제주도에서 자율주행을 

윤재호, 이현수

제주도에서 자율주행을 🚗

윤재호, 이현수

제주도에서 자율주행을 🚗

윤재호, 이현수

Contents

01. Planning

Planning

차선인식 알고리즘

1. bird's eye view
2. HSV로 변환
3. 화면 반전하여 흰색 차선으로 탐지
4. 초기 차선 위치 설정
5. sliding window 진행

예외 처리

1. 평균 밝기 유지
2. 탐지 안될 경우 대칭
3. 지난 프레임 윈도우 사용

02. Code

Code (Include, namespace, main)

```
#include <iostream>
#include "opencv2/opencv.hpp"
#include <fstream>
#include <algorithm >

using namespace std;
using namespace cv;
```

include 선언

namespace 선언

```
int main()
{
    VideoCapture cap("../data/subProject.avi");

    if (!cap.isOpened()) {
        cerr << "Camera open failed" << endl;
        return -1;
    }
```

영상 불러오기

파일이 없을 경우 종료

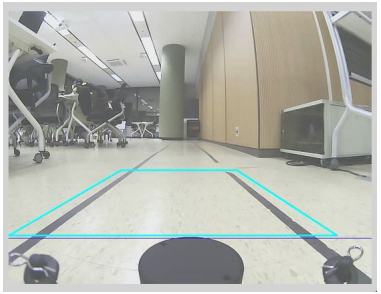
```
int width = cvRound(cap.get(CAP_PROP_FRAME_WIDTH));
int height = cvRound(cap.get(CAP_PROP_FRAME_HEIGHT));
```

영상의 크기 불러와서 설정

Code (ROI)

```
int w = (int)width * 1.5, h = (int)height * 1.5;
```

```
vector<Point2f> src_pts(4);
vector<Point2f> dst_pts(4);

// 파란색 선 없는 roi
src_pts[0] = Point2f(0, 395); src_pts[1] = Point2f(198, 280); src_pts[2] = Point2f(403, 280); src_pts[3] = Point2f(580, 395);
dst_pts[0] = Point2f(0, h - 1); dst_pts[1] = Point2f(0, 0); dst_pts[2] = Point2f(w - 1, 0); dst_pts[3] = Point2f(w - 1, h - 1);
```



```
Mat per_mat_todst = getPerspectiveTransform(src_pts, dst_pts);
Mat per_mat_tosrc = getPerspectiveTransform(dst_pts, src_pts);
```

원근변환

```
while (true) {
    (...)

    Mat roi;
    warpPerspective(frame, roi, per_mat_todst, Size(w, h), INTER_LINEAR);
    polylines(frame, pts, true, Scalar(255, 255, 0), 2);
```

ROI 설정

Code (binary)

```
Mat hsv;
Mat v_thres = Mat::zeros(w, h, CV_8UC1);
cvtColor(roi, hsv, COLOR_BGR2HSV);

vector<Mat> hsv_planes;
split(roi, hsv_planes);
```

```
Mat v_plane = hsv_planes[2];
v_plane = 255 - v_plane;

int means = mean(v_plane)[0];
v_plane = v_plane + (100 - means);

GaussianBlur(v_plane, v_plane, Size(), 1.0);
inRange(v_plane, 125, 255, v_thres);
```

HSV 색공간

색 반전, 성능 ↑

불안정한 밝기 보정

가우시안 필터, 이진화

Code (lane x index initialization)

```
int left_l_init = 0, left_r_init = 0;
int right_l_init = 960, right_r_init = 960;
for (auto x = 0; x < w; x++) {
    if (x < w / 2) {
        if (v_thres.at<uchar>(h - 1, x) == 255 && left_l_init == 0) {
            left_l_init = x;
            left_r_init = x;
        }
        if (v_thres.at<uchar>(h - 1, x) == 255 && left_r_init != 0) {
            left_r_init = x;
        }
    }
    else {
        if (v_thres.at<uchar>(h - 1, x) == 255 && right_l_init == 960) {
            right_l_init = x;
            right_r_init = x;
        }
        if (v_thres.at<uchar>(h - 1, x) == 255 && right_r_init != 960) {
            right_r_init = x;
        }
    }
}

int left_start = (left_l_init + left_r_init) / 2;
int right_start = (right_l_init + right_r_init) / 2;
```

