Finding Lane Lines on the Road

The goal of this project 1 of CarND course is to detect the basic road lane lines on the image and videos. 6 test images and 3 test videos (1 challenge) are provided to test the final pipeline.

The project is done at 2 main steps:

I. Detect the lines on the images,

II. Extend the pipeline developed for lane detection to videos

---

The following test images are used:

Reflection

I. Detect the lines on the images:

My pipeline consisted of the following main steps:

1) Edge Detection
    a. Convert image to grayscale
    b. Apply Gaussian blur
    c. Apply Canny function
2) Lane Detection Using Hough Method
    a. Select region of interest
    b. Apply Hough method to detect the lanes
    c. Add lines to the original image
3) Extending the Lanes
    a. Split the lanes to left and write
    b. Find the weighted average of left/right lines slop and intercept
    c. Extrapolate to extend
    d. Add the extrapolated lines to the original image

Each step and sub-step is explained below and the output of each step is shown:

1) Edge Detection Pipeline:

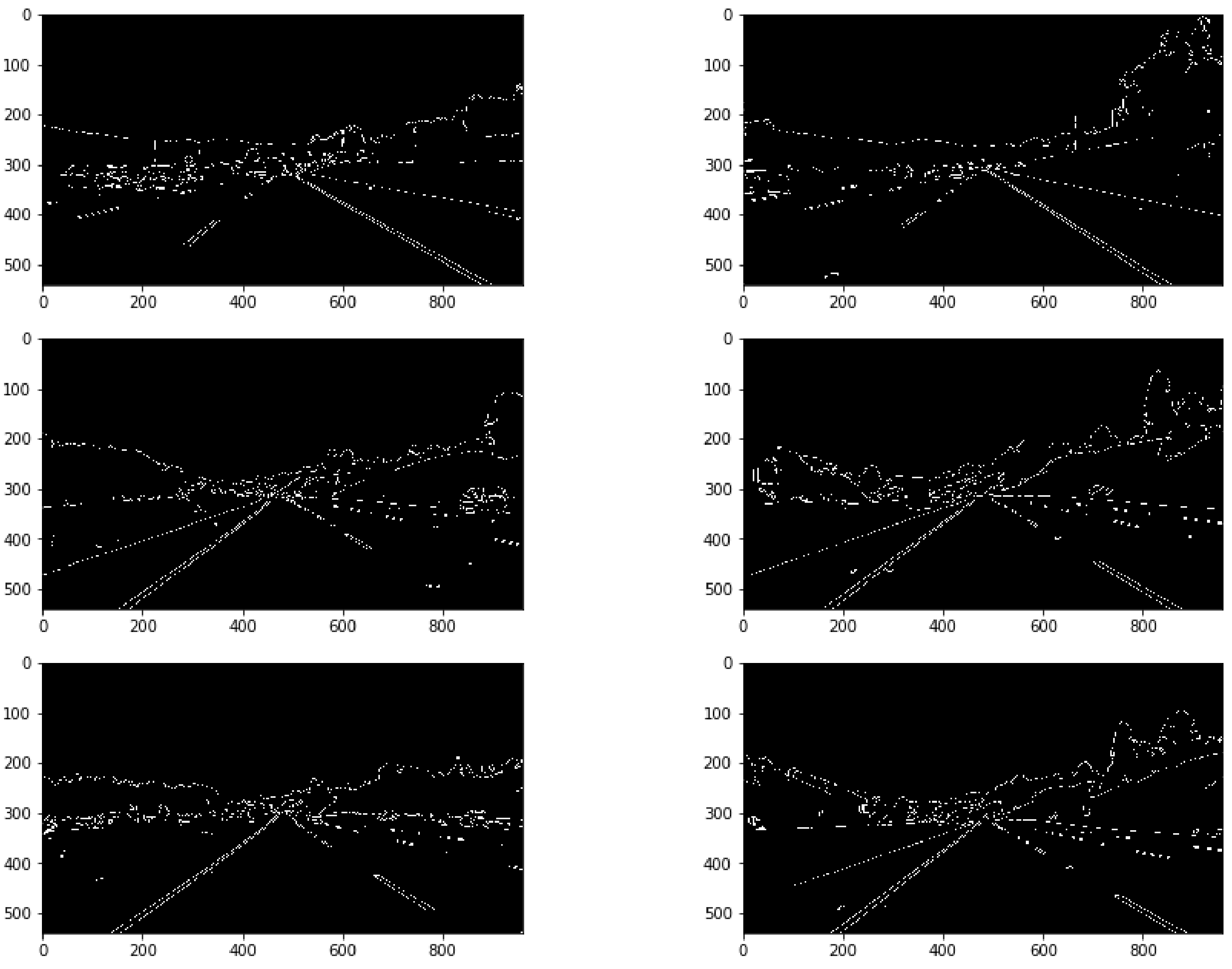
The pipeline below implements Canny edge detection algorithm for all 6 images. It has 3 steps:

