**Advanced Lane Finding**

Udacity Self Driving Engineer –Term 1 Project 4

**Overview**

The goals 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.

The entire project code with relevant comments are implemented in advanced\_lane\_finding.py file.

**Camera Calibration**

The code for this step is contained in the function calibrate\_camera(). 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 and obtained this results. The distortion coefficients thus found are used for further pipelining for images and video.

|  |  |
| --- | --- |
| **Distorted** | **Undistorted** |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\test_images\calibration18.jpg | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\undistorted_calibration18.jpg |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\1_input_image.png | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\2_undistorted_image.png |

**Pipeline for images**

The image pipe line is detailed in steps below:

**Step1:** Each image is undistorted using cv2.undistort() using the camera calibration matrix found from the initial camera calibration step.

|  |  |
| --- | --- |
| **Distorted** | **Undistorted** |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\1_input_image.png | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\2_undistorted_image.png |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test_image.jpg | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\undistorted_test_image.jpg |

**Step2:** Undistorted image is converted to edge detected thresholded image by using magnitudes of the sobel with kernel size of 9 and thresholds set to 75 and 255. In addition, S component of the HLS converted image with thresholds of 175 and 255. These are combined to form a combined binary image. The methods and threshold values are derived after experimentation with different methods of sobel operation and thresholds. The chosen values are shown to provide satisfactory results. These steps are implemented in the function edge\_detect() and mag\_thresh(). Functions abs\_sobel\_thresh() and dir\_threshold() are developed to get absolute and directional value of the sobel operated image values. These functions are used for experimentation and are not used in the final pipeline.

|  |  |
| --- | --- |
| **Input undistorted image** | **Binary combined image** |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\1_input_image.png | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\3_combined_image.png |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test_image.jpg | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\combined_binary_test_image.jpg |

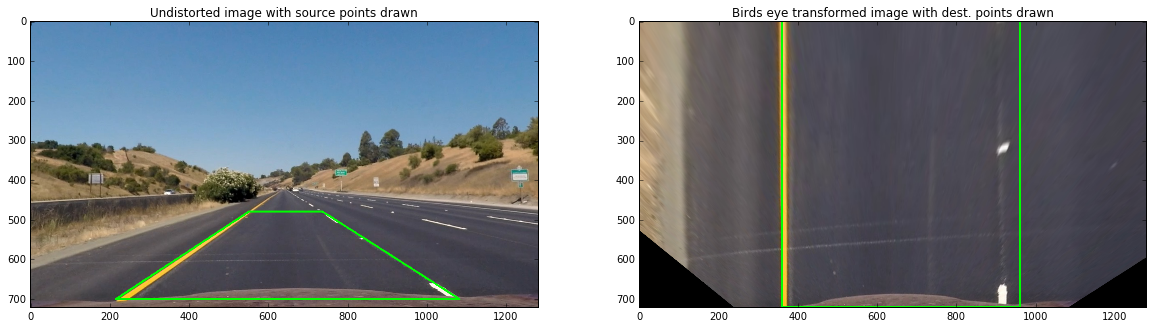
**Step 3:** In this step, the binary image is perspective transformed to provide a bird’s eye view. The bird’s eye image is subjected to lane detection. The detected lane pixels are curve fitted using 2nd order polynomial.

The code for my perspective transform is includes a function called perspective\_transformation (). The function takes source (src) and destination ( dst ) points. I chose the hardcoded the source and destination points in the following manner:

src = np.array([[215,700], [1080,700], [735,480],[550,480]], np.int32)

dst = np.array([[360,720], [960,720], [960,0], [360,0 ]], np.int32)

These values are chosen based on observation and visualization. Below figure illustrates the approach:



**Step 4:** The resulting transformation matrix from the bird’s eye perspective transformation is used to convert combined binary image into perspective transformed image. The resulting binary warped image is provided to detect\_lanes() routine to detect the lanes using histogram and window search method. The same routine calculates the polynomial fitting the lane lines, curvature of the lanes and the vehicle offset from center and returns these values. I have based these measurements on US regulation which is - the lane is about 30 meters long and 3.7 meters wide for converting pixel measurements to measurements in meters.

The figure below illustrates the steps with input combined binary image. The output visualization shows the window search and the curve fit drawn on the image (zoom the image to check the yellow lane which represents the curve fit lanes).

|  |  |
| --- | --- |
| **Input combined binary image** | **Perspective transformed lane detected image with fitted curves shown.** |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\3_combined_image.png | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\4_lane_datected_and_line_fit.png |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\combined_binary_test_image.jpg | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\lane_detected_window_search_and_curve_fit_lines.png |

**Step 5:** In this final step, the lane lines are drawn on the undistorted image for visualization. In addition, the radius of curvature (which is the average of radius of curvature of left lane and right lane) is written and the vehicle offset from the lane center. This is implemented in draw\_lanes\_on\_road() function.

|  |  |
| --- | --- |
| **Input distorted image** | **Final image after the pipelined steps** |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\1_input_image.png | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\5_final_image.png |
| E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test_image.jpg | E:\courses\project4\CarND-Advanced-Lane-Lines_v0.1\output_images\test\5_final_image.png |

**Pipeline for Video:**

All the above steps for image pipeline are called from process\_video\_frame() function. This function accepts image are as its input and returns the final visualization as output. For video frame processing, below additional implementations are made:

* Line class – this class is implemented to store and process lane lines detected. To minimize the jitter, the co-efficient of the line fit are averaged over 3 frames. The Line class also stored and provide the radius of curvature and vehicle offset from center distance.
* Optimized lane detect- This is implemented in function opt\_detect\_lanes(). This function uses the information from the previous lane detections to perform a quick lane detect. In case if not enough lane pixels are detected (tested and empirically set to 7200), complete lane detect algorithm with histogram and window search is run.
* The program prints the number of times complete histogram and window search based lane detect is called and number of times optimized quick lane detect is called.

**Results Conclusions and further work**

The output for the project\_video.mp4 (<https://youtu.be/fSoVP9dX6V8> ) is shown below which shows minimized jitter with good lane detection under various road conditions.

****

The project build run output shows the number of times histogram and window search used (20) and the number of times quick lane detect is used (1241).

[MoviePy] >>>> Building video p4\_project\_video.mp4

[MoviePy] Writing video p4\_project\_video.mp4

100%|█████████████████████████████████████████████████████████████████████████████▉| 1260/1261 [04:33<00:00, 4.51it/s]

[MoviePy] Done.

[MoviePy] >>>> Video ready: p4\_project\_video.mp4

Number of times histogram and window search is used: 16

Number of times quick lanes detect is used : 1245

. . .

The pipeline did not perform well on the challenge video. To improve performance on the challenge videos, the pipeline need to be tuned for better lane detection and prediction. Some of important improvements include – improving the edge/lane detection, improving the lane prediction based on previous lane detections. Overall, this project provided a deeper understanding into the lane detection challenges and thanks to Udacity for putting this up as part of the course.