October 21, 2017

CarND-term1_P1_write-term_Andriy Kumanovskyy – Reflection Update

1.) Summary

Greyscaling. The first step involves grey-scaling the image. This is done to not only make the gaussian filter more effective, but to prevent certain colored lines on the road from being detected more easily than others -- leveling the playing field.

Gaussian filter is applied in order to remove any potential noise in the image so that it does not register in the following line detecting steps. A 5x5 matrix is chosen in order to have precision without losing the detector's sensitivity to noise or increasing the localization error and which invertly affects edge detection.

Canny edge detection is applied to the image in order to take the slope between each two adjacent pixels and to show this image of intensity. Following this, non-maximum suppression thins the gradient lines, while double gradient applies the threshold values for which anything under the threshold is rejected (weak edge), anything above is preserved (strong edge). The last step --hysteresis ensures that anything in between the two thresholds is shown as long as it is connected to a strong edge.

Masking
Masking exploits the natural "triangle" shape of lane lines as is visible from the top of a car where a camera would be mounted.I used the quadrilateral masking shown in the lesson with adjusted values.

Hough Space Detection
The Hough space method is utilized in order to detect lines in the image. Lines detected by the image via Canny Edge Detection -- while visible to us, are not recognized by the computer (and any subsequent steps cannot be performed, such as line extrapolation) unless a very clever mathematical approach is used. The edges visible in an image after a canny edge detection was performed are identified as pixels for a computer, or mathematical points. One point serves as a parametric axis over which an infinite number of lines in image space -- all related to each other through a function in hough scape -- can rotate over. If two or more points exist in image space, then when two rotating lines align to the same y-intercept and slope -- we obtain a crossover between who functions in hough space. This is basis over which computer vision can detect lines in a stream of images, which is essentially the definition of a video stream.

Various parameters from this .cv2 module can be adjusted in order to change the line detecting algorithm depending on the situation such as the angle resolution in hough space, min_length for it to be considered a line, max line gap, ect

Line Drawing and Extrapolation
Following hough image detection, the lines can be drawn back onto the image space. While this seems effective, this is not adequate for any control purposes as we are obtaining a hefty set of lines for both the left and right side which would render an effective and safe control system useless. For this reason the various lines are needed to be converted into a single line utilizing various mathematical tools such as regression lines, line of best fit, and others if necessary.

For this pipeline's "draw lines" function I firstly separated the slopes of various points' interaction with another point into the left and right lanes -- and appended this information into applicable variables (right_x1, right_y1, ect.). For both the left and right lanes this numerical list -- or array of information was then averaged and processed to obtain an average slope of many different lines to obtain a very close approximation of all of the left and all of the right lines.

The true values for x were then iterated from the latest average x values and the incremental difference between the next values given by the equation ((right_y1 - avg_right_y1) / right_slope) which returns a delta increment of x. The true values used for y were guesses for both the left and the right lanes for which the cv2.line function was implemented to match the left and right lines into a single line from the bottom of the line near the car, to the far reaches of the lanes as limited by the masking function.

I used float to maximize precision at the expense of computation time.

Weighted Image Combination
The last step is the moment of truth which combines the original image with the lines created by our pipeline up to this point. This will show us whether the code has been able to adequately detect the lines. I decided to make the lines slightly transparent when I add-weighted the function defined by:

Video

Finally, the pipeline was wrapped in order to be able to apply it to a video input and drawing the "draw_lane_lines" on the two first videos.

2.) Shortcoming of my pipeline

While I was able to successfully apply my pipeline to detect lines for the first two videos, I was unable to do so for the last "challenge" video. The challenge video combined moving curves as well as various different lighting conditions which caused my pipeline to perform in unexpected ways – even bypassing the masking outline.

Perhaps applying the suggestions will take care of the above stated issue, if not I will experiment with different ways to define an array (tuples, ect) and other methods hardcoding and "forcing" the masking to take place.

3.) Suggestions for improving the algorithm

The algorithm can be improved by increasing the min_line_len and the max_line_gap to numbers over 100, which would have a direct impact on the solid annotated output line – effectively increasing its length to perhaps fill the entire masked region. Increasing the lower threshold for canny above 50 up to 60 can be advantageous in ruling out false lines for other cases which involve curved lanes and shadows. Increasing the kernel-size in the Gaussian filter would remove the noise – making the image less blurry.

Finally, the output line's shakiness can be reduced by taking the average of the previous several lines in order to reduce the line shakiness. This can only become visible after the first few frames as initially the video feed has no prior knowledge of the frame before. I can do this method taking advantage of python's ability to redefine the same variables later utilizing a loop. This will allow any control systems reading the lines to exhibit a less jerky response. One thing to be extremely careful about is to not average too many previous images (frames) as this will create a time gap between what is being computed and what is actually happening right now, potentially causing an accident. I am thinking averaging in the realm of the last 2-3 images, no more than 5.