종합설계 프로젝트 보고서

QR코드 기반 목적지 지정 자율이송 분류 플랫폼

2026. 05. 27.

한성대학교

전자 트랙

개인정보(ID) 이호성

개인정보(ID) 김은중

개인정보(ID) 강성민

개인정보(ID) 임재엽

개인정보(ID) 안휘훈

종합설계 프로젝트 보고서

QR코드 기반 목적지 지정 자율이송 분류 플랫폼

작 성 자 : 개인정보(ID) 이호성

개인정보(ID) 김은중

개인정보(ID) 강성민

개인정보(ID) 임재엽

개인정보(ID) 안휘훈

지도교수 : 오 종 택

차 례

Ⅰ. 프로젝트 개요

1. 개발 배경 및 필요성

최근 물류 및 제조 산업에서는 다품종 소량 생산과 전자상거래의 급증으로 인해 창고 내 물류 이동의 유연성과 효율성이 극대화되어야 하는 요구가 커지고 있다. 고전적인 AGV(Automated Guided Vehicle)는 바닥에 부착된 마그네틱 테이프나 레일을 따라 이동하는 방식으로, 동적인 장애물이 발생하거나 레이아웃이 변경될 경우 대응이 어렵다는 한계가 있다.

이를 해결하기 위해 인프라(관제 카메라)와 로봇 간의 실시간 통신을 기반으로 유동적인 경로 탐색이 가능하고, 정밀한 위치 보정을 통해 자율적으로 물품을 운송할 수 있는 스마트 물류 운송 로봇 관제 시스템의 개발이 필요하다.

2. 개발 목표

본 프로젝트의 최종 목표는 '인프라 카메라 기반의 반실시간 장애물 회피 및 QR 코드/IMU 센서 융합을 통한 고정밀 자율 주행 물류 운송 로봇 시스템'을 개발하는 것이다. 이를 달성하기 위한 구체적인 목표는 다음과 같다.

- 다방향 주행 메커니즘: 메카넘 휠(Mecanum Wheel)과 DRI0039 모터 드라이버를 활용하여 좁은 창고 내에서도 전후좌우 및 대각선 이동(보정용)이 가능한 하드웨어 플랫폼 구축.
- 자이로 기반 자세 보정 및 측위: MPU-9250 센서로부터 획득한 자이로 데이터를 활용하여 주행 중 발생하는 방향 오차를 비례 제어(P-Control) 로직으로 실시간 보정하고, 바닥의 QR 코드를 통해 절대 좌표를 확립하여 ±10mm 이내의 정밀도를 확보.
- 이벤트 기반 장애물 회피: 높은 위치에 설치된 인프라 카메라로 작업 공간(Grid) 내의 장애물을 인식하고, MQTT 통신을 통해 로봇에게 전달하며, 특정 이벤트 시점에 갱신하여 충돌을 방지하고 최적의 경로를 재탐색.
- 통합 관제 웹 서버 구축: 사용자가 다수의 물류 주문(Queue)을 입력하고 실시간으로 로봇의 위치, 상태, 장애물 현황을 모니터링 및 제어(비상 정지 등)할 수 있는 웹 기반(Flask) 관제 시스템 개발.

3. 개발 범위 및 주요 특징

본 프로젝트는 단일 로봇의 제어를 넘어, 시스템 간의 유기적인 연동을 포함하는 3-Tier 아키텍처로 구성된다.

- 인프라 감시 모듈 (Raspberry Pi A): Background Subtraction(MOG2) 알고리즘을 이용해 특정 관심 영역(ROI) 내의 장애물을 감지하고, MQTT 토픽(test/json)으로 지속 발행한다.
- 관제 및 비전 모듈 (Raspberry Pi B - 로봇 탑재): 웹 UI를 통해 사용자의 주문 명령을 수신하고 대기열을 관리한다. 동시에 바닥의 QR 코드를 스캔해 얻은 위치 및 오차 데이터를 UART 통신으로 하위 제어기에 전달한다.
- 구동 및 주행 제어 모듈 (ESP32 - 로봇 탑재): 상위 모듈에서 전달받은 명령(ORD), 장애물 정보(BLOCK), 좌표 정보(POS)를 종합하여 비용(Cost) 기반 라우팅 알고리즘으로 최단 거리를 산출하고, 4개의 DC 모터와 하역용 서보 모터를 정밀 제어한다.

4. 기대 효과

- 경제성 및 확장성 향상: 고가의 라이다(LiDAR) 센서 대신 범용 카메라(인프라 측면)와 저비용 고효율 마이크로컨트롤러(ESP32, Raspberry Pi)를 결합하여 시스템 구축 비용을 절감하였으며, 추후 다수 로봇(Multi-Robot) 도입 시 인프라 확장 비용을 최소화할 수 있다.
- 산업 안전성 확보: 물리적 버튼을 통한 적재 확인 로직과 주행 중 경로가 막혔을 때 강제로 전진하지 않고 대기하며 경로를 재탐색(WAIT_PATH_RECOVERY)하는 페일세이프(Fail-Safe) 기능, 그리고 웹을 통한 긴급 정지(CMD,STOP) 기능을 도입하여 산업 현장에서 요구되는 높은 수준의 안정성을 확보하였다.

II. 시스템 구성

본 시스템은 효율적인 물류 운송 및 실시간 관제를 수행하기 위하여 인프라 인식 계층, 중앙 관제 계층, 물리 구동 계층의 3단계로 구분된 계층적 아키텍처를 채택한다. 각 계층은 독립적인 연산 장치를 보유하며, 상호 간의 데이터 정합성을 유지하기 위해 네트워크 프로토콜 및 고속 직렬 통신을 활용한다. 최상위 인프라 계층은 작업 환경 전반을 감시하며, 중앙 관제 계층은 주문 관리 및 비전 인식을 수행하고, 최하위 구동 계층은 실제 하드웨어의 움직임을 정밀하게 제어하는 구조이다.

시스템을 구성하는 핵심 하드웨어는 두 대의 Raspberry Pi와 한 대의 ESP32 마이크로컨트롤러로 구성된다. 작업 영역 상단에 설치된 인프라 인식용 Raspberry Pi는 관제 카메라를 통해 동적 장애물을 판단하며, 로봇에 탑재된 메인 Raspberry Pi는 하향 카메라를 이용한 QR 코드 인식 및 Flask 기반 웹 서버 구동을 담당한다. 하위 제어기인 ESP32는 DRI0039(Dual Digital DC Motor Driver)를 통해 4개의 메카넘 휠을 독립적으로 구동하며, MPU-9250 IMU 센서를 활용하여 주행 중 발생하는 자세 오차를 실시간으로 연산한다. 특히 메카넘 휠의 구동 특성을 고려하여 전후좌우 이동을 주행의 기본으로 설정하되, 대각선 이동은 위치 정밀 보정 단계에서만 제한적으로 활용되도록 설계하여 주행의 안정성을 확보한다.

소프트웨어 및 통신 환경은 시스템의 안정성과 실시간성을 최우선으로 고려하여 설계되었다. 모듈 간 데이터 교환을 위해 인프라 모듈과 관제 서버 간에는 MQTT 프로토콜을, 관제 서버와 하위 제어기 간에는 115200bps 속도의 UART 직렬 통신을 구축하였다. 통신 과부하 및 주행 중 데이터 충돌에 의한 제어 지연을 방지하기 위해, 장애물 맵 업데이트는 주행 중 상시 이루어지지 않고 특정 이벤트(주문 접수, 적재 완료, 하역 완료 등) 발생 시에만 일시적으로 업데이트 락(Lock)을 해제하여 동기화하도록 설계하였다. 전체 시스템의 개발은 VS Code 환경에서 진행되었으며, 원격 디버깅 및 실시간 모니터링을 위해 RealVNC가 활용된다. 또한, 로봇의 주행 및 하역 작업을 체계화하기 위하여 서보 모터를 이용한 적재함 제어와 물리 버튼(GPIO 17)을 활용한 사용자 승인 로직이 하드웨어 계층에 통합되어 있다.

운용 공간은 논리적인 격자형 맵(Grid Map)으로 정의되며, 전체 작업 영역은 4x3 구조의 좌표계로 분할 관리된다. 로봇의 초기 대기 및 복귀 지점은 (3, 1) 좌표로 설정되며, 물류 적재는 (3, 0) 및 (3, 2) 구역에서, 물류 하역은 (0, 0), (0, 1), (0, 2) 구역에서 각각 수행되도록 정의된다. 각 좌표 지점에는 QR 코드가 배치되어 로봇이 자신의 절대 위치를 파악할 수 있는 지표로 활용되며, 갱신 시점에 수신된

장애물 정보에 따라 해당 좌표를 동적으로 폐쇄 처리함으로써 경로 탐색 시 가변적인 환경에 대응하도록 설계하였다.

III. 각 모듈별 동작 원리

인프라 감시 모듈은 작업 환경 내 동적 장애물을 실시간으로 감지하기 위하여 OpenCV 라이브러리의 BackgroundSubtractorMOG2 알고리즘을 핵심적으로 활용한다. 해당 모듈은 미리 정의된 각 그리드 셀(Cell) 영역에 대하여 초기 배경 학습을 수행하며, 학습된 배경 데이터와 현재 입력 영상 간의 차이를 분석하여 이질적인 물체를 식별한다. 감지된 물체는 윤곽선(Contour) 추출 및 면적 계산 과정을 거치며, 설정된 임계치 이상의 면적을 가진 물체만을 유효한 장애물로 판단함으로써 센서 노이즈를 억제한다. 최종적으로 식별된 장애물의 격자 좌표는 MQTT 프로토콜을 통해 0.5초 주기로 발행되어 전체 시스템 맵의 동적 갱신을 가능하게 한다.

중앙 관제 및 비전 모듈은 시스템의 두뇌 역할을 수행하며, Flask 프레임워크를 기반으로 웹 인터페이스 제공과 비전 처리를 동시에 수행한다. 로봇 하단에 장착된 카메라를 통해 바닥의 QR 코드를 스캔하며, 인식된 데이터로부터 로봇의 절대 좌표를 획득한다. 이때 단순히 좌표값만을 읽는 것에 그치지 않고, 영상 프레임 내 QR 코드의 중심점 좌표와 회전 각도를 계산하여 픽셀 단위의 미세 오차(errX, errY, angle)를 산출한다. 산출된 정밀 오차 데이터와 웹 UI로부터 입력된 주문 정보는 UART 직렬 통신을 통해 하위 구동 제어기로 실시간 전달되어 주행의 정확도를 높이는 기초 자료로 활용된다. 특히 인프라로부터 수신한 외부 장애물 정보는 내부 변수에 캐싱(Caching)해 두었다가, 주행 시작 전이나 주요 거점 정차 시 등 데이터 동기화가 필요한 특정 시점에만 ESP32로 일괄 전송하여 통신 효율을 극대화한다.

물리 구동 제어 모듈은 상위 계층에서 전달받은 데이터를 물리적인 움직임으로 변환하는 최종 실행 단계이다. ESP32 프로세서 내부에 설계된 상태 머신(State Machine)은 로봇의 현재 상태를 유기적으로 전환하며 주행 로직을 관리한다. 로봇의 직진성을 유지하기 위하여 MPU-9250 센서의 자이로 Yaw 값을 실시간으로 모니터링하며, 설정된 목표 각도와 현재 각도의 편차에 비례 상수(Kp)를 적용하는 P-제어 알고리즘을 수행한다. 이 보정 알고리즘은 주행 중 로봇이 좌우로 편향되는 것을 방지하기 위해 각 모터의 PWM 출력을 실시간으로 가감한다. 특히 메카넘 휠의 특성

을 극대화하여 정방향 주행 중 발생하는 위치 오차를 대각선 이동 로직으로 보정하며, 목적지 도달 시에는 서보 모터를 구동하여 물리적인 하역 작업을 수행한다. 또한 최단 경로 산출 알고리즘을 내장하여 특정 시점에 전달받은 장애물 정보를 바탕으로 이동 구간의 비용을 계산하고, 가장 효율적인 운송 동선을 스스로 생성한다.

IV. 모듈별 설계

인프라 감시 모듈은 OpenCV의 BackgroundSubtractorMOG2 알고리즘을 기반으로 하여 동적 장애물을 실시간으로 식별하도록 설계되었다. 고정된 시점에서 입력되는 영상의 배경을 학습하고 가우시안 혼합 모델을 적용함으로써 전경과 배경을 분리한다. 연산 효율을 높이기 위해 전체 프레임이 아닌 특정 셀 영역(ROI)만을 분석 대상으로 설정하며, 감지된 물체의 면적이 사전에 정의된 임계치를 초과할 경우 해당 좌표를 장애물로 판정한다. 이렇게 식별된 장애물 정보는 인프라 수준에서는 실시간으로 발행(Publish)되지만, 이는 시스템 전체의 주행 로직과는 분리되어 독립적인 데이터 스트림으로 존재하도록 설계되었다.

중앙 관제 및 비전 모듈은 시스템의 논리적 제어와 사용자 인터페이스를 담당하는 핵심 계층으로 설계되었다. Flask 웹 프레임워크를 사용하여 관리자가 주문을 입력하고 로봇의 상태를 모니터링할 수 있는 환경을 제공한다. 비전 설계 측면에서는 하향 카메라로부터 유입되는 영상에서 QR 코드를 검출하고, 코드에 내장된 좌표 정보를 기반으로 로봇의 위치를 실시간 갱신한다. 특히 로봇의 중심과 QR 코드의 중심 간 편차 및 회전 각도를 계산하여 정밀 오차 보정치(errX, errY, angle)를 생성한다. 통신 설계 측면에서는 인프라 모듈로부터 수신되는 장애물 정보를 즉시 로봇에게 전송하지 않고 내부 변수에 캐싱(Caching)하도록 설계되었으며, 주행 중 발생할 수 있는 데이터 병목 현상을 방지하기 위해 특정 이벤트 시점에만 전송 프로세스를 활성화하는 전송 제어 로직을 포함한다.

물리 구동 제어 모듈은 ESP32 마이크로컨트롤러를 중심으로 메카넘 휠의 전방향 구동 및 정밀 자세 제어를 수행한다. MPU-9250 센서로부터 유입되는 각도 오차 데이터에 비례 상수를 곱하여 모터 속도를 조정하는 비례 제어 피드백 루프를 형성함으로써, 물리적 주행 환경에서 발생할 수 있는 직진성 저하 문제를 해결하도록 설계되었다. 특히 맵 데이터의 무결성을 보장하기 위해 유한 상태 머신(Finite State Machine)의 전환 시점(주문 수락 전, 하역 후 대기 중 등)에서만 외부 데이터 수신

을 승인하도록 설계되었다. 또한 경로가 폐쇄되었을 때 진입하는 '경로 재탐색 대기(WAIT_PATH_RECOVERY)' 상태를 설계하여, 맵이 최신화될 때까지 주행 프로세스를 일시 중지하고 안전하게 다음 경로를 생성하는 예외 처리 메커니즘을 내장하고 있다.

