Final Project: Lane Detection

資科工碩	309551178	秦紫頤
電資學士班	0610021	鄭伯俞
清華	H092505	翁紹育

Introduction

- Lane detection is an important and fundamental task in self-driving car.
- We are trying to implement through three different techniques.
- Dataset : TuSimple (tusimple-benchmark)

Contains 6,408 road images on US highways.

Image resolution is 1280×720.

Outline

- **Traditional**
  - Method 1: Edge detection + Hough transform
  - Method 2: Perspective transform
- **Deep learning**
  - Hourglass Network
- **Result comparison**
- **Conclusion**

Traditional Method

- We implement 2 different lane detection pipeline
- Break through: We can detect more than 2 lane lines
- The challenging part is extracting out edges of lane lines.

Method 1

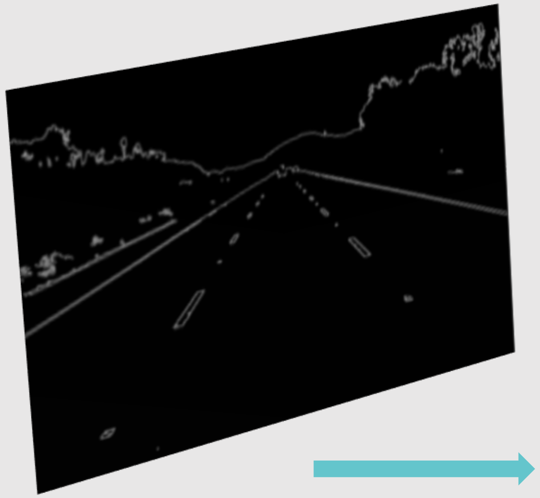
Edge detection + Hough transform

Method 1 : Edge detection + Hough transform

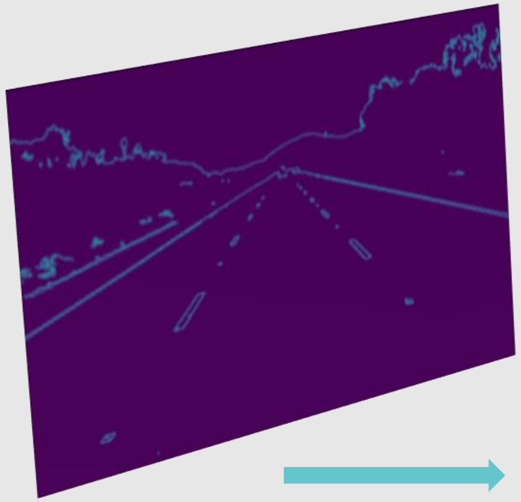
Input image

Grayscale image

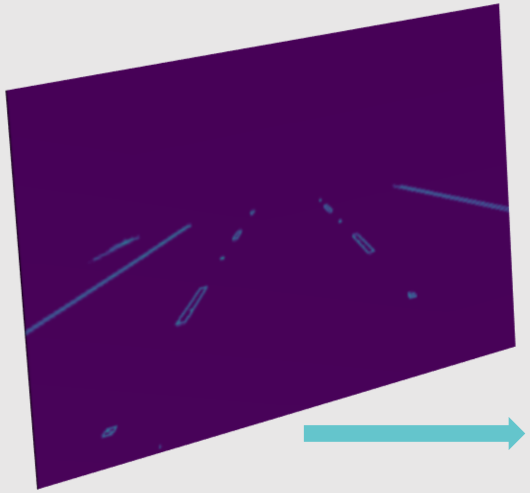
Gaussian blur



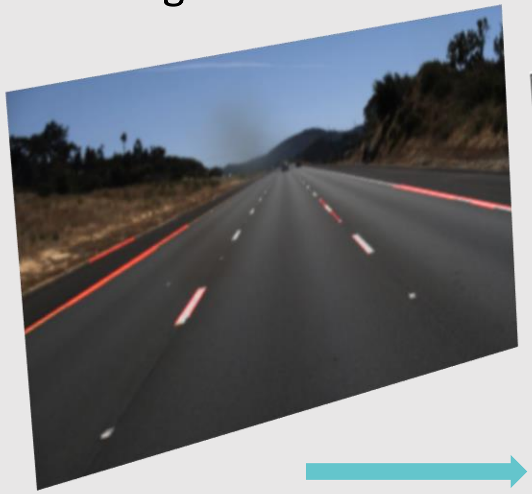
Canny edge



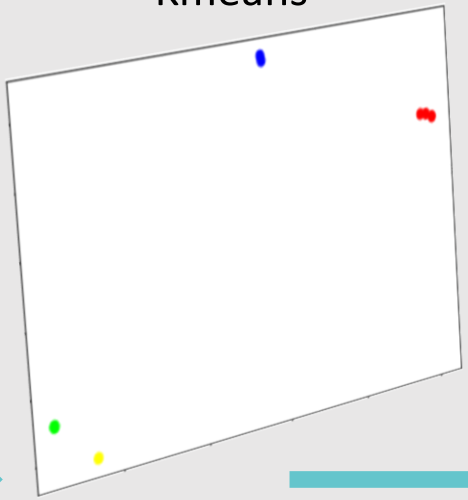
Circle region of interest



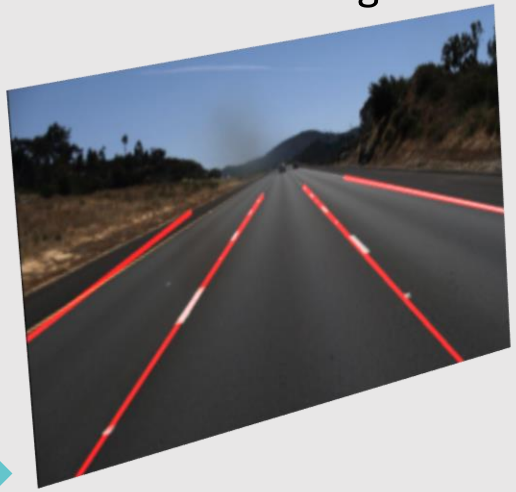
Hough transform



Kmeans

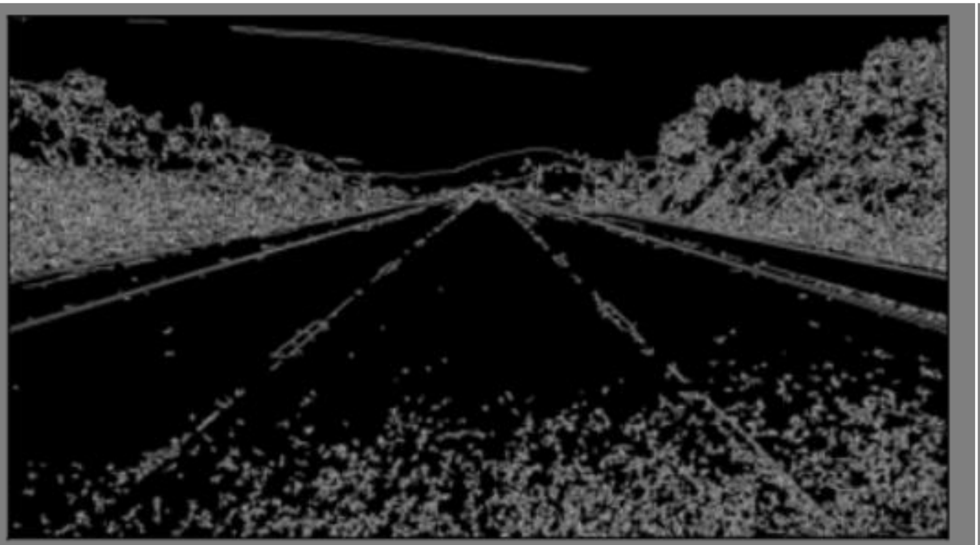


Curve Fitting



Canny Edge Detector

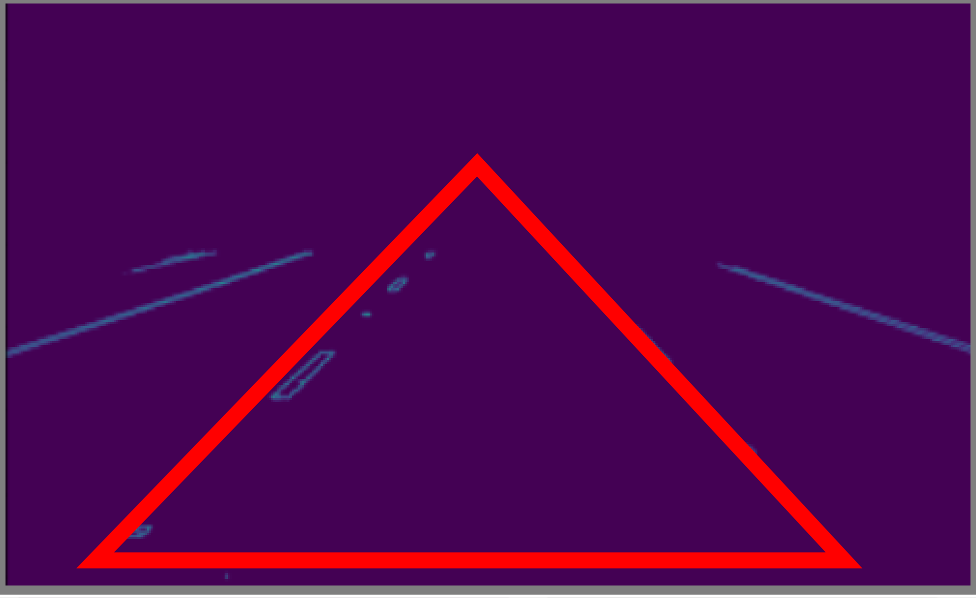
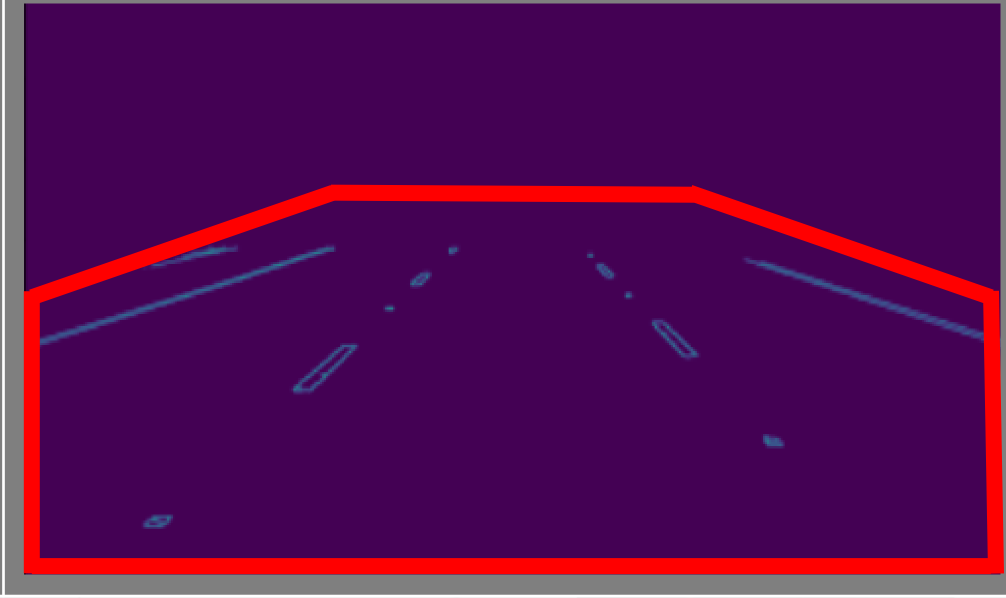
Idea: Lane lines they will survive the bluring, because they are long and have strong gradient.

Without Smoothing	Gaussian Blur with kernel_size = 15
	

Region of interest

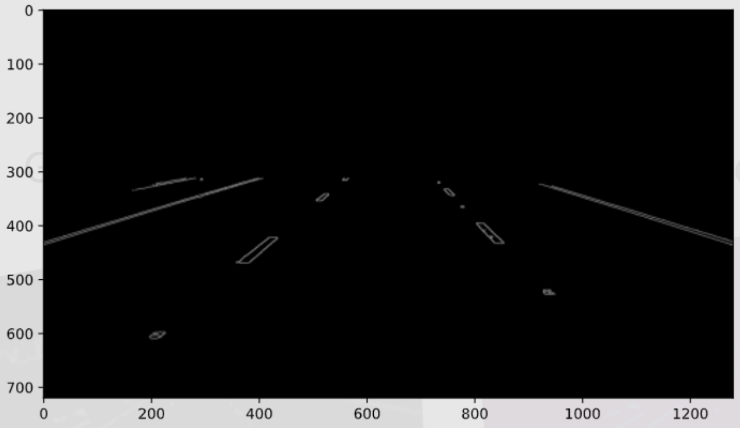
Lanes are normally restricted in particular regions of the image.

Instead of circle a triangle, we circle polygon for more lane to be detect

Original ROI	Modified ROI
	

Hough Line Transform

Idea: Lane lines are straight.

