




No description, website, or topics provided.


Edit

[Manage topics](#)

 17 commits

 1 branch

 0 releases

 1 contributor

Branch: master ▾











New pull request



Create new file

Upload files

Find File

Clone or download ▾

| | |
|---|--|
|  ameter Merge branch 'master' of github.com:ameter/CarND-LaneLines-P1 | Latest commit 60c8334 on Nov 29, 2017 |
|  .ipynb_checkpoints | added writeup images a year ago |
|  examples | initial commit 2 years ago |
|  test_images | Working pretty well. Good enough to turn in finally. 2 years ago |
|  test_videos | initial commit 2 years ago |
|  test_videos_output | cleaned up notebook a year ago |
|  writeup_images | added writeup images a year ago |
|  P1.html | added html of notebook a year ago |
|  P1.ipynb | added writeup images a year ago |
|  README.md | Update README.md a year ago |

 README.md 

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 consisted of 6 steps. First, I converted the images to grayscale. Second, I applied a gaussian blur. Third, I performed Canny edge detection. Fourth, I applied a mask to the region in front of the camera where I wanted to find lane lines to follow. Fifth, I applied a Hough transform to identify the lines. Sixth, I overlaid the identified lane lines on the original image.

I set a number of tunable parameters. I did this as a separate function because I wanted to set them all in one place. This simlified tuning as I was able to tweak the parameters and then run my test images or video through my pipeline by only re-executing two cells.

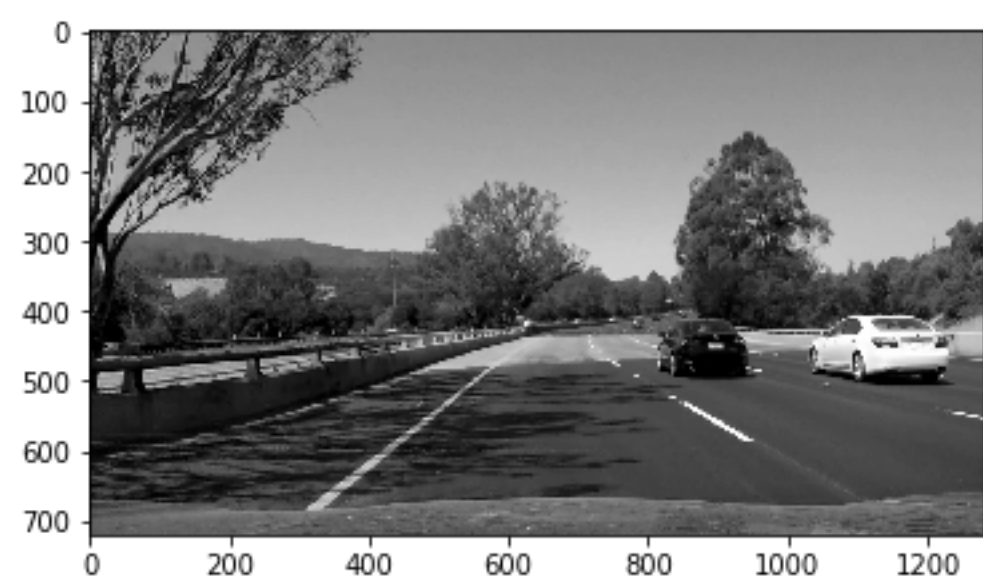
In order to draw a single line on the left and right lanes, I modified the draw_lines() function to bin all the identified line segments into 2 bins (left and right) based on whether they had a positive or negative slope. I then extended and averaged the lines in each bin to form a single left and right line.

Below is an example frame at each stage of my processing pipeline. Note that this is one of the more challenging frames from the challenge video.