V. 전체 시스템 설계

전체 시스템은 비전 인식, 통신 네트워크, 물리 제어 계층이 유기적으로 통합된 결합체로 설계되었다. 본 시스템은 통신 부하를 최적화하고 주행 안정성을 확보하기 위하여 모든 데이터를 실시간으로 동기화하는 대신, 특정 이벤트 시점에 맵 데이터를 갱신하는 전략적 동기화 방식을 채택한다. 사용자가 웹 인터페이스를 통해 주문을 입력하면, 중앙 관제 서버는 인프라 모듈로부터 수신하여 캐싱(Caching)해 두었던 최신 장애물 정보를 ESP32로 즉시 전송하여 하위 제어기의 맵을 최신화한다. 하위 제어기는 갱신된 맵을 기반으로 경로 탐색을 수행하며, 목적지까지의 가용한 경로가 확보된 경우에만 주문을 최종 수락하고 주행을 개시한다.

주행 과정에서 로봇은 각 그리드 지점에 도달할 때마다 하향 카메라를 활용한 위치 정렬(Alignment) 과정을 수행한다. 이때 자이로 센서 기반의 방향 보정과 QR 코드 기반의 위치 보정이 순차적으로 이루어지며, QR 코드로부터 산출된 위치 오차 데이터(errX, errY, angle)가 허용 범위 이내로 수렴할 때까지 미세 보정 주행을 반복한다. 보정이 완료된 후에야 다음 좌표로의 이동을 승인함으로써 누적 오차에 의한 주행 탈선 가능성을 원천적으로 차단한다. 특히, 주행 중 급격한 데이터 유입으로 인한 제어 로직의 간섭을 방지하기 위해 이동 구간에서는 맵 업데이트를 일시적으로 제한하도록 설계되었다. 대신, 시스템은 주행 중 발생할 수 있는 환경 변화에 대응하기 위해 주요 정지 이벤트 시점을 활용하여 맵을 재갱신한다. 물류 구역에 도착하여 사용자가 적재 완료 버튼을 누른 직후나, 목적지에서 서보 모터를 활용한 하역 동작을 마친 직후가 이에 해당한다. 이 시점에 로봇은 즉시 이동하지 않고 잠시 대기하며 상위 서버로부터 최신 장애물 정보를 수신하여 경로의 유효성을 다시 한번 검증한다.

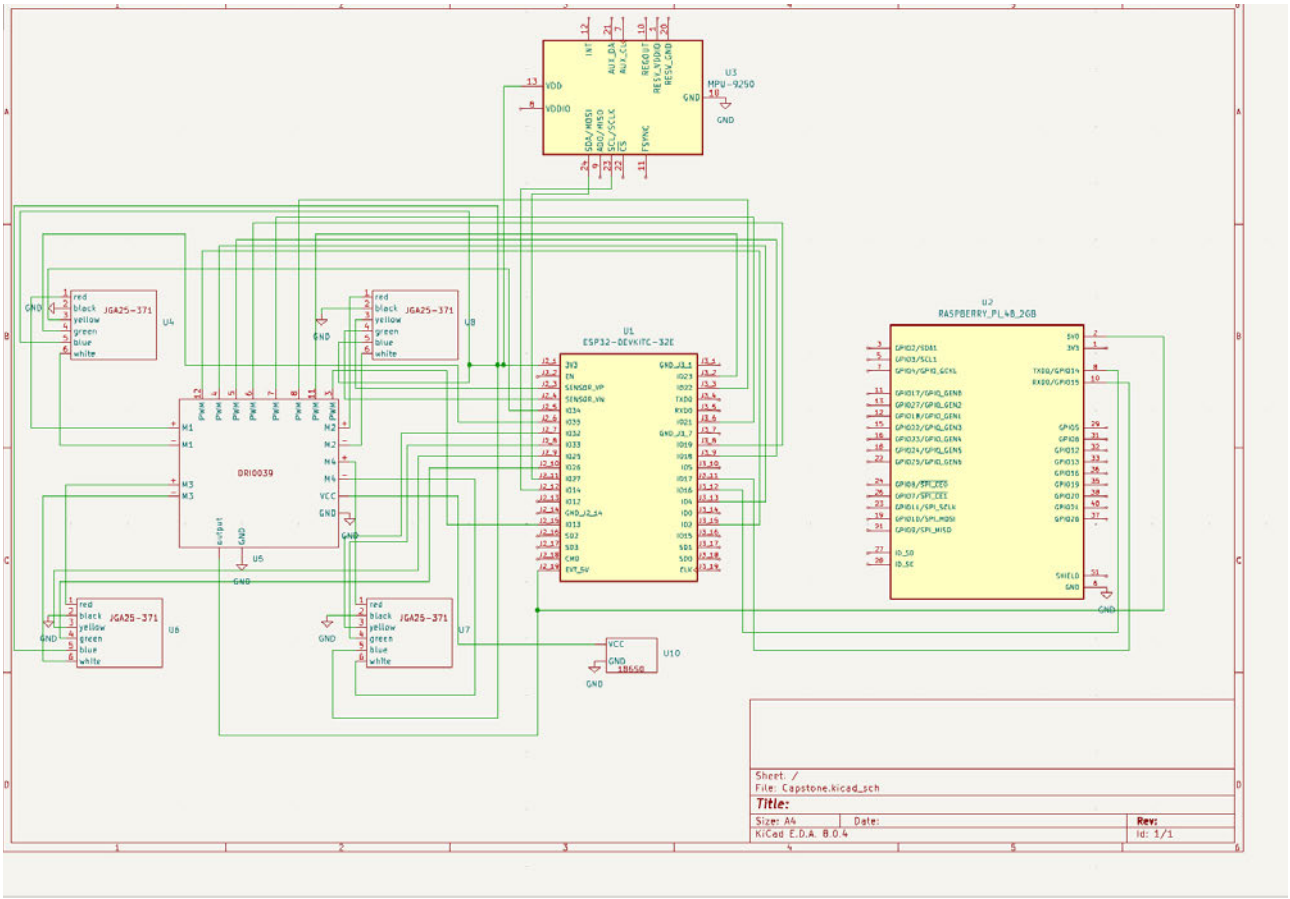
만약 주행 경로상의 모든 행(Row)이 장애물로 폐쇄되어 이동이 불가능한 예외 상황이 발생할 경우, 로봇은 강제로 주행을 시도하지 않고 상위 서버에 경로 막힘 신

호(EVT,START_BLOCKED 또는 EVT,PATH_BLOCKED)를 보고한다. 이 신호를 수신한 서버는 업데이트 락(Lock)을 일시적으로 해제하여 인프라 모듈로부터 장애물 해제 정보를 지속적으로 모니터링한다. 로봇은 '경로 재탐색 대기(WAIT_PATH_RECOVERY)' 상태에서 주기적으로 갱신된 맵 정보를 확인하며, 경로가 다시 확보되어 대체 경로 생성이 가능해지는 시점에 주행을 자동으로 재개하도록 설계되었다.

최종 목적지에서 모든 공정을 마친 로봇은 마지막으로 맵 최신화 과정을 거친 뒤 시작 지점인 (3, 1) 좌표로 자동 회항하며 업무를 종료한다. 이러한 전체 운용 과정은 웹 관제 화면을 통해 실시간 영상 스트리밍과 그래픽 기반의 상태 배지로 사용자에게 전달되며, 긴급 상황 시 즉각적인 소프트웨어적 제동이 가능하도록 설계되어 시스템 전반의 운용 안정성과 신뢰성을 극대화한다.

VI. 제작내용 (회로도, 소스코드 등 첨부)

- 회로도



- 소스 코드

[인프라 감시 모듈 (Raspberry Pi A)]

```
from picamera2 import Picamera2
import cv2
import numpy as np
import time
import json
import os
import paho.mqtt.client as mqtt

BROKER_IP = "192.168.137.194"
BROKER_PORT = 1883
MQTT_TOPIC = "test/json"
MQTT_SEND_INTERVAL = 0.5

CELL_ORDER = [
    (1, 0), (2, 0),
    (1, 1), (2, 1),
    (1, 2), (2, 2),
]

ROI_FILE = "roi_poly.json"
STATE_HISTORY_LEN = 5
STATE_BLOCK_THRESHOLD = 3
FIXED_BOTTOM_PX = 25

roi_map = {}
current_pts = []
current_idx = 0
global_bg_sub = None
global_roi_mask = None
cell_masks = {}
state_history = {}
last_mqtt_send_time = 0

def get_cell_params(cell):
    x, y = cell
    if y == 2:
        return {"min_area": 120}
    elif y == 1:
        return {"min_area": 60}
```

```
    else:
        return {"min_area": 45}

def mqtt_send(blocked, client):
    payload = {
        "timestamp": time.time(),
        "blocked": blocked
    }
    client.publish(MQTT_TOPIC, json.dumps(payload))
    print("[MQTT SENT]", payload)

def save_roi():
    data = {f"{x},{y}": pts for (x, y), pts in roi_map.items()}
    with open(ROI_FILE, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)
    print("[ROI] saved")

def load_roi():
    global roi_map
    if not os.path.exists(ROI_FILE):
        return False
    with open(ROI_FILE, "r", encoding="utf-8") as f:
        data = json.load(f)
    roi_map = {tuple(map(int, k.split(","))): v for k, v in data.items()}
    print("[ROI] loaded")
    return True

def mouse(event, x, y, flags, param):
    global current_pts
    if event == cv2.EVENT_LBUTTONDOWN:
        current_pts.append((x, y))

def draw_existing_rois(frame):
    for cell, pts in roi_map.items():
        pts_np = np.array(pts, np.int32)
        cv2.polylines(frame, [pts_np], True, (255, 0, 0), 2)
        x, y, w, h = cv2.boundingRect(pts_np)
        cv2.putText(frame, f"{cell}", (x + 5, y + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

def draw_current_points(frame):
    for i, p in enumerate(current_pts):
        cv2.circle(frame, p, 4, (0, 255, 255), -1)
        cv2.putText(frame, str(i), (p[0] + 5, p[1] - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255),
```

1)

```
    if len(current_pts) >= 2:
        pts_np = np.array(current_pts, np.int32)
        cv2.polylines(frame, [pts_np], False, (0, 255, 255), 1)

def draw_calib_guide(frame):
    if current_idx < len(CELL_ORDER):
        target = CELL_ORDER[current_idx]
        msg1 = f"Draw ROI for {target} with clicks"
        msg2 = "ENTER: finish / z: undo / c: clear all / s: save"
        cv2.putText(frame, msg1, (20, 35), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,255,255), 2)
        cv2.putText(frame, msg2, (20, 70), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,255), 2)
    else:
        cv2.putText(frame, "All ROI done. Press 's' to save", (20, 35), cv2.FONT_HERSHEY_SIMPLEX, 1.0,
(0,255,0), 2)

def calibrate(cam):
    global current_idx, current_pts, roi_map
    current_idx = 0
    current_pts = []
    roi_map = {}

    cv2.namedWindow("calib")
    cv2.setMouseCallback("calib", mouse)

    while True:
        frame = cam.capture_array()
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        frame = cv2.flip(frame, -1)
        disp = frame.copy()

        draw_existing_rois(disp)
        draw_current_points(disp)
        draw_calib_guide(disp)

        cv2.imshow("calib", disp)
        key = cv2.waitKey(30) & 0xFF

        if key == 13:
            if current_idx < len(CELL_ORDER) and len(current_pts) >= 3:
                roi_map[CELL_ORDER[current_idx]] = current_pts.copy()
                print(f"[ROI SET] {CELL_ORDER[current_idx]} = {current_pts}")
                current_pts.clear()
```

```
                current_idx += 1
            elif key == ord('z'):
                if len(current_pts) > 0: current_pts.pop()
            elif key == ord('c'):
                roi_map.clear(); current_pts.clear(); current_idx = 0
                print("[ROI] cleared")
            elif key == ord('s'):
                if len(roi_map) == len(CELL_ORDER):
                    save_roi()
                    break
                else:
                    print("[ROI] not complete yet")
            elif key == 27:
                break

        cv2.destroyWindow("calib")

    def save_background(frame):
        global global_bg_sub, global_roi_mask, state_history, cell_masks
        global_roi_mask = np.zeros(frame.shape[:2], dtype=np.uint8)
        cell_masks.clear()

        for cell, pts in roi_map.items():
            pts_np = np.array(pts, np.int32)
            cv2.fillPoly(global_roi_mask, [pts_np], 255)

            c_mask = np.zeros(frame.shape[:2], dtype=np.uint8)
            cv2.fillPoly(c_mask, [pts_np], 255)
            cell_masks[cell] = c_mask

        global_bg_sub = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=40,
    detectShadows=True)

        frame_blurred = cv2.GaussianBlur(frame, (15, 15), 0)
        frame_masked = cv2.bitwise_and(frame_blurred, frame_blurred, mask=global_roi_mask)

        for _ in range(5):
            global_bg_sub.apply(frame_masked, learningRate=1.0)

        state_history.clear()
        for cell in CELL_ORDER:
            state_history[cell] = []
        print("[BG] Color MOG2 Background initialized")
```

```
def analyze(frame):
    raw_states = {cell: 0 for cell in CELL_ORDER}
    stable_states = {cell: "FREE" for cell in CELL_ORDER}
    debug_info = {"contours": [], "bottom_boxes": []}

    if global_bg_sub is None:
        return raw_states, stable_states, debug_info

    frame_blurred = cv2.GaussianBlur(frame, (15, 15), 0)
    frame_masked = cv2.bitwise_and(frame_blurred, frame_blurred, mask=global_roi_mask)

    fg_mask = global_bg_sub.apply(frame_masked, learningRate=0)
    _, th = cv2.threshold(fg_mask, 254, 255, cv2.THRESH_BINARY)

    kernel_small = np.ones((3, 3), np.uint8)
    th = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel_small)
    th = cv2.morphologyEx(th, cv2.MORPH_CLOSE, kernel_small)

    contours, _ = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    valid_contours = []
    bottom_boxes = []

    for cnt in contours:
        if cv2.contourArea(cnt) < 45:
            continue

        valid_contours.append(cnt)

        x, y, w, h = cv2.boundingRect(cnt)

        bottom_h = min(FIXED_BOTTOM_PX, h)
        bottom_y = y + h - bottom_h

        box_area = w * bottom_h
        if box_area == 0: continue

        best_cell = None
        max_overlap = 0

        for cell, c_mask in cell_masks.items():
            c_mask_crop = c_mask[bottom_y:bottom_y+bottom_h, x:x+w]
            overlap_area = cv2.countNonZero(c_mask_crop)
```

```
            if overlap_area > max_overlap:
                max_overlap = overlap_area
                best_cell = cell

        if best_cell is not None and max_overlap > (box_area * 0.15):
            raw_states[best_cell] = 1

        bottom_boxes.append((x, bottom_y, w, bottom_h, best_cell))

    for cell in CELL_ORDER:
        hist = state_history.get(cell, [])
        hist.append(raw_states[cell])
        if len(hist) > STATE_HISTORY_LEN: hist.pop(0)
        state_history[cell] = hist
        stable_states[cell] = "BLOCKED" if sum(hist) >= STATE_BLOCK_THRESHOLD else "FREE"

    debug_info["contours"] = valid_contours
    debug_info["bottom_boxes"] = bottom_boxes

    return raw_states, stable_states, debug_info

def draw_runtime(frame, raw_states, stable_states, debug_info):
    cv2.putText(frame, "b: save background", (20, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,0), 2)
    cv2.putText(frame, "r: reset background", (20, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,0), 2)
    cv2.putText(frame, "q: recalibrate ROI", (20, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,0), 2)
    cv2.putText(frame, "ESC: quit", (20, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,0), 2)

    for cell, pts in roi_map.items():
        pts_np = np.array(pts, np.int32)
        state = stable_states.get(cell, "FREE")
        raw = raw_states.get(cell, 0)
        color = (0, 255, 0) if state == "FREE" else (0, 0, 255)

        cv2.polylines(frame, [pts_np], True, color, 2)
        x, y, w, h = cv2.boundingRect(pts_np)
        cv2.putText(frame, f"{state} raw={raw} {cell}", (x + 5, y + 20), cv2.FONT_HERSHEY_SIMPLEX,
0.55, color, 2)

    for cnt in debug_info.get("contours", []):
        cv2.drawContours(frame, [cnt], -1, (0, 255, 255), 2)

    for box in debug_info.get("bottom_boxes", []):
```

```
        x, by, w, bh, cell = box

        overlay = frame.copy()
        cv2.rectangle(overlay, (x, by), (x + w, by + bh), (0, 0, 255), -1)
        cv2.addWeighted(overlay, 0.4, frame, 0.6, 0, frame)
        cv2.rectangle(frame, (x, by), (x + w, by + bh), (0, 0, 255), 2)

        if cell:
            cv2.putText(frame, str(cell), (x, by - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

def get_blocked_cells(stable_states):
    return [list(cell) for cell in CELL_ORDER if stable_states.get(cell) == "BLOCKED"]

def runtime_mode(cam, client):
    global last_mqtt_send_time, global_bg_sub, state_history

    while True:
        frame = cam.capture_array()
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        frame = cv2.flip(frame, -1)
        disp = frame.copy()

        background_ready = (global_bg_sub is not None)

        if not background_ready:
            cv2.putText(disp, "WAITING FOR BG - PRESS 'b'", (350, 360), cv2.FONT_HERSHEY_SIMPLEX, 1.2,
(0, 0, 255), 3)

        raw_states = {cell: 0 for cell in CELL_ORDER}
        stable_states = {cell: "FREE" for cell in CELL_ORDER}
        debug_info = {cell: {} for cell in CELL_ORDER}

        if background_ready:
            raw_states, stable_states, debug_info = analyze(frame)

            now = time.time()
            if (now - last_mqtt_send_time) >= MQTT_SEND_INTERVAL:
                blocked = get_blocked_cells(stable_states)
                mqtt_send(blocked, client)
                last_mqtt_send_time = now

        draw_runtime(disp, raw_states, stable_states, debug_info)
        cv2.imshow("runtime", disp)
```

```
        key = cv2.waitKey(30) & 0xFF

        if key == ord('b'):
            save_background(frame)
        elif key == ord('r'):
            global_bg_sub = None
            state_history.clear()
            print("[BG] reset")
        elif key == ord('q'):
            cv2.destroyWindow("runtime")
            calibrate(cam)
            global_bg_sub = None
            state_history.clear()
        elif key == 27:
            break

def main():
    client = mqtt.Client()
    client.connect(BROKER_IP, BROKER_PORT, 60)
    client.loop_start()

    cam = Picamera2()
    config = cam.create_preview_configuration(main={"size": (1280, 720), "format": "RGB888"})
    cam.configure(config)
    cam.start()
    time.sleep(1.0)

    if not load_roi():
        calibrate(cam)

    runtime_mode(cam, client)

    cam.stop()
    client.loop_stop()
    client.disconnect()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

[관제 및 비전 모듈 (Raspberry Pi B)]

