C : 네트워크/시스템

실내 지도 데이터 생성 로봇

지도교수 : 김원석

참새크면비둘기조

201724611 최호진

201724589 조창현

201924650 박지호

<목차>

1 서론 3
1.1 연구 배경
1.2 연구 목표
2 연구 환경 3
2.1 데이터 수집
2.2 Github
3 연구 내용 4
3.1 데이터 수집 장치
3.2 서버
3.3 데이터 전처리
3.4 3D 모델링
4 연구 결과 분석 및 평가 15
5 결론 및 향후 연구 방향 16
6 구성원별 역할 및 개발 일정 18
7 참고 문헌 20

1. 서론

1.1. 연구배경

최근 군에서 위치 인식과 측정을 통해 계측한 데이터를 토대로 3D 모델을 생성하는 로봇이 개발 및 연구가 진행중이다. 이와 비슷한 기술이 적용된 예로는 로봇청소기를 들 수 있다. 이러한 로봇들에서는 정확한 계측을 위해서 정교한 센서들을 많이 요구되므로 비용이 올라가게 된다. 따라서 본 과제를 통해 해당 기술을 직접구현 및 제작해보며 해당 기술을 간접적으로 체험해보고자 해당 주제를 선정하였다.

1.2. 연구 목표

직접 측정가능한 장치만들고 데이터를 분석하여 3D모델링을 하는 것으로 해당 과제를 풀고자 한다. 아두이노를 기반으로 하고 거리를 측정하기 위해 Lidar 센서를 사용한다. 측정된 데이터를 기반으로 다음 위치를 파악하고 해당 위치로 이동하여 방 전체를 측정할 때까지 이 과정을 반복한다. 이렇게 받아온 데이터는 평준화를 위해 전반적인 처리과정을 겪어 다듬어준다. 전처리된 데이터는 Unity에서 읽어들여 모델링을 시켜준다. 또한 만들어진 모델링들은 각각 저장되어 원하는 모델을 볼 수 있도록 한다.

2. 연구 환경

2.1. 데이터 수집

알고리즘 및 데이터 처리를 위해 테스트용 데이터를 수집해야 했다. 테스트용 데이터는 117 X 230 (cm) 의 임의의 공간에서 수집하였다. 측정 해야하는 케이스마다 장애물을 두어 더욱 다양한 데이터를 수집하도록 하였다. Lidar센서를 통한 데이터는 서브모터를 통해 360°로 측정하여 회당 235개의 데이터를 얻는다. 이를 전처리과정을 통해서 3D 모델을 만들기 적합한 형태로 가공해준다.

2.2. Github

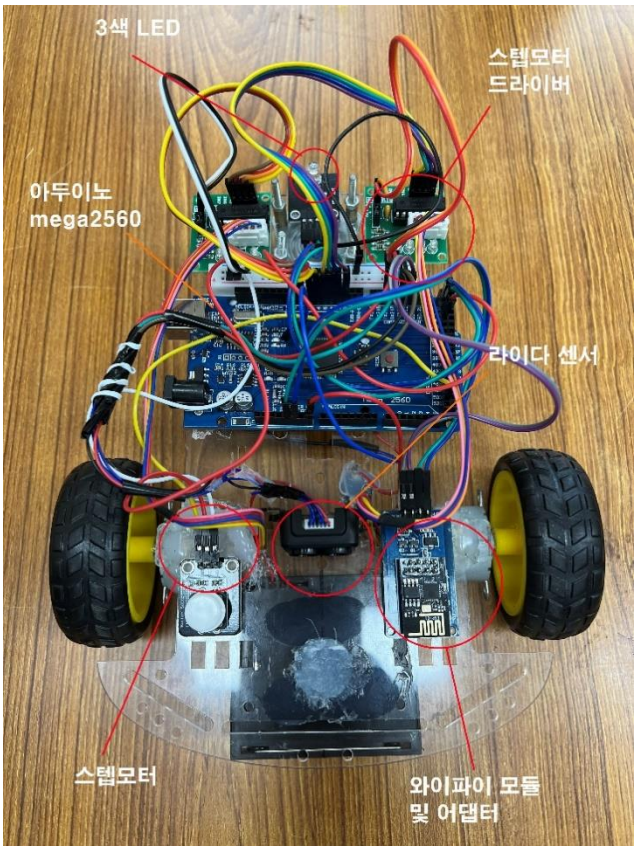
팀원 간의 효율적인 데이터 공유 및 협업을 위해 Github을 주로 사용하였다. 서로간의 상호작용이 자주 일어나는 과제이기에 파일을 쉽게 접근이 가능한 Github repository를 저장공간으로 활용하였다.

3. 연구 내용

3.1. 데이터 수집 장치

앞서 데이터 수집에서 언급하였듯이 Lidar 센서를 통해 장치와의 거리를 측정한다. 이는 오픈 소스를 기반으로 한 단일 보드 마이크로컨트롤러 보드인 아두이노를 바탕으로 사용하였다. 측정한 데이터는 Json형태로 서버로 전송하게 되며 서버에서 명령어를 입력받아 작동하게 된다.

