Robotics Project

Implements of Localization & Planning & Navigation

제출일	2024.12.13	전공	컴퓨터공학과
과목	로봇공학개론	학번	32224332
담당교수	최용근 교수님	이름	조남웅

1.서론

로봇 공학은 현대 기술의 중요한 연구 분야로, 다양한 환경에서의 자율성과 문제 해결 능력을 요구한다. 본 프로젝트는 로봇의 위치 추정과 경로 계획 문제를 해결하기 위한 시뮬레이션 시스템을 설계하고 구현하는 것을 목표로 한다. Unity와 Python을 활용한 이 프로젝트는, 로봇 공학의 기본 원리와 이를 구현하기 위한 실질적인 프로그래밍 기술을 응용하여, 로컬라이제이션, 플래닝, 네비게이션, 파티클 필터, A*알고리즘 및 스무딩 알고리즘 등 다양한 알고리즘을 구현하고 여러 상황을 가정하여 실험한다.

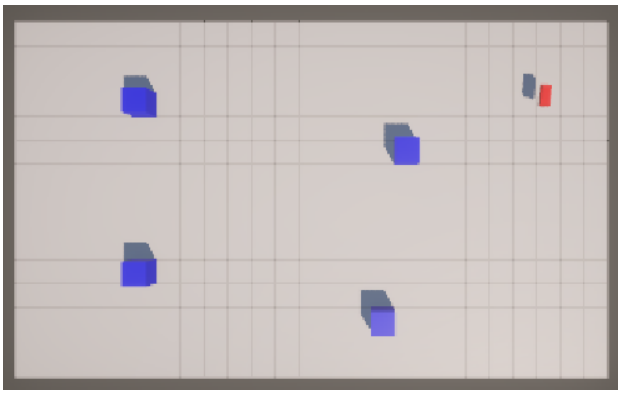
특히, 랜드마크 기반의 위치 추정 알고리즘과 최적 경로 탐색 알고리즘을 설계하고, 이를 시각화하여 구현 과정을 확인하는 것이 주요 과제이다. 이를 통해 로봇 공학의 이론적 지식을 실제 구현으로 연결하며, 실제 로봇과 최대한 유사한 형태로 프로젝트를 진행한다.

본 보고서에서는 프로젝트의 설계 과정과 구현 방법, 그리고 이를 통해 얻은 주요 결과와 학습 내용을 다루고자 한다.

2. 본론

2-1. SLAM 기본환경

SLAM을 위한 시뮬레이션 환경은 다음과 같다.

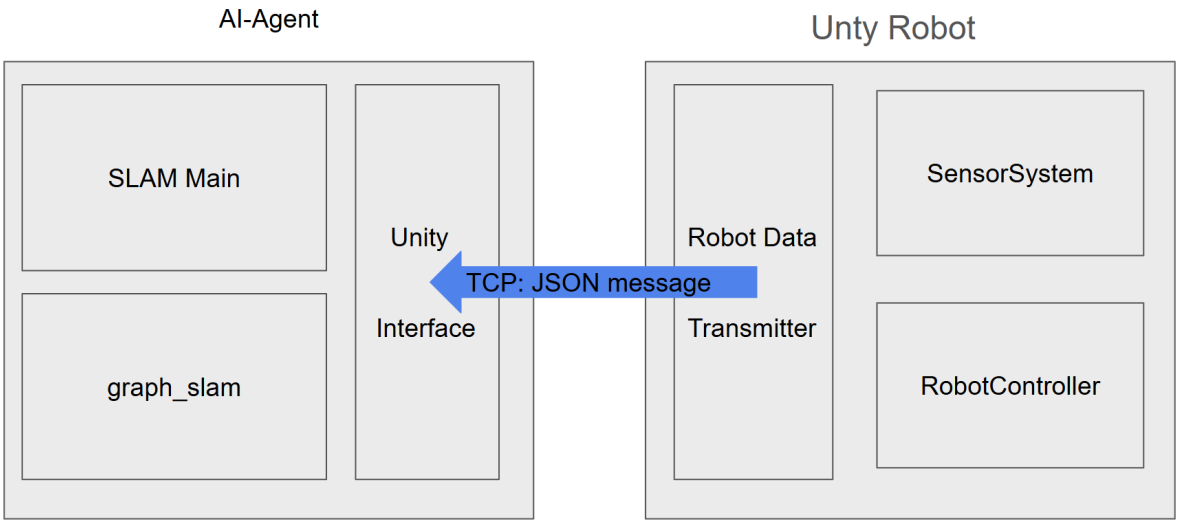


로봇의 초기Pose는 (22.5,11.5,1.5pi)에 위치해 있고 4개의 파란색 랜드마크는 장애물로 표현되며, 각각의 좌표는 (5,11) (16,9),(5,4),(15,2)에 위치한다. map의 총 크기는 25 x 15로 구성되고 하나의 셀은 1m x 1m 로 설정한다. 문제의 가정인 motion은 총 50회로 bicyle모델로 0.5m 앞으로 움직이면서 한번 움직일때마다 핸들을 2pi/50씩 움직이는 것을 실제 action모델로 선정하여 RobotController 라는 C# 스크립트로 구현했다. 또한 랜드마크와 로봇 사이의 가로거리 x, 세로거리 y를 측정하는 센싱모델을 SensorSystem 이라는 C#스크립트로 구현하여 실제 센싱에 대한 하드웨어 자체의 가우시안 노이즈와 액션에 대한 가우시안 노이즈를 설정할수 있도록 구현했다. 추가로 랜드마크의 감지거리는 제한하지 않고 센싱모델을 구현했다.

액션 모델과 센싱 모델을 통해 도출한 action data와 Sensing data는 실시간으로 RobotDataTransmitter.cs 에서 AI-Agent라고 볼 수있는 python코드로 TCP소켓 통신을 통해 전달된다. 전송 방식은 Json String을 버퍼를 통한 dyte파싱을 선택하였으며 한번의 스텝을 이동할때마다 스텝 단위의 데이터를 실시간으로 전송하도록 구현했다.

python에서는 모션 노이즈(MOTION_NOISE)와 측정 노이즈(MEASUREMENT_NOISE)는 신뢰도를 표현했다. 즉, 전송받은 데이터를 그대로 사용하는 것이아닌 일종의 노이즈(불확실성)를 추가하여 최대한 현실적인 환경을 시뮬레이션 하였다.

다음 사진은 SALM에 전체적인 구조도이다.



2-1-1. 문제 1 해결 시나리오

랜드마크 감지 거리를 제한하지 않았으므로 로봇은 모든 랜드마크를 센싱한다. 또한 문제 1에서의 맵핑은 랜드마크의 맵핑이기 때문에 특별히 맵의 경계 부분을 센싱하지 않도록 가정했다. 따라서 action모델도 한스텝당 2pi/50씩 왼쪽으로 회전하며 0.5m씩 이동하도록 설정하였으며 이는 지정된 map인 25 x 15를 벗어나도 SLAM에는 영향이 없도록 구현했다.

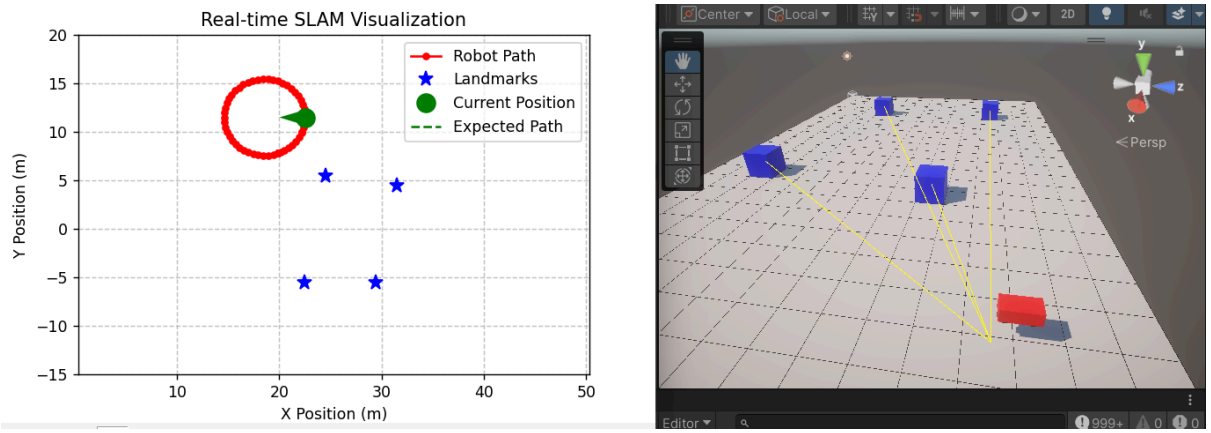
문제 1을 해결하기 위해서 구현한 Graph SLAM 알고리즘은 다음의 단계에 걸쳐 진행된다. 첫번째로, 정보행렬(오메가)과 정보벡터를 초기화 한다. 로봇의 초기 위치인 22.5와 11.5에대한 강한 제약조건(높은 신뢰도)를 설정한다. 이는 문제의 기본 가정인 초기 위치를 로봇이 인지하고 있다는 가정을 적용한것이다.

