Pusan National University
Computer Science and Engineering
Technical Report 2023-10

LiDAR 센서 데이터 기반 3D 공간 구축

201824616 홍주혁

202055640 남예진

202055647 우현우

지도교수 김원석

목 차

1. 서론 1

1.1. 연구 배경 1

1.2. 연구 목표 1

1.3. 기대 효과 2

2. 연구 내용 2

2.1. 포인트 클라우드 데이터 수집 2

2.1.1. 장치 구성도 2

2.1.2. 데이터 수집 장치 및 테스트 공간 3

2.1.3. ROS 4

2.1.4. python 5

2.2. Rectangle 추출 프로그램 7

2.2.1. 전반 구조 7

2.2.2. 파일 입력 및 전처리 단계 7

2.2.3. 포인트 그룹화 단계 8

2.2.4. 포인트 그룹 Rectangle 변환 단계 11

2.2.5. Rectangle 합병 단계 12

2.3. 바닥, 벽, 장애물 구분 및 3차원 공간 구현 14

3. 연구 결과 분석 및 평가 18

3.1. 생성과정 및 결과 18

3.2. 결과 분석 및 평가 22

3.2.1. 데이터 수집 과정 22

3.2.2. Rectangle 추출 프로그램 결과 분석 25
3.2.3. 바닥, 벽, 장애물 구분 및 3차원 공간 구현 25
4. 결론 및 향후 연구 방향 28
5. 개발 일정 29
6. 참고 문헌 31

1. 서론

1.1. 연구 배경

최근 다양한 분야에서 3차원 공간의 정보 생성, 구축, 활용에 관한 연구를 진행하고 있다. 특히 디지털 트윈(Digital Twin) 시장이 큰 주목을 받고 있다. 디지털 트윈이란 현실 세계에 실재하는 것을 가상 공간에 구현한 후, 현실의 실시간 데이터로 가상 공간을 업데이트하며 시뮬레이션, 머신 러닝, 추론을 통해 의사 결정을 돕는 가상 모델을 의미한다.

디지털 트윈에 필요한 3차원 공간 정보를 생성하기 위해 사용한 센서는 LiDAR(light detection and ranging)이다. LiDAR 센서는 레이저 펄스를 쏘고, 반사되어 돌아오는 시간을 측정하여 반사체의 위치 좌표를 측정한다. 조명 등 외부 환경적 방해를 적게 받고, 비교적 넓은 범위의 3차원 정보를 포인트 클라우드(point cloud)로 획득할 수 있어 비용 대비 높은 정밀도로 주목받는 시스템이다.

LiDAR를 이용한 3차원 가상 공간 구현과 관련된 선행 연구는 빠르고 효율적으로 물체를 감지하는 기술을 주제로 많이 다루고 있었다. 본 과제에서는 공간에 더욱 주목해보고자 하였다. 따라서 LiDAR의 레이저가 장애물에 가려져 투과하지 못해 손실된 벽 또는 바닥의 포인트를 보간하여 손실된 부분이 없는 공간과 장애물을 가상공간에 구현하는 것을 과제의 주제로 선택하였다.

1.2. 연구 목표

LiDAR 센서를 탑재한 원격 제어 카트를 만들고 실제 공간을 스캔하여 얻은 데이터를 3D 모델로 만들어 Unity에 렌더링하고자 한다. 이를 위해 라즈베리파이를 기반으로 한 카트에 LiDAR 센서를 부착하여 실시간 포인트 클라우드를 수집하고 축 센서의 데이터와 함께 SLAM 라이브러리에서 처리하여 방 전체의 포인트 클라우드를 만들어낸다. 여기서 방과 장애물들의 형태는 육면체로 제한한다. 그 후 방의 포인트 클라우드를 Python에서 분석하여 각 육면체의 면에 해당하는 rectangle 들로 변환한다. 여기서 장애물에 일부가 가려진 면들도 완전한 rectangle로 복구할 것이다. 변환된 사각형들은 Unity에서 최종적으로 3D cube 들로 변환되어 렌더링 된다. 이때 렌더링 된 cube 들을 벽과 장애물로 분리하고 UI 버튼을 통해 장애물들의 렌더링 여부를 제어할 수 있도록 한다.

1.3. 기대 효과

SLAM을 통해 수집한 3차원 지도가 주로 사용되는 분야는 로봇청소기, 자율주행 차량의 주차 등이 있다. 이런 분야에서는 수집된 3차원 지도에 존재하는 벽, 천장, 바닥과 장애물을 구분하지 않고 사용한다. 앞선 분야는 구분없이 사용할 수 있지만, 장애물을 구분하는 기술은 이외의 여러 분야에 사용될 것으로 기대된다.

집이나 방 내부를 촬영하여 배치된 가구들을 분리하고, 이를 통해 가구가 없는 방의 치수를 재고 빈방을 시각화할 수 있을 것이다. 가구를 배치하기 위해 길이를 잴 필요가 없어질 것이다. 부동산에서 방을 안내할 때 가구를 치울 필요 없이 빈방을 시뮬레이션하여 보여줄 수 있을 것이다. 화재 현장에서 장애물들을 파악하여 진입로를 확보하는 데에도 사용할 수 있을 것이다.

2. 연구 내용

2.1. 포인트 클라우드 데이터 수집

2.1.1. 장치 구성도

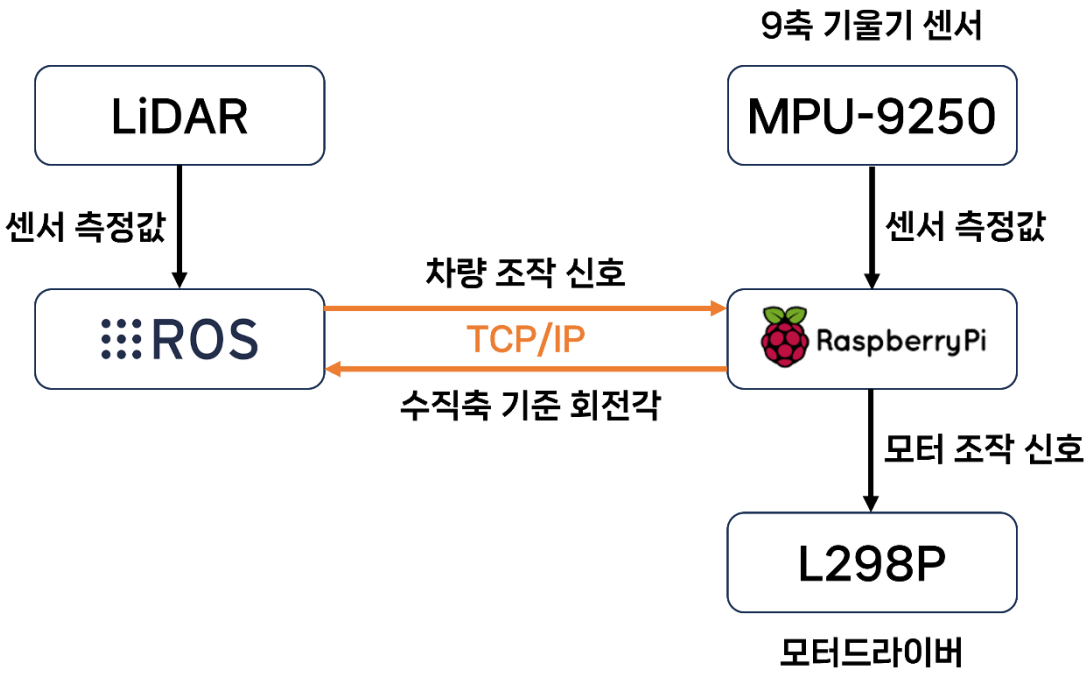

그림 1 장치 구성도

2.1.2. 데이터 수집 장치 및 테스트 공간

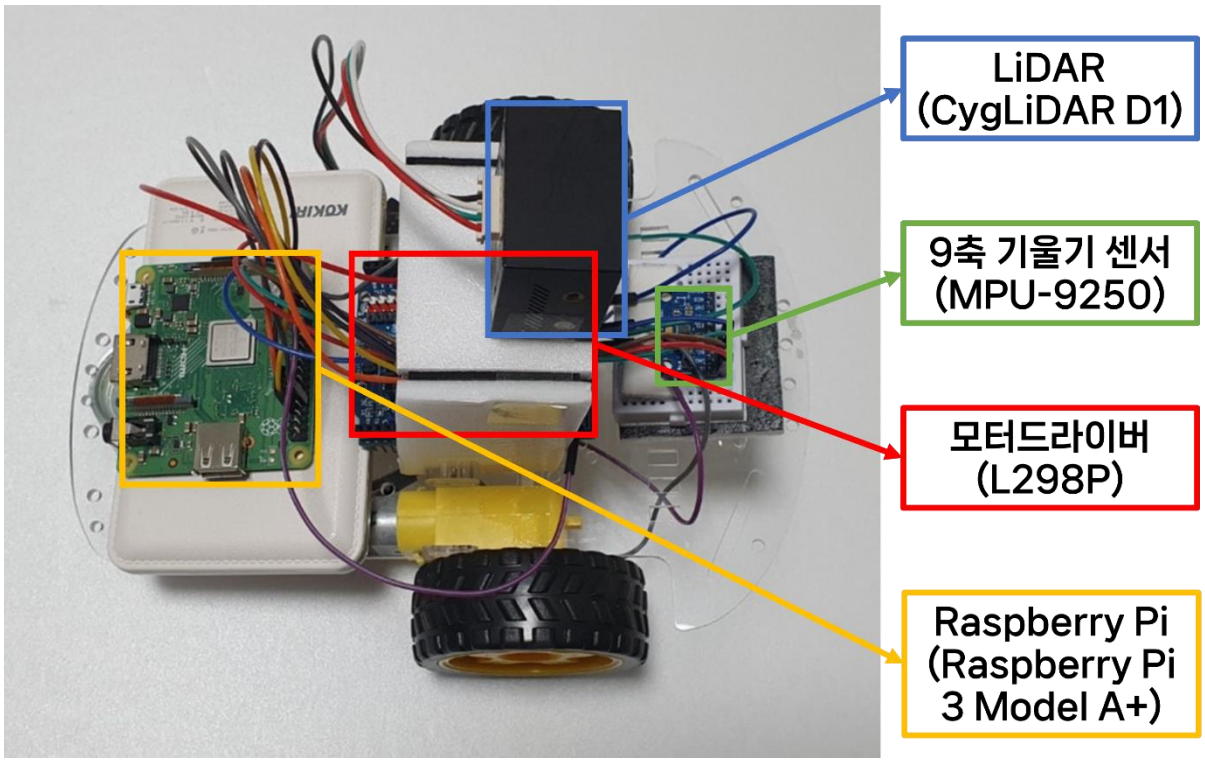

그림 2 데이터 수집 장치의 주요 구성

- LiDAR

연구에 사용한 주요 데이터 수집 장치로는 Cygbot사의 Cyglidar_D1을 사용하였다. 2D/3D 데이터를 동시에 측정할 수 있고 3차원 포인트 클라우드 데이터를 ToF 방식으로 수집한다. 직접 조작하는 차량에 부착되어 함께 움직이고 수집된 데이터는 유선 연결된 컴퓨터의 ROS에서 처리된다.

