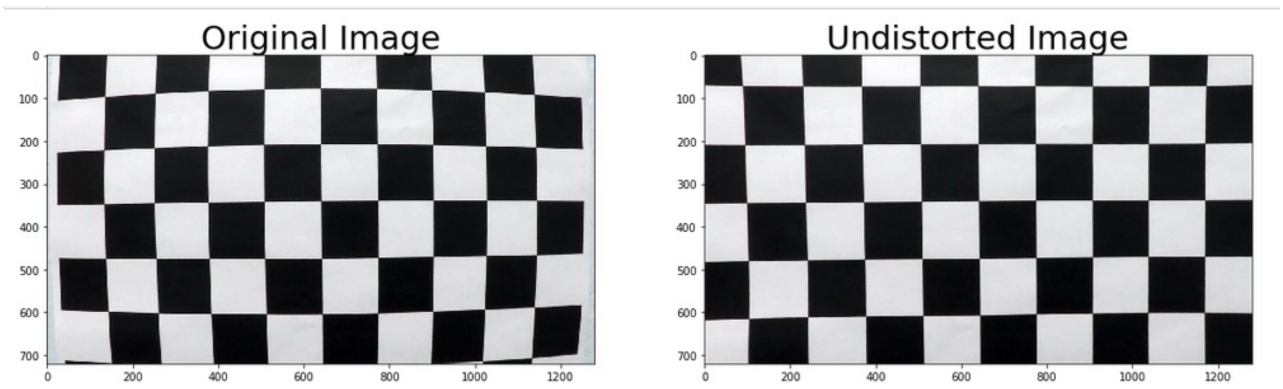
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step in Jupyter Notebook Project4, cells 1 , 2 and 3. and is based on Lesson 15 “Finding Corners” and “Calibrating your Camera”

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:
Output from Cell 3:

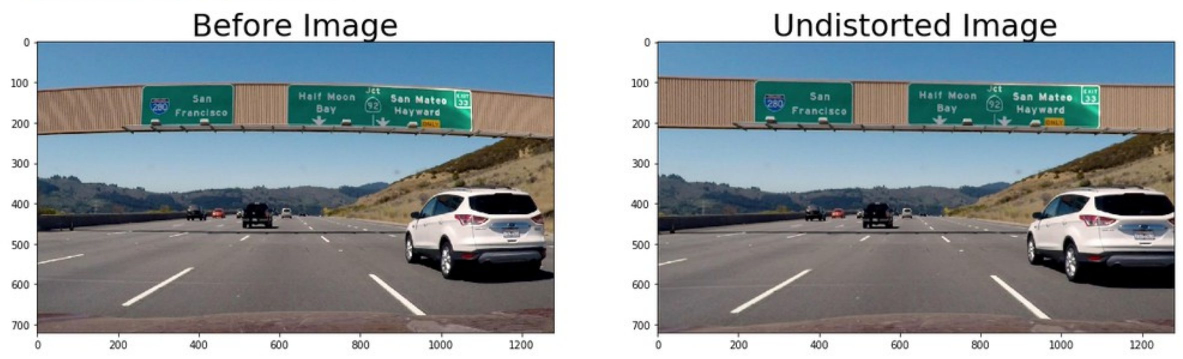


Pipeline (single images)

1. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one. This is the output from cell 4 and you can clearly see the effects of undistortion on the horizontal lines of the overhead road sign. The undistortion effect is the same one as the chessboard from earlier cells.

```
Out[4]: <matplotlib.text.Text at 0x899f080>
```



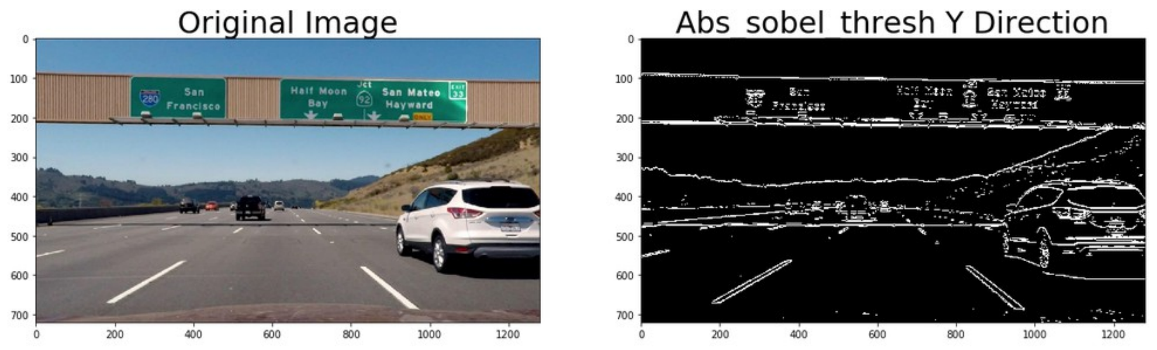
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image. In cell 5 based on Lesson 15, video 2 there is a function "def abs_sobel_thresh" which calculates Sobel values for either the X or Y directions. The following cells show experiments with using Sobel as a way of finding lines and you can see various levels tried in the code cells and the output images show my preferred threshold levels.

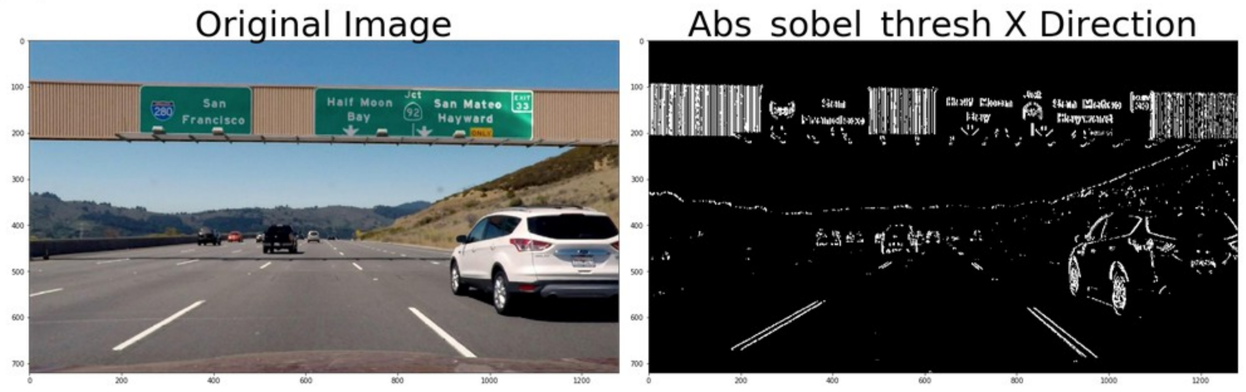
PLEASE NOTE: After Cell 5 the numbering of the cells jumps to 21. I believe this is because I added these cells later but it may cause some confusion. At the time of writing I don't know how to fix this so please just be aware in the notebook the cell after 5 is 21.

Output from Cell 21: Sobel Y direction threshold Min 30 Max 255

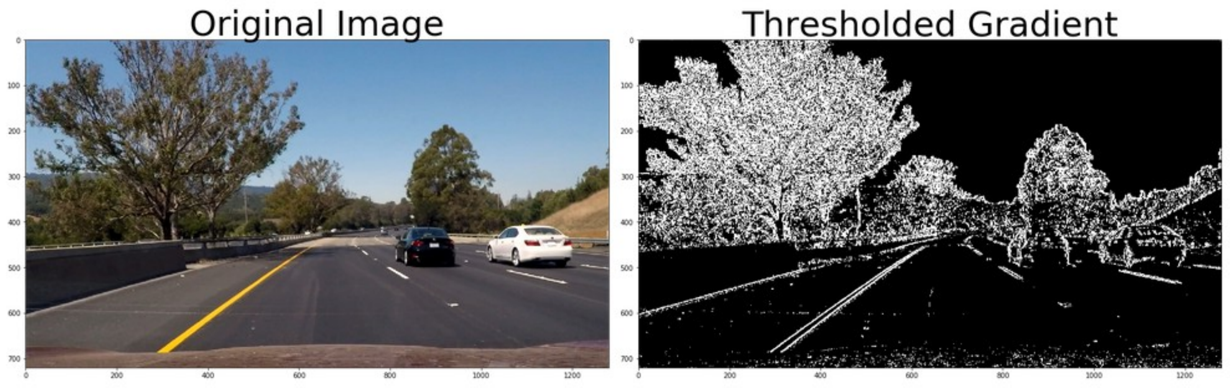
```
Out[21]: <matplotlib.text.Text at 0xb8a8da0>
```



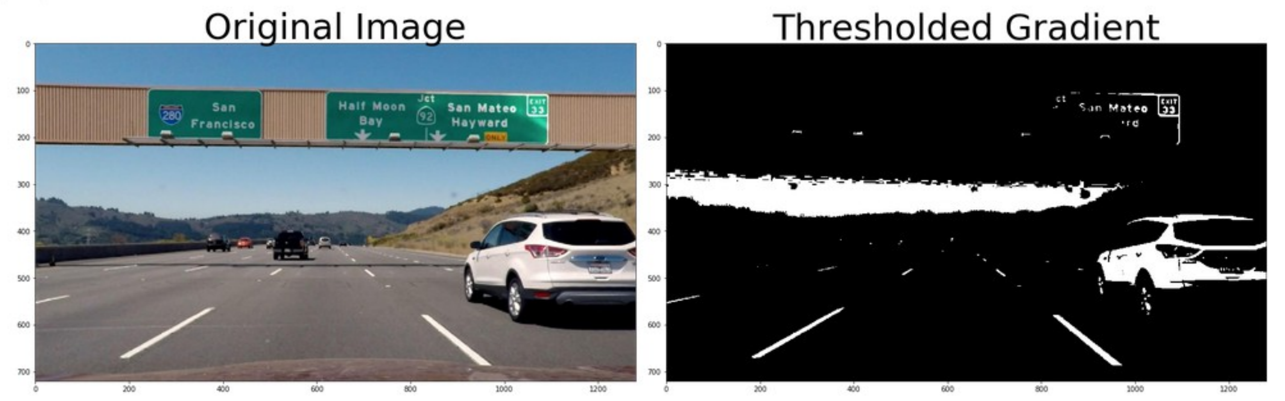
Output from Cell 23: Sobel X direction threshold Min 30 Max 255



Output from Cell 8 (*the next cell in sequence*) Sobel X direction threshold Min 30 Max 255 test_image_6

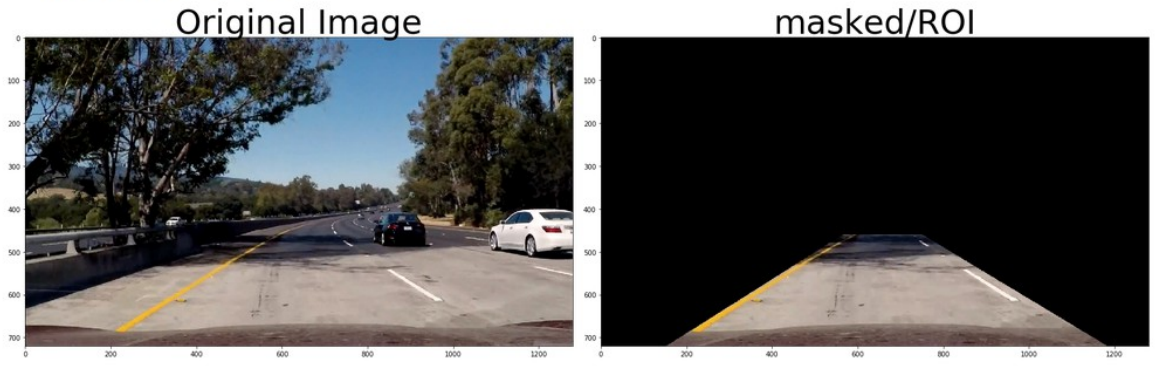


Output from Cell 27 and Cell 30 (*the next cell in sequence*) GrayScale threshold Min 190 Max 255:

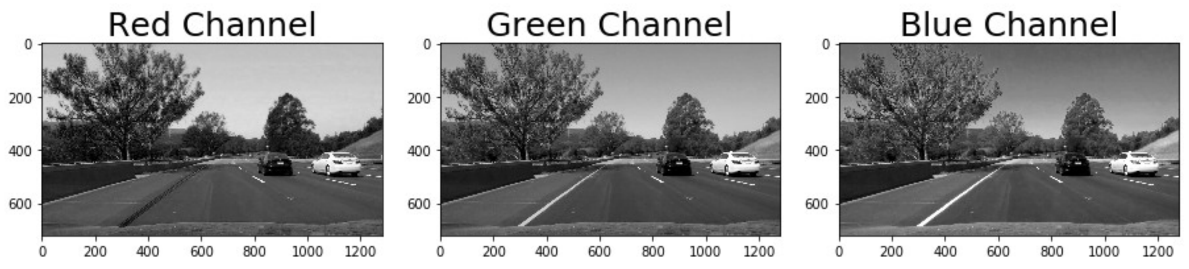


Output from Cell 29: This is where I work with the Region of Interest. I added here in this part of the notebook because in defining the edge of the Region of Interest it was important to get all of the road lanes but reduce the unuseful information and it took some experimenting to get it right. There are some references in the code comments here also, from Stack overflow and other students work in github.

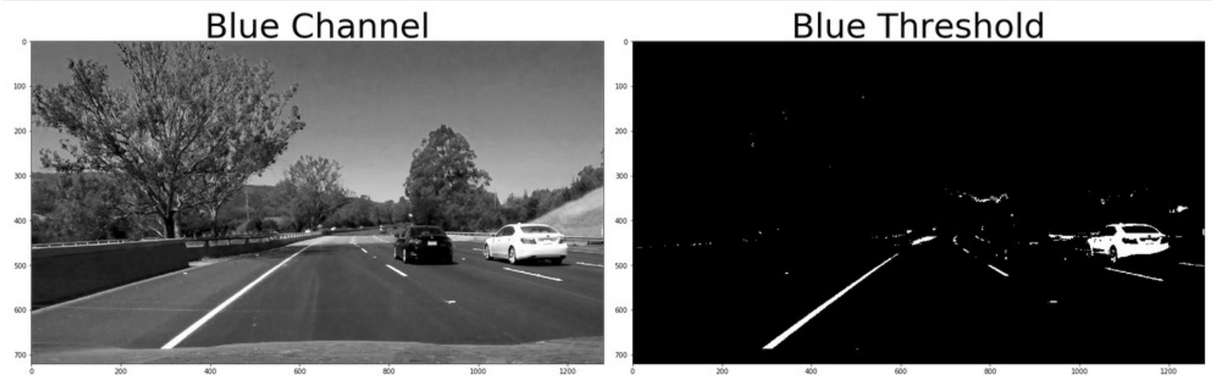
```
Out[29]: <matplotlib.text.Text at 0xbe35e80>
```



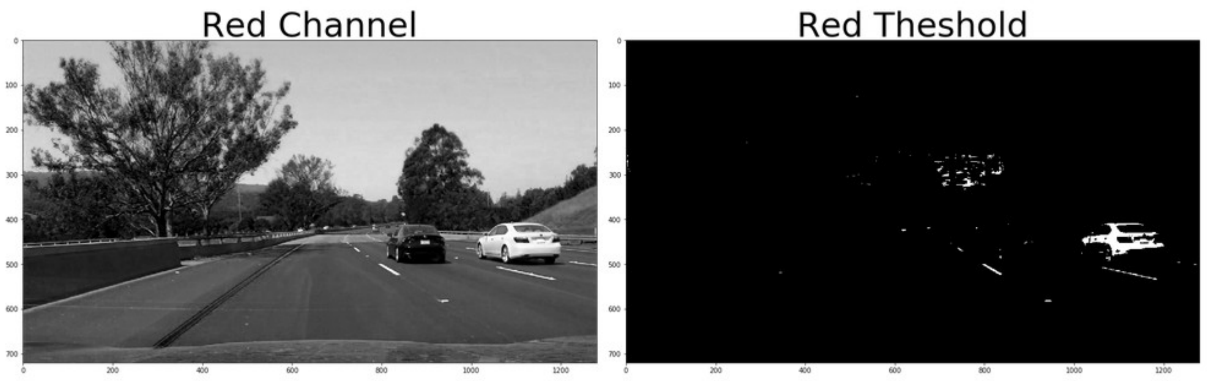
Output from Cell 31: This shows experiments in separating the RGB channels of an image. Note the usefulness of the Blue channel in finding lane lines



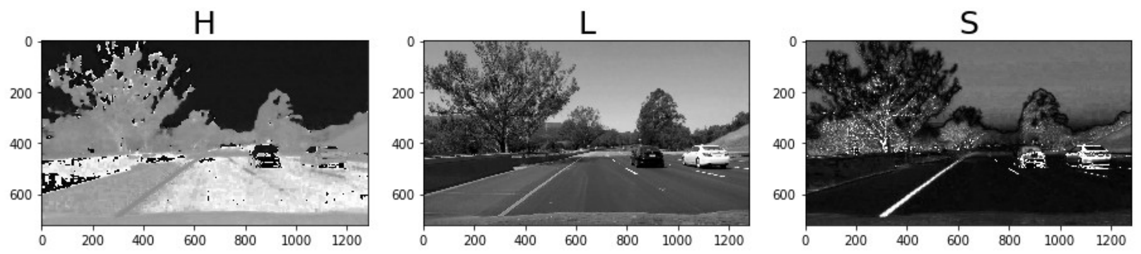
Output from Cell 33: Experiments with Blue Channel and threshold values.



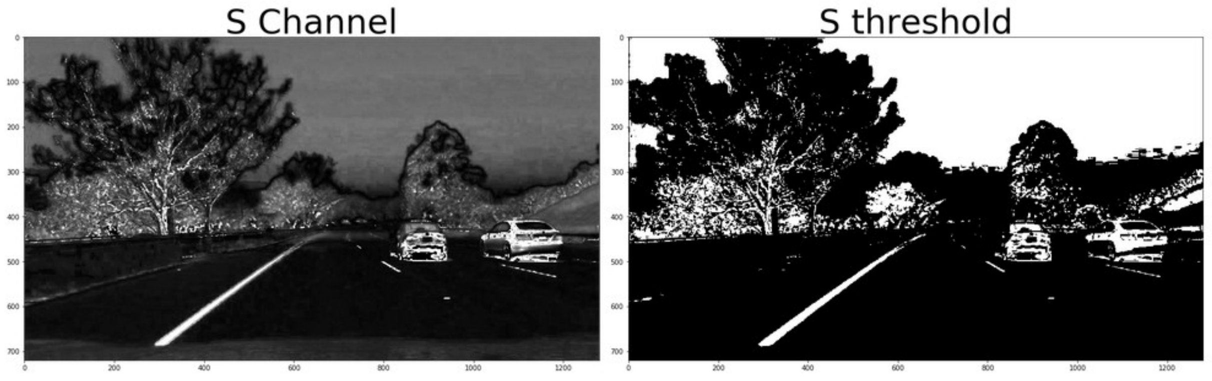
Output from Cell 34: Experiments with Red channel and threshold values



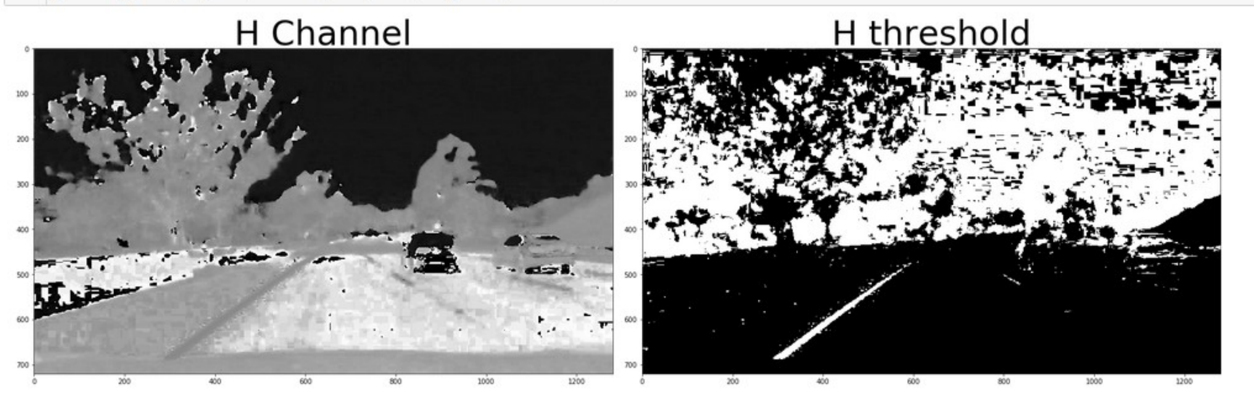
Output from Cell 35: Experiments separating the image in HLS components:



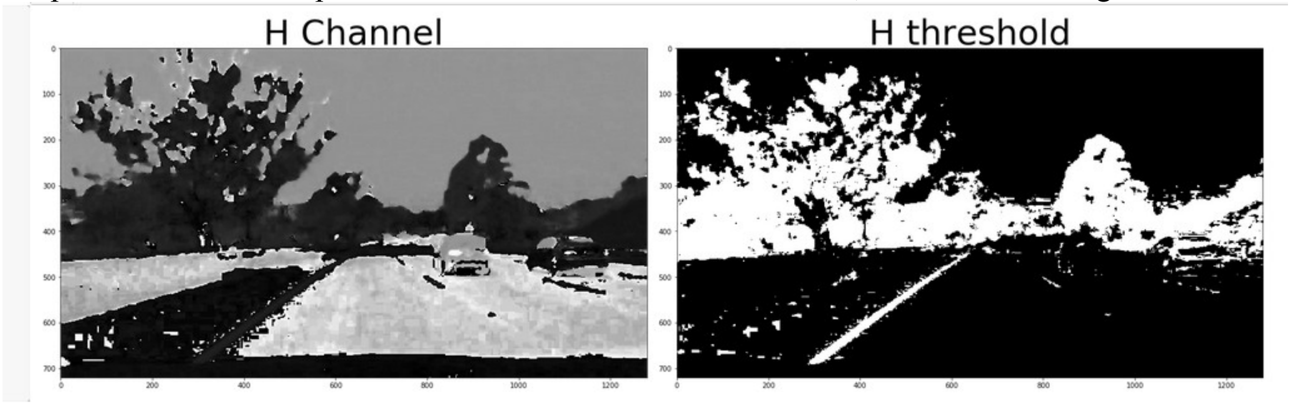
Output from Cell 36: Experiments with S-Channel and Threshold Values



