Assignment: Lane Detection

Date: 2025-04-04

Author: Joungbin Choi 22200757

Github: repository link

Introduction

1. Objective

Goal: Find the lanes on the road and a vanishing point to drive the center of the road.

2. Preparation

Software Installation

- OpenCV 3.83, Clion2024.3.5
- CUDA 12.6, cudatoolkit 12.6, C++

Dataset

There are 2 road pictures:

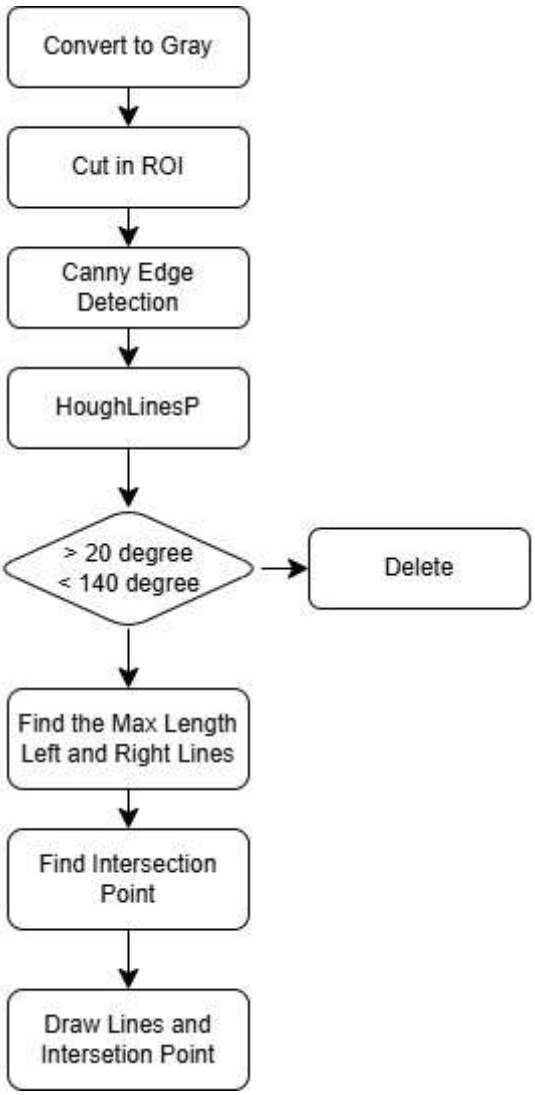
- Driving at the center of lanes
- Shifted to left lane

Dataset link:

- Download the Lane_center image
- Download the Lane_changing image

Algorithm

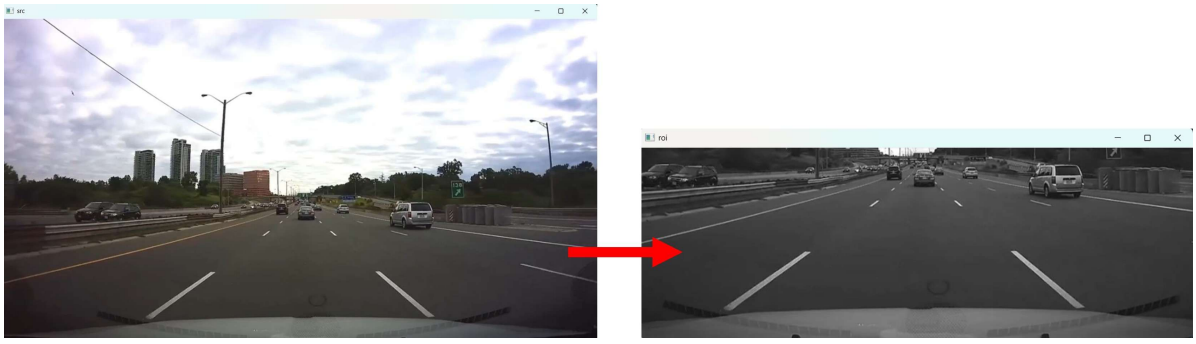
1. Overview



2. Procedure

Filtering and ROI

To detect lines clearly, some image processes are needed. First process is converting to gray scale. Because the line is white and the road is black, it is possible to see more differences in gray scale. Second process is cropping the image to a certain ROI. Since the lanes are on the road, it is not necessary to see all ROI.



