Advanced Lane Finding Project

\* 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.

[//]: # (Image References)

[image1]: ./examples/undistort\_output.png "Undistorted"

[image2]: ./test\_images/test1.jpg "Road Transformed"

[image3]: ./examples/binary\_combo\_example.jpg "Binary Example"

[image4]: ./examples/warped\_straight\_lines.jpg "Warp Example"

[image5]: ./examples/color\_fit\_lines.jpg "Fit Visual"

[image6]: ./examples/example\_output.jpg "Output"

[video1]: ./project\_video.mp4 "Video"

# Camera Calibration

1. Chessboard images processing.

Call function GetObjImagePoints(images,chessboard\_grid=(9,6)), get object points and image points.

## 2.Get camera calibration parameters.

Call function cv2.calibrateCamera(objpoints, imgpoints, image.shape[1::-1], None, None) ,get (ret, mtx, dist, rvecs, tvecs).

# Pipeline (single image)



1. Undistort image.

Call function UndistortImage(image, objpoints, imgpoints) get undistorted image.

|  |  |
| --- | --- |
| Original image | Undistorted image |
|  |  |

## 2. Combine thresholds.

I use sobel,mag,dir,hls,lab,luv thresholds’combination.

|  |  |
| --- | --- |
| Undistorted image | Thresholed binary image |
|  |  |

## 3. Perspective transform.

Call function GetPerspectiveTransformMatrix() get perspective transform matrix ,M(src->dst) and INV\_M(dst->src).

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.

|  |  |
| --- | --- |
| Undistorted image | Thresholed binary image |
|  |  |

## 4. Find lane lines.

Call FindLaneLine(warped\_binary\_image) to find lane’s left and right lines,and call DrawLaneZone(undistorted\_image,warped\_binary\_image,inverse\_matrix,left\_fit, right\_fit) draw lane zone color onto undistorted original image.

## 5. Calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

curvature,distance\_in\_car\_and\_road\_center = GetCurvatureAndPosition(warped\_binary\_image,left\_fit, right\_fit)

## 6. Draw lane zone color onto original undistorted image.

Call Pipline(img)

|  |  |
| --- | --- |
| Original image | Pipline image |
|  |  |

# Pipeline (video)

## 1.Final video output.

Here's a [link to my video result](./project\_video.mp4)

---

# Discussion

## 1. When process video ,conversion speed is so slowly, it can’t be used in real world self driving car.

## 2. When the light is more complex, the processing is not ideal