System Design Document (SDD)

프로젝트 제목: Auto Valet Garage(무인 주차 관제 시스템)
날짜: 2025.12.04
버전: 5.1

—

1. 개요 (Overview)

Auto Valet Garage는 도심지의 심각한 주차난과 이중 주차로 인한 사회적 비용을 기술적으로 해결하기 위해 고안된 AMR(Autonomous Mobile Robot) 기반의 무인 자동 주차 관제 시스템입니다. 본 프로젝트는 기존 유인 주차 방식의 한계를 극복하고, AI 기반의 정밀 상황 인식과 다중 로봇 군집 제어(Fleet Management) 기술을 융합하여 입차부터 주차, 출차에 이르는 전 과정을 무인 자동화(Full-Automation)함으로써 주차 공간의 효율성을 극대화하고 사용자에게 최상의 편의성을 제공하는 것을 목표로 합니다.

본 시스템은 기능적 독립성과 안정성을 확보하기 위해 역할이 분리된 2대의 제어 PC를 중심으로 운용됩니다. PC 1(입차 관리 서버)은 입차 구역의 CCTV 영상을 YOLOv8 모델로 분석하여 차량 인식 및 최적의 주차 구역 배정 로직을 전담하며, 입차 요청 시 AMR에게 해당 위치로 이동하여 차량을 이송하도록 명령을 하달합니다. PC 2(출차 관리 서버)는 모든 출차 요청을 수신하여 일반 출차뿐만 아니라 이중 주차 상황 발생 시 협동 제어 알고리즘을 통한 장애물 해결 및 출차 프로세스를 총괄합니다. 결제 처리는 사용자의 편의를 위해 별도의 웹 기반 결제 시스템에서 독립적으로 이루어지며, 결제 완료 정보가 PC 2로 연동되어 출차 로직을 트리거합니다. 또한, 관리자는 장소에 구애받지 않고 전체 시스템을 감독할 수 있도록 웹 기반의 별도 모니터링 시스템을 통해 실시간 관제 환경을 제공받습니다. 특히, 본 시스템은 출차 경로상의 이중주차된 차량을 감지하면 복수의 로봇이 협력하여 해당 차량을 안전 위치로 이동시키고, 출차 완료 시 원상 복구하는 협동 제어(Cooperative Control) 알고리즘을 핵심 솔루션으로 탑재하고 있으며, 본 문서는 이러한 기술적 요소들을 축소 모형 환경에서 검증하기 위한 구체적인 설계 사양을 정의합니다.

—

2. 시스템 아키텍쳐 (System Architecture)

본 시스템은 기능 분산형 제어 아키텍처(Functionally Distributed Control Architecture)를 기반으로 설계되었습니다. 시스템의 핵심 로직인 입차와 출차 프로세스를 물리적으로 분리된 2대의 제어 PC에서 각각 전담하여 처리 부하를 분산하고, 전체 시스템의 안정성과 확장성을 확보합니다. 모든 구성 요소는 Wi-Fi 6 무선 네트워크 환경에서 ROS 2 FastDDS (Data Distribution Service) 미들웨어를 통해 실시간으로 데이터를 교환하며 유기적으로 연동됩니다.

2.1 주요 설계 특징

• 기능별 역할 분산 및 전문화:

- PC 1 (입차 관리 서버): 입차 구역에 설치된 카메라 영상을 실시간으로 수집하고, YOLOv8 기반의 AI 추론 엔진을 구동하여 진입 차량을 인식합니다. 차량의 종류(Class)를 식별하고 주차장 전체 맵 정보를 바탕으로 최적의 빈 주차 구역을 배정한 뒤, AMR에게 이동 명령(Action)을 하달하는 입차 프로세스를 전담합니다.
- PC 2 (출차 관리 및 협동 제어 서버): 사용자의 출차 요청을 수신하고 처리합니다. 특히, 출차 경로 상에 장애물(이중 주차 차량)이 감지될 경우, 다중 로봇 협동 제어(Cooperative Control) 알고리즘을 수행하여 복수의 로봇에게 장애물 회피 및 원상 복구 명령을 내리는 고도화된 출차 시퀀스를 총괄합니다.

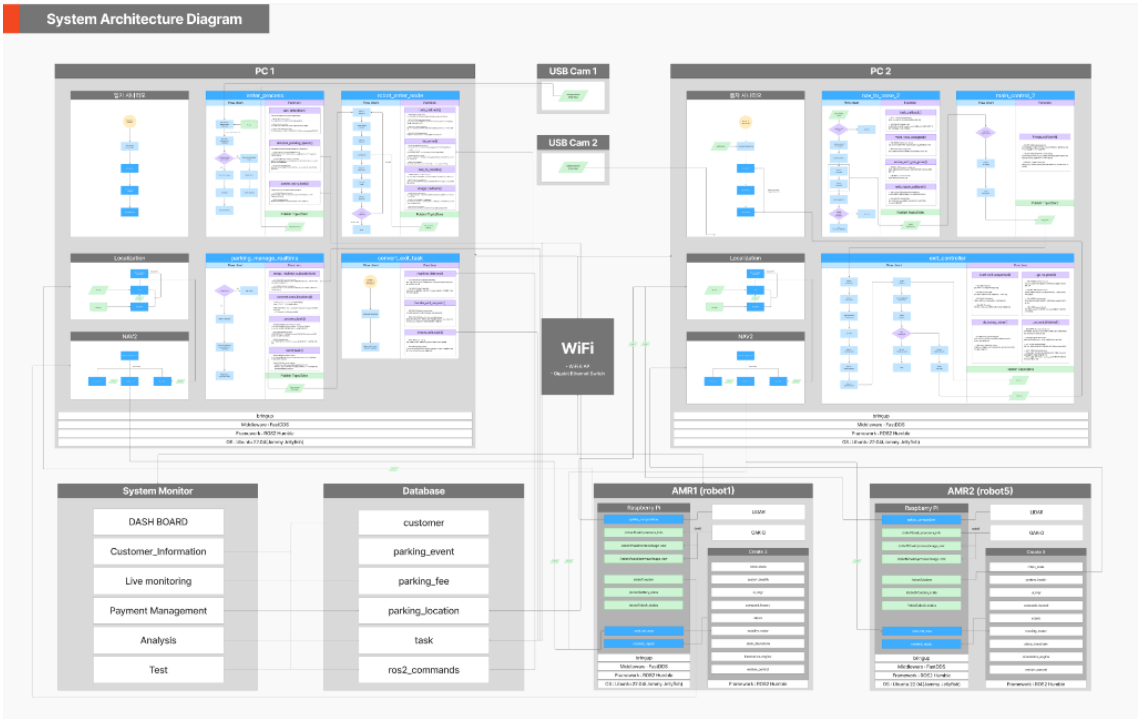
• 이중 주차 해결을 위한 협동 제어 (Cooperative Control):

- 출차 시 이중 주차 차량으로 판단되면, PC 2는 즉시 '협동 모드'를 활성화합니다.
- 서버는 두 대의 로봇을 동시에 호출하여, 한 대는 장애물 차량을 임시 구역(Buffer Zone)으로 이동시키고, 다른 한 대는 대상 차량을 출차시키는 정교한 다단계 시퀀스(Sequence)를 제어하여 충돌 없는 안전한 출차를 보장합니다.

