[4 일차]

[lane detect : lane_detector.cpp + lane_detect.mp4]

<code>

```
// vector 안에 vector.. 3 차원이라보면됨. tensor 같은거.
//template code 오타나기쉬움.
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <string>
#include <vector>
//번거로워도 cv::붙여주는이유가 namespace cv; namespace sd;안하기때문. dsp 라이브러리랑 충돌나는게있어서 이렇게 함.

class LaneDetector {
private:
    double img_size;
    double img_center;
    bool left_flag = false;
    bool right_flag = false;
    cv::Point right_b;
    double right_m;          // y = m*x + b
    cv::Point left_b;
    double left_m;

public:
    cv::Mat deNoise(cv::Mat inputImage);
    cv::Mat edgeDetector(cv::Mat img_noise);
    cv::Mat mask(cv::Mat img_edges);
    std::vector<cv::Vec4i> houghLines(cv::Mat img_mask);
    std::vector<std::vector<cv::Vec4i> > lineSeparation(
        std::vector<cv::Vec4i> lines, cv::Mat img_edges);
    std::vector<cv::Point> regression(
        std::vector<std::vector<cv::Vec4i> > left_right_lines,
        cv::Mat inputImage);
    std::string predictTurn();
    int plotLane(cv::Mat inputImage,
        std::vector<cv::Point> lane,
        std::string turn);
};

cv::Mat LaneDetector::deNoise(cv::Mat inputImage) {
        cv::Mat output;

        cv::GaussianBlur(inputImage, output, cv::Size(3, 3), 0, 0);

        return output;
}

cv::Mat LaneDetector::edgeDetector(cv::Mat img_noise) {
        cv::Mat output;
        cv::Mat kernel;
        cv::Point anchor;

        cv::cvtColor(img_noise, output, cv::COLOR_RGB2GRAY);
        cv::threshold(output, output, 140, 255, cv::THRESH_BINARY);

        anchor = cv::Point(-1, -1);
        kernel = cv::Mat(1, 3, CV_32F);
        kernel.at<float>(0, 0) = -1;
        kernel.at<float>(0, 1) = 0;
        kernel.at<float>(0, 2) = 1;

        cv::filter2D(output, output, -1, kernel,
```

```
            anchor, 0, cv::BORDER_DEFAULT);

        return output;
}

cv::Mat LaneDetector::mask(cv::Mat img_edges) {
        cv::Mat output;
        cv::Mat mask = cv::Mat::zeros(img_edges.size(),
                    img_edges.type());

#if 0
        cv::Point pts[4] = {
                cv::Point(210, 720),
                cv::Point(550, 450),
                cv::Point(717, 450),
                cv::Point(1280, 720)
        };
#endif
        // 210 / 1280 ==>> x / 450, 73
        // 550 / 1280 ==>> x / 450, 193
        // 450 / 720  ==>> x / 300, 187
        // 717 / 1280 ==>> x / 450, 252
        cv::Point pts[4] = {
                cv::Point(73, 300),
                cv::Point(193, 187),
                cv::Point(252, 187),
                cv::Point(450, 300)
        };

        cv::fillConvexPoly(mask, pts, 4, cv::Scalar(255, 0, 0));
        cv::bitwise_and(img_edges, mask, output);

        return output;
}

std::vector<cv::Vec4i> LaneDetector::houghLines(cv::Mat img_mask) {
        std::vector<cv::Vec4i> line;

        HoughLinesP(img_mask, line, 1, CV_PI/180, 20, 20, 30);
      // https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html
        return line;
}

std::vector<std::vector<cv::Vec4i> >
LaneDetector::lineSeparation(
    std::vector<cv::Vec4i> lines, cv::Mat img_edges) {
        std::vector<std::vector<cv::Vec4i> > output(2);
        size_t j = 0;
        cv::Point ini;
        cv::Point fini;
        double slope_thresh = 0.3;
        std::vector<double> slopes;
        std::vector<cv::Vec4i> selected_lines;
        std::vector<cv::Vec4i> right_lines, left_lines;

        for (auto i : lines) {
                ini = cv::Point(i[0], i[1]);
                fini = cv::Point(i[2], i[3]);

                double slope =
            (static_cast<double>(fini.y) -
            static_cast<double>(ini.y)) /
            (static_cast<double>(fini.x) -
            static_cast<double>(ini.x) +
            0.00001);

                if (std::abs(slope) > slope_thresh) {
                        slopes.push_back(slope);
```

```
                        selected_lines.push_back(i);
                }
        }

        img_center = static_cast<double>((img_edges.cols / 2));
        while (j < selected_lines.size()) {
                ini = cv::Point(selected_lines[j][0],
                selected_lines[j][1]);
                fini = cv::Point(selected_lines[j][2],
                selected_lines[j][3]);

                if (slopes[j] > 0 &&
        fini.x > img_center &&
        ini.x > img_center) {
                        right_lines.push_back(selected_lines[j]);
                        right_flag = true;
                } else if (slopes[j] < 0 &&
        fini.x < img_center &&
        ini.x < img_center) {
                        left_lines.push_back(selected_lines[j]);
                        left_flag = true;
                }
                j++;
        }

        output[0] = right_lines;
        output[1] = left_lines;

        return output;
}

std::vector<cv::Point> LaneDetector::regression(
    std::vector<std::vector<cv::Vec4i> > left_right_lines,
    cv::Mat inputImage) {
        std::vector<cv::Point> output(4);
        cv::Point ini;
        cv::Point fini;
        cv::Point ini2;
        cv::Point fini2;
        cv::Vec4d right_line;
        cv::Vec4d left_line;
        std::vector<cv::Point> right_pts;
