System Design Document

프로젝트 제목: Two Cops – 협동 자율 로봇 기반 도주 차량 추적·차단 시스템

팀명: Team 공조

버전: 1.0

날짜: 2025.11.21

---

1. 개요

Two Cops 시스템은 두 대의 자율 이동 로봇(TurtleBot4)이 협력하여 도주 차량(RC-Car)을 추적·차단하는 PoC(Proof of Concept)를 위한 시스템이다. 단일 로봇이 빠른 속도의 대상(10km/h)을 추격하는 것은 물리적으로 불가능하므로, **협동(공조) 기반 전술**을 통해 지형 구조(골목) 활용·포위·차단하는 구조로 설계되었다.

Two Cops는 다음의 특징을 가진다:

- robot1, robot3: 하나의 로봇이 도주 차량 실시간 탐지·추적, 차량의 좌표를 발행 / 반대편에 있는 로봇이 좌표를 구독하여 이동 및 탐지·추적
- 서버 PC, 클라이언트 PC가 각각 한 로봇에 연결, 하나의 클라이언트 PC가 모니터링 실행
- 실시간 모니터링 도구로 **Flask** 사용

본 문서는 Two Cops 시스템의 전체 아키텍처, 구성 요소, 데이터 흐름, 보안, 오류 처리 및 테스트 전략을 설계 관점에서 정의한다.

