# Report Project 1

## Finding lanes in images

### Lane fitting

I have adjusted the helper method "draw_lines" to 1. separate between the left lane and the right lane and 2. to identify a line between the x1, x2 values. Separation of lines was carried out according to the project template using the slope of two points: ((y2-y1)/(x2-x1)).
The negative slope was identified as part of the left lane and the positive slope as part of the right lane.

The corresponding x and y values have been stored in to separate arrays for further processing.
Now, in order to identify the line that represents the left and right lanes, I have used a simple linear regression provided by nympy (polyfit). Since the return value of polyfit represents the m and b values in y=xm+b and since I needed the corresponding function, I have used numpy's poly1d to get the actual linear function.

Due to the fact that the linear function in 2D space represents a line, I needed only two points to plot the corresponding lanes; which were represented by my field of view (region of interest)

### Parameter Selection

W.r.t. the Canny thresholds, my opinion is that the suggested ratio of 1:3 was not sufficient. While trying different parameters, I have selected 90,120 as my preferred choice. In combination with a kernel size of 7 for the gaussian smoothing, this approach seemed to result in the best outcome.

For rho I have identified that the value 1 in combination with a higher value for the Hough threshold (i.e. 60) implied the better results. Based on my observation, these parameters had the maximum impact on the resulting lane display.

Note that I have defined variables that represent my field of view as these were required in the draw function.

# Finding lanes in videos

## Parameter selection

It turned out that my parameter selection used for the image detection was not good enough for videos. Therefore, I had to adjust the parameters for:

1. The Gaussian Blur (as in some frames the resulting matrices were too sparse for a linear fit). A higher value enabled to smoothen and get better results (a higher value also implied less "shaky" projection lines).
2. The Canny thresholds as the edge detection had to be in a different greyscale
3. The rho value for the Hough transform to cover more intersections
4. The Hough threshold as some frames included less lines and the resulting min/max values for the lines

In order to get rid of most non-relevant lines however, I had to adjust my "range of interest" (referred to as field of view in my code comments). I have noticed that the camera mounting position was different compared to the one in the images. This adjustment seemed to resolve the remaining issues that I have noticed (I was actually fighting with that for some time).

## Limitations and potential improvements

---

### A: Less Shaky Lines (using image flow)

The projection of the lines that cover the lanes is calculated for each video frame and since these imply smaller changes and moreover, since the gaussian blur applied prior to the Canny algorithm has its limitations, a further improvement for "smoother" lines would be to introduce a "memory" that stores a set of previously calculated points (at frame_i-1,i-2,...i-n). These points should be for the (linear) fitting for the current frame (frame_i).

---

### B: Improved Fitting

The lines projected using the code are calculated with a simple linear fit. This however would lead to poor results at least in the following cases:
1. Curves
2. Steep hills (e.g. San Francisco)

Therefore, a non linear fit should be introduced to enable an optimal fit in non linear environments.
In my test runs, I have applied a non linear fit by adjusting the degree in np.polyfit (e.g. fittedRight=np.polyfit(rightLaneX, rightLaneY, 3).

With this application one single line projection would not be sufficient, of course.
Example adjustment in code:

```
    XX = np.linspace(510, 930, (930-509))
    while i < len(list(XX))-2:
        y1=fitPolyR(int(XX[i]))
        y2=fitPolyR(int(XX[i+1]))
        cv2.line(img, (int(XX[i]), int(y1)), (int(XX[i+1]), int(y2)),
color, thickness)
        i=i+1
```

In my observations, however, I have experienced that the extrapolation in linear environments was rather poor (implying further improvements).
Therefore, I was not able to correct the outlier at the 11th second of the video file `solidYellowLeftWithLanes.mp4`

---

## C: Advanced Fitting

Since the lanes follow a distinct pattern with smooth changes, an interesting approach to improved fitting could be a Gaussian Process (GP).
As a GP identifies (multivariate) functions that follow a Gaussian distribution, lane fitting using GPs seems worth a try.
This represents a Machine Learning approach that requires a certain amout of data.

In my test run (refer code below) I experienced issues with shortage on training data as my predictions seemed to have no proper outcomes.

```
from sklearn.gaussian_process import GaussianProcessRegressor
 from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C

  kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
  gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
  gp.fit(np.array(rightLaneX), np.array(rightLaneY))
  XX, YY = np.meshgrid(np.arange(510, 930), np.arange(510, 930))
  y_pred, sigma = gp.predict(XX, return_std=True)
```