Canny

Canny edge detection is utilized to find the edges of images.

Canny Edge Detection Flow

1. Gaussian Filtering

Decrease the noise

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2+y^2}{2\sigma})$$

2. Calculate Gradient

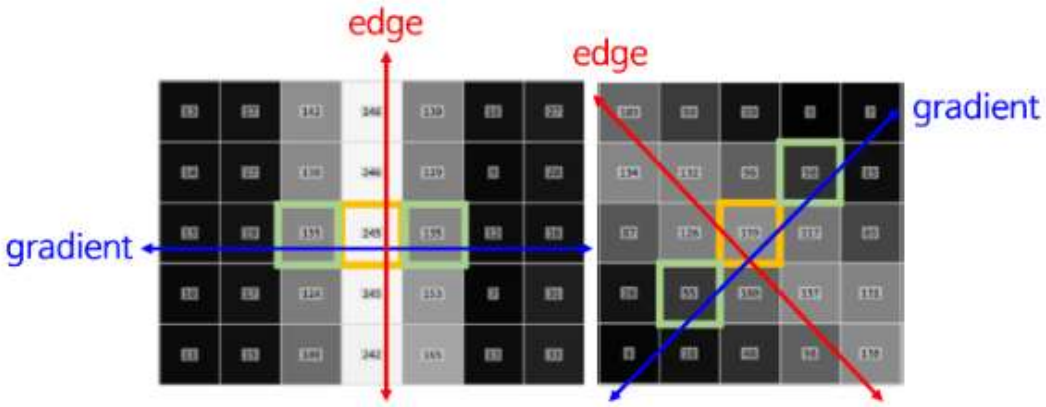
   With Sobel filter, find the differential values in x, y direction. With those values, the magnitude and direction of gradient can be calculated.

$$magnitude : ||f|| = \sqrt{f_x^2 + f_y^2}$$

$$phase : \theta = tan^{-1}(\frac{f_y}{f_x})$$

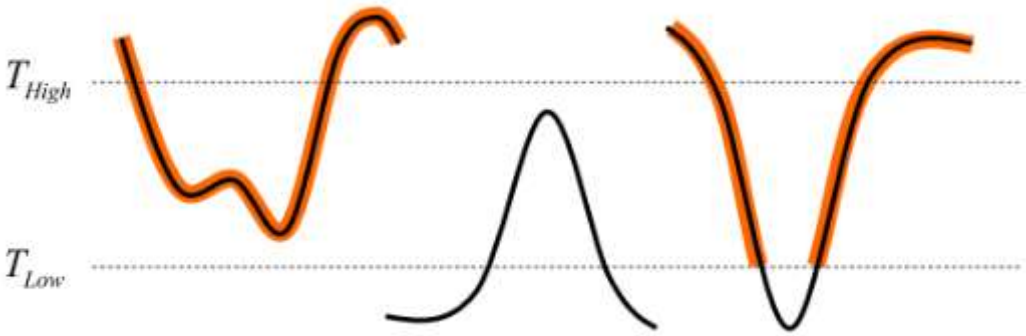
3. NMS(Non-maximum Suppression)

   This process is the method to decrease noise of line and have a thinner line which is closer to real line. Find the a local maximum gradient value of orthogonal direction of edge.

