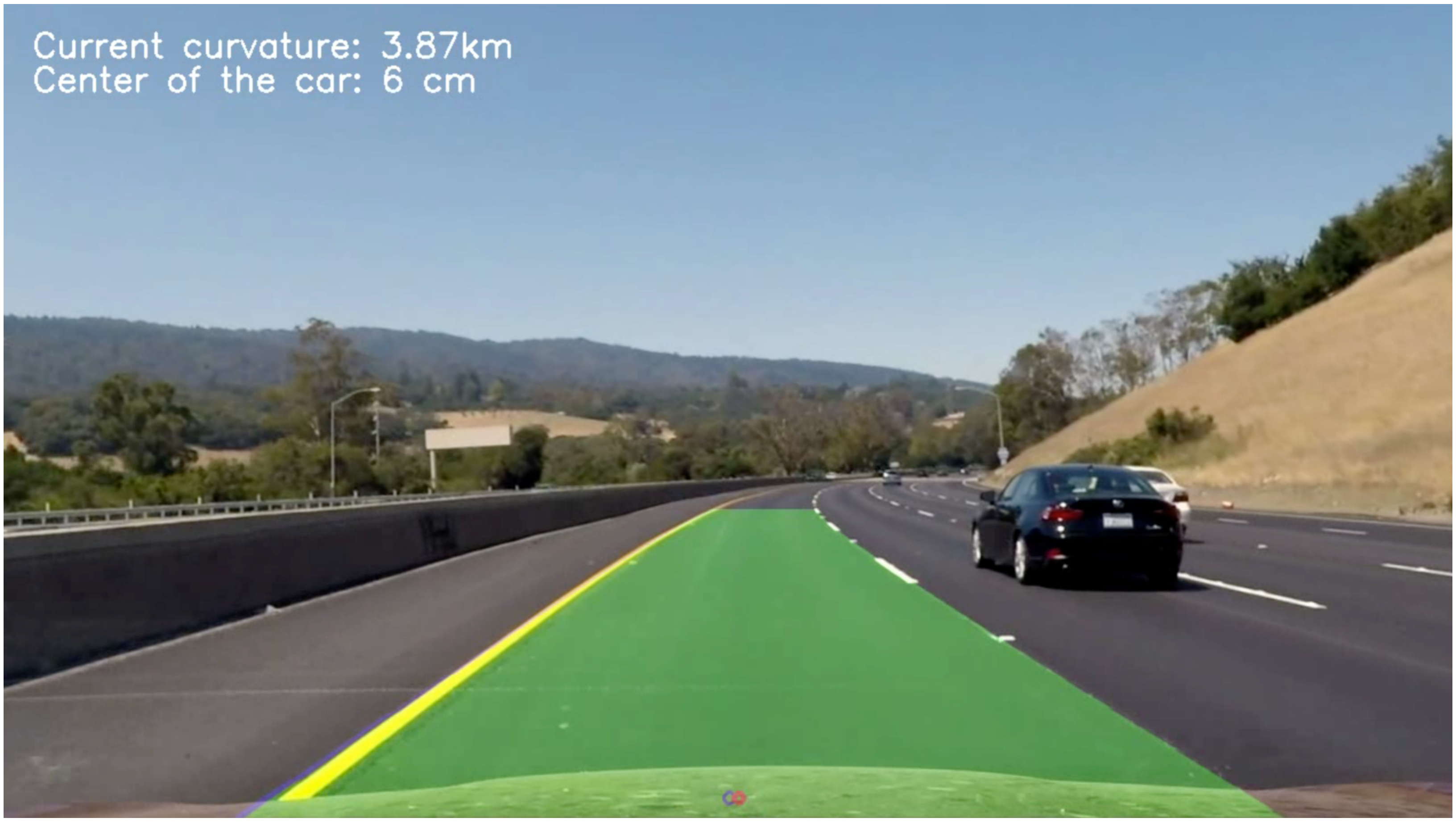
ADVANCED LANE LINE FINDING

AUGUST 09 2021



Introduction

This current write-up corresponds to the delivery of the second project of self driving car nano-degree. All the code can be found in:
https://github.com/sergio-omar/advance_lane_finding/blob/main/lane_line_finding.py

And the video in:

https://github.com/sergio-omar/advance_lane_finding/blob/main/advance_laneline_detection.mov

GOALS OF THIS PROJECT ARE THE FOLLOWING:

1) Compute the matrix camera and distortion coefficients
2) Use the camera matrix and distortion coefficients to remove distortion from the camera
3) Use color transform, gradients and other color techniques to create a refined binary image to quickly spot lane lines
4) Apply perspective transformation to get a birds-eye view
5) Detect lane line pixels using “sliding window” technique
6) Get the corresponding polylines to the left and right lane lines
7) Obtain the curvature of the lane.
8) Obtain the position of the car with respect of the center of the lane
9) Reverse the perspective transformation to get the original perspective
10) Add curvature and position of the car over the image

