Advanced Line Detection

Author: Diogo Pontes

Email: dpontes11@gmail.com

Objective of the project

In this project, the goal is to write a software pipeline to identify the lane boundaries in a video from a front-facing camera on a car.

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 ("birdseye view").
- Detect lane pixels and fit to find 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 and distortion correction

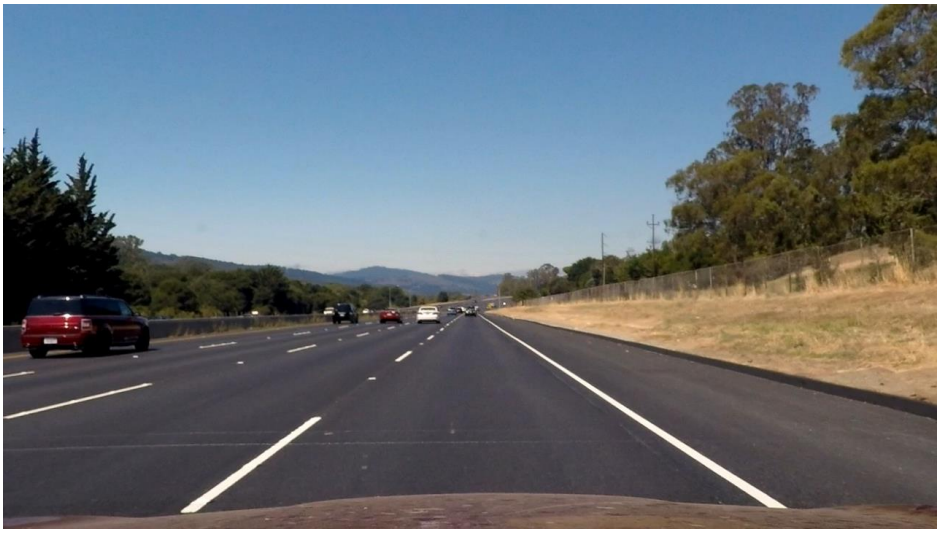
Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image

The code for the camera calibration is in the *calibrate.py* file. It simply loads the images from */camera_cal/* directory and uses the chessboard corners to compute the camera matrix and distortion coeficients.

The result data is stored in the *wide_dist_pickle.p* file which will be used by the rest of the scripts.

Original Image

Calibrated / Undistorted



Pipeline (test images)

1. Provide an example of a distortion-corrected image

Example given previously

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

All image processing is in the *process_image* function. It receives an image and returns an array of warped, filtered images.

The process is:

- Undistort the image
- Convert to HSV and separate the channels
- Apply EqualizeHist to value and saturation channels
- Apply a GaussianBlur with a 7x7 kernel to value and saturation channels
- Call a special remove_dark function (later explained)
- Apply a set of filters and warp the resulting images

The remove_dark function was originally used to remove dark lines and soft some edges. Finally it became a more generalized light equalization. It's process is:

- Divide the lower half of the image (with the exception of the bottom 20 pixels) in n horizontal slides
- For each slide compute the average V value in the center of the image (between 20% and 80% of the width)
- Select all points below this average value and apply to them a BoxFilter with 50x50 pixels in S and V channels
- Just for the V channel move all below average intensities between 0 and 50 and expand the ones over the average to 50-255

The idea is that low intensity features are blurred and will generate less gradients as high intensities have a greater range.

Here is an example of the value channel, first the original image:

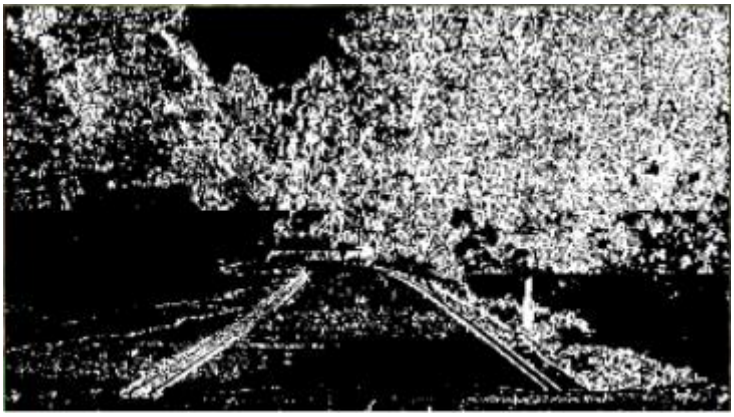
Then this one is just after the equalization and GaussianBlur:

And this one is the same as the previous one but after remove_dark with 3 slides:

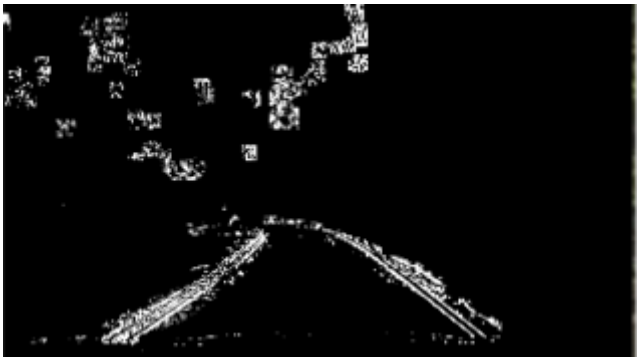
To get the bitmap images I have used 2 filters and an additional color filter.

First one is the one talked in the course, just selecting pixels with gradient in x and y direction and those special values and saturations. It is in the gradient_filter function that uses the color_threshold and the abs_sobel_thresh. Here is an example of the same image with said filters:

The other algorithm uses similar sobel computations linked with values in gradient x and saturation but "or's" them together so usually is messier. It is implemented in the complex_sobel function.



Selecting hue is difficult but a way to do it is select just the road, expand its borders and intersect with the results of the other algorithm so we have a mask that may be intersected with the other results, giving:



which is a lot better.

So we have 4 different possibilities in total. Depending on the situation we may like to use one or another result so that the main filtering function, **super_filter** may return an array of all identified as "gr", "so", "grt" and "sot" corresponding to the images presented.

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

At the beginning the perspective transformation was calculated manually. Afterwards a small piece of code was developed in order to generate the transformation from just one parameter that is linked to the camera focal length and information about how to compress the images so that the curves on the road are visible when warped.

It is in the *perspective* function:

```
def perspective(focal=1.3245, maxHeight=460, size=(1280,720), shrink=0.0, xmpp=0.004):
```

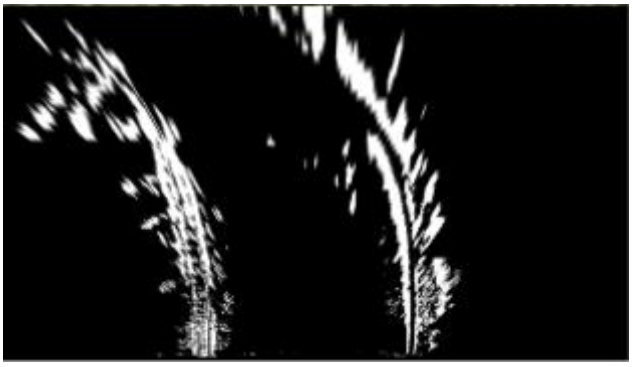
The conversion in the X axis from pixels to meters is modified accordingly and returned

Original Image

Warped Image

As seen in the warped image, lane lines are parallel to each other.

For the previous filtered images we get:



which are similar and quite parallel at the bottom but not so clear at the top.

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

Before going to line recognition we must decide which of the filters to use. That is an interesting question and surely there must exist a way to select them before applying. I just compute all lines in every case and get the ones that seem better.

I use two methods, the sliding window method and an existing fit method for selecting points and fitting second order polynomials to the points.

The two methods build a Measure object with information from the sides.

Sliding Window is as explained in the course. We get the two centers at the bottom by getting the maximum of an histogram of a convolution of all ones window over the bottom 1/4 of the image.

From here we look at next level centers around last level centers with the optimization that if no points are found in the window, the "movement" of the centroid from last one is carried on to the next level.

it is implemented in the sliding_window and find_window_centroids functions. Points are packed as a Fit object that also computes the fit and the residual and some data as radius.

World coefficients are computed from the warped image units as it is not necessary to do another fit.

Also did some work to check that the average of the residuals are really the sigma^2 of the points against the computed points.

In fact the residuals interpreted as the squared standard deviation of the "fit" are very important in the algorithm.

The existing_fit method is implemented in known_lines_fit function. It just looks for points in a "margin" around the current fit with the exception that we use an advanced fit.