```
import cv2
import numpy as np
import datetime
import json
import re
import time
import serial
import threading
import RPi.GPIO as GPIO
import math
import paho.mqtt.client as mqtt
from picamera2 import Picamera2
from flask import Flask, render_template_string, request, jsonify, Response

try:
    from pyzbar.pyzbar import decode
except ImportError:
    decode = None

app = Flask(__name__)

try:
    ser = serial.Serial('/dev/ttyS0', 115200, timeout=1)
    ser.flush()
except:
    ser = None

uart_lock = threading.Lock()

def send_uart(msg):
    if ser and ser.isOpen():
        with uart_lock:
            try: ser.write(msg.encode('utf-8'))
            except: pass

BUTTON_PIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

data_store = {
    'order_A': 0, 'order_B': 0, 'robot_location': '대기중',
    'robot_state': '대기중',
    'current_coord': '3,1', 'history': [], 'loc_history': [], 'has_visited_box': False,
```

```
    'obstacles': [],
    'order_queue': [],
    'current_order': None
}

SPECIAL_LOCATIONS = {
    "0,0": "Box1(0,0)", "0,1": "Box2(0,1)", "0,2": "Box3(0,2)",
    "3,0": "물류A(3,0)", "3,2": "물류B(3,2)"
}
BOX_COORDS = ["0,0", "0,1", "0,2"]

obstacle_lock = False
web_update_lock = False
current_obstacles = set()

def is_path_fully_blocked():
    for y in range(3):
        row_clear = True
        for x in range(4):
            if f"{x},{y}" in current_obstacles:
                row_clear = False
                break
        if row_clear:
            return False
    return True

def on_message(client, userdata, msg):
    global current_obstacles
    try:
        data = json.loads(msg.payload.decode('utf-8'))
        blocked_list = data.get("blocked", [])
        robot_loc = data_store.get('current_coord', '')
        current_obstacles = {f"{x},{y}" for x, y in blocked_list if f"{x},{y}" != robot_loc}

        if not web_update_lock:
            data_store['obstacles'] = list(current_obstacles)

        if not obstacle_lock:
            send_uart("CLEAR_MAPWn")
            time.sleep(0.01)
            for obs in current_obstacles:
                send_uart(f"BLOCK,{obs}Wn")
                time.sleep(0.01)
```

```
        if data_store['current_order'] is None and len(data_store['order_queue']) > 0:
            process_queue()
    except:
        pass

mqtt_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
mqtt_client.on_message = on_message
try:
    mqtt_client.connect("192.168.137.194", 1883, 60)
    mqtt_client.subscribe("test/json")
    mqtt_client.loop_start()
except:
    pass

snapshot_running = False
last_retry_time = 0

def temporary_lock_and_update(duration):
    global obstacle_lock, web_update_lock
    obstacle_lock = False
    web_update_lock = False
    time.sleep(duration)
    obstacle_lock = True
    web_update_lock = True
    data_store['obstacles'] = list(current_obstacles)
    send_uart("CLEAR_MAPWn")
    time.sleep(0.05)
    for obs in current_obstacles:
        send_uart(f"BLOCK,{obs}Wn")
        time.sleep(0.02)

def uart_receiver():
    global snapshot_running
    while True:
        if ser and ser.in_waiting:
            try:
                line = ser.readline().decode('utf-8').strip()
                if not line: continue

                if line.startswith("STATE,"):
                    data_store['robot_state'] = line.split(",")[1]
```

```
            elif line == "EVT,JOB_DONE":
                now = datetime.datetime.now().strftime("%H:%M:%S")
                if data_store['current_order']:
                    box_id = data_store['current_order']['box_id']
                    data_store['history'].append({'time': now, 'msg': f"□ [Box {box_id}] 배달 완료
및 복귀"})
                data_store['current_order'] = None
                process_queue()

            elif line == "CMD,LOAD_START":
                threading.Thread(target=temporary_lock_and_update,                    args=(3.0,),
daemon=True).start()

            elif line == "EVT,START_BLOCKED":
                now = datetime.datetime.now().strftime("%H:%M:%S")
                if   len(data_store['history'])   ==   0   or   "모든   경로   막힘"   not   in
data_store['history'][-1]['msg']:
                    data_store['history'].append({'time': now, 'msg': "⚠ 모든 경로 막힘. 뚫릴 때까
지 대기합니다."})

                if data_store['current_order'] is not None:
                    data_store['order_queue'].insert(0, data_store['current_order'])
                    data_store['current_order'] = None

            elif line == "EVT,PATH_BLOCKED":
                data_store['robot_state'] = '경로 대기중'
                now = datetime.datetime.now().strftime("%H:%M:%S")
                if   len(data_store['history'])   ==   0   or   "모든   경로   막힘"   not   in
data_store['history'][-1]['msg']:
                    data_store['history'].append({'time': now, 'msg': "⚠ 모든 경로 막힘. 뚫릴 때까
지 대기합니다."})

                if not snapshot_running:
                    snapshot_running = True
                    def run_snapshot():
                        global snapshot_running
                        temporary_lock_and_update(3.0)
                        snapshot_running = False
                    threading.Thread(target=run_snapshot, daemon=True).start()

            elif line == "EVT,READY_TO_RETURN":
                if not snapshot_running:
                    snapshot_running = True
```

```
                        def run_snapshot():
                            global snapshot_running
                            temporary_lock_and_update(3.0)
                            snapshot_running = False
                        threading.Thread(target=run_snapshot, daemon=True).start()
            except:
                pass
        time.sleep(0.01)

def button_listener():
    while True:
        if GPIO.input(BUTTON_PIN) == 1:
            send_uart("BTN,303Wn")
            time.sleep(1.0)
        time.sleep(0.1)

def process_queue():
    global last_retry_time
    if time.time() - last_retry_time < 1.0: return
    last_retry_time = time.time()

    if data_store['current_order'] is None and len(data_store['order_queue']) > 0:
        if is_path_fully_blocked():
            if data_store['robot_state'] != '경로 대기중':
                data_store['robot_state'] = '경로 대기중'
                now = datetime.datetime.now().strftime("%H:%M:%S")
                data_store['history'].append({'time': now, 'msg': "⚠ 모든 경로 막힘. 뚫릴 때까지 대기
합니다."})
            return

        data_store['current_order'] = data_store['order_queue'].pop(0)
        order = data_store['current_order']

        global obstacle_lock, web_update_lock
        obstacle_lock = True
        web_update_lock = True
        data_store['obstacles'] = list(current_obstacles)

        send_uart("CLEAR_MAPWn")
        time.sleep(0.05)
        for obs in current_obstacles:
            send_uart(f"BLOCK,{obs}Wn")
            time.sleep(0.01)
```

```
        time.sleep(0.1)
        send_uart(f"ORD,{order['box_id']},{order['count_a']},{order['count_b']}Wn")
try:
    picam2 = Picamera2()
    config = picam2.create_video_configuration(main={"size": (640, 480)})
    picam2.configure(config)
    picam2.start()
except:
    picam2 = None

basic_detector = cv2.QRCodeDetector()
last_logged_loc = ""
last_uart_time = 0

def generate_frames():
    global last_logged_loc, last_uart_time
    while True:
        if picam2:
            try: frame = picam2.capture_array()
            except: time.sleep(0.1); continue
        else:
            time.sleep(1); continue

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        detected_qr = None
        pts_for_calc = None

        if decode:
            decoded_objects = decode(gray)
            for obj in decoded_objects:
                detected_qr = obj.data.decode('utf-8')
                points = obj.polygon
                if len(points) >= 4:
                    pts_for_calc = np.array(points, np.int32).reshape((-1, 1, 2))
                    cv2.polylines(frame, [pts_for_calc], True, (0, 255, 0), 4)
                    cv2.putText(frame,  detected_qr,  (pts_for_calc[0][0][0],  pts_for_calc[0][0][1]  -
10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
        else:
            detected_qr, bbox, _ = basic_detector.detectAndDecode(frame)
            if detected_qr and bbox is not None:
                pts_for_calc = bbox[0].astype(int).reshape((-1, 1, 2))
                cv2.polylines(frame, [pts_for_calc], True, (0, 255, 0), 4)
```

```
current_time = time.time()
if detected_qr:
    match = re.search(r'(Wd+)Ws*,Ws*(Wd+)', detected_qr)
    if match:
        x, y = match.group(1), match.group(2)
        clean_coord = f"{x},{y}"
        display_text = ""

        err_x, err_y, angle_deg = 0, 0, 0
        if pts_for_calc is not None and len(pts_for_calc) >= 4:
            pts_list = [p[0] for p in pts_for_calc]
            cx = int(sum([p[0] for p in pts_list]) / 4)
            cy = int(sum([p[1] for p in pts_list]) / 4)
            err_x, err_y = cx - 320, cy - 240
            sorted_by_y = sorted(pts_list, key=lambda p: p[1])
            top_two = sorted(sorted_by_y[:2], key=lambda p: p[0])
            tl, tr = top_two[0], top_two[1]
            angle_deg = int(math.degrees(math.atan2(tr[1] - tl[1], tr[0] - tl[0])))

        if current_time - last_uart_time > 0.1:
            send_uart(f"POS,{x},{y},{err_x},{err_y},{angle_deg}Wn")
            last_uart_time = current_time

        if clean_coord in BOX_COORDS: data_store['has_visited_box'] = True

        if clean_coord == "3,1":
            display_text = "대기중" if data_store['has_visited_box'] else "좌표(3,1) 통과중"
            if data_store['robot_state'] == '대기중':
                global obstacle_lock, web_update_lock
                obstacle_lock = False
                web_update_lock = False
            if data_store['current_order'] is None and len(data_store['order_queue']) > 0:
                process_queue()
        elif clean_coord in SPECIAL_LOCATIONS:
            place_name = SPECIAL_LOCATIONS[clean_coord]
            display_text = f"{place_name} 도착"
            if "물류" in place_name: data_store['has_visited_box'] = False
        else:
            display_text = f"좌표({clean_coord}) 통과중"

        data_store['robot_location'] = display_text
        data_store['current_coord'] = clean_coord
```

```
                if clean_coord != last_logged_loc:
                    now = datetime.datetime.now().strftime("%H:%M:%S")
                    if display_text == "대기중": log_msg = "좌표(3,1) 대기중"
                    elif clean_coord in SPECIAL_LOCATIONS: log_msg = f"□ {SPECIAL_LOCATIONS[clean_coord]} 도착"
                    else: log_msg = f"□ {display_text}"
                    data_store['loc_history'].append({'time': now, 'msg': log_msg})
                    last_logged_loc = clean_coord
            else:
                data_store['robot_location'] = f"인식됨: {detected_qr}"
        else:
            thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 51, 10)
            kernel = np.ones((5,5), np.uint8)
            thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
            contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            best_contour, max_area = None, 0
            for cnt in contours:
                area = cv2.contourArea(cnt)
                if 500 < area < 25000:
                    x, y, w, h = cv2.boundingRect(cnt)
                    aspect_ratio = float(w) / h
                    if w < 250 and h < 250 and 0.4 < aspect_ratio < 2.5:
                        if area > max_area:
                            max_area = area
                            best_contour = cnt
            if best_contour is not None:
                M = cv2.moments(best_contour)
                if M["m00"] != 0:
                    cx, cy = int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"])
                    err_x, err_y = cx - 320, cy - 240
                    cv2.drawContours(frame, [best_contour], -1, (0, 255, 255), 2)
                    cv2.circle(frame, (cx, cy), 5, (0, 255, 255), -1)
                    if current_time - last_uart_time > 0.1:
                        send_uart(f"PART,{err_x},{err_y}Wn")
                        last_uart_time = current_time

            current_text = data_store['robot_location']
            if not any(x in current_text for x in ["완료", "대기중", "Box", "물류"]):
                data_store['robot_location'] = "□ 이동중..."

