Advanced Lane Finding¶

Udacity Self Driving Car Engineer Nanodegree - Project 4¶

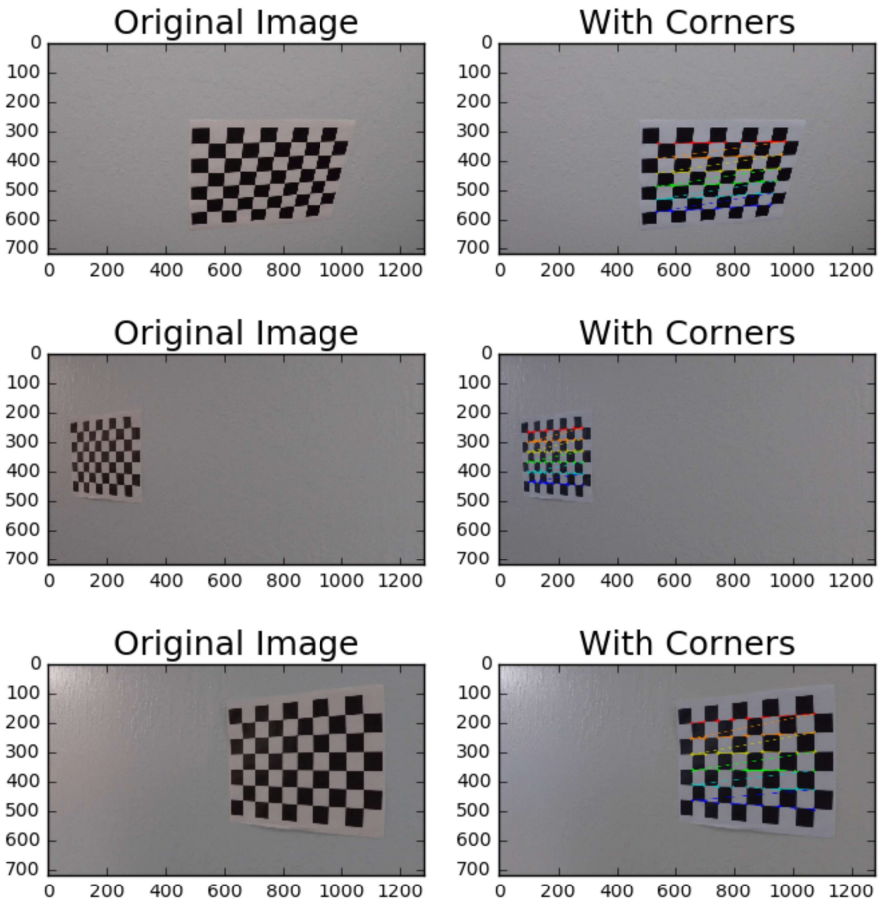
The goal of this project is to develop a pipeline to process a video stream from a forward-facing camera mounted on the front of a car, and output an annotated video which identifies:

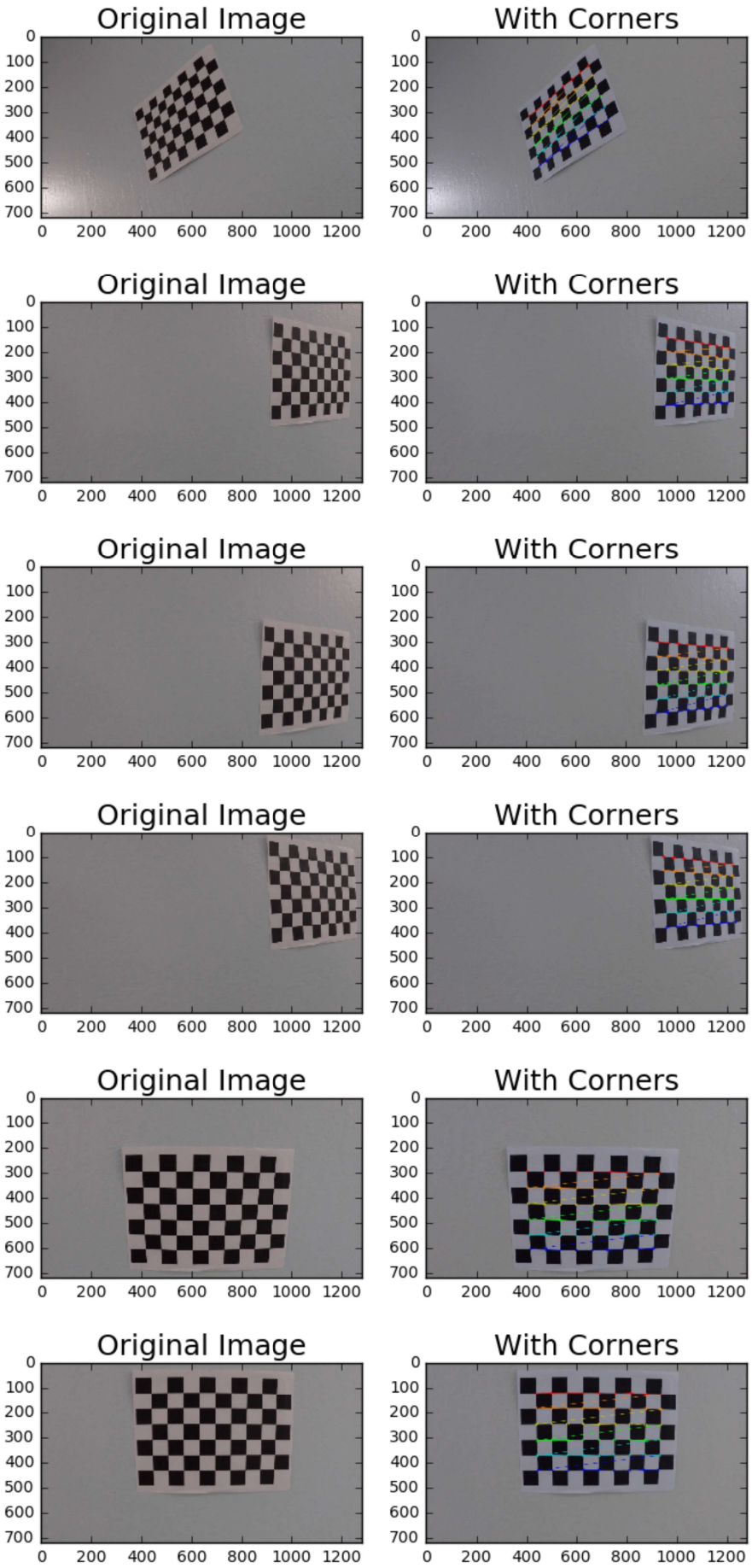
- The positions of the lane lines
- The location of the vehicle relative to the center of the lane
- The radius of curvature of the road

