

Doosan Rokey 5기

협동 자율 로봇 기반 도주 차량 추적·차단 시스템

TwoCops: 공조
2025. 11.21

C-1 성송현 정지원 이성우 김도원 유인선

C-3 김해민 장지민 나윤석 김경훈

멘토 Andy Kim

목차

01

프로젝트 개요

02

프로젝트 팀 구성 및
역할

03

프로젝트 수행 스케줄

04

프로젝트 세부 사항

05

자체 평가 의견

1. 프로젝트 개요

Chapter 1

프로젝트 주제
및 선정 배경 ,
기획의도

Chapter 2

프로젝트
내용

Chapter 3

활용 장비 및
재료

Chapter 4

프로젝트
구조

Chapter 5

활용방안 및
기대 효과

What do we solve?

최신뉴스

경찰차 교통사고로 연평균 230여명
다쳐..."저감대책 필요"

송고 2018-10-28 09:36



최근 5년 경찰차 사고 1000건 육박..."출동 안전 담보돼야"

이관주 기자

입력 2018.10.10 14:45 | 00분 26초 소요



최신뉴스

순찰차 6대 들이받으며 도주한 운전자…13km
추격전 끝 검거(종합)

송고 2025-06-17 14:27

강영훈 기자
+구독

경찰 조사 과정서 자해, 응급입원 검토…약물 검사 진행 예정

(안산=연합뉴스) 강영훈 기자 = 경찰의 정차 요구를 무시한 채 순찰차를 들이받으면서 도심 도
주극을 벌인 50대 여성 운전자가 붙잡혔다.

기존 유사 시스템



미국

순찰/감시/번호판 인식 가능
But 추격 또는 차단 기능 X



국내

건물 내부 자율 순찰
화재/침입 감지 가능
But 추격 또는 차단 기능 X



중국

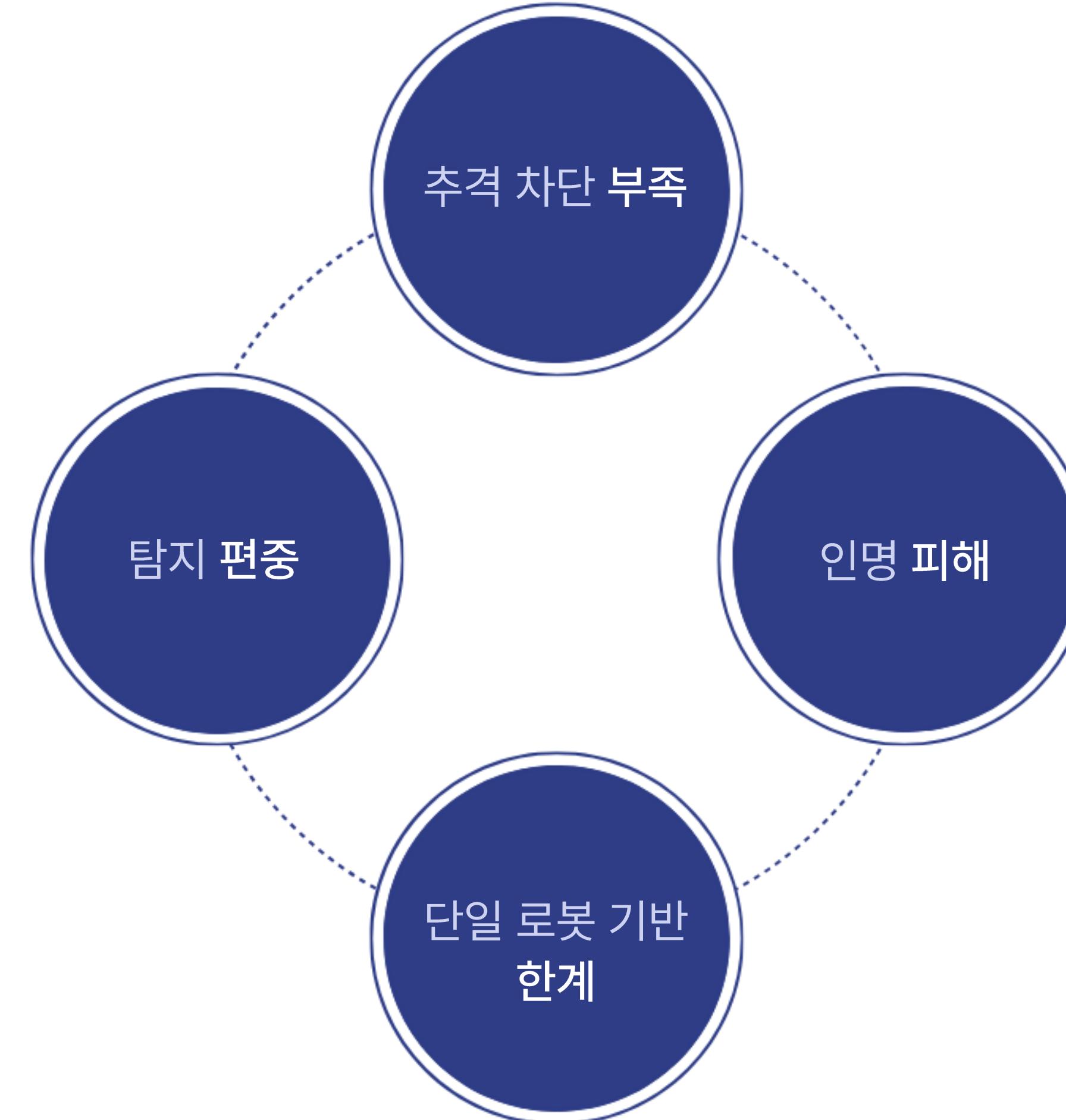
자율 순찰
But 추격 또는 협력 기능 X

기존 보안, 순찰 로봇은 '탐지'에만 집중
추격, 차단 능력은 부족

단일 로봇 기반 시스템은 로봇의 시야가
제한되고 넓은 공간/ 복잡한 골목에서
동시 관찰하거나 경로 차단 어려움

사람이 직접 범죄차량을 인지하고
추격하기엔 교통 사고의 위험

**"인명 피해 없이 다중 로봇의 협력을
이용한 추격, 차단 시스템"**



프로젝트 컨셉(Concept)

"협력형 다중 AMR 자율 추격 차단 시스템"

기존 보안 로봇이 수행하던 정적 감시 단계를 넘어서 두 로봇이 함께 동적 상황에 대응하는 자율 추격 차단 기능을 구현

- 멀티 로봇 협력(Collaborative Robotics)
=> 두 로봇이 서로 정보를 교환하여 작전 수행
- 시스템 자동화
=> 탐지 => 추격 => 협력요청 => 지원 => 차단 => 포위
- 대응 자율화
=> 중앙 서버를 통해 로봇이 탐지한 좌표 전달

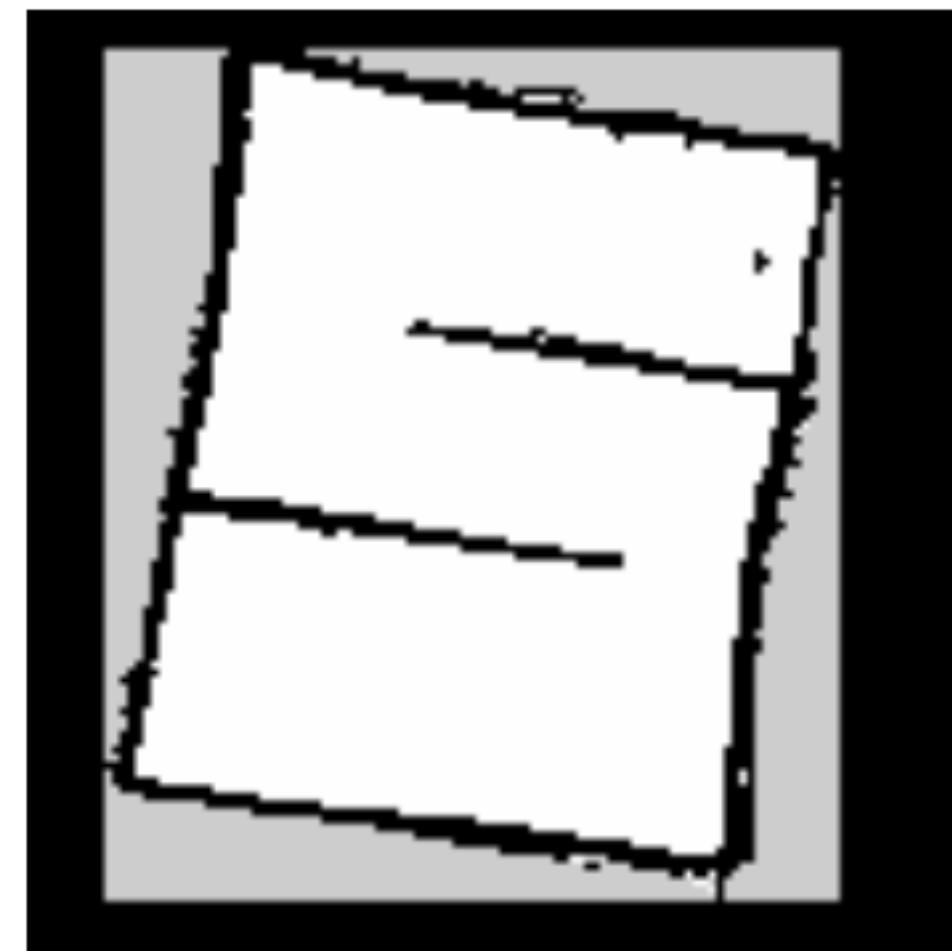
프로젝트 구현 내용

1 실시간 객체 탐지 기반 범죄차량 위치인식

- SLAM을 이용해 Map 완성
- YOLO 기반 AMR의 OAK-D 카메라로 범죄 차량 Detection
- 추격 대상의 좌표를 Stereo, RGB Camera를 이용해 계산
- 추격 대상의 좌표를 연속적으로 계산하여 Tracking

