**프로젝트 제목:** 도슨트 안내 로봇
**버전:** 1.3
**날짜:** [25.07.19]

## 1. 개요

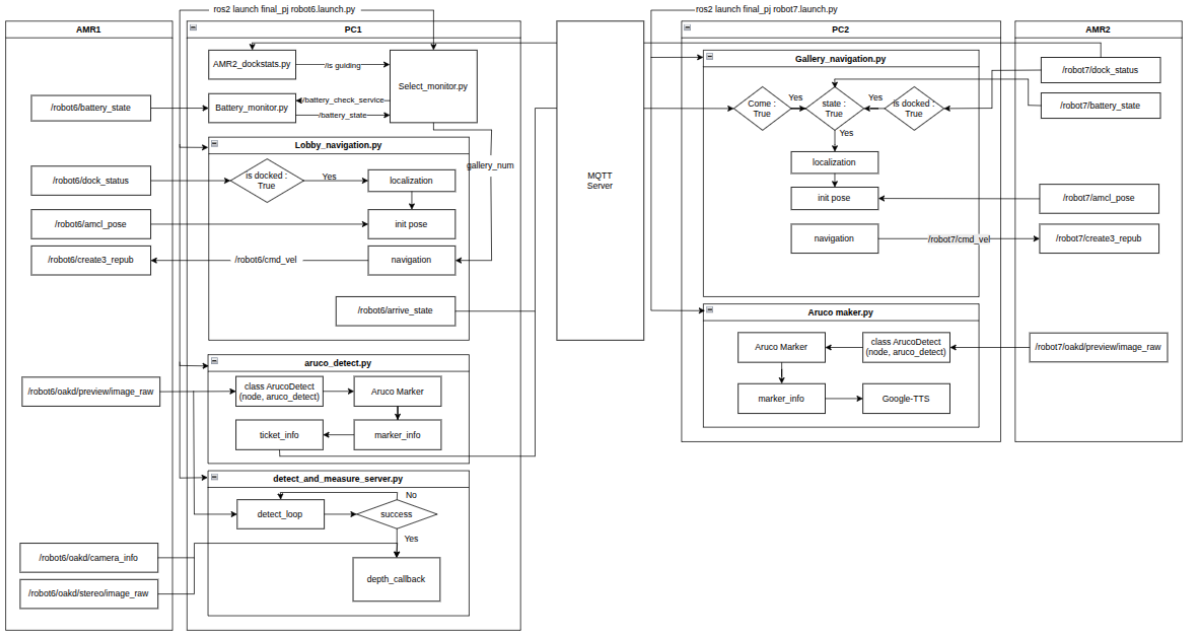
도슨트 로봇(AMR)은 미술관 내에서 전시장까지 길을 안내하는 로봇과 전시장 내에서 작품에 대한 설명을 해주는 로봇으로 구성됩니다. 이 프로젝트 내에서 두 로봇은 각각 도슨트 안내 로봇(AMR 1), 도슨트 설명 로봇(AMR 2)으로 명명되며 두 시스템은 단일 AI 기반 로봇을 사용하여 지도 상에서 경로 이동, 객체 인식/작품 인식 기능을 제공하도록 설계되었습니다.