---

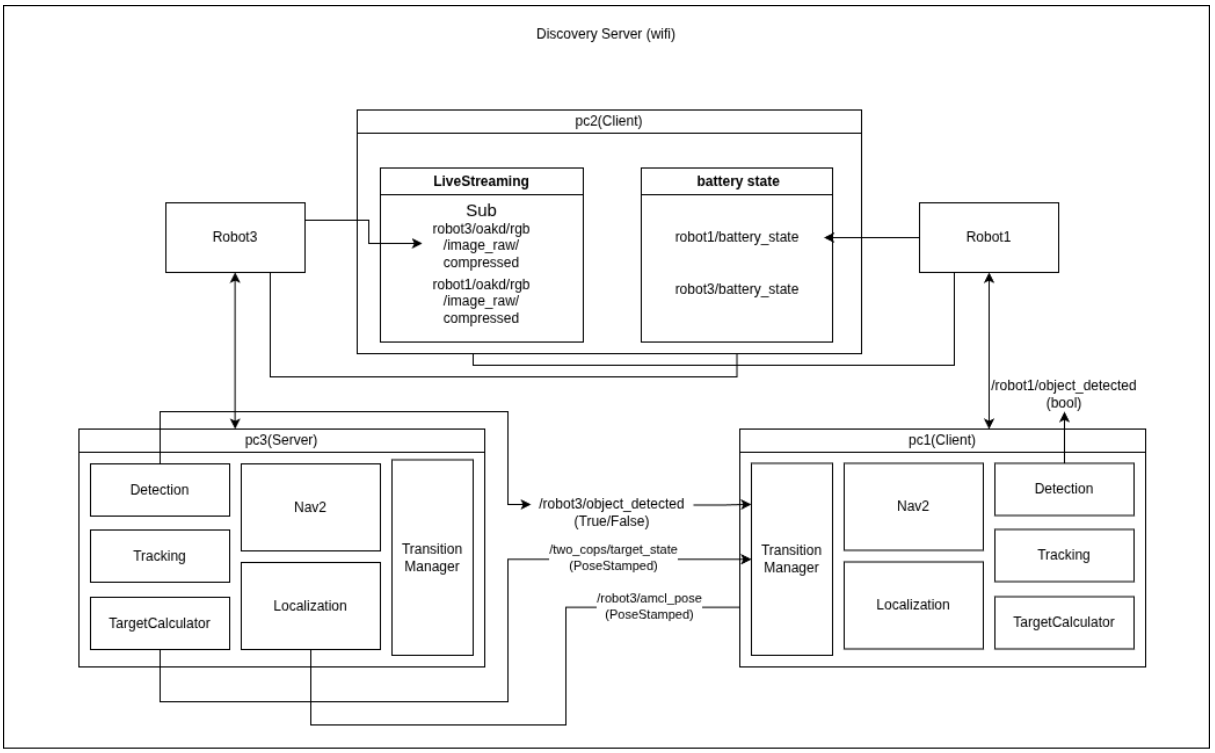
2. 시스템 아키텍처

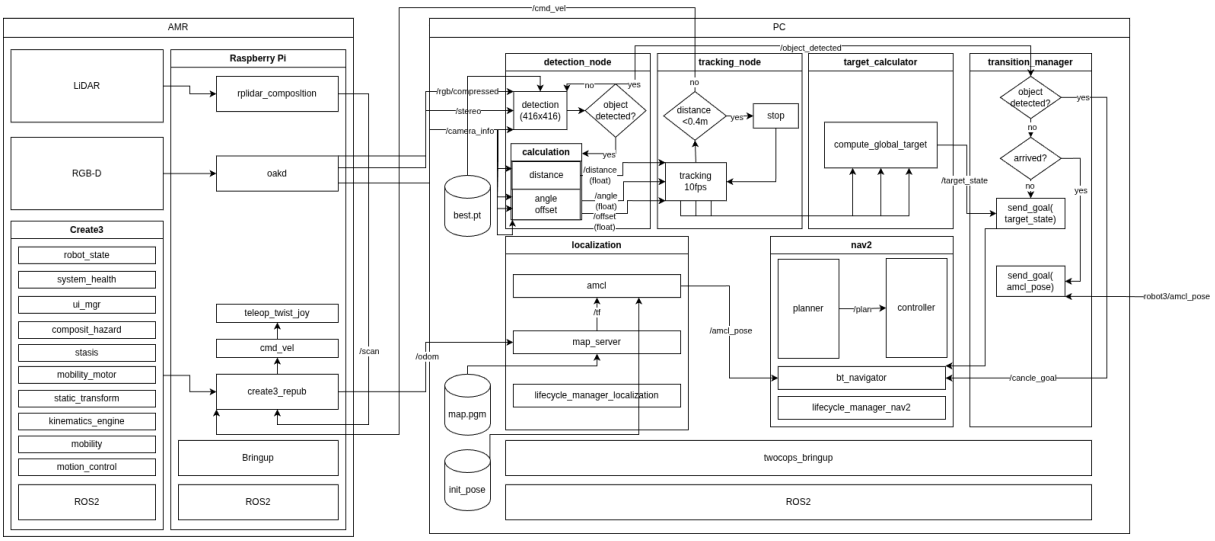
Two Cops 시스템은 서버–로봇(robot3), 클라이언트1–로봇(robot1), 클라이언트2–모니터링의 구조로 구성되며 ROS2 DDS를 통해 세 PC와 두 로봇이 동일 네트워크에서 통신한다. 두 로봇 중 최초 범죄 차량을 탐지한 로봇은 지속적으로 추적하면서 반대편 로봇에 좌표를 전송하고 반대편 로봇이 그 좌표로 이동을 수행하며 범죄 차량을 탐지, 추적을 실행한다. 마지막으로 두 로봇이 범죄 차량을 포위하고 도망가지 못하게 하면서 투캅스 시스템을 종료시킨다.

2.1 주요 설계 특징

- 하나의 robot은 **YOLO 기반 객체 탐지·추적**을 수행하며 탐지 대상 좌표에 관한 이벤트를 생성
- 이벤트(`/target_state`)는 ROS2 토픽으로 다른 로봇에 연결된 PC에게 전달
- 반대편 robot은 수신 즉시 Nav2를 이용해 **지정된 좌표**로 이동과 동시에 **YOLO 기반 객체 탐지·추적**
- 이후 두대의 robot이 **양방향 차단** 수행

2.2 고레벨 아키텍처 다이어그램





3. 구성 요소 설계

3.1 하드웨어 구성 요소

1) 서버 PC (robot3 담당)

- CPU: 8-core 8-thread
- RAM: 32GB
- GPU: NVIDIA RTX 3050
- OS: Ubuntu 22.04

2) 클라이언트 PC (robot1 담당)

- CPU: 16-core 22-thread
- RAM: 32GB
- GPU: NVIDIA RTX 4070
- OS: Ubuntu 22.04

3) 클라이언트 PC (모니터링 담당)

- CPU: Intel® Core™ Ultra 7 155H × 22
- RAM: 32GB

- GPU: Mesa Intel® Arc(tm) Graphics (MTL)
- OS: Ubuntu 22.04

4) 자율 이동 로봇 (AMR) – TurtleBot4 (robot1 & robot3 동일)

- 프로세서: Raspberri Pi 4B 4GB(On-Board Computer)
- 운영 체제: Ubuntu 22.04, ROS2
- 센서: LiDAR, RGB Camera, Stereo Camera
- 위치 추정: Odometry
- 이동속도: ~1km/h
- 배터리: 약 2–3시간

