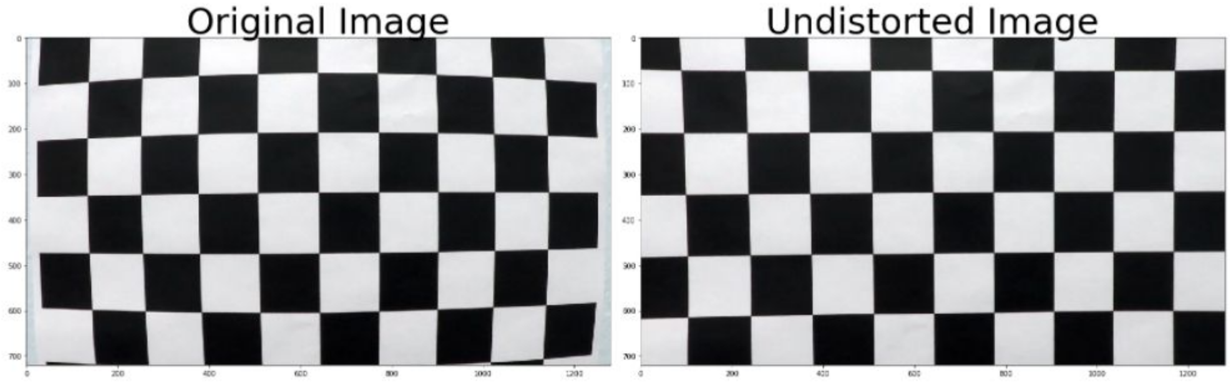
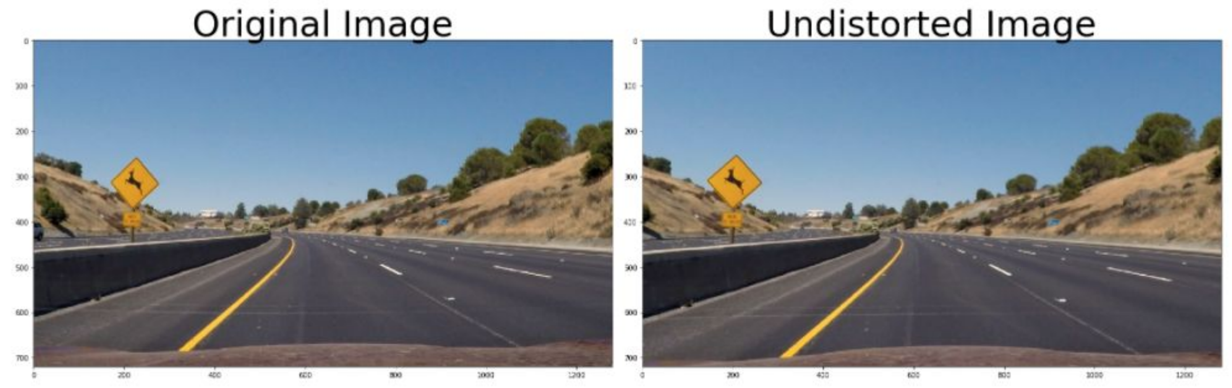
CarND - Advanced Lane Lines

By James Marshall

Camera Calibration

Using the 20 calibration images for the camera, the calibration coefficients were found using the cv2.findChessboardCorners and cv2.drawChessboardCorners functions. These functions gave image points, which were then used in the cv2.calibrateCamera function to give the calibration coefficients needed.
With these coefficients, the undistort function is defined in the fifth code cell, which returns the undistorted images found in code cells 4 and 7.