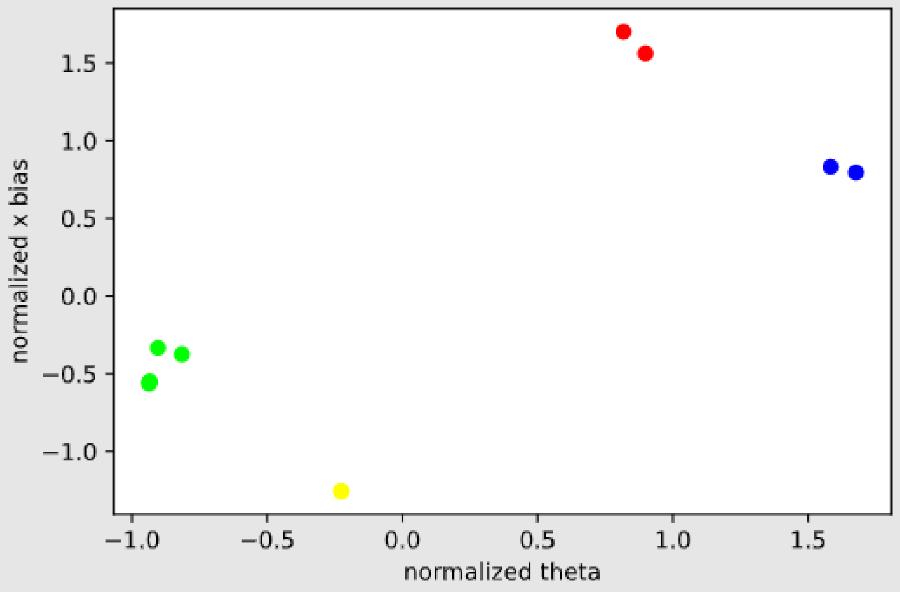


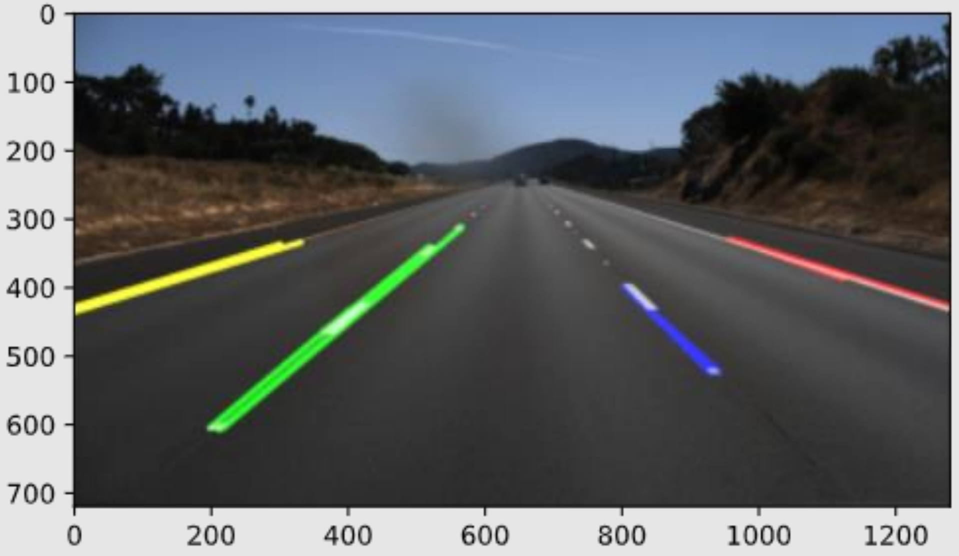
max_line_gap = 50	Max_line_gap = 300
	

Kmeans

Cluster the line segments in to several lanes.

- Different Lane lines have different slope and x-intercepts.
- We can find more than two lanes by Kmeans.





Curve fitting

Fit the end points of the line segments with mse

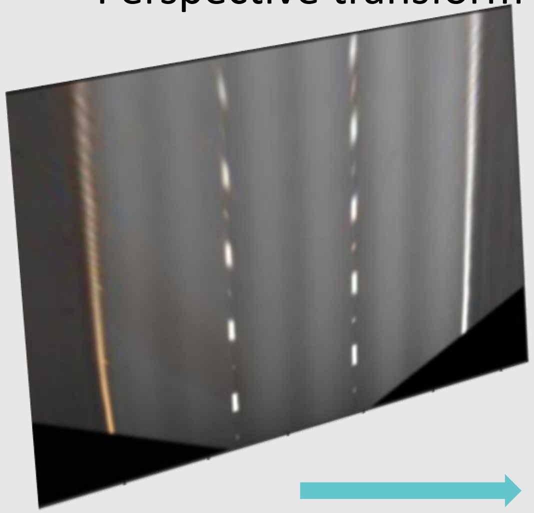
Method 2

Perspective transform

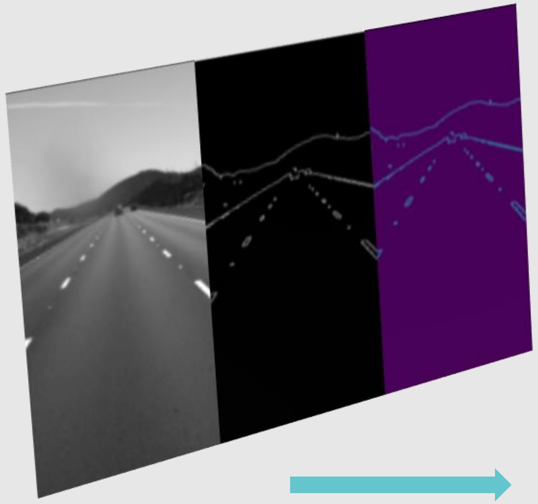
Method 2 : Perspective transform

Input image

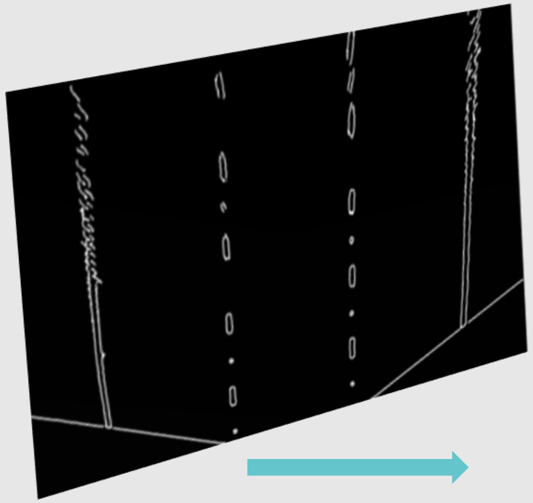
Perspective transform



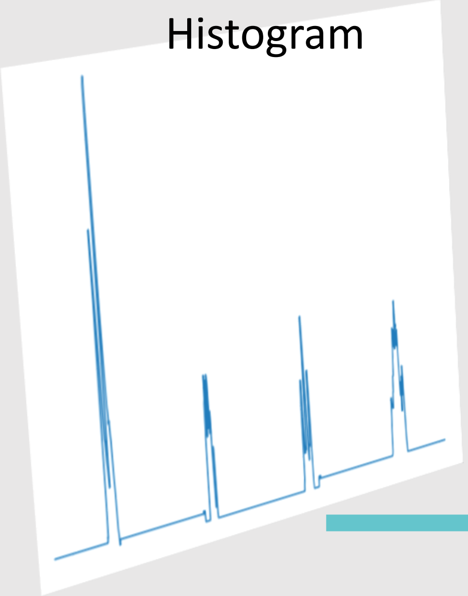
Grayscale + Gaussian + Canny



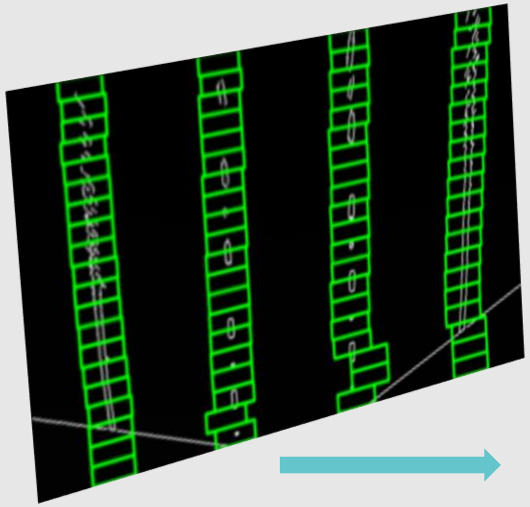
Binary image



Histogram

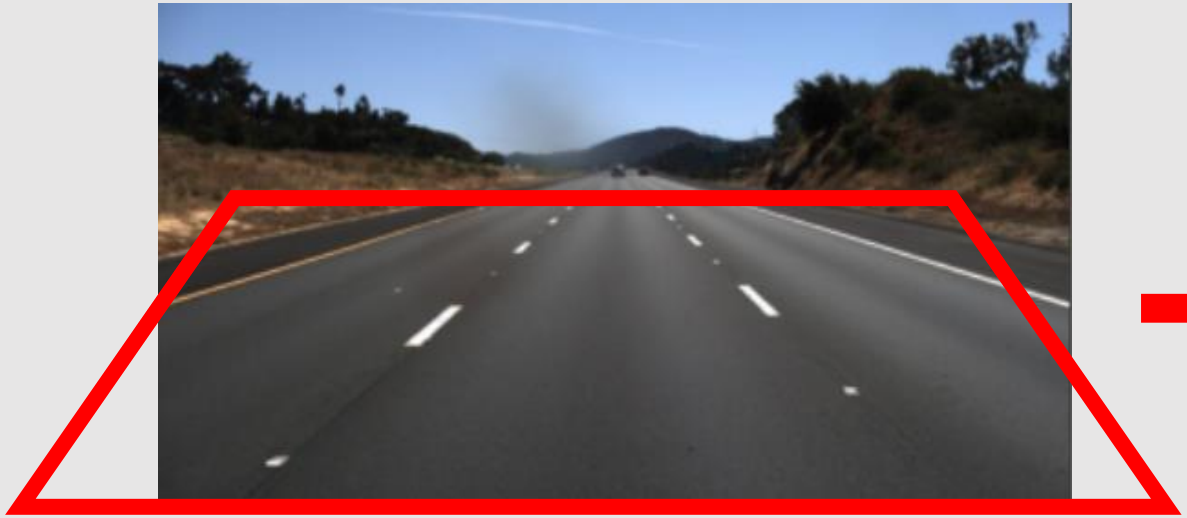


Sliding window address lane

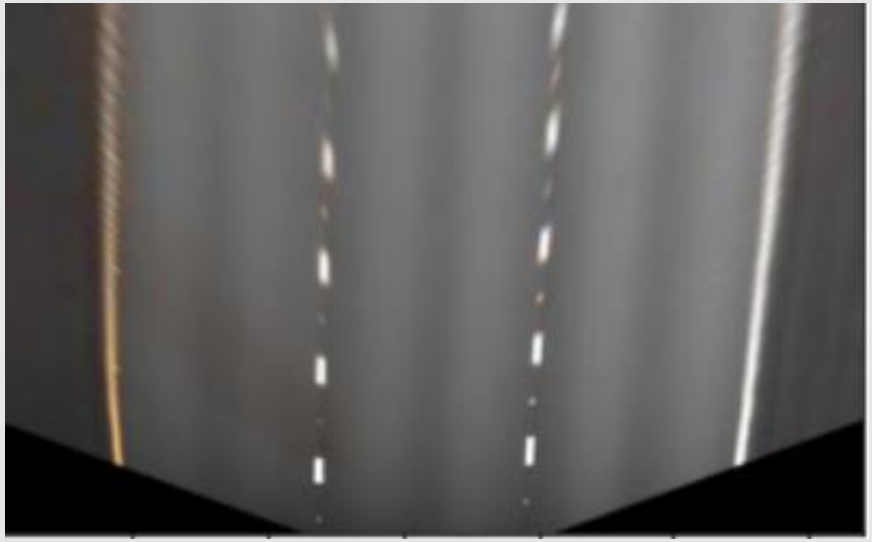


Curve Fitting

Perspective transform



Region of interest

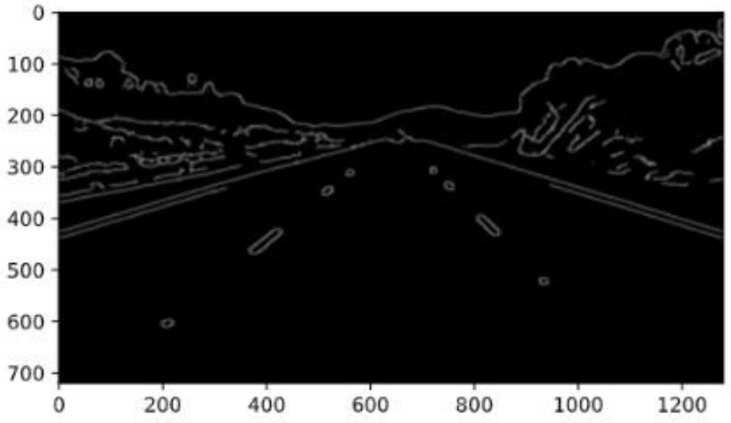
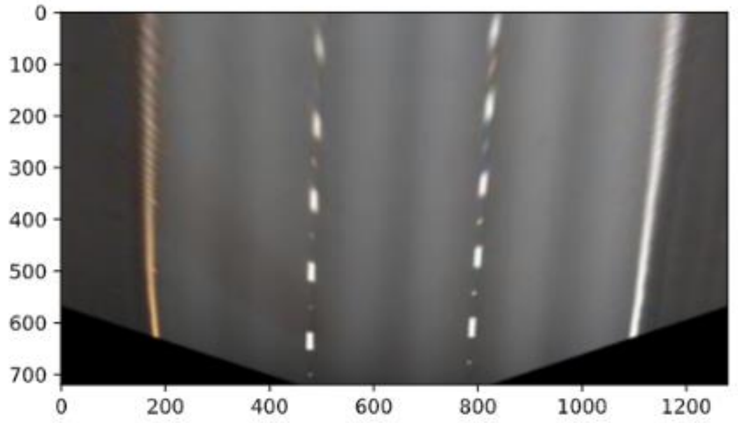
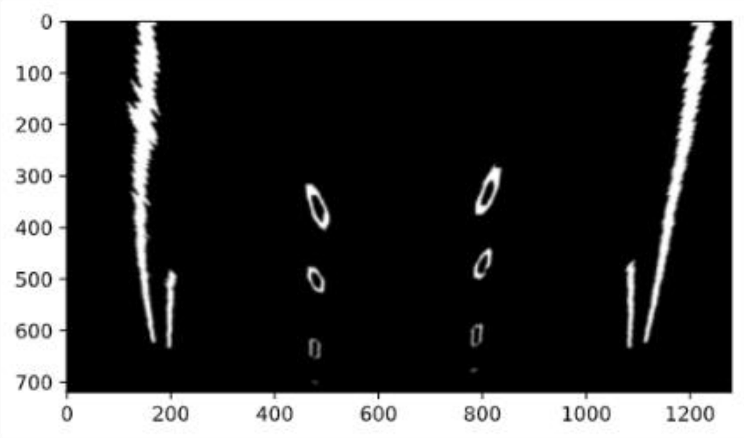
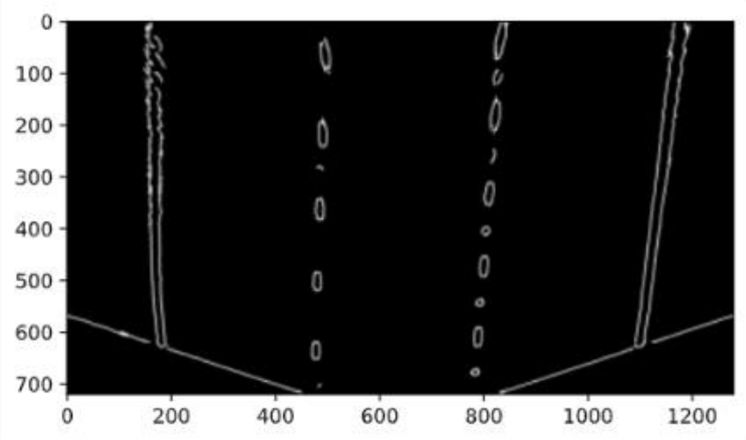