Output from Cell 37: Experiments with H-Channel and Thresholds

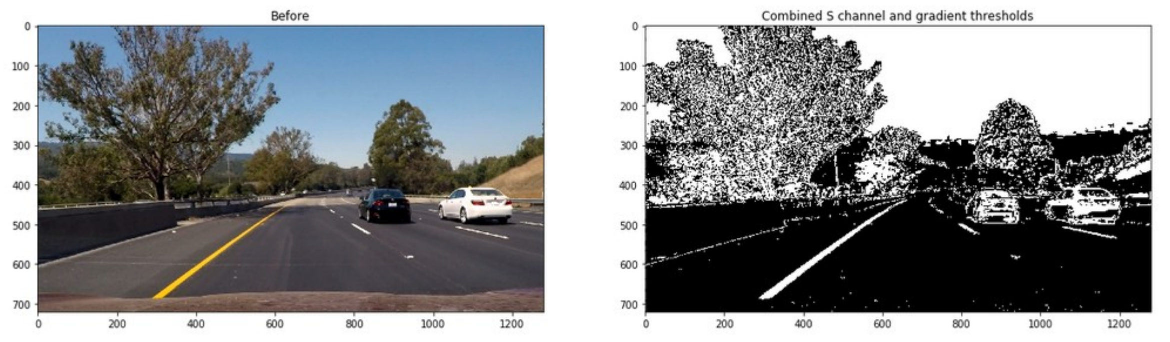


Output from Cell 38: Experiments with H-Channel and Threshold , different Test image



Output from Cell 40: This is the final experiment and is based on Udacity provided code.

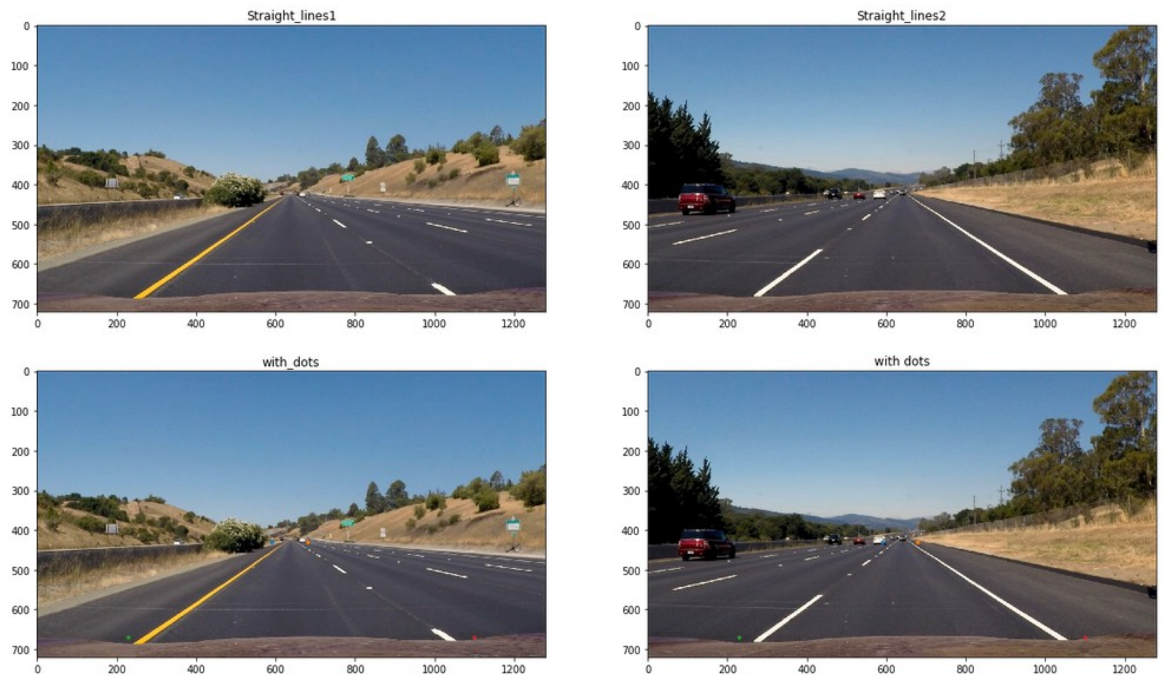
```
Out[40]: <matplotlib.image.AxesImage at 0xe46c978>
```



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

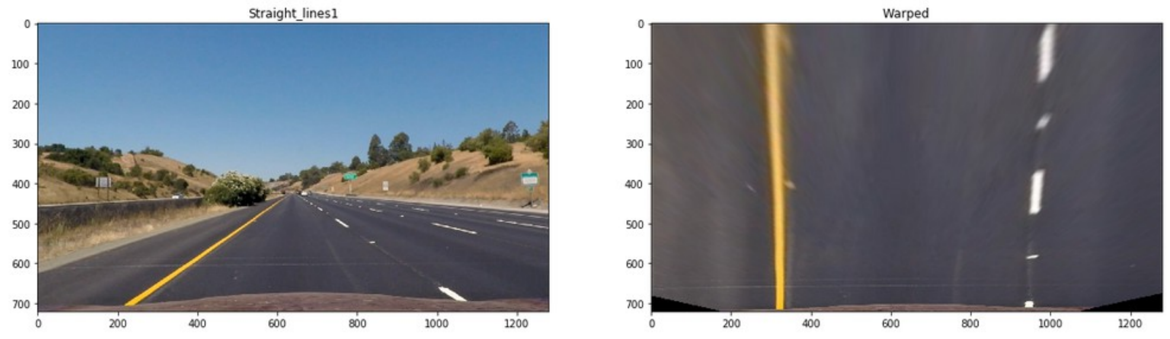
Cell 42 is the beginning of the transformation/warp section. In Cell 42 I was experimenting by putting dots into the image to figure out what dimension to use for the transform. The dots are very hard to see but you can see in the code what I was doing. I did a lot of experimentation for this as it was very difficult to get the warp effect right. In the end I took the values from the sample write up and adjusted them slightly. Output from cell 42 below.

```
Out[42]: [<matplotlib.lines.Line2D at 0xc22c5c0>]
```



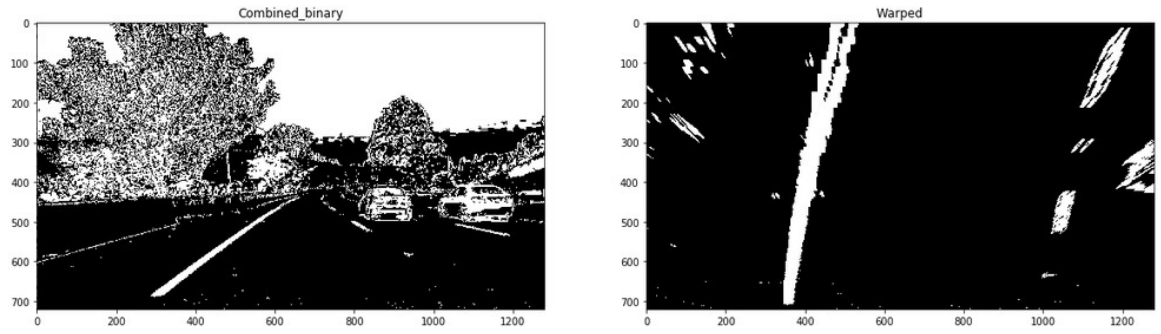
Output from Cell 43: This shows the final warp effect

```
Out[43]: <matplotlib.image.AxesImage at 0xb2bcac8>
```



Output from Cell 44: This shows the warp effect applied to a combined binary and this is then used in the final pipleine

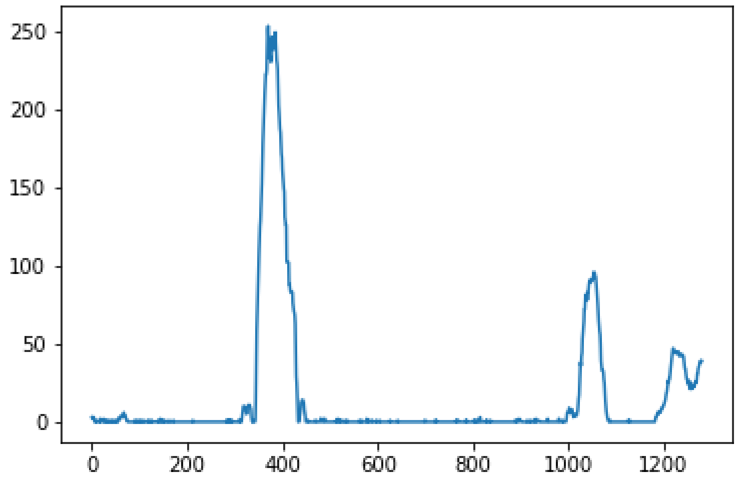
```
Out[44]: <matplotlib.image.AxesImage at 0xc61dd68>
```



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

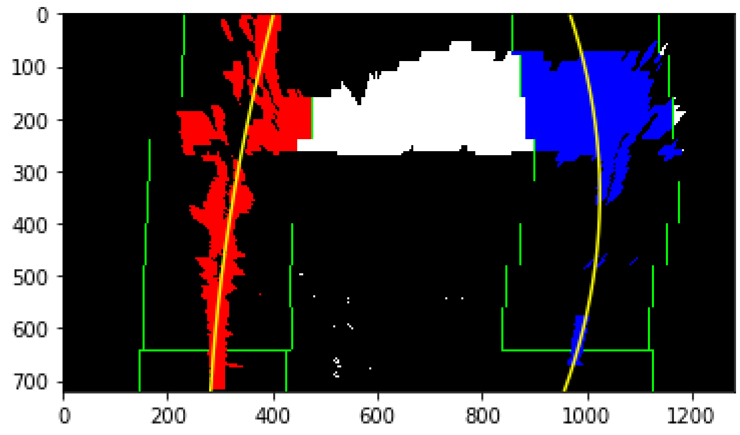
Cell 46 is the beginning of the lane finding section. A histogram technique is used to find the likely starting points of the lane lines and the output of Cell 46 demonstrates the technique applied to a test image that is a combined binary and warped same as the pout put from the previous cell.

```
Out[46]: [<matplotlib.lines.Line2D at 0xe820ef0>]
```



Cell 47 is the most important Cell. It is where the Sliding Window technique is used to find the lane lines. Additions to the code supplied by Udacity includes lines 78 to 86 where data for the calculation of the road position is extracted. Basically when the function finds the first pair of valid left and right sliding window centre points then that is taken for the measurement of the road position against the centre of the image. Lines 90 to 95 were for testing as I was trying to see how often no valid left or right value was found, it was many in practice.

The output below is from the next cell after 47 which shows a sliding window.