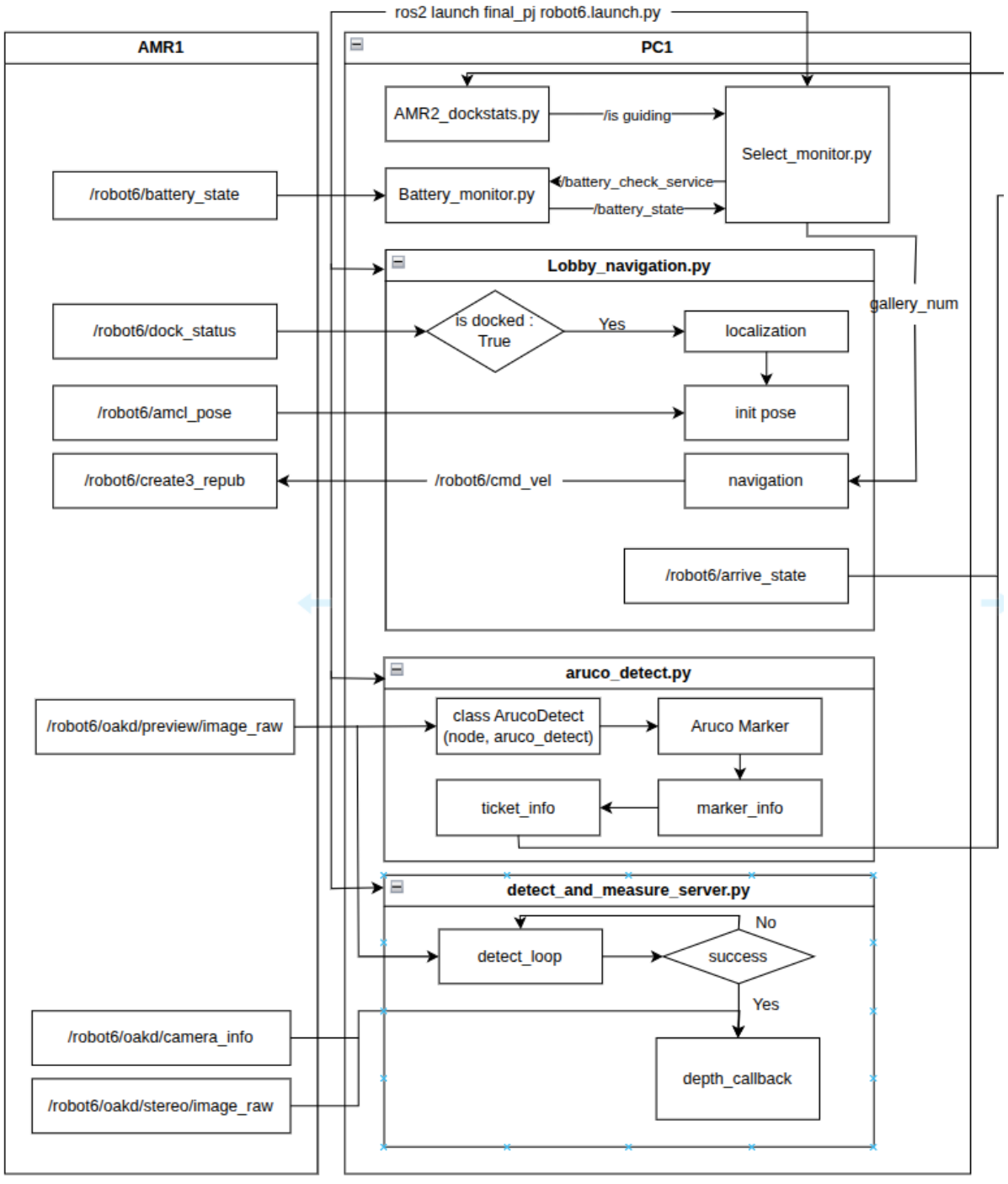
## 2. 시스템 아키텍처

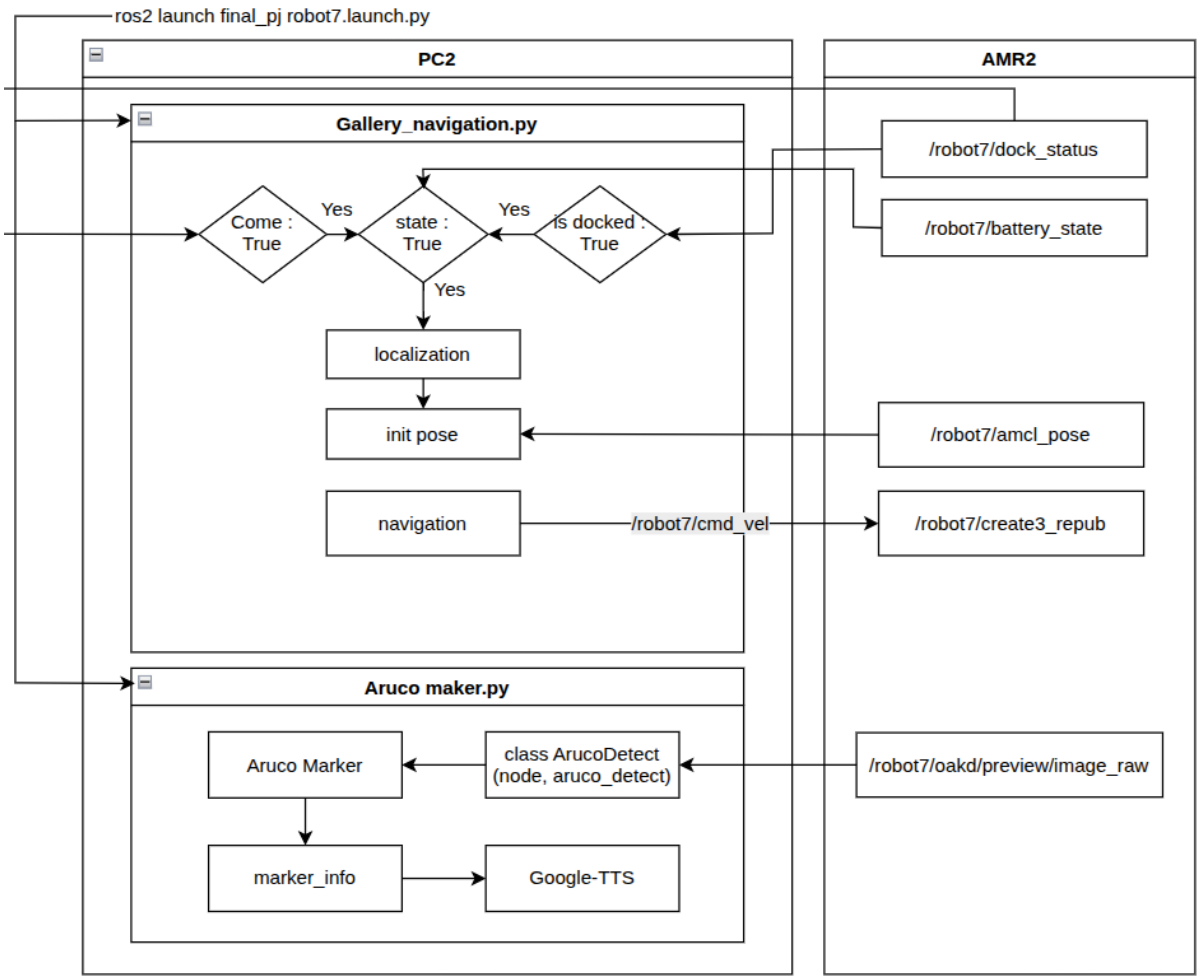
이 시스템은 두 대의 AMR(TurtleBot4)을 기반으로 구성되어 있으며, 각 로봇은 역할에 따라 도슨트 안내 로봇과 도슨트 설명 로봇으로 구분됩니다. 도슨트 안내 로봇은 미술관을 방문한 사용자를 인식한 뒤, 자율 주행 기능을 통해 전시장까지 안전하고 효율적으로 안내하는 기능을 수행합니다. 반면 도슨트 설명 로봇은 전시장 내부에서 작품에 부착된 ArUco 마커를 인식하고, 해당 마커에 연결된 정보를 바탕으로 작품에 대한 설명을 제공하는 역할을 담당합니다.

각 AMR은 개별적으로 할당된 PC를 통해 센서 및 영상 데이터를 실시간으로 처리하며, WIFI 및 ROS2 DDS 통신을 통해 제어 명령을 Raspberry Pi로 전송합니다. 이후 Raspberry Pi가 TurtleBot4를 구동함으로써 로봇의 주행 및 안내 기능이 수행됩니다. 이러한 구조는 각 로봇의 독립적인 기능 수행을 가능하게 하며, 전체 시스템의 안정성과 유연성을 높이는 데 기여합니다.

## 2.1 고레벨 아키텍처 다이어그램




ros2 launch final_pj robot6.launch.py
AMR1
PC1
AMR2_dockstats.py
/is guiding
Select_monitor.py
/robot6/battery_state
Battery_monitor.py
/battery_check_service
/battery_state
Lobby_navigation.py
gallery_num
/robot6/dock_status
is docked : True
Yes
localization
/robot6/amcl_pose
init pose
/robot6/create3_repub
/robot6/cmd_vel
navigation
/robot6/arrive_state
aruco_detect.py
/robot6/oakd/preview/image_raw
class ArucoDetect (node, aruco_detect)
Aruco Marker
ticket_info
marker_info
detect_and_measure_server.py
No
detect_loop
success
Yes
/robot6/oakd/camera_info
depth_callback
/robot6/oakd/stereo/image_raw




