Advanced Lane Detection

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:

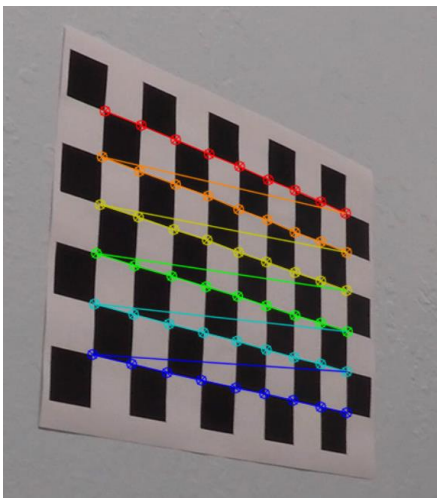


Fig: Uncalibrated Image

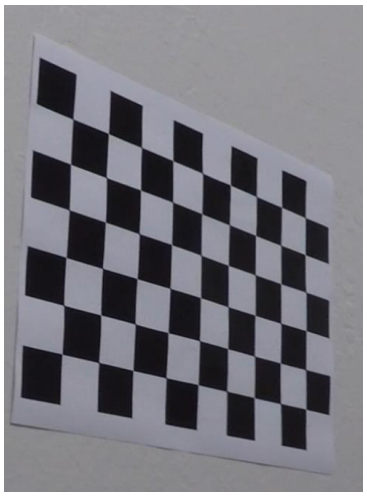


Fig: Calibrated Image

The code for this step is contained in the first code cell of the “example.ipynb” IPython notebook.

We start by defining objpoints which are the (x, y, z) coordinates of the chessboard corners in the real world space and “imgpoints” which are the (x,y) coordinates in image plane.

Thus, `objp` is just a replicated array of coordinates which are fixed, 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.

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img_size,None,None)

I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained the above result. The code for this can be found in code cell 2 of my python notebook

img_undist = cv2.undistort(img, mtx, dist, None, mtx)

Pipeline (single images)

1. Provide an example of a distortion-corrected image.

Below is the result of undistorting the original image after calibrating on the chess board images.

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.

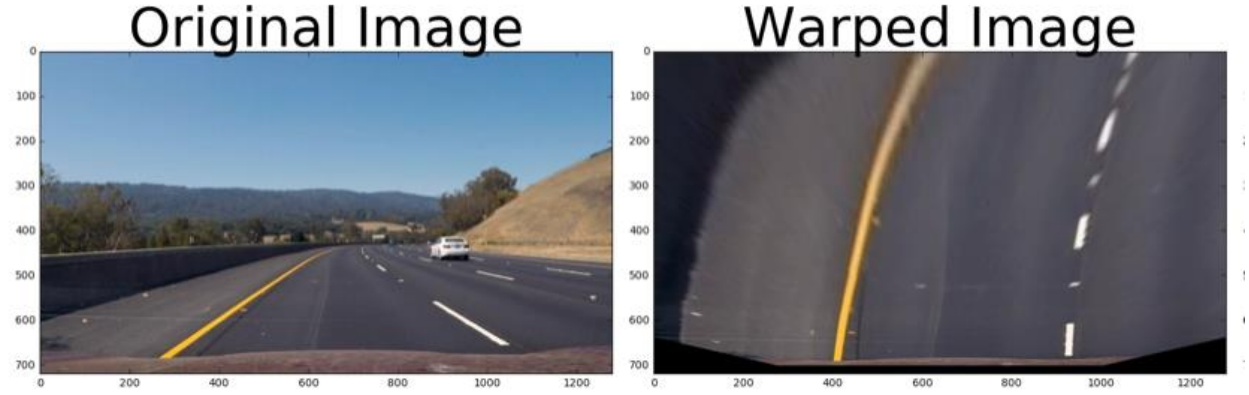
I used a combination of absolute sobel, magnitude color and gradient thresholds to generate a binary image which can be seen in the code cell 4 of python notebook. The function “find_lane” (code cell 5) combines all thresholds to create a single binary image. Here's an example of my output for this step.

3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

For perspective transform, I hardcoded the source and destination points and checked the results manually. The code for the same can be found in code cell 6 (line 1- line14). The selection that gave me best result is:

```
src = np.float32(
    [[760,450],
     [1270,680],
     [10,680],
     [520,450]])

dst = np.float32(
    [[1280,0],
     [1050,690],
     [230,690],
     [0,0]])
```

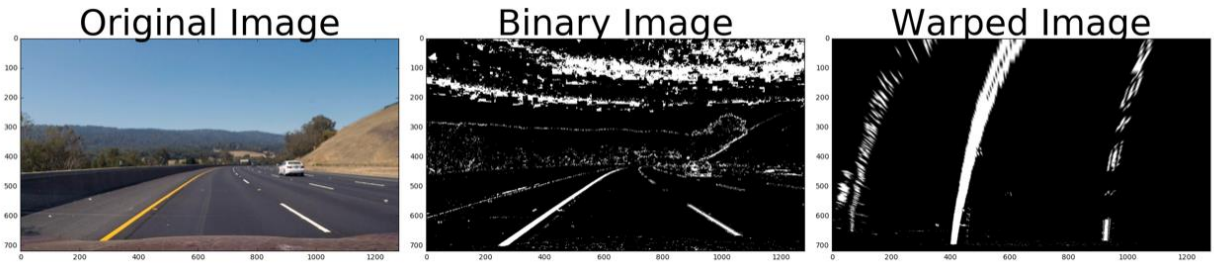


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The function “lane_pipeline” is responsible for finding the lanes. The steps followed are:

a. Undistort the input image
b. Threshold the undistorted image
c. Perspective transform
d. Compute histogram of the lower half of image to find lane starting points
e. Find left and right lane points
f. Fit Polynomial

Below is the result of the input image after first three steps.

