Finding Lane Lines on the Road

Neeraj Phadnis

I am submitting two notebooks:

- Project1_Image.ipynb: This notebook contains my pipeline in separate cells which was convenient for development and troubleshooting. I will reference cells from here below.
- Project1_video.ipynb: This notebook contains my final **process_image** function pipeline and subsequent video processing.

My **process_image** function pipeline has the following steps:

1) Grayscale the image: Here I grayscaled the image for ease of identifying lane lines later during Canny edge detection. I used cv2.COLOR_RGB2GRAY because importing images using matplotlib results in an RGB image (Project1_Image.ipynb cell 3).
2) Gaussian smoothing: Here I sharpened image pixels. I used cv2.GaussianBlur with a kernel size of 5 similar to the lecture example (Project1_Image.ipynb cell 4).
3) Canny edge detection: The pixel gradient across edges within the image (in white) and the background (in black) can be detected and isolated with Canny edge detection. I used cv2.Canny with low and high thresholds of 50 and 150 respectively. This combination seemed to best detect the lane lines in the sample images (Project1_Image.ipynb cell 5).
4) Creating masked edges: This step creates a blank image with just the lane markings as a "mask" of white pixels. I created a white polygon that best crops out the portion of the road with lane lines. Then I performed a bitwise AND operation between the polygon and lane lines (Project1_Image.ipynb cell 7).
5) Hough Transform: This step transforms image pixels from the XY coordinate system into the Hough space where they are represented as lines. Intersecting lines in this space represent collinear points in the XY space. That helps group multiple points into one straight line. Many such lines together form one lane line. I began with a high value for *threshold* (minimum number of "votes" or intersections in the Hough grid cell) before realizing that lowering this value helped identify more points along the lanes. Similarly reducing *max_line_gap* helped collect more points together as a single line. At this point, some stray lines not in the direction of the lane lines began appearing in the output image. These were bright patches on the road or short square lane reflectors being identified as complete lines. Increasing *min_line_length* helped discard these stray lines but I realized that some useful pixels were being discarded too. Making the Hough grid finer by reducing *rho* and *theta* helped better discard these lines without having to

raise *min_line_length* too much. It did not seem to increase image processing time by a lot either. The output was an array containing endpoints (x1, y1, x2, y2) for each line segment (Project1_Image.ipynb cell 9).

6) Identify left and right lanes: My strategy was to separate the line segments belonging to the left and right lanes and then use each set of points to draw a best fit line over each lane. In my first attempt, I calculated the slope of each line segment and hoped to classify lines with a negative slope as left lanes and positive slope as right lanes. This approach worked for most images but there were certain images where a square lane marker was partially present at the bottom of the image (even though I attempted to discard them in step 5). During the Hough Transform, this marker became a line segment with a slope opposite to the other segments in the same lane and got classified into the wrong lane. On performing a polyfit operation, the final lane line cut across the image. My second attempt classified line segments by checking whether they lay in the left or right half of the image respectively. This performed well on all images (Project1_Image.ipynb cell 12).
7) Draw lane lines on original image: During the polyfit operation, I identified the endpoints of each lane line within the polygon defined in step 4. Using cv2.line, I drew both lines over a blank image and took a weighted average between this and the original image to superimpose the lines over the original lanes (Project1_Image.ipynb cell 14).

Some steps in my pipeline became specific to the kind of lanes present in the assignment videos; continuous lanes or fairly long line segments in a lane. It occasionally struggled in the presence of smaller lane reflectors and I would expect it to struggle even more if only these lane reflectors were present. Identifying them would require a different tuning of the Hough Transform parameters.

Also my approach of classifying left and right lane line segments would fail during sharper turns that cause the lane line to cut across the center of the image. I would have to reduce the size of my polygon from step 4 which would compromise how much further from the car can the pipeline identify lanes.

I am certain there is a more efficient way to code step 6 instead of my solution. There is a lot of code repetition that seems unnecessary but I am limited in my Python skills!

I found my final videos to have some discontinuity between augmented lane lines in consecutive images. One area for potential improvement could be to apply some filtering on the lane coordinates from one image to another or have a common start point for all lane lines at the bottom of each image.