- 9축 기울기 센서

차량의 회전을 감지하기 위해서 MPU9250 센서를 사용하였다. 가속도, 자이로, 지자기 센서가 포함되어 각각을 측정하고 이를 이용하여 roll, pitch, yaw를 계산할 수 있다. I2C 통신을 이용하여 Raspberry Pi에 측정된 데이터를 전송한다.

- 모터드라이버

차량의 바퀴 조작을 위해서 L298P 모터드라이버 쉴드를 사용하였다. 차량은 전진, 후진, 제자리 360도 회전을 지원해야 하므로 모터를 양방향으로 회전시킬 필요가 있었다. Raspberry Pi에서 모터 조작 신호를 받아 바퀴 2개를 회전시킨다.

- Raspberry Pi

차량의 조작 및 센서 측정값 전송을 위해서 Raspberry Pi 3 Model A+를 사용하였다. 9축 기울기 센서에서 측정값을 받아 회전각을 계산한다. 모터 조작을 위한 신호를 모터 드라이버에 보내 카트가 원하는 방향으로 이동하게 한다. 컴퓨터와 TCP 통신을 통해 연결되어 ROS에 수직축 기준 회전각을 전송하고, 차량 조작 신호를 전송받는다.

- 테스트 공간

우드락으로 만들어진 4개의 벽과 1개의 바닥이 결합되어 900x900x300 크기의 테스트 공간을 구성한다. 장애물은 340x250x200 크기를 가지는 상자 1개를 사용한다. 이 장애물을 테스트 공간 내의 임의의 위치에 배치한다.

2.1.3. ROS

ROS는 로봇 애플리케이션 구축에 사용하는 라이브러리 및 도구들의 모음이다. 장비나 프로그램에 부여된 노드 간의 데이터 통신 인터페이스를 정의하여 지원한다. 연구에 사용된 CygLiDAR D1이나 OctoMap은 ROS 라이브러리를 지원한다. 이를 통해 CygLiDAR D1에서 수집된 데이터를 OctoMap에 전달하고 지도를 작성할 수 있다. 또한 python 코드를 작성하여 ROS 내부에서 실행시킬 수 있는 rospy 라이브러리를 지원한다.

- CygLiDAR_ROS[1]

CygLiDAR_ROS는 Cygbot사에서 제공하는 ROS 라이브러리이다. CygLiDAR D1에서 수집된 데이터를 ROS 내부에서 다른 노드와 공유할 수 있도록 한다. 공유되는 데이터는 포인트 클라우드 형태의 scan_2D, scan_3D이나 Image 형태의 scan_image, ROS의 자료형 중 하나인 LaserScan 형태의 scan이 있다. 연구에서는 3차원 포인트 클라우드 데이터인 scan_3D를 OctoMap에 전달하여 사용하였다. 차량 움직임을 추적하는 transformation frame을 fixed frame으로 설정하여 위치 정보가 반영된 포인트 클라우드 데이터를 얻을 수 있도록 하였다.

- OctoMap[2][3][4]

OctoMap은 로봇이 주변 환경을 인식하고 위치를 파악함과 동시에 지도를 작성하는 기술인 SLAM의 한 종류이다. Octree를 기반으로 작동하며 센서 측정값을 추가하는데

제한이 없고 map의 범위를 정할 필요 없이 동적으로 확대되는 장점이 있다. Cygbot 공식 유튜브에서도 OctoMap을 사용한 SLAM을 시연한 적이 있어 이를 여러 SLAM 중에서 선택하게 되었다. CygLiDAR-ROS에서 포인트 클라우드 데이터를 받아 3차원 지도를 생성한다. OctoMap의 frame은 이동하지 않는 고정 frame으로 설정해 지도가 중심을 기준으로 생성되도록 하였다. octomap_mapping.launch의 resolution 파라미터를 작게 수정하여 작은 크기의 테스트 공간에서도 충분한 양의 데이터가 모일 수 있도록 하였다.

2.1.4. python

- Raspberry Pi

Raspberry Pi에서는 차량 이동을 위한 모터 조작 신호와 회전각 계산을 담당한다. 회전각과 모터 조작 신호를 컴퓨터의 ROS와 주고받기 위해서 TCP 통신을 수행한다.

모터 조작은 python에서 Raspberry Pi GPIO를 위해 사용하는 RPi.GPIO 패키지를 사용한다. Raspberry Pi의 GPIO 5, 6, 12, 13번을 output으로 설정한다. 모터드라이버에서 모터 속도 조절을 위한 PWM 신호는 GPIO 12, 13번에 설정하여 보낸다. ROS에서 차량 이동에 대한 신호가 오면 그에 따라 모터 속도와 방향을 결정하는 신호를 보낸다.

회전각 계산은 9축 기울기 센서와의 i2c 통신을 위한 smbus 패키지, 센서값을 읽어오고 계산하기 위한 imusensor 패키지를 사용한다[5]. 먼저 측정의 정밀함을 위해서 각 센서의 교정을 진행하고 교정값을 저장하였다. 이후 저장된 교정값을 적용하여 센서값을 읽어오고 노이즈를 제거하기 위한 Kalman filter를 적용하여 회전각을 계산한다. 연구에서 차량이 수직을 제외한 다른 축을 기준으로 회전하는 경우는 없기 때문에 ROS에는 yaw 값만을 전송한다.

- ROS

ROS에서는 rospy 라이브러리를 통해 내부에 python 코드를 실행한다[6]. python 코드를 통해서 ROS 내부에 공유되는 데이터들을 사용할 수 있다. 이번 연구에서 ROS의 python 코드는 Raspberry Pi에서 전송받은 회전각이나 키보드 조작을 따라 움직이는 transformation frame을 생성한다. 또한 키보드 조작에 따라 차량의 움직임을 지시하는 신호를 Raspberry Pi로 전송한다.

키보드의 up 방향키 또는 down 방향키를 누르면, Raspberry Pi에 전진 또는 후진하라는 신호를 보내고 transformation frame을 앞 또는 뒤로 이동시킨다. 키보드의 left 방향

키 또는 right 방향키를 누르면, Raspberry Pi에 왼쪽으로 회전 또는 오른쪽으로 회전 신호를 보낸다. 방향키에서 손을 떼는 경우에는 모든 움직임을 멈추라는 신호를 Raspberry Pi에 전송한다.

Raspberry Pi에서 yaw 값을 전송받으면 이를 통해 transformation frame을 회전시킨다. Yaw 값이 Kalman filter를 거쳤음에도 노이즈가 발생하는 경우들이 발견되었다. 이 노이즈에 의한 영향을 최소화하기 위해 temp, count, maxCount, threshold, noiseCount, maxNoiseCount 변수를 설정하였다.

temp, count, maxCount는 측정된 값들의 평균을 구하기 위해 사용한다. 측정된 값들이 하나 넘어올 때마다 count를 하나씩 증가시키며 temp에 측정된 값들을 누적한다. Count가 maxCount와 같아지는 경우 temp를 maxCount로 나눠 평균을 계산한다. threshold는 회전각의 한계치를 정하기 위해 사용한다. 이전 회전각과 평균값인 temp의 차이를 계산하여 threshold를 넘지 않는 경우만 회전각으로 적용하였다. noiseCount와 maxNoiseCount는 -180도와 180도 부근을 측정할 때 생기는 노이즈[그림 3]를 해결하기 위해 사용한다. 이전 회전각과 측정된 값 하나의 차이를 계산하여 threshold를 넘는 경우에는 noiseCount를 하나씩 증가시키고, count가 maxCount와 같아지는 경우 maxNoiseCount와 비교한다. maxNoiseCount보다 noiseCount가 큰 경우에는 회전각을 -180도로 고정하였다.

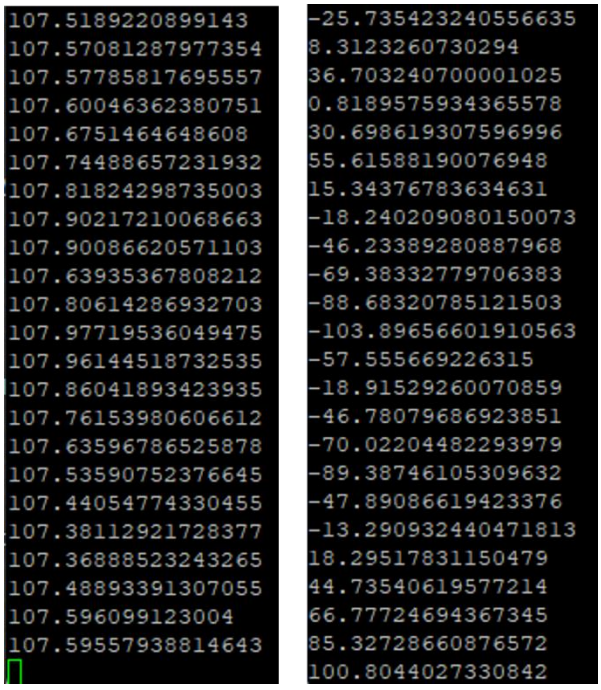

그림 3 노이즈가 없는 경우(좌) 노이즈가 발생한 경우(우)

2.2. Rectangle 추출 프로그램

2.2.1. 전반 구조

데이터 수집 장치로 얻은 데이터와 SLAM을 통해 방의 포인트 클라우드를 수집하고 나면 rectangle 추출 프로그램을 통해 해당 포인트 클라우드를 기하학적으로 분석하여 벽과 장애물들의 각 면에 해당하는 rectangle 들을 추출한다. 이때 rectangle 들을 입력받아 3D 모델로 렌더링하는 Unity 프로그램의 처리 로직에서 높이 축과 수평에 가까운 rectangle만을 필요로 하므로 이러한 rectangle만을 추출한다. rectangle 추출 프로그램은 다음의 4단계의 프로세스를 가진다.

① 파일 입력 및 전처리

n개의 3차원 포인트를 가진 포인트 클라우드 텍스트 파일을 읽고 NumPy의 (n,3) 배열로 변환한다. 그 후 바닥 제거 등의 전처리 작업을 수행한다.

② 포인트 그룹화

(n,3) 포인트 배열을 입력받아 RANSAC과 위상 기반 BFS 클러스터링을 적용하여 포인트들을 사각형 별로 그룹화하고 해당 rectangle에 대한 평면(한 점과 법선) 배열도 출력한다.

③ 포인트 그룹 rectangle 변환

각 rectangle을 이루는 포인트 그룹과 평면 정보를 받아 rectangle의 네 모서리 포인트를 출력한다.

④ rectangle 합병

Rectangle 들을 입력받아 유사한 변을 가지며 평면의 각도가 적게 차이 나는 사각형들을 합병하고 결과를 출력한다.

2.2.2. 파일 입력 및 전처리 단계

- 파라미터

해당 단계에서 사용하는 파라미터들과 예시 값은 다음과 같다. 파라미터들은 처리 과정 파트에서 *기울임꼴*로 표기된다.

int floorPoint : 150

float threshold : 0.1

- 처리 과정

파일 입력 및 전처리 단계는 n개의 포인트가 저장된 포인트 클라우드 텍스트 파일을 읽어 NumPy의 (n,3) 배열로 변환하여 출력하고 전처리하는 단계이다. 이때 포인트 파일은 앞선 SLAM의 출력으로 pcd 확장자를 가진다. pcd 파일은 11줄의 헤더 데이터와 12줄부터의 포인트 데이터들이 존재한다. 포인트 데이터는 줄마다 3개의 float가 한 포인트의 x, y, z 좌표를 나타내며 n개의 줄이 존재한다.

