Report
Finding lane lines project

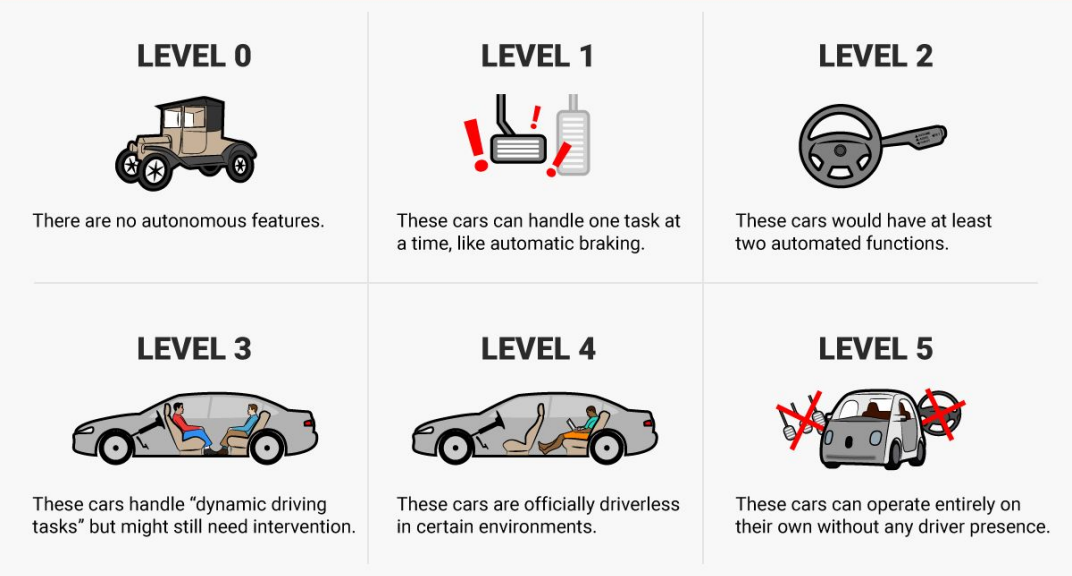
Team members

ThoTD2 & AnDTP & PhuVT2

Contents

Problems in Self-driving car PJ

Self-driving car

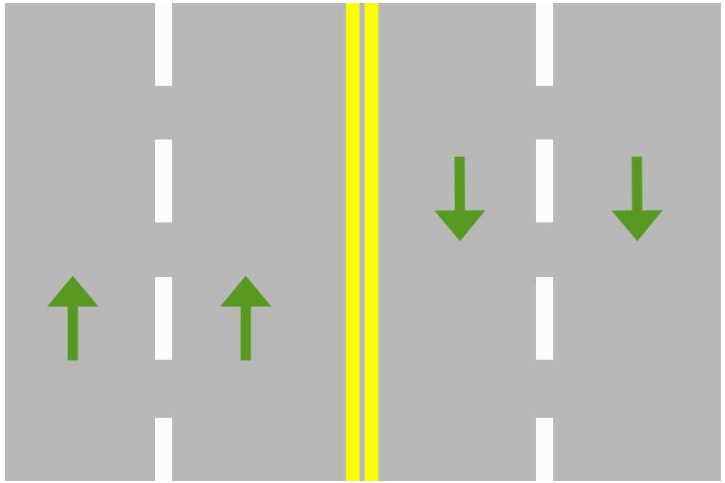


6 levels of self-driving car in **SAE (J3016) Automation Levels**

Problems in Self-driving car

- **Goal**: reach each level so that the car itself becomes a self driving car.
- **Basic step**: turn car from level 0 to level 1 (Driver Assistance).

- The problem must be solved in **Computer Vision: Detection & Classification**.
- The most basic object to determine is 2 lanes, to know where the vehicle can run.