Warped Image

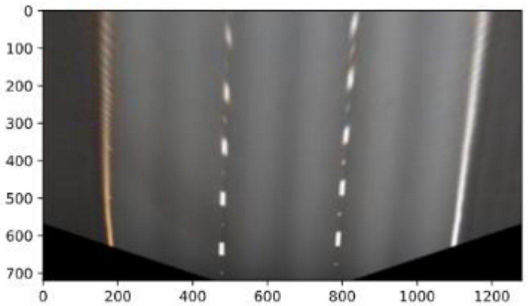
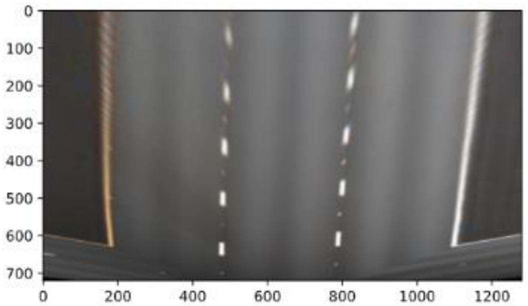
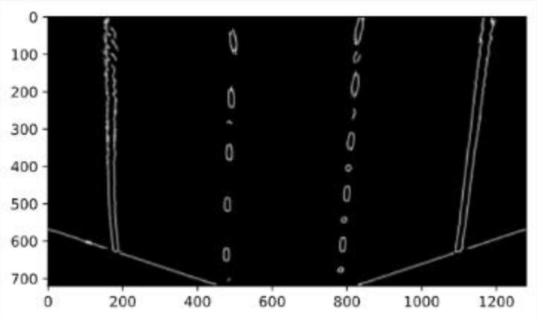
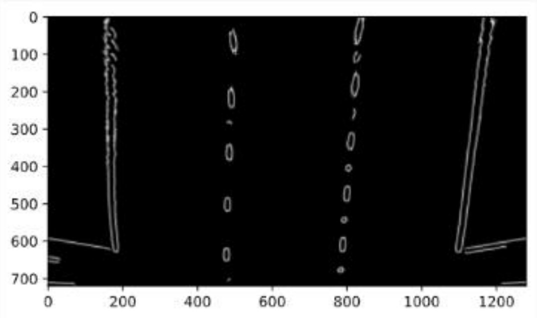
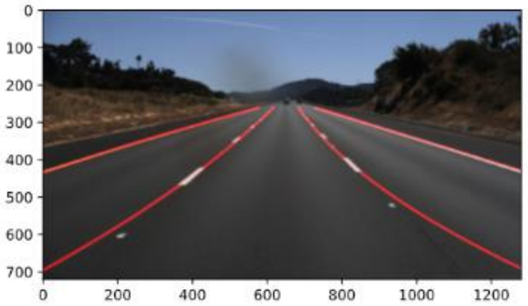
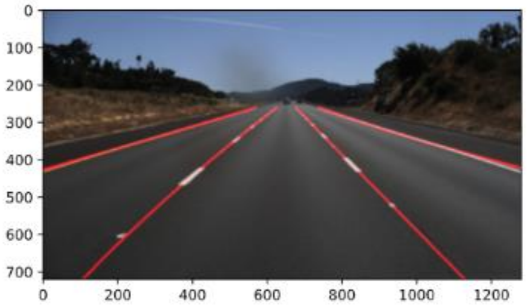
Benefits of Bird's eye view perspective:

- We can use x values to separate different lane lines
- We can traces points on a same lane along vertical direction

Canny Edge detection

	Canny -> Warp	Warp -> Canny
Step1		
Step2		

Canny Edge detection

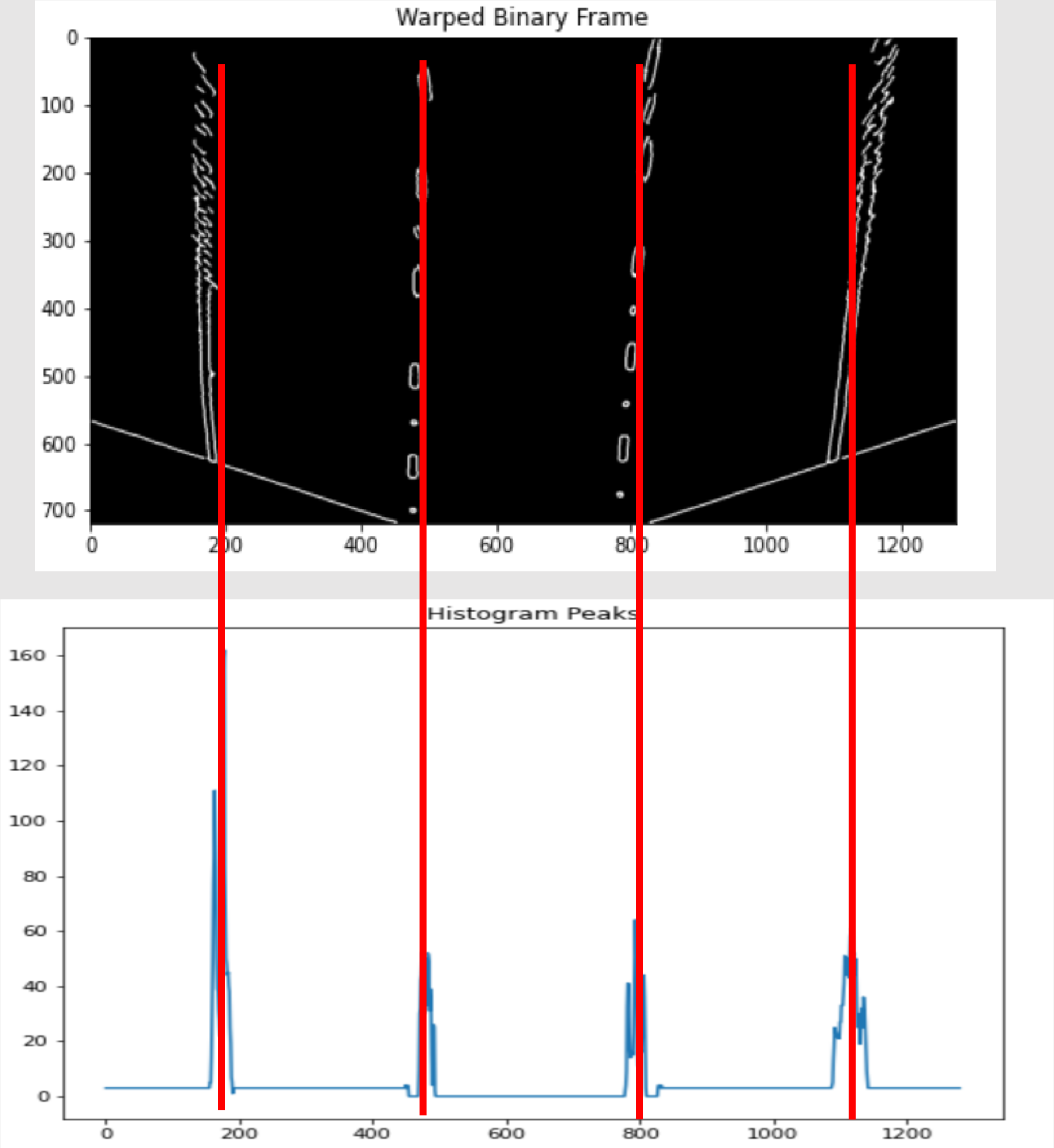
	Warp original image (Before Modification)	Warp circular repeated image (After Modification)
After Warping		
After Canny Edge		
Final Result		

Histogram

After binary the image into black/white, we sum up the values along y axis

The peak represent that there are high frequency appearance of values
→ candidate position for lanes

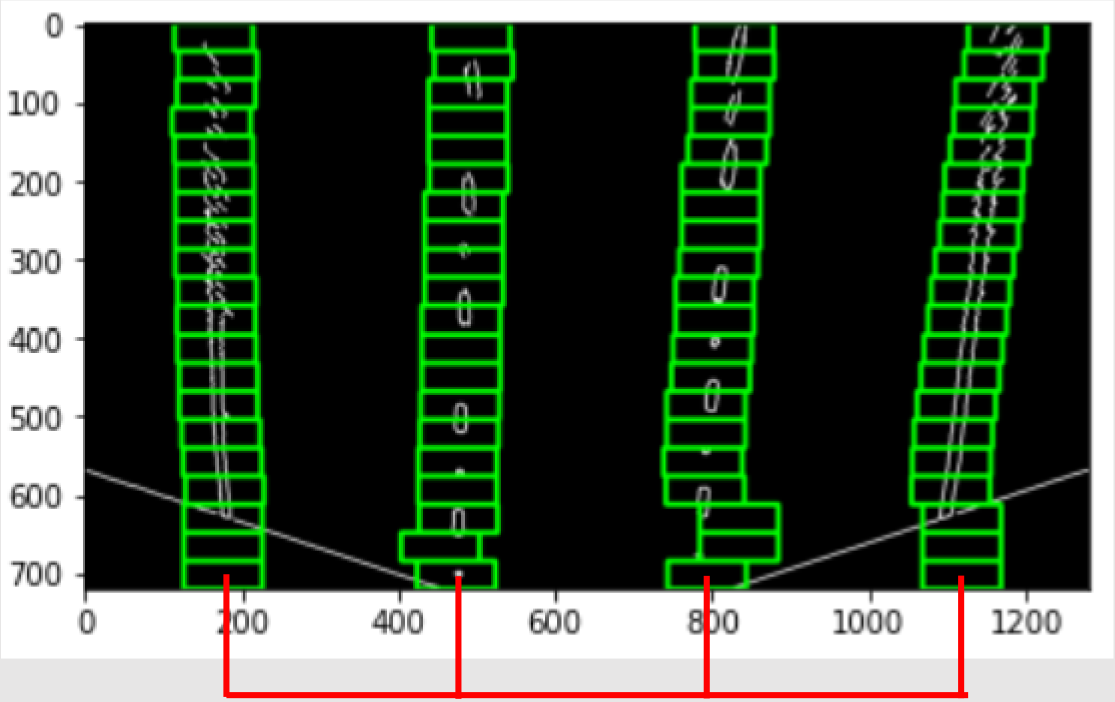
If the shape of each distribution is narrow, then the quality of lane will be better



Sliding windows

According to result of histogram, we are able to identify the position of lanes

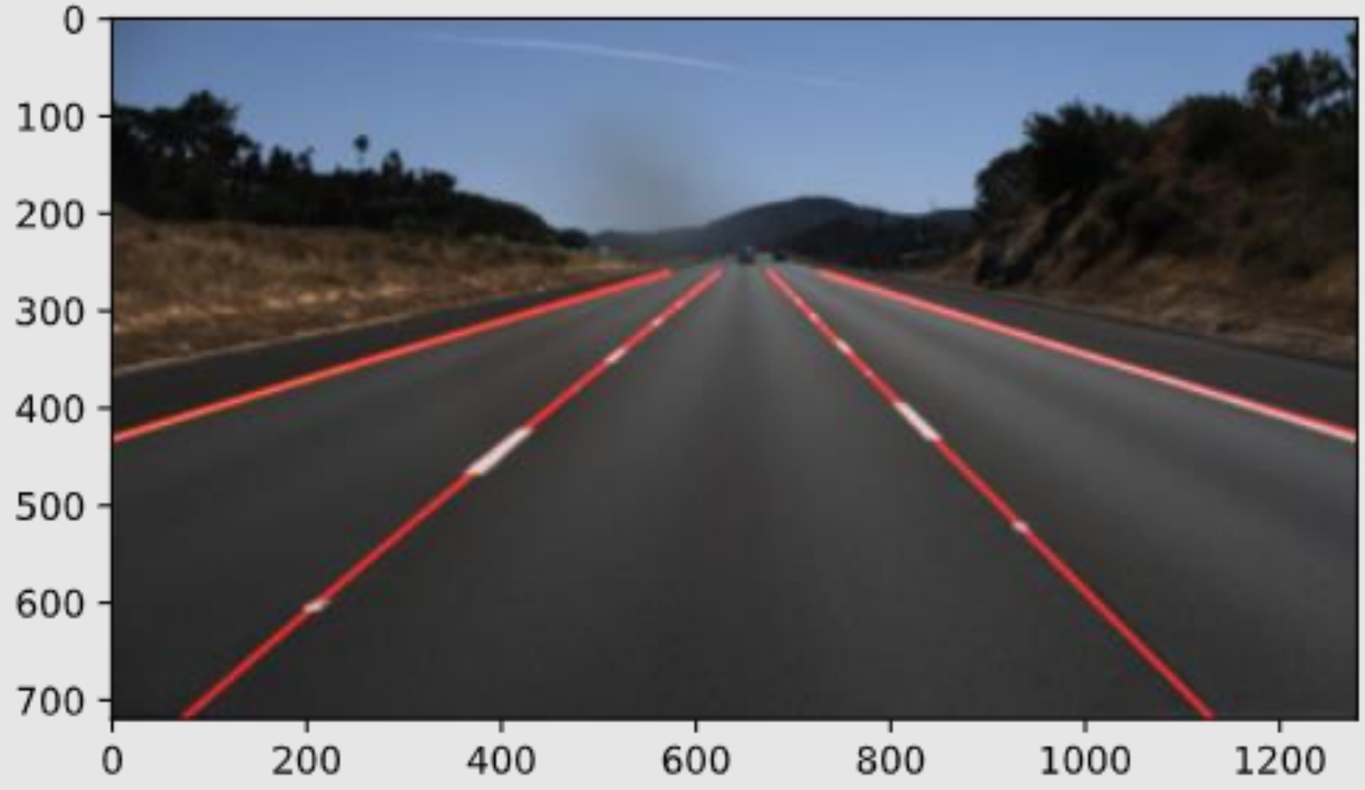
1. Apply sliding window from bottom to top of image
2. we can gradually adjust our position if the mean in sliding window get change