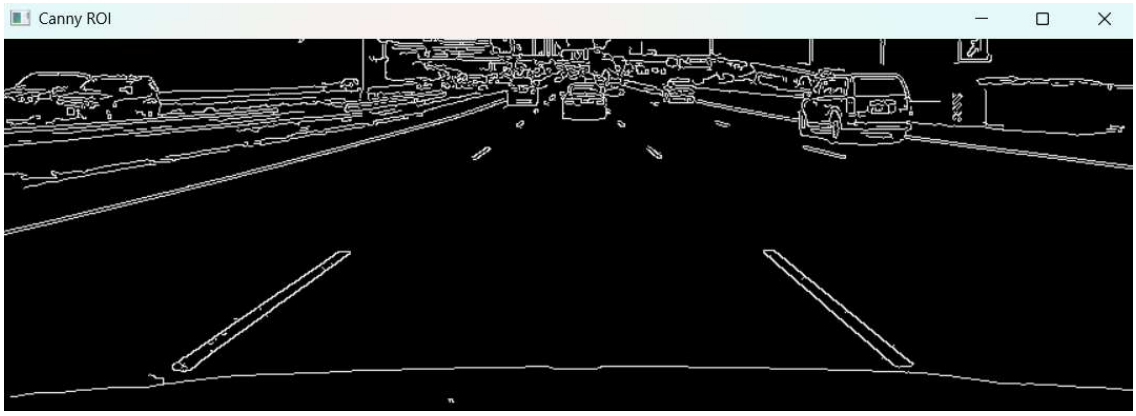


4. Hysteresis Edge Tracking

   This method it to find the strong edge with threshold values which the user provides. If the gradient value is over the high threshold value, it means there is a line. If the gradient value is middle of high threshold and low threshold value, it could be a line or not. If that potential line is continued until it overs the high threshold, it is a line. Otherwise, if it falls under the low threshold, it is not a line. If the gradient value is less than low threshold value, it is not a line.



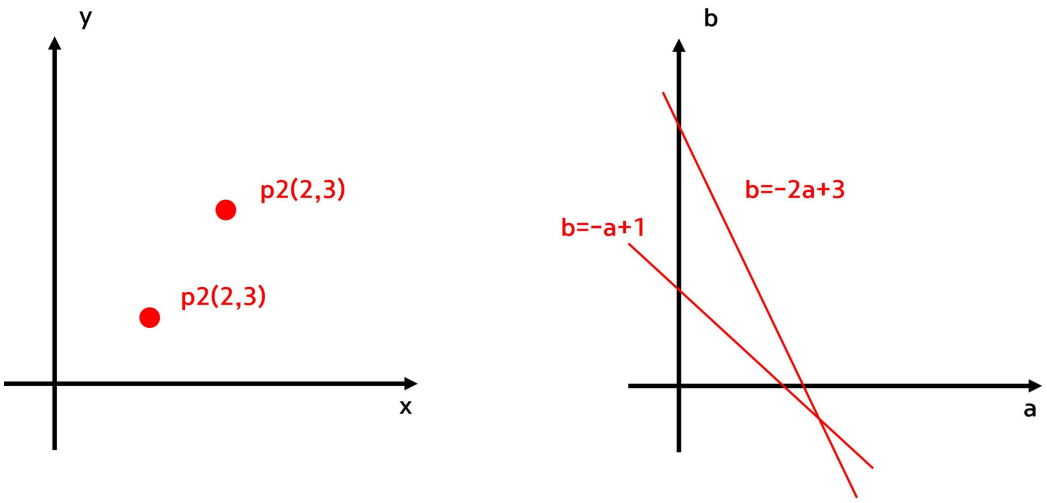
The result of `Canny` is shown below.