• 웹 기반 실시간 모니터링:

- 관리자는 장소에 구애받지 않고 시스템 상태를 확인할 수 있도록 별도의 웹 기반 모니터링 시스템을 사용합니다.
- 서버로부터 전송받은 로봇의 배터리 상태, 작업 진행률 등을 직관적인 대시보드 형태로 시각화하며, 비상 상황 발생 시 웹 인터페이스를 통해 비상 정지나 수동 제어 명령을 내릴 수 있습니다.

2.2 고레벨 아키텍처 다이어그램(High-Level Architecture Diagram)



※ 별첨서류 : (별첨) System Architecture Diagram.pdf

—

3. 구성 요소 설계 (Component Design)

3.1 하드웨어 구성 요소(Hardware Components)

3.1.1 PC1(입차 관제 서버)

• 장비: MSI 고성능 랩탑
  - 운영 체제: Ubuntu 22.04 (Jammy Jellyfish)
  - 프레임워크: ROS 2 Humble
  - CPU: Intel Ultra9 185H (다중 로봇 경로 계획 및 스케줄링 연산용)
  - GPU: NVIDIA RTX 4070 8GB(YOLOv8 실시간 객체 인식 및 학습용)
  - 네트워크: Wi-Fi 6 (802.11ax) 지원 무선 랜카드 (다중 로봇과의 저지연 통신 보장)

• 역할:
  - ROS 2 Master: 전체 노드 간 통신 중재 및 Discovery Server 역할 수행
  - AI Inference: CCTV 영상 수신 및 실시간 차량 인식 수행

3.1.2 PC2(출차 관제 서버)

• 장비: MSI 고성능 랩탑
  - 운영 체제: Ubuntu 22.04 (Jammy Jellyfish)
  - 프레임워크: ROS 2 Humble
  - CPU: Intel Ultra9 185H (다중 로봇 경로 계획 및 스케줄링 연산용)
  - GPU: NVIDIA RTX 4070 8GB(YOLOv8 실시간 객체 인식 및 학습용)
  - 네트워크: Wi-Fi 6 (802.11ax) 지원 무선 랜카드 (다중 로봇과의 저지연 통신 보장)

• 역할:
  - Manual Override: 비상 정지 및 로봇 원격 수동 제어(Teleoperation) 입력 장치

3.1.3 자율 이동 로봇 (AMR)

• 장비 : TurtleBot4 (총 2대 운용)
  - 베이스 플랫폼: iRobot Create 3 (이동성 및 배터리 관리)
  - 메인 제어기: Raspberry Pi 4 Model B (4GB RAM)
  - 센서 시스템:
    1) RPLIDAR A1M8: 360도 2D 레이저 스캐너 (2m 범위, SLAM 및 동적 장애물 회피용)
    2) OAK-D Pro: AI 기반 심도(Depth) 카메라 (고품질 Depth Image 생성, 저조도 환경 객체 탐지)
    3) IMU: 9축 관성 측정 장치 (회전각 및 이동 거리 보정)
  - 운영 체제: Ubuntu 22.04 Server (ROS 2 Humble 호환)
  - 전원: 리튬이온 배터리 (최소 2시간 배터리 수명, 자동 도킹 및 충전 지원)

3.1.4 인프라 센서 및 충전 스테이션

• 입구 카메라 (WebCam 1): 입차 구역에 고정 설치되어 진입 차량의 이미지를 캡처하고 서버로 전송합니다. (1080p, 30fps)

• 천장 카메라 (WebCam 2): 주차장 전체를 조망하는 위치에 설치되어, 주차 구역의 점유

상태를 2차 검증(Double Check)하고 사각지대를 보완합니다.

• 충전 스테이션: 수동 개입 없이 AMR의 자동 충전을 지원하는 도킹 스테이션입니다.

3.2 소프트웨어 구성 요소(Software Components)

3.2.1 PC 소프트웨어 패키지 (Server-Side)

서버에서 구동되는 소프트웨어 스택은 데이터 처리, 판단, 통신의 핵심 로직을 포함합니다.

• 운영 환경: Python 3.10+ (비동기 처리 지원)

• 미들웨어 (Middleware):
- ROS 2 Humble: FastDDS(Data Distribution Service) 기반의 실시간 로봇 통신 프레임워크
- Nav2 Server: 다중 로봇 경로 계획을 위한 Navigation Stack 서버 인스턴스

• AI & 영상 처리:
- Ultralytics YOLOv8n: RGB 이미지 기반 차량 탐지 및 분류 모델
- OpenCV: 영상 전처리(Resize, Color Space Convert) 및 주차 라인 추출 알고리즘
- cv_bridge: ROS 이미지 메시지와 OpenCV 이미지 포맷 간 변환 인터페이스

• 백엔드 & 데이터베이스:
- REST API : 클라이언트 요청 수신 및 로봇 제어 명령을 연결하는 통신 인터페이스
- Supabase cloud(AWS, PostgreSQL): 실시간 주차 현황 및 차량 입출차 로그를 관리하는 데이터베이스

• 프론트엔드:
- React : 웹 기반 대시보드 및 사용자 인터페이스

3.2.2 AMR 소프트웨어 패키지 (Client-Side)

로봇 내부에서 실행되는 소프트웨어는 주행 제어와 센서 데이터 처리에 집중합니다.

• 운영 환경: Python 3.10+, C++ (고성능 센서 드라이버용)

• 제어 및 통신:
- Micro-ROS Agent: 라즈베리 파이와 하위 제어기(Create 3) 간의 통신 브리지
- Robot State Publisher: 로봇의 TF(Transform) 트리 및 상태 정보 브로드캐스팅

• 자율 주행 (Autonomous Navigation):
- SLAM Toolbox: 주차장 환경 지도 작성(Mapping) 및 저장
- AMCL: 2D LiDAR 스캔 매칭을 통한 실시간 자기 위치 추정(Localization)
- Nav2 Stack:
  1) Planner Server: 최단 경로 생성 (Global Path)
  2) Controller Server: 동적 장애물 회피 및 경로 추종 (DWB Controller)
  3) Behavior Tree: 주행, 대기, 리프팅, 복귀 등의 복합 행동 시퀀스 관리
- Cooperative Package: 다중 로봇 제어를 위한 커스텀 패키지 (협동 제어 로직 수행)

• 비전 처리 (Vision):

- DepthAI API: OAK-D 카메라의 하드웨어 가속을 활용한 경량 객체 인식 및 위협 분석
- OpenCV: 이미지 전처리 및 특징점 추출

—

4. 데이터 흐름 설계 (Data Flow Design)

4.1 데이터 수집 (Data Collection)
시스템은 외부 환경(주차장)과 로봇의 상태를 인지하기 위해 다양한 센서로부터 원시 데이터를 수집합니다.