        std::vector<cv::Point> left_pts;

        if (right_flag == true) {
                for (auto i : left_right_lines[0]) {
                        ini = cv::Point(i[0], i[1]);
                        fini = cv::Point(i[2], i[3]);

                        right_pts.push_back(ini);
                        right_pts.push_back(fini);
                }

                if (right_pts.size() > 0) {
                        cv::fitLine(right_pts, right_line,
                    CV_DIST_L2, 0, 0.01, 0.01);
                        right_m = right_line[1] / right_line[0];
                        right_b = cv::Point(right_line[2],
                        right_line[3]);
                }
        }

        if (left_flag == true) {
                for (auto j : left_right_lines[1]) {
                        ini2 = cv::Point(j[0], j[1]);
                        fini2 = cv::Point(j[2], j[3]);

                        left_pts.push_back(ini2);
```

```
                        left_pts.push_back(fini2);
                }

                if (left_pts.size() > 0) {
                        cv::fitLine(left_pts, left_line,
                CV_DIST_L2, 0, 0.01, 0.01);
                        left_m = left_line[1] / left_line[0];
                        left_b = cv::Point(left_line[2], left_line[3]);
                }
        }

        int ini_y = inputImage.rows;
        int fin_y = 165;

        double right_ini_x = ((ini_y - right_b.y) / right_m) +
                right_b.x;
        double right_fin_x = ((fin_y - right_b.y) / right_m) +
                right_b.x;

        double left_ini_x = ((ini_y - left_b.y) / left_m) + left_b.x;
        double left_fin_x = ((fin_y - left_b.y) / left_m) + left_b.x;

        output[0] = cv::Point(right_ini_x, ini_y);
        output[1] = cv::Point(right_fin_x, fin_y);
        output[2] = cv::Point(left_ini_x, ini_y);
        output[3] = cv::Point(left_fin_x, fin_y);

        return output;
}

std::string LaneDetector::predictTurn() {
        std::string output;
        double vanish_x;
        double thr_vp = 10;

        vanish_x = static_cast<double>(((right_m*right_b.x) -
    (left_m*left_b.x) - right_b.y + left_b.y) / (right_m - left_m));

        if (vanish_x < (img_center - thr_vp))
                output = "Left Turn";
        else if (vanish_x > (img_center + thr_vp))
                output = "Right Turn";
        else if (vanish_x >= (img_center - thr_vp) &&
        vanish_x <= (img_center + thr_vp))
                output = "Straight";

        return output;
}

int LaneDetector::plotLane(cv::Mat inputImage,
    std::vector<cv::Point> lane, std::string turn) {
        std::vector<cv::Point> poly_points;
        cv::Mat output;

        inputImage.copyTo(output);
        poly_points.push_back(lane[2]);
        poly_points.push_back(lane[0]);
        poly_points.push_back(lane[1]);
        poly_points.push_back(lane[3]);
        cv::fillConvexPoly(output, poly_points,
            cv::Scalar(0, 0, 255), CV_AA, 0);
        cv::addWeighted(output, 0.3, inputImage,
            1.0 - 0.3, 0, inputImage);

        cv::line(inputImage, lane[0], lane[1],
        cv::Scalar(0, 255, 255), 5, CV_AA);
        cv::line(inputImage, lane[2], lane[3],
        cv::Scalar(0, 255, 255), 5, CV_AA);
```

```
        cv::putText(inputImage, turn, cv::Point(50, 90),
    cv::FONT_HERSHEY_COMPLEX_SMALL, 3,
    cv::Scalar(0, 255, 0), 1, CV_AA);

        cv::namedWindow("Lane", CV_WINDOW_AUTOSIZE);
        cv::imshow("Lane", inputImage);
        return 0;
}

int main(int argc, char *argv[]) {
        if (argc != 2) {
                std::cout << "Not enough parameters" << std::endl;
                return -1;
        }

        std::string source = argv[1];
        cv::VideoCapture cap(source);
        if (!cap.isOpened())
                return -1;

        LaneDetector lanedetector;
        cv::Mat frame;
        cv::Mat img_denoise;
        cv::Mat img_edges;
        cv::Mat img_mask;
        cv::Mat img_lines;
        std::vector<cv::Vec4i> lines;
        std::vector<std::vector<cv::Vec4i> > left_right_lines;
        std::vector<cv::Point> lane;
        std::string turn;
        int flag_plot = -1;
        int i = 0;

        while (i < 540) {
                if (!cap.read(frame))
                        break;

                img_denoise = lanedetector.deNoise(frame);
                img_edges = lanedetector.edgeDetector(img_denoise);
                img_mask = lanedetector.mask(img_edges);
                lines = lanedetector.houghLines(img_mask);

                if (!lines.empty()) {
                        left_right_lines =
            lanedetector.lineSeparation(lines, img_edges);
                        lane = lanedetector.regression(left_right_lines,
                            frame);
                        turn = lanedetector.predictTurn();
                        flag_plot = lanedetector.plotLane(frame, lane,
                                turn);

                        i += 1;
                        cv::waitKey(25);
                } else {
                        flag_plot = -1;
                }
        }
        return flag_plot;
}
```