2 AMR의 협력 요청

- 추격 대상이 탐지된 경우 대상의 좌표를 반대편 AMR에 발행
- 반대편 AMR은 발행한 좌표를 구독해 NavToPose로 이동
- 이동과 동시에 추격 대상 탐지 시스템 가동



프로젝트 구현 내용

3 AMR의 자율 추격

- 추격 대상이 탐지되지 않는 경우 서로의 현재 좌표를 발행
- SLAM 기반 좌표 인식과 경로를 계획
- 장애물이 있는 경우 Nav2를 이용해 회피 및 지속적 추격

4 자동 포위 검거 시나리오 구현

- 추격조가 보낸 추격 대상의 좌표로 지원조가 즉시 이동
- 추격과 동시에 차단을 하는 시스템으로 두 AMR 양쪽 길목 차단
- 추격 대상을 가운데 두고 양쪽에 AMR이 대치하는 상황으로 검거

훈련 내용과의 연관성

1 ROS2 기반 로봇 제어 및 통신 기술 활용

- 토픽 /서비스/액션 구조를 사용해 로봇 간 협력 구현
- ROS2 기반 터틀봇 제어 및 Multi-robot DDS 구조 사용
=> 훈련에서 배운 ROS2 전체 과정 실전 활용

2 AI 객체 탐지 모델 활용

- YOLOv8n 모델에 추가적인 데이터 학습
- 해당 카메라 토픽에 적용하여 실시간 탐지 및 추적
=> AI 객체 탐지 및 추격 실전 적용

3 SLAM 및 Navigation 활용

- SLAM으로 지도 생성 및 로봇 위치 추정
- Nav2 기반 경로 계획 및 장애물 회피 적용
=> 자율 주행 로봇 실전 적용

훈련 내용과의 연관성

4 멀티 로봇 협력 설계

- DDS를 통해 여러대의 PC와 AMR을 연결해 협력 구조 설계
=> 멀티 로봇 협업과 통신 실전 적용

5 문제 해결 능력

- 센서 데이터, 장애물 회피, 실시간 제어 등 여러 리얼 타임 문제 해결
- 종속성 및 타이밍 문제 등, 개발 과정에서 발생한 문제들의 근본적인 이유와 원리를 이해하여 문제 해결

프로젝트 목적

1 최종 목표

- 두 대의 자율 이동 로봇(AMR)이 협동으로 도주 차량(RC-Car)을 추적 및 포위, 차단하는 PoC 시스템을 성공적으로 구현하는 것

2 핵심 기술

- 실시간 객체 탐지 및 추적(YOLOv8n)
- 정확한 거리 /각도/이격 계산을 기반으로 한 자율 주행 및 추격 제어
- ROS2 DDS 기반의 실시간 로봇 간 협력 통신(좌표 공유)
- Nav2를 활용한 정확한 Localization 및 Navigation
- 실시간 모니터링(Flask)

3 운영 요구 사항

- 추격 간격 오차 10cm 내외
- 이동 오차 20cm내외
- 거리 계산 오차 20cm이내

비즈니스 및 실무 활용성

1 공공 및 안전 분야

- 경계 및 순찰 임무에서 넓은 지역을 순찰하는 다중 로봇이 비인가 침입자를 포착하고, 출구 방향을 예측하여 포위
=> 협력 기반 전술 및 경로 예측을 통해 양방향 차단 로직 활용

2 생태계 분야

- 보호가 필요한 멸종위기 동물이나, 유기견, 유기묘 등을 협동 추격 로직을 이용하여 포획

2. 프로젝트 팀 구성 및 역할

TwoCops: 공조

훈련생	역할	담당 업무
유인선	매니저	프로젝트 총괄, Integration
김경훈	팀원	YOLO 모델 학습/라벨링, launch파일 통합
김도원	팀원	nav_to_pose Node 개발
김해민	팀원	YOLO 모델 학습/라벨링, Tracking Object 좌표 연산
나윤석	팀원	YOLO 모델 학습/라벨링, Detection Node 개발
성송현	팀원	system design 및 system architecture diagram
이성우	팀원	nav_to_pose Node 개발
장지민	팀원	YOLO 모델 학습/라벨링, Tracking Node 개발
정지원	팀원	Flask 모니터링

3. 프로젝트 수행 스케줄

TwoCops: 공조

구분	기간	활동	비고
사전 기획	11/14	프로젝트 기획 및 주제 선정, BRD 작성	
YOLO 모델 학습	11/14	YOLOv8n 활용하여 데이터 수집/라벨링 및 커스텀 모델 학습	
Vision 및 추적 시스템 개발	11/15 ~ 11/17	Detection/Tracking Node 개발 및 Loss 감지 후 좌표 연산	초반 인식 가능
로봇 자율주행 시스템 구축	11/14 ~ 11/17	NavToPose Node 개발 및 타겟 상태 업데이트 로직	로봇 주행 가능
모니터링 시스템 구축	11/16 ~ 11/18	카메라 스트리밍, 로그인 기능, 비디오 로그 저장 기능 구현	웹/관제 시스템
유닛 테스트	11/16 ~ 11/18	개별 유닛 정상 작동 여부 및 로봇 간 통신 여부 확인	
시스템 통합	11/18 ~ 11/19	Node 동기화, Nav/Tracking 간 전환 로직 작성, E2E 통합 테스트	기능 연동 및 마무리
문서화	11/15 ~ 11/20	SRD, ADD, SDD, 발표자료 작성	
총 개발기간	11/14 ~ 11/20 (총 7일)		

