Advanced Lane Lines

Camera Calibration:
The camera calibration is happening in the code cell 2. The first thing I did, was to prepare the object points which are coordinates in 3 dimensions (x,y,z). Due to the nature of the images, the z value is always 0. These fixed coordinates are stored in the list called objp. The chessboard image has 9x6 points.
So, taking distorted samples of the chessboard images, whenever the chessboard detection happens, objp list is automatically appended to the objpoints list. Also, during this process, the imgpoints list, which are the pixel coordinates of the distorted images are also detected. Thus, having two values, object points (essentially the goal points) and the image points (the pixel locations of the chessboard), using the cv2.calibrateCamera() function, the camera has been calibrated. After investigating all the samples and calibrating the camera, the next step was to undistort the image, using the cv2.undistort() function and test the result.

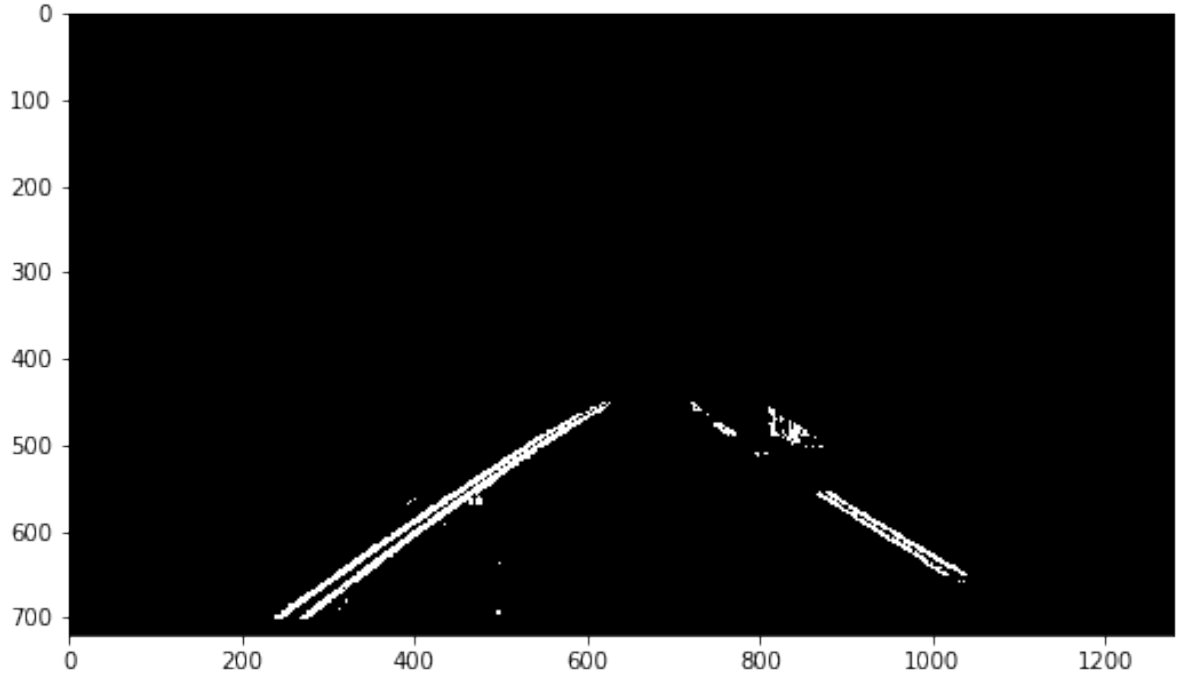
Pipeline

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

After calibrating the camera, I will show the undistorted image from one of the video frames, which looks like this:

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 (R and G to identify the yellow lines) and gradient thresholds (S and L from the HLS, to identify the bright white lines and to eliminate any shadows) to generate a binary image (in code cells 4 and 5). Here's an example of my output for this step.



