**Udacity Self-Driving Car Engineer**

**Report**

* **Project:** Advanced Lane Finding
* **Name:**  Shao Hongxu

**Let me just use some brief sentence to explain the thing I did, basically these are 8 big parts here:**

**Part 1:** import necessary lib

**Part 2:** compute the camera matrix and distortion coefficients

**Part 3:** image pre-processing

**Part 4:** convert image to a "birds-eye view"

**Part 5:** lane line detection

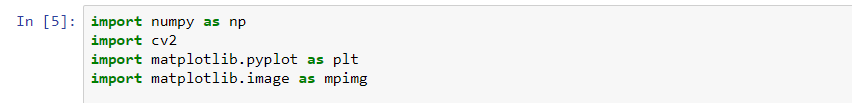
**Part 6:** meter calculation

**Part 7:** warp back the image

**Part 8:** Summary

**To be specific as below, you will find some note on Script, key take away and Area could be improved:**

**Part 1:** import necessary lib:



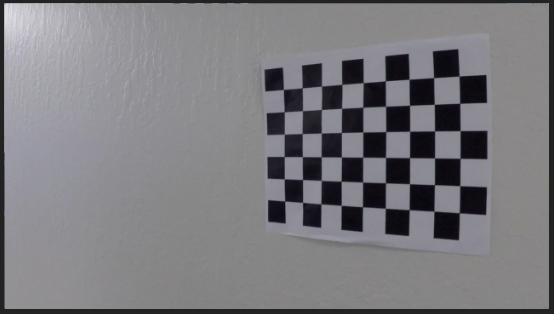
**Part 2:** compute the camera matrix and distortion coefficients:



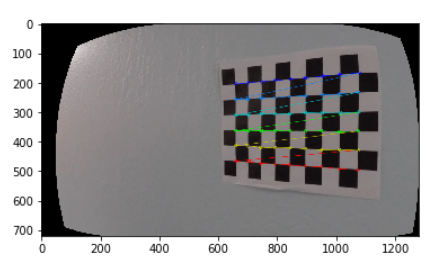
**-> Key take away:** To archive the goal of distortion correction, the main parameter is to identify **chessboard size, in this case, it is 9x6**.

