September 23, 2021

This document describes the implementation of a program to identify lane lines on the road.

Pipeline

The input image goes through a series of transformations:

- **Grayscale**: lines can be of different colors, also because of the lighting, so we rather identify lines as edges, i.e. areas of the image with a sudden variation of color.
- **Gaussian blur**: we apply a Gaussian kernel of a given provided size (typically, 3 or 5), in order to average pixel values based on their neighbor cells, and this way eliminate noisy edges that will hence not be detected as such by the Canny edge detection algorithm we apply below.
- **Canny**: we use the Canny edge detection algorithm on the grayscale image, which identifies edges as areas with a pixel value gradient being greater than a given provided value, which is a parameter that we need to fine-tune. The lower gradient threshold is eliminating, while areas with a gradient between the lower and upper thresholds make it to the output if they are connected to an area with a gradient beyond the upper threshold.
- **Region of interest**: we assume lane lines are always only in a given area of the input image, and apply a mask to only keep this region. The rest of the image is set to black.
- **Hough lines**: we apply the HoughLinesP function of OpenCV, which identifies lines thanks to the representation of points in Hough space, whereby points are represented as lines reflecting the position of the point in the original image. A set of points that somewhat represent a “line” in the original image would have lines that more or less intersect in a given grid cell of the Hough space. Depending on the granularity of this grid, which we control with parameters given to the HoughLinesP function, it will be more or less easy for a set of points to be considered as belonging to the same line. The intersection cell finally gives a line in the original image, so at this point a line of interest has been identified.
- **Averaging and connecting**: we observe that the borders of lane lines are each identified as edges, but ideally, we want to merge them and have a thicker, single line drawn on the original image to mark lane lines clearly. Also, most lane lines are not continuous, so we end up with chunks that we want to connect together. The algorithm needs to connect the right chunks together. This is done in a simplistic and imperfect way, only taking into account the “slope bucket” into which the slope belongs, and computing a new line going through the center of all lines belonging to the same chunk, and extending from the bottom of the image to a given fixed height which we assume is always more or less the upper part of the road on the image (which is obviously too simplistic to work safely in reality).

Parameters

The main parameters of interest are the HoughLinesP parameters:

```
rho = 4 # distance resolution in pixels of the Hough space grid
theta = 3.0 * math.pi / 180.0 # angular resolution in radians
threshold = 70 # minimum number of intersections in a grid cell for a line to be considered as such
           # (corresponding to the points making up this candidate line, all represented by a line
           # in Hough space, and those lines intersecting)
min_line_length = 5 # minimum line length in pixels that will be accepted in the output
max_line_gap = 10 # maximum distance in pixels between segments that can be merged into the same line
line_image = hough_lines(image_masked, rho, theta, threshold, min_line_length, max_line_gap)
```

Those values were obtained after many trial-and-errors.
The granularity had to be permissive enough, as lines from real images are not really “perfect lines” as one would draw with a ruler.
The threshold has to be large enough to avoid connecting together a set of points that don’t really fit all on the same line, but small enough to account for the fact mentioned in the sentence above, namely that lane lines in real images are not perfect lines, hence points of these lines will not have lines in Hough space that perfectly intersect.
Finally, the min line length had to be small enough to account for the observed fact that some lane lines are not perfect and sometimes have big gaps with just a small line chunk in-between that is still very useful to identify to further be able to build to full length line by connecting chunks together.

Drawbacks and potential improvements

We attempt to run the above-described algorithm on the challenge video, and observe the following:

- the region of interest has a different shape from the hard-coded shape that we worked on for the 2 other simpler videos, and many irrelevant edges are identified. Somehow, the algorithm should be able to adapt the region of interest dynamically to the context
- some parts of the lane lines are not identified anymore when the road becomes light-colored, since the contrast of color with the line becomes too low. Maybe the algorithm should extrapolate lines across successive frames based on the gradient of a line slope in the past frames.
- we observe that because of the noise in detected edges, the simplistic averaging algorithm for lines, that merges chunks of similar gradients, does not work at all anymore here, as it tries to merge together chunks that actually do not belong to the same line