초기 위치 설정

차선의 시작점과
끝점 탐색

찾은 시작점과 끝점을 통해
차선의 중간점 저장

Code (sliding window)

```
left_line, right_line = n_window_sliding(left_start, right_start, roi, v_thres,
                                         w, h, lpoints, rpoints, per_mat_tosrc);
```

n window sliding 함수

```
Vec4f n_window_sliding(int left_start, int right_start, Mat roi, Mat v_thres, int w, int h,
                       vector<Point>& lpoints, vector<Point>& rpoints, Mat per_mat_tosrc) {

    int nwindows = 12;
    int window_height = (int)(h / nwindows);
    int window_width = (int)(w / nwindows * 1.5);
```

window 개수,
window 크기 지정

```
int margin = window_width / 2;
```

중앙값이 검출되므로
중앙값 기준 좌우 크기

```
vector<Point> mpoints(nwindows);
```

차선 중앙을 검출하고,
저장하기 위한 변수들

```
int lane_mid = w / 2;

int win_y_high = h - window_height;
int win_y_low = h;

int win_x_leftb_right = left_start + margin;
int win_x_leftb_left = left_start - margin;

int win_x_rightb_right = right_start + margin;
int win_x_rightb_left = right_start - margin;
```

window 상/하/좌/우
크기 지정

Code (sliding window)

```
lpoints[0] = Point(left_start, (int)((win_y_high + win_y_low) / 2));
rpoints[0] = Point(right_start, (int)((win_y_high + win_y_low) / 2));
mpoints[0] = Point((int)((left_start + right_start) / 2), (int)((win_y_high + win_y_low) / 2));


rectangle(roi, Rect(win_x_leftb_left, win_y_high, window_width, window_height), Scalar(0, 150, 0), 2);
rectangle(roi, Rect(win_x_rightb_left, win_y_high, window_width, window_height), Scalar(150, 0, 0), 2);
```

```
for (int window = 1; window < nwindows; window++) {

    win_y_high = h - (window + 1) * window_height;
    win_y_low = h - window * window_height;

    win_x_leftb_right = left_start + margin;
    win_x_leftb_left = left_start - margin;

    win_x_rightb_right = right_start + margin;
    win_x_rightb_left = right_start - margin;

    int offset = (int)((win_y_high + win_y_low) / 2);

    int pixel_thres = window_width * 0.2;
```

왼/오른 차선 좌표,
차선 중앙 좌표 저장

window draw

1번 window부터 시작

window 좌표 재설정

window 중앙을 차선 탐색
위치로 지정

nonzero에 대한
threshold 지정

Code (sliding window)

```
int ll = 0, lr = 0; int rl = 960, rr = 960;
int li = 0;
vector<int> lhigh_vector(window_width + 1);
for (auto x = win_x_leftb_left; x < win_x_leftb_right; x++) {
    li++;
    lhigh_vector[li] = v_thres.at<uchar>(offset, x);


    if (v_thres.at<uchar>(offset, x) == 255 && ll == 0) {
        ll = x;
        lr = x;
    }
    if (v_thres.at<uchar>(offset, x) == 255 && lr != 0) {
        lr = x;
    }
}
 int ri = 0;
 vector<int> rhigh_vector(window_width + 1);
 for (auto x = win_x_rightb_left; x < win_x_rightb_right; x++) {
    ri++;
    rhigh_vector[ri] = v_thres.at<uchar>(offset, x);
    if (v_thres.at<uchar>(offset, x) == 255 && rl == 960) {
        rl = x;
        rr = x;
    }
    if (v_thres.at<uchar>(offset, x) == 255 && lr != 960) {
        rr = x;
    }
 }
```

```
int lnonzero = countNonZero(lhigh_vector);
int rnonzero = countNonZero(rhigh_vector);
```