5) 네트워크

- Wi-Fi: AMR과 PC의 연결 통신

3.2 소프트웨어 구성 요소

1) PC

- Python3
- OpenCV
- CvBridge
- ROS2 Humble
- Ultralytics YOLOv8
- Detection Node: 추적 대상을 디텍션
- Tracking Node: 추적 대상을 실시간으로 트래킹
- nav_to_pose Node: 정해진 작전 위치로 주행
- Flask Studio: 실시간 영상 모니터링 및 영상 저장

1) TurtleBot4 소프트웨어

- 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
- Odometry Node: Remapping 신호 처리

---

4. 데이터 흐름 설계

4.1 사전 데이터 수집

- YOLOv8n에 AMR로 수집한 추적 대상 이미지를 학습 (`best.pt`)
- SLAM으로 map 데이터를 수집 (`map.pgm`)

4.2 데이터 처리 및 분석

- AMR - 구동부(Create3)를 제어하고 각 센서(RGB-D, LiDAR)로부터 들어오는 신호를 처리
- 원격 PC - AMR로부터 카메라와 라이다, 구동부(Creat3), Odomery의 신호를 받아, 객체 감지 및 추적, 좌표 이동을 위한 연산을 진행

1) detection_node.py

- 416×416의 사이즈로 사진을 resize하고 Oakd 카메라 센서(stereo, rgb)를 통해 YOLOv8n로 사전 학습시켜 놓은 탐지 대상을 탐지
- 탐지한 경우 `/object_detected` 토픽 발행
- Oakd 카메라 센서(stereo)를 통해 추적 대상과의 거리를 계산, `/detected_distance` 토픽 발행
- Oakd 카메라 센서(stereo, rgb)를 통해 추적 대상의 각도와 이격 계산, `/detected_angle` , `/detected_offset` 토픽 발행

2) target_calculator.py

- `/amcl_pose` 토픽을 구독하여 SLAM 기반 로봇의 전역 위치(x, y) 및 Orientation(Quaternion → yaw) 정보를 저장
- `/detected_distance` , `/detected_angle` , `/object_detected` 토픽을 구독하여 detection_node에서 계산된 추적 대상의 상대 거리(m)·각도(°)·탐지 유무를 수신
- 상대 좌표(distance, angle)를 로봇의 전역 위치(robot_x, robot_y, robot_yaw)와 결합하여 추적 대상의 **map 기준 전역 좌표**를 계산

- 계산된 전역 좌표를 `PoseStamped` 형태로 `/two_cops/target_state` 토픽으로 발행하여 **반대편 로봇이 협력 추적(nav_to_pose)** 을 수행할 수 있도록 지원

3) tracking_node.py

- `/object_detected` 토픽을 구독하여 탐지 유무를 판단
- detection_node에서 발행하는 `/detected_distance`, `/detected_angle`, `/detected_offset` 토픽을 10fps로 구독하여 AMR의 이동 제어에 활용
- 감지 중일 때는 각도(angle)와 거리(distance)를 기반으로 선형·각속도를 계산하여 `/cmd_vel` 로 퍼블리시
- 탐지 대상이 카메라에서 사라진 경우(일정 시간 object_detected 신호가 들어오지 않을 때) 마지막으로 수신한 좌표를 `/robot/target_state` 로 발행하여 반대편 로봇에게 협력을 요청
- 탐지 대상과의 거리가 0.4m 이하로 접근한 경우 더 이상의 추종을 중단하고 시스템을 정지

4) Localization

- SLAM을 통해 만들어 놓은 map, LiDaR를 통해 발행된 `/scan` 과 Odometry을 통해 발행된 `/odom`으로 로봇의 현재 위치를 Localization
- 현재 위치의 좌표를 `/amcl_pose` 로 토픽 발행

5) nav_to_pose

- 반대편 로봇의 협력 요청(좌표)을 BT(Behavior Tree)로 받아 `/goal_pose` 로 토픽 발행
- Planner가 `/goal_pose` ,map으로 경로를 계획하여 `/plan` 토픽 발행
- Controller가 `/plan` , `/scan` , `/odom` 을 통해 `/cmd_vel` 토픽을 발행하고 AMR 구동

4.3 모니터링 및 데이터 저장

토픽 수신 → 화면 출력 흐름

- `DualCamNode`는 다음 네 개의 ROS2 토픽을 구독:
  - `/robot1/oakd/rgb/image_raw/compressed` (robot1 카메라)
  - `/robot3/repub/compressed` (robot3 카메라 재퍼블리시 영상)
  - `/robot1/battery_state`, `/robot3/repub/battery_state` (배터리 상태)
- 카메라 콜백(`robot1_callback`, `robot3_callback`)에서:

    - `CompressedImage.data`를 `np.frombuffer()`로 가져와서
    - `cv2.imdecode()`로 디코딩하여 **OpenCV BGR 프레임**으로 변환
    - 변환된 프레임을 `self.robot1_frame`, `self.robot3_frame`에 **항상 최신 한 장만 유지**
- Flask의 `/robot1_feed`, `/robot3_feed` 엔드포인트:
    - `generate(lambda: node.robotX_frame)`를 호출해서
    - 최신 프레임을 JPEG로 인코딩(`cv2.imencode(".jpg", frame)`)한 뒤
    - `multipart/x-mixed-replace; boundary=frame` 형식으로 계속 흘려보냄 (MJPEG 스트리밍)
    - HTML에서는 `<img src="/robot1_feed">` 같은 태그로 연결해서 **끊김 없이 실시간 영상처럼 보이게** 함
- 배터리 콜백(`battery1_callback`, `battery3_callback`)에서는:
    - `msg.percentage` (0~1)를 받아서 `round(msg.percentage * 100, 1)`로 퍼센트(%)로 변환
    - `node.battery_percentage_r1`, `node.battery_percentage_r3`에 저장
    - Flask `/battery_status`에서 JSON(`{"robot1": r1, "robot3": r3}`)으로 응답 →
      프론트엔드에서 AJAX/fetch로 주기적으로 호출해 **UI에 실시간 표시**

2-2. 저장(녹화) 방식

- UI에서 **녹화 버튼**을 누르면:
    - `/toggle_record_robot1`, `/toggle_record_robot3` 라우트가 호출되고
    - 전역 변수 `robot1_enabled`, `robot3_enabled`를 `True/False`로 토글
- 각 카메라 콜백에서:
    - 해당 로봇의 `_enabled`가 `True`일 때만 녹화 로직 실행
    - `VideoWriter`가 아직 없으면:
        - `new_file(robot)`을 호출해서
            - `recordings/robot1/20250101_123456.mp4` 같은 **타임스탬프 기반 파일명** 생성
            - `cv2.VideoWriter_fourcc(*"mp4v")` 코덱, 30fps, 현재 프레임 해상도로 `VideoWriter` 초기화
    - 이후 들어오는 프레임들을 `writer.write(frame)`으로 **연속 저장**
- 녹화 종료:
    - 다시 버튼을 눌러 `_enabled`가 `False`가 되면

- 해당 `VideoWriter`를 `release()` 하고 `None`으로 초기화

---

5. 상세 설계

5.1 최초 탐지 및 추격하는 로봇

- 최초 탐지: 두 로봇 중 최초로 YOLOv8n 모델을 통해 추적 대상(RC-Car) detection
- 추격: YOLO tracking으로 타겟을 지속적으로 추적
- 협력 요청: 반대편 로봇에 추적 대상의 좌표를 지속적으로 발행
- 이동 및 경로 방어: 추격 대상이 시야에서 없어진 경우 반대편 로봇이 추격 대상을 탐지해 협력(`/robot/target_ pose`)요청을 보낼 때까지 경로 방어
- 검거: 타겟과의 거리가 0.4m인 경우 검거

5.2 협력 요청을 받아 이동 및 추격하는 로봇

- 협력: 타겟의 좌표를 받아 그 자리로 이동
- 탐지: YOLO 모델을 통해 추격 대상을 detection
- 추격: 타겟을 detection한 경우 YOLO tracking으로 타겟을 지속적으로 추적
- 검거: 타겟과의 거리가 0.4m인 경우 검거

5.3 모니터링 및 데이터 저장

- Flask 툴 사용 (Websocket)
- `/robot3/oakd/rgb/image_raw/compressed`, `/robot1/oakd/rbg/image_raw/compressed`토픽들을 구독해 실시간 카메라 스트리밍
- 실시간 들어오는 사진들을 numpy array로 변환하여 파일 저장

---

6. 보안 설계

- Wi-Fi 암호화: WPA3-Personal SAE 기반으로 비밀 번호를 알아야 접속 가능
- SSH: 관리자 IP 주소로만 접근을 제한하고, 비밀번호 인증만이 아니라 키 기반 인증을 강제
- ROS2 DDS: AMR/PC 간에만 해당 포트 범위의 인바운드/아웃바운드 트래픽 허용

- Topic 구조: `/robot3/oakd/rgb/image_raw/compressed` 와 같이 주제별로 분리하여 정확한 노드에서 오용 없이 사용
- Port 보안: '11811' 포트 번호를 사용하여 정해진 포트에서 통신하므로 방화벽/라우터에서 이 포트를 허용해 사용
- Id/Password: Flask 접속시 id/password 사용