HoughLinesP

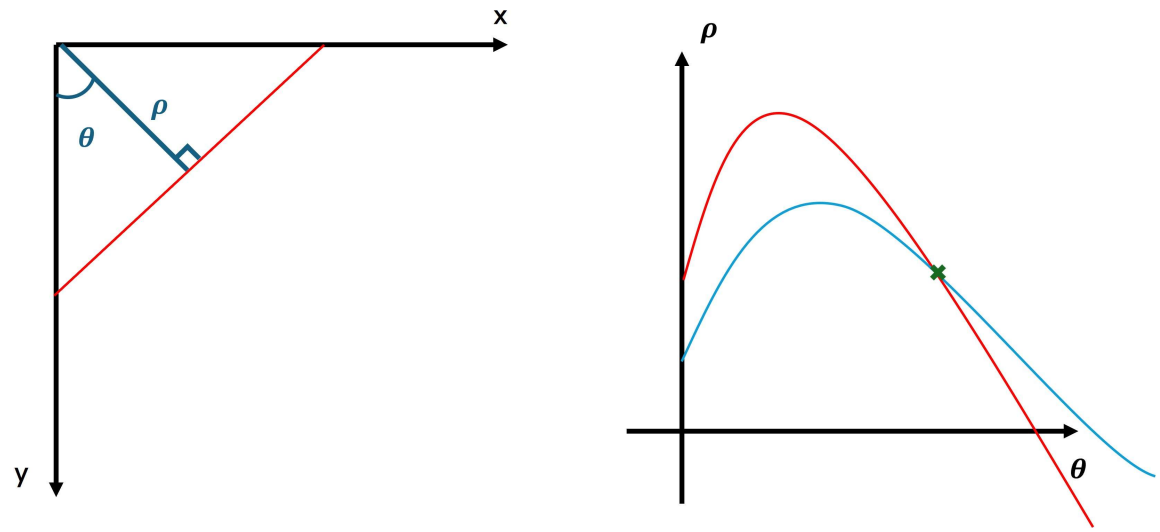
`HoughLinesP` is picking random pixels to find the lines. It is based on `HoughLines` with Probabilistic.

In xy-plane the line has a y-intersection and a slope. Let's say a slope as `a` and y-intersection as `b`, it is able to take the points in xy-plane to ab-plane. Then, the points in xy-plane are being line in ab-plane. The intersection of those two liens in ab-plane is the line that connects two points in xy-plane.

