**Open Software Project**

**Homework7 Technical Report**

**경영학부 1517034 정성원**

**1.**

**A. Code**

-feature matching과 image stitching을 위한 코드는 lab02 와 lab06의 code를 재사용했다.

-RANSAC 코드는 다음과 같이 작성했다.

if (ransac) {

// RANSAC Parameters

int K = 3; // 3;

int S = 50;

double thres = 3.84;

int bestScore = 0;

Mat bestT;

int\* sptl\_x = (int\*)malloc(sizeof(int) \* K);

int\* sptl\_y = (int\*)malloc(sizeof(int) \* K);

int\* sptr\_x = (int\*)malloc(sizeof(int) \* K);

int\* sptr\_y = (int\*)malloc(sizeof(int) \* K);

for (int s = 0; s < S; s++) {

// Randomly sample k data

srand((unsigned int)time(NULL));

int\* picked = (int\*)calloc(num\_points, sizeof(int));

int k = 0;

while (k < K) {

int randn = rand() % num\_points;

if (!picked[randn]) {

sptl\_x[k] = ptl\_x[randn];

sptl\_y[k] = ptl\_y[randn];

sptr\_x[k] = ptr\_x[randn];

sptr\_y[k] = ptr\_y[randn];

picked[randn] = 1;

k ++;

}

}

// Estimate the affine transformation

Mat A12 = cal\_affine<float>(ptl\_x, ptl\_y, ptr\_x, ptr\_y, num\_points);

// Score by computing the number of inliers

int score = 0;

for (int i = 0; i < num\_points; i++) {

float Tpx = A12.at<float>(0) \* dstPoints[i].y + A12.at<float>(1) \* dstPoints[i].x + A12.at<float>(2);

float Tpy = A12.at<float>(3) \* dstPoints[i].y + A12.at<float>(4) \* dstPoints[i].x + A12.at<float>(5);

float dist = sqrt((Tpx - srcPoints[i].y)\*(Tpx - srcPoints[i].y) + (Tpy - srcPoints[i].x)\*(Tpy - srcPoints[i].x));

if (dist < thres)

score++;

}

// Select the best affine transformation

if (score > bestScore) {

bestScore = score;

bestT = A12;

}

}

// Re-estimate the affine transformation by solving Mx = b with Tb's inliers

// Get inliers

int\* iptl\_x = (int\*)malloc(sizeof(int)\*bestScore);

int\* iptl\_y = (int\*)malloc(sizeof(int)\*bestScore);

int\* iptr\_x = (int\*)malloc(sizeof(int)\*bestScore);

int\* iptr\_y = (int\*)malloc(sizeof(int)\*bestScore);

int j = 0;

std::cout << "BestT: " << bestT << endl;

std::cout << "BestScore: " << bestScore << endl;

for(int i = 0; i < num\_points; i++) {

float Tpx = bestT.at<float>(0) \* dstPoints[i].y + bestT.at<float>(1) \* dstPoints[i].x + bestT.at<float>(2);

float Tpy = bestT.at<float>(3) \* dstPoints[i].y + bestT.at<float>(4) \* dstPoints[i].x + bestT.at<float>(5);

float dist = sqrt((Tpx - srcPoints[i].y)\*(Tpx - srcPoints[i].y) + (Tpy - srcPoints[i].x)\*(Tpy - srcPoints[i].x));

if (dist < thres) {

iptl\_x[j] = dstPoints[i].y;

iptl\_y[j] = dstPoints[i].x;

iptr\_x[j] = srcPoints[i].y;

iptr\_y[j] = srcPoints[i].x;

j++;

}

}

// Re-estimate

//cout << "A12" << A12 << endl;

//cout << "A21" << A21 << endl;

A12 = cal\_affine<float>(iptl\_x, iptl\_y, iptr\_x, iptr\_y, bestScore);

A21 = cal\_affine<float>(iptr\_x, iptr\_y, iptl\_x, iptl\_y, bestScore);

//cout << "A12" << A12 << endl;

//cout << "A21" << A21 << endl;

}

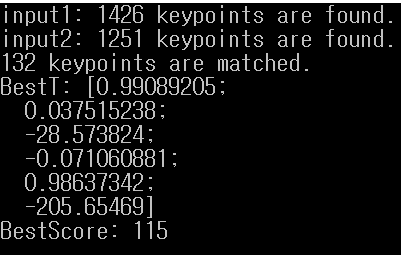
**B. Result**

**CASE 1. (Not using RANSAC)**



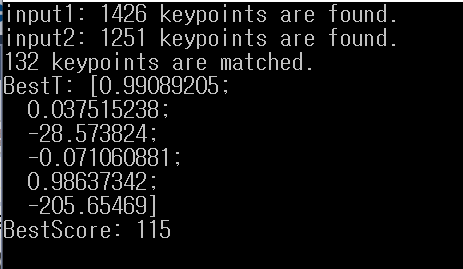
**CASE 2. (Using RANSAC)**

**A. K = 3**





**B. K = 4**





**2.**

**A. Code**

먼저 Canny edge detector를 사용해 이미지의 edge를 검출한다.



