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!

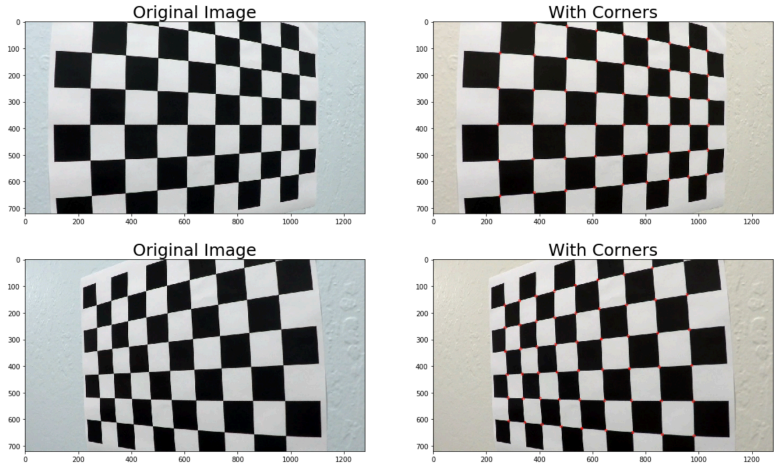
Camera Calibration

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 cells of the IPython notebook.

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.

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:



Detecting Chessboard Corners

Undistorted Images

Pipeline (single images)

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

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

2. 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 `getPerpectiveTransform()`, which appears in jupyter notebook. The `getPerpectiveTransform()` function takes as inputs an image (`image`)and uses given source (`src`) and destination (`dst`) points. I chose to hardcode the source and destination points in the following manner:

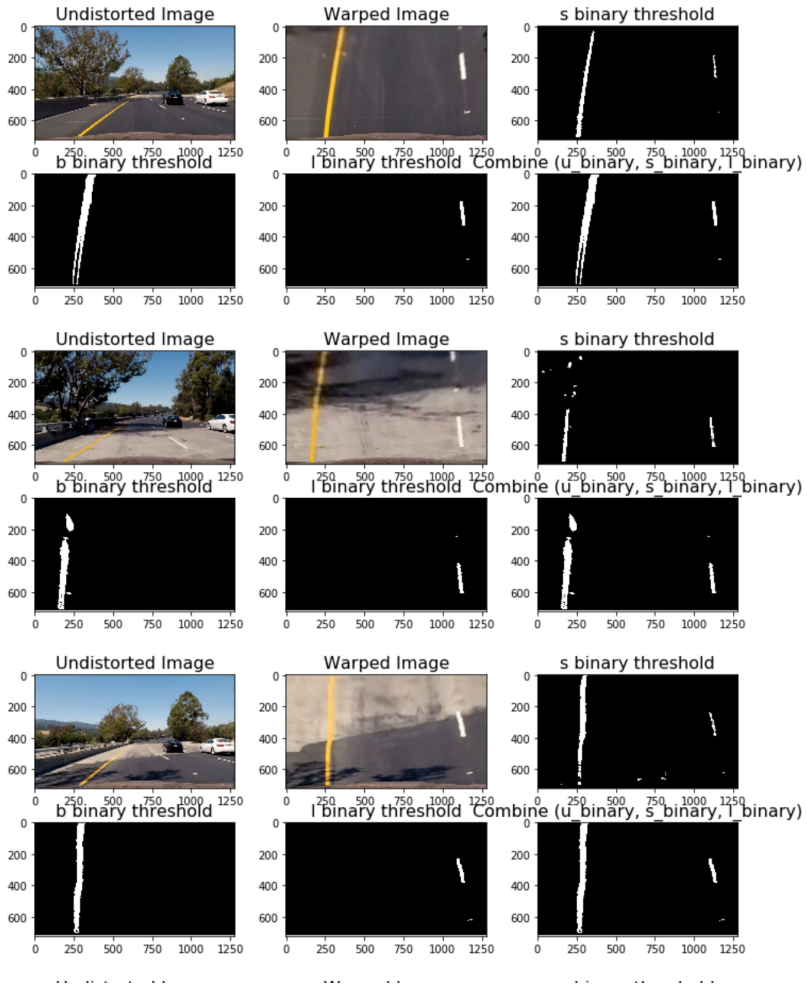
```python

    source = np.float32([[490, 482],[810, 482],
                      [1250, 720],[40, 720]])
    destination = np.float32([[0, 0], [1280, 0],
                     [1250, 720],[40, 720]])
```
This resulted in the following source and destination points:

```
Perspective Transform: [[-7.88079470e-01 -1.62251656e+00  1.16821192e+03]
 [-1.27675648e-15 -2.38410596e+00  1.14913907e+03]
 [-2.27682456e-18 -2.48344371e-03  1.00000000e+00]]

Perspective Transform: [[-7.88079470e-01 -1.62251656e+00  1.16821192e+03]
 [-1.27675648e-15 -2.38410596e+00  1.14913907e+03]
 [-2.27682456e-18 -2.48344371e-03  1.00000000e+00]]

Perspective Transform: [[-7.88079470e-01 -1.62251656e+00  1.16821192e+03]
 [-1.27675648e-15 -2.38410596e+00  1.14913907e+03]
 [-2.27682456e-18 -2.48344371e-03  1.00000000e+00]]
```