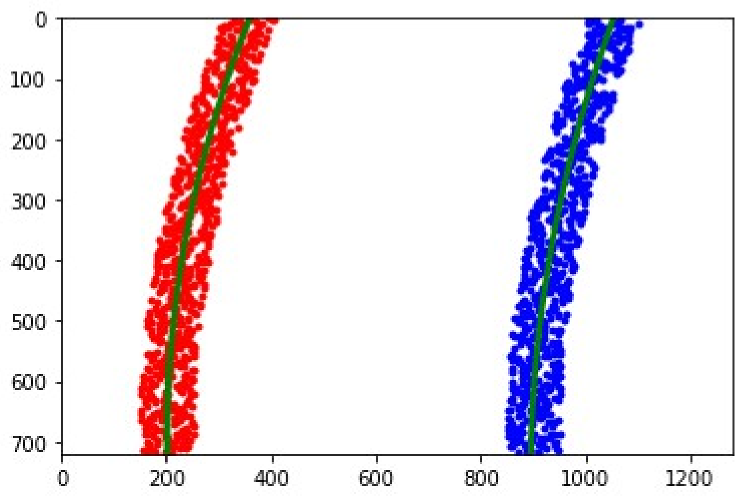


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

Radius of Road Calculation:

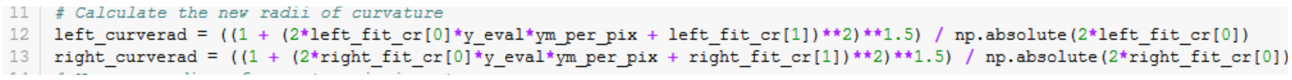
Cells 52, 53 and 54 are functions ***fake_plot()*** , *get_radius_in_pixels()* and *get_radius_in_meters()* are the cells that calculate the radius of curvature. These functions are straight out of Lesson 15 Video 34 with only a couple of adjustments to make them functions instead of code snippets.

Cell 52, creates fake data around the already identified lines , this is to create data to feed into an equation that will calculate a radius of a line from point on that line. Output below



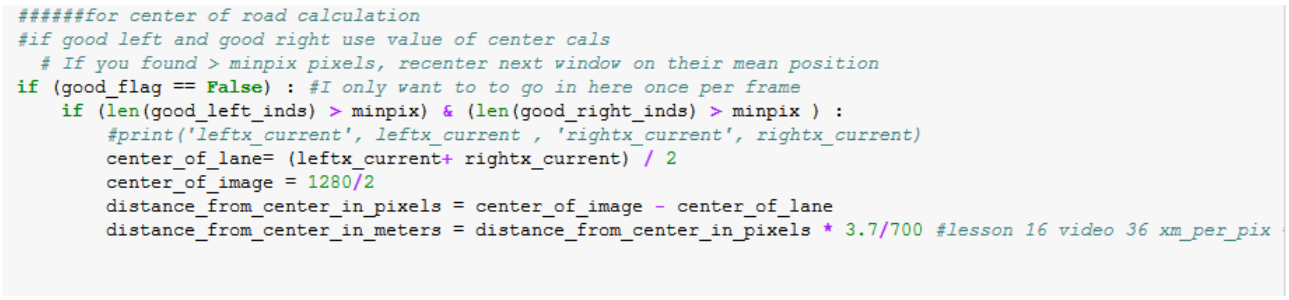
Cell 53: This cell creates an numerical output of the raidu in pixels which is shown in the lesson but I decided not to use it

Cell 54: This cell is similar to 53 except it creates an output in Meters instead of pixels and this is the one used in the pipeline. Worth noting is the output is a float with multiple decimal poitns and this gets rounded to an integer to display on the video. The important lines are 11 and 12 where the calculation is done.


```
11  # Calculate the new radii of curvature
12  left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
13  right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
```

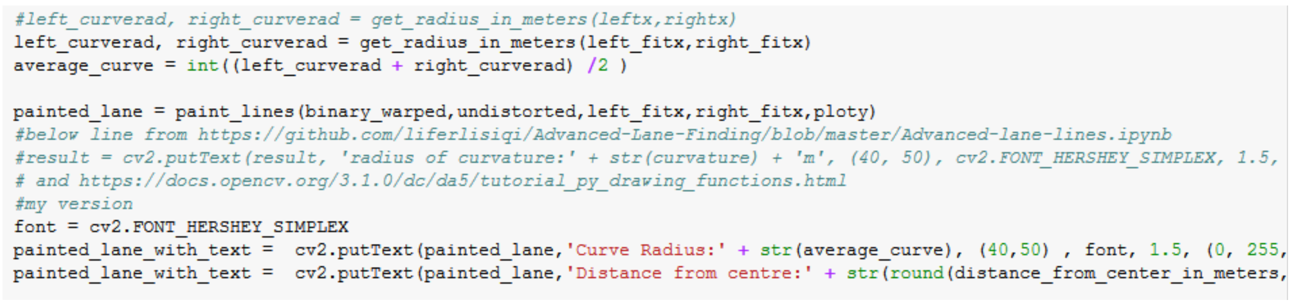
Center of Road Calculations:

There are 2 cells involved in the center of road calculations Cell 47 Sliding Windows and Cell 62 PipeLine4 . The lane measurements are taken from the first valid pair of good left and good right lane points. So as the sliding window method goes throughout the image as it finds a good position on the left and a good one on the right it takes those values as the left and right lane points. These values are added then halved and that result taken from the center point of the image to generate a distance from center point value in pixels which is then converted to meters. A boolen "good_flag" is used to ensure that once a successful value is found the process is not repeated for every other set of sliding windows in a given frame. As shown below



```
######for center of road calculation
#if good left and good right use value of center cals
  # If you found > minpix pixels, recenter next window on their mean position
if (good_flag == False) : #I only want to to go in here once per frame
    if (len(good_left_inds) > minpix) & (len(good_right_inds) > minpix ) :
        #print('leftx_current', leftx_current , 'rightx_current', rightx_current)
        center_of_lane= (leftx_current+ rightx_current) / 2
        center_of_image = 1280/2
        distance_from_center_in_pixels = center_of_image - center_of_lane
        distance_from_center_in_meters = distance_from_center_in_pixels * 3.7/700 #lesson 16 video 36 xm_per_pix
```

The rest of the work is done in Cell 62 where the values are averaged and added to the text in the final image. Shown below.

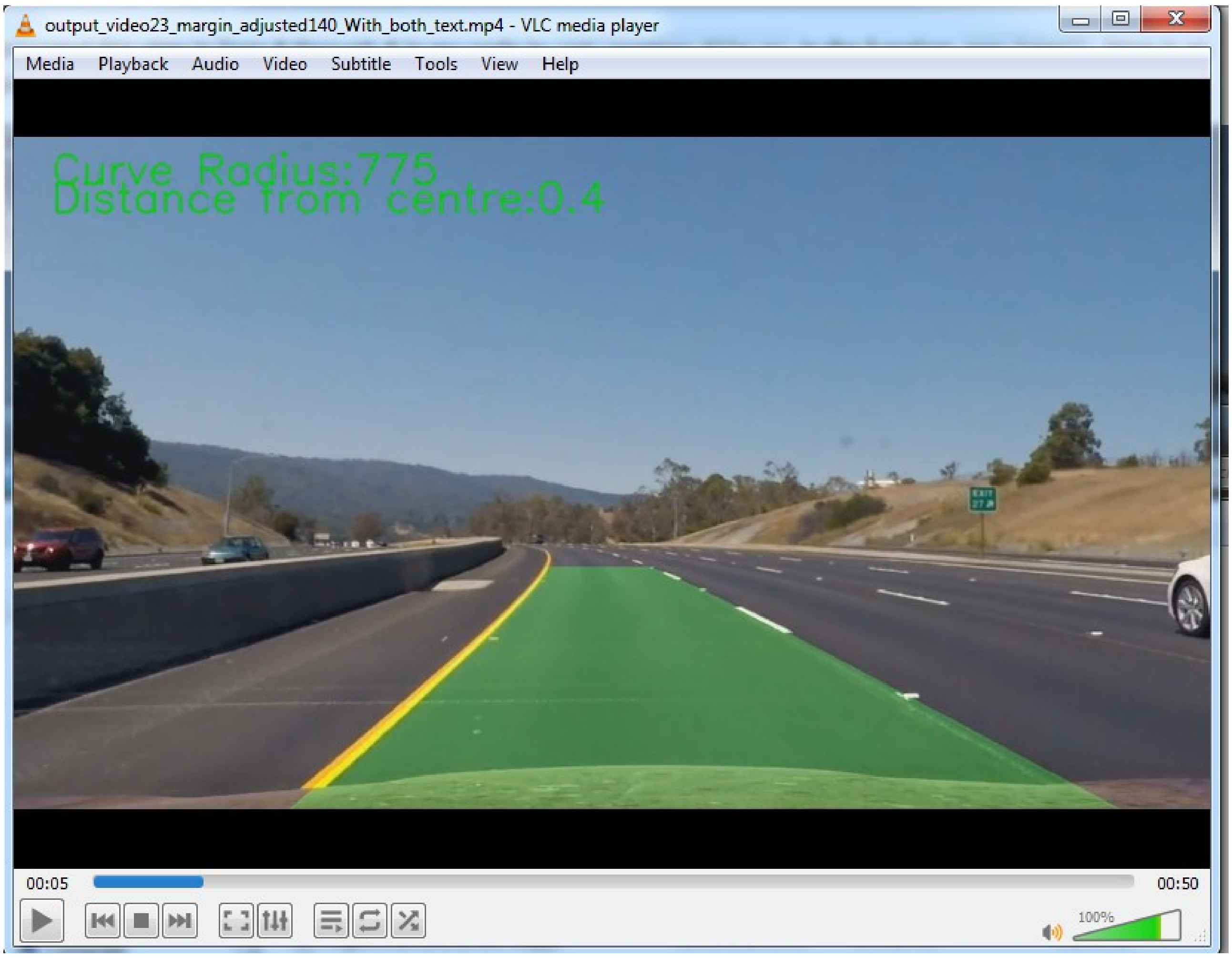


```
#left_curverad, right_curverad = get_radius_in_meters(leftx,rightx)
left_curverad, right_curverad = get_radius_in_meters(left_fitx,right_fitx)
average_curve = int((left_curverad + right_curverad) /2 )

painted_lane = paint_lines(binary_warped,undistorted,left_fitx,right_fitx,ploty)
#below line from https://github.com/liferlisiqi/Advanced-Lane-Finding/blob/master/Advanced-lane-lines.ipynb
#result = cv2.putText(result, 'radius of curvature:' + str(curvature) + 'm', (40, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.5,
# and https://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html
#my version
font = cv2.FONT_HERSHEY_SIMPLEX
painted_lane_with_text =  cv2.putText(painted_lane,'Curve Radius:' + str(average_curve), (40,50) , font, 1.5, (0, 255,
painted_lane_with_text =  cv2.putText(painted_lane,'Distance from centre:' + str(round(distance_from_center_in_meters,
```

