Finding Lane Lines on the Road

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

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consists of the following steps:

- Convert the input color image to grayscale.
- Apply Gaussian blur to smooth the image.
- Compute an "edge-map" image using Canny edge detection.
- The left & right lane lines appear in distinct and relatively consistent areas of the test images/videos, so the rest of the steps in the pipeline are performed independently for left & right lines.
    - Apply region-of-interest mask to Canny edge-map to block out most edges not on the lane line.
    - Detect line segment features in the Canny edge-map using the Hough Transform.
    - At this point we likely have multiple line segments that actually lie along the lane line, as well as a small number of outlier segments. The pipeline includes a custom algorithm for filtering & combining the line segments into a single lane line.
- Draw the left & right lane lines on an overlay image (no background).
- Blend the original image with the lane- lines overlay to produce the final result image.

!canny-edges !hough-lines !lane-lines

For code clarity I decomposed my pipeline into several functions somewhat different from the provided utility functions. Thus my logic for combining Hough line segments into lane lines doesn't appear in the draw_lines() function (which I didn't use).

Instead this algorithm is defined in the segments_to_lane_lines() function. The following is a description of the algorithm. Keep in mind this is called separately for left & right lane lines.

- For each line segment, compute the midpoint, direction angle, and segment length.
- Ignore segments that are nearly vertical or horizontal, as the lane lines are always slanted one way or the other.
- Combine remaining segments into a single lane line as follows:
    - Point on the lane line = weighted average of segment midpoints, where the weights are the

    segments lengths.
    - Lane line direction angle = weighted average of segment angles, where the weights are again the segment lengths.
    - An important detail is that the segment direction angles are constrained to "point" in a consistent positive or negative direction. Thus they can be averaged without cancelling out.

2. Identify potential shortcomings with your current pipeline

The primary shortcoming of the current pipeline is sensitivity to outlier line segments. Because the ROI dimensions and Canny/Hough parameters are chosen to reliably capture lane line edges & lines with reasonable image variation, it's unavoidable that some outlier line segments will also be detected. Some of the outliers are removed via slope filtering, but remaining outliers clearly distort the lane line averaging.

Another shortcoming is the trade-off between ROI masking and acceptable variation in lane line location. Tighter ROI's filter out more noise, but also reduce where the lane lines may be found in the images. The chosen ROI dimensions are effective for the given test images/videos, but may not be robust to other image sets.

Finally, the nature of the assignment is to detect lane LINES, but real-world lanes within the driver's field-of-view are curved. So a truly robust solution would need to detect curves rather than lines.

3. Suggest possible improvements to your pipeline

A possible improvement to the outlier sensitivity issue would be to compute lane lines from the edge-map data directly, rather than using Hough Line detection.

- Apply RANSAC algorithm to extract edge points consistent with a linear model.
- Compute best-fit line from the extracted edge points, via a least-squares fit.
- For even better accuracy, apply an "iteratively re-weighted" least-squares fit with a loss function. (see OpenCV fitLine() for details)

An improvement for the lines-vs-curves issue would be to least-squares fit a spline curve to the edge data. I suspect this is what "industrial strength" algorithms do, but it's obviously beyond the scope of this assignment.