

robocallee

Rotus

8 September, 2025

Table of Contents

1	기획	8
1.1	Map 구성도.....	8
1.2	Robocallee 통합 코드 구성도 및 빌드 유의사항.....	9
1.2.1	디버깅 유의사항	9
1.2.2	코드 사용 규칙.....	10
1.3	1차 기획안	14
1.4	2차 기획안	16
1.5	Test Plan.....	20
1.6	interface specifaication.....	25
2	설계	26
2.1	User Requirements - 25.07.08.....	26
2.1.1	User Requirements	26
2.2	System Requirements 25.07.11.....	27
2.3	HW Architecture.....	31
2.4	SW Architecture	32
2.5	ER Diagram.....	34
2.6	Class Diagram.....	36
2.7	Sequence Diagram.....	36
2.8	State Diagram	40
2.8.1	Robot Arm State Diagram	41
2.8.2	AMR State Diagram	41
3	Technology Survey.....	43
3.1	FMS(로봇 관제 시스템)	43
3.1.1	Open RMF.....	43
3.1.2	FMS 기능	43
3.1.3	RMF(Robot Middleware Framework)	44
3.1.4	Open RMF 작업 코드	46
3.2	C++로 VS Code & OpenCV & rclcpp	46
3.2.1	예제 코드 추가	50

3.3	jet sub pub 코드	50
3.4	Hand-Eye Calibration & 좌표 변환	52
3.4.1	파악한 문제점.....	52
3.4.2	Hand Eye Calibartion 플로우	52
3.4.3	개요	52
3.4.4	AX = XB 이해	53
3.4.5	필요한 데이터 구하기	53
3.4.6	미터 단위일 경우(최종 코드 아님)	57
3.4.7	타겟의 위치를 base 좌표계로 변환	65
3.4.8	joint 값 받는 코드	79
3.4.9	Hand Eye Calibration (Python code).....	80
3.5	관제 카메라 실행 코드 (bmp파일로 저장)	85
3.5.1	관제 캠 (검은색) 캘리	86
3.5.2	관제 캠 (흰색) 캘리	88
3.5.3	Numpy	91
3.6	Motion Planning.....	91
3.6.1	전체 개념 요약	92
3.6.2	💡 핵심 포인트	92
3.6.3	MyCobot Motion Planning.....	94
3.7	내부 파라미터 & 왜곡계수 & Reprojection error	94
3.7.1	카메라 Calibration으로 내부 파라미터와 왜곡 계수 구하기	95
3.7.2	(두번째 관제 캠)내부 파라미터 & 왜곡계수 & Reprojection error.....	113
3.8	다중 로봇 경로 계획 정리본	117
3.9	음성 인식, LLM 관련	117
3.10	이미지 처리, 카메라 관련	117
3.11	다중 주행	117
3.11.1	다중 로봇 경로 조사	119
3.11.2	MAPF-Multi Agent Path Finding (A* 확장)	119
4	스터디	123
4.1	선형대수	123
4.1.1	선형대수-(기본+심화).....	123
4.2	관제	123

4.2.1	다중 로봇 경로 기술 조사 (MAPF-Multi Agent Path Finding).....	123
4.2.2	behavior tree(구조도)	143
4.3	AMR	152
4.3.1	PID 개념 및 코드 정리.....	152
4.3.2	pinky nav2 공부.....	152
4.3.3	Nav2 패키지	161
4.3.4	Cartographer 빌드	163
4.3.5	pinky gazebo	165
4.4	ROBOT ARM	167
4.4.1	카메라로부터 위치와 자세 추출:.....	167
4.4.2	DH Parameter & 변환행렬 // OpenCV로 QR인식	168
4.4.3	로봇팔 임의 좌표	188
4.5	VISION	191
4.5.1	Homography.....	191
4.5.2	ROS2 OpenCV c++ build	192
4.5.3	YOLOv8 + ROS2 (Python ver.)	197
4.5.4	ArUco marker + RPi 카메라 + Flask (작성중).....	205
4.5.5	Visual C++, about 비트맵.....	214
4.5.6	Anaonda 사용법	217
4.6	서비스 기능	219
4.6.1	음성 명령 with SST, GPT API (작성 예정 아마 ~ 7/13?).....	219
4.7	강의자료/튜토리얼	222
4.7.1	Jecobot Rviz 환경구축	222
4.7.2	ROS2 humble 공부 자료.....	222
4.7.3	Tutorials - ROS Jazzy.....	222
4.7.4	핑크랩 강의목록	224
5	Sprint 기록.....	227
5.1	<RT-18> <곽제우> <traffic manager(멀티 로봇 경로 계획) 설계> <7/15>	227
5.1.1	멀티로봇 동시에 NAV2 수행.....	227
5.1.2	Terminal.....	227
5.1.3	MAPF:멀티 로봇 경로 계획.....	228
5.1.4	구현	240

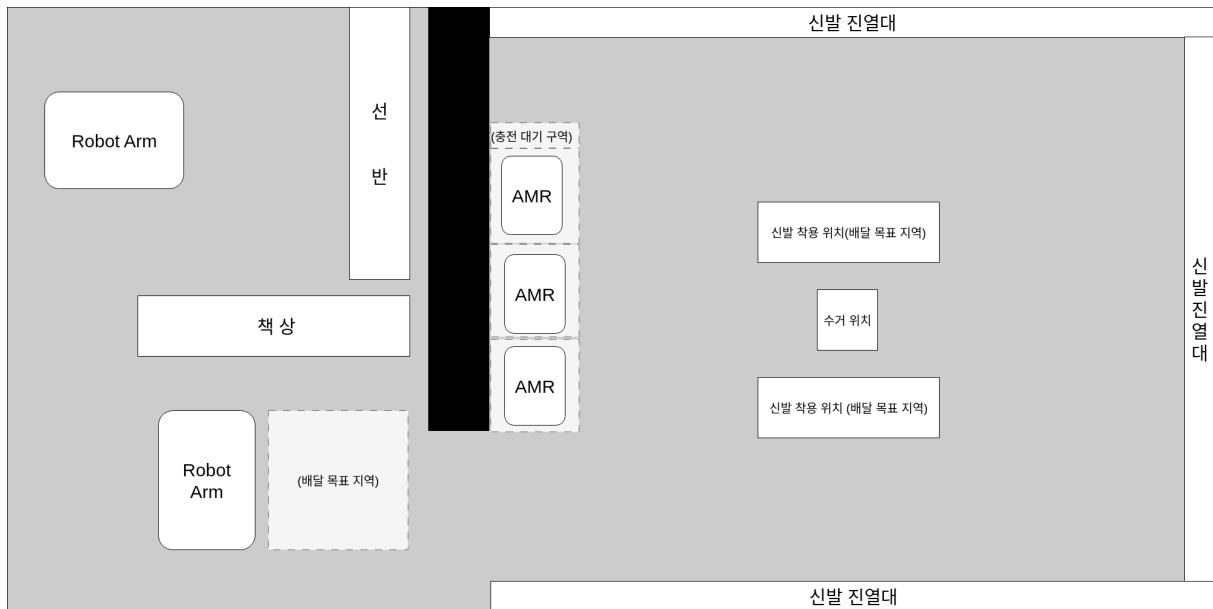
5.1.5 1단계: waypoints 기준 A*	240
5.2 <RT-8> <문자언> <로봇팔_카메라 캘리브레이션> <7/11 내용>	248
5.2.1 DH Parameter를 사용한 카메라와 엔드이펙터 사이의 변환 행렬 구하기	248
5.2.2 메모 및 개념 정리 (전반 흐름 설명 포함)	250
5.3 <RT-37> <이찬종> <OCR & Detection> <7/31>	252
5.4 <RT-17> <이창연> <멀티 로봇 위치 ROS2 연동(Nav2 package)> <7/16>	256
5.5 <RT-22> <문자언> <로봇팔 - 비전 캘리브레이션> <7/11 >	257
5.5.1 X는 해(solution)지 정답과 비교할 수 있는 GT(Ground Truth)가 없음	257
5.6 <RT-19> <이은혜> <SW 구체적 설계 (통신 , 구조)> <25.07.18>	260
5.7 <RT-27><황다연><PID제어>	262
5.7.1 setup.py	265
5.8 <RT-27> <곽제우> <PID제어 적용> <2025.7.22>	266
5.8.1 로봇 속도를 0.6배로 감속	266
5.8.2 PID 대신 nav2_params.yaml 수정	266
5.8.3 aruco maker 기준으로 P제어 적용	267
5.8.4 nav_to_goal_pid.py	267
5.8.5 코드 실행 Terminal	271
5.8.6 setup.py	272
5.8.7 localization_launch.2xml	273
5.8.8 initialpose_bringup_launch2.xml	274
5.9 <RT-31> <곽제우> <로봇 waypoint 추종> <2025.7.22>	276
5.9.1 MAPF 실행	276
5.9.2 Global path 지그재그 현상	283
5.9.3 GoalPoint Approaching Control	284
5.10 <RT-17> <곽제우> <멀티 로봇 위치 ROS2 연동(Nav2 package)> <2025.7.22>	286
5.10.1 ROS 버전 Aruco Maker 실행 방법	286
5.10.2 ARUCO MAKER 설정	286
5.10.3 Tracked_pose VS 관제 카메라 Robot_pose	289
5.10.4 TF이란?	301
5.10.5 결과 확인!	305
5.11 <RT-23> <박민수> <로봇팔_Moveit> <7/28>	307
5.12 <RT-35> <문자언> <jetcobot 소프트웨어 개발> <7/31>	310

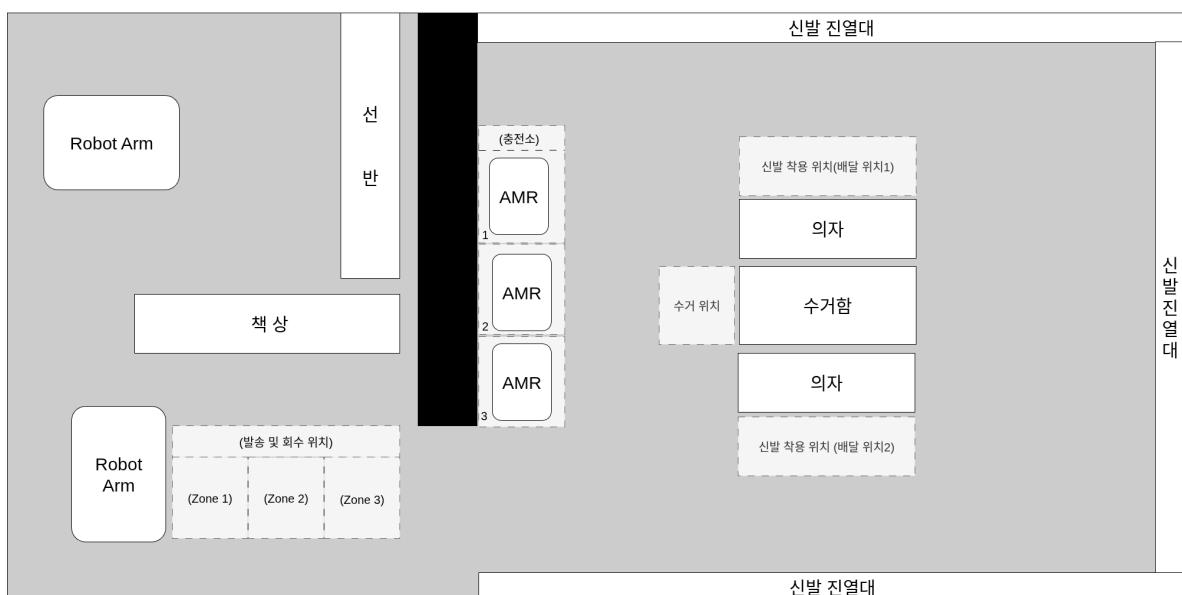
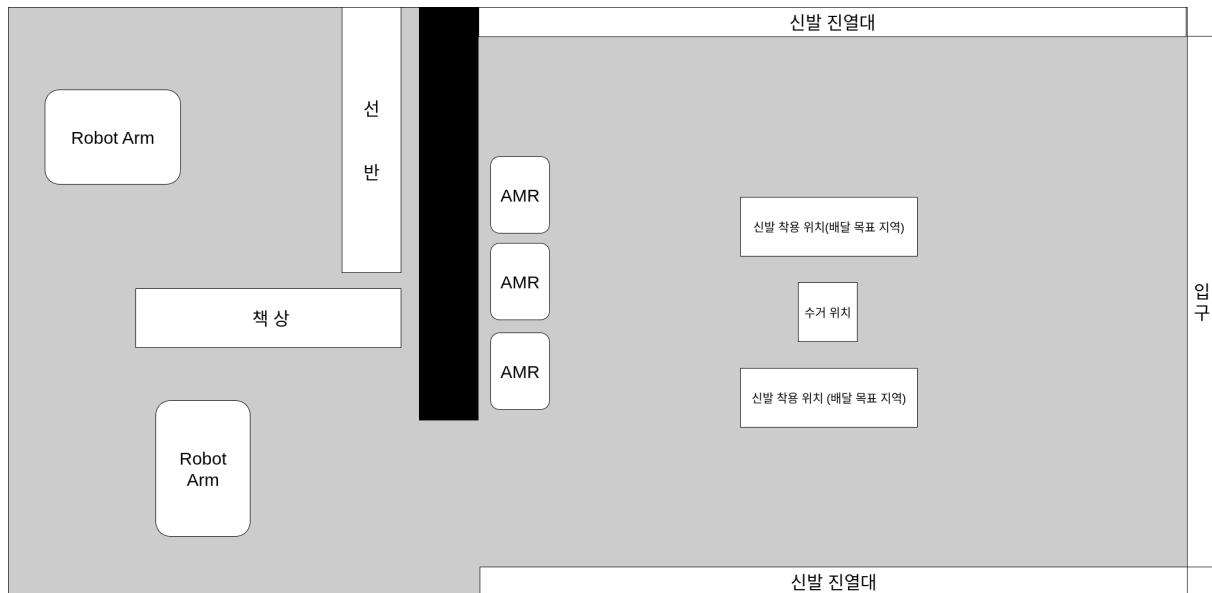
5.12.1 코드 플로우 -pick and place	310
5.12.2 위치	312
5.13 <RT-35><선태욱> 추가 정리	313
5.13.1 1번 문제 - 카메라 버퍼 문제	314
5.13.2 1번 문제 해결	315
5.14 <RT-36> <황다연> <AMR-ADAPTER>.....	316
5.15 <RT-XX><이창연><PID_GUI.origin.ver맵>	317
5.16 <RT-29> <이찬종> <UI 및 서버 개발> <7/31>	317
5.17 <RT-39> <곽제우> <Traffic Planner 개발> <8/4>	318
5.18 <RT-38> <이찬종> <Robot Arm Adapter 개발> <8/5>	318
5.19 <RT-41> <문지언> <robot arm 연동 테스트> <8/4>	319
5.19.1 로봇팔 환경 구성	319
5.19.2 버퍼에 따른 로봇팔 보정값	322
5.20 <RT-34><이창연><PID_GUI맵 subplot.ver>	324
5.21 <RT-40> <이은혜> <Vision 통한 회피주행> <25.08.06>.....	325
5.21.1 [ZMQ 설치]	325
5.21.2 [YOLO 설치].....	326
5.21.3 [UDP, ZMQ를 사용한 이미지 전달]	326
5.22 <RT-41> <문지언> <robot arm 연동 테스트> <8/14>.....	340
5.22.1 로봇팔2 동작 영상	340
5.22.2 로봇팔1 동작 영상	340
6 Robot 실행 메뉴얼	342
6.1 AMR(pinky) 실행 코드 정리	342
6.1.1 SLAM	342
6.1.2 Navigation	343
6.1.3 P 제어 이동	345
6.1.4 Domain Bridge 예제.....	352
6.2 ROS_DOMAIN_ID.....	353
6.3 통합 코드 빌드 & Git 명령어	354
6.3.1 통합 코드 빌드	354
6.3.2 오동이 계속 들어오는지(ROS IF 변환 결과)	357
6.3.3 goal_mover가 속도 내는지	358

6.4	Django Server 실행 명령어	358
6.4.1	[Git clone].....	358
6.4.2	[가상환경 설정].....	359
6.4.3	[장고 환경 설정].....	360
6.4.4	[서버 실행 방법].....	360
6.4.5	계정 생성???	361
6.4.6	Django 서버와 로봇팔 연동 후 OCR 연산	365
6.5	Robot Arm Service 실행	371
6.5.1	<생략 가능한 부분>	371
6.5.2	이미 있는 환경 이용하기	371

1 기획

1.1 Map 구성도





1.2 Robocallee 통합 코드 구성도 및 빌드 유의사항

1.2.1 디버깅 유의사항

- gdb 설치되어 있어야함(디버깅 툴?)

```
sudo apt install gdb
```

- debug할 패키지에서 source install 되어 있어야함

- ~/.bashrc에 다음 넣어 놓기

```
source ~/roscamp-repo-3/robocallee_fms/install/local_setup.bash
```

1.2.2 코드 사용 규칙

- Class 대문자로 쓰기
- Callback 함수는 앞에 cb 붙이기
ex)

```
void RosInterface::cbDoneService(const std::shared_ptr<DoneServiceType::Request>
request, std::shared_ptr<DoneServiceType::Response> response)
```

