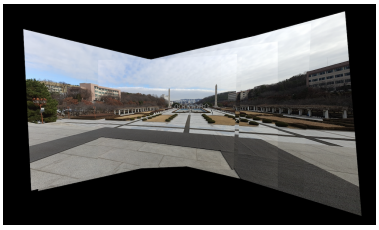
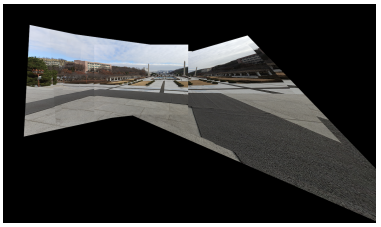
컴퓨터비전 최종 보고서

2020105695 김희성

0. Table of Contents

1. 결과 이미지
2. 과정 설명
    a. 실행시 주의할 점
    b. Flow chart
3. 수업 외의 시도해서 성공한 것
    a. chain Homography(using Dynamic Programming)
    b. RANSAC training set heuristic (4 pair로 학습한 것이 가장 효과적임)
    c. Back warping
    d. blending (averaging, first come occupied)
4. 수업 외의 시도해서 실패한 것
    a. rotation translation decomposition
    b. ORB를 이용한 panorama
        i. BF Match 직접 구현
        ii. Good match pruning
    c. gauss-newton method Optimizing

1. 결과 이미지

[그림1] : reference image	[그림2] : bottom up stitching(panorama)	[그림3] : bottom up homography + stitching
		

[그림1]은 opencv library의 실제 이미지를 사용했을 때의 이미지이다. [그림2]는 라이브러리를 사용한 Homography에 직접 개발한 stitching module을 쓴 것이다. [그림3]은 직접 구한 homography 에
stitching을 한 것이다.
github repository : https://github.com/luckydipper/bottom-up-panorama
gdrive link :
https://drive.google.com/drive/folders/1e2bNPxIDIjEYvDXFevZyDmSq7nJPd9LN?usp=sharing

2. 과정 설명

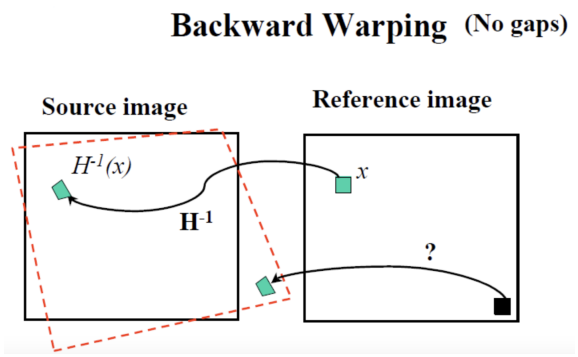
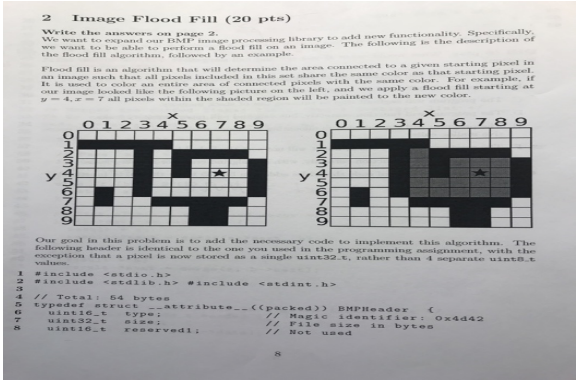
- **2.a** 주의 할 점

1) stack overflow를 방지하기 위해 반드시 아래 명령어를 쳐야 한다. 해당 명령어를 치지 않을 경우 segment fault error가 발생한다.

```
unlimt -s unlimited
```

Backward Warping [그림4]을 직접 구현하는 과정에서 **DFS**로 **Flood Fill [그림5]** 알고리즘을 사용했다. stackoverflow를 방지하기 위해 BFS로도 구현해봤지만 속도가 매우 느렸다. 메모리도 더 많이 잡아먹어 컴퓨터가 멈췄다.

2) 모든 이미지와 keypoint들의 **index**는 **1**부터 시작한다.
3) 이미지**(Mat)**는 **y,x**의 **indexing**을**, pixel**의 좌표는 **x,y,**의 **indexing**을 사용한다.
4) 구현이 아닌 함수에 관한 annotation은 header파일에 들어있다.
5) RANSAC의 sampling이 잘 되지 않은 경우, Homography matrix가 발산하여 검은 이미지를 return 할 수 있다.

