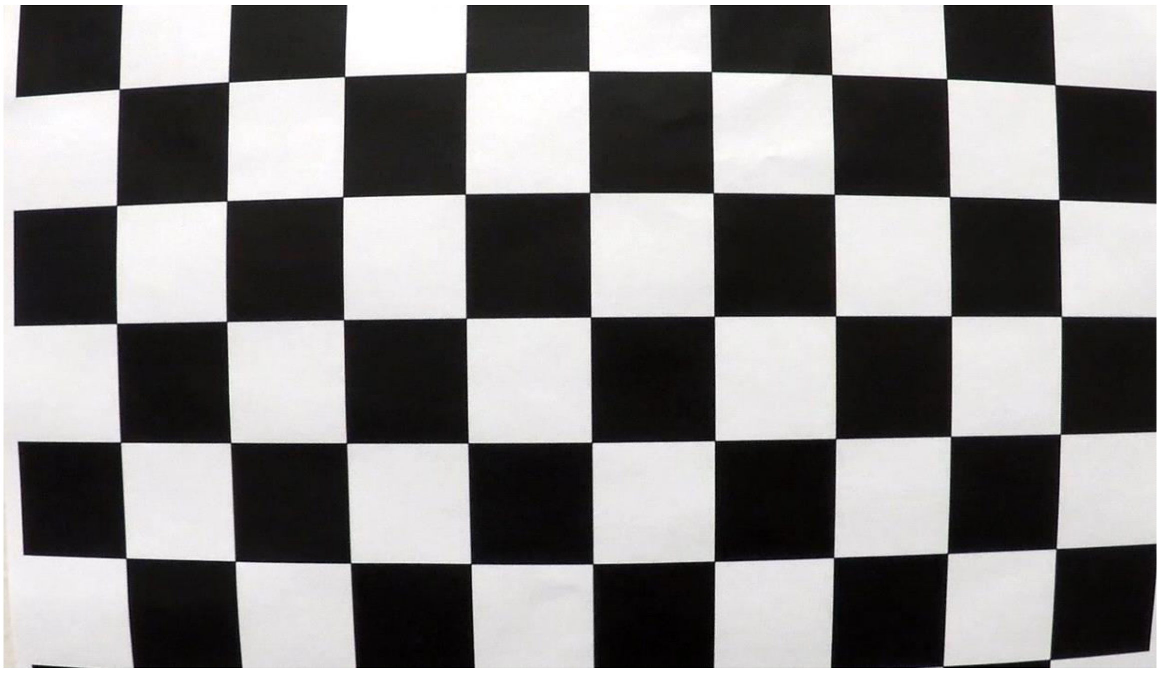
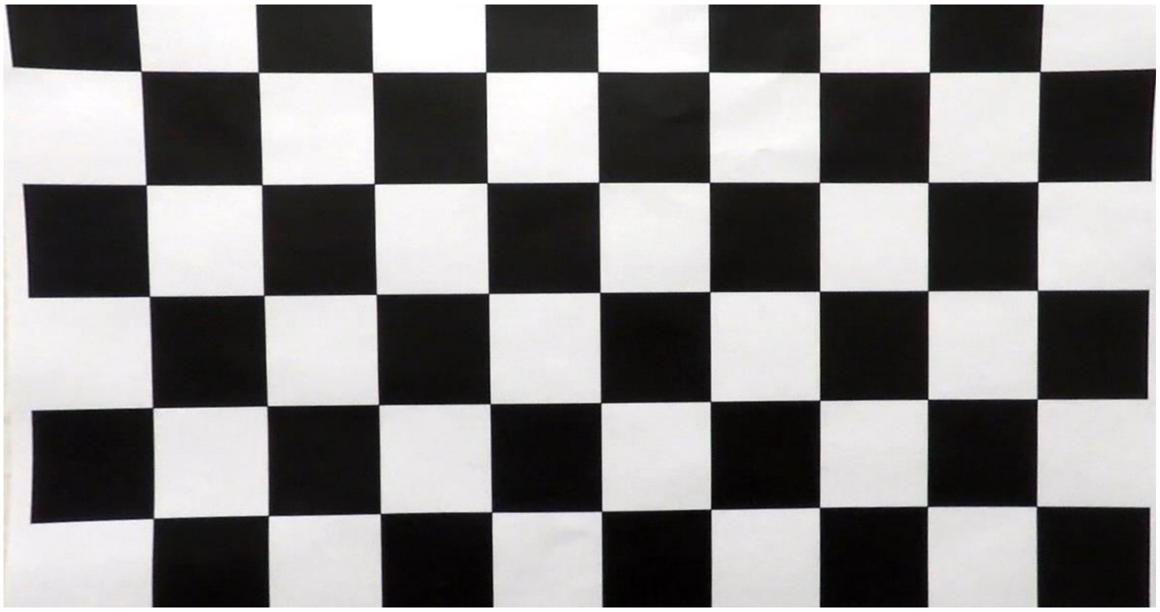
For this project, I created a program that detects lane lines, the curvature of the lane lines, and the offset of the car to the center of the lane. I achieved this in seven steps. The seven steps are: calibrating the camera, converting the video to images, defining functions for applying un-distortions and thresholds, applying a perspective transform, finding the lane lines and calculating curvature, making a function to find lane lines, curvature of the lines, and the center offset for each frame, and getting the lines of each frame and converting it to video.