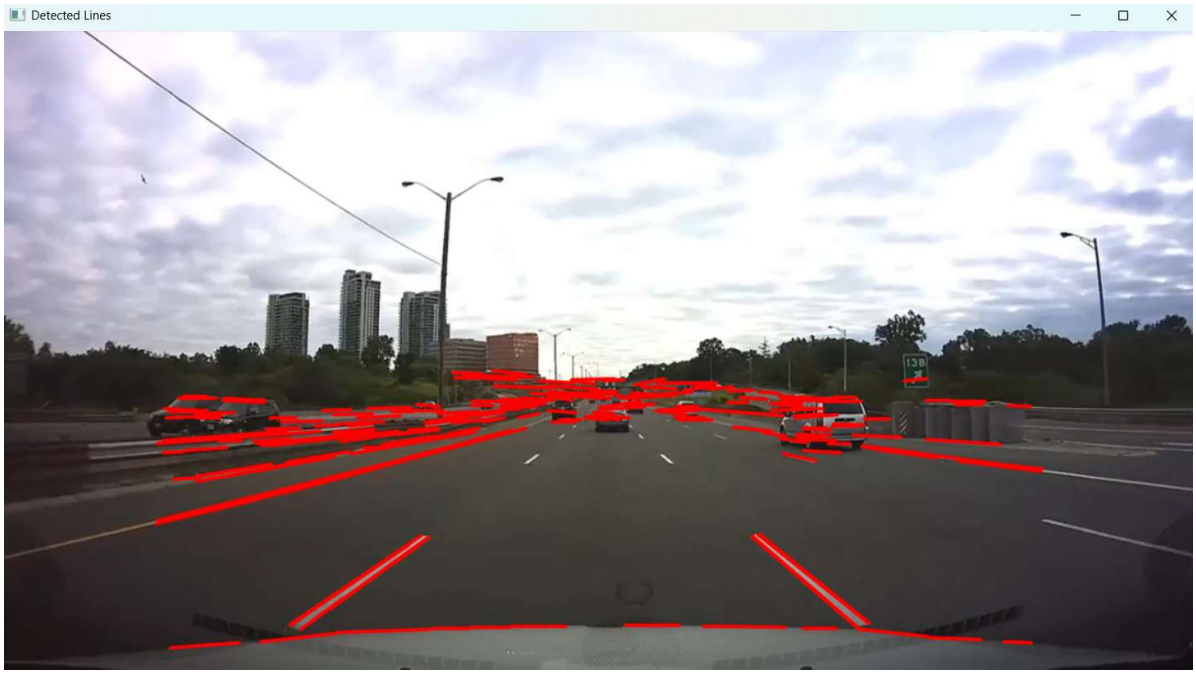


However, there is a problem when the line is parallel to y-axis. Therefore, it transports to a polar-coordinate. In xy-plane, each line has a angle(theta) and length(rho) from the origin. They can be expressed as an equation below.

$$x\sin(\theta) + y\cos(\theta) = \rho$$

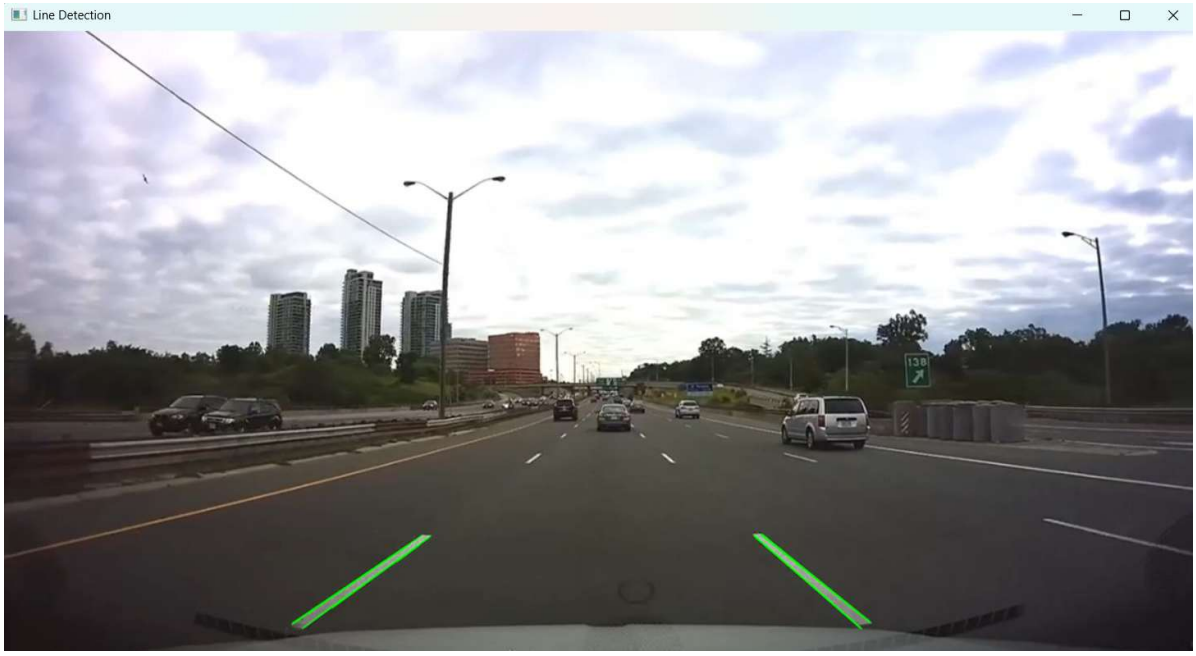


The result of `HoughLinesP` is shown below.

