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.

To test each one of above steps on an image or video please refer to README.md or README.pdf file. [//]: # (Image References)

Rubric 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 is a template writeup for this project you can use as a guide and a starting point.

You're reading writeup, I also prepare a README file contains information on how to use scripts to run image pipeline and video pipeline.

Camera Calibration

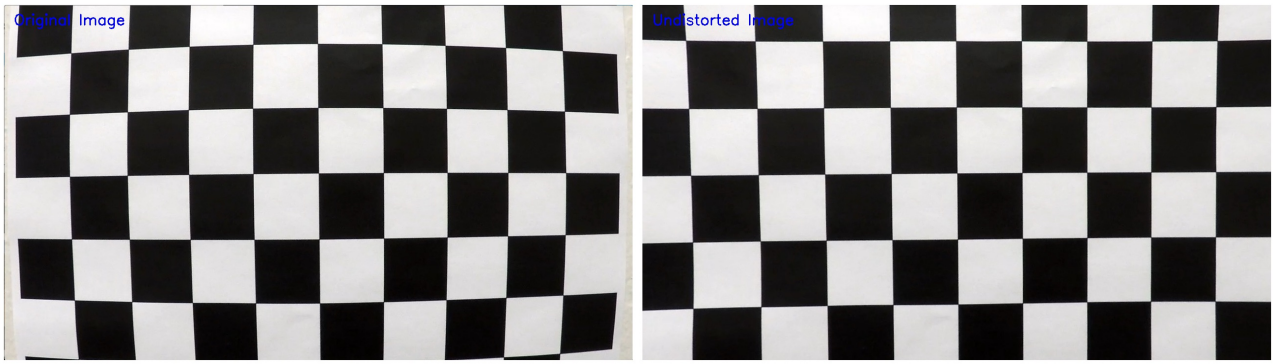
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is in utils/campera_cal.py file in CameraCalibration class. CameraCalibration class has 5 methods.

- compute: (camera_cal.py, Lines 12-30) this method reads all camera calibration images, find chessboard corners and finally compute camera calibration and distortion coefficients using calibrateCamera function. 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.

- save: (camera_cal.py, Lines 31-35) this method saves camera calibration matrix to a pickle file (camera.p).
- load: (camera_cal.py, Lines 36-42) this method loads camera calibration matrix from file.
- undistort: (camera_cal.py, Lines 43-48) this method applys undistortion on an input image using camera calibration and distortion coefficients.



- visualize: (camera_cal.py, Lines 49-61) this method stack original and undistorted images for visualization purposes.

Example:

```
from camera_cal import CameraCalibration
cam_cal = CameraCalibration()
cam_cal.compute(image_list)
cam_cal.save(camera_file)
```

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 used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

The code for this step is in utils/binary_image.py file in BinaryImage class. this class has these methods that compute different threshold mechanisms and build a voting binary image.

- __init__: (binary_image.py, Lines 6-26) constructor of the class generate HLS image, gray image, JET colormapped image, binary images of each threshold mechanism and a voting binary image. In this method I smooth the image using bilateralFilter function.
- _color_thresh: (binary_image.py, Lines 35-43) this method generate a binary image using 3 color thresholds. One from saturation value, another from lightness value and the final one from red value of JET color mapped image. This method adds **2** to the voting binary image if color thresholds condition was met.
- _abs_sobel_thresh: (binary_image.py, Lines 44-50) this method generate a binary image using scaled absolute sobel on x direction. This method adds **2** to the voting binary image if color thresholds condition was met.
- _mag_thresh: (binary_image.py, Lines 51-59) this method generate a binary image using magnitude of sobel edges. This method adds **1** to the voting binary image if color thresholds condition was met.
- _dir_thresh: (binary_image.py, Lines 60-67) this method generate a binary image using direction of sobel edges. This method adds **1** to the voting binary image if color thresholds condition was met.
- get: (binary_image.py, Lines 68-76) this method generate a binary image using vote_binary image with vote_binary > 2 condition.

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 is in utils/perspective_transorm.py file in PerspectiveTransform class. This class has these methods.

