# 움직이는 카메라와 특징점 추출을 이용한 스테레오 비전

3학년 X반 XX번 이XX
3학년 X반 X번 서규호
지도교사 : 이XX

**초 록**

현재 사용되는 대부분의 스마트폰에는 한 면에 카메라가 하나씩만 달려 있어 공간 안의 장애물을 파악할 수 없다. 이는 AR 애플리케이션에서 가상의 물체를 나타낼 때 물체가 장애물을 통과하는 등 비현실적인 모습이 나타나는 문제를 유발한다. 본 연구는 한 개의 카메라가 서로 다른 위치와 다른 방향에서 찍은 사진들을 이용한 공간의 입체적인 인식을 통해 공간 내의 장애물을 파악함으로써 단일 카메라로도 현실감 있는 AR을 구현하는 것을 목표로 한다.

## 1. 서론(Introduction)

### 1.1 필요성과 목적

현재, 여러 AR 애플리케이션들이 개발되어 있으며, AR을 이용한 애플리케이션들은 앞으로도 늘어날 것이다. AR은 실제 공간을 보여주는 화면에 가상의 물체를 띄움으로써 실제 공간에 가상의 물체가 있는 듯 한 효과를 준다. 하지만 디바이스가 공간의 장애물에 대한 정보를 알지 못하면 멀리 바닥에 있어야 하는 물체가 바로 앞의 벽에 나타나는 등 현실감이 떨어진다. 공간의 장애물에 대한 정보를 얻기 위해서는 거리 측정용 센서 또는 스테레오 카메라 등의 장치가 필요하다. 하지만, 가장 많이 보급되어 있는 모바일 디바이스인 스마트폰에는 대부분이 카메라가 앞면과 뒷면에 하나씩 달려있어 한 개의 카메라만 사용할 수 있어 거리 측정이 불가능하다.

스테레오 비전은 두 카메라가 떨어져 위치해 나타나는 시각의 차이를 이용한다. 단일 카메라로도 서로 다른 위치에서 찍은 두 이미지와 위치 차이를 이용하면 같은 효과를 낼 수 있다. 이 연구의 목적은 모바일 디바이스의 위치 이동을 파악하고, 서로 다른 위치에서의 이미지들을 이용해 단일 카메라로도 거리 계산을 통해 공간의 장애물을 인식함으로써 더욱 현실감 있는 AR을 구현하는 데에 있다.

## 1.2 선행연구

### 1.2.1 거울의 이용

“Depth Perception with a Single Camera”, “Single-camera stereo vision for obstacle detection in mobile robots” 등과 같은 논문에서는 그림 16과 같이 거울을 이용해서 하나의 카메라로 stereo camera와 같은 효과를 내었다. 이렇게 찍게 되면 하나의 화면에 stereo camera로 찍은 것처럼 보이는 사진 두 개를 찍을 수 있기 때문에 하나의 카메라로 stereo vision을 구현할 수 있다.


그림 16. 거울 장비의 예

### 1.2.2 핸드폰을 수평 이동해 찍은 두 개의 사진 이용

“Measuring Distance with Mobile Phones Using Single-Camera Stereo Vision” 논문에서는 위와 같이 거울을 이용하지 않고 휴대폰에 있는 Single Camera로 Stereo Vision을 구현하였고 이를 이용해 장애물과의 거리를 계산하였다. 이 논문에서 구현한 것은 핸드폰으로 한 번 사진을 찍고 그 다음 옆으로 살짝 옮겨서 사진을 찍은 뒤에 그 사진 두 개로 물체와 핸드폰 사이의 거리를 알아내게 한 것이다. 이 논문의 원리는 휴대폰에 있는 가속도 센서로 카메라가 얼마나 이동했는지를 구한 다음에 두 이미지의 disparity를 이용해 두 사진을 매칭시키는 것이다. 이렇게 두 사진을 매칭시키게 되면 어떤 물체(장애물)과 찍은 카메라 사이의 거리를 계산할 수 있다.

## 1.3 관련 이론

- 부록 참고

# 2. 화면에 수평하게 이동한 두 사진을 이용한 거리 인식 실험

이 실험에서는 SIFT와 SURF 특징점 추출 알고리즘을 이용해 서로 다른 위치에서 촬영한 두 이미지에서 특징점을 추출한다. 그리고 추출된 특징점들을 비교해 같은 점들을 찾은 후 각 점들의 화면에서의 위치 변화를 이용해 그 점까지의 거리를 계산한다. SIFT, SURF, 매칭 알고리즘은 openCV 라이브러리를 사용한다.

스테레오 비전을 구현하는데 특징점 추출 알고리즘을 이용한 이유는 최종적으로 모바일 디바이스를 이동, 회전시켜도 문제없이 장애물 인식을 하는 것을 목표로 하기 때문이다. 블록으로 픽셀 값들을 비교하는 것보다는 scale, rotation invariant한 특징점 추출 알고리즘을 이용하는 편이 두 이미지를 찍은 위치 차이가 크고 회전되어도 같은 물체를 인식하기 쉬울 것이며, affine invariant 조건까지 만족하는 ASIFT를 사용하면 인식은 더 잘 이루어 질 것이다.

이 실험의 목적은 SIFT, SURF, 매칭 알고리즘이 작동하는 개발 환경 구축과, 제시한 방법으로 거리를 인식할 때 생길 수 있는 오류와 문제점들을 점검하는 것이다. 이 실험에서는 두 사진을 찍은 위치 차이는 입력하지 않았고, 따라서 실험의 결과는 점들까지의 정확한 거리가 아니라 점들 사이의 상대적인 거리이다.

## 2.1 연구 방법

### 2.1.1 사진 촬영

거리차가 존재하는 물체들이 보이는 사진을 촬영한다. 한 번 촬영 후, 스마트폰 카메라를 최대한 화면에 평행하게, 수평 방향으로 임의의 거리만큼 이동시키고 두 번째 사진을 촬영한다. 카메라의 회전은 최소한으로 한다.

### 2.1.2 특징점 추출

촬영된 두 사진은 연산을 줄이기 위해 가로, 세로 길이를 1/4로 줄인다. 줄어든 사진은 SIFT 알고리즘을 이용해 특징점(keypoint)을 찾아낸 후 각 특징점의 영상 기술자(descriptor)를 계산한다. 계산된 영상 기술자는 그 특징점의 특징을 나타내며, 두 영상 기술자가 같으면 두 특징점은 같은 모양이다.

### 2.1.3 특징점 매칭

다음으로 한 이미지의 영상 기술자들은 다른 이미지의 영상 기술자들과 비교된다. FLANN(Fast Approximate Nearest Neighbor Search) based matcher를 이용한다. 매칭의 결과는 매칭된 두 특징점의 인덱스와 distance(영상 기술자의 차이) 값으로 구성되는 DMatch 구조체의 벡터로 산출된다. Distance가 클수록 잘못된 매치일 확률이 높기 때문에, DMatch 벡터를 distance 순으로 정렬하고, distance가 작은 DMatch들을 DMatch 벡터 길이의 1/16만 선별해 goodmatches 벡터를 만든다.

goodmatches가 너무 적다고 판단되면 선별 개수를 조절한다.

이 실험에서는 두 이미지 사이에서 모든 점의 이동 방향은 동일하다. 따라서 disparity(매치된 두 특징점의 위치 차이)의 방향 또한 모두 동일해야 하며, 이 방향이 다른 매치는 틀린 매치임이 분명하다. 이러한 특징을 이용해 goodmatches 중 40개의 disparity의 방향을 (y 좌표 변화량)/(x 좌표 변화량)으로 기울기로 나타내고, 평균을 계산해 그 기울기에서 ±0.01의 범위 내에 들어오는 기울기를 가지는 매치들만 선별해 parallelmatches 벡터를 만든다. 이러한 방법으로 틀린 매치의 수를 크게 줄일 수 있다. parallelmatches의 수가 너무 적으면 기울기를 ±0.03의 범위로 실험한다.

#### 2.1.4 결과 시각화

마지막으로 실험 결과는 이미지의 형태로 산출된다. 먼저, parallelmatches 내의 DMatch에서 disparity를 계산하고, disparity의 크기의 최댓값 distmax, 최솟값 distmin을 구한다. 이후 두 이미지를 겹쳐 보이도록 혼합해 한 이미지로 만들고, 매칭된 두 특징점을 연결하는 선을 그린다. 이때, 선과 특징점의 색은 disparity를 distmin, distmax에 비교해 distmax는 빨간색, distmin은 파란색으로 내삽 하여 결정한다. 따라서 가까이 있는 점일수록 빨간색에 가까운 색, 멀리 있는 점일수록 파란색에 가까운 색으로 나타난다.

SIFT로 실험 한 후에는 SURF로 동일하게 진행하고 두 결과를 비교하여 SIFT와 SURF 중 더 적합한 알고리즘을 찾는다.

### 2.2 결과(Data & Result)

#### 2.2.1 실험 1

실험 1은 거리가 다른 여러 종류의 물체를 가지고 실험해 보았다. 그림 2에서 SIFT가 SURF보다는

더 정확한 결과가 나왔다. SIFT와 SURF 모두 물체의 모서리를 특징점으로 잡은 경우에 잘못된 매치가 생겼다. SURF는 노트북 모서리에서 위치 차이의 편차가 가장 크게 나타났다. SIFT와 SURF는 서로 특징점을 잡은 위치가 달랐고, 특징점의 개수는 SURF가 더 많다.


그림 2 실험 1 SIFT 이용 결과


그림 3 실험 1 SURF 이용 결과

### 2.2.2 실험 2

실험 2에서는 실험 1과 같은 환경에서 카메라를 더 앞으로 이동시켜 다시 실험해 보았다. SIFT는 주로 뒤의 상자와 키보드에서 잘못된 매치가 생겼고, SURF는 뒤의 상자의 특징점은 잘 찾았지만 노트북

에 잘못된 매치가 매우 많을뿐더러 원통의 특징점은 하나도 잡지 않았다. 그림 20에서 노트북 화면은 주황색으로 나타나야 한다. 특징점 크기가 클수록, 그리고 내부에 독특한 특징이 있을수록 정확한 매칭이 되는 것이 보인다. 또한, SIFT의 경우 서로 다른 키보드 버튼끼리 매칭되었다.

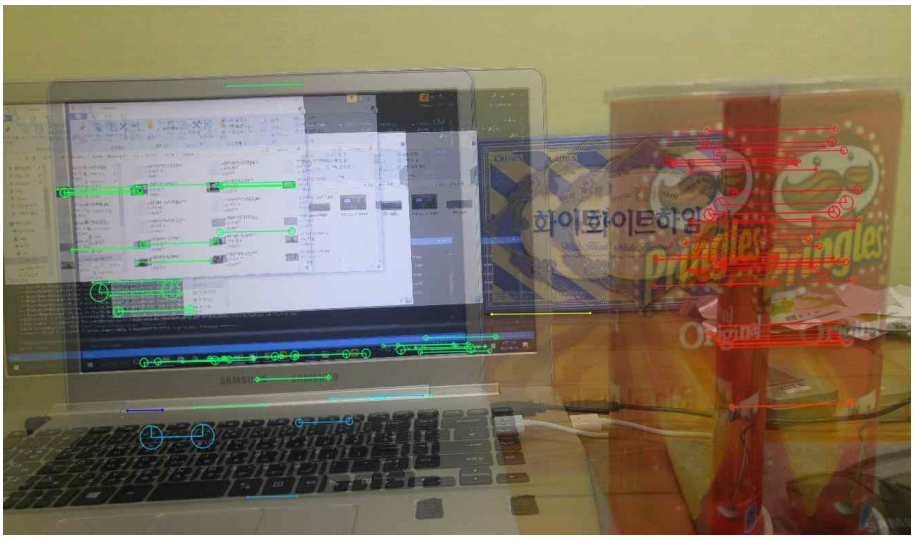



그림 4 실험 2 SIFT 이용 결과

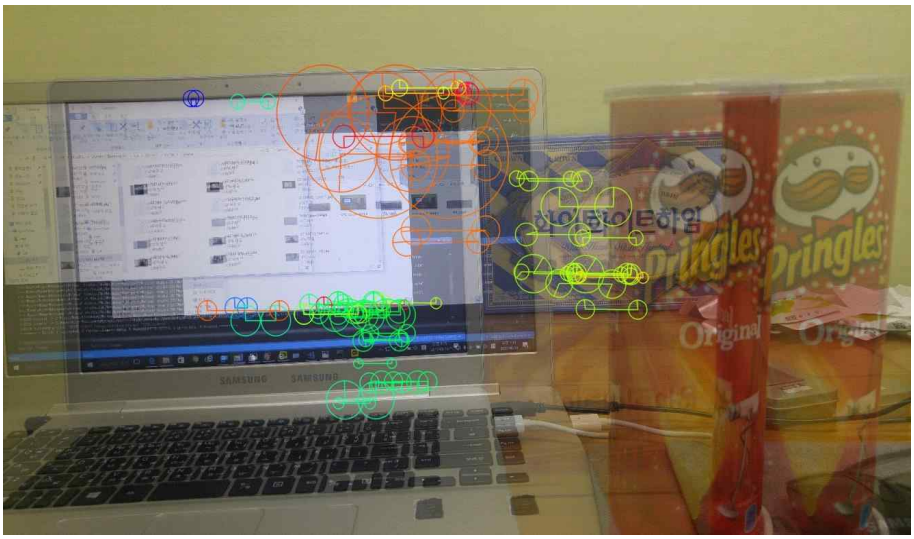



그림 5 실험 2 SURF 이용 결과

### 2.2.3 실험 3

실험 3부터는 컴퓨터실에서 찍은 사진이다. 똑같이 생긴 모니터와 책상들이 서로 다른 거리로 위치한다. SIFT는 특징점의 수도 적고, 모두 잘못된 매치이다. 반면 SURF는 하나의 잘못된 매치 외에는 모두 제대로 된 매치이다.

SIFT를 사용했을 때 평균 기울기를 구한 것이 잘못된 기울기를 구하게 된 것으로 보인다. 평균을 사용했기 때문에 생겨난 문제로 보이며, 최빈값을 사용하는 등의 개선이 필요하다고 보인다.