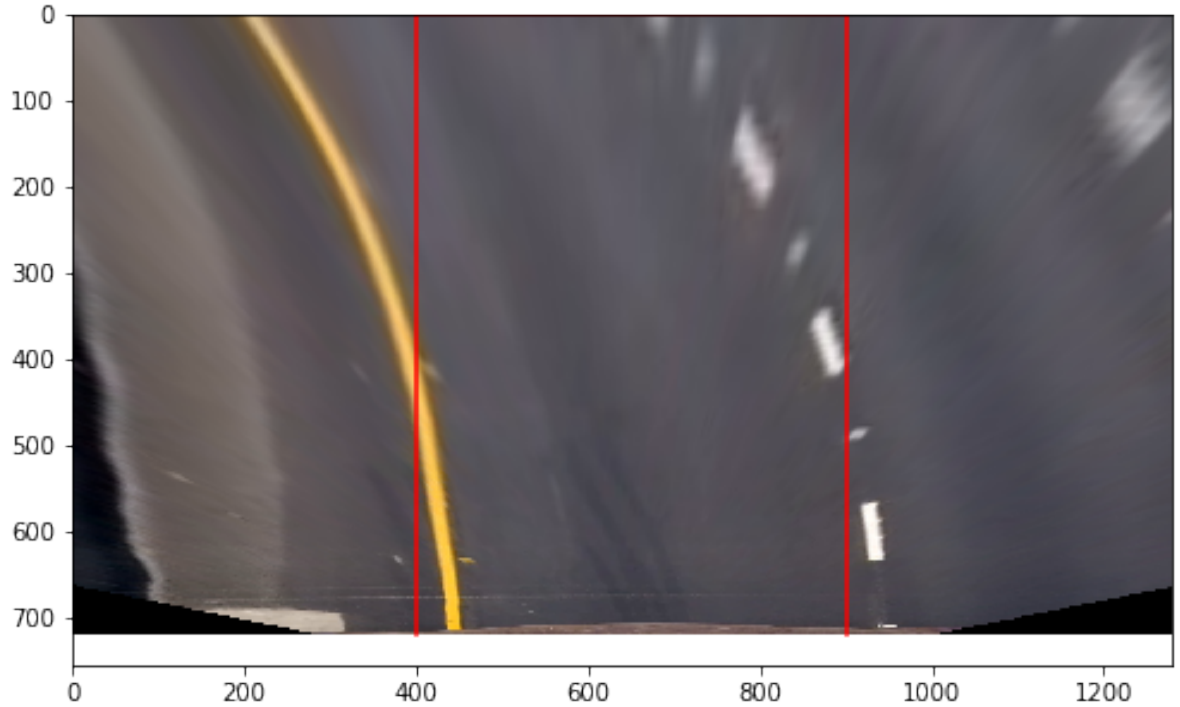
I have also applied a mask to remove any noise.