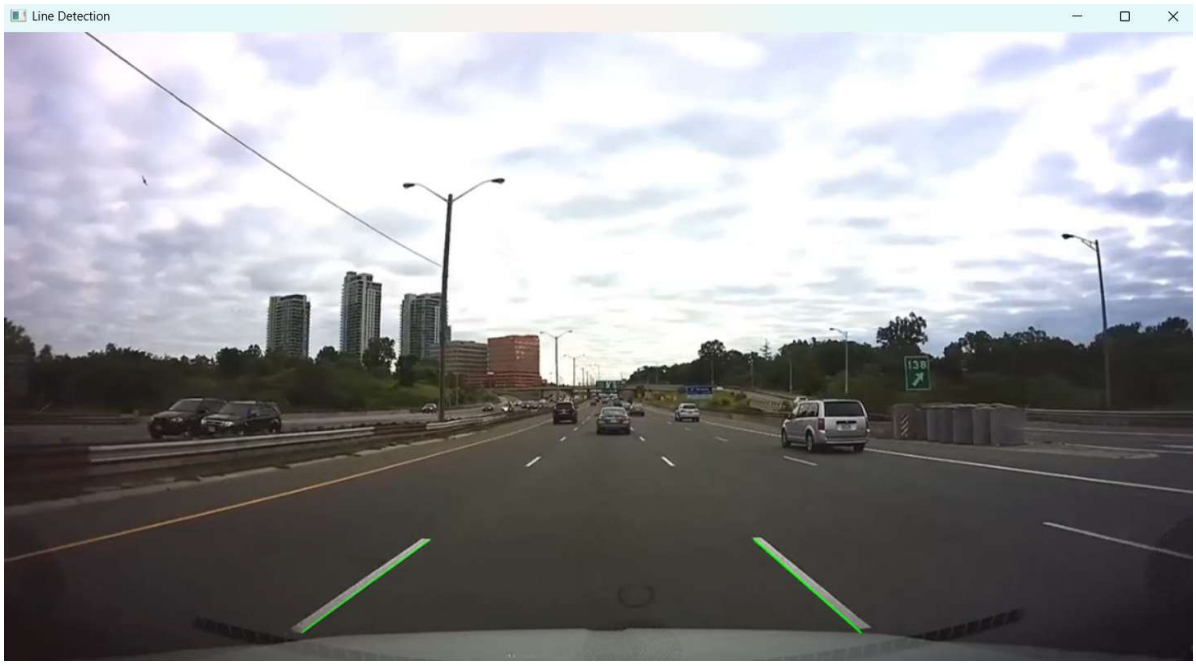


Selecting Lines

Beside lines on the road, too many other lines are detected because they are all edges in after Canny method. Since lines on the road are in certain range of angle, some lines are excluded which is less than 20 degree or bigger than 140 degree.



After angle excluding process, multiple lines are detected on each side of line block. The line with the maximum length is chosen on left and right ROI.



Draw the Line and Find the Intersection

The logic of drawing the line end to end is based on the slope of lines. The start and end points on x-axis is 0 to column size of original source image. Using the linear equation the start and end points on y-axis can be calculated. By the way, left and right lines are crossed which means their starting point should be about the opposite side.

```
cv::Vec4i leftLine = lines[leftindex];
double leftSlope = (double)(leftLine[3] - leftLine[1]) / (leftLine[2] -
leftLine[0]);
int leftEndX = 0;
int leftEndY = (leftLine[1]+y_roi) - leftSlope * (leftLine[0]+x_roi);
int leftStartX = src.cols;
int leftStartY = leftSlope * leftStartX + leftEndY;

cv::Vec4i rightLine = lines[rightindex];
double rightSlope = (double)(rightLine[3] - rightLine[1]) / (rightLine[2] -
rightLine[0]);
int rightStartX = 0;
int rightStartY = (rightLine[1]+y_roi) - rightSlope * (rightLine[0]+x_roi);
int rightEndX = src.cols;
int rightEndY = rightSlope * rightEndX + rightStartY;
```

To find the intersection point of left and right lines, a simple linear equation is used as below.

$$y = m \cdot x + b$$

where `m` is the slope and `b` is the y-interception. Both left and right lines have their own equation. Since we want to get y value which both two equations are crossed, both two equations can be simplified as below

$$Left\ Slope \cdot b_{left} = Right\ Slope \cdot b_{right}$$
$$x_{intersection} = (b_{right} - b_{left})/(Left\ Slope - Right\ Slope)$$

The y-intersection point can be calculated with the x-intersection point.