**-> Result:**

*Original image:*

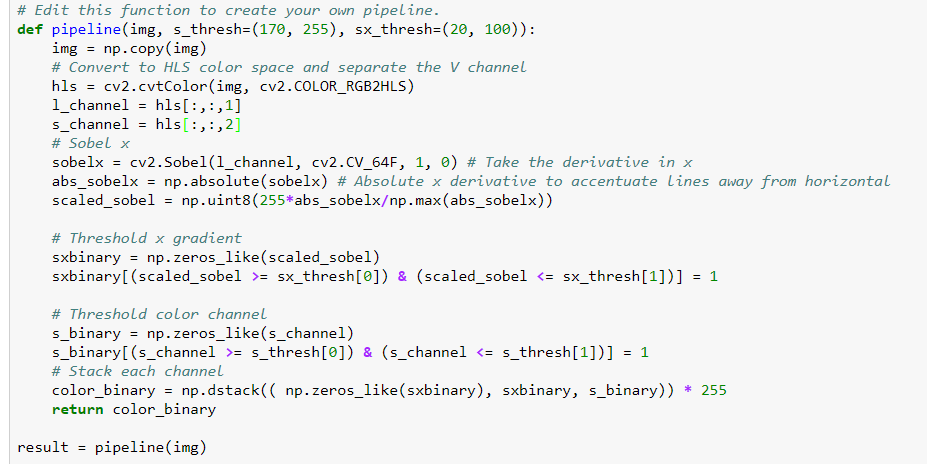


*Image after distortion correction:*

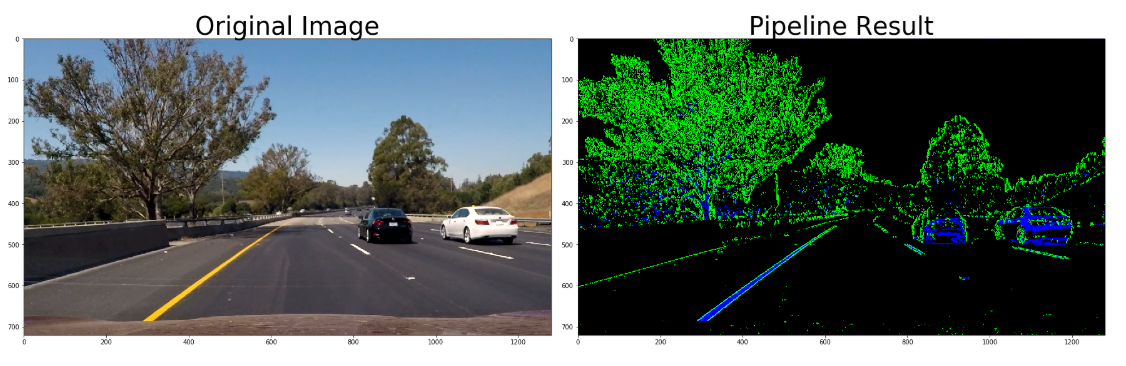


**Part 3:** image pre-processing

Create a function like in Udacity lesson, draw the pipeline for input image.

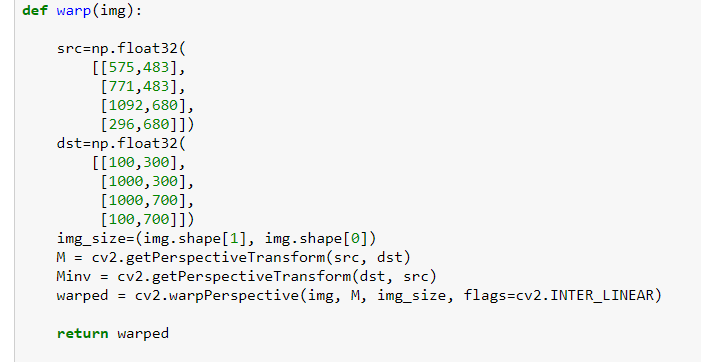


As a result:

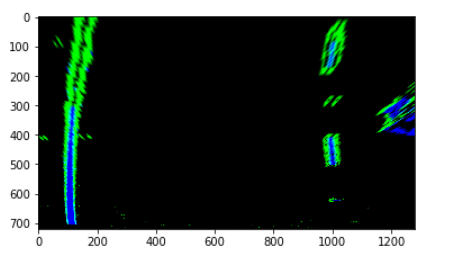


**Part 4:** Convert image to a "birds-eye view"

convert the pipeline image into bird view using below function. Here you can see, I have identified 4 points to map between pipeline image & bird view image.



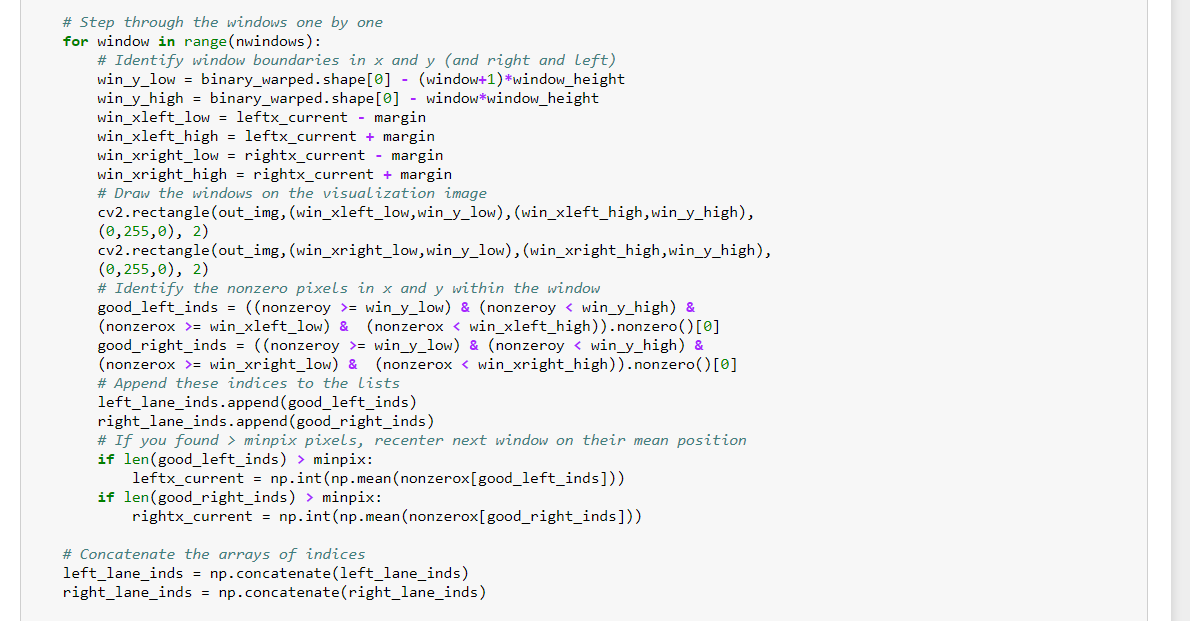
As a result, you can see below image as a bird view image output:



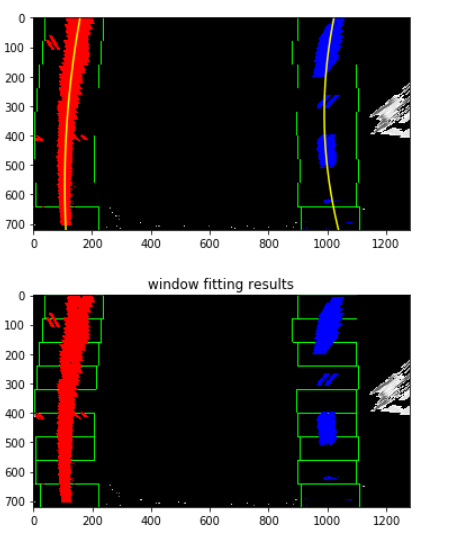
Here the key take away is, better to specific the area for 2 lines only. I have tried to include as much information as much from original pipeline image, but the result not that properly. Sometimes left side road corner has been considered as lane line. And I realized I should give more restriction, and that indeed improved the code as a result.

**Part 5:** lane line detection

For next step, is to identify the left & right lane line respectively in different colour, here is the lesson code to archive that:

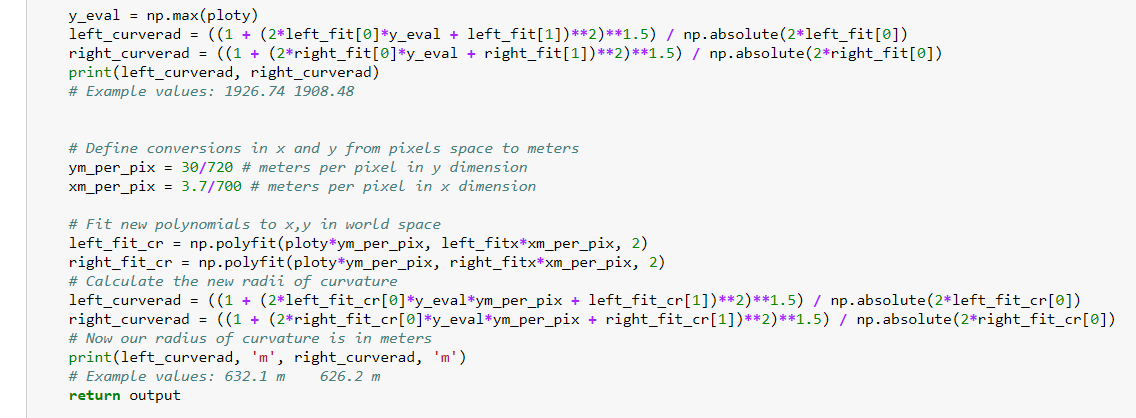
 

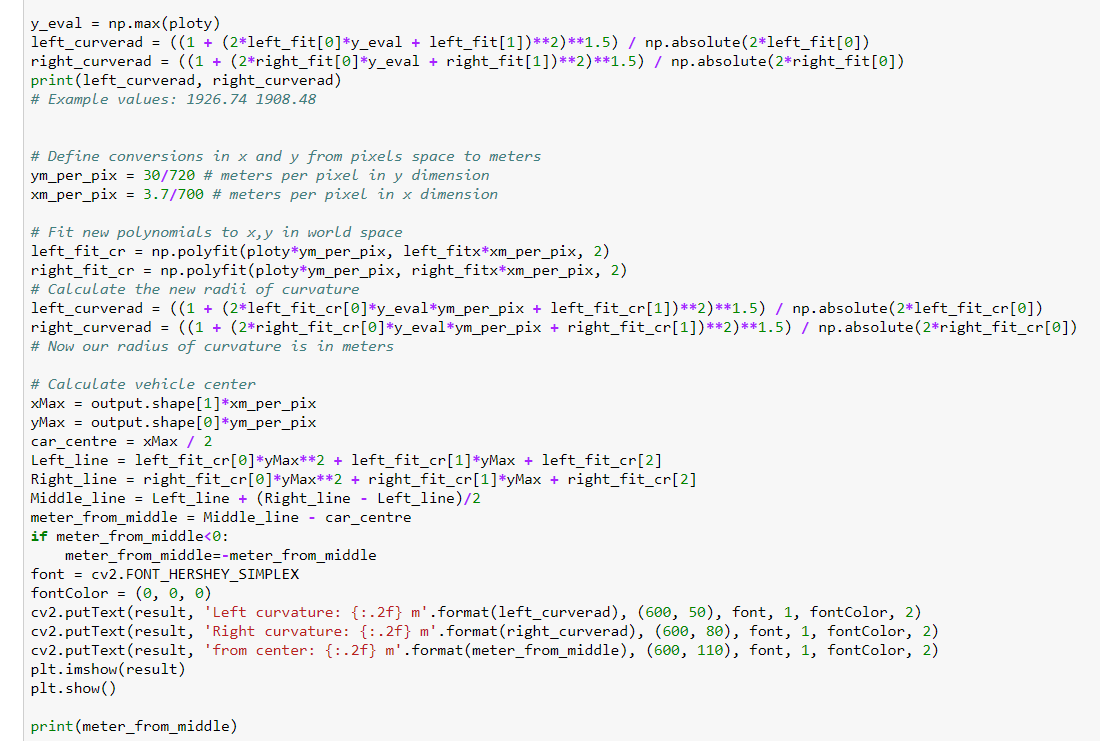
As a result, code is able to detect the 2 lines with colour red on left & blue on right.



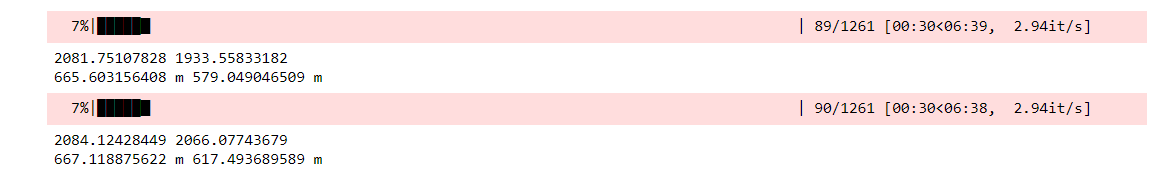
**Part 6:** meter calculation

At the same time, lane meters will be calculated using below code:



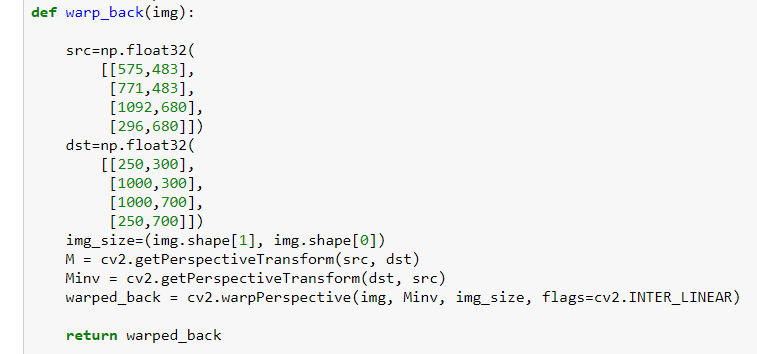


As a result, code will print out the number after each image conversion:

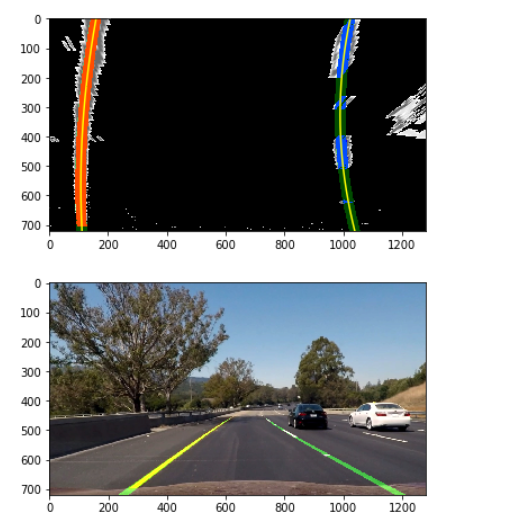


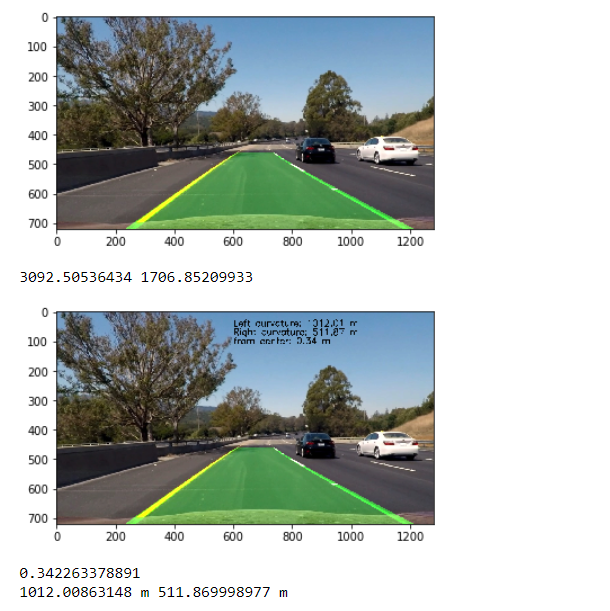
**Part 7:** warp back the image

Next step after lane line detection, is to convert lane line back from bird view into normal view. Combine them together into a single picture using below code:

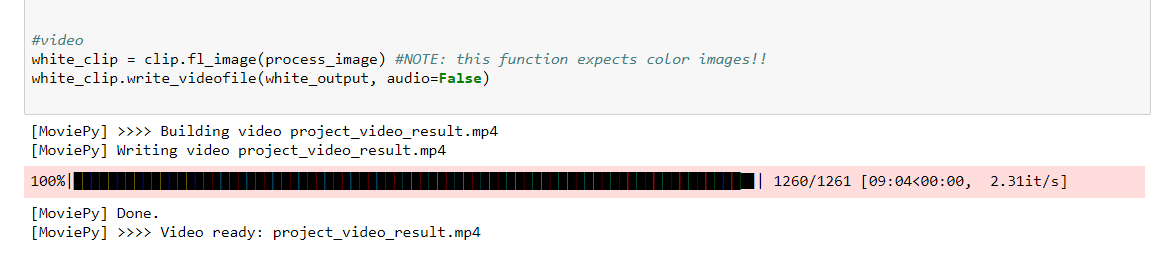


As you can see, same conversion matrix used for warping, but opposite way. As a result, image after all method will be shown as below with lane line in light green.





Inside project, I made video generation code as a separate set and to let first part of code used only for showing single image result. For video part, code will capture set of images from video and put them through the same conversion logic as we have done for image above. And showing process as below:



**Part 8:** Summary

These are the steps I could think of for project 4, as a base line. Still a lot of improvement needed to achieve a better accuracy.

Like the meter calculation, maybe it can be given a better exchange rate, now left & right meters are too much different most of the time.

Overall, I have learnt a lot from this project and compared with the first project, the lane line detection solution is much much better!