Starting position according to the result of histogram

Curve Fitting

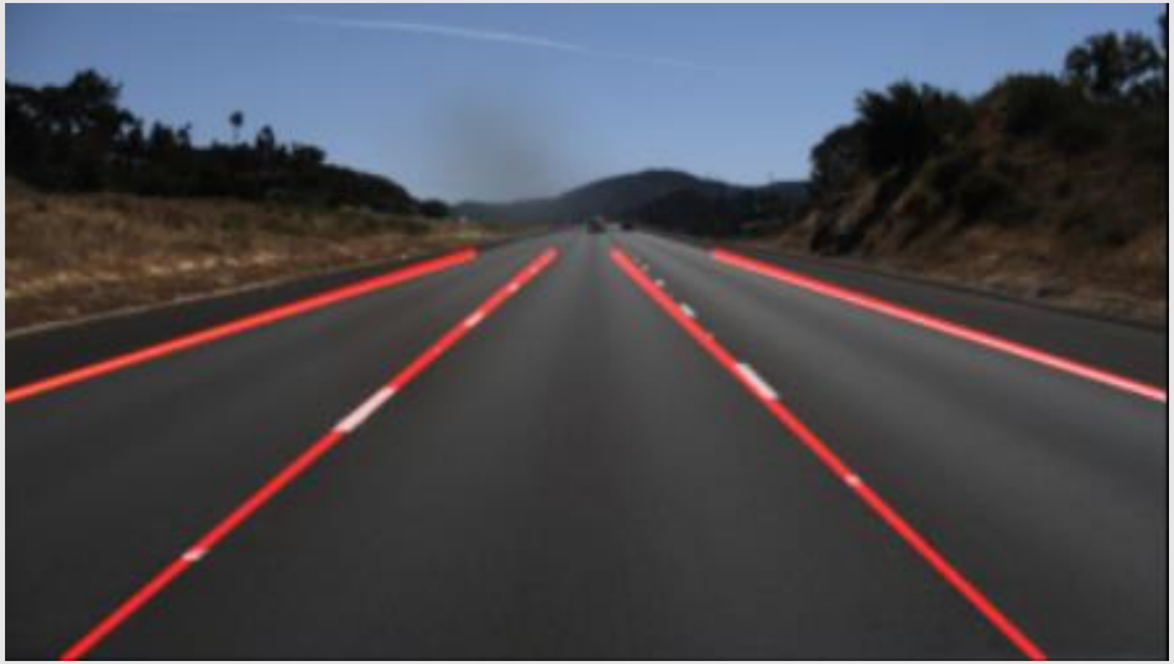
- Transform the edge pixels captured by sliding windows back to original coordinate.
- Fit each lane line with a degree 1 of degree 2 polynomial



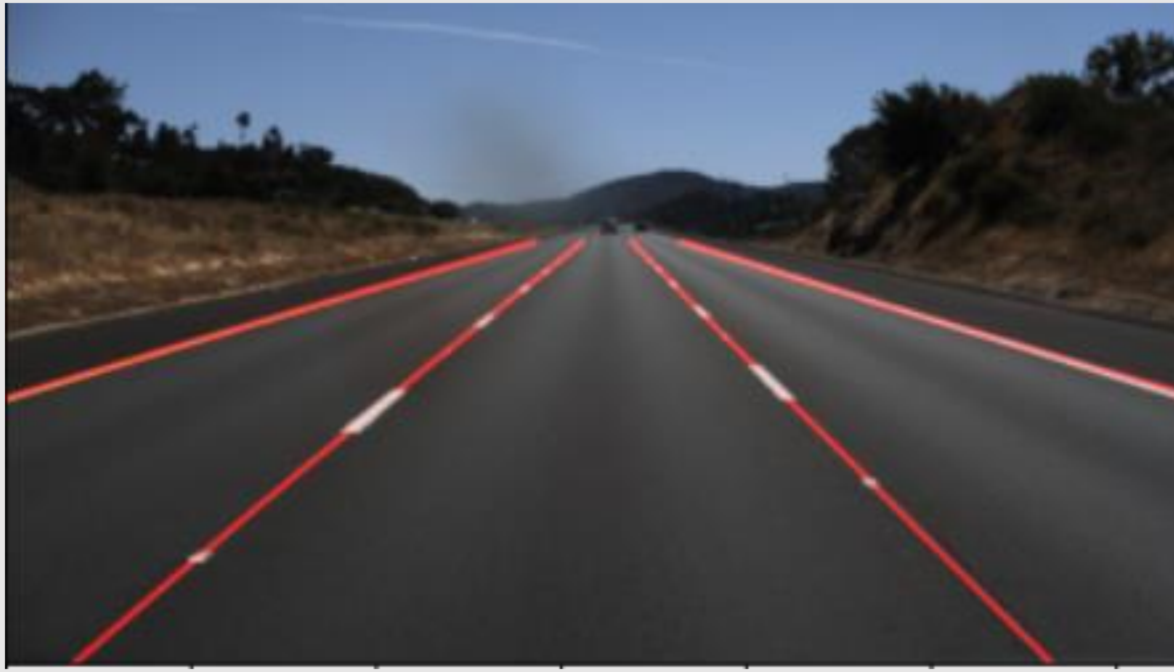
Result
[Good]

Raw image

Method 1

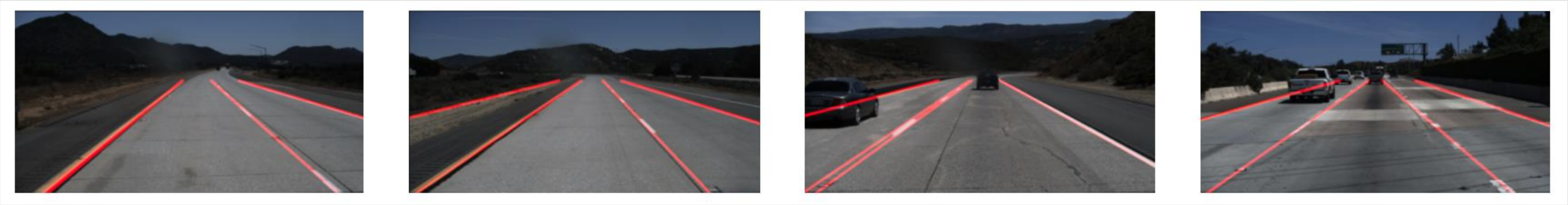


Method 2

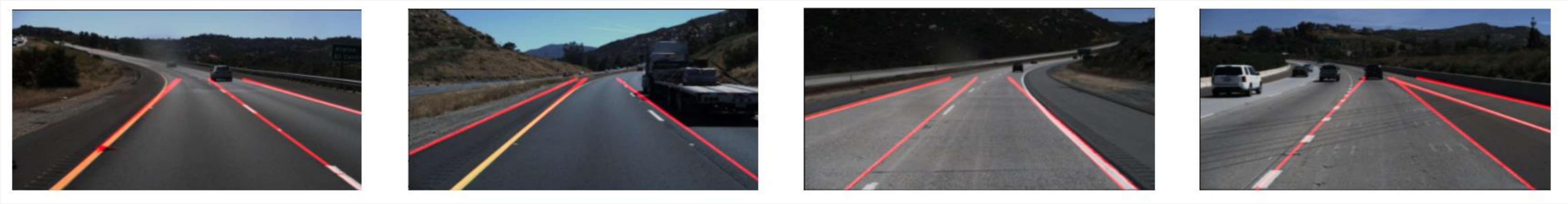


Method 1

Good



Okay



Shitty

Method 2

Good

Okay

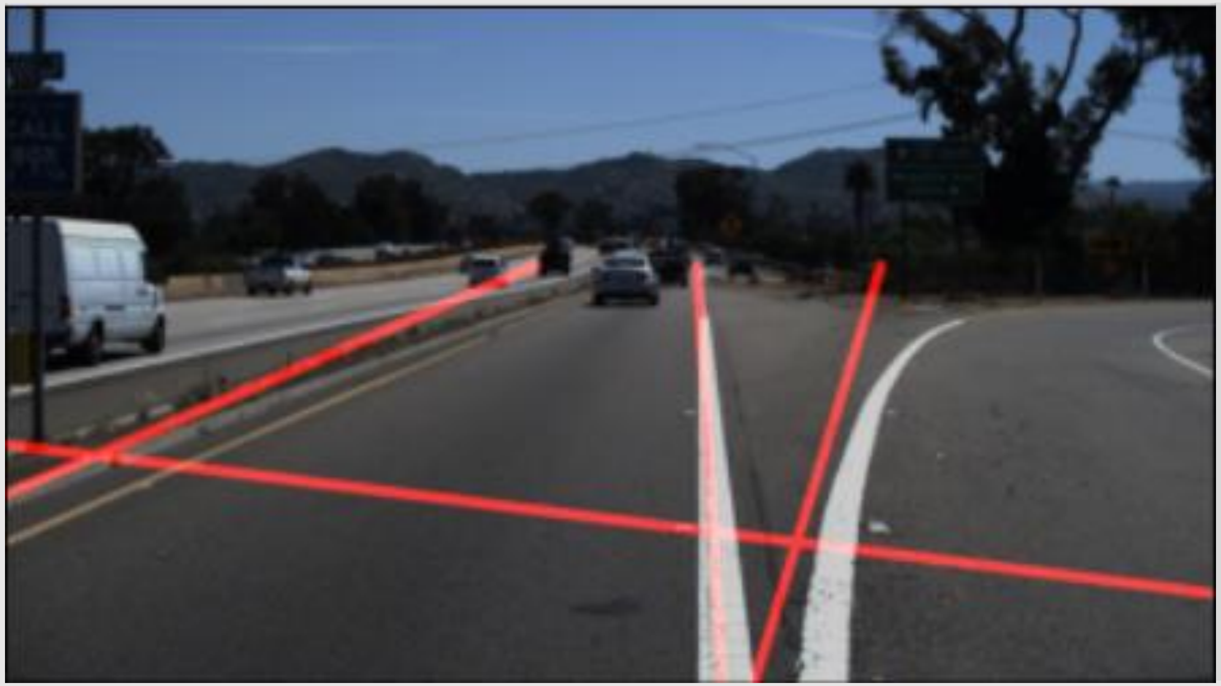
Shitty

Result

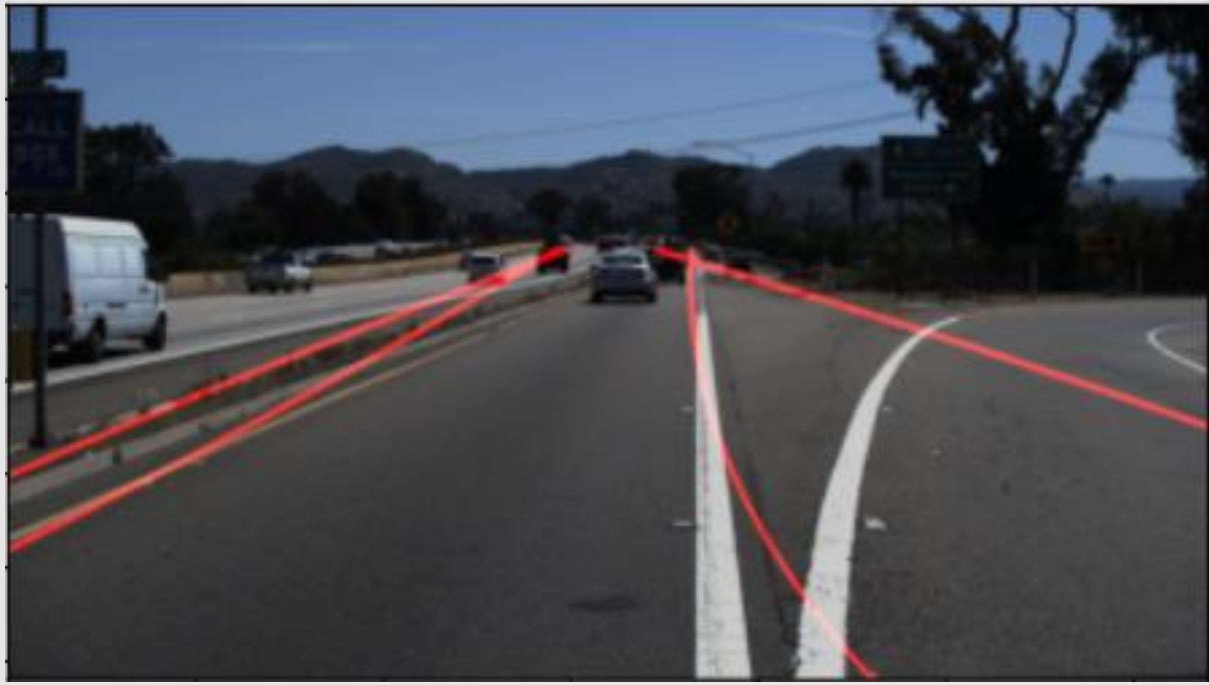
[Good]