4. 프로젝트 세부사항

TwoCops: 공조

개발 환경

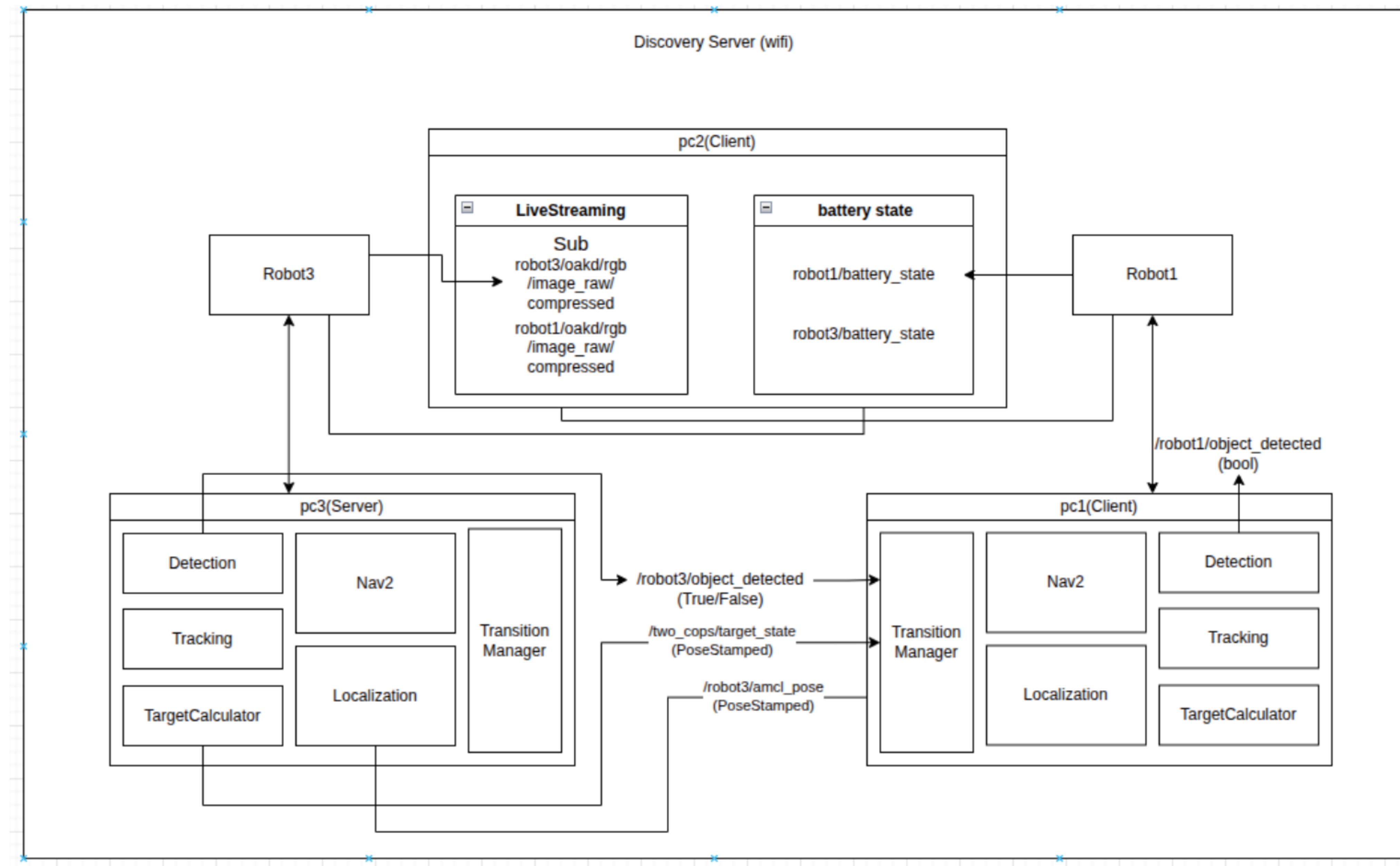
1 하드웨어 환경

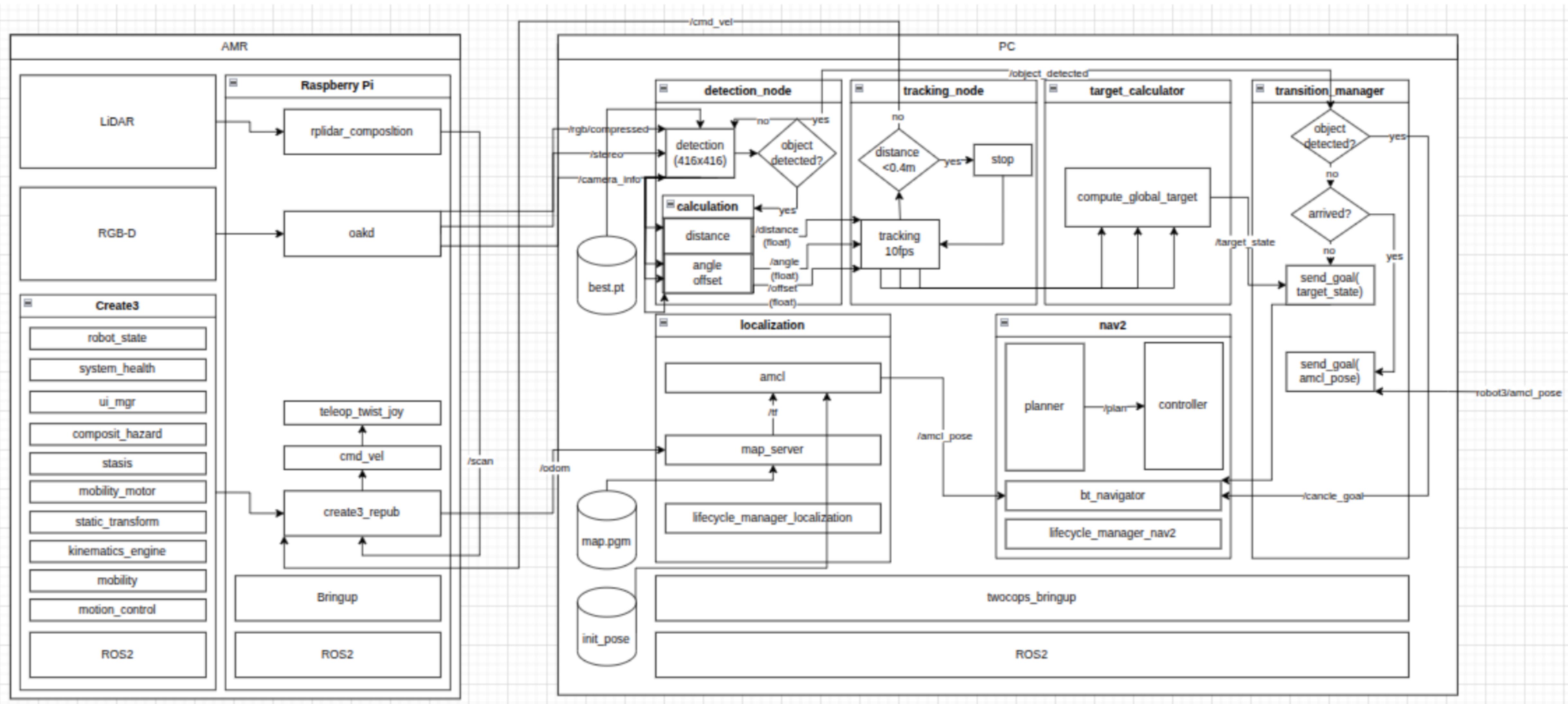
구분	상세	구분	상세
AMR (자율주행로봇)	프로세서: Raspberry Pi 4B 4GB(On-Board Computer)	클라이언트 PC (robot1 담당)	cpu: 16-core2-thread RAM: 32GB GPU:NVIDIA RTX 4070 OS:Ubuntu
센서	OAK-D Depth Camera LiDaR	클라이언트 PC (모니터링 담당)	CPU: Intel® Core™ Ultra 7 155H × 22 RAM: 32GB GPU: Mesa Intel® Arc(tm) Graphics (MTL) OS: Ubuntu 22.04
서버 PC (robot3 담당)	cpu: 8-core 8-thread RAM: 32GB GPU:NVIDIA RTX 3050 OS:Ubuntu	네트워크	Wi-Fi FastDDS

개발 환경

2 소프트웨어 환경

구분	상세	구분	상세
PC	<ul style="list-style-type: none">- Python3- OpenCV- CvBridge- ROS2 Humble- Ultralytics YOLOv8n- Detection Node: 추적 대상을 디텍션- Tracking Node: 추적 대상을 실시간으로 트래킹- NavToPose Node: 정해진 작전 위치로 주행- Transition Manager: 전반적인 시나리오 로직 관리- Flask Studio: 실시간 영상 모니터링 및 영상 저장	AMR	<ul style="list-style-type: none">- ROS2 Humble- Create3 Node: 구동부(Create3)의 상태 확인 및 제어- LiDAR Node: 라이다의 신호를 처리- OAKD Node: RGB 이미지, 스테레오 이미지, 카메라 보정 정보 신호를 처리





구체적인 실행 내역

1 소프트웨어 환경 설정(Setup)

- 환경: 서버/클라이언트 PC 및 Turtlebot4에 Ubuntu 22.04 및 ROS2 Humble
- 네트워크/보안: 모든 장치 Wi-Fi 연결 및 SSH 키 기반 인증, ROS2 Discovery server 설정
- 데이터 준비: YOLOv8n 모델에 추적대상(RC-Car) 학습 완료, SLAM을 통해 테스트 환경에 계획한 Map 파일 생성