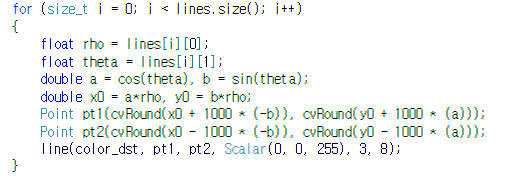
이후 openCV의 houghLines를 이용해 Hough transform을 시행한다. 전달해주는 인자는 순서대로 edge detect의 결과 이미지(dst), detected lines의 시작좌표와 끝좌표를 저장할 벡터(lines), r(rho), 세타(theta), line detect 시 교점의 최소수(threshold), 선을 형성할 점의 최소수(minLinLength), 같은 직선이라고 고려된 두 점의 최대 차이(maxLineGap)이다.



먼저 이미지 상에서의 직선은 y = ax + b 즉, 파라미터 (a, b)로 표현될 수 있다. 이 때 좌표 시스템을 바꾼다면 직선은 극좌표 즉 파라미터 (rho, theta)로 표현할 수 있다. 따라서 직선은 아래와 같이 쓰일 수 있다.

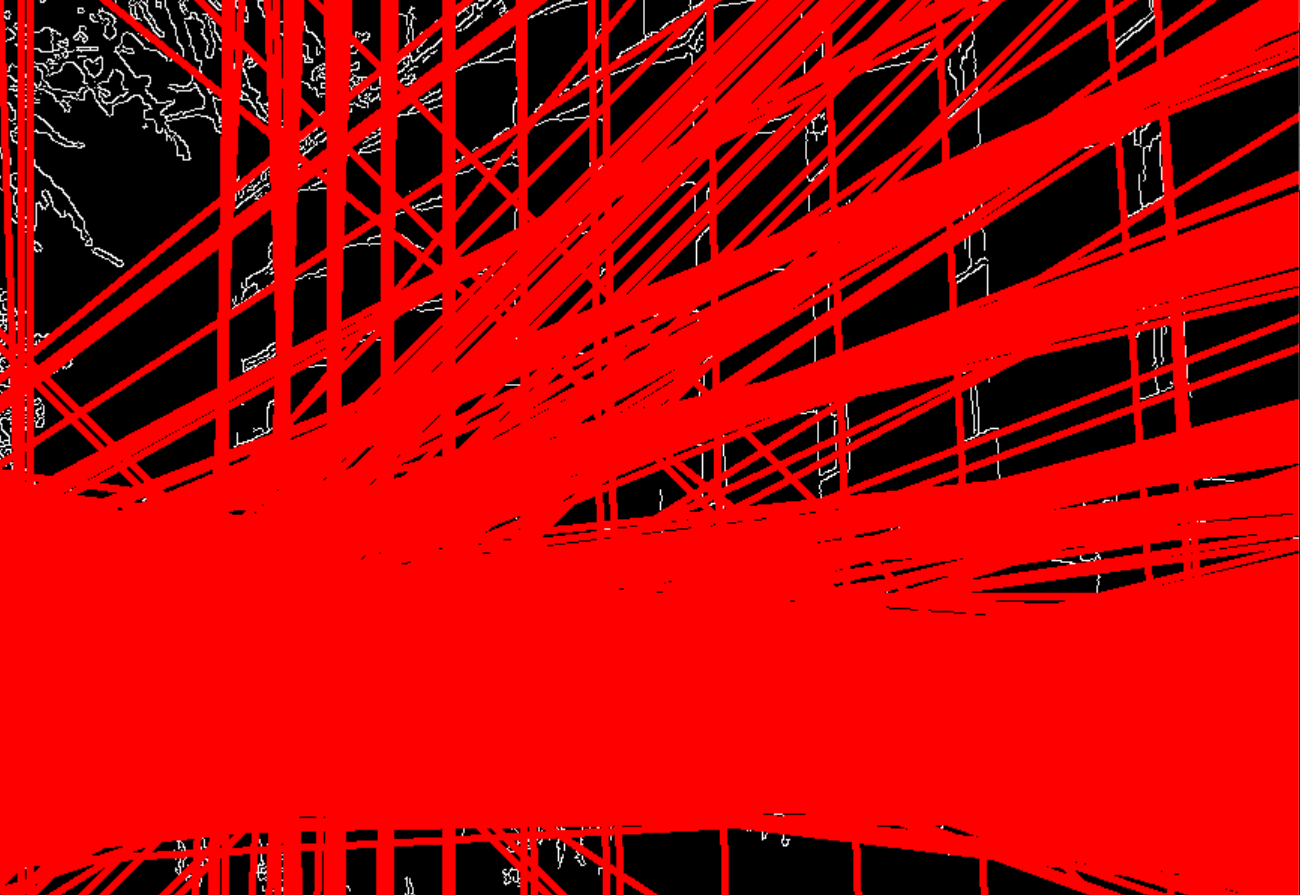


변환된 좌표 체계에서 교집합의 수가 가장 많은 지점을 찾는 것이 Hough Transform이다. Hough Transform을 마친 후 시각화를 위해 line을 그린다.



**B. Result**

**HoughLines**



**HoughLinesP**