Raw image

Method 1

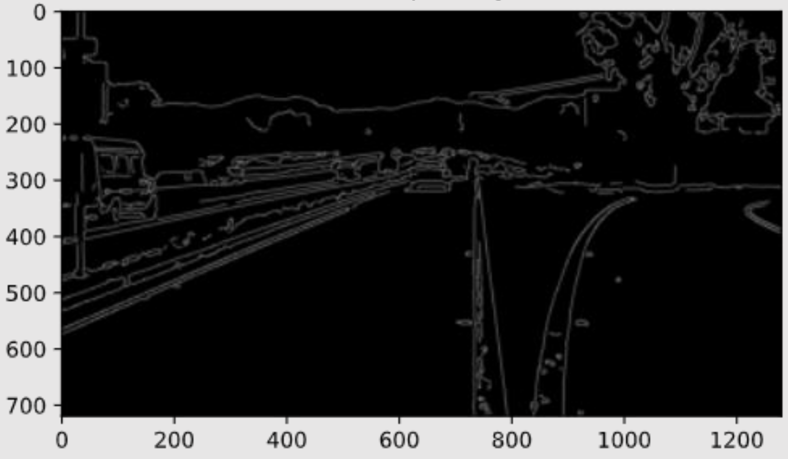


Method 2

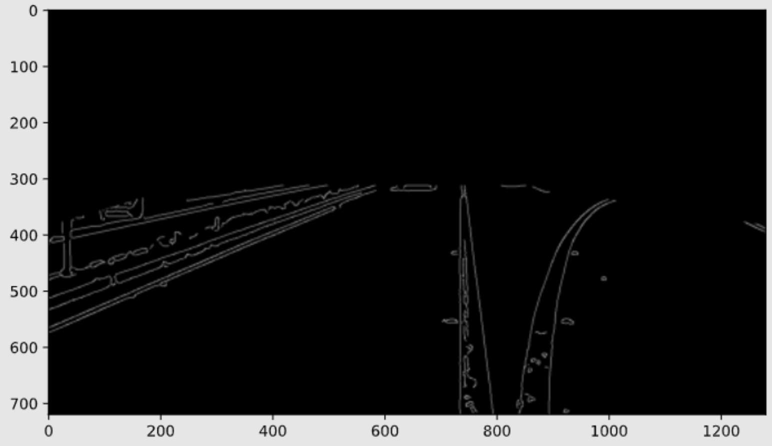


Method 1 – which step makes it broken?

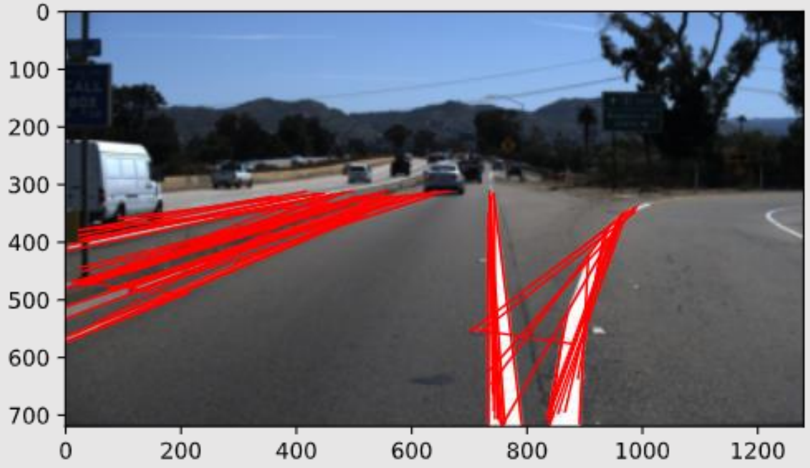
Canny edge



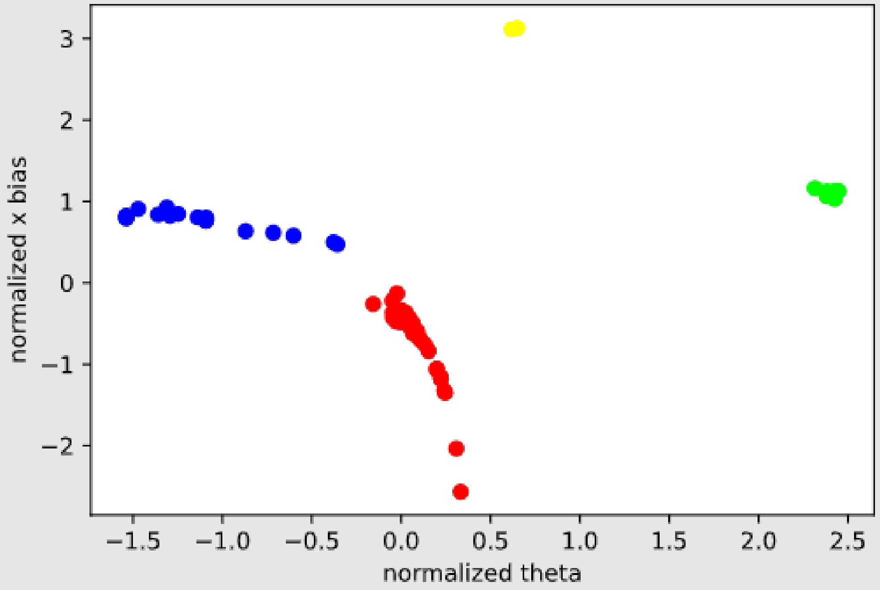
Region of interest



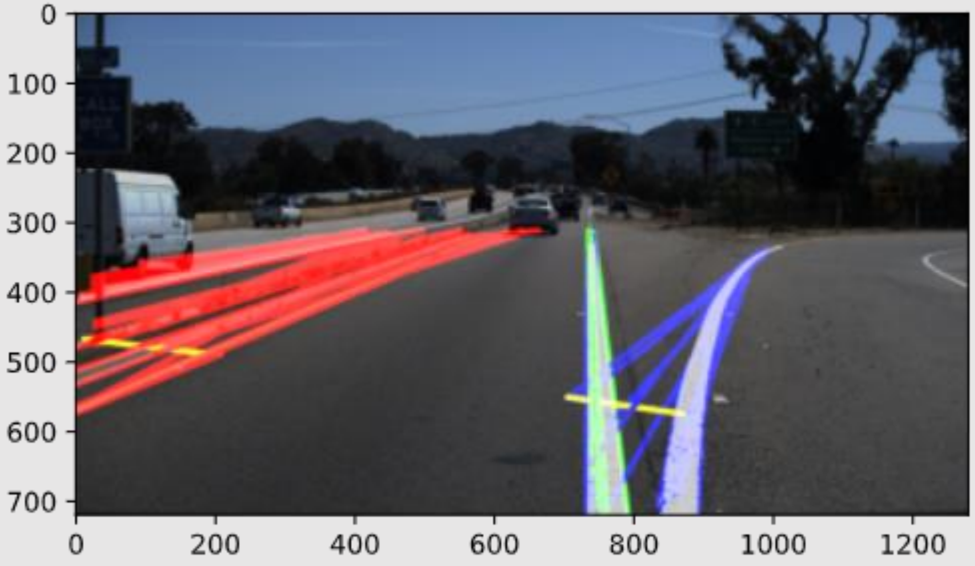
Hough transform



Kmeans



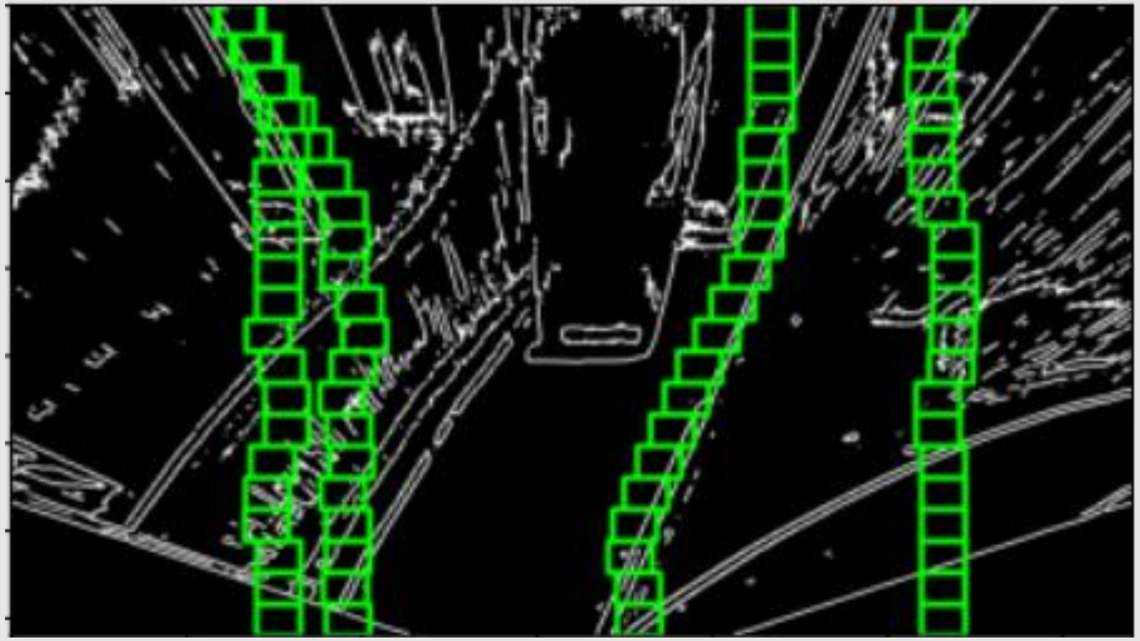
Result after Kmeans



Method 2 – which step makes it broken?

Perspective transform

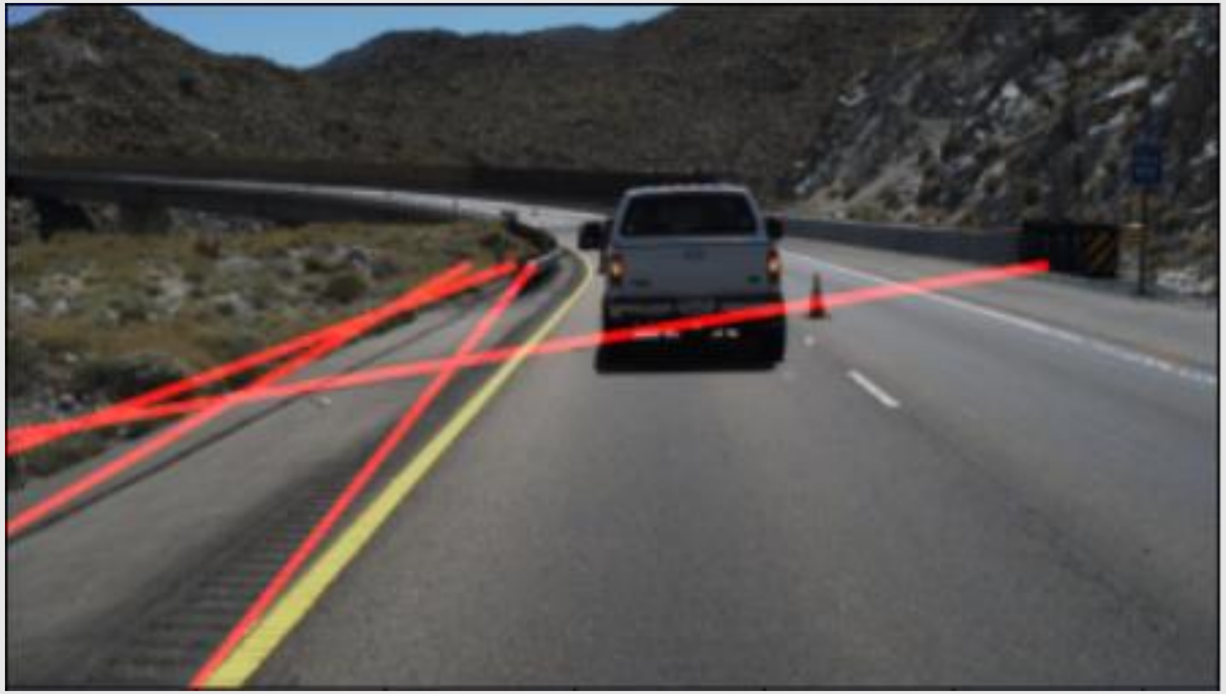
Sliding windows



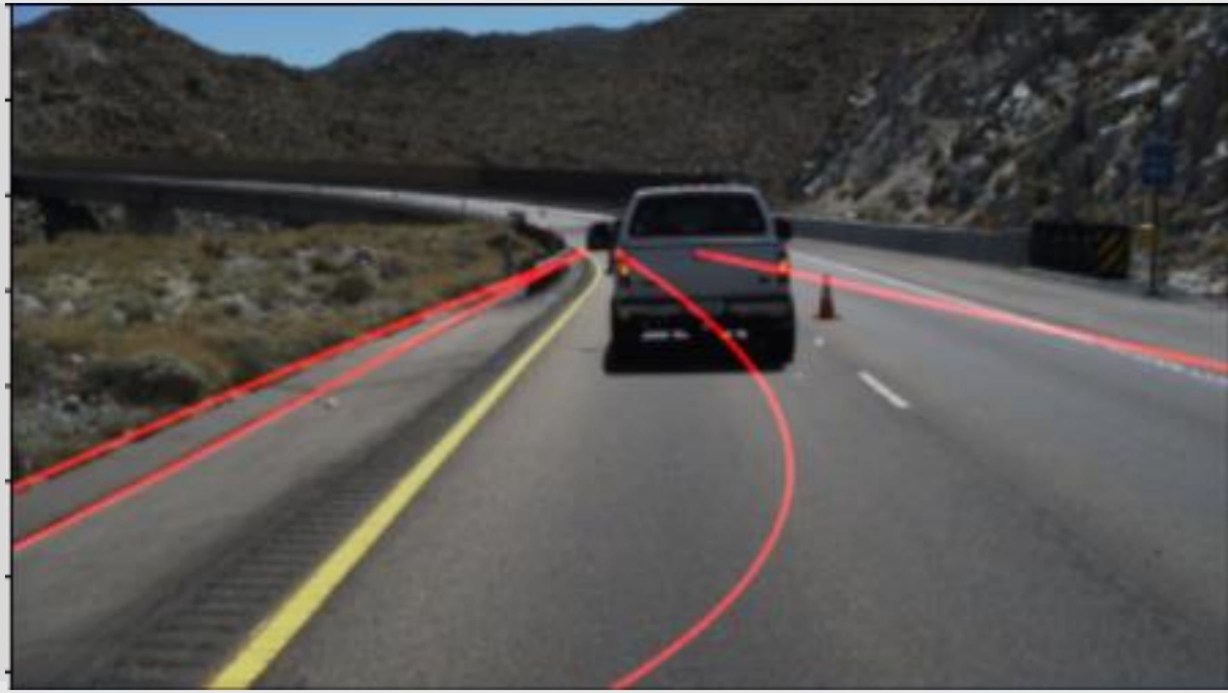
Result
[Good]