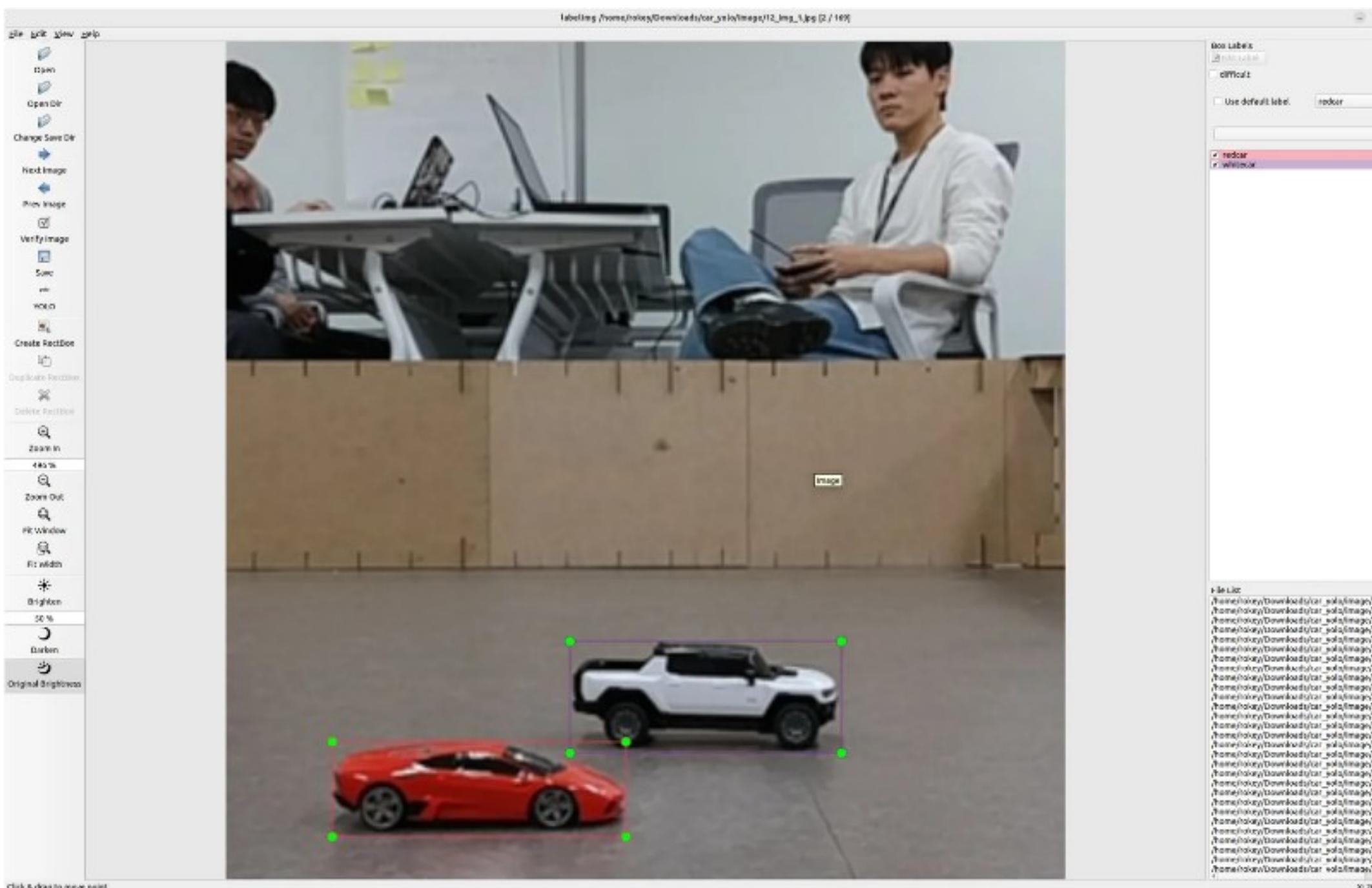
구체적인 실행 내역

2 Detection & Tracking

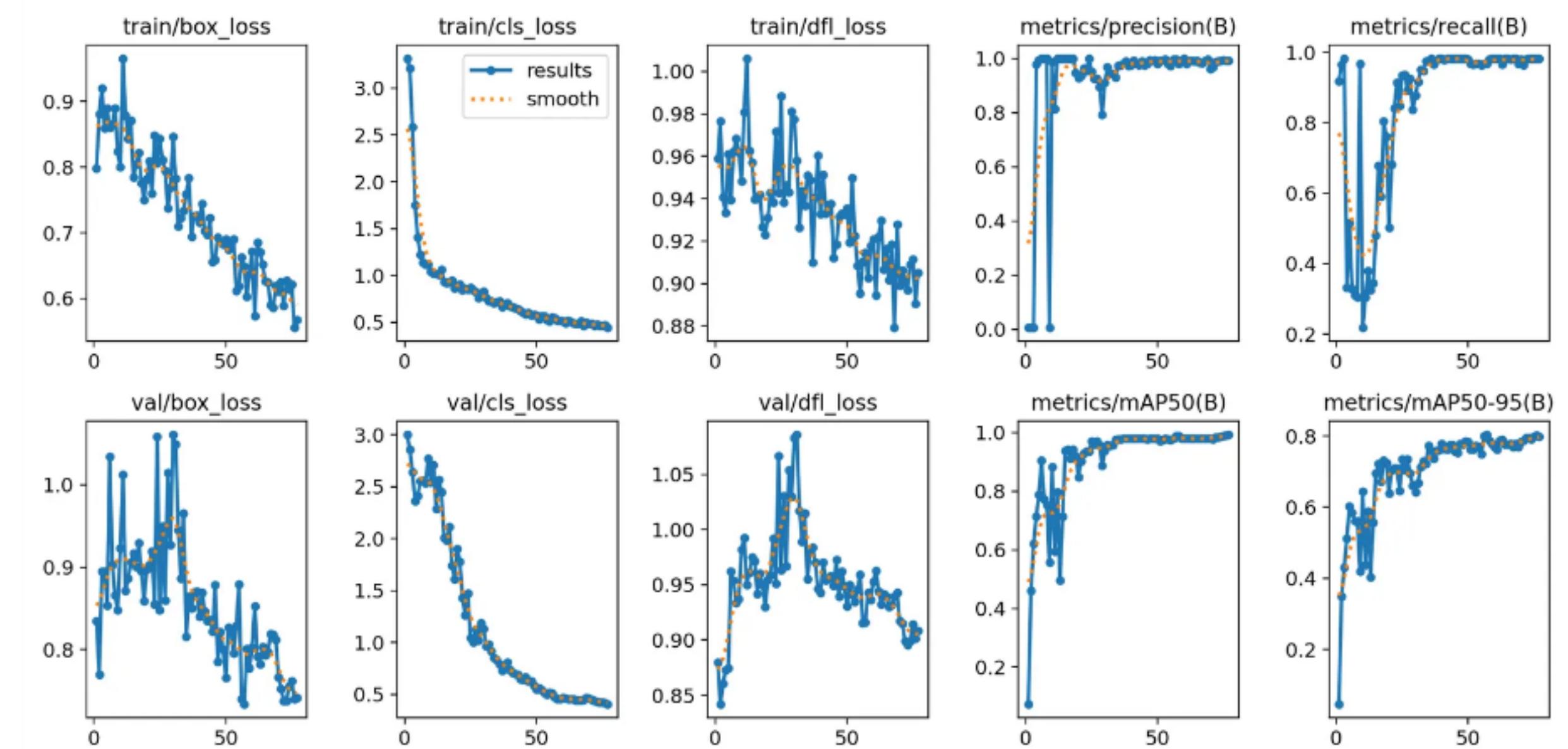
- 실제 환경에서의 성능을 고려하여 AMR에 장착된 카메라 모듈을 사용해 데이터 수집
- 추격 대상(Red car) 비추격 대상(White car)의 구분을 위해 두 차량의 이미지 데이터 모두 수집 및 라벨링
- 학습된 모델을 이용하여 RGB-D카메라로 들어오는 데이터를 실시간으로 감지
- 감지된 타겟의 거리와 각도를 계산
- 계산된 거리와 각도를 바탕으로 cmd_vel을 제어하여 타겟 위치로 이동하도록 AMR제어

데이터 학습 (YOLOv8n)

AMR(turtlebot4)의 cam을 통해 capture하여 데이터 수집
이후 labelImg를 통해 redcar와 whitecar 학습데이터 라벨링