        ret, buffer = cv2.imencode('.jpg', frame)
        yield (b'--frameWrWn' b'Content-Type: image/jpegWrWnWrWn' + buffer.tobytes() + b'WrWn')
```

```
@app.route('/')
def index():
    return         render_template_string(html_template,         location=data_store['robot_location'],
qty_a=data_store['order_A'], qty_b=data_store['order_B'])

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/status')
def status():
    return jsonify(data_store)

@app.route('/order', methods=['POST'])
def order():
    box_id = int(request.form.get('box_id', 1))
    count_a = int(request.form.get('count_a', 0))
    count_b = int(request.form.get('count_b', 0))
    if count_a > 0 or count_b > 0:
        data_store['order_A'] += count_a
        data_store['order_B'] += count_b
        now = datetime.datetime.now().strftime("%H:%M:%S")
        new_order = {"box_id": box_id, "count_a": count_a, "count_b": count_b}
        data_store['order_queue'].append(new_order)
        data_store['history'].append({'time':  now,  'msg':  f"[Box{box_id}]  접수  대기  (A:{count_a},
B:{count_b})"})
        process_queue()
    return jsonify({"status": "success"})

@app.route('/delete_queue', methods=['POST'])
def delete_queue():
    idx = int(request.form.get('idx', -1))
    if 0 <= idx < len(data_store['order_queue']):
        removed = data_store['order_queue'].pop(idx)
        data_store['order_A'] = max(0, data_store['order_A'] - removed.get('count_a', 0))
        data_store['order_B'] = max(0, data_store['order_B'] - removed.get('count_b', 0))
        now = datetime.datetime.now().strftime("%H:%M:%S")
        data_store['history'].append({'time': now, 'msg': f"□ 주문 취소: Box {removed['box_id']}"})
    return jsonify({"status": "success"})

@app.route('/reset_totals', methods=['POST'])
```

```
def reset_totals():
    data_store['order_A'] = 0
    data_store['order_B'] = 0
    now = datetime.datetime.now().strftime("%H:%M:%S")
    data_store['history'].append({'time': now, 'msg': "□ 총계 수량이 초기화되었습니다."})
    return jsonify({"status": "success"})

@app.route('/emergency_stop', methods=['POST'])
def emergency_stop():
    now = datetime.datetime.now().strftime("%H:%M:%S")
    if data_store['current_order']:
        curr = data_store['current_order']
        data_store['order_A'] = max(0, data_store['order_A'] - curr.get('count_a', 0))
        data_store['order_B'] = max(0, data_store['order_B'] - curr.get('count_b', 0))
        data_store['history'].append({'time': now, 'msg': f"□ 긴급정지 작동! [Box {curr['box_id']}] 주
문 취소됨"})
        data_store['current_order'] = None
    else:
        data_store['history'].append({'time': now, 'msg': "□ 긴급정지 작동!"})
    send_uart("CMD,STOPWn")
    return jsonify({"status": "success"})

@app.route('/reset_order', methods=['POST'])
def reset_order():
    data_store['history'] = []
    return jsonify({"status": "success"})

@app.route('/reset_loc', methods=['POST'])
def reset_loc():
    global last_logged_loc
    data_store['loc_history'] = []; data_store['robot_location'] = "대기중"
    data_store['current_coord'] = "3,1"; data_store['has_visited_box'] = False; last_logged_loc = ""
    return jsonify({"status": "success"})

html_template = """
<!DOCTYPE html>
<html>
<head>
    <title>스마트 물류 관제 시스템</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body { font-family: 'Malgun Gothic', sans-serif; padding: 10px; background-color: #f4f6f9;
```

```
margin: 0; }
        .container { display: flex; flex-wrap: wrap; justify-content: center; gap: 15px; max-width: 1200px; margin: 0 auto; }
        .column { flex: 1; min-width: 320px; }
        .box { background: white; border: 1px solid #ddd; padding: 15px; margin-bottom: 15px; border-radius: 12px; box-shadow: 0 4px 6px rgba(0,0,0,0.05); }
        .log-box { height: 150px; overflow-y: auto; background-color: #f8f9fa; border: 1px solid #eee; padding: 10px; border-radius: 8px; font-size: 13px; }
        .log-item { padding: 4px 0; border-bottom: 1px solid #e9ecef; color: #333; }
        .log-time { color: #888; font-size: 11px; margin-right: 5px; font-weight:bold;}
        .loc-log-item { color: #0056b3; }
        .map-grid { display: grid; grid-template-columns: repeat(4, 1fr); gap: 8px; margin-top: 10px; }
        .map-cell { position: relative; border: 2px solid #ccc; border-radius: 6px; padding: 10px 0; text-align: center; font-weight: bold; color: #aaa; background-color: #f9f9f9; font-size: 12px; word-break: break-all;}
        .active-cell { background-color: #ffeb3b; color: #d32f2f; border-color: #d32f2f; transform: scale(1.05); box-shadow: 0 0 10px rgba(255, 235, 59, 0.8); transition: 0.2s;}
        .cone { position: absolute; top: 2px; right: 5px; font-size: 16px; }
        .camera-screen { width: 100%; max-width: 280px; height: auto; border-radius: 8px; border: 2px solid #333; display: block; margin: 0 auto; }
        .btn { padding: 8px; width: 100%; font-size: 14px; background: #007bff; color: white; border: none; border-radius: 6px; cursor: pointer; }
        .btn-order { background: #28a745; font-size: 16px; font-weight: bold; }
        .btn-reset-order { background: #dc3545; margin-top: 5px; }
        .btn-reset-loc { background: #6c757d; margin-top: 5px; }
        .btn-estop { background: #dc3545; font-weight:bold; }
        .btn-del-small { background: #ff4d4d; color: white; border: none; border-radius: 4px; padding: 3px 8px; font-size: 12px; cursor: pointer; margin-left: 10px; font-weight: bold; }
        .btn-reset-total { background: #ffc107; color: black; font-weight: bold; margin-bottom: 5px; }
        input, select { padding: 8px; text-align: center; border: 1px solid #ccc; border-radius: 5px; }
        input[type="number"] { width: 40px; }
        select { width: 100%; margin-bottom: 10px;}
        h2 { margin: 10px 0; text-align: center; font-size: 24px; color:#333; }
        h3 { margin-top: 0; margin-bottom: 10px; font-size: 18px; border-bottom: 2px solid #eee; padding-bottom: 5px;}
        .status-badge { display: inline-block; padding: 5px 15px; border-radius: 20px; color: white; font-weight: bold; font-size: 16px; }
        .bg-대기중 { background-color: #6c757d; }
        .bg-경로 { background-color: #dc3545; }
        .bg-이동중 { background-color: #007bff; }
        .bg-보정중 { background-color: #fd7e14; }
        .bg-적재중 { background-color: #ffc107; color: black; }
        .bg-하역중 { background-color: #28a745; }
```

```
        .order-panel { background: #f8f9fa; padding: 10px; border-radius: 8px; margin-bottom: 10px; border: 1px solid #ddd; }
        .order-list { list-style: none; padding: 0; margin: 0; font-size: 14px; }
        .order-list li { padding: 5px 0; border-bottom: 1px dashed #ccc; display: flex; justify-content: space-between; align-items: center; }
    </style>
    <script>
        function updateData() {
            fetch('/status').then(r => r.json()).then(data => {
                document.getElementById('loc').innerText = data.robot_location;
                document.getElementById('qty_a').innerText = data.order_A;
                document.getElementById('qty_b').innerText = data.order_B;
                let badge = document.getElementById('robot-state-badge');
                badge.innerText = data.robot_state;
                badge.className = "status-badge bg-" + data.robot_state.split(' ')[0];
                let currOrder = document.getElementById('current-order');
                if (data.current_order) {
                    currOrder.innerHTML = `<span><strong>Box ${data.current_order.box_id}</strong> (A: ${data.current_order.count_a}, B: ${data.current_order.count_b})</span>`;
                    currOrder.style.color = "#0056b3";
                } else {
                    currOrder.innerHTML = "진행중인 주문 없음";
                    currOrder.style.color = "#888";
                }
                let waitHtml = "";
                if (data.order_queue.length === 0) {
                    waitHtml = "<li style='color:#aaa; justify-content:center;'>대기중인 주문 없음</li>";
                } else {
                    data.order_queue.forEach((o, idx) => {
                        waitHtml += `<li><span>[대기 ${idx+1}] Box ${o.box_id} (A: ${o.count_a}, B: ${o.count_b})</span>
                                    <button onclick="deleteQueue(${idx})" class="btn-del-small">삭제</button></li>`;
                    });
                }
                document.getElementById('waiting-orders').innerHTML = waitHtml;
                updateMap(data.current_coord, data.obstacles);
                updateOrderLog(data.history);
                updateLocationLog(data.loc_history);
            });
        }
        let lastHistoryJSON = ""; let lastLocHistoryJSON = "";
```

```
        function updateMap(loc, obstacles) {
            let cells = document.getElementsByClassName('map-cell');
            for(let c of cells) {
                c.classList.remove('active-cell');
                let cone = c.querySelector('.cone');
                if (cone) cone.remove();
            }
            if (obstacles) {
                obstacles.forEach(obs => {
                    let match = obs.match(/(Wd+),(Wd+)/);
                    if (match) {
                        let el = document.getElementById('cell-' + match[1] + '-' + match[2]);
                        if (el) el.innerHTML += '<div class="cone">□</div>';
                    }
                });
            }
            if (loc) {
                let match = loc.match(/(Wd+)Ws*,Ws*(Wd+)/);
                if (match) {
                    let el = document.getElementById('cell-' + match[1] + '-' + match[2]);
                    if (el) el.classList.add('active-cell');
                }
            }
        }
        function updateOrderLog(history) {
            let logBox = document.getElementById('order-log-container');
            let currentJSON = JSON.stringify(history);
            if (lastHistoryJSON !== currentJSON) {
                lastHistoryJSON = currentJSON;
                logBox.innerHTML = history.length ? "" : '<div style="color:#aaa; text-align:center;">
내역 없음</div>';
                [...history].reverse().forEach(item => {
                    logBox.innerHTML += `<div class="log-item"><span
class="log-time">[${item.time}]</span>${item.msg}</div>`;
                });
            }
        }
        function updateLocationLog(history) {
            let logBox = document.getElementById('loc-log-container');
            let currentJSON = JSON.stringify(history);
            if (lastLocHistoryJSON !== currentJSON) {
                lastLocHistoryJSON = currentJSON;
                logBox.innerHTML = history.length ? "" : '<div style="color:#aaa; text-align:center;">
```

```
이동 기록 없음</div>';
                [...history].reverse().forEach(item => {
                    logBox.innerHTML += `<div class="log-item loc-log-item"><span class="log-time">[${item.time}]</span>${item.msg}</div>`;
                });
            }
        }
        function submitOrder(e) { e.preventDefault(); fetch('/order', { method: 'POST', body: new FormData(e.target) }).then(updateData); }
        function submitResetLoc(e) { e.preventDefault(); fetch('/reset_loc', { method: 'POST' }).then(updateData); }
        function submitResetOrder(e) { e.preventDefault(); fetch('/reset_order', { method: 'POST' }).then(updateData); }
        function callEStop() { fetch('/emergency_stop', { method: 'POST' }).then(updateData); }
        function deleteQueue(idx) {
            let fd = new FormData(); fd.append('idx', idx);
            fetch('/delete_queue', { method: 'POST', body: fd }).then(updateData);
        }
        function resetTotals() { if(confirm("총계 수량을 초기화하시겠습니까?")) fetch('/reset_totals', { method: 'POST' }); }
        setInterval(updateData, 500);
    </script>
</head>
<body>
    <h2>□ 스마트 물류 관제 시스템</h2>
    <div class="container">
        <div class="column">
            <div class="box" style="text-align:center;">
                <h3>□ 로봇 상태 및 제어</h3>
                <div id="robot-state-badge" class="status-badge bg-대기중" style="margin-bottom:15px;">대기중</div>
                <div style="display:flex; justify-content:center;">
                    <button onclick="callEStop()" class="btn btn-estop">□ 로봇 긴급 정지</button>
                </div>
            </div>
            <div class="box">
                <h3>□ 실시간 위치 (Map)</h3>
                <p style="margin:5px 0;">상태: <strong id="loc" style="font-size:18px;">{{ location }}</strong></p>
                <div class="map-grid">
                    <div class="map-cell" id="cell-0-0">Box1<br>(0,0)</div> <div class="map-cell" id="cell-1-0">(1,0)</div>
                    <div class="map-cell" id="cell-2-0">(2,0)</div> <div class="map-cell"
```

```
id="cell-3-0">물류A<br>(3,0)</div>
                    <div  class="map-cell"  id="cell-0-1">Box2<br>(0,1)</div>  <div  class="map-cell"
id="cell-1-1">(1,1)</div>
                    <div  class="map-cell"  id="cell-2-1">(2,1)</div>  <div  class="map-cell"
id="cell-3-1">(3,1)</div>
                    <div  class="map-cell"  id="cell-0-2">Box3<br>(0,2)</div>  <div  class="map-cell"
id="cell-1-2">(1,2)</div>
                    <div  class="map-cell"  id="cell-2-2">(2,2)</div>  <div  class="map-cell"
id="cell-3-2">물류B<br>(3,2)</div>
                </div>
            </div>
            <div class="box">
                <div style="display:flex; justify-content:space-between; align-items:center;">
                    <h3>□ 이동 경로</h3>
                    <form  onsubmit="submitResetLoc(event)"><button  type="submit"  class="btn
btn-reset-loc" style="font-size:12px;">□ 경로 초기화</button></form>
                </div>
                <div id="loc-log-container" class="log-box"></div>
            </div>
        </div>
        <div class="column">
            <div class="box" style="background:#e3f2fd;">
                <h3>□ 주문 현황 (대기열)</h3>
                <div class="order-panel">
                    <span style="font-size:13px; color:#555;">▶ 진행중:</span>
                    <div  id="current-order"  style="font-size:16px;  margin-top:5px;  display:flex;
justify-content:space-between; align-items:center;">진행중인 주문 없음</div>
                </div>
                <div class="order-panel">
                    <span style="font-size:13px; color:#555;">□ 대기열:</span>
                    <ul id="waiting-orders" class="order-list"></ul>
                </div>
                <hr style="border:0; border-top:1px solid #ccc; margin:15px 0;">
                <div style="text-align:center; margin-bottom:10px;">
                    <button  onclick="resetTotals()"  class="btn  btn-reset-total"  style="padding:4px
10px; font-size:12px;">□ 총계 수량 초기화</button><br>
                    □ 총계 A: <b id="qty_a">{{ qty_a }}</b> □ 총계 B: <b id="qty_b">{{ qty_b }}</b>
                </div>
                <form onsubmit="submitOrder(event)">
                    <div style="display:flex; justify-content:center; gap:5px; margin-bottom:10px;">
                        <select name="box_id" style="width: auto;">
                            <option value="1">Box 1 도착지</option>
                            <option value="2">Box 2 도착지</option>
```