[그림4]	[그림5]
	

- **2.b flow chart**

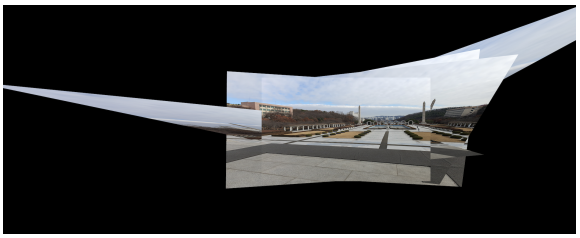
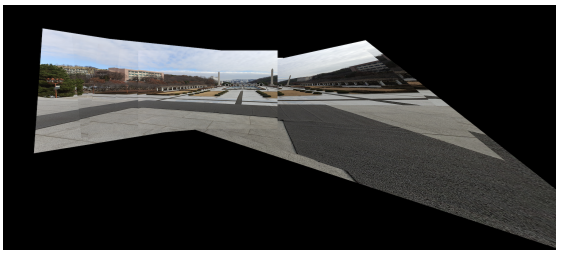
[그림6]은 알고리즘의 flow chart를 나타낸다.
Input : 10장의 pure rotation image
Output : panorama images
10장의 이미지가 순차적으로 들어올 때, 5번째 image를 reference image로 두고, 다른 이미지들을 perspective transform하여 붙힌다.

3. 수업 외의 시도해서 성공한 것

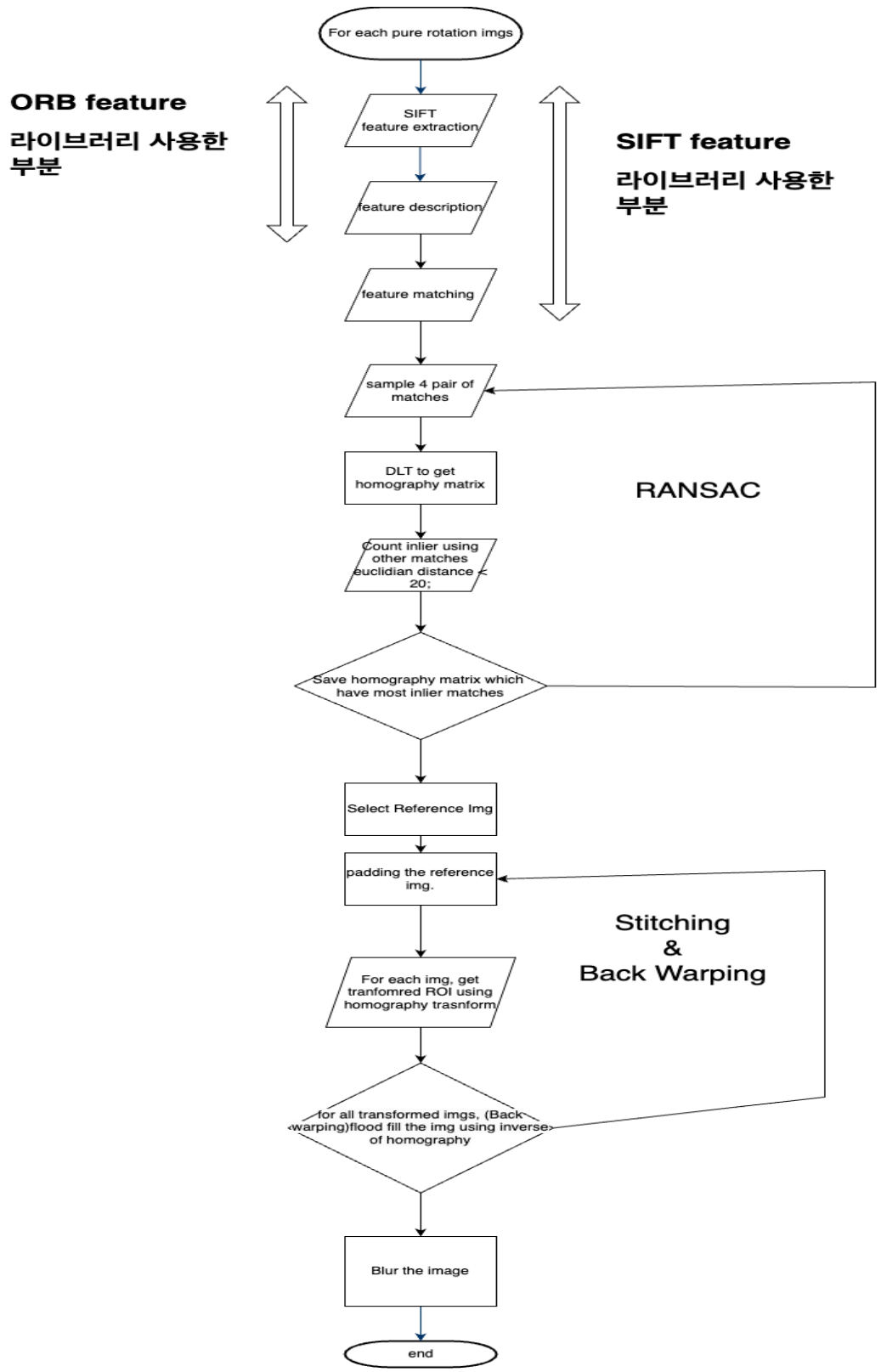
- **3.a chain homography (using Dynamic Programming)**