두번째로 Unity에서 받은 json데이터를 기반으로 행렬 업데이트를 수행한다. json데이터에 담겨있는 데이터는 로봇의 현재 x좌표, 현재y좌표, 현재 로봇의 방향(라디안), 랜드마크 ID, 각 랜드마크별 가로거리 x, 세로거리 y가 포함되어 있다. 이때 현재 좌표x와 현재 좌표 y는 이전 스텝의 로봇 위치와 비교하여 이동거리를 계산하며,SLAM의 Motion Model에 사용된다. 랜드마크 데이터는 로봇이 센싱한 각 랜드마크별 데이터가 있고 이는 SLAM의 Measurement Model에 반영된다.

세번째로 이렇게 얻은 데이터들을 기반으로 Motion Noise(이동 노이즈),Measurement Noise(측정 노이즈)로 액션데이터와 센싱 데이터에 대한 신뢰도를 설정하고 업데이트된 오메가와 상태를 이용해 로봇의 위치와 랜드마크의 위치를 계산한다. 이때 깨끗하지 않은 데이터 처리 즉, 랜드마크에 대한 정보가 없거나 현재 Pose에 대한 정보가 없다면 정보행렬(오메가)와 정보행렬(ξ)에 대한 해(상태: 로봇 위치와 랜드마크 위치의 최적해)를 구하기 어려운 경우가 생길수 있기 때문에`np.linalg.pinv(self.omega) @ self.xi`를(의사역행렬)사용하여 선형 방정식으로 풀리지 않은 식에 대한 예외처리를 하였다.

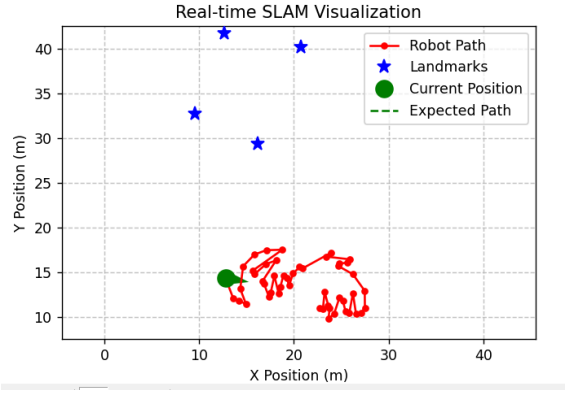
2-1-2 다양한 경우 테스트

1. 모든 노이즈가 0인 경우



위 사진에서 오른쪽은 Unity로 구현한 시뮬레이션환경에서 실제 로봇이 센싱하는 사진이고 왼쪽은 python에서 SLAM 알고리즘을 적용하여 시각화한 사진이다. 위 사진의 경우는 하드웨어 노이즈(현실적 환경에서 발생하는 노이즈) 즉 Unity코드에서의 액션모델, 센싱모델의 노이즈를 0으로 하고 python에서 Motion Noise=0.0, Measurement Noise = 0.0으로 설정하여 테스트 하였다.

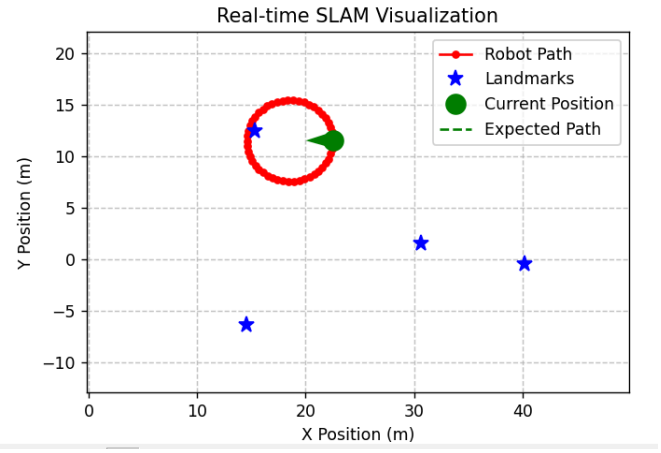
2. 액션 & 센싱 노이즈(하드웨어 노이즈)만 있는 경우



위 사진은 액션 노이즈 시그마를 0.1 그리고 센싱 노이즈 시그마를 0.4로 설정하고 SLAM의 Motion 노이스와 Measurement 노이즈를 모두 0으로 설정하고 SLAM 진행이 완료된 상태를 시각화한 것이다. 액션과 센싱 노이즈의 시그마는 가우시안 분포의 파라미터로 사용하였다. 가우시안 노이스 메서드는 파라미터인 시그마 값을 입력받으면 0~1사이의 균일 분포 난수 두개 u1,u2를 생성한뒤 Box-Muller 변환 방식을 사용하여 표준 정규 분포 N(0,1) 즉 평균이

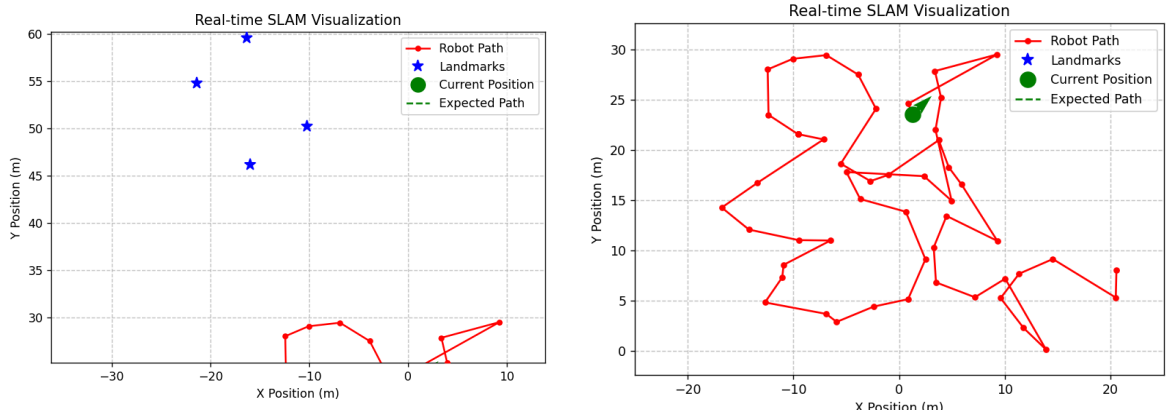
0이고 분산이 1인 분포를 생성한다. 이후 만들어진 표준정규분포 N의 분산산과 입력받은 시그마값을 곱하여 (x=μ+σ·z) 원하는 분산과 표준편차를 갖는 N(μ,σ^2)를 도출한다. 이를 기반으로 사진의 로봇의 움직임은 확실히 노이즈로 인해 입력된 각도로 움직이지 않고 40%의 오차를 지니며 움직이고, 랜드마크 역시 10%정도의 차이를 보이며 최종적으로 노이즈가 0인상태에서 약 41%정도 차이가 난다는것을 알수 있다.

3. 센싱 노이즈 시그마 ,Motion 노이즈 값이 크고 나머지 노이즈는 모두 0 일경우



다음으로, 액션 노이즈 시그마가 0,센싱 노이즈 시그마가 10이고 Motion 노이즈 값이 매우 큰값, Measurement 노이즈는 0일때의 상황이다. 실제 액션모델의 노이즈 시그마가 0이여서 완벽한 액션 데이터를 수집하지만 SLAM적용시 Motion 노이즈를 매우 큰 수로 설정했기 때문에 센싱 데이터에 대한 의존성이 증가하여 위와 같은 시각화 결과가 나온다는것을 알수 있다. Measurement 노이즈의 신뢰도가 높지만, 랜드마크에 대한 맵핑이 전혀 안되고 있다는것을 확인할 수 있다. 또한 여기서 만약 자신의 예상위치까지 도출한다면 로봇의위치가 다르게 나오기 때문에 일반적인 SLAM알고리즘이 적용이 안될 수 있다.

4. 액션 노이즈 시그마, Measurement 노이즈가 크고 나머지 노이즈는 모두 0일 경우



마지막으로, 액션 노이즈 시그마가 0.3 센싱 노이즈 시그마가 0 Motion 노이즈 값은 0, Measurement 노이즈가 매우큰값 일때의 상황이다. 이번에는 완벽한 센싱값이 입력되었지만, SLAM적용시 Measurement 노이즈를 매우 큰 수로 설정했기 때문에 다른 데이터 즉, 액션 데이터에 대한 의존성이 증가하여 위과 같은 시각화 결과가 나온다는것을 알 수있다.

2-2 Localization 기본환경 (Particle Filter)

마찬가지로 Unity 시뮬레이션 환경 및 실시간 TCP 방식을 사용했으며, 문제 2번의 Localization은 파란색 랜드마크만을 이용하여 로봇의 자신의 위치를 특정하기 때문에 map은 문제 1과 같다. 하지만 하드웨어 action모델의 스펙은 전후 좌우 한칸(1m) 이동으로 고정되어 있기 때문에 문제 1과 다른 action 모델을 구현하여 사용했다. action데이터를 얻는 방법으로는 두가지 경우의 선택을 할 수 있었는데, 첫번째는 수동조작으로 action 데이터를 생성하여 Localization하는 방법과 특정한 경로 혹은 랜덤한 조작 데이터를 생성하여 Localization하는 방법이 있었다. 여기서 선택한 데이터 획득 방식은 특정한 경로를 지정하여 action하도록 하여 action데이터를 수집하는 방식을 선택했다. 그 이유는 센싱에 대한 범위가 중요한 가정이기 때문에 최대한 많은 랜드마크를 센싱 할 수있는 움직임을 action데이터로 선정하면 가장 이상적인 Localization이 될 수 있다고 생각하여 두번째 방식인 특정 경로 action데이터를 수집했다.