Raw image

Method 1

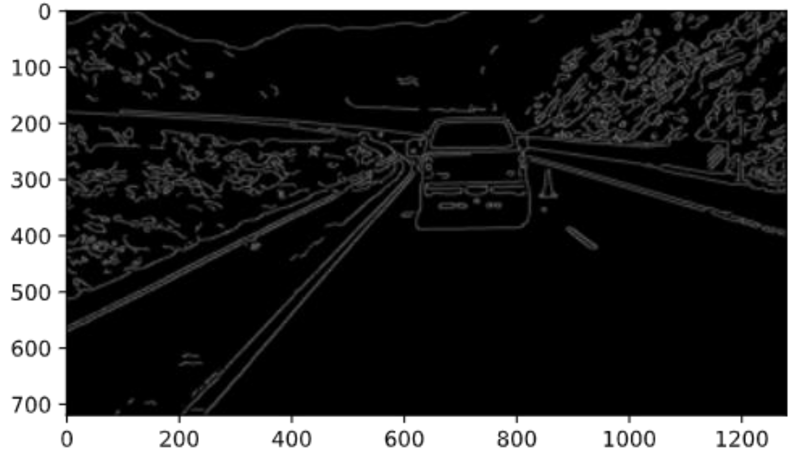


Method 2

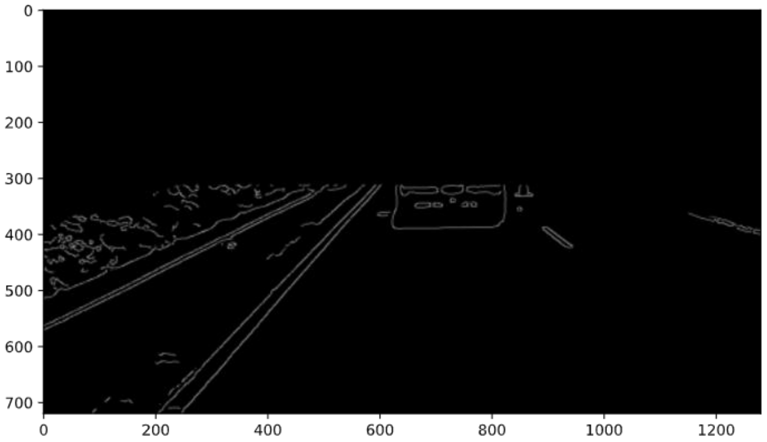


Method 1 – which step makes it broken?

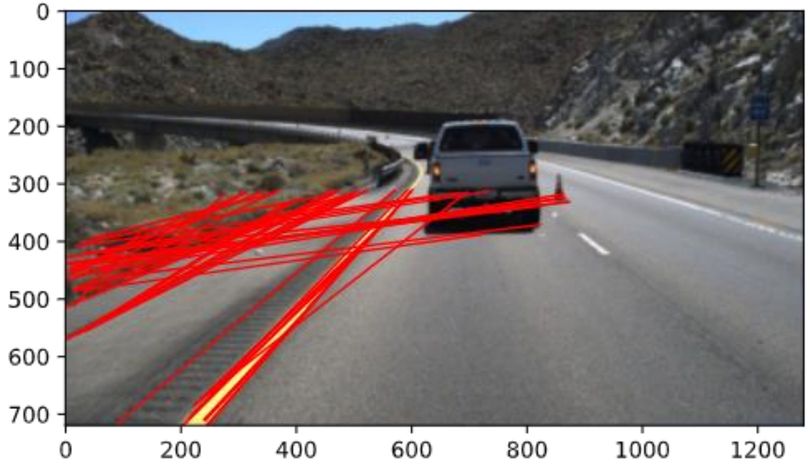
Canny edge



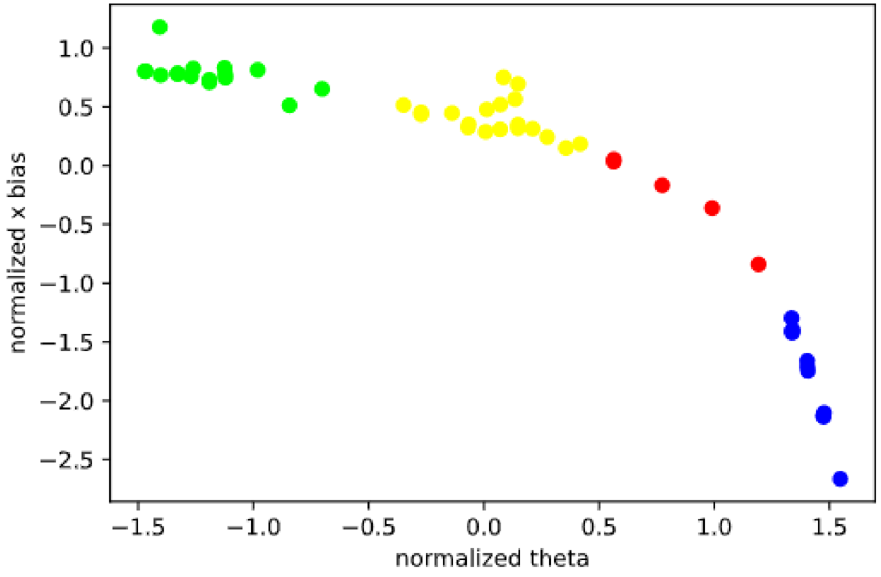
Region of interest



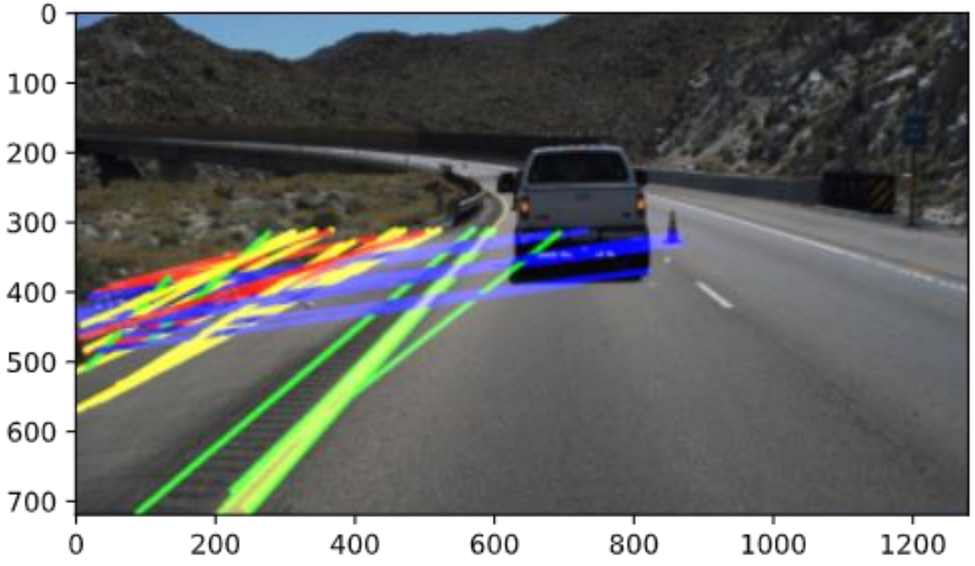
Hough transform



Kmeans



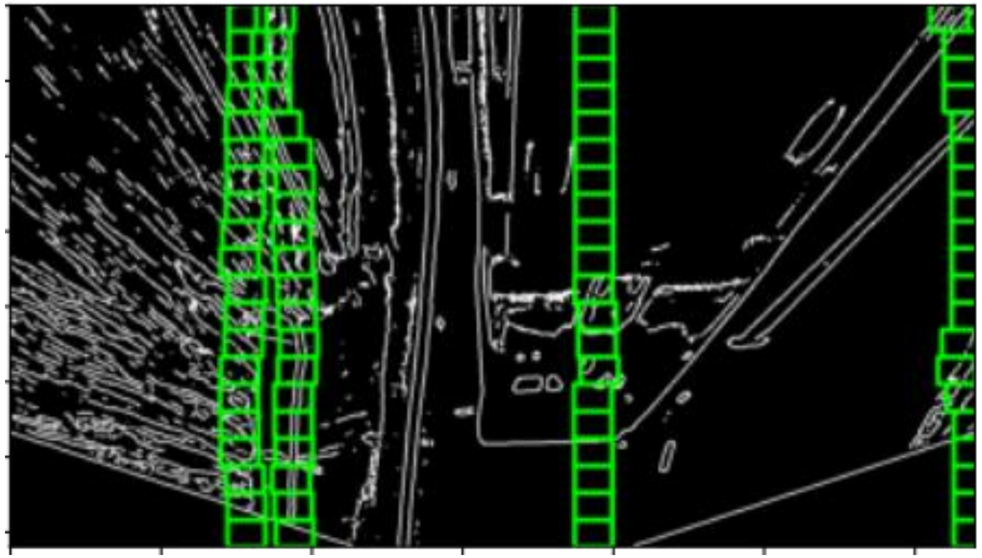
Result after Kmeans



Method 2 – which step makes it broken?

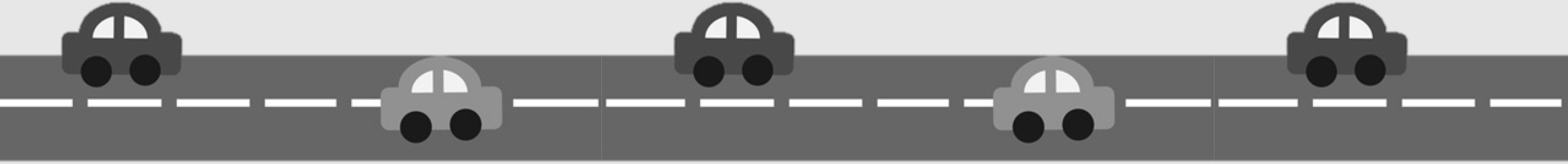
Perspective transform

Sliding windows



Deep leaning

Hourglass Network



Dataset

- In every clip folder, there will be a json file that contains the ground truth coordinates
- We split the data by its number

kivinju commented on Oct 20, 2018 Collaborator

LANE DETECTION CHALLENGE

dataset

- train_set.zip
- test_set.zip
- test_baseline.json

ground truth

- test_label.json

Num	Training set	Test set
Five	239	569
Four	2982	468
Three	404	1740
Two	1	5
Total	3626	2782

Directory Structure:

```
|
|----readme.md                  # description
|
|----clips/                     # video clips, 3626 clips
|------|
|------|----some_clip/          # Sequential images for the clip, 20 frames
|------|----...
|
|----label_data_0313.json           # Label data for lanes
|----label_data_0531.json           # Label data for lanes
|----label_data_0601.json           # Label data for lanes
```

Data Preprocess

For each image we have done some changes:

- Random flip
- Random translation (affine transformation)
- Random rotation
- Add random Gaussian noise
- Random change the intensity of the image
- Add random shadow to the image

Training: Read data

We set our batch size to 8. For each image, we generate a random number between [0,1] and from the random number we decide which category we choose and pick and image from there and go into training.

```
for _ in range(start, end):
    choose = random.random()
    if 0.8 <= choose:
        data = random.sample(self.train_data_five, 1)[0]
    elif 0.3 <= choose < 0.8:
        data = random.sample(self.train_data_four, 1)[0]
    elif 0.05 <= choose < 0.3:
        data = random.sample(self.train_data_three, 1)[0]
    elif choose < 0.05:
        data = random.sample(self.train_data_two, 1)[0]
```

Training--- Read Data (Continued)

