OpenCV 8차시

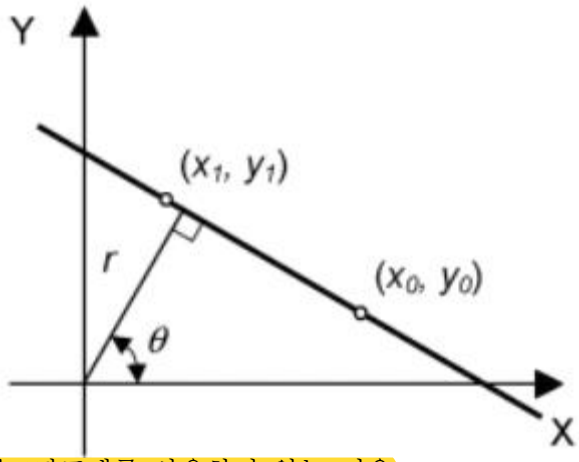
ㅇ Hough transform

직선검출은 영상 내에서 공간 구조를 분석하는데 유용
예를 들어,
도로 주행영상에서 차선 검출하는데 이용
또는 명함 스캔 프로그램에서 명함의 윤곽선을 검출하는 데 이용

이러한 직선검출 방법 중에서 가장 널리 사용되는 것이 'Hough transform'

허프변환이란, 직교 좌표계로 표현되는 영상의 에지 점들을 극 좌표계로 옮겨서 검출하고자 하는 물체의 파라미터 (rho, theta)를 추출

y = a x + b (직교 좌표계) <======> rho = x cos theta + y sin theta (극 좌표계)



직교좌표계를 사용하지 않는 이유:
검출 직선의 기울기와 절편을 구해야 하는 데, 수직선이나, 수평선의 경우, y 혹은 x좌표가 사라지기 때문이다.

극좌표계에서는 (x,y) 좌표에 대해서 상응하는 rho와 theta가 늘 존재한다.

Opencv에서 지원하는 함수 Houghlines( )를 사용하면 직선들의 방정식을 얻을 수 있다. 그런데, 이 방정식들은 rho와 theta값들로 주어진다. 원점으로부터 직선까지의 직교거리 rho, 직교선이 x축과 이루는 각도 theta이다.

rho와 theta를 알 때, 이를 직선으로 그리는 방법은 다음과 같다.
(1) 직교선과 직선이 만나는 좌표 (x0, y0)는 다음과 같이 구할 수 있다.
x0 = rho * cos (theta)
y0 = rho * sin (theta)

(2) (x0, y0)는 이미지의 한 가운데 있을 수 있으므로, 이미지의 경계 밖의 두 점을 찾아서 직선으로 이어주면 된다.
우선 이미지 밖에 있는 점 A의 좌표를 구해보면,
xa = x0 - 1000 * sin (theta)
ya = y0 + 1000 * cos (theta)
(xa, ya)는 다음 방적식을 만족시킨다.
xa * cos (theta) + ya * sin(theta) = rho
따라서
(x0 - 1000 * sin(theta)) * cos (theta) + (y0 + 1000 * cos(theta)) * sin(theta) = rho.

여기에서 - 1000 * sin(theta)와 1000 * cos (theta)가 어떻게 나왔는지에 대해서 이해해 보자.
(x0, y0)는 이미지 영역내의 점일 것이다.
이 때 이미지 영역 바깥의 두 점을 찾아서 직선을 그어야, 이미지를 관통하는 직선을 그릴 수 있다.
x0로부터 멀리 떨어진 점을 찾아야 하는데,
일단 x0 - 1000 이라고하면, x0에서 1000 픽셀정도 떨어질 것이다.
이제 x * cos (theta) + y * sin (theta) = rho 를 만족시켜야 하므로,
x0 - 1000 * sin(theta) 를 새로운 x로 하고,
y0 + 1000 * cos(theta)를 새로운 y로 하면, 위 수식을 만족시키는 (x, y)값이 나올 것이다.