초기 위치 지정

offset의 nonzero픽셀 개수 구함

차선의 시작점과 끝점 저장

li, ri는 nonzero를 구하기 위한 벡터에만 사용됨

nonzero 픽셀 수를 구함

Code (sliding window)

```
left_start = (ll + lr) / 2;
right_start = (rl + rr) / 2;



int lane_mid = (right_start + left_start) / 2;
int left_diff = lane_mid - left_start;
int right_diff = -(lane_mid - right_start);
```

구한 시작점과 끝점으로 window위치 지정

차선 중간과 대칭에 사용하기 위한 중앙값과의 차 추출

```
if (lnonzero < pixel_thres && rnonzero > pixel_thres) {
    left_start = lane_mid - right_diff;
    lane_mid = right_start - right_diff;
}
else if (lnonzero > pixel_thres && rnonzero < pixel_thres) {
    right_start = lane_mid + left_diff;
    lane_mid = left_start + left_diff;
}
```

한 쪽 차선을 잡지 못할 경우 반대편 차선과의 대칭을 통해 구함

```
if (lnonzero < pixel_thres && rnonzero > pixel_thres) {
    left_start = lpoints[window].x;
    lane_mid = (right_start + left_start) / 2;
}
else if (lnonzero > pixel_thres && rnonzero < pixel_thres && rpoints[window].x != 0) {
    right_start = rpoints[window].x;
    lane_mid = (right_start + left_start) / 2;
}
```

또는 한 쪽 차선을 잡지 못할 경우 지난 프레임의 값 사용

Code (sliding window)

```
mpoints[window] = Point(lane_mid, (int)((win_y_high + win_y_low) / 2));
lpoints[window] = Point(left_start, (int)((win_y_high + win_y_low) / 2));
rpoints[window] = Point(right_start, (int)((win_y_high + win_y_low) / 2));
```

윈도우마다 탐색한 좌표 저장

```
Vec4f left_line, right_line, mid_line;
fitLine(lpoints, left_line, DIST_L2, 0, 0.01, 0.01);
fitLine(rpoints, right_line, DIST_L2, 0, 0.01, 0.01);
fitLine(mpoints, mid_line, DIST_L2, 0, 0.01, 0.01);
```

fitline을 통해 차선에 대한 직선 추출

line = Vec4f(x vector, y vector, x center, y center)

Code (sliding window)

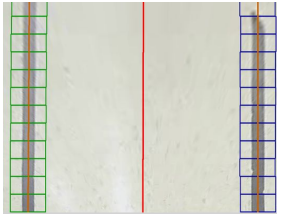
```
if (left_line[1] > 0) {
    left_line[1] = -left_line[1];
}
if (right_line[1] > 0) {
    right_line[1] = -right_line[1];
}
if (mid_line[1] > 0) {
    mid_line[1] = -mid_line[1];
}
```

항상 직선의 방향을 위에서 아래로 그림

```
int lx0 = left_line[2], ly0 = left_line[3];
int lx1 = lx0 + h/2 * left_line[0], ly1 = ly0 + h/2 * left_line[1];
int lx2 = 2 * lx0 - lx1, ly2 = 2 * ly0 - ly1;

int rx0 = right_line[2], ry0 = right_line[3];
int rx1 = rx0 + h/2 * right_line[0], ry1 = h/2 * right_line[1];
int rx2 = 2 * rx0 - rx1, ry2 = 2 * ry0 - ry1;

int mx0 = mid_line[2], my0 = mid_line[3];
int mx1 = mx0 + h/2 * mid_line[0], my1 = my0 + h/2 * mid_line[1];
int mx2 = 2 * mx0 - mx1, my2 = 2 * my0 - my1;

line(roi, Point(lx1, ly1), Point(lx2, ly2), Scalar(0, 100, 200), 3);
line(roi, Point(rx1, ry1), Point(rx2, ry2), Scalar(0, 100, 200), 3);
line(roi, Point(mx1, my1), Point(mx2, my2), Scalar(0, 0, 255), 3);
```