해당 텍스트 파일을 읽고 NumPy 배열로 변환하고 나면 전처리를 수행한다. 우선 프로그램의 성능을 위해 10만 개 이상의 포인트가 입력되면 그 중 무작위로 10만 개의 인덱스를 뽑고 중복된 인덱스를 제거하여 남은 인덱스의 포인트만 남겨둔다. 그 후 데이터 수집 환경의 한계에 의해 발생한 벽 너머의 이상 값들을 제거하기 위해 z축 (높이 축)에 thresholding를 적용하여 특정 값 이상의 높이를 가진 포인트를 제거한다. 여기서 특정 값은 환경마다 다르다. 마지막으로 바닥 포인트 제거를 위하여 높이 축의 값이 *floorPoint* 번째로 낮은 포인트를 선별한다. 그 후 선별된 포인트의 z 값에 *threshold*를 더하고 해당 값에 thresholding을 적용하여 더 낮은 z 값을 가진 포인트들을 제거한다.

2.2.3. 포인트 그룹화 단계

- 파라미터

해당 단계에서 사용하는 파라미터들과 예시 값은 다음과 같다. 파라미터들은 처리 과정 파트에서 *기울임꼴*로 표기된다

int groupSize : 500

int RANSACloop : 3000

int RANSACSampleSize : 20

float inlierDis : 0.035

float mergeDis : 0.05

int minInlier : 300

float edgeDis : 0.1

int removeSize : 200

int checkAxisNeighbors : 20

int minPlanePoints : 7500

- 처리 과정

입력으로 받은 포인트 배열에서 랜덤한 한 포인트를 추출하고 그와 인접한 *groupSize* 개의 포인트들을 추출한다. 그리고 해당 포인트들에 RANSAC을 적용하여 최적의 한 평면에 대한 inliers를 추출한다. 여기서 RANSAC은 다음을 *RANSACloop*만큼 반복하는 함수이다. 포인트 배열 중 *RANSACSampleSize* 개의 랜덤한 포인트를 추출하고 선형회귀로 평면 방정식을 만든다. 이때 평면의 법선과 up vector(0,0,1)를 내적 했을 때 절댓값이 0.1을 초과하면 해당 평면은 높이 축에 평행이 아니므로 처리하지 않고 다음 루프로 넘어간다. 절댓값이 0.1 이하면 평면과의 거리가 *inlierDis*보다 작은 inlier들을 구하고 지금까지 가장 inlier가 많은 평면과 inlier 개수를 비교하여 더 많은 쪽을 갱신한다. 그렇게 구한 가장 많은 수의 inlier 들을 반환한다. 이후 RANSAC의 inlier의 결과가 *minInlier*보다 적다면 루프의 처음으로 돌아가 다시 시작한다. 만약 20번 연속 루프 동안 *minInlier*보다 적었다면 *minInlier*를 절반으로 줄여준다.

*minInlier*보다 inlier 수가 많다면 해당 inlier 들에 선형회귀를 적용하여 평면을 얻고 전체 포인트 그룹에서 해당 평면과 *mergeDis*보다 가까운 포인트들을 추출한다. 그리고 그 결과에 거리 기반 BFS 클러스터링을 수행한다. 여기서 BFS 클러스터링은 다음과 같이 동작한다. 우선 포인트들을 각 축 값에 대해 정렬한다. 각 축에 대한 정렬 배열에서 임의의 한 포인트 p1의 인덱스에서 앞뒤로 checkAxisNeighbors /2 이하로 떨어진 포인트들 중 p1과의 실제 3차원 거리가 edgeDis보다 작은 포인트들은 p1과 인접한 것으로 간주한다. BFS 클러스터링은 이렇게 형성된 그래프에서 한 클러스터를 구성하는 포인트들끼리 분류하여 반환한다. 그 후 이렇게 분류된 그룹 중 가장 포인트 수가 많은 그룹을 추출한다. 만약 *minPlanePoints*보다 해당 그룹의 inlier 수가 같거나 많다면 결과 배열에 해당 그룹을 넣고 다음 루프로 넘어간다. 그렇지 않고 *minPlanePoints*보다 해당 그룹의 inlier수가 적다면 20번의 루프 연속으로 적었던 건지를 확인한다. 그렇지 않다면 다음 루프로 넘어가고 그렇다면 *minPlanePoints* /= 2, *mergeDis* *= 0.8를 적용한다. 이때 *minPlanePoints*가 *removeSize*보다 작거나 같아졌다면 루프를 종료하고 결과 배열을 출력하고 아니라면 다음 루프를 수행한다.

이를 플로우 차트로 표현하면 다음과 같다.

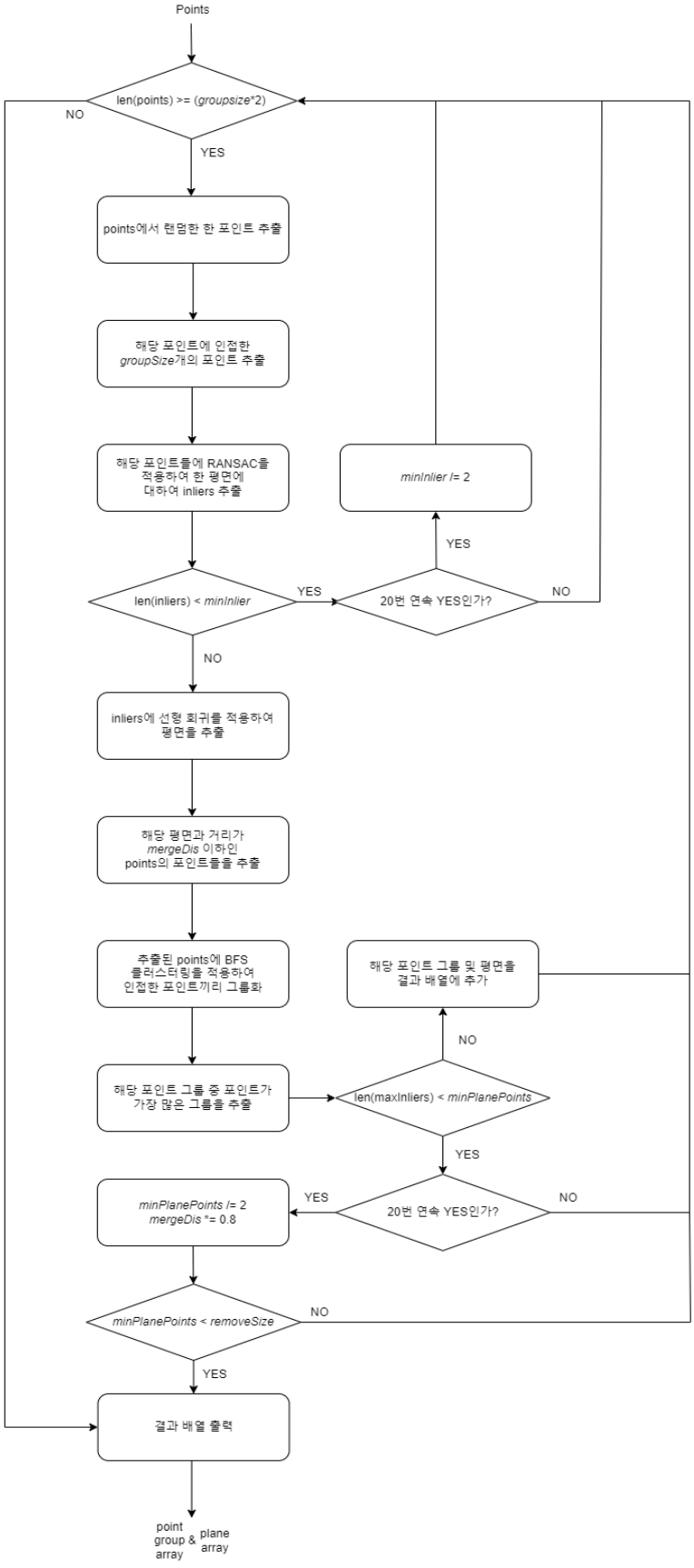

그림 4 포인트 그룹화 단계 플로우 차트

2.2.4. 포인트 그룹 Rectangle 변환 단계

- 파라미터

해당 단계에서 사용하는 파라미터들과 예시 값은 다음과 같다. 파라미터들은 처리 과정 파트에서 *기울임꼴*로 표기된다

int numSampleLine : 1

int RANSACloop : 300

int n : 2

float inlierDis : 0.05

float edgeWidth : 0.03

- 처리 과정

해당 단계에서는 포인트 그룹의 포인트들을 평면에 투영시키고 평면의 법선을 z축으로 하는 정규 직교 기저를 구한다. 여기서 xy축은 정규 직교이기만 하면 아무축이라도 상관이 없다. 그 후 각 포인트에 convex hull 알고리즘을 적용하기 위해 해당 기저로 좌표 변환하고 getEdgePoints 함수에 해당 포인트들을 적용한다. getEdgePoints는 넘겨진 포인트들에 convex hull을 적용해 볼록 다각형 포인트들을 추출한다. 그리고 볼록 다각형 포인트들의 위치 평균을 구하고 해당 방향으로 *edgeWidth*만큼 이동시킨 뒤 해당 볼록 다각형 바깥의 점들을 edgePoints로써 출력한다.

그 후 edgePoints들에 RANSAC을 적용하여 최적의 line을 찾아낸다. RANSAC은 *n*개의 포인트를 무작위 추출한 뒤 선형회귀로 line을 구하고 해당 line과의 거리가 *inlierDis*보다 작은 inlier 개수를 세어 *RANSACloop*번의 루프 동안 가장 inlier가 많은 line을 찾아내어 후보 line으로 둔다. 그렇게 *numSampleLine* 번 반복하여 *numSampleLine* 개의 후보 line을 구하고 그중 길이가 가장 긴 것을 출력한다.

Line을 얻었으면 평면의 법선을 z축으로, line의 방향을 x축으로 하는 정규 직교 기저를 구하고 전체 포인트들을 해당 기저로 좌표 변환한다. 그 후 각 포인트의 최대, 최소 xy좌표를 구하고 이를 통해 네 개의 모서리 포인트 좌표를 얻는다. 마지막으로 해당 포인트를 표준 기저로 변환하여 출력한다.

2.2.5. Rectangle 합병 단계

- 파라미터

해당 단계에서 사용하는 파라미터들과 예시 값은 다음과 같다. 파라미터들은 처리 과정 파트에서 *기울임꼴*로 표기된다

float linePointsDis : 0.2

float limitNormalAngle : 20

- 처리 과정

해당 단계에서는 입력으로 들어온 각 rectangle의 평면 방정식을 계산하고 다음의 루프를 실행한다. Rectangle 배열에서 한 rectangle과 그에 대한 평면 방정식을 pop하고 해당 rectangle과 합병을 할 수 있는 rectangle이 rectangle 배열에 있는지 탐색한다. 이때 합병이 가능한 조건은 한 rectangle의 한 변의 두 점과 다른 rectangle의 한 변의 두 점을 한 점씩 짝지었을 때 각 거리가 *linePointsDis* 이하면서 두 평면 사이의 각도가 *limitNormalAngle* 이하이면 합병 가능하다고 판단한다. 만약 합병 가능한 rectangle이 없다면 해당 rectangle을 결과 배열에 넣고 다음 루프를 진행한다. 합병 가능한 rectangle이 있다면 해당 rectangle과 합병을 진행한다. 이때 합병은 평면의 법선을 z축으로, 한 rectangle의 한 변의 방향을 x축으로 하는 정규 직교 좌표로 두 rectangle의 점들을 좌표 변환하고 최대, 최소 xy좌표를 구하여 합병된 사각형의 네 모서리 포인트를 구한다. 이후 합병된 rectangle에 대해 rectangle 배열 및 결과 배열에 있는 rectangle 중 합병이 가능한 rectangle이 있는지 찾고 있다면 다시 합병하여 다시 탐색을 진행한다. 그렇게 반복하다 합병 가능한 rectangle이 더 이상 나오지 않으면 해당 rectangle을 결과 배열에 넣고 다시 루프를 진행한다. 그렇게 루프를 반복하다 rectangle 배열에 있는 rectangle의 수가 2보다 작아지면 루프를 빠져나가고 rectangle 배열에 남은 모든 사각형을 결과 배열에 넣은 뒤 출력한다.