이것은 하나의 점이므로, 다른 한 점의 좌표도 1000 픽셀 정도 떨어진 위치에서 잡으면 되는데, 이때는 1000 앞에 붙는 부호를 바꾸면 된다.
그러면,
x0 + 1000 * sin (theta) * cos(theta),
y0 - 1000 * cos (theta) * sin(theta)가 될 것이다.

이렇게 이미지 영역 밖의 두 점을 잡으면,
이 두 점은 서로 정반대 쪽에 있는 두 점이 될 것이고,
이를 기준으로 직선을 그으면 이미지를 관통하는 직선을 얻을 수 있다.

```
void cv::HoughLines ( InputArray   image,   // 직선 찾을 이미지 => Canny edge로 넘겨준다.
                      OutputArray  lines,   // rho, theta 으로 나온다.
                      double       rho,
                      double       theta,   // 몇도씩 올릴꺼니?
                      int          threshold, // 점 몇개가 만나는지의 정도 (여러개일때 직선이라고 판단단한)
                      double       srn = 0,
                      double       stn = 0,
                      double       min_theta = 0,
                      double       max_theta = CV_PI
                    )
```

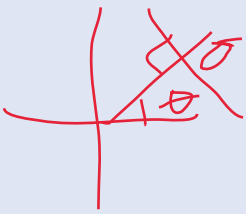


image--> edge detection이 된 binary image
lines --> vector이고 각 요소는 2개의 floating number로 구성된다. 첫 번째 값이 rho이고, 두 번째 값이 theta값이다.
rho --> rho값의 resolution이다. increment를 얼마씩 할 것인가를 결정한다.
theta --> theta값의 resolution이다. increment를 얼마씩 할 것인가를 결정한다.
threshold --> rho와 theta space에서 각 (rho, theta) 영역에 매칭되는 빈도가 기록되는데, 그 빈도가 threshold 이상 인 것만 발견된 직선으로 한다.

OpenCV의 HoughLines가 동작하는 방식

Hough transform은 n개의 2차원 좌표들이 주어졌을 때, 이 좌표들을 지나는 직선의 방정식을 구하는 방법이다. 이것은 직선의 방정식을 (x, y) 좌표계에서 (theta, p) 좌표계로 변환시키는 것을 말한다.

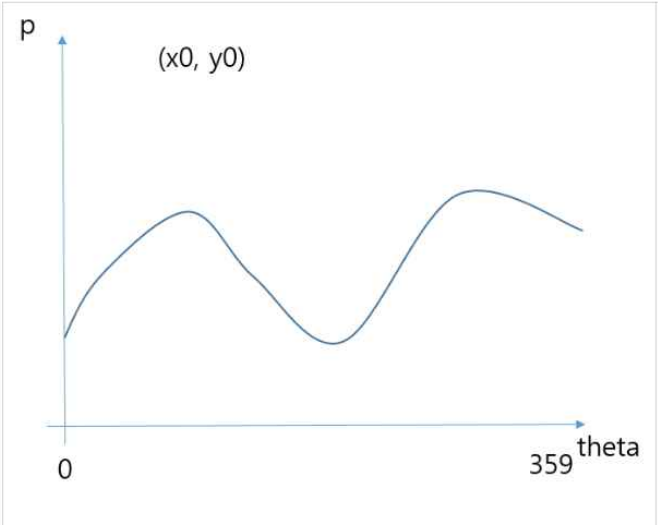
직선을 나타내기 위해서 Hesse normal form을 이용한다.

직선 L이 있고, 원점으로부터 이 직선까지의 직교선의 길이가 p, 직교선과 x축이 이루는 각도를 theta라고 하자.
그러면, L의 방정식은
x cos theta + y sin theta = p 된다.