Fit lines are computed in Fit.compute_fit(self):

```
def compute_fit(self):

   aux = np.polyfit(self.y_values, self.x_values, 2, full=True)
   self.coeficients = aux[0]
   if len(aux[1]) > 0:
      self.residuals = aux[1][0] / len(self.x_values)

   else:
      self.residuals = 500

   self.compute_world_coeficients()
```

A difference with what has been taught is that my fit units are reversed in Y.

Y=0 is the bottom of the screen. That has some interesting properties for the fit (Ay^2 + By + C):

- C is the position at x=0
- B is the "direction" of the lane at x=0
- A is the "curvature"

The first property makes it very easy to get the position of the car and the lanes and move one fit to check parallelism with another one. Just made C = 0.

This is clear in the lane creation in the sliding_windows_fit and the known_lines_fit in:

```
left_lane = Lane_Measure.new_lane_measure_from_data(leftx,
                                    maxy-lefty,
                                    l_points,
                                    side='left',
                                    filter_x=filter_name,
                                    method='windows', xm=world.calibration.xm,
                                    ym=world.calibration.ym)
```

where lefty has been changed to maxy-lefty.

5. Describe how (and identify where in the code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center

Radius computed in the **radius** and **world_radius** methods of the **Fit** class:

```
def world_radius(self, y):
    curverad = ((1 + (2 * self.world_coeficients[0] * y + self.world_coeficients[1]) ** 2) ** 1.5) / np.absolute(
        2 * self.world_coeficients[0])

    return curverad
```

Position is computed in the **position** and the **position_w** methods of the **Belief** class:

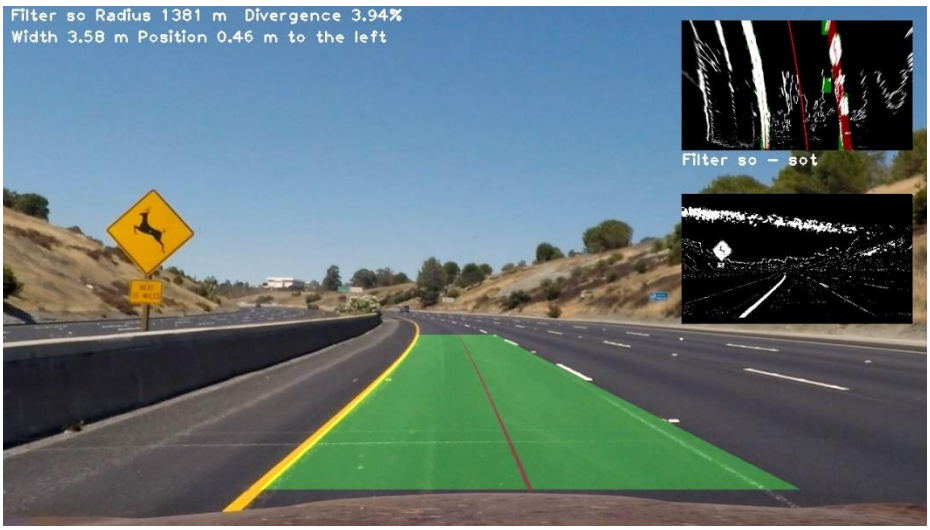
```
def position(self):

    image_center = self.left_data.get_shape()[1] / 2.0
    lane_center = self.center_lane.get_x(0)

    lane_offset = lane_center - image_center

    return lane_offset
```

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



Here we have an image where the lanes are clearly marked, some important stats are visible, is it specified which filter has been selected ("so"), and an image of the warped image with the sliding windows and adjusted lines.

Pipeline (video)

Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video

The video can be viewed here: https://youtu.be/c88UoVzh6s8

(Video is not listed)

In the video, in the small window with the warped image, the red line markers at the left and right show the error. There is also a blue line, not easy to see, that shows the last frame adjustment.

Discussion

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?

I have spent more time on this project than ever before, even though it felt easier.

Most problems have been with image processing. How to eliminate shadows, identify what really is important in the image.

The video only has the "clear lane" problems and these can be solved by the "forget frame" as they are quite stable sessions.

Looking back I feel that I overcomplicated my code, but the algorithm in my head made sense to me, and I didn't want to start over, as it always felt that I was close to the finish.