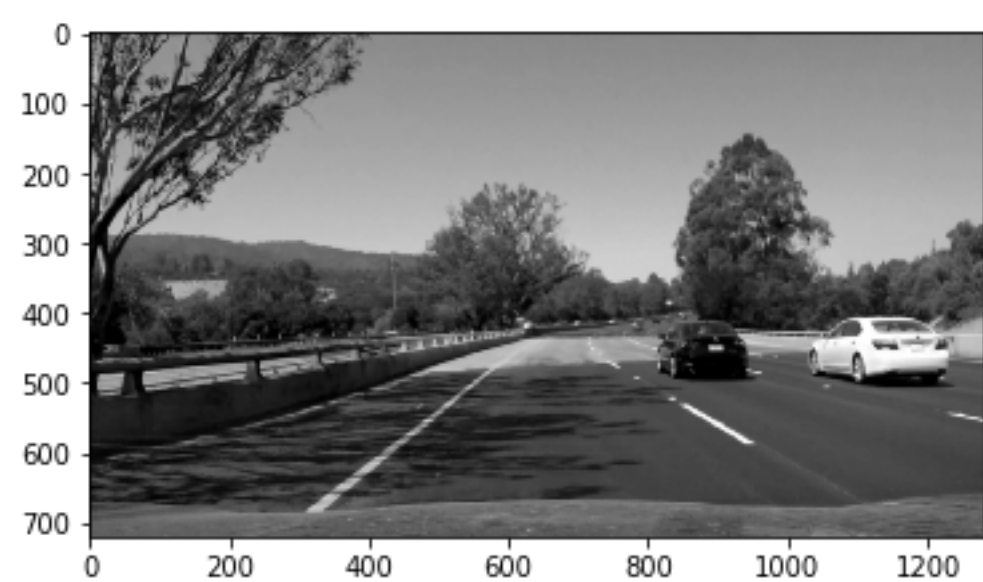
Input frame from challenge video



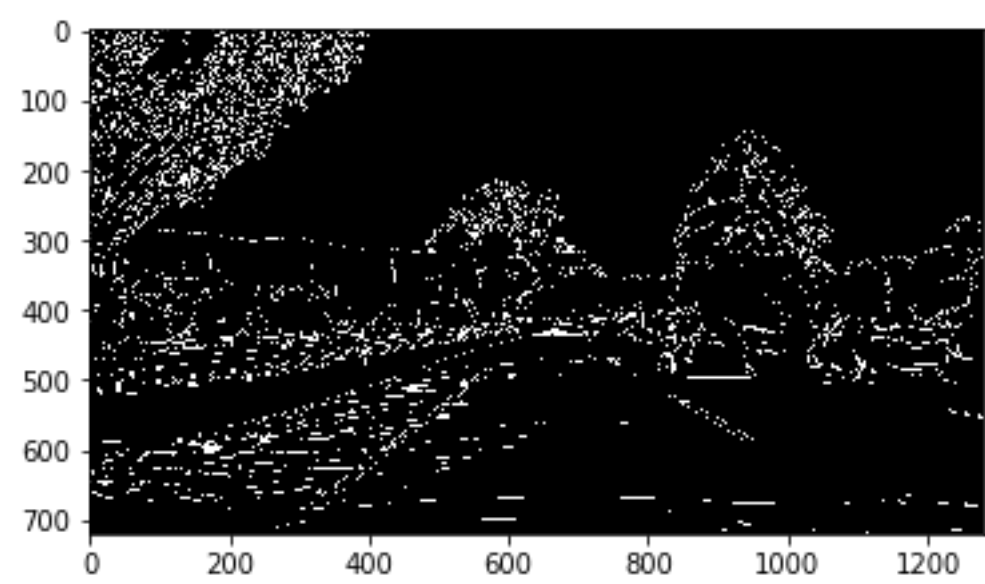
1. Grayscale



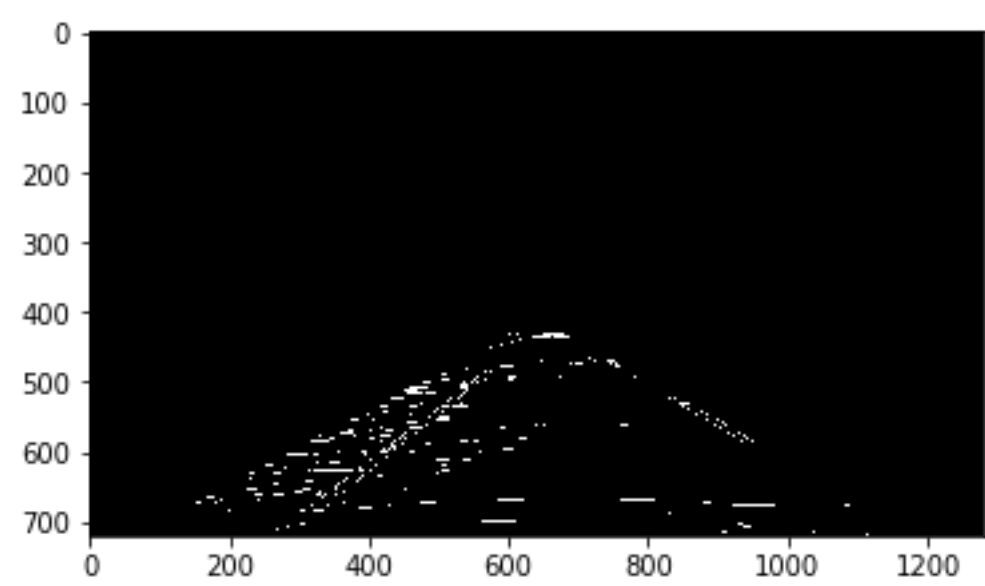
2. Gaussian blur



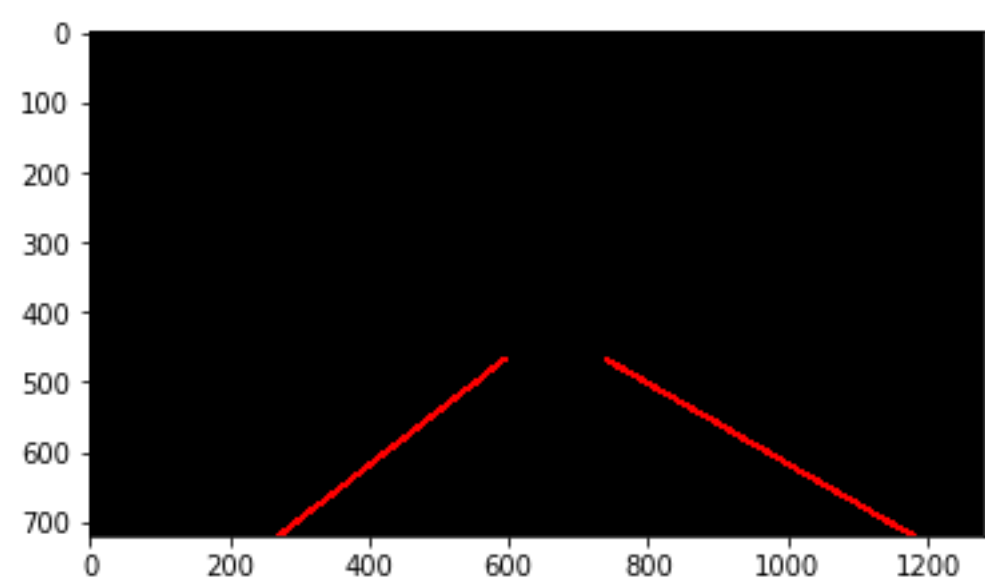
3. Canny edge detection



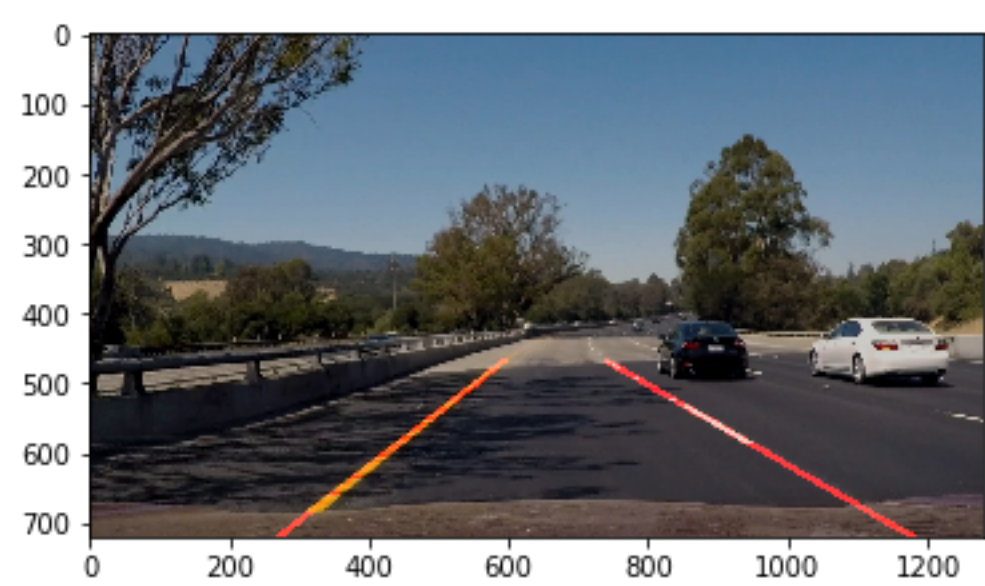
4. Region mask



5. Hough transform to identify the lines



6. Result



2. Identify potential shortcomings with your current pipeline

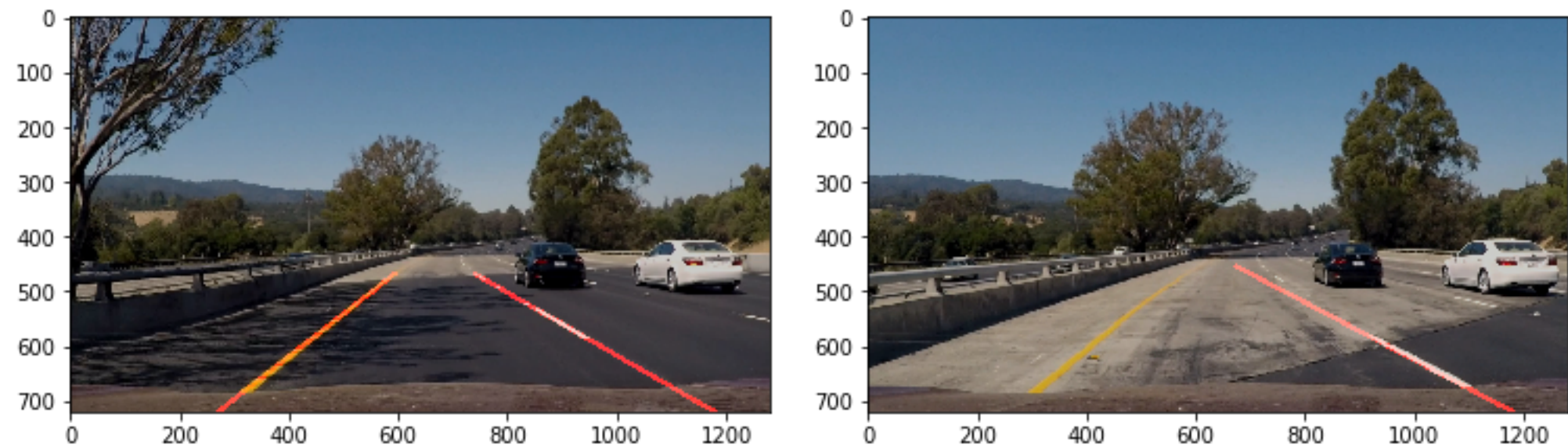
One potential shortcoming would be what would happen when there are no lane lines. Other techniques would need to be employed to follow the road.

Another shortcoming could be when there is a lane split, as in certain situations at Interstate exit ramps. In this case, the car needs to decide which lane to follow.

Finally, as seen in the challenge video, there are situations where there are multiple lines on the road (that are not lane lines) caused by shadows, changes in pavement types, bridges, etc. More work is needed to confidently address all of these types of situations that can and will be encountered in a real world situation.

The challenge video was challenging. I tried a number of techniques, to include tuning all my parameters, refining my mask area, etc, to improve performance on the challenge. Most significantly, I wrote an outlier defeat function that dropped outlier line segments before averaging. This helped, and in the end, my pipelines performance on the challenge video was pretty good, with a couple of glitches on particularly troublesome frames. I attempted to address this by capturing still shots of the most problematic frames and attempting to tweak my pipeline to handle them. In the end, it was close, but not perfect.

Here are 2 examples of the most problematic frames from the challenge video:



3. Suggest possible improvements to your pipeline

A possible improvement would be to give additional weight to longer line segments instead of simply taking the average of line segments when combining them.

Another potential improvement could be to maintain some sort of state information between frames to favor continuity of lines between frames when multiple candidate lines (shadows, etc) are present in a frame.

Ultimately, I believe some more advanced techniques, such as deep learning with a Convolutional Neural Network, are needed to create a truly robust lane following capability.