센싱 모델도 5m까지의 감지 범위가 있기때문에 새로운 센싱 모델을 구현하여 사용했다. 이때의 센싱모델은 거리제한이 있다는것을 염두하여 거리에 따른 노이즈 변화 로직도 추가하여 구현하였다. 문제 2의 기본 노이즈인 0.4부터 1m씩 멀어질 때마다 0.1의 노이즈 증가 계수를 추가하여 최대 노이즈 1.0까지 구현하였다. 이는 현실적으로 거리가 멀어질 때마다 센싱 모델의 노이즈가 증가하는 것을 표현하였다.

Localization의 알고리즘으로는 Particle Filter를 선정하였고 Particle Depletion 현상을 방지하기 위해 충분한 파티클 개수인 1000개의 파티클로 진행하였다. 노이즈 모델 역시 문제 1과 마찬가지로 가우시안 분포를 사용하여 가중치를 계산하였다. 이 때의 json 데이터의 내용은 랜드마크 까지의 직선거리와 랜드마크 ID 가 주요한 데이터 내용이다. 리샘플링 조건으로는 유효 입자 수(n_eff)가 전체 입자 수의 절반 미만일 경우 리샘플링을 수행한다. 유효 입자 수는 가중치 분포의 집중도를 나타내며, 특정 입자에 가중치가 과도하게 집중될 경우 리샘플링이 트리거된다.

유효 입자는 가중치 분포의 균등성을 나타내는 지표로 전체 다음과 같이 계산된다

$$n_{\text{eff}} = \frac{1}{\sum w_i^2}$$

w_i 는 각입자의 가중치를 나타낸다. 이는 특정 가중치가 특정 입자에 집중될 경우 입자의 다양성이 부족해질 수 있으므로 이를, 방지하기 위한 매커니즘이다. 따라서 입자가중키가 특정입자에 집중되면서 유효입자수가 절반 이하로 감소하면 리샘플링을 통해 입자 분포를 재조정 한다.

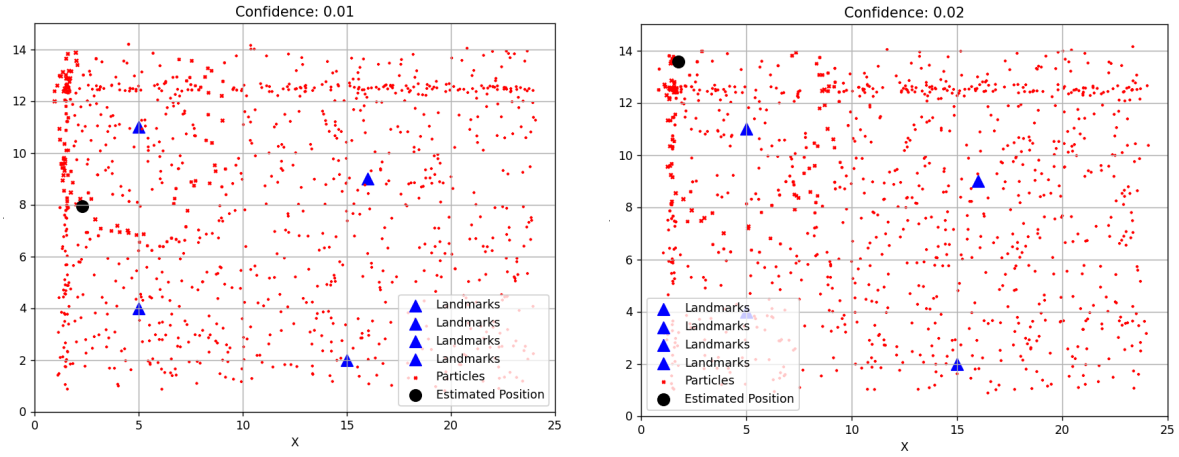
2-2-1 문제 2 해결 시나리오

먼저 파티클 초기화 하는 _intiialize_particles() 를 사용하여 25 x 15 맵안의 파티클들을 초기 가중치를 1/N으로 설정하여 랜덤하게 초기화 한다. 다음으로 predct 예측단계에서 로봇의 이동명령을 파티클에 반영하고, 이동과정에서의 노이즈를 추가한다. 이때 이동명령(dx,dy) max_movement라는 파티클 최대 이동거리를 초과하지 않도록 제한하여 단계별 시각화가 느리게 흘러가도록 조절했다. 또한 이동거리에 Motion 노이즈를 추가하여 실제 액션데이터에 대한 신뢰도를 반영하였으며 위의 SLAM과는 다르게 Localization의 기본적으로 제공된 map정보를 로봇은 이미 알고 있기 때문에 맵바깥으로 나가는 파티클은 맵경계에 고정하여 시각화에 더 직관적이게 표현했다.

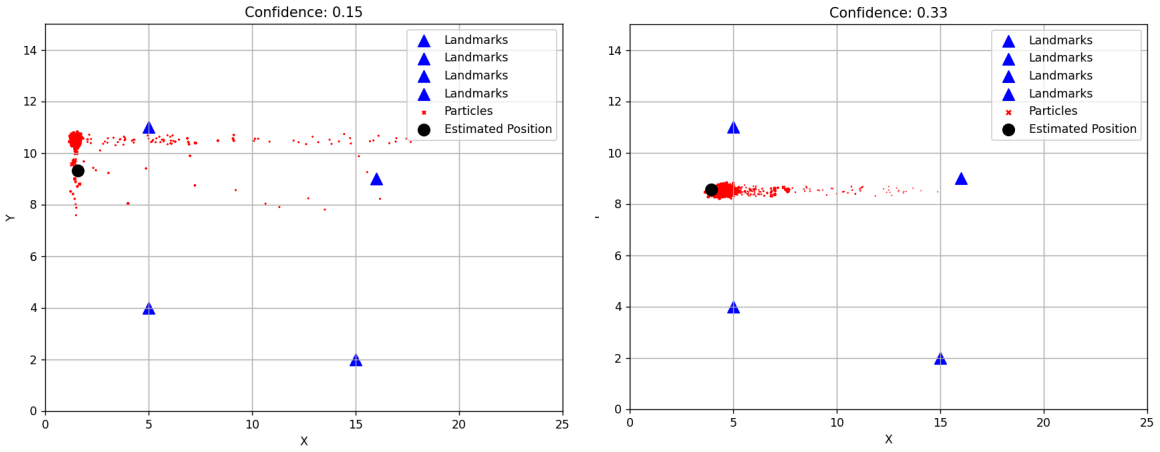
다음으로 update() 메서드를 사용하여 센서 데이터를 기반으로 파티클 가중치를 업데이트 하여 로봇의 위치를 더 정확히 추정했다. Json 데이터 입력으로 들어온 랜드마크 ID와 측정된 거리정보를 확인하고, 각파티클에서 랜드마크까지의 거리를 계산한후 측정값과 예측값 간의 차이를 기반으로 가중치를 업데이트 하였다. 그 다음 가중치 정규화를 하여 모든 파티클의 가중치 합이 1로 되게하여 scale을 맞췄으며, 리샘플링 판단을 위한 유효파티클 개수를 새는 로직도 추가하였다 유효파티클 개수가 전체 파티클의 반이 되었을 때 리샘플링을 수행하도록 하였다. 또한 0.5 + 0.5* likehood 방식으로 가중치를 업데이트하여 모든 파티클이 일정 수준 이상의 기여도를 유지하도록 했으며 훨씬 부드럽게 시각화되는것을 알수 있었다.

