FINDING LANE LINES ON THE ROAD

Objectives:

Finding Lane Lines on the Road

- Make a pipeline that finds lane lines on the road
- Reflect work and understanding describing steps in a writeup report.

Introduction

Lane line detection is one of the essential components of self-driving cars. There are many approaches to doing this. Here, we'll look at the simplest approach using **Hough Transformation**.

General Pipeline Steps:

1. ***Convert the images to grayscale***. This helps by increasing the contrast of the colours, making it easier to identify changes in pixel intensity.
2. Apply ***Gaussian blur*** to the image. The purpose of the gaussian filter is to reduce noise in the image. We do this because the gradients in Canny are really sensitive to noise, so we want to eliminate the most noise possible.

3. Apply ***Canny Edge Detection*** (Canny already includes Gaussian Blur, however we use it also independently). This is where we detect the edges in the image. What it does is it calculates the change of pixel intensity (change in brightness) in a certain section in an image. Luckily, this is made very simple by OpenCV.

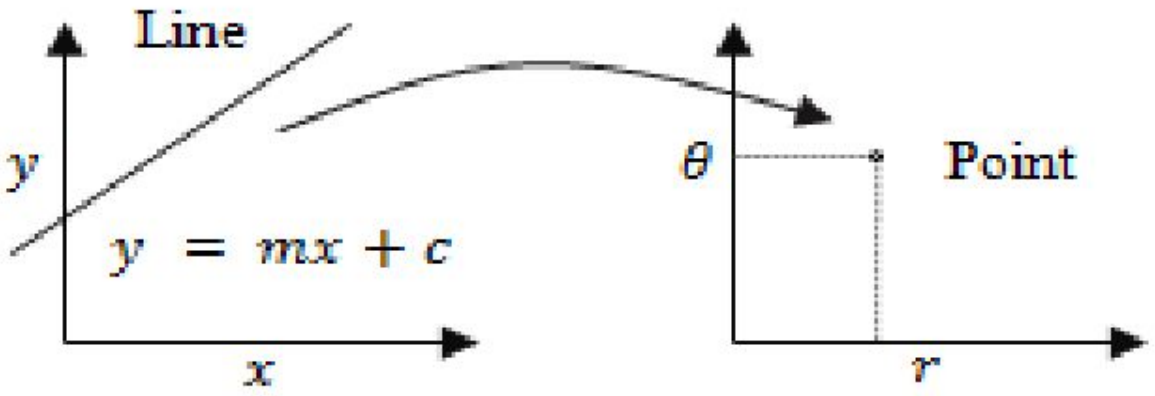
4. Define ***area of interest*** (in this case we used polyfit, defining vertices) and applying the mask. This function will isolate a certain hard-coded region in the image where the lane lines are. It takes one parameter, the Canny image and outputs the isolated region.

   Function Example:

```
def region(image):
    height, width = image.shape
    triangle = np.array([
                        [(100, height), (475, 325), (width, height)]
                        ])
    mask = np.zeros_like(image)
    mask = cv2.fillPoly(mask, triangle, 255)
    mask = cv2.bitwise_and(image, mask)
    return mask
```

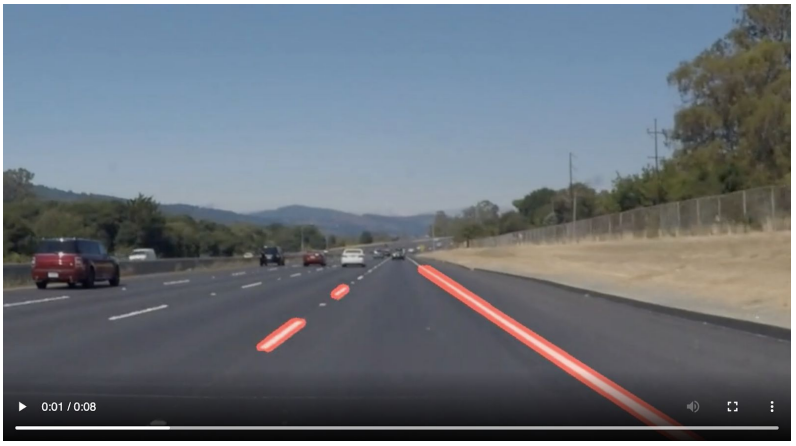
5. Run ***Hough Transformation***

Principle Fig 1.1



- Lines in the cartesian plane are represented as points in Hough Space
- Points in the cartesian plane are represented as lines in the Hough Space
- You can **find the line of best fit** of two points in cartesian space by finding the *m* and *b* coordinates

6. Combining initial image with the image after Hough Transform, applying ***weighted_line() function*** to turn the edges into lines/
7. Test Pipeline in images
8. Run pipeline with different clips
    - **Clip 1**: Solid white clip (Using simple helper function)



- **Clip 2:** Solid yellow clip (Using optimized helper function)

- **Clip 3**: Challenge Clip (Using challenge other helper functions)

Potential exploration to **solve these shortcomings** would be:

1. Explore other filters and functions available in OpenCV or other algorithms
2. Other parameters selections playing (tuning) different thresholds, min/max, angles, curves, slopes,etc
3. Understand and explore the requirements for the input image (camera location, quality, etc) in order to discard implications from the original image.