Finding lanes line PJ

Naive solution

Canny and Hough algorithm applied solution

Naive solution

Solving

1. **Color selection:** select white objects in the image
2. **Region masking:** determine region of the lane lines
3. **Color the identified lane lines**

Solving - Color selection's Fundamentals

- The image is made up of three color channels (red, green and blue)
- Each color channel contains pixels whose values range from 0 to 255
- The darkest color value is 0
- The brightest color value is 255

Solving - Color selection

1. Define the minimum color values (R, G, B) for white objects to be selected.
2. Black out all the pixels whose value is below the threshold.

```
red_threshold = 200
green_threshold = 200
blue_threshold = 200
rgb_threshold = [red_threshold, green_threshold, blue_threshold]

# Identify pixels below the threshold
color_threshold = (image[:,:,0] < rgb_threshold[0]) \
            | (image[:,:,1] < rgb_threshold[1]) \
            | (image[:,:,2] < rgb_threshold[2])
color_select[color_threshold] = [0,0,0]
```

Implement in python language

Solving - Color selection result

Original image

Image after color selection

There are some objects detected around that are not lane lines

Solving - Region masking

```
# Define a triangle region of interest
left_bottom = [0, 539]
right_bottom = [900, 539]
apex = [480, 320]

# Fit lines (y=Ax+B) to identify the  3 sided region of interest
fit_left = np.polyfit((left_bottom[0], apex[0]), (left_bottom[1], apex[1]), 1)
fit_right = np.polyfit((right_bottom[0], apex[0]), (right_bottom[1], apex[1]), 1)
fit_bottom = np.polyfit((left_bottom[0], right_bottom[0]), (left_bottom[1], right_bottom[1]), 1)


# Find the region inside the lines
XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))
region_thresholds = (YY > (XX*fit_left[0] + fit_left[1])) & \
                    (YY > (XX*fit_right[0] + fit_right[1])) & \
                    (YY < (XX*fit_bottom[0] + fit_bottom[1]))
```

Expected result & Implementation in python language

Solving - Color selection & Region masking combined

image[color_threshold] = [0, 0, 0]

image[~region_threshold] = [0, 0, 0]

Solving - Color the identified lane lines

```
image[~color_threshold & region_threshold] = [255, 0, 0]
```

Evaluation

Time to apply our solution

Image with yellow lane line on the left

Color selection fails
to detect the yellow lane line

Evaluation - Ignored facts

- Vehicles don’t always run in lane
- **Lane lines’ colors** are not always white
- **Light intensity** does change in different conditions (night time, shade of trees on the side, etc)

Evaluation

Advantage	Disadvantage
• Time complexity is kind of quick **O(number of pixels in image)** • The solution is simple and easy to apply.	• Lane lines are not always white. • The same color affected by other factors (lightning, shadow, etc) may not be detected.

We need a solution that is more complicated and effective

Advanced Solution

Assumption

- Vehicles always run in lane
- **Light intensity** does not change too much
- No or some objects that cover 2 lane lines but not too much

⇒ We find lane lines using:

- **GrayScale & Gaussian Filter**
- **Canny Edge Detection**
- **Hough Line Transform**

Why need GrayScale & Gaussian Filter?

- Help to reduce the processing time.
- Convert RGB 3-channel image to GrayScale 1-channel image.
- Reduce noise & make image smooth.

Grayscale & gaussian filter applied example image

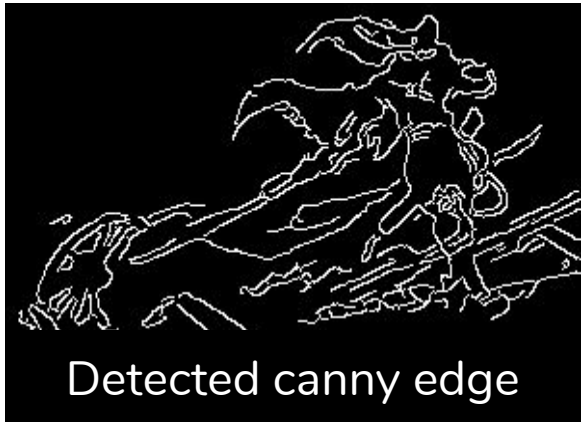
Some pieces about Canny Edge Detection

The use of **Canny Edge Detection** to help identify edges and discard all other unnecessary pixels.

