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.

Rubric Points Camera Calibration

1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

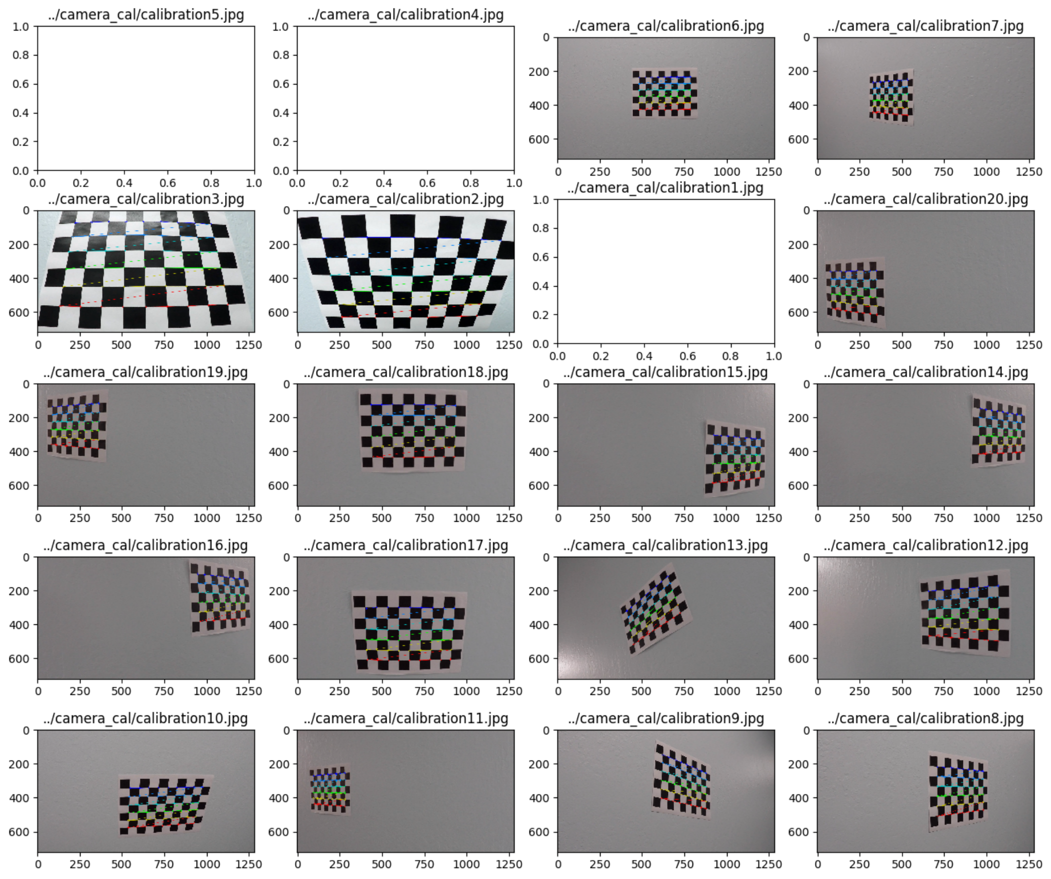
First, compute the camera calibration using chessboard images¶

The code for this step is contained in (Advanced-Lane-Lines/project.py[line 22 to 45]).

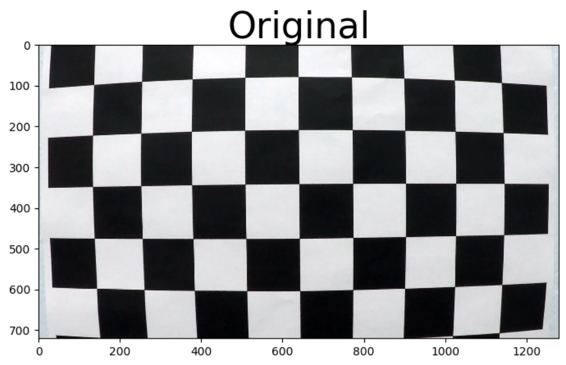
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image.