-

1.2.2.1 robocalle_fms 파일 구조

robocallee_fms/	:root 폴더
└── Adapters/	:실제 로봇의 관제 제어 로직이 들어간 폴더
├── include/	
│ └── AmrAdapter.hpp	:AMR 관제 로직이 들어간 헤더파일
└── src/	:Robot Arm 관제 로직이 들어간 헤더파일
└── AmrAdapter.cpp	:AMR 관제 로직이 구현된 파일
└── RobotArmAdapter.cpp	:Robot Arm 관제 로직이 구현된 파일
└── Core/	:FMS 프로그램의 life cycle 및 통합 관리
├── include/	
│ └── Commondefine.hpp	:프로그램 전반에 사용되는 구조체 및 define
└── src/	:Core의 인터페이스(순환참조 방지를 위해 사용- mediator pattern)
└── Core.cpp	:Core의 로직이 구현된 파일
└── Interface/	
├── include/	
│ └── RosInterface.hpp	:ros 통신과 관련된 로직 정의(ros dependency는 모두 여기만 include 할 것)
└── src/	:ros 통신과 관련된 로직(ros dependency는 모두 여기만 include 할 것)
└── Request/	
├── include/	
│ └── RequestManager.hpp	:서버에서 요청받은 request를 AMR에 분배해주는 로직 정의
└── src/	

```

    └── RequestManager.cpp      :서버에서 요청받은 request를 AMR에 분배해주는 로직 구현

    └── Task/
        └── include/
            ├── Dispatcher.hpp
            └── Executor.hpp
        └── src/
            ├── Dispatcher.cpp
            └── Executor.cpp

    └── Traffic/
        └── include/
            └── Adapter.hpp

    └── srv/
        ├── DoneMsg.srv
        └── ShoeRequest.srv

    └── log/                  # 로그 저장 디렉토리
    └── _build/                # 중간 빌드 결과
    └── build/                 # colcon 빌드 출력
    └── install/               # colcon install 출력

    └── CMakeLists.txt
    └── package.xml
    └── main.cpp   :프로그램의 실제 시작 위치
    └── README.md

```

*package 이름은 소문자로 해야됨

1.2.2.2 CMakeLists.txt

```

cmake_minimum_required(VERSION 3.10.0)

project(robocallee_fms VERSION 0.1.0 LANGUAGES C CXX)

set(CMAKE_CXX_STANDARD 20)

#GCC or Clang
#-Wall: 기본 경고 전부 표시
#-Wextra: 추가 경고
#-Wpedantic: 표준에 어긋나는 코드도 경고

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
    add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# 경로 설정
set(ADAPTERS_INCLUDE_DIR      "${CMAKE_SOURCE_DIR}/Adapters/include")
set(CORE_INCLUDE_DIR           "${CMAKE_SOURCE_DIR}/Core/include")

```

```

set(LIB_INCLUDE_DIR      "${CMAKE_SOURCE_DIR}/lib")
set(REQUEST_INCLUDE_DIR  "${CMAKE_SOURCE_DIR}/Request/include")
set(INTERFACE_INCLUDE_DIR "${CMAKE_SOURCE_DIR}/Interface/include")
set(TASK_INCLUDE_DIR     "${CMAKE_SOURCE_DIR}/Task/include")
set(TRAFFIC_INCLUDE_DIR  "${CMAKE_SOURCE_DIR}/Traffic/include")
set(LOGGER_INCLUDE_DIR   "${CMAKE_SOURCE_DIR}/../Utile/Logger/include")
set(UTILE_INCLUDE_DIR    "${CMAKE_SOURCE_DIR}/../Utile/include")
set(COMMON_INCLUDE_DIR   "${CMAKE_SOURCE_DIR}/../Utile/Common/include")

find_package(ament_cmake REQUIRED)                      # ament 빌드시스템 사용
find_package(rclcpp REQUIRED)                         # ROS2 C++ 클라이언트 라이브러리
find_package(std_msgs REQUIRED)                       # 표준 메시지 타입들 (e.g.
std_msgs::String)
find_package(rosidl_default_generators REQUIRED)       # 서비스 메시지 생성에 사용

# include 경로 등록
include_directories(
    ${ADAPTERS_INCLUDE_DIR}
    ${CORE_INCLUDE_DIR}
    ${LIB_INCLUDE_DIR}
    ${REQUEST_INCLUDE_DIR}
    ${INTERFACE_INCLUDE_DIR}
    ${TASK_INCLUDE_DIR}
    ${TRAFFIC_INCLUDE_DIR}
    ${UTILE_INCLUDE_DIR}
    ${LOGGER_INCLUDE_DIR}
    ${COMMON_INCLUDE_DIR}
)

# 서비스 메시지 생성
rosidl_generate_interfaces(${PROJECT_NAME}
    "srv/ShoeRequest.srv"
    "srv/DoneMsg.srv")

add_executable(robocallee_fms_node
    main.cpp
    ../Utile/Logger/src/Logger.cpp
    Core/src/Core.cpp
    Task/src/Dispatcher.cpp
    Task/src/Executor.cpp
    Interface/src/RosInterface.cpp
    Adapters/src/AmrAdapter.cpp
    Adapters/src/RobotArmAdapter.cpp
    Request/src/RequestManager.cpp
)

target_include_directories(robocallee_fms_node PUBLIC
    ${CMAKE_CURRENT_BINARY_DIR}/rosidl_generator_cpp
)

# 실행 파일(robocallee_fms_node)를 사용하는 ROS2 패키지들을 link 해줌
ament_target_dependencies(robocallee_fms_node
    rclcpp
)

# 필요한 OpenCV 모듈 수동 연결 (필요한 .a/.so 파일 이름만 작성)
target_link_libraries(robocallee_fms_node
)

```

```

rclcpp::rclcpp
${std_msgs_TARGETS}
)

install(TARGETS
robocallee_fms_node
DESTINATION lib/${PROJECT_NAME})

#메시지 타입이(자동 생성된 헤더)을 이 실행 파일(robocallee_fms_node)이 사용할 수 있도록 연결
rosidl_target_interfaces(robocallee_fms_node
${PROJECT_NAME} "rosidl_typesupport_cpp")

ament_package()

```

#경로 설정

set(ADAPTERS_INCLUDE_DIR "\${CMAKE_SOURCE_DIR}/Adapters/include")

에서 CMAKE_SOURCE_DIR는 CMakeLists.txt가 존재하는 폴더

set은 ADAPTERS_INCLUDE_DIR을"\${CMAKE_SOURCE_DIR}/Adapters/include"로 맵핑

header 파일 경로를 include_directories에 넣어줘야함.

```

include_directories(
${ADAPTERS_INCLUDE_DIR}
${CORE_INCLUDE_DIR}
${LIB_INCLUDE_DIR}
${REQUEST_INCLUDE_DIR}
${INTERFACE_INCLUDE_DIR}
${TASK_INCLUDE_DIR}
${TRAFFIC_INCLUDE_DIR}
${UTILE_INCLUDE_DIR}
${LOGGER_INCLUDE_DIR}
${COMMON_INCLUDE_DIR}
)

```

add_executable에 cpp파일 위치 추가해 줘야함.

```

add_executable(robocallee_fms_node
main.cpp
../Utile/Logger/src/Logger.cpp
Core/src/Core.cpp
Task/src/Dispatcher.cpp
Task/src/Executor.cpp
Interface/src/RosInterface.cpp
Adapters/src/AmrAdapter.cpp
Adapters/src/RobotArmAdapter.cpp

```

```
Request/src/RequestManager.cpp
)
```

1.2.2.3 c_cpp_properties.json

vscode에 header 파일 위치 알려주는 json 파일

include 폴더 추가시 이 위치에 경로 넣어줘야함.