• 인프라 영상 데이터 (Infrastructure Vision):
- 입차 카메라 (WebCam 1): 입차 구역(Entry Zone)에 진입하는 물체를 감지하기 위해 1920x1080 해상도의 영상을 RTSP 프로토콜로 중앙 서버(PC 1)에 스트리밍합니다.
- 천장 카메라 (WebCam 2): 주차장 전체를 조망하는 Top-down 영상을 수집하여, 주차 구역의 실제 점유 여부와 로봇/차량의 위치를 파악하는 데 사용합니다.

• AMR 센서 데이터 (Robot Telemetry):
- LiDAR: 로봇 주변 360도 반경의 거리 데이터를 /scan 토픽으로 수집하여 SLAM 및 장애물 감지에 활용합니다.
- OAK-D 카메라: 로봇 전방의 RGB 및 Depth 이미지를 /oakd/rgb/preview 토픽으로 수집합니다.
- Odometry 엔코더 기반의 주행 정보(/odom)를 1초 주기로 수집합니다.
- Battery: 배터리 전압/잔량 정보(/battery_state)를 10분 주기로 수집합니다.

• 사용자 입력 데이터 (User Input):
- 웹 인터페이스를 통해 사용자의 입차 요청, 결제 정보, 출차 요청(차량 번호) 데이터를 보여줍니다.

4.2 데이터 처리 및 분석 (Data Processing & Analysis)
수집된 데이터는 중앙 관제 서버(PC1)에서 알고리즘을 통해 의미 있는 정보로 변환되고, 로봇 제어 명령을 생성합니다.

• 객체 인식 및 분류 (AI Perception):
- 입차 인식: enter_process 노드는 입차 카메라 영상을 프레임 단위로 캡처하여 YOLOv8n 모델로 추론합니다. 차량(Car)으로 분류되고 신뢰도(Confidence)가 0.75 이상일 경우 입차 프로세스를 트리거합니다.
- 랜덤 ID 생성: 차량 번호 인식 과정을 대체하여, 시스템 내부적으로 7자리 난수(Random ID)를 생성하고 해당 차량의 고유 식별자로 할당합니다.
• 경로 계획 및 작업 할당 (Planning & Allocation):
- 주차 배정: DB에서 Empty 상태인 주차 구역을 조회하고 배차합니다.
- 협동 제어 로직: 출차하려는 차량 앞에 이중 주차 차량이 있다면, exit_controller 노드가 활성화되어 [장애물 이동 -> 출차 -> 원복] 순서의 시퀀스 명령을 생성하고 AMR 1, 2에게 작업을 분배합니다.
• 결제 및 요금 산정:

- 출차 요청 시점의 타임스탬프와 DB에 저장된 Entry Time의 차이를 계산하여 주차 요금을 산정합니다.

4.3 데이터 저장 (Data Storage)
처리된 데이터와 시스템 로그는 데이터베이스 및 로컬 스토리지에 저장되어 관리됩니다.

• 관계형 데이터베이스 (PostgreSQL):
- Task table: 각 로봇이 어느 구역에 입차/출차를 해야하는지, 어떤 로봇이 할당 받았는지, 일을 마쳤는지 등을 저장합니다.
- Logs Table: 시스템 오류, 입출차 트랜잭션, 로봇 배터리 상태 기록 등을 타임스탬프와 함께 영구 저장합니다.

4.4 경고 및 알림 (Alerts & Notifications)
• 관리자 알림 (Admin Alarm)
- 로봇 배터리 부족 (Low Battery):
1) 조건: AMR이 전송하는 /battery_state 토픽을 모니터링하여, 배터리 잔량이 30% 미만으로 떨어질 경우 즉시 알림을 발생시킵니다.
2) 조치: 대시보드에 해당 로봇의 상태를 '충전 필요(Low Battery)'로 표시하고, 현재 작업을 마치는 대로 충전 복귀 명령을 준비합니다.
- 로봇 작업 실패 (Task Failure):
1) 조건: 로봇이 물리적 경로 막힘, 하드웨어 오류(센서/모터), 또는 위치 소실(Localization Lost)로 인해 할당받은 태스크를 완수하지 못하고 'Aborted' 상태를 반환할 경우 알림을 발생시킵니다.
2) 조치: 대시보드에 [작업 실패] 팝업을 띄우고, 관리자가 원격 제어(Teleoperation)나 현장 조치를 취할 수 있도록 실패 위치와 원인 코드를 표시합니다.
• 로봇 오디오 피드백:
- 목표 좌표 도착 시 로봇 내부 스피커를 통해 행동을 대체합니다.

—

5. 상세 설계 (Detailed Design)

본 섹션에서는 시스템의 핵심 기능을 수행하는 소프트웨어 노드(Node)들의 내부 로직, 상태 전이(State Transition), 그리고 데이터 처리 파이프라인을 상세히 기술합니다.

5.1 AMR 네비게이션 및 주행 제어 (Navigation & Control)
• AMR 공통 기능:
- 네비게이션 (Navigation): ROS 2 Nav2 Stack을 기반으로 구동됩니다. Planner Server는 주차장 정적 맵(Static Map) 위에서 A* 알고리즘으로 전역 경로를 생성하고, Controller Server는 DWB(Dynamic Window Approach) 컨트롤러를 사용하여 동적 장애물(타 로봇)을 실시간으로 회피하며 경로를 추종합니다.
- 위치 추정 (Localization): AMCL(Adaptive Monte Carlo Localization) 알고리즘을 사용하여 2D LiDAR 스캔 데이터와 사전 작성된 맵을 매칭해 로봇의 현재 위치(Pose)를 실시간으로

보정합니다.
- 정밀 정렬 (Fine Docking): 주차 구역 진입 시, OAK-D Pro 카메라가 바닥의 주차 라인(Line)을 인식하여 로봇의 Yaw(회전각) 및 Lateral(측면) 오차를 ±2cm 이내로 보정하는 Align_Action을 수행합니다.

5.2 서버 측 핵심 노드 및 로직 (Server-Side Logic)
서버 시스템은 입차, 주차 배정, 출차, 로봇 제어 등 핵심 기능을 담당하는 ROS 2 노드들의 집합으로 구성되며, 각 노드는 Topic, Service, Action 인터페이스를 통해 유기적으로 데이터를 교환합니다.

5.2.1 enter_process.py (입차 관리 및 판단 노드)
입차 구역의 상황을 인식하고, 적절한 주차 공간을 배정하여 입차 태스크를 생성하는 의사 결정 노드입니다.

• 역할 (Role):
- 차량 인식 및 분류: YOLOv8 모델을 활용하여 진입 차량을 감지하고 차종을 분류합니다.
- 주차 공간 탐색: 데이터베이스를 조회하여 가용한 주차 공간을 탐색합니다.
- 이중 주차 판단: 만차 시 이중 주차 가능 여부를 판단하고 태스크를 생성합니다.

• 주요 함수 (Key Functions):
- run_detection(): YOLO 모델을 구동하여 객체 감지 및 유형 분류(소/중/대형)를 수행합니다.
- allocate_parking_space(): DB를 조회하여 빈 주차면을 필터링하고, 동일 타입 우선 배정 또는 대체 공간 탐색 로직을 수행합니다.
- create_entry_task(): 생성된 Task 정보를 시스템 스키마에 맞게 가공하고 DB에 새로운 레코드로 등록합니다.