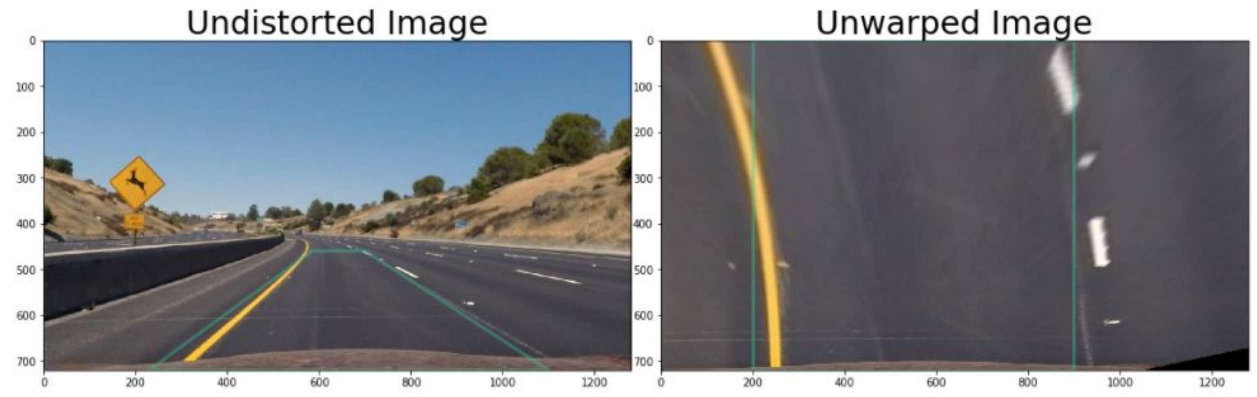


Perspective Transform

To apply a perspective transform to the images, I hardcoded the source and destination points as follows (can be seen in lines 4-14 in code cell 9 in the notebook):

Source	Destination
230, 720	300, 720
585, 460	300, 0
700, 460	900, 0
1100, 720	900, 720

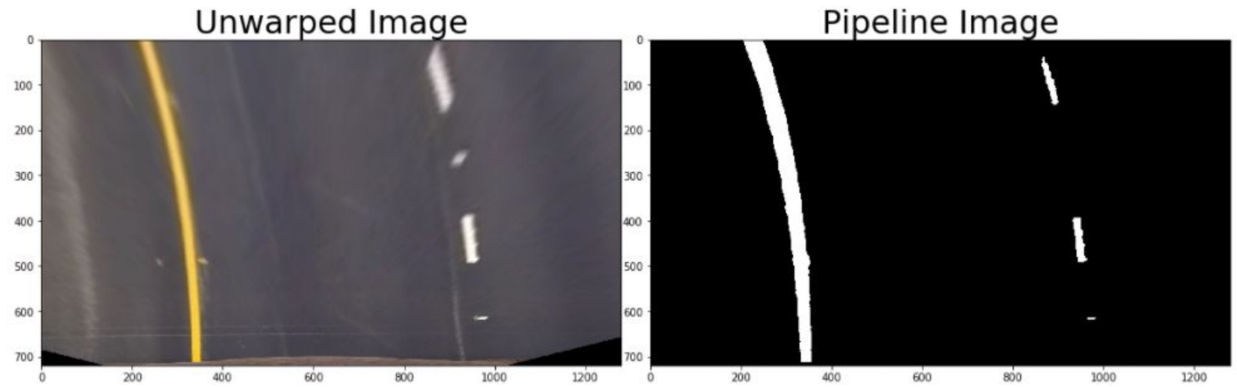
These points are then used in the warp function (shown in code cell 8) to return the warped image, plus the two transform matrices. The example image is created in code cell 9.



Threshold Binary

After experimenting with gradient, magnitude and directional thresholds as well as thresholds in HLS colour space, in particular the S and L channels, I went with a combination of the L channel OR B channel. If either of these returned 1 in the binary image, the overall binary image returned a 1 for that pixel.

The experimenting with each individual threshold are shown in the code cells 10-22, and the image pipeline function that turns the original camera image into the binary image, is written in code cell 23, with the example image being shown in code cell 24.



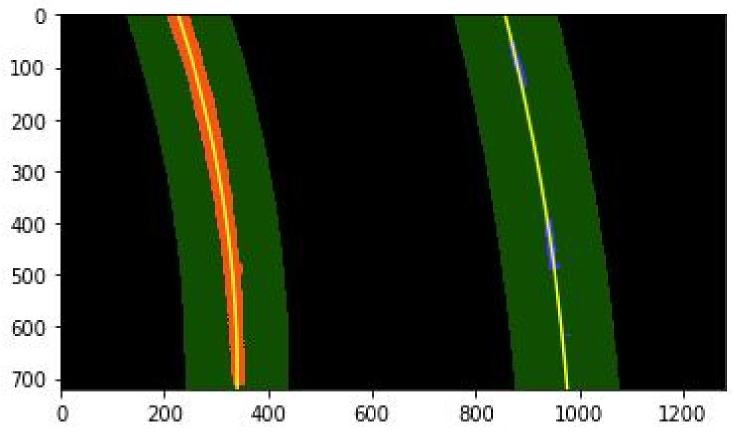
Sliding Window Lane Finding

To find where the lane lines are in the binary image, I firstly created a histogram of the distribution of the binary image. I also select the amount of windows that will be in each frame, and get the base of each lane by taking the x pixel value where there is the highest concentration of pixels in each half of the binary image.