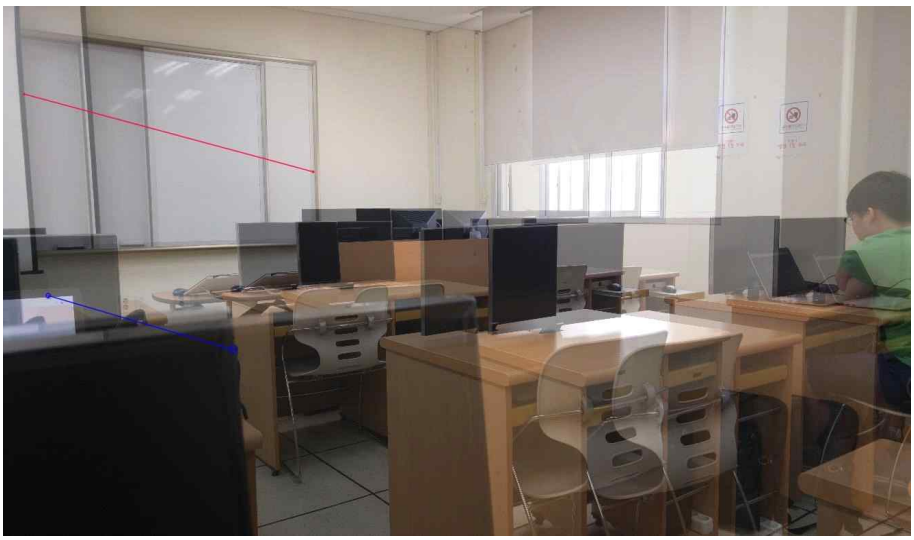


그림 6 실험 3 SIFT 이용 결과

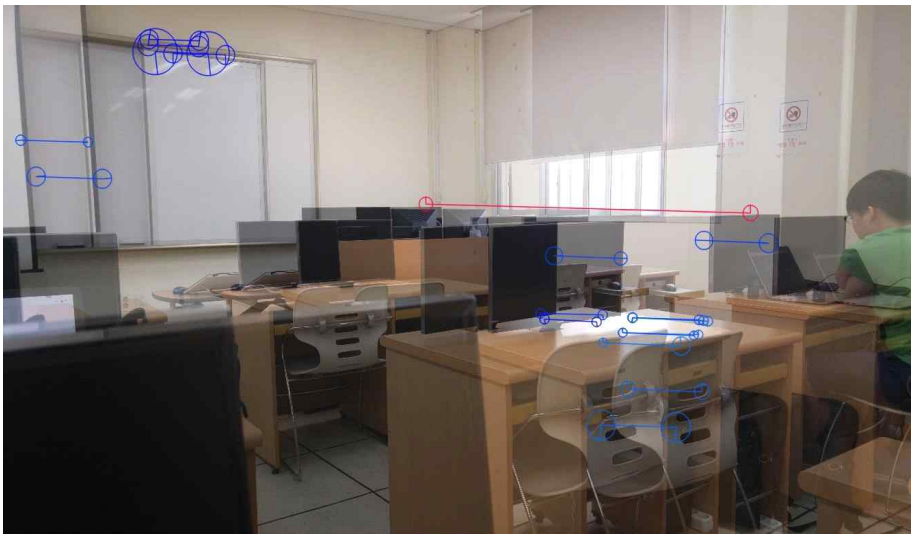


그림 7 실험 3 SURF 이용 결과

### 2.2.4 실험 4, 5

-부록에 첨부

### 2.2.5 실행 시간

| 표 1. 2장의 SIFT를 이용한 실험의 실행 시간 | | | | | | |
|---|---|---|---|---|---|---|
| 과정 | 실험 1 | 실험 2 | 실험 3 | 실험 4 | 실험 5 | 평균 |
| 특징점 추출 | 3.339 | 3.454 | 2.935 | 2.990 | 3.111 | 3.166 |
| 영상 기술자 계산 | 2.740 | 2.903 | 1.983 | 2.118 | 2.328 | 2.414 |
| 특징점 매칭 | 1.870 | 2.624 | 0.550 | 0.813 | 1.226 | 1.417 |
| 매치 필터링 | 0.008 | 0.010 | 0.002 | 0.004 | 0.005 | 0.006 |
| 총 실행 시간 | 7.957 | 8.991 | 5.470 | 5.925 | 6.670 | 7.003 |

| 표 2. 2장의 SURF를 이용한 실험의 실행 시간 | | | | | | |
|---|---|---|---|---|---|---|
| 과정 | 실험 1 | 실험 2 | 실험 3 | 실험 4 | 실험 5 | 평균 |
| 특징점 추출 | 2.517 | 2.656 | 1.836 | 2.107 | 2.542 | 2.332 |
| 영상 기술자 계산 | 9.481 | 10.187 | 5.399 | 6.838 | 8.610 | 8.103 |
| 특징점 매칭 | 4.583 | 5.833 | 2.147 | 3.341 | 4.667 | 4.114 |
| 매치 필터링 | 0.025 | 0.024 | 0.010 | 0.018 | 0.024 | 0.020 |
| 총 실행 시간 | 16.606 | 18.700 | 9.392 | 12.304 | 15.843 | 14.569 |

SIFT를 이용한 경우 평균적으로 특징점 추출에 3.166초, 영상 기술자 계산에 2.414초, 매칭에 1.417초, 매치 필터링에 0.006초가 걸려, 총 7.003초가 걸렸다. SURF를 이용한 경우 평균적으로 특징점 추출에 2.332초, 영상 기술자 계산에 8.103초, 매칭에 4.114초, 매치 필터링에 0.020초가 걸려, 총 14.569초가 걸렸다. SURF는 특징점 추출에는 시간이 적게 걸리지만 추출하는 특징점의 수가 많아 이후의 과정에 시간이 더 걸렸다. 장애물 파악을 위해서는 실행이 빨라야 하므로 실행 속도 면에서는 SIFT가 더 적합하다.

## 2.3 고찰(Discussion)

앞선 실험에서 SIFT, SURF, FLANN based matching 등의 알고리즘을 사용할 환경을 갖추었고, 특징점 추출과 특징점 매칭을 통해 스테레오 비전을 구현할 경우 생기는 여러 문제들을 파악하였다.

### 2.3.1 직선 모서리 특징점

결과에서 볼 수 있었듯이 특징점이 직선 모서리 위에 있는 경우에는 똑같은 형태의 특징점이 직선 모서리의 다른 위치에 존재할 수 있다. 이는 disparity에 오차를 발생시켜 계산된 거리에 오차를 발생시킨다. 이 문제를 해결하기 위해서는 직선 모서리를 의미하는 영상 기술자의 특징을 알아내어 필터링하거나, 꼭짓점 검출 알고리즘을 특징점 추출에 사용해야 할 것이다.

### 2.3.2 Disparity 기울기 평균 이용

실험 3의 SIFT 이용 결과와 실험 4의 SURF 이용 결과를 살펴보면 disparity의 방향이 카메라의 이동 방향과 일치하지 않는다. 이는 disparity 기울기의 평균을 이용했기 때문에 40개의 기울기 중 잘못된 매치들의 기울기가 한쪽으로 편향된 경우 문제가 생기는 것으로 보인다. 평균 대신 최빈값을 이용

하면 문제가 줄어들 것으로 보인다.

#### 2.3.3 특징점 크기

앞선 실험들에서 주로 잘못된 매치들이 발생하는 특징점들은 대부분 크기가 작다. 특징점이 작을수록 영상 기술자의 유일성이 줄어들기 때문이며, 특징점의 크기가 큰 매치에는 가중치를 주어 goodmatches에 들어갈 수 있도록 하면 정확한 매치의 수가 늘어날 것이다.

#### 2.3.4 불균일한 특징점 분포

공간의 장애물을 파악하기 위해서는 화면의 모든 점의 거리를 알아야 한다. 하지만 특징점 추출 알고리즘을 사용했기 때문에 디바이스와 특징점의 거리밖에 알 수 없고, 특징점이 불균일하게 분포해 있기 때문에 장애물의 유무를 알지 못하는 공간이 생기기도 한다. 이 문제를 해결할 방법이 필요하다.

#### 2.3.5 더 적합한 특징점 추출 알고리즘

앞선 실험의 결과를 보면 SIFT를 사용했을 때 잘못된 매치의 수가 더 적은 등 SIFT의 정확도가 더 높았고, 실행 시간도 짧았다. SIFT 알고리즘이 더 적합하다고 판단된다.

# 3. RANSAC 알고리즘의 이용

2장의 실험에서는 추출된 특징점들의 특징을 비교해 오차제곱합 방법을 이용해 같은 점들을 찾고 그 점을 매칭시켰다. 하지만 이 과정에서 잘못된 매치들이 나타나는 것을 확인할 수 있었고 이를 해결하기 위해 RANSAC 알고리즘을 도입하였다. RANSAC 알고리즘은 한 이미지에서 다른 이미지를 찾는 목적으로도 쓰이며, 본 실험에서는 평행이동만 일어난 두 이미지 중 한 이미지를 다른 이미지에서 찾으려 하는 꼴이 된다. 그럼에도 틀린 매치를 필터링하는 효과는 가질 것이고, RANSAC 알고리즘의 성질을 파악해 다르게 활용할 수 있을 방법을 찾기 위해 RANSAC 알고리즘을 적용해 실험했다.

## 3.1 RANSAC 적용 실험

#### 3.1.1 연구 방법(Method)

전체적인 연구 방법은 2장의 실험과 동일하다. 매치들 중 잘못된 매치를 걸러내는 과정을 RANSAC 알고리즘으로 대체한다는 점만 달라졌다.

goodmatches는 전체 matches 중 distance가 작은 상위 50%만 선별되어 구성되고, 이 중 RANSAC 알고리즘을 이용해 inlier로 판별된 keypoint와 매치만이 시각화 되었다. OpenCV 라이브러

리의 findHomography 함수가 사용되었다. 특징점 추출은 SIFT를 사용하였다.

### 3.1.2 실험 결과

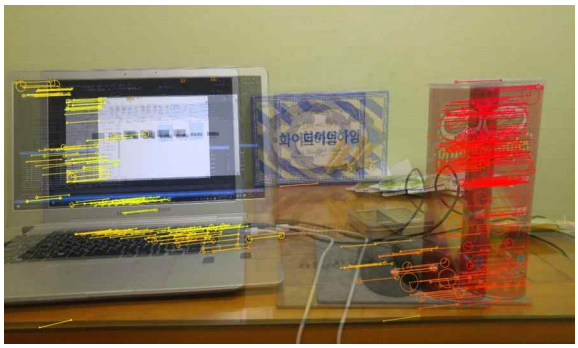


그림 8 실험 1에 RANSAC 적용 결과

그림 9 실험 2에 RANSAC 적용 결과

그림 10 실험 3에 RANSAC 적용 결과

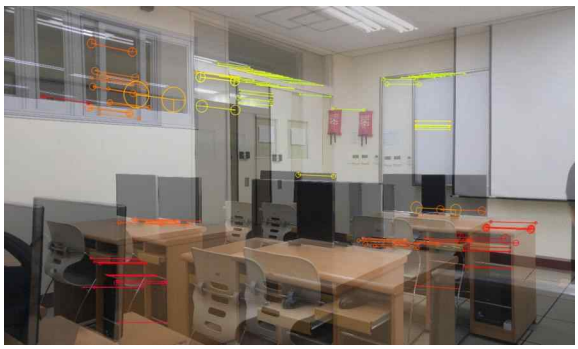


그림 11 실험 4에 RANSAC 적용 결과

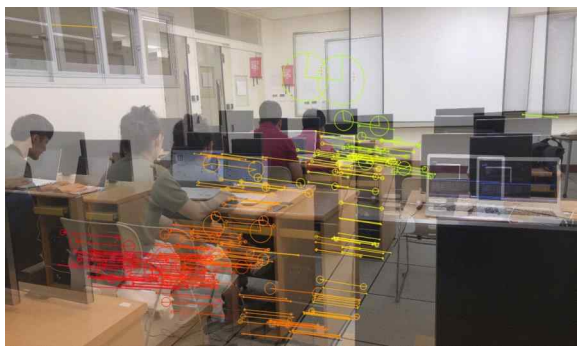


그림 12 실험 5에 RANSAC 적용 결과

그림 8부터 그림 12까지 관찰하면, 공간상의 특정한 평면에 가까이 있는 특징점들만 나타난 것으로 보인다. 나타난 특징점들은 평면을 벗어난 매우 적은 수의 잘못된 매치를 제외하고는 모두 정확한 매치이다. RANSAC 알고리즘을 통해 이미지 일부분의 틀린 매치는 필터링 할 수 있었지만, 평면에서 떨어져 있는 특징점들의 정확성은 확인할 수 없다는 문제점이 있다. 언급한 특정한 평면은 두 이미지에서 RANSAC 알고리즘으로 찾아낸 서로 공통되는 이미지를 포함하는 평면일 것으로 추정된다. 특정 평면

은 실험 1과 실험 2의 이미지에서 가장 명확하게 확인할 수 있다. 실험 1과 실험 2의 결과에서 모니터와 자판에서의 평면과의 접촉선은 서로 다른 기울기를 가지며, 이를 통해 카메라로부터의 거리를 직관적으로 확인할 수 있다.

### 3.1.3 고찰

RANSAC 알고리즘을 적용한 결과, 특정 평면에 가까이 있는 특징점들만 나타나며, 이 점들의 매치는 정확하다는 흥미로운 결과를 얻을 수 있었다. 잘못된 매치를 필터링 할 수 있는 탁월한 방법이지만, 그 평면 근처의 특징점의 거리만을 측정할 수 있다는 큰 단점이 있다. 이에 문제를 해결할 수 있는 한 가지 방법을 제시한다. 그 평면의 위치나 회전 상태를 변화시키는 방법이다. 추세선 역할을 하는 평면의 위치나 회전 상태를 변화시킬 수 있는지 추가적인 실험이나 정보가 필요하지만, 기본적인 개념은 여러 개의 평면으로 RANSAC을 적용해 모든 특징점이 알맞은 매치를 가지도록 하는 것이다.

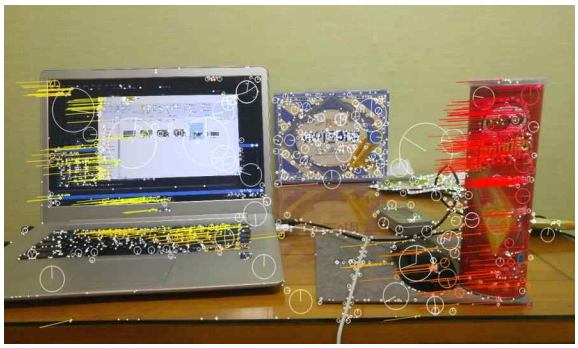


그림 13 SIFT로 검출된 특징점과 RANSAC에 걸러진 특징점

그림 14 SURF로 검출된 특징점과 RANSAC에 걸러진 특징점

그림 13에서 SIFT 알고리즘이 찾아낸 특징점들을 확인할 수 있다. 부분적으로만 분포되어 있는 것을 확인할 수 있다. 반면, 그림 14를 보면 SURF 알고리즘이 찾아낸 특징점들은 넓게 분포되어 있다는 것을 알 수 있으며, 위에서 제시한 방법으로 모든 특징점들의 카메라부터의 거리를 알 수 있다면, 특징점이 넓게 분포되어 있는 SURF 알고리즘으로 특징점을 찾아내는 것이 유리하다고 생각된다.

## 3.2 RANSAC 중첩 적용 실험

RANSAC을 사용한 앞선 실험에서는 잘못된 매치를 걸러내는 기능은 탁월했지만, 일부의 특징점만을 구할 수 있다는 문제가 있었다. 따라서 그 문제를 해결하기 위해 고찰에서 제시한 방법과 유사한 방법으로 RANSAC을 여러 번 적용해 더 많은 특징점을 구했다.

### 3.2.1 실험 방법

앞의 실험에서 RANSAC을 적용하는 과정에서만 차이가 있다. RANSAC을 적용해 전체 matches

중에 특정 평면에 위치하는 매치들만 goodmatches 이동시킨다. 그 후에 goodmatches로 이동된 매치들을 제외한 나머지 매치들에 다시 RANSAC을 적용해 다른 특정 평면에 위치하는 매치들을 goodmatches로 이동시킨다. 이를 3~6회 정도 시행하면 RANSAC을 한번만 적용했을 때와는 달리 일부의 매치가 아니라 대부분의 정확한 매치들을 얻을 수 있다.