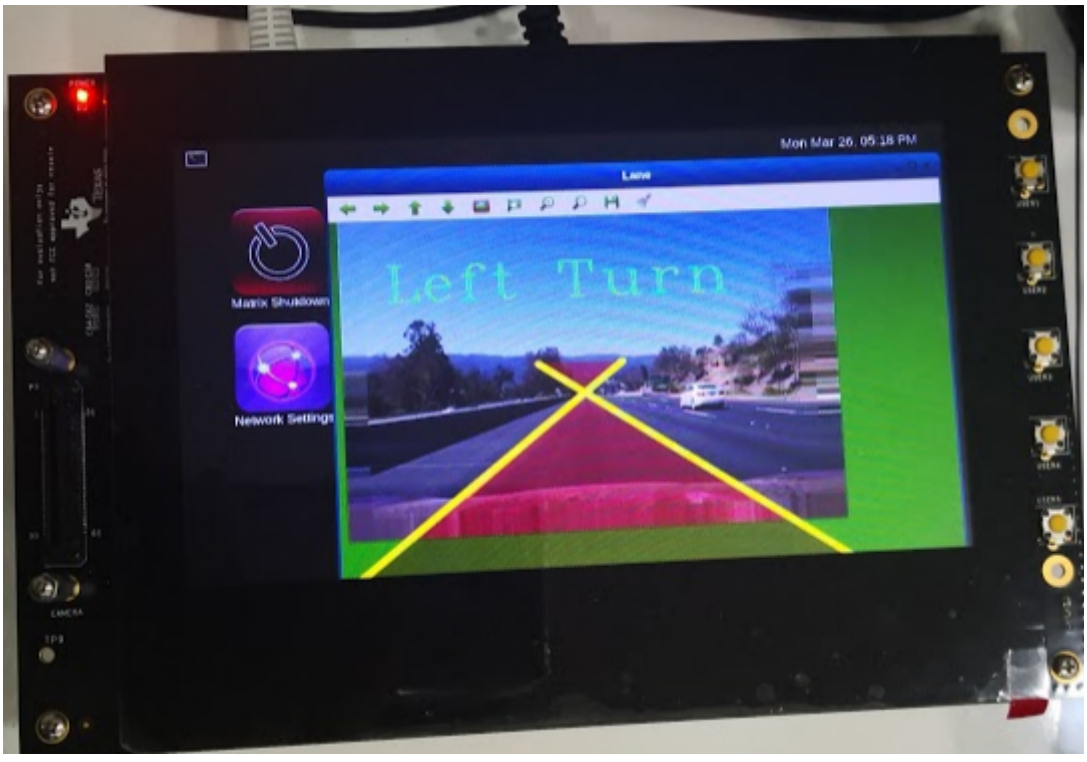
```
root@am57xx-evm:~/gihwahong# g++ -std=c++11 -lOpenCL -locl_util -L/usr/share/OpenCV/3rdparty/lib -lopencv_videostab -lo
pencv_aruco -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_cvv -lopencv_dnn -lopencv_dpm -lopencv_fuzzy
-lopencv_line_descriptor -lopencv_optflow -lopencv_plot -lopencv_reg -lopencv_saliency -lopencv_stereo -lopencv_structu
red_light -lopencv_rgbd -lopencv_surface_matching -lopencv_tracking -lopencv_datasets -lopencv_text -lopencv_face -lope
ncv_xfeatures2d -lopencv_shape -lopencv_video -lopencv_ximgproc -lopencv_calib3d -lopencv_features2d -lopencv_flann -lo
pencv_xobjdetect -lopencv_objdetect -lopencv_ml -lopencv_xphoto -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -l
opencv_photo -lopencv_imgproc -lopencv_core -L$(pwd) -Wl,-rpath=$(pwd) -g -larm_serial lane_detector.cpp
root@am57xx-evm:~/gihwahong# ./a.out
Not enough parameters
root@am57xx-evm:~/gihwahong# ./a.out lane_detect.mp4
init done
Using Wayland-EGL
wlpvr: PVR Services Initialised
```