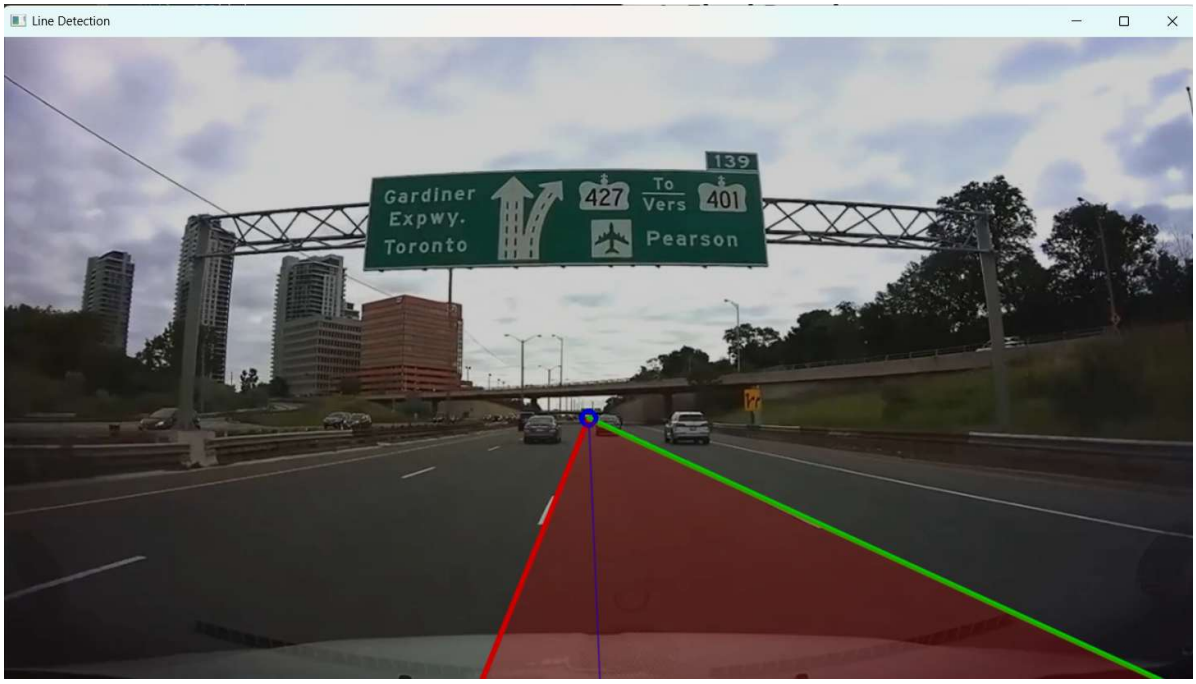
The functions `line` and `circle` are utilized to draw left, right, and intersection point. The area of lines and vanishing(intersection) points are colored with the function `fillPoly`.

Result and Discussion

1. Final Result

The image of driving on the center of the road.

The image of changing the lines.



2. Discussion

As the result images show, the lines are detected well in both condition. However, it is not tested on real driving situation, so it cannot be robust enough to utilize on real driving. More preprocessing and controlling exceptions are needed.

Conclusion

In automatic driving industry, camera processing is essential technique such as line detecting. To detect lines with image processing, many other methods exist. Among them, `Canny Edge Detection` and `HoughLinesP` are used in this assignment. In addition, the intersection point is calculated with two linear equations. The ratio of length between the intersection point to left and

right tells the car driving on the center or not.

