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.

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 (17-68) of the file called (*camera_calibration.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 (9, 6) 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 (9, 6) 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:

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 by using the camera calibration and distortion coefficients from the camera calibration file and applied 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.

I used a combination of colors thresholds to generate a binary image in lines (35 to 55) in file called (detect_lanes.py). Here's an example of my output for this step.

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 warper(), which appears in lines 65 through 80 in the file (detect_lanes.py). The warper() function takes as inputs an image (img), I chose the hardcode the source and destination points in the following manner:

```
src = np.float32(
    [[(img_size[0] / 2) - 55, img_size[1] / 2 + 100],
    [((img_size[0] / 6) - 10), img_size[1]],
    [(img_size[0] * 5 / 6) + 60, img_size[1]],
    [(img_size[0] / 2 + 55), 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.

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

I did this in lines 95 through 141 in my code in (detect_lanes.py). I first take a histogram along all the columns in the lower half of the image, Then I used a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame and fit my lane lines with a 2nd order polynomial kinda like this:

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 151 through 163 in my code in (detect_lanes.py). First i calculated the radius of curvature for each lane line in the real world space. Then i calculated the position of the vehicle relative to the center of the lane.

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 177 through 197 in my code in (detect_lanes.py) in the function detect_lane(). 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 (https://github.com/Ahmed-M-Shehata/Advanced_lane_finding/blob/master/CarND-Advanced-Lane-Lines-master/result.mp4)

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 had issue while i was using s channel from HLS color space to determine the min and max threshold, Then i changed to use b channel from Lab color space with threshold (145, 200) and l channel from LUV color space with threshold (215, 255), This pipeline failed in challenge videos when the vehicle goes from light to shadow and the yellow line change from lighter to darker which mean i increase the threshold.