전에 긴거 컴파일 했던거 가져와서 쓰면됨.

g++ -std=c++11 -lOpenCL -locl_util -L/usr/share/OpenCV/3rdparty/lib -lopencv_videostab -lopencv_aruco -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -lopencv_cvv -lopencv_dnn -lopencv_dpm -lopencv_fuzzy -lopencv_line_descriptor -lopencv_optflow -lopencv_plot -lopencv_reg -lopencv_saliency -lopencv_stereo -lopencv_structured_light -lopencv_rgbd -lopencv_surface_matching -lopencv_tracking -lopencv_datasets -lopencv_text -lopencv_face -lopencv_xfeatures2d -lopencv_shape -lopencv_video -lopencv_ximgproc -lopencv_calib3d -lopencv_features2d -lopencv_flann -lopencv_xobjdetect -lopencv_objdetect -lopencv_ml -lopencv_xphoto -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -lopencv_photo -lopencv_imgproc -lopencv_core -L$(pwd) -Wl,-rpath=$(pwd) -g -larm_serial lane_detector.cpp

./a.out lane_detect.mp4 하면 실행됨.

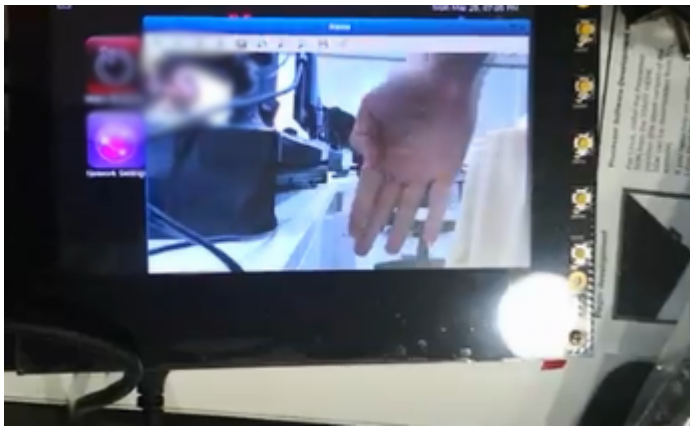
동영상 실행되면서, 라인 잡아 노랗게 그려준다.



