Advanced Lane Finding Project

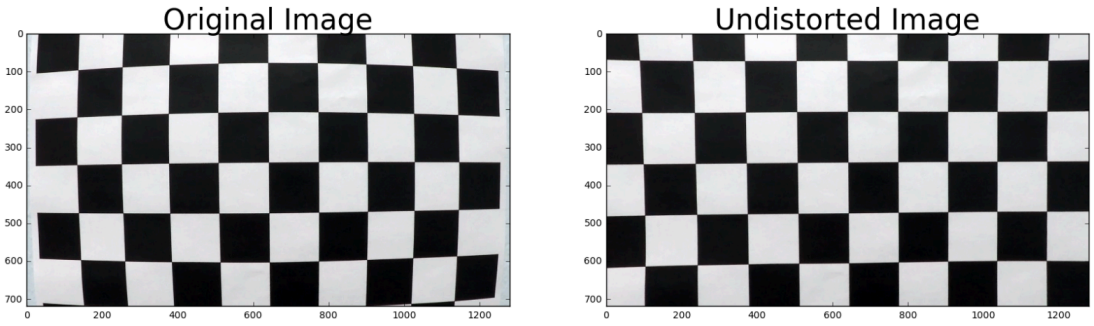
1-Camera calibration:

The code for this step is contained in the cells 2 and 3 of the IPython notebook.

Following the course material, I created the "object points", which will be the 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. The result is like this:



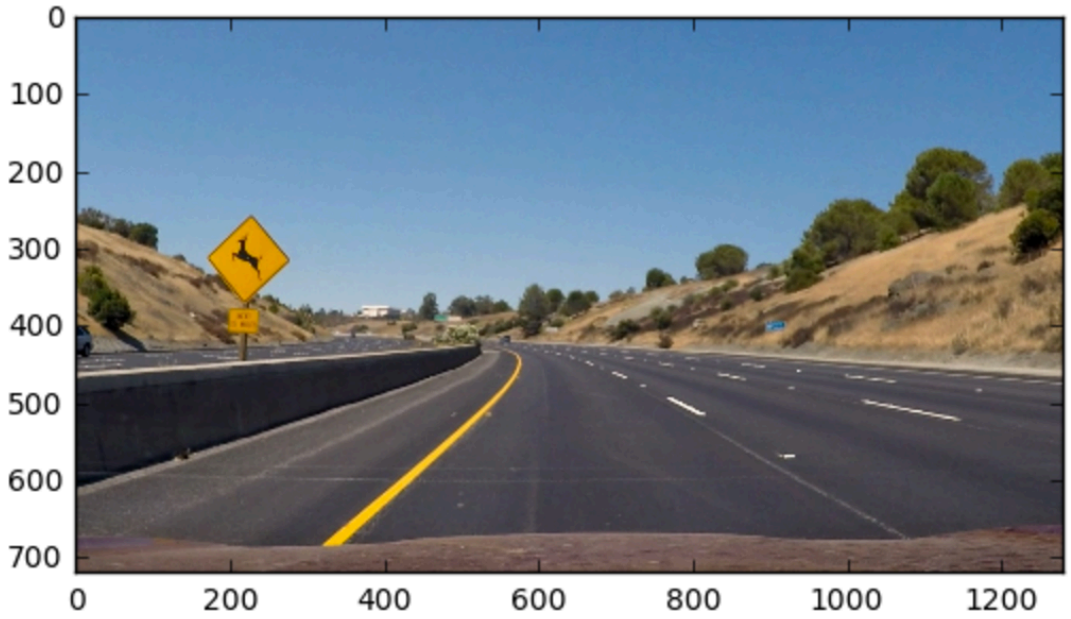
After camera calibration step, the camera parameters will be kept in ret, mtx, dist, rvecs, tvecs variables and can be used in the pipeline to undistort the images.

2-Pipeline

Before creating the pipeline, I tried to test each individual pipeline step with different sample images in order to adjust each pipeline step first and then connect them together.