1.a. Convert the images to grayscale (since canny is more efficient with greyscale).

1.b. Apply Gaussian blur to smooth the image colors a little, a threshold of 5 is used. A lower value doesn't filter all the noises in the image and a higher value will over-smooth the image and doesn't allow the canny to detect images.

1.c. Apply canny function to filter the gradients beyond 60 and 150 (color change per pixel).

The output of this step is below:



2) Lane Line Detection Pipeline:

Hough line detection algorithm is used to detect the lane lines. In fact, the Hough method is applied to the output of previous pipeline, Edge Detection pipeline. However, since there're many lines around that we're not interested in, first a region of interest of the image is selected and then the Hough method is applied to the region of interest to detect only the lane lines. This pipeline has 4 steps:

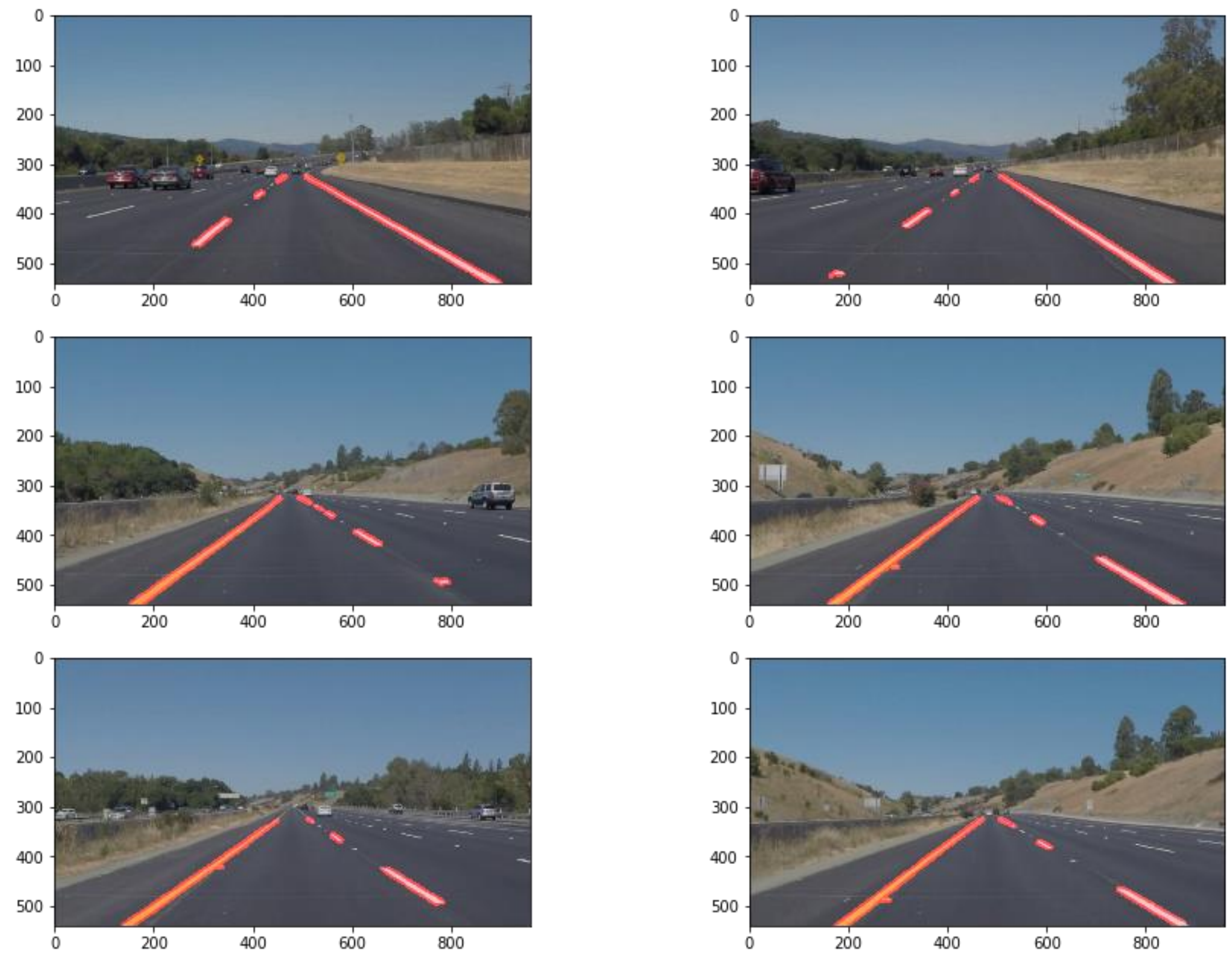
2.a. Apply Region of interest detection: since the camera location is fixed, a fixed region of the image works (at least in this case) interestingly well for all the images!

