Finding Lane Lines on the Road

Andreas Gotterba

The goals / steps of this project are the following:

1. Make a pipeline that finds lane lines on the road
2. 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 consisted of 5 steps:

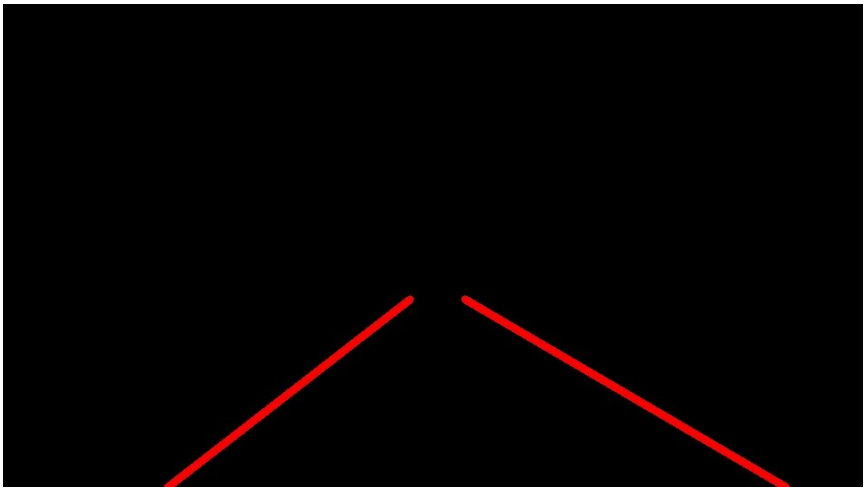
1. Convert image to gray-scale, so that the canny transform will give simple, predictable results.

2. Apply Gaussian blur, to remove high frequency noise (beyond the blur that the canny transform will apply). I'm applying a rather heavy kernel of size 7, since the lane lines are still clearly visible for this case.

3. Use Canny edge detection to detect well-defined edges.

4. Cut out the parts of the image that aren't of interest. In addition to the trapezoidal cutout we used in class, I'm cutting out a triangle inside the lane, since we don't expect lane lines in the bottom-center area. Still, it's not perfect, and can't be, since lane-lines will vary (which is why we need to find them!)

5. Apply the Hough transform to detect lines. At first the lines detected by the Hough transform are simply drawn on the image, but after modifying draw_lines, they're simply used as input data to derive singular, strong, straight lines



Here’s the final result:

In order to draw a single line on the left and right lanes, I modified the draw_lines() function to use linear regression to derive a single lane line for each side. I do this by:

1. Separating the lines detected by the Hough transform into left and right sides, based on the lower x coordinate. If there's a bend in the road, I'm willing to allow the top of the lane line to cross the center (though with a dotted line, what I'm doing isn't sufficient).

2. Filtering lines based on their slope. If the slope is too far from expected, eg, if the line slopes towards the outside of the image, then I discard it. With my current settings, this filter is pretty permissive.

3. Breaking the lines for each side up into individual points (ie, I calculate a y-value for every x-coordinate along the line). I do this because the Hough transform will have marked a broken line as two lines, if the break exceeded my max_gap, and I don't want that line to 'vote' more strongly than a solid line, as it would if I only used the endpoints. If a line is longer than the others, it makes sense for it to be given more weight.

4. With this collection of points for each side, I perform a linear regression to find the single, straight line that minimizes the sum of least-squares error to the lines’ points.

5. With the equation for the line of best fit from the linear regression, I calculate the start and end points that fit within the y coordinates of my region of interest. I’m willing to let the x coordinates fall outside the region: if that’s where the lane line has been found, I’ll go ahead

and mark it as such. But bounding the y coordinate is important to not extend the line beyond where I gave the Hough transform input data (or even into the sky). I do this by setting the x coordinates to the edge of image and calculating the y coordinates, but as is almost always the case, these y coordinates are out of bounds. I detect this, and recalculate the start and end x coordinates for the y coordinates at the bounds of the region.

2. Identify potential shortcomings with your current pipeline

1. By filtering the region that I'm looking for lines in, I'm expecting the car to already be positioned in a well-defined lane. The more permissive this region can be while still getting good results, the better the lane-line detector would handle more challenging situations. Even if the car is well positioned in a lane, all the examples we've worked on have been on freeways. A sharp bend on a smaller road would be more challenging

2. As per the instructions, my updated draw_lines function draws a single *straight* line for each side of the lane. This assumes that straight lane lines isn't too big of an error, and for the cases we've looked at (on the freeway), that isn't a bad assumption. But it wouldn't be appropriate for other roads.

3. One place I've seem some hiccups in my lane lines is on dashed line, when the first dash is relatively far away. Without data closer in, the far part of the lane controls the direction of the line, which can lead to more obvious error closer to the camera, at least in some of the video frames.

3. Suggest possible improvements to your pipeline

1. To detect the lane-line, the canny edge detection is looking for single, high-contrast transitions. Indeed, for the lines close to us, we see it detect both edges of the lane-line. If its edge detection was signed, we could detect a dark-light-dark transition (imagine we're scanning the image from left to right) within a reasonable distance for the width of the lane line, instead of just light-dark and dark-light. That would help with the challenge video, where my pipeline gets confused by the transition from concrete to asphalt.

2. Since I'm using linear regression to improve draw_lines, it would be simple to use the error values for each point to calculate a confidence estimate for how certain I am that I've found the lane-line correctly.

3. With more test images, I could refine the parameters. I'm happy with my performance on the still images, but see a few hiccups in the videos, and don't do well with the challenge video until the end. To debug these cases, I'd like to take single problematic frames and see what output I'm getting from Canny edge detection and the Hough transform before the linear regression.

4. It would be easy to use a higher-order regression to allow the simplified lane-lines to curve.

5. There's been some talk on Slack about using data from previous frames to fill in missing data on the current frame. I think this could help with dashed lines, if the dash is missing from the bottom of the frame.