i가 reference index 보다 작을 때에 아래 식이 성립한다. 재귀적 구조를 띈다. 즉, Dynamic programming을 이용해 1씩 증가하는 값을 미리 저장해두면, 더 빠르게 homography를 구할 수 있다. matching 되는 feature가 적은 index 1과 10의 image도 robust하게 stitching할 수 있었다. [그림7,8] 비교.

$$H_{[i][refernce]} = H_{[i][i+1]} \cdot H_{[i+1][i+2]} \cdot \ldots \cdot H_{[reference-1][refernce]}$$

[그림7] : Before applying chain Homography	[그림8] : After applying chain Homography
	

[그림6] Flow Chart



- **2.b RANSAC training set heuristic**

DLT에서 matrix를 만들 때 4point pair를 RANSAC으로 이용해서 homography를 구할 때가 가장 좋은 homography를 얻을 수 있었다. 4쌍 이상으로 올리면 RANSAC에서 더 적은 inlier가 나오는 결과를 보였다.

- **2.c Back warping**

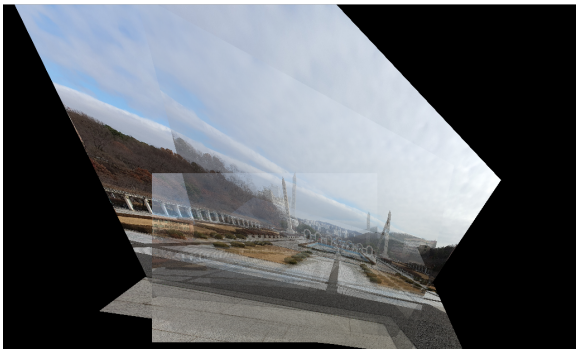
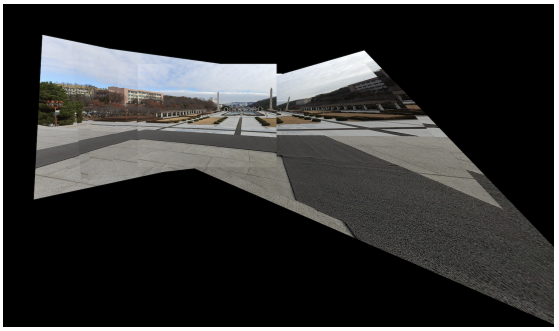
수업 시간에서는 back warping의 과정에 대해서 배우지 않았다. 이미지의 끝점을 perspective 변환하여 이동시킨 후, transform 된 사각형의 중점부터 flood fill algirhthm을 적용하였다. 이 때 원본 이미지의 pixel에 접근하여 interpolation하여 transformed pixel에 지정했다.

- **2.d blending (averaging, first come occupied)**

stitched 된 이미지를 가져다 붙일 때, 두 가지 blending을 시도해봤다.

1) 먼저 온 pixel을 우선적으로 고려한다.
2) averaging을 통해 blending 을 실시한다.

-> goasting 문제 때문에 reference image를 중간으로 기준을 둔 후 first come occupied algorithm을 사용 하는 것이 더욱 효과적이었다.

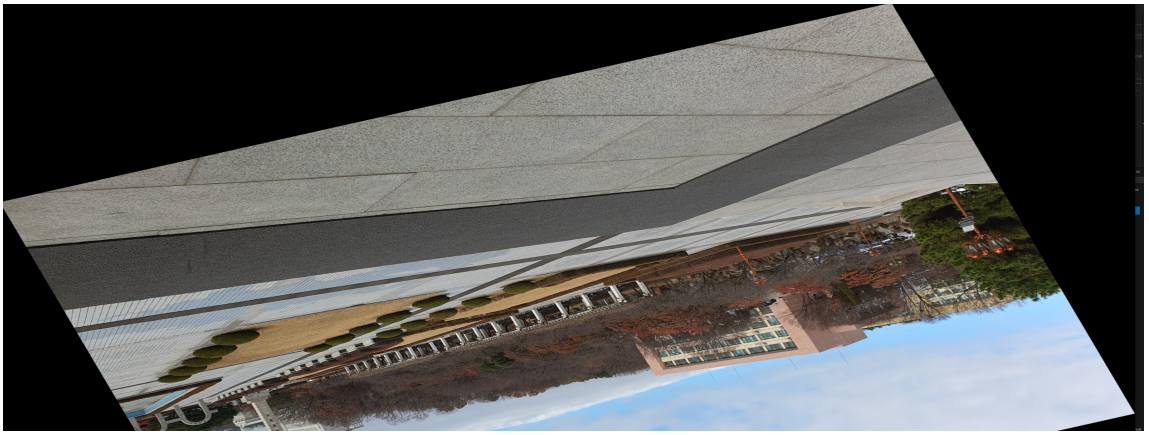
[그림7] average blending	[그림8] first come occupied
	

