1. Description of the final pipeline

The final pipeline for lane line detection is described below:

i. In this step, the pipeline resizes an input image so that it can automatically adapt different camera resolutions. In this step, an input image is resized by `cv2.resize()` function.

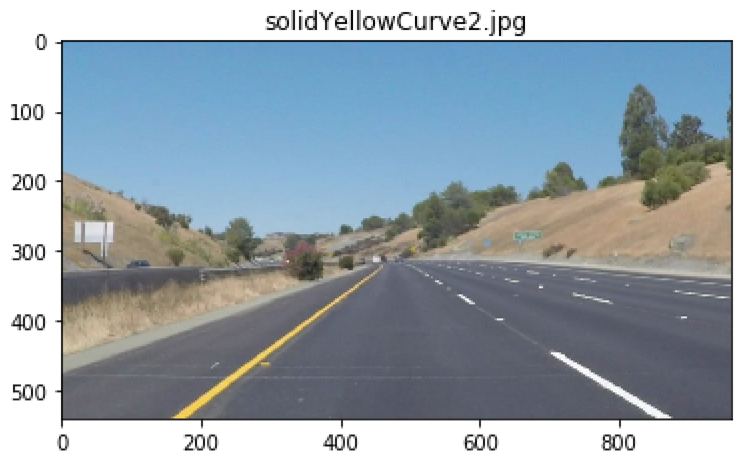

Image 1. A resized image

ii. In this step, the pipeline applies a HSV color filter to the image that is passed from the previous step. In this step, `cv2.cvtColor()` is used to convert the image to HSV format, and `cv2.inRange()` is applied for the color filter. This step was a key to improve the performance of lane line detection especially for **the optional challenge**. In the optional challenge, there is a border of shadow over the car lane that makes lots of noise in an output from an edge line detection algorithm such as canny. By applying a HSV color filter, such noise can be drastically reduced.

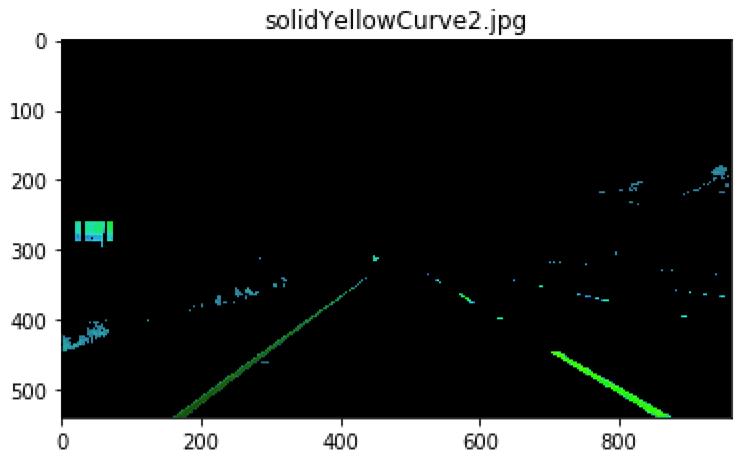

Image 2. An image after the color filter

iii. In this step, the pipeline applies canny edge detection algorithms to detect edges from the image passed from the previous step. There are 2 transformations before the canny detection algorithms. First, `cv2.cvtColor()` is applied to convert the input image to grayscale. This is needed because the canny edge detection algorithm only takes a grayscale image as an input. Second, `gaussian_blur()` is applied with `kernel_size=5` to reduce noise for the canny edge detection. After these functions, `canny()` is applied with `low_threshold=100` and `high_threshold=150`. The following image is a sample output from this step.

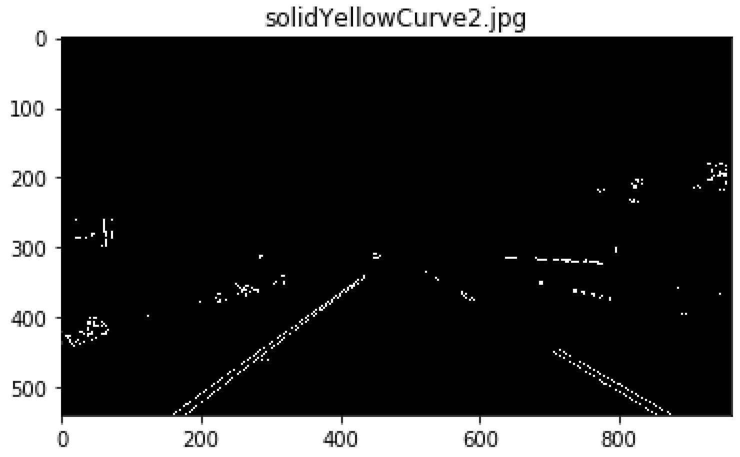

Image 3. An image after canny edge detection

iv. In this step, the pipeline extracts the region of interests from the image that is passed from the previous step. Since the left and right lanes are always seen in a trapezoid area in the image, the pipeline extracts the objects in the area. As shown in the following image, only the lane lines are visible after this step.