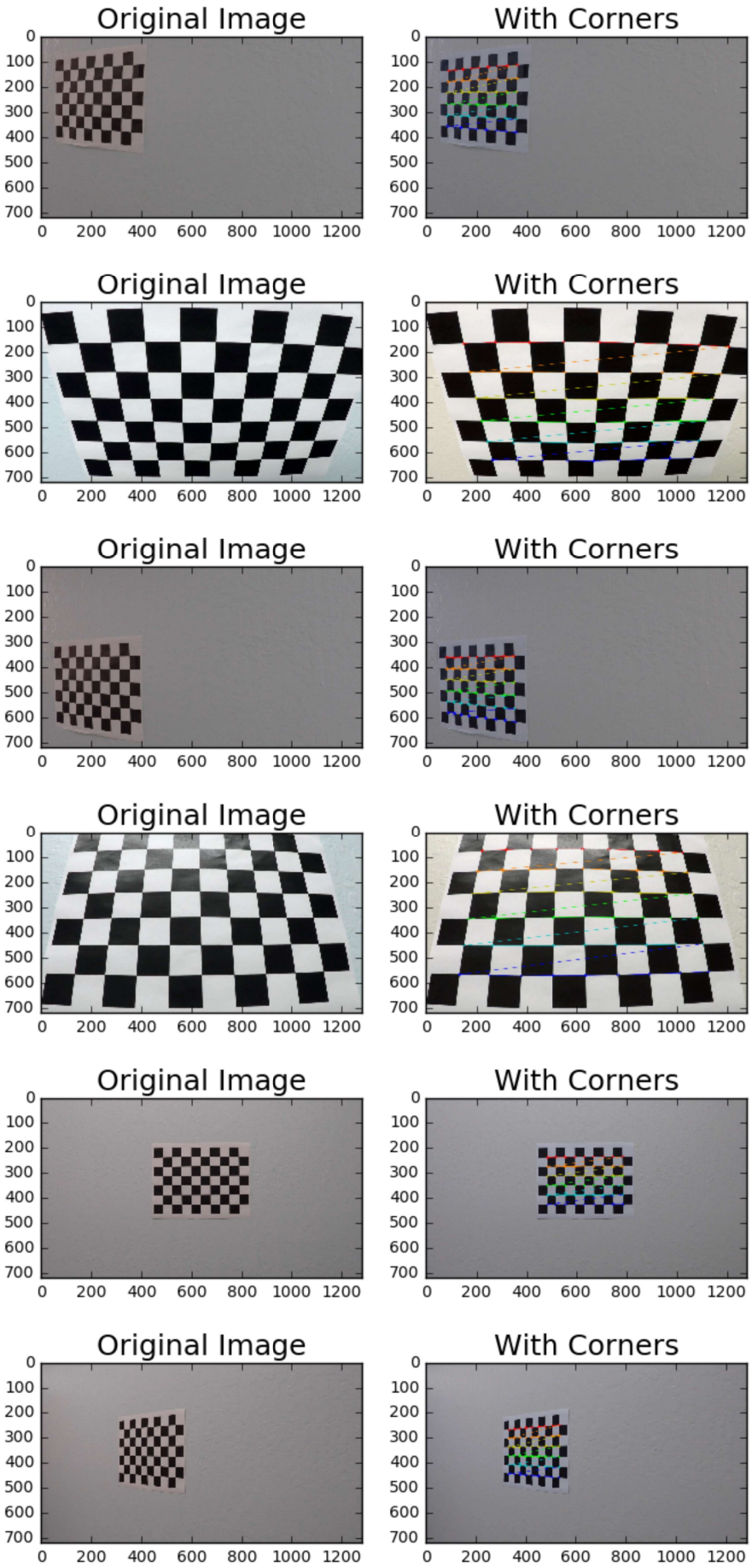
Step 1: Distortion Correction¶

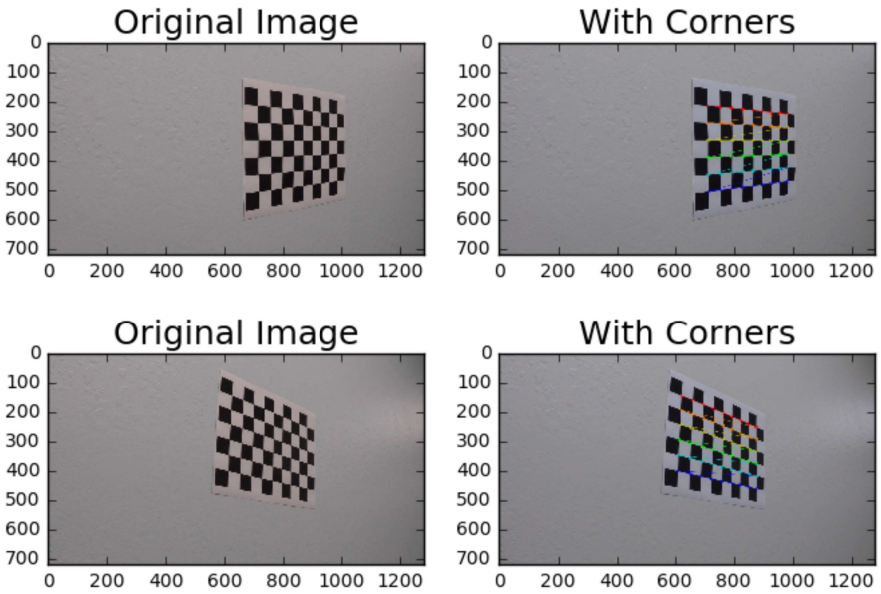
The first step in the project is to remove any distortion from the images by calculating the camera calibration matrix and distortion coefficients using a series of images of a chessboard.

The output is as below:



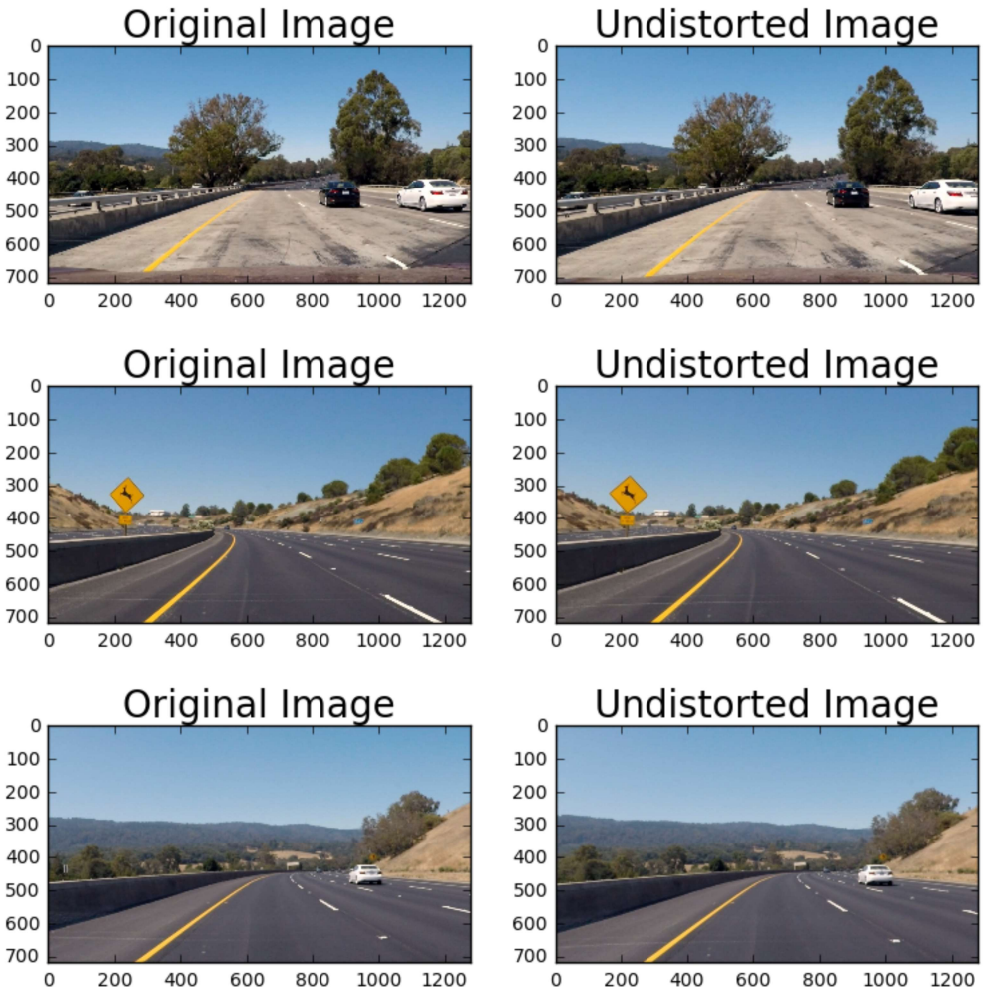

Original Image
With Corners
Original Image
With Corners
Original Image
With Corners
Original Image
With Corners
Original Image
With Corners
Original Image
With Corners

