Advanced Lane Finding Project
Laura Le

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.

Camera Calibration

1. Compute the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the second code cell of the IPython notebook located in
"./P4_Advance_Lane_Finding.ipynb"
I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

Original Image

Corrected Image

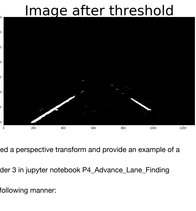
Pipeline (single images)

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:

Original Image

Corrected Image



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 can be found in 5th cell in Jupyter Notebook, under 2nd big header, the function get_threshold(img)
I use 2 kinds of gradient threshold
Along the X axis (since lane line is vertical)
Directional gradient with threshold 30 and 90 degree (since I'm interested in only edges of a particular orientation)

I then apply color threshold as followings:

R & G channel threshold to detect yellow line
L (Lightness) channel threshold so that edge generated due to shadow will not be taken into account
S (Saturation) channel

The image test5.jpg in test_image after applied threshold is shown below

Original Image

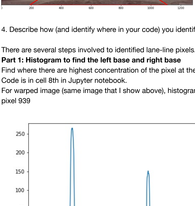
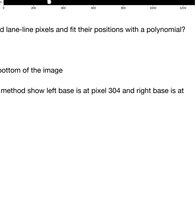


Image after threshold

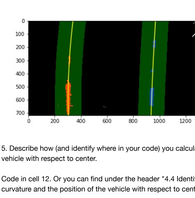


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.
The code for perspective transform is in cell 6th under big header 3 in jupyter notebook P4_Advance_Lane_Finding
I use function cv2.warpPerspective.
I chose the hardcode the source and destination points in the following manner:

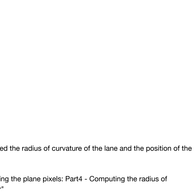
```
# Mannually extract vertices
bottom_left = [190,720]
bottom_right = [1110, 720]
top_left = [570, 470]
top_right = [725, 470]

# Destination points are manually chosen
bottom_left = [320,720]
bottom_right = [920, 720]
top_left = [320, 1]
top_right = [920, 1]
```

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

Original Image

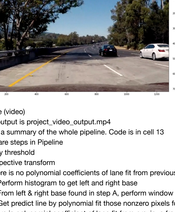


Warped Image

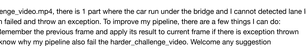


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

There are several steps involved to identified lane-line pixels.
Part 1: Histogram to find the left base and right base
Find where there are highest concentration of the pixel at the bottom of the image
Code is in cell 8th in Jupyter notebook.
For warped image (same image that I show above), histogram method show left base is at pixel 304 and right base is at pixel 939



Part 2: Sliding window and Fit Polynomial
Code is in cell 9th. Or you can look for it under "4.2 Identify the lane pixels: Part 2 - Sliding Window and Fit a Polynomial"
I perform sliding window starting with the left and right base calculated by histogram above
After identify non-zero pixels by sliding window, I perform polynomial fit (2nd order) to get left lane and right lane
The visualization of sliding window is shown below:



Part 3: Search around detected line in previous frames
Code in cell 10. Or you can find it under the header "4.3 Identifying the lan pixels: Part 3 - Search around detected line in previous frames"
As part of pipeline, when we go to next frame in a video, the line should be similar to its preceding frame. So we don't need to go through the first-search i.e. we don't need to perform histogram to search for base and then sliding window. We can use the lanes from the preceding frame and apply a margin to get non-zero pixels. We then use polynomial fitting (2nd order) to get the left & right lane
If there is no non-zero pixels found , we go back to first-search i.e. histogram and sliding window
the green part in the pic below is the area where we search for non-zero pixels

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.

Code in cell 12. Or you can find under the header "4.4 Identifying the plane pixels: Part4 - Computing the radius of curvature and the position of the vehicle with respect to center"

I Use method mentioned in classroom. I need to convert from pixel to metter because curvature calculation is performed in pixel instead of real world meters
The center of the image is the position of the car. The mean of the pixels closest to the car is the center of the lane. The difference between these 2 values is the offset

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

I use the method cv2.fillPoly to plotted back down onto the road the lane area. It's line 147 to line 157 in cell 12, as part of function pipeline(img). To demonstrate the whole pipeline of test5.jpg, you can see cell 13 under header "Apply Pipeline to test image"

Original Image

Processed Image

Pipeline (video)
Video output is project_video_output.mp4
Here is a summary of the whole pipeline. Code is in cell 13
Below are steps in Pipeline
1. Apply threshold
2. Perspective transform
3. If there is no polynomial coefficients of lane fit from previous frame, perform first search of the image as followings:
   A. Perform histogram to get left and right base
   B. From left & right base found in step A, perform window sliding to get all nonzero pixels within the window
   C. Get predict line by polynomial fit those nonzero pixels found in step B
4. If there is polynomial coefficient of lane fit from previous frame, find the nonzero pixels within a margin of the predicted lanes from previous framework and run a polynomial fit of these non-zero pixels tog et predicted line
5. Get Cuevature radius and center offset
6. Inverse transform perspective of the frame

Discussion

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?

In challenge_video.mp4, there is 1 part where the car run under the bridge and I cannot detected lane line. My pipeline function failed and throw an exception. To improve my pipeline, there are a few things I can do:
1. Remember the previous frame and apply its result to current frame if there is exception thrown

I don't know why my pipeline also fail the harder_challenge_video. Welcome any suggestion