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "${workspaceFolder}/Core/include",
                "${workspaceFolder}/Adapters/include",
                "${workspaceFolder}/Request/include",
                "${workspaceFolder}/Task/include",
                "${workspaceFolder}/Traffic/include",
                "${workspaceFolder}/Interface/include",
                "${workspaceFolder}/../Utile/Logger/include",
                "${workspaceFolder}/../Utile/Common/include",
                "/opt/ros/jazzy/include",
                "/opt/ros/jazzy/include/rclcpp"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "c17",
            "cppStandard": "gnu++17",
            "intelliSenseMode": "linux-gcc-x64"
        }
    ],
    "version": 4
}
```

1.3 1차 기획안

과정명	[심화] ROS2와 AI를 활용한 자율주행 로봇개발자 부트캠프 (2025.06.23~2025.08.22)
작성일	09 Apr 2025
프로젝트명	선반위에 물건을 비전을 사용해 인식하고 Pick & Place 가능

팀명	Rotus
팀원	@박승우 @CJ Lee @곽제우 @황다연 @이은혜 @Sun TaeUk @문지언 @박민수 @이창연
프로젝트 기간	23 Jun 2025 ~ 14 Jul 2025
프로젝트 목표	<ul style="list-style-type: none"> 카메라에서 보이는 물체를 인식하고 그위치로 로봇팔을 이동시키는 기능 구현 선반 위에 물건을 학습 할 수 있도록, 중간 경유지를 추가하여 기구물에 간섭이 없도록 구현
프로젝트 배경	<ul style="list-style-type: none"> 현재 로봇은 사람을 수고를 덜기 위해 많은 분야에서 사용되고 있음 많은 공장에서 로봇팔을 장비에 결합하여 사용하고 있지만, 상황이 변경될 때마다 새롭게 세팅 해야하는 단점이 있음
예상 산출물	<ul style="list-style-type: none"> 지능형 로봇 제어 시스템 <ul style="list-style-type: none"> 카메라 기반 객체 인식 및 위치 추정 모듈 로봇 경로 계획 및 중간 경유지 설정 기능 로봇팔-카메라 연동을 통한 학습 동작 구현 실증 시연 자료 <ul style="list-style-type: none"> 선반 물체 자동 학습 시나리오 영상 장애물/기구물 회피 시연 영상 다중 물체 인식 및 연속 동작 시연 기술 문서 <ul style="list-style-type: none"> ROS2 기반 제어 아키텍처 문서화 알고리즘 및 시스템 구성 보고서
문제 정의	<ul style="list-style-type: none"> 물체 인식과 경로 계획이 별개로 동작하여 실시간 대응력이 낮음 기존 공장 로봇은 반복 작업에는 강점이 있지만, 환경 변화에 대한 유연한 적응 능력 부족
제안 솔루션	<ul style="list-style-type: none"> 카메라 기반 객체 인식 <ul style="list-style-type: none"> OpenCV, 딥러닝 기반 물체 검출 및 위치 추정 인식 결과를 로봇 좌표계로 변환 경로 계획 및 간섭 회피 <ul style="list-style-type: none"> ROS2 기반 MoveIt! 활용 중간 경유지 자동 생성으로 기구물 충돌 회피 학습 동작 최적화 <ul style="list-style-type: none"> 로봇팔의 그리퍼 제어 및 안정적 물체 잡기 동작 구현 다중 학습/정리 시나리오 지원 통합 제어 시스템 <ul style="list-style-type: none"> ROS2 + Python 기반 노드 아키텍처 설계 센서-로봇-제어 모듈 간 동기화

기대 효과	<ul style="list-style-type: none"> 산업적 측면: 환경 변화에 적응 가능한 유연한 로봇 제어 기술 확보 기술 확장성: <ul style="list-style-type: none"> 인간–로봇 협업(HRC) 기술 연구 기반 마련 HRI(Human-Robot Interaction) 연구에 실증 사례 제공 활용 가능성: 전시, 체험형 콘텐츠, 과학관, 로봇 캠프 등으로 확장 가능
보유 자원	<ul style="list-style-type: none"> 3060 GPU가 탑재된 PC 9대 최대 그리핑력 150g의 6축 로봇팔 2대 관제 웹캠 1대 무선 공유기 1대
필요 자원	<ul style="list-style-type: none"> 테스트 베드 구축을 위한 1m x 1m 품보드 5장 100 x 200 x 300 mm 작업대 모형 3개 3층 선반

1.4 2차 기획안

과정명	[심화] ROS2와 AI를 활용한 자율주행 로봇개발자 부트캠프 (2025.06.23~2025.08.22)
작성일	09 Apr 2025
프로젝트명	RoboCallee (Robot + Callee): 창고형 매장에서 운용하는 서비스 로봇
팀명	Rotus
팀원	@박승우 @CJ Lee @곽제우 @황다연 @이은혜 @Sun TaeUK @문지언 @박민수 @이창연
프로젝트 기간	21 Jul 2025 ~ 21 Aug 2025
프로젝트 목표	<ul style="list-style-type: none"> 다수의 로봇팔과 자율주행 로봇을 활용한 창고형 신발 매장의 완전 자동화 시스템 구축 고객 주문부터 신발 핵입, 배송, 반납까지의 전 과정을 무인으로 처리하는 통합 솔루션 개발 웹캠 기반 실시간 모니터링 및 고객 만족도 분석을 통한 서비스 품질 향상 효율적인 재고 관리와 고객 경험 최적화를 동시에 달성하는 스마트 매장 시스템 구현

프로젝트 배경	<ul style="list-style-type: none"> • 고객들이 매장 직원을 의식하여 신발 시착을 충분히 하지 못하고, 반복적인 대기 시간으로 불편감을 느끼는 문제점 해결 필요 • 인건비 상승과 24시간 운영 수요로 인한 무인 매장의 필요성 증대 • 아마존 Go, 무인 편의점 등 무인 매장 기술이 빠르게 확산되어 신발 매장으로의 적용 시점 도래 • 신발 매장 특성상 사이즈별, 브랜드별 정확한 재고 관리와 빠른 픽업이 핵심이며, 로봇 자동화가 이를 효과적으로 해결 가능 • 매장 운영 측면에서 로봇팔과 주행 로봇을 통한 재고 운반 및 신발 박스 전달 자동화로 운영 효율성 극대화 • 창고형 매장의 높은 보관 효율성과 자동화 기술을 결합한 혁신적 비즈니스 모델 구현 기회
예상 산출물	<ul style="list-style-type: none"> • 매장 내 주문 처리 시나리오 <ul style="list-style-type: none"> • 고객이 앱 주문 → 로봇팔 픽업 → 주행 로봇 배송 → 고객 수령 • 매장 내 물품 반납 시나리오 <ul style="list-style-type: none"> • 관리자 수거 요청 → 수거 물품을 주행 로봇에 상차 → 주행 로봇 배송 → 로봇팔 하차 → 자동 재고 정리 • 로봇팔-주행 로봇 스케줄 관리 시스템 • 웹캠 기반 매장 전체 모니터링 시스템 • 창고형 매장 다중 로봇 관제 및 물류 통합 관리 FMS 시스템
문제 정의	<ul style="list-style-type: none"> • 기존 신발 매장은 인건비 부담이 크고 24시간 운영이 어려움 • 고객들은 기다림 없는 즉각적인 배송을 원함 • 사이즈별, 재고별 정확한 관리와 빠른 찾기 서비스의 어려움 • 시착 후 정리, 반품 처리 등 반복적 업무로 인한 비효율성

<p>제안 솔루션</p>	<p>RoboCallee 통합 시스템 아키텍처</p> <ul style="list-style-type: none"> 핵심 구조: 3-Layer 아키텍처로 구성된 통합 로봇 관제 시스템 <ul style="list-style-type: none"> Interface Layer: 웹 브라우저 기반 운영자/고객 인터페이스 Service Layer: 메인서버(제어), AI서버(분석), 신발 DB 관리 Hardware Controller Layer: AMR, 로봇팔, 카메라 디바이스 제어 <p>핵심 운영 시나리오</p> <ol style="list-style-type: none"> 상품 배달 시나리오 (고객 주문 처리) <ul style="list-style-type: none"> 1단계 - 상품 요청: 고객이 모바일 앱에서 "Nike 260" 신발 주문 2단계 - 경로 생성: AI 서버가 재고 위치 확인 후 최적 경로 생성 3단계 - 상자 핀업: 로봇팔이 신발 진열대에서 해당 상자를 정밀 핀업 4단계 - 배달: AMR(자율주행로봇)이 상자를 고객 대기 장소로 배송 5단계 - 배달 완료: 고객이 상품 수령 후 시스템에 완료 신호 전송 상품 수거 시나리오 (직원/반품 처리) <ul style="list-style-type: none"> 1단계 - 수거 요청: 반품/교환 상품 수거 요청 접수 2단계 - 상자: AMR이 고객 위치에서 반품 상자 수거 3단계 - 배달: 수거한 상자를 검수/정리 구역으로 이송 4단계 - 하차: 로봇팔이 AMR에서 상자를 받아 검수 대기 구역 배치 5단계 - 창고내 정리: 검수 완료 후 해당 상품을 원래 진열 위치로 재배치 <p>로봇팔 시스템 (myCobot 2대)</p> <ul style="list-style-type: none"> 모델: myCobot_gripper_Ag_white 그리퍼 탑재 핵심 사양: <ul style="list-style-type: none"> 클램핑 범위: 20-45mm (신발상자 측면 파지) 최대 클램핑력: 150g (경량 신발상자 전용) 반복 정확도: 1mm (정밀한 핀앤플레이스) 담당 업무: <ul style="list-style-type: none"> ROBOT ARM 1: 고객 주문 상품 선별 및 AMR 전달 ROBOT ARM 2: 반품/시착 완료 상품 정리 및 재배치 제어 방식: 시리얼 포트 제어를 통한 ROS2 연동 <p>AMR (자율주행 모바일 로봇) 3대</p> <ul style="list-style-type: none"> 핵심 기능: 매장 내 상품 운반 및 고객 서비스 운영 방식: <ul style="list-style-type: none"> 배터리 및 거리 기준으로 우선순위를 결정해서 상품 배송 여러 주문이 동시에 들어온 경우 다중 경로를 생성해서 가용 주행 로봇이 동시에 배송 제어 방식: ROS 통신을 이용한 제어 <p>RoboCallee FMS (Fleet Management System)</p> <ul style="list-style-type: none"> 시스템 구조: <ul style="list-style-type: none"> TASK DISPATCHER: 다중 로봇 작업 할당 및 우선순위 관리 REQUEST MANAGER: 고객 요청 접수 및 처리 상태 관리 STORAGE MANAGER: 실시간 재고 위치 추적 및 관리 AMR/ROBOT ARM ADAPTER: 이기종 로봇 통합 제어 인터페이스
----------------------	---