• 동작 흐름 (Operational Flow):
1) Classification of vehicle types (YOLOv8n):
- 입차 구역 카메라 영상을 best.pt 가중치를 로드한 YOLOv8 모델에 입력합니다.
- 차량을 대형(Large), 중형(Mid), 소형(Small)으로 분류합니다.
2) Search all parking spaces in database:
- PostgreSQL 데이터베이스에서 현재 모든 주차면의 상태 정보를 조회합니다.
3) Is there a parking space inside? (Decision):
- YES (내부 공간 있음): Place in inner space 단계로 이동하여 안쪽 일반 주차면을 배정합니다.
- NO (내부 공간 없음): Place as double parking space 단계로 이동하여 이중 주차 구역을 배정합니다.
4) Create task:
- 배정된 주차면 ID, 차량 정보(차종, 번호), 작업 유형(입차)을 포함한 태스크 객체를 생성합니다.
5) Save to database:
- 생성된 태스크 정보를 DB에 기록하여 트랜잭션을 시작합니다.

6) Publish Topic/Data:
- parking_manage_realtime 노드로 생성된 태스크 정보를 토픽으로 발행합니다.

5.2.2 parking_manage_realtime.py (작업 할당 및 좌표 변환 노드)
논리적인 주차 태스크를 로봇이 실행 가능한 물리적 좌표 명령으로 변환하고, 적절한 로봇에게 작업을 할당하는 매니저 노드입니다.

• 역할 (Role):
- 태스크 큐 관리: 대기 중인 주차/출차 태스크를 관리합니다.
- 좌표 변환: 주차면 ID를 로봇 네비게이션을 위한 Global 좌표로 변환합니다.
- 작업 할당: 가용한 AMR을 선정하여 작업을 지시합니다.

• 주요 함수 (Key Functions):
- setup_realtime_subscription(): DB의 변경 사항을 실시간으로 감지하고 조건에 맞는 작업(Pending 상태)을 필터링합니다.
- convert_task_locations(): 주차면 ID를 좌표로 변환하고, 안전한 이동을 위한 경로(Start -> Waypoint -> Target)를 생성합니다.
- process_task(): 태스크 타입(입차/단일출차/이중출차)을 식별하고 실행 카테고리(Simple/Double Task)로 분류합니다.
- send_task(): 지정된 로봇의 명령 토픽으로 메시지를 발행하고 전송을 보장합니다.

• 동작 흐름 (Operational Flow):
1) Is the task in? (Monitoring):
- NO: Wait a task 상태로 대기하며 신규 태스크 수신을 기다립니다.
- YES: 신규 태스크가 수신되면 다음 단계로 진행합니다.
2) Receive a notification:
- enter_process 또는 convert_exit_task로부터 태스크 생성 알림을 받습니다.
3) Converting registered parking positions to coordinates:
- 태스크에 포함된 주차면 ID(예: A-1)를 사전에 정의된 맵 좌표 데이터베이스와 매핑하여 (x, y, yaw) 좌표로 변환합니다.
4) Publish topics to be sent to AMR:
- 변환된 좌표와 작업 명령을 담아 특정 AMR(예: /robot1/cmd_task)에게 토픽을 발행합니다.

5.2.3 convert_exit_task.py (출차 요청 처리 및 데이터 동기화 노드)
사용자의 출차 요청을 시스템 내부 태스크로 변환하고, 관련 데이터베이스 상태를 동기화하여 출차 프로세스를 시작하는 노드입니다.

• 역할 (Role):
- 사용자 요청 수신: 앱/키오스크로부터 출차 요청을 받습니다.
- DB 업데이트: 차량 상태 및 주차면 상태를 출차 중으로 변경합니다.
- 태스크 생성: 출차 태스크를 생성하여 매니저 노드에 전달합니다.

• 주요 함수 (Key Functions):

- realtime_listener(): 출차 요청 테이블을 모니터링하며 신규 삽입(INSERT) 이벤트를 포착합니다.
- handle_exit_request(): 원시 요청 데이터를 인출하여 시스템 스키마 규격에 맞게 포맷팅합니다.
- create_exit_task(): 가공된 데이터를 인자로 받아 Task Table에 새로운 출차 작업으로 등록(Commit)합니다.

• 동작 흐름 (Operational Flow):

1) Customer's application to leave the car:
   - 사용자가 앱에서 '출차 요청' 버튼을 누르면 API를 통해 요청이 수신됩니다.
2) Update another database:
   - 결제 시스템 등 외부 DB와 연동하여 결제 완료 여부를 확인하고 상태를 동기화합니다.
3) Update task DB values from other databases:
   - 내부 태스크 DB에 출차 작업 정보를 생성하고, 해당 차량의 상태를 EXIT_PROCESS로 갱신합니다.
4) Publish Topic/Data:
   - 생성된 출차 태스크를 parking_manage_realtime 노드로 발행하여 로봇 할당을 요청합니다.

5.2.4 robot_enter_node.py (로봇 입차 시퀀스 제어 노드)

할당받은 입차 태스크를 수행하기 위해 AMR의 이동, 정렬, 리프팅 등 세부 동작을 순차적으로 제어하는 상태 머신(State Machine) 노드입니다.

• 역할 (Role):
- 주행 제어: Nav2 액션을 통해 목표 지점으로 이동합니다.
- 정밀 제어: 카메라 기반 라인 인식을 통해 정밀 주차를 수행합니다.
- 작업 상태 관리: 작업 완료 여부를 판단하고 다음 작업을 결정합니다.

• 주요 함수 (Key Functions):
- task_callback(): 로봇의 가용 상태(Busy/Idle)를 점검하고 자신에게 할당된 명령을 필터링하여 수신합니다.
- do_enter(): 파라미터를 파싱하고 전체 입차 시퀀스(Undock -> Pick -> Place -> Return)를 순차적으로 호출합니다.
- nav_to_coords(): 내비게이터 객체를 통해 경로 생성 및 추종(Path Planning & Following)을 수행합니다.
- image_callback(): OAK-D 카메라 영상에서 ROI 설정, HSV 필터링, 윤곽선 검출을 통해 주차 라인을 인식합니다.

• 동작 흐름 (Operational Flow):

1) Move to start point:
   - 입차 구역(Pick-up Zone)으로 이동하여 대기 중인 차량 앞으로 접근합니다.
2) Pick up the car on AMR:
   - 차량 하부로 진입하여 가상 리프팅 동작(LED 점멸/대기)을 수행합니다.
3) Move to way-point:
   - 주차 구역 진입 전, 안전한 경로 확보를 위해 경유지(Waypoint)로 이동합니다.
4) Line detection:

- OAK-D 카메라를 이용하여 바닥의 주차 라인을 인식하고, 로봇의 정렬 오차 (Lateral/Heading Error)를 계산합니다.

5) Move to target-point:
- 보정된 경로를 따라 최종 주차 구역(Target Point)으로 정밀하게 진입합니다.

6) Drop a car at AMR:
- 차량을 하차(Drop)하는 가상 동작을 수행하고 주차를 완료합니다.

7) Move to way-point (Exit):
- 주차 구역을 안전하게 빠져나오기 위해 출차 경유지로 이동합니다.

