Finding Lane Lines on the Road

SELF – DRIVING CAR ENGINEER

Sai Subhakar T | 11-05-2018

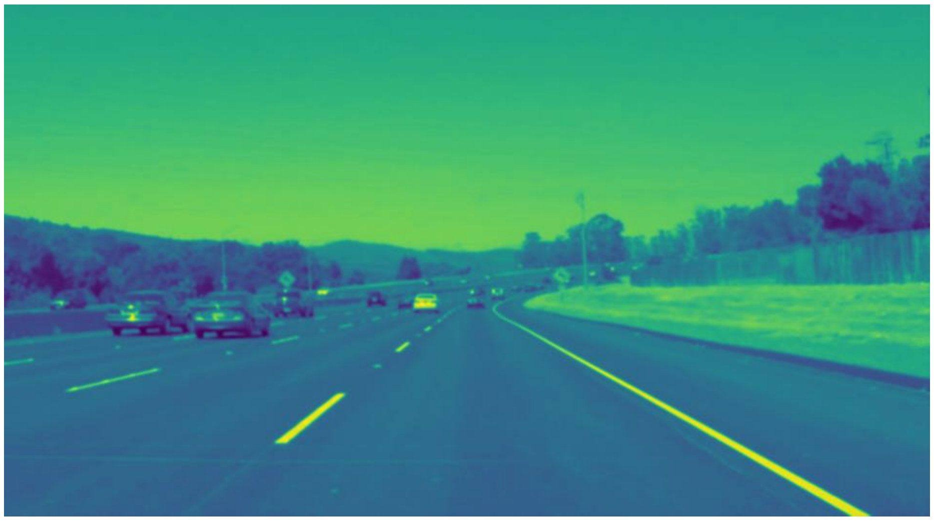
My Pipeline:

1. Image Processing: As a Signal Processing student, this part is very crucial in determining several characteristics of an image. The following has been done in this process:

- Converting the original image into **gray scale** as it will be easier to find it's gradient later on.
- Suppressing noise by **Gaussian Blur** algorithm.

This algorithm from OpenCV averages gradients and suppresses additional noise in the image by choosing the kernel_size as a parameter. A larger kernel_size implies averaging, or smoothing, over a larger area (5 in this case looks better).

Fig: Solid white curve after gray scaling and blurring:

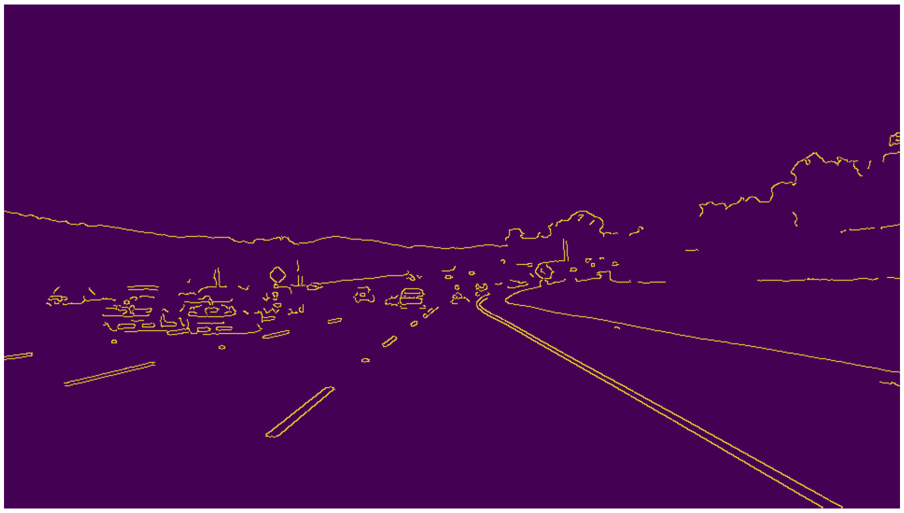


2. Canny to Detect Lane Lines: This algorithm detects the edges in an image by means of applying thresholds and taking gradients (rapid changes in brightness). This uses two parameters namely, low_threshold and high_threshold to pass a detected edge or not.

Generally, this algorithm will first detect strong edge (strong gradient) pixels above the high_threshold, and reject pixels below the low_threshold. Next, pixels with values between the low_threshold and high_threshold will be included as long as they are connected to strong edges.

As far as a ratio of low_threshold to high_threshold, **John Canny himself** recommended a low to high ratio of 1:2 or 1:3.