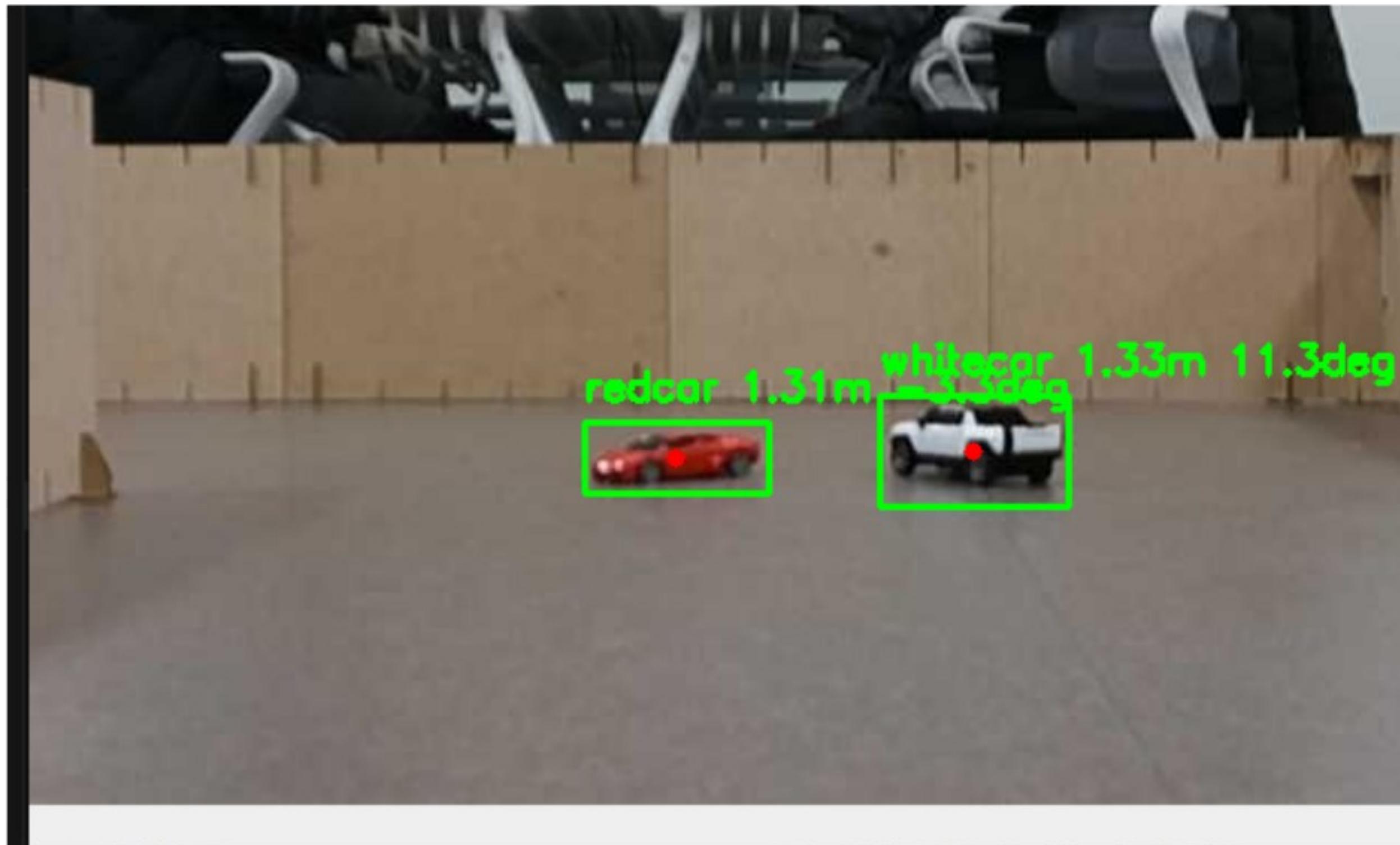


Box Loss 0.41 → 위치 정확도 매우 높음
metrics/mAP50(B) → IoU 0.5 기준 mAP 98.9%
metrics/precision(B) & recall(B) → Precision 99.3%, Recall 98.3% →
오탐·미탐 거의 없음



Detection

AMR(TurtleBot4)가 주행중에 가져오는 OAK-D 카메라 RGB·Depth 데이터를 기반으로 YOLO 모델이 redcar와 whitecar를 실시간 탐지하고, 각 객체의 거리(m)와 각도(deg)를 추정해 퍼블리시.



카메라 모델 기반 각도 계산

- pinhole 모델:

$$X = d \cdot \frac{(u - cx)}{fx}$$

$$\text{angle} = \text{atan2}(X, d)$$

Tracking

YOLODepthFusion이 퍼블리시한 거리(distance), 각도(angle), 객체 존재 여부(object_detected)를 기반으로 AMR0| redcar를 향해 움직이는 추종 제어를 수행.



$$w = -0.01 \cdot \text{angle} \quad (|\text{angle}| > \text{deadzone})$$

$$\text{deadzone} = 2^\circ$$

작은 흔들림 제거 → “부드러운 회전”

$$\text{err} = \text{dist} - 1.0$$

가까울수록 천천히 접근

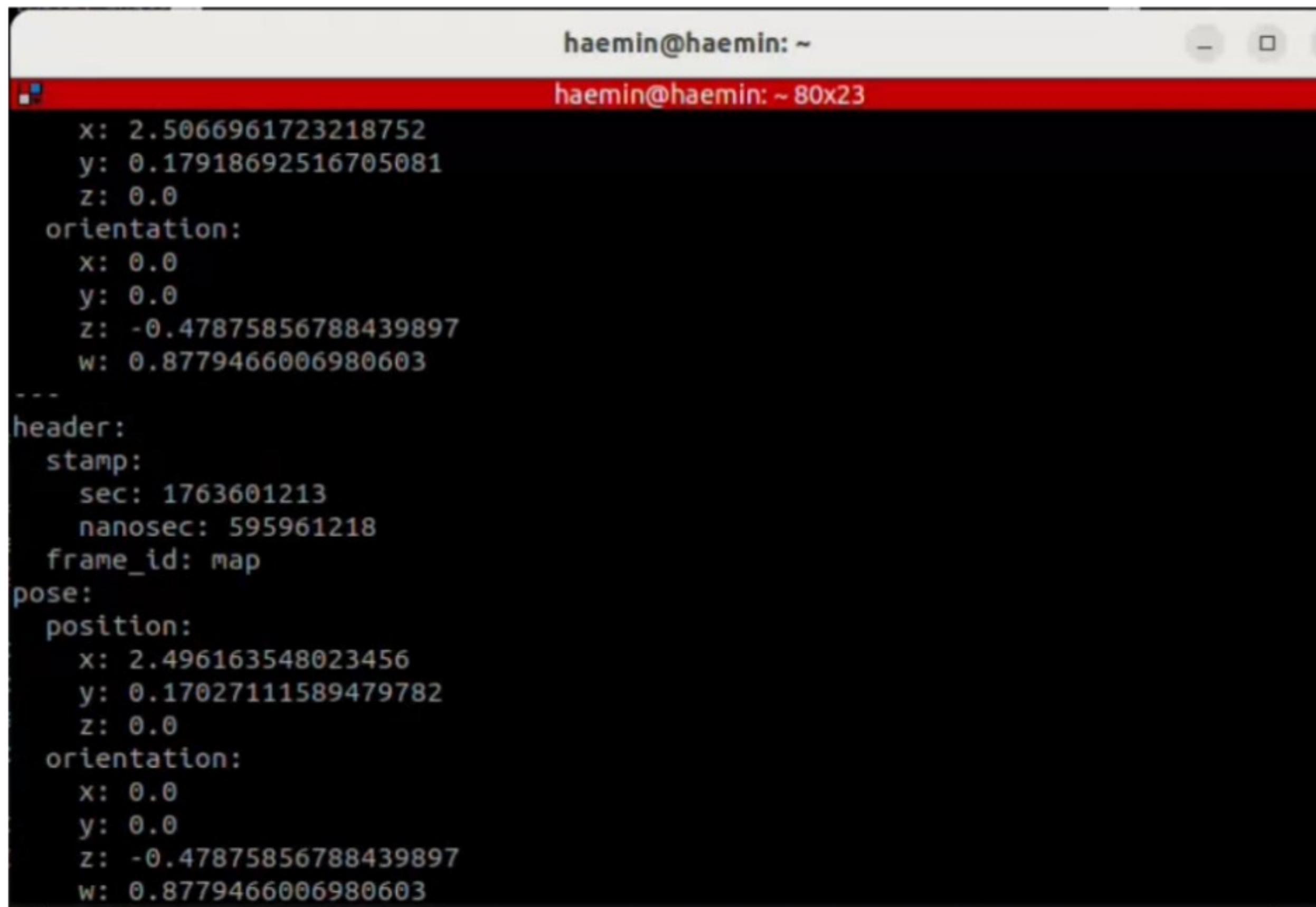
$$v = \min(0.9, 0.4 \cdot \text{err})$$

멀면 빠르게 접근

최대 속도 0.9 m/s 제한

Target Calculator

Detection node에서 redcar의 거리(distance)와 각도(angle)를 획득하고, AMCL로 추정된 로봇의 현재 pose(x, y, yaw)를 결합하여 redcar의 실제 위치를 map 좌표계로 변환.



A terminal window titled 'haemin@haemin: ~' showing ROS message output. The message is a 'Target' object with fields: header, stamp, frame_id, pose, and orientation. The pose field contains position (x: 2.496163548023456, y: 0.17027111589479782, z: 0.0) and orientation (x: 0.0, y: 0.0, z: -0.47875856788439897, w: 0.8779466006980603). The timestamp is sec: 1763601213, nanosec: 595961218.

```
haemin@haemin: ~
haemin@haemin: ~ 80x23

x: 2.5066961723218752
y: 0.17918692516705081
z: 0.0
orientation:
x: 0.0
y: 0.0
z: -0.47875856788439897
w: 0.8779466006980603

header:
stamp:
sec: 1763601213
nanosec: 595961218
frame_id: map
pose:
position:
x: 2.496163548023456
y: 0.17027111589479782
z: 0.0
orientation:
x: 0.0
y: 0.0
z: -0.47875856788439897
w: 0.8779466006980603
```

$$\theta = \text{yaw} + \text{angle}_{rad}$$

$$x_{target} = x_{robot} + d \cdot \cos(\theta)$$

$$y_{target} = y_{robot} + d \cdot \sin(\theta)$$

구체적인 실행 내역

3 로봇 자율 주행 및 협력 로직

- Localization: 맵, /scan, /odom 데이터를 사용하여 AMCL 기반의 정확한 로봇 현재 위치(/amcl_pose) 발행 및 검증
- Navigation: nav_to_pose 노드를 이용, BT Navigator를 통해 목표 좌표 (/goal_pose) 수신 및 Planner/Controller와 연동하여 자율주행(/cmd_vel) 기능 구현
- TransitionManager: 전체 시나리오 로직을 관할하는 노드. Detection 발생 시 NavToPose를 즉시 중단하고 Tracking 실행. Detection이 일어나지 않은 경우 가장 최신으로 갱신된 target 좌표, 또는 (이미 그 좌표로 이동한 경우) 상대 AMR의 현재 좌표로 이동
- 최초 dock 상태의 AMR을 undock하는 것부터 마지막 검거 시나리오까지 하나의 통합 launch 파일(twocops_bringup)로 실행