I calibrated the camera by finding the chessboard corners on each of the calibration images by using the findChessboardCorners function from openCV to provide parameters for the calibrateCamera function from openCV. The calibrateCamera function would then return vital values to undistort the camera with the undistort function from openCV. The image undistorted correctly as shown here.

Original image:



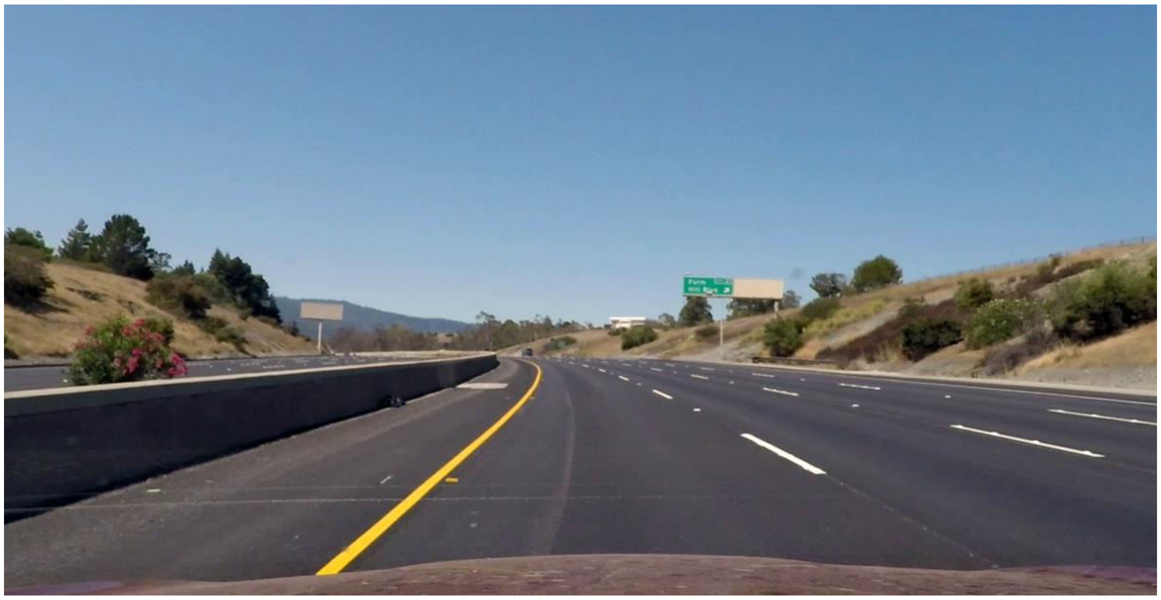
Undistorted Image:



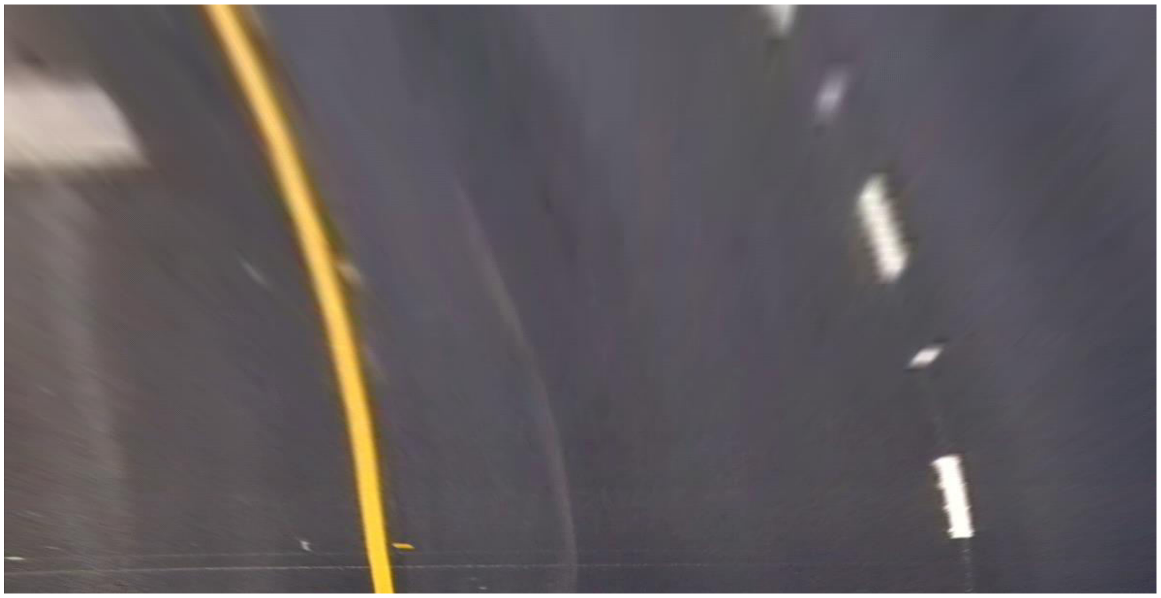
I converted the video to images by grabbing every frame using the VideoCapture function from openCV. I passed every frame into an images array to use later.

Before I got the threshold of the image, I applyed a perspective transform in order to see the lane lines from a bird's-eye view. I did this by grabbing the lane of the original image and then using the getPerspectiveTransform function to transform the trapazoidal shape of the lane lines to a rectangular shape as shown below.

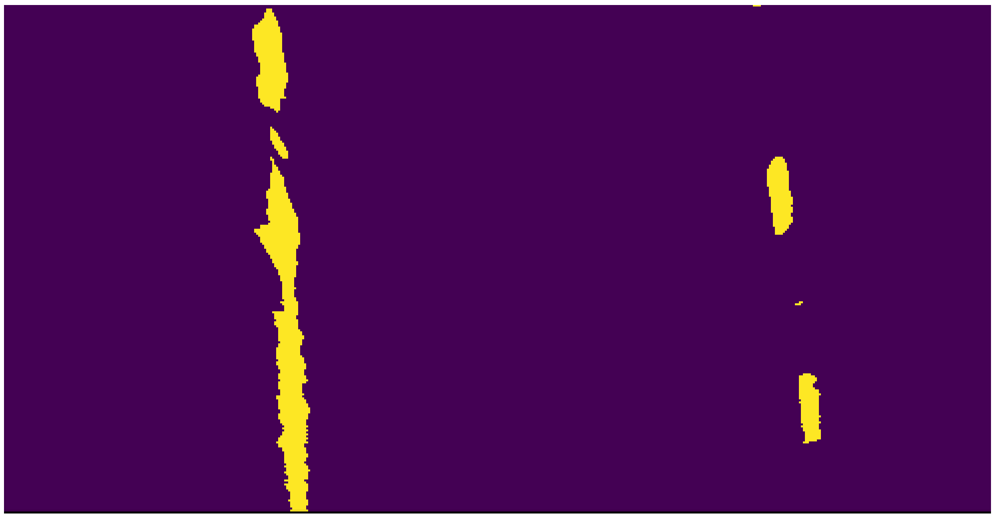
Original Image:



Applied Perspective Transform:

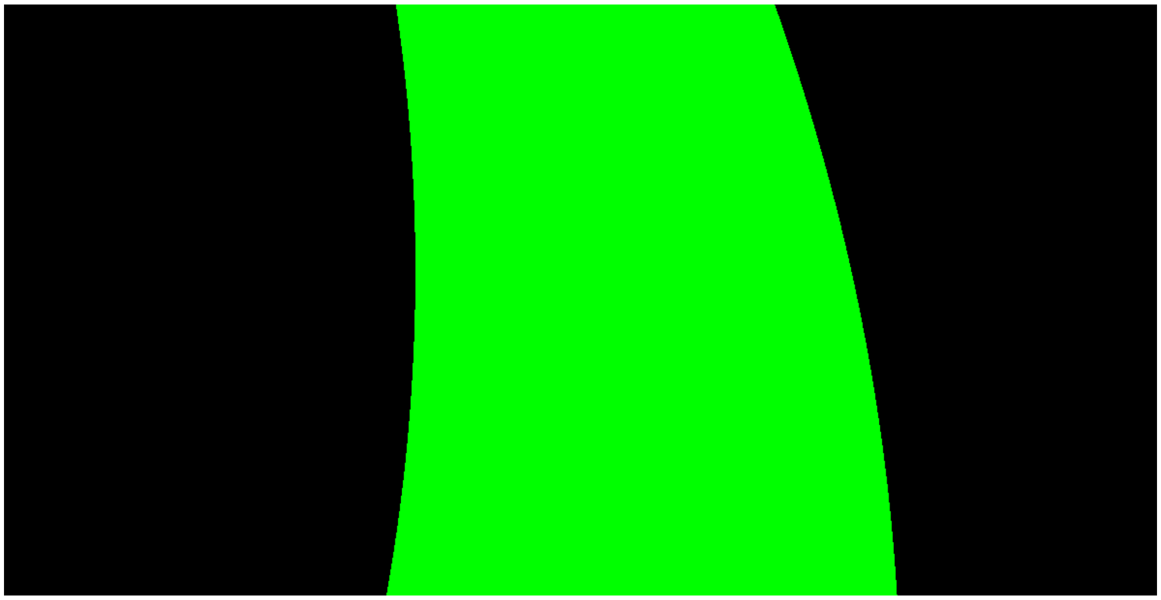


The threshold I used for this project is a a combined color space channel threshold. The threshold consists of the HLS's S channel and HSV's V channel. The combination of these two color channels provided me with a very accurate threshold to detect lane lines as shown here.

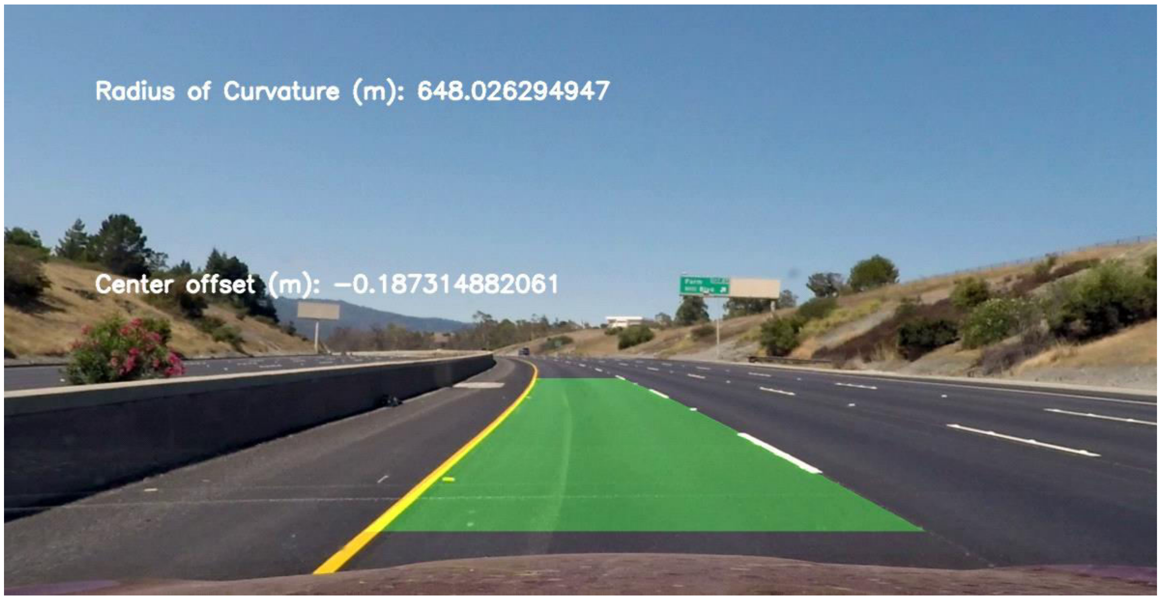


I found lines in the image by finding the peaks in a histogram of the thresholded image. I found this technique to work very accurately. I drew the space between the lines on the original image by overlaying the space on a shaped image with the same dimensions of the warped image. This line image then inverses the perspective transform to overlay over the original image.