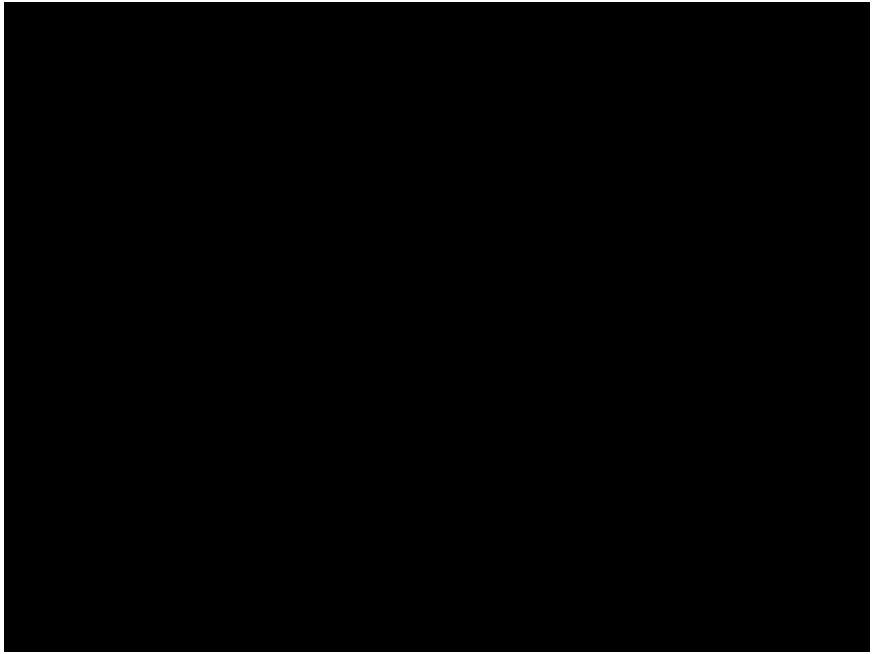
x1 = 중앙값 + h/2 * x vector
y1 = 중앙값 + h/2 * y vector = 0

y2 = y1 - (y0 - y1) = h



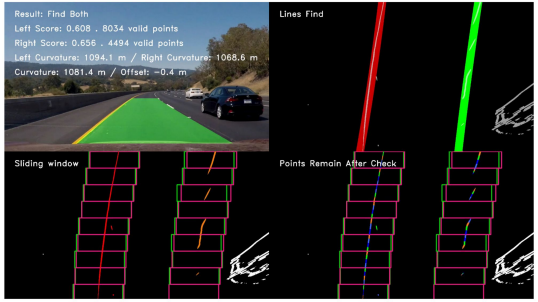
03. Review

review (결과물)

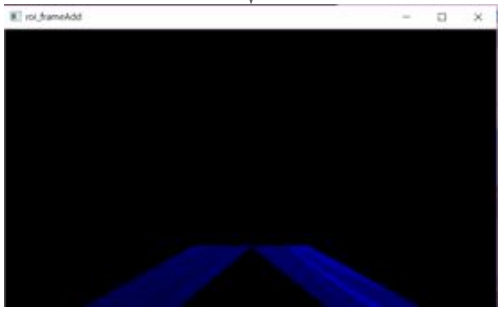


review (차선인식 알고리즘 종류)