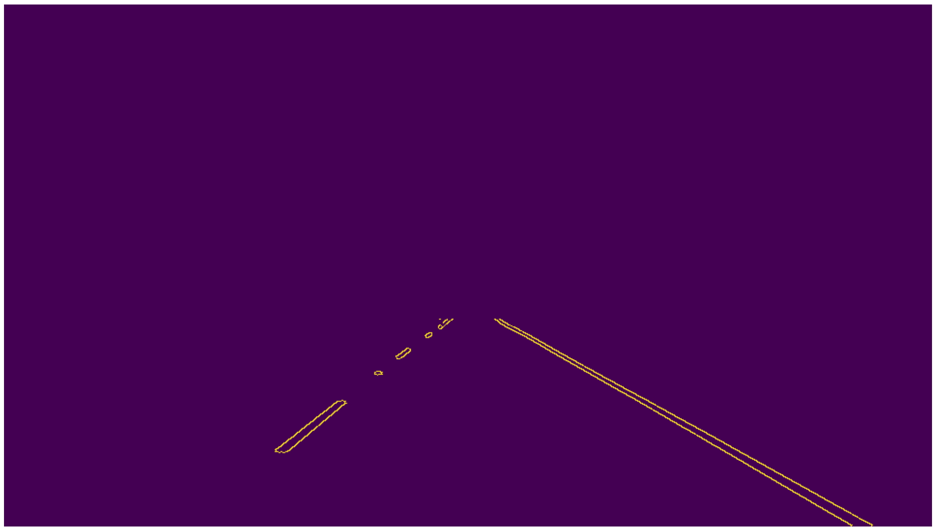
Fig: Solid white curve after Canny:



3. Region Masking: In order to avoid overlapping of lane lines and detection of any unnecessary objects, it is important that we specify the region of interest in an image. We only consider pixels where we expect lanes lines to be, in this case, from a mounted front facing camera.

In this step, We define a four sided polygon which can mask out extra edges apart from our region of interest. Parameters like vertices of the polygon are considered (left_bottom, right_bottom, left_top, right_top).

Fig: Solid white curve after region masking:



4. Hough Transform to Find and Extract Lane Lines from Canny Edges: We all know that the equation of line is $y = mx + b$ and in order to connect all the dots in edges detected from Canny to form a line, We need to consider the equation of line and Hough Transform.

The Hough Transform is just the conversion from image space to Hough space. So, the characterization of a line in image space will be a single point at the position (m, b) in Hough space. In Hough space, I can represent my "x vs. y" line as a point in "m vs. b" instead.

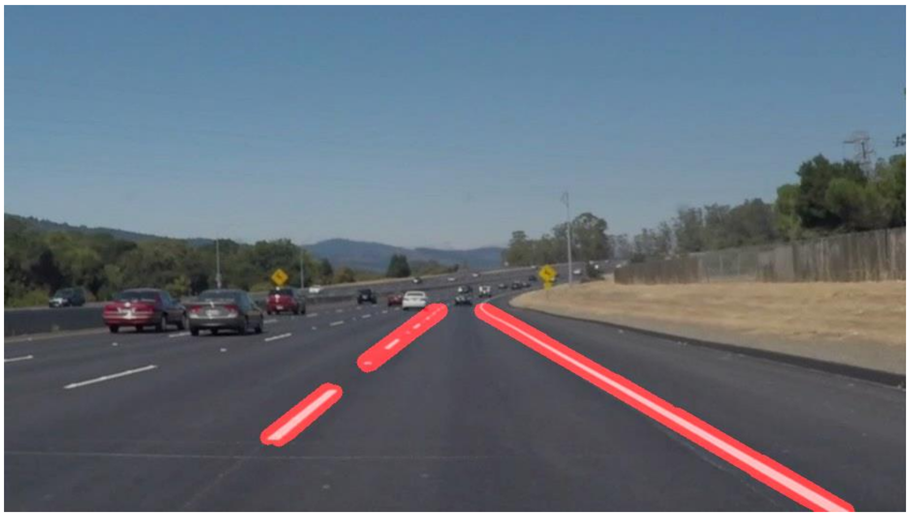
Parameters:

- rho and theta - distance (pixels) and angular resolution (degrees) of our grid in Hough Space. Reasonable values are 1 for both and however, tuning is recommended.
- threshold - it specifies the minimum number of votes (intersections in a given grid cell) a candidate line needs to have to make it into the output.

- min_line_length - is the minimum length of a line (pixels) that we accept in the output.
- max_line_gap - is the maximum distance (pixels) between segments that we allow to be connected into a single line.

5. Averaging and Extrapolating the Lane Lines | Improving the draw_lines function: The output from Hough Transform is shown below which is just a plot with small lines. We need to map out the full picture (full extent of the line till the end) for our Self-Driving car to better help in sensing. This is where averaging/extrapolating the lines comes into action!