4 . 수업 외의 시도해서 실패한 것

- **4.a Rotation Translation Decomposition**

$\boxed{\tilde{\mathbf{p}}_{ij}^{k} = \mathbf{K}_i \mathbf{R}_i \mathbf{R}_j^T \mathbf{K}_j^{-1} \tilde{\mathbf{u}}_j^l.}$ pure rotation의 경우 옆의 식으로 최적화를 진행할 수 있다고 ‘Automatic Panoramic Image Stitching using Invariant Features’ 해당 논문에 나와있었다. 그래서 rotation matrix와 intrinsic matrix로 나눈 후 최적화를 진행하여 봤지만, perspective transform이 아니라 Euclidean transform을 한 이미지만 얻어졌다. [그림9]은 focal length가 12mm로 같을 때, z축을 기준으로 PI, PI/9 만큼 회전한 이미지를 위의 식으로 예측한 결과이다.

[그림9] : Perspective transform이 아닌 Euclidean transform이 나온 모습.



- **4.b ORB**를 사용한 **panorama**

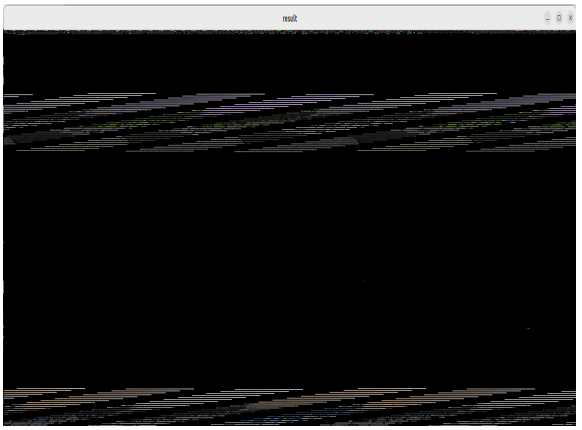
원래는 ORB feature를 이용해서 위의 과정들을 수행하려고 했다. ORB 논문도 코드도 읽어보고, **Matcher**까지 직접 구현했었다. descriptor부터 **hamming distance**도 직접 구해서, 이를 기준으로 match들을 sorting한 후 **good match**만 **pruning**해봤다. 그러나, **orb feature**에서는 좋은 결과를 얻을 수 없었다**. matcher.cpp**를 보면 고생의 흔적이 남아있다**.** 그러나 사용하지 않았다**.** matching되는 sample들이 sparse 하게 연결 되어 있기 때문 같다.

[그림10] Bottom up ORB matcher implementation	[그림11] Bottom up ORB error-2

```
int hammingDistance(const uchar code1, const uchar code2){
    int result = 0;
    int XOR = (code1 ^ code2);
    while(XOR >0){
        if(XOR%2 > 0)
            result++;
        XOR >>= 1;
    }
    return result;
}

        struct bottom_up::FeatureMapping

vector<FeatureMapping> orbFeatureMatch(const Mat & descriptor1, const Mat &descriptor2){
    assert(descriptor1.rows == descriptor2.rows && descriptor1.cols == descriptor2.cols);
    vector<FeatureMapping> result_match;
    int num_features = descriptor1.rows;
    for(int i = 0; i < num_features; i++){
        for(int j = 0; j < num_features; j++){
            int distance=0;
            for(int bit_seq = 0; bit_seq < descriptor1.cols; bit_seq++){
                uchar batch1 = descriptor1.at<int>(i,bit_seq);
                uchar batch2 = descriptor2.at<int>(j,bit_seq);
                distance += hammingDistance(batch1, batch2);
            }
            result_match.push_back(FeatureMapping(i, j, distance));
        }
    }
    return result_match;
}
```



- **4.c gauss-newton method Optimizing**

opencv documentation을 읽어보면 Levenberg-Marquardt 방식으로 Homography를 얻었다고 한다. 그래서 그 하위 버전 인 gaussian newton 방식으로 homography를 최적화 하고 싶었지만 불가능하였다.

- **4.d** 이 밖에도 **spherical coordinate**이나 **cylindrical coordinate**으로 변환하는 것도**, PROSAC**도 해보고 싶었는데 시간이 부족했다**.**