```
                        <option value="3">Box 3 도착지</option>
                    </select>
                </div>
                <div style="display:flex; justify-content:center; align-items:center; gap:5px;">
                    A <input type="number" name="count_a" value="0" min="0"> B <input type="number"
name="count_b" value="0" min="0">
                    <button type="submit" class="btn btn-order">주문 접수</button>
                </div>
            </form>
        </div>
        <div class="box">
            <div style="display:flex; justify-content:space-between; align-items:center;">
                <h3>□ 시스템 주문 로그</h3>
                <form    onsubmit="submitResetOrder(event)"><button    type="submit"    class="btn
btn-reset-order" style="font-size:12px;">□ 로그 초기화</button></form>
            </div>
            <div id="order-log-container" class="log-box"></div>
        </div>
        <div class="box" style="text-align:center; background:#222; color:white;">
            <h3 style="color:white; border-bottom:1px solid #555;">□ QR 실시간 인식</h3>
            <img src="/video_feed" class="camera-screen">
        </div>
    </div>
</body>
</html>
"""

if __name__ == '__main__':
    threading.Thread(target=button_listener, daemon=True).start()
    threading.Thread(target=uart_receiver, daemon=True).start()
    app.run(host='0.0.0.0', port=5000, debug=False, threaded=True)
```

[구동 및 주행 제어 모듈 (ESP32)]

```
#include <Arduino.h>
#include <MPU9250.h>
#include <Wire.h>
#include <ESP32Servo.h>

struct Pos { int x; int y; };
```

```
#define RXp2 16
#define TXp2 17

const int M1_PWM = 13; const int M1_DIR = 4;
const int M2_PWM = 23; const int M2_DIR = 2;
const int M3_PWM = 18; const int M3_DIR = 22;
const int M4_PWM = 19; const int M4_DIR = 21;

Servo myServo;
const int SERVO_PIN = 5;
const int PWM_STRAIGHT = 75;
const int PWM_STRAFE   = 100;

const int FWD_LEFT_TRIM = 0;
const int BWD_LEFT_TRIM = 3;

const int ALIGN_PWM_X = 60;
const int ALIGN_PWM_Y = 50;
const int ALIGN_PWM_ANG = 55;
const int ALIGN_PWM_DIAG = 75;

const int TOL_X = 10;
const int TOL_Y = 10;
const int TOL_ANG = 1;
const int ALIGN_TICK_MS = 40;
const int ALIGN_WAIT_MS = 300;

const int ENTRY_DELAY_FWD = 200;
const int ENTRY_DELAY_BWD = 200;
const int IGNORE_PART_MS  = 700;

int errX = 0, errY = 0, errAngle = 0;
int alignSuccessCount = 0;

MPU9250 mpu;
float targetYaw = 0.0;
float currentYaw = 0.0;
float gyroBiasZ = 0.0;
const float Kp = 2.5;
const float Kp_X = 0.15;
unsigned long lastTime = 0;

enum State {  IDLE, ALIGNING, WAIT_PATH_RECOVERY, GO_TO_BOX, AT_BOX_WAIT, RETURN_TO_HOME, RECOVERING,
```

```
CANCEL_REVERSING };
State state = IDLE;
State nextStateAfterAlign = IDLE;

unsigned long stateEnterMs = 0;
unsigned long ignorePartTimer = 0;
unsigned long lastNotify = 0;
bool isReturning = false;

Pos pos = {3, 1};
Pos currentPath[30];
int pathLen = 0;
int pathIdx = 0;
String rxLine = "";
int currentAction = -1;
int cancelRevAction = 0;

int targetBoxId = 1;
bool needToStopAtA = false;
bool needToStopAtB = false;

const int GRID_X = 4;
const int GRID_Y = 3;
bool blocked[GRID_X][GRID_Y] = {false};

bool tempBlocked[GRID_X][GRID_Y] = {false};
bool mapUpdatePending = false;
unsigned long lastMapMsgMs = 0;

String lastDisplayState = "";

void uartReadLines();
void handleLine(const String& line);

String getDisplayState() {
  switch (state) {
    case IDLE: return "대기중";
    case WAIT_PATH_RECOVERY: return "경로 대기중";
    case ALIGNING:
    case RECOVERING:
    case CANCEL_REVERSING: return "보정중";
    case GO_TO_BOX:
    case RETURN_TO_HOME: return "이동중";
```

```
    case AT_BOX_WAIT: return "하역중";
    default: return "알수없음";
  }
}

void mpuUpdateRoutine() {
  if (mpu.update()) {
    unsigned long currentTime = micros();
    float dt = (currentTime - lastTime) / 1000000.0;
    lastTime = currentTime;
    if (dt > 0.1) dt = 0.01;
    float gz = mpu.getGyroZ() - gyroBiasZ;
    if (abs(gz) > 0.5) { currentYaw += (gz * dt); }
    if (currentYaw > 180) currentYaw -= 360;
    if (currentYaw < -180) currentYaw += 360;
  }
}

void smartDelay(unsigned long ms) {
  unsigned long start = millis();
  while (millis() - start < ms) { mpuUpdateRoutine(); }
}

void calibrateGyro() {
  float sumGz = 0;
  for(int i=0; i<300; i++) { if(mpu.update()) sumGz += mpu.getGyroZ(); delay(10); }
  gyroBiasZ = sumGz / 300.0; currentYaw = 0.0; targetYaw = 0.0; lastTime = micros();
}

void setMotor(int pwmPin, int dirPin, int speed) {
  speed = constrain(speed, -255, 255);
  if (speed >= 0) { digitalWrite(dirPin, HIGH); analogWrite(pwmPin, speed); }
  else { digitalWrite(dirPin, LOW); analogWrite(pwmPin, -speed); }
}

void moveMecanum(int s1, int s2, int s3, int s4) {
  setMotor(M1_PWM, M1_DIR, s1); setMotor(M2_PWM, M2_DIR, s2);
  setMotor(M3_PWM, M3_DIR, s3); setMotor(M4_PWM, M4_DIR, s4);
}

void stopDrive() { moveMecanum(0, 0, 0, 0); currentAction = 0; }

void waitForButton() {
```

```
  stopDrive();
  Serial2.println("STATE,적재중");
  lastDisplayState = "적재중";
  rxLine = "";
  while(Serial2.available()) Serial2.read();
  Serial2.println("EVT,READY_TO_RETURN");

  while (true) {
    mpuUpdateRoutine();
    uartReadLines();
    if (rxLine == "BTN,303") { rxLine = ""; return; }
    if (rxLine == "CMD,STOP" || rxLine == "CMD,CANCEL") { return; }

    if (mapUpdatePending && (millis() - lastMapMsgMs > 150)) {
      for(int x=0; x<GRID_X; x++) for(int y=0; y<GRID_Y; y++) blocked[x][y] = tempBlocked[x][y];
      mapUpdatePending = false;
    }
    delay(10);
  }
}

void driveTo(Pos target) {
  int dx = target.x - pos.x; int dy = target.y - pos.y;
  float angleError = targetYaw - currentYaw;
  if (angleError > 180) angleError -= 360;
  if (angleError < -180) angleError += 360;
  float gyroCorrection = angleError * -Kp;
  gyroCorrection = constrain(gyroCorrection, -40, 40);
  float xCorrection = 0;

  if (abs(errX) > 10 && (state == GO_TO_BOX || state == RETURN_TO_HOME)) { xCorrection = errX * Kp_X; }

  if (dx < 0) {
    if (currentAction != 1) { stopDrive(); smartDelay(200); targetYaw = 0.0; currentAction = 1; }
    moveMecanum(PWM_STRAIGHT + gyroCorrection - xCorrection, PWM_STRAIGHT - gyroCorrection +
xCorrection + FWD_LEFT_TRIM,
                PWM_STRAIGHT + gyroCorrection - xCorrection, PWM_STRAIGHT - gyroCorrection +
xCorrection + FWD_LEFT_TRIM);
  }
  else if (dx > 0) {
    if (currentAction != 2) { stopDrive(); smartDelay(200); targetYaw = 0.0; currentAction = 2; }
    int base = -PWM_STRAIGHT;
    moveMecanum(base + gyroCorrection + xCorrection, base - gyroCorrection - xCorrection -
```

```
BWD_LEFT_TRIM,
               base + gyroCorrection + xCorrection, base - gyroCorrection - xCorrection -
BWD_LEFT_TRIM);
  }
  else if (dy > 0) {
    if (currentAction != 4) { stopDrive(); smartDelay(150); currentAction = 4; }
    moveMecanum(-PWM_STRAFE, PWM_STRAFE, PWM_STRAFE, -PWM_STRAFE);
  }
  else if (dy < 0) {
    if (currentAction != 3) { stopDrive(); smartDelay(150); currentAction = 3; }
    moveMecanum(PWM_STRAFE, -PWM_STRAFE, -PWM_STRAFE, PWM_STRAFE);
  }
  else { stopDrive(); }
}

bool isRowClear(int y) {
  for (int x = 0; x <= 3; x++) if (blocked[x][y]) return false;
  return true;
}

void clearBlockedMap() { for (int x = 0; x < GRID_X; x++) for (int y = 0; y < GRID_Y; y++)
blocked[x][y] = false; }

void addSafeSegment(int targetX, int targetY) {
  int currX = (pathLen == 0) ? pos.x : currentPath[pathLen - 1].x;
  int currY = (pathLen == 0) ? pos.y : currentPath[pathLen - 1].y;

  if (currX == 0 || currX == 3) {
    while (currY != targetY) { currY += (targetY > currY) ? 1 : -1; currentPath[pathLen++] = {currX,
currY}; }
  }
  while (currX != targetX) { currX += (targetX > currX) ? 1 : -1; currentPath[pathLen++] = {currX,
currY}; }
  while (currY != targetY) { currY += (targetY > currY) ? 1 : -1; currentPath[pathLen++] = {currX,
currY}; }
}

bool generateRoute() {
  pathLen = 0;
  int targetY = (targetBoxId == 1) ? 0 : ((targetBoxId == 2) ? 1 : 2);

  int bestRow = -1;
  int minDiff = 999;
```

```
  for(int y=0; y<=2; y++) {
    if(isRowClear(y)) {
      if(abs(y - targetY) < minDiff) { minDiff = abs(y - targetY); bestRow = y; }
    }
  }
  if (bestRow == -1) return false;

  if (needToStopAtA && needToStopAtB) {
    int costA_then_B = abs(pos.y - 0) + 2 + abs(2 - bestRow);
    int costB_then_A = abs(pos.y - 2) + 2 + abs(0 - bestRow);

    if (costA_then_B <= costB_then_A) {
      addSafeSegment(3, 0);
      addSafeSegment(3, 2);
    } else {
      addSafeSegment(3, 2);
      addSafeSegment(3, 0);
    }
  }
  else if (needToStopAtA) { addSafeSegment(3, 0); }
  else if (needToStopAtB) { addSafeSegment(3, 2); }

  addSafeSegment(3, bestRow);
  addSafeSegment(0, bestRow);
  addSafeSegment(0, targetY);
  return true;
}

bool generateReturnRoute() {
  pathLen = 0;
  int bestRow = -1;
  int minDiff = 999;
  for(int y=0; y<=2; y++) {
    if(isRowClear(y)) { if(abs(y - 1) < minDiff) { minDiff = abs(y - 1); bestRow = y; } }
  }
  if (bestRow == -1) return false;

  addSafeSegment(0, bestRow);
  addSafeSegment(3, bestRow);
  addSafeSegment(3, 1);
  return true;
}
```

```
void uartReadLines() {
  while (Serial2.available()) {
    char c = (char)Serial2.read();
    if (c == '\n') {
      rxLine.trim();
      if (rxLine.length() > 0) {
        if (rxLine != "BTN,303") { handleLine(rxLine); rxLine = ""; }
      }
    }
    else if (c != '\r') rxLine += c;
  }
}

void commitMapUpdate() {
  if (mapUpdatePending && (millis() - lastMapMsgMs > 400)) {
    for(int x=0; x<GRID_X; x++) {
      for(int y=0; y<GRID_Y; y++) { blocked[x][y] = tempBlocked[x][y]; }
    }
    mapUpdatePending = false;
  }
}

void handleLine(const String& line) {
  if (line == "CMD,STOP") {
    stopDrive();
    myServo.write(0);
    state = IDLE;
    currentAction = 0;
    pathLen = 0;
    pathIdx = 0;
    isReturning = false;
    return;
  }

  if (line == "CMD,CANCEL") {
    myServo.write(0);
    if (state == IDLE) return;

    if (state == GO_TO_BOX || state == RETURN_TO_HOME) {
      cancelRevAction = currentAction;
      state = CANCEL_REVERSING;
    } else {
      stopDrive();
```

```
      if (generateReturnRoute()) {
        pathIdx = 0; state = RETURN_TO_HOME; isReturning = true;
      } else {
        state = IDLE;
      }
    }
    return;
  }

  if (line.startsWith("POS,")) {
    int c[5]; int lastPos = 0;
    for(int i=0; i<5; i++) { c[i] = line.indexOf(',', lastPos + 1); lastPos = c[i]; }
    if (c[1] > 0) {
      pos.x = line.substring(c[0] + 1, c[1]).toInt(); pos.y = line.substring(c[1] + 1, c[2]).toInt();
      errX = line.substring(c[2] + 1, c[3]).toInt(); errY = line.substring(c[3] + 1, c[4]).toInt();
errAngle = line.substring(c[4] + 1).toInt();
    }

    if (state == CANCEL_REVERSING) {
      stopDrive();
      state = ALIGNING;
      alignSuccessCount = 0;
      if (generateReturnRoute()) {
        pathIdx = 0; isReturning = true;
        nextStateAfterAlign = RETURN_TO_HOME;
      } else {
        nextStateAfterAlign = IDLE;
      }
    }
    return;
  }

  if (line.startsWith("PART,")) {
    if (state == GO_TO_BOX || state == RETURN_TO_HOME) return;

    int c1 = line.indexOf(','); int c2 = line.indexOf(',', c1 + 1);
    if (c1 > 0 && c2 > 0) {
      errX = line.substring(c1 + 1, c2).toInt();
      errY = line.substring(c2 + 1).toInt();
      errAngle = 0;
    }

    if (state == CANCEL_REVERSING) {
```

```
      stopDrive();
      state = ALIGNING;
      alignSuccessCount = 0;
      if (generateReturnRoute()) {
        pathIdx = 0; isReturning = true;
        nextStateAfterAlign = RETURN_TO_HOME;
      } else {
        nextStateAfterAlign = IDLE;
      }
    }
    return;
  }

  if (line == "CLEAR_MAP") {
    if (mapUpdatePending) {
        for(int x=0; x<GRID_X; x++) for(int y=0; y<GRID_Y; y++) blocked[x][y] = tempBlocked[x][y];
    }
    for(int x=0; x<GRID_X; x++) for(int y=0; y<GRID_Y; y++) tempBlocked[x][y] = false;
    mapUpdatePending = true;
    lastMapMsgMs = millis();
    return;
  }

  if (line.startsWith("BLOCK,")) {
    int c1 = line.indexOf(','); int c2 = line.indexOf(',', c1 + 1);
    if (c1 > 0 && c2 > 0) {
        int bx = line.substring(c1 + 1, c2).toInt();
        int by = line.substring(c2 + 1).toInt();
        if (bx >= 0 && bx < GRID_X && by >= 0 && by < GRID_Y) {
            tempBlocked[bx][by] = true;
            mapUpdatePending = true;
            lastMapMsgMs = millis();
        }
    }
    return;
  }

  if (line.startsWith("ORD,") && state == IDLE) {
    int c1 = line.indexOf(','); int c2 = line.indexOf(',', c1 + 1); int c3 = line.indexOf(',', c2 + 1);
    targetBoxId = line.substring(c1 + 1, c2).toInt();
    needToStopAtA = (line.substring(c2 + 1, c3).toInt() > 0); needToStopAtB = (line.substring(c3 +
1).toInt() > 0);
    isReturning = false;
```