구체적인 실행 내역

3 로봇 자율 주행 및 협력 로직

- 협력 요청: 최초 탐지 로봇이 추격 대상 탐지 후 대상의 좌표 발행
- 협력 이동: 반대편 로봇이 /two_cops/target_state를 구독 시, 해당 좌표를 goal_pose로 설정하여 nav_to_pose 이동
- 경로 방어: 추격 중 로봇이 대상을 잃으면 마지막 좌표로 이동 및 경로 방어
- 검거: 두 로봇 사이(40cm 이격)에 추격 대상이 있고 두 로봇이 탐지한 상황이면 검거 성공이라고 판단

twocops_bringup

undock-> loc -> init pose-> nav -> sub 'two_cops/target_state' ->nav_to_goal -> mission succeed

하나의 launch 코드로
수행

```
/bin/bash 95x46
[behavior_server-11] [INFO] [1763685818.796367482] [robot1.behavior_server]: Destroying
[INFO] [waypoint_follower-13]: process has finished cleanly [pid 4880]
[INFO] [smoother_server-9]: process has finished cleanly [pid 4872]
[planner_server-10] [INFO] [1763685818.833206524] [robot1.planner_server]: Destroying
[INFO] [amcl-4]: process has finished cleanly [pid 4774]
[controller_server-8] [INFO] [1763685818.852880337] [robot1.controller_server]: Destroying
[INFO] [lifecycle_manager-15]: process has finished cleanly [pid 4884]
[ERROR] [detection-16]: process has died [pid 4972, exit code -6, cmd '/home/rokey/turtlebot4_ws/install/turtlebot4_tracking/lib/turtlebot4_tracking/detection --ros-args -r __node:=detection -r __ns:=/robot1 --params-file /tmp/launch_params_i0asrypg']. 
[INFO] [behavior_server-11]: process has finished cleanly [pid 4876]
[INFO] [planner_server-10]: process has finished cleanly [pid 4874]
[INFO] [controller_server-8]: process has finished cleanly [pid 4870]
[bt_navigator-12] [INFO] [1763685819.015657967] [robot1.bt_navigator]: Completed Cleaning up
[bt_navigator-12] [INFO] [1763685819.015704301] [robot1.bt_navigator]: Destroying bond (bt_navigator) to lifecycle manager.
[bt_navigator-12] [INFO] [1763685819.031833774] [robot1.bt_navigator]: Destroying
[INFO] [bt_navigator-12]: process has finished cleanly [pid 4878]
rokey@rokey-Victus-by-HP-Laptop-16-d1xxx:~$ ^C
rokey@rokey-Victus-by-HP-Laptop-16-d1xxx:~$ robot-restart
The daemon has been stopped
The daemon has been started
rokey@rokey-Victus-by-HP-Laptop-16-d1xxx:~$ sb
rokey@rokey-Victus-by-HP-Laptop-16-d1xxx:~$ ros2 launch twocops_bringup twocops_bringup.launch.py
robot_id:=1 initial_pose_file:=$HOME/turtlebot4_ws/config/robot1_initial_pose.txt map:=$HOME/turtlebot4_ws/maps/key_map.yaml params_file:=$HOME/turtlebot4_ws/maps/local2.yaml nav2_params_file:=$HOME/turtlebot4_ws/maps/nav2_net2.yaml robot_id:=1 partner_robot_id:=3
[INFO] [launch]: All log files can be found below /home/rokey/.ros/log/2025-11-21-09-45-03-78625-rokey-Victus-by-HP-Laptop-16-d1xxx-5433
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [dock_state_manager-1]: process started with pid [5434]
[dock_state_manager-1] [INFO] [1763685904.099956981] [dock_state_manager]: DockStateManager initialized (ns=/robot1) - waiting for dock_status...
[dock_state_manager-1] [INFO] [1763685905.435791527] [dock_state_manager]: Dock status received: is_docked=True
[dock_state_manager-1] [INFO] [1763685905.436355090] [dock_state_manager]: Robot is docked at startup → sending UNDOCK goal
[dock_state_manager-1] [INFO] [1763685905.441319275] [dock_state_manager]: Undock goal accepted
[dock_state_manager-1] [INFO] [1763685906.467214562] [dock_state_manager]: Dock status received: is_docked=False
[dock_state_manager-1] [INFO] [1763685907.489647966] [dock_state_manager]: Dock status received: is_docked=False
[dock_state_manager-1] [INFO] [1763685908.425235230] [dock_state_manager]: Dock status received: is_docked=False

```

```
/bin/bash
ms
64 bytes from 192.168.107.56: icmp_seq=62 ttl=64 time=599
ms
64 bytes from 192.168.107.56: icmp_seq=63 ttl=64 time=6.04
ms
64 bytes from 192.168.107.56: icmp_seq=64 ttl=64 time=28.3
ms
64 bytes from 192.168.107.56: icmp_seq=65 ttl=64 time=28.9
ms
64 bytes from 192.168.107.56: icmp_seq=66 ttl=64 time=2.10
ms
64 bytes from 192.168.107.56: icmp_seq=67 ttl=64 time=24.5
ms
64 bytes from 192.168.107.56: icmp_seq=68 ttl=64 time=4.98
ms
64 bytes from 192.168.107.56: icmp_seq=69 ttl=64 time=2.09
ms
64 bytes from 192.168.107.56: icmp_seq=70 ttl=64 time=25.7
ms
64 bytes from 192.168.107.56: icmp_seq=71 ttl=64 time=17.4
ms
64 bytes from 192.168.107.56: icmp_seq=72 ttl=64 time=2.01
ms
64 bytes from 192.168.107.56: icmp_seq=73 ttl=64 time=5.89
ms
64 bytes from 192.168.107.56: icmp_seq=74 ttl=64 time=22.4
ms
64 bytes from 192.168.107.56: icmp_seq=75 ttl=64 time=1.90
ms
64 bytes from 192.168.107.56: icmp_seq=76 ttl=64 time=23.6
ms
64 bytes from 192.168.107.56: icmp_seq=77 ttl=64 time=32.7
ms
64 bytes from 192.168.107.56: icmp_seq=78 ttl=64 time=2.33
ms
64 bytes from 192.168.107.56: icmp_seq=79 ttl=64 time=23.5
ms
64 bytes from 192.168.107.56: icmp_seq=80 ttl=64 time=2.26
ms
64 bytes from 192.168.107.56: icmp_seq=81 ttl=64 time=6.71
ms
64 bytes from 192.168.107.56: icmp_seq=82 ttl=64 time=23.7
ms
64 bytes from 192.168.107.56: icmp_seq=83 ttl=64 time=24.6
ms

```

```
/bin/bash 35x46
rokey@rokey-Victus-by-HP-Laptop-16-d1xxx:~$ dock 1
Waiting for an action server to become available...
Sending goal:
{}

Goal accepted with ID: 0a8c9af0abc646deb04c02f2b2c44fa7
Result:
is_docked: true
Goal finished with status: SUCCEEDED
rokey@rokey-Victus-by-HP-Laptop-16-d1xxx:~$ 
```