#### 3.2.2 실험 결과

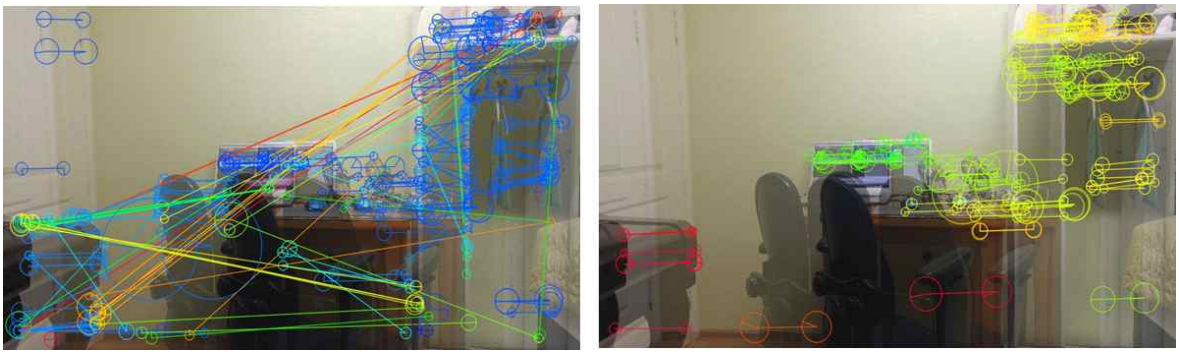


그림 15 RANSAC 적용 전 RANSAC 적용 후

RANSAC과 여러 휴리스틱 알고리즘을 적용해 좋은 매치들을 아주 효과적으로 분리하는데 성공하였고 이 매치들을 이용해 물체까지의 상대적인 거리를 구하였다. 그림 15에서 볼 수 있듯이 정확한 매치들을 선별할 수 있었다. 하지만, RANSAC을 적게 적용하면 일부의 특징점밖에 구하지 못하고, 많이 적용하면 잘못된 매치들이 들어가는 문제가 있어 RANSAC을 몇 번 적용해야 할지 결정해야 한다는 문제가 있다.

# 4. 특징점의 공간 배치 시각화

## 4.1 연구 방법

### 4.1.1 좌표 계산

특징점들이 카메라에 대해서 x,y,z축 방향으로 얼마나 떨어져 있는지를 알기 위해 특징점의 공간상의 위치를 벡터를 이용해 계산하였다. 카메라 화면에서의 점은 공간에서의 선에 대응된다. 따라서 화면의 특징점은 특징점에 대응되는 공간상의 직선 위에 있다는 것을 알 수 있다. 카메라로 두 번 촬영했기 때

문에 특징점은 각각 대응되는 두 직선의 교점 위에 있다는 것을 알 수 있다. 하지만 실제로는 오차가 있어 정확히 그 두 직선이 삼차원 공간 상에서 교차를 하지 않기 때문에 물체의 위치를 그 두 직선에서 가장 가까운 점이라 정의한다. 즉 두 직선의 공통 수선의 중간지점을 물체의 위치라고 정의하였다. 그렇게 정의를 하게 되면 물체의 위치는 아래와 같이 나온다. 왼쪽에서 찍은 카메라의 위치벡터를 A라고 하고 그 카메라 화면의 특징점에 대응되는 직선의 방향벡터를 $\vec{a}$라고 한다. 마찬가지로 오른쪽 카메라도 위치벡터 B와 방향벡터 $\vec{b}$를 구할 수 있다. 그리고 $\vec{c}$는 $\overrightarrow{AB}$라고 정의했다. 이렇게 정의를 하게 되면 위치벡터 D, E를 그림 16과 같은 식으로 계산할 수 있고 그러면 물체의 위치는 D와 E의 중간 지점이 되게 된다. 따라서 벡터방정식을 잘 풀어 물체의 위치를 잘 구하였다.

$$D = A + \vec{a}\frac{-\left(\vec{a}\cdot\vec{b}\right)\left(\vec{b}\cdot\vec{c}\right)+\left(\vec{a}\cdot\vec{c}\right)\left(\vec{b}\cdot\vec{b}\right)}{\left(\vec{a}\cdot\vec{a}\right)\left(\vec{b}\cdot\vec{b}\right)-\left(\vec{a}\cdot\vec{b}\right)\left(\vec{a}\cdot\vec{b}\right)}$$

$$E = B + \vec{b}\frac{\left(\vec{a}\cdot\vec{b}\right)\left(\vec{a}\cdot\vec{c}\right)-\left(\vec{b}\cdot\vec{c}\right)\left(\vec{a}\cdot\vec{a}\right)}{\left(\vec{a}\cdot\vec{a}\right)\left(\vec{b}\cdot\vec{b}\right)-\left(\vec{a}\cdot\vec{b}\right)\left(\vec{a}\cdot\vec{b}\right)}$$

그림 16 D, E 계산식

#### 4.1.2 좌표 시각화

좌표를 시각화하기 위해 Unity 게임엔진을 이용하였다. 최종적으로 프로그램을 실행시키면 특징점들을 openCV라이브러리를 이용해 선별하고 이것을 매칭한 후 좌표를 계산해 바로 Unity 공간상에 띄워주는 것까지 완성하였다. 물체는 구로 표현하였고 한눈에 알아볼 수 있게 구의 크기를 잘 조절하였다. 또한 이렇게 물체를 3차원 공간상에 표시를 하게 되면 카메라로부터 물체까지의 거리를 구할 수 있다. 따라서 이제는 물체까지의 거리를 정확하게 구할 수 있게 되었다.

### 4.2 결과

#### 4.2.1 카메라를 평행이동해 거리를 계산한 결과

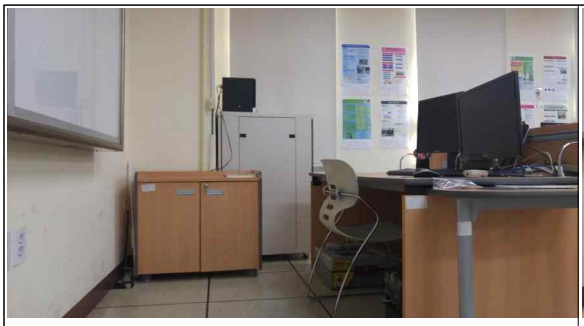


그림 17 왼쪽 사진

그림 18 오른쪽 사진

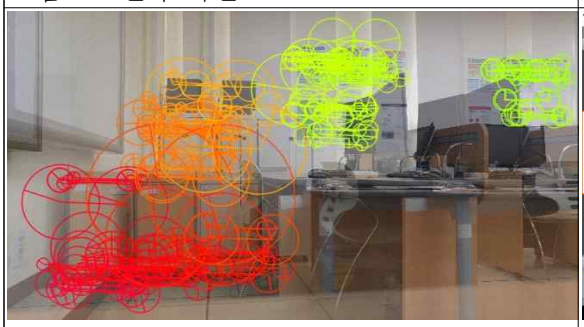


그림 19 RANSAC 3회 결과

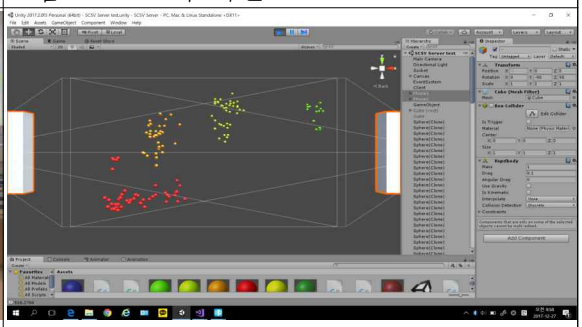


그림 20 카메라 위치에서 본 결과

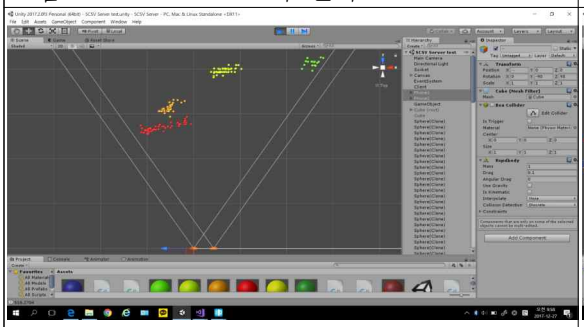


그림 21 위에서 직교투영한 결과

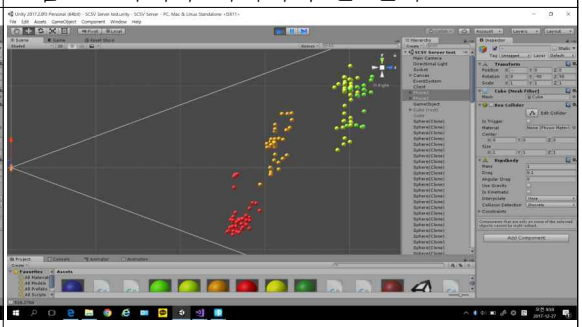


그림 22 옆에서 직교투영한 결과

카메라 이동: x축 43cm

| 물체 | 실제 거리(cm) | | 측정 거리(cm) | | 상대오차(%) | |
|---|---|---|---|---|---|---|
| 서랍장 | 306 | 300 | 287 | 281 | 6.20 | 6.33 |
| 스피커 | 361 | 332 | 351 | 325 | 2.77 | 2.10 |
| 포스터1 | 429 | | 425 | | 0.93 | |
| 포스터2 | 515 | | 492 | | 4.46 | |

표 3 4.2.1의 거리 계산 결과와 실제 거리 비교

이 실험은 카메라를 x축 방향으로 평행하게 43cm을 움직여서 찍은 사진이다. 총 6개의 지점까지의 거리를 구해보았으며 상대오차는 위와 같이 나왔다. 물체들에 대해 평균적인 상대오차는 3.79%가 나왔다. 서랍장과 스피커의 경우 두 개의 거리가 있는 것을 볼 수 있는데 이는 스피커의 왼쪽까지의 거리와

오른쪽까지의 거리처럼 물체의 여러 가지 지점에 대해 측정한 거리이다.

### 4.2.2 카메라를 평행이동, 회전해 거리를 계산한 결과

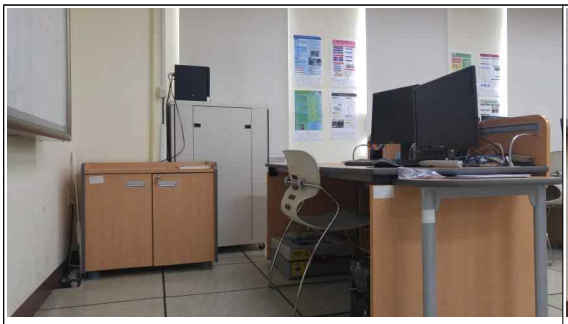


그림 23 왼쪽 사진

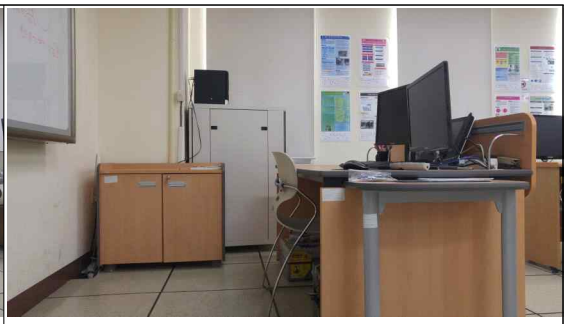


그림 24 오른쪽 사진

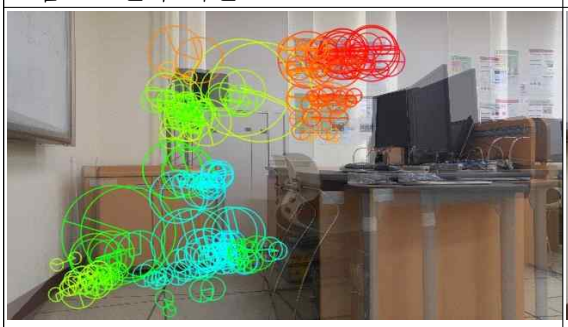


그림 25 RANSAC 3회 결과

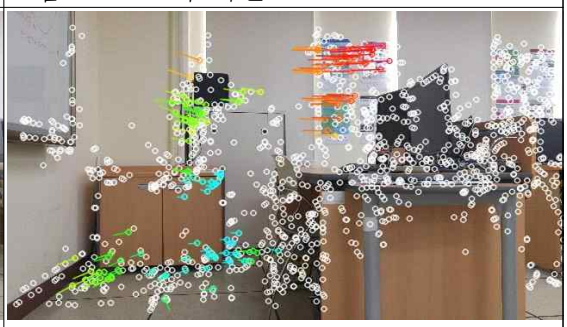


그림 26 오른쪽 사진의 특징점

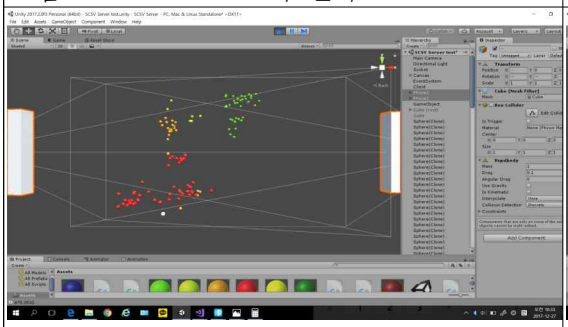


그림 27 카메라 위치에서 본 결과

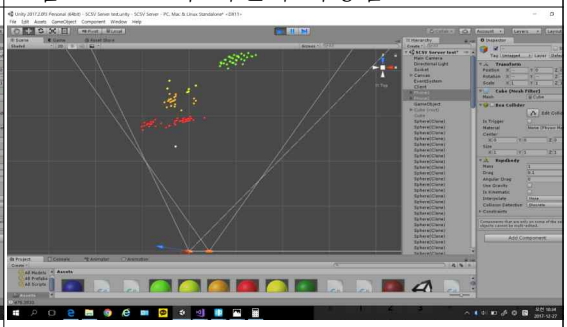


그림 28 위에서 직교투영한 결과

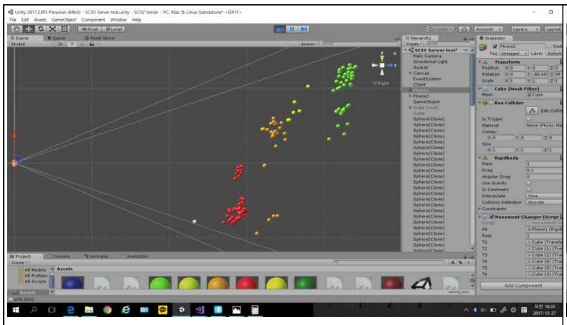


그림 29 옆에서 직교투영한 결과

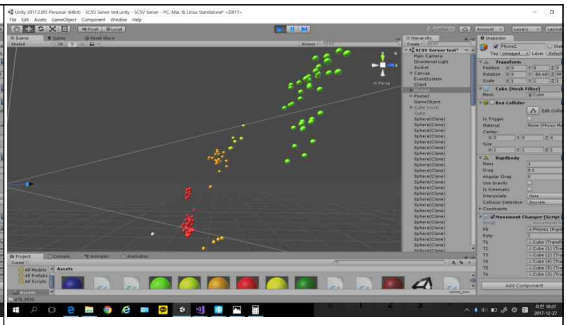


그림 30 오른쪽 뒤, 벽에 평행하게 본 결과

카메라 이동: 왼쪽 y축 9.335° 회전, 오른쪽 x축 43cm 이동

| 물체 | 실제 거리(cm) | | 측정 거리(cm) | | 상대오차(%) | |
|---|---|---|---|---|---|---|
| 서랍장 | 305 | 290 | 306 | 289 | 0.32 | 0.34 |
| 스피커 | 342 | 342 | 342 | 340 | 0 | 0.58 |
| 포스터 | 411 | 448 | 426 | 471 | 3.64 | 5.13 |

표 4 4.2.2의 거리 계산 결과와 실제 거리 비교

이 실험은 카메라를 x축 방향으로 평행하게 43cm을 움직이고 카메라를 회전까지 시켜 찍은 사진이다. 카메라가 회전한 각도를 구하기 위해 줄자를 이용해 길이를 재고 arcsin함수를 이용해 각도를 구하였다. 총 6개의 지점까지의 거리를 구해보았으며 상대오차는 위와 같이 나왔다. 물체들에 대해 평균적인 상대오차는 1.66%가 나왔다.