The function Reg_of_Int is added to encapsulate some part of the code related to the Region of Interest selection.

```
In [3]: import math

        def Reg_of_Int(image):
            ysize = image.shape[0]
            xsize = image.shape[1]
            bottom_left  = [xsize*0.13, ysize*1.0]
            top_left     = [xsize*0.47, ysize*0.6]
            bottom_right = [xsize*0.95, ysize*1.0]
            top_right    = [xsize*0.57, ysize*0.6]

            vertices = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int32)

            image_RofInt = region_of_interest(image, vertices)

            return image_RofInt
```

2.b. Apply Hough Line Detection

2.c. Add lines to the original image



3) Lane extension pipeline has 4 steps:

The pipeline consists of 4 steps:

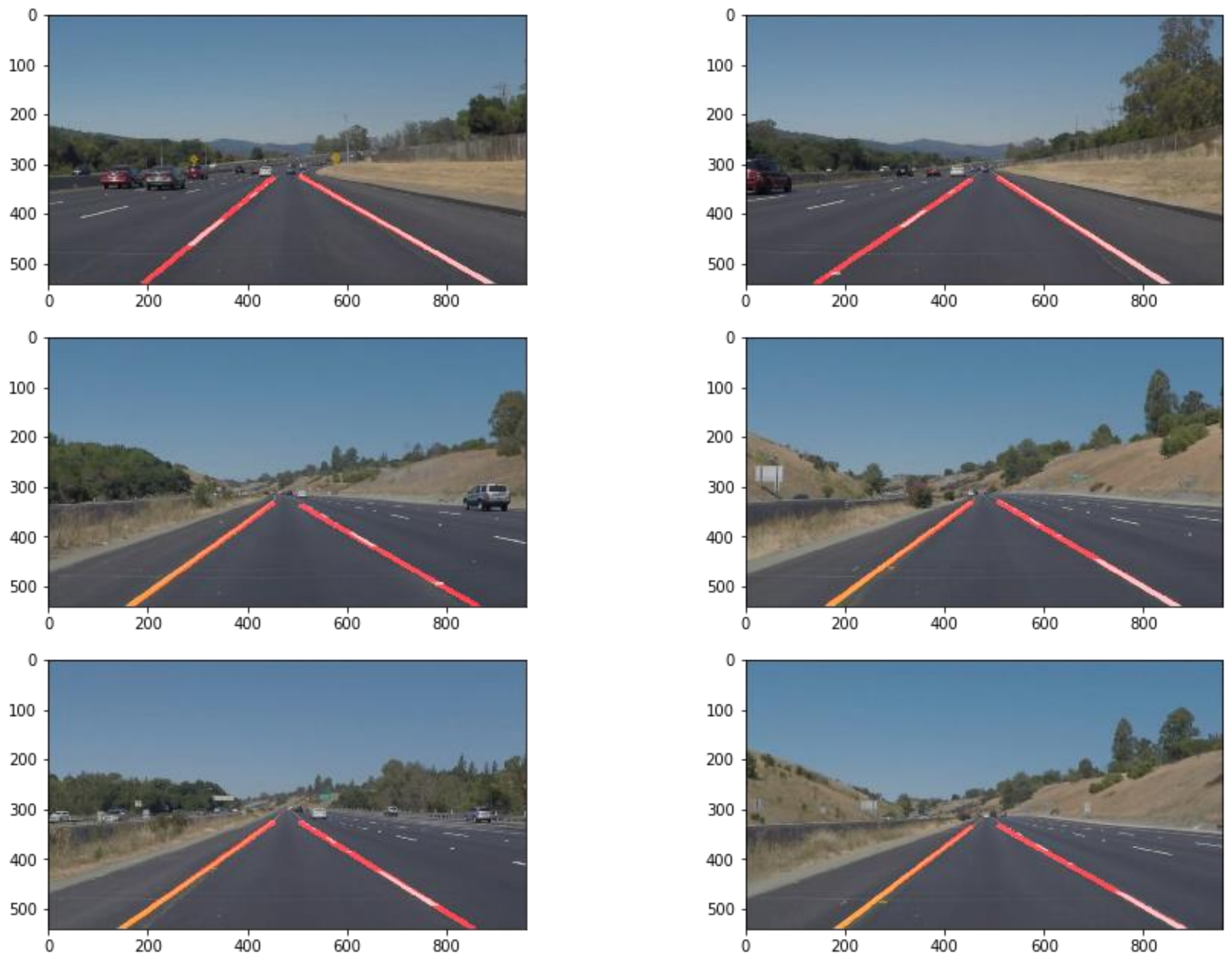
3.a. Split the lanes in 2 categories: left and right; this is done by the function left_right_lanes. ,

Note: for robustness 3rd category can be introduced for ignoring the detected lines that aren't on either left and right lines to filter them out (not implemented in this code) ,

3.b. Do a recursive weighted average of left/right lines slope and intercept (see left_right_lanes). The weight is based on the line length. The recursive element is chosen for code efficiency as it requires much less memory than the batch weighted average calculation; because it requires less memory and becomes useful especially for lane detection on videos. ,

3.c. Extrapolate the lanes to bottom of the image and top of region of interest. ,