Appendix

```
//
// Created by joung on 25. 4. 1.
//

#include <iostream>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>

using namespace cv;
using namespace std;

int main() {
  Mat src, gray, roiImg, cImg;

  // Load Image
  src = imread("../Image/Lane_changing.jpg");
  if (src.empty()) {
    cerr << "Failed to load image!" << endl;
    return -1;
  }

  // Preprocess
  cvtColor(src, gray, COLOR_BGR2GRAY);
  int y_roi = 330;
  int x_roi = 150;
  Rect roi(x_roi, y_roi, gray.cols - 2 * x_roi, gray.rows - y_roi);
  roiImg = gray(roi);

  // Finding Line
  Canny(roiImg, cImg, 50, 200, 3);
  vector<Vec4i> lines;
  HoughLinesP(cImg, lines, 1, CV_PI / 180, 30, 20, 5);

  // Find the max and min length
  double minTheta = 20 * CV_PI / 180;
  double maxTheta = 140 * CV_PI / 180;
  double leftmaxlength = 0;
  double rightmaxlength = 0;
  int leftindex = 0;
  int rightindex = 0;

  vector<Vec4i> filtered_line;
  int count = 0;

  for (size_t i = 0; i < lines.size(); i++) {
    Vec4i l = lines[i];
    double length = sqrt(pow((l[3] - l[1]),2)+pow((l[2] - l[0]), 2));
    double theta = atan2((l[3] - l[1]), (l[2] - l[0]));
    theta = abs(theta);
```

```
    if (theta > minTheta && theta < maxTheta) {
      if (l[0]<roiImg.cols/2 && length > leftmaxlength) {
        leftmaxlength = length;
        leftindex = i;
      }
      else if (l[0]>roiImg.cols/2 && length > rightmaxlength) {
        rightmaxlength = length;
        rightindex = i;
      }
    }
  }

  Vec4i leftLine = lines[leftindex];
  Vec4i rightLine = lines[rightindex];

  // Left Line
  double leftSlope = (double)(leftLine[3] - leftLine[1]) / (leftLine[2] -
leftLine[0]);
  int leftEndX = 0;
  int leftEndY = (leftLine[1]+y_roi) - leftSlope * (leftLine[0]+x_roi);
  int leftStartX = src.cols;
  int leftStartY = leftSlope * leftStartX + leftEndY;

  cout << leftStartX << " " << leftStartY << endl;
  cout << leftEndX << " " << leftEndY << endl;

  // Right Line
  double rightSlope = (double)(rightLine[3] - rightLine[1]) / (rightLine[2] -
rightLine[0]);
  int rightStartX = 0;
  int rightStartY = (rightLine[1]+y_roi) - rightSlope * (rightLine[0]+x_roi);
  int rightEndX = src.cols;
  int rightEndY = rightSlope * rightEndX + rightStartY;

  cout << rightStartX << " " << rightStartY << endl;
  cout << rightEndX << " " << rightEndY << endl;

  // Interception Point
  double Left_Intersection = leftEndY - leftSlope * leftEndX;
  double Right_Intersection = rightStartY - rightSlope * rightStartX;

  double X_intersection = (Right_Intersection - Left_Intersection) / (leftSlope
- rightSlope);
  double Y_intersection = leftSlope * X_intersection + Left_Intersection;

  Point inter = Point(cvRound(X_intersection) , cvRound(Y_intersection) );

  // Inline Area
  vector<Point> area = {
    Point(leftEndX , leftEndY ),
    Point(rightEndX , rightEndY ),
    inter
  };

  // Draw Lines and Circle on interception
  line(src, inter, Point(src.cols/2, src.rows), Scalar(255, 0, 0), 1, LINE_AA);
  line(src, Point(leftEndX, leftEndY), inter, Scalar(0, 0, 255), 3, LINE_AA);
  line(src, inter, Point(rightEndX, rightEndY), Scalar(0, 255, 0), 3, LINE_AA);
```

```
  circle(src, inter, 7, Scalar(255, 0, 0), 3);

  // Color Area
  vector<vector<Point>> areaPoly = {area};
  Mat temp = src.clone();
  fillPoly(temp, areaPoly, Scalar(0, 0, 255));
  addWeighted(temp, 0.2, src, 0.6, 0, src);

  imshow("Line Detection", src);
  waitKey(0);

  return 0;
}
```