예를 들어, 3개의 2차원 좌표 A, B, C가 주어졌다고 하고, 각각의 좌표는 다음과 같다.
A = (x0, y0), B = (x1, y1), C = (x2, y2)
이제 이 세 점을 지나는 직선의 방정식을 구한다고 하자.

theta를 0에서 359도까지 1도씩 증가시켜가면서,
A = (x0, y0)에 대해서, p를 다음 식을 이용해서 구한다.

x0 cos theta + y0 sin theta = p
이렇게 해서 구해진 (theta, p)를 theta와 p의 좌표축 위에 그려보자.
그러면 다음과 같이 하나의 선이 나올 것이다.



x0 cos theta + y0 sin theta = p의 그래프.

theta의 값은 0에서 1도씩 증가시켜나간다.
x0, y0는 fixed된 값이다.
p는 단순 계산값이다.
따라서, theta에 의해 p가 결정된다.

Hough transform은 n개의 2차원 좌표들이 주어졌을 때, 이 좌표들을 지나는 직선의 방정식을 구하는 방법니다.
이것은 직선의 방정식을 (x, y) 좌표계에서 (theta, p) 좌표계로 변환시키는 것을 말한다.

직선을 나타내기 위해서 Hesse normal form을 이용한다.

직선 L이 있고, 원점으로부터 이 직선까지의 직교선의 길이가 p, 직교선과 x축이 이루는 각도를 theta라고 하자.
그러면, L의 방정식은
x cos theta + y sin theta = p 된다.
(왜 이렇게 되는지에 대해서는 다음 노트를 참조한다.)

예를 들어, 3개의 2차원 좌표 A, B, C가 주어졌다고 하고, 각각의 좌표는 다음과 같다.
A = (x0, y0), B = (x1, y1), C = (x2, y2)
이제 이 세 점을 지나는 직선의 방정식을 구한다고 하자.

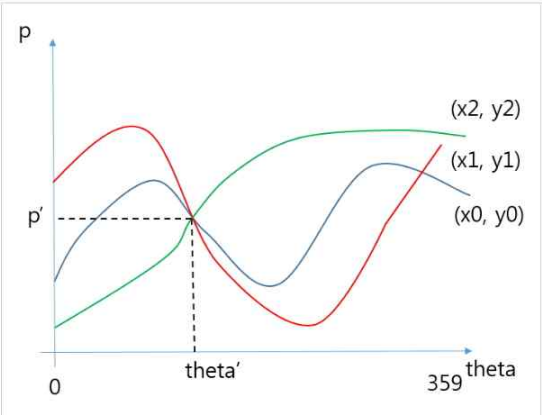
theta를 0에서 359도까지 1도씩 증가시켜가면서,
A = (x0, y0)에 대해서, p를 다음을 이용해서 구해보자.
x0 cos theta + y0 sin theta = p
이렇게 해서 구해진 (theta, p)를 theta와 p의 좌표축 위에 그려보자.
그러면 다음과 같이 하나의 선이 나올 것이다.

x0,y0를 지나는 직선?
x0 cos theta + y0 sin theta = p의 그래프.

(x1,y1)
(x2,y2)
:
하나씩 다 넣어보고 Theta와 rho 값을 구한다.

theta의 값은 0에서 1도씩 증가시켜나간다.
x0, y0는 fixed된 값이다.
p는 단순 계산값이다.
따라서, theta에 의해 p가 결정된다.

이번에는 B = (x1, y1)와 C = (x2, y2)를 이용해서 동일하게 그래프를 그린다.



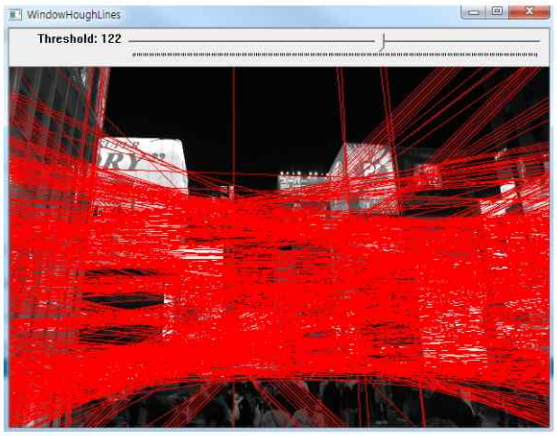
왼쪽은 세 점 (xi, yi) i = 0, 1, 2에 대해서 그래프를 그린 것이다. 세 곡선이 만나는 교점 (theta', p')은 세 점을 지나는 직선의 방정식에 사용되는 theta와 p의 값이 된다.