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

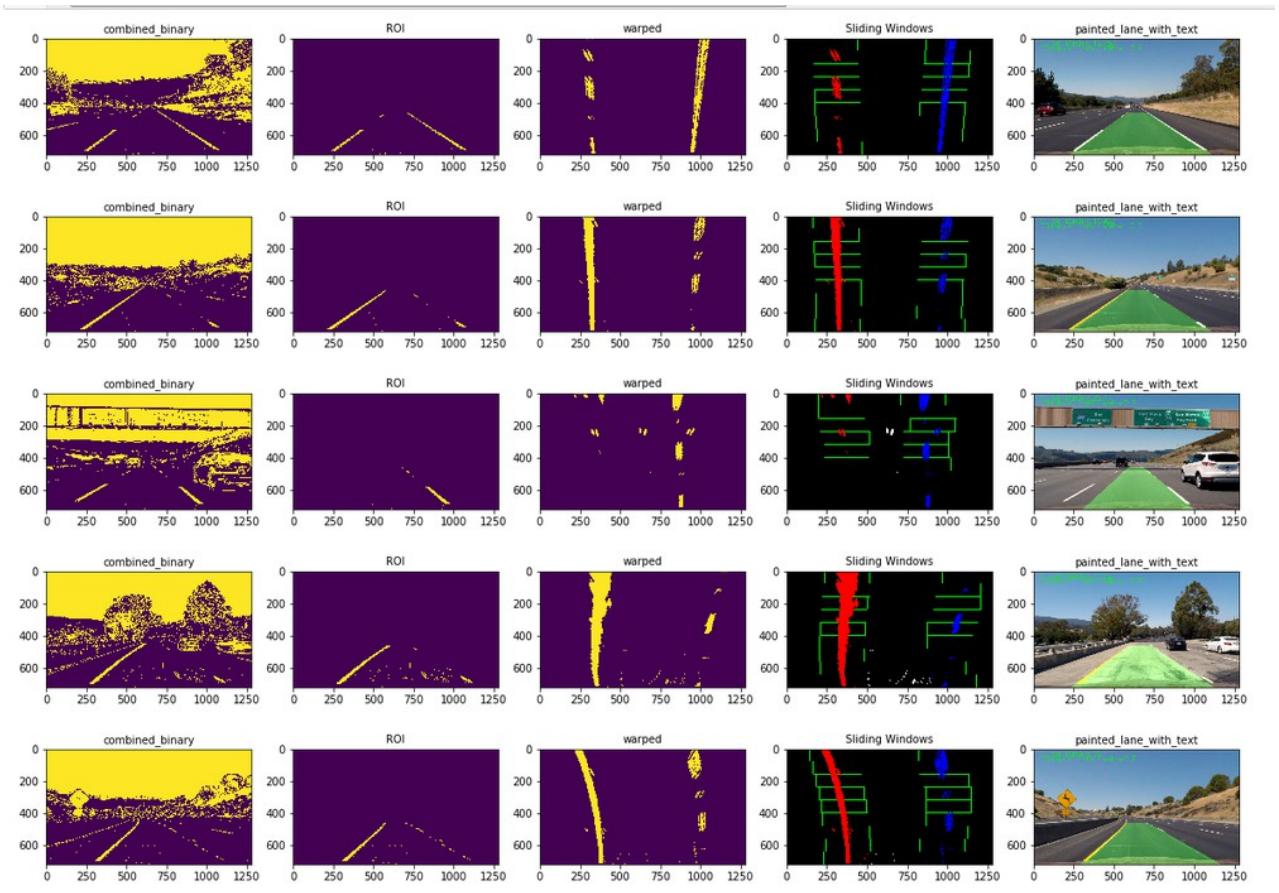
Below is a still image taken from the output video to show it working. The lane lines are plotted back and "Unwarped" onto the image in Cell 40 in a function called *paint_lines*



Also worth showing is some extra functionality I included in the pipeline. I created plots for each frame or test image showing all the stages of the pipeline. In this way threshold values and Regions Of Interest could be adjusted, margins in the sliding windows were also adjusted.

This was very useful in troubleshooting. It works well on individual images as shown below. The code is disabled by a Boolean to make sure it does not run on the videos because even though it technically can work it uses too much memory and will probably freeze your machine so it runs for test images only

If you run cell 62 pipeline 4 you will see the below output.



Pipeline (video)

1. The final video is called final_video.mp4, it is in the ZIP file......to tell you the truth I don't know if it is good enough....

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

I could write a lot here.... attempting to be brief:

As a general point the project was difficult and Udacity provided many code snippets to assist the students. I included references to the code in comments and generally I found as I assembled the code snippets and created functions for the pipeline I would find lots of errors and what was happening was I had created a function but not included enough parameter being sent into and out from the functions and from the errors I would see that some function needed to be sending another one so information and I would follow that and get it done. That was fine, just time consuming.

There was some code offered about an alternative to the sliding window function and also a faster less computationally intensive function than sliding windows to use one some frames once some lane lines were found. I did not implement this as I found the sliding windows needed some adjustments and I was happy with the eventual result so I didn't implement the alternatives but I might as a side project for practice.

I found all the ways of representing images, RGB GrayScale HLS etc quite fascinating and it amazes me that still in this day and age we cannot easily say pick out the yellow lines. I may explore other colour schemes to perhaps ones that deal with yellow better. That is to say can I generate from RGB channels a Yellow channel and use that to find yellow lane lines. Probably but that will be for another day.

Clearly dark areas and areas of shade are still and issue for this pipeline and the pipeline is quite poor on the harder challenge video. Some ideas I had to address that included. When one lane line is good eg solid yellow to mirror that data on the other side to create a lane boundary on the opposite side to make a good final image.

Also there are issues around finding lines in the center of the lane ie a change of tarmac or road surface. Ignoring lines that are too near or too near compared to the previous frame would reduce false “half lanes” from road shadows etc.

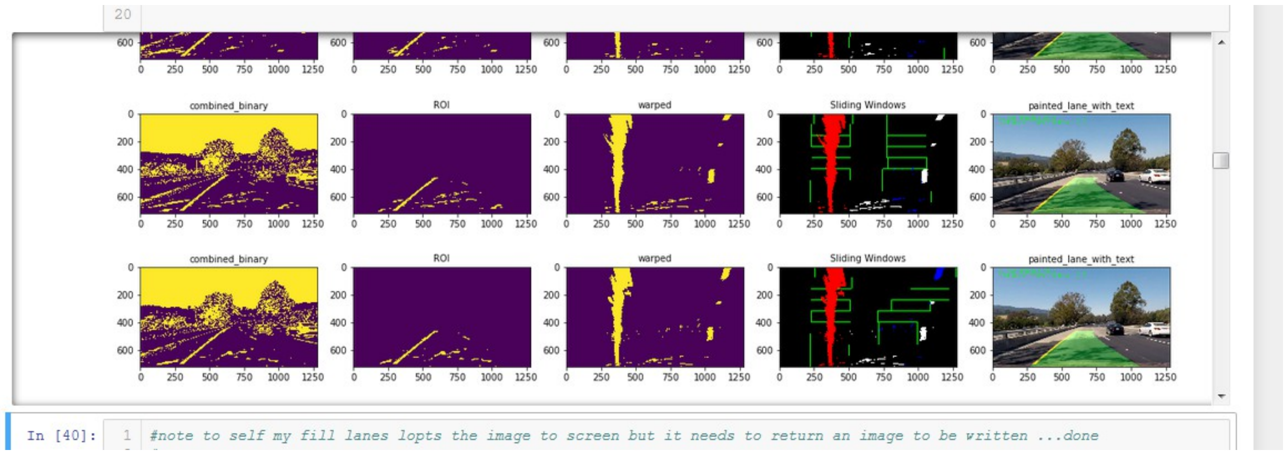
Also how the pipeline will fare with other video footage when other cars etc cross the lane lines is an important consideration not dealt with at all in my implementation.

All that is for follow on projects.

Post Review

Analysis, thresholds improvements:

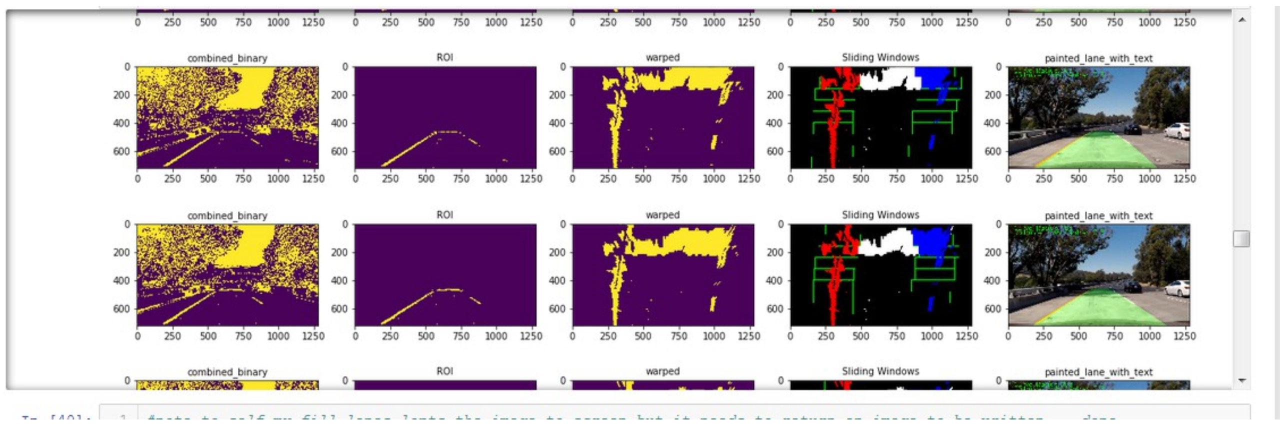
The right lane in this section approx sec 0.41 in video is not drawn correctly. Breaking the frame down into the pipeline images we can see that the lines are shown in the combined binary and ROI and sliding windows. However the sliding windows are not placed over the lines on the right correctly causing the Polyline to be badly drawn. Other similar frames were drawn correctly.