1. 모듈 설계

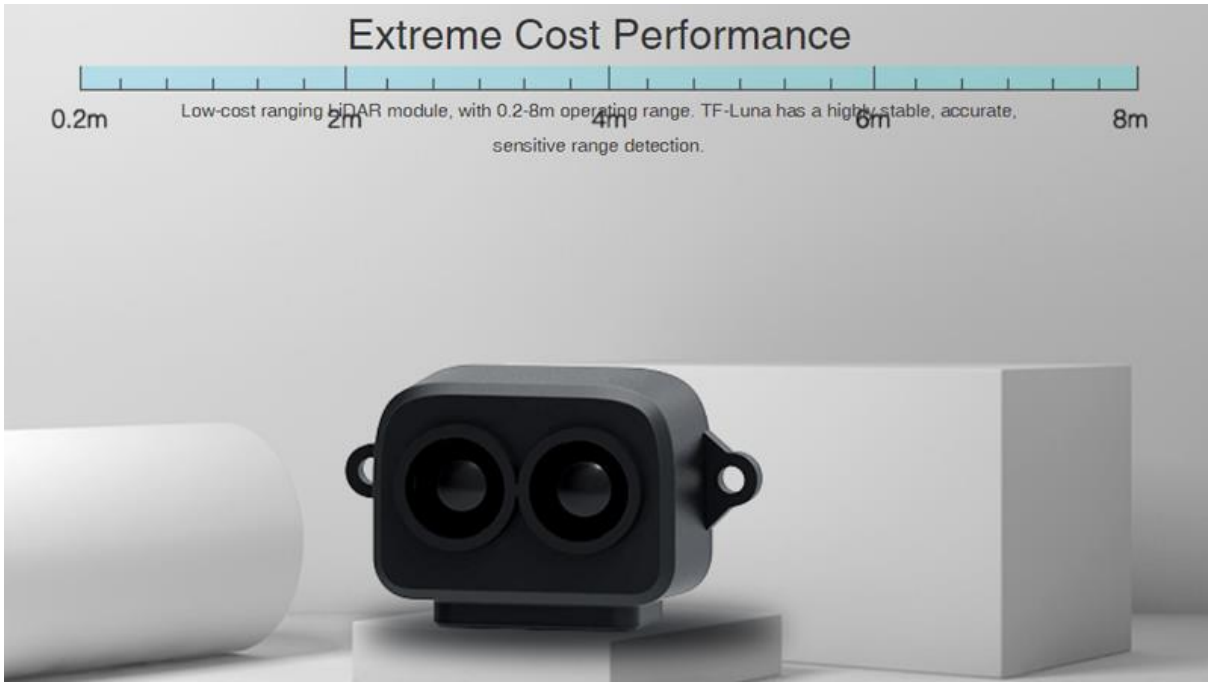


[데이터 수집 장치]

- LiDAR

실내 측위를 하기위한 모듈로 카메라, LiDAR 등을 활용하게 되며 본 과제에서는 LiDAR 센서를 활용할 것이다. LiDAR 센서의 경우 자율주행차량에도 장착될 만큼 정확도면에서 큰 장점을 가지고 있다. 또한 카메라를 활용하게 되면 어두운 환경이나 카메라 인식이 불안정한 환경에선 활용이 불가능하며 개인 프라이버시의 문제점이 있으나 LiDAR를 활용하게 되면 평면 이미지로 파악이 불가능한 깊이 정보 파악, 길이 값만 나오므로 개인 프라이버시 문제 해결 및 카메라를 활용하지 못하는 환경에서도 활용이 가능하므로 많은 가능성이 있을 것으로 생각되어서 LiDAR 센서를 채택하게 되었다.

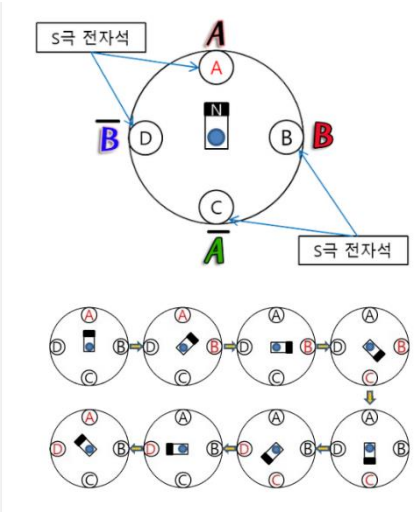
LiDAR 센서로 저가형 one point LiDAR를 활용하였으며 장치를 360도 회전하는 것으로 View를 360으로 만들어서 값을 읽어올 것이다.



0.2m 최대 8m까지 지원하는 모델을 사용하였다.

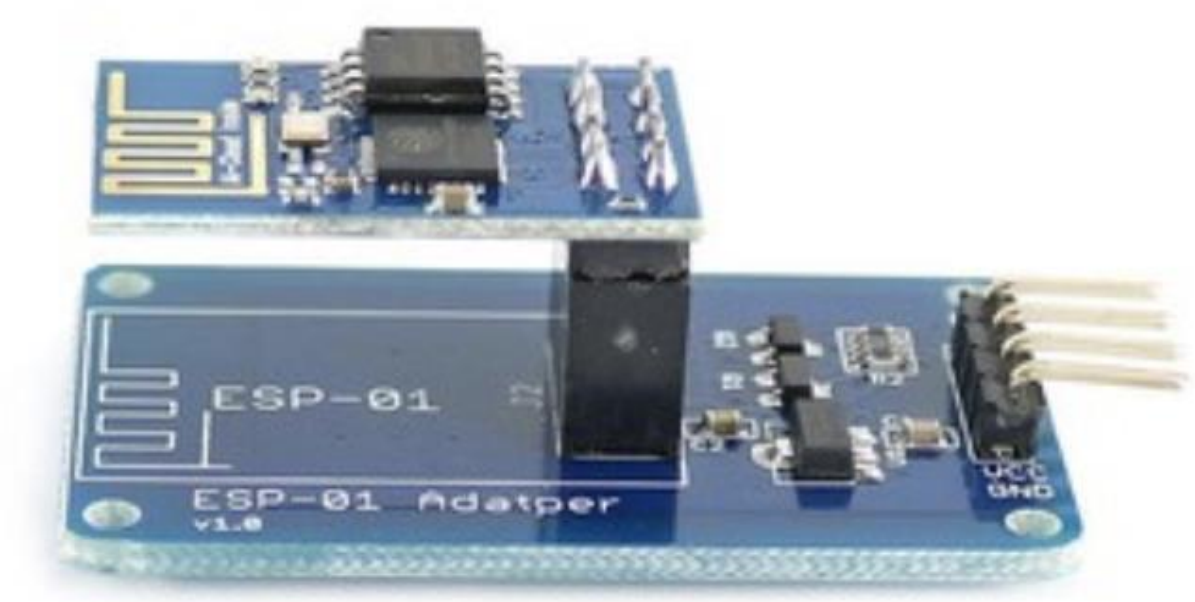
- 스텝모터 + 스텝모터 드라이버

장치의 이동을 위해서 DC모터가 아닌 스텝모터를 활용하였다. DC모터를 활용하게 되면 장치의 회전각이나 이동거리에 대해서 판단하기가 매우 어렵다. 그러나 스텝모터는 각 스텝별로 동작하기 때문에 한바퀴 회전하는 스텝을 활용하게 되면 장치를 특정 각도로 회전이 가능하며 이동거리 또한 파악이 가능하다.



- WiFi 모듈(esp01) + 어댑터

장치와 서버간 통신을 하기위한 모듈로 장치는 서버로 측정한 결과값을 보내주고 서버는 받을 결과 값을 전처리 후 다음 명령어를 장치에 전달하게 된다. http 통신을 통하여 서버로 Json을 전달하고 명령어를 받아온다.



