Finding Lane Lines on the Road

Autor: Michael Scharf
Email: mitschen@gmail.com
Date: 29th may 2017

Finding Lane Lines on the Road

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

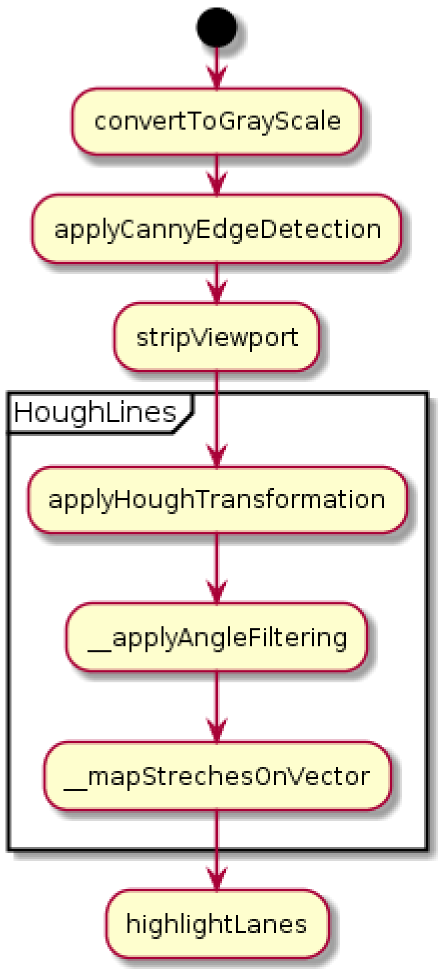
Reflection

The so called **LaneFinder** module realizes a straight forward pipeline of different processing steps which allows to find street lanes in images as well as in video clips. The module hides the complete processing in a single **findLanes** function which expects an image (matplotlib.image). Inside the LaneFinder 5 steps are executed which applies different convertions on a working copy of the original image. As a result of the findLanes function, an image containing the original image as well as the highlightes lanes is returned.
By applying the LaneFinder.findLanes on a sequence of images a complete video clip could be used as input for the LaneFinder.

remark: unfortuantely I didn't notice the already given sequence of pipeline steps that were available in the P1.ipynb, therefore the structure in my solution looks slidly different but in principle reflects the same doings.

Processing steps (the pipeline)

The pipeline of the LaneFinder contains 5 steps. As you can see in the flowchart, two of these steps are more or less implicit called during the HoughLines extraction.



In principle the LaneFinder starts with the convertion of the image into a grayscale image. The result is stored in the internal member so that at any given time the original imgaes stays as it is. The reason for converting the image into grayscale is that it allows a more simpler and faster criteria for the following edge detection by simply reducing the number of colors and therefore the degrees of freedom. For the CannyEdge dectection we've chosen the threshold between 50 and 150. The extraced edged are stored again inside the working copy of the original image. As the next step, we strip the image to the relevant part - this is done by applying a polygon mask in shape of a trapeze to the internal working copy.

The houghLines extraction is splitted into three steps. It operates on the gray, edgy, masked image. The parameters for the hough transformation are:

```
theta = 1 * pi / 180
rho = 2
threshold = 15
min_line_length = 20
max_line_gap = 20
```

As a result of the hough transformation we get a bunch of non-connected linesegments. Unfortunately the houghtransformations also provides lines we're not really interested in e.g. the shape of the engine bonnet. In order to get rid of this noise, we apply a filter based on line-angle.
Our assumption is that all lanes - as long as the autonomous car is not doing any stunts - refereing to the left bottom corner of the image are in a angle range of 35+/-15 degrees. Any lines not matching this filter are getting removed.

The next step is combining the remaining line-stretches to a maximum of two vectors. This is done by using the given points of each line-stretch and calculating the grade **m** as well as the section **b**. This is done by changing the equation

$$y = x * m + b$$

$$<=> b = y - x * m$$

while the grade is given by

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

By doing that for all given points we're getting a list of m_+ and b_+ as well as for m_- and b_-. The resulting vectors are received by calculating the median (not the average) of m and b and identifying two lines which start at the x-axis and reaching to roughtly the middle of the image.

The last step simply combines the original image with the working copy - which means in essence the resulting lines are drawn on top of the image.

2. Identify potential shortcomings with your current pipeline

So from my point of view there are several drawbacks of the solution so far:

- Grayscale convertion: as i've noticed during my development i've figured out that convertion into grayscale might fail in case that the overall brightness of the picture is very high. The "challenge.mp4" has a few of these situation in which the grayscale-based canny did fail with the parameters given
- stripping operation: the limitation of the viewport by the trapeze polygon might cause misdetection in case of lane-changes as well as in steep curves
- hough transformation seems to be a very expensive operation. Especially in a real world scenario, the CPU might run very fast to its limits which will result in non-realtime desicion making
- resulting line extrapolation is based on the current image ony. In order to apply a smooth and more reliable transition from image i to image i+1 some kind of history should be taken into account.

And overall - the LaneFinder is not really flexible at all. The parameters are hardcoded and optimized for the scenario given. Furthermore I'm quite sure due to fact that I'm in touch which great libs like cv, numpy, matplotlib, ... for the first time, there is much space for me for optimization in using these libs.

3. Suggest possible improvements to your pipeline

I guess concerning the issue with the grayscaled images, using another color-space like **YUV** or **HSV** might be worth to give a try.
Concerning the stripping of the view-port, I would say a feasible approach is to adjust the polygon depending on the steer angle. In this case the trapeze might be changed to a parallelogram directed to the curve.
Converning HoughTransformation - I'm not really familar with the content yet. But maybe there is already some other kind of line-detection. I saw a method in cv2 but didn't yet dive into the details of it.

And for sure the overall concept of the LaneFinder could be optimized. I guess one step might be to make the parameters configureable. This could be happening dynamically e.g. by examine the overall brightness of the image in process. Furthermore there are some steps which either could be replaced by predefined functions (as mentioned, I'm not yet familiar with the libraries) or maybe optimized in the way that seperated steps are done in a single step. As an example I would say the functions *applyAngleFiltering* as well as *mapStretchesOnVector* could be merged to a single function.