Vincente Nguyen
February 2019

Finding Lane Lines on the Road

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

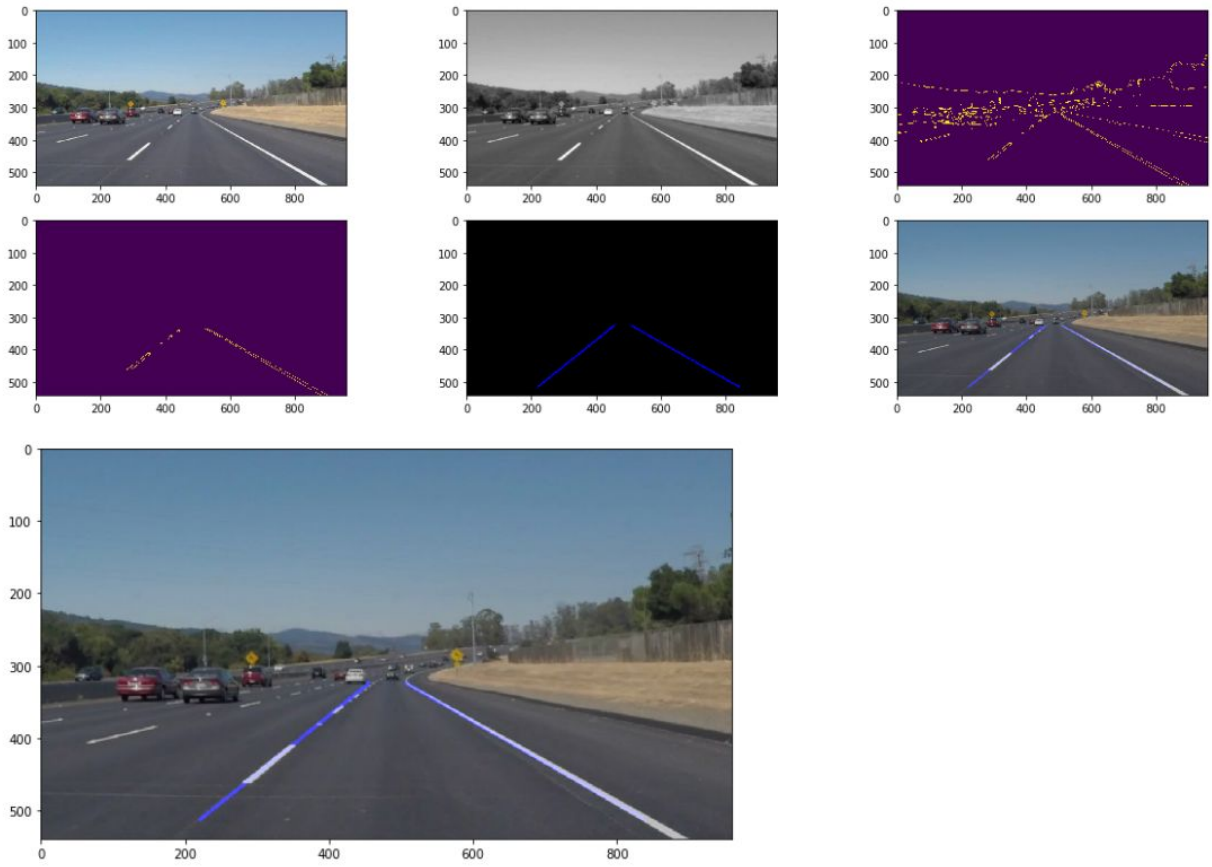
In order to draw lane lines onto the road I had to perform the following steps:

1. Convert the image or video frame to a gray scaled image to reduce color channels and bits.
2. Afterwards the gray scaled image is fed into a Gaussian blur function to smooth out the image of extra noise enabling easier edge detection.
3. With this “smoother” image we apply the Canny transform to find edges in an image using a low and high threshold. Canny himself recommends a 1:2 or 1:3 ratio, I personally chose 1:3 (50:150) as I thought that gave me the best picture.
4. Afterwards we “crop” or “mask” the image to only use the area of the image that we find useful i.e. the road and its lanes in this case.
5. Next we we use the Hough transform to convert our image from image space to Hough space to find where the lines intersect. These intersecting lines are what we are interested in as we can use this to draw lines back onto our image in image space.
6. Finally, we take our originally image and the image we got from the Hough transform and merge them together to show lane line markings on the original colored image.

The draw_lines() function was modified to enable the algorithm to “guess” based on the slope of found points or lines to fill in the area where we don’t see any lane markings. This is done by separating both sides of the lanes by determining if the slope is negative or positive. From here we basically extend the line to the bottom of the image and to roughly 60 percent of the image where we can no longer really see the lines.

Vincente Nguyen
February 2019

In the matplotlib plots below you can see each individual step done to produce lane lines on the original image.



Because a video really is just a series of photos we can apply this same technique / pipeline onto a video by separating out the frames and feeding it back into this pipeline and rewriting the video with the “merged” pictures.

2. Identify potential shortcomings with your current pipeline

The current pipeline is rather slow, applying to a live stream for example on a real self-driving car would require significant processing power or to tweak the algorithm or hardware. Maybe something like slowing down the frames per second on a camera to allow the pipeline more time to process.

Another shortcoming is that [obviously] this algorithm will not work on curved lanes due to the way I currently draw and interpret lane lines that are not there. This can be seen in the challenge video. My current best guess on how to do this would be calculate the slope over significantly shorter lines to be able to draw a curved line better.

Vincente Nguyen
February 2019

3. Suggest possible improvements to your pipeline

Regarding my pipeline if I had more time I wouldn't have put all my subplots in my pipeline itself and made it into a seperate display function to be called. This is because when the video is called it calls the pipeline on every frame and spams the Jupyter notebook of subplots. In my final submission I commented out the subplots so the project grader doesn't have to get spammed on every video frame of subplots. Feel free to uncomment and see the cool plots.

Some possible improvements code-wise would be why don't we give masking or cropping the image at the very beginning a try? Would that not enable the algorithm less area to work with thus speeding up lane line detection?

My draw_lines() function could have been broken up into more singular functions to enable better reuse for future projects, but oh well.