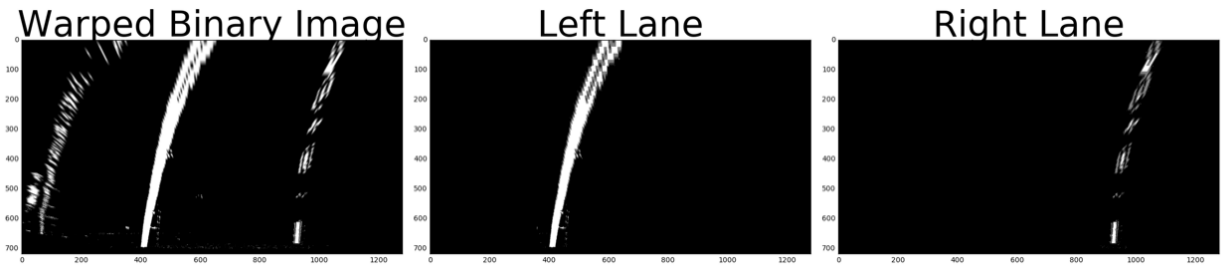


After we get the warped image, we compute histogram of the lower half of the image to find the lane starting points. We the select the local maxima in the areas where we roughly expect the left lane and right lane to start (Code cell 6 :: Line 83-94). It is done by:

```
histogram = np.sum(b_img[b_img.shape[0]/2:,:], axis=0)
left_lane_hist=histogram[250:600]
right_lane_hist=histogram[850:1100]
```

After we get the starting points, we divide the image into 8 segments based on height. We find the histogram, for the section and find the local maxima in a window of 50 pixel on either side to find the expected lane start position for the upper image segment. We repeat the same for other segments.

Doing this, we get the left and right lanes points. The code can be found in code cell 6(Line 106-127) The image for the same can be seen below:



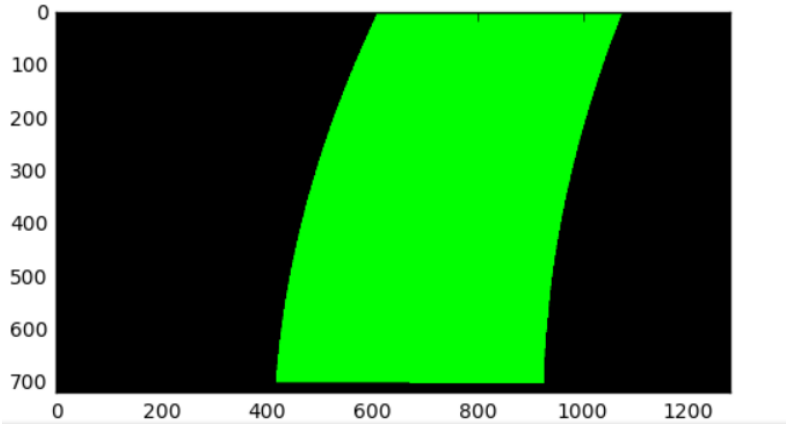
Then I try to find left lane and right point coordinates in order to fit a 2nd order polynomial to the lanes(code cell 6: Line 25-34). This is done by:

```
points_left,points_right=get_lane_points(left_lane_image,right_lane_image,imshape)
```

```
def get_lane_points(left_lane_image,right_lane_image,imshape):
    left_lane_image_array=np.asarray(left_lane_image)
    p_left=np.where(left_lane_image_array!=0)
    points_left=np.column_stack((p_left[1],p_left[0]))

    right_lane_image_array=np.asarray(right_lane_image)
    p_right=np.where(right_lane_image_array!=0)
    points_right=np.column_stack((p_right[1],p_right[0]))
    return points_left,points_right
```

After this, I fit a 2nd order equation to the lane points in the function “fit_equation”. It is done in code cell 6 (line 38-74). I get something as below:



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 code cell 6: lines 60-72. We had stored the coordinates of the left and right lane in the above step, as well as the 2nd order equation and its coefficients. Then we convert the pixel space to meters. This would give us the radius of curvature in real world.

We find the left and right lane curvatures, by using the equation:

$$R_{curve} = \frac{|2A|}{(1+(2Ay+B)^2)^{3/2}}$$

The radius of curvature of lane would be the average of left and right lane curvatures.

To find the position of the center of the vehicle with respect to center, we find the center of left and right lane begin positions. The difference value from the half of image width is the positon of vehicle. In order to get real world estimate, we scale it by multiplying 3.7/700.

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

I implemented all the steps in the function "lane_pipeline" in code cell 6.

After fitting the 2nd order equation, I create a blank image, draw lane lines, fill the space in between lanes with color and wrap the image back into original image.

Below is the image of the final result.

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!).

Please see the below link for the video of my result:

https://www.youtube.com/watch?v=FkFfl-Zc_lw

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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

The function “lane_pipeline” is responsible for finding the lanes. The steps followed are:

a. Undistort the input image
b. Threshold the undistorted image
c. Perspective transform
d. Compute histogram of the lower half of image to find lane starting points
e. Find left and right lane points
f. Fit Polynomial

In my approach I used a combination of absolute sobel, magnitude color and gradient thresholds to generate a binary image. I used parameters that worked fairly well on individual images. More refining of the parameters would make this pipeline perform better. I also use a fixed window to check for the left and right lane to start, which could be improved.