##Writeup Template
###You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Advanced Lane Finding Project

The goals / steps of this project are the following:

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

[//]: # (Image References)

[image1]: ./examples/undistort_output.png "Undistorted"
[image2]: ./test_images/test1.jpg "Road Transformed"
[image3]: ./examples/binary_combo_example.jpg "Binary Example"
[image4]: ./examples/warped_straight_lines.jpg "Warp Example"
[image5]: ./examples/color_fit_lines.jpg "Fit Visual"
[image6]: ./examples/example_output.jpg "Output"
[video1]: ./project_video.mp4 "Video"

[Rubric](https://review.udacity.com/#!/rubrics/571/view) Points
###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

###Writeup / README

####1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. [Here](https://github.com/udacity/CarND-Advanced-Lane-Lines/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

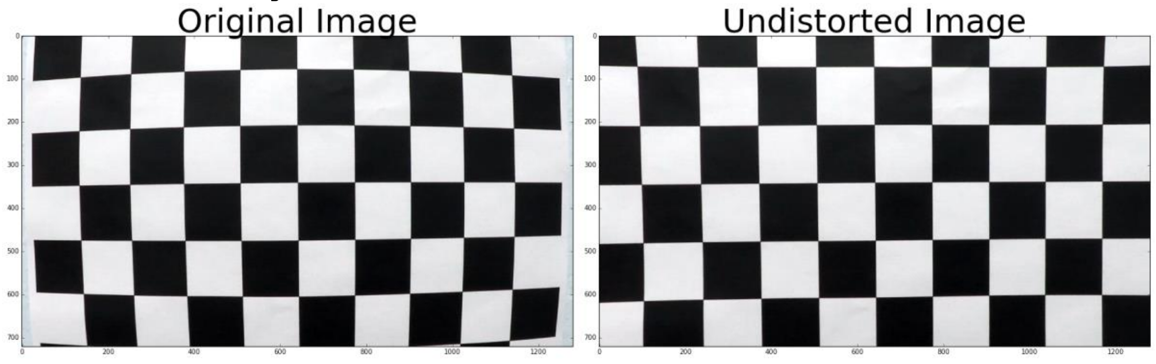
You're reading it!
###Camera Calibration

####1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first and second code cell of the IPython notebook located in "Andvanced_lane_lines.ipynb".

I start 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.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

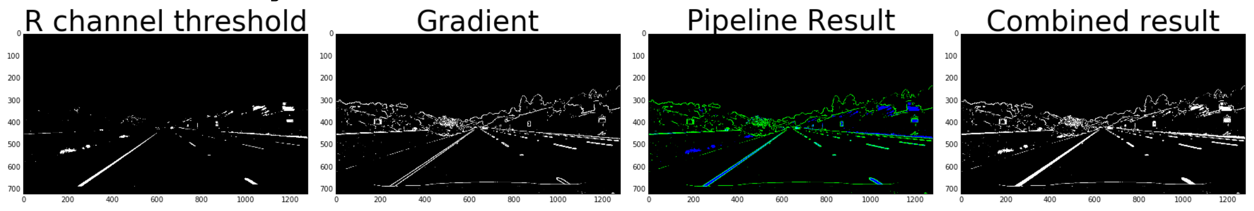
![alt text][image1]


###Pipeline (single images)

####1. Provide an example of a distortion-corrected image.
As mentioned above I have used cv2.undistort() function to get an undistorted image and result is as follows:
![alt text][image2]

####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 and gradient thresholds to generate a binary image (The code for this step is contained in the third code cell of the IPython notebook located in "Andvanced_lane_lines.ipynb".). Here's an example of my output for this step. (note: this is not actually from one of the test images)

![alt text][image3]



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

The code for my perspective transform includes a function called `pers_transform()`, which appears in the 4th code cell of the IPython notebook). The `pers_transform()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose the hardcode the source and destination points in the following manner:

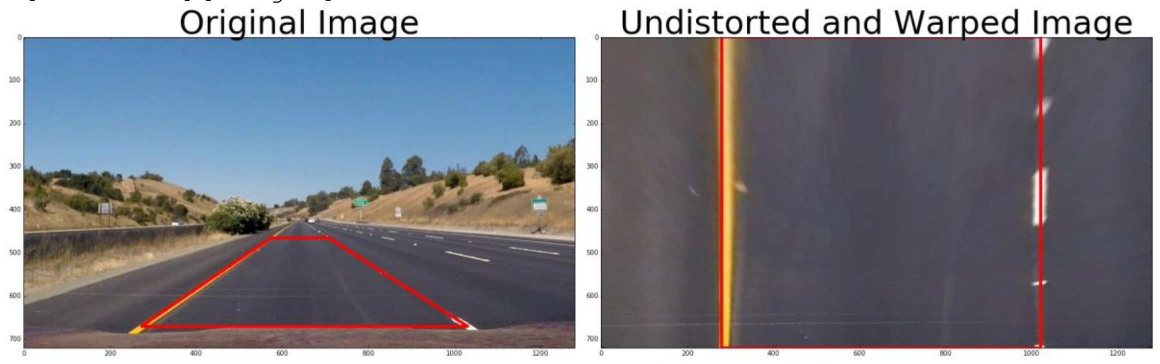
```
src = np.float32([[1030,670],[712,468],[570,468],[270,670]])
dst = np.float32([[1010,720],[1010,0],[280,0],[280,720]])