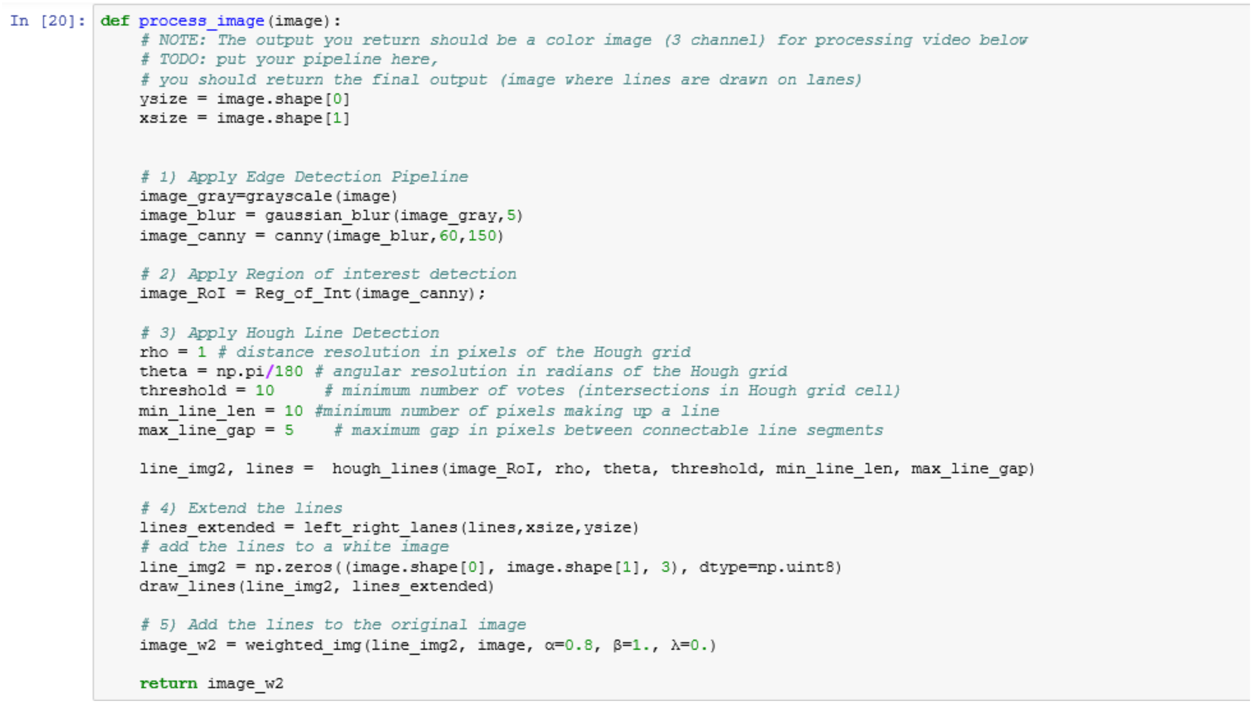
4.d. Add the extrapolated lanes to the original image and plot them.



II. Extend the pipeline developed for lane detection to videos

Videos are sequence of images, then the pipeline just splits the video into frames and send the frames as image to process_image() to run it on the lane detection pipeline.

The function process_image encapsulates the pipeline for lane detection.



```
In [20]: def process_image(image):
             # NOTE: The output you return should be a color image (3 channel) for processing video below
             # TODO: put your pipeline here,
             # you should return the final output (image where lines are drawn on lanes)
             ysize = image.shape[0]
             xsize = image.shape[1]


             # 1) Apply Edge Detection Pipeline
             image_gray=grayscale(image)
             image_blur = gaussian_blur(image_gray,5)
             image_canny = canny(image_blur,60,150)

             # 2) Apply Region of interest detection
             image_RoI = Reg_of_Int(image_canny);

             # 3) Apply Hough Line Detection
             rho = 1 # distance resolution in pixels of the Hough grid
             theta = np.pi/180 # angular resolution in radians of the Hough grid
             threshold = 10     # minimum number of votes (intersections in Hough grid cell)
             min_line_len = 10 #minimum number of pixels making up a line
             max_line_gap = 5    # maximum gap in pixels between connectable line segments

             line_img2, lines =  hough_lines(image_RoI, rho, theta, threshold, min_line_len, max_line_gap)

             # 4) Extend the lines
             lines_extended = left_right_lanes(lines,xsize,ysize)
             # add the lines to a white image
             line_img2 = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)
             draw_lines(line_img2, lines_extended)

             # 5) Add the lines to the original image
             image_w2 = weighted_img(line_img2, image, α=0.8, β=1., λ=0.)

             return image_w2
```

The output videos are here: \test_videos_output

solidWhiteRight.mp4 solidYellowLeft.mp4

2. Identify potential shortcomings with your current pipeline

The algorithm needs to be robustified for different conditions including: curvy lanes (similar to challenge exercise), night, snowy, rainy, extra light conditions, etc.

3. Suggest possible improvements to your pipeline

To detect curvy lanes such as those presented in the challenge video we can use the polynomial functions rather than the lines. Another option is to use shorter line segments in the Extend line function; for example the maximum line length shouldn’t be more than 2 m.