## 3. 구성 요소 설계

### 3.1 하드웨어 구성 요소

1. **PC (모니터링 시스템) - AMR 제어용 2대**
2. **운영 체제:** Ubuntu 22.04
    - **네트워크:** AMR과 안정적인 Wi-Fi 연결 통신 지원
3. **자율 이동 로봇 (AMR) – TurtleBot4 2대 (ID 6 & 7)**
    - **프로세서:** Raspberry Pi 4B (On-board Computer)
    - **운영 체제:** Ubuntu 22.04 및 ROS2 (로봇 제어)

- **센서:**
  - **RPLIDAR:** 2m 범위의 360도 레이저 범위 스캐너, 로봇 주변 환경의 2D 스캔을 생성
  - **OAK-D Pro:** 고품질 depth Image 생성, 저조도 환경에도 사용 가능
- **배터리:** 최소 2시간 배터리 수명, 자율 도킹 및 충전 지원

4. **충전 스테이션**
   - 수동 개입 없이 AMR의 충전을 위한 자율 도킹 스테이션

### 3.2 소프트웨어 구성 요소

1. **PC 소프트웨어**
   - **Python3:** 인터페이스 관련 작업을 위한 기본 언어
   - **ROS2:** AMR과의 통신을 위한 플랫폼
   - **OpenCV 및 Ultralytics (YOLO)**: RGB 이미지 기반 한 객체 탐지 및 시각화 처리, Aruco 마커 인식
   - **cv_bridge**: ROS 이미지와 OpenCV 이미지 간 변환을 위한 인터페이스
   - **Google -TTS** : 요청 받은 작품에 대한 설명을 음성으로 출력
2. **AMR 소프트웨어**
   - **ROS2:** 네비게이션, 제어 및 데이터 통합의 핵심 플랫폼
   - **Python3:** AMR 작동 및 AI 처리에 사용하는 기본 프로그래밍 언어
   - **OpenCV 및 Ultralytics (YOLO):** Raspberry Pi 4B 에서 실행되는 객체 감지 및 위협 분석

## 4. 데이터 흐름 설계

1. **데이터 수집**

- **AMR 센서:** AMR은 RPLIDAR, OAK-D Pro 센서를 사용하여 자율 네비게이션을 수행하고 잠재적 위협을 감지합니다.
- **사용자 정보**: MQTT를 사용하여 담당할 사용자를 배정받습니다.

2. **데이터 처리 및 분석**
   - **온보드 처리:** 비디오 및 센서 데이터는 AMR에서 YOLO결과는 PC1에서 처리되어 객체 감지 및 이미지 분석이 이루어집니다.
   - **Aruco 마커 인식:** Aruco 마커가 입력되면 해당 작품에 대한 설명을 출력합니다.

**(1) battery_monitor_node.py**

- 도슨트 안내 로봇(AMR 6번)과 도슨트 설명 로봇(AMR 7번) 공통, AMR 카메라 토픽 /battery_state 토픽을 구독하고 로봇의 배터리 상태 확인.
- 로봇의 배터리 상태가 50% 이상인 경우 /battery_check_service(service)의 response.success = True 응답
- /battery_check_service는 로봇의 이용이 가능하다는 결과를 서비스 응답으로 전송

**(2) select_monitor_mdfy.py**

- battery_monitor_node에게 배터리 상태를 확인 요청.
- battery_monitor_node에게 /battery_check_service(service)의 response.success = True 받음.
- 만약 robot_state_service도 응답 True면 사용자 입력 대기.
- 사용자 입력을 받으면(목적지인 전시관 번호) lobby_guide_navigator_node에 입력을 서비스 요청으로 보냄.

**(3) lobby_guide_navigator_node.py**

- select_monitor_mdfy에서 사용자 입력을 /robot7/gallery_select_service로 받음.

○ 로봇은 Undock한 다음 사용자가 가진 아르코 마커 인식.
○ 사용자 입력에 맞는 전시관으로 이동 시작.
○ waypoint 1에 도착하면, /robot7/cmd_vel를 발행.
○ AMR이 받아서 수치만큼 회전.
○ detect_and_measure 액션 goal를 발행.(거리 측정해라) -> detect_and_measure_server.py가 받음
○ success = True(성공)을 받으면 목적지 진행방향으로 회전한 후 재이동.
○ 목적지에 도착하면, 로봇7에게 도착했다고 서비스 요청.