[video capture : cam_record.cpp]

<코드>

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main(void)
{
    Mat frame;
    VideoCapture inputVideo(1);

    if(!inputVideo.isOpened())
    {
        cout << "can't open video!" << endl;
        return 0;

    }

    Size size = Size((int)inputVideo.get(CAP_PROP_FRAME_WIDTH),
(int)inputVideo.get(CAP_PROP_FRAME_HEIGHT));

    cout << " size = "<<endl;

    int fourcc = VideoWriter::fourcc('x', 'v', 'i', 'd');
    double fps=30;
    bool isColor = true;
    VideoWriter outputVideo("output.avi", fourcc, fps, size, isColor);

    if(!outputVideo.isOpened())
    {
        cout<< "cant open video!" <<endl;
        return -1;

    }
    if(fourcc != -1)
    {
        imshow("frame", NULL);
        waitKey(100);
    }

    int delay=1000/fps;

    for(;;)
    {
        inputVideo >> frame;
        if(frame.empty())
            break;
        outputVideo << frame;
        imshow("frame", frame);

        int ckey=waitKey(delay);
        if(ckey==27)                //esc
            break;
    }

    return 0;


}
```



[과제 : assignment.avi 로 라인 따라 그리게 코드 수정해보자. lane_detector_assignment.cpp]

<코드>

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <string>
#include <vector>

class LaneDetector {
private:
    double img_size;
    double img_center;
    bool left_flag = false;
    bool right_flag = false;
    cv::Point right_b;
    double right_m;          // y = m*x + b
    cv::Point left_b;
    double left_m;

public:
    cv::Mat deNoise(cv::Mat inputImage);
    cv::Mat edgeDetector(cv::Mat img_noise);
    cv::Mat mask(cv::Mat img_edges);
    std::vector<cv::Vec4i> houghLines(cv::Mat img_mask);
    std::vector<std::vector<cv::Vec4i> > lineSeparation(
        std::vector<cv::Vec4i> lines, cv::Mat img_edges);
    std::vector<cv::Point> regression(
        std::vector<std::vector<cv::Vec4i> > left_right_lines,
        cv::Mat inputImage);
    std::string predictTurn();
    int plotLane(cv::Mat inputImage,
        std::vector<cv::Point> lane,
        std::string turn);
};

cv::Mat LaneDetector::deNoise(cv::Mat inputImage) {
        cv::Mat output;

        cv::GaussianBlur(inputImage, output, cv::Size(9, 9), 0, 0);

        return output;
}

cv::Mat LaneDetector::edgeDetector(cv::Mat img_noise) {
        cv::Mat output;
        cv::Mat kernel;
        cv::Point anchor;

        cv::cvtColor(img_noise, output, cv::COLOR_RGB2GRAY);
        cv::threshold(output, output, 140, 255, cv::THRESH_BINARY);

        anchor = cv::Point(-1, -1);
        kernel = cv::Mat(1, 3, CV_32F);
        kernel.at<float>(0, 0) = -1;
        kernel.at<float>(0, 1) = 0;
        kernel.at<float>(0, 2) = 1;

        cv::filter2D(output, output, -1, kernel,
            anchor, 0, cv::BORDER_DEFAULT);

        return output;
}

cv::Mat LaneDetector::mask(cv::Mat img_edges) {
        cv::Mat output;
```

```
        cv::Mat mask = cv::Mat::zeros(img_edges.size(),
                    img_edges.type());

#if 0
        cv::Point pts[4] = {
                cv::Point(210, 720),
                cv::Point(550, 450),
                cv::Point(717, 450),
                cv::Point(1280, 720)
        };
#endif
        // 210 / 1280 ==>> x / 450, 73
        // 550 / 1280 ==>> x / 450, 193
        // 450 / 720  ==>> x / 300, 187
        // 717 / 1280 ==>> x / 450, 252
        cv::Point pts[4] = {
                cv::Point(20, 240),
                cv::Point(10, 480),
                cv::Point(460, 480),
                cv::Point(460, 240)
        };

        cv::fillConvexPoly(mask, pts, 4, cv::Scalar(255, 0, 0));
        cv::bitwise_and(img_edges, mask, output);

        return output;
}

std::vector<cv::Vec4i> LaneDetector::houghLines(cv::Mat img_mask) {
        std::vector<cv::Vec4i> line;

        HoughLinesP(img_mask, line, 1, CV_PI/180, 40, 30, 20);

        return line;
}

std::vector<std::vector<cv::Vec4i> >
LaneDetector::lineSeparation(
    std::vector<cv::Vec4i> lines, cv::Mat img_edges) {
        std::vector<std::vector<cv::Vec4i> > output(2);
        size_t j = 0;
        cv::Point ini;
        cv::Point fini;