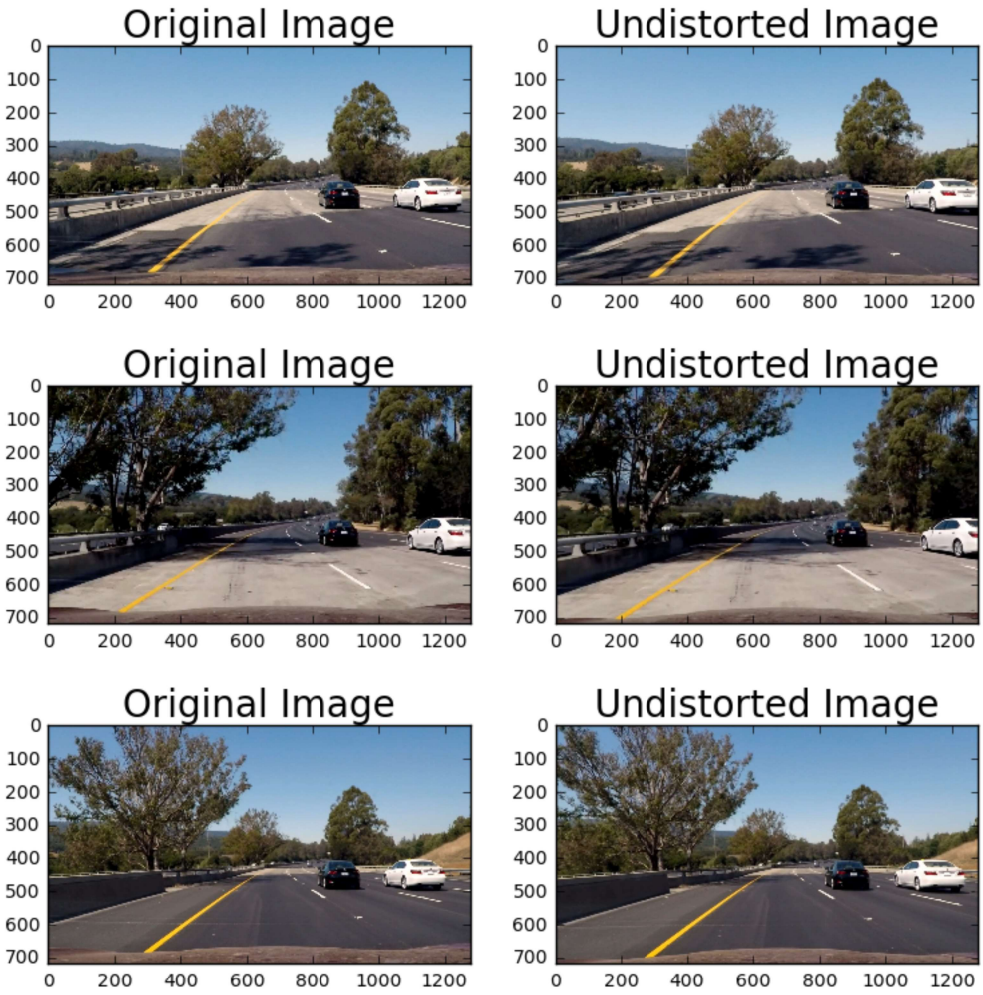

Original Image
With Corners
Original Image
With Corners
Original Image
With Corners
Original Image
With Corners
Original Image
With Corners
Original Image
With Corners



Next I will define a function `undistort()` which uses the calculate camera calibration matrix and distortion coefficients to remove distortion from an image and output the undistorted image.

The undistorted results are shown below:

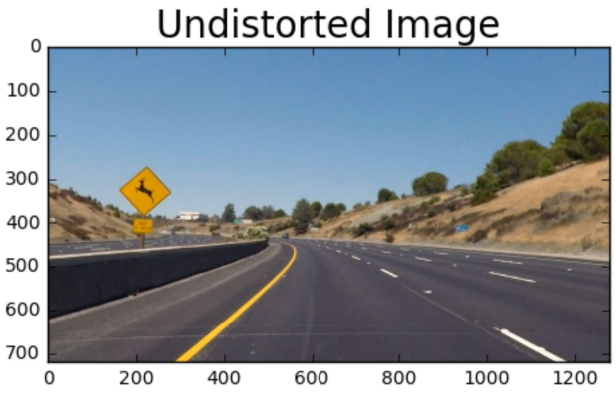


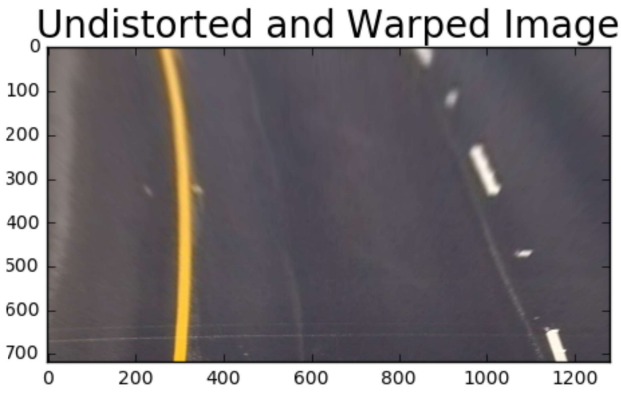


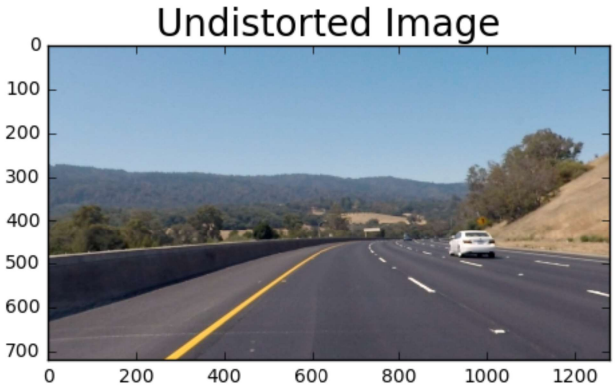
Step 2: Perspective Transform¶

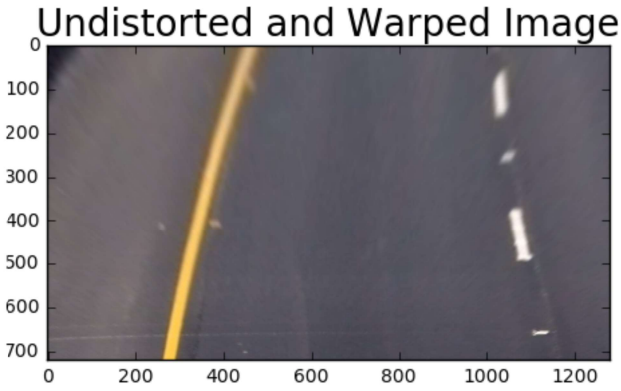
In this step I will define a function `birds_eye()` which transforms the undistorted image to a "birds eye view" of the road which focuses only on the lane lines and displays them in such a way that they appear to be relatively parallel to eachother. This will make it easier later on to fit polynomials to the lane lines and measure the curvature.