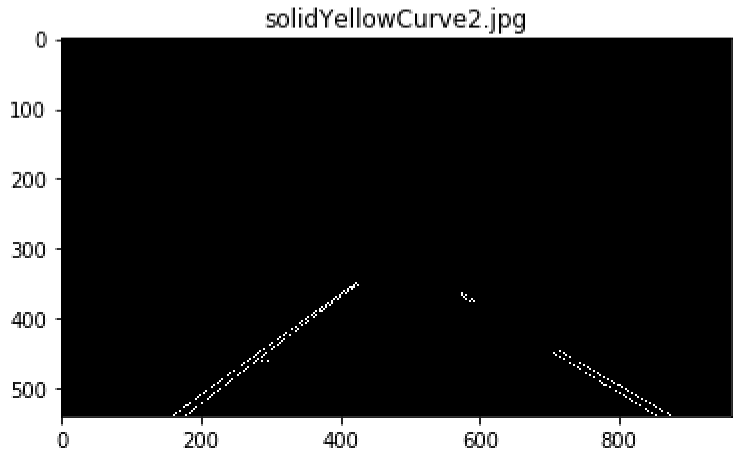

Image 4. An image after extracting the region of interest

v. In this step, the pipeline draws lines of lanes based on the result of the lane detection algorithm. Before drawing the lines, the pipeline applies 1) Hough Line Transformation and 2) Linear Regression to calculate coefficients and bias of the lines. Since the output of Hough Line Transformation is not a single line segment for each left or right lane but just many line segments on lane lines, it was necessary to use Linear Regression to find the line segments which represent line segments on left or right lane.

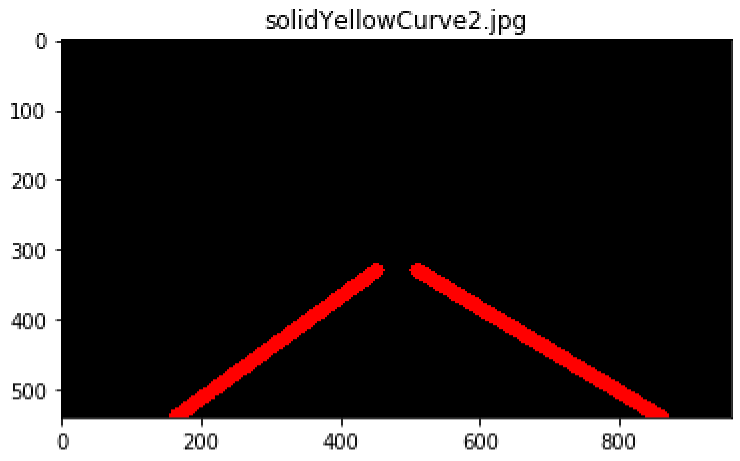

Image 5. Line segments calculated in step 5

vi. In this last step, the pipeline overlays calculated line segments over the resized image. To do this, a `weighted_img()` function is used.

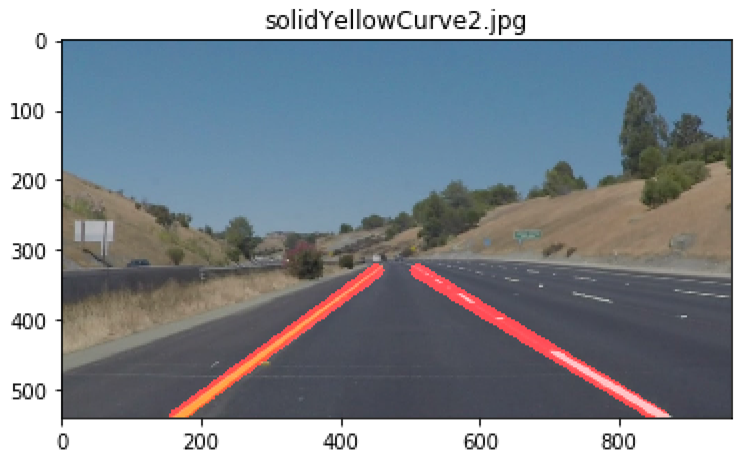

Image 6. Line segments overlayed on the resized image

2. Potential shortcomings with my current pipeline

As seen in the optional challenge, the lane line detection is broken when there is an unexpected scene in a video by the algorithm. For example, the algorithm was easily broken with a bordar of shadow in the optional challenge. In addition to a border of shadow, the algorithm will be broken by different weather, different light conditions and many cut-in vehicles.

3. Suggest possible improvements to your pipeline

In this practice, I’ve implemented rule based lane line detection algorithm by combining several opencv features. However, a rule-based approach needs lots of manual work, and makes it difficult to adapt to many edge cases. To reduce the amount of manual work, I suggest applying a machine learning based approach with a rule-based approach. With a machine learning based approach, you can reduce your time to manually create rules if you have a bunch of training images.