PROJECT 4

ADVANCED LANE FINDING

One thing that I would like to mention before anything else is that of all the projects that I have done, so far, this was hands down the most difficult.

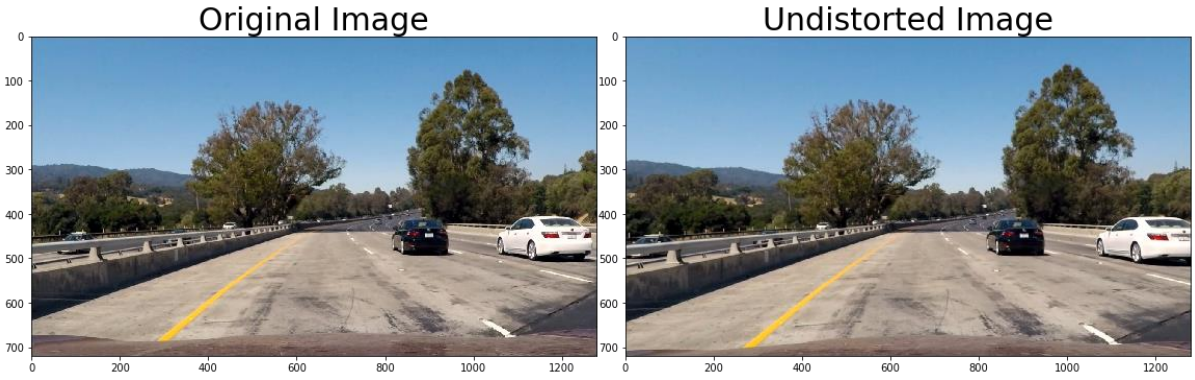
I was literally stuck on the same error for like 2-3 weeks, in the end it turned out that I was simply missing **.shape()** at one point (Yeah, it made me quite sad ;-;), and at one point I mistakenly swapped x and y in the nonzero() function which made the lines plot in oblivion. Apart from all these silly mistakes the hardest part was using the class(), while I am familiar with how it works in JAVA, here it was quite new and had me seeking for help a number of times, but I was fun to learn nevertheless.

Alright moving on to the actual report…….

The actual report:

So, to start things of we have to calibrate the camera. This was simply done as portrayed in the lesson.

We need to calibrate the camera so as to remove the distortion from the images we do this by obtaining the camera matrix (obtained by detecting the chess board corners). This camera matrix can then be used to undistort the images, while in this case it is minimal and can only be seen ever so slightly, it is more prominent on some cameras.



The effect can be most easily seen on the hood of the car, and on the white car.

Next we need to change our perspective, to get a bird's eye of the whole scenario and to also remove unwanted clutter such as cars, shadows, and tree and only focus on what is important , i.e. the road and the lane markings.

The perspective transformation is fairly easy, the important part though is guessing the source and destination points. The more parallel lines we get the better our detected lane lines can be.

The more parallel lines also help in better detection of lanes, the system as an easier times fitting polynomials to the lanes.

VISULAIZATION:

After this I visualized various channels, to choose the channel for our purpose, there were many choices HSV, HLS and LAB looked best. So, I used the B channel from LAB space and S channel from the HSV channel (they did a pretty good job detecting the lanes). I originally played around with these parameters quite a lot (getting varying results) then one of my friends recommended me these channels.

Then combined the results from all the required channels, to get the results.

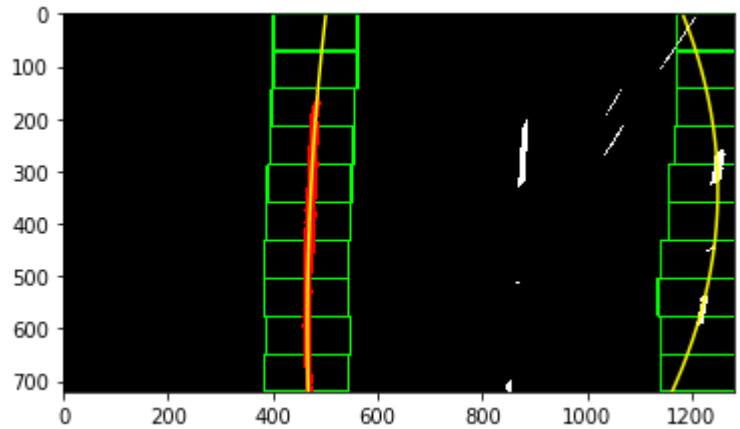
SLIDING WINDOW AND PREVIOUS FIT TECHNIQUE

After this I simply did the sliding window technique and the polynomial fitting method as explained in the lesson, nothing special there.