	<ul style="list-style-type: none"> • TRAFFIC PLANNER: 로봇 간 충돌 방지 및 경로 최적화 <p>웹캠 기반 모니터링 시스템</p> <ul style="list-style-type: none"> • 기능: <ul style="list-style-type: none"> • 핸드폰으로 상품 옆에 배치된 QR를 인식해서 제품 배송 주문하는 기능 • 핸드폰으로 사이트에 들어가서 제품을 주문하는 기능 • 주행 로봇을 관리하는 기능과 긴급 정지 기능 <p>신발 데이터베이스 시스템</p> <ul style="list-style-type: none"> • DB 구조: <ul style="list-style-type: none"> • shoes_id (PK): 고유 상품 식별자 • model_name: 브랜드 및 모델명 (예: Nike 260) • size: 사이즈 정보 • color: 색상 정보 • fitting_position: 창고 내 정확한 진열 위치 • stock_quantity: 실시간 재고 수량 • 연동 기능: 로봇 작업 완료 시 자동 재고 업데이트 및 위치 정보 갱신 <p>통합 제어 및 통신 시스템</p> <ul style="list-style-type: none"> • 플랫폼: ROS2 기반 분산 처리 시스템 • 통신 방식: <ul style="list-style-type: none"> • 하드웨어 ↔ 소프트웨어: ROS2 DDS 메시지 통신 • 웹 인터페이스: HTTP 기반 REST API • 로봇 제어: TCP/UDP 및 시리얼 포트 • 핵심 기능: <ul style="list-style-type: none"> • 실시간 다중 로봇 협업 제어 • 고객 대기시간 최소화 스케줄링 • 예외상황 자동 대응 및 복구 시스템
기대 효과	<p>경영적 효과</p> <ul style="list-style-type: none"> • 인건비 절감: 24시간 운영으로 인건비 절감 • 운영 효율성: 정확한 재고 관리와 빠른 주문 처리 • 고객 만족도: 대기시간 단축 및 개인화된 서비스 제공 <p>기술적 성과</p> <ul style="list-style-type: none"> • 다중 로봇 협업 시스템의 실용적 구현 사례 • 소매업계 자동화 기술의 새로운 패러다임 제시 • 고객 행동 분석 기반 서비스 최적화 기술 개발 <p>확장 가능성</p> <ul style="list-style-type: none"> • 다른 의류/잡화 매장으로의 기술 확산 • 프랜차이즈 무인 매장 패키지 사업화 • 스마트 물류센터 기술로의 업그레이드

보유 자원	<ul style="list-style-type: none"> 컴퓨팅 자원: 3060 GPU 탑재 PC 9대 로봇팔: 최대 그리핑력 150g 6축 로봇팔 2대 모바일 로봇: 디스플레이 탑재 자율주행 로봇 3대 모니터링: 웹캠 1대 인프라: 자율주행 패키지, 무선 공유기 1대
필요 자원	<ul style="list-style-type: none"> 고해상도 웹캠 1대 주행 로봇 위에 물건을 싣기 위한 받침대 학습을 위한 고사양 GPU 컴퓨터

1.5 Test Plan

TC_ID				통과 기준	결과	상세
TC_01	엘리베이터 호출 버튼 조작	Robot GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> [배송중] 화면 테스트 단계 <ul style="list-style-type: none"> 로봇이 엘리베이터 입구에 도착 	<ul style="list-style-type: none"> 로봇 엘리베이터 입구 도착시 로봇팔 버튼 조작 시작시 [로봇팔 버튼 조작 인내] 화면 표시 엘리베이터 호출 후 [엘리베이터 호출 중] 화면 표시 	PASS	
TC_02	엘리베이터 탑승	Robot GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> [엘리베이터 호출 중] 화면 테스트 단계 <ul style="list-style-type: none"> 엘리베이터 현재 층(1층) 도착 	<ul style="list-style-type: none"> 엘리베이터 현재 층(1층) 도착시 상행 도착 시 <ul style="list-style-type: none"> [엘리베이터 탑승중] 화면 전환 하행 도착 시 <ul style="list-style-type: none"> [엘리베이터 호출 중] 화면 유지 	PASS	
		Arm Controller	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> [내기] 자세 테스트 단계 <ul style="list-style-type: none"> 로봇 위치 엘리베이터 중심점 도착 로봇이 버튼을 바라보는 방향으로 회전 	<ul style="list-style-type: none"> 로봇이 버튼을 바라보는 방향으로 회전 로봇이 목표층(6층) 버튼을 클릭 	FAIL	엔드이펙터의 지지가 약해 안눌러짐. 

TC_ID	항목	분류		통과 기준	결과	상세
TC_1	GUI 통해서 FMS에게 고객 요청 전송	Customer GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기화면 테스트 단계 <ul style="list-style-type: none"> 서비스 시작 	<ul style="list-style-type: none"> 고객 요청이 들어오면 AMRI가 지정되고 창고의 픽업존으로 이동 배달 요청이 들어온 수만큼 대기 번호부여 각 고객마다 다른 고객 번호 부여 	PASS	

TC_2	GUI 통해서 FMS에게 배달 완료 메시지	Customer GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 대기 번호표를 배정받은 화면 테스트 단계 <ul style="list-style-type: none"> 배달 서비스 완료 	<ul style="list-style-type: none"> 배달 완료 메시지가 GUI를 통해 전송되면 AMR이 충전소로 이동 	PASS	
TC_3	GUI 통해서 FMS에게 수거 요청 전송	Customer GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기화면 테스트 단계 <ul style="list-style-type: none"> 서비스 시작 	<ul style="list-style-type: none"> 수거 요청이 들어오면 AMR이 지정되고 수거존으로 이동 배달 요청이 들어온 수만큼 대기 번호부여 	PASS	
TC_4	GUI 통해서 FMS에게 수거 완료 메시지 전송	Customer GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 대기 번호표를 배정받은 화면 테스트 단계 <ul style="list-style-type: none"> 수거 완료 	<ul style="list-style-type: none"> 수거 완료 메시지가 GUI를 통해 전송되면 AMR이 창고 드랍존으로 이동 	PASS	
TC_5			<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> asdf 테스트 단계 <ul style="list-style-type: none"> ㅁㄴㄹㅇ 		PASS	
TC_6	GUI를 통해서 AMR 이동	Admin GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 화면 테스트 단계 <ul style="list-style-type: none"> 관리자 모드로 AMR 주행 	<ul style="list-style-type: none"> GUI에서 MAP에 target_pose를 찍으면 해당 위치로 이동 	PASS	
TC_	GUI를 통해서 AMR 긴급 정지	Admin GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 화면 / AMR 주행 중 테스트 단계 <ul style="list-style-type: none"> 서비스 도중 	<ul style="list-style-type: none"> GUI에서 AMR ID 번호 선택하고 긴급 정지 누르면 해당 AMR 긴급 정지 	PASS	
TC_	GUI를 통해서 AMR 투닝	Admin GUI	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 화면 테스트 단계 <ul style="list-style-type: none"> 서비스 시작 전 	<ul style="list-style-type: none"> GUI에서 AMR PID 파라미터 조절 시 AMR 최고, 최저 속도와 PID 설정 반영 	PASS	

TC_			<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> asdf 테스트 단계 <ul style="list-style-type: none"> ㅁㄴㄹㅇ 		PASS	
TC_			<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> asdf 테스트 단계 <ul style="list-style-type: none"> ㅁㄴㄹㅇ 		PASS	
TC_	로봇1이 선반에 서 버퍼로 Pick & Place	Robot Arm Controller	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 위치에 서 대기 테스트 단계 <ul style="list-style-type: none"> 버퍼에 물건 배송 완료 	<ul style="list-style-type: none"> pick 성공 여부 <ul style="list-style-type: none"> 그리퍼 닫 힘 정도 place 성공 여부 <ul style="list-style-type: none"> april tag 인식 여부 	PASS	
TC_	로봇1이 콜렉션 에서 선반으로 pick & place	Robot Arm Controller	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 위치에 서 대기 테스트 단계 <ul style="list-style-type: none"> 선반에 물건 수거 완료 	<ul style="list-style-type: none"> pick 성공 여부 <ul style="list-style-type: none"> 그리퍼 닫 힘 정도 place 성공 여부 <ul style="list-style-type: none"> april tag 인식 여부 	PASS	
TC_	로봇2가 버퍼에 서 핑키로 pick & place	Robot Arm Controller	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 위치에 서 대기 테스트 단계 <ul style="list-style-type: none"> 핑키에 물건 배송 완료 	<ul style="list-style-type: none"> pick 성공 여부 <ul style="list-style-type: none"> 그리퍼 닫 힘 정도 place 성공 여부 <ul style="list-style-type: none"> april tag 인식 여부 	PASS	
TC_	로봇2가 핑키에 서 버퍼로 pick & place	Robot Arm Controller	<ul style="list-style-type: none"> 초기 조건 <ul style="list-style-type: none"> 초기 위치에 서 대기 테스트 단계 <ul style="list-style-type: none"> 버퍼에 물건 수거 완료 	<ul style="list-style-type: none"> pick 성공 여부 <ul style="list-style-type: none"> 그리퍼 닫 힘 정도 place 성공 여부 <ul style="list-style-type: none"> april tag 인식 여부 	PASS	
TC_					PASS	
TC_					PASS	
TC_					PASS	

TC_					PASS	
TC_					PASS	