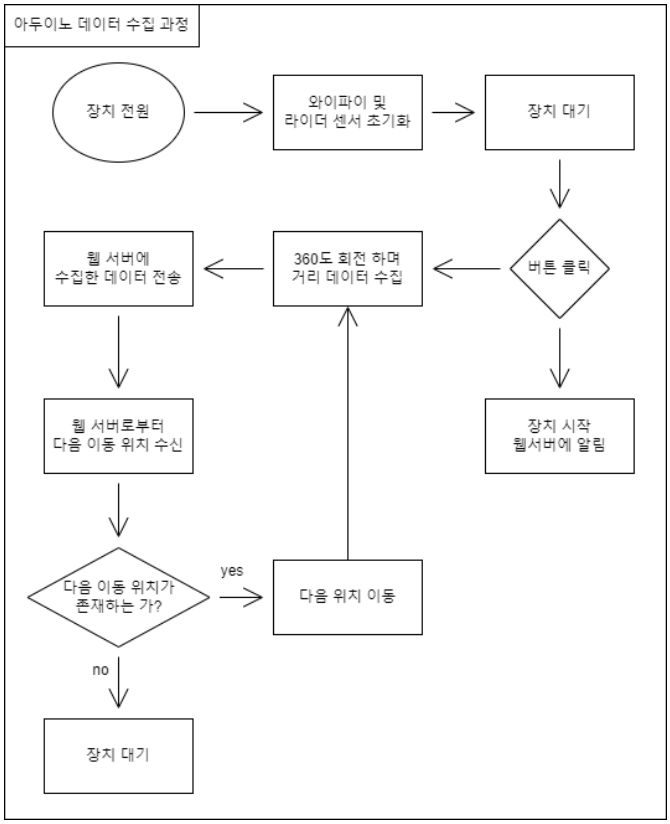
- 기타장치(버튼, LED, 배터리팩)

- 무선으로 장치를 동작하기 위해 전원공급을 위한 배터리팩
- 장치 시작을 위한 버튼
- 장치의 현재 상태를 파악하기 위한 LED

LED 색에 따른 장치 상태	
빨강	최초 시작 대기상태 및 에러
파랑	데이터 수집 중
녹색	웹서버 데이터 송수신 중 및 측정 완료 후 대기상태
보라	장치 이동 중

2. 데이터 수집 과정

스텝 모터 한바퀴를 회전하기 위해선 2048 step이 필요하게 된다. 그러나 장치의 무게 및 바퀴의 마찰력으로 인해서 테스트하여 확인해본 결과 장치를 360도 회전시키기 위해서는 총 4700 step이 필요하게 되며 이를 양쪽 스탭 모터 인가 시에 장치가 1회전 한다. 이를 이용해 주변 360도의 거리 데이터를 라이더 센서를 통해 얻는다. 1회전 시 4700을 20으로 나누어 총 235개의 거리 데이터를 가져오게 되며 데이터 사이의 각도는 360/235 = 1.5 도이며 거리 데이터를 전송하게 되면 서버에서 이를 X,Y 좌표로 환산하게 된다. 데이터 수집 사이클은 다음 다이어그램과 같다.



3. 길 찾기 알고리즘

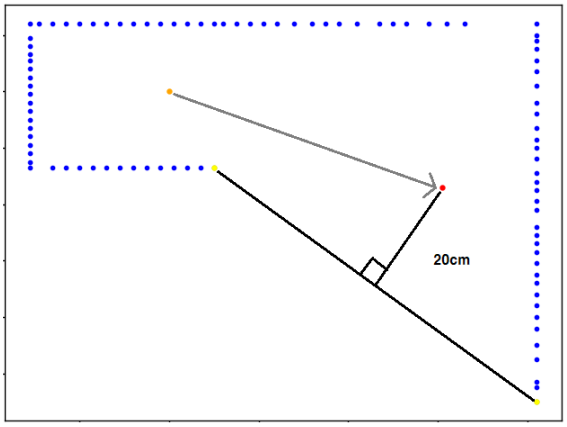
데이터 전처리 과정에서 나온 끊어진 점을 활용하여 다음 이동 위치를 특정한다. 끊어진 양 끝점의 중간 점에서 반시계 방향으로 장치 회전을 위해 20cm 떨어진 지점을 벡터 회전을 통해 반환하게 된다.

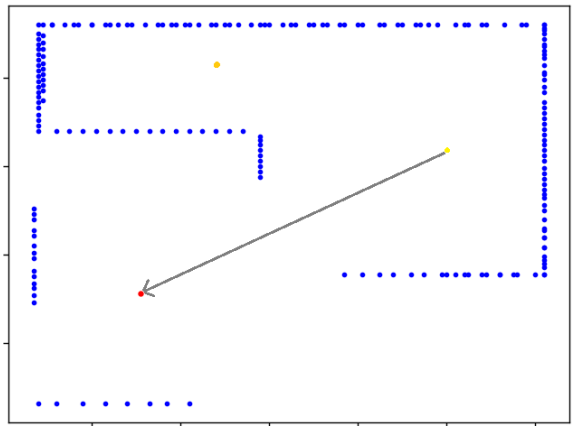
$$gap = (((x_1 - x_2)^2 + (y_1 - y_2)^2)^{1/2}$$

$$\theta = \mathrm{atan}(20/gap)$$

$$x = \frac{(x_2 - x_1)}{2}\frac{1}{\cos(\theta)}\cos(\theta) - \frac{(y_2 - y_1)}{2}\frac{1}{\cos(\theta)}\sin(\theta) + x_1$$

$$y = \frac{(x_2 - x_1)}{2}\frac{1}{\cos(\theta)}\sin(\theta) - \frac{(y_2 - y_1)}{2}\frac{1}{\sin(\theta)}\sin(\theta) + y_1$$





측정 이후 이전 지도 데이터를 참고하여 현재 끊어진 좌표가 이전 지도 데이터에 존재하는 경우, 다음 이동위치에 대한 지점에서 배재하게 되며 위의 과정을 반복하여 전체 지도를 구성하며 서버에서 끊어진 점이 없다고 명령어가 전달하게 되면 장치는 측정을 중지하게 되고 대기상태에 들어가게 된다.

3.2. 서버

데이터 수집 장치 자체적으로 데이터 저장, 길 찾기 알고리즘, 종료 시점을 판단하기에는 무겁다고 판단되었고 만일 알고리즘에 문제가 생겼을 경우 수정이 어렵기 때문에 서버를 활용하여 서버에서 알고리즘이 동작하게 된다면 서버상 알고리즘을 수정하면 되기 때문에 유지 보수 및 관리가 쉽다. 그러므로 장치-서버간 통신을 활용하여 데이터 수집 장치는 오직 데이터만 수집하고 특정 명령어만 수행하게 된다.