```
This resulted in the following source and destination points:

Source	Destination
1030, 670	1010, 720
712, 468	1010, 0
570, 468	280, 0
270, 670	280, 720

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

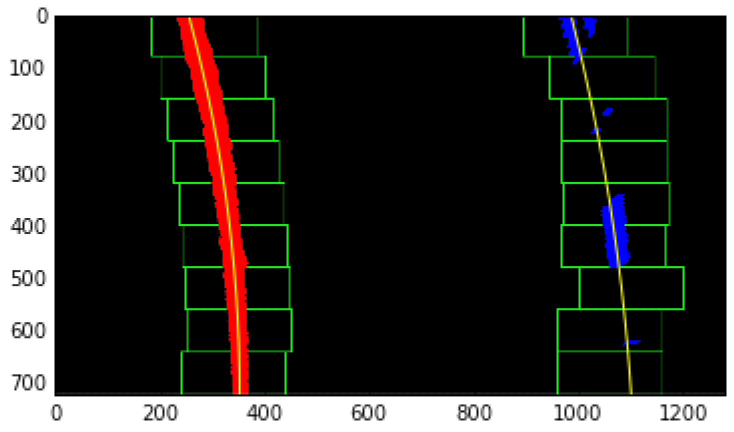
![alt text][image4]



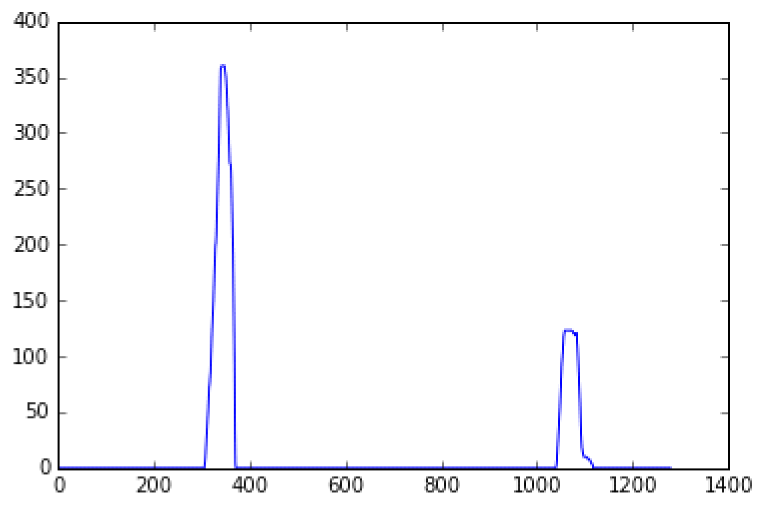
####4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Then I calculated the histogram of the lower half of image and found the peaks for the left and right lines. Then using the sliding search found the lane points. And using the np.polyfit() method, I fit 2nd order polynomial for both the lanes. I did this in the sliding_window() function.My resultant 2nd order polynomial fit is as follows:

![alt text][image5]



Example histogram plot



####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.

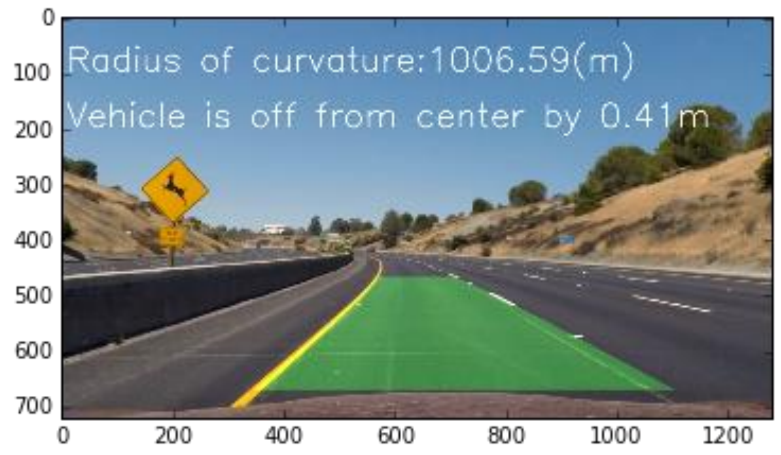
The code for this step is contained in the 8th code cell of the IPython notebook located in "Andvanced_lane_lines.ipynb".

I have calculated the radius of curvature and center offset use the equations discussed in the course and converted them from pixels to meters.

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

I implemented this step in the 9th code cell of the IPython notebook located in "Andvanced_lane_lines.ipynb".
Here is an example of my result on a test image:

![alt text][image6]


Radius of curvature:1006.59(m)
Vehicle is off from center by 0.41m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

###Pipeline (video)

####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!).

output video is stored as project_output.mp4 in this folder.

###Discussion

####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?

Threshold values of R channel and gradient were tested intensively with various values using trial and error bases and choose the current values to maximize the detection.

By replacing the current sliding window search with convolution sliding window search might improve the performance.

Currently my pipeline doesn't work on the challenge videos this need to be improved.

If checks are added to ensure the distance between left and right lanes is constant, and logic to interchange between sliding window and skip sliding window functions might improve my pipeline.