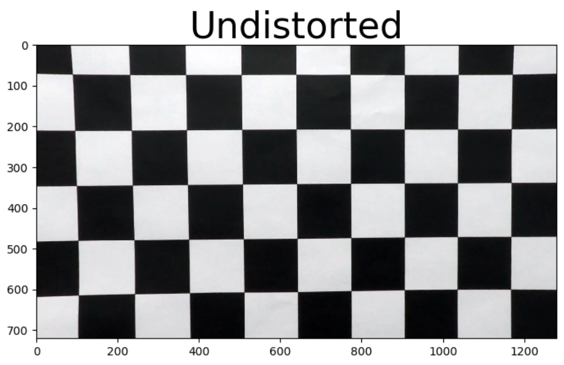
`imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

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


../camera_cal/calibration5.jpg
../camera_cal/calibration4.jpg
../camera_cal/calibration6.jpg
../camera_cal/calibration7.jpg
../camera_cal/calibration3.jpg
../camera_cal/calibration2.jpg
../camera_cal/calibration1.jpg
../camera_cal/calibration20.jpg
../camera_cal/calibration19.jpg
../camera_cal/calibration18.jpg
../camera_cal/calibration15.jpg
../camera_cal/calibration14.jpg
../camera_cal/calibration16.jpg
../camera_cal/calibration17.jpg
../camera_cal/calibration13.jpg
../camera_cal/calibration12.jpg
../camera_cal/calibration10.jpg
../camera_cal/calibration11.jpg
../camera_cal/calibration9.jpg
../camera_cal/calibration8.jpg

to the test image using the `cv2.undistort()` function and obtained this result: The code for this step is contained in (Advanced-Lane-Lines/project.py[line 48 to 72]).





Pipeline (single images)

1. Has the distortion correction been correctly applied to each image?

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

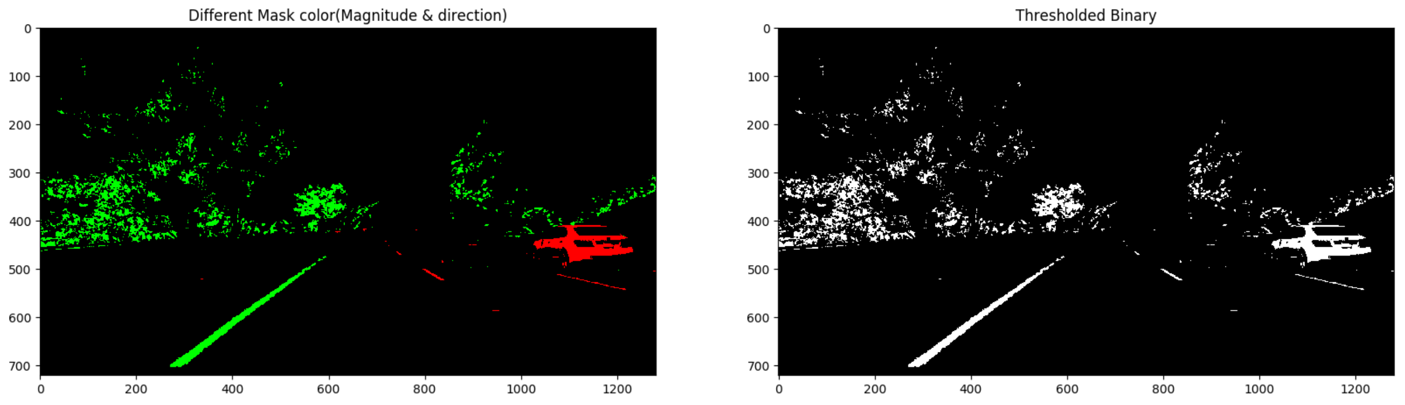
The code for this step is contained in (Advanced-Lane-Lines/project.py[line 83]).:

I use opencv undistort function to undistort.(with calibration parameter mtx and dist)

2. Has a binary image been created using color transforms, gradients or other methods?

I create direction and magnitude mask, merge it to threadholded binary image(you could see the image as below)

The code for this step is contained in (Advanced-Lane-Lines/libs/threshold.py[line 1 to 72]).:



3. Has a perspective transform been applied to rectify the image?

The perspective transformation was done using following source and destination points.