8) If don't have a task (Decision):
- More task: 대기 중인 태스크가 있다면 Move to start point로 돌아가 반복 수행합니다.
- No more task: Dock 명령을 실행하여 충전 스테이션으로 복귀합니다.

5.2.5 nav_to_pose_2.py (자율 주행 및 작업 관리 노드)

Nav2 스택과 연동하여 로봇을 목표 지점까지 이동시키는 액션 클라이언트(Action Client) 역할을 수행하며, 주행 완료 후 튜닝 모드 진입 등 후속 작업을 관리합니다.

• 역할 (Role):
- 주행 명령 실행: Nav2 액션을 호출하여 로봇을 특정 웨이포인트로 이동시킵니다.
- 작업 상태 관리: 로봇의 작업 가능 여부(ready_for_task)를 관리하고 작업 완료 시 시스템에 알립니다.
- 튜닝 모드 전환: 주행 완료 후 정밀 제어가 필요한 경우 튜닝 모드로 진입합니다.
- 주요 함수 (Key Functions):
- task_callback(): 새로운 작업 메시지를 수신하면 호출되며, 상태 플래그(ready_for_task)를 갱신합니다.
- mark_task_assigned(): 특정 작업 ID를 식별하여 해당 작업의 상태를 assigned(배정됨)로 변경하고 확정합니다.
- move_with_pre_pose(): 로봇을 지정된 웨이포인트 좌표로 물리적으로 이동시키는 주행 명령을 수행합니다.
- exit_ready_callback(): 시스템이 새로운 작업을 할당받을 수 있는 상태로 복귀하고 가용성을 활성화합니다.

• 동작 흐름 (Operational Flow):
1) Task Reception: task_command_robot: 토픽을 통해 작업 명령을 수신합니다.
2) Availability Check: ready_for_task가 True인지 확인합니다. False라면 명령을 무시합니다.
3) Task Parsing: 작업 데이터를 파싱하여 목표 웨이포인트 좌표를 저장합니다.
4) Execute Movement: move_with_pre_pose()를 호출하여 로봇을 이동시킵니다.
5) Arrival Check: 목표 지점에 도달하면 Start tuning mode로 진입합니다.
6) Wait for Completion: /robot5/exit_ready 토픽을 수신할 때까지 대기합니다.
7) Reset Status: exit_ready_callback()을 실행하여 로봇을 다시 작업 가능 상태로 전환하고 /robot5/waypoint_done 토픽을 발행합니다.

5.2.6 main_control_2.py (정밀 주차 및 라인 트레이싱 제어 노드)

주차 구역 진입 시 카메라 센서를 활용하여 바닥의 라인을 인식하고, 로봇을 정밀하게 정렬하는 비전 기반 제어(Vision-based Control) 노드입니다.

• 역할 (Role):
- 라인 트레이싱: 카메라 영상에서 주차 라인을 검출하고 로봇의 중심을 라인에 맞춥니다.
- 정밀 이동: 계산된 오차 값을 바탕으로 미세 조정 주행을 수행합니다.
- 작업 완료 신호: 정렬이 완료되면 튜닝 완료 신호를 발행합니다.

• 주요 함수 (Key Functions):
- image_callback():
1) ROI 설정: 전체 프레임 중 바닥 라인이 위치하는 하단 1/3 영역만 연산 범위로 설정합니다.
2) 색상 필터링: HSV 색상 공간에서 흰색(White) 범위를 추출하여 마스크를 생성합니다.
3) 윤곽선 검출: 이진 이미지에서 컨투어(Contour)를 추출하여 라인의 중심점을 획득합니다.

• 동작 흐름 (Operational Flow):
1) Trigger: waypoint_done 토픽을 구독하면 정밀 제어 모드가 시작됩니다.
2) Line Tracing: Perform line tracing 로직을 수행하여 로봇을 정렬합니다.
3) Completion: 정렬이 완료되면 tuning_done 토픽을 발행하여 다음 단계(출차 시퀀스 등)로 넘어갑니다.

5.2.7 exit_controller.py (출차 시퀀스 통합 관리 노드)
출차 프로세스 전반을 총괄하며, 데이터베이스 상태 업데이트, 로봇 이동, 도킹 등 일련의 과정을 순차적으로 제어하는 시퀀스 매니저(Sequence Manager)입니다.

• 역할 (Role):
- 출차 시퀀스 제어: 튜닝 완료 신호를 받은 시점부터 로봇이 주차 구역을 빠져나와 최종 목적지까지 이동하는 흐름을 관리합니다.
- 상태 동기화: 출차 완료 시 DB의 차량 상태를 업데이트하고 시스템을 대기 모드로 전환합니다.
- 도킹 복귀: 작업이 없을 경우 로봇을 충전 스테이션으로 복귀시킵니다.

• 주요 함수 (Key Functions):
- start_exit_sequence(): 출차 프로세스의 시작점으로, 상태 플래그를 busy로 설정하고 목표 지점으로 이동 명령을 내립니다.
- _go_to_pose(): Nav2 액션을 통해 이동 명령을 수행하고 도착 상태를 지속적으로 점검합니다.
- cb_tuning_done(): main_control_2로부터 튜닝 완료 신호를 수신하면 즉시 출차 로직을 가동합니다.
- _on_exit_finished(): 출차 완료 후 후속 작업을 결정합니다. 새 작업이 있으면 중간 대기 지점으로, 없으면 도킹 장소로 이동합니다.

• 동작 흐름 (Operational Flow):
1) Start: tuning_done 토픽을 수신하면 start_coords(출차 시작점)로 이동합니다.
2) Action: 알람음을 울리고(Alarm beep sound) 픽업 완료 동작을 수행합니다.
3) Rotation: 180도 회전(Turn 180)하여 출차 방향을 잡습니다.
4) Navigation: start way-point -> middle way-point -> exit target point 순서로 경유지를

거쳐 이동합니다.

5) Completion: exit_done 토픽을 발행하고 DB의 차량 상태를 Car-out completion으로 업데이트합니다.

6) Next Step: 대기 중인 작업(pending task) 유무를 확인합니다.

- Yes: 중간 대기 지점(middle way-point)으로 이동 후 exit_ready를 발행하여 새 작업을 수행합니다.
- No: Dock 명령을 실행하여 충전 스테이션으로 복귀합니다.

5.3 Detection Pipeline (객체 인식 데이터 흐름)

카메라 센서부터 제어 명령 생성까지의 데이터 파이프라인 구조입니다.

5.3.1 Image Acquisition

- WebCam에서 Raw Image를 캡처합니다.
- WebCam: 입차 구역에 설치된 USB 카메라로부터 실시간 영상 스트림을 수신합니다.

5.3.2 Preprocessing (enter_process)

- 압축 해제 및 변환: 수신된 CompressedImage 형식의 이미지를 OpenCV 포맷(cv::Mat)으로 디코딩합니다.
- 리사이징 및 정규화: YOLO 모델의 입력 크기에 맞춰 이미지를 640x640 픽셀로 리사이징하고, 픽셀 값을 0~1 사이로 정규화(Normalization)하여 추론 성능을 최적화합니다.