### 4.2.3 이전 실험 가시화 결과

- 4.2.1, 4.2.2는 가장 최근에 최대한 정확하게 실험한 결과이다. 추가적으로 이전 실험 결과를 공간상에 표현한 결과가 부록에 수록되어 있다.

## 4.3 고찰

먼저 RANSAC 알고리즘을 돌리는 횟수에 대해 실험을 해봤는데 5번 정도 돌리게 되면 더 많은 매치들을 찾아낼 수 있지만 그만큼 틀린 매치가 증가하기 때문에 3번 정도 돌릴 때 가장 안정적으로 틀린 매치 없이 잘 찾아낼 수 있음을 알아내었다. 그 다음으로 물체를 시각화해서 가상공간에 표현했더니 현실과 똑같이 표현되는 것을 볼 수 있었다. 실제로 물체와 동일한 모양으로 가상 공간에 표현되었음을 위의 사진을 통해 시각적으로 확인할 수 있다. 즉 이 방법이 물체를 잘 인지할 수 있음을 실험을 통해 확인하였다. 두 번째로 이것이 얼마나 정확하게 측정되었는가를 확인하기 위해 물체 사이의 거리와 측정 거리를 비교하였다. 실제로 비교를 해보니 각각의 상황에 대해 상대오차가 평균적으로 3.79%, 1.66% 정도로 매우 정확하게 공간상에 표현되는 것을 확인할 수 있었다. 그리고 오차 원인과 해결방법

은 아래와 같이 정리하였다.

#### 4.3.1 잘못된 매치

앞선 실험에서와 같이 매치가 잘못되는 경우가 존재하며, RANSAC으로 걸러지지 않는 매치들이 존재하기도 한다. 이는 거리 계산에 큰 오차를 발생시킨다. 해결을 위해서는 RANSAC 시행 횟수를 결정하는 방법이 필요하다.

#### 4.3.2 카메라 이동 정보

카메라의 이동한 거리와 회전한 정도가 정확하게 입력되어야 정확한 거리를 계산할 수 있다. 4.2.1, 4.2.2에서는 43±0.5cm의 거리를 이동했다. 카메라의 회전 또한 정확하게 입력되어야 하는데, 부록의 평행이동, 회전된 실험 결과에 오차가 큰 이유가 x축에 대한 회전이 고려되지 않았기 때문이다.

#### 4.3.3 카메라의 화각

카메라 화면의 점을 직선으로 만드는 과정에서 카메라의 화각 정보가 필요하다. 따라서 카메라의 화각이 정확히 입력되지 않는다면 오차가 발생하게 된다.

## 5. 핸드폰과 컴퓨터의 무선 통신을 이용해 정보를 송수신하기

### 5.1 연구 방법

C#으로 코딩해 핸드폰과 컴퓨터의 무선 통신을 할 수 있게 만들었다. 핸드폰에 만든 앱을 설치하고 실행시키게 되면 계속 가속도 센서와 자이로스코프 센서의 값이 컴퓨터로 송신되게 만들었다. 컴퓨터에서는 이 받은 센서값을 이용해 카메라가 움직인 거리를 적분을 이용해 계산하고 마찬가지로 얼마나 회전했는지도 적분을 이용해 계산하였다. 또한 추가적으로 사진 또한 통신으로 전송할 수 있게 코딩을 했으나 사진을 핸드폰에서 찍게 되면 전체 사진이 전송되는 것이 아니라 일부분만 전송되는 문제가 있었고 이는 현재 해결 중에 있다.

### 5.2 결론

아래의 그림과 같이 핸드폰을 회전시키면 그에 맞춰서 Unity공간상에 있는 저 박스가 똑같이 움직이는 것을 볼 수 있다. 또한 핸드폰이 이동하는 것 또한 Unity 공간상에서 잘 움직이는 것을 확인할 수 있었다. 하지만 생각보다 오차가 살짝 있었고 이를 해결하기 위해 칼만필터 같은 것을 이용해볼까 했지만 검색해본 결과 오직 저 두 개의 센서값만으로는 정확하게 보정할 수 없다는 글을 발견하였다. 상보필터 같은 것이 존재하기는 했지만 그것만으로는 유의미한 보정이 나오지 않는다는 글이 있었고 그래서

현재 어떤 식으로 이 센서값들을 보정해서 적분할 것인지 고민하고 있다. 만약 이것이 해결 된다면 항상 길이를 측정하지 않더라도 자연스럽게 카메라를 움직이면서 사진 두 장을 찍으면 바로 공간을 인식할 수 있게 된다. 즉 아주 편리하게 공간을 정확하게 인식할 수 있게 되는 것이다.

# 6. 결론 및 제언

## 6.1 결론

SIFT와 SURF를 사용해 특징점을 추출하고 특징점을 매칭해 스테레오 비전을 구현하려 하면 특징점들 사이에 잘못된 매치가 생기는 오류가 많이 나타난다는 것을 알 수 있었으며, 따라서 스테레오 비전이 정상적으로 작동하기 위해서는 잘못된 매치의 필터링이 중요하다는 것을 알 수 있었다. 잘못된 매치를 줄이기 위해 두 영상 기술자의 차이가 적은 매치들만 선별하고, 그 중 이동 방향이 같은 매치들만 선별하는 등 잘못된 매치를 줄이기 위한 여러 방법을 적용해 보았지만, 그럼에도 필터링 되지 않는 잘못된 매치가 존재했고, 이를 유발하는 몇 가지의 요인들을 찾아낼 수 있었다.

RANSAC을 이용해 틀린 매치를 필터링 하려는 시도는 나타난 특징점 매치들의 정확성은 매우 높았지만, 공간상의 특정 평면 근처에 위치하는 특징점 매치들만 나타나는 단점이 있었다. 이에 대해서 RANSAC을 중첩적으로 적용하여 대부분의 특징점 매치들을 구할 수 있었다.

정확하게 구한 특징점 매치들을 가지고 실제로 물체가 3차원 공간상에 어디에 위치하는 지를 벡터방정식을 풀어 구하였다. 그리고 이를 실제로 가시화시키기 위해 Unity 게임엔진을 이용해 Unity 공간상에 표현하였다.

그 다음, 실제로 얼마나 정확하게 표현되었는가를 측정하기 위해 실제 물체와의 거리와 측정된 물체와의 거리를 구하였고 상대오차를 구하였다. 상대오차는 실험 2개에 대해서 각각 평균적으로 3.79%, 1.66%로 매우 정확하게 공간상에 표현되었음을 알 수 있다.

핸드폰과 컴퓨터의 무선 통신에 성공하였으며 핸드폰의 가속도 센서, 자이로스코프 센서의 값을 컴퓨

터로 송신할 수 있었다. 또한 핸드폰에서 찍은 사진을 일부 송신할 수 있었다. 센서의 값에 오차가 있었지만 카메라의 위치 변화를 직접 입력한 실험 결과가 정확히 나온 것을 보면, 카메라의 위치 이동 오차만 해결할 수 있다면 핸드폰을 자유롭게 움직여서 측정해도 3차원 공간을 잘 파악할 수 있을 것이다.

### 6.2 제언

후속 연구는 틀린 매치의 수를 줄이는 방법, 특징점을 찾아내지 못한 공간에서도 거리를 파악해 모든 3차원 공간을 정확하게 파악하는 것과, 스마트폰의 센서와 카메라를 이용해 스마트폰의 위치와 회전 상태를 정확하게 파악할 수 있는 알고리즘의 개발, RANSAC을 몇 번 시행할지 결정하는 알고리즘이 개발이 필요하다. 세부적으로는 디바이스의 위치와 회전에 누적되는 오차를 줄일 수 있는 알고리즘, 위치, 회전을 카메라에 보이는 결과와 종합해 위치를 파악하는 알고리즘, RANSAC 결과에서 잘못된 매치가 있는지 알아내는 알고리즘이 필요할 것이며, 알고리즘을 최적화 해 수행시간을 줄이는 연구가 필요하다.

## 7. 참고 문헌


[1] Jonathan R. Seal, Donald G. Bailey, Gourab Sen Gupta. Depth Perception with a Single Camera.

[2] William Lovegrove. Single-camera stereo vision for obstacle detection in mobile robots.

[3] Clemens Holzmann, Matthias Hochgatterer. Measuring Distance with Mobile Phones Using Single-Camera Stereo Vision. 2012 32nd International Conference on Distributed Computing Systems Workshops, pp. 88 - 93.

[4] David G. Lowe. SIFT - The Scale Invariant Feature Transform Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.

[5] Herbert Bay1, Tinne Tuytelaars2, and Luc Van Gool12. SURF: Speeded Up Robust Features.

[6] Guoshen Yu, and Jean-Michel Morel, ASIFT: An Algorithm for Fully Affine Invariant Comparison, Image Processing On Line, 1, 2011.

[7] Marius Muja and David G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. Pattern Analysis and Machine Intelligence (PAMI), Vol. 36, 2014.

[8] Marius Muja and David G. Lowe. Fast Matching of Binary Features. Conference on Computer and Robot Vision (CRV) 2012.

[9] Marius Muja and David G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration", in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009.

[10] M. A. Fischler, R. C. Bolles. Random Sample Consensus. A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

[11] Serge Belongie & David Kriegman (2007). Explanation of Homography Estimation. Department of Computer Science and Engineering, University of California, San Diego.

# 부록

## 1. 관련 이론

### 1.1 SIFT

SIFT(Scale Invariant Feature Transform)란 크기와 회전에 불변하는 특징을 추출하는 알고리즘이다. SIFT는 David Lowe라는 사람이 2004년에 논문을 제출하면서 알려지게 되었다. SIFT가 나온

이후에 수많은 논문들이 나왔지만 현재까지 SIFT를 이용하지 않은 알고리즘 중에SIFT보다 뛰어난 성능을 가진 알고리즘은 없다. SIFT 알고리즘은 물체 인식이나 파노라마 영상을 만드는 등의 작업을 할 때 쓰인다.

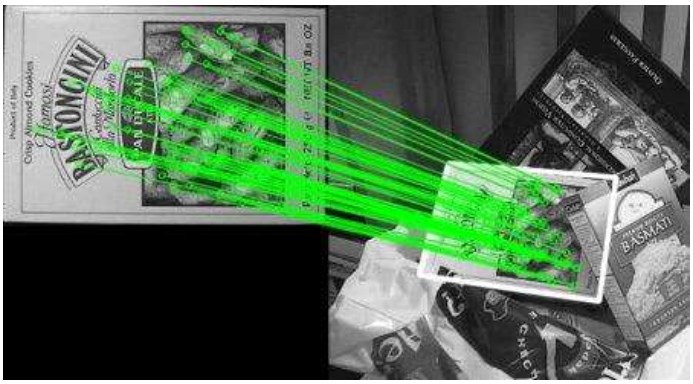
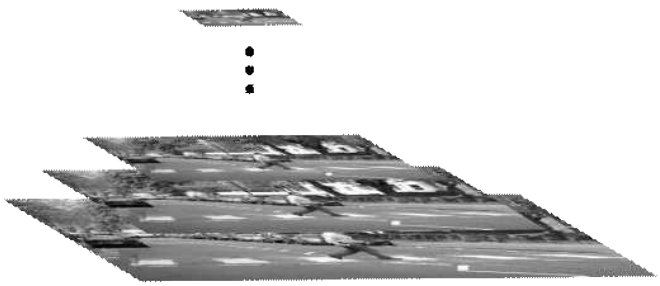



그림 1. SIFT로 물체를 인식한 사진

SIFT 알고리즘은 4가지 단계로 나눌 수 있다. 첫 번째 단계는 크기와 방향에 불변할 것이라고 추측되는 Interest Point(관심영역)을 검출한다. 보통 DOG(Difference of Gaussian)를 기반으로 이미지와 스케일에 대해서 코너성이 극대인 점을 찾는다. 스케일에 대해 불변한 특징점을 찾기 위해서 입력한 이미지를 단계적으로 축소시켜서 일련의 축소된 이미지를 먼저 생성한다. 즉 그림 2와 같은 image pyramid를 생성한다.




이 때

그림 2. 이미지 피라미드

각 스케일의 이미지마다 코너점(코너성이 로컬하게 극대이면서 임계값 이상)들을 찾는다. 그러면 각 스케일 이미지마다 코너점들이 검출되는데, 대부분의 경우 인접한 여러 영상 스케일에 걸쳐서 동일한 지점이 코너점으로 검출된다. 동일한 지점이 여러 영상 스케일에 걸쳐 코너점으로 검출된 경우, 그 중 스케일 축을 따라서도 코너성이 극대인 점을 찾으면 스케일에 불변한 특징점이 된다. SIFT에서 코너성이 극대인 점을 찾는 방법은 Laplacian 함수값을 사용한다. Laplacian은 이미지의 밝기 변화에 대한2차

미분값으로서 그림 3과 같이 계산된다.

$$L = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$
$$= I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)$$

그림 3. Laplacian 계산 방법

이 식을 설명하면 변화가 일정한 곳에서는 0에 가까운 값을 가지고 영역의 경계와 같이 밝기 변화가 급격한 곳에서는 높은 값(절대값)을 나타낸다. 그러나 실제 SIFT에서는 속도 문제로 인해 Laplacian을 직접 계산하지는 않고 DOG(Difference of Gaussian)를 이용하여 각 스케일 별 Laplacian을 근사적으로 계산한다. DOG는 입력영상에 Gaussian 필터를 점진적으로 적용하여 blurring 시킨 이미지들에서 인접 이미지들 간의 차 영상을 의미하며 이론적으로는LOG(Laplacian of Gaussian) 필터를 적용한 것과 거의 동일한 결과를 가지게 된다. 그리고 이렇게 얻어진 DOG 피라미드에서 극대 또는 극소점을 찾으면 SIFT의 특징점을 얻어낼 수 있다.