이를 플로우 차트로 표현하면 다음과 같다.

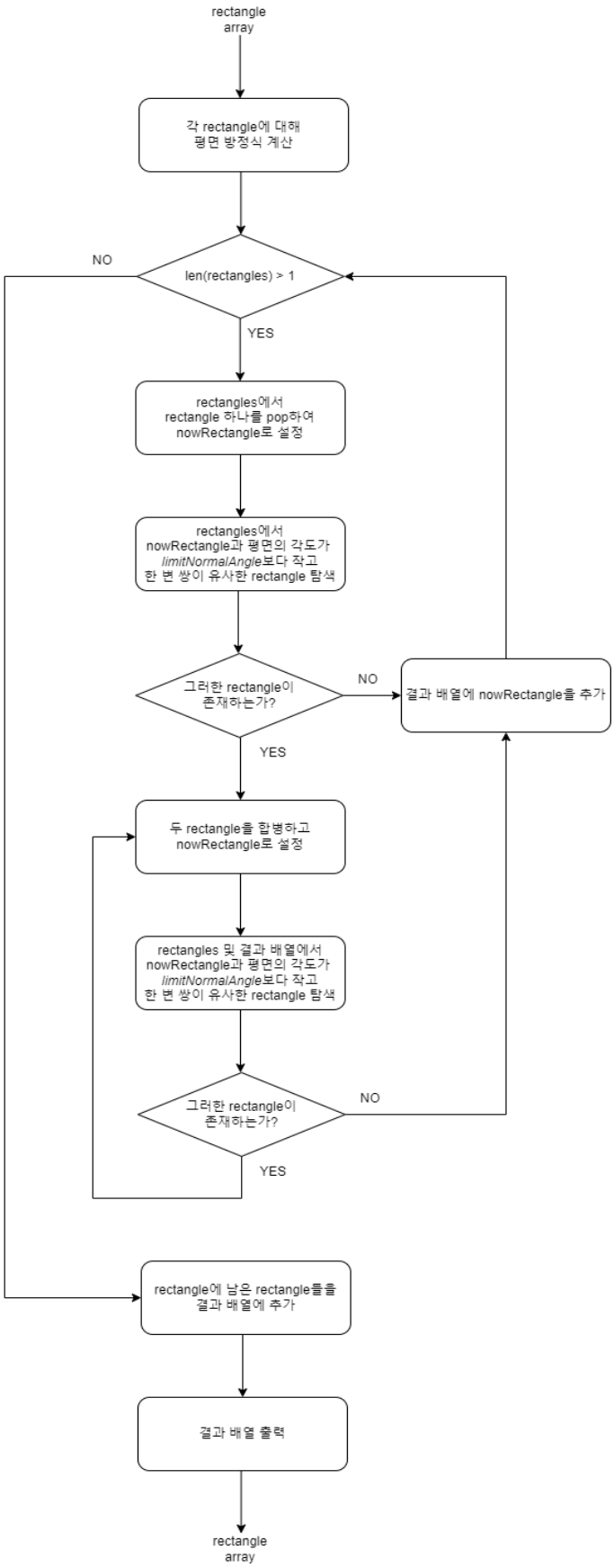

그림 5 Rectangle 합병 단계 플로우 차트

2.3. 바닥, 벽, 장애물 구분 및 3차원 공간 구현

2.2의 과정을 통해 추출한 rectangle들을 이용하여 3차원 공간에 바닥, 벽과 장애물을 구분하여 구현하기 위해 다음의 과정을 수행하였다.

포인트 클라우드로부터 추출한 각 rectangle의 4개의 3차원 좌표를 한 줄로 작성한 텍스트 파일(그림 6 위)을 읽는다. 4개 점은 이웃하게 나열되어 있다. xz 평면에 수직하여 바르게 세워져 있는 실제 공간의 벽과 달리 rectangle들은 기울어져 있는 경우가 많기 때문에 오차를 최소화하기 위해 xz 평면에 투영시켰을 때 수직하여 4개의 점이 아니라 2개의 점으로 겹쳐 나타나도록 1차 보정을 수행한다. 4개 점을 이용하여 구한 면의 법선 벡터의 y(높이) 성분이 0이 아니거나 높이가 높은 두 점의 높이가 다른 경우 면이 회전하여 있음을 알 수 있다. 사영시켰을 때 겹쳐져야 할 2개 점을 찾아 평균을 내어 새로운 1개의 점을 생성한다. 보정된 면의 법선 벡터, 최고 최저 높이, 대각선 정보 등도 함께 저장한다. 그림 6에서 추출한 rectangle들의 정보와 그에 따른 텍스트 파일, 1차 보정을 수행한 후 실제 저장되어 사용되는 점을 볼 수 있다. 그림 5의 rectangle이 11개이므로 텍스트 파일에도 총 11개의 줄 각각에 면을 나타내는 4개의 점에 대한 좌표 값 x, y, z이 4개씩 들어있다. 그림 6의 아래 오른쪽에서 텍스트 파일의 몇 번째 줄을 읽은 결과인지를 나타내기 위해 번호를 표시하였고, 아래 왼쪽 rectangle을 나타낸 그림에서 일치하는 색의 rectangle을 찾아 1차 보정 전의 모습을 확인할 수 있다.

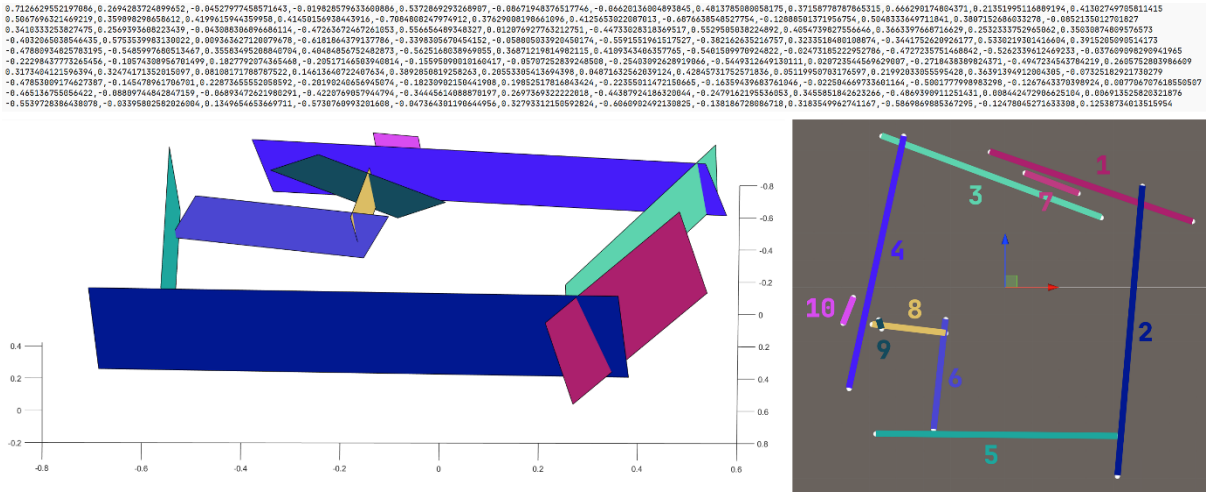

그림 6 2.2의 단계에서 추출한 rectangle들(아래 왼쪽)과
rectangle들의 점을 나타낸 텍스트 파일(위)에 따라 실제 저장되어 사용되는 점(아래 오른쪽)

저장한 모든 점을 대상으로 선분끼리 중복된 점을 제거한 후 Convex hull 알고리즘을 수행한다. 결과로 모든 점을 포함하는 볼록다각형인 convex hull (그림 7의 (b))을 구할 수 있다. Convex hull로부터 공간을 이룰 4개의 정점을 구하기 위해 점의 개수가 4개가 될 때까지 가장 거리가 가까운 2개의 점을 1개의 새로운 점으로 바꾸는 작업을 반복한다. 새로운 점은 2개의 점이 포함되어 있던 면의 대각선 값에 따라 다르게 생성된다. 대각선 길이가 같은 경우 2개 점의 평균 위치에 새로운 점을 생성하고, 다른 경우는 대각선 길이가 큰 점이 새로운 점이 된다. Convex hull로부터 구한 4개의 점(그림 7의 (c))을 기준으로 공간이 xz 평면의 1사분면에 위치할 수 있도록 이동, 회전 변환한다(그림 7의 (d)). 3차원 공간에 구축해야 할 공간은 직사각형이므로 4개 점이 직사각형을 이루도록 추가로 보정한다 (그림 7의 (e)). 4개 점을 이용하여 만들 수 있는 가장 큰 직사각형과 가장 작은 직사각형의 평균이 최종적으로 공간을 구성하는 바닥이 된다. 이때 이 4개 점이 직사각형의 꼭짓점으로 변환되는 것에 맞추어 모든 점들을 homography 변환하였다. 이 직사각형의 4개 정점으로 바닥과 4개 벽을 이루는 cube를 생성하여 공간을 구현하였다.

다음으로 공간에 존재하는 장애물을 구현하기 위해 벽 주변에 존재하지만, 장애물이 아니라고 판단되는 점들을 제거한다 (그림 7의 (f)). 제거할 점을 판단하는 기준은 벽으로 두었다. 벽의 안팎으로 점을 제외할 공간을 정한 후, 선분의 두 점이 모두 벽의 외부에 있다면 제외한다. 선분의 한 점만 벽의 외부에 있어, 선분이 벽에 걸쳐지는 경우는 벽으로 나눠진 선분의 외부 부분이 더 큰 경우만 제외하였다. 나머지 점들이 이루는 선분들끼리 최단 거리에 따라 오름차순으로 정렬하여, 남는 선분이 없을 때까지 같은 장애물을 이루는 점으로 분류하였다. 각 장애물 그룹 점 집합을 대상으로도 벽을 구했던 것과 비슷한 과정을 수행한다. 최종 4개 점으로 구할 수 있는 직사각형을 구하기 위해 별도의 좌표 변환이 추가로 필요하다는 점, 그 직사각형으로 벽과 바닥을 구성하는 5개의 cube가 아닌 1개의 cube만 생성한다는 점, 생성한 cube가 벽을 통과하는 경우 장애물 cube가 벽을 넘어가지 않게 하기 위한 재보정 과정이 있다는 점 등이 다르다. 장애물의 색상은 하얀색으로 나타내는 벽과 쉽게 구분해야 하고, 여러 개의 장애물이 함께 있는 방을 스캔할 수 있기 때문에 랜덤하게 색이 정해지도록 하였다.