Fig: Solid white curve defined just from helper functions gives us only small lines: (Hough Transform Output)



➢ **draw_lines function:** To extrapolate the lines, the following modifications has been done in the draw_lines function:
  - First, we define the empty lists to work with (as seen in code and references).
  - Then we separate the small lines x and y into two groups to find out if they belong to left or right part of the image.

- By considering the slope of the line equation, $(y_2 - y_1)/(x_2 - x_1)$, We consider values from left or right parts depending on the slope is positive or negative.
- We then use polyfit function to fit a line to these points.
- We now take average of slope and intercept line and derive upper and lower parts of the image (as seen in code).
- Finally, we plot the single solid line on both the lanes as output. Some of the examples are shown below:

Fig: Solid white curve after averaging:

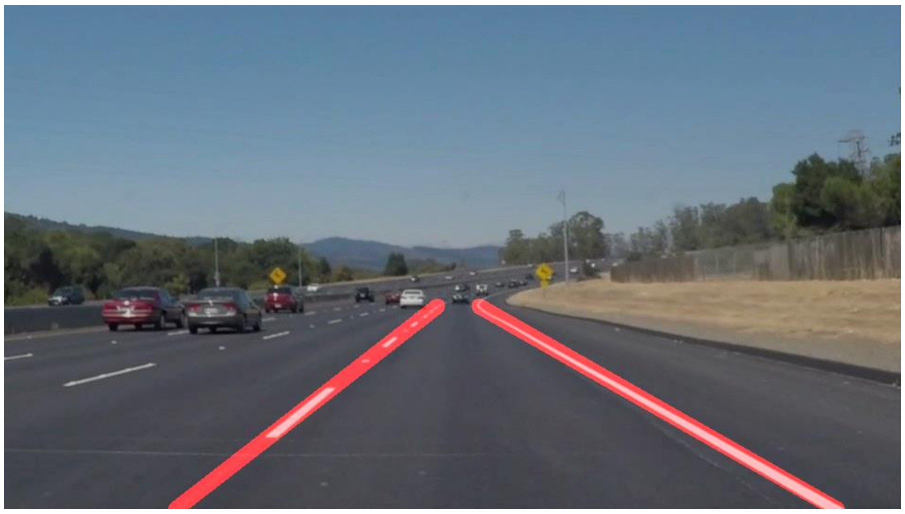


Fig: Solid yellow curve after averaging:

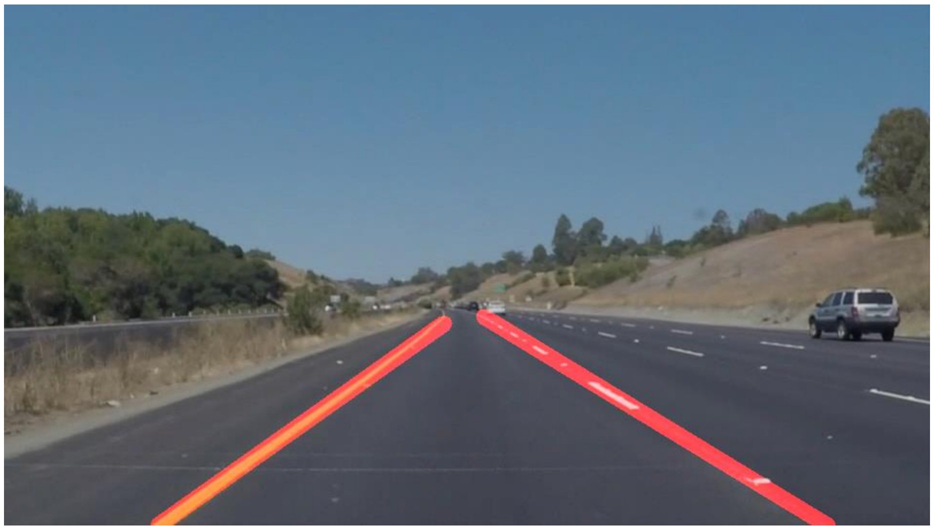


Fig: Solid yellow left after averaging:

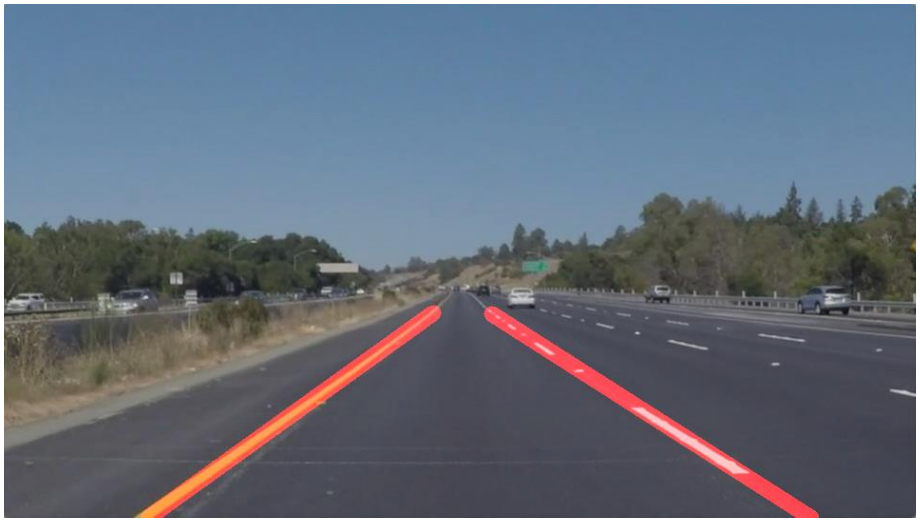
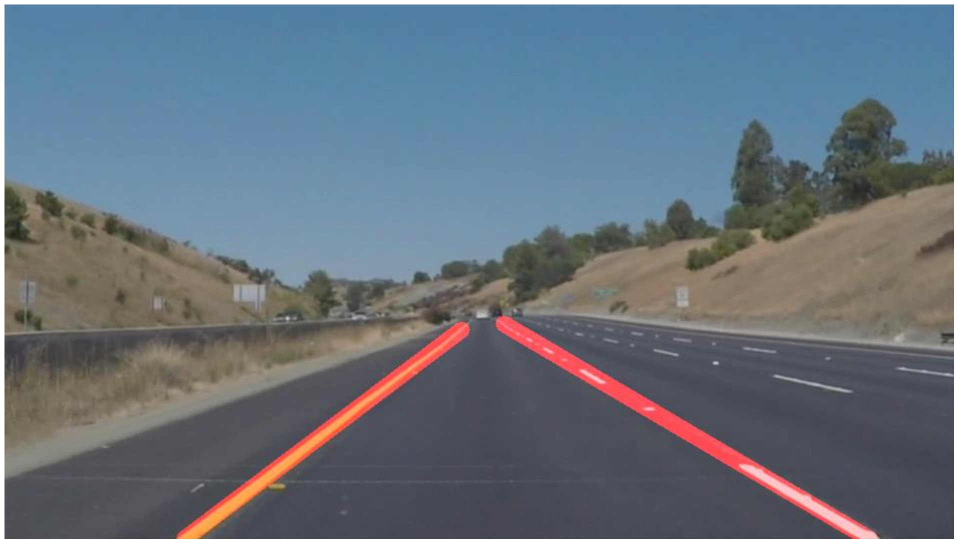


Fig: White car lane switch after averaging:



Potential Shortcomings:

- Our Self-Driving car probably need more than a polygon mask to increase it's perception of roads. Like roads with curves, damaged lanes, various night time-only scenarios, need more than a mask and straight lines for our car to figure out how to actually rule the roads.
- Canny edge detection is great as it goes but is not reliable as it can detect objects other than lane lines. I think our car can manage up to most extent but at some point, it will face trouble classifying objects to lane lines.
- Lane lines intensity (low, high, dim, shadows etc) varies accordingly by changing Canny's threshold parameters. It is not flexible as it can mess up with the gradients and lines might not even be detected in some cases. (causes flickering)
- The lanes could intersect and overlap with each other if the region masking isn't done accurately.
- Parameter tuning is too sensitive and above all can seriously mess up in refining line extrapolation.

Possible Improvements:

- Evaluating a deep learning approach to update all parameters independently with respect to changes in real-time.
- Introducing a reliable classifier to detect objects and classify them with respect to lane lines.
- Optional: We can use a median filter to remove further noise and blobs between the video frames. A matlab example which I've used for my background detection project:

```
(FilteredImage=medfilt2(Binaryimage,[5 5]);#applies med
filter to output image (binary in this case with order 5)
```

- Converting the image into a different color space like HSV, HSL can help in robust object discrimination in conditions with varying light intensities, shadows etc.
- Threshold parameters have to be refined to perfection to avoid any possible flickering in the video frames.
- And of course, spending more time on tuning the parameters!

References:

https://peteris.rocks/blog/extrapolate-lines-with-numpy-polyfit/

Thank you for an amazing project,
Sai