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.

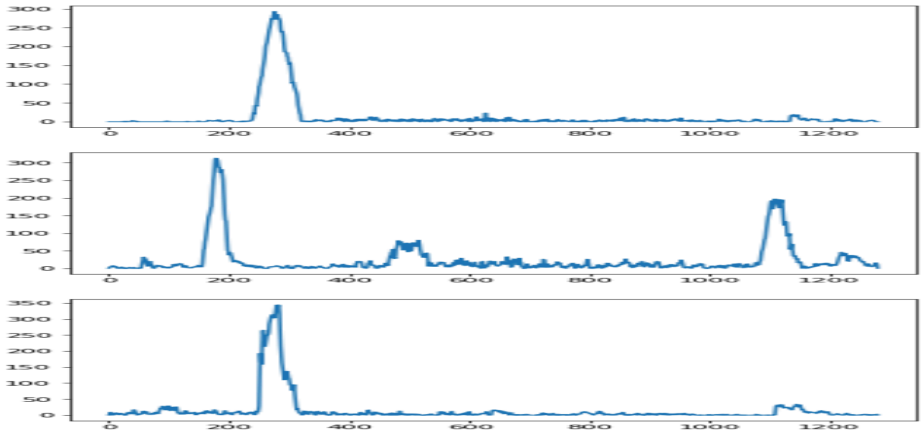
3. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used several combinations of color channel and gradient to threshold the images.I also found that combination of np.dstack((u_binary, s_binary, l_binary))gives me the best result.