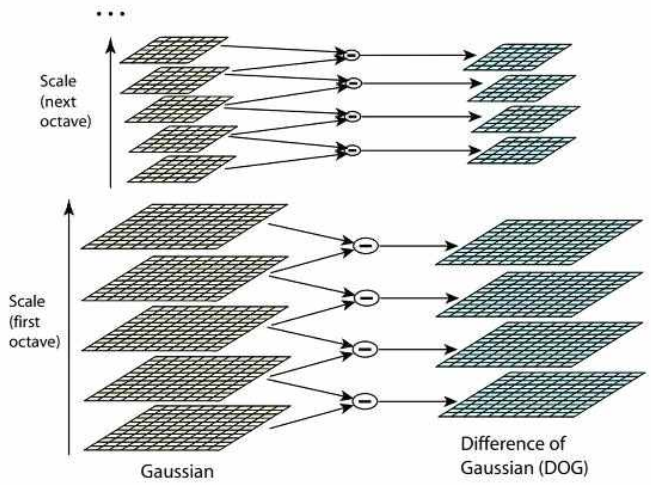



그림 4. DOG를 이용한 스케일별 Laplacian의 계산

두 번째 단계는 첫 번째 단계에서 추출한 특징점들에서 특징이 좋지 않은 점들을 제거 하는 단계이다. 방법은 다음과 같다. 먼저 keypoint의 후보로 만들어진 점 중 matching을 함에 있어 안정적이지 못한 점을 제거한다. 그 다음 특징점들을 정수영역(integer domain)이 아닌 연속공간에 위치시킨다. 그 다음 taylor 급수로 DOG를 전개한다. 이렇게 되면 이진영상이 연속영상에 근사되는데, 이렇게 근사된 영상은 이진 영상과 다른 극점을 가지게 되므로, 이의 차이를 가지고 keypoint 위치를 계속 조정해줄 수 있게 된다. 이러한 과정을 거치게 되면 이상한 keypoint들이 제거된다.

세 번째 단계는 keypoint 들의 방향성분을 결정하는 단계이다. 이는 그림 5와 같은 식으로 구할 수 있다.

$$m(x,y) = \sqrt{\left(L(x+1,y) - L(x-1,y)\right)^2 + \left(L(x,y+1) - L(x,y-1)\right)^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

그림 5. 방향성분 결정

이렇게 구한 다음 keypoint 주변으로 16*16 영역을 할당한 후 그 영역 이미지에 Gaussian Blurring을 적용한다. 그렇게 되면 keypoint 주변의 영역에 대한 Gradient의 방향과 영역을 결정할 수 있게 된다. Gradient의 방향과 크기를 구하면 Orientation Histogram을 형성한 후 가장 값이 큰 orientation을 keypoint의 orientation으로 결정하게 된다. 이렇게 되면 모든 keypoint에 대해 방향이 달라도 keypoint가 같음을 인식할 수 있게 된다.

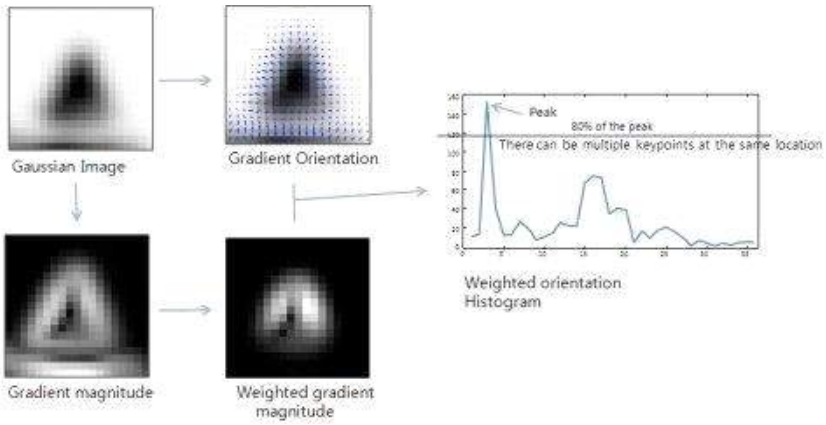



그림 6. keypoint의 orientation 정렬 과정

네 번째 단계는 각 keypoint마다 descriptor라는 것을 생성하는 단계이다. SIFT에서는Gaussian Weight Function을 이용해 descriptor를 생성하게 된다. 이 단계에서는 각 keypoint를 중심으로 16*16 pixel 영역을 할당하여 그림 7과 같이 image gradients를 구할 수 있다.

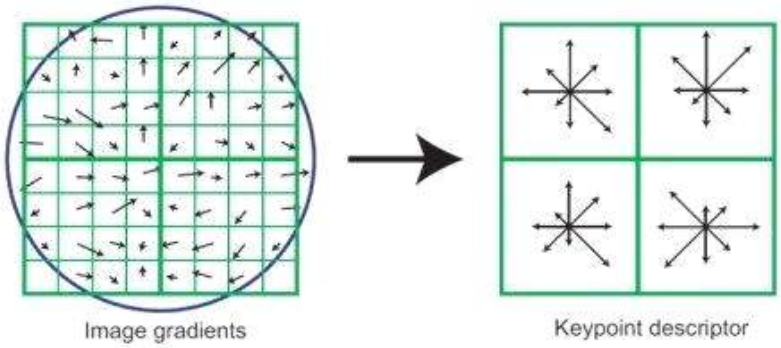



그림 7. keypoint의 descriptor 계산

그리고 저 gradients 값들을 4개의 묶음으로 나눈 뒤 그것들을 Gaussian Weight Function에 곱한다. 이제 나온 값들로 히스토그램을 그려주게 되는데 이 때 두 개의 descriptor를 비교할 때 방향에 무관하게 비교하기 위해 아까 전에 구한 Orientation들을 고려해서 히스토그램을 그린다. 이렇게 구하면 최종적으로는 각 keypoint에 대해128개의 descriptor값(실수형 데이터)들이 저장된다.

## 1.2 SURF

2004년에 SIFT의 등장으로 keypoint와 descriptor에 대한 많은 연구가 있었고 SIFT의 최대 단점인 많은 계산량을 개선시킨 알고리즘이 바로 SURF이다. SURF는 SIFT 못지않게 성능을 유지하며 계산 속도를 올린 알고리즘이다. 그리고 이 알고리즘은 3가지 단계로 정리할 수 있다.

첫 번째 단계는 Integral Image(적분 이미지)를 구하는 것이다. SURF는 SIFT와 다르게 적분 영상을 이용해 관심영역과 특징점을 찾는다. 적분이미지란 아래와 같이 계산하는 것으로 부분합을 이용하면 빠르게 계산할 수 있다.

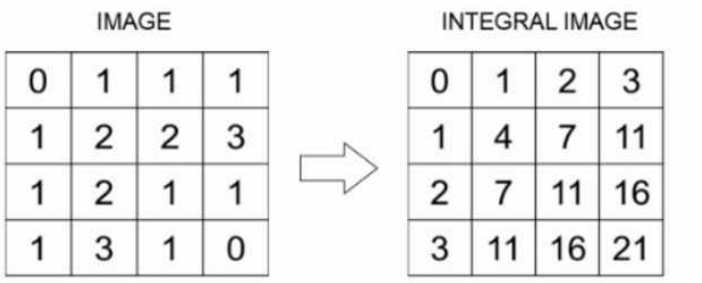



그림 8. 적분 이미지

두 번째 단계는 적분 이미지에 고속 헤시안 검출(Fast Hessian Detector)를 이용해 특징점들을 추출하는 단계이다. 그림 9와 같이 박스 필터를 사용해 적분이미지 계산으로 영역의 넓이를 빠르게 구하고 구한

넓이로 헤시안 행렬을 계산해 특징점을 찾는다. 또한 SURF도 SIFT와 마찬가지로 이미지 피라미드와 같은 것을 이용해 다양한 스케일에 대해서 이 작업을 반복하고 결국 특징점들을 스케일에 상관없이 찾게 된다.

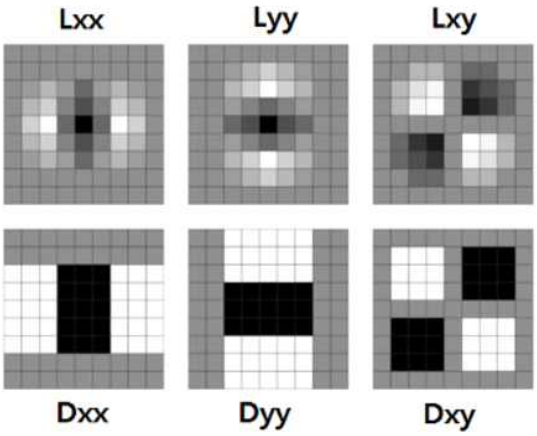



그림 9. 근사화한 박스 필터

세 번째는 이렇게 구한 특징점들을 가지고 descriptor를 계산하는 일이다. 먼저 각 특징점에 대해 6s(scale)의 원 안에 있는 이웃들에 관해 x, y 방향의 Haar wavelet response를 계산한다.

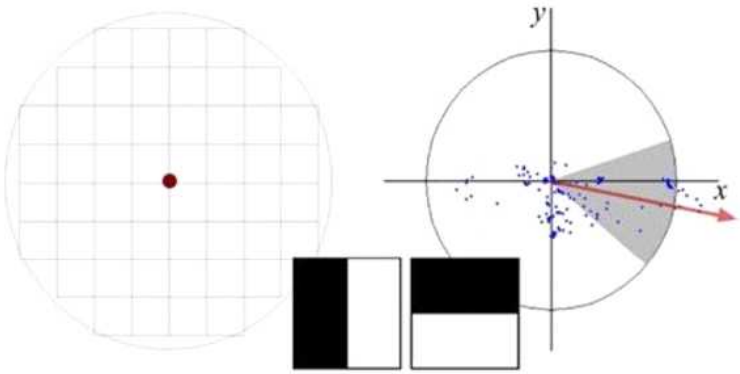



그림 10. Haar wavelet response

그림 10에서 오른쪽 원 안에 있는 파란 점이 response 값들의 분포이다. 그리고 빨간색 화살표는 부채꼴 모양의 영역 안에 있는 response의 합을 벡터로 표현한 것이다. 이렇게 60도 크기의 window를 슬라이딩 하면서 x, y 축의 response의 합을 계산해서 벡터를 생성시키고 벡터 길이가 가장 긴 것이 orientation으로 할당된다. 이 때 Haar wavelet response를 이용하기 때문에 적분 이미지의 장점 즉 빠른 계산을 이용할 수 있다. Orientation을 구했으니 이제는Descriptor를 계산할 차례이다. 이는 관심점 주위 20scale 크기의 window를 구성해 이미지를 회전에 invariant 하기 위해 회전시킨 후 계산한

다. 이 때 앞에서 구한 orientation을 활용해서 구한다. 또한 descriptor window를 4*4로 나누고 각 구역은 5*5 Haar wavelet로 구성된다. 결국 이를 통해 각 특징점들마다 descriptor를 추출하게 되었다.

## 1.3 ASIFT

Asift(An Algorithm for Fully Affine Invariant Comparison)는 SIFT를 응용한 알고리즘이다. 기존의 SIFT는 밑의 사진과 같이 회전하고 기울어짐이 있는 물체에 대해서는 특징점들을 잘 매칭시키지 못한다는 단점이 있었다. 그래서 이것을 개선한 것이 바로 ASIFT이다.

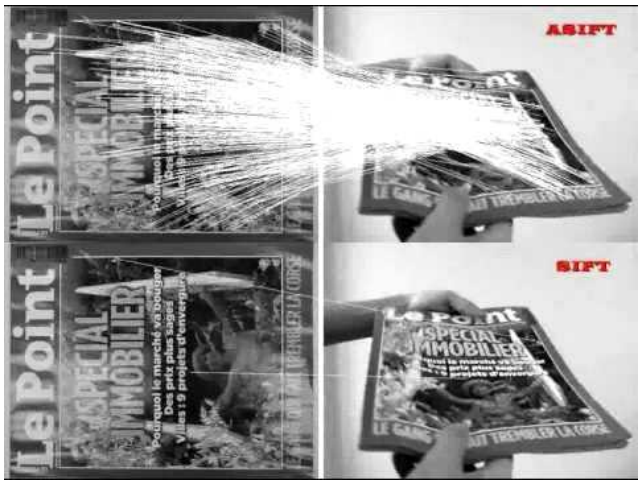



그림 11. SIFT와 ASIFT의 비교

ASIFT의 원리는 여러 가지 각도에 대해 전부SIFT를 효과적으로 돌려가지고 각 특징점에 대해 descriptor를 잘 구하는 것이다.

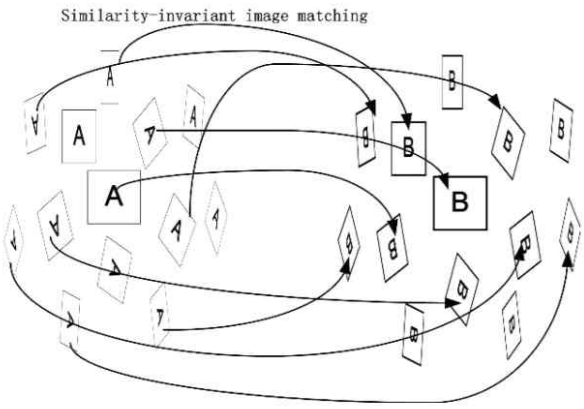



Figure 5: ASIFT algorithm overview.

그림 12. ASIFT의 원리

즉 그림 12과 같이 이미지 A와 B를 잘 회전시킨 다음 각각에 대해 SIFT를 돌려서 image matching을 시킨다는 것이다. 이를 이용하면 적당한 회전이 있어도 SIFT와는 달리 이미지들을 잘 매칭시킬 수 있다. 또한 ASIFT는 거의 대부분의 상황에서 사용되는 RANSAC 알고리즘을 사용하지 않고 ORSA(Optimized Random Sampling Algorithm)를 사용한다.

### 1.4 BF Matching

BF Matching은 Brute Force Matching으로 모든 쌍의 특징점들에 대해 그 특징점들이 얼마나 차이 나는지(distance)를 계산한 다음 distance가 작으면 매칭을 시켜주는 하나의 고전적인 매칭 방법이다. 그러나 BF Matching은 느리기 때문에 여러 가지 Matching법들이 생겨났다.

### 1.5 Flann based matching

Flann(Fast Library for Approximate Nearest Neighbors)는 Opencv의 라이브러리다. 이 라이브러리에는 가장 가까운 이웃 검색을 빠르게 할 수 있는 알고리즘들이 들어있다. 즉 가장 가까운 k개의 점을 찾는 knn algorithm이 들어간다. 즉 이를 이용해 기존의 BF Match와는 다르게 훨씬 더 빠르게 distance가 작은 매치들을 찾을 수 있게 되었다.

### 1.6 RANSAC algorithm

RANSAC 알고리즘은 거의 모든 영상처리나 컴퓨터 비전에서 사용되는 알고리즘이라고 볼 수 있다.