The transformed results are shown below:

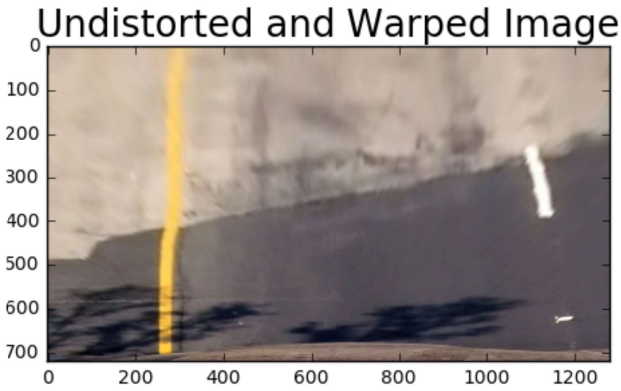

Undistorted Image

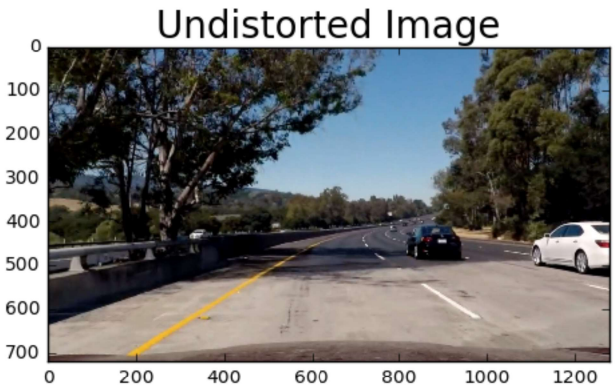

Undistorted and Warped Image

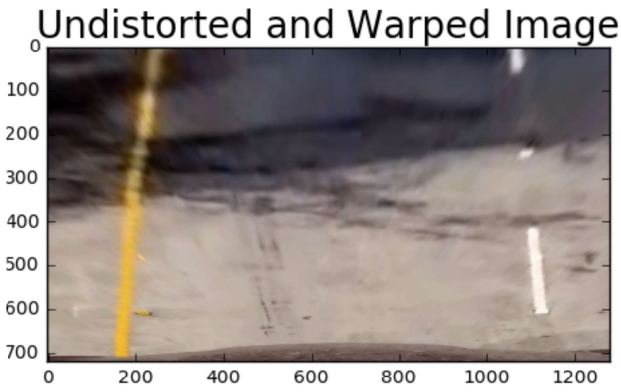

Undistorted Image

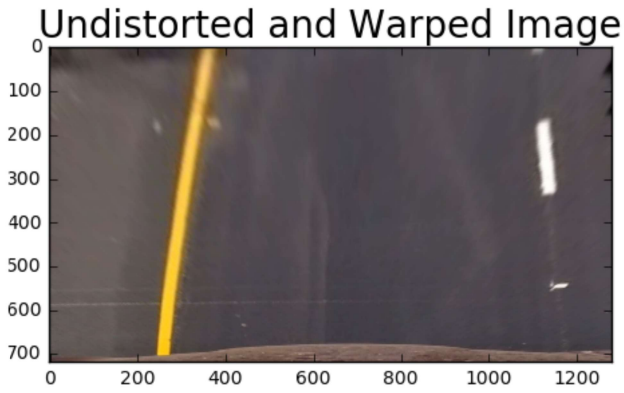

Undistorted and Warped Image


Undistorted Image


Undistorted and Warped Image


Undistorted Image


Undistorted and Warped Image



Step 3: Apply Binary Thresholds¶

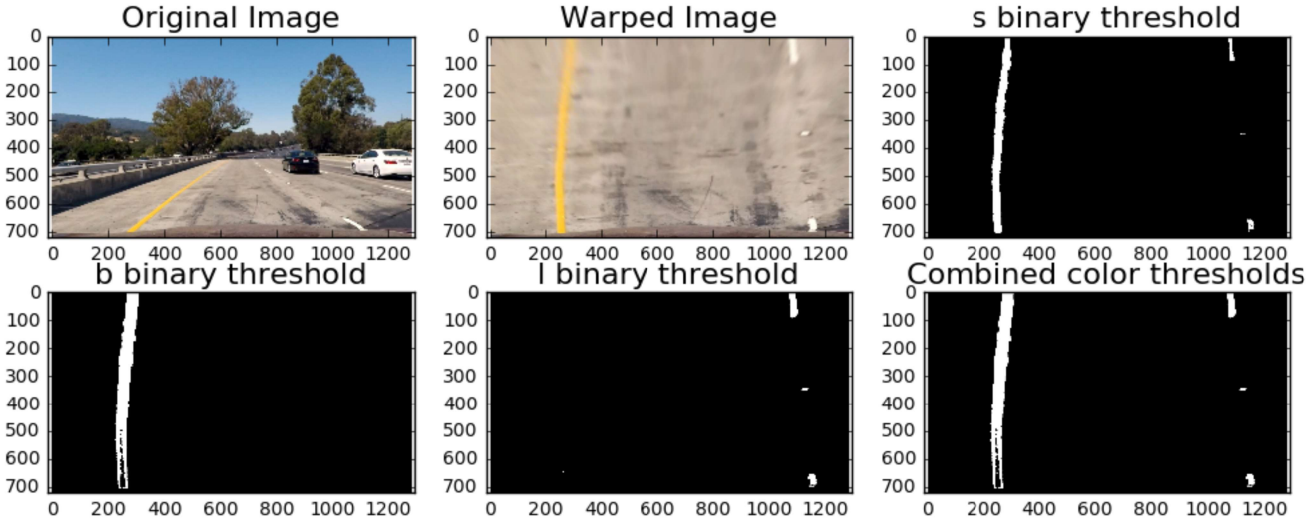
In this step I attempted to convert the warped image to different color spaces and create binary thresholded images which highlight only the lane lines and ignore everything else. I found that the following color channels and thresholds did a good job of identifying the lane lines in the provided test images:

- The S Channel from the HLS color space, with a min threshold of 180 and a max threshold of 255, did a fairly good job of identifying both the white and yellow lane lines, but did not pick up 100% of the pixels in either one, and had a tendency to get distracted by shadows on the road.
- The L Channel from the LUV color space, with a min threshold of 225 and a max threshold of 255, did an almost perfect job of picking up the white lane lines, but completely ignored the yellow lines.
- The B channel from the Lab color space, with a min threshold of 155 and an upper threshold of 200, did a better job than the S channel in identifying the yellow lines, but completely ignored the white lines.