2-2-2 Localization Test

1. 랜드마크 갯수 4개 최대감지 범위 5m

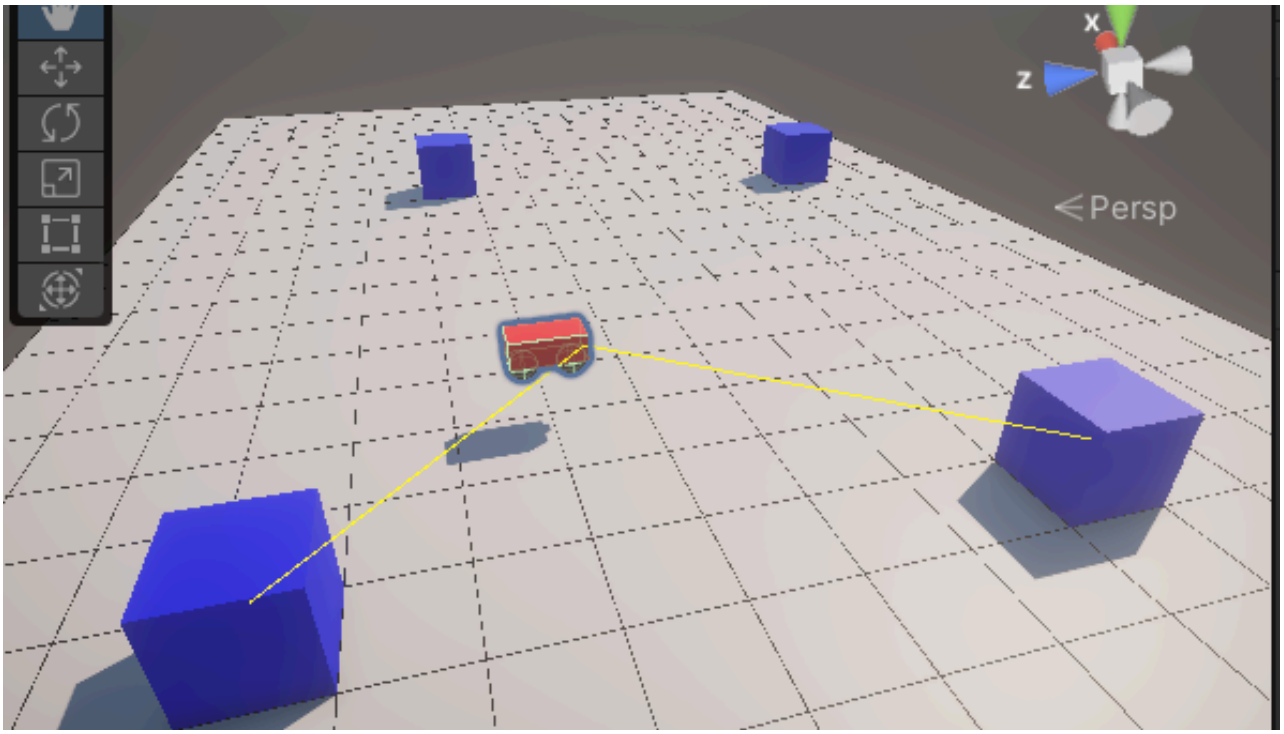


먼저 처음 테스트를 시작하자마자 랜덤한 분포로 map에 파티클들이 분포된다. 이때의 confidence는 매우 적은 수치이고 로봇 자신의 위치역시 어디있는지 잘 모르는 상태이다.

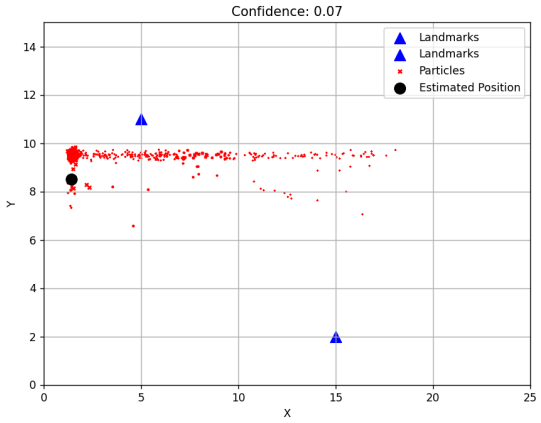


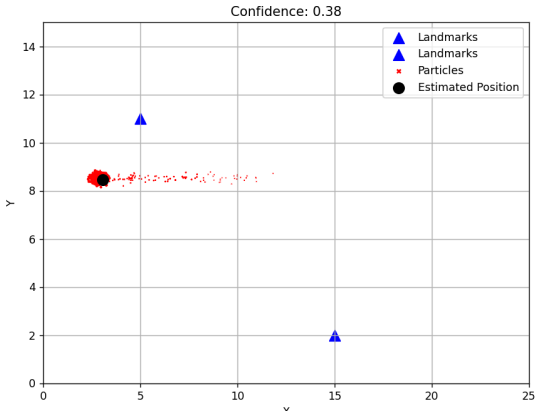
다음으로 로봇의 점점 로봇이 이동함에 따라 파티클들의 분포는 로봇의 위치와 비슷해지게 리샘플링 되며, 최종적으로 로봇의 위치가 어느정도 확정이 된 순간의 confidence의 값을 0.8기준으로 세워 자신의 위치에 대한 신뢰성이 0.8이상이 되면 particle 시각화를 마무리한다.

다음 사진은 지정된 경로를 한칸씩 이동함에 따라 로봇이 센싱하는 랜드마크가 생긴다는 것을 알 수 있는 사진이다



2. 랜드마크 개수 2개





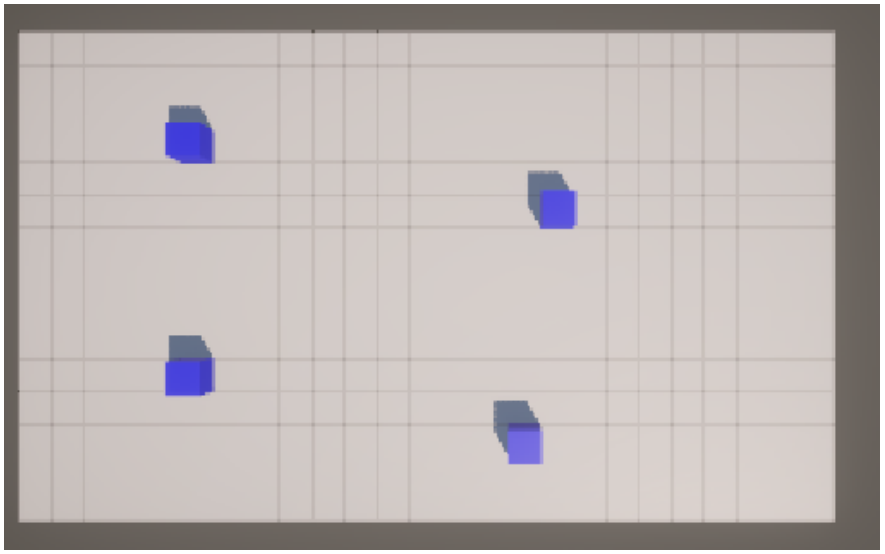
다음으로 위사진은 랜드마크가 2개일때의 사진이다. 사진의 particle 분포를 보면 별 차이가 없어 보일 수있지만 confidence값을 보면 확연한 차이가 있다. 자신의 위치에 대한 신뢰도즉 추정 정확도와 수렴 속도에서 차이가 있는 것을 확인 할수 있다. 각 파티클의 가중치 계산에 도움을 주는 랜드마크의 개수가 줄어들어 으므로 파티클의 상태(위치와 방향)가 좋은 상태인지 평가하는데 오래걸리고, 그에 따라 파티클의 수렴 속도가 줄어든다.1번의 경우엔 시각화 총 시간이 약 3초이지만 2번의 랜드마크가 2개인 경우에는 총걸린 시간이 약 5.53초로 수렴하는데 더 오랜시간이 걸린다는 것을 알 수 있다. 랜드마크의 개수가 줄어들면 각 파티클이 계산해야할 랜드마크와의 거리 또는 각도 데이터가 줄어들지만, 근복적을 랜드마크가 제공하는 정보가 부족하기 때문에 파티클의 추정 정확도가 떨어져 수렴속도가 느려진다는것을 테스트를 통해 확인하였다.

3. 4개 랜드마크 개수 & 센서의 최대거리 증가

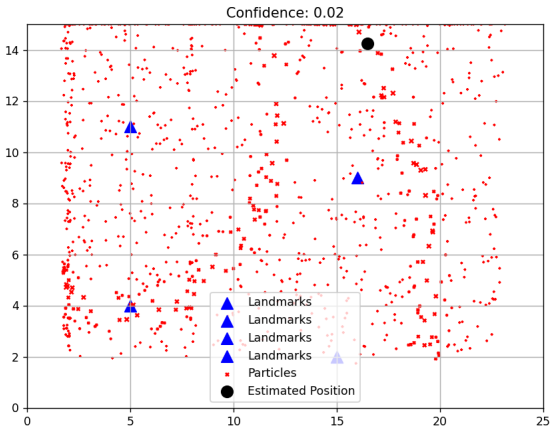
3-1. 하드웨어 노이즈 시그마 없을경우

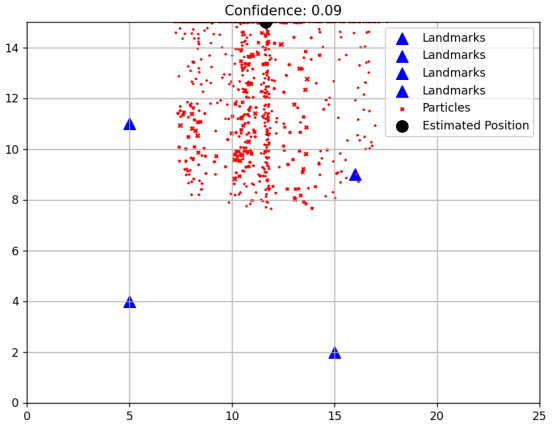
랜드마크의 개수를 4개로 하고 센서의 최대거리를 40으로 하고 이번엔 하드웨어 노이즈 시그마 센서와 액션 시그마를 모두 0으로 설정하였을 때 많은 양의 랜드마크 데이터 처리를 위해 약간 느려지지만 마지막엔 매우 높은 신뢰도 즉 매우높은 정확성을 갖는것을 알 수 있다. 따라서 관측 데이터가 증가하여 위치 추정 정확도가 향상된다는 것을 알 수 있다. 이는 기존의 시각화 자료와 크게 다르지 않는것을 알수 있으며, 2개의 센싱데이터를 추가로 처리하기때문에 1,2번 경우와 시간의 차이가 난다는것을 알수 있다.