        double slope_thresh = 0.3;
        std::vector<double> slopes;
        std::vector<cv::Vec4i> selected_lines;
        std::vector<cv::Vec4i> right_lines, left_lines;

        for (auto i : lines) {
                ini = cv::Point(i[0], i[1]);
                fini = cv::Point(i[2], i[3]);

                double slope =
            (static_cast<double>(fini.y) -
            static_cast<double>(ini.y)) /
            (static_cast<double>(fini.x) -
            static_cast<double>(ini.x) +
            0.00001);

                if (std::abs(slope) > slope_thresh) {
                        slopes.push_back(slope);
                        selected_lines.push_back(i);
                }
        }

        img_center = static_cast<double>((img_edges.cols / 2));
        while (j < selected_lines.size()) {
                ini = cv::Point(selected_lines[j][0],
```

```
                selected_lines[j][1]);
                fini = cv::Point(selected_lines[j][2],
                selected_lines[j][3]);

                if (slopes[j] > 0 &&
        fini.x > img_center &&
        ini.x > img_center) {
                        right_lines.push_back(selected_lines[j]);
                        right_flag = true;
                } else if (slopes[j] < 0 &&
        fini.x < img_center &&
        ini.x < img_center) {
                        left_lines.push_back(selected_lines[j]);
                        left_flag = true;
                }
                j++;
        }

        output[0] = right_lines;
        output[1] = left_lines;

        return output;
}

std::vector<cv::Point> LaneDetector::regression(
    std::vector<std::vector<cv::Vec4i> > left_right_lines,
    cv::Mat inputImage) {
        std::vector<cv::Point> output(4);
        cv::Point ini;
        cv::Point fini;
        cv::Point ini2;
        cv::Point fini2;
        cv::Vec4d right_line;
        cv::Vec4d left_line;
        std::vector<cv::Point> right_pts;
        std::vector<cv::Point> left_pts;

        if (right_flag == true) {
                for (auto i : left_right_lines[0]) {
                        ini = cv::Point(i[0], i[1]);
                        fini = cv::Point(i[2], i[3]);

                        right_pts.push_back(ini);
                        right_pts.push_back(fini);
                }

                if (right_pts.size() > 0) {
                        cv::fitLine(right_pts, right_line,
                    CV_DIST_L2, 0, 0.01, 0.01);
                        right_m = right_line[1] / right_line[0];
                        right_b = cv::Point(right_line[2],
                        right_line[3]);
                }
        }

        if (left_flag == true) {
                for (auto j : left_right_lines[1]) {
                        ini2 = cv::Point(j[0], j[1]);
                        fini2 = cv::Point(j[2], j[3]);

                        left_pts.push_back(ini2);
                        left_pts.push_back(fini2);
                }

                if (left_pts.size() > 0) {
                        cv::fitLine(left_pts, left_line,
                CV_DIST_L2, 0, 0.01, 0.01);
                        left_m = left_line[1] / left_line[0];
```

```
                        left_b = cv::Point(left_line[2], left_line[3]);
                }
        }

        int ini_y = inputImage.rows;
        int fin_y = 165;

        double right_ini_x = ((ini_y - right_b.y) / right_m) +
                right_b.x;
        double right_fin_x = ((fin_y - right_b.y) / right_m) +
                right_b.x;

        double left_ini_x = ((ini_y - left_b.y) / left_m) + left_b.x;
        double left_fin_x = ((fin_y - left_b.y) / left_m) + left_b.x;

        output[0] = cv::Point(right_ini_x, ini_y);
        output[1] = cv::Point(right_fin_x, fin_y);
        output[2] = cv::Point(left_ini_x, ini_y);
        output[3] = cv::Point(left_fin_x, fin_y);

        return output;
}

std::string LaneDetector::predictTurn() {
        std::string output;
        double vanish_x;
        double thr_vp = 10;

        vanish_x = static_cast<double>(((right_m*right_b.x) -