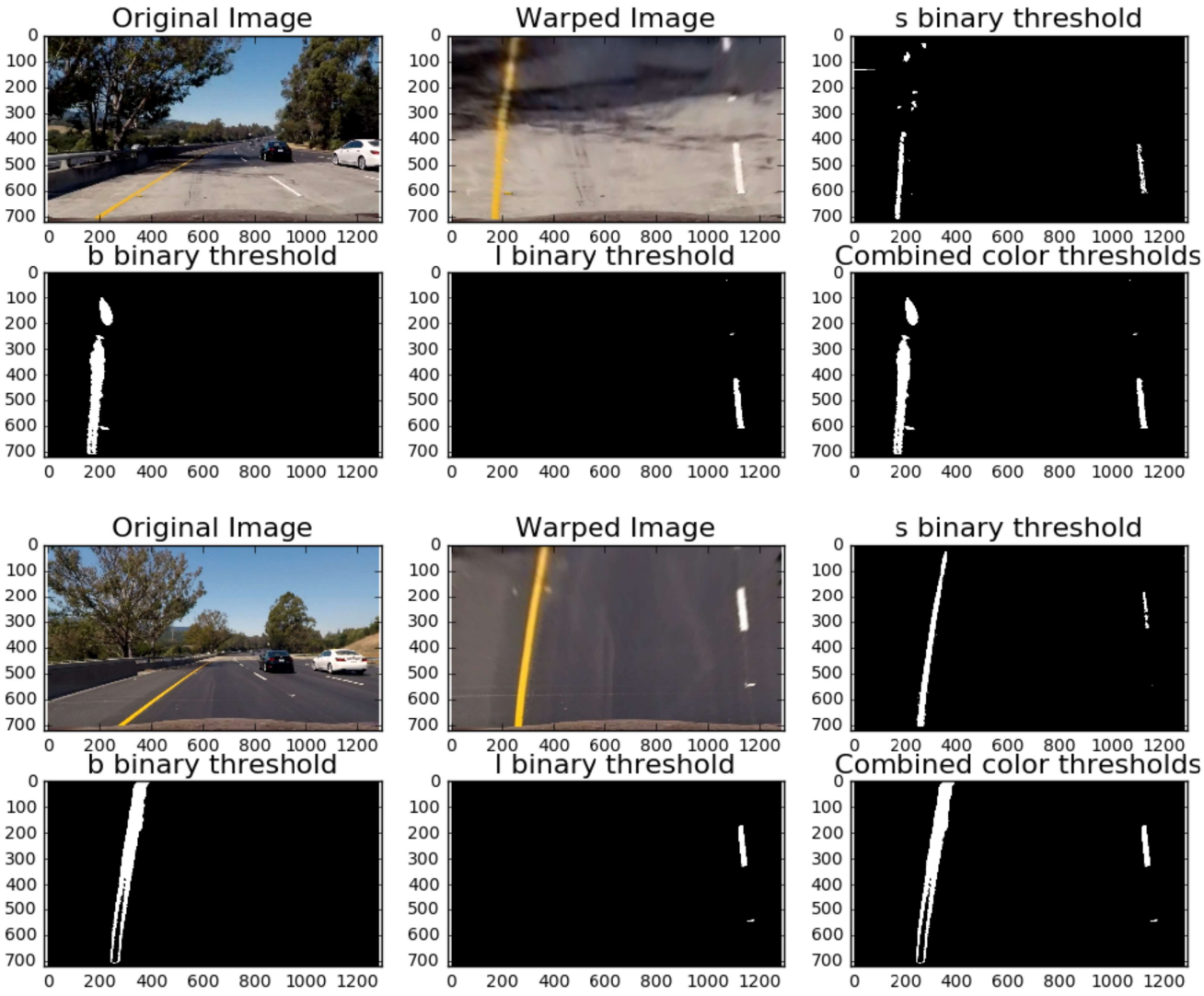
I chose to create a combined binary threshold based on the three above mentioned binary thresholds, to create one combination thresholded image which does a great job of highlighting almost all of the white and yellow lane lines.

Note: The S binary threshold was left out of the final combined binary image and was not used in detecting lane lines because it added extra noise to the binary image and interfered with detecting lane lines accurately.

From the results below we could see the effectiveness of various transformation methods.




Original Image
Warped Image
s binary threshold
b binary threshold
l binary threshold
Combined color thresholds
Original Image
Warped Image
s binary threshold
b binary threshold
l binary threshold
Combined color thresholds
Original Image
Warped Image
s binary threshold
b binary threshold
l binary threshold
Combined color thresholds



Steps 4, 5 and 6: Fitting a polynomial to the lane lines, calculating vehicle position and radius of curvature:¶

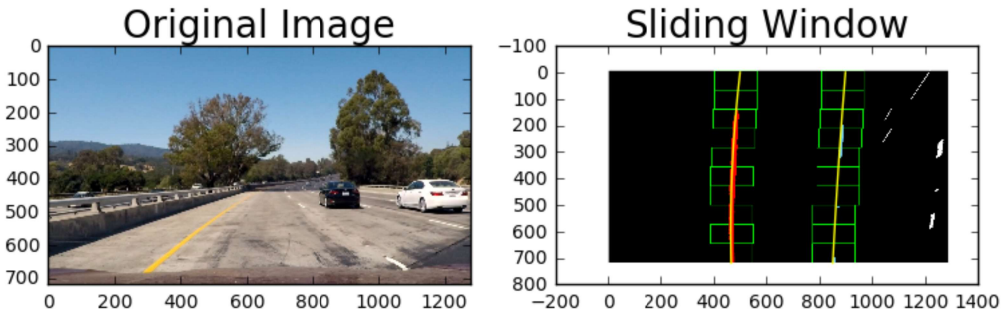
At this point I was able to use the combined binary image to isolate lane line pixels and fit a polynomial to each of the lane lines. The space in between the identified lane lines is filled in to highlight the driveable area in the lane. The position of the vehicle was measured by taking the average of the x intercepts of each line.

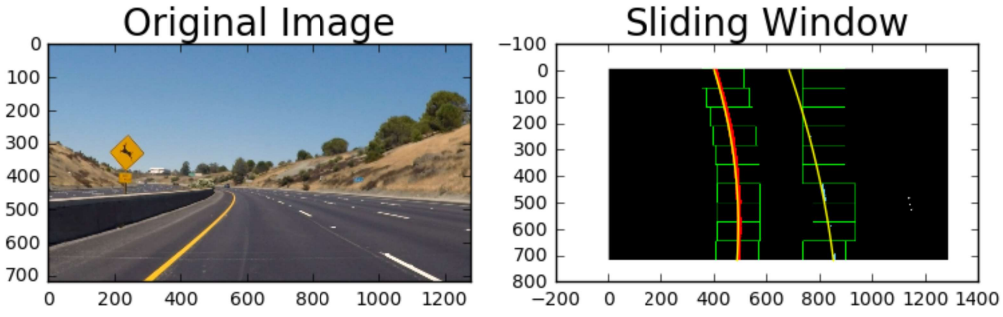
The equation for calculating radius of curvature was discovered at this page: http://www.intmath.com/applications-differentiation/8-radius-curvature.php

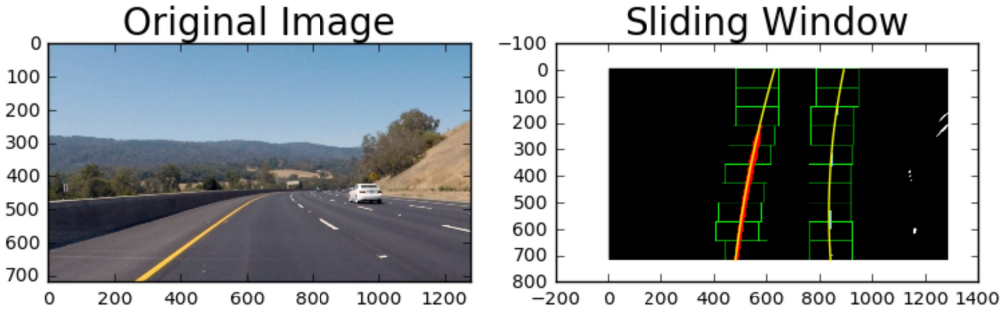
In a function `fill_lane()` , lane lines are detected by identifying peaks in a histogram of the image and detecting nonzero pixels in close proximity to the peaks.