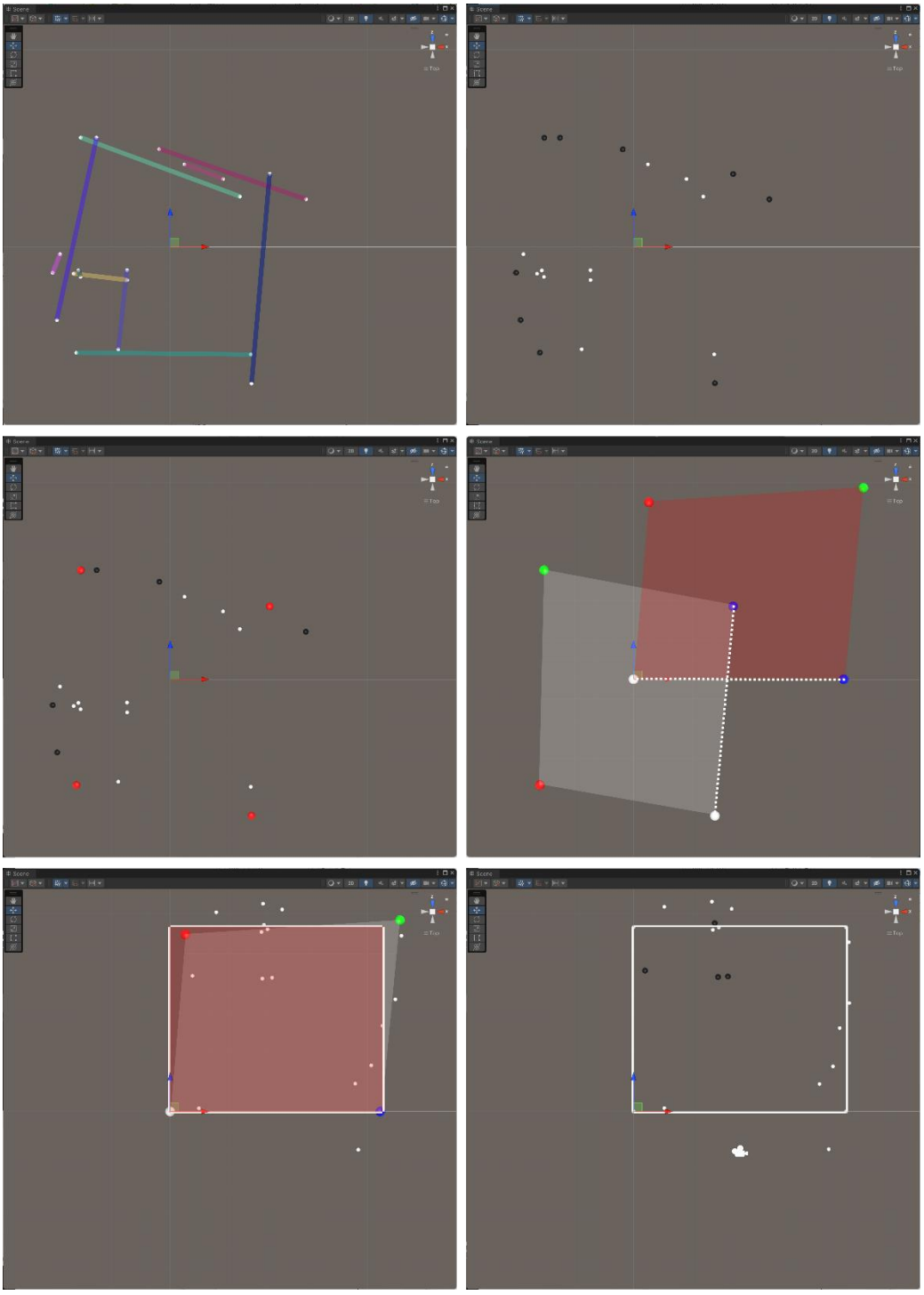

그림 7 벽 생성과 장애물 점을 분리하기 까지의 과정을 나타낸 Unity scene 화면.
(a) 텍스트 파일을 읽고 1차 보정 후.
(b) convex hull을 검정색 점으로 표시.
(c) (b)의 convex hull을 이용하여 벽을 구성할 4개 점을 빨간색으로 표시.
(d) (c)의 4개 점이 xz 평면의 1사분면에 위치할 수 있도록 좌표 변환.
(e) (d)의 4개 점을 이용하여 생성한 직사각형(흰색 벽)으로 homography 변환.
(f) 장애물에 생성에 사용하기 위해 분리된 점을 검정색으로 나타냄.

벽과 바닥, 장애물을 생성하는 cube는 벽을 이루는 4개 정점으로부터 직육면체의 8개 정점을 이용하여 만들었다. 위 과정까지는 xz 평면에 사영했을 때 나타나는 벽과 바닥, 장애물의 바닥 면만을 구한 상태이기 때문에, 저장해 두었던 높이 정보를 함께 사용하면 직육면체를 이루는 8개 정점을 구할 수 있다. 유니티에서 메시는 3개의 점으로 정의되는 삼각형들로 구성된다. 6개의 면으로 이루어진 직육면체 cube를 만들기 위해서는 1개 면에 2개의 삼각형을 위해 6개의 정점이 필요하므로 총 24개의 정점이 필요하다. 메시의 뒷면은 렌더링되지 않으므로 면이 보여야 할 방향과 정점의 순서에 유의하여 8개 정점으로 cube를 생성한다.

화면 오른쪽 아래에 있는 버튼은 장애물을 숨기거나 다시 나타나게 하는 토글 역할을 수행한다. 장애물이 나타나 있는 상태일 때는 장애물을 숨기기 위해 "OFF", 장애물이 숨겨진 상태일 때는 다시 나타나게 하기 위해 "ON"이 버튼 텍스트에 나타나도록 했다.

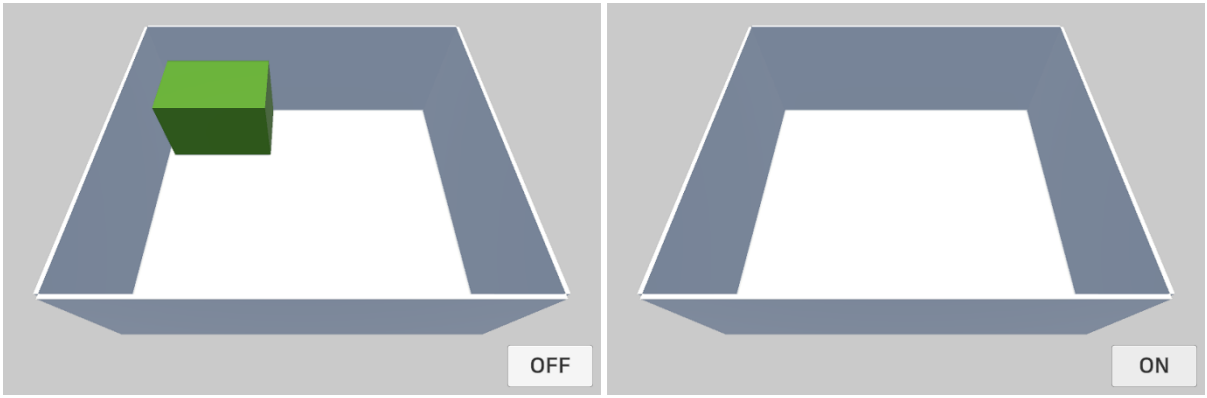

그림 8 Unity 시각화 화면. 오른쪽의 버튼을 통해 장애물을 나타내거나(왼쪽) 숨길 수 있음(오른쪽). 버튼을 누르지 않은 처음 상태는 (왼쪽) 화면.

공간을 시각화하는 것이 목표이므로 카메라는 공간의 중심점을 주목한 상태를 유지하며 이동하도록 하였다. 수직 방향으로는 이동하지만 수평 방향으로는 회전만 가능하다. 이는 키보드 방향키로 제어할 수 있다. 카메라 확대와 축소는 카메라가 바라보는 방향 앞, 뒤로 이동하게 하여 마우스 스크롤로 제어할 수 있도록 구현하였다. 카메라가 비추는 공간과 너무 멀어지지 않도록 이동할 수 있는 거리에 제한을 두었다.

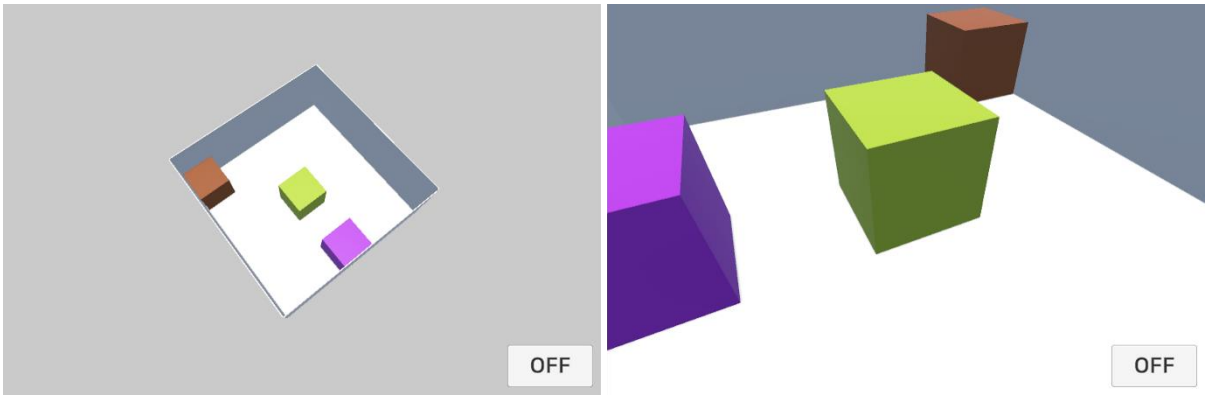

그림 9 Unity 시각화 화면. (왼쪽) 카메라를 가장 멀리 이동시켰을 때. (오른쪽) 카메라를 확대하였을 때.

3. 연구 결과 분석 및 평가

3.1. 생성과정 및 결과

Cube가 직육면체 방에서 위치할 수 있는 3가지 경우의 수를 테스트케이스로 설정하였고, 이 경우의 수가 모두 존재하도록 코드로 생성한 이상적인 케이스를 마지막 테스트케이스로 설정하였다.

실제 공간과 장애물의 사진(1), rectangle 추출 과정(2a ~ 2f), 최종 결과로 나타나는 Unity 시각화 화면(3)을 각 테스트 케이스마다 아래에 그림으로 나타내었다.