**(4) detect_and_measure_server.py**

○ lobby_guide_navigator_node에게 액션 요청을 받으면, 객체 검출 시작.
○ AMR의 /robot7/oakd/rgb/preview/image_raw 토픽 받아서 Yolo모델로 객체 검출.
○ 성공시 AMR의 /robot7/oakd/stereo/image_raw 토픽 받아서 AMR과 객체 사이의 거리 측정 시작.
○ 거리 측정이 목표값 이내 들어오면 success = True 액션 결과로 lobby_guide_navigator_node에 보냄.

**(5) select_then_nav_gallery.py**

○ 도슨트 안내 로봇이 서비스 응답 요청시, 도슨트 설명 로봇이 Undock한 다음 사용자 방향으로회전(왼쪽 90도) 한 후 대기.
○ 사용자의 아르코마커 인식, 성공시 도슨트 안내 로봇에게 서비스 응답 보냄.
○ 도슨트 안내 로봇은 서비스 응답 받으면 도킹 스테이션으로 복귀 시작.
○ 도슨트 설명 로봇은 사용자에게 전시관 내 작품 안내 시작.
○ 설명 종료시 도슨트 안내 로봇에게 종료 전달.
○ 도슨트 설명 로봇도킹 스테이션으로 복귀.

**(6) aruco_detect.py**

- ○ /robot7/oakd/rgb/preview/image_raw 토픽에서 실시간 이미지를 수신하여 OpenCV 이미지로 변환
- ○ 변환된 이미지에서 ArUco 마커 검출을 수행하며, 검출된 마커의 ID와 코너 좌표를 추출하여 마커 시각화
- ○ 검출된 마커 ID를 키로 하는 딕셔너리 형태로 전시장 번호 정보(1~20번)와 작품 설명 정보(21~40번)를 관리하여 마커 인식 시 활용함.
  - i. **사용자 인식용 마커(1~20)**가 검출될 경우, 내부 딕셔너리로부터 매칭된 전시장 번호 정보를 참조하여 사용자의 방문 전시장 위치 파악 및 동선 안내에 활용
  - ii. **작품 인식용 마커(21~40)**가 검출될 경우, 해당 작품의 설명을 딕셔너리에서 참조하여 일정 시간 간격을 두고 중복 없이 TTS(음성 안내)를 출력하여, 음성 안내 생성 및 재생 과정에서 발생 가능한 오류는 예외 처리를 통해 방지.
- ○ 마커 처리 결과를 포함한 시각화 이미지를 /image_marked 토픽으로 발행하여 모니터링 및 디버깅 지원

3. **데이터 저장**
   - ○ **작품 설명**: Aruco 마커마다 작품에 대한 설명(TTS, txt)를 도슨트 안내 로봇과 연결된 PC2에 생성, 저장합니다.

## 5. 상세 설계

### 5.1 AMR 네비게이션 및 객체 탐지, 작품 인식

### AMR 공통 기능

- **네비게이션**: ROS2와 SLAM(동시 위치 추정 및 매핑) 및 RPLIDAR를 사용하여 AMR이 생성된 지도 구역 내에서 자율적으로 이동하고 장애물을 회피합니다.

- **충돌 방지:** 사용자, 장애물과의 충돌을 방지하기 위해서 OAK-D Pro 센서를 사용해서 최소 오프셋 거리를 측정 후 사용합니다.
- **사용자 인식**: 각 AMR은 Aruco 마커를 사용하여 본인이 담당한 객체(사람)이 맞는지 확인합니다.

### 5.2 모니터링 시스템 (PC1, PC2)

- **도슨트 안내 로봇의 목적지 선택:**
  - **목적지 결정:** 사용자로부터 도착할 목적지(전시관) 값을 연결된 PC1에서 입력받아 해당 경로를 목적지로 인식, 안내를 시작합니다.
  - **객체(손님) 탐지**: AMR1과 연결된 PC1에서 Yolo 모델을 사용하여 객체 인식. AMR1의 카메라 토픽(okad/rgb/preview/img_raw)을 받아 객체 인식. 객체 인식을 성공하면 AMR1의 카메라 토픽(okad/stereo/img_raw)을 받아 검출된 객체와의 거리를 계산합니다.
- **도슨트 설명 로봇의 요청 처리:**
  - **작품 인식, 설명 요청:** 전시장 내 존재하는 작품의 아르코 마커를 인식하면 그에 맞는 설명을 출력합니다. AMR2와 연결된 PC2에서 처리하며 텍스트, Google TTS(gTTS)로 음성을 출력합니다.