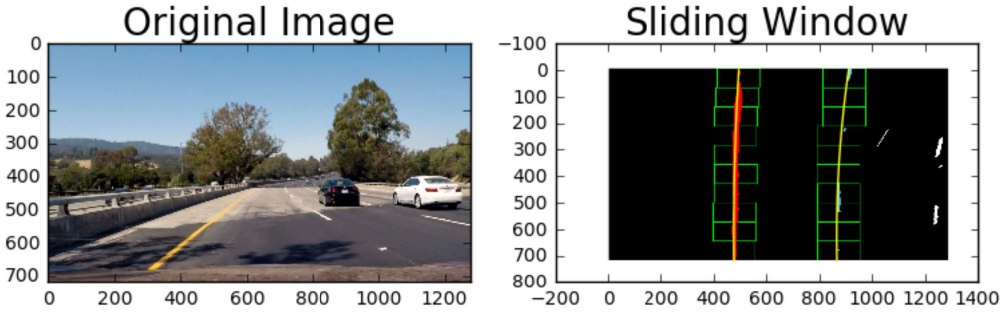
Visualize sliding window polyfit on example image

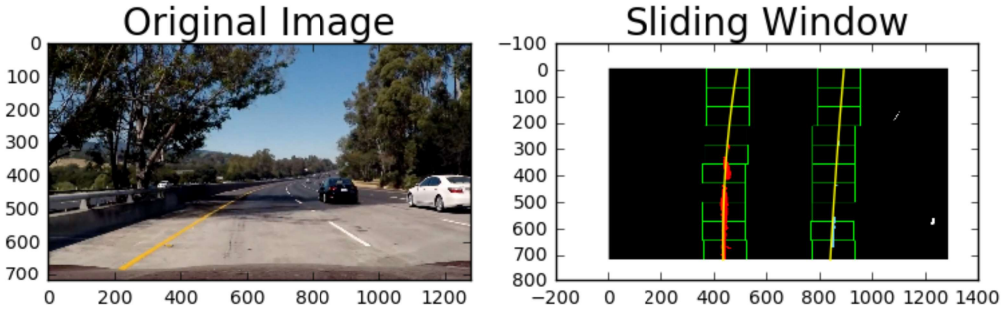
Sliding window Method to Track Lanes¶

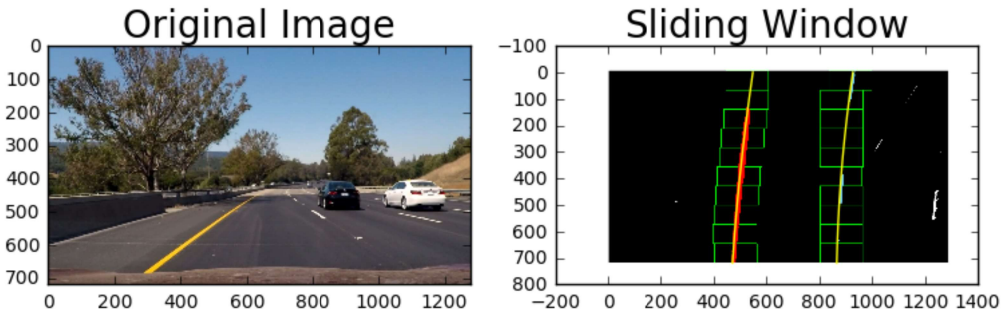

Original Image
Sliding Window


Original Image
Sliding Window


Original Image
Sliding Window

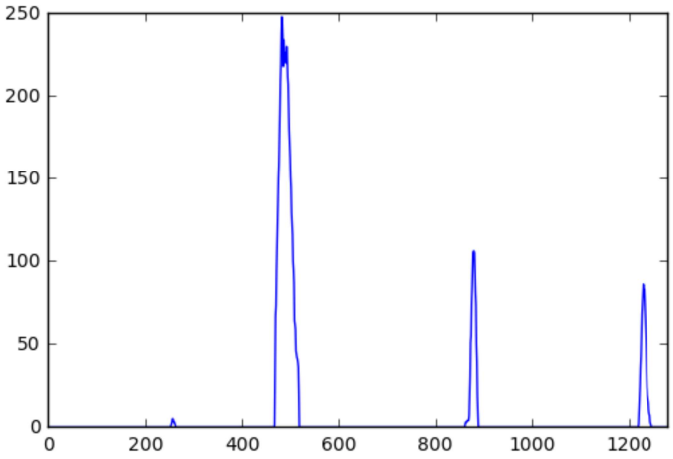

Original Image
Sliding Window


Original Image
Sliding Window