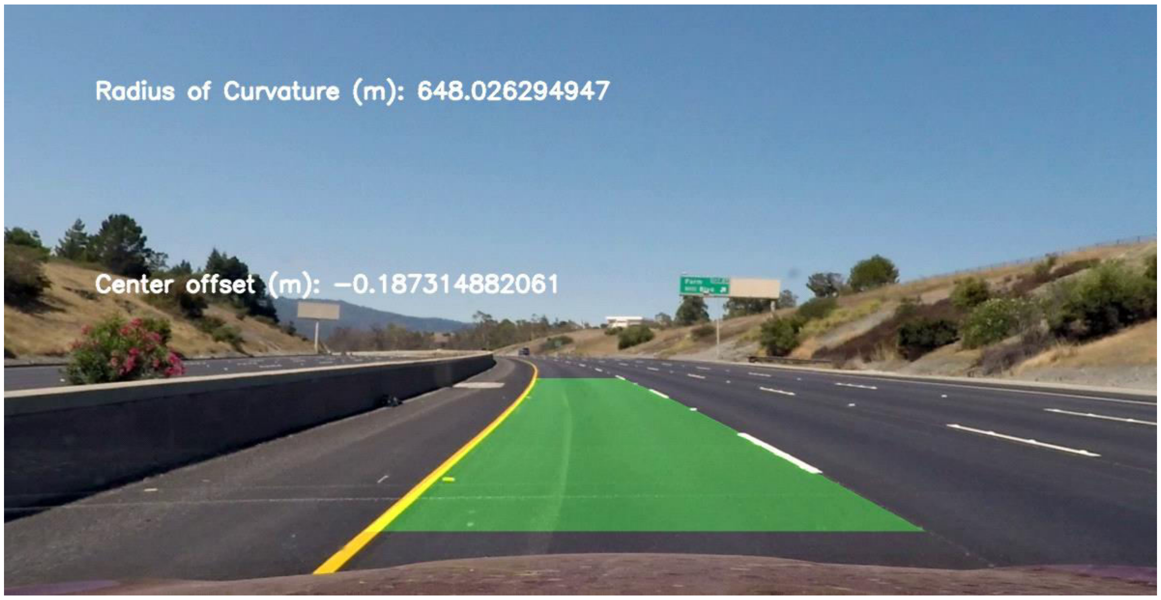
Space between the Lines:



Space between the Lines on Original Image:


Radius of Curvature (m): 648.026294947
Center offset (m): −0.187314882061

I then calculated the curvature of the lines and the center offset by using various formulas. I calculated the center offset by adding the first point of the right line with the first point of the left line, dividng by 2, subtracting it by the center of the image, and then multiplying that value by the pixel to meters conversion. I displayed these values on the image as shown here.


Radius of Curvature (m): 648.026294947
Center offset (m): −0.187314882061

I got the lane lines and calculations of each frame by looping through all the frames in the images array and then calling a draw_lines function. This function finds the lane lines in the image and then calculates and displays the curvature of the lane lines and center offset of the car. After every frame altered, I added the altered frame to the output video by using the VideoWriter function from openCV.

Overall, the pipeline performs very well. However, the pipeline overshoots the lanes when the frame is very dark or very bright. A way to improve the pipeline would to add averaging to the lanes, so that the lanes do not "jump" to another wrong position for a few frames due to the darkness or brightness. Another way is to add more layers to the thresholding, like adding more color channels, so it is more accurate.

All of the figures proving that the functions work is in the jupyter notebook in the "Test each function"   part of the program.

Linked is the output video: https://www.dropbox.com/s/ct8wr03dmi0zlmi/output.avi?dl=0