Source Points:

[563, 455],

[720, 455],

[1130, 720],

[190 , 720]

Destination Points:

[200, 0],

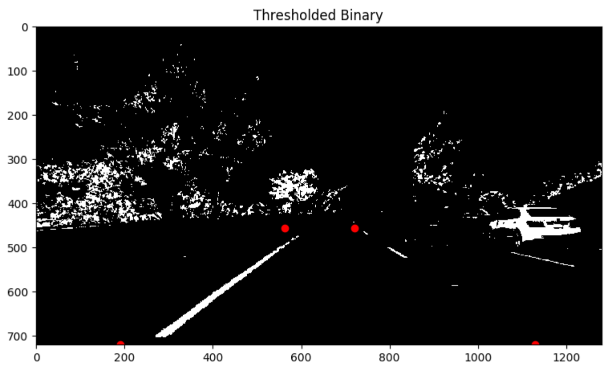
[1080, 0],

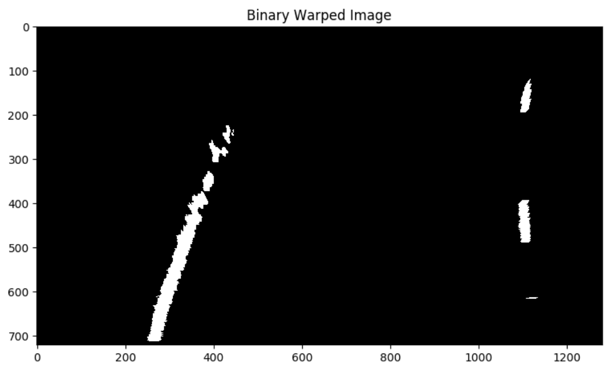
[1080, 720],

[200 , 720]

The code for this step is contained in (Advanced-Lane-Lines/project.py[line 116 to 123]).

The perspective transform for threadholded binary is as below image:





4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

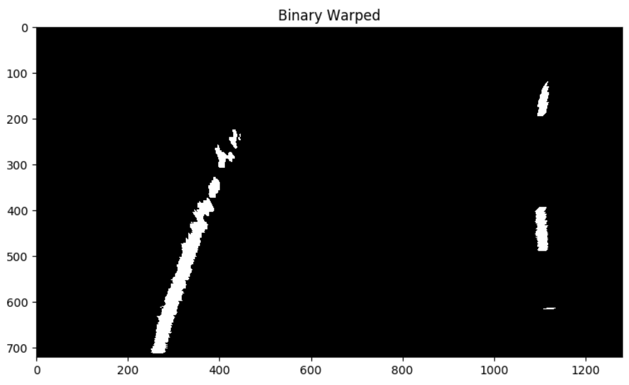
First, find peak on histogram, help us to find the starting point of left and right lane.

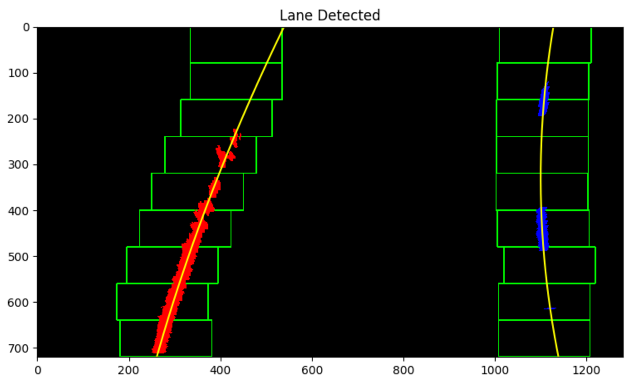
The code for this step is contained in (Advanced-Lane-Lines/libs/pipeline.py[line 44 to 120]).

Create a window that margin is 100px, and start from bottom, identifies ten windows from which to identify lane pixels, each one centered on the midpoint of the pixels from the window below.