- __init__: (perspective_transorm.py, Lines 6-21) constructor of the class insitialized src and dst points and computed transformation matrix, M, and inverse transformation matrix, Minv. I chose the hardcode the source and destination points in the following manner:

```
self.src = np.float32(
```

```
    [[(self.img_size[0] / 2) - 65, self.img_size[1] / 2 + 100],
        [((self.img_size[0] / 6) - 10), self.img_size[1]],
        [(self.img_size[0] * 5 / 6) + 60, self.img_size[1]],
        [(self.img_size[0] / 2 + 65), self.img_size[1] / 2 + 100]])
self.dst = np.float32(
    [[(self.img_size[0] / 5), -20],
        [(self.img_size[0] / 5), self.img_size[1]],
        [(self.img_size[0] * 4 / 5), self.img_size[1]],
        [(self.img_size[0] * 4 / 5), -20]])
```

This resulted in the following source and destination points:

Source	Destination
575, 460	256, -20
203.33, 720	256, 720
1126.66, 720	1024, 720
705, 460	1024, -20

- get: (perspective_transorm.py, Lines 22-25) this method returns perspective transformed image using transformation matrix.
- get_inverse: (perspective_transorm.py, Lines 26-29) this method returns perspective transformed image using inverse transformation matrix.
- visualize:(perspective_transorm.py, Lines 30-47) this method visualize perspective transformation effect by stacking original image and transformed image.

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.

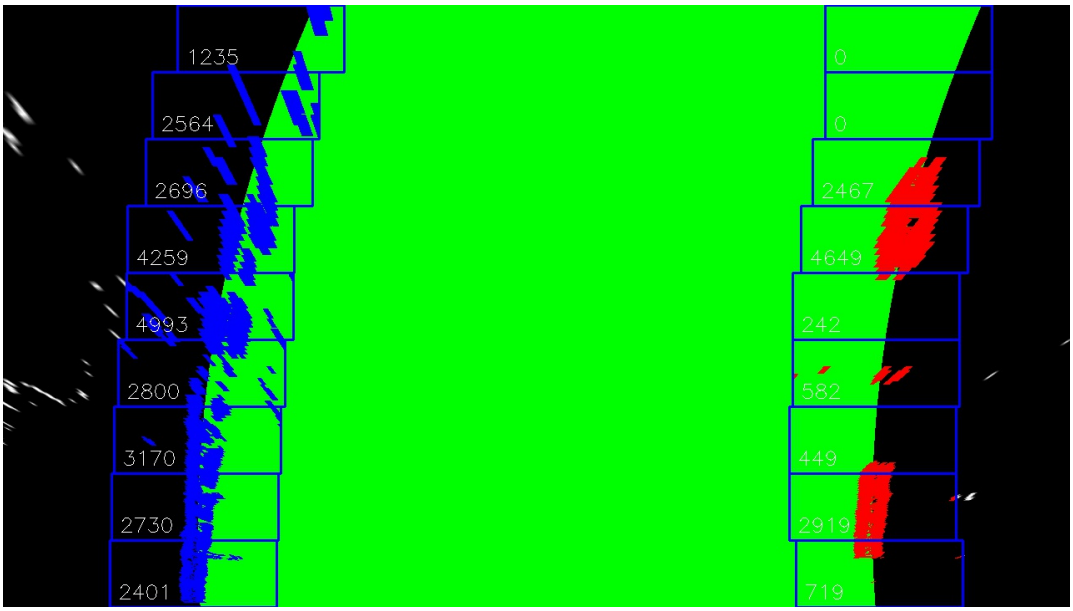
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The code for this step is in utils/lane_finder.py file in LaneFinder class. This class has these methods:

- __init__: (lane_finder.py, Lines 88-113) constructor of this class computed histogram on below half of image, Then find two peaks in left and right section of image using np.argmax function. Also distance

from center of lane was computed.

- slide_window: (lane_finder.py, Lines 114-209) this method slides a window on both left and right starting points computed in constructor. In each slide, index of pixels in window were identified and appended to a list, left_lane_inds list for left lane pixels and right_lane_inds for right lane pixels (lines 146-147). If more than min_pixel pixels were found in a window then next window will be recentered (line 148-154). Then I fit 2nd order polynomial fit on these pixels for left and right lanes (lines 175-183). Finally in this method, left and right curves were computed and stored for later usage.
- visualize: (lane_finder.py, Lines 210-251) this method draws lane lines, slide windows and lane polygon.
- draw_info: (lane_finder.py, Lines 253-271) this method draw left and right curve measures and distance from center on input image.



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.

As I describe in previous section, the code for this section is in LaneFinder class in lane_finder.py file in lines 187 through 208.

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 lines # through # in my code in adv_lane_detection.py in the function lane_visualizer(). Here is an example of my result on a test image:

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!).

Here's a link to my video result

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?

One of problems that I still struggling with that is finding binary image in a better way so I can get better results on the challenge videos. I use a voting technique and get better results than combining threshold binary images.

One thing that I did and get better results in video pipeline was in second order polynomial fit (`lane_finder.py, Lines 43-54). I used pixels on previous frames of video together with pixels on the current frame in order to fit second order polynomial.

Another problem is in challenge video, which lanes not founded properly and it's because of obstacles and road cracks. I think that proper binary image generation is the entry point to solve this problem.