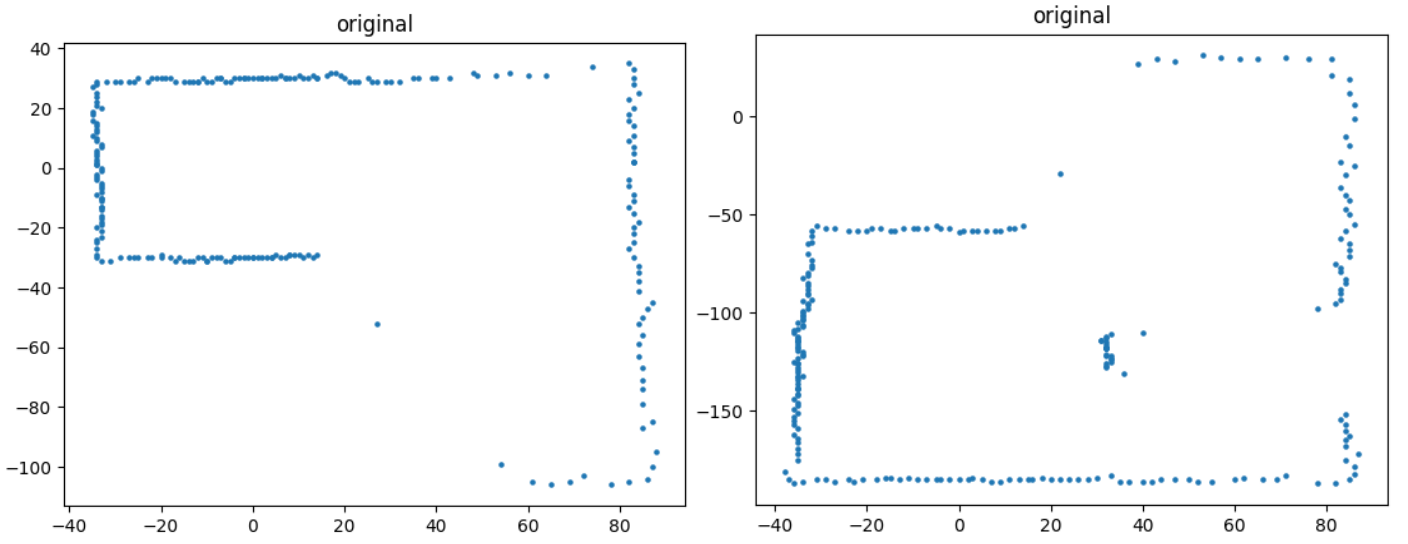
nodejs기반 Web서버를 사용하며 장치가 측정한 데이터는 서버를 통해 배포 될 수 있도록 하며 앞서 언급된 이동 알고리즘을 서버에서 동작 시켜 장치가 가야할 다음 위치도 보내준다. 기존에 사용하기로 했던 AWS에서는 CPU 성능 문제로 인해 서버가 주기적으로 다운되는 현상을 확인하여 AWS보다 CPU 성능이 우수한 NAS로 교체하여 동작시켰다.

또한 완성된 모델을 보여주기 위한 인터페이스로서 Unity를 활용하였으며 WebGL로 컴파일하여 Web서버에서도 동작할 수 있게 하였다. 이를 통해 별도의 툴 설치 필요없이 웹 주소만으로 쉽게 접근이 가능하고 이용할 수 있도록 한다.

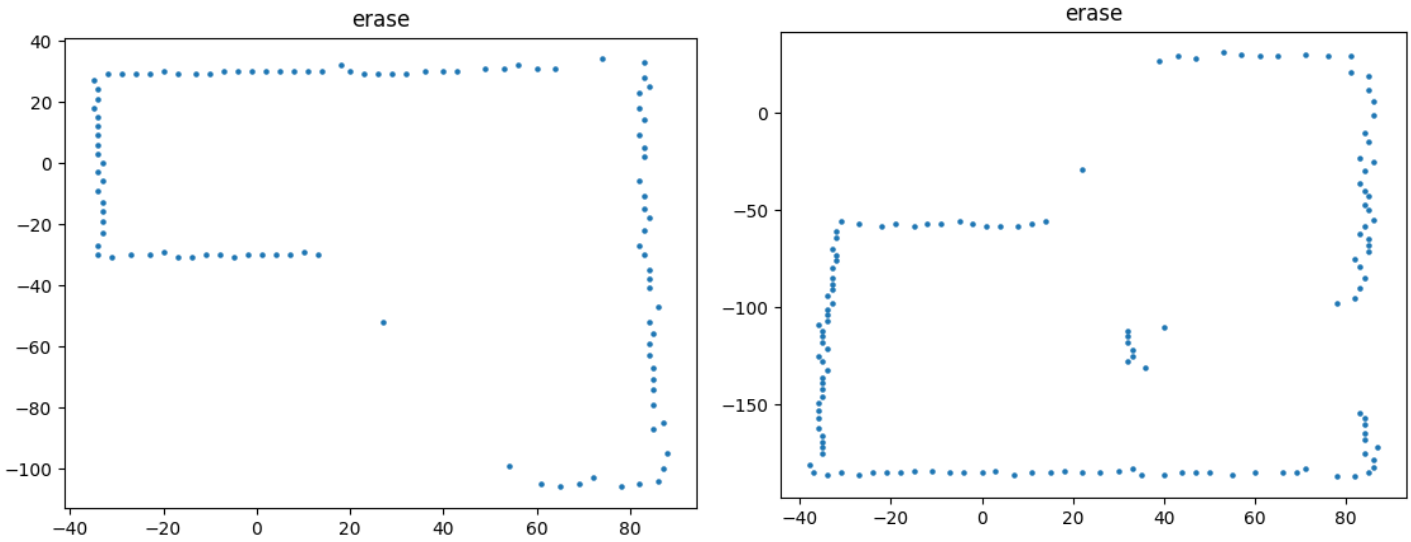
3.3. 데이터 전처리

1. 과정

- Downsampling



[Original]

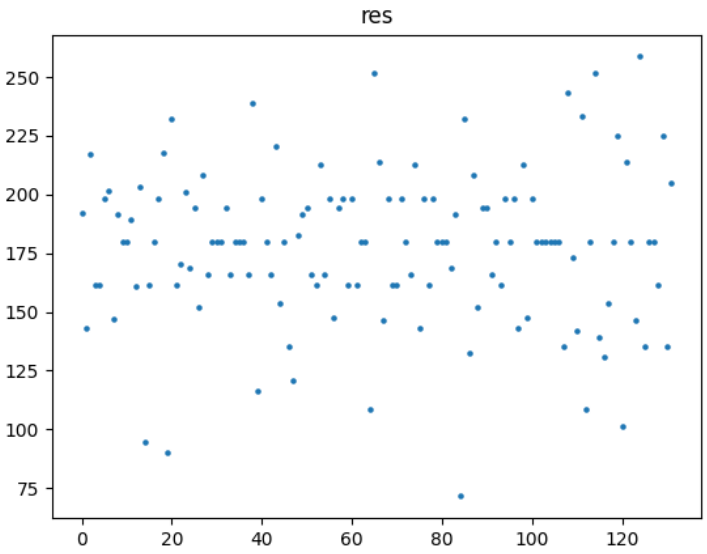


[Downsampling]