sliding window

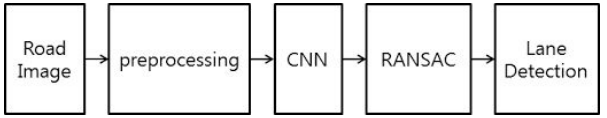


hough transform



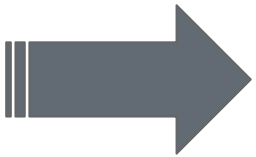
V - ROI

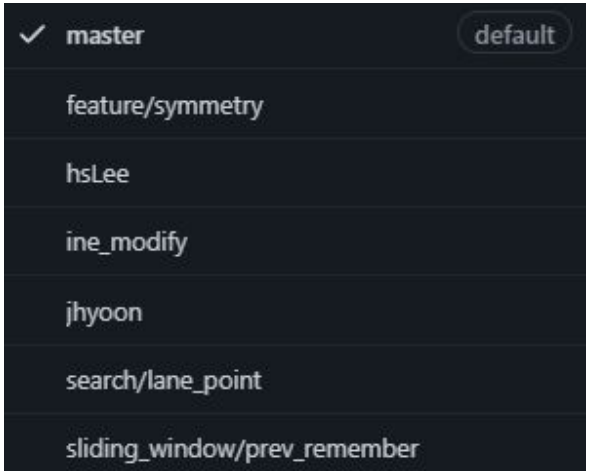
deep learning (CNN)



review (협업)

기능마다 브랜치 생성





review (차선 탐지)

현재 윈도우 픽셀에서 탐색 후 draw인식

```
leftx_current = np.argmax(histogram[:midpoint])
rightx_current = np.argmax(histogram[midpoint:]) + midpoint

window_height = np.int(lane.shape[0]/nwindows)
```

```
for window in range(nwindows):
    win_yl = lane.shape[0] - (window+1)*window_height
    win_yh = lane.shape[0] - window*window_height

    win_xll = leftx_current - margin
    win_xlh = leftx_current + margin
    win_xrl = rightx_current - margin
    win_xrh = rightx_current + margin

    cv2.rectangle(out_img,(win_xll,win_yl),(win_xlh,win_yh),(0,255,0), 2)
    cv2.rectangle(out_img,(win_xrl,win_yl),(win_xrh,win_yh),(0,255,0), 2)

    good_left_inds = ((nz[0] >= win_yl)&(nz[0] < win_yh)&(nz[1] >= win_xll)&(nz[1] < win_xlh)).nonzero()[0]
    good_right_inds = ((nz[0] >= win_yl)&(nz[0] < win_yh)&(nz[1] >= win_xrl)&(nz[1] < win_xrh)).nonzero()[0]

    left_lane_inds.append(good_left_inds)
    right_lane_inds.append(good_right_inds)

    if len(good_left_inds) > minpixel:
        leftx_current = np.int(np.mean(nz[1][good_left_inds]))
    if len(good_right_inds) > minpixel:
        rightx_current = np.int(np.mean(nz[1][good_right_inds]))

    lx.append(leftx_current)
    ly.append((win_yl + win_yh)/2)
    rx.append(rightx_current)
    ry.append((win_yl + win_yh)/2)
```

주어진 파이썬 코드를 보면
맨 처음 탐색한 결과를 바탕으로 그림을
그리고, 그 window의 위치에서 255 픽셀들의
평균을 구해 다음 window의 시작점을 찾는
과정을 반복한다.

곡선의 변화에 둔감해질 것이라 생각함

review (차선 탐지)

현재 윈도우 픽셀에서 탐색 후 draw인식

원본의 차선을 탐색하는 과정은 해당 박스안에서의 nonzero들의 x좌표 평균을 구해서 다음 윈도우의 중앙점으로 정한다.

기존의 방법

해당 윈도우 위치에서의 offset을 기준으로 nonzero들의 x좌표들의 시작점과 끝점의 평균 구해 x좌표 구하고, 윈도우를 그린다.

