Technical Report

1829008 김민영

1. practice : Image Stitching Using Affine Transform


Compare Case1) and Case2) : RANSAC을 추가한 Case2가 더 성능이 좋은 것을 확인할 수 있다.

Compare k=3 and k=4 : k=4일 때가 k=3일 때 보다 성능이 더 좋은 것을 확인할 수 있다.

Explanation of Codes

Case1) Mx = b (StitchingAffine.cpp)

앞서 lab2(stitching)과, lab8(SIFT)에서 구현했던 코드를 합친 코드이다. SIFT에서 구한 모든 점을 기준으로 stitching 하면 된다.

Main 함수 : siftFeatureDetector, SiftDescriptorExtractor를 만든다. Keypoints 와 descriptor를 출력 화면에 보여준다. Nearest neighbor pairs를 찾고, 그 쌍을 그려 matching image를 출력한다.

euclidDistance 함수 : 유클리디안 거리를 계산하고 이를 반환한다.

nearestNeighbor1, nearestNeighbor2 함수: Find the index of nearest neighbor point from keypoints.

findPairs 함수 : Find pairs of points with the smallest distace between them

cal_affine 함수 : 행렬 연산을 사용하여 affineM을 구한다.

blend_stitching 함수 : Blend two Images

Case2) Mx = b + RANSAC (StitchingAffineRansac.cpp)

위의 Case1) 코드에 RANSAC 부분을 추가하여 구현하면 된다.

랜덤샘플링하여 inlier 개수가 많은 best affine transformation matrix 선택하는 것이다.

추가되는부분 : main 함수, cal_affine 함수 부분

1. k개의 data를 randomly sample

2. (Mx=b 수행)

3. |Tp-p'|^2 < sigma^2에 대해 inliers를 counting (이상치 제거)

4. best affine transfromation Tb를 뽑는다(T1~Ts)

5. re-estimate the affine transformation by solving Mx = b with Tb's inliers.

Analysis

다양한 유형의 outlier가 포함되어 있을 수 있기 때문에 모든 점을 매칭할 때에는 잘못된 대응점들을 매칭할 수 있는 경우가 발생한다. 따라서 더 정확하게 하기 위해 RANSAC 알고리즘을 사용하여 inlier를 뽑으면 더 좋은 성능을 얻을 수 있다.

(전체적인 과정)

1. Feature matching

 1) Run SIFT descriptor for two images

 2) Perform the feature matching using NN

 3) Refine feature matching using cross-checking & ratio-thresholding

2. Affine transform estimation

3. perform image stitching

Practice2 : Line Fitting using Hough Transform

Explain how to estimate the line segments in the Hough transform

representation of Lines in the Hough Space

$y = ax + b$

$\Rightarrow r = x\cos\theta + y\sin\theta \Leftrightarrow y = -\frac{\cos\theta}{\sin\theta}x + \frac{r}{\sin\theta}$

1. edge detection
2. mapping of edge points to the Hough Space and storage in accumulator
3. interpretation of the accumulator to yield lines of th

-> 몇 개의 data set을 r=xcos세타 + ysin세타 에 대입하여 가장 빈번하게 나오는 (r,세타)를 찾음

create a copy image2 of the input edge image1

if(image2 = empty) ···①

finish

x_{start} x_{end} y_{start} y_{end}

update the accumulator with randomly selected pixel from image2

remove the pixel from image2

if (bin with the largest value in the accumulator < threshold)

goto ①

search in image1 along a corridor specified by BINX, and find the longest segment of pixels either continuous or exhibiting gaps not exceeding a given threshold

remove the pixels in the segment from image2

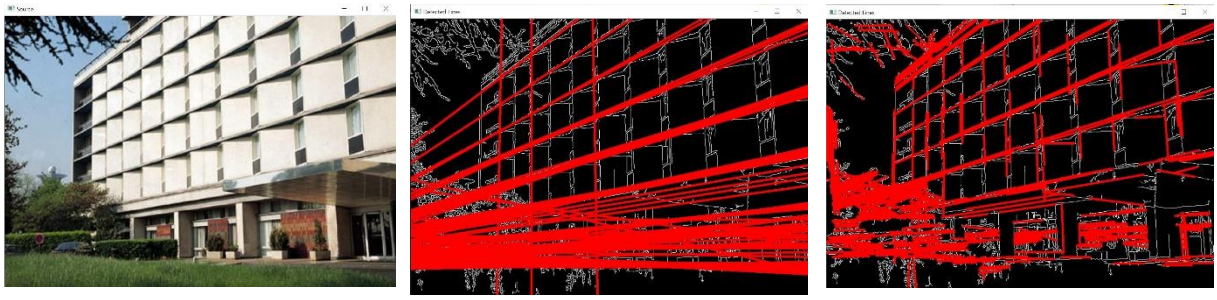
clear BINX

if (detected line segment > minLineLength)

add outputlist

return outputlist.start , outputlist.end

Practice3 : Line Fitting using Hough Transform



Explain the Codes (HoughTransform.cpp)

1. Canny edge detector 실행

2. HoughLines 실행 , HoughLinesP 실행

3. draw the line fitting result

Analysis

HoughLines를 사용한 결과보다 HoughLinesP를 사용한 결과가 더 성능이 좋은 것을 확인할 수 있다.