From there, each successive window's base is the average x value of all pixels in the previous window, which is repeated until all windows have been drawn. While these windows are being created, every non-zero pixel within those windows are stored, as the final step uses all of these non-zero pixels to do a polyfit, getting the best equation of the lane line. I used a quadratic polyfit, which is all shown in code cell 26.

Then, once the lane lines have been found in one frame, a new function (code cell 28) can be written, which saves time looking for the lane lines by using the previous best fit for the lines.

The output of this function is shown in code cell 29.



Radius of Curvature and Distance from centre of lane

The first thing that needed to be done was to figure out how many metres there were per pixel where I was calculating the radius and centre from in both the x and y directions.

Next I had to calculate new polyfits, this time in real world space, so that I could apply the radius of curvature equation (shown on this slide) to get the radius of the curve.

To find how far the car was from the centre of the lane was to get the car's position, which is clearly the centre of the image, and get the midpoint of the two lane lines, by averaging their positions at the bottom of the image and subtracting the two from each other to get the distance from the centre.

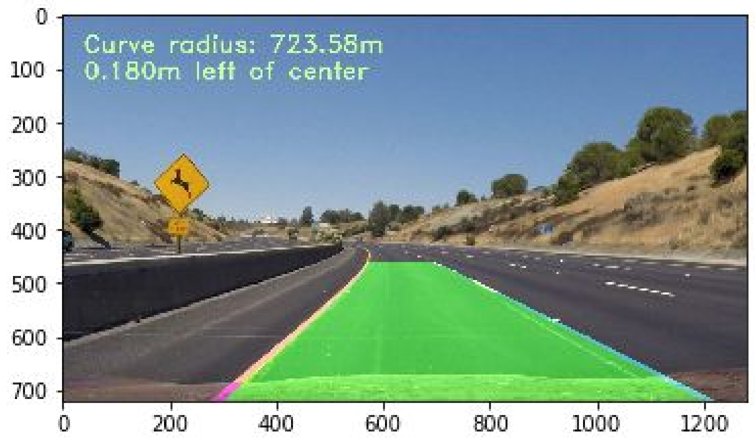
This function is written in code cell 32.

Image Overlay

Overlaying both the lane projection and the lane data on the image was split into two functions. In code cell 30, the draw_lines function creates arrays that contain the pixels where the lane lines are. cv2.polylines and cv2.fillPoly draw the lanes, and fill the space between them, before unwarping them so that they can be placed over the original image.

To print the curve radius and the distance from centre I made a draw_data function (code cell 34) to print the data on the screen, which is straightforward, however I had to check whether the car was on the left or right hand side of the lane, based on the sign of the centre measurement.

The example output is shown in code cell 35.



Video Pipeline

The video pipeline is included in the submission folder.

Discussion and Reflection

The largest problem that I faced with this project was the part of the video where there were changes in road surface colour and shadows across the lane simultaneously. This was fixed by using multiple colour channels in the HLS colour space, instead of just the S channel, which I was originally doing.

Another problem I faced, which I still believe could use a lot of improvement, is the accuracy and consistency of the curve radius measurement. I think the main way for the accuracy to be improved is first of all get a much more accurate measurement of metres per pixel, which could be done by finding a frame in the video where an inside lane line is closest to the bottom of the image, and using pythagoras and an estimate of the angle from horizontal.

A problem that I know my pipeline has an issue with as I have seen from the challenge videos, is both when the road surface becomes even more consistent, in which my pipeline doesn't correctly identify the lane lines, and also when a corner becomes incredibly sharp and a line leaves the camera frame. My pipeline gets very confused in both these conditions, both of which can't happen in the real world. This could be improved with better threshold methods for inconsistent surfaces and better uses of past frames to predict where extremely sharp lanes curve.