## 6. 보안 설계

- **데이터 암호화:** AMR과 PC 간 전송되는 모든 데이터는 TLS 1.3을 사용해 암호화됩니다.
- **접근 제어:** 대시보드는 다중 요소 인증을 통해 접근이 제한되어 있습니다.
- **로컬 저장 보안:** PC에 저장된 로그 및 경고 데이터는 AES-256을 사용해 암호화하여 민감한 정보를 보호합니다.
- **Broker Address:** EMQX Cloud에서 제공하는 MQTT 브로커 주소 (`*.mqtt.emqxcloud.com`)를 사용하여, 공용 네트워크 상에서도 신뢰할 수 있는 중앙 서버를 통해 통신합니다. 모든 연결은 TLS를 통해 암호화되어 데이터 도청 및 변조를 방지합니다.

- **Port 보안:** '8883' 포트를 사용하며, 이는 MQTT over TLS(MQTTS)를 위한 표준 포트입니다. 일반 MQTT(1883 포트)와 달리 암호화된 전송을 보장하여 보안성을 강화합니다.
- **Client ID:** 각 로봇 또는 제어 PC에 고유한 'Client ID'를 부여하여 연결을 식별합니다. 이로써 무단 연결 탐지 및 추적이 용이하며, 인증 정책 및 권한을 개별 클라이언트 단위로 제어할 수 있습니다.
- **Username / Password:** EMQX Cloud에서 발급한 인증 정보를 이용해 접근 권한을 제어합니다. 비밀번호는 안전하게 관리되어야 하며, 평문 저장 금지 및 주기적인 갱신이 요구됩니다. MQTT 브로커는 잘못된 인증 정보 사용 시 연결을 즉시 거부합니다.
- **TLS 사용:** 'client.tls_set()'을 통해 TLS 기반의 암호화 채널을 구축합니다. 이로써 모든 MQTT 메시지는 중간자 공격(MITM)이나 패킷 가로채기로부터 보호됩니다.
- Topic 구조: '/robot/1/status' 와 같은 명확한 주제 구조를 사용하며, 주제별로 권한을 분리하여 오용을 방지합니다.
- **ACL (Access Control List):** 각 사용자에게 읽기/쓰기 가능한 토픽을 제한하여, 다른 로봇의 명령 주제에 대한 무단 접근을 차단합니다. 이는 시스템 전체의 무결성을 보장합니다.
- **보안 설정 파일 관리:** 인증 정보('username', 'password')는 코드에 직접 노출하지 않고 '.env' 또는 보안 설정 파일을 통해 로드하도록 하여 보안 노출을 최소화합니다.
- **로그 및 모니터링:** EMQX Cloud는 접속/메시지 로그를 기록하여 이상 접근 시 추적이 가능합니다. 이를 통해 사고 발생 시 원인 분석 및 대응이 가능해집니다.

## 7. 성능 요구사항

- **시스템 가동률:** AMR은 최소 1시간 동안 연속 운영이 가능해야 하며, 필요 시 충전 독으로 복귀할 수 있어야 합니다.

- **확장성:** 단일 AMR 운영에 맞춰 설계되었으나, 추가 유닛이 필요할 경우 최소한의 아키텍처 변경으로 지원 가능합니다.
- **객체 감지**: AMR은 YOLO 기반 객체 감지가 가능해야 하며, 최소 0.6의 신뢰도를 가져야 한다.
- **AMR 이동 (Waypoint 지정)**: AMR이 사전에 지정된 waypoint 좌표에 정확히 도달해야 하며, 도착 시 위치 오차는 ±0.2m 이내로 유지되어야 한다.
- **AMR Localization (위치 추정)**: AMR은 초기 pose 설정 이후 지속적으로 자신의 위치를 안정적으로 추정해야 하며, SLAM으로 생성된 맵과의 정확성이 유지되어야 한다.
- **AMR SLAM + Navigation**: AMR은 SLAM 기반으로 실시간 맵을 구성하며, 장애물을 회피하고 목적지까지 자율 주행이 가능해야 한다.
- **객체 인식 (RGB 카메라 기반 YOLO)**: 차량 및 사람 등 주요 객체를 YOLO 모델을 통해 신뢰도(confidence) 0.6 이상으로 안정적으로 인식해야 한다.
- **거리 인식 (Depth Stereo 기반)**: Depth 센서를 통해 추적 대상과의 거리 측정이 ±0.3m 이내 오차로 수행되어야 한다.
- **거리 기반 안내 재개**: 객체 인식으로 사용자가 감지되었더라도 일정 거리(예: 3.0m) 이내로 접근 하지 않았다면 사용자를 기다려야 한다.
- **아르코 마커 인식**: ArUco 마커를 일정 거리의 정면에서 인식할 수 있어야 하며, 마커별 ID를 정확히 구분해야 한다.
- **아르코 작품 안내 (TTS 출력)**: 마커 인식 이후 작품 설명 음성이 1~2초 이내 재생되어야 하며, 내용 누락 없이 출력되어야 한다.
- **사용자 인식:** 도슨트 안내 로봇(AMR)은 최소 10초 이내에 손님을 인식해야 합니다.
- **작품 설명 출력 :** 도슨트 설명 로봇은 작품 설명에 대한 텍스트 정보를 사용자에게 제공해야 합니다.