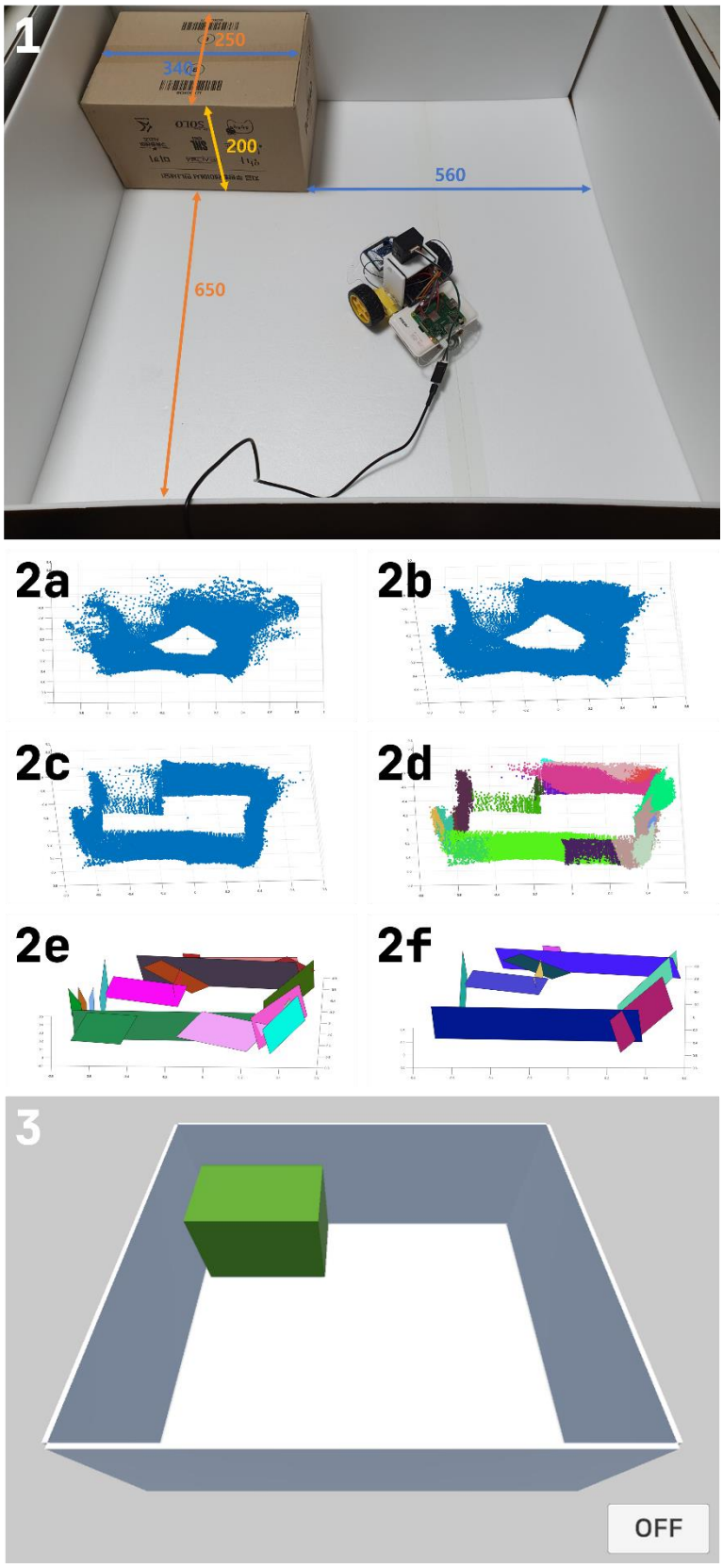

그림 10 1번 케이스 (장애물 코너 배치 맵)

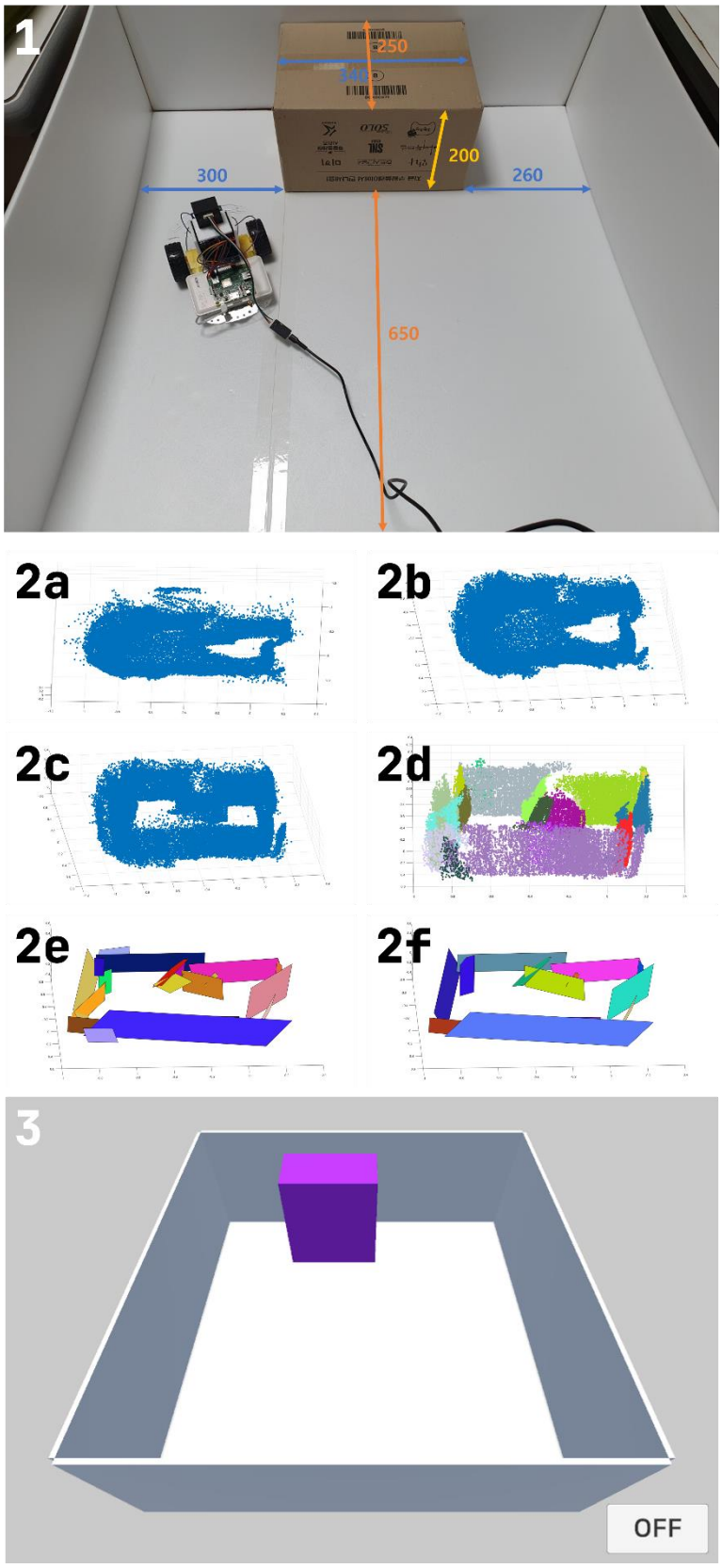

그림 11 2번 케이스 (장애물 벽 배치 맵)

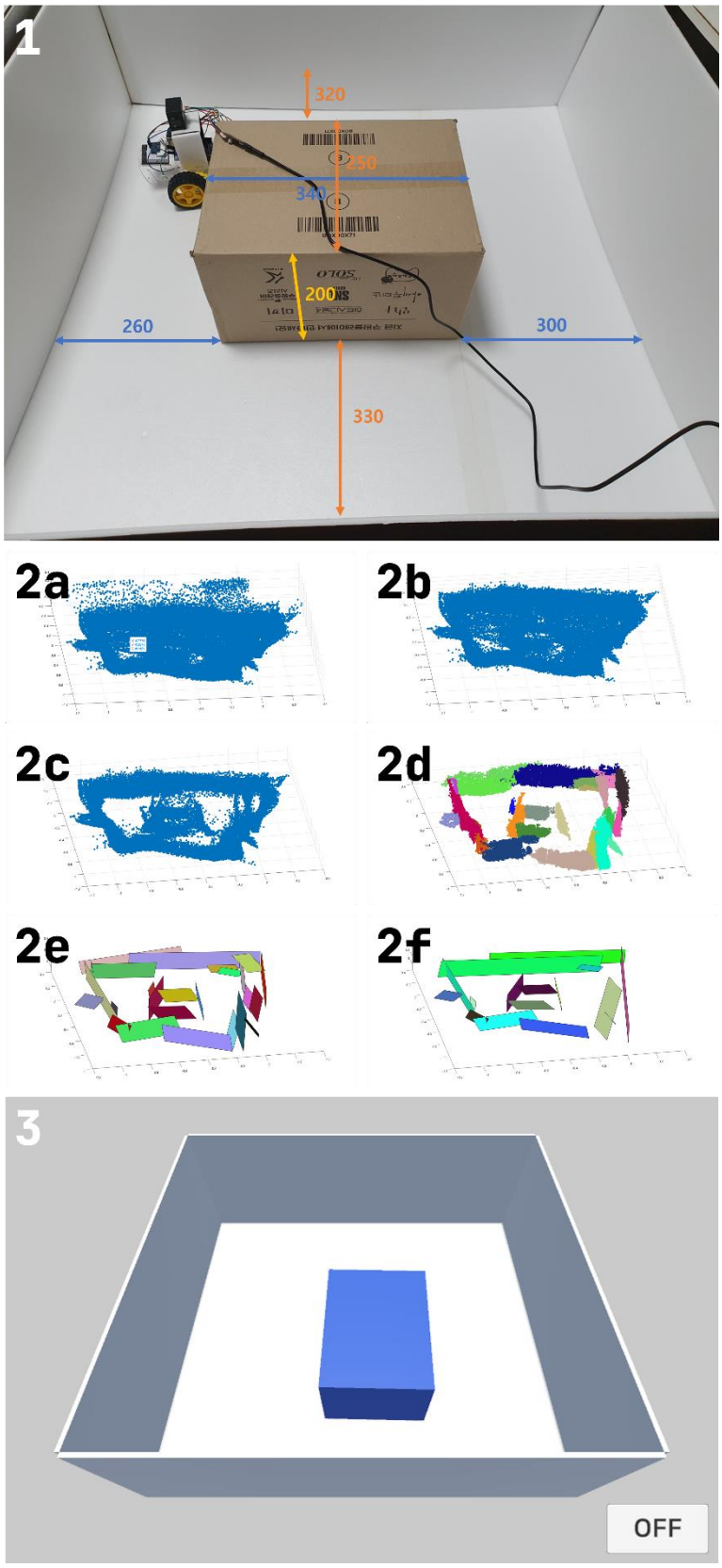

그림 12 3번 케이스 (장애물 중앙 배치 맵)