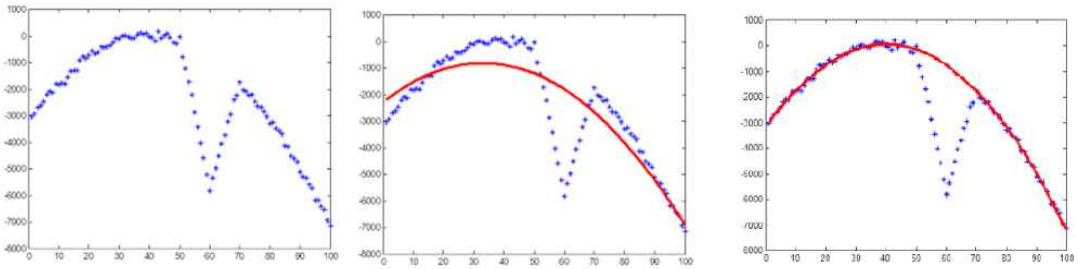


그림 13. 데이터, 최소자승법, RANSAC 알고리즘의 비교

첫 번째 그래프가 현재 우리가 가진 데이터고 현재 우리는 이 데이터에 맞는 추세선을 그리려고 한다고 가정해보자. 두 번째 그래프는 오차의 제곱의 합이 작도록 구하는 최소자승법으로 구한 추세선이고 세 번째 그래프는 RANSAC를 이용해 구한 추세선이다. 즉 RANSAC가 가진 장점은 데이터에 outlier(벗어난 데이터)가 있을 때 그것들을 효과적으로 배제하면서 추세선 같은 것을 구할 수 있다는 것이다.

저 상황에서 RANSAC 알고리즘은 다음과 같다. 먼저 몇 개의 표본 데이터를 추출한 다음에 그것들을 모두 지나는 선을 그린다. (N개의 점을 선택했으면 N-1차 다항식으로 선이 표현이 될 것이다.)

그런 다음 그 선과 거리가 T이하인 점의 개수를 센다. 그리고 이 작업을 여러 번 반복하는데 이 때 우리는 아까 전에 세었던 점의 개수가 가장 많았던 선을 기록해 놓고 그 선을 추세선으로 가정한다. 이것이 RANSAC 알고리즘의 대략적인 내용이다. 이 방법이 성공하기 위해서는 적어도 한 번 모든 표본 데이터가 inlier(올바른 데이터)였던 적이 있어야 한다. 또한 T값과 같은 값을 잘 설정해야지만 잘 동작한다는 단점이 있다. 다른 특징으로는 우리가 inlier의 비율을 어느 정도 알고 있다면 이 정보를 이용해 우리가 몇 번 표본 데이터를 추출할 지도 계산할 수 있다. 결론적으로 RANSAC 알고리즘을 이용하면 outlier가 있어도 inlier들을 잘 추출할 수 있다는 것이다.

## 1.7 Homography

Homography는 평면물체의 2D 이미지 변환 관계를 설명할 수 있는 가장 일반적인 모델이다. Homography는 homogeneous 좌표계에서 정의되며 그 일반식은 아래와 같다.

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

그림 14. Homography의 일반식

Homography는 자유도가 8이기 때문에 homography를 결정하기 위해서는 최소 4개의 매칭쌍이 있어야 한다. Homography를 실제로 구하는 과정은 수학적으로 매우 복잡하기 때문에 여기서는 논문 제목으로 대체해야 될 것 같다. 자세한 내용은 “Homography Estimation 2009”의 논문에 있다. 그래도 Homography를 간단히 설명하자면 2D평면에서 임의의 사각형을 임의의 사각형으로 매핑시킬 수 있는 변환이 Homography이다. 아래 그림과 같은 느낌으로 이해하면 편하다.

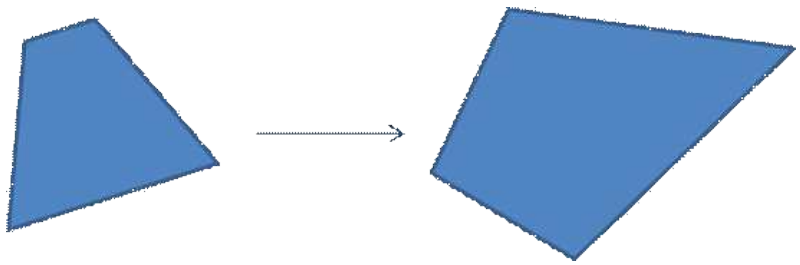


그림 15. Homography의 단순한 개념

## 2. 2.2의 추가 실험 결과

### 2.1 실험 4

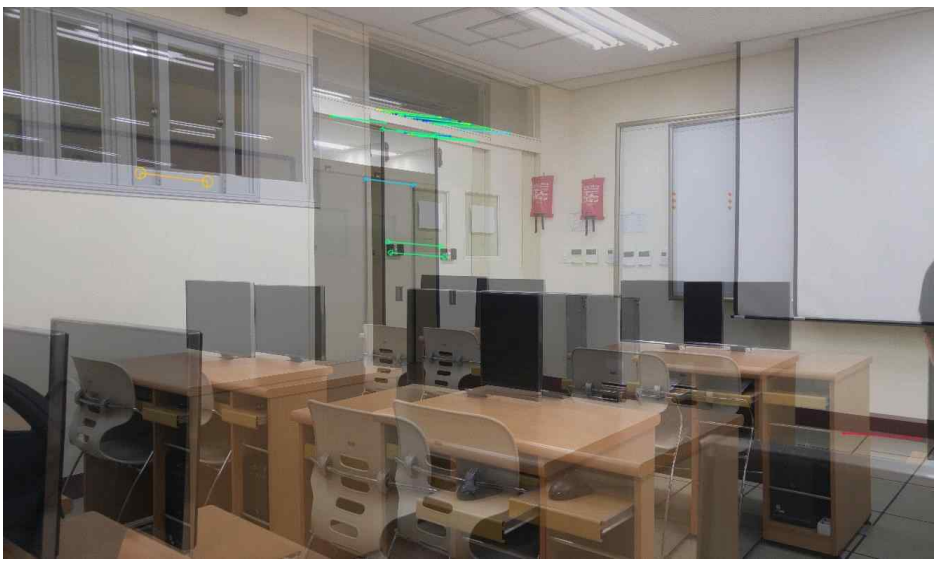


그림 16. 실험 4 SIFT 이용 결과

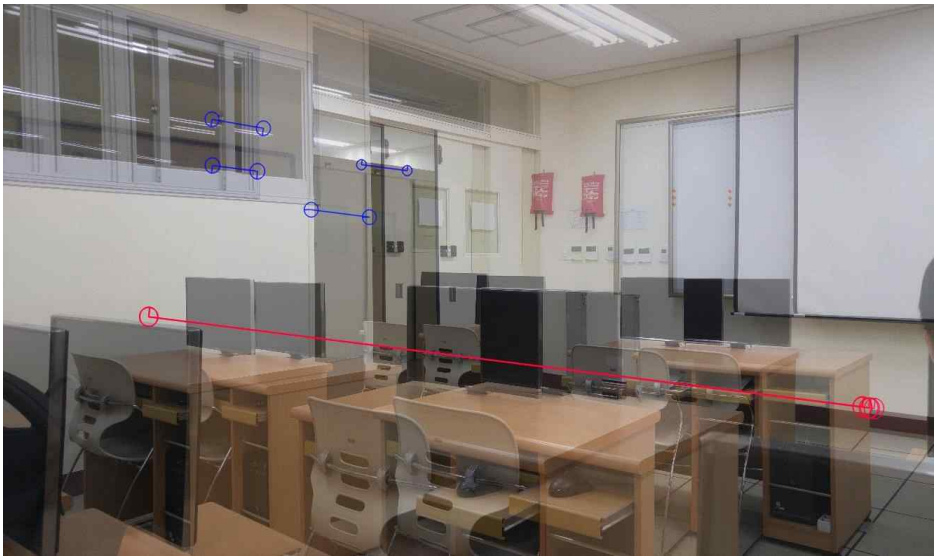


그림 16. 실험 4 SURF 이용 결과

SIFT는 벽 쪽에서만 특징점들을 찾았다. 그리고 문 위쪽에 특히 특징점들이 많이 분포하는데, 특징점의 크기가 작고, 형태의 모서리 부분이라 잘못된 매치들이 여러 개 있다. 오른쪽 하단에는 빨간색으로 매치가 표시되었는데, 정확한 매치로 보인다.

SURF는 모두 잘못된 매치이다. 문 근처의 파란색으로 표시된 매치들도 실제 매치되어야 하는 위치에서 조금씩 떨어져 있다.

## 2.2 실험 5

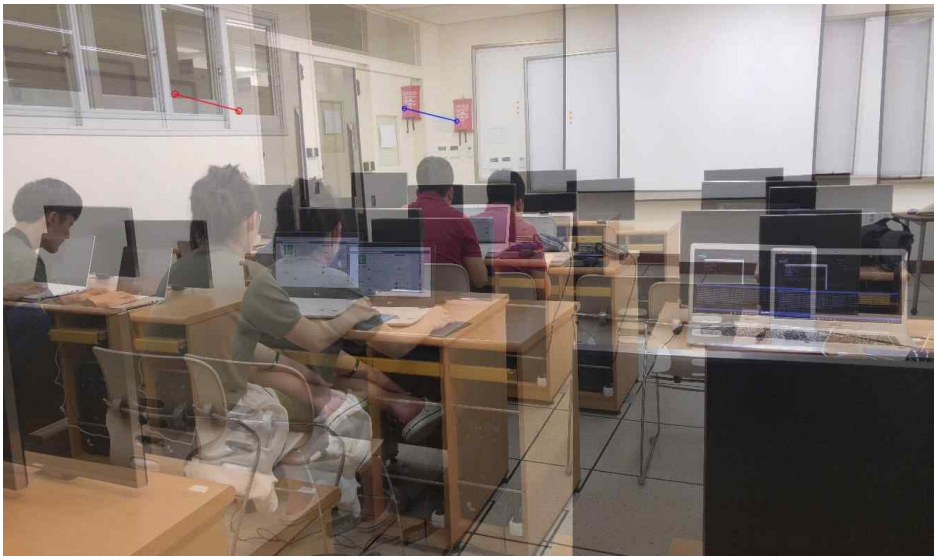


그림 25. 실험 5 SIFT 이용 결과

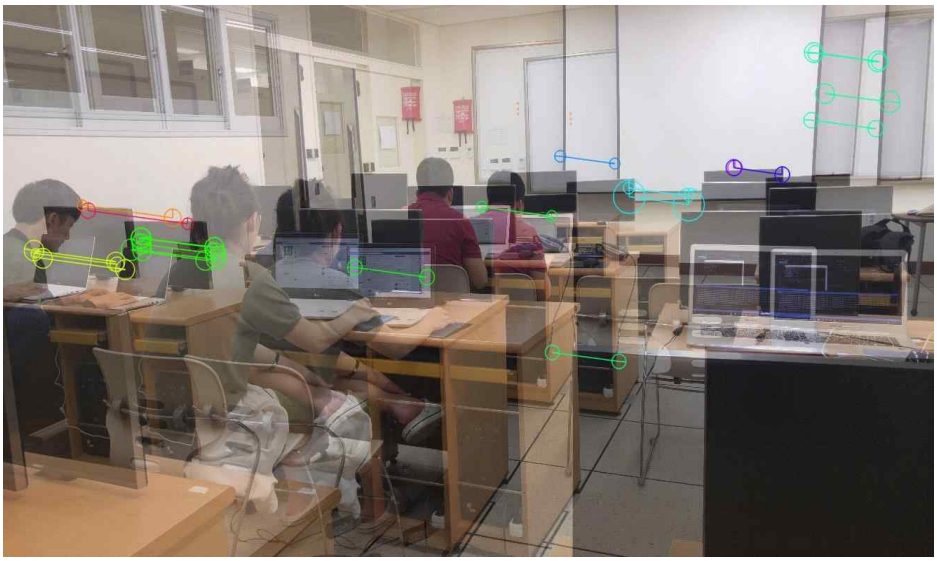
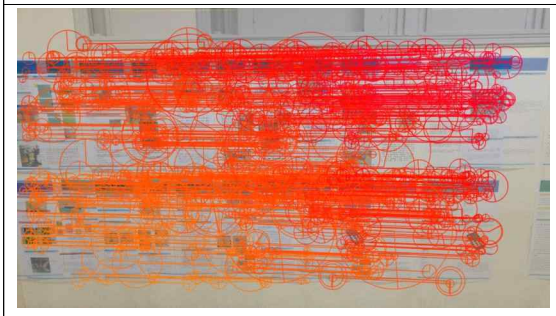
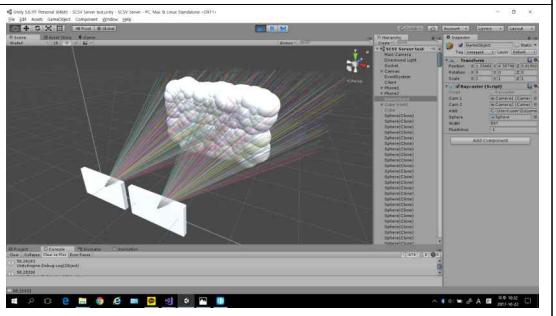
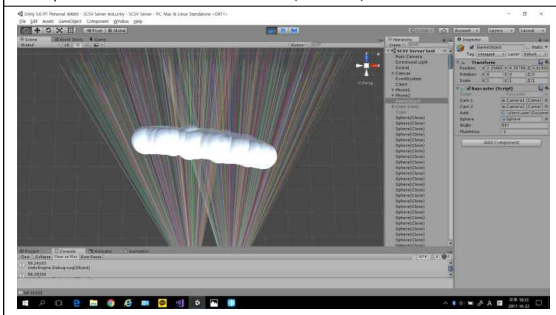
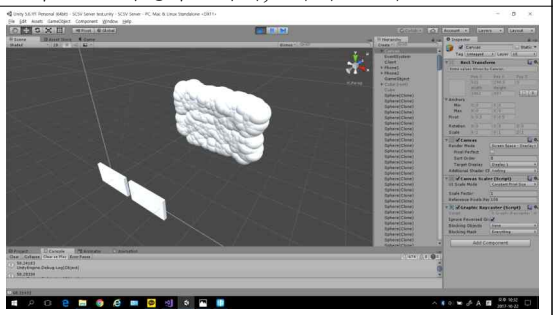
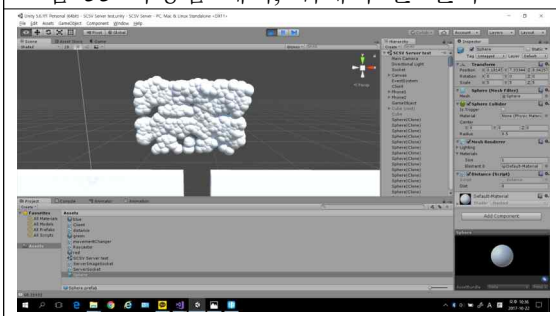
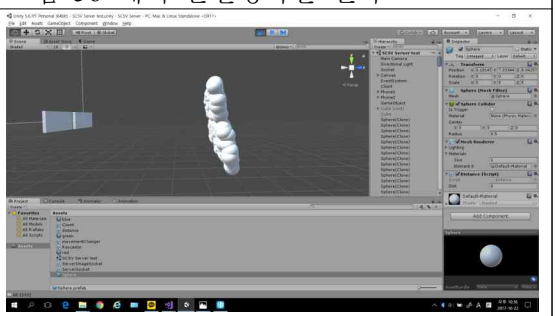


그림 26. 실험 5 SURF 이용 결과

SIFT는 매칭은 정확하지만 특징점의 수가 너무 적다. SURF는 형태의 모서리에 위치해 잘못 매칭되는 경우가 많이 나타났다.