이런식으로 주어진 n개의 점들에 대해서 그래프를 모두 그린 후, 교차하는 직선들이 많은 점들에 해당하는 (theta, p)값 들을 찾으면, n개의 점들을 지나는 직선의 방정식을 구할 수 있다.

Hough threshold 변화에 따른 검출 직선의 변화

아래 그림들은 HoughLines( )에서 사용되는 threshold의 변화에 따라, 검출되는 직선들이 어떻게 변하는지를 보여주고 있다. threshold는 rho와 theta가 직선으로 인정받기 위해서 매칭되어야 하는 최소 점의 개수이다.

threshold = 173으로, 직선으로 인정받기 위해서 최소 173개의 픽셀이 일렬로 늘어서 있어야 한다. 그러므로 검출되는 직선의 숫자가 몇 개 되지 않는다.



threshold = 122로, 많은 수의 직선들이 검출된 것을 볼 수 있다.

```
6   void draw_houghLines(Mat src, Mat& dst, vector<Vec2f> lines, int nline)
7   {
8       cvtColor(src, dst, CV_GRAY2BGR); // 컬러 선 그리기 위해서
9       for (int i = 0; i < min((int)lines.size(), nline); i++)
10      {                    // 라인갯수랑, 그리고자 하는 선갯수 중 작은애로 선택
11          float rho = lines[i][0];
12          float theta = lines[i][1];
13          double a = cos(theta); // rho * cos(theta) = x좌표
14          double b = sin(theta); // y좌표
15
16          Point2d pt(a*rho, b*rho); // (직선이 지나는 점의 좌표
17          Point2d delta(1000 * -b, 1000 * a);
18          line(dst, pt + delta, pt - delta, Scalar(0, 255, 0), 1, LINE_AA);
19      }
20  }
```

// 이미지 밖에 있는 좌표가 나오게
=> 라인을 그릴 수 있게

* line을 내가 원하는 부분만 그리게 할려면?

```cpp
int main(int argc, char *argv[])
{
    Mat img = imread(argv[1], IMREAD_UNCHANGED);

    if (img.empty() == true)
    {
        cout << "Unable to read image" << endl;
        return -1;
    }
    Mat gray;
    cvtColor(img, gray, CV_BGR2GRAY);

    double rho = 1;
    double theta = CV_PI / 180; //1도씩 증가 pi/180

    Mat canny, dst; // Hough는 반드시 Canny
    GaussianBlur(gray, canny, Size(5, 5), 2, 2); // 노이즈를 제거
    Canny(canny, canny, 100, 150, 3); // binary canny image

    vector<Vec2f> lines; // 찾은 라인을 넘겨주는거
    HoughLines(canny, lines, rho, theta, 50); //최소 50개 지나야 직선으로 판단.
    draw_houghLines(canny, dst, lines, 10);
                    // 바이너리 이미지로

    imshow("org", img);
    imshow("canny", canny);
    imshow("hough", dst);
    waitKey(0);

    return 0;
}
```

실습:

1. HoughLines를 이미지 위에 표시하는 클래스를 만들어 보자
2. Trackbar를 이용해서 threshold를 변화시켜 가면서, 검출되는 line들을 표시해 보자.
3. 블랙박스 컬러영상 위에 HoughLines를 '적색'으로 표시해 보자. 블랙박스 동영상을 play하면서, 실시간으로 Line을 모두 찾아 표시한다.