3-2. 하드웨어 노이즈 시그마 있을경우



맵에서조차 로봇이 보이지 않음



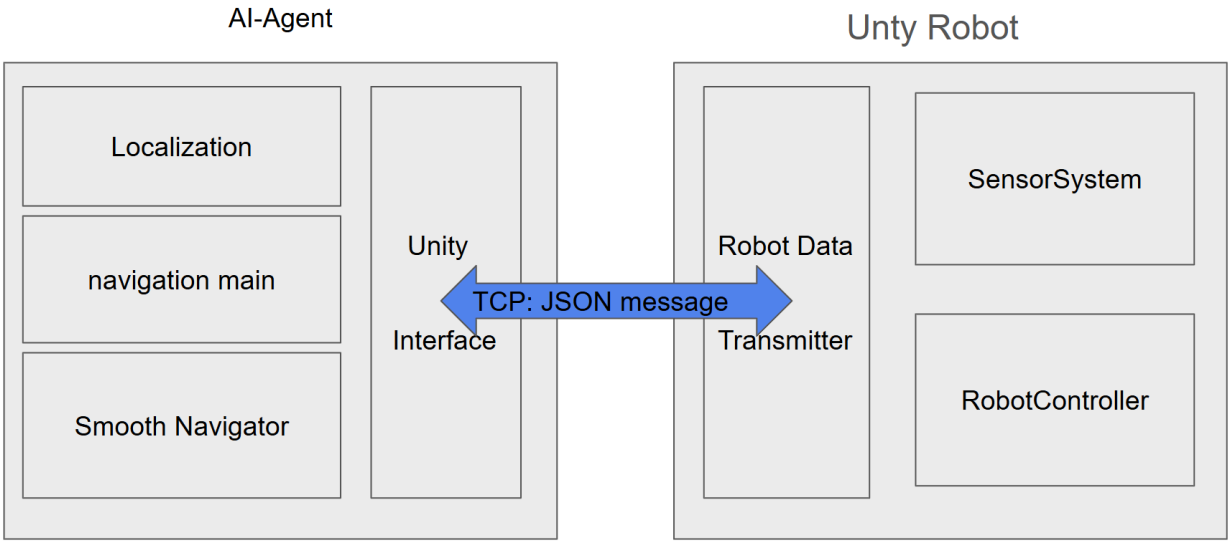


랜드마크의 개수 4개 센싱 범위 40 액션 노이즈 시그마 10 센싱 노이즈 시그마 40 으로 설정했을때의 경우이다 이 경우 센싱 데이터와 액션데이터 모두 실제 로봇의 위치와는 매우 다르기 때문에 실제 로봇의 위치와 매우다르고 센싱 범위는 넓기 때문에 4개의 랜드마크에 관한 데이터 처리 까지 합하여 매우 안좋은 성능을 보여준다 정확도와 시간 모두 오래걸리며, 그렇게 해서 얻은 신뢰도도 매우 낮다는 것을 확인할 수 있다.

3.Planning & Navigation 기반 환경 (PID, Smoothing)

문제3 Planning에 대한 기본환경은 랜드마크 4개만 있는 환경을 구성하였다 1,2 번문제의 map을 그대로 유지하였으며 Localization 알고리즘도 동일하게 문제 2번에서 사용한 particle filter를 유지했다. 하지만 기존의 Particle Filter와 다른점은 로봇의 특정한 이동없이 Localization을 진행한 것이다. 랜드마크의 위치와 개수가 동일 하기 때문에 사실상 랜드마크 1개 를 사용하여 Localization을 진행하였다. 이때 결과값을 얻어내기위해 각 파티클 가중치를 크게 올렸으며 하나의 랜드마크만을 센싱하기 때문에 Localization을 할때의 시간이 너무 오래걸려 결국 0.01초마다 resampling하여 매우 빠른 속도로 신뢰도 0.8에 도달하도록 하였다. 문제 3번의 가장 중요한 핵심은 Planning과 Navigation이기 때문에 경로를 탐색하고 목적지로 향하는 것에 중점을 두었다. 나아가 Planning & Navigation을 하는 도중에도 로봇은 0.1초 주기로 Localization을 진행하여 신뢰도 0.8의 수치를 유지하도록 구현했다. 이를 위해 기존 SLAM에서의 통신방법인 하나의 step을 완료하면 이벤트 트리거를 작동하여 python에게 센싱과 액션 데이터를 주는 방식이 아닌 1초에 한번씩 주기적으로 액션과 센싱데이터를 보내도록 변경하였다.

3-1. 문제 3 해결 시나리오



위 사진은 Planning과 Navigation의 전체적인 구조를 표현한 구조도이다 동작의 순서는 다음의 단계에 따른다.

첫번째로 로봇은 움직이지 않고 랜드마크 하나만을 이용해 Localization을 수행한다. 신뢰도가 0.8이 되지 않으면 경로생성 클래스가 수행되지 않도록하여 확실하게 자신의 위치를 알고있다는 상태를 기반으로 구현했다. 파티클의 개수는 1000개를 사용했고 일단 motion noise 0.1 , measurement noise 0.4 로 설정한다.

두번째 단계로, Loclaization을 통해 신뢰도가 0.8이상이 되면 그즉시 목적지 까지의 최적경로를 생성한다. 경로 생성 알고리즘은 A*알고리즘을 사용하였으며 각 격자 좌표의 휴리스틱 값은 목적지로부터의 유클리드 거리를 사용했다. 또한 로봇의 이동은 각 격자 사이의 중심으로 이동한다는것을 가정하여 x와 y에 0.5를 더하여 격자 중심점으로 변환한뒤 각 격자의 중심을 통해 목적지까지 도달하도록 하였다.

세번째로 최적 경로가 생성되면 Smoothing 알고리즘을 수행하여 스무딩된 경로를 도출한다. Smoothing경로로 사용한 알고리즘은 예제의 Smoothing방정식을 사용하였으며 원래경로를 numpy배열로 변환하여 smooth_tolerance이하의 오차가 발생할 때까지 반복을 수행했다. x를 원래 경로라 하고 y를 스무딩된 경로라 한다면 x-y와 스무딩경로 상의 이웃간의 점의 차이를 Gradient Descent의 원리를 적용하여 스무딩 경로를 도출했다. 스무딩 알고리즘에는 weight_data, weight_smooth, smooth_tolerance 라는 세가지 파라미터를 설정하였다. weight_data는 원래 경로를 유지하려는 힘(데이터 보존강도)을 제어하는 가중치, weight_smooth는 스무딩 경로의 연속성을 유지하려는 힘(부드러움 강도)을 제어하는 가중치, smooth_tolerance 는 알고리즘의 반복 종료 조건을 정의하는 허용 오차이다. 이 세가지 값을 이용하여 다양한 경우의 smoothing 경로를 도출했다.

네번째로 도출한 스무딩 경로를 이동하기위한 PID제어를 진행한다 PID제어에 필요한 Kp(비례 게인) 값은 0.3, Ki(적분 게인)값은 0.05, Kd(미분 게인) 값은 0.4로 설정하고 `_compute_lookahead_point` 메서드를 사용하여 항상 앞의 경로의 목표점을 설정하도록 하였고 Ki(적분 게인)의 값은 Anti-windup을 사용하여 생성된 적분값을 -10에서 10사이로 제한하여 시스템 안정성과 적분항이 무한히 커지는 것을 방지했다. 이렇게 생성한 PID 제어를 dx,dy형태인 벡터로 변환하여 Unity로봇에게 TCP통신을 통해 Json 메세지를 보낸다.

다섯번째로 dx,dy의 값이 로봇에게 도착하면 로봇은 선정된 벡터 명령을 수행한다. 이동 중 다음 PID 값의 도착으로 인한 오버헤드를 줄이기 위해 3초에 한번씩 PID계산과 통신주기를 설정하여 로봇이 충분히 움직이고 난뒤 다음 PID를 수행하도록 하였다.

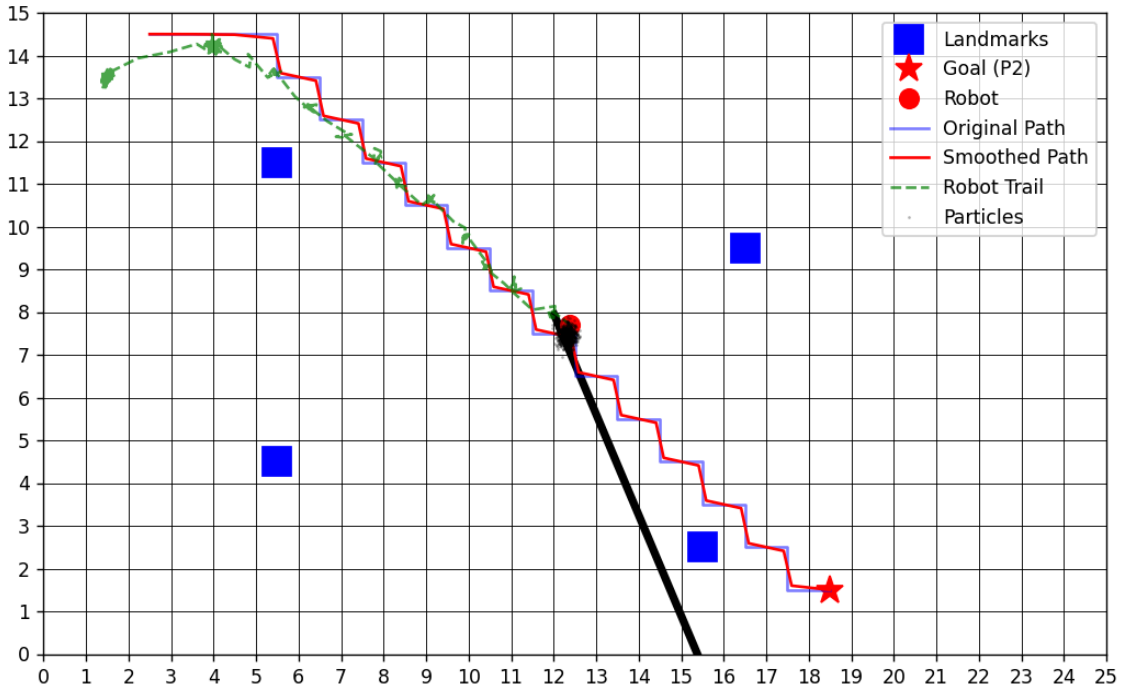
3-2. Planning & Navigation Test

3-2-1. Planning : smootng path 파라미터에 따른 경로추정 케이스

weight_data,weight_smooth 의 값들을 변화시켜 여러 테스트를 진행한뒤 스무딩 알고리즘에 대해 고찰하고 각 가중치들은 어떤 특징이 있는지 설명한다.