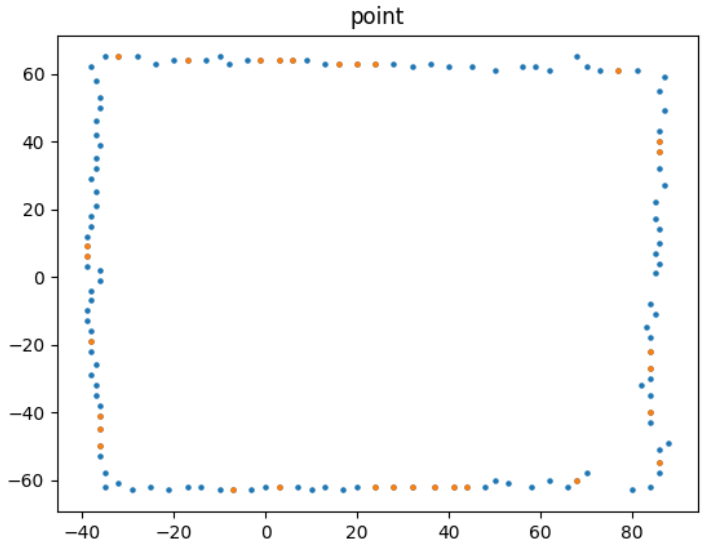
원본 데이터의 경우 짧은 거리에서 측정 시 점들이 모이게 되어 쓸모 없는 값이 존재하게 된다. 또한 장치의 진동으로 인하여 꼭지점의 위치도 불분명해지는 현상을 발견하여 직선을 활용하여 꼭지점을 예측할 것이기 때문에 Downsampling을 진행하였다.

- 내각 측정

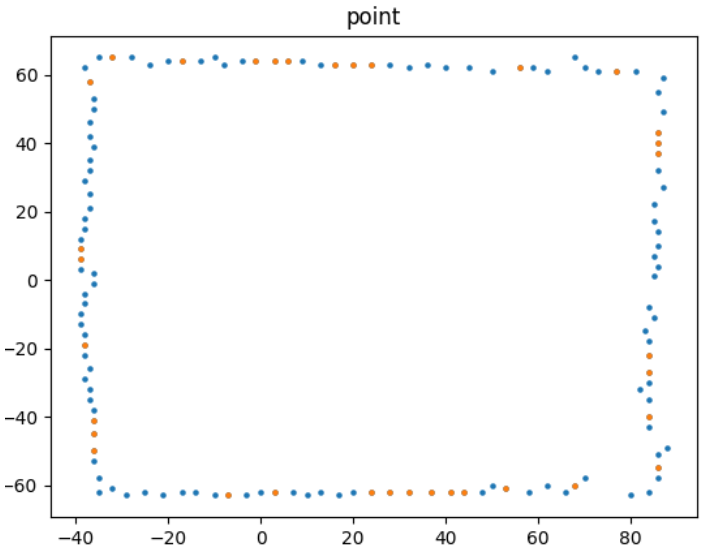
직선을 활용하여 꼭지점을 예측할 것이기 때문에 직선으로 예상되는 점을 특정할 필요가 있다. 점을 특정하기 위해 세 점을 활용하여 내각을 측정하였다.



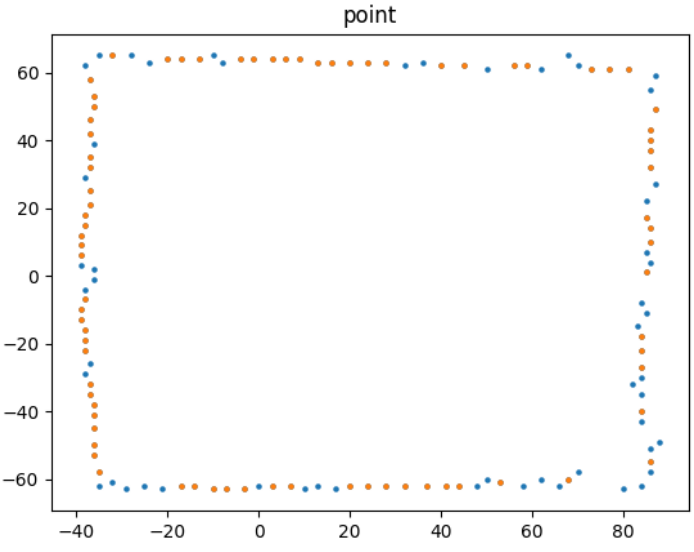
[세 점을 활용한 내각 값]



[res = 180인 Point]



[170 < res < 190인 Point]]



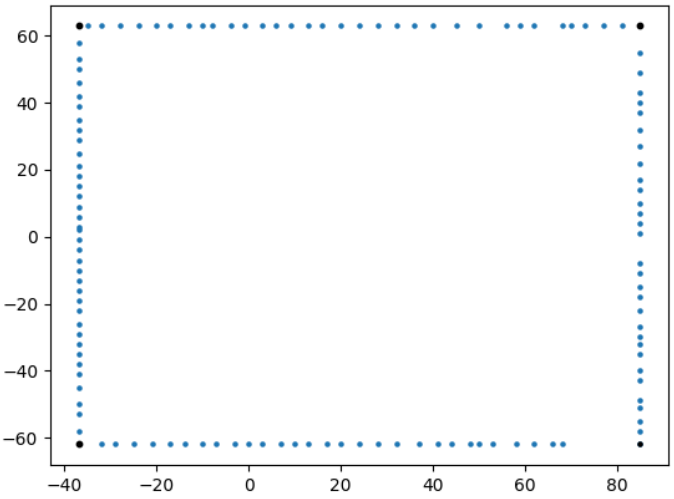
[160 < res < 200인 Point]

Res의 범위에 따라 확인해본 결과 160 < res < 200일 경우 직선인 Point의 집합을 잘 찾는 것을 확인할 수 있다.

