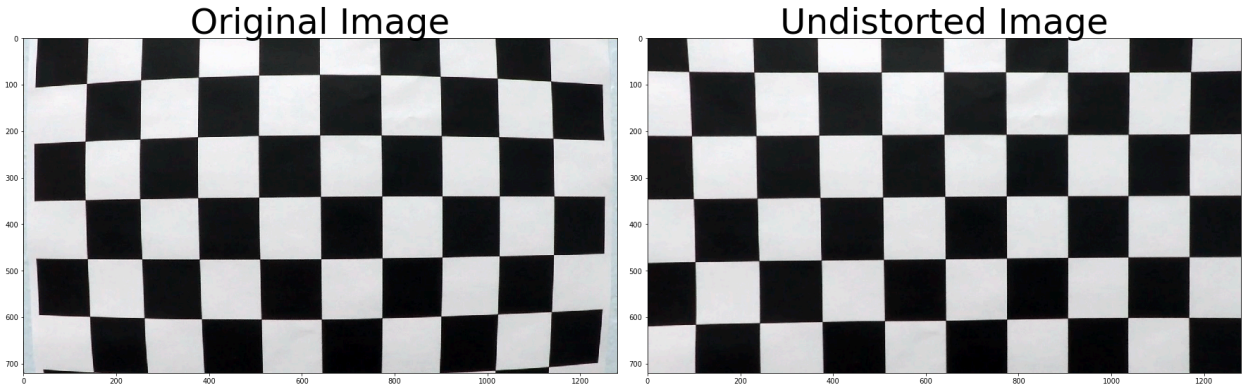
CarND Advanced Lane Detection project:

The goals / steps of this project are the following:

1. *Compute the camera calibration matrix and distortion coefficients given a set of chessboard images. Apply a distortion correction to raw images.*
2. *Use colour transforms, gradients, etc., to create a thresholded binary image.*
3. *Apply a perspective transform to rectify binary image ("birds-eye view").*
4. *Detect lane pixels and fit to find the lane boundary.*
5. *Determine the curvature of the lane and vehicle position with respect to centre.*
6. *Warp the detected lane boundaries back onto the original image.*
7. *Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.*

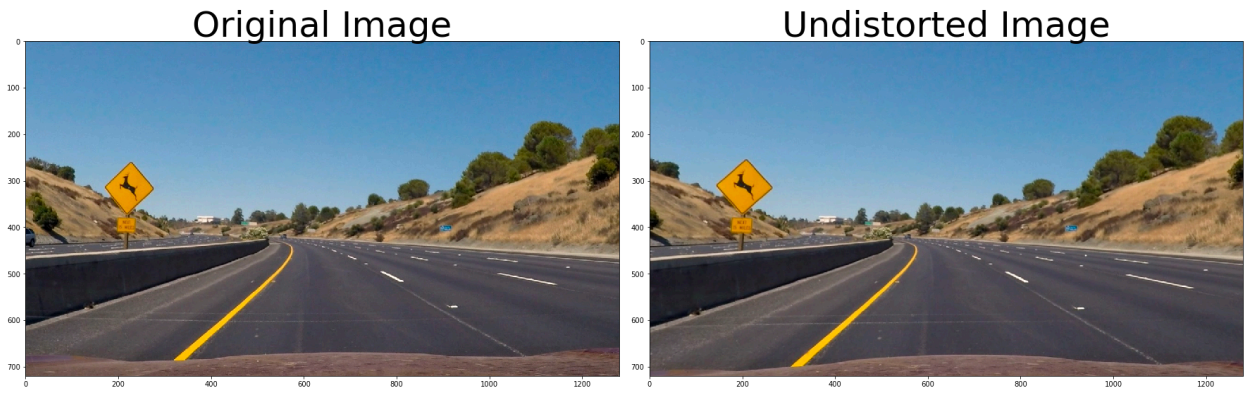
Camera Calibration

I started by preparing "object points", which will be the (x, y, z) 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.