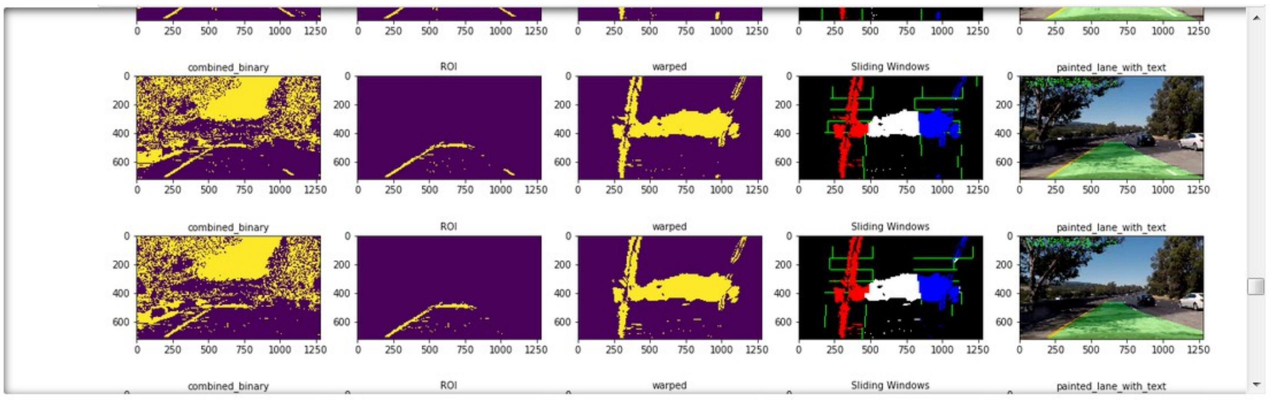


```
In [40]:  #note to self my fill lanes lopts the image to screen but it needs to return an image to be written ...done
```

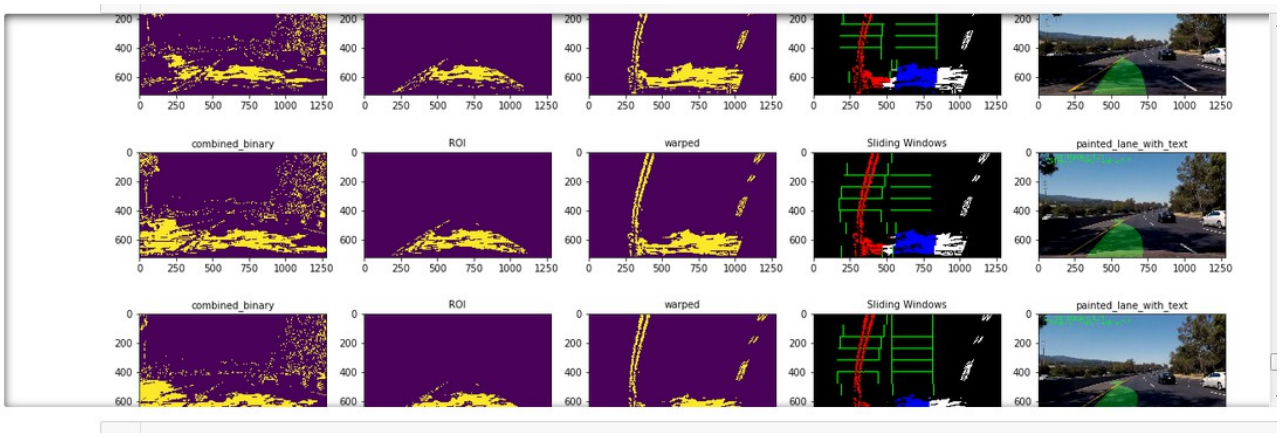
Given that the lines were detected in the sliding windows section there seems to be no need to adjust sections before the sliding window. Also note worthy is the left lane is detected and correctly drawn.

The below frames from approx 0.40 second show an area of pixels in the Warped and Sliding windows frames that are throwing off the lane positioning. It maybe possible to adjust the pipeline so such pixels are not put in or it may be useful to ignore them

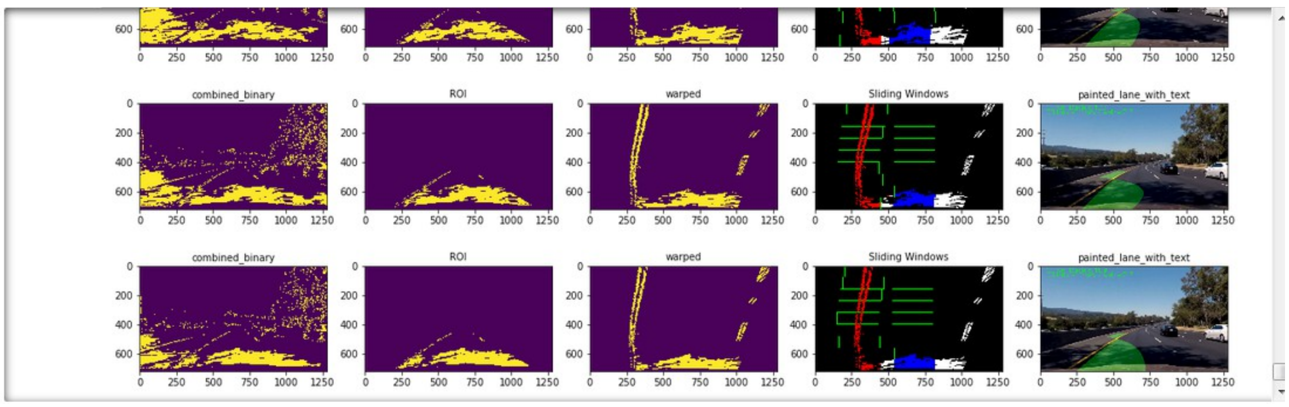




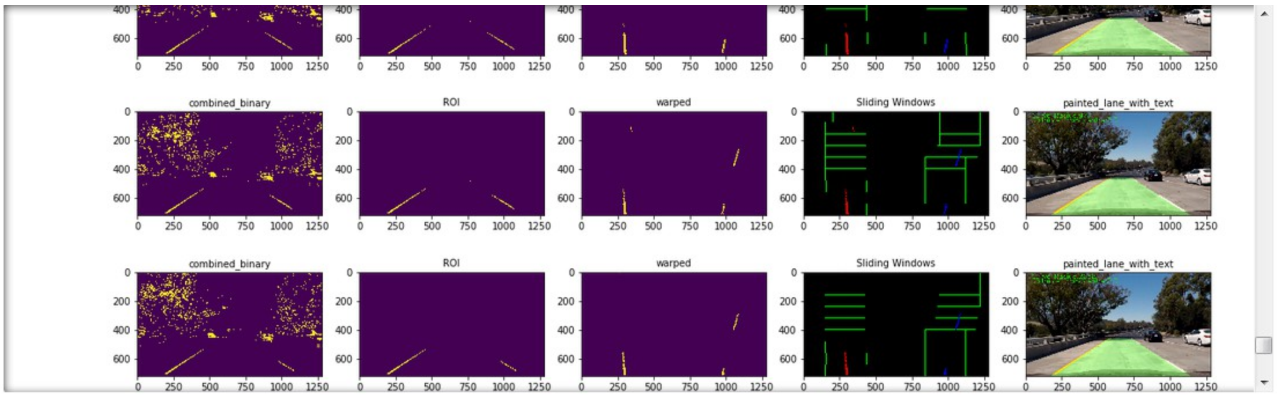
Adjusting the threshold values for Sobel X direction and S color gave some improvement in the number of non lane line pixels detected but and the lines were visible in the sliding windows but not detected see below



The last row below is a good example that I would expect a minimum width approach to work well with

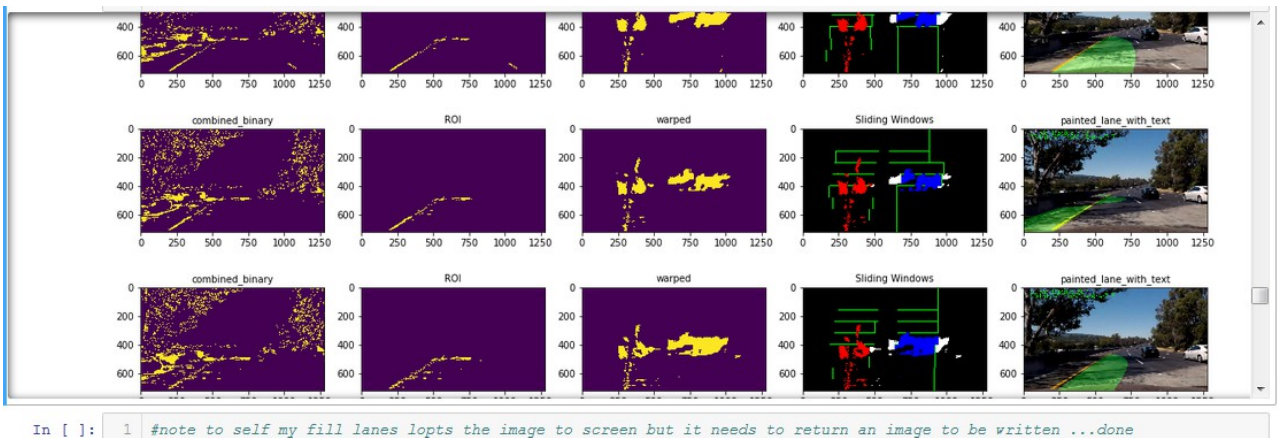


An experiment to see what happens if I removed the Sobel function worked really well. My thinking was the changes in colour were registering in the Sobel but the colours were just shaded on the road not white lines or yellow lines but shadow on the tarmac. These did not seem to be picked up using S threshold so an experiment to use only S threshold seems better see below



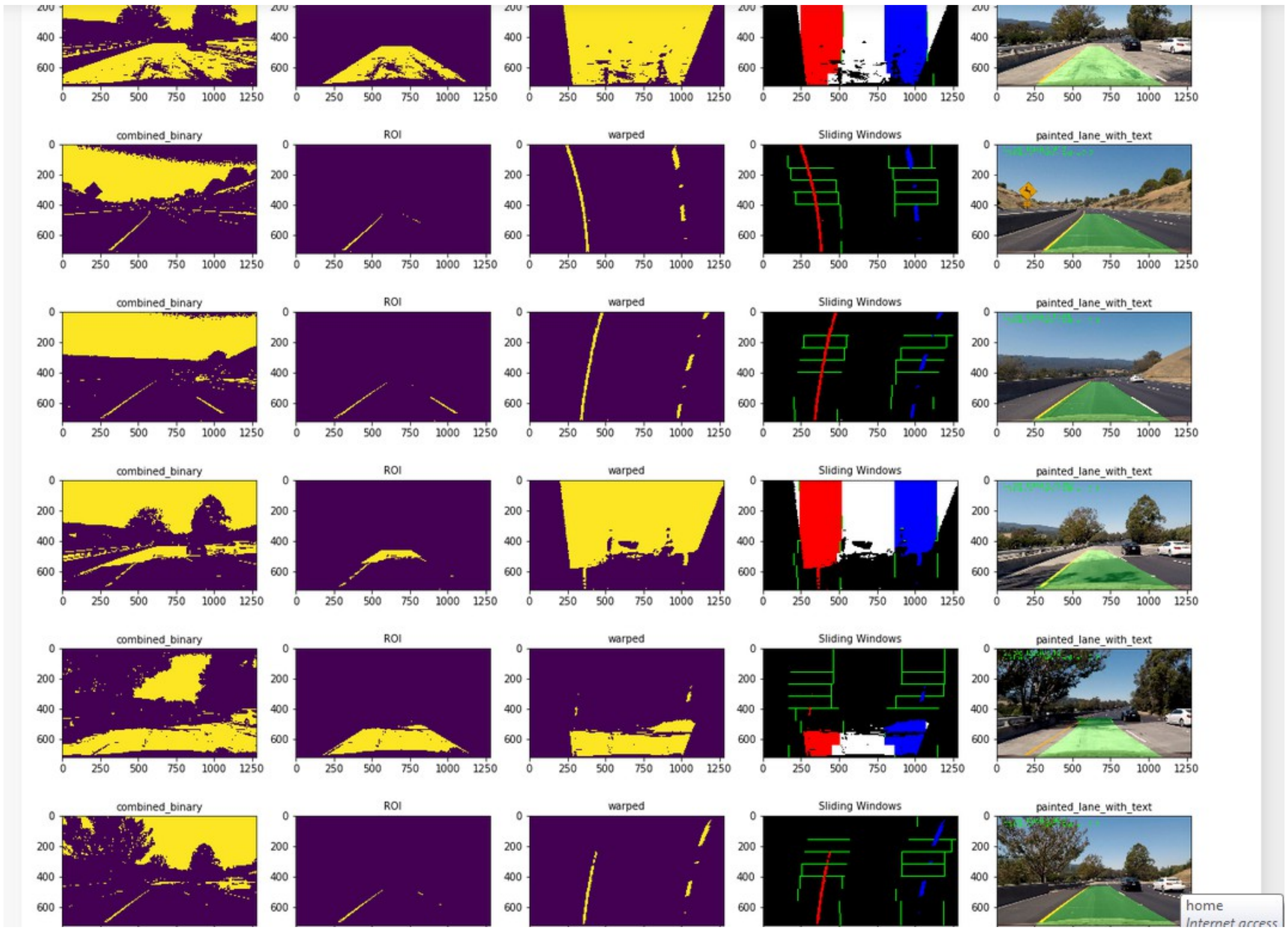
Using no Sobel at all with just S-Channel worked on testimages buit crashed on the video with some frames not detecting any lines. I decided to included a little Sobel to help prevent this.