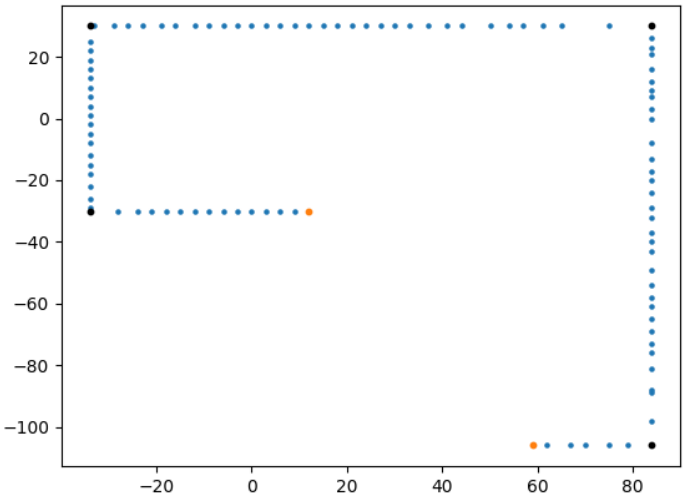
● 꼭지점 및 끊어진 점 탐색

직선을 확인하기 위해 Point의 집합에서 X, Y 변화량을 확인하여 한 직선에 대한 Point들을 모아서 Curvefitting을 진행하여 직선의 방정식을 알아낸 후 원본 Point들을 정렬하고 직선 간 교점을 확인하고 근처에 해당 점이 존재하면 해당 점을 꼭지점으로 판별하여 꼭지점을 반환한다.

또한 점들 간의 거리가 X, Y 변화량 크게 변화하는 경우를 끊어진 점으로 판단하고 장치는 해당 점을 활용하여 다음 탐색 지점으로 이동하게 된다.

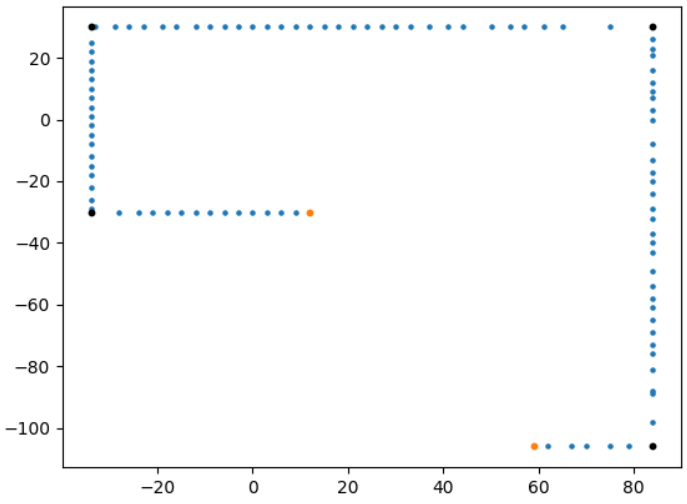


[꼭지점]

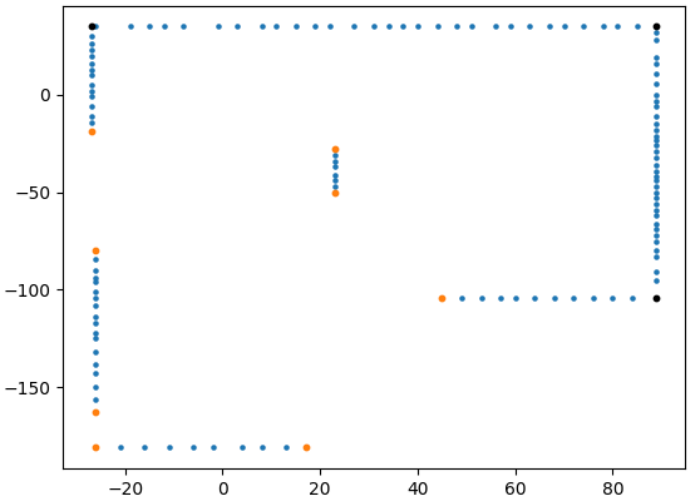


[꼭지점 + 끊어진 점]

● 데이터 정합

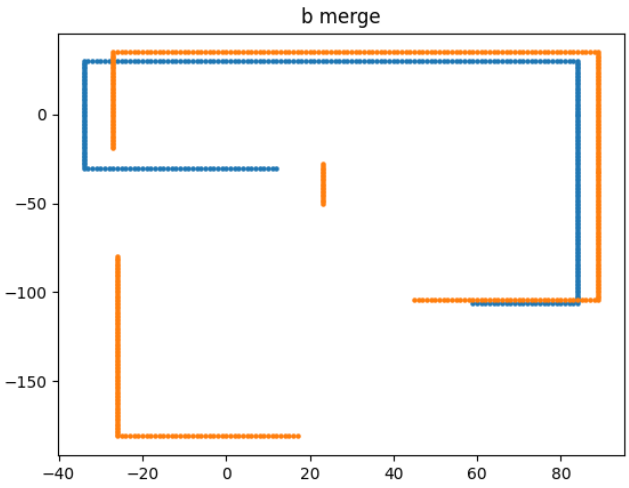


[1번 Point set]

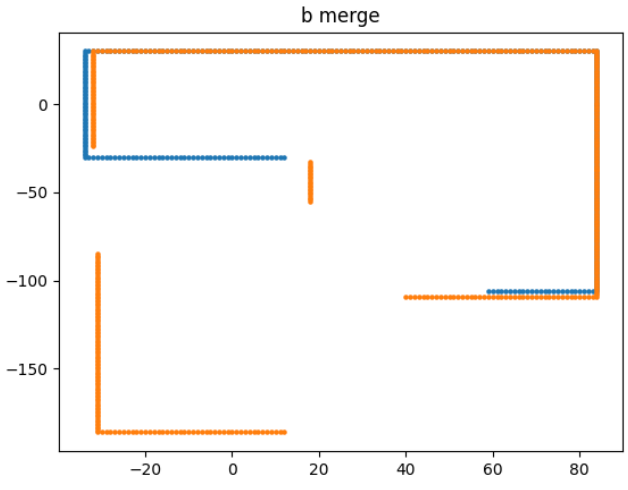


[2번 Point set]

최종 결과물을 만들기 위해서는 데이터들을 정합 할 필요가 있다. 장치의 위치와 관계 없이 데이터를 정합하기 위해서 특정된 꼭지점들을 비교하여 얼마나 차이가 있는지를 확인하여 1번 set을 기준으로 2번 set을 옮겨 온다.