Distortion

Next, I undistorted the image by applying the distortion matrix we calculated from camera calibration .



Create a thresholded binary image

We use threshold to extract gradients that interest us

1. Using Sobel operator

Firstly, I set an absolute threshold on the given axis, so that we can extract the gradient with value that fits in the range. I set this range to be (20, 100) for both X and Y axis.

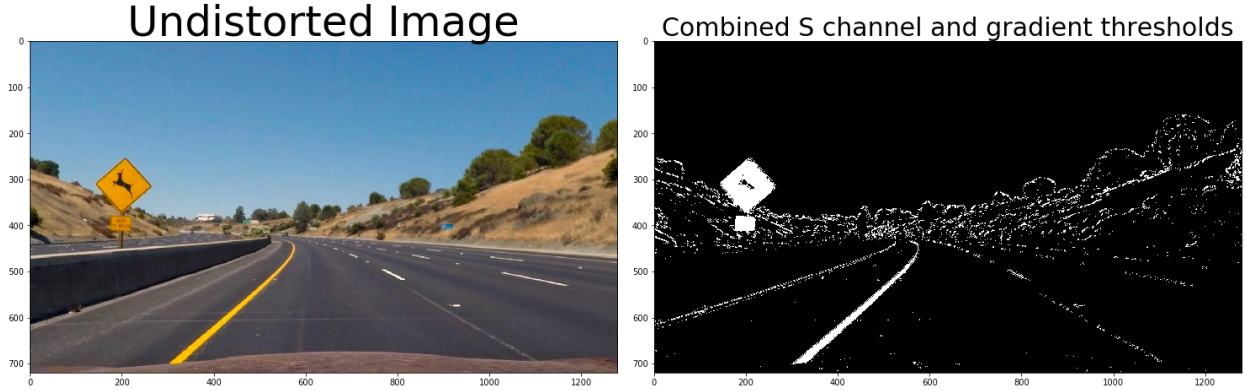
Secondly, I set a magnitude threshold to extract the gradient that has a magnitude within the range. I set this range to be (50, 100).

I set the last threshold which focuses on direction. This threshold helps us extract the gradient that has a direction value that fits our need. The range I used is (0.8, 1.3) with sobel kernel 15.

Finally, we combine the three threshold together, which means that we only take gradient that meets absolute threshold or (magnitude threshold and direction threshold)

2. Using colour shift

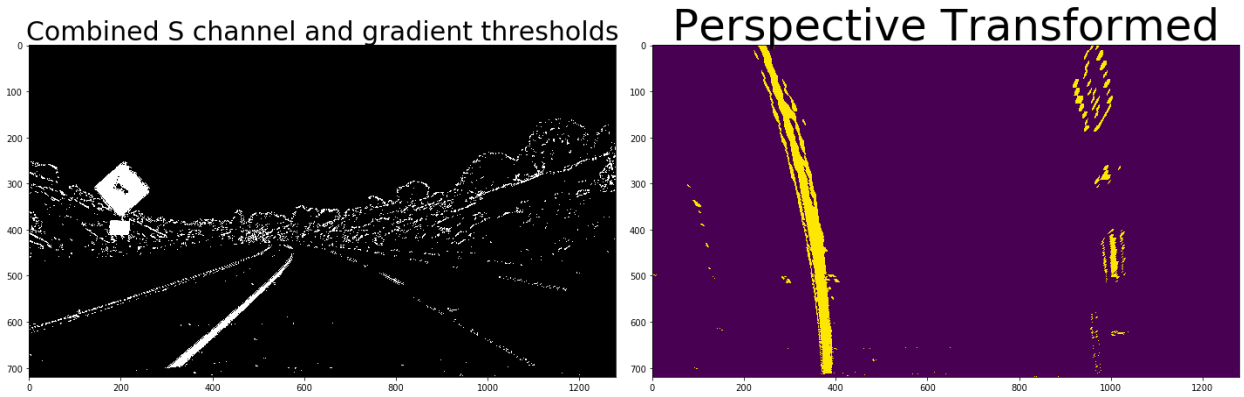
Since saturation channel has the best performance extracting lanes, we use range (170, 255) as a threshold to filter gradients that fit.