```
    if(!generateRoute()) {
      stopDrive();
      Serial2.println("EVT,START_BLOCKED");
      return;
    }

    pathIdx = 0;

    bool atA = (pos.x == 3 && pos.y == 0); bool atB = (pos.x == 3 && pos.y == 2);
    if ((atA && needToStopAtA) || (atB && needToStopAtB)) {
      if (atA && needToStopAtA) needToStopAtA = false; if (atB && needToStopAtB) needToStopAtB = false;
      waitForButton();
      if (state == IDLE || state == CANCEL_REVERSING) return;

      stopDrive();
      state = WAIT_PATH_RECOVERY;
      return;
    }

    state = ALIGNING;
    nextStateAfterAlign = GO_TO_BOX;
    alignSuccessCount = 0;
  }
}

void updateStateMachine() {
  switch (state) {
    case IDLE: stopDrive(); break;

    case WAIT_PATH_RECOVERY: {
      if (!generateRoute()) {
        stopDrive();
        if (millis() - lastNotify > 2000) {
          Serial2.println("EVT,PATH_BLOCKED");
          lastNotify = millis();
        }
      } else {
        pathIdx = 0;
        state = ALIGNING;
        nextStateAfterAlign = isReturning ? RETURN_TO_HOME : GO_TO_BOX;
        alignSuccessCount = 0;
      }
```

```
      break;
    }

    case CANCEL_REVERSING: {
      float angleError = targetYaw - currentYaw;
      if (angleError > 180) angleError -= 360;
      if (angleError < -180) angleError += 360;
      float gyroCorrection = angleError * -Kp;
      gyroCorrection = constrain(gyroCorrection, -40, 40);

      if (cancelRevAction == 1) {
        int base = -PWM_STRAIGHT;
        moveMecanum(base + gyroCorrection, base - gyroCorrection, base + gyroCorrection, base - gyroCorrection);
      } else if (cancelRevAction == 2) {
        moveMecanum(PWM_STRAIGHT + gyroCorrection, PWM_STRAIGHT - gyroCorrection, PWM_STRAIGHT + gyroCorrection, PWM_STRAIGHT - gyroCorrection);
      } else if (cancelRevAction == 3) {
        moveMecanum(PWM_STRAFE, -PWM_STRAFE, -PWM_STRAFE, PWM_STRAFE);
      } else if (cancelRevAction == 4) {
        moveMecanum(-PWM_STRAFE, PWM_STRAFE, PWM_STRAFE, -PWM_STRAFE);
      } else {
        stopDrive();
      }
      break;
    }
    case ALIGNING: {
      if (abs(errAngle) > TOL_ANG) {
        if (errAngle > 0) moveMecanum(ALIGN_PWM_ANG, -ALIGN_PWM_ANG, ALIGN_PWM_ANG, -ALIGN_PWM_ANG);
        else moveMecanum(-ALIGN_PWM_ANG, ALIGN_PWM_ANG, -ALIGN_PWM_ANG, ALIGN_PWM_ANG);
        smartDelay(ALIGN_TICK_MS); stopDrive(); smartDelay(ALIGN_WAIT_MS); alignSuccessCount = 0;
      }
      else if (abs(errX) > TOL_X || abs(errY) > TOL_Y) {
        if (abs(errX) > TOL_X) { if (errX > 0) moveMecanum(-ALIGN_PWM_X, ALIGN_PWM_X, ALIGN_PWM_X, -ALIGN_PWM_X); else moveMecanum(ALIGN_PWM_X, -ALIGN_PWM_X, -ALIGN_PWM_X, ALIGN_PWM_X); }
        else if (abs(errY) > TOL_Y) { if (errY > 0) moveMecanum(ALIGN_PWM_Y, ALIGN_PWM_Y, ALIGN_PWM_Y, ALIGN_PWM_Y); else moveMecanum(-ALIGN_PWM_Y, -ALIGN_PWM_Y, -ALIGN_PWM_Y, -ALIGN_PWM_Y); }
        smartDelay(ALIGN_TICK_MS); stopDrive(); smartDelay(ALIGN_WAIT_MS); alignSuccessCount = 0;
      }
      else {
        alignSuccessCount++;
        if (alignSuccessCount >= 4) {
          stopDrive(); currentYaw = (float)errAngle; targetYaw = 0.0;
```

```
          if (nextStateAfterAlign == AT_BOX_WAIT) {
             Serial2.println("CMD,LOAD_START");
             stateEnterMs = millis();
             myServo.write(40);
          }
          else if (nextStateAfterAlign == IDLE && isReturning) {
             Serial2.println("EVT,JOB_DONE");
             isReturning = false;
          }

          state = nextStateAfterAlign;
          ignorePartTimer = millis() + IGNORE_PART_MS;
        } else { smartDelay(100); }
      }
      break;
    }
    case RECOVERING: {
      if (abs(errX) > TOL_X || abs(errY) > TOL_Y) {
        if (abs(errX) > TOL_X) { if (errX > 0) moveMecanum(-ALIGN_PWM_X, ALIGN_PWM_X, ALIGN_PWM_X,
-ALIGN_PWM_X); else moveMecanum(ALIGN_PWM_X, -ALIGN_PWM_X, -ALIGN_PWM_X, ALIGN_PWM_X); }
        else if (abs(errY) > TOL_Y) { if (errY > 0) moveMecanum(ALIGN_PWM_Y, ALIGN_PWM_Y, ALIGN_PWM_Y,
ALIGN_PWM_Y); else moveMecanum(-ALIGN_PWM_Y, -ALIGN_PWM_Y, -ALIGN_PWM_Y, -ALIGN_PWM_Y); }
        smartDelay(ALIGN_TICK_MS); stopDrive(); smartDelay(ALIGN_WAIT_MS);
      } else { stopDrive(); state = ALIGNING; alignSuccessCount = 0; }
      break;
    }
    case GO_TO_BOX: {
      if (pathIdx < pathLen) {
        Pos target = currentPath[pathIdx];
        if (pos.x == target.x && pos.y == target.y) {
          bool isFinalDest = (pathIdx == pathLen - 1);

          bool isCorner = false;
          if (!isFinalDest) {
            Pos nextTarget = currentPath[pathIdx + 1];
            if ((currentAction == 1 || currentAction == 2) && nextTarget.y != pos.y) isCorner = true;
            if ((currentAction == 3 || currentAction == 4) && nextTarget.x != pos.x) isCorner = true;
          }

          bool atA = (pos.x == 3 && pos.y == 0); bool atB = (pos.x == 3 && pos.y == 2);
          if ((atA && needToStopAtA) || (atB && needToStopAtB)) {
            if (atA) needToStopAtA = false; if (atB) needToStopAtB = false;
```

```
        waitForButton();
        if (state == IDLE || state == CANCEL_REVERSING) break;

        stopDrive();
        state = WAIT_PATH_RECOVERY;
      }
      else if (isFinalDest) {
        stopDrive();
        state = ALIGNING;
        nextStateAfterAlign = AT_BOX_WAIT;
        pathIdx++;
        alignSuccessCount = 0;
      }
      else if (isCorner) {
        stopDrive();
        state = ALIGNING;
        nextStateAfterAlign = GO_TO_BOX;
        pathIdx++;
        alignSuccessCount = 0;
      }
      else { pathIdx++; ignorePartTimer = millis() + IGNORE_PART_MS; }
    } else { driveTo(target); }
  } break;
}
case AT_BOX_WAIT: {
  if (millis() - stateEnterMs >= 3500) {
    if (myServo.read() != 0) {
        myServo.write(0);
        smartDelay(1500);
    }

    if(!generateReturnRoute()) {
        stopDrive();
        if (millis() - lastNotify > 2000) {
            Serial2.println("EVT,PATH_BLOCKED");
            lastNotify = millis();
        }
    }
    else {
        pathIdx = 0;
        state = RETURN_TO_HOME;
        isReturning = true;
        alignSuccessCount = 0;
```

```
        }
      } break;
    }
    case RETURN_TO_HOME: {
      if (pathIdx < pathLen) {
        Pos target = currentPath[pathIdx];
        if (pos.x == target.x && pos.y == target.y) {
          bool isFinalDest = (pathIdx == pathLen - 1);
          bool isCorner = false;
          if (!isFinalDest) {
            Pos nextTarget = currentPath[pathIdx + 1];
            if ((currentAction == 1 || currentAction == 2) && nextTarget.y != pos.y) isCorner = true;
            if ((currentAction == 3 || currentAction == 4) && nextTarget.x != pos.x) isCorner = true;
          }
          if (isFinalDest) {
            stopDrive();
            state = ALIGNING;
            nextStateAfterAlign = IDLE;
            pathIdx++;
            alignSuccessCount = 0;
          }
          else if (isCorner) {
            stopDrive();
            state = ALIGNING;
            nextStateAfterAlign = RETURN_TO_HOME;
            pathIdx++;
            alignSuccessCount = 0;
          }
          else { pathIdx++; ignorePartTimer = millis() + IGNORE_PART_MS; }
        } else { driveTo(target); }
      } break;
    }
  }
}

void setup() {
  Serial.begin(115200); Serial2.begin(115200, SERIAL_8N1, RXp2, TXp2); Wire.begin(27, 14);
  myServo.attach(SERVO_PIN); myServo.write(0); delay(2000);
  pinMode(M1_PWM, OUTPUT); pinMode(M1_DIR, OUTPUT); pinMode(M2_PWM, OUTPUT); pinMode(M2_DIR, OUTPUT);
  pinMode(M3_PWM, OUTPUT); pinMode(M3_DIR, OUTPUT); pinMode(M4_PWM, OUTPUT); pinMode(M4_DIR, OUTPUT);
  stopDrive(); if (!mpu.setup(0x68)) { while (1) { delay(500); } } delay(1000); calibrateGyro();
}
```

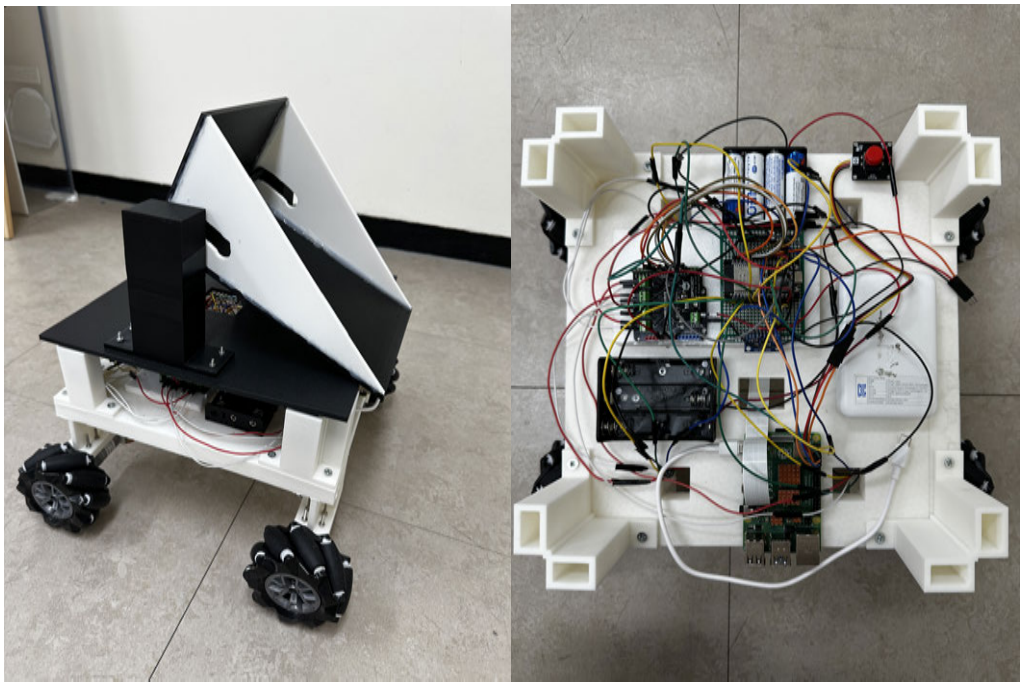
```
void loop() {
  mpuUpdateRoutine();
  uartReadLines();
  commitMapUpdate();
  updateStateMachine();