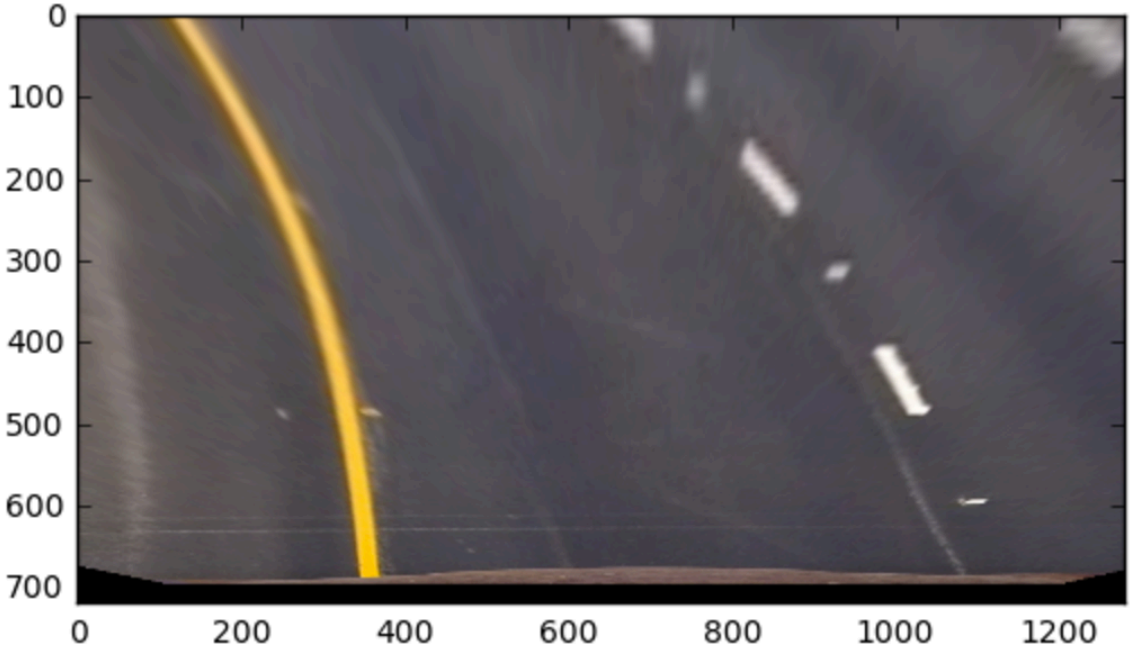
2-1- Undistort the image:
Cells 4 and 5 show an image from camera and the undistorted image. It can be seen that the edges of the image have been affected by undistorting. Also in cell 5, I tried to find 8 points on the image to be used for perspective transform.

Image from Camera:

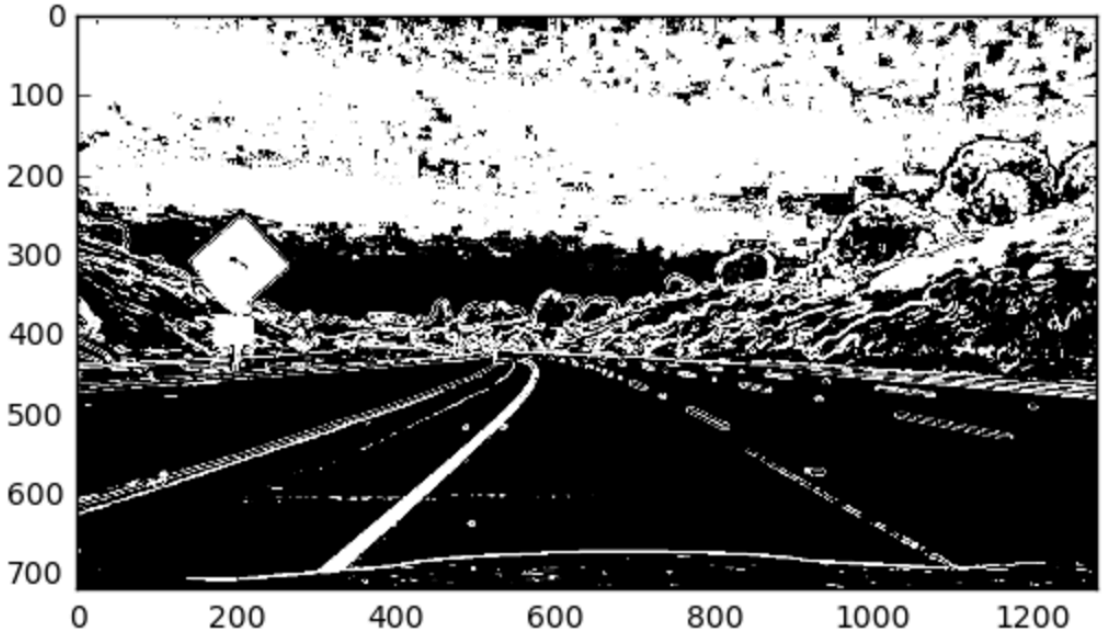


Undistorted image:

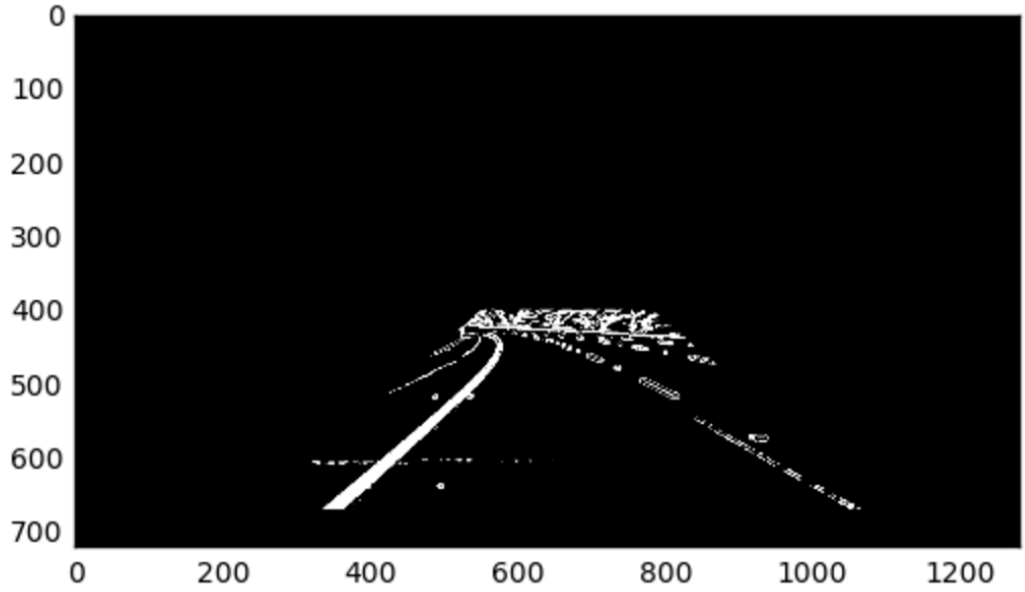
2-2- In cell 6, I used the points which I found in previous step to be passed to cv2.getPerspectiveTransform and created 2 functions, one to do perspective transform and the other to do reverse perspective transform. **I applied the perspective transform to image from previous step only to observe the result. This step will not be part of the pipeline**



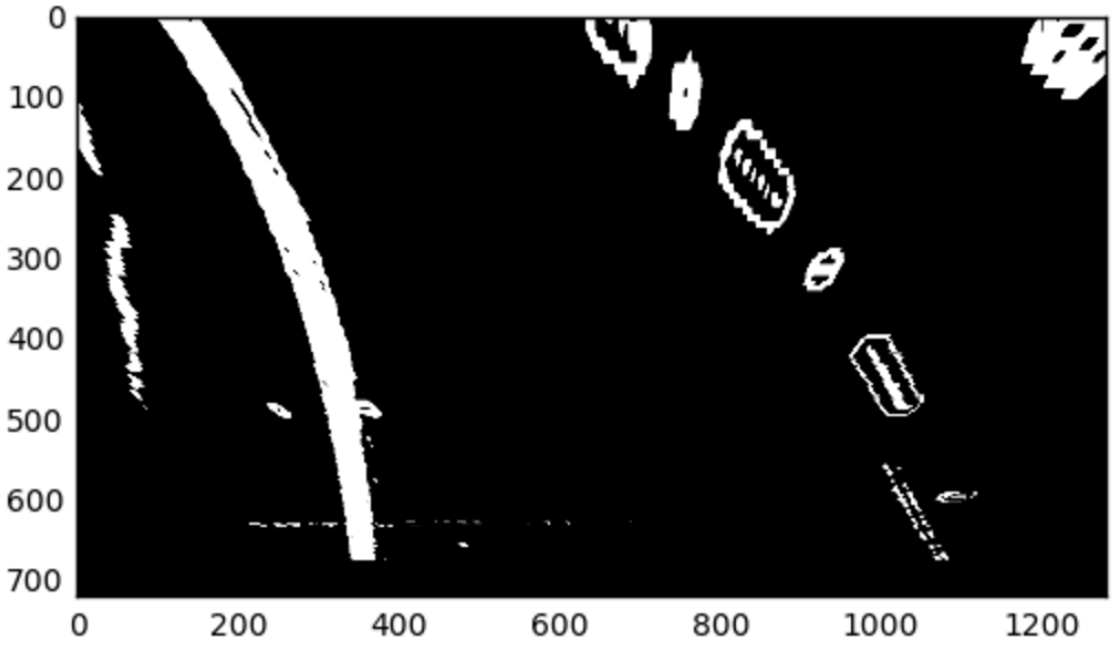
2-3- In cell 7, I created and tested a function which applies gradient and color thresholds using the ideas from course samples and then tested it with different arguments to get a good output. Here's the result:



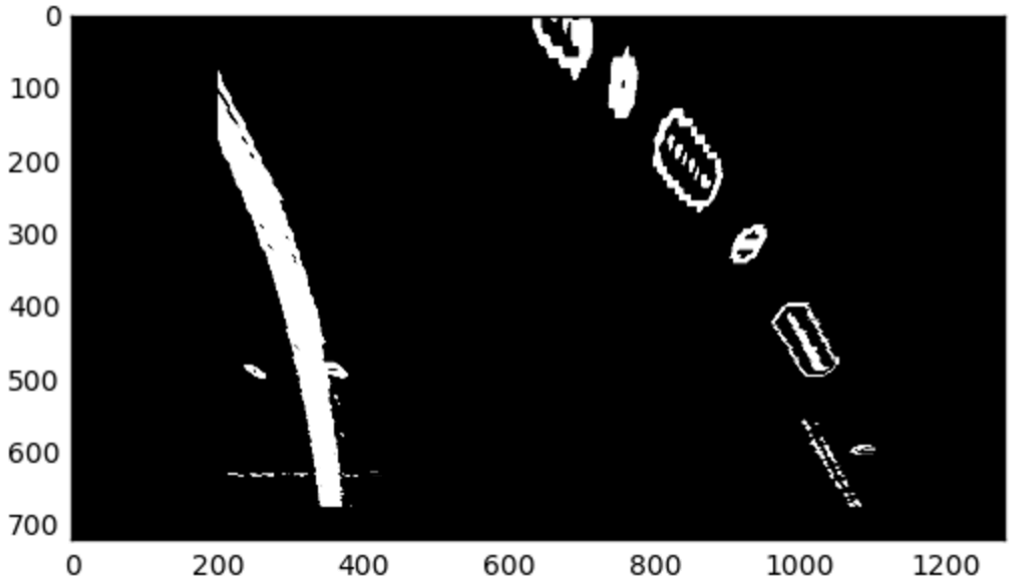
2-4- In cell 8, I used the function "region_of_interest" from project 1 to mask the image and remove the areas outside the road. Here's the result from this step:



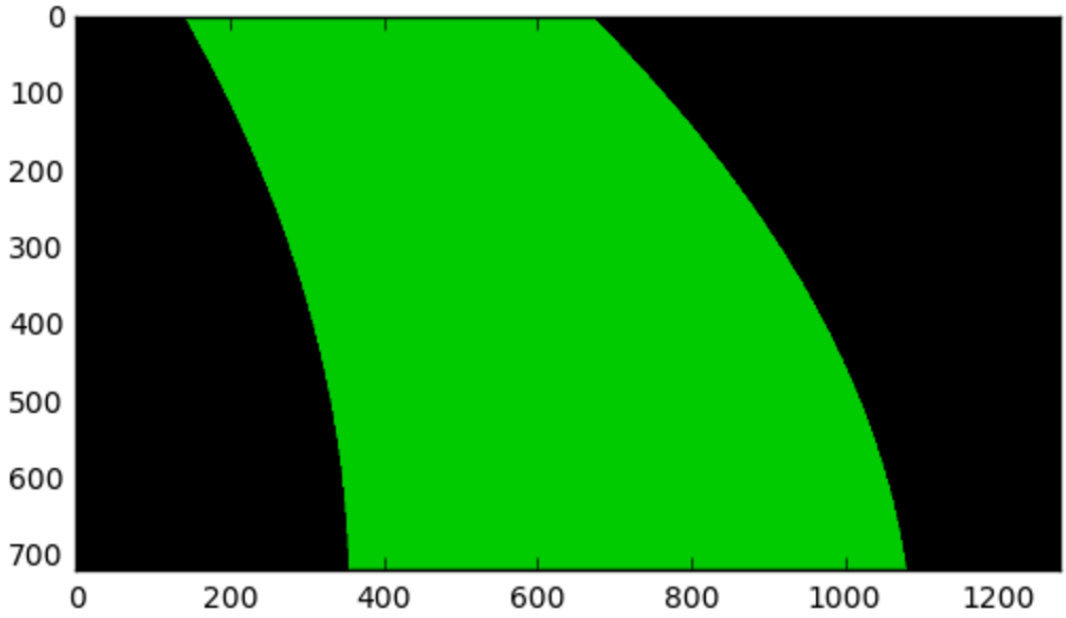
2-5- In cell 9 I applied the perspective transform to previous image. Here's the result:



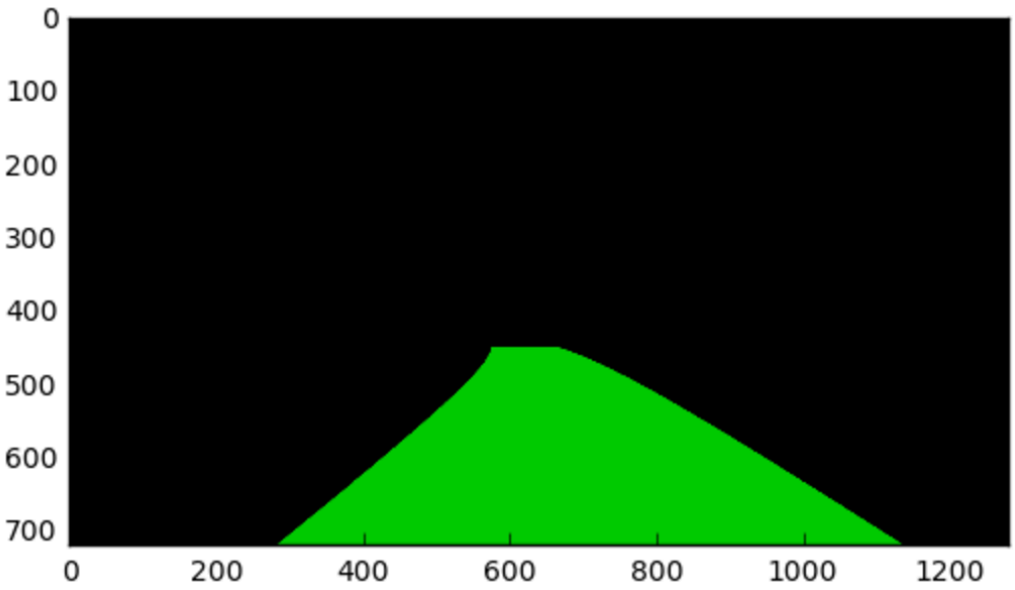
2-6- Looking at different sample images, I noticed there are still extra pixels around the lines. So in cell 10 I applied another mask to the perspective again to have filter out some of the extra pixels. Here's the result:



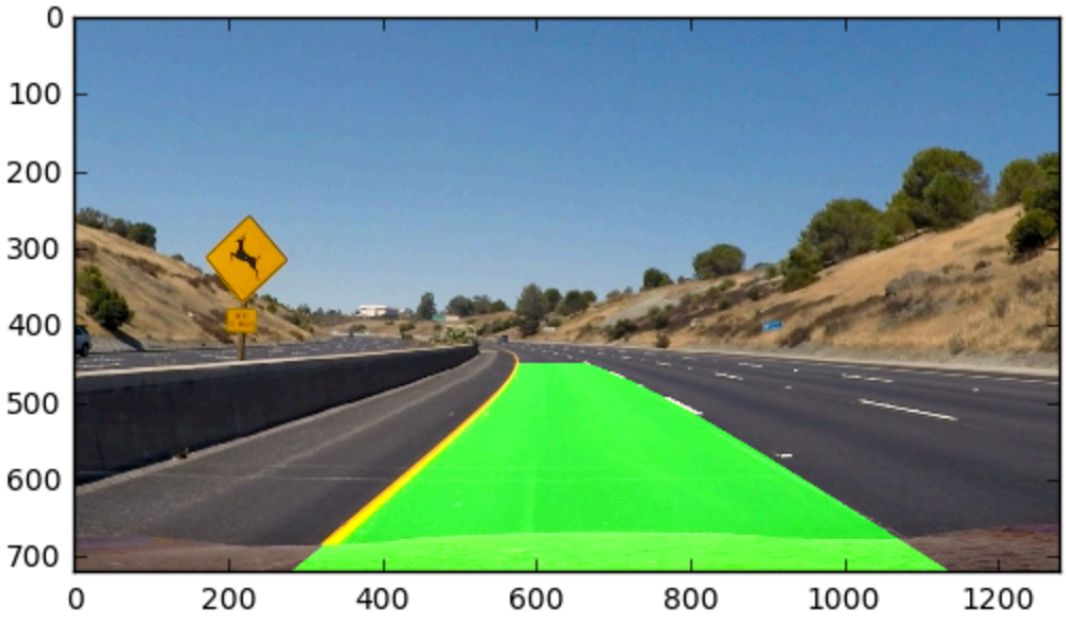
2-7- In cell 11, I used the code from course and made a function which find the lines using peaks in histogram. The function fits second order polynomial to the curves it finds. Also I filled the area between the curves with solid color. I did the calculations to find the radius of the curvatures and the distance from the lines in this function. The values are stored in global variables to be used in the pipeline. Here's the result of applying this function to previous image:



2-8- In cell 12, I used the reverse perspective transform to get the curves which will be matching the original image. Here's the result:

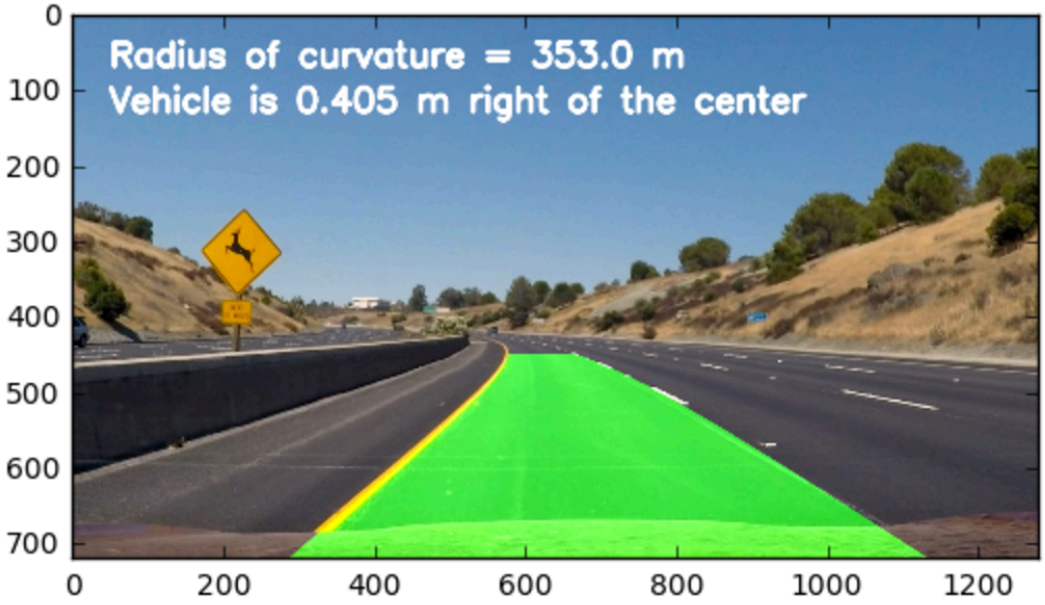


2-9- In cell 13, I combined the original image with the previous result, showing the area between lines for the sample image:



2-10- Combining the steps explained above, in cell 14, I built the overall pipeline which output of each step will be fed to the next step. Also I printed the radius and distance in to the image inside the pipeline. Here's the result of applying

pipeline to original sample image:



3- Creating the videos:
In cell 16, I created the output video with applying the input video and feeding each frame to the pipeline.

Discussion

As I was trying different parts of the pipeline, I experienced that adjusting the threshold is the very critical part of the pipeline and finding the sweet spot is not easy. As the parameters to the apply_threshold function change, it works better for road areas with shadows for example but works worse for areas which the pavement color is different. A good threshold step in the pipeline will make the result much better.

I did some experiment on challenge videos too although because of the deadlines I won't have enough time to get an acceptable result. For the "challenge_video.mp4", the side line and the middle line cause problems. Some modifications on search lines algorithms may help. I was thinking of finding 4 lines for example and pick the 2 which their distance is close to distance of the road lines.
The other improvement can do a smarter search instead of complete search, as rubric is suggesting.

For “harder_challenge_video.mp4”, the perspective should change. Using the same perspective dimensions used for the first video doesn’t work for this video, because the visible part of the road is shorter in this video.