1. Computing the camera and distortion coefficients

From the chess_pictures folder, we could obtain several pictures that were taken by the same camera. By using these chess board pictures we could obtain the camera matrix and distortion coefficients with the following code:

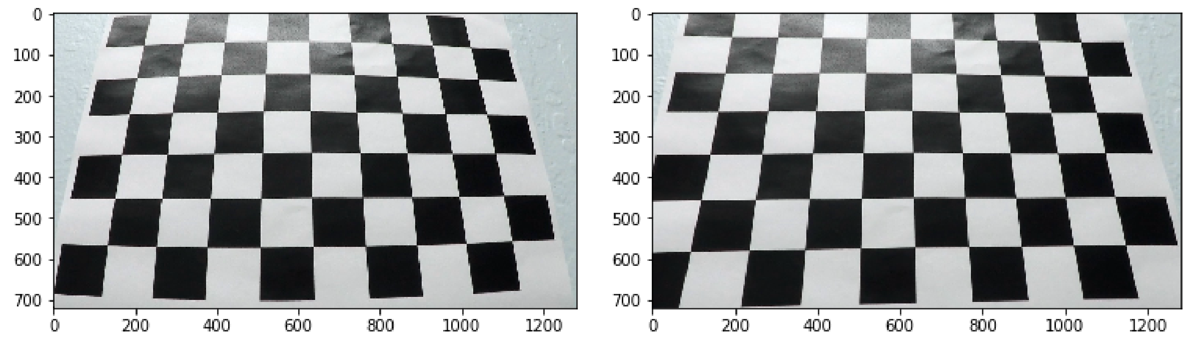
```python
def camera_calibration():
    obj_points = []
    img_points = []
    chess_dir = 'camera_cal/*.jpg'
```

```
list_of_files = []
chess_images = []
objp = np.zeros((9*6,3),np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
for address in glob.glob(chess_dir):
    list_of_files.append(address)

for file in list_of_files:
    img = cv2.imread(file)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray,(9,6),None)
    if (ret == True):
        img_points.append(corners)
        obj_points.append(objp)

ret,mtx,dist,rvecs,tvecs = cv2.calibrateCamera(obj_points,img_points,gray.shape[::-1],None,None)
print(ret)
print('camera matrix successfull')
with open('camera_matrix.pickle','wb') as camera_matrix_file:
    pickle.dump((mtx,dist),camera_matrix_file,protocol=pickle.HIGHEST_PROTOCOL)
```

After we find the camera matrix and distortion coefficients we store them in a pickle file for later use.

2. Remove camera distortion

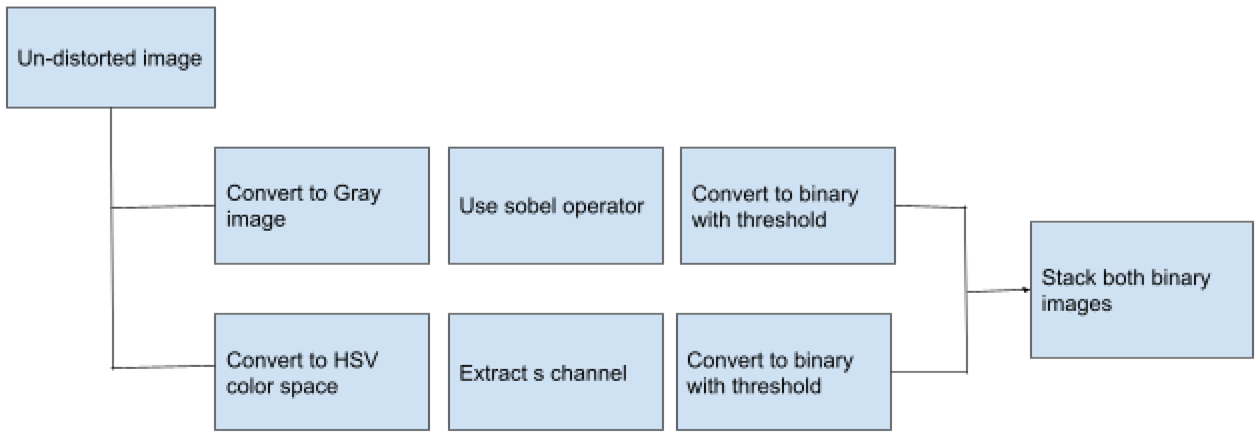
Using the values from camera matrix and distortion coefficients we could proceed to undistort images. Next we have a sample of a “raw” image and an image with distortion removed



Where we can clearly see that the image on the left has small curvature in the center of the image. In the right image we can appreciate the same image but applying the "undistortion" function.

3. Color transform

In this section I use several techniques that can be resumed in this schema:



4. Mask region of interest

From the previous project I decided to mask the region of interest, then we could avoid some line detections caused by the shade of the left fence. The shape of the mask is a trapeze.