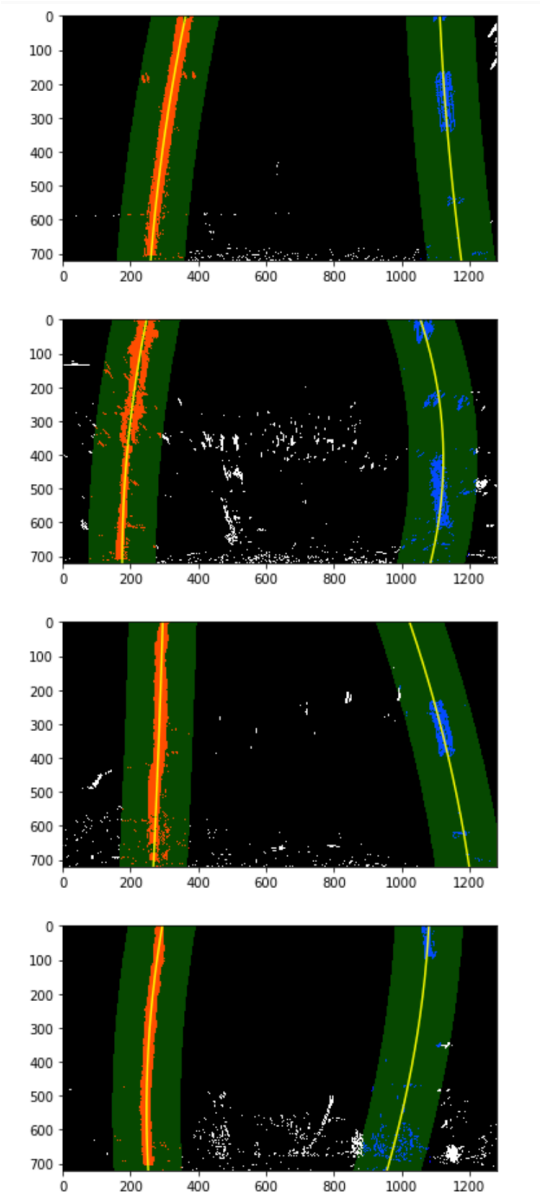


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

In order to find lane lines, I applied the same technique taught in lectures. According to it, I first took histogram along all the columns in the lower half of the image like this.



After that I applied the sliding window technique. As my thresholded binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. I can use that as a starting point for where to search for the lines. From that point, I can use a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame. From the above method once I get the lines, I am searching along the found line positions. The green shaded area shows where I searched for the lines this time. So, once I know where the lines are in one frame of video, I can do a highly targeted search for them in the next frame. This is equivalent to using a customized region of interest for each frame of video, and should help me track the lanes through sharp curves and tricky conditions.

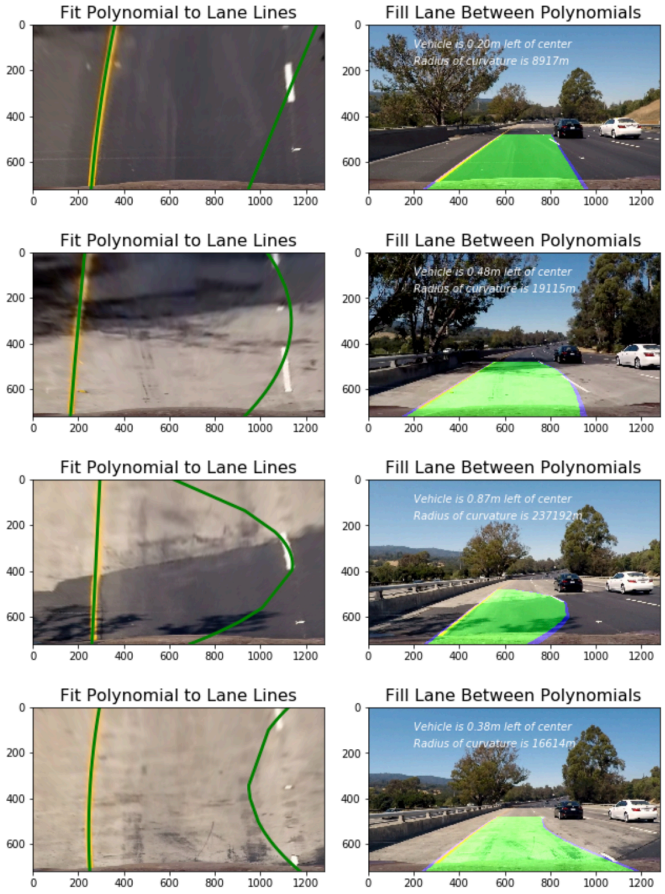


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

For calculating the radius of the curvature, I used the link https://www.intmath.com/applications-differentiation/8-radius-curvature.php given by Udacity where I am finding first and second order derivatives and calculating value of Rcurve. As I've calculated the radius of curvature based on pixel values, so the radius I am reporting is in pixel space, which is not the same as real world space. So I actually need to repeat this calculation after converting our x and y values to real world space. After I wanted to derive a conversion from pixel space to world space in my own images, so I compared it with U.S. regulations that require a minimum lane width of 12 feet or 3.7 meters, and the dashed lane lines are 10 feet or 3 meters long each.

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 using method finalPloyFitAndCurvatureMethod()in my jupyter notebook.



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!).

Link: https://streamable.com/kvjfc

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?

I have found that my model is performing well for project video as well as for challenge video whereas it fails badly on given harder_challenge video.It's because the roads are having the kind of curvatures which I didn't anticipate while working on my model. So I need to further apply some other technique in order to detect lane in mountain roads as given in this challenge video.