Writeup

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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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 lines 102 through 116 of the file called `Advanced_Lane_Lines.py`).

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, but I didn't save the result, it will also be shown on Pipline results.

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: And the result shown as below:

2. Describe how you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `corners_unwarp()`, which appears in lines 150 through 173 in the file `Advanced_Lane_Lines.py`. The `corners_unwarp()` function takes as inputs an image (`img`), as well as (`mtx`) and (`dist`) points. I chose the hardcode the source and destination points in the following manner:

```
src = np.float32(
    [[(img_size[0] / 2) - 50, img_size[1] / 2 + 100],
    [((img_size[0] / 5) + 15), img_size[1]],
    [(img_size[0] * 5 / 6) + 15, img_size[1]],
    [(img_size[0] / 2 + 60), img_size[1] / 2 + 100]])
dst = np.float32(
    [[(img_size[0] / 4), 0],
    [(img_size[0] / 4), img_size[1]],
    [(img_size[0] * 3 / 4), img_size[1]],
    [(img_size[0] * 3 / 4), 0]])
```

This resulted in the following source and destination points:

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

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 a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines 118 through 147 in `Advanced_Lane_Lines.py`). Here's an example of my output for this step.

Canny Output

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

Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:

Lane

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.

I did this in lines 223 through 228 in my code in `Advanced_Lane_Lines.py`

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 70 through 100 in my code in `Advanced_Lane_Lines.py` in the function `draw_curve()`. Here is an example of my result on a test image:

Final Output

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?

1.The first problem is the method to find the lanes, I didn't use the method in the tutorial which I cann't understand totally. My method code as follows:

```
    histogram = np.sum(img[:int(img.shape[0]/2),offset:img.shape[1]-100], axi
    midpoint = np.int(histogram.shape[0]/2)
    leftx_base = np.argmax(histogram[:midpoint])
    rightx_base = np.argmax(histogram[midpoint:])
    for i in range(img.shape[0]):
        histogram = np.sum(img[i:i+10,offset:img.shape[1]-100], axis=0)
        midpoint = np.int(histogram.shape[0]/2)
        left = histogram[:midpoint]
        right = histogram[midpoint:]
        if(np.amax(left)>9):
            thresh_left=np.where(left > 9)
            if (thresh_left[0][0] > (leftx_base-100)) and (thresh_left[0][-1]
                leftx=np.mean(thresh_left[0])+offset
                left_lanex.append(leftx)
                left_laney.append(i)
                leftx_base = leftx - offset
        if(np.amax(right)>9):
            thresh_right=np.where(right > 9)
            if (thresh_right[0][0] > (rightx_base-100)) and (thresh_right[0][
                rightx=np.mean(thresh_right[0])+midpoint+offset
                right_lanex.append(rightx)
                right_laney.append(i)
                rightx_base = rightx - midpoint - offset
    left_fit = np.polyfit(left_laney, left_lanex, 2)
    right_fit = np.polyfit(right_laney, right_lanex, 2)
```

The main idea is to find the boundary of nozero which probably be lanes boundary, it use 10 pixes as a unit.
I reduced the searching area to avoid distrub.
2.The second problem is to make my pipeline more robust.
I did it by 3 parts:
1) In the finding lane function as shown above, I add some offset and threshold to append to the lane array.
2) I try to reduce the noise by adjust the threshold of x gradient and color channel, but it only can reduce some part of the noises.
3) I add a line_verification function as below:

```
def line_verification(leftfit,rightfit,leftx,rightx):
    if leftline.best_fit is None:
        leftline.best_fit = leftfit
    if leftline.bestx is None:
        leftline.bestx = np.mean(leftx)
    if leftline.current_fit is None:
        leftline.current_fit = leftfit
    if rightline.best_fit is None:
        rightline.best_fit = rightfit
    if rightline.bestx is None:
        rightline.bestx = np.mean(rightx)
    if rightline.current_fit is None:
        rightline.current_fit = rightfit
    leftlinebest = leftline.best_fit[0]*100000+leftline.best_fit[1]*10+leftli
    leftlinecurrent = leftfit[0]*100000+leftfit[1]*10+leftfit[2]/100
    leftlinediff = np.abs(leftlinebest-leftlinecurrent)
    rightlinebest = rightline.best_fit[0]*100000+rightline.best_fit[1]*10+rig
    rightlinecurrent = rightfit[0]*100000+rightfit[1]*10+rightfit[2]/100
    leftlinediff = np.abs(leftlinebest-leftlinecurrent)
    rightlinediff = np.abs(rightlinebest-rightlinecurrent)
    leftxdiff = np.abs((leftline.bestx - np.mean(leftx)))
    rightxdiff = np.abs((rightline.bestx - np.mean(rightx)))
    lefterror.append(leftlinediff)
    righterror.append(rightlinediff)
    if leftlinediff < 120 and leftxdiff < 100:
        leftline.best_fit = (leftline.best_fit + leftfit)/2.0
        leftline.bestx = (leftline.bestx +np.mean(leftx))/2.0
        if  leftlinediff < 10 and leftxdiff < 10:
            leftline.current_fit = leftfit
    if rightlinediff < 120 and rightxdiff < 100:
        rightline.best_fit = (rightline.best_fit + rightfit)/2.0
        rightline.bestx = (rightline.bestx +np.mean(rightx))/2.0
        if rightlinediff < 10 and rightxdiff < 10:
            rightline.current_fit = rightfit

    return leftline.current_fit,rightline.current_fit
```

The main idea is adding threshold of polynomial coefficients by learning exsiting lanes.And it works very well.