저희의 방법

review (차선 탐지)

현재 윈도우 픽셀에서 탐색 후 draw인식

```
int offset = (int)((win_y_high + win_y_low) / 2);
```

```
int ri = 0;
vector<int> rhigh_vector(window_width + 1);
for (auto x = win_x_rightb_left; x < win_x_rightb_right; x++) {
    ri++;
    rhigh_vector[ri] = v_thres.at<uchar>(offset, x);
    if (v_thres.at<uchar>(offset, x) == 255 && rl == 960) {
        rl = x;
        rr = x;
    }
    if (v_thres.at<uchar>(offset, x) == 255 && lr != 960) {
        rr = x;
    }
}
```

```
rectangle(roi, Rect(win_x_leftb_left, win_y_high, window_width, window_height), Scalar(0, 150, 0), 2);
rectangle(roi, Rect(win_x_rightb_left, win_y_high, window_width, window_height), Scalar(150, 0, 0), 2);
```

review (대칭)

탐지 안될 경우 대칭으로 차선 예측

```
if (lnonzero < pixel_thres && rnonzero > pixel_thres) {
    left_start = lane_mid - right_diff;
    lane_mid = right_start - right_diff;
}
else if (lnonzero > pixel_thres && rnonzero < pixel_thres) {
    right_start = lane_mid + left_diff;
    lane_mid = left_start + left_diff;
}
```

offset에서의 nonzero가 임계값을 넘지 못할 경우 탐지한 차선을 대칭시켜 예측한다.

탐지 안될 경우 지난 프레임 결과값을 통해 예측

```
if (lnonzero < pixel_thres && rnonzero > pixel_thres) {
    left_start = lpoints[window].x;
    lane_mid = (right_start + left_start) / 2;
}
else if (lnonzero > pixel_thres && rnonzero < pixel_thres && rpoints[window].x != 0) {
    right_start = rpoints[window].x;
    lane_mid = (right_start + left_start) / 2;
}
```

offset에서 nonzero가 임계값을 넘지 못할 경우 지난 프레임에서의 결과값을 다시 사용한다.

review (점 원근 변환)

ROI 좌표 -> mat_tosrc -> 원본 frame 좌표

```
vector<Point> matrix_oper(Mat frame, Mat per_mat_tosrc, int lx1, int ly1, int lx2, int ly2, int rx1, int ry1, int rx2, int ry2) {
    vector<Point> warp_left_line, warp_right_line;

    int new_lx1, new_ly1, new_lx2, new_ly2;
    new_lx1 = (per_mat_tosrc.at<double>(0, 0) * lx1 + per_mat_tosrc.at<double>(0, 1) * ly1 + per_mat_tosrc.at<double>(0, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * lx1 + per_mat_tosrc.at<double>(2, 1) * ly1 + per_mat_tosrc.at<double>(2, 2));

    new_ly1 = (per_mat_tosrc.at<double>(1, 0) * lx1 + per_mat_tosrc.at<double>(1, 1) * ly1 + per_mat_tosrc.at<double>(1, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * lx1 + per_mat_tosrc.at<double>(2, 1) * ly1 + per_mat_tosrc.at<double>(2, 2));

    new_lx2 = (per_mat_tosrc.at<double>(0, 0) * lx2 + per_mat_tosrc.at<double>(0, 1) * ly2 + per_mat_tosrc.at<double>(0, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * lx2 + per_mat_tosrc.at<double>(2, 1) * ly2 + per_mat_tosrc.at<double>(2, 2));

    new_ly2 = (per_mat_tosrc.at<double>(1, 0) * lx2 + per_mat_tosrc.at<double>(1, 1) * ly2 + per_mat_tosrc.at<double>(1, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * lx2 + per_mat_tosrc.at<double>(2, 1) * ly2 + per_mat_tosrc.at<double>(2, 2));

    int new_rx1, new_ry1, new_rx2, new_ry2;
    new_rx1 = (per_mat_tosrc.at<double>(0, 0) * rx1 + per_mat_tosrc.at<double>(0, 1) * ry1 + per_mat_tosrc.at<double>(0, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * rx1 + per_mat_tosrc.at<double>(2, 1) * ry1 + per_mat_tosrc.at<double>(2, 2));

    new_ry1 = (per_mat_tosrc.at<double>(1, 0) * rx1 + per_mat_tosrc.at<double>(1, 1) * ry1 + per_mat_tosrc.at<double>(1, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * rx1 + per_mat_tosrc.at<double>(2, 1) * ry1 + per_mat_tosrc.at<double>(2, 2));

    new_rx2 = (per_mat_tosrc.at<double>(0, 0) * rx2 + per_mat_tosrc.at<double>(0, 1) * ry2 + per_mat_tosrc.at<double>(0, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * rx2 + per_mat_tosrc.at<double>(2, 1) * ry2 + per_mat_tosrc.at<double>(2, 2));

    new_ry2 = (per_mat_tosrc.at<double>(1, 0) * rx2 + per_mat_tosrc.at<double>(1, 1) * ry2 + per_mat_tosrc.at<double>(1, 2)) /
        (per_mat_tosrc.at<double>(2, 0) * rx2 + per_mat_tosrc.at<double>(2, 1) * ry2 + per_mat_tosrc.at<double>(2, 2));
```