Using the following setting things went fairly well S threshold thresh = (200, 255), Soble 200, 255

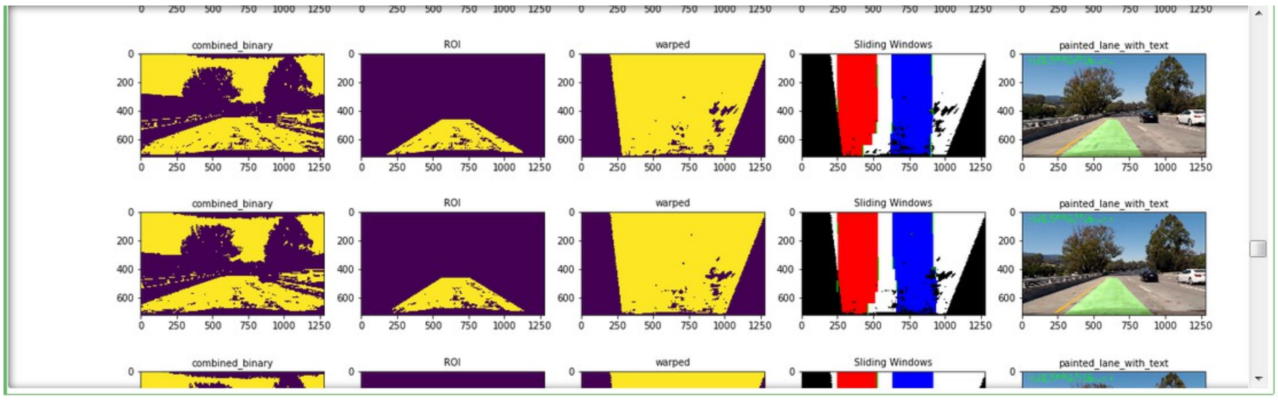


```
In [ ]: #note to self my fill lanes lopts the image to screen but it needs to return an image to be written ...done
```

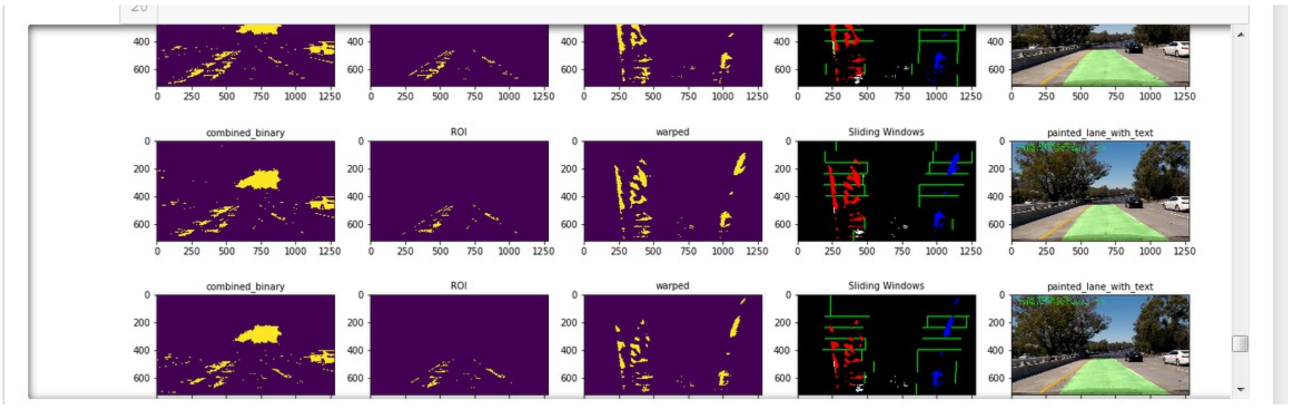
Using just Grayscale and S-channel threshold no Sobel I got this interesting...



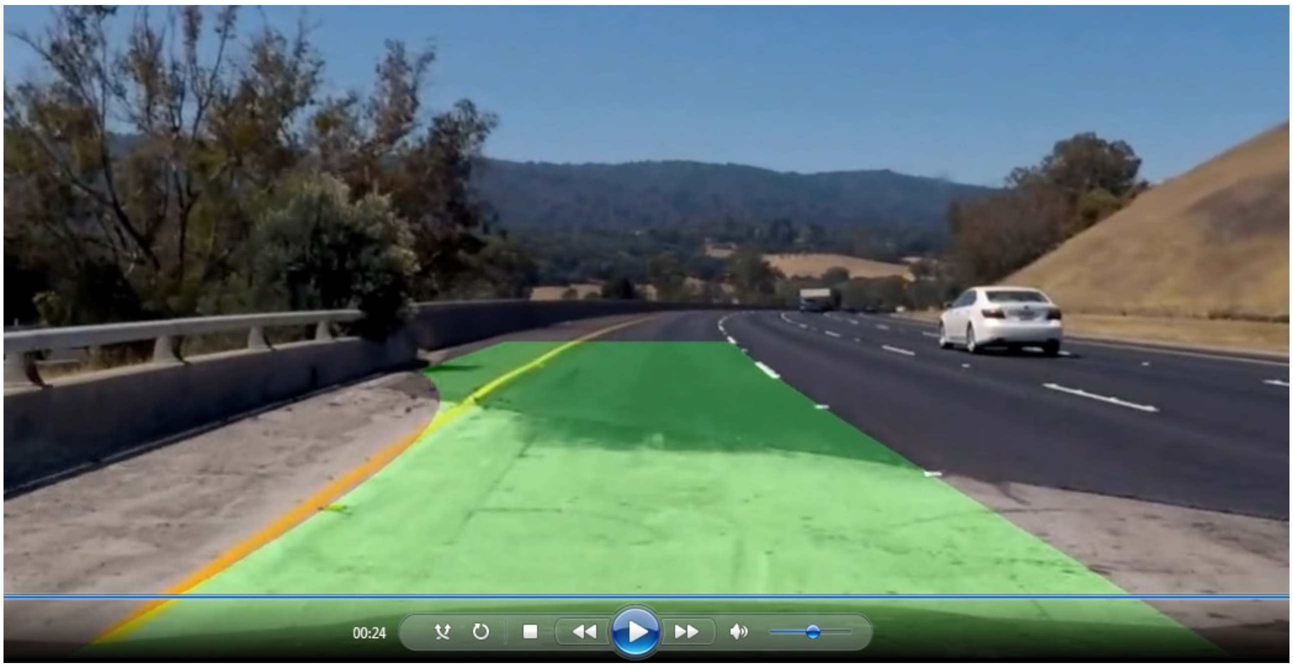
Using Grayscale 150 , 255 at and S-Channel 170,255 at give....



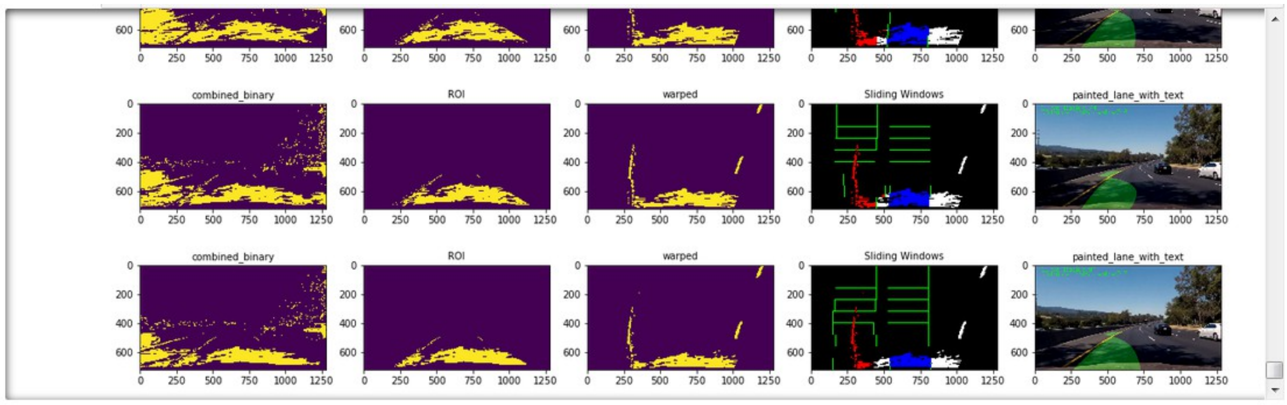
Not bad with Graysacle and S-channel



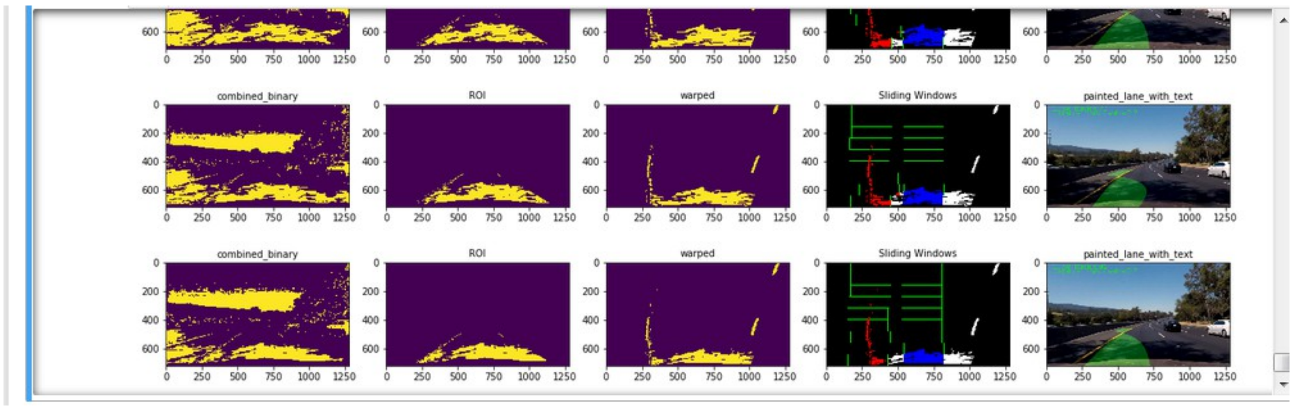
This is just before the program crashes note it is yellow line but in the shade and error indicates it got no line immediately after this...need better dark yellow detection.