TC_	경로 생성 확인	AMR Controller	<ul style="list-style-type: none"> • 초기 조건 <ul style="list-style-type: none"> • 대기 • 테스트 <ul style="list-style-type: none"> • 서로 다른 Time step의 경로 생성 	<ul style="list-style-type: none"> • GUI를 통해서 경로를 확인 했을 때 겹치지 않는 waypoints 발생 	PASS	
TC_	목표점 이동	AMR Controller	<ul style="list-style-type: none"> • 초기 조건 <ul style="list-style-type: none"> • 대기 • 테스트 <ul style="list-style-type: none"> • 목표 및 경로 점으로 PID 제어로 이동 	<ul style="list-style-type: none"> • 각도(degree) < 5 도, 위치(x,y) 오차 < 5cm 으로 목표점 도달 	PASS	
TC_	다중 로봇 서버와 연결 확인	AMR Controller	<ul style="list-style-type: none"> • 초기 조건 <ul style="list-style-type: none"> • 대기 • 테스트 <ul style="list-style-type: none"> • 각 로봇의 제어 및 상태 값을 서버에서 받기 	<ul style="list-style-type: none"> • Domain_Bridge 를 실행 시키고 ROS2 topic list 를 실행해서 제어 및 상태 값 받음 • Admin GUI에 각각의 값이 들어옴 	PASS	
TC_	회피 주행	AMR Controller	<ul style="list-style-type: none"> • 초기 조건 <ul style="list-style-type: none"> • 이동 • 테스트 <ul style="list-style-type: none"> • 동적 물체를 회피해서 이동 다음 경로 점으로 이동 	<ul style="list-style-type: none"> • 동적 물체와 정적 물체를 회피해서 이동. 	PASS	

TC_	통합 테스트(배송)	fms 서버	<ul style="list-style-type: none"> • 초기 조건 <ul style="list-style-type: none"> • 각 amr 충전 소에 위치 • 각 robot arm 초기 자세 • 테스트 단계 <ul style="list-style-type: none"> • 고객이 GUI에서 신발 신어 보기 클릭 	<ul style="list-style-type: none"> • AMR 할당 • AMR이 창고 픽업존으로 이동 • Robot Arm 1번 선반에서 버퍼로 상품 이동 • Robot Arm 2번 버퍼에서 AMR로 상품 전달 • AMR 픽업존에서 배달 위치로 상품 운반 • 고객이 배달 완료 버튼 클릭하면 AMR은 충전소로 이동 	PASS	
TC_	통합 테스트(수거)	fms 서버	<ul style="list-style-type: none"> • 초기 조건 <ul style="list-style-type: none"> • 각 amr 충전 소에 위치 • 각 robot arm 초기 자세 • 테스트 단계 <ul style="list-style-type: none"> • 직원이 GUI에서 수거 요청 클릭 	<ul style="list-style-type: none"> • AMR 할당 • AMR이 수거존으로 이동 • 직원이 GUI에서 수거 완료 버튼 누르면 AMR이 창고 드랍존으로 이동 • Robot Arm 2번 AMR에서 버퍼로 상품 하차 • Robot Arm 1번 버퍼에서 선반으로 상품 반납 • AMR 충전소로 이동 	PASS	
TC_					PASS	
TC_					PASS	
TC_					PASS	
TC_					PASS	
TC_					PASS	

1.6 interface specification

제공서비스	From	To	프로토콜	인터페이스 항목	Request	Response	비고
AMR interface	AMR	LM	ROS2 Service	request_service	int32 requester string size string model string color float32 x float32 y int32 customer_id	int32 wait_list	GUI 요청 정보 및 목적지 좌표
	AMR	LM	ROS2 Service	done_service	int32 requester int32 customer_id	bool accepted	작업 완료 후 처리 수락 여부
	AMR	RobotArm	ROS2 Service(client)	arm1_send_request	int32 shelf_num int32 pinky_num	bool success	로봇팔 요청 성공 여부
	LM	AMR	ROS2 Topic	/aruco_pose{i}		float32 x float32 y	aruco 마커로 인식된 위치 좌표
	LM	RobotArm	ROS2 Service	arm2_service	int32 shelf_num string action int32 pinky_num string shoe_info	bool success	action: buffer_to_pinky or pinky_to_buffer

2 설계

2.1 User Requirements - 25.07.08



작성자 : 박승우 일시 : 25.07.08 17시 최종 수정

2.1.1 User Requirements

UR_ID	UR Description	Priority	Reporter
UR_01	앱 설치 필요 없이 간편한 서비스 이용	H	박승우
UR_02	직원 도움 없이 재고파악 가능	H	황다연
UR_03	서비스 내에서 직원호출 기능	L	이은혜
UR_04	빠른 결제(온라인 페이)	L	문지언
UR_05	사용자에게 도착/대기 예정시간 알림	H	박민수
UR_06	로봇이 신발 착용 위치 안내	H	선태욱
UR_07	로봇이 손님에게 물건을 직접 배송	H	이창연
UR_08	로봇이 자동으로 충전위치로 이동	H	박민수
UR_09	수거과정의 편리(직원)	H	곽제우
UR_10	착용 완료한 신발 반납과정 편리	H	이찬종
UR_11	창고에서 자동으로 신발 정리(직원)	H	황다연
UR_12	응대로봇이 호감형이면 좋겠다	M	문지언
UR_13	선택한 항목과 유사한 제품 추천기능	M	박승우
UR_14	발 특성에 따른 제품 안내	L	곽제우

UR_15	마케팅 효과를 위해 로봇팔 작업공간 시각화	M	이은혜
UR_16	로봇의 초기 세팅 편리(직원)	H	선태욱
UR_17	시스템 내에서 재고관리	M	이창연
UR_18	시스템에서 간편결제가 되었으면 좋겠다.	L	박승우
UR_19	로봇이 사람과 부딪치지 않게 안전요소가 충분했으면 좋겠다	H	박민수
UR_20	여러대의 로봇이 동시에 손님을 응대했으면 좋겠다	H	황다연

2.2 System Requirements 25.07.11

작성자 : 박승우 , 25.07.08

SR_ID	SR Name	Description	Pri or it y	비 고	UR _ID
SR_01	웹 서비스 제공	고객이 매장 입장이 별도의 어플리케이션 설치 없이 서비스를 이용할 수 있도록 하는 기능	R	QR Code	UR_01
SR_02	사용자가 원하는 신발을 검색하는 기능 (신발 검색 기능)	<p>서비스를 이용해서 고객이 원하는 신발을 검색해서 신발 정보를 확인 할 수 있는 기능</p> <ul style="list-style-type: none"> • 신발 정보 <ul style="list-style-type: none"> • 신발 모델명 • 신발 사이즈 • 신발 색상 • 재고 수량 • 검색하는 방법 <ul style="list-style-type: none"> • 바코드로 검색 • 검색어를 선택하여 필터링된 결과 검색 <ul style="list-style-type: none"> • 신발 색상 • 신발 제품명 • 신발 활용 범위 (ex 러닝화, 운동화 .etc) • 이미지 인식을 통해 검색 <ul style="list-style-type: none"> • 신발 사진 	R	신발 검색 방법 신발 바코드, 이미지 인식	UR_02 UR_17

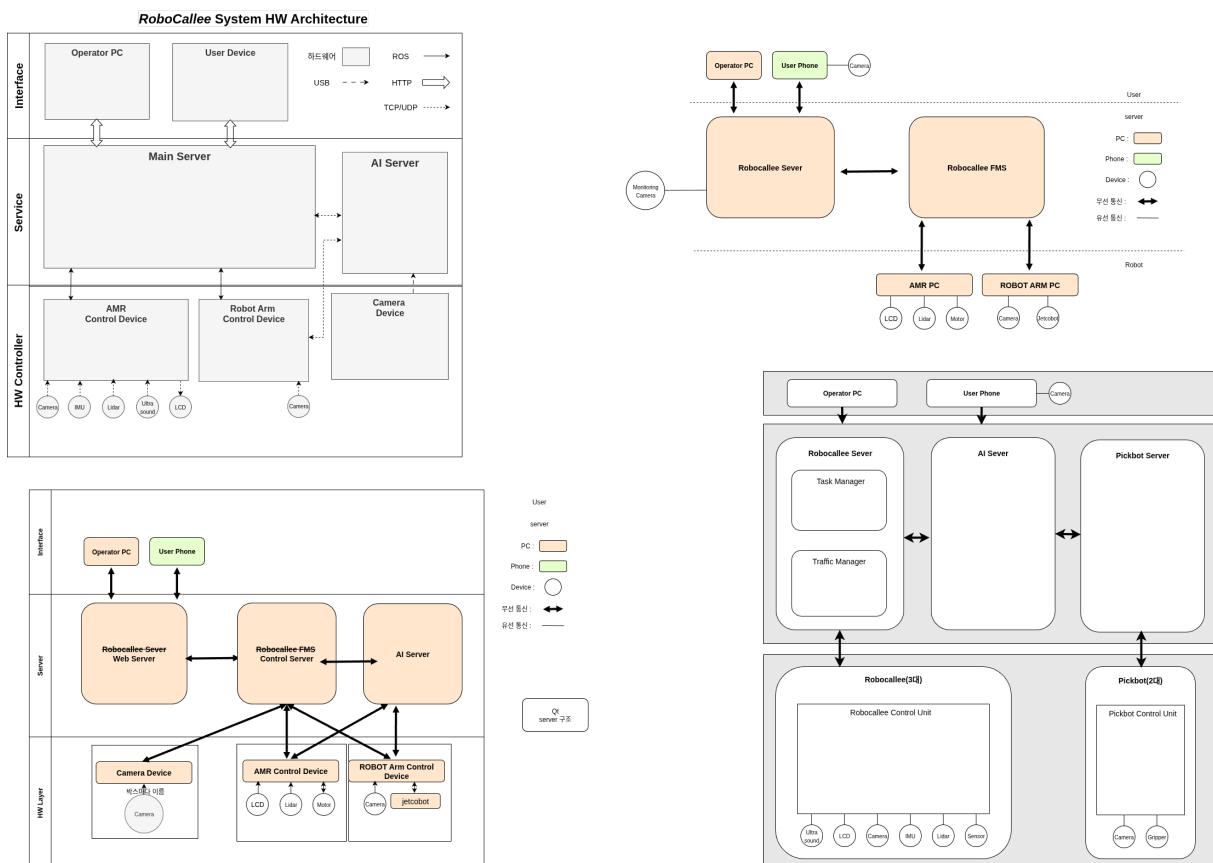
SR_ID	SR Name	Description	Priority	비고	UR_ID
SR_03	재고 요청 기능	구매를 원할시 재고 요청하는 기능 • 재고 요청 방법	장바구니 개념 (여러개 선택 가능)	UR_02 UR_17	
SR_04	예약주문 기능	현재 물건의 재고가 없는 서비스를 통해 예약주문을 하는 기능	O L		
SR_05	제품 착용 위치 안내 기능	고객이 서비스 로봇에서 요청을 했을시 로봇이 제품 착용 위치를 안내 해주는 기능 • 착용 위치 안내 • 서비스를 사용한 위치 안내 • 로봇이 움직이며 직접 위치 안내 • 음성 안내 (고려 중)	R		UR_06
		<p>i 착용 위치 : 고객이 물건을 인도 받아 물품을 착용해볼 수 있는 위치</p>			
SR_06	상품 추천 기능	서비스를 통해 고객에게 상품을 추천해 주는 기능 • 추천 기능 • 사용자 맞춤형 추천 기능 (명확하게) • 회원 정보 • 구매 이력 • 간단한 설문조사 • Best 상품 추천 기능 • 나이 및 성별 별로 추천 기능 • 고객의 발 특성 (발볼, 발등)을 고려한 추천 기능(선택)	O L		UR_14 UR_13

SR_ID	SR Name	Description	Pri or it y	비 고	UR _ID
SR_07	재고 관리 기능	<p>현재 해당 제품이 창고 및 매장에 있는지, 창고에서 출고될때마다 실시간으로 재고 관리 서비스에 반영</p> <ul style="list-style-type: none"> 사용자 주문후 로봇팔이 하차 작업 했을때 상태 변경 및 매장에서 창고로 상차 작업 했을때 상태 변경 카테고리 정리 후 신발 종류 설정 (데모 시나리오 반영) 제고 관리 데이터 (데이터 범위) 	R	신발 사이트에서 상품 목록 조사	UR_02
SR_08	창고의 재고를 자동으로 상하차 하는 기능	창고 안의 선반의 물건을 자동 인식하여 물건을 자동으로 상하차 하는 기능	R	로봇 팔이 창작되어 있는 시나리오 고려	UR_16 UR_11
SR_09	창고에서 고객에게 제품 배송 기능	재고 요청된 제품을 모바일 로봇이 고객에게 배송하는 기능	R		UR_07
SR_10	매장에서 창고로 제품 반납 기능	반납 요청된 제품을 모바일 로봇이 창고로 반납하는 기능	R		
SR_11	로봇 배터리 관리 기능 (자동충전기능)	<p>로봇이 자동으로 배터리 관리를 하고, 배터리가 부족하면 일 할당을 받지 않고, 자동으로 충전위치로 이동해서 충전</p> <ul style="list-style-type: none"> 배터리를 실시간으로 체크하는 기능 일을 하기 위해 최소한으로 필요한 배터리량을 체크하는 기능 특정 수치 기준으로 단계별 실행 <ul style="list-style-type: none"> 20% 이상 : 작업 할당 가능 20% 이하: 작업 할당 불가, 충전 위치로 이동 	R	특정 수치를 기준으로 단계별 관리	UR_08
SR_12	배달 시간 공지 기능	<p>고객이 서비스를 이용해서 물건을 배달하는 경우 배달 시간을 공지해 주는 기능</p> <ul style="list-style-type: none"> 배달 요청 후 고객에게 이동까지 걸리는 시간 공지 모든 로봇이 작업중이면 예상 대기 시간을 제공 	R		UR_05

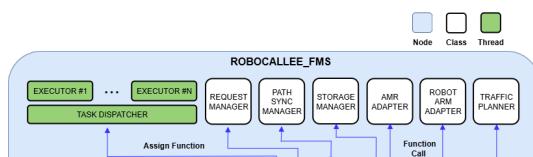
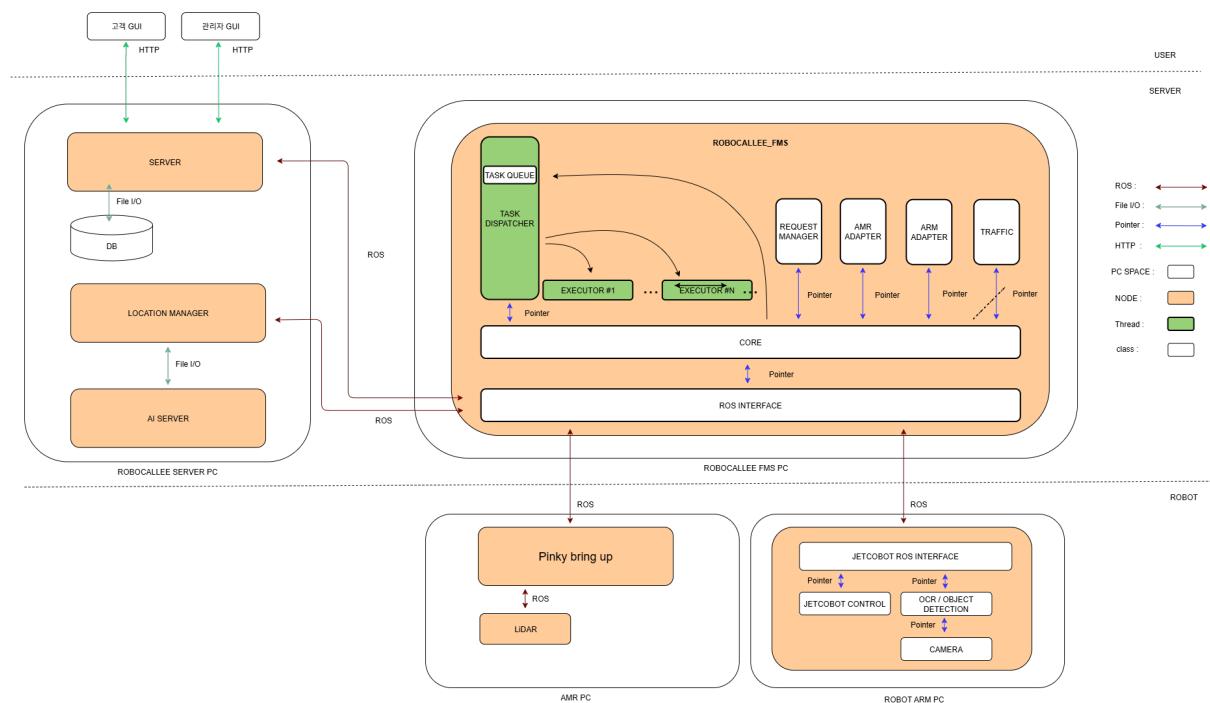
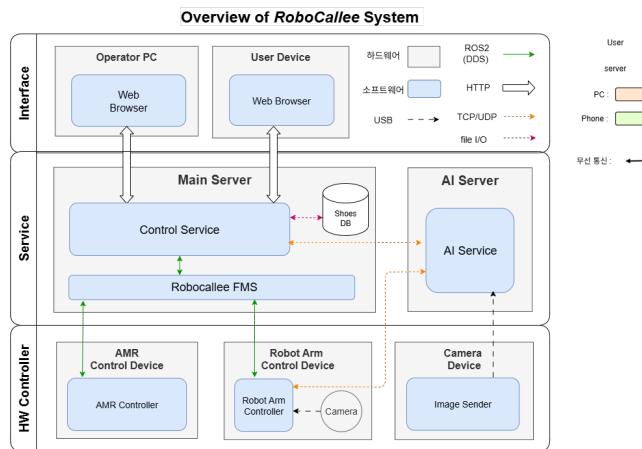
SR_ID	SR Name	Description	Priority	비고	UR_ID