구체적인 실행 내역

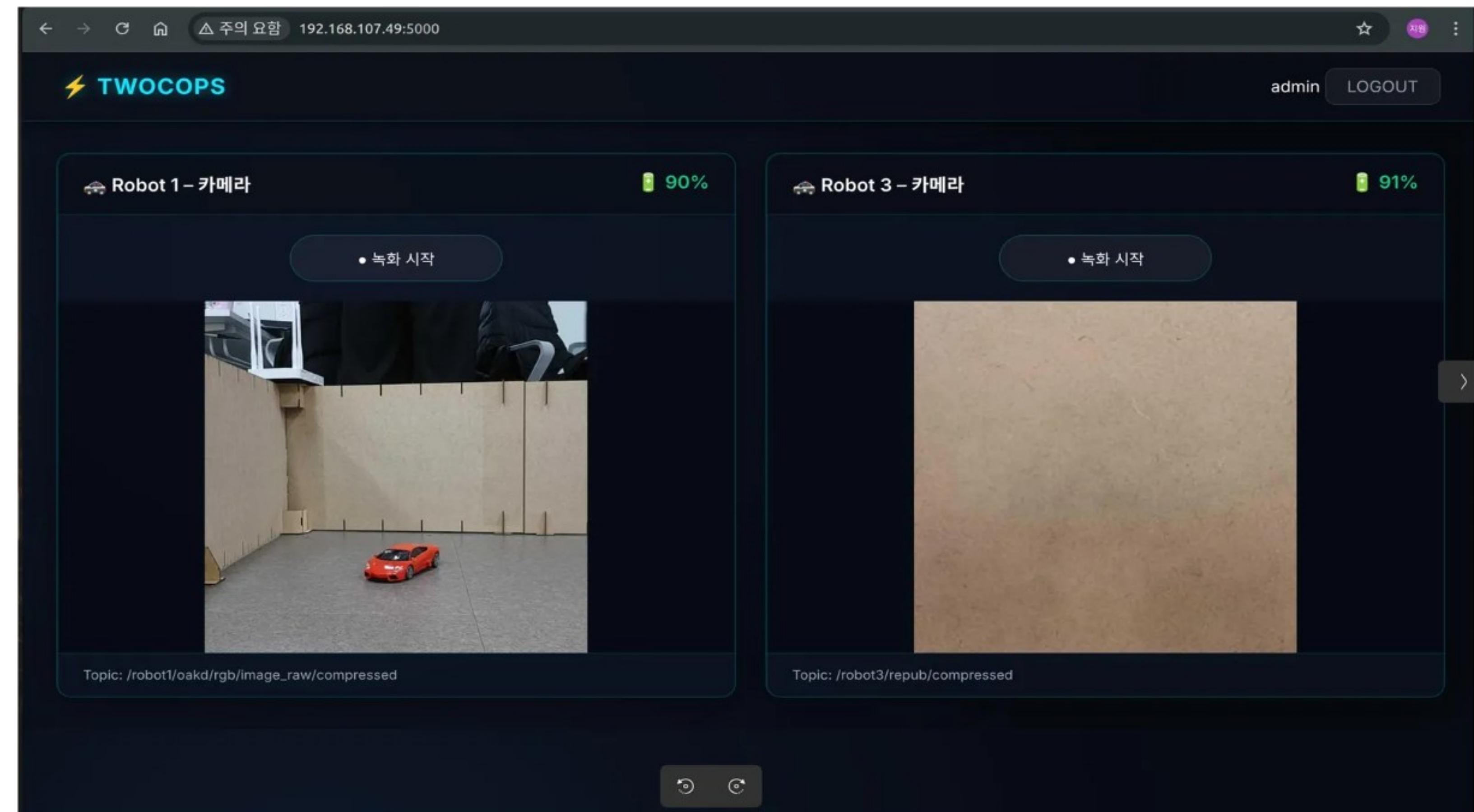
4 모니터링 및 테스트

- 모니터링: Flask(WebSocket)가 /robot3/oakd/rgb/image_raw/compressed, /robot1/oakd/rgb/image_raw/compressed 토픽을 실시간으로 구독하여 실시간 웹 스트리밍 구현
- 데이터 저장: 수신된 이미지를 numpy 형식으로 변환하여 mp4 형식으로 저장
- 통합 테스트: launch Navigator(localization > initial pose > navigation) - launch Tracker(Detection & Tracking) - launch Transition Manager(Tracking > NavToPose)
- 오류 처리: Localization 오류 발생 시 선행 조건(odom_frame, base_link_frame, laser_frame, scan_topic) 체크 스크립트(prep_checker.sh)를 통해 문제 진단 복구

Monitoring

ROS2 노드가 robot1/robot3의 카메라 압축 영상 토픽과 배터리 상태 토픽을 실시간으로 구독해, Flask 서버에서 MJPEG 스트리밍으로 웹 대시보드에 출력.

스트리밍 중인 프레임은 녹화 버튼으로 mp4 파일로 저장되며, 배터리 퍼센트는 /battery_status로 UI에 실시간 갱신.