This is with Gray scale and S channel best so far but around 42 second this. Note lines are detected but spurious pixels throws off sliding windows



Including the R channel improved things a little some white lane line detected but not covered by sliding window a minimum lane width might fix that



Conclusions about Thresholds:

The previous submitted version used Sobel with a lower threshold and S-Channel. Currently I am using Sobel with a hight threshold and S-Channel (for Yellow) and GrayScale (for white)

This can be seen in Cell 22 funtion create_binary2 line 67 where I combine 3 elements

combined_binary[(gray_binary == 1) |(s_binary == 1)| (R_binary ==1)] = 1

This current strategy is giving slightly better dark image performance and yellow colour detection.

POST REVIEW:

Discarding poor fits:

New approach 4 July comparing curves to last one if more than x% out use last known good one. The code is Cell 40 , Line 64 in function ***image_pipeline6***

if (left_curverad > (last_good_left_curverad*1.5)) or (left_curverad < (last_good_left_curverad*0.5)) or (left_curverad < 500):

In the above line I am making decision about which lines to discard. There is a corresponding line for right curves.

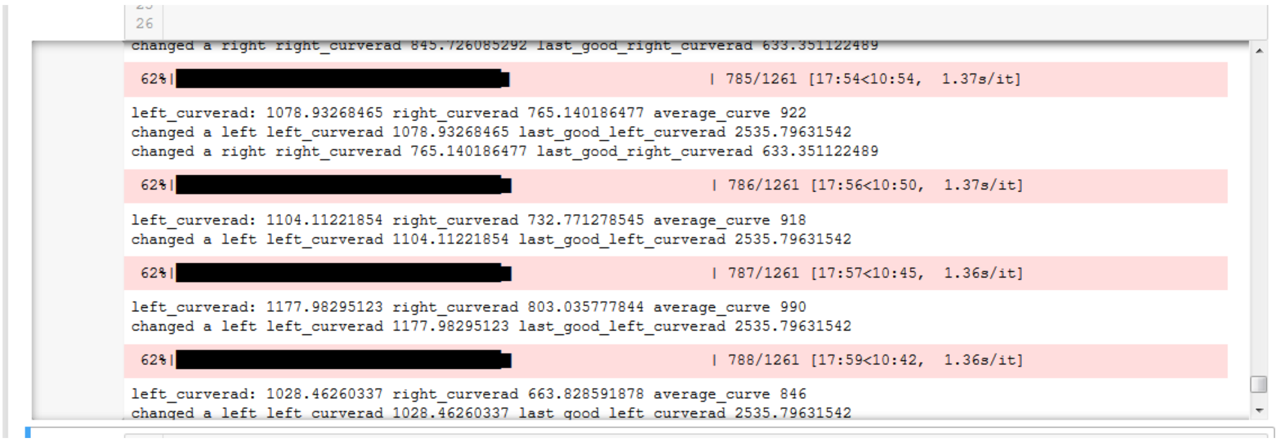
There are 3 criteria for discarding a left or right curve.

1) if the curve radius is more than 50% margin greater than the last known good curve
2) If the curve radius is less than 50% margin of the last known good curve
3) If the curve radius is below 500M from observing the read outs this seems to be a good cut off point

Margins of 10,20,30,40,50% were experimented with and 1000M and 500M were all experimented with.

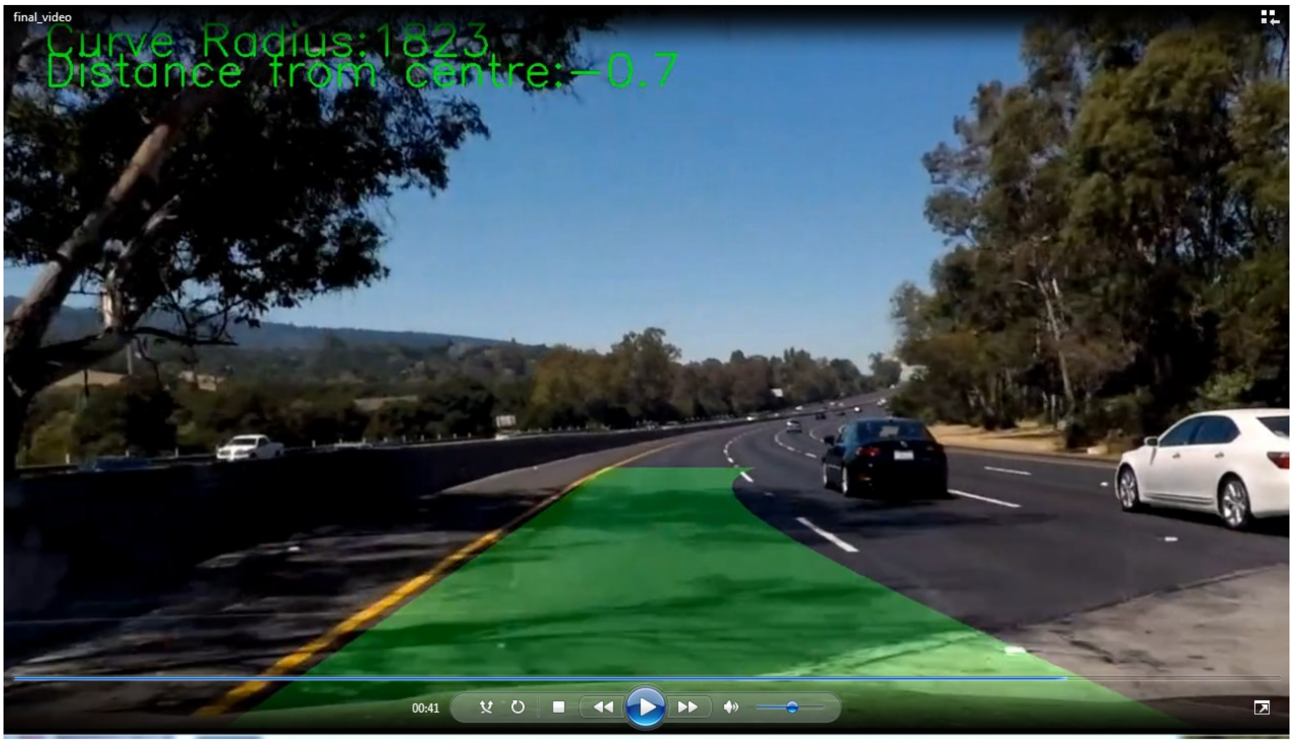
50% margin for previous good lane lines and 500M minimum radius seemed to give the best results.

Below is an output show the video being processed with the curve radius being outputted and the ones being changes noted.

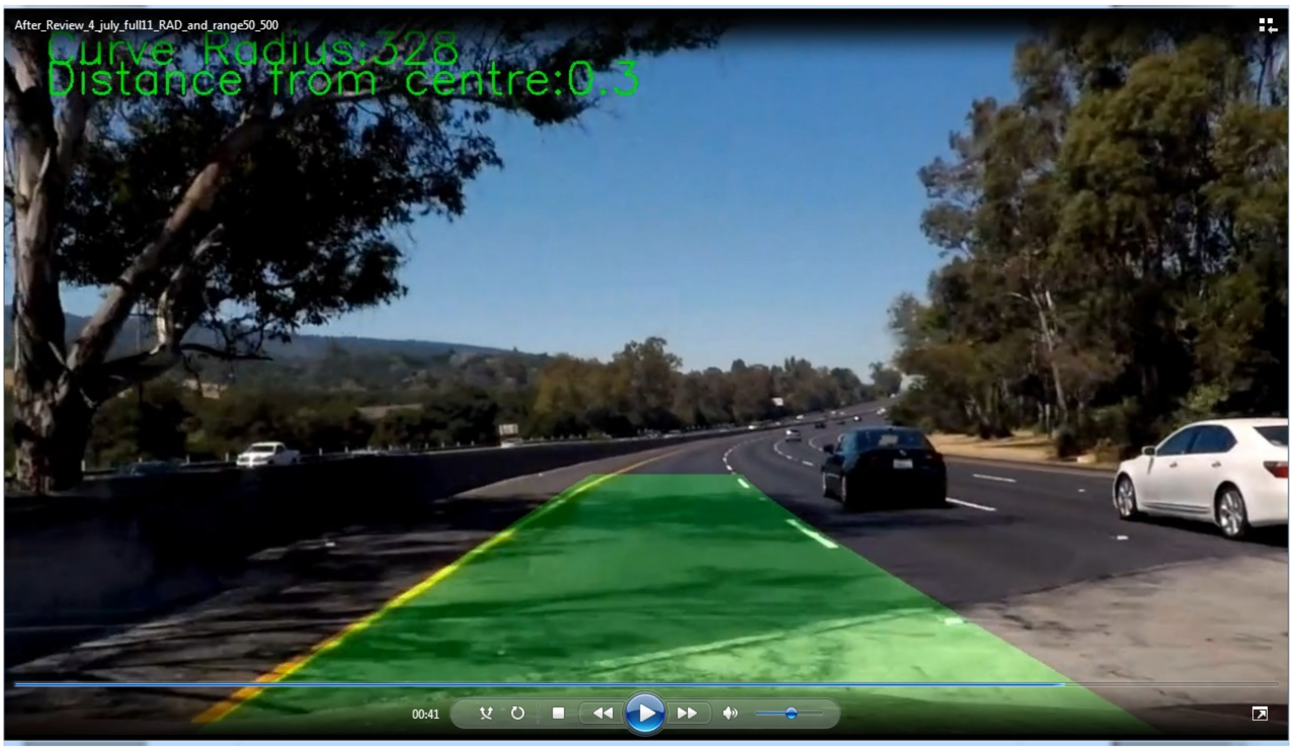


```
changed a right right_curverad 845.726085292 last_good_right_curverad 633.351122489
 62%|██████████████████████████                | 785/1261 [17:54<10:54,  1.37s/it]
left_curverad: 1078.93268465 right_curverad 765.140186477 average_curve 922
changed a left left_curverad 1078.93268465 last_good_left_curverad 2535.79631542
changed a right right_curverad 765.140186477 last_good_right_curverad 633.351122489
 62%|██████████████████████████                | 786/1261 [17:56<10:50,  1.37s/it]
left_curverad: 1104.11221854 right_curverad 732.771278545 average_curve 918
changed a left left_curverad 1104.11221854 last_good_left_curverad 2535.79631542
 62%|██████████████████████████                | 787/1261 [17:57<10:45,  1.36s/it]
left_curverad: 1177.98295123 right_curverad 803.035777844 average_curve 990
changed a left left_curverad 1177.98295123 last_good_left_curverad 2535.79631542
 62%|██████████████████████████                | 788/1261 [17:59<10:42,  1.36s/it]
left_curverad: 1028.46260337 right_curverad 663.828591878 average_curve 846
changed a left left_curverad 1028.46260337 last_good_left_curverad 2535.79631542
```

Before: As submitted first attempt.



After resubmission.



Discussion of improvements:

From the above it is clear that the poor fitting right lane has been replaced by the last good right lane and fits well. It is also clear that the left lane does not fit perfectly. This is because it too was rejected and replaced with the last good left lane which fits quite well but at the top of the left side near the centre of the picture the green colour extends a little beyond the yellow line. In fact there are several frames in the video where this happens.

Overall the pipeline works reasonably well there are no very bad fits but there are a few frames similar to the above that are not perfect.

At this point it is definitely better than the first attempt and I am going to resubmit but I don't know if that is quite good enough.