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

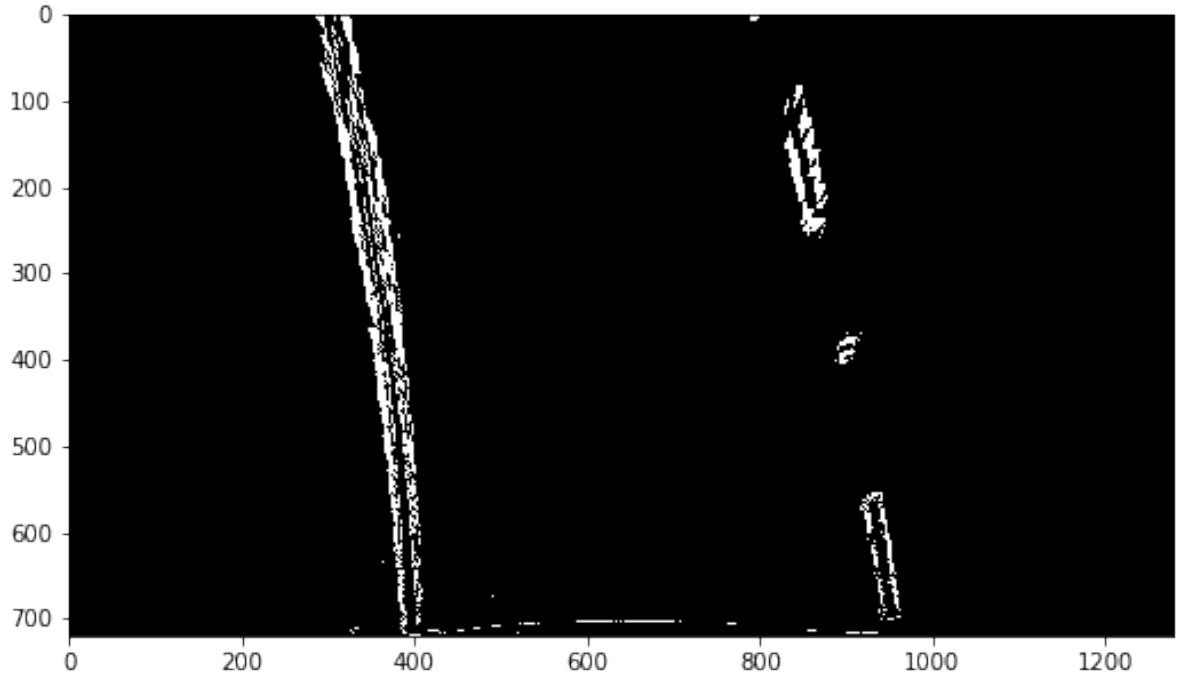
After applying a threshold the next step was to perform a perspective transform, by firstly finding the M value from cv2.getPerspectiveTransform(src,dest) and also M_inv in order to unwarp at the final stage, by simply changing the src and dest with each other. For the source (src) points, I have chosen the values like:
[200,720], [599,446], [680,446], [1100,720]

Which takes this region:

Then, the destination points are:
[200+offset,720], [200+offset,0], [1100-offset,0], [1100-offset,720], offset being 200.
After warping using those points, the warped non binary image looked like this (destination points shown):



So the binary warped image is:



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

In order to find the lane pixels, I have squashed all the binary pixels using the np.sum and found the spikes. Those spikes (x values of the spikes) were the starting point of the lane-lines.
After that, I have divided the image into 9 portions (9 windows) from the Y perspective. Within those 9 windows I have started to search for the pixels by starting from the starting point and only searching within +/- 100 pixels across X axis. If the pixels were found (minimum 50 pixels to satisfy), the next iteration (next window) will start from taking the average of the location from the identified pixels (x axis values). If less than 50 pixels were fond, the next iteration will continue at previous x axis value.

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.

To measure the curvature for the each line, I have used the coefficients values from my polynomial fits and then fitted again but this time, against the empty y values (0-719)

After finding the y positions, I have fitted the polynomial and then applied the formula, for finding the radius of curvature.

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

I will show the final image, from the example mage above:

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

https://youtu.be/jXlH0ru9PNE

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?

The biggest issue I have experienced is with shadows. I have tried numerous experiments by picking the right thresholds and removed most of the shadow by using the L value from the HLS spectrum. Still, it is not perfect and if the shadow will exist for a longer distance, my detection will probably fail big.
The improvement here could be to use the previously saved polynomials in order to approximate the lanes and use better thresholds or different spectrum to remove the shadow completely.
I have edited the spectrum and used R & G or L(lum) or H(hls) and tuned it to the best output. Sometimes, when the right lane is not able to fully detect the lane, it wobbles a bit and I have tried to average, use the previous polynomial, adding the midpoint values to the left values in order to construct a virtual polynomial and even measured the slope of both lines to detect the unusual lanes. However, I did saved the polynomial coefficients for the right lane if the first coefficient is smaller than 1e-4, which I have observed produces the best results. Therefore, if for some reason, the right lane will get out of control and its first polynomial coefficient is larger than 1e-5, then the code just takes the previous polynomial coefficients.