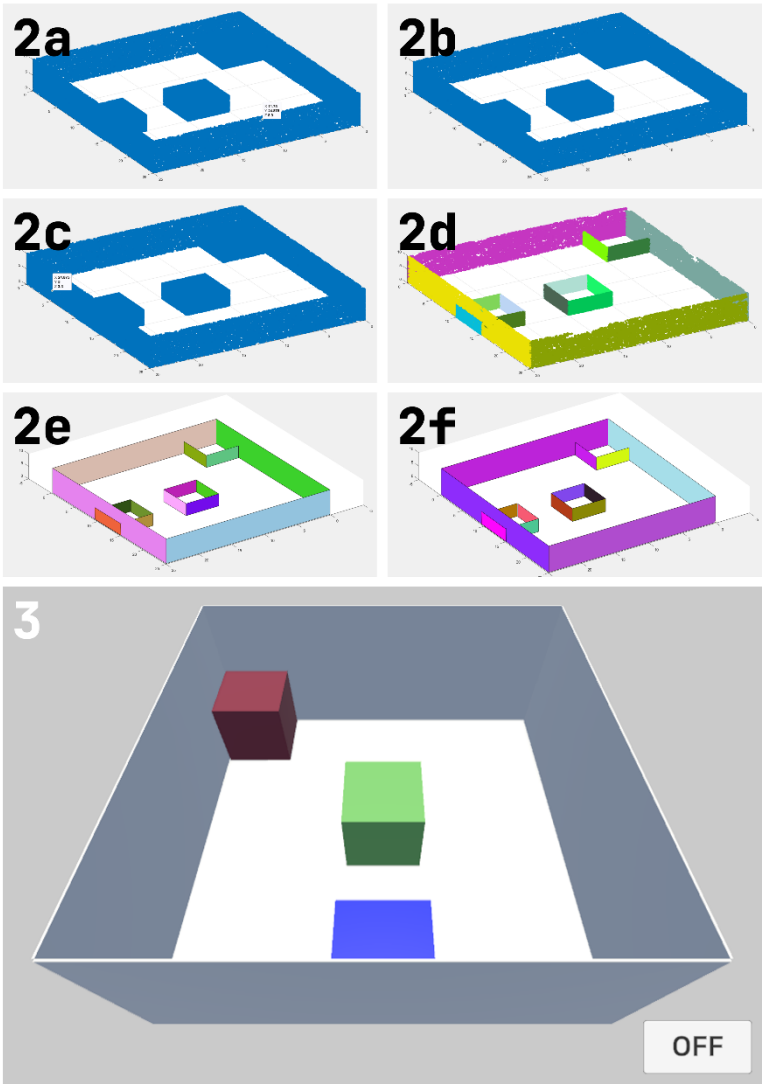

그림 13 4번 케이스 (코드로 생성된 이상적인 맵)

3.2. 결과 분석 및 평가

3.2.1. 데이터 수집 과정

- 공통 제약사항

테스트 공간을 평평한 바닥으로 하고 볼 베어링을 장착했지만, 마찰력으로 인해 회전이나 전진, 후진에 오차가 발생하였다. 회전은 제자리에서 회전을 가정하고 코드를 구현하였지만, 축이 고정된 회전이 아닌 조금씩 움직이는 모습을 보였다. 전진이나 후진 또한 키 입력 시간에 비례하여 일정한 거리만큼을 이동하도록 코드를 구현하였지만 실제 전진 거리나 후진 거리는 차이가 있었다. 또한 키보드 입력에 딜레이가 생겨 ROS 내부

transformation frame의 이동 거리도 일정하지 않았다.

- 1번 케이스

1번 케이스는 전진이나 후진 없이 제자리에서 회전만으로 수집을 진행하였다. 코너에 장애물이 있는 경우에는 테스트 공간 중앙에서 회전하는 움직임만으로 벽면 전체와 장애물에서 노출된 2개의 면 데이터를 수집할 수 있었다. 다른 케이스에 비해서 움직임이 적고, 오차가 발생하기 쉬운 전진이나 후진이 과정에 들어가지 않아 공간을 거의 온전히 담을 수 있었다.

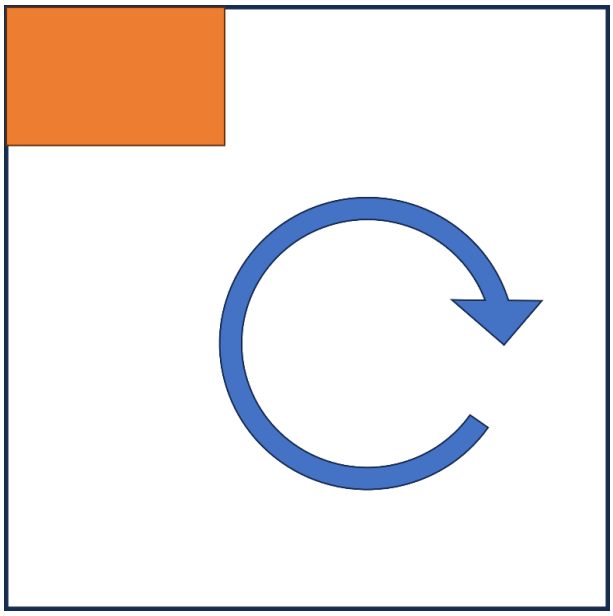

그림 14 1번 케이스의 회전

- 2번, 3번 케이스

2번과 3번 케이스는 장애물에서 노출된 모든 면 데이터를 수집하기 위해서 전진과 후진이 필요했다. 전진, 후진, 회전이 모두 오차가 발생하는 상황에서 더 많은 움직임이 발생할수록 수집되는 3차원 지도의 형태가 일그러졌다. 장애물 데이터가 직육면체 형태가 아닌 직육면체가 여러 개 겹쳐진 형태로 수집되거나, 벽 데이터가 여러 겹 겹쳐져 두꺼워지고 하나의 벽이 여러 개로 수집되었다.

2번 케이스는 장애물에서 3면이 노출되어 있어 이를 수집하기 위해서 장애물 기준 양옆으로 이동이 필요했다. 데이터 수집은 2번의 회전과 1번의 전진으로 진행되었다. 첫 번째 회전과 두 번째 회전에서 장애물의 위치가 다른 곳에 있는 것처럼 수집되어 장애물에서 큰 왜곡이 발생하였다.

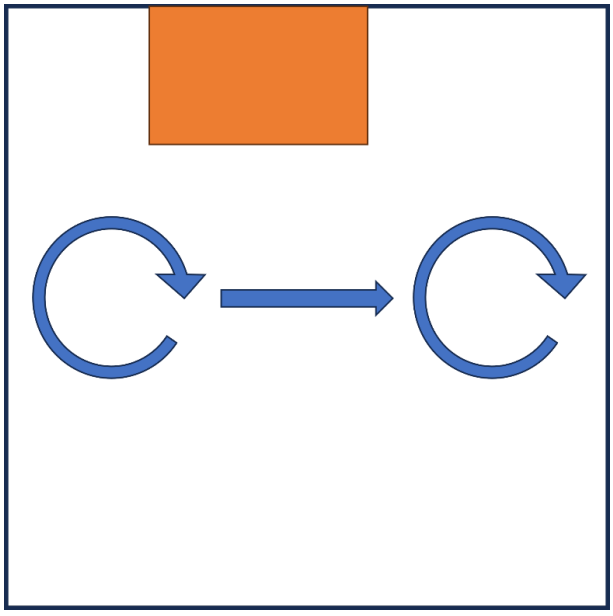

그림 15 2번 케이스의 이동 및 회전

3번 케이스는 장애물 4면이 모두 노출되어 있어 이를 수집하기 위해 4개의 코너로 이동이 필요했다. 데이터 수집은 4번의 회전과 3번의 전진으로 진행되었다. 2번 케이스에 비해 이동 거리도 길고 회전도 많아져 큰 오차들이 발생했다. 장애물은 직육면체의 형체를 알아보기 힘들 정도로 왜곡되었으며, 벽은 두꺼워져 새로운 장애물이 놓여 있는 것처럼 수집되었다.

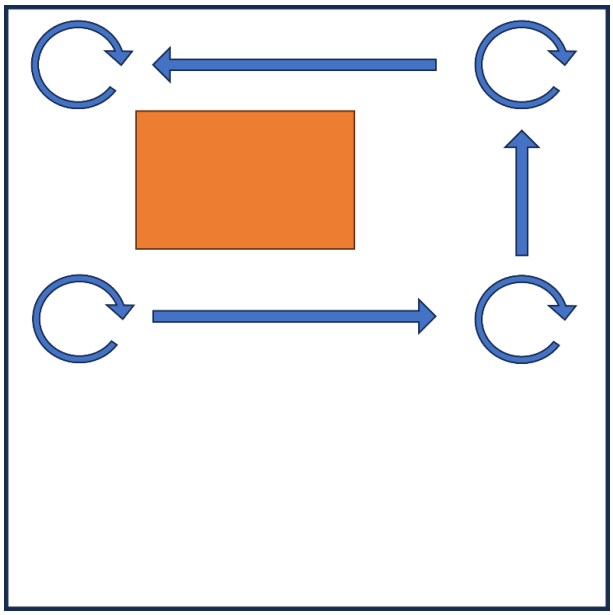

그림 16 3번 케이스의 이동 및 회전

3.2.2. Rectangle 추출 프로그램 결과 분석

코드를 통해 생성한 4번 케이스를 제외한 다른 케이스들을 실제 환경의 모습과 Rectangle 추출 결과를 비교해 보았을 때 형태와 위치가 실제와 대략적으로는 유사하였으나 세부적인 오차가 많이 발생하였다. 4번 케이스의 경우 최초 데이터와 매우 흡사한 결과가 나타났다.

- 벽면 Rectangle 추출 결과 분석

벽면의 경우 1번 케이스는 실제 환경과 달리 왼쪽 위 모서리에 빈틈이 발생하였다. 포인트들을 rectangle 별로 그룹화하는 과정에서 위상적으로 이어진 평면상의 포인트들로 그룹화를 진행하고 이를 기반으로 rectangle을 생성한다는 점을 고려했을 때 장애물 뒤의 벽을 인식하기 위해선 최소한 장애물의 위나 옆으로 가려지지 않은 벽에 대한 포인트들이 필요하다. 1번 케이스의 경우 장애물 뒤의 벽에 대한 포인트가 부족하여 빈틈이 발생하였다. 2번 케이스의 경우 장애물 위쪽의 벽에 대한 포인트가 충분하여 빈틈이 발생하지 않았다. 3번 케이스의 경우 육면체와 동떨어진 방의 형태로 벽이 생성되었는데 이는 프로그램이 평면 위의 점들을 기반으로 rectangle을 생성하는 것에 그칠 뿐 대략적인 형태를 기반으로 합리적인 방의 형태를 유추하는 기능은 없기에 입력 데이터의 오차를 그대로 벽으로 인식해 실제 형태와 유사하지 않은 결과가 나오게 되었다. 4번 케이스의 경우 안정적인 밀도와 하나의 평면으로 거의 완벽한 rectangle의 형태를 만드는 포인트들로만 이루어져 있기에 최초 데이터와 흡사한 결과가 나타났다.

- 장애물 Rectangle 추출 결과 분석

장애물의 경우 1번 케이스는 실제 장애물과 상대적으로 유사한 형태가 나왔지만 2, 3번 케이스는 그렇지 않았다. 1번 케이스의 경우 입력 데이터의 장애물이 각 면이 뚜렷한 rectangle의 형태를 잘 나타내어 주었다. 그에 비해 2, 3번 케이스는 장애물의 형태가 잘 나타나지 못하였다. 이는 장애물 주변의 오차 포인트들까지 사각형에 합병되어 사각형의 회전 각도나 범위가 잘못 인식된 것이 원인으로 판단된다. 4번 케이스의 경우 벽면과 동일한 이유로 최초 데이터와 흡사한 결과가 나타났다.

3.2.3. 바닥, 벽, 장애물 구분 및 3차원 공간 구현

실제 공간의 크기는 가로 900mm, 세로 900mm, 높이 300mm이며, 장애물의 크기는 가로 340mm, 세로 250mm, 높이 200mm이다. 그림 17에서 실제 공간과 가상 공간을 비교해볼 수 있다.