## 8. 오류 처리 및 복구

- **네트워크 손실:** AMR1 혹은 AMR2의 구동 중 Wi-Fi 연결이 끊기면 각 AMR과 연결된 PC로 ping을 확인, 연결이 복원되면 데이터 전송을 재개합니다.
- **하드웨어 오류:** AMR은 정기적으로 센서 및 하드웨어 상태를 점검합니다. 문제가 감지되면 모니터링 PC에 오류를 보고합니다.
- **사용자(객체)인식 오류**: 도슨트 안내 로봇은 사용자를 10초 내로 인식하지 못하면 사용자가 떠났다고 판단, 도킹 스테이션으로 복귀합니다.
- **안내 중 사용자 인식 오류**: 도슨트 안내 로봇은 목적지까지 이동하며 정해진 중간 포인트마다 뒤로(180도) 돌아 사용자가 안내 경로를 따라 잘 따라오고 있는지 확인합니다. 사용자가 따라오지 않는다면 사용자가 일정 거리로 접근할 때까지 기다립니다.

## 9. 테스트 및 검증

1. **단위 테스트:** ROS2 노드 및 Flask 서버를 포함한 모든 소프트웨어 구성 요소를 개별적으로 테스트하여 기능을 검증합니다.

   **(1) AMR 이동 (Waypoint 지정) 테스트**: AMR이 사전에 지정된 waypoint 좌표에 정확히 도달해야 하며, 도착 시 위치 오차는 ±0.2m 이내로 유지되어야 한다.

   **(2) AMR Localization (위치 추정) 테스트:** AMCL 기반 위치 추정 결과와 수동 제어에 의한 위치 이동 데이터를 비교하여 추정 오류를 시각적으로 확인한다. 지도 상 위치와 실제 위치가 일치하는지 검증한다.

   **(3) AMR SLAM + Navigation 테스트:** 복잡한 경로 및 장애물 환경에서 SLAM 맵 생성이 적절히 이루어지는지 확인하고, 새로운 장애물 등장 시 회피 및 재탐색 경로 생성 여부를 확인한다.

   **(4) 객체 탐지 테스트:** YOLO 기반 모델을 사용, AMR의 카메라를 통해 객체 감지를 확인합니다.

- ○ AMR을 통해서msg(service / action) 요청이 발생한 상황을 client/action_client 파일로 구현해 대신합니다.
- ○ msg(service / action) 요청이 발생하면 AMR의 카메라 토픽 /oakd/rgb/preview/image_raw, /oakd/stereo/image_raw 을 받아오며 객체 감지가 동작.
- ○ 객체가 감지되지 않았다면 객체 인식 성능을 측정하고, 해당 결과를 터미널에 출력.
- ○ 객체가 감지되었고 신뢰도 0.6 이상이면 depth 계산 실행, 객체 감지 노드와 depth 계산 노드는 멀티쓰레드 형태로 구성되어있으며 하나의 노드에서 병렬적으로 실행됩니다.
- ○ depth값이 계산되었고 특정 기준값 이내(3.0m 이내)면 목표가 달성했다는 응답을 서버로 송신합니다.