1. Make ground truth point: x, y coordinates of the lane's ground truth sort by y coordinates and return

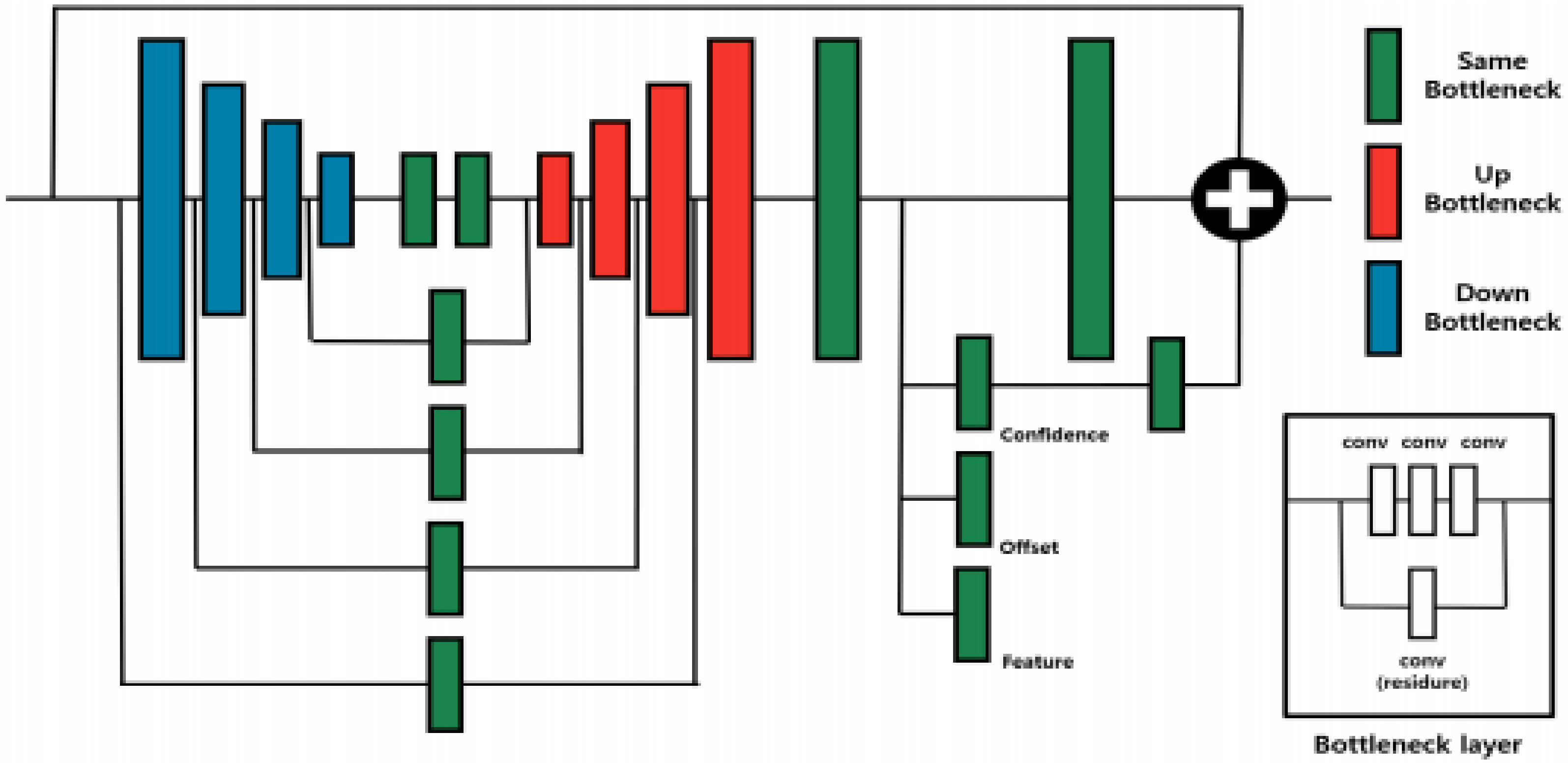
2. Make ground truth instance: classify the ground truth points into each instance and class
    a. Same instance (same lane): 1
    b. Different instance but same class (different lane but close point): 2
    c. Different instance and different class: 3

Because all the image have been resize to the same size, so the coordinates should adjust according to resize ratio

Format

```
{
  'raw_file': str. Clip file path.
  'lanes': list. A list of lanes. For each list of one lane, the elements are width values on image.
  'h_samples': list. A list of height values corresponding to the 'lanes', which means len(h_samples) == len(lanes[i])
}
```

Training--- Model



Two hourglass network

Training--- SGPN Loss

- Lane Loss
  - Exist confidence loss → if the predicted point exist
  - Non-exist confidence loss → if the predicted point not exist
  - Offset loss
- Instance Loss
  - Same instance
  - Different instance but same class

$$L_{SIM} = \sum_{i}^{N_p} \sum_{j}^{N_p} l(i, j)$$

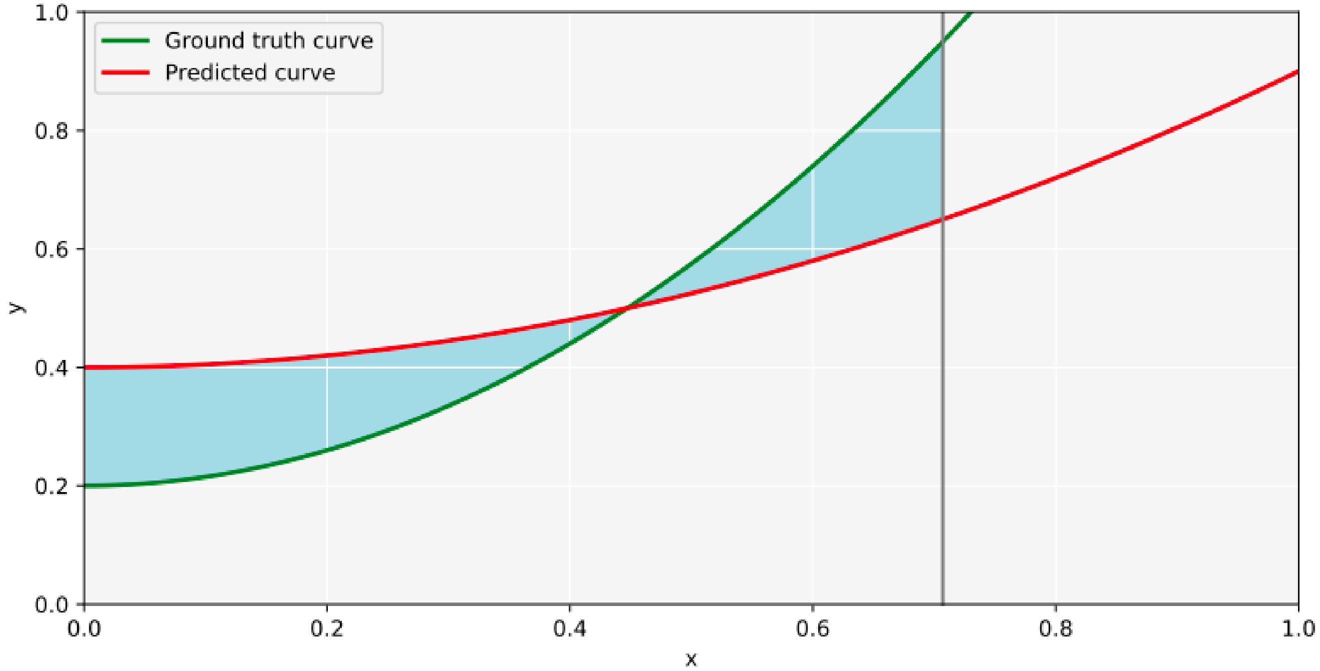
$$l(i,j) = \begin{cases} ||F_{SIM_i} - F_{SIM_j}||_2 & C_{ij} = 1 \\ \alpha \max(0, K_1 - ||F_{SIM_i} - F_{SIM_j}||_2) & C_{ij} = 2 \\ \max(0, K_2 - ||F_{SIM_i} - F_{SIM_j}||_2) & C_{ij} = 3 \end{cases}$$

Total Loss = Lane Loss + Instance Loss

Reference: SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation

Training--- Add OLE Loss

OLE Loss is the modified version of Geometric Loss



It minimizes the (squared) area between the predicted curve and ground truth curve up to a point

Evaluation

- We do evaluation every 10 epochs
- We calculate loss, accuracy, FP and FN

Before adding OLE Loss:

We train for 500 epochs the best evaluation result we can get right now is in the 100th epoch

Loss	Accuracy	FP	FN
1.11	0.97	0.02	0.14

After adding OLE Loss

We train for 1000 epochs the best evaluation result we can get right now is in the 543th epochs

Loss	Accuracy	FP	FN
1.57	0.94	0.08	0.20

Testing and Post Processing

We do testing (output with lanes on testing image) every 1000 iterations (an epoch is 454 iterations)

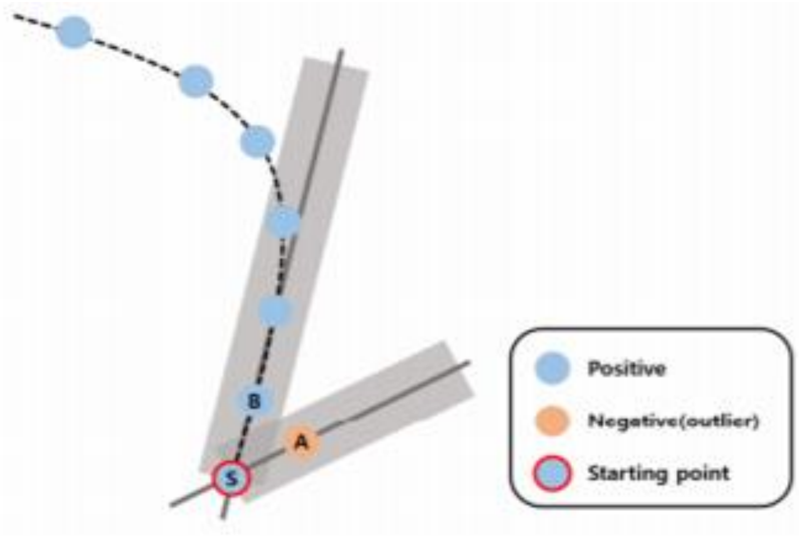
Post processing:

- Step 1: Find six starting points. Starting points are defined as the three lowest points and the three leftmost or rightmost points. If the predicted lane is on the left related to the center of the image, the leftmost point are selected
- Step 2: Select three closest points to the starting point among points that are higher than each starting point
- Step 3: Consider a straight line connecting two points that are selected at step 1 and 2
- Step 4: Calculate the distance between the straight line and other points
- Step 5: Count the number of the points that are within the margin. The margin γ , is set to 1

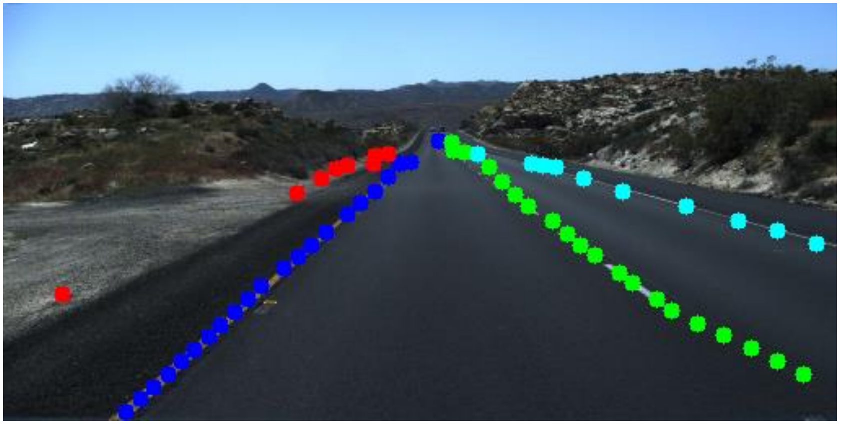
Testing and Post Processing (Continued)

Post processing:

- Step 6: Select the point that has maximum and larger count than threshold as new starting point, and consider that the point to the same cluster with starting point. We set the threshold to twenty percent of the remaining points
- Step 7: Repeat from step 2 to step 6 until no points are found at step 2
- Step 8: Repeat from step 1 to step 7 for all starting point, and consider the maximum length cluster as a result lane
- Step 9: Repeat from step 1 to step 8 for all predicted lanes