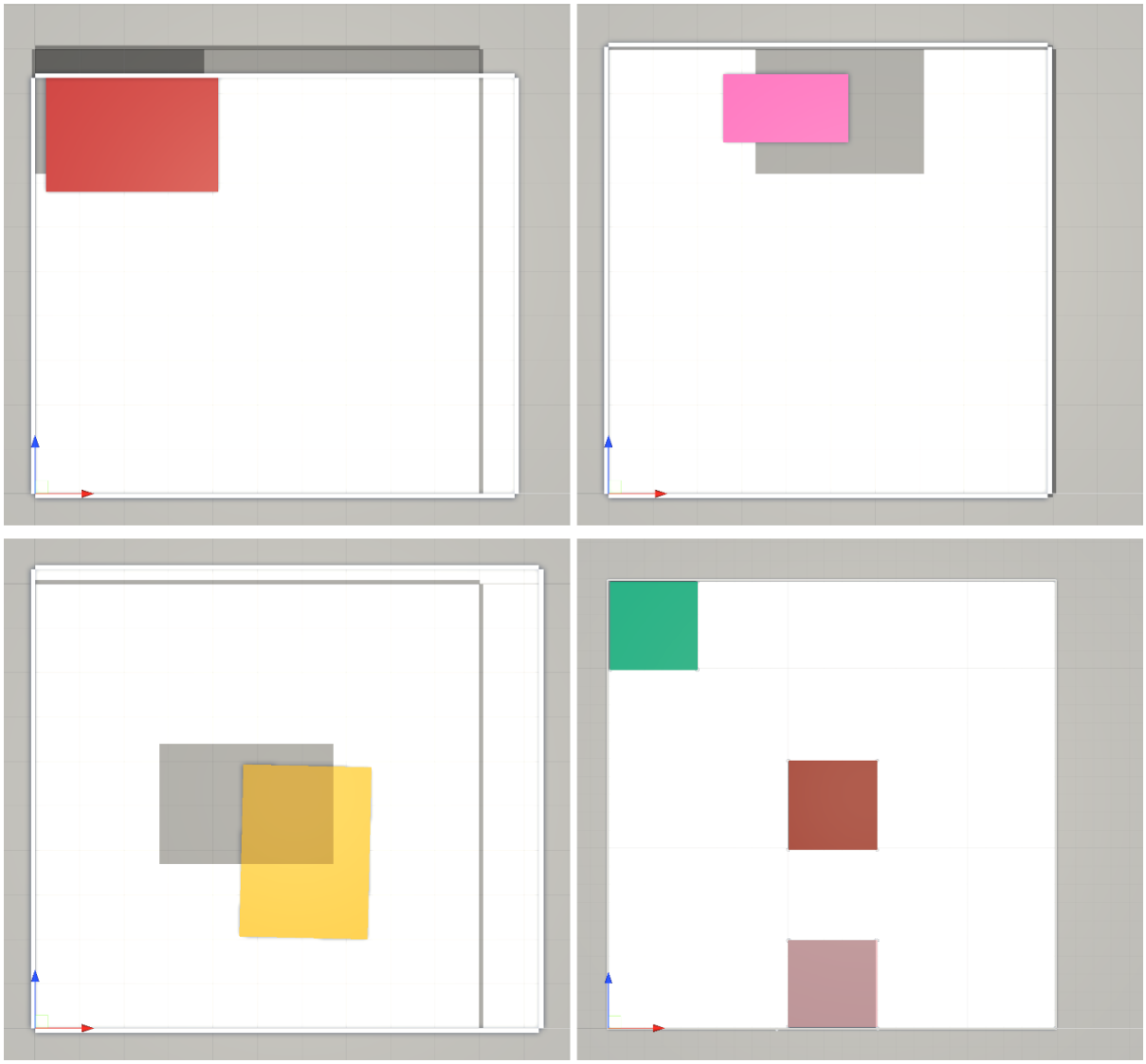

그림 17 실제 공간과 가상 공간을 나타낸 평면도.
실제의 벽과 장애물을 반투명 검정색 Cube로 표시하였다.

생성된 가상 공간과 장애물에 대해 실제 공간과 길이를 비교한 결과는 아래와 같다.

	실제 공간	1번 케이스	2번 케이스	3번 케이스
가로	900	970	890	816
세로	900	842	905	743
높이	300	348	306	268

표 1 가상 공간의 길이 비교(mm)

	실제 공간	1번 케이스	2번 케이스	3번 케이스
가로	340	348	254	208
세로	250	231	139	279
높이	200	227	276	155

표 2 장애물의 길이 비교(mm)

(실제)/(측정)	1번 케이스	2번 케이스	3번 케이스
X	170/197	470/359	430/437
Y	775/726	775/780	455/285

표 3 장애물의 중심 좌표(mm) (xy 좌표 기준)

가상 공간과 장애물에 대해 가로, 세로, 높이와 장애물의 중심 좌표를 실제 공간과 비교하였다. 참값(실제 공간의 수치)과 측정값(가상 공간의 수치)에 대해 다음과 같이 오차를 측정하였다.

$$오차\ 백분율 = \frac{\left|참값 - 측정값\right|}{참값} \times 100\ (\%)$$

	1번 케이스	2번 케이스	3번 케이스
공간 길이	10.13	1.20	12.43
장애물 길이	7.77	36.02	24.35
장애물 중심 좌표	11.11	8.59	19.51
총 오차 평균	9.49	16.10	18.67

표 4 오차 백분율 평균(%)

실제 공간과 가상 공간을 함께 나타난 평면도 그림 17을 보면 임의로 이상적인 맵을 만들어 구현한 4번 케이스는 실제 공간과 가상 공간이 일치하여 겹쳐 보이는 것과 달리 실제 공간을 LiDAR 센서로 스캔한 데이터로 만든 1, 2, 3번 케이스는 크기가 같은 공간과 장애물을 썼음에도 불구하고 모두 차이를 확인할 수 있다. 비교적 실제와 유사하게 나타난 1번 케이스는 점들이 실제 공간과 매우 유사하게 분포했지만, 2번과 3번 케이스에서는 실제로는 빈 공간인 위치에도 점들이 꽤나 분포하여 장애물이 1번 만큼 유사하게 생성되지 못했다. 그러나 2번 케이스의 경우 공간의 길이에 대해 벽을 이루는 rectangle들 집합이 실제와 비슷하게 배치되어 있어 가장 오차가 적게 나타났다. 2번과 3번 케이스는 데이터 수집 과정에서 카트의 이동이 포함되어 1번 케이스에 비해 왜곡된 부분이 있었기 때문에, 장애물의 구현에 특히 영향을 미친 것으로 추측된다.

4. 결론 및 향후 연구 방향

연구를 통해 데이터 수집, rectangle 추출, 시각화 총 세 개의 단계에 걸쳐 LiDAR 센서를 부착한 카트를 이용하여 3차원 가상공간을 생성하는 시스템을 구축하였다. 그 결과 일부의 단순한 배치를 가진 구조의 경우 상대적으로 실제와 비슷한 결과가 생성되었으나 대부분의 경우 실제와 동떨어진 결과물이 생성되었다. 이러한 결과의 원인과 그 외 부실한 부분들을 고려했을 때 단계 별 향후 연구 방향은 다음과 같다.

데이터 수집 단계의 경우 데이터 수집 결과와 멘토 의견서의 내용을 고려하였을 때, 더 다양한 기능을 제공해 주고 높은 정밀도를 제공해 주는 LiDAR 센서의 도입이 필요하다. 데이터 수집 중 SLAM 결과에서 오차를 발생시키는 가장 큰 원인으로는 실제 카트의 회전 및 이동과 ROS에 입력되는 회전 및 이동 예상값의 오차로 분석되었다. 이러한 점들을 고려했을 때 자체적인 센서를 통해 이동 및 회전을 측정해 주는 LiDAR 센서의 도입이 필요하다고 판단된다.

Rectangle 추출 단계의 경우 데이터 수집의 오차를 극복하고 더 나아가 단순한 cube형태의 구조를 벗어난 복잡한 형태의 구조까지 변환할 수 있도록 기능을 향상시킬 필요가 있다. 이를 위해선 데이터의 오차를 극복하기 위한 추가적인 단계나 기존 단계의 개선이 연구되어야 한다. 추가로 기하학적 분석 대신 CNN 등의 머신러닝 기법을 활용하는 방안도 연구되어야 한다.

시각화 단계의 경우 현재 변환된 cube 데이터들을 Unity 오브젝트를 통하여 실시간 렌더링하는 기능만이 존재하기에 결과물을 fbx등의 범용 확장자 파일로 출력하는 기능이 구현될 필요가 있다.

마지막으로 현재 rectangle 출력 및 시각화 단계에서는 이전 단계의 출력 결과를 사전에 처리되어 저장된 파일을 읽어 처리하는 방식으로 되어있다. 이러한 방식 외에 전 단계에서 처리된 결과를 프로세스간 통신을 통해 실시간으로 받아 처리하는 방식이 추가될 필요가 있다.

5. 개발 일정

개인 일정은 노란색, 공통은 회색으로 표시

- 홍주혁

<table>
<tr><th colspan="4">6월</th><th colspan="4">7월</th><th colspan="4">8월</th><th colspan="4">9월</th><th colspan="2">10월</th></tr>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th></tr>
<tr><td colspan="4">ROS, SLAM 관련 기술 스터디</td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="3"></td><td colspan="2">SLAM 구현</td><td colspan="3"></td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4">카트 조작 프로그램 개발</td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="3">중간보고서 작성</td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4">알고리즘 구현 지원</td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="3"></td><td colspan="4">카트, 테스트 공간 제작</td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="3">테스트 및 성능 평가</td><td colspan="1"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="3"></td><td colspan="2">알고리즘 수정</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="3"></td><td colspan="2">최종보고서 작성</td></tr>
</table>

표 5 개발 일정 (홍주혁)

- 우현우

<table>
<tr><th colspan="4">6월</th><th colspan="4">7월</th><th colspan="4">8월</th><th colspan="4">9월</th><th colspan="2">10월</th></tr>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th><th>3</th><th>4</th><th>1</th><th>2</th></tr>
<tr><td colspan="4">RANSAC, 계산 기하학 관련 기술 스터디</td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4">Rectangle 추출 프로그램 개발</td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="3">중간보고서 작성</td><td colspan="4"></td><td colspan="4"></td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4">프로그램 성능 개선</td><td colspan="2"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="3">테스트 및 성능 평가</td><td colspan="1"></td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="3"></td><td colspan="2">최종보고서 작성</td></tr>
</table>

표 6 개발 일정 (우현우)

- 남예진

6월				7월				8월				9월				10월	
1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
계산기하학, Unity 관련 기술 스터디																	
				벽, 장애물 구분 알고리즘 구현													
						중간보고서 작성											
							유니티 시각화 프로그램 개발										
										알고리즘 수정							
													테스트 및 성능 평가				
															최종보고서 작성		

표 7 개발 일정 (남예진)

6. 참고 문헌

[1] GitHub CygLiDAR-ROS/cyglidar_d1. Available: https://github.com/CygLiDAR-ROS/cyglidar_d1
[2] OctoMap Document. Available: https://octomap.github.io/
[3] GitHub Octomap/octomap_mapping. Available: https://github.com/OctoMap/octomap_mapping
[4] GitHub OctoMap/octomap. Available: https://github.com/OctoMap/octomap
[5] Hands-on with RPi and MPU9250 Part 3. Available: https://medium.com/@niru5/hands-on-with-rpi-and-mpu9250-part-3-232378fa6dbc
[6] rospy_tutorials - ROS Wiki. Available: https://wiki.ros.org/rospy_tutorials