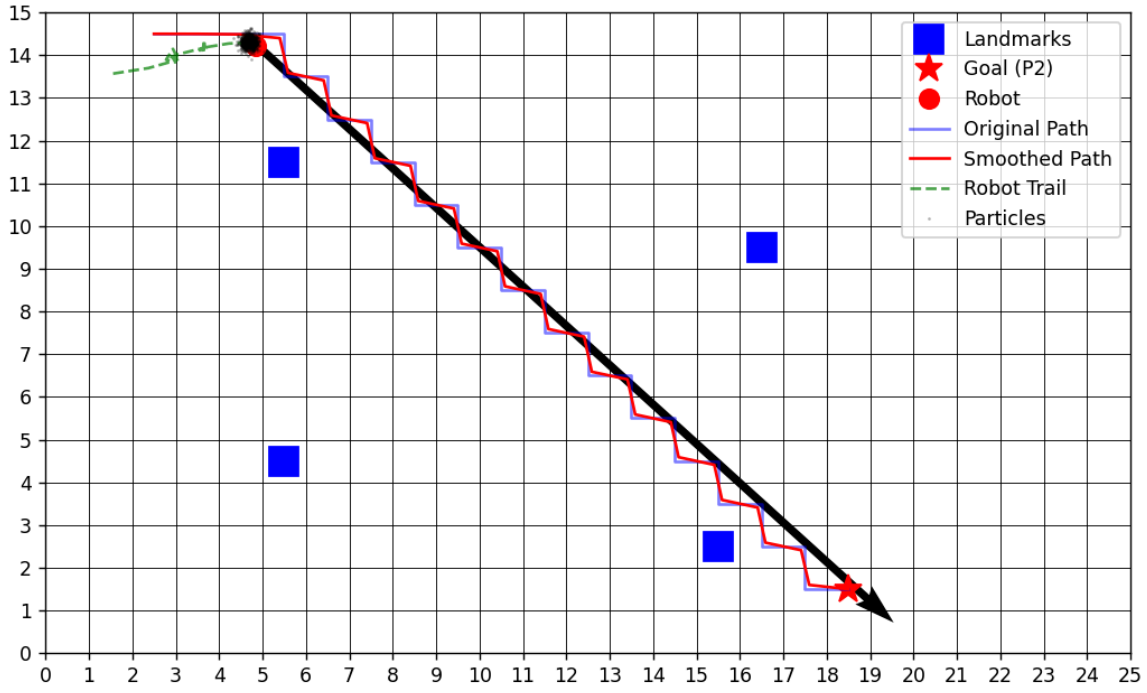
1. weight_data : 원래 경로를 유지하려는 힘(데이터 보존강도)을 제어하는 가중치
2. weight_smooth : 스무딩 경로의 연속성을 유지하려는 힘(부드러움 강도)을 제어하는 가중치
3. smooth_tolerance : 알고리즘의 반복 종료 조건을 정의하는 허용 오차

1.weight_data=0.9, weght_smooth = 0.1 , sooth_tolerance = 0.00001



일단 Localization의 신뢰도가 0.8에 도달할때까지 기다린다. 오른쪽의 사진은 0.8이 도달된 즉시 신뢰도 0.8을 유지하며 smooting 경로를 도출한다. weight_data를 0.9로 선정했기 때문에 A*생성한 원래 경로와 큰차이가 없다는것을 알 수있다. 검정색은 그지점에서의 로봇의 벡터 값을 시각화 한것이다.

2. weight_data=0.1, weght_smooth = 0.9 , sooth_tolerance = 0.00001

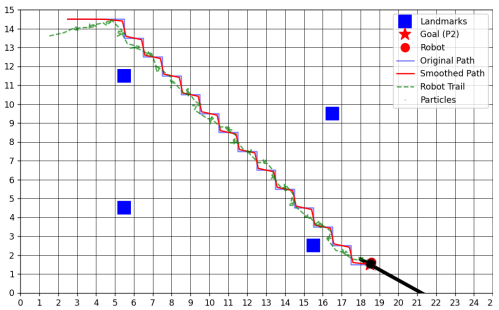


경로가 크게 달라지지 않는다는 것을 알 수 있다. 이유는 sooth_tolerance = 0.00001 가 동일하기 때문에 동일한 오차범위로 수렴할때 까지 충분한 반복을 진행하여 두 경우의 smoothing 경로 차이는 없다고 볼수 있다. 이는 충분한 반복을 진행하여 오차 허용 범위에 진입하기 때문에 1번 경우에는 천천히 원래경로를 따라 안정적으로 변화하여 스무딩 경로를 도출하고 2번 경우에는 빠르고 급격하게 변화하여 스무딩 경로를 도출한다는 것을 실험을 통해 증명할수 있다.

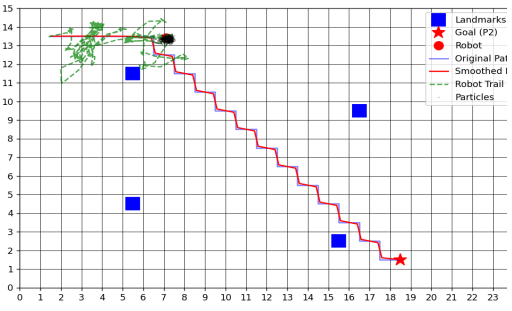
경로간의 차이를 두기 위해서는 반복횟수를 제한하거나 sooth_tolerance범위를 더 작게 설정하고 스무딩경로가 도출되는 중간단계를 시각화 하는것이 1,2,번 경우 차이를 볼수 있다. 이는 어떤식으로 스무딩 경로가 도출되는지 즉, 중간단계는 다르지만 최종적인 스무딩경로는 같다는 것을 알수 있다. 선택한 스무딩 알고리즘은 [변화량 = weight_data * (원래점 - 현재점) + weight_smooth * (이웃점들의 평균 - 현재점)] 즉 선형 시스템을 기반으로 스무딩 경로를 도출하기 때문에 선형 시스템에서는 가중치들이 단지 "얼마나 빨리 해에 도달하는가"만 결정할 뿐, 최종 해 자체는 시스템의 구조에 의해 결정된다를 알수 있다. 따라서 3개의 파라미터의 변화로 알수 있는것은 충분한 반복이 일어날 경우 비선형 항이나 최대 변화량을 제한하거나 곡률에 따른 적응형 가중치를 추가 하지 않는이상 최종적인 결론이 같다는 것을 도출 할 수 있다. 즉,weight_data와 weight_smooth의 값이 어떻게 설정되든 결국 같은 결과가 나오게 된다는 것을 알수 있다.

3-2-2. Navigation: PID 제어 gain 파라미터 변화에 따른 로봇의 이동기록 테스트

1. Kp:0.3 Ki:0.05 Kd:0.4



2. Kp:1.5 Ki:0.2 Kd:0.1



1번일 경우에서는 비례 게인(Kp)이 낮아 목표 값에 도달하기 위한 제어 신호의 크기가 상대적으로 작고, 적분 게인(Ki)이 낮아 오차 누적 보정도 느리게 이루어진다. 대신, 미분 게인(Kd)이 상대적으로 높아 목표에 접근할 때 오차 변화율을 효과적으로 억제하여 안정적인 상태로 수렴할 수 있다. 이러한 균형 잡힌 게인 설정은 시스템이 과도한 제어 신호로 인해 불필요한 진동을 피할 수 있도록 돕는다.