Simply finding out all the non zero pixels in the binary image and then using them to plot the rectangle on the detected lanes. Then re-centring the windows to the new location when a certain number of one hot pixels have been found.

Then according the indices found in the function we can find the constants a,b and c which are used to define a quadratic equation. These constants can then be used to plot the detected lane lines on a copy of the original image.

However when I first made the images it looked something like this.



As can be seen this is wrong, so to correct this I made changes in the histogram feature , instead of searching for one hot encoded pixels in both the halves of the images, lets us just search for them in only the quarters, where the lanes lines have the maximum probability of existing, this not only corrected the image but also reduced the work load as the now the code has search for one hot pixels in just a small region.

Then there is the previous window method, nothing special there either. Just a simple method of using the equation constants from the previous frame instead of finding new ones.

[NOTE: This method has certain limitation such as the fact the next image must be a direct continuation of the previous image, so as to use this technique most effectively.]

RADIUS OF CURVATURE:

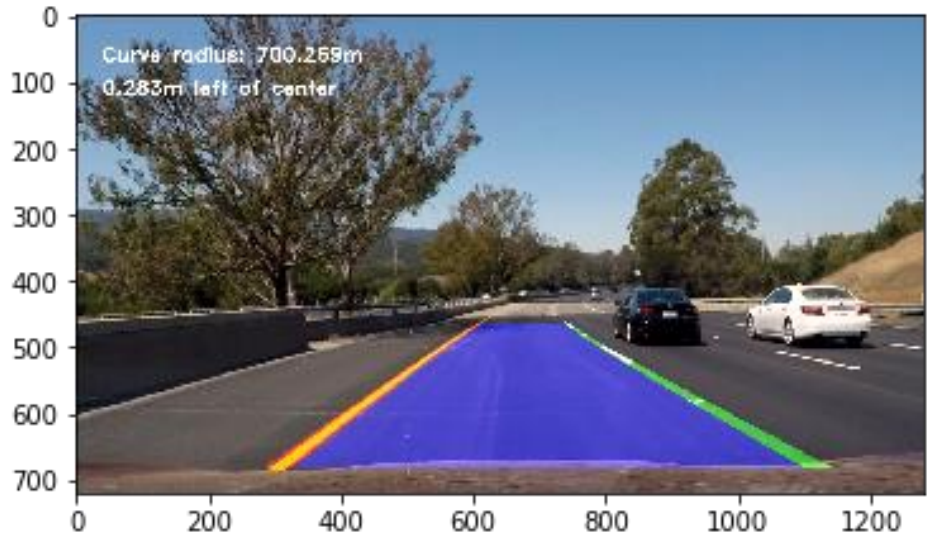
I really figure this out, something just were not falling in place. Obviously having a lane line having a radius of curvature of 1000km is quite wrong.

Anyways, the most important part about this part is converting the distances from the pixels distances to real life distance.

Apart from this everything is simple as should be done.

FINAL RESULTS

Then I simply plotted the results on a copy of the original image.



As can be seen the detected lane lines have been plotted on the original image along with the radius of the curvature.

FINAL VIDEO:

Finally I made a pipeline for making the video, now I am not quite familiar about how classes work in python (I know how they work in java), so this part took some understanding even though the code was already available from the lesson itself.

The best part about this is multi window output, I really like it and seeing how easy it is to make I was really happy, and also it looks really cool.

I actually made another video from my mobile of my college campus road. Also took images and tried my code on it. The results were bad, you can see the results in the copy of the jupyter notebook.

Few changes that I did do were to use a mask to remove unnecessary areas of the road such as the edges and the centre part, I also applied Gaussian blurring.

But because of the resolution differences the results are not good at all in no way.

IMPROVEMENTS:

My code didn't really work on the challenge videos. Certain improvements that can be done:

1. Using sobel threshold, absolute sobel and sobel direction.
2. Masks.
3. Better lane detection by using more than 3 channels.

So, you have reached the end of my boring report.

Also, I would really love to hear anything that you can suggest

Thank you!!