Apply a perfective transform to rectify binary image

Next, I use openCV's warpPerspective function to convert the binary image into a birds-eye view, since we are interested in the properties of the lane such as curvature. I defined four points that surround the interested area, as well as another four points to be mapped. To help us change the bird view back to normal later, I also calculated the inverse matrix of the mapping.

Source	Destination
585, 460	320, 0
203, 720	320, 720
1127, 720	960, 720
695, 460	960, 0



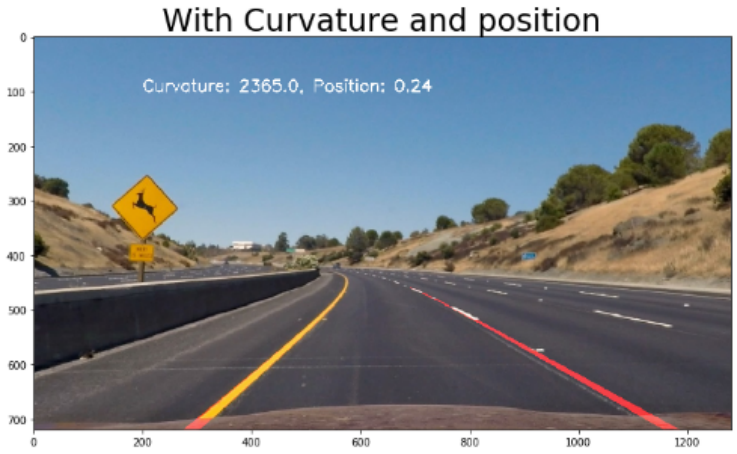
Detect lane pixels and fit to find the lane boundary.

I first take a histogram the bottom half of the image. To find the peak of the left and right halves of the histogram, we loop a number of sliding windows on each half. We define the number of windows to be 9, the margin of the window to be 100, and the minimum pixels can be found in a window to be 50.

I remember the good pixels in the for loop to determine the polynomial. Then, I draw the two curves defined by the polynomials on an empty numpy array that has the same shape as the bird view. I next convert the two curves on the bird view to its original view using the inverse matrix we defined before. Finally, we stack the two curves onto the original picture.

Calculating curvature and position

We calculate the curvature of each curve using the polynomial we calculated. We can also calculate the position of the ego car relative to the lane on the left and lane on the right. We then put the result on the screen.



How to improve my project:

~~The pipeline still shows some instabilities when there are other shapes in the picture, for example shadows of the tree. I need to adjust my parameters or potentially use a memory to remember the continuous gradient to avoid irregular gradients that we are not interested in.~~

How I improved my project:

Thanks to the advices from my reviewer, I spotted my problems and took his suggestions. Here are what I have done to overcome the problem I had:

1. Adjusting perspective transform src dst

Another problem I had was that the bird view is a little bit off. By adjusting the src dst points, I got a better transform.

2. Draw on the area between left lane and right lane

I also drew the area between two lanes in green to indicate what's between the left lane and the right lane

3. Memory system:

I created a numpy array to remember the last 10 left fit polynomials and the right fit polynomials. When a new frame comes in, I first calculate if the left fit polynomial or the right fit polynomial is too off by setting a few thresholds. If it is, we drop this frame, and use the average values from the memory. Otherwise, we pop the oldest polynomial pair out, push the new polynomial pair in, calculate the average, and use the average as the polynomial values we draw on the frame.

4. Using matchShapes function by openCV

Only using the memory system isn't enough to deal with the my lane detection problem when shadows come in. Therefore, I added matchShapes function to compare if the contour drawn on the new frame matches the shape in the previous frame. By setting anther threshold of the value returned by matchShapes function, I found the problem is decently solved.