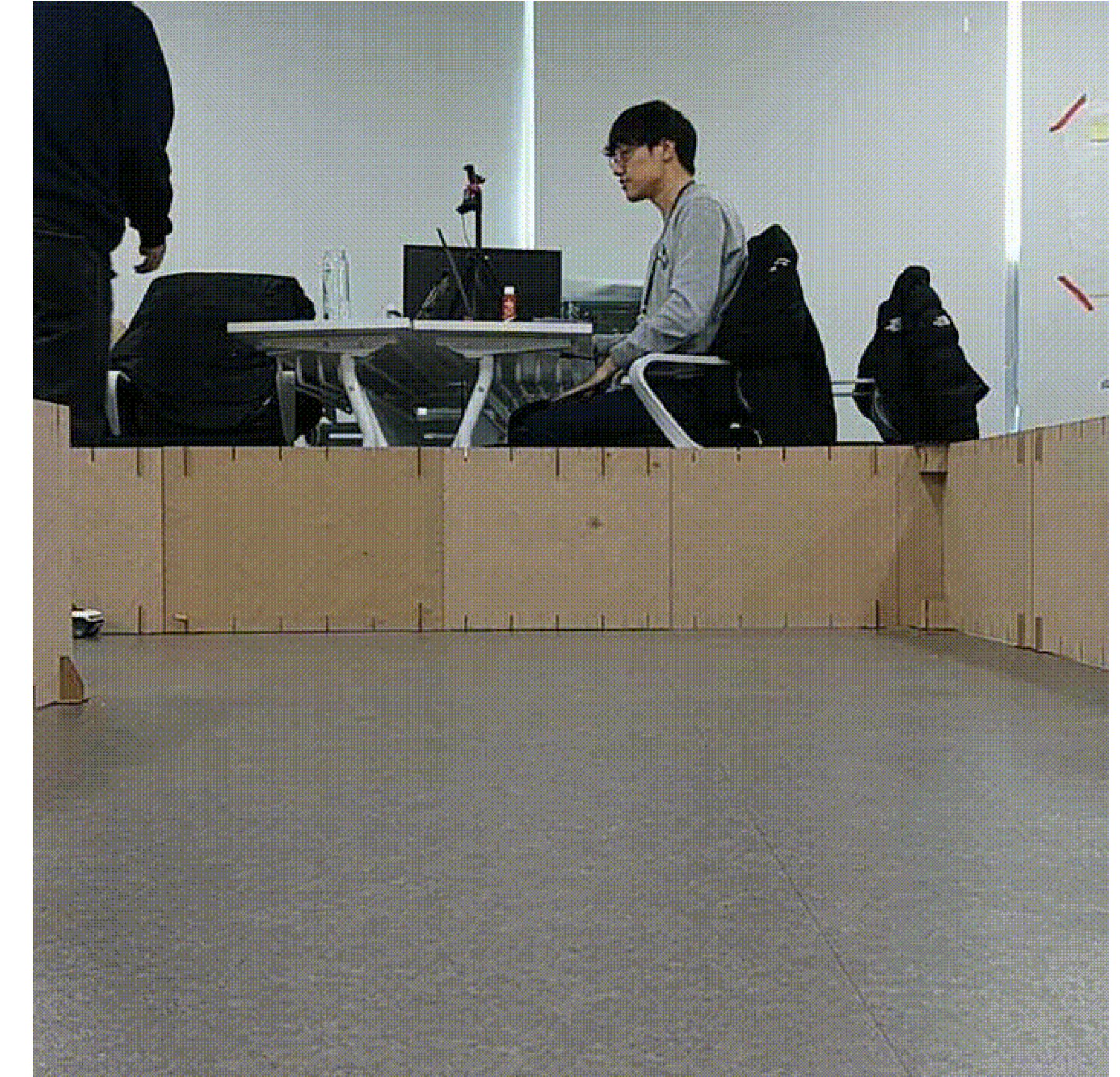


UI

robot1



robot3



5. 자체 평가 의견

TwoCops: 공조

Key issues and challenges

```
/bin/bash 95x46
ebot4_ws/install/turtlebot4_tracking/lib/turtlebot4_tracking/target_calculator --ros-args -r __
node:=target_calculator -r __ns:=/robot1'.
[INFO] [velocity_smoothen-14]: process has finished cleanly [pid 4882]
[lifecycle_manager-15] [INFO] [1763685818.742159988] [robot1.lifecycle_manager_navigation]: Des
stroying lifecycle_manager_navigation
[ERROR] [transition_manager-19]: process has died [pid 4978, exit code 1, cmd '/home/rokey/turt
lebot4_ws/install/twocops Bringup/lib/twocops Bringup/transition_manager --ros-args -r __node:=
transition_manager_robot1 --params-file /tmp/launch_params_zvk6fbby --params-file /tmp/launch_p
arams_ebz9vzi4'].
[planner_server-10] [INFO] [1763685818.745398000] [robot1.planner_server]: Destroying bond (pla
nner_server) to lifecycle_manager.
[INFO] [map_server-3]: process has finished cleanly [pid 4772]
[INFO] [lifecycle_manager-5]: process has finished cleanly [pid 4776]
[planner_server-10] [INFO] [1763685818.771514930] [robot1.global_costmap.global_costmap]: Destr
oying
[controller_server-8] [INFO] [1763685818.775979950] [robot1.controller_server]: Destroying bond
(controller_server) to lifecycle_manager.
[controller_server-8] [INFO] [1763685818.793951650] [robot1.local_costmap.local_costmap]: Destr
oying
[behavior_server-11] [INFO] [1763685818.796367482] [robot1.behavior_server]: Destroying
[INFO] [waypoint_follower-13]: process has finished cleanly [pid 4880]
[INFO] [smoother_server-9]: process has finished cleanly [pid 4872]
[planner_server-10] [INFO] [1763685818.833206524] [robot1.planner_server]: Destroying
[INFO] [amcl-4]: process has finished cleanly [pid 4774]
[controller_server-8] [INFO] [1763685818.852880337] [robot1.controller_server]: Destroying
[INFO] [lifecycle_manager-15]: process has finished cleanly [pid 4884]
[ERROR] [detection-16]: process has died [pid 4972, exit code -6, cmd '/home/rokey/turtlebot4_w
s/install/turtlebot4_tracking/lib/turtlebot4_tracking/detection --ros-args -r __node:=detection
-r __ns:=/robot1 --params-file /tmp/launch_params_i0asrypg'].
[INFO] [behavior_server-11]: process has finished cleanly [pid 4876]
[INFO] [planner_server-10]: process has finished cleanly [pid 4874]
[INFO] [controller_server-8]: process has finished cleanly [pid 4870]
[bt_navigator-12] [INFO] [1763685819.015657967] [robot1.bt_navigator]: Completed Cleaning up
[bt_navigator-12] [INFO] [1763685819.015704301] [robot1.bt_navigator]: Destroying bond (bt_navi
gator) to lifecycle_manager.
[bt_navigator-12] [INFO] [1763685819.031833774] [robot1.bt_navigator]: Destroying
[INFO] [bt_navigator-12]: process has finished cleanly [pid 4878]
rokey@rokey-Victus-by-HP-Laptop-16-dixxx:~$ ^C
rokey@rokey-Victus-by-HP-Laptop-16-dixxx:~$ robot-restart
The daemon has been stopped
The daemon has been started
rokey@rokey-Victus-by-HP-Laptop-16-dixxx:~$ sb
rokey@rokey-Victus-by-HP-Laptop-16-dixxx:~$ ros2 launch twocops Bringup twocops Bringup.launch.
py robot_id:=1 initial_pose_file:=$HOME/turtlebot4_ws/config/robot1_initial_pose.txt map:
=SHOME/turtlebot4_ws/maps/key_map.yaml params_file:=$HOME/turtlebot4_ws/maps/local2.yaml na
v2_params_file:=$HOME/turtlebot4_ws/maps/nav2_net2.yaml robot_id:=1 partner_robot_id:=3
```

1) nav2 실행 시 잦은 오류 발생

-> localization과 navigation 실행에 필요한 노드와 토픽들을 정확하게 파악

-> nav2 체크 스크립트 구현, nav 실행 전 사전 검사를 통해 nav가 실행될 수 있는 환경인지 판단하여 오류 변수를 줄임

2) nav to pose로 tracking을 할 경우 planning -> control 사이의 지연과 보수적인 이동은 추격에 적합하지 않음

-> nav2 goal로 목표 이동이 아닌 cmd_vel로 빠른 추적

1) 사전 기획의 관점에서 프로젝트 결과물에 대한 완성도 평가

본 프로젝트는 사전 기획 단계에서 설정한 핵심 목표인

- (1) YOLO 기반 도주 차량 인식,
- (2) robot1의 실시간 추격 및 좌표 전달,
- (3) robot3의 목표 지점 이동 및 차단 동작,
- (4) 협업 기반 검거 시나리오 구현을 명확하게 충족.

두 로봇 간의 역할 분담 구조가 효과적으로 설계되었으며, YOLO 모델을 통한 RC 차량 식별이 실제 환경에서도 안정적으로 이루어졌고, Nav2 기반 이동 제어 역시 기획 당시 의도한 경로·속도·정지 조건을 정확히 수행

2) 개인 또는 우리 팀이 잘한 점

✓ 잘한 점

(1) 역할 분담: 이번 프로젝트에서 팀은 계획 단계에서 설정한 역할 분담 구조를 실제 구현으로 효과적으로 연결 함

-특히 robot3의 YOLO 기반 도주 차량 탐지 및 추격 기능이 매우 안정적으로 동작하였고, 인식된 좌표를 robot1이 수신하여 자연 없이 목표 지점으로 이동하는 협업 흐름을 구축한 점은 높은 수준의 실무 능력을 보여줌

(2) 디버깅: 특정 노드 실행시 종속성이나 타이밍 문제를 해결하기 위해 상태 체크 스크립트를 사용하였고, 로봇과 피씨 사이의 노드 관계를 정확하게 파악함으로써 오류의 변인을 줄였음

(3) 통합: 두 로봇이 서로 다른 역할(추적·차단)을 수행하면서도 전체 시나리오가 끊김 없이 진행되도록 노드 구성, 네임스페이스 관리, Nav2 제어, TF 좌표계 처리를 스스로 해결한 점은 기술적으로 우수하다. 또한 YOLO 검출을 단순 시각화에 그치지 않고, 검출 결과 → 좌표 변환 → 로봇 행동(Action) 으로 연계한 설계는 로봇 SW 개발에서 중요한 역량

2) 프로젝트 결과물의 아쉬운 점 및 추후 개선점

(1) 로봇 간 협업 구조의 확장성 강화

현재 robot1 → robot3 좌표 전달 방식은 단순 Topic 기반이므로 상황 변화에 대한 상호 상태 공유(State Synchronization) 가 제한적. 향후에는 Service/Action 또는 Behavior Tree 기반 구조를 적용하면 협력 동작의 안정성과 반응성을 향상 가능성

(2) 정지 조건 및 거리 제어 알고리즘의 정밀화

현재 정지 로직은 조건 기반으로 잘 동작하지만, 실제 현장에서 요구되는 정밀성을 고려할 때 목표 접근 속도 조절을 비선형으로 적용하면 로봇3의 차단 동작이 훨씬 안정적으로 수행될 것으로 판단

(3) 프로젝트 진행 초반에 구현 용이성에만 너무 치우친 설계를 했다는 점

프로젝트의 목적을 생각하지 않고 구현 용이성만 생각하여, 완성도가 떨어진 설계로 진행해 초반에 방향성이 약간 어긋났지만, 이후 피드백을 받고 설계에 반영하여 목적에 맞는 설계와 구현을 완성함

3) 프로젝트를 수행하면서 느낀 점이나 경험한 성과 (경력 계획 등과 연관)

- (1) 본 프로젝트를 통해 비전 인식–자율 이동–다중 로봇 협업이라는 로봇 시스템 핵심 요소를 통합적으로 경험
- (2) YOLO 기반 추격, 좌표 공유, 협업 차단 과정을 직접 설계·구현하며 로봇이 인식→판단→행동하는 전체 구조에 대한 실질적 이해를 확보
- (3) 실험 중 발생한 문제를 스스로 분석·해결하는 과정에서 디버깅 역량·시스템적 사고·로봇 제어 SW 이해도가 크게 향상

발표를 들어주셔서 감사합니다

TwoCops: 공조

Appendix

삼성 S1 보안로봇 UBTECHAIMBOT Knightscope K5/K1 가 뭔데

삼성 S1 보안로봇(국내): [SMNR] 청소하고 집도 지키는 영상 보완 로봇 청소기!

UBTECHAIMBOT(중국): UUV-C 소독 기능, 자율 주행 및 센서 탑재, 모니터링 및 보고서 기능, 비접촉 작업 대응 하는 로봇

Knightscope K5/K1: K5: 야외용 자율 보안 로봇. 24시간 순찰 가능하며, 비디오/센서/탐지 기능을 갖춤.
K1 Hemisphere: 정지형(스테이셔너리) 보안 장치로, 특정 지점(예: 출입구, 복도 등)에 설치되어 사람·물체 탐지 기능을 수행.

<https://news.samsung.com/kr/449>

https://manuals.plus/ko/ubtech/ah0061-aimbot-virus-protection-robot-manual?utm_source=chatgpt.com#ubtech_ah0061_aimbot_virus_protection_robot

<https://knightscope.com/>

Appendix

실시간 카메라 스트리밍을 할때 어떻게 qos 설정을 했습니까? 그걸 쓴 이유는?

BEST_EFFORT를 사용했습니다.

영상 스트리밍은 약간의 프레임 손실보다 자연 없는 최신 화면이 더 중요하기 때문입니다.

BEST_EFFORT는 네트워크 상황이 나쁠 때 일부 프레임이 떨어질 수 있지만,

재전송을 하지 않기 때문에 딜레이 없이 실시간성이 보장됩니다.

반대로 **RELIABLE**를 쓰면 프레임 재전송 때문에 영상이 끊기거나 버퍼링이 생길 수 있습니다.

그래서 실시간 카메라 스트리밍에서는 **BEST_EFFORT**가 표준입니다.

Appendix

왜 Flask를 사용했나요? Foxglove 쓴다 했었는데?

Foxglove는 시각화엔 뛰어나지만:

MJPEG 다중 스트림 다루기 힘듦

네트워크 의존성이 높아 프로젝트와 동시에 스트리밍 하기 힘듦

UI 커스터마이징 제한적

Flask는:

카메라 스트림 2대 단독으로 가볍게 처리

UI 커스터마이징 용이

로그인 구현 가능

동영상 저장 지원