    (left_m*left_b.x) - right_b.y + left_b.y) / (right_m - left_m));

        if (vanish_x < (img_center - thr_vp))
                output = "Left Turn";
        else if (vanish_x > (img_center + thr_vp))
                output = "Right Turn";
        else if (vanish_x >= (img_center - thr_vp) &&
        vanish_x <= (img_center + thr_vp))
                output = "Straight";

        return output;
}

int LaneDetector::plotLane(cv::Mat inputImage,
    std::vector<cv::Point> lane, std::string turn) {
        std::vector<cv::Point> poly_points;
        cv::Mat output;

        inputImage.copyTo(output);
        poly_points.push_back(lane[2]);
        poly_points.push_back(lane[0]);
        poly_points.push_back(lane[1]);
        poly_points.push_back(lane[3]);
        cv::fillConvexPoly(output, poly_points,
            cv::Scalar(0, 0, 255), CV_AA, 0);
        cv::addWeighted(output, 0.3, inputImage,
            1.0 - 0.3, 0, inputImage);

        cv::line(inputImage, lane[0], lane[1],
        cv::Scalar(0, 255, 255), 5, CV_AA);
        cv::line(inputImage, lane[2], lane[3],
        cv::Scalar(0, 255, 255), 5, CV_AA);

        cv::putText(inputImage, turn, cv::Point(50, 90),
    cv::FONT_HERSHEY_COMPLEX_SMALL, 3,
    cv::Scalar(0, 255, 0), 1, CV_AA);

        cv::namedWindow("Lane", CV_WINDOW_AUTOSIZE);
        cv::imshow("Lane", inputImage);
```

```
        return 0;
}

int main(int argc, char *argv[]) {
        if (argc != 2) {
                std::cout << "Not enough parameters" << std::endl;
                return -1;
        }

        std::string source = argv[1];
        cv::VideoCapture cap(source);
        if (!cap.isOpened())
                return -1;

        LaneDetector lanedetector;
        cv::Mat frame;
        cv::Mat img_denoise;
        cv::Mat img_edges;
        cv::Mat img_mask;
        cv::Mat img_lines;
        std::vector<cv::Vec4i> lines;
        std::vector<std::vector<cv::Vec4i> > left_right_lines;
        std::vector<cv::Point> lane;
        std::string turn;
        int flag_plot = -1;
        int i = 0;

        while (i < 540) {
                if (!cap.read(frame))
                        break;

                img_denoise = lanedetector.deNoise(frame);
                img_edges = lanedetector.edgeDetector(img_denoise);
                img_mask = lanedetector.mask(img_edges);
                lines = lanedetector.houghLines(img_mask);

                if (!lines.empty()) {
                        left_right_lines =
            lanedetector.lineSeparation(lines, img_edges);
                        lane = lanedetector.regression(left_right_lines,
                            frame);
                        turn = lanedetector.predictTurn();
                        flag_plot = lanedetector.plotLane(frame, lane,
                                turn);

                        i += 1;
                        cv::waitKey(25);
                } else {
                        flag_plot = -1;
                }
        }
        return flag_plot;
}
```

```
root@am57xx-evm:~/gihwahong# g++ -std=c++11 -lOpenCL -locl_util -L/usr/share/OpenCV/3rdpart
y/lib -lopencv_videostab -lopencv_aruco -lopencv_bgsegm -lopencv_bioinspired -lopencv_ccali
b -lopencv_cvv -lopencv_dnn -lopencv_dpm -lopencv_fuzzy -lopencv_line_descriptor -lopencv_o
ptflow -lopencv_plot -lopencv_reg -lopencv_saliency -lopencv_stereo -lopencv_structured_lig
ht -lopencv_rgbd -lopencv_surface_matching -lopencv_tracking -lopencv_datasets -lopencv_tex
t -lopencv_face -lopencv_xfeatures2d -lopencv_shape -lopencv_video -lopencv_ximgproc -lopen
cv_calib3d -lopencv_features2d -lopencv_flann -lopencv_xobjdetect -lopencv_objdetect -lopen
cv_ml -lopencv_xphoto -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -lopencv_photo -
lopencv_imgproc -lopencv_core -L$(pwd) -Wl,-rpath=$(pwd) -g -larm_serial lane_detector_assi
gnment.cpp
root@am57xx-evm:~/gihwahong# ./a.out assignment.avi
init done
Using Wayland-EGL
wlpvr: PVR Services Initialised
root@am57xx-evm:~/gihwahong#
```

동영상소스 : 680x480 이라

관심영역 설정하고, + blur 설정 + houghlines 설정 + threshold 는 140 그대로. 아래는 그 결과값.