**(5) 거리 인식 (Depth Stereo 기반) 테스트**: 실제 측정 거리와 센서 측정 값을 비교하고, 중심 좌표 기반 평균 depth 값을 통해 정밀도 테스트를 수행한다.

**(6) 거리 기반 안내 재개 테스트**: 사용자가 멀리 있을시 사용자를 기다리고 일정 거리내(3.0m) 내로 들어왔을 시 다시 안내를 진행하는 테스트를 수행한다.

**(7) 아르코 마커 인식 테스트**: 아르코 마커 인식 기능의 안정성과 정확성을 검증하기 위한 테스트는 aruco_detect.py 노드를 기반으로 수행된다.

- ○ aruco_detect.py 노드는 AMR의 RGB 카메라로부터 /oakd/rgb/preview/image_raw 토픽을 구독하여 이미지 데이터를 수신하고, OpenCV의 aruco.detectMarkers() 함수를 활용해 마커를 검출한다.
- ○ 테스트는 다양한 ID의 ArUco 마커를 정면에서 제시, 마커 인식 성공 여부는 로그를 통해 확인한다. 마커가 인식되었을 경우

drawDetectedMarkers() 함수가 호출되어 내부 이미지 처리 결과가 퍼블리시되며, 인식된 마커 ID가 marker_info 딕셔너리 내에 포함되는지를 판별한다.

- ○ 마커가 인식되지 않은 경우에는 조도, 거리, 각도 등 조건을 변경하여 다시 시도하며, 인식 실패 원인을 기록한다. 또한 rosbag을 활용해 다양한 환경(정지, 이동 중, 사용자 접근 시)에서의 인식 성능을 추가적으로 검증한다.

**(8) 아르코 마커 TTS 출력 테스트**: TTS 출력 테스트는 아르코 마커 인식 이후 연결된 작품 설명이 음성으로 정확히 재생되는지를 확인하기 위한 단계다. 이 기능은 aruco_detect.py 내부의 say_text() 함수와 Google TTS(gTTS), playsound 모듈을 기반으로 수행된다.

- ○ 아르코마커가 인식되면 해당 ID에 매핑된 설명 텍스트가 marker_info 딕셔너리로부터 조회되며, 해당 문자열이 gTTS를 통해 음성으로 변환되고, 생성된 MP3 파일이 playsound를 통해 즉시 재생된다. 이때, 음성 재생 여부, 지연 시간 등을 함께 확인한다.
- ○ 또한, 동일 마커에 대해 일정 시간 간격 내 중복 음성 출력이 되지 않도록 구현된 self.last_spoken 딕셔너리와 self.speech_interval 기반 로직의 정상 동작 여부도 함께 검증한다. 예를 들어, 마커를 반복적으로 보여주는 경우에도 음성 출력 간 최소 간격(예: 10초)이 유지되는지 확인한다.
- ○ rosbag을 사용하여 시나리오를 재현하면서 마커 인식 → 텍스트 출력 → 음성 재생의 전체 흐름이 잘 동작하는지를 확인하고, TTS 음성이 끝난 후 시스템이 다음 안내를 받을 준비 상태인지도 함께 검토한다.

2. **통합 테스트:** AMR과 모니터링 PC를 통합하여 원활한 데이터 흐름과 명령 실행을 보장합니다.
3. **현장 테스트:** 목표 보안 구역에서 AMR을 테스트하여 자율 네비게이션, 위협 탐지 및 경고 기능을 실제 조건에서 검증합니다.

4. **통신 테스트**: EMQX Cloud, MQTT 가입 후 제공받은 클라우드 서버를 이용하여 한 PC 내 토픽 통신, 다른 PC와의 토픽 통신 여부를 확인한다.이때 python3 기반과 Json 메시지 모두 테스트 후 결과를 확인한다.

## 10. 배포 및 유지보수 계획

- **배포 단계:**
  - PC에 ROS2를 설치합니다.
  - 모니터링 대시보드를 구성하고 AMR을 Wi-Fi로 연결합니다.
- **유지보수 일정:**
  - 매월 AMR과 PC의 소프트웨어 업데이트로 보안 패치 적용
  - 분기마다 AMR의 하드웨어 점검, 센서 보정 및 배터리 검사

## 11. 부록

- **라이브러리 및 종속성:** ROS2, OpenCV, YOLO 등 사용된 소프트웨어 종속성 목록
- **용어 사전:** AMR, SLAM, YOLO 등 주요 용어의 정의
- **참고 문헌:** ROS2, Raspberry Pi 4B 4GB, TurtleBot4 및 관련 기술에 대한 문서