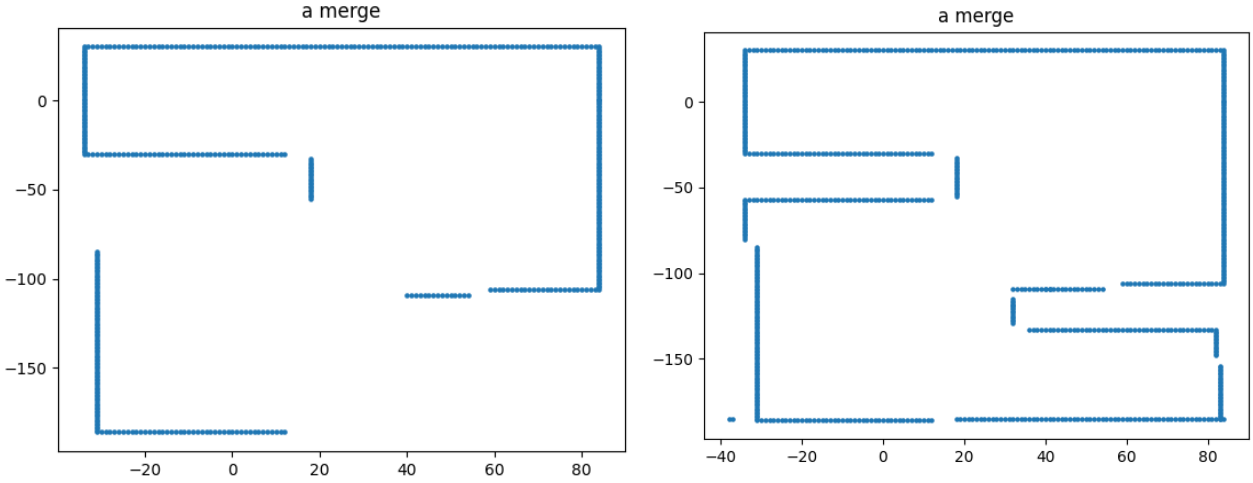


[이동 전]



[이동 후]

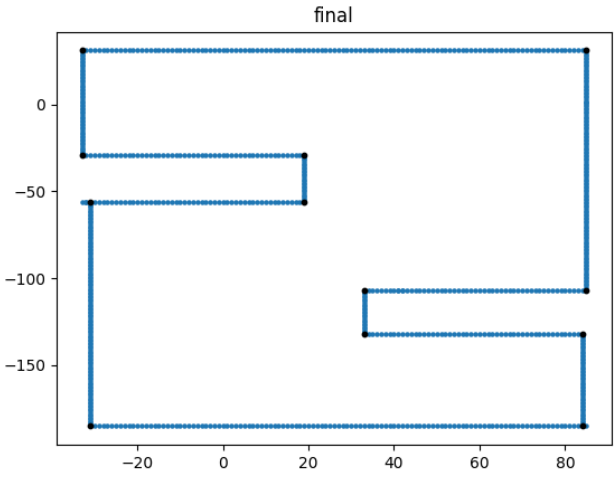
옮긴 이후 근처의 점이라고 판단되는 점들을 제거하여 하나의 set으로 정합한다. 정합이 완료될 때마다 Unity에서 완료된 set을 확인 할 수 있다.



[정합 후]

끊어진 점이 사라질 때까지 해당 과정은 반복되며 끊어진 점이 사라지게 되면 해당 set을 정렬시킨다.

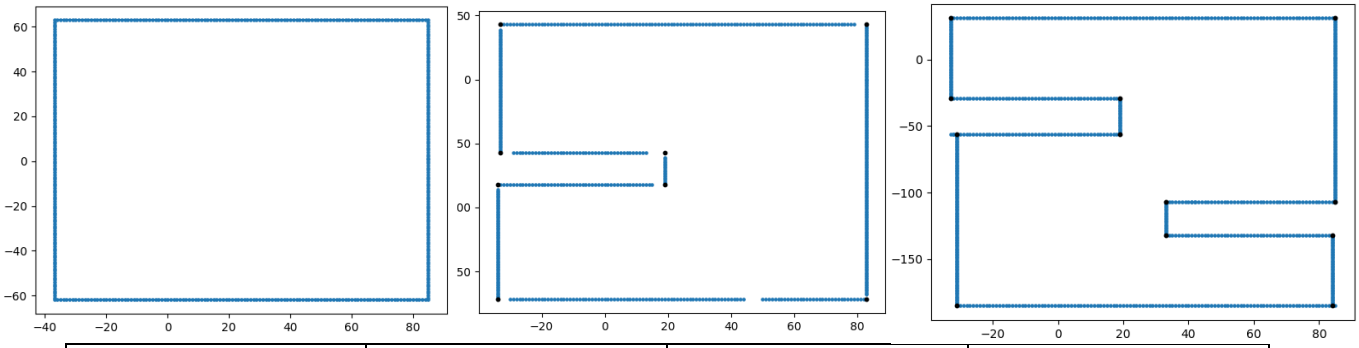
정렬 후 위의 과정을 거치게 되면 아래의 그림처럼 나오게 된다.



[최종 결과물]

2. 검증

ㅁ, ㄷ, ㄹ 형태의 실내 모형을 만들어 테스트하였으며 각각 최종 결과물은 아래와 같다.



(실제)/(측정)	ㅁ	ㄷ	ㄹ
가로	117/114	117/120	117/121
세로	112/116	230/240	230/245
장애물	X	50/48	50/48, 45/42

실제 값과 측정 값 사이에 3 ~ 15의 오차 범위를 가지게 된다.

단순한 형태에선 오차 범위가 적게 나오지만 장애물이 생기고 측정을 여러 번 하게 되는 경우 오차가 커지게 되며 최대 10의 오차가 발생하게 된다. 이는 정합 과정에서 발생하게 된다. 오차가 발생하는 원인을 분석해본 결과 기본적으로 LiDAR 센서는 진동에 민감하여 값이 거나 차체의 LiDAR의 위치 또한 정중앙에 위치하지 않아서 data set간의 오차가 발생하는 것으로 분석된다.

3.4. 3D 모델링 준비

수집된 데이터를 모델링하기 위한 툴로 Unity를 활용한다.

데이터 전처리가 완료된 Json파일은 Unity에서 3D Object를 만들 준비를 한다. 장치에서 수집된 LiDAR 데이터셋를 서버에 전송하게 되면 전처리과정을 거쳐 정재된 Json을 저장하고 해당 Json 데이터를 읽어와 X,Y 좌표에 Prefab에 저장된 Block 쌓아 연결시킨다. 이렇게 완성된 모델은 InputText에 측정한 방의 이름을 입력하면 Database에 해당 이름으로 읽어온 Json파일이 저장된다.