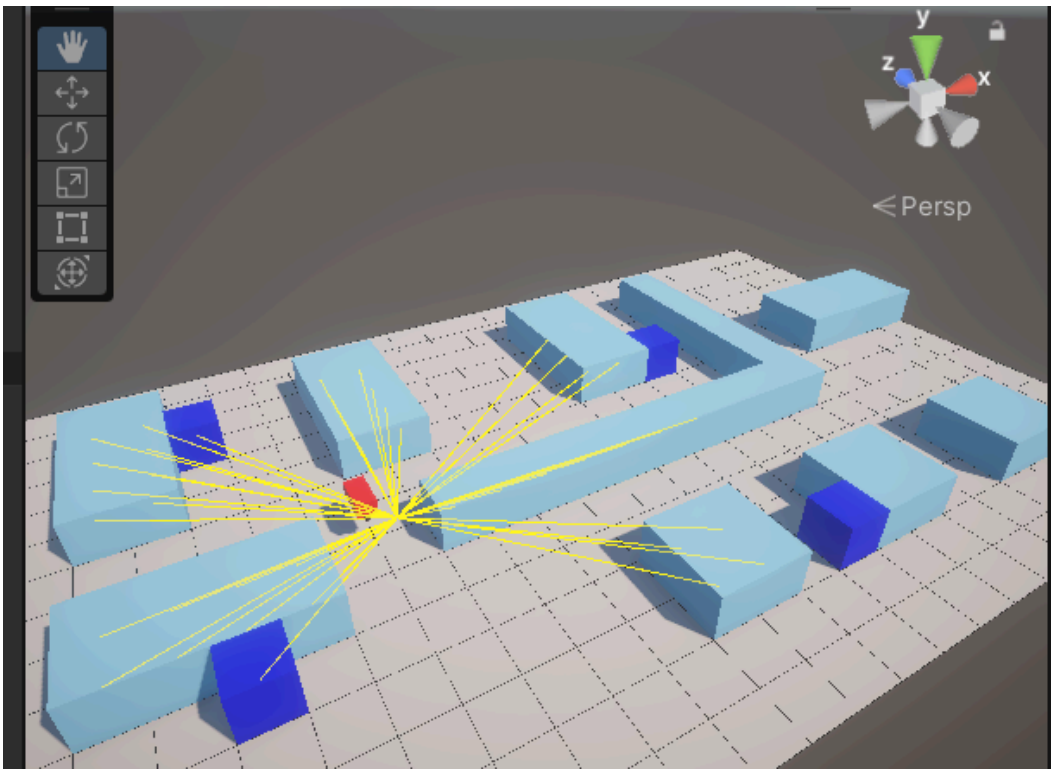
2번일 경우에서는 비례 게인(Kp)이 높아 오차에 대한 제어 신호가 매우 강하게 반응한다. 적분 게인(Ki) 역시 높아 오차 누적 보정이 빠르게 이루어지지만, 미분 게인(Kd)이 낮아 오차 변화율을 억제하는 효과가 약화된다. 이로 인해 목표 값 근처에서 과도한 제어 신호가 발생하고, 이를 상쇄하려는 제어 신호가 반복적으로 발생하면서 진동이 유발된다. 요약하자면, 진동이 발생하는 주요 원인은 제어 시스템이 너무 "과민하게" 반응하기 때문이라는 것을 알수 있다. 따라서 Kp(비례)는 현재 오차에 대한 즉각적인 반응의 강도, Ki(적분)는 과거의 누적된 오차를 처리하는 정도, Kd(미분)는 급격한 변화를 억제하고 진동을 감소시키는 정도를 결정한다는 사실을 도출할 수 있다.

게인값 이외에도 진동을 유발하는 요소는 많다. PID 제어주기가 너무 짧거나 너무 가까운 목표점을 선택하여 진동하는 경우, 또는 적분항의 계산시 누적되어 많은 오차를 일으킬수 있다. 따라서 적절한 조절을 통해 PID값을 제어하여 dx,dy값을 도출하고 로봇의 이동을 검사해야한다.

4. 문제 4,5 를 위한 기본환경

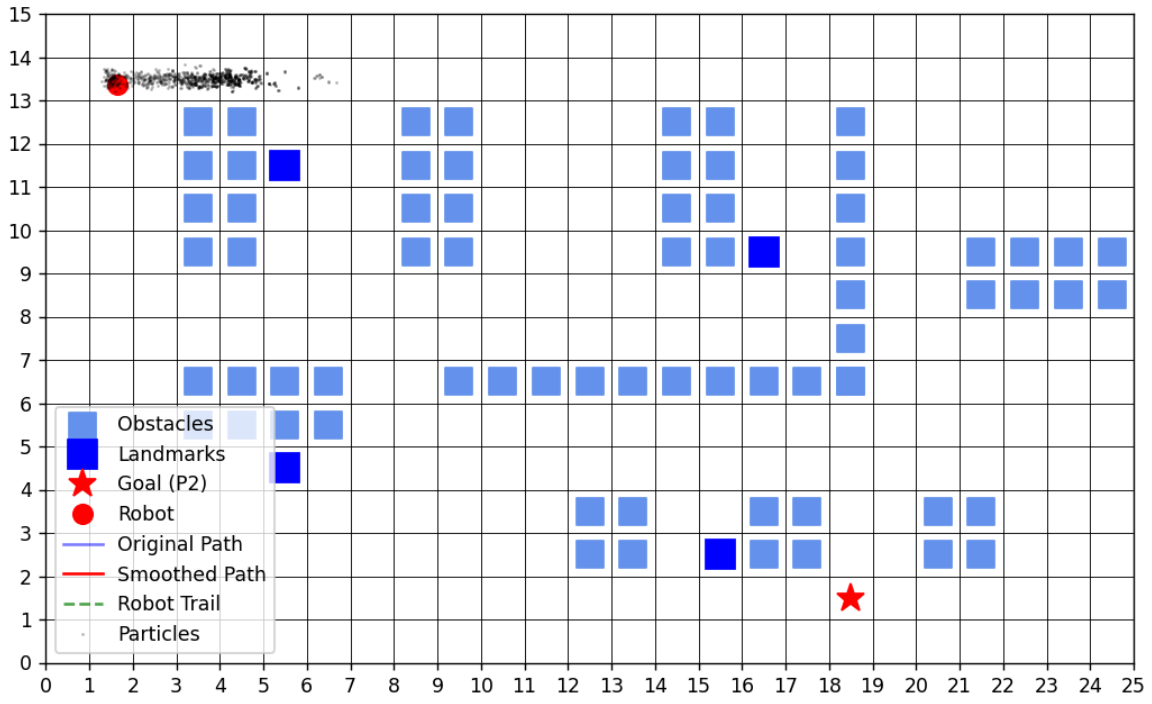
문제 2번의 센싱모델을 감지거리를 8로 확장하고, 그리고 감지요소에 장애물까지 추가하여 진행했다. 10도 단위로 360도 전체방향으로 거리를 측정하여 랜드마크 뿐만 아니라 감지되는 요소에 장애물도 추가하여 전체적인 Localization 과 Planning & Navigation을 진행하였다. 스무딩 알고리즘도 기존의 선형적 시스템에 적용하는것이 아닌 장애물에 따른 곡률의 변화를 비선형형 항으로 추가하여 더욱 개선하였다. 로봇의 액션모델은 위의 문제들과 동일한 전후 좌우 1m 씩 움직이는 모델을 그대로 사용했다. 아래 사진은 실제 unity로봇이 주변 사물들의 정보를 수집하는 사진이다.

실제 Unity 로봇이 센싱하는 사진



4-1 Localization의 변화 (문제 4)

기존의 감지범위 5에서 8로 증가함과 자신의 위치에 근거가 되는 장애물요소가 추가투입됨에 따라 기존의 Localization에 걸리는 시간보다 긴 시간이 소요된다는 것을 알 수 있었다. 이는 더 많은 참조점으로 인한 정확한 위치를 추정할수 있다는 장점을 가졌지만, 실질 적인 센싱 데이터량의 처리량 증가와 연산량을 증가하여 더 많은 계산이 필요하다는 것을 알수 있었다. 아래의 사진은 감지 범위8, 장애물 추가 한 이후 Localization하는 사진이다.

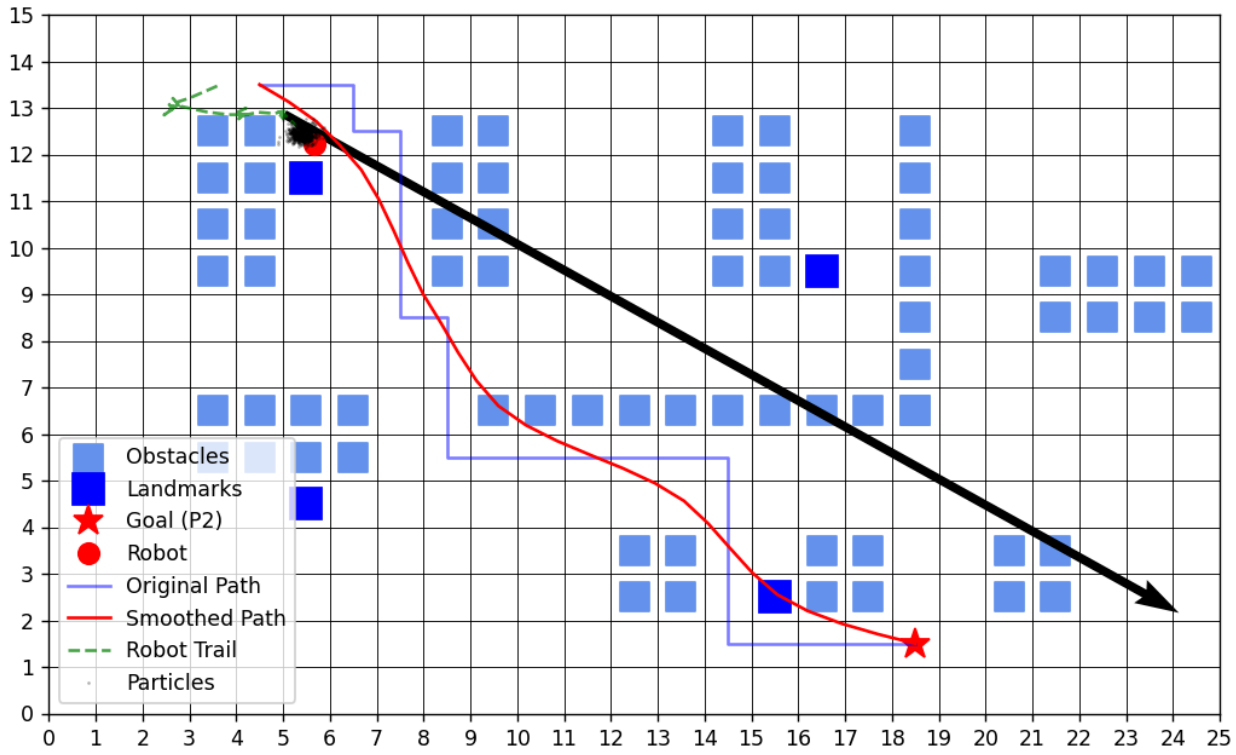


4-2 Planning & Navigation 의 변화(문제 5)