$$\texttt{dst}(x, y) = \texttt{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

review (점 원근 변환)

ROI 좌표 -> mat_tosrc -> 원본 frame 좌표

```
int offset = 400;
int lpos = int((offset - warp_left_line[0].y) * ((warp_left_line[1].x - warp_left_line[0].x) /
               (warp_left_line[1].y - warp_left_line[0].y)) + warp_left_line[0].x);

int rpos = int((offset - warp_right_line[0].y) * ((warp_right_line[1].x - warp_right_line[0].x) /
               (warp_right_line[1].y - warp_right_line[0].y)) + warp_right_line[0].x);
vector<Point> pos;
pos.push_back(Point(lpos, rpos));

return warp_left_line, warp_right_line, pos;
```

- $(x_1, y_1), (x_2, y_2)$ 통과 $(x_1 \neq x_2)$

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

위의 행렬 연산을 통해 얻은 점 좌표 두 개를 통해 직선의 방정식을 세워 offset = 400 에서의 차선을 찾는다.

```
vector<Point> n_window_sliding(int left_start, int right_start, Mat roi, Mat v_thres, int w, int h,
    vector<Point>& lpoints, vector<Point>& rpoints, Mat per_mat_tosrc, Mat frame) {
```

...

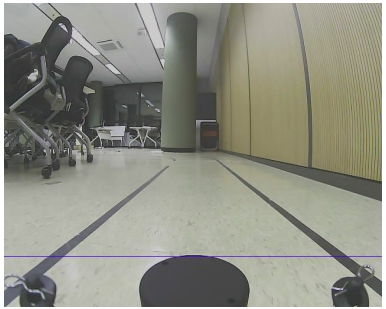
```
vector<Point> warp_left_line(2), warp_right_line(2), pos;
warp_left_line, warp_right_line, pos = matrix_oper(frame, per_mat_tosrc, lx1, ly1, lx2, ly2, rx1, ry1, rx2, ry2);

return warp_left_line, warp_right_line, pos;
```

review (점 원근 변환)

변환은 정확하게 되었으나 직선의 방정식의 문제가 생긴 것으로 보임

review (결과)

직선



O	X
O	X
O	O
O	O
O	O
O	O
O	O
O	O
O	O

정답률

Left : 64.81 %

right : 50.93 %

곡선

O	X
X	X
O	X
O	X
O	X
O	X
O	O
X	O
O	O

직선 구간 : 높은 정답률!

곡선 구간 : 아쉬움

회고

아쉬운 점 🤔

- ❖ 점 원근 변환
- ❖ 탐색한 좌표로 원본 이미지에 초록색 표시
- ❖ 가상 (확장)스크린
- ❖ 새로 찾은 차선 위치가 너무 많이 차이날 경우 처리

나아갈 점 😎

- ❖ 곡선 구간 정확도 개선
- ❖ 특정 구간 노이즈 제거


제 행복을 찾아 떠납니다~~~!