  String currentDisplayState = getDisplayState();
  if (currentDisplayState != lastDisplayState) {
    Serial2.println("STATE," + currentDisplayState);
    lastDisplayState = currentDisplayState;
  }
}
```

VII. 결과물 설명 (결과물 관련 이미지(사진) 첨부)

[인프라 감시 모듈 (Raspberry Pi A)]

- 인프라 감시 모듈은 작업 환경 전체를 조망할 수 있도록 고안된 수직 삼각대 상단에 배치된다. 이는 천장 카메라의 역할을 수행하며, 넓은 화각을 확보하여 4x3 격자 맵 전체의 동적 장애물 유무를 사각지대 없이 판별하기 위한 설계이다. 삼각대 상단의 Raspberry Pi A는 OpenCV 기반의 배경 차분 알고리즘을 구동하며, 감지된 장애물 정보를 MQTT 프로토콜을 통해 중앙 관제 서버로 실시간 전송하는 인프라 망의 핵심 노드 역할을 수행한다.

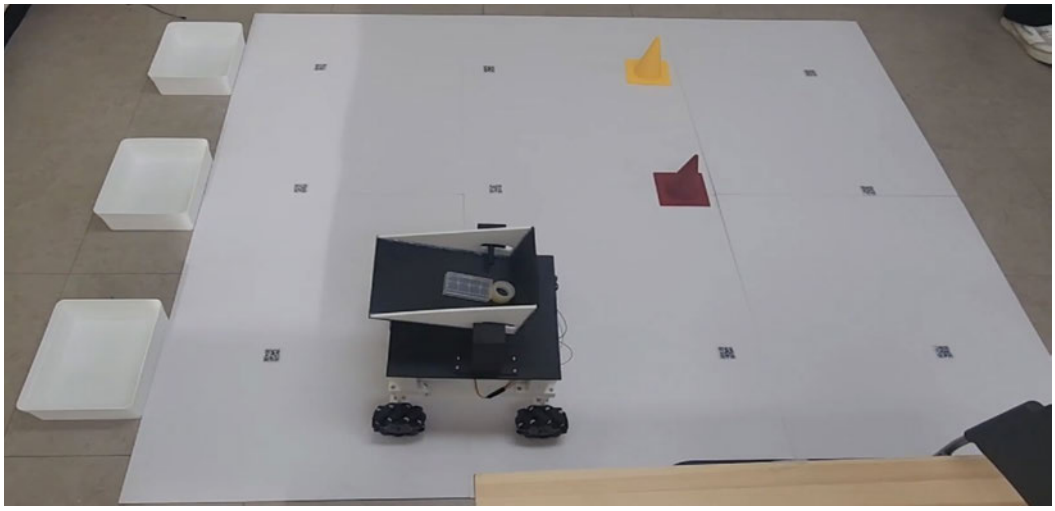
[로봇 (Raspberry Pi B),(ESP32)]

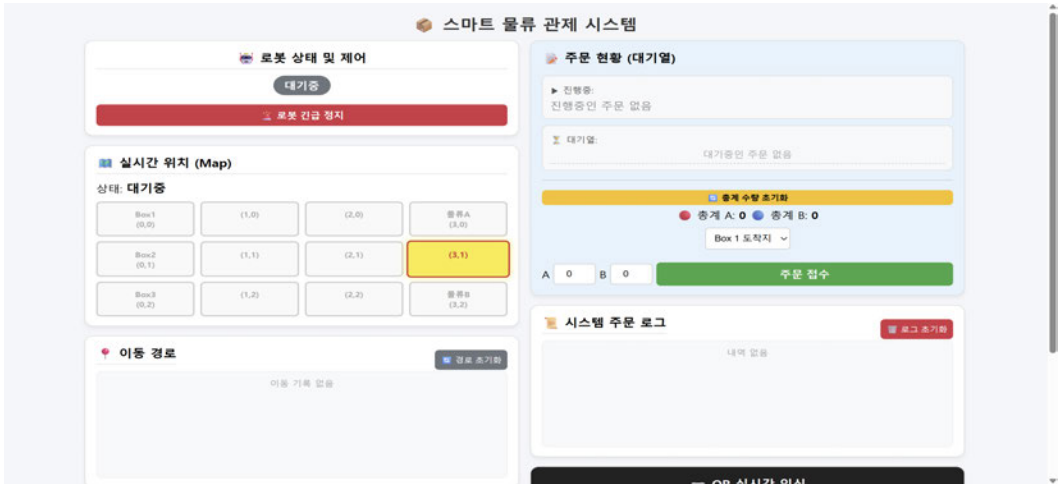


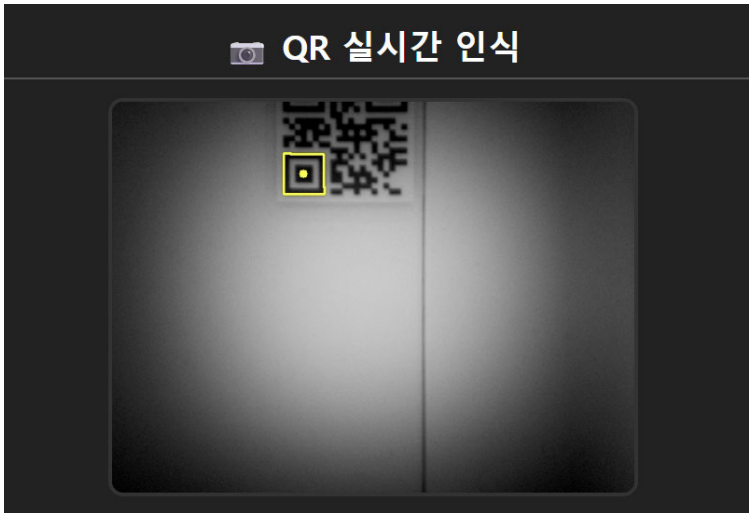
- 로봇 본체는 기동성과 적재 효율을 극대화하도록 설계되었다. 구동부에는 4개의 메카넘 휠을 장착하여 전방향 이동 및 제자리 회전이 가능하도록 하였으며, 상단부에는 서보 모터와 연동된 경사형 적재함을 설치하여 목적지 도달 시 자동으로 물품을 하역할 수 있는 구조를 취한다.

로봇 내부의 제어부는 상위 연산을 담당하는 Raspberry Pi B와 하위 구동을 담당하는 ESP32의 결합으로 구성된다. Raspberry Pi B는 비전 인식 및 웹 서버 운영을 수행하며, ESP32는 DRI0039 모터 드라이버와 연결되어 메카넘 휠의 PWM 제어 및 IMU 기반 자세 보정 알고리즘을 실행한다. 배선 설계 시 각 모듈 간의 간섭을 최소화하도록 배치하였으며, 외부 전원 공급을 위한 배터리 팩과 보조배터리를 바닥 양옆에 배치하여 무게 중심을 맞춤으로써 주행 안정성을 확보하였다. 또한, 배선 이후 납땜하여 최종 하드웨어를 완성하였다.

[전체 작업 환경 및 웹 화면]




스마트 물류 관제 시스템
로봇 상태 및 제어
대기중
로봇 긴급 정지
주문 현황 (대기열)
진행중:
진행중인 주문 없음
대기열:
대기중인 주문 없음
총계 수량 초기화
총계 A: 0 총계 B: 0
Box 1 도착지
A 0 B 0
주문 접수
실시간 위치 (Map)
상태: 대기중
Box1 (0,0)
(1,0)
(2,0)
물류A (3,0)
Box2 (0,1)
(1,1)
(2,1)
(3,1)
Box3 (0,2)
(1,2)
(2,2)
물류B (3,2)
이동 경로
경로 초기화
이동 기록 없음
시스템 주문 로그
로그 초기화
내역 없음


QR 실시간 인식

- 작업 환경은 로봇의 측위 및 주행을 위한 격자형 맵으로 구성된다. 바닥면에는 각 좌표를 식별할 수 있는 QR 코드가 부착되어 있으며, 이는 로봇의 하향 카메라와 연동되어 절대 위치 좌표 및 정밀 오차 보정 데이터를 생성하는 지표가 된다. 사진 내의 원뿔형 오브젝트는 동적 장애물을 모사하며, 인프라 감시 모듈에 의해 감지되어 시스템 맵에 실시간으로 반영된다.

동시에 제공되는 웹 기반 관제 시스템 UI는 Flask 서버를 통해 구현되었으며, 로봇의 현재 상태(대기중, 이동중, 보정중 등), 실시간 위치 정보, 이동 경로 로그 및 주문 대기열을 직관적으로 시각화한다. 사용자는 해당 인터페이스를 통해 배송 주문을 접수하거나 로봇의 주행 상태를 실시간으로 모니터링하며, 긴급 상황 시 비상 정지 명령을 하달할 수 있는 통합 관제 환경을 갖추고 있다.

VIII. 프로젝트 수행 결과 분석

1. 재학중 취득한 기초지식의 활용 내용

전공 교과 과정에서 학습한 프로그래밍 언어 과목에서 학습한 C언어를 바탕으로 ESP32 마이크로컨트롤러의 펌웨어 개발을 진행하였다. 회로이론 과목으로는 전압, 전류, 저항의 관계를 바탕으로 DRI0039 모터 드라이버와 4개의 메카넘 휠 모터 간의 전원 계통을 설계하였으며, 전력 소모를 고려하여 전압 강하를 최소화하는 배선을 수행하였다. 제어공학 이론을 바탕으로는, MPU-9250 센서의 자이로 데이터를 분석하여 로봇의 방향 오차를 보정하는 P-제어(비례 제어) 로직을 설계하고 구현하였다. 데이터통신 및 컴퓨터네트워크 과목으로는 네트워크 모델에 대한 이해를 바탕으로 TCP/IP 기반의 MQTT 통신망을 구축하여 인프라 카메라와 중앙 서버 간의 데이터를 중계하였다. 디지털 영상처리 과목에서는 기초 이론을 바탕으로 인프라 감시 카메라의 장애물 인식 알고리즘을 설계하였다. 가우시안 혼합 모델(MOG2)을 이용한 배경 차분(Background Subtraction) 기법을 적용하여 동적 장애물을 분리하였으며, 모폴로지 연산(Morphology)과 윤곽선 추출(Contour Detection) 및 면적 필터링 기술을 활용하여 영상 내 노이즈를 제거하고 유효한 객체만을 식별하는 로직을 구현하였다. 또한 효율적인 연산을 위해 특정 영역만을 분석하는 ROI(Region of

Interest) 설정 기법을 적용하여 시스템의 실시간성을 높였다.

2. 재학중 취득한 실험지식의 활용 내용

전공 교과 과정에서 학습한 전기전자회로 실험 과목에서는 회로 구성 및 배선 실습 경험을 활용하여 DRI0039 모터 드라이버, ESP32, Raspberry Pi 등 다양한 모듈 간의 전원 및 신호 선을 체계적으로 배치하였다. 특히 모터 구동 시 발생하는 노이즈와 전압 강하를 최소화하기 위해 전력선의 굵기를 고려한 배선과 안정적인 납땜 처리를 수행하여 시스템의 전기적 신뢰성을 확보하였다. 마이크로프로세서 실험 과목에서는 직렬 통신 프로토콜의 이론과 실습 지식을 바탕으로 Raspberry Pi와 ESP32 간의 데이터 교환 체계를 구축하였다. 두 장치 간의 통신 속도를 115200bps로 설정하고, 상위 계층의 이동 명령(ORD) 및 좌표 오차(POS) 데이터를 하위 계층으로 손실 없이 전송하기 위한 데이터 패킷 구조화 기술을 적용하였다. 또한 임베디드 시스템 과목에서는 리눅스 운영체제 기반의 임베디드 환경 활용 능력을 바탕으로 로봇의 메인 두뇌인 Raspberry Pi에 Flask 웹 서버와 비디오 스트리밍 환경을 구축하였다. Picamera2 라이브러리를 이용한 하향 카메라 제어와 비전 처리 알고리즘을 실장하였으며, 인프라 카메라의 데이터를 수신하기 위한 MQTT 클라이언트 환경을 조성함으로써 임베디드 시스템 중심의 통합 제어를 실현하였다.

3. 본 프로젝트 수행과정에서의 설계 능력 향상 내용

상위 관제 계층, 인프라 인식 계층, 하위 물리 구동 계층으로 구분된 3-Tier 아키텍처를 직접 설계하며 복합 시스템 통합 능력을 배양하였다. 로봇의 자율 주행 시나리오를 체계적으로 관리하기 위해 유한 상태 머신(Finite State Machine) 기반의 제어 구조를 설계하였으며, 이는 예외 상황 발생 시 로봇의 행동을 논리적으로 정의하는 계기가 되었다. 또한, 격자형 맵(Grid Map) 환경에서 최단 경로를 산출하기 위한 비용 기반 라우팅 알고리즘을 설계하고 적용함으로써 소프트웨어적 문제 해결 역량을 강화하였다.

4. 본 프로젝트 수행과정에서의 문제 해결 내용

개발 과정 중 발생한 통신 병목 및 맵 업데이트 충돌 문제를 해결하기 위해 이벤트 기반 동기화 로직을 도입하였다. 주행 중 상시 맵을 업데이트할 경우 발생하는 제어 지연을 방지하고자 주문 수락, 적재 완료, 하역 직후 등 특정 정지 이벤트 시점에만 업데이트 락(Lock)을 해제하여 정보를 수신하도록 개선하였다. 또한, 모든 경로가 폐쇄되었을 때 시스템이 교착 상태에 빠지는 문제를 해결하기 위해 '경로 재탐색 대기(WAIT_PATH_RECOVERY)' 상태를 추가하여 경로 확보 시 주행을 자동 재개하도록 로직을 고도화하였다. 또한 초기 설계 단계에서는 일반적인 이륜 구동 구조를 계획하였으나, 협소한 격자형 물류 환경에서 제자리 회전 및 방향 전환 시 발생하는 주행 부정확성이 설계 목표 달성의 저해 요소로 확인되었다. 이를 해결하기 위해 전후좌우 이동 및 즉각적인 방향 전환이 가능한 메카넘 휠 시스템으로 구동부를 교체하여 기동성을 확보하고 주행 오차를 최소화하였다. 그러나 메카넘 휠은 지면과의 마찰력 및 평탄도에 매우 민감하여, 일반적인 바닥면에서는 불규칙한 미끄러짐 현상으로 인해 정밀한 위치 보정이 어렵다는 문제가 발생하였다. 주행 환경의 변수를 통제하고 일관된 주행 성능을 확보하기 위하여, 균일한 표면 특성을 가진 포맥스(Formex) 판을 구매하여 전체 작업 평면의 바닥재로 사용하였다. 이를 통해 휠의 접지력을 일정하게 유지하고 하향 카메라의 QR 코드 인식 환경을 최적화함으로써 시스템의 완성도를 높였다.

5. 본 프로젝트 수행과정에서의 실무 능력 향상 내용

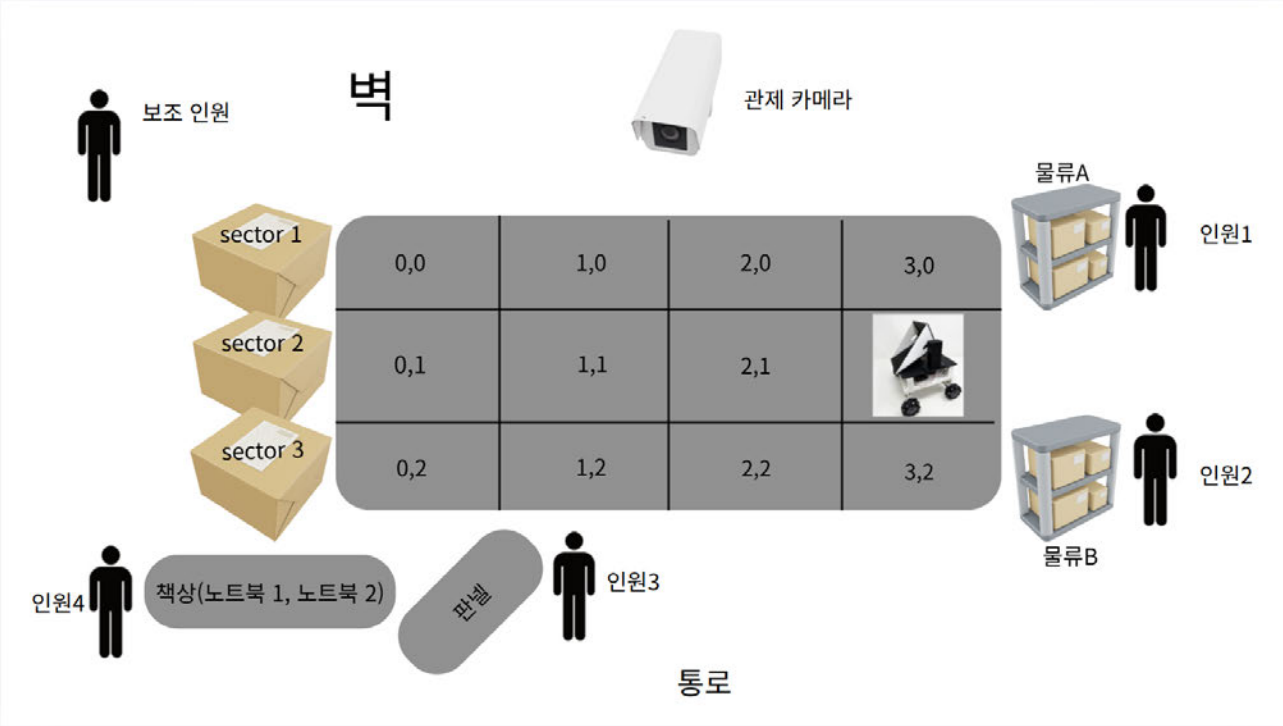
산업 현장에서 활용되는 MQTT, UART 등 다양한 통신 프로토콜을 실무적으로 적용하며 네트워크 프로토콜에 대한 이해도를 높였다. Flask 프레임워크를 이용해 실시간 비디오 스트리밍과 맵 시각화가 가능한 관제 서버를 구축함으로써 풀스택 제어 시스템 개발 능력을 확보하였다. OpenCV 라이브러리를 활용한 배경 차분(MOG2) 및 QR 코드 비전 처리 기법을 실제 환경에 적용하며 컴퓨터 비전 기반의 물체 감지 및 정밀 위치 측위 기술을 습득하였다. 또한 하드웨어 측면에서는 Raspberry Pi와 ESP32라는 이기종 MCU를 결합하여 상·하위 제어 체계를 구축함으로써 복합 시스템 통합 설계 능력을 확보하였다. 저전력 고효율의 DRI0039 모터 드라이버를 실장하여 4개의 메카넘 휠을 독립적으로 제어하는 전방향 구동 회로를 성공적으로 구현하였

다. 서보 모터를 활용한 경사형 적재함의 하역 메커니즘을 설계하고 이를 소프트웨어 상태 머신 로직과 연동함으로써 물리적인 액추에이터 제어 실무 역량을 강화하였다. 특히 메카넘 휠의 구동 특성을 고려하여 포맥스 바닥재를 시공하고 센서 및 카메라의 최적 배치안을 도출함으로써, 실제 구동 환경에서 발생할 수 있는 물리적 변수를 통제하고 하드웨어의 신뢰성을 높이는 실무적 노하우를 습득하였다.

6. 본 프로젝트 수행과정에서의 팀원간 협동 내용