A pixel is considered as an edge pixel if its gradient is higher than the high threshold or its gradient is between the threshold values and it is connected to another edge.

Original

Gaussian filter

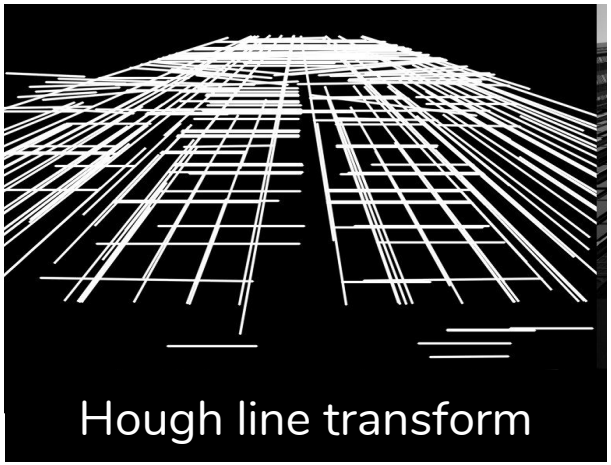


Detected canny edge

Gaussian filter & Canny edge detection algorithm applied example image

What's Hough Line Transform used for?

- To find all the lines on which edges points lie.
- To remove some short and useless edges with specific parameters.



Grayscale & Hough Line Transform applied example image

Steps to solve

1. GrayScale & Gaussian Filter
2. Canny Edge Detection
3. Region of interest & Hough line transform
4. Draw lines

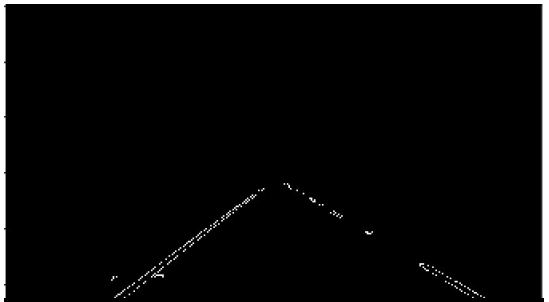
Step 1 - 3

Original

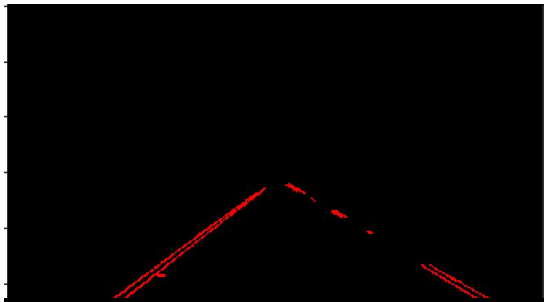
Grayscale

Gaussian filter

Detected canny edge



Region of interest



Hough lines

Draw lines from Hough lines

1. **Classify lines & reduce the noise lines**.
   1.1. **Left lane lines** have a **slope smaller than -0.5**
   1.2. **Right lane lines** have a **slope larger than 0.5**
   1.3. The remaining lines are noise lines
2. Calculate average left line & average right line
3. Draw 2 average lines

Draw lines for multiple frames

1. **Classify lines & reduce the noise lines**.
   1.1. **Left lane lines** have a **slope smaller than -0.5**
   1.2. **Right lane lines** have a **slope larger than 0.5**
   1.3. The remaining lines are noise lines
2. Calculate average left line & average right line
3. Draw average lines if the slope is not too different from the average lines of previous frames.
   If not just draw the previous lines.

Evaluation

Advantage	Disadvantage
• **Time complexity is a little larger** than time complexity of Naive method but but give **acceptable results.** • **Reduce some assumptions** in Naive solution.	• Determine wrong lane lines if there is shade in the bend. • Only feasible in some special conditions. • Difficult to apply to the actual environment.

Thank you for listening

Reference

Self-driving car course Udacity

Self-driving car Wikipedia