앞선 문제들에서 사용한 경로 추적 알고리즘인 A*알고리즘에 장애물 판단 로직을 추가하였다. 해당 좌표를 피해서 최적 경로를 계산하여 장애물과의 충돌상황을 미리 인지하여 로봇의 움직임에 적용하였다. 다음으로 스무딩 알고리즘의 개선사항이다 장애물이 추가함에 따라 스무딩경로를 더욱 부드럽게 생성하기 위해 더욱 개선된 형태의 스무딩 알고리즘을 사용하였다. 기존의 스무딩 알고리즘은 원래 경로점으로 당기는 힘(weight_data)와 이웃한 점들의 평균으로 당기는힘(weight_smooth)만들 고려하여 스무딩 경로를 생성했지만, 개선된 스무딩 알고리즘은 max_curvature(곡률 제한 변수)라는 변수를 추가하여 곡률로 인한 으로 경로가 너무 급격하게 꺾이는 것을 방지하였고 , 주변 장애물을 피하는 장애물로부터의 반발력을 계산하고, 장애물과의 거리에 반비례하는 힘을 추가하여 안전한 거리 유지가 가능하게 하였다.

4-2-1 개선된 Smoothing경로 test

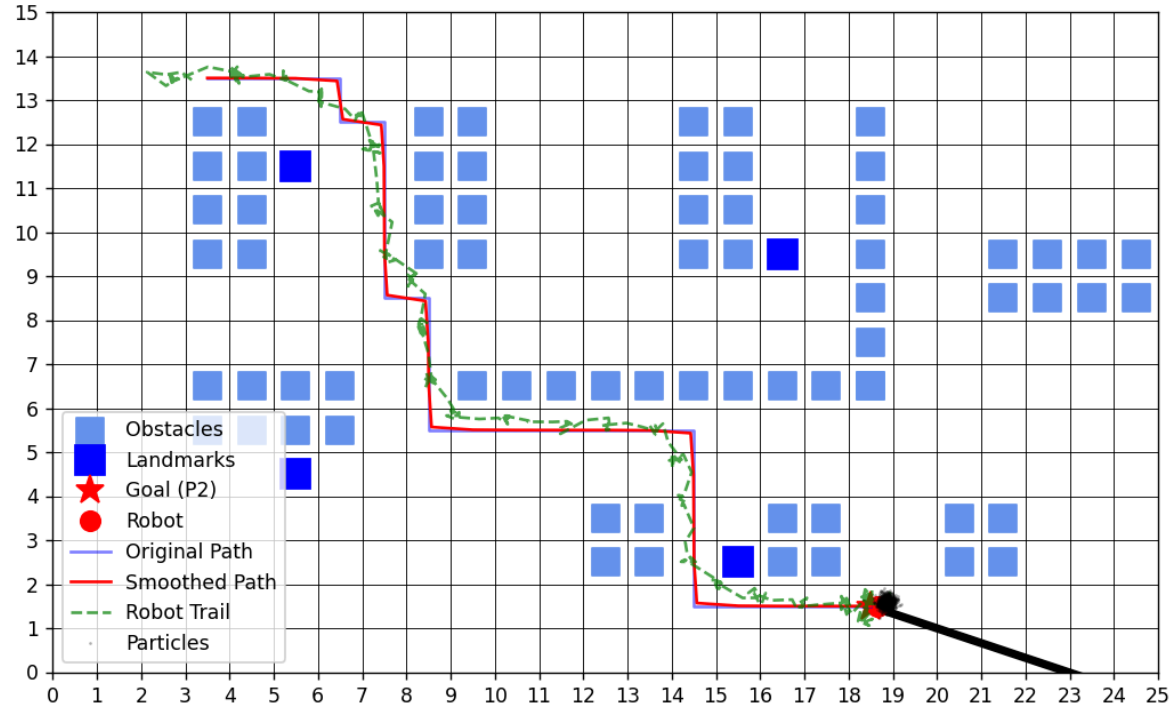
weight_data=0.1 weight_smooth=0.9, max_curvature=0.5(라디안), 반발력 상수=0.2



위 사진은 weight_data=0.1 weight_smooth=0.9 인상황에서 스무딩 경로를 탐색한 예이다 극단적인 앞서서 말했던 비선형 항들(곡률기반 힘,장애물 회피력)을 추가였다. 이로인해 기존의 경로보다 극단적으로 스무딩경로가 변화 하는것을 테스트를 통해 알수 있었다. 앞서서 제시한 max curvature변수로 인해 이 변수가 클수록 더 급격한 회전을 허용하고 더 짧은 경로가 생성된다는 것을 알수 있었다. max_crvature 변수는 임계각도를 제한하는 변수로 numpy의 across 함수를 사용하여 두 벡터의 내적과 두 벡터의 길이의 곱으로 나누어 cos(θ) 를 구한 이후 numpy의 clip함수로 cos(θ)을 -1~1 로 scale을 맞춘후 최종각도(라디안)을 계산하였다. 따라서 곡률을 계산하기 위한 기본적인 방식을 사용하여 곡률 제한을 설정하였다.

다음으론 장애물 회피력에 관한 로직으로 -1,0,1로 총 8방향의 범위(360도를 센싱 개념을 도입) 를 검사하여 장애물이 있다면 장애물까지의거리를 구하고 거리에 반비례하는 힘을 구한 후 -dx/dist 로 장애물로부터 멀어지는 방향을 결정하여 최종적으로 반발력의 크기를 조절하는 상수를 곱하여 requlsive_force를 구하였다. 여기서 나중에 곱하는 반발력 상수는 장애물로부터 얼마나 떨어진 스무딩 경로를 구하느냐에 따라 달라진다.

4-2-1 최종 Planning & Navigation test



문제 5를 위한 PID 알고리즘은 문제 3번의 PID제어 구조를 동일 하게 사용하였으며 최종적으로 particle filter와 개선된 스무딩 알고리즘과 장애물을 고려한 A*알고리즘을 모두 사용하여 시각화한 모습이다.

5. 결론

이번 프로젝트는 로봇의 SLAM, Localization, 경로 계획 및 제어를 구현하며 Unity 환경과 실시간 통신을 통해 실제 로봇 시스템과 유사한 시뮬레이션을 구성한 점에서 의의가 있다. 특히, 다양한 테스트를 통해 여러 시나리오에서의 성능을 검증하고 결과를 시각화하여 직관적으로 확인할 수 있었다.

SLAM에서는 파티클 필터와 그래프 SLAM 알고리즘을 활용하여 로봇의 위치를 추정하였으며, Unity 데이터를 기반으로 다양한 노이즈 조건에서도 안정적으로 동작하도록 설계하였다. Localization 단계에서는 랜드마크와 거리 센서를 활용하여 로봇의 위치를 추정하며, 센서의 최대 범위와 정확도가 성능에 미치는 영향을 실험적으로 분석하고 이를 시각적으로 표현했다. 경로 계획에서는 A* 알고리즘을 사용해 장애물을 회피하는 최적 경로를 생성하고, 경로 평활화 및 PID 제어기를 통해 로봇이 안정적으로 목표 지점에 도달하도록 구현했다. 이러한 과정은 Unity 시뮬레이션에서 로봇의 이동 궤적과 제어 성능을 실시간으로 확인하며 테스트했다.

또한, Unity와 TCP 통신을 통해 로봇 센서 데이터를 실시간으로 수집하고 제어 명령을 전송하며, 실제 로봇 시스템과 유사한 상호작용 구조를 설계하였습니다. 여러 환경 조건과 변수 설정을 바탕으로 다양한 테스트를 수행하며 시스템의 견고성과 성능을 검증하였습니다. 시각화 도구를 활용해 로봇의 위치 추정, 경로 계획 및 이동 궤적을 명확히 표현하며, 결과를 직관적으로 분석할 수 있었다.

결론적으로, 본 프로젝트는 로봇 공학의 이론적 개념을 실무적으로 구현하고 다양한 환경에서 검증하며 SLAM, Localization, Planning, Navigation을 수학적으로 증명하며 각각의 문제들의 해답을 완전히 도출해냈다.이러한 경험은 자율 주행 및 로봇 공학의 복잡한 문제 해결에 소중한 밑거름이 될 것이라고 생각한다.