5.3.3 Inference (enter_process)

- 객체 감지 (Object Detection): 전처리된 이미지를 YOLOv8 모델에 입력하여 추론을 수행합니다.
- 정보 추출: 감지된 객체의 Bounding Box (x, y, w, h), Class ID (차량 종류), Confidence Score를 추출합니다.
- 결과 발행: 추출된 정보를 좌표 변환 및 추적 노드에서 활용할 수 있도록 합니다. 디버그를 위해 /yolo/debug_image 를 발행해 rqt에서 확인할 수 있도록 한다.

5.4 모니터링 시스템 상세 (Monitoring & Telemetry)

• 대시보드 동기화 (Web_Socket_Bridge):

- ROS 2의 /robot_states (위치, 배터리) 토픽을 JSON으로 직렬화하여 웹 클라이언트로 10분 주기로 파싱.

—

6. 보안 설계 (Security Design)

본 시스템은 개방형 네트워크 환경에서도 데이터의 기밀성(Confidentiality), 무결성(Integrity), 가용성(Availability)을 보장하기 위해 네트워크 전송, 접근 제어, 데이터 저장 전반에 걸쳐 강력한 보안 아키텍처를 적용합니다.

6.1 통신 및 네트워크 보안 (Communication Security)

AMR, 중앙 관제 서버, 클라우드 브로커 간의 모든 통신은 암호화된 채널을 통해 이루어지며,

중간자 공격(MITM) 및 도청을 방지합니다.

- 전송 구간 암호화 (TLS 1.3):
  - AMR과 PC(서버) 간 전송되는 모든 제어 명령 및 상태 데이터는 TLS 1.3 (Transport Layer Security) 프로토콜을 사용하여 암호화 전송됩니다.

6.2 인증 및 접근 제어 (Authentication & Access Control)
모든 디바이스와 사용자는 엄격한 인증 절차를 거쳐야 하며, 권한에 맞는 리소스에만 접근할 수 있습니다.

- 기기 및 사용자 인증:
  - Username / Password: 정해진 인증 정보를 사용하여 접근을 통제합니다.
  - Client ID 식별: 각 로봇(AMR), 제어 PC, 관리자 앱에 고유한 'Client ID'를 부여하여 연결 주체를 식별하고, 중복 연결이나 비인가 기기의 접속을 차단합니다.

- 권한 관리 (Authorization):
  - Topic 구조화: /robot/{id}/status, /robot/{id}/cmd_vel과 같이 계층적이고 명확한 토픽 구조를 설계하여, 주제별 권한 분리를 명확히 하고 오용을 방지합니다.

6.3 데이터 보호 및 설정 관리 (Data Protection)
- 저장 데이터 암호화:
  - PC 및 서버에 저장되는 시스템 로그, 비상 경고 이력, 차량 정보 등의 민감 데이터는 암호화되어 저장됩니다.

- 보안 설정 파일 관리:
  - username, API Key 등 민감한 인증 정보는 소스 코드에 하드코딩하지 않습니다.
  - 대신, 외부의 .env 파일이나 암호화된 설정 파일(Config)을 통해 로드하는 방식을 적용하여, 코드 유출 시에도 보안 키가 노출되지 않도록 합니다.

6.4 보안 모니터링 및 감사 (Audit & Monitoring)
- 실시간 로그 감사:
  - 서버의 접속 로그와 메시지 트래픽을 실시간으로 기록하고 모니터링합니다.

—

7. 성능 요구사항 (Peformance Requirements)

본 시스템은 상용 서비스 수준의 안정성과 반응성을 확보하기 위해 다음의 구체적인 성능 지표를 준수해야 합니다. 모든 지표는 정량적으로 측정 가능해야 하며, 테스트 및 검증 단계의 기준이 됩니다.

7.1 시스템 가용성 및 확장성 (Availability & Scalability)
- 시스템 가동률 (Uptime):
  - 전체 시스템(서버 및 AMR 군집)은 데모 시연 및 테스트 기간 중 99.9% 이상의 가동률을 유지해야 합니다.

- AMR은 완충 시 최소 1시간 이상 연속 운영(Continuous Operation)이 가능해야 한다.

• 유연한 확장성 (Scalability):
- 시스템은 기본적으로 2대의 PC(서버/관제)와 2대의 AMR 구조로 설계되었으나, 향후 로봇 대수 증가 시 서버의 코드 수정 없이 노드 확장이 가능한 Namespace 기반 분산 아키텍처를 지원해야 합니다.
- 단일 관제 서버는 최대 4대의 AMR을 성능 저하(Latency 증가) 없이 동시에 제어할 수 있어야 합니다.

7.2 객체 인식 및 탐지 성능 (Perception Performance)
• 객체 감지 신뢰도 (Detection Confidence):
- RGB 카메라 기반: 입차 구역 및 로봇에 탑재된 카메라는 YOLOv8n 모델을 통해 차량(대형, 중형, 소형) 및 사람, 장애물을 감지해야 하며, 최소 0.75 (75%) 이상의 신뢰도(Confidence Score)를 유지해야 유효한 객체로 인정합니다.
- 오인식률 최소화: 주차장 내 구조물(기둥, 벽)을 차량으로 오인식하는 비율(False Positive)은 5% 미만이어야 합니다.

• 거리 측정 정밀도 (Depth Estimation):
- OAK-D Pro: 스테레오 비전(Stereo Vision) 기반의 Depth 센서는 전방 0.5m ~ 3.0m 범위 내의 장애물 및 대상 차량과의 거리를 ±0.3m 이내의 오차로 측정해야 합니다.

• 사용자/차량 인식 속도:
- 입차 구역에 진입한 차량은 정지 후 3초 이내에 차종 분류 및 ID 생성이 완료되어야 합니다. AMR은 전방의 장애물이나 차량을 2초 이내에 인식하고 회피 또는 정지 판단을 내려야 합니다.

7.3 자율 주행 및 제어 정밀도 (Navigation & Control Precision)
• Waypoint 도달 정밀도:
- AMR이 서버로부터 할당받은 목표 지점(Goal Pose)에 도착했을 때, 허용되는 위치 오차(Position Tolerance)는±0.2m (20cm) 이내, 방향 오차(Yaw Tolerance)는 ±0.1 rad (약 5.7도) 이내여야 합니다.

• Localization 안정성:
- AMCL: AMR은 초기 위치(Initial Pose) 설정 이후 주행 중 지속적으로 자신의 위치를 추정해야 하며, SLAM으로 생성된 정적 지도(Static Map)와 실제 LiDAR 스캔 데이터 간의 매칭 점수가 일정 수준 이상 유지되어야 합니다. (위치 소실 발생 시 5초 이내 복구)

• 주행 및 회피 성능:
- 경로 생성: Nav2 Planner는 목적지까지의 전역 경로를 1초 이내에 생성해야 합니다.
- 동적 회피: 주행 경로 상에 갑작스러운 장애물(사람, 타 로봇)이 나타날 경우, 1.5초 이내에 감지하여 정지하거나 우회 경로를 생성해야 합니다.
- 이동 속도: 안전을 위해 공차 시 최대 1.5m/s, 차량 운반 시뮬레이션 시 최대 0.8m/s의 속도 제한을 준수해야 합니다.