SR_13	다중 로봇 관리 기능	고객의 배송요청이 들어왔을때 어떤 로봇이 작업을 실시하는지, 결정하는 기능 <ul style="list-style-type: none">• 다중로봇이 매장안에서 주행할때 서로 겹치지 않게 경로를 결정하는 기능• 어떤 로봇에 일을 할당할지 결정하는 기능	R		UR_20
SR_14	로봇의 주행중 장애물 회피 기능	로봇이 실시간으로 이동하면서 정적인 장애물 및 사람 및 상대 로봇 등 물체를 회피해서 경로를 계획하는 기능 <ul style="list-style-type: none">• 정적 장애물<ul style="list-style-type: none">• 박스• 동적 장애물<ul style="list-style-type: none">• 사람• 작업을 수행 중인 로봇	R		UR_19
SR_15	응대로봇의 표정 변화 기능	응대 로봇이 고객에게 서비스를 제공하는 동안 호감형 표정등을 화면에 표시하여 친근감을 표현하는 응대 /기능 <ul style="list-style-type: none">• 웃는 표정 : 고객 앞(목적지)에 도착하여 고객을 응대하고 있을 때• 당황하는 표정 : 고객이 주문한 제품이 재고가 없거나, 동적 장애물이 나타났을 때• 열심히 일하는 표정 : 물건을 고객에게 배달하거나, 수거할 때	O M		UR_12
SR_16	결제 기능	고객이 카운터 까지 갈필요 없이 서비스를 이용하여 구매하는 경우 온라인 간편 결제 기능 <ul style="list-style-type: none">• 카카오 페이 및 페이북 등 온라인 페이 기능을 사용한 간편 결제• 삼성페이 등 핸드폰에 등록된 카드로 서비스 상에서 간편하게 결제할 수 있는 기능	O L	제자리에서 결제 가능	UR_18
SR_17	착용 완료한 상품 반납 기능	직원이 호출 배달로봇을 호출하여 창고로 이송하는 기능	R	지게차 형태 고려	UR_10 UR_09

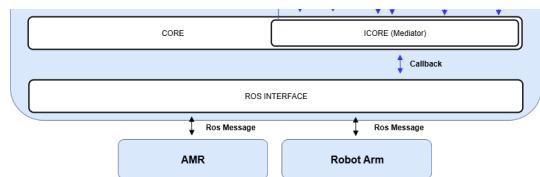
SR_ID	SR Name	Description	Priority	Bug	UR_ID
SR_18	상태 알람 기능	<p>로봇팔, 주행 로봇 상태를 주변 사람에게 항상 알람하고 있어야 한다.</p> <ul style="list-style-type: none"> • 로봇이 물건을 상하차 할때 실패 할 경우 - error 알림 • 로봇앞의 동적 장애물이 10분 이상 있어서서 로봇이 이동하지 못하는 경우 - 로봇 앞 장애물 치워 달라는 알림 	알람에 대한 인터페이스 종류 고려 (LED, 부저)		UR_12

2.3 HW Architecture

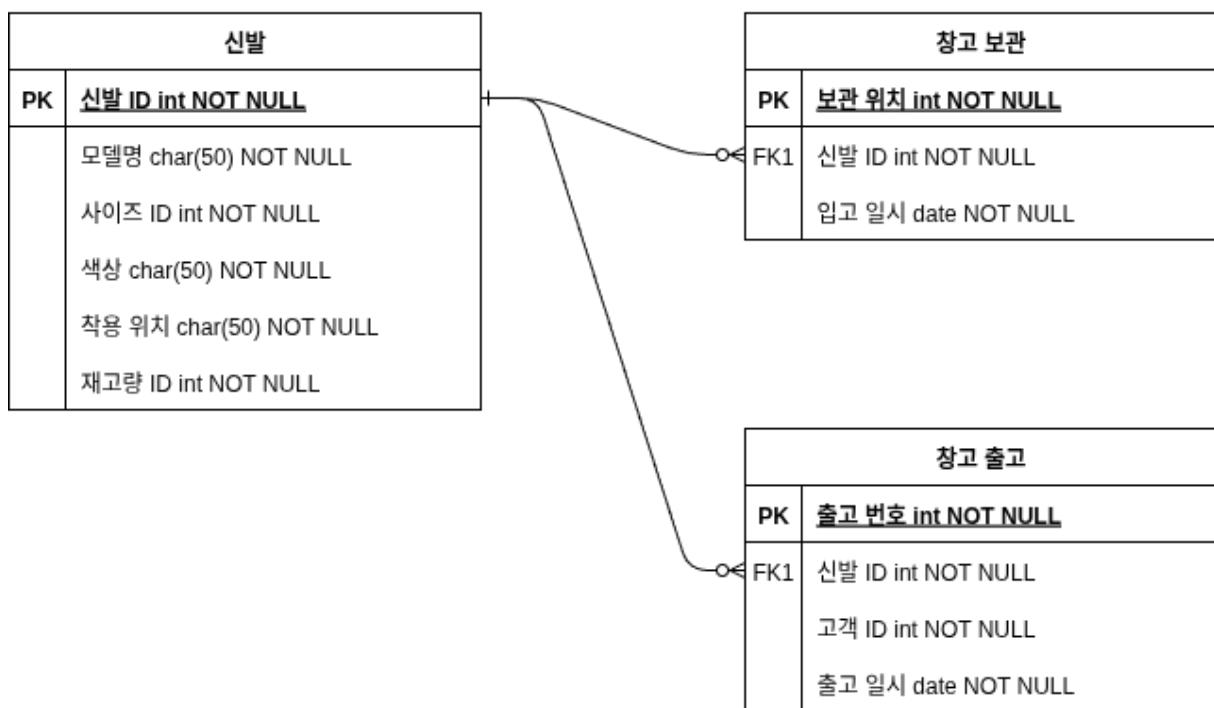
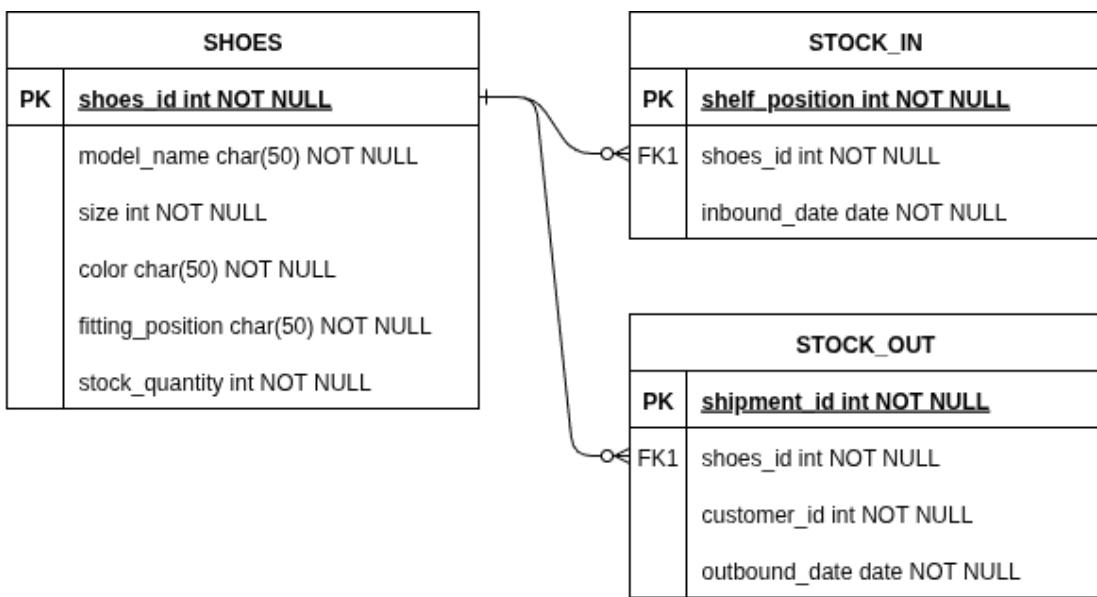


2.4 SW Architecture

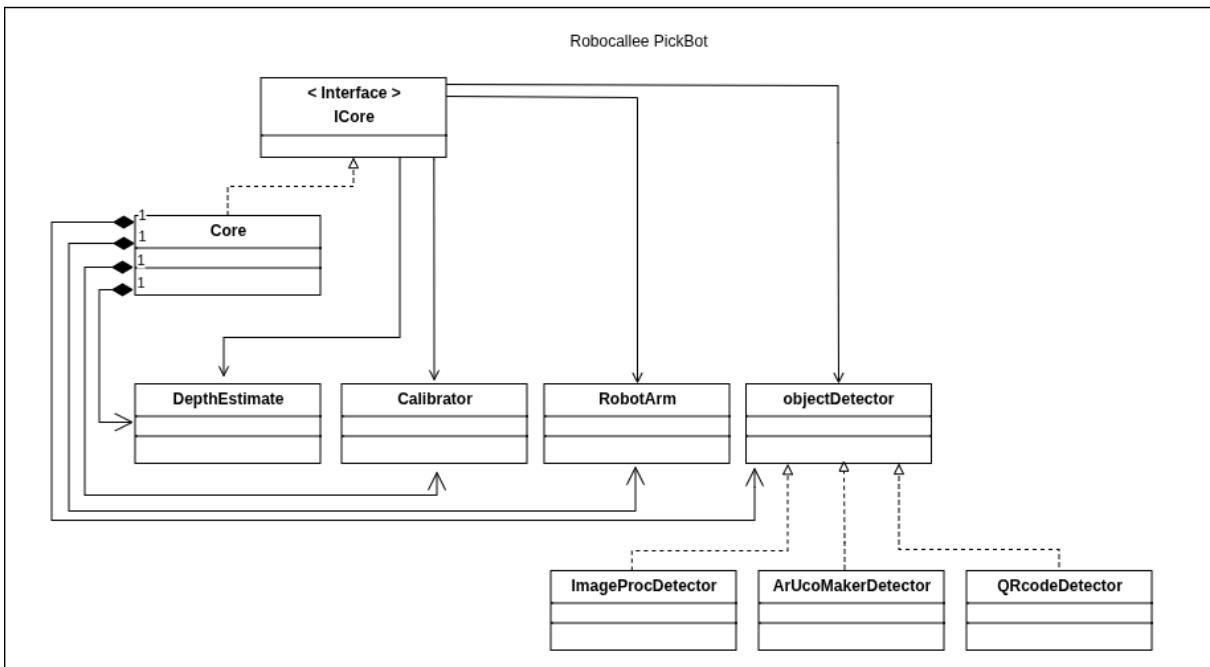
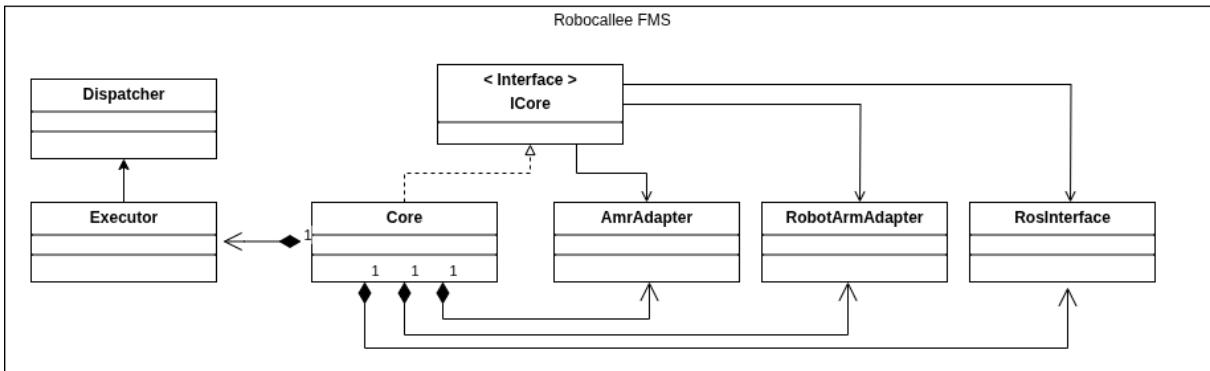




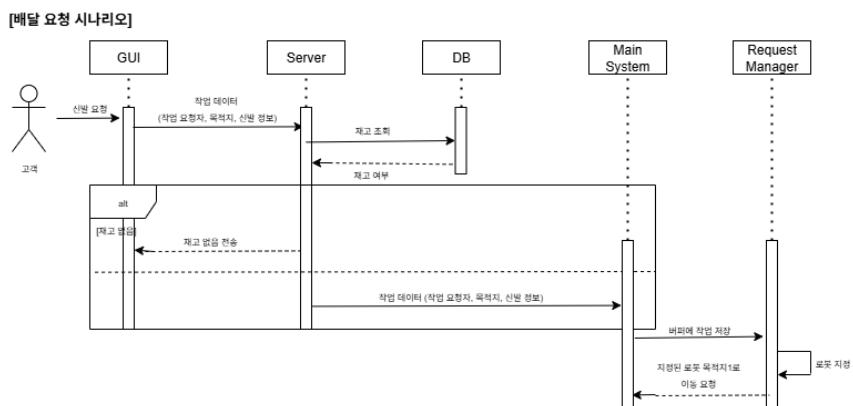
2.5 ER Diagram



2.6 Class Diagram

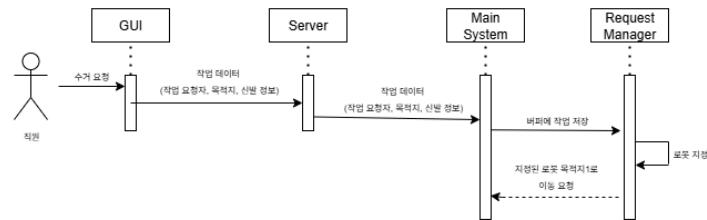


2.7 Sequence Diagram

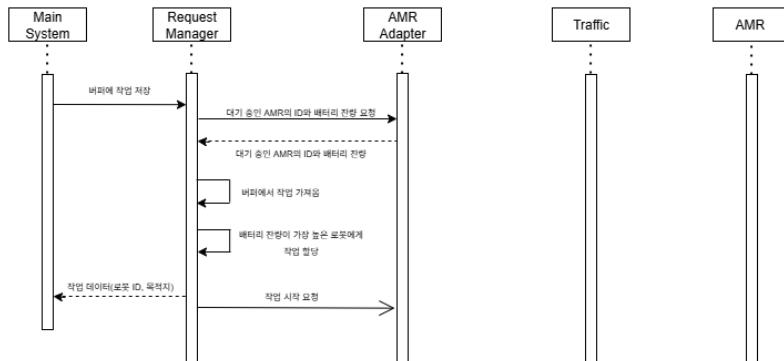




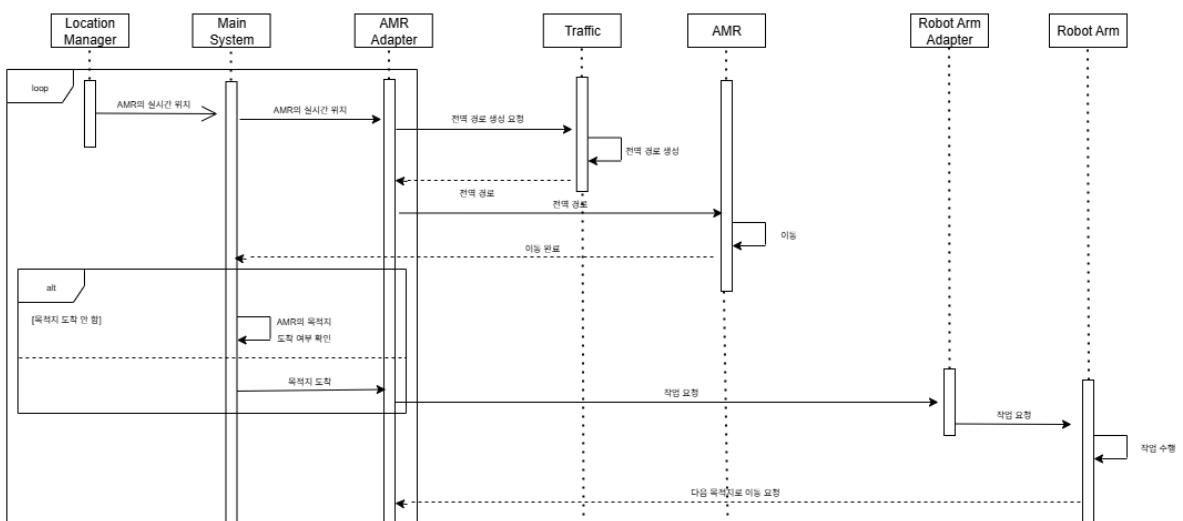
[수거 요청 시나리오]



[작업 스케줄링 시나리오]

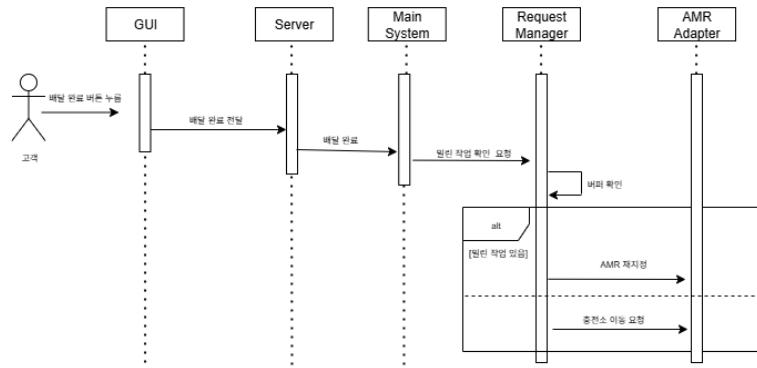


[작업 수행 시나리오]

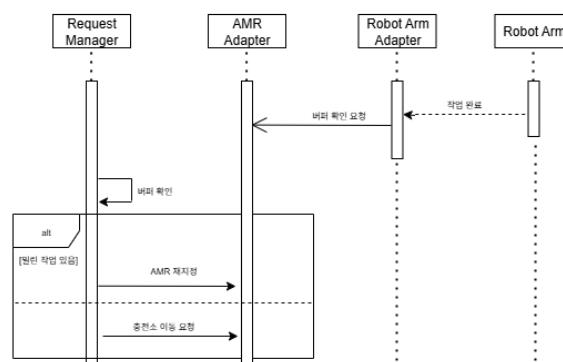




[배달 완료 확인 시나리오]

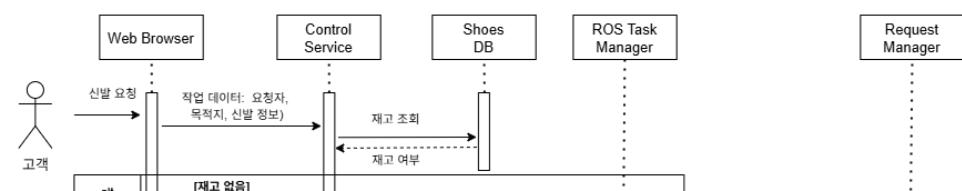


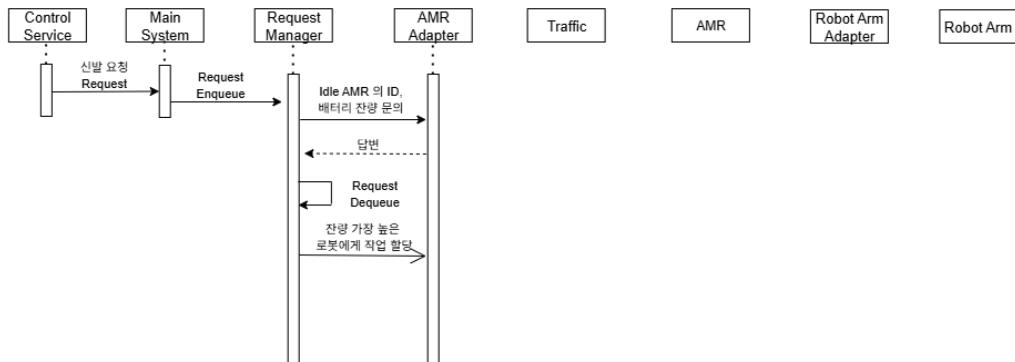
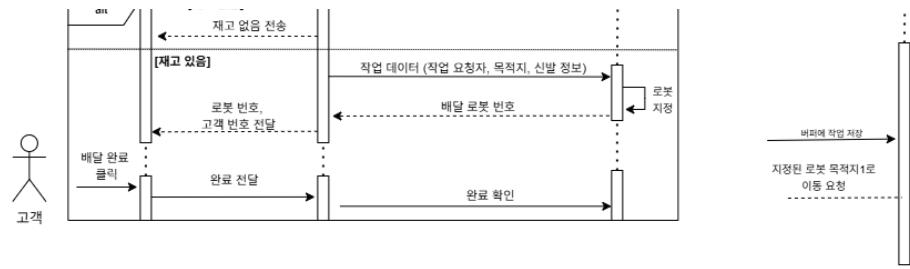
[수거 완료 시나리오]



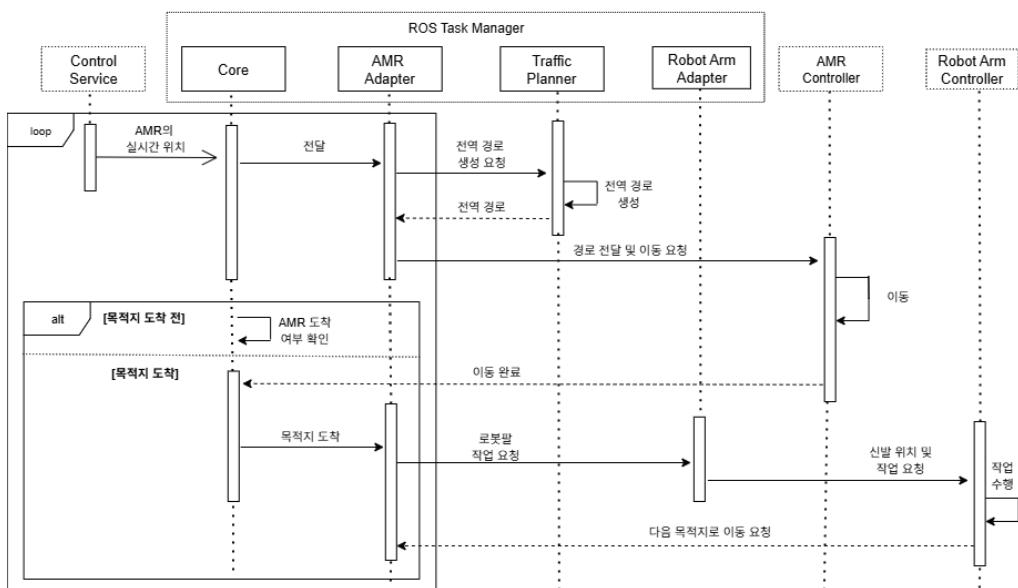
8/4 발표용

[배달 요청 - 완료 시나리오]





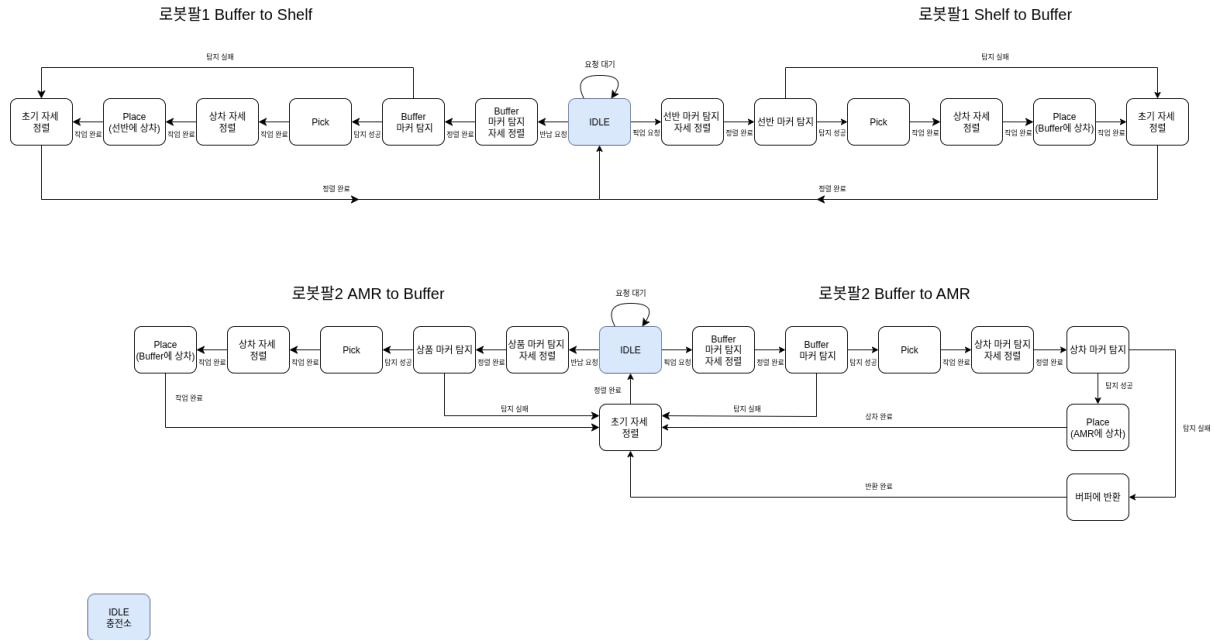
[작업 수행 시나리오]



2.8 State Diagram

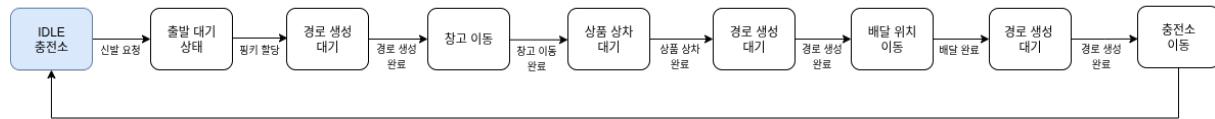


2.8.1 Robot Arm State Diagram

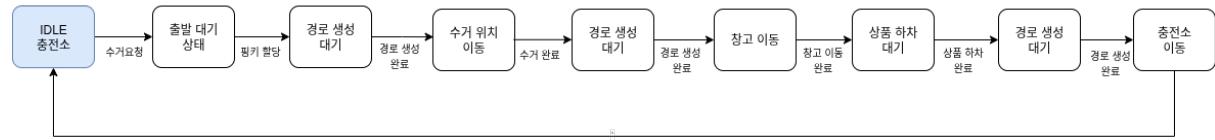


2.8.2 AMR State Diagram

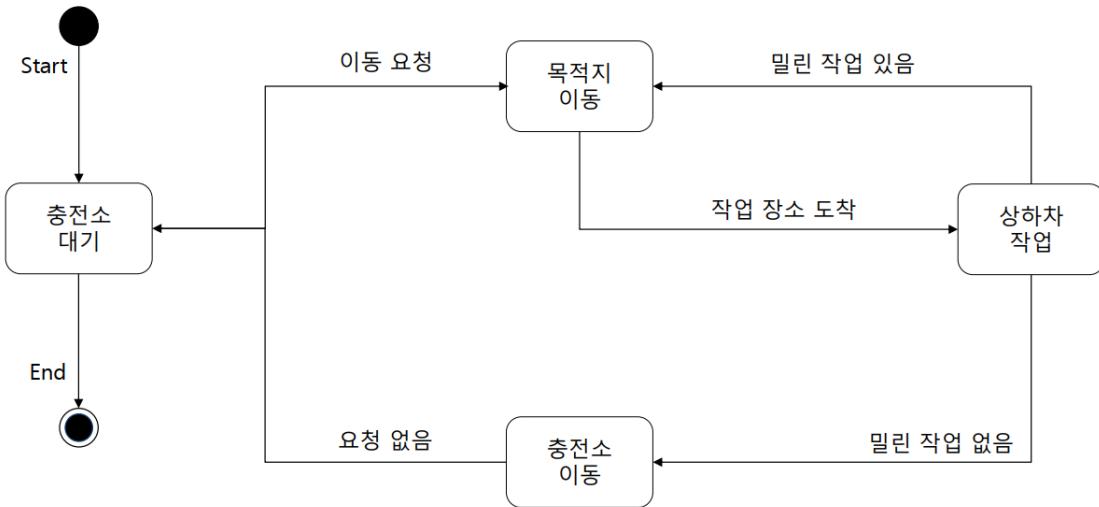
AMR 배달 로직



AMR 수거 로직

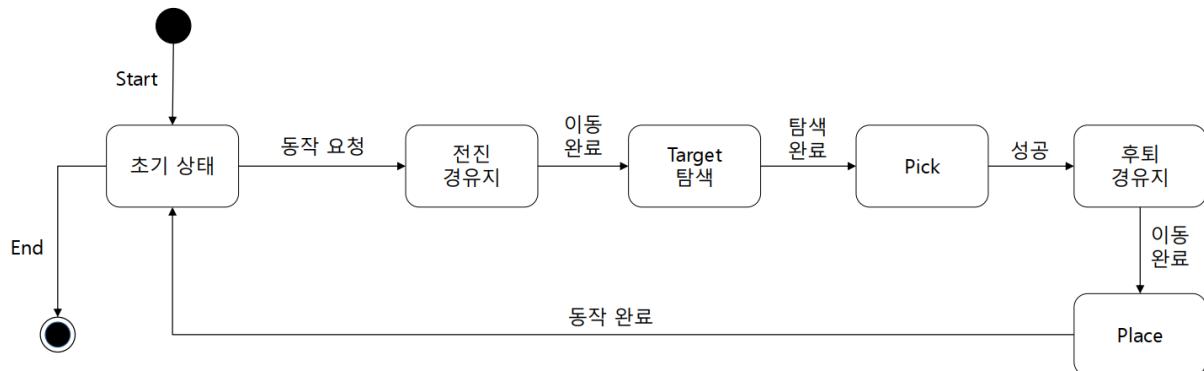


State Diagram – AMR (주행 로봇)



- 목적지 4종류 : 1) 창고 - 상/하차 2) 신발 수거함 - 상차 3), 4) 착용 장소 1, 2 - 하차

State Diagram – Robot Arm 1, 2



- 동작 4종류 : shelf_to_buffer , buffer_to_pinky , pinky_to_buffer , buffer_to_shelf

3 Technology Survey

3.1 FMS(로봇 관제 시스템)

참고 : <https://osrf.github.io/ros2multirobotbook/integration.html>

3.1.1 Open RMF

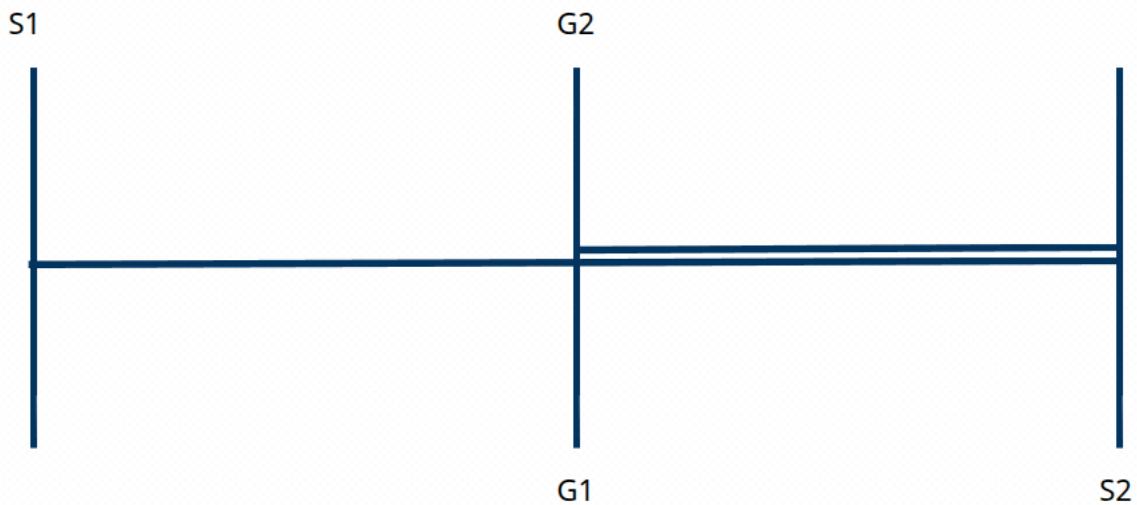
- fleet adapter을 사용해서 ROS와 연결

3.1.2 FMS 기능

3.1.2.1 Task Planner: 로봇 할당

3.1.2.2 Traffic Manager: 교차상태 해소

- RMF는 로봇 경로 지도를 사용하여 해당 환경에서 작업하는 로봇의 이동 경로를 예측한다.
- RMF는 환경 내 모든 활성 로봇에 대한 경로 예측을 생성하여 다양한 로봇 경로 계획 간의 충돌을 사전에 방지하는 데 활용할 수 있다.



3.1.2.3 Minimum Map Information Required

- 경로점 또는 노드 목록
 - 웨이포인트 이름
 - 레벨 이름(B1, L1, L2 등)
 - (x, y) 레벨 내 미터 단위 위치

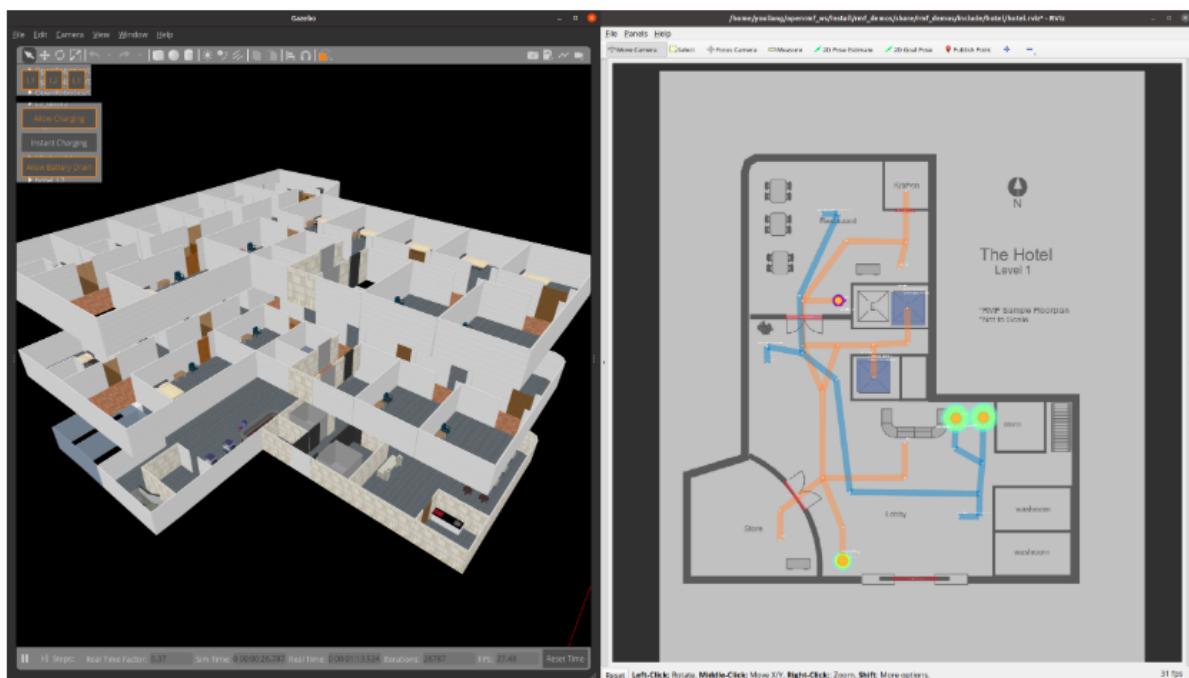
- 다음과 같은 특수 속성이나 플래그:
 - 여기가 승하차 주차장인가요?
 - 이게 충전기예요?
 - 비상 경보가 울리면 이곳은 안전한 주차 장소인가요?
- 노드 사이의 에지 또는 "이동 경로" 목록
 - (시작, 종료) 경로점 이름
 - 양방향 교통인가 일방통행인가?
 - 일방통행인 경우 이동 방향을 확인하세요
 - 이 구간의 속도 제한과 같은 기타 정보

해당 task에 FIT하게 새로 구조를 짬.

3.1.3 RMF(Robot Middleware Framework)

3.1.3.1 Open Robotics Middleware Framework (Open-RMF)

https://github.com/open-rmf/rmf_demos



Task Planner (rmf_task::agv::TaskPlanner)

- 각 Fleet Adapter가 새로운 작업 요청을 어떻게 처리할지 결정
- 배터리 충전 스케줄링, 최적 경로 계산, 작업 순서 최적화 담당
- 로봇이 연속적인 작업을 수행하기 위한 충전 필요 여부 판단

Traffic Manager (rmf_traffic)

- 로봇 간 경로 충돌 방지 및 교통 관리
- 공유 자원(엘리베이터, 문, 통로) 사용 조정
- 실시간 경로 재계획 및 데드락 해결

Dispatcher (rmf_dispatcher_node)

- 작업 할당을 위한 입찰 프로세스 관리
- 새 작업 요청 시 모든 Fleet Adapter에 `BidNotice` 전송
- 최적 제안을 선택하여 `DispatchRequest` 전송

3.1.3.1.1 2. Fleet Adapter 타입들

Full Control Fleet

- 로봇의 완전한 제어 권한
- 경로 계획, 속도 제어, 작업 실행 모두 관리

Traffic Light Fleet

- 단순한 일시정지/재시작 명령만 가능
- `EasyTrafficLight.hpp` API 기반으로 구현
- 예시: `office_mock_traffic_light.launch.xml`

Read-Only Fleet

- 로봇 위치만 추적, 제어 불가
- 수동 조작 차량(지게차 등)에 적용
- 다른 로봇들이 이를 피해서 경로 계획

3.1.3.1.2 3. 실제 구현 흐름

1. **작업 요청:** 웹 대시보드나 CLI를 통해 작업 요청
2. **입찰 과정:** Dispatcher가 모든 Fleet Adapter에 입찰 요청
3. **비용 계산:** 각 Fleet Adapter가 TaskPlanner로 비용 계산
4. **작업 할당:** 최적 제안을 선택하여 작업 할당
5. **실행:** Fleet Adapter가 로봇에 명령 전송 및 상태 모니터링

3.1.4 Open RMF 작업 코드

3.1.4.1 C++ API¹

https://osrf.github.io/ros2multirobotbook/integration_fleets.html

3.1.4.2 다중로봇 명령 할당

https://github.com/open-rmf/free_fleet

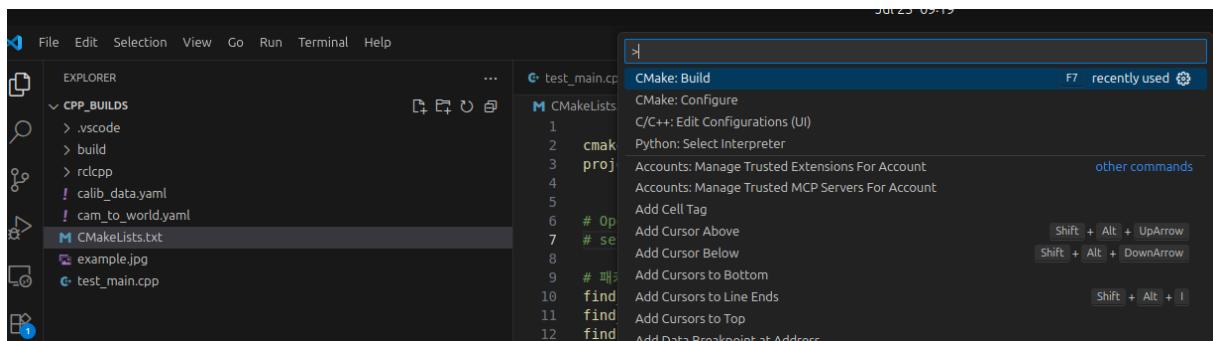
3.1.4.3 Task manager

https://github.com/open-rmf/rmf_task

3.2 C++로 VS Code & OpenCV & rclcpp

extended from [ROS2 OpenCV c++ build \(see page 192\)](#)

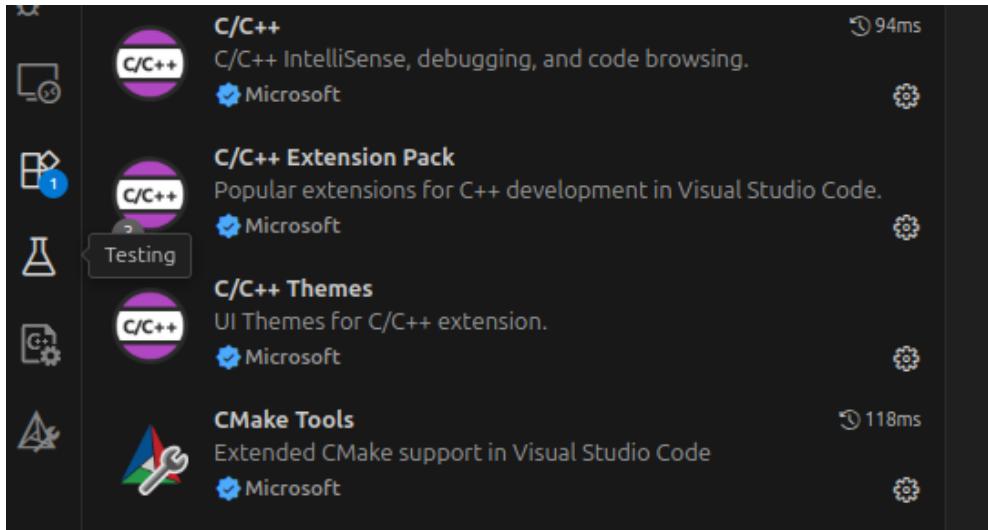
- CMake는 빌드랑 컴파일을 훨씬 편하게 해 주는 툴. 아주 간단히만 얘기해서 우선은 'gcc 명령어 칠 때 include, link 등등 해야할 수고를 덜어준다' 정도로만 이해하고 넘어가자



1 CMake: Build 하기 전에 해당 디렉토리(화면의 경우 cpp_builds)에 CMakeLists.txt 가 있어야 함

Code에서 사용 시에는 Ctrl + Shift + p로 팔레트 고르는 창에서 CMake: Build, CMake: Configure를 골라서 실행하며 된다. 근데 configure는 build만 해도 보통 저절로 되는듯

1. https://osrf.github.io/ros2multirobotbook/integration_fleets.html#c-api



2 물론 사용 전에 관련 extension을 설치해주어야 한다

CMakeList.txt의 매우 단순한 예시. Build하면 build 폴더 안에 실행 파일이 생기는데 아래의 경우는 cmake_practice 인데,

add_executable(실행 파일명 소스파일1 소스파일2 ... 헤더파일1 헤더파일2 ...)에 따른 것

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a project structure with CPP_BUILDS, build, .cmake, ament_cmake_core, ament_cmake_package_templates, ament_cmake_uninstall_target, CMakeFiles, Testing, AMENT_IGNORE, cmake_install.cmake, cmake_practice, CMakeCache.txt, compile_commands.json, CTestConfiguration.ini, CTestCustom.cmake, CTestTestfile.cmake, Makefile, rclcpp, tmp_test, and CMakeLists.txt. The CMakeLists.txt file is selected in the editor, showing the following code:

```

1 cmake_minimum_required(VERSION 3.10)
2 project(cmake_practice)
3
4
5 # OpenCV 경로 직접 지정
6 # set(OpenCV_DIR "/home/addinedu/for_opencv/build")
7
8 # 패키지 찾기
9 find_package(rclcpp REQUIRED)
10 find_package(std_msgs REQUIRED)
11 find_package(OpenCV REQUIRED)
12
13 # 실행 파일 생성
14 add_executable(cmake_practice test_main.cpp)
15
16
17
18
19 # 링크할 라이브러리
20 target_link_libraries(cmake_practice
21   rclcpp
22   ${std_msgs_TARGETS}
23   ${OpenCV_LIBS}
24 )
25
26

```

여기서 말하는 패키지는 외부 라이브러리나 모듈 가져오는 단위를 뜻하는데

find_package(패키지명) 으로 포함시킬 수 있다. (REQUIRED는 못 찾았을 때 CMake 에러 발생시키고자)

다음으로 **target_link_libraries**(실행파일명 링크할_라이브러리1 링크할_라이브러리2 ..)와 같은 식으로 해주면

```

1
2
3 #include "std_msgs/msg/string.hpp"
4 #include "rclcpp/rclcpp.hpp"
5 #include "opencv2/opencv.hpp" //opencv 헤더
6 #include <opencv2/aruco.hpp>
7
8 #include <sys/socket.h>
9 #include <arpa/inet.h>
10 #include <unistd.h>
11
12 #include <chrono>
13 #include <memory>
14 #include <string>
15 #include <iostream>
16

```

#include로 불러와서 쓸 수 있게 되고, 빨간줄은 CMake: Build를 한 번 해주면 사라진다.

<iostream>, <string>, <memory> 같은 표준 라이브러리는 CMakeLists.txt에서 추가 안 해줘도 무방하다.

- rclcpp 사용법

사실 우리는 ros2 jazzy 를 sudo apt install 명령을 통해 설치했기 때문에 추가적인 작업을 해 줄 필요는 없다.
std_msgs 의 경우도 그렇고 위의 캡처들처럼 **find_package()** 와 **target_link_libraries()** 로

간단하게 불러와서 사용이 가능, 다만 평소 ros2 쓸 때처럼 setup.bash 가 된 상태여야 한다고 한다

<https://docs.ros.org/en/rolling/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html>

<https://github.com/ros2/rclcpp>

<https://docs.ros2.org/foxy/api/rclcpp/>

사용 예시, 소스, API의 경우 위 링크들에서 확인이 가능하다

- OpenCV 빌드 및 설치하기

파이썬으로 할 때 pip install로 opencv를 설치하는 것에 대응되는 과정. 이대로 따라하면 위 내용처럼 opencv를 간단하게 불러와 C++에서도 사용할 수 있게 된다.

1. 이거 안 해주면 나중에 cv2.imshow() 같은 거 못 써서 다시 빌드해야 함

```
sudo apt update
sudo apt install -y libgtk-3-dev pkg-config
```

2. 공식 홈페이지의 **Build with opencv_contrib** 을 # Create build directory ~ 까지만 실행.

OpenCV 를 받아올 폴더를 원하는 이름으로 만들고, cd로 들어간 후 수행하길 추천한다.

https://docs.opencv.org/4.x/d7/d9f/tutorial_linux_install.html

Configure # Build 단계의 경우 맨 아래의 **Install** 부분의 Warning 을 참고해서, 만들어지는 bin, lib 등을 /usr/ 쪽이 아니라 /home/addinedu/.local 에 설치하기 위해 다음 커맨드로 대신했음

```
# Configure
cmake \
-DOPENCV_EXTRA_MODULES_PATH=../opencv_contrib-4.x/modules \
-DCMAKE_BUILD_TYPE=Debug \
-DCMAKE_INSTALL_PREFIX=$HOME/.local \
-DOPENCV_GENERATE_PKGCONFIG=ON \
../opencv-4.x

# Build
cmake --build . --parallel $(nproc)
```

지금은 Debug로 빌드하는데 나중에 쓸 땐 Release로 하게될 듯. 왜냐면 전자는 최적화가 안 된 상태라 꽤나 느림

nproc은 내 pc에서 사용 가능한 논리 cpu 갯수인데, 위와 같이 하면 생각보다 빨리 된다 16개 기준 10분 정도?
빌드 끝난 후엔 이미 들어와 있을 'build' 디렉토리에서 다음을 수행한다.

```
cmake --install . # 뒤에 . 잊지 말것!
```

하면 /home/addinedu/.local/ 아래 lib, bin 등 파일들이 복사됨

cmake 할 때 -DOPENCV_GENERATE_PKGCONFIG=ON 을 했기 때문에 **find_package**(패키지명) 으로 빌드된 OpenCV를 쉽게 불러올 수 있다.

3.2.1 예제 코드 추가

```

-- The CMake C compiler identification is Clang 11.0.0
-- The C++ C++ compiler identification is Clang 11.0.0
-- Check for working C compiler: /usr/bin/clang - works
-- Check for working C++ compiler: /usr/bin/clang++ - works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C++ compiler ABI info
-- Detecting C++ compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/robocallee/catkin_ws/src/roboarm_tutorial/build

```

빌드 후엔 build/cmake_practice 로 실행하면 된다.

3.3 jet sub pub 코드

subscriber

```

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Point

class SubscriberNode(Node):
    def __init__(self):
        super().__init__('robot_arm_subscriber') # 노드 이름
        self.subscription = self.create_subscription(
            Point, # 메시지 타입
            'robot_arm_position', # 구독할 토픽 이름
            self.listener_callback, # 데이터를 받을 때 호출될 함수
            10 # 큐 사이즈
        )

    def listener_callback(self, msg):
        # 받은 좌표를 출력
        self.get_logger().info(f'Received coordinates: X: {msg.x}, Y: {msg.y}, Z: {msg.z}')
        # 여기에 받은 좌표로 로봇팔을 이동시키는 코드를 추가할 수 있어
        self.move_robot_arm(msg.x, msg.y, msg.z)

    def move_robot_arm(self, x, y, z):

```

```

# 로봇팔을 이동시키는 코드 (여기서는 로그로 출력)
self.get_logger().info(f"Moving robot arm to: X: {x}, Y: {y}, Z: {z}")

def main(args=None):
    rclpy.init(args=args)
    subscriber_node = SubscriberNode()
    rclpy.spin(subscriber_node)

if __name__ == '__main__':
    main()

```

publisher

```

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Point

class PublisherNode(Node):
    def __init__(self):
        super().__init__('robot_arm_publisher') # 노드 이름
        self.publisher = self.create_publisher(Point, 'robot_arm_position', 10) # 퍼블리
서 생성
        self.timer = self.create_timer(1.0, self.publish_coordinates) # 1초마다 호출

    def publish_coordinates(self):
        msg = Point()
        msg.x = 10.0 # X 좌표
        msg.y = 20.0 # Y 좌표
        msg.z = 30.0 # Z 좌표
        self.publisher.publish(msg) # 메시지 발행
        self.get_logger().info(f"Publishing coordinates: {msg.x}, {msg.y}, {msg.z}")

def main(args=None):
    rclpy.init(args=args)
    publisher_node = PublisherNode()
    rclpy.spin(publisher_node)

if __name__ == '__main__':
    main()

```

3.4 Hand-Eye Calibration & 좌표 변환

[Hand Eye Calibration Python Code \(see page 80\)](#)

[타겟 좌표 -> 베이스 좌표계 변환 \(see page 65\)](#)

3.4.1 파악한 문제점

1. 만약 연산을 m로 하고 결과는 mm로 알고 싶다면, 절대 연산 중간의 단위를 건들지 말고, 마지막의 결과를 변경해야 함. (아마 지금까지 이런 실수를 계속 반복하지 않았을까 싶음)
2. target → base 좌표계를 계산할 때, joint 값을 degree로 연산하고 있었던 것도 문제가 된다고 판단 중

3.4.2 Hand Eye Calibartion 플로우

$AX = XB$ (Eye-in-Hand의 경우)

A: gripper2base (로봇 움직임)

B: target2cam (타겟 관측)

X: cam2gripper(구하려는 값)

1. Robot의 DH 파라미터를 알고 있고, Joint 값을 알고 있다.
2. 이를 활용하여 FK를 계산하여, base to gripper 행렬을 얻는다.
3. 역행렬을 구하여 gripper to base 행렬을 얻는다.
4. 카메라의 내부 파라미터와 왜곡 계수를 알고 있다.
5. 이를 활용하여 PnP 알고리즘을 통해, target to camera 행렬을 얻는다.
6. gripper to base와 target to camera 행렬을 이용하여 Hand Eye Calibration을 한다.
7. camera to gripper 행렬, 즉 X 행렬을 얻는다.

3.4.3 개요

1. 목적

Hand-Eye Calibration은 카메라와 로봇 팔 끝단(End-Effector) 사이의 상대 위치 관계를 구하는 과정입니다. 이때 구해지는 것은 두 프레임 사이의 고정된 변환 행렬 (R, t)입니다.

2. 필요한 데이터

- **R_gripper2base / t_gripper2base**

로봇의 End-Effector가 base로부터 어떤 위치에 있는지를 나타냅니다.
(로봇의 Forward Kinematics로 구함)

- **R_target2cam / t_target2cam**

카메라가 마커(Checkerboard나 ArUco 등)를 관찰해서 얻은 변환 정보입니다.
(CV에서 cv2.solvePnP() 등으로 계산)

3. 수행 절차

위 데이터를 5세트 이상 준비한 후, 아래 함수를 사용합니다:

```
R_cam2gripper, t_cam2gripper = cv2.calibrateHandEye(
    R_gripper2base, t_gripper2base,
    R_target2cam, t_target2cam
)
```

4. 결과 의미

- `R_cam2gripper` 와 `t_cam2gripper` 는 카메라가 로봇 팔에 어떻게 장착되어 있는지 (즉, 카메라 → 로봇 손의 위치)를 나타냅니다.
- 이 결과를 통해, 향후 로봇이 보는 영상 정보(예: 마커 좌표)를 정확한 로봇 좌표계로 변환할 수 있습니다.

3.4.4 $\mathbf{AX} = \mathbf{XB}$ 이해

```
A_i · X_i ≈ X_i · B_i
A_i = T_gripper2base(i)^{-1} · T_gripper2base(i+1)
B_i = T_target2cam(i)^{-1} · T_target2cam(i+1)
X_i = T_cam2gripper
```

로봇 Hand-Eye Calibration은 로봇 그리퍼의 움직임(A)과 카메라가 본 캘리브레이션 패턴의 움직임(B) 사이의 관계를 이용해, 카메라가 그리퍼에 어떻게 장착되어 있는지(X)를 찾는 과정

<https://qkrtmddn0507.atlassian.net/wiki/spaces/196616/pages/edit-v2/10453010/>

3.4.5 필요한 데이터 구하기

3.4.5.1 🔥 `R_gripper2base`, `t_gripper2base`

로봇 팔의 각 관절에 대해 DH 파라미터를 정의한 뒤, 이 함수를 사용해 각 관절의 변환 행렬을 계산.

이 행렬들을 곱해 전체 로봇의 운동학적 체인(kinematic chain)을 구함.

결과적으로, 로봇의 최종 위치와 자세를 계산할 수 있음.

```
# _____
# 1) DH 변환 함수 (기준 코드)
#
def dh_transform(theta, d, a, alpha):
    """Std-DH 4x4 homogeneous transform A_{i-1}^i"""
    ct, st = np.cos(theta), np.sin(theta)
    ca, sa = np.cos(alpha), np.sin(alpha)
    return np.array([
        [ct, -st * ca, st * sa, a * ct],
        [st, ct * ca, -ct * sa, a * st],
        [0, sa, ca, d],
```

```

        [0 ,      0,      0,      1]
    ])

# _____
# 2) myCobot 280 DH 상수 (단위: mm, rad, 기준 코드)
#
d_vals     = [131.22,   0,      0,   63.4 , 75.05, 45.6 ]    # d_i
a_vals     = [ 0 , -110.4, -96,   0 ,   0 ,   0 ]    # a_i
alpha_vals = [ 1.5708,   0,      0,   1.5708, -1.5708, 0]    # alpha_i
offsets    = [ 0 , -1.5708, 0, -1.5708, 1.5708, 0]    # theta_offset_i

```

Forward Kinematics(FK)는 로봇의 관절 각도(또는 관절 변수)를 입력으로 받아 말단 장치(end-effector)의 위치와 자세를 계산하는 과정

FK는 DH 파라미터를 기반으로 한 변환 행렬을 사용해 구현

- α (alpha): 비틀림 각도(twist angle), 두 축 사이의 회전 각.
- a : 링크 길이(link length), 두 관절 축 사이의 거리.
- d : 링크 오프셋(link offset), 한 축에서 다음 축까지의 수직 거리.
- θ (theta): 관절 각도(joint angle), 관절의 회전 각.

```

# _____
# 3) 전진 기구학 함수 (기준 코드)
#
def forward_kinematics(q):
    """
    q : 길이 6 리스트/배열, [q1..q6] (rad)
    반환: (T_BE 4x4, R 3x3, p 3x1) – base to gripper
    """
    if len(q) != 6:
        raise ValueError("6-axis 로봇이므로 q 길이는 6이어야 합니다.")

    T = np.eye(4)
    for i in range(6):
        theta_i = q[i] + offsets[i]
        A_i = dh_transform(theta_i, d_vals[i], a_vals[i], alpha_vals[i])
        T = T @ A_i

    R = T[:3, :3]
    p = T[:3, 3]
    return T, R, p

# _____
# 4) 역변환 함수 (SE(3)의 특성을 활용한 역행렬)
#
def inverse_transform(T):
    """SE(3) 행렬의 역행렬(빠른 방법)"""
    R = T[:3, :3]
    p = T[:3, 3]
    T_inv = np.eye(4)
    T_inv[:3, :3] = R.T

```

```
T_inv[:3, 3] = -R.T @ p
return T_inv
```

3.4.5.2 🔥 R_target2cam, t_target2cam

체스보드, 카메라 캘리브레이션 파라미터 설정

```
# === 체스보드 설정 ===
CHECKERBOARD = (6, 9)
SQUARE_SIZE = 0.02 # meter (Note: Will convert tvec to mm later)

# === 카메라 파라미터 로드 ===
camera_matrix = np.load("calibration_matrix.npy")
dist_coeffs = np.load("distortion_coefficients.npy")
```

6 X 9 체스판에는 54개의 검은, 흰색 사각형이 있음. → 이들의 ‘교차점’이 우리가 찾는 코너 각 코너는 ‘가로 몇칸’, ‘세로 몇칸’으로 파악

objp 를 만드는 목적 = PnP 알고리즘(3D-2D 대응 알고리즘)에 이미지의 좌표가 실제 좌표에 어디인지 알려주기 위해서

PnP는 3D공간의 점(체스보드 위 격자점)과 이미지에서 검출된 2D 코너 좌표를 짹짓고 카메라가 어떤 각도와 위치에서 이미지를 찍었는지를 계산

Z축은 체스보드가 평평(2D)하기 때문에 모두 0

objp[0] 은 (0, 0, 0),
objp[1] 은 (1 * 0.02, 0 * 0.02, 0), ...

즉, objp 에는 k번째 체스보드 격자점이 실제 어디의 (X, Y, 0)에 있는지를 저장

이 3D 좌표 리스트와, 이미지에서 검출된 2D 코너 리스트를 solvePnP() 에 넣으면 camera → target 변환 행렬을 알 수 있음

카메라가 찍은 이미지와 실제 물리적 좌표계를 연동하여 “카메라가 어디에, 어떻게 놓여 있는지” 계산하려는 것
이 objp 생성의 궁극적 목적

```
# === 3D 체스보드 포인트 생성 ===
objp = np.zeros((CHECKERBOARD[0]*CHECKERBOARD[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)
objp *= SQUARE_SIZE

# === 저장 리스트 초기화 ===
```

```
R_target2cam_list = []
t_target2cam_list = []
joint_angles_list = []
```

```
# === 이미지 경로 ===
image_dir = "/home/addinedu/handeye_test/pics2"
image_files = sorted(Path(image_dir).glob("*.png"))

# === 루프 시작 ===
for image_path in image_files:
    image = cv2.imread(str(image_path))
    if image is None:
        print(f"[X] 이미지 로딩 실패: {image_path.name}")
        continue

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD, None)

    if not ret:
        print(f"[X] 체스보드 인식 실패: {image_path.name}")
        continue

    # 코너 정제
    corners_subpix = cv2.cornerSubPix(
        gray, corners, (11,11), (-1,-1),
        (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001))

    # PnP 계산
    retval, rvec, tvec = cv2.solvePnP(objp, corners_subpix, camera_matrix, dist_coeffs)

    # 회전벡터를 행렬로
    R_cam, _ = cv2.Rodrigues(rvec)

    # !!! 여기서 tvec 단위를 미터에서 밀리미터로 변환 !!!
    # tvec는 현재 미터 단위이므로 1000을 곱하여 밀리미터로 만듭니다.
    # tvec_mm = tvec * 1000.0

    # 결과 저장
    R_target2cam_list.append(R_cam)
    # t_target2cam_list.append(tvec_mm) # 밀리미터로 변환된 tvec 저장
    t_target2cam_list.append(tvec)