1. 완성본 저장

Unity에서 완성된 모델을 Database에 저장하여 관리한다. Web 상에서 Database에 저장된 데이터를 List화 하여 표시하고 사용자가 원하는 모델을 선택하게 되면 해당 모델을 Database에서 읽어와 해당 모델을 사용자에게 제공한다.

Unity에선 완성된 모델링을 저장하기 위해 MySQL과 동일한 소스 코드를 기반으로 하는 Mariadb를 사용한다. Mariadb에는 2개에 Table이 저장되어 있으며 모델의 이름을 나타내는 'Name'과 불러올 모델의 Json 파일이 저장된 'LidarData'가 있다.

Dropdown으로 현재 Mariadb에 저장된 목록의 'Name' Table을 나열한다. 이전에 완성된 모델을 보기위해 목록 중 하나를 선택하였으면 Load 버튼을 통해 Json파일인 'LidarData' Table을 불러온다. 해당 데이터는 Unity 상에서 읽어와 3D Object로 만들어 사용자에게 제공한다.

4. 연구 결과 분석 및 평가

1. 정합 알고리즘

정합 알고리즘은 정확도를 높이기 위해 배열을 하나씩 탐색하게 되며 N개의 데이터셋을 탐색하게 되면 시간 복잡도는 N의 제곱승으로 증가하게 되어 시간 복잡도가 기하급수적으로 증가하는 형태를 보인다. 또한 LiDAR의 장점으로는 낮은 오차가 있는데 본 과제에 적용된 알고리즘을 사용하게 되면 오차가 상당히 많이 나는 것을 확인할 수 있다. 또한 정확도가 어느정도 보장된다 하더라도 자율

주행자동차나 실시간으로 LiDAR 데이터가 필요한 경우 실시간을 보장을 못하게 되는 단점이 있어 이를 보완할 필요가 있다.

2. 테스트 환경

정해진 환경에서 장치를 테스트하게 되어 해당 환경에서는 특정 step으로 장치의 한바퀴를 특정하였지만 다양한 환경에서는 적용이 불가능하다. 이를 보완하기 위해선 나침반센서를 활용하여 장치를 360도 회전을 보장할 수 있을 것으로 예상된다.

3. LiDAR

해당 과제에서 이용한 LiDAR는 One Point만 반환하다 실시간으로 주변환경 파악이 어려운 점이 있다. 즉, 장치가 360를 회전해야만 주변 환경을 파악이 가능하게 된다. 또한 LiDAR센서는 진동에 민감한데 장치가 회전하면서 진동이 발생하고 이는 데이터셋에 영향을 미치게 된다. 이를 해결하기 위해선 FOV(Field Of View) 즉, 시야각이 넓은 LiDAR를 활용하게 된다면 실시간으로 주변환경을 파악하거나 진동이 발생하더라도 전처리 과정 없이 정합 알고리즘 만으로 모델링이 가능할 것으로 예상된다.

5. 결론 및 향후 연구 방향

우리는 직접 장치를 제작하고 받아온 데이터를 가공하여 이상적인 3D 모델을 만들고 이를 Web을 통해 제공하고자 했다. 데이터 처리 및 장치 이동을 위해 직접 Python을 활용하여 가공하였다. 가공 한 데이터를 Unity를 통해 온전한 모델로 제작하고 이를 Web을 통해 시각적으로 제공함으로써 데이터 전처리, 알고리즘, 서버 제작 과정 전체를 경험해보았다. 완성된 데이터를 저장 및 불러오기 위한 Database도 구축하여 사용해보았다. 빠른 속도로 측정할 수는 없었지만 정확도 적인 부분을 좀 더 높이는 방향으로 진행했다.

비록 본 과제에서는 상당히 많은 전제조건하에 진행되었기 때문에 범용성이 상당히 부족하다. 그러나 이러한 전제조건은 다양한 센서를 활용하면 충분히 극복 가능한 조건이기 때문에 향후 장치에 다양한 센서 및 향상된 센서를 활용하게 되

면 범용성 있게 장치를 활용가능 할 것으로 예상되며 특히 LiDAR 센서를 one point에서 FOV가 넓은 LiDAR 센서를 활용하게 되면 본 과제에 적용되어 있는 전처리 알고리즘을 제거할 수 있으며 실시간으로 주변환경 파악이 가능할 것이다. 추후 다양한 센서 추가 및 FOV가 120 정도인 LiDAR 센서를 활용하여 알고리즘의 시간 복잡도 개선 및 범용성 높은 장치를 제작하여 테스트해볼 계획이다. 또

한 LiDAR를 활용하여 측정한 결과값은 2D 흑백 이미지의 구조와 비슷한 특성을 가지기 때문에 CNN을 활용하게 된다면 성능적인 부분에서 개선될 가능성이 높아질 것으로 예상된다.

6. 구성원별 역할 및 개발 일정

- 최호진

5월			6월					7월				8월					9월			
3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주	3주	4주	5주	1주	2주	3주	4주
모듈 제작																				
		Unity, 아두이노, Web관련 기술 스터디																		
						데이터셋 수집 및 3D 모델링 알고리즘 구현														
											데이터 전처리 구현									
										중간보고서										
													아두이노, Web 지원							
																테스트 및 디버깅				
																	최종 발표 및 보고서 준비			

- 조창현

5월			6월					7월				8월					9월			
3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주	3주	4주	5주	1주	2주	3주	4주
모듈 확인																				
		아두이노 관련 기술 스터디																		
							장치 구현 및 Web 통신													
										실내 탐색 알고리즘 구현 및 데이터 수집										
										중간보고서										
																테스트 및 디버깅				
																	최종 발표 및 보고서 준비			

- 박지호

5월			6월					7월				8월					9월			
3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주	3주	4주	5주	1주	2주	3주	4주
Node.JS, 서버 관련 기술 스터디																				
					Web 제작															
								WebGL, 서버 연동												
										알고리즘 구현 지원										
										중간보고서										
																테스트 및 디버깅				
																	최종 발표 및 보고서 준비			

이름	**역할 분담**
최호진	3D 모델링 프로그램 개발 및 데이터 전처리
조창현	데이터 수집장치 구현 및 길찾기 알고리즘 구현
박지호	Web 제작 및 서버,DB 관리
공통	보고서 작성 및 발표 테스트 및 성능 평가

7. 참고 문헌

[1] https://www.add.re.kr/board?menuId=MENU02742&siteId=SITE00002

[2] https://blog.panoply.io/a-comparative-vmariadb-vs-mysql

[3] https://lhbhb21c.tistory.com/88

[4] https://www.zenez.org/1596

[5] https://github.com/ZENEZ-ORG/AccelStepper