7.4 협동 제어 및 시나리오 반응성 (Cooperative Control Response)

- 협동 모드 전환 시간:
  - 출차 경로 상의 이중 주차가 감지되면, 서버는 1초 이내에 이를 인지하고 협동 제어 를 트리거해야 합니다.
- 시퀀스 동기화:
  - 협동 로봇(Obstacle 담당)과 주 로봇(Target 담당) 간의 작업 인계(Handover) 지연 시간은 3초 미만이어야 합니다. (예: 장애물 제거 완료 신호 후 주 로봇 출발까지의 시간)
- 원상 복구 정확도:
  - 장애물 차량을 임시 구역으로 이동시켰다가 다시 원래 위치로 복귀시킬 때, 초기 위치와의 오차는 ±0.3m 이내여야 합니다.

7.5 인터페이스 및 통신 성능 (Interface & Communication)

- UI 반응 속도:
  - 사용자 앱 및 관리자 대시보드의 버튼 입력(입차/출차/비상정지)에 대한 시스템 반응(ACK)은 1초 이내에 화면에 표시되어야 합니다.
- 알림 전송 지연:
  - 입차 완료, 결제 요청, 출차 완료 등의 중요 이벤트 알림은 발생 시점으로부터 1초 이내에 사용자 단말기로 전송되어야 합니다.

—

8. 오류 처리 및 복구 (Error Handling and Recovery)

8.1 네트워크 및 통신 장애 (Network Failure)

- 네트워크 손실 감지: AMR은 중앙 관제 서버와의 연결 상태를 1초 주기의 Heartbeat 메시지로 확인합니다. 5초 이상 응답이 없을 경우 '통신 두절(Disconnected)' 상태로 간주합니다.
- Fail-Safe 동작: 통신이 끊긴 즉시 AMR은 안전을 위해 제자리 정지(Stop)하고, 현재 수행 중인 Navigation Action을 취소합니다.

8.2 하드웨어 및 시스템 오류 (Hardware Failure)

- 상태 모니터링: AMR 내부의 Diagnostics Node는 모터 드라이버, 배터리 전압, 센서 데이터(LiDAR, Camera)의 상태를 실시간 점검합니다.

8.3 주행 및 위치 인식 오류 (Navigation & Localization Failure)

- 위치 추정 실패 (Localization Lost): AMCL의 추정 신뢰도(Covariance)가 임계값 이하로 떨어질 경우, AMR은 제자리 회전(Recovery Behavior)을 통해 주변 지형지물(Lidar Scan)을 재스캔하여 위치를 다시 찾습니다. 실패가 지속되면 관리자에게 '위치 소실' 알림을 보냅니다.
- 경로 차단 및 고립: 동적 장애물로 인해 경로가 막히면 DWB 컨트롤러가 우회 경로를 생성합니다. 모든 경로가 막혀 이동이 불가능할 경우(Stuck), 서버에 '이동 불가' 상태를 보고하고 대기 모드로 전환합니다.

• 원격 구난 (Rescue): 소프트웨어적으로 해결 불가능한 고립 상태나 경로 이탈 시, 관리자가 웹 대시보드의 '가상 조이스틱(Virtual Joystick)'을 활성화하여 수동으로 로봇을 안전 구역으로 이동시킬 수 있어야 합니다.

—

9. 테스트 및 검증 (Testing and Validation)

시스템의 완성도를 보증하기 위해 단위 테스트(Unit Test), 통합 테스트(Integration Test), 시나리오 테스트(Scenario Test), 현장 테스트(Field Test)의 4단계 검증 프로세스를 수행합니다.

9.1 단위 테스트 (Unit Testing)

각 소프트웨어 모듈이 개별적으로 정상 동작하는지 검증합니다.

• 객체 인식 (YOLO):
- 방법: 다양한 각도/조명에서 촬영된 차량 이미지 100장을 입력으로 주입.
- 기준: 차량(Car, SUV) 탐지 성공률 75% 이상, 오탐지(False Positive) 5건 미만.

• 주차/출차 라인 디텍션 (Line Detection):
- 방법: OAK-D 카메라에 주차 구역의 'T자형' 라인과 출차 유도선 이미지를 입력.
- 기준: 라인의 색상(흰색/노란색) 및 엣지(Edge)를 정확히 추출하고, 로봇 중심과의 편차(Lateral Error)를 ±2cm 이내로 계산하는지 확인.

• AMR 주행 (Nav2):
- 방법: Rviz에서 특정 좌표(x, y)로 이동 명령 전송.
- 기준: 목표 지점 도착 오차 ±20cm 이내, 이동 중 장애물 회피 성공 여부.

• 위치 추정 (Localization):
- 방법: 로봇을 수동으로 주행시키며 AMCL 추정 위치와 실제 위치(Ground Truth) 비교.
- 기준: 회전 및 직진 주행 시 위치 추정 오차가 맵 상에서 발산하지 않고 유지되는지 확인.

9.2 통합 테스트 (Integration Testing)

서버와 로봇 간의 데이터 흐름과 인터페이스 연동을 검증합니다.

• 통신 안정성:
- 방법: Wi-Fi 환경에서 서버와 AMR 간 1시간 동안 연속 데이터 패킷 송수신.
- 기준: 패킷 손실률 1% 미만, 평균 지연 시간(Latency) 200ms 미만.

• DB 트랜잭션:
- 방법: 입차 -> 데이터 생성 -> DB 저장 -> 조회 -> 출차 -> 상태 업데이트 프로세스 반복.
- 기준: 데이터 누락 없이 모든 트랜잭션이 정확히 기록되고 조회되어야 함.

• 관제 시스템 연동:
- 방법: AMR의 상태 변화(배터리, 작업상태)가 실시간으로 반영되는지 확인.

9.3 시나리오 테스트 (Scenario Test)

실제 운영 환경을 모사하여 핵심 비즈니스 로직을 검증합니다.

• 입차 및 자동 주차 (Auto Parking):
  - 시나리오: 차량 진입 -> 인식(ID생성) -> 로봇 배정 -> 이동 -> [라인 인식 정렬] -> 주차 -> 복귀.
  - 검증: 로봇이 차량을 정확히 인식하고, 주차 라인을 따라 정밀하게 진입하여 주차하는지 확인.

• 단일 차량 출차 (Single Exit):
  - 시나리오: 사용자 앱 출차 요청(장애물 없음) -> 로봇 이동 -> 차량 리프팅(가상) -> [출차 유도선 추종] -> 출차 구역 하차 -> 복귀.
  - 검증: 장애물이 없는 상황에서 최단 경로로 신속하게 출차를 완료하는지, 출차 구역 정차 위치가 정확한지 확인.

• 이중 주차 협동 제어 (Cooperative Control):
  - 시나리오: 출차 요청 -> 장애물 감지 -> AMR 2대 호출 -> [회피 -> 출차 -> 원복] 시퀀스 수행.
  - 검증:
    1) 두 로봇이 충돌 없이 교차 이동하는지.
    2) 작업 인계(Handover) 타이밍이 5초 이내로 매끄러운지.
    3) 장애물 차량이 원래 위치로 정확히 복구되는지.