When first frame is processed, follows the lane lines up to the top of the binary image, and speeds processing by only searching for activated pixels over a small portion of the image. Having identified the lane lines, has the radius of curvature of the road been estimated. And the position of the vehicle with respect to center in the lane.

You could see the result as below image.





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.

The radius of curvature is based upon this website(http://www.intmath.com/applications-differentiation/8-radius-curvature.php) and calculated in the code cell titled "Radius of Curvature and Distance from Lane Center Calculation" using this line of code (altered for clarity):

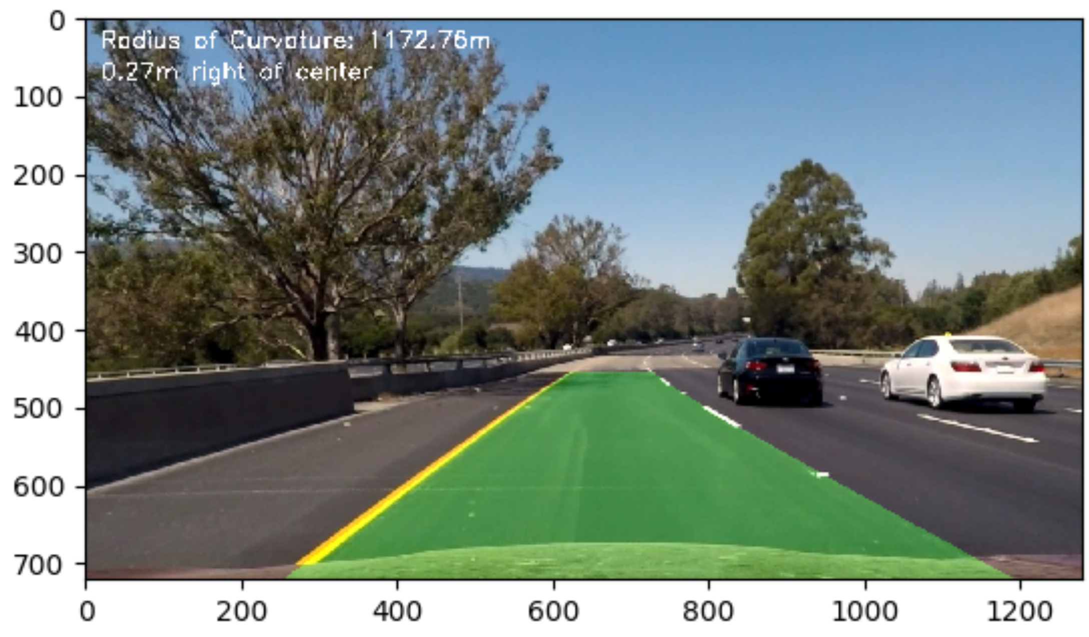
The code for this step is contained in (Advanced-Lane-Lines/libs/fit_line.py[line 39 to 54]).

```
y1 = (2*fit[0]*self.y_eval + fit[1])*self.xm_per_pix/self.ym_per_pix
y2 = 2*fit[0]*self.xm_per_pix/(self.ym_per_pix**2)
curvature = ((1 + y1*y1)**(1.5))/np.absolute(y2)
```

Distance from lane center calculation

```
def get_distance_from_center(self):
    x_left = self.left_fit[0]*(self.y_eval**2) + self.left_fit[1]*self.y_eval + self.left_fit[2]
    x_right = self.right_fit[0]*(self.y_eval**2) + self.right_fit[1]*self.y_eval + self.right_fit[2]
    return ((x_left + x_right)/2.0 - self.midx) * self.xm_per_pix
```

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



Pipeline (video)

1. Does the pipeline established with the test images work to process the video?

It sure does! Here's a link to my video as below link.

https://youtu.be/8rtri9PhG_M

Discussion

This project is heavy loading for me and do not have much to try many parameters.

Parameters likes thresholds, sobel kernel & thresh range, magnitute thresh and so on, turning parameters may help image clean and sharp.

I realize the source point and destinate points for perspective transform that very important. Different image(or frame) have bias of postition, I have no idea, does this affect the accuracy of curvature and distance measure?

And I can improve sliding window in the future, I think this is performance consumption.