## 3. 4.2.3 추가 실험

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 51 왼쪽 사진</td><td>그림 52 오른쪽 사진</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 53 RANSAC 2회 결과</td><td>그림 54 특징점 레이, 배치 결과</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 55 특징점 레이, 위에서 본 결과</td><td>그림 56 레이 불활성화한 결과</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 57 정면, 구체 크기 감소</td><td>그림 58 측면에서 본 경우</td></tr>
</table>

왼쪽 아래 부분이 disparity가 더 크고, 3d공간에 배치된 상황도 이에 따른다. 아래쪽을 보며 촬영했고, 약간 왼쪽 방향으로 촬영했다.

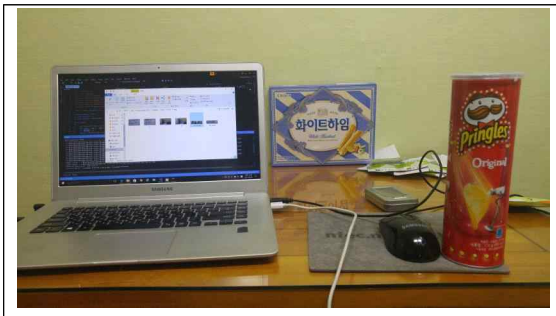



그림 59 왼쪽 사진

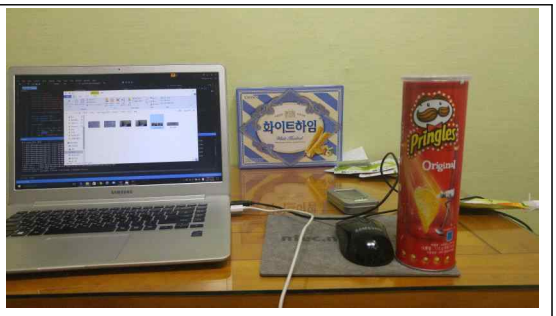



그림 60 오른쪽 사진

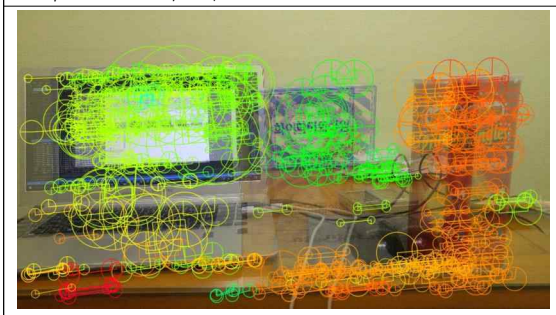


그림 61 RANSAC 5회 결과

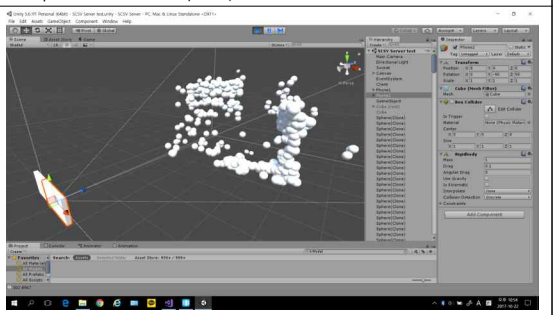


그림 62 가상 공간 배치 결과

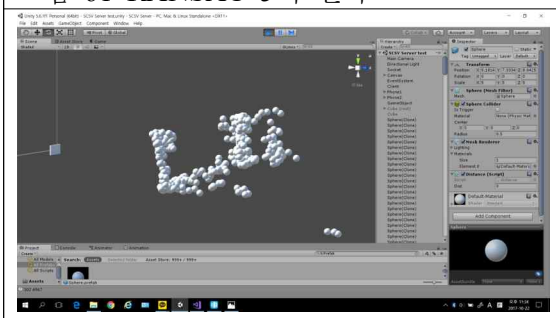


그림 63 측면 직교 투영 결과

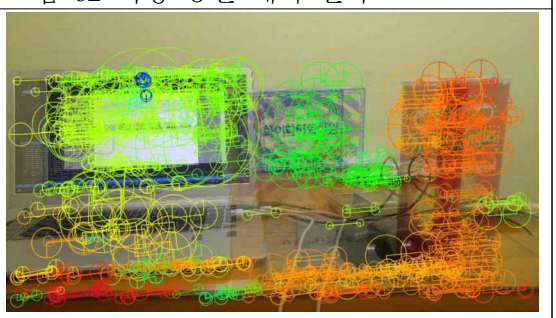


그림 64 RANSAC 7회 결과

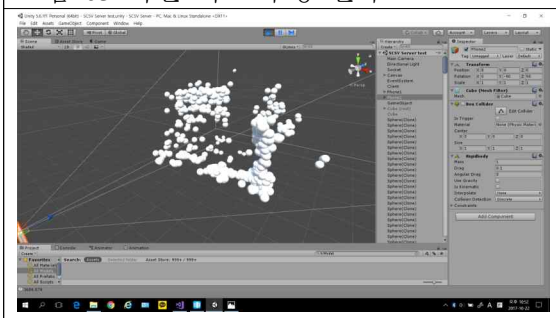


그림 65 가상 공간 배치 결과

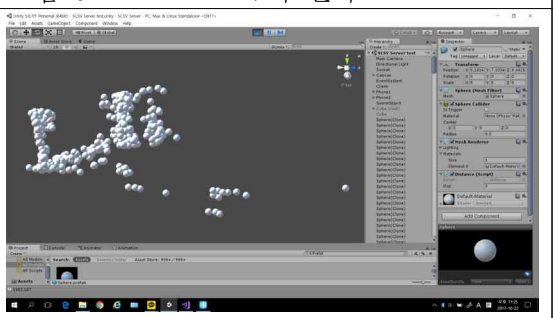
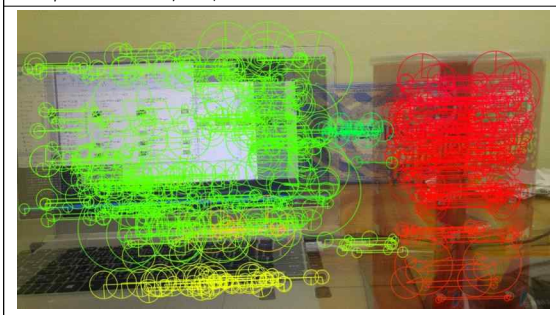
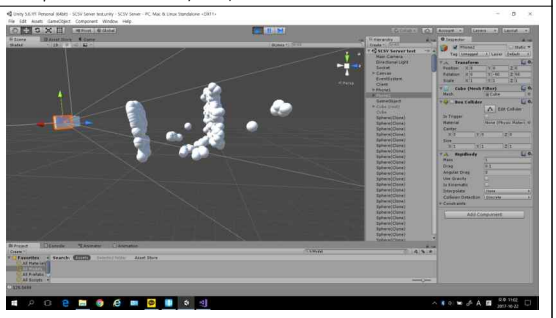
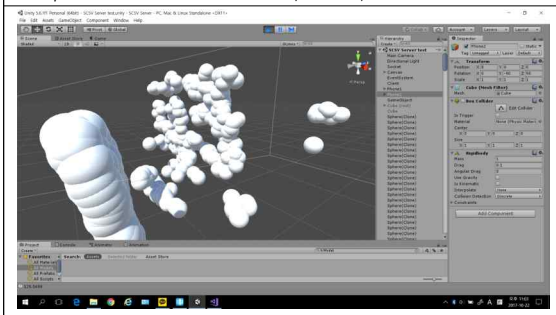
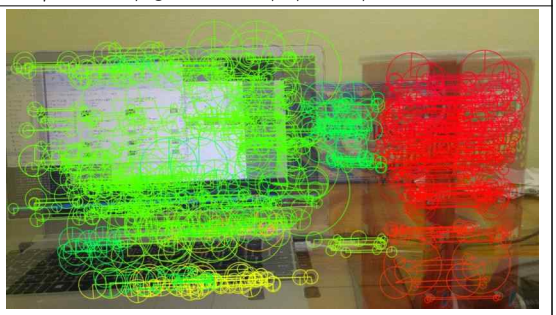
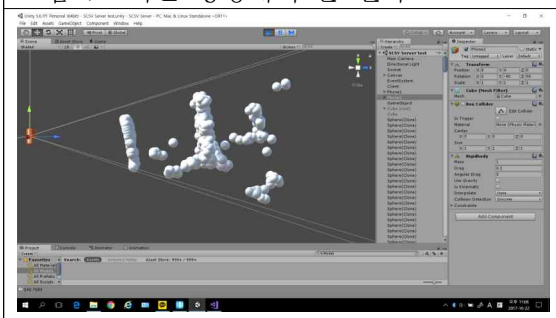
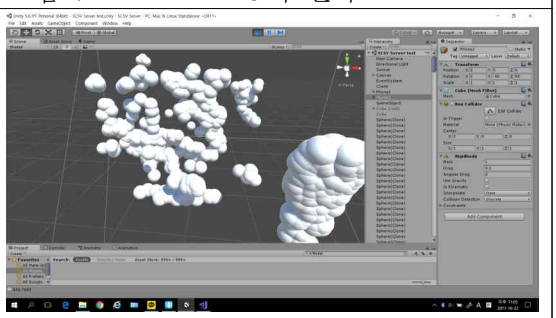


그림 66 측면 직교 투영 결과

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 67 왼쪽 사진</td><td>그림 68 오른쪽 사진</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 69 RANSAC 3회 결과</td><td>그림 70 가상 공간 배치 결과</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 71 다른 방향에서 본 결과</td><td>그림 72 RANSAC 5회 결과</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>그림 73 가상 공간 배치, 직교 투영</td><td>그림 74 정면에서 본 결과</td></tr>
</table>

5개의 평면으로 RANSAC 필터링을 하니 더 많은 점들을 나타낼 수 있지만 잘못된 매치들이 나타난다.