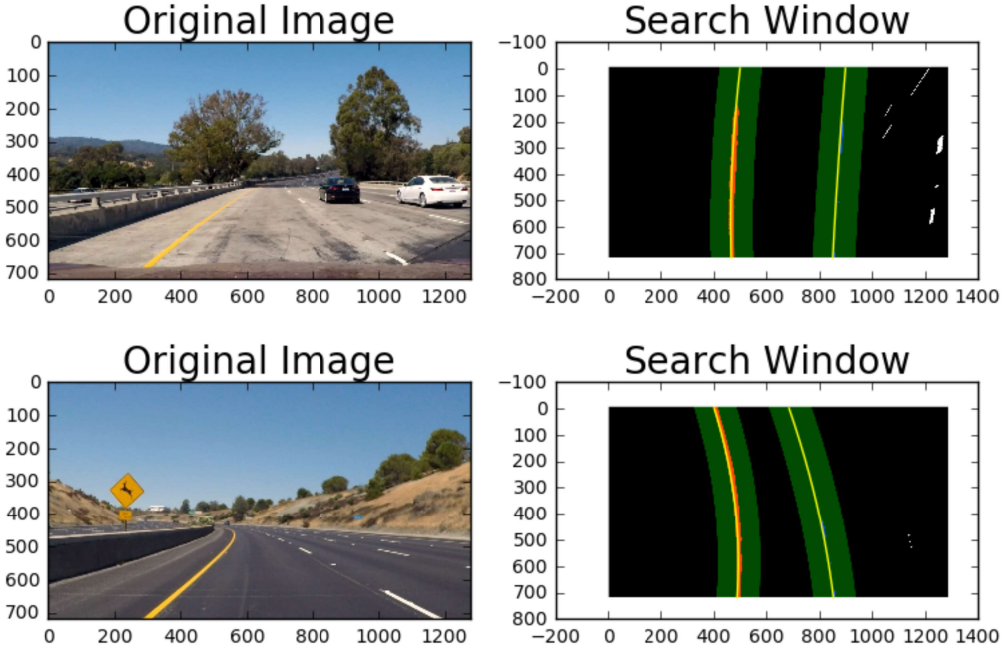
Show one example of histogram¶

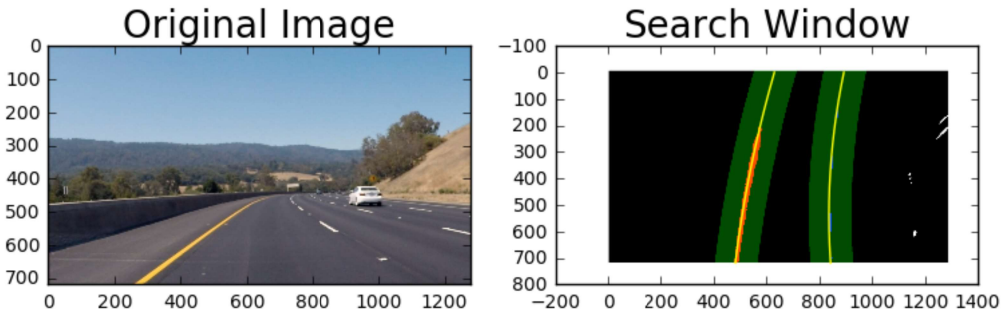
Histogram is a good way to see the candidate lanes in the perspective transformed image.

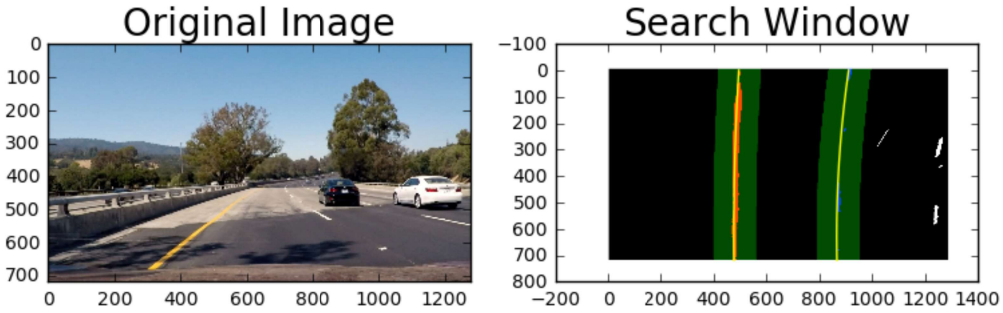


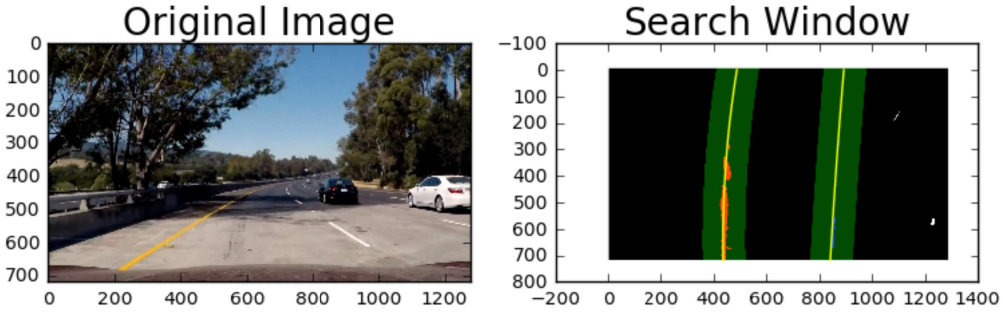
Visualize with search window¶

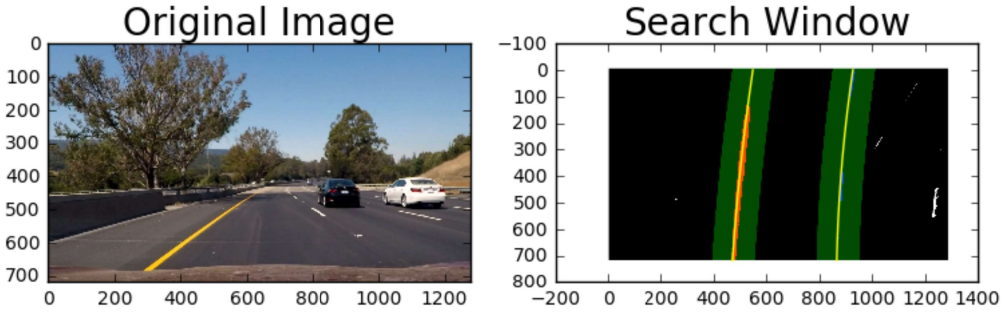
And we can further visualize the search window like in images below:



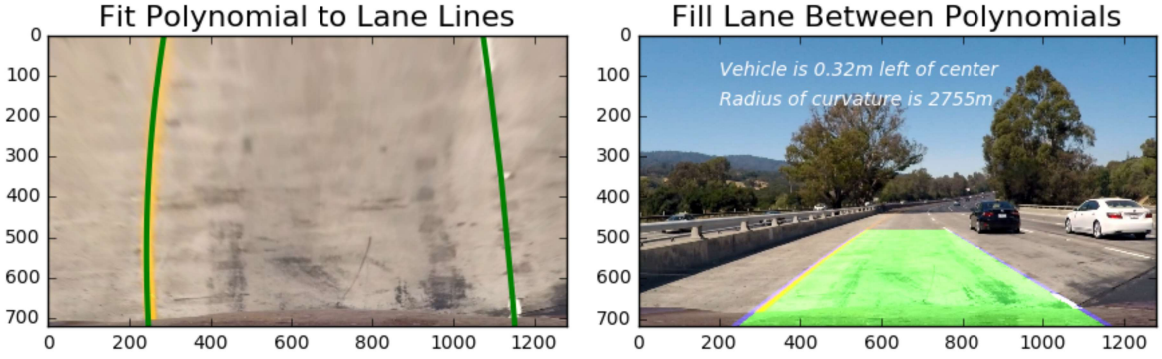


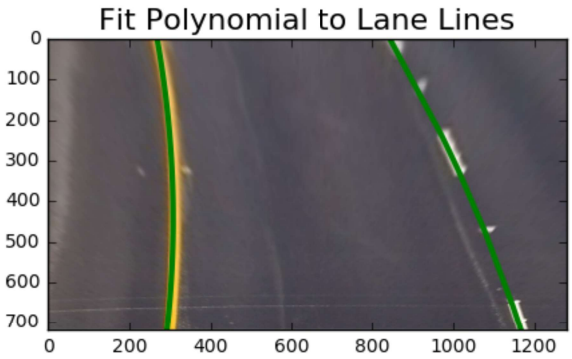


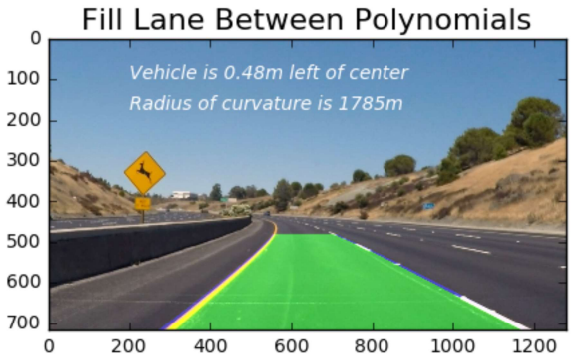


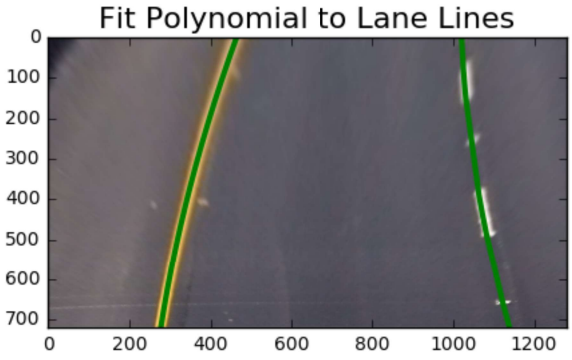


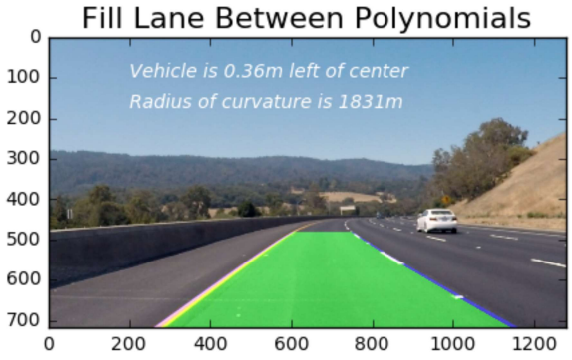
Step 7: Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.¶

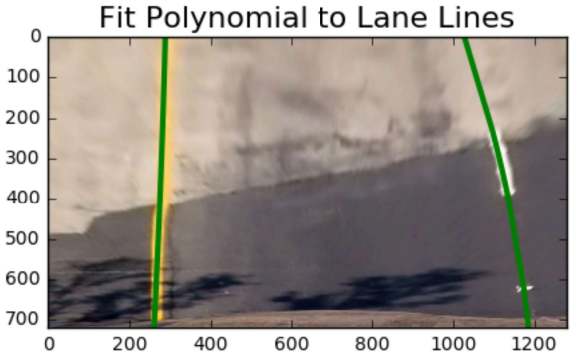


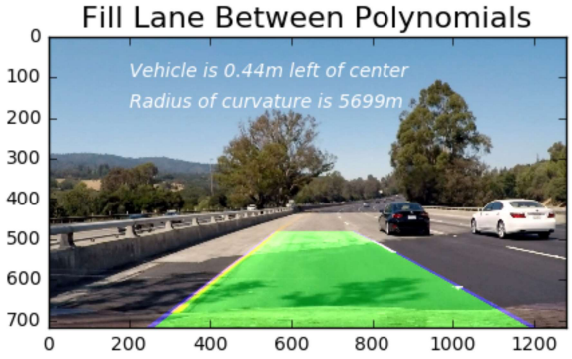

Fit Polynomial to Lane Lines

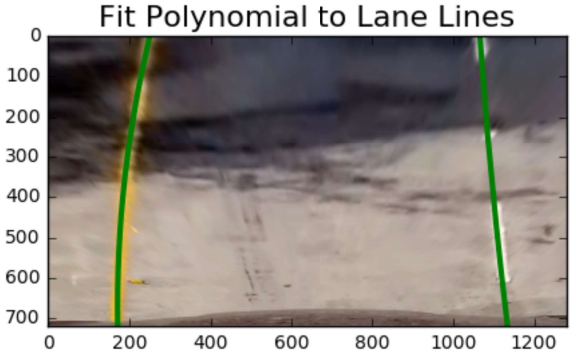

Fill Lane Between Polynomials
Vehicle is 0.48m left of center
Radius of curvature is 1785m

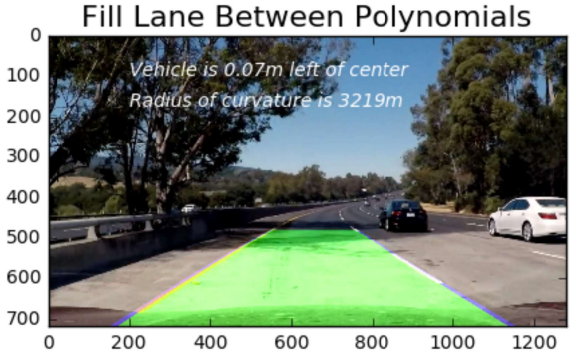

Fit Polynomial to Lane Lines

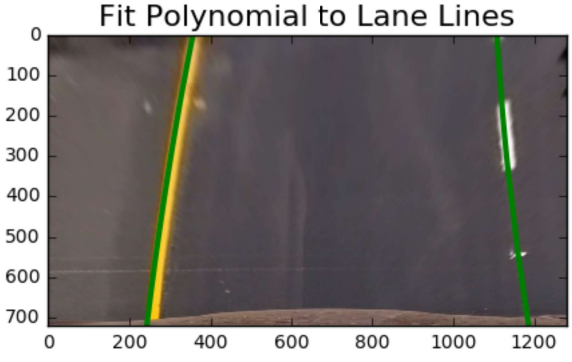

Fill Lane Between Polynomials
Vehicle is 0.36m left of center
Radius of curvature is 1831m

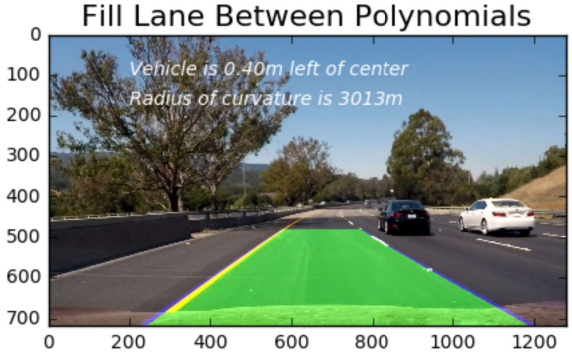

Fit Polynomial to Lane Lines


Fill Lane Between Polynomials
Vehicle is 0.44m left of center
Radius of curvature is 5699m


Fit Polynomial to Lane Lines


Fill Lane Between Polynomials
Vehicle is 0.07m left of center
Radius of curvature is 3219m


Fit Polynomial to Lane Lines


Fill Lane Between Polynomials
Vehicle is 0.40m left of center
Radius of curvature is 3013m

Video Processing Pipeline:¶

First I am going to establish a class Line() for the lines to store attributes about the lane lines from one frame to the next. Inside the class I will define several functions which will be used to detect the pixels belonging to each lane line.

Next I create a function `process_vid()` which processes a video frame by frame and outputs an annotated video with lane lines, radius of curvature and distance from center.

The video processing pipeline is very similar to the `fill_lanes()` function established earlier in the report, except that the video pipeline stores information about the lane lines across frames to average the lane positions and ensure a smooth output which is less impacted by outliers.

The video pipeline also knows whether or not the lane line was detected in the previous frame, and if it was, it only checks for lane pixels in a tight window around the previous polynomial, ensuring a high confidence detection. If the lane line was not detected in the previous frames (and for the first 5 frames of the video) The pipeline performs the same search which was performed in the fill_lanes() function based on identifying peaks in a histogram.