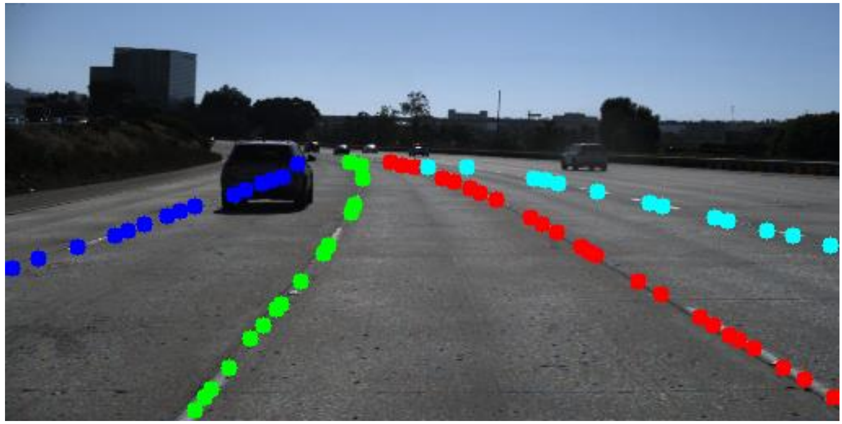


Results - In TuSimple test set

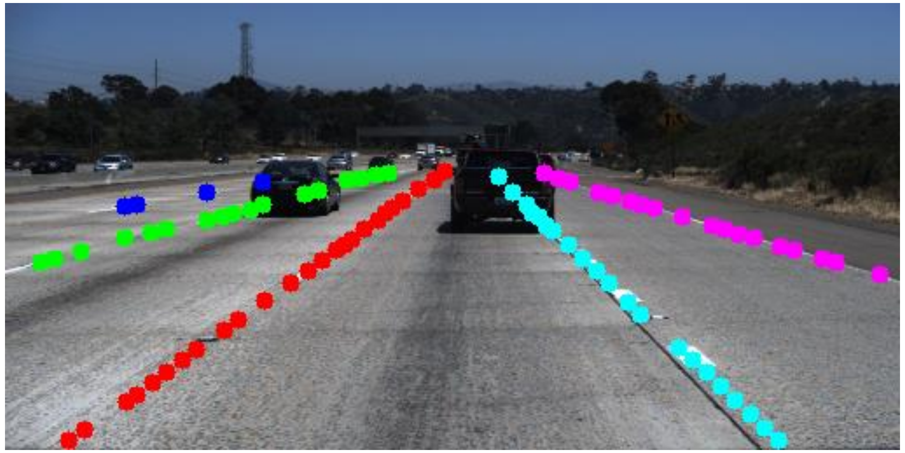


2000 iterations

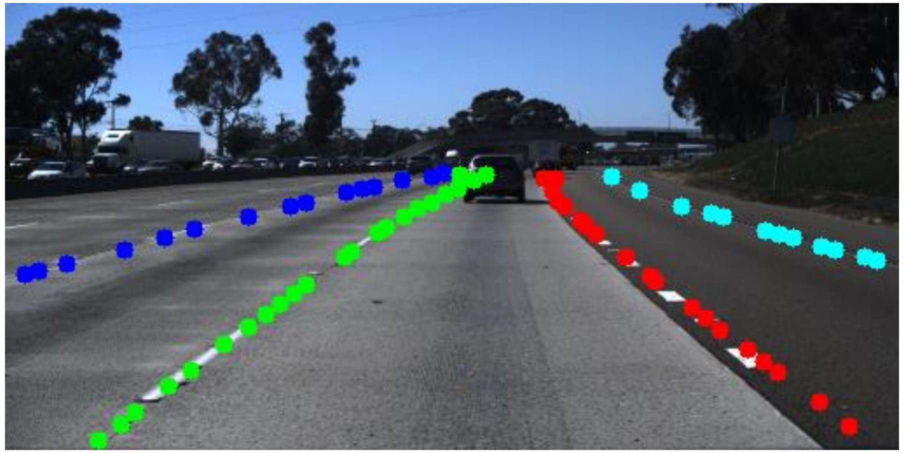
56000 iterations



102000 iterations



143000 iterations



225000 iterations

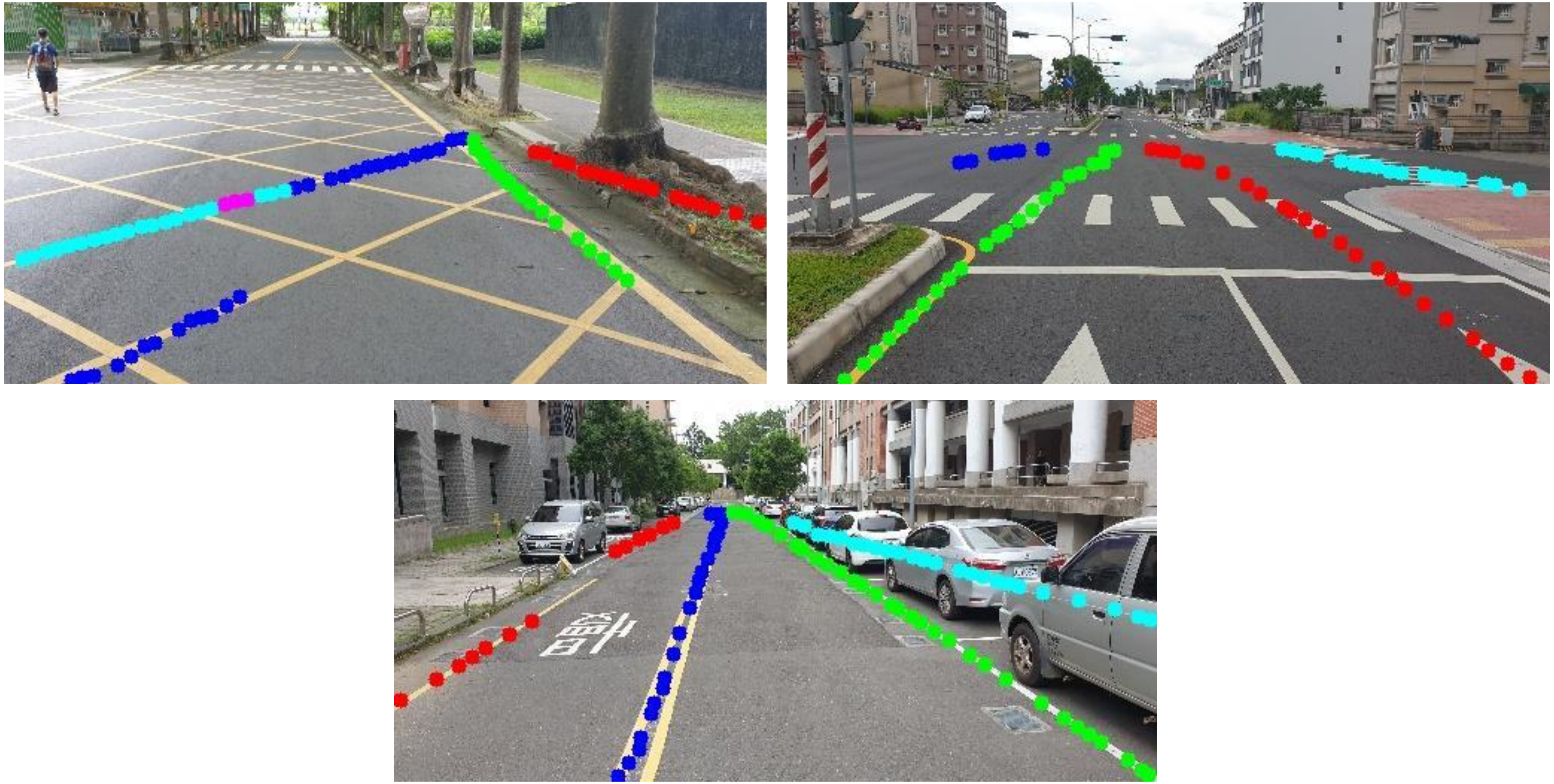
Results

Results

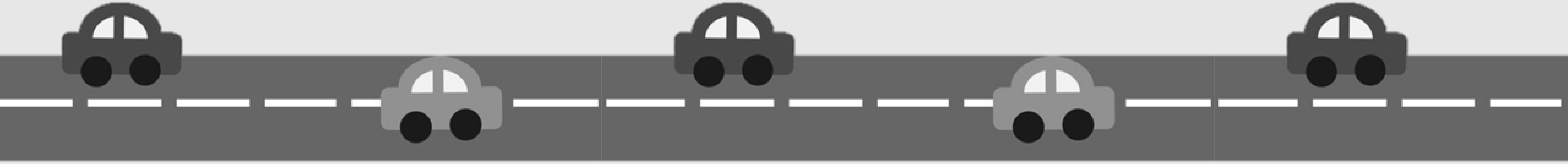
The images recorded from different angle may lead to different accuracy.

Results

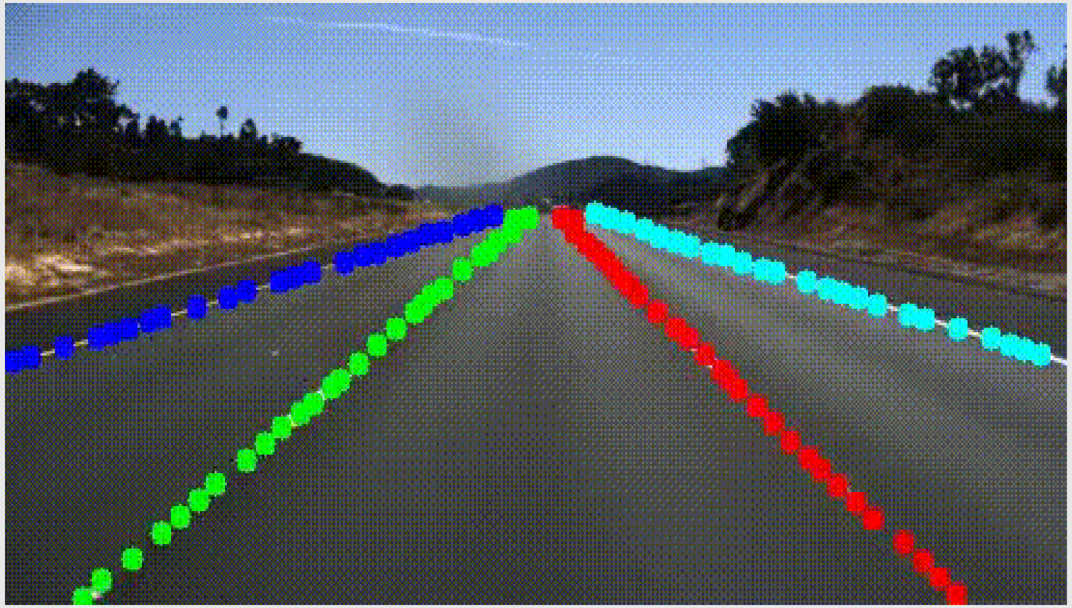
But in this situation...



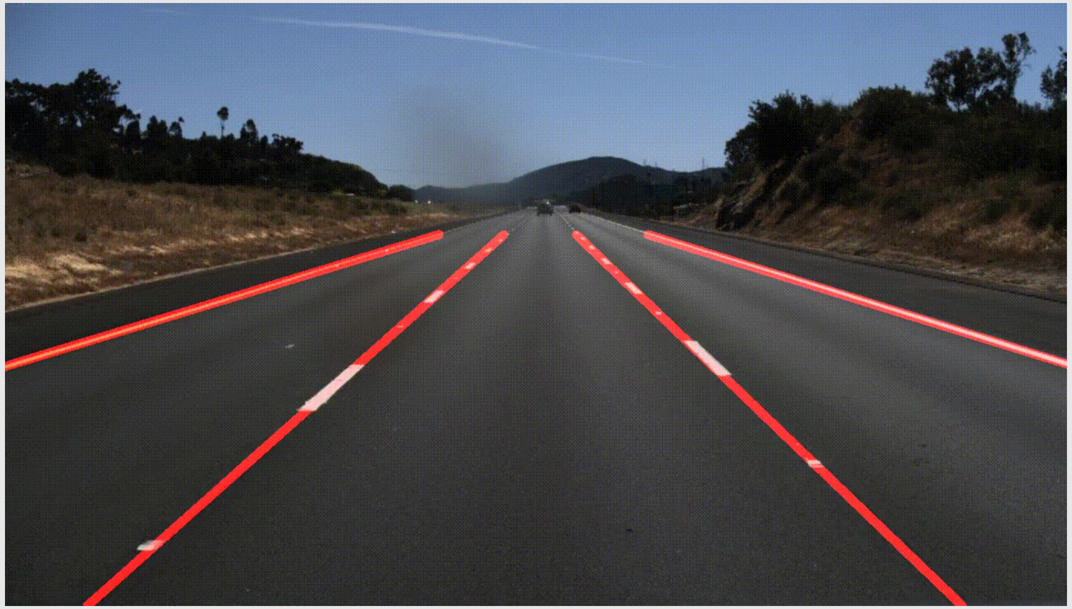
Result comparison



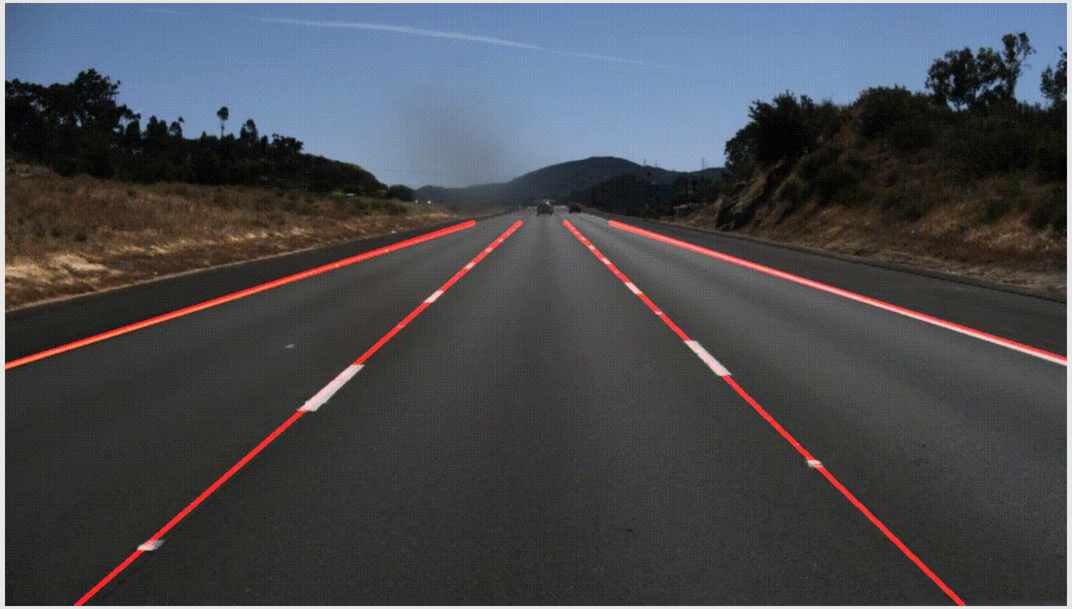
Method 3 ▸



▼ Method 1



▼ Method 2



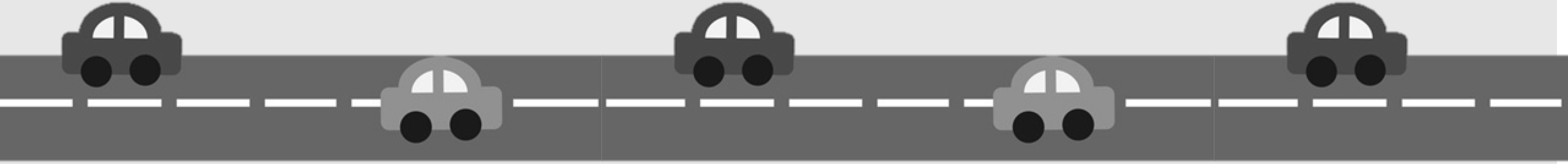
Accuracy:

For Method1 and Method2 we only test for the first 500 images in the test set

	Method 1	Method 2	Method 3
Accuracy	0.73	0.86	0.97
FP	0.57	0.40	0.02
FN	0.48	0.27	0.14

Conclusion

- In this final project, we succesfully detect multiple lanes with both traditional and deep learning methods.
- Traditional methods has limited accuracy, and the quality of resutl can be easily affected by cars and obstacles in the image.
- Using Hourglass network and SGPN loss, deep learning method achive very high accuracy.
- Both traditional and deep learning methods are sensitive to camera perspective. However, deep learning methods still output reasonable result in those condition where traditional methods fails dramatically.



Thank you for listening!