5. Apply “BIRDS-EYE”perspective transformation.

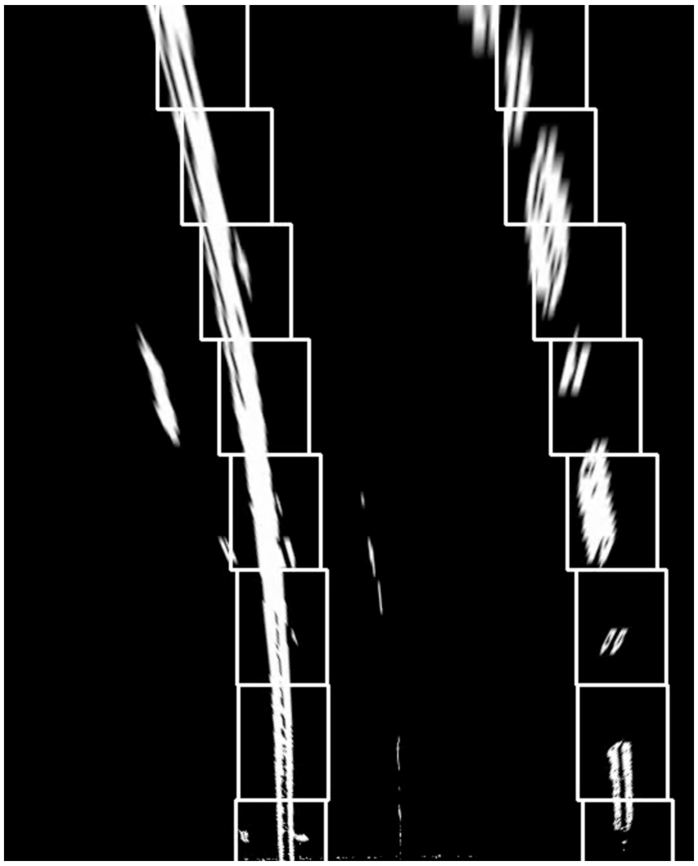
In this part of the project I was able to take the stacked image and apply a birds eye transformation. The most difficult part of this section was to figure out the importance of use of really straight lane, then find the coordinates of four points corresponding to the 4 corners of the lane. If the 4 points were inaccurate all transformations will distort the road in the “birds-eye” perspective.

5. Apply “BIRDS-EYE”perspective transformation

In this section it is really important to understand each step of the process that are required to make a sliding window.

1) Using only half of the picture, apply Histogram to find the biggest sum of “True” of the vertical pixels to identify where the left or right lane line is.
2) Starting from those two points the algorithm will use the size of the window to search for an average of all the “True” pixels to give the coordinates of the next Window.
3) Repeat the second step until all the y axis is examined.

After surveying all the y axis we will obtain the next image:



6. Find polylines with the points found by sliding window algorithm

With the points found by the sliding window, now we can proceed to find the polylines that will define the edge of the lane lines. We find both polylines, one for the left and one for the right. In my algorithm I decided to make a green rectangle that will paint the road color green. For that i used the two polylines with the next cv2 library.

```
cv2.fillPoly(out_img,np.array([lane_points],dtype=np.int32),(0,255,0))
```

7. Obtain the curvature

After obtaining the polylines is quite straightforward to calculate the curvature. Using the cvw library:

```
left_curverad = ((1 + (2*left_fit[0]*y_eval*ym_per_pix + left_fit[1])**2)**1.5)
/ np.absolute(2*left_fit[0])
```

8. Obtain the position of the car with respect to the lane line.

To obtain the position of the car respect to the lane we first take in account that the middle of the car is the middle of the image: "center_of_the_car = img_size[0]/2" and the middle of the lane with: "middle_of_lane = (y**2*(left_fit[0])+y*(left_fit[1]+left_fit[0])) - (y**2*(right_fit[0]+y*right_fit[1]+right_fit[2]))" with y = 720

To make it more soft I added a low pass filter that can help stabilize the measurement

9. Reverse the perspective

To go back again to the original perspective we can use the previous numbers to reverse this operation.

10. Add curvature and position of the car over the image.

Once we have the curvature and position of the car with respect to the center of the lane we can easily overlapped with the next line of code:

```
cv2.putText(img,str_curvature,pos,font,scale,color,line_type)
cv2.putText(img,str_center,pos,font,scale,color,line_type)
```

I enjoyed this project a lot, although I encounter some difficulties in:

1) Choose the sweet spot in thresholds for sobel operator, as well as the binary image
2) When trying to find the lane line the software sometimes gets lost with other lines similar to the lane lines, so that's why I used the mask, leaving just the region of interest.
3) When trying to use the algorithm to make more efficient the lane-line finding by only using a search around the polynomial equation, the algorithm easily get lost and was hard to return to the correct position of the lane. That's why I keep using the sliding window.
4) In my opinion i didn't like that the curvature was very “jumpy” that is the reason why I added a filter that could help to stabilize the curvature.

As a personal project I want to add this software to Jetson Nano and install it in the middle of the windshield.