본 프로젝트를 수행하는 과정에서 5명의 인원이 각자의 전문 영역을 분담함과 동시에 유기적으로 협업하는 구조로 운영되었다. 2명의 팀원은 로봇의 내부 구조 설계 및 3D 프린터를 활용한 프레임 제작 등 하드웨어 전반을 담당하였다. 로봇의 적재함 구조 설계와 부품의 최적 배치를 수행하여 물리적 안정성을 확보하는 데 주력하였다. 다른 2명의 팀원은 로봇의 기동을 위한 ESP32 펌웨어 개발과 Raspberry Pi 상위 소프트웨어 설계를 담당하였다. 메카넘 휠의 구동 알고리즘 구현, 자이로 센서 기반의 P-제어 로직 설계, 그리고 QR 코드 인식 비전 프로세스 개발을 수행하였다. 1명의 팀원은 프로젝트 전반의 기술 통합 및 진척 관리를 수행함과 동시에, 천장에 설치되는 인프라 관제 카메라용 Raspberry Pi 시스템을 전담하였다. OpenCV를 활용한 장애물 감지 알고리즘을 개발하고 MQTT 통신을 통한 맵 데이터 중계 시스템을 구축하였다. 개발 공정 중 인력 보강이 시급하거나 기술적 병목 현상이 발생하는 경우, 담당 파트가 아니더라도 상호 지원을 아끼지 않는 팀워크를 발휘하였다. 특히 하드웨어 조립 및 전체 시스템 통합 테스트 단계에서는 전 팀원이 합심하여 현장에서 발생하는 변수들을 실시간으로 해결하며 프로젝트의 완성도를 높였다. 이러한 협업 경험은 복합적인 시스템 설계 환경에서 팀원 간 소통의 중요성과 유연한 역할 분배의 필요성을 체득하는 계기가 되었다.

7. 개발된 결과물에 대한 전시 방법 계획



인원 1, 2 : 물류 적재 및 로봇 이벤트 버튼 제어(상황에 따라 1명으로 운영)
인원 3 : 프로젝트 관련 설명
인원 4 : 프로젝트 체험 전담(노트북 화면 컨트롤 및 시스템 운용)
* 상황에 맞추어 보조 인원 투입 예정

< 참고 문헌 >

주백석・성영휘, 「메카넘휠 기반의 전방향 이동로봇 주행성능 평가」, 『한국생산제조학회지』, 2014.08

오인진・권건우・양현석, 「메카넘 휠 이동로봇의 바퀴 슬립을 고려한 위치 추정 연구」, 『한국군사과학기술학회지』, 2019.06

이강철, 「QR 코드를 이용한 실내 이동로봇의 위치인식 및 자율주행 기법에 관한 연구」, 『한국산업기술대학교 대학원』, 2017

김정유, 「AI・자동화 로봇 기술…쿠팡 '로켓배송' 핵심으로」, 『이데일리』, 2025.03.26.

(https://www.edaily.co.kr/News/Read?newsId=01646566642107256&mediaCodeNo=257).

< 종합설계 프로젝트 수행 후기 >

개인정보(ID) 이호성

이번 캡스톤디자인은 QR 기반 자율주행 물류 운송 로봇을 만들면서 지금까지 배운 전공지식을 실제로 적용해볼 수 있었던 좋은 경험이었습니다. 특히 이번 주제는 제어와 통신이 중심이 되었고, 로봇을 안정적으로 움직이기 위해 제어공학에서 배운 P제어를 직접 사용해볼 수 있었습니다. MPU 센서로 현재 상태를 받아 주행을 보정하면서 이론으로 배운 내용이 실제 동작에 어떻게 연결되는지 알 수 있었습니다.

또한 UART, MQTT, 웹서버 라우팅 등을 사용하면서 임베디드 시스템에서 데이터가 어떻게 주고받아지는지도 경험할 수 있었습니다. 경로 탐색에서는 A*와 같은 범용 알고리즘 대신, 4×3 그리드 구조에 맞춘 규칙 기반 휴리스틱 방식을 적용하였습니다. 장애물이 없는 행을 찾고 목적지와 가까운 행을 선택하는 방식이라 구조는 단순하지만, 제한된 자원에서는 효율적이라고 느꼈습니다. 또한 A와 B 지점의 방문 순서를 비용 계산으로 정해 불필요한 이동을 줄이려 했습니다. 주행 중 장애물이 생기면 멈춘 뒤 다시 경로를 생성하도록 하여 실제 환경 변화에도 대응할 수 있게 했습니다.

이번 프로젝트를 통해 제어, 통신, 알고리즘, 임베디드 구현이 따로 떨어진 지식이 아니라 하나의 시스템 안에서 함께 동작한다는 것을 느꼈습니다. 전체적으로 부족한 점도 있었지만, 직접 문제를 해결해가며 전공 지식을 한 단계 더 발전시킬 수 있었던 의미 있는 경험이었습니다.

개인정보(ID) 김은중

이번 전자 캡스톤 디자인 프로젝트는 지난 4년간 전자공학도로서 학습한 이론과 실습 지식들을 처음으로 하나의 완성된 실물 시스템으로 구현해 본 가장 뜻깊은 프로젝트였다. 나는 팀에서 강성민 학우와 함께 하드웨어 제작을 담당하여 전체적인 회로 배선과 납땜, 그리고 3D 프린팅을 활용한 기구부 및 프레임 설계를 맡아 진행하였다.

가장 기억에 남고 부분은 부품 정하기, 회로 배선과 납땜 과정이었다. 먼저 로봇에 어떤 RPM 값의 모터가 적합할 지, 로봇의 총 무게와 주행 속도 어떻게 정해야 할 지 등이 처음 마주한 난관이었다. 데이터시트와 동영상과 같은 다양한 자료를 조사하며 최종 부품들을 정할 수 있었다. 또한, 그렇게 정한 부품들을 전기적으로 연결하는 것도 쉬운 일이 아니었다. 브레드보드에서는 작동하던 회로가 만능 기판에 납땜했을 때는 오작동 할 수도 있기 때문이었다. 다행이도, 핀 하나하나 조심히 작업하며 멀티미터로 찍어본 덕분에 전원을 넣었을 때 시스템이 목표한 대로 정확히 구동하는 모습을

보며 뿌듯함을 느꼈다.

또한, 3D 프린터를 이용해 기체의 프레임과 부품 마운트를 직접 설계하고 출력하면서 기계적 구조와 하드웨어 배치의 중요성도 깨달았다. 단순히 외형을 만드는 것을 넘어 배터리팩과 모터의 무게 중심, 부품 간의 조립 여유 공간까지 고려해야 했기에 도면 수정과 재출력을 여러번 반복해야 했다. 이 과정을 통해 아무리 훌륭한 소프트웨어 알고리즘이라도 그것을 받쳐주는 물리적인 하드웨어 뼈대가 견고하지 않으면 시스템이 제대로 동작할 수 없다는 것을 배웠습니다. 결국 소프트웨어와 합쳐져 완성된 회로와 프레임이 최종적으로 하나의 주행 로봇을 이루었을 때 매우 큰 성취감을 느꼈다.

최종적으로, 이번 캡스톤 프로젝트의 경험은 앞으로 엔지니어로서 실무를 수행하고 문제를 해결해 나가는 데 있어 튼튼한 밑거름이 될 것이라고 생각한다.

개인정보(ID) 강성민

이번 종합설계프로젝트는 학부 과정 동안 학습한 이론적 지식을 바탕으로, 우리만의 실체적인 결과물을 직접 구현해 볼 수 있는 뜻깊은 활동이었습니다. 이는 단순한 이론 학습을 넘어, 실제적인 공학적 문제 해결 역량을 기르는 중요한 토대가 되었습니다. 특히 QR코드라는 일상적인 소재를 산업적 관점에서 재해석하고, 머릿속으로만 구상하던 아이디어를 실제 운용 가능한 시스템으로 구현해 보는 과정은 매우 뜻깊었습니다.

본 프로젝트에서 하드웨어 개발 파트를 담당하며 3D 프린팅 출력 이슈, 배선 및 납땜 불량 등 다양한 기술적 난관에 부딪혔습니다. 특히 '직진 주행'과 같이 겉보기에는 단순해 보이는 기초적인 동작조차도, 실제 하드웨어로 구현하고 제어하는 과정에서는 수많은 변수를 통제해야 한다는 점을 깊이 깨달았습니다.

이러한 기술적 시행착오와 더불어 프로젝트 진행 중 팀원들과 발생했던 의견 차이 또한 귀중한 성장의 밑거름이 되었습니다. 졸업 후 실무 환경에서 수행하게 될 업무는 결국 타인과의 협업을 전제로 합니다. 이번 프로젝트는 팀원들과 호흡을 맞추고 이견을 조율하며 최선의 결과를 도출해 내는 협력의 가치를 체득할 수 있었던 값진 경험이었습니다.

개인정보(ID) 임재엽

본 종합설계 프로젝트는 대학 재학 기간 동안 습득한 학술적 이론과 실험 지식을 하나의 통합된 시스템으로 구현해 볼 수 있는 뜻깊은 기회였다. 초기 설계 단계에서 이륜 구동 로봇으로 계획했던 플랫폼을 협소한 격자형 공간에서의 기동성 확보를 위해 메카넘 휠 구동계로 전면 수정하고, 이 과정에서 발생하는 휠의 미끄러짐 현상을 제어

하기 위해 직접 포맥스로 바닥을 제작하는 등 현실적인 제한 요소들과 끊임없이 직면하였다. 이러한 물리적 변수들을 하드웨어와 소프트웨어 양측면에서 조율하고 최적의 주행 환경을 구축해 나가는 과정은 단순한 이론 학습을 넘어선 실제적인 공학적 해결 역량을 기르는 토대가 되었다.

특히, 다수의 연산 장치와 서로 다른 통신 프로토콜이 얽힌 복합 시스템을 구축하면서 데이터 동기화의 중요성을 깊이 깨달았다. 초기 구상과 달리 주행 중 실시간 데이터 유입이 제어 루프에 간섭을 일으키는 문제를 발견하고, 이를 해결하기 위해 특정 정지 이벤트 시점에만 업데이트 락을 해제하는 이벤트 기반 동기화 로직과 경로 폐쇄 시 대기하는 예외 처리 메커니즘을 고안해 낸 경험은 시스템의 무결성과 신뢰성을 확보하는 중요한 기술이 되었다.

5명의 팀원이 하드웨어 기구 설계, 제어 펌웨어 및 비전 알고리즘 개발, 그리고 인프라 시스템 통합 관리로 역할을 명확히 분담하면서도, 기술적 병목이 발생할 때마다 자신의 파트를 불문하고 합심하여 지원했던 유연한 협업 과정은 프로젝트를 성공적으로 마칠 수 있었던 가장 큰 원동력이었다. 혼자서는 해결하기 어려웠던 방대한 시스템 통합 과제를 소통과 협력을 통해 완수해 내며 팀워크의 가치를 깨달을 수 있었다. 본 프로젝트를 통해 축적한 시스템 통합, 비전 처리 지식, 그리고 실무적인 문제 해결 경험은 향후 산업 현장에서 마주할 어떠한 복합적인 공학 과제도 주도적으로 해결할 수 있다는 강한 자신감을 얻게 되었다.

개인정보(ID) 안휘훈

캡스톤 디자인 프로젝트는 끊임없는 사고의 연속이라고 생각한다. 프로젝트 초기 무엇을 제작할지 정하는 것부터 난관이었다. 만일 괜찮은 아이디어가 나왔다하더라도 그 아이디어의 구현 가능성, 구현 가능하다면 어떤식으로 구현할지 다시 방안을 강구해야 하는 과정을 겪었다. 또한 이러한 과정은 하드웨어 제어와 소프트웨어 연동, 특히 센서 데이터의 오차나 통신 지연과 같은 오차 해결, 주요 기능 등을 구현할 때도 예외가 아니었으며 이러한 문제 등을 해결하며 수많은 시행착오가 있었다.

특히 프로젝트 수행 과정에서 가장 까다로웠던 기술적 과제는 영상 처리 데이터와 센서 값을 유기적으로 융합하여 기체의 방향을 실시간으로 정확하게 정렬하는 것이었다. 초기에는 하드웨어의 물리적 오차와 통신 및 연산 지연으로 인해 기체가 중심에 위치하지 못하고 이탈하는 현상이 빈번하게 발생했는데, 이를 해결하기 위해 알고리즘과 모터 제어 보정 로직을 세밀하게 최적화한 끝에 원하는 정밀도의 기체 정렬 시스템을 완성할 수 있었다.

이러한 일련의 과정을 통해 문제해결을 위한 자세를 키울 수 있는 귀중한 경험이 되었다. 올바르게 동작하는 것에 그치지 않고 소프트웨어적으로 기능을 더 향상시키기 위한 토의와 사고를 통해 엔지니어로서 한 단계 더 크게 성장할 수 있었다. 결론적으로 이번 캡스톤 디자인은 복잡한 문제를 마주했을 때 주도적으로 원인을 분석하고 해결 방안을 강구하는 끈기를 배웠다.