그림 82 왼쪽 사진

그림 83 오른쪽 사진

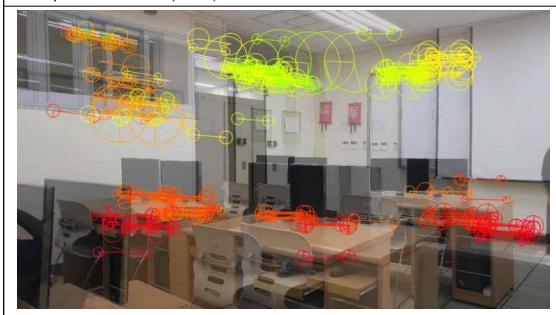


그림 84 RANSAC 3회 결과

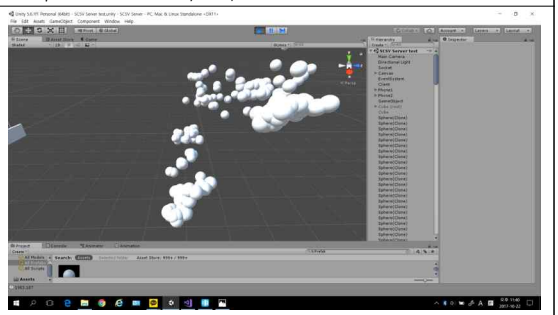


그림 85 측면

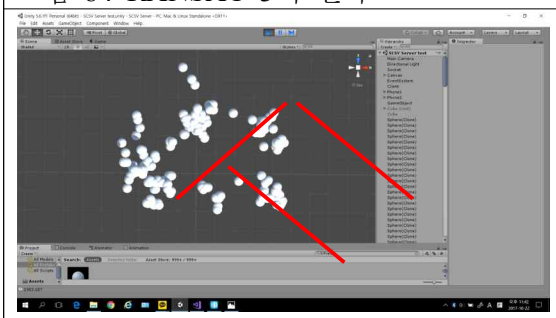


그림 86 위에서 직교 투영한 결과

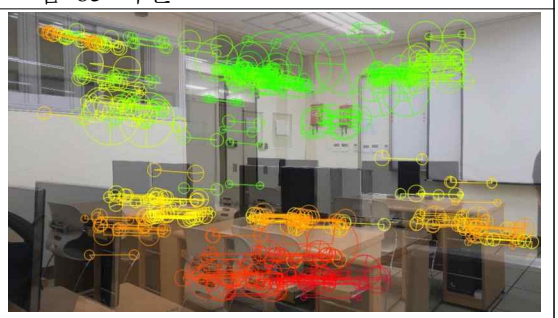


그림 87 RANSAC 6회 결과

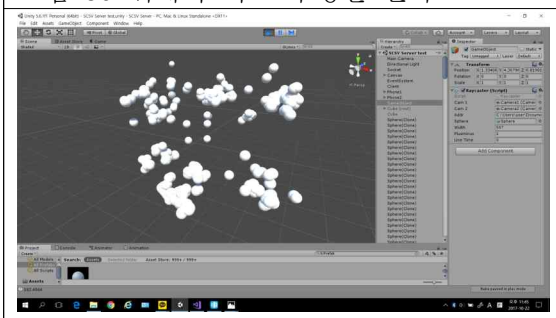


그림 88 조금 오른쪽에서 본 결과

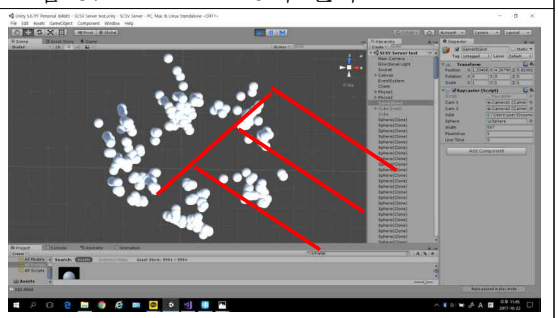


그림 89 위에서 직교 투영한 결과

그림 90 왼쪽 사진

그림 91 오른쪽 사진

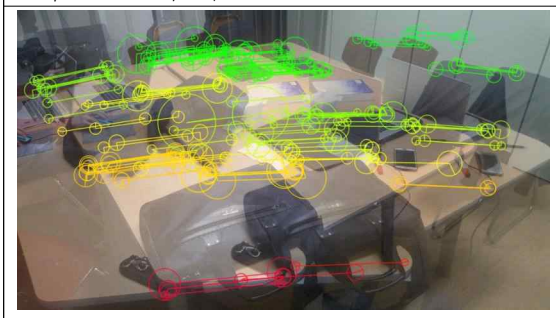


그림 92 RANSAC 4회 결과

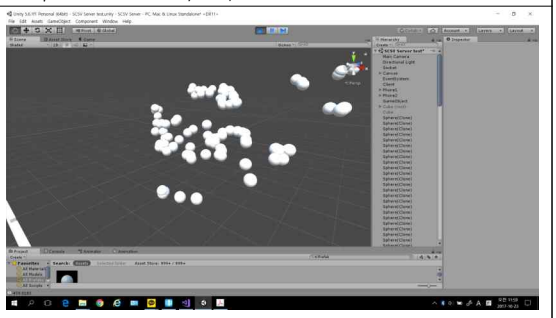


그림 93 공간 배치 결과

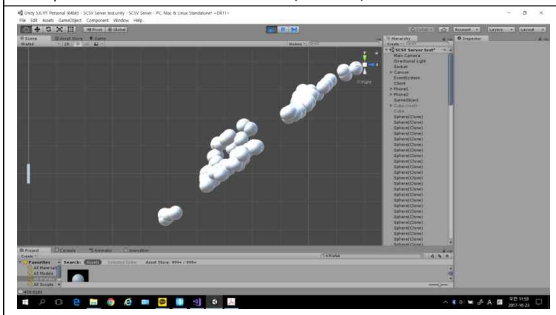


그림 94 측면 직교 투영

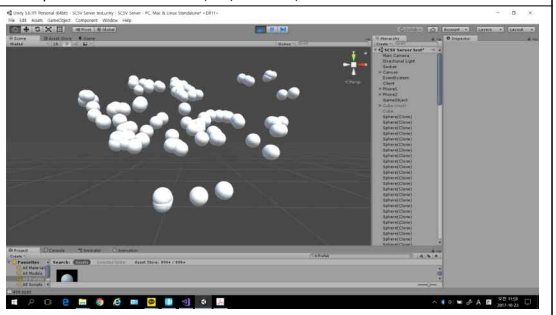


그림 95 카메라 위치에서 본 결과

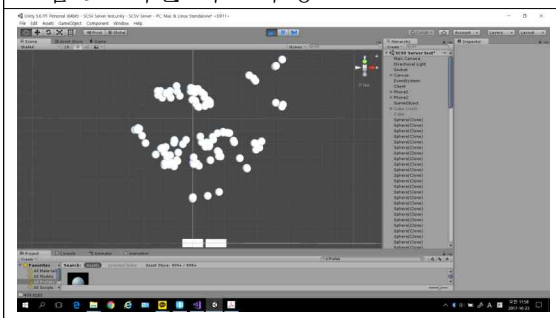


그림 96 위에서 직교 투영

그림 97 왼쪽 사진

그림 98 오른쪽 사진

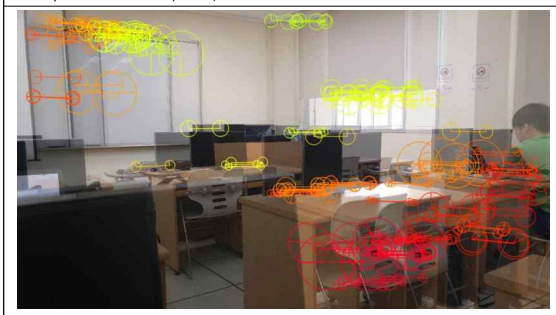


그림 99 RANSAC 3회 결과

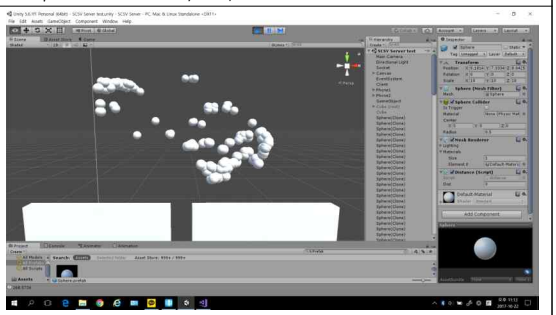


그림 100 카메라 위치에서 본 결과

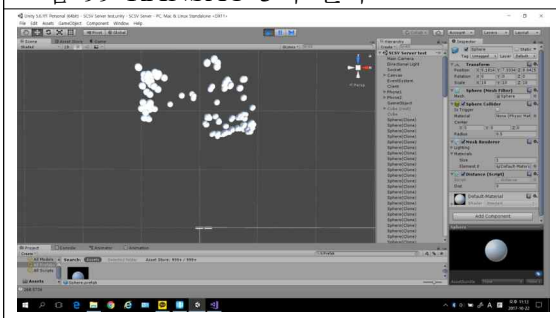


그림 101 위에서 본 결과

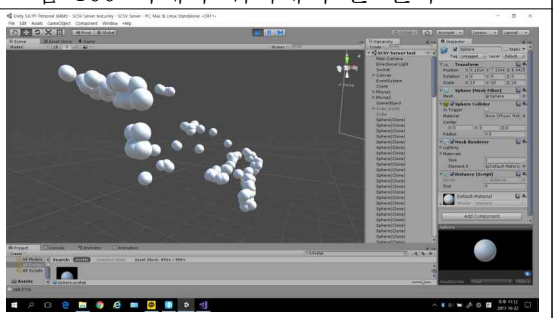


그림 102 왼쪽에서 본 결과

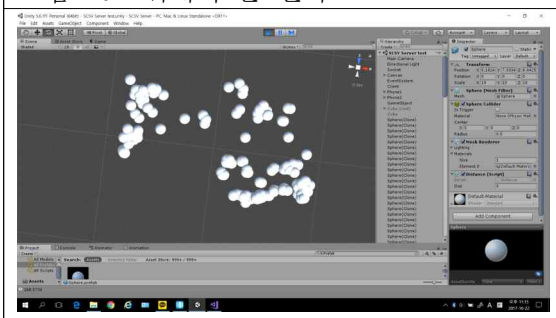


그림 103 위에서 직교 투영한 결과

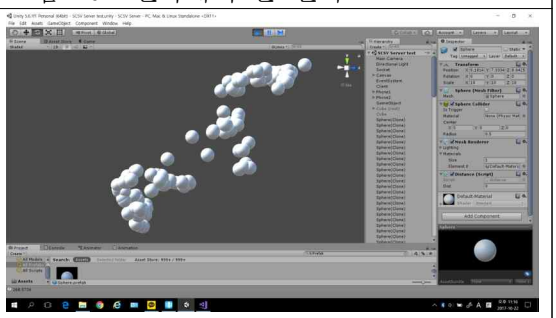


그림 104 오른쪽 측면 직교 투영

그림 105 왼쪽 사진

그림 106 오른쪽 사진

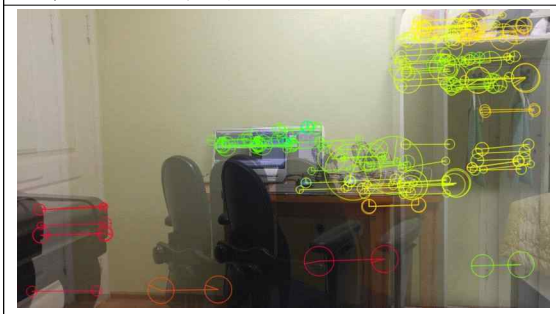


그림 107 RANSAC 6회 결과

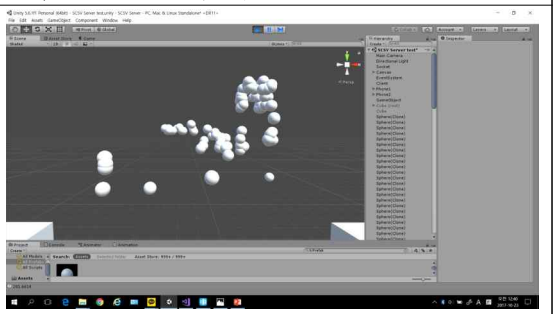


그림 108 카메라 위치에서 본 결과

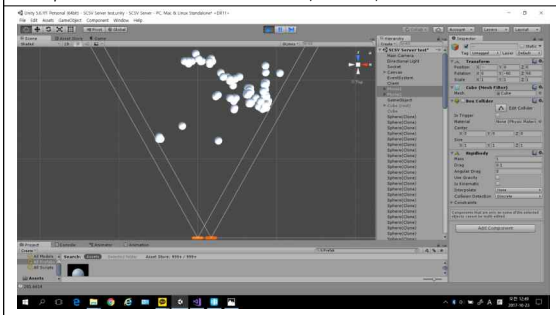


그림 109 위에서 직교 투영한 결과

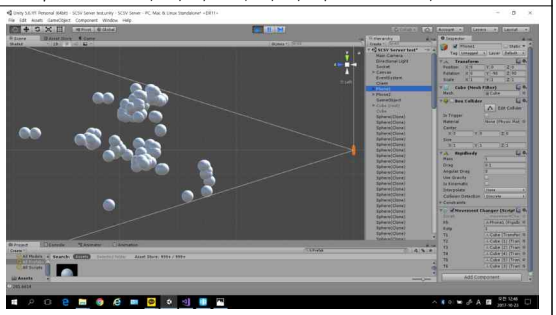


그림 110 왼쪽에서 직교 투영한 결과

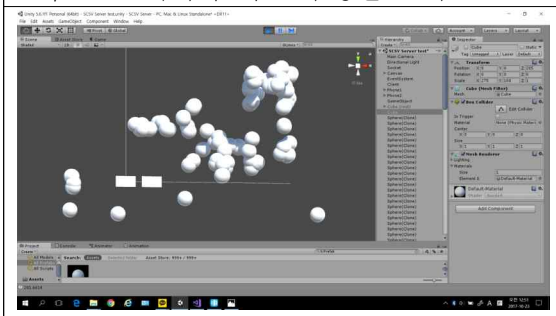


그림 111 정면에서 직교 투영한 결과

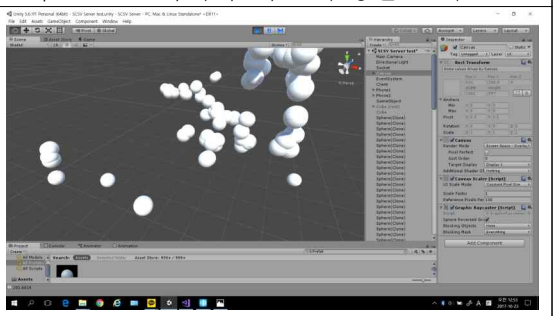


그림 112 다른 방향에서 본 결과

그림 113
실제 옷장

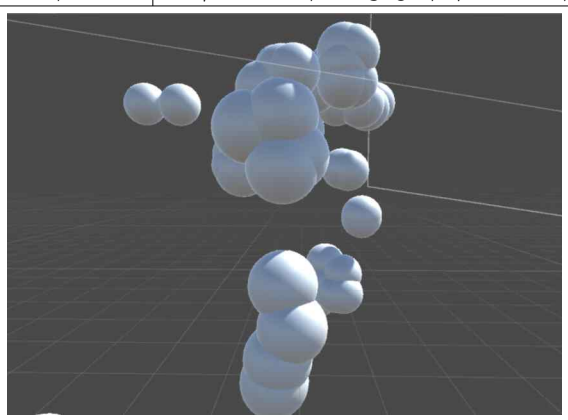


그림 114
인식된
옷장

카메라 이동 x: 17cm
피아노 실제: 118~127cm  측정: 137~139cm  재측정: 116~124cm
노트북 실제: 194~200cm  측정: 221~233, 272, 281cm  재측정: 182~192, 224, 231cm
옷걸이 부분 실제: 173cm  측정: 190~196cm  재측정: 162~168cm
의자 등 실제: 125cm  측정: 148cm  재측정: 124cm
의자 팔걸이 실제: 100cm  측정: 128cm  재측정: 108cm
책상 실제: 145cm  측정: 169~182cm  재측정: 142~151cm
옷장 실제: 176cm  측정: 191~196cm  재측정: 164~166cm
선반 실제: 173cm  측정: 188~197, 210cm  재측정: 161~172, 182cm
벽 실제: 228cm  측정: 238cm  재측정: 205cm

모든 값들이 실제보다 작게 나왔다. 실제 카메라의 시야각과 가상 공간의 카메라의 시야각이 차이가 났기 때문이라고 생각되어 시야각을 조절한 후 재측정함.

그림 115 왼쪽 사진

그림 116 오른쪽 사진

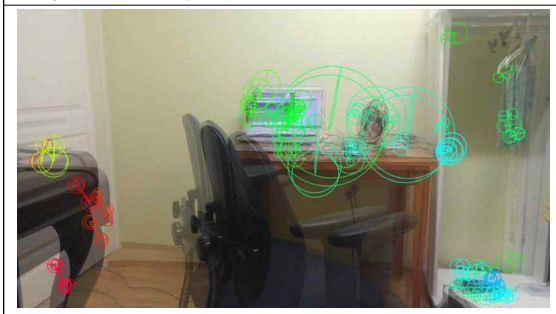


그림 117 RANSAC 4회 결과

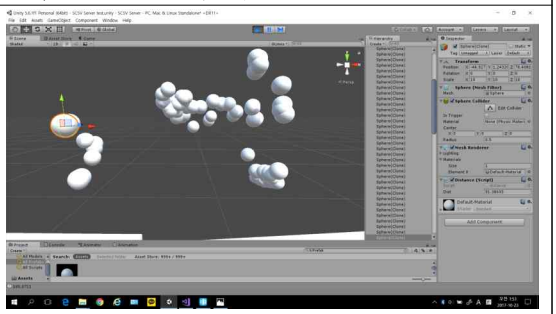


그림 118 카메라 위치에서 본 결과

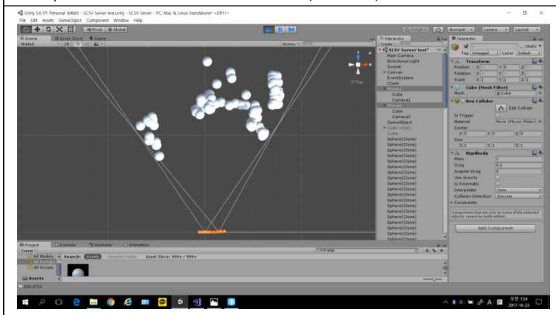


그림 119 위에서 직교 투영 결과

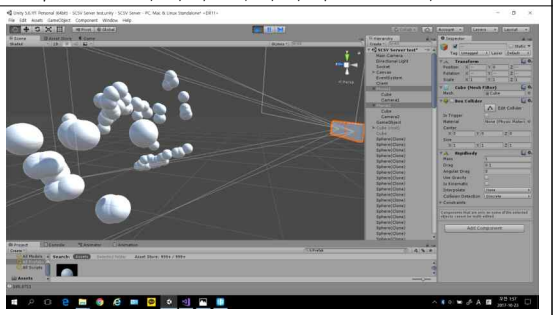


그림 120 왼쪽 앞에서 본 경우

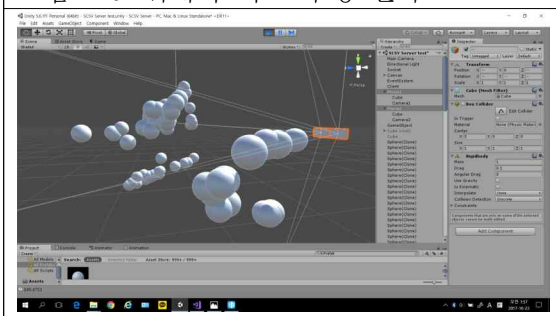


그림 121 왼쪽 뒤에서 본 경우

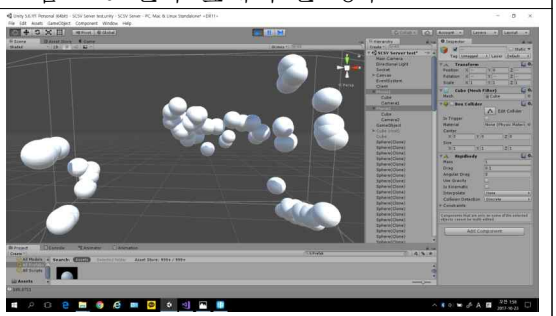


그림 122 정면에서 본 경우

카메라 이동 x: 14.5cm z: 0.4cm

카메라 회전 y축 -5.355° -> z축 회전 고려 필요했을 듯

피아노 실제: 118~127cm 측정: 91, 106~128, 123~131cm

노트북 실제: 194~200cm 측정: 123~142, 175cm

옷걸이 부분 실제: 173cm 측정: 117~120cm

오른쪽 아래 실제: 154~225cm 측정: 118~183(연속)cm