---

7. 성능 요구사항

- YOLO 탐지: 실시간 10FPS로 들어오는 사진을 과부화 없이 처리
- 확장성: 두대의 AMR 운영만 시현해봤으나 구동 코드가 같아 여러대의 협력도 가능
- AMR 이동: AMR이 추격 차량의 좌표로 이동시 정확히 도착해야 하며, 오차는 ±20cm 이내
- AMR SLAM: SLAM을 기반으로 정확한 좌표를 가진 map을 제작
- AMR Navigation: 생성된 map을 기반으로 장애물을 회피 기동하며 정해진 위치까지 자율 주행 가능
- AMR Localization: AMR의 initilal pose를 설정하고 AMR의 좌표를 실시간으로 발행해야 하며, map과 비교해 정확해야한다.
- 거리 계산: Stereo 타입의 카메라를 통해 추적 대상과의 거리 차이는 0.2m 이내의 오차로 계산
- 각도 계산: Stereo, RGB 카메라를 통해 추적 대상과의 각도 차이는 5도 이내의 오차로 계산
- 이격 계산: Stereo, RGB 카메라를 통해 추적 대상과의 이격 차이는 0.1m 이내의 오차로 계산
- 추격 간격: AMR과 추격 대상과의 최소 간격은 40cm 오차는 ±10cm 이내

---

8. 오류 처리 및 복구

- 대상 탐지 손실 오류: 추격 대상을 최초 탐지 후 대상이 맵의 장애물 뒤로 숨은 상황이면 대상을 마지막으로 잃은 좌표까지 이동, 반대편 로봇역시 탐지 불가능할 시 반대편 로봇의 좌표로 이동하면서 탐지
- Localization 오류: 오류로 인해 navigation이 불가능한 경우 선행 조건인 ODOM_FRAME, BASE_LINK_FRAME, LASER_FRAME, SCAN_TOPIC,

MAP_TOPIC, TIMEOUT을 차례로 체크하는 프로그램(`prep_checker`)을 실행시켜 navigation이 실행 가능한 환경인지 검증

- 하드웨어 오류: AMR에서 발행되는 토픽을 지속적으로 점검 및 보수
- 네트워크 오류: AMR과 연결된 PC로 ping을 확인하여 평균적으로 100ms의 수치가 넘어가면 프로그램 종료 및 점검

---

9. 테스트 및 검증

9.1 단위 테스트

- YOLO 탐지 테스트: 실시간 10FPS로 들어오는 사진을 실시간으로 탐지 가능한지 확인
- AMR 이동: AMR이 추격 차량의 좌표로 이동시 정확히 도착해야 하며, 오차는 ±20cm 이내인지 확인
- AMR SLAM: SLAM을 기반으로 정확한 좌표를 가진 map을 제작
- AMR Navigation: 생성된 map을 기반으로 장애물을 회피 기동하며 정해진 위치까지 자율 주행 가능
- AMR Localization: AMR의 initilal pose를 설정하고 AMR의 좌표를 실시간으로 발행해야 하며, map과 비교해 정확해야한다.
- 거리 계산: Stereo 타입의 카메라를 통해 추적 대상과의 거리 차이는 0.2m 이내의 오차로 계산
- 각도 계산: Stereo, RGB 카메라를 통해 추적 대상과의 각도 차이는 5도 이내의 오차로 계산
- 이격 계산: Stereo, RGB 카메라를 통해 추적 대상과의 이격 차이는 0.1m 이내의 오차로 계산
- 추격 간격: AMR과 추격 대상과의 최소 간격은 40cm 오차는 ±10cm 이내

9.2 통합 테스트

- tracking → loss → backup coordinate → nav 이동

9.3 시나리오 테스트

- 실제 골목 환경에서 RC-Car 도주 상황 재현
- 로봇 검거 성공률 검증

9.4 모니터링 테스트

- Flask에서 모든 토픽 정상 렌더링

---

10. 배포 및 유지보수 계획

10.1 배포 단계

- 서버/클라이언트 PC ROS2 환경 통합 버전으로 설치
- 프로젝트 Bringup 자동화 스크립트 구성
- 모든 AMR, PC 동일 Wi-Fi 연결

10.2 유지보수

- 매월 AMR의 하드웨어, 배터리, 센서 점검
- 분기마다 소프트웨어 업데이트 및 호환성 점검
- 프로젝트 종료 후 User guide 및 bug report 문서화

---

11. 부록

- 라이브러리 및 종속성: ROS2, YOLOv8, OpenCV, Flask 등 사용된 소프트웨어 종속성 목록
- 참고 문헌: ROS2 공식 문서, OpenCV 공식 문서, Flask 공식 문서, Turtlebot4 공식 문서