9.4 현장 테스트 및 시연 (Field Testing & Demo)
실제 구축된 데모 환경에서 최종 리허설을 수행합니다.

• 환경: 4.75*3.45m 이상의 평탄한 공간, 실제 조명 조건.

• 항목:
  1) 10회 연속 입출차(단일/협동 혼합) 시나리오 수행 시 성공률 90% 이상.
  2) 비상 정지 버튼 누름 시 1.5초 이내 즉시 정지 확인.

—

10. 배포 및 유지보수 계획 (Deployment and Maintenance Plan)

10.1 배포 단계 (Deployment Phase)
시스템을 실제 운영 환경에 설치하고 초기화하는 절차입니다.

• 인프라 및 네트워크 구축:
  - PC 1 (서버): Ubuntu 22.04 LTS 및 ROS 2 Humble 전체 패키지(ros-humble-desktop-full)를 설치하고, 고정 IP(Static IP)를 할당하여 네트워크의 중심 노드로 설정합니다.
  - PC 2 (모니터링): ROS 2 및 웹 브라우저 환경을 구성하고, 서버와 동일한 ROS_DOMAIN_ID를 설정하여 통신을 연결합니다.
  - AMR (로봇): TurtleBot4 2대의 펌웨어를 최신 버전으로 업데이트하고, Wi-Fi 6 AP에 연결합니다. 각 로봇에 고유한 NAMESPACE (예: /tb4_1, /tb4_2)를 부여하여 식별 충돌을 방지합니다.

• 시스템 초기화 (Bring-up):

- 맵핑 (Mapping): AMR 1대를 수동 조작하여 주차장 전체를 SLAM으로 스캔하고, 생성된 정적 지도(parking_lot.pgm)를 서버의 Nav2 설정 경로에 저장합니다.
- 좌표 설정: 맵 상에서 주차 구역(Zone A, B...), 입차 구역, 충전 스테이션의 좌표(x, y, yaw)를 측정하여 DB 및 설정 파일(params.yaml)에 등록합니다.

• 자동화 스크립트:
- 시스템 부팅 시 로봇 제어 노드와 웹 서버가 자동으로 실행되도록 system.md 서비스 또는 startup.sh 스크립트를 등록합니다.

10.2 유지보수 일정 (Maintenance Schedule)
시스템의 안정성과 성능을 지속적으로 유지하기 위한 주기별 점검 계획입니다.

• 월간 점검 (Monthly Maintenance):
- 소프트웨어 업데이트: Ubuntu OS 보안 패치 및 ROS 2 패키지 업데이트를 수행하여 보안 취약점을 개선합니다.
- 로그 분석: 지난 한 달간 서버에 저장된 에러 로그(syslog, rosout)를 분석하여 반복되는 오류 패턴(통신 지연, 경로 생성 실패 등)을 식별하고 버그를 수정합니다.
- AI 모델 재학습: 운영 중 수집된 오인식(False Positive) 이미지를 선별하여 라벨링을 보완하고, YOLO 모델을 재학습(Fine-tuning)하여 인식 정확도를 향상시킵니다.

• 분기별 점검 (Quarterly Maintenance):
- 하드웨어 점검: AMR의 타이어 마모도, 모터 구동 소음, LiDAR 센서 렌즈 오염 상태를 육안으로 점검하고 청소합니다.
- 배터리 검사: 배터리 완충 후 방전 테스트를 수행하여 수명(Health)을 체크하고, 효율이 80% 이하로 떨어진 배터리는 교체합니다.
- 센서 보정 (Calibration): 주행 중 발생하는 오차(Drift)를 최소화하기 위해 OAK-D 카메라의 캘리브레이션과 휠 오도메트리 보정 작업을 수행합니다.

• 문서화 (Documentation):
- 프로젝트 종료 후 또는 주요 업데이트 시마다 사용자 가이드(User Guide)와 버그 리포트(Bug Report)를 최신화하여 관리합니다.

—

11. 부록 (Appendix)

11.1 라이브러리 및 종속성 (Libraries & Dependencies)
본 시스템 구축에 사용된 주요 소프트웨어 라이브러리와 버전 정보는 다음과 같습니다.

• Core Framework:
- ROS 2 Humble Hawksbill: 로봇 제어 미들웨어 및 통신 백본.
- Python 3.10+: 서버 로직 및 AI 추론 엔진 주 언어.
- C++ (Standard 17): 고성능 센서 드라이버 및 Nav2 플러그인 개발 언어.

• AI & Computer Vision:

- Ultralytics YOLOv8: 차량 및 장애물 실시간 탐지.
- OpenCV 4.x: 이미지 전처리, 변환, 주차 라인 추출.
- cv_bridge: ROS 이미지 메시지와 OpenCV 포맷 간 변환.
- PyTorch: 딥러닝 모델 학습 및 추론 가속.

• Navigation & Control:
- Nav2 Stack: 경로 계획(Planner), 주행 제어(Controller), 행동 트리(BT).
- SLAM Toolbox: 정적 지도 작성 및 저장.
- AMCL: 2D LiDAR 기반 실시간 위치 추정.

• Web & Database:
- FastAPI (or Flask): 비동기 REST API 서버 및 WebSocket 처리.
- PostgreSQL 14: 관계형 데이터베이스 관리 시스템.
- SQLAlchemy: ORM(Object Relational Mapping) 기반 DB 접근.
- React.js: 관리자 대시보드 프론트엔드 프레임워크.

• Drivers:
- turtlebot4_robot: TurtleBot4 제어 및 상태 모니터링 패키지.
- depthai_ros: OAK-D Pro 카메라 드라이버 및 ROS 2 인터페이스.
- rplidar_ros: RPLIDAR A1 레이저 스캐너 드라이버.

11.2 용어 사전 (Glossary)
• AMR (Autonomous Mobile Robot): 자율 이동 로봇. 주변 환경을 인식하여 목적지까지 스스로 이동하는 로봇.
• AVG (Auto Valet Garage): 본 프로젝트의 명칭. 무인 자동 주차 관제 시스템.
• DDS (Data Distribution Service): ROS 2에서 사용하는 실시간 데이터 분산 서비스 미들웨어 표준.
• FMS (Fleet Management System): 다수의 로봇(군집)을 통합 관리하고 작업을 할당하는 관제 시스템.
• SLAM (Simultaneous Localization and Mapping): 로봇이 미지의 환경에서 지도를 작성하면서 동시에 자신의 위치를 파악하는 기술.
• YOLO (You Only Look Once): 이미지 내의 객체를 실시간으로 탐지하고 분류하는 딥러닝 알고리즘.
• Costmap: 로봇 주행 시 장애물 정보를 비용(Cost)으로 표현하여 경로 계획에 사용하는 격자 지도.

11.3 참고 문헌 (References)
• ROS 2 Documentation: https://docs.ros.org/en/humble/
• Navigation 2 (Nav2) Documentation: https://www.google.com/search?q=https://navigation.ros.org/
• TurtleBot4 User Manual: https://turtlebot.github.io/turtlebot4-user-manual/
• Ultralytics YOLOv8 Docs: https://docs.ultralytics.com/
• Luxonis OAK-D DepthAI Docs: https://docs.luxonis.com/