    # 조인트 각도 추출
    try:
        joint_angles = ast.literal_eval(image_path.stem)
        joint_angles_list.append(joint_angles)
    except Exception as e:
        print(f"[!] 조인트 파싱 실패: {image_path.stem} → {e}")
```

3.4.5.3 🔥 cv2.calibrateHandEye() 를 통해 R_cam2gripper, t_cam2gripper 구하기

- R_cam2gripper : 카메라→그리퍼 사이의 최종 회전 행렬 (3x3)
- t_cam2gripper : 카메라→그리퍼 사이의 최종 병진 벡터 (3x1)

```
# Hand-Eye Calibration
try:
    R_cam2gripper, t_cam2gripper = cv2.calibrateHandEye(
        R_gripper2base, t_gripper2base, R_target2cam, t_target2cam, method=method
    )
except cv2.error as e:
    raise RuntimeError(f"Hand-Eye Calibration 실패: {str(e)}")
```

3.4.5.4 특이사항 메모

- opencv 버전에 따른 연산 결과 다름 확인 (4.6.0 ↔ 4.11.0)

3.4.5.4.1 문제 예상

1. 단위 불일치
 - > target2cam의 단위를 미터 단위를 밀리미터로 변경
 - > gripper2base의 도를 라디안으로 변경
2. 표본 부족
3. 조인트 각도 단위 (도 → 라디안)
4. target2cam list를 만들 때, joint_angles_list와 순서가 같은지?

3.4.6 미터 단위일 경우(최종 코드 아님)

3.4.6.1 target2cam.py

```
import cv2
import numpy as np
import os
import ast
from pathlib import Path
```

```

# === 체스보드 설정 ===
CHECKERBOARD = (6, 9)
SQUARE_SIZE = 0.02 # meter (Note: Will convert tvec to mm later)

# === 카메라 파라미터 로드 ===
camera_matrix = np.load("calibration_matrix.npy")
dist_coeffs = np.load("distortion_coefficients.npy")

# === 3D 체스보드 포인트 생성 ===
objp = np.zeros((CHECKERBOARD[0]*CHECKERBOARD[1], 3), np.float32)
objp[:, :, 2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)
objp *= SQUARE_SIZE

# === 저장 리스트 초기화 ===
R_target2cam_list = []
t_target2cam_list = []
joint_angles_list = []

# === 이미지 경로 ===
image_dir = "/home/addinedu/handeye_test/pics"
image_files = sorted(Path(image_dir).glob("*.png"))

# === 루프 시작 ===
for image_path in image_files:
    image = cv2.imread(str(image_path))
    if image is None:
        print(f"[X] 이미지 로딩 실패: {image_path.name}")
        continue

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD, None)

    if not ret:
        print(f"[X] 체스보드 인식 실패: {image_path.name}")
        continue

    # 코너 정제
    corners_subpix = cv2.cornerSubPix(
        gray, corners, (11, 11), (-1, -1),
        (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001))

    # PnP 계산
    retval, rvec, tvec = cv2.solvePnP(objp, corners_subpix, camera_matrix, dist_coeffs)

    # 회전벡터를 행렬로
    R_cam, _ = cv2.Rodrigues(rvec)

    # !!! 여기서 tvec 단위를 미터에서 밀리미터로 변환 !!!
    # tvec는 현재 미터 단위이므로 1000을 곱하여 밀리미터로 만듭니다.
    # tvec_mm = tvec * 1000.0

    # 결과 저장
    R_target2cam_list.append(R_cam)
    # t_target2cam_list.append(tvec_mm) # 밀리미터로 변환된 tvec 저장
    t_target2cam_list.append(tvec)

    # 조인트 각도 추출

```

```

try:
    joint_angles = ast.literal_eval(image_path.stem)
    joint_angles_list.append(joint_angles)
except Exception as e:
    print(f"[!] 조인트 파싱 실패: {image_path.stem} → {e}")

# === 시작화 ===
img_vis = image.copy()
cv2.drawChessboardCorners(img_vis, CHECKERBOARD, corners_subpix, ret)

# 축 그리기 (체스보드 원점 기준 XYZ) - 시작화는 그대로 미터 단위로 (또는 mm로 조정 가능)
# PnP 계산 결과인 rvec, tvec는 여전히 미터 단위로 사용
axis = np.float32([[0.05,0,0], [0,0.05,0], [0,0,0.05]]) # 5cm = 0.05 meters
imgpts, _ = cv2.projectPoints(axis, rvec, tvec, camera_matrix, dist_coeffs)
origin = tuple(corners_subpix[0].ravel().astype(int))
imgpts = imgpts.reshape(-1, 2).astype(int)
cv2.line(img_vis, origin, tuple(imgpts[0]), (0,0,255), 2) # X - Red
cv2.line(img_vis, origin, tuple(imgpts[1]), (0,255,0), 2) # Y - Green
cv2.line(img_vis, origin, tuple(imgpts[2]), (255,0,0), 2) # Z - Blue

# 창에 띄우기
cv2.imshow("PnP Result", img_vis)
key = cv2.waitKey(0)
if key == ord('q'):
    break

cv2.destroyAllWindows()

# === 결과 저장 ===
np.save("R_target2cam_list.npy", R_target2cam_list)
np.save("t_target2cam_list.npy", t_target2cam_list) # t_target2cam_list는 이제 밀리미터 단위
np.save("joint_angles_list.npy", joint_angles_list)

print(f"\n✓ 완료: {len(R_target2cam_list)}개의 유효한 이미지에서 포즈 추정함")

```

3.4.6.2 gripper2base.py

```

import numpy as np

# _____
# 1) DH 변환 함수 (기준 코드)
# _____
def dh_transform(theta, d, a, alpha):
    """std-DH 4x4 homogeneous transform A_i-1^i"""
    ct, st = np.cos(theta), np.sin(theta)
    ca, sa = np.cos(alpha), np.sin(alpha)
    return np.array([
        [ct, -st * ca, st * sa, a * ct],
        [st, ct * ca, -ct * sa, a * st],
        [0, sa, ca, d],
        [0, 0, 0, 1]
    ])
# _____

```

```

# 2) myCobot 280 DH 상수 (단위: mm, rad, 기존 코드)
#
d_vals      = np.array([131.22,    0,      0,   63.4 ,  75.05, 45.6 ]) * 0.001 # d_i
a_vals      = np.array([    0, -110.4, -96,     0 ,     0 ,     0 ]) * 0.001 # a_i
alpha_vals = [ 1.5708,    0,      0,  1.5708, -1.5708, 0]      # alpha_i
offsets     = [    0 , -1.5708, 0, -1.5708,  1.5708, 0]      # theta_offset_i

#
# 3) 전진 기구학 함수 (기존 코드)
#
def forward_kinematics(q):
    """
    q : 길이 6 리스트/배열, [q1..q6] (rad)
    반환: (T_BE 4x4, R 3x3, p 3x1) - base to gripper
    """
    if len(q) != 6:
        raise ValueError("6-axis 로봇이므로 q 길이는 6이어야 합니다.")

    T = np.eye(4)
    for i in range(6):
        theta_i = q[i] + offsets[i]
        A_i = dh_transform(theta_i, d_vals[i], a_vals[i], alpha_vals[i])
        T = T @ A_i

    R = T[:3, :3]
    p = T[:3, 3]
    return T, R, p

#
# 4) 역변환 함수 (SE(3)의 특성을 활용한 역행렬)
#
def inverse_transform(T):
    """SE(3) 행렬의 역행렬(빠른 방법)"""
    R = T[:3, :3]
    p = T[:3, 3]
    T_inv = np.eye(4)
    T_inv[:3, :3] = R.T
    T_inv[:3, 3] = -R.T @ p
    return T_inv

#
# 5) joint_angles_list.npy를 사용해 R_gripper2base, t_gripper2base 계산
#
def compute_gripper2base(joint_angles_file, angles_in_degrees=True):
    """
    joint_angles_list.npy 파일을 읽어 R_gripper2base와 t_gripper2base를 계산
    입력:
        joint_angles_file (str) - 조인트 각도 파일 경로
        angles_in_degrees (bool) - 입력 조인트 각도가도(degree) 단위인지 여부.
                                    True이면도에서 라디안으로 변환합니다.
    출력: R_gripper2base (N, 3, 3), t_gripper2base (N, 3, 1)
    """
    # 조인트 각도 리스트 로드
    q_list = np.load(joint_angles_file)

    # 입력 검증
    if q_list.ndim != 2 or q_list.shape[1] != 6:

```

```

        raise ValueError("joint_angles_list.npy는 (N, 6) 형태의 NumPy 배열이어야 합니다.")

# 각도를 라디안으로 변환 (필요한 경우)
if angles_in_degrees:
    q_list = np.radians(q_list)

# 결과를 저장할 리스트
R_gripper2base = []
t_gripper2base = []

# 각 포즈에 대해 전진 기구학 수행
for q in q_list:
    # base to gripper 변환 계산
    T_base2gripper, _, _ = forward_kinematics(q)

    # gripper to base 변환 계산 (역변환)
    T_gripper2base = inverse_transform(T_base2gripper)

    # R과 t 추출
    R = T_gripper2base[:3, :3]
    t = T_gripper2base[:3, 3]

    R_gripper2base.append(R)
    t_gripper2base.append(t.reshape(3, 1)) # t는 (3, 1) 형태로 저장

# NumPy 배열로 변환
R_gripper2base = np.array(R_gripper2base)
t_gripper2base = np.array(t_gripper2base)

return R_gripper2base, t_gripper2base

# _____
# 6) 사용 예시
# _____
if __name__ == "__main__":
    # joint_angles_list.npy 파일 경로
    joint_angles_file = "joint_angles_list.npy"

    # R_gripper2base, t_gripper2base 계산
    # joint_angles_list.npy 파일의 각도가 도(degree) 단위인 경우, angles_in_degrees=True로 설정
    R_gripper2base, t_gripper2base = compute_gripper2base(joint_angles_file,
angles_in_degrees=True)

    # 출력 설정
    np.set_printoptions(precision=6, suppress=True)

    # 결과 출력
    print(f"R_gripper2base (shape: {R_gripper2base.shape}): \n", R_gripper2base)
    print(f"\nt_gripper2base (shape: {t_gripper2base.shape}): \n", t_gripper2base)

    # 필요 시 파일로 저장
    np.save("R_gripper2base_list.npy", R_gripper2base)
    np.save("t_gripper2base_list.npy", t_gripper2base)
    print("\nResults saved as R_gripper2base_list.npy and t_gripper2base_list.npy")

    # 검증: 한 포즈에 대해 역변환이 올바른지 확인
    if len(R_gripper2base) > 0:

```

```

# 검증을 위해 원본 'degrees' 각도를 다시 로드하고, 라디안으로 변환하여 사용
q_test_degrees = np.load(joint_angles_file)[0]
q_test_radians = np.radians(q_test_degrees) # 여기에 추가된 부분

T_base2gripper, _, _ = forward_kinematics(q_test_radians)
T_gripper2base = inverse_transform(T_base2gripper)

# T_base2gripper * T_gripper2base ≈ I인지 확인
identity_check = T_base2gripper @ T_gripper2base
print(f"\n검증 (T_base2gripper @ T_gripper2base ≈ I):\n{identity_check}")

```

3.4.6.3 hand_eye_calibration.py

```

import numpy as np
import cv2
import os

def validate_data(R_gripper2base, t_gripper2base, R_target2cam, t_target2cam):
    """입력 데이터의 형상과 유효성을 검증."""
    assert len(R_gripper2base) == len(t_gripper2base) == len(R_target2cam) ==
len(t_target2cam), "데이터 쌍의 수가 일치하지 않습니다."
    assert R_gripper2base.shape[1:] == (3, 3), "R_gripper2base는 (N, 3, 3) 형식이어야 합니다."
    assert R_target2cam.shape[1:] == (3, 3), "R_target2cam은 (N, 3, 3) 형식이어야 합니다."
    assert t_gripper2base.shape[1:] in [(3,), (3, 1)], "t_gripper2base는 (N, 3) 또는 (N, 1) 형식이어야 합니다."
    assert t_target2cam.shape[1:] in [(3,), (3, 1)], "t_target2cam은 (N, 3) 또는 (N, 1) 형식이어야 합니다."
    for R in R_gripper2base:
        assert np.allclose(R @ R.T, np.eye(3), atol=1e-5), "R_gripper2base는 직교 회전 행렬이어야 합니다."
    for R in R_target2cam:
        assert np.allclose(R @ R.T, np.eye(3), atol=1e-5), "R_target2cam은 직교 회전 행렬이어야 합니다."

def run_hand_eye_calibration(input_dir, output_dir, method=cv2.CALIB_HAND_EYE_PARK):
    """
    Hand-Eye Calibration을 수행하여 카메라-그립퍼 변환을 계산.

    Args:
        input_dir (str): 입력 데이터 파일이 있는 디렉토리.
        output_dir (str): 출력 파일을 저장할 디렉토리.
        method: cv2 Hand-Eye Calibration 메서드 (기본값: CALIB_HAND_EYE_TSAI).

    Returns:
        tuple: (R_cam2gripper, t_cam2gripper) - 카메라에서 그립퍼로의 회전 행렬과 이동 벡터.
    """
    # 데이터 로드
    try:
        R_gripper2base = np.load(f"{input_dir}/R_gripper2base_list.npy",
allow_pickle=True)
        t_gripper2base = np.load(f"{input_dir}/t_gripper2base_list.npy",
allow_pickle=True)
        R_target2cam = np.load(f"{input_dir}/R_target2cam_list.npy", allow_pickle=True)
    
```

```

    t_target2cam = np.load(f"{input_dir}/t_target2cam_list.npy", allow_pickle=True)
except FileNotFoundError as e:
    raise FileNotFoundError(f"입력 파일을 찾을 수 없습니다: {str(e)}")
except Exception as e:
    raise RuntimeError(f"파일 로드 중 오류: {str(e)}")

# 데이터 검증
validate_data(R_gripper2base, t_gripper2base, R_target2cam, t_target2cam)

# 데이터 수
N = len(R_gripper2base)
print(f"총 데이터 수: {N}")

# Hand-Eye Calibration
try:
    R_cam2gripper, t_cam2gripper = cv2.calibrateHandEye(
        R_gripper2base, t_gripper2base, R_target2cam, t_target2cam, method=method
    )
except cv2.error as e:
    raise RuntimeError(f"Hand-Eye Calibration 실패: {str(e)}")

# 출력 설정
np.set_printoptions(precision=5, suppress=True)
print("\nⓐ Hand-Eye Calibration 결과 (Camera → Gripper 변환):")
print("\nⓐ 회전 행렬 (R_cam2gripper):\n", R_cam2gripper)
print("\nⓑ 이동 벡터 (t_cam2gripper):\n", t_cam2gripper)

# 결과 저장
os.makedirs(output_dir, exist_ok=True)
np.save(f"{output_dir}/R_cam2gripper.npy", R_cam2gripper)
np.save(f"{output_dir}/t_cam2gripper.npy", t_cam2gripper)
print(f"\n✓ 결과가 {output_dir}/R_cam2gripper.npy, {output_dir}/t_cam2gripper.npy로 저
장되었습니다.")

return R_cam2gripper, t_cam2gripper

if __name__ == "__main__":
    input_dir = "." # 현재 디렉토리
    output_dir = "results"
    run_hand_eye_calibration(input_dir, output_dir, method=cv2.CALIB_HAND_EYE_TSAI)

```

3.4.6.4 hand_eye_reprojection_error.py

```

import numpy as np

# _____
# 1) hand-eye 결과 로드 (여기에 calibrateHandEye() 결과를 직접 넣거나 npy 로드)
# _____
# 예시: calibrateHandEye()로 구한 값
# R_cam2gripper = np.array([[...],[...],[...]])
# t_cam2gripper = np.array([[...],[...],[...]])
#
# 또는 파일에서 불러오기:
R_cam2gripper = np.load("results/R_cam2gripper.npy")      # shape (3,3)

```

```

t_cam2gripper = np.load("results/t_cam2gripper.npy")           # shape (3,1)

# _____
# 2) 데이터 로드
# _____
R_gripper2base = np.load("R_gripper2base_list.npy", allow_pickle=True) # (N,3,3)
t_gripper2base = np.load("t_gripper2base_list.npy", allow_pickle=True) # (N,3,1)
R_target2cam   = np.load("R_target2cam_list.npy", allow_pickle=True) # (N,3,3)
t_target2cam   = np.load("t_target2cam_list.npy", allow_pickle=True) # (N,3,1)

assert len(R_gripper2base) == len(R_target2cam), "샘플 수 불일치"

# _____
# 3) A·X ≈ X·B 오차 계산
# _____
rot_errors    = []
trans_errors  = []

M = np.hstack((R_cam2gripper, t_cam2gripper))

N = len(R_gripper2base)
for i in range(N):
    A_R = R_gripper2base[i]
    A_t = t_gripper2base[i].reshape(3,1)
    B_R = R_target2cam[i]
    B_t = t_target2cam[i].reshape(3,1)

    # 회전 오차: A_R·X_R - X_R·B_R
    errR = A_R @ R_cam2gripper - R_cam2gripper @ B_R
    rot_err = np.linalg.norm(errR) # Frobenius norm
    rot_errors.append(rot_err)

    # 병진 오차: A_R·X_t + A_t - (X_R·B_t + X_t)
    left_t  = A_R @ t_cam2gripper + A_t
    right_t = R_cam2gripper @ B_t + t_cam2gripper
    trans_err = np.linalg.norm(left_t - right_t)
    trans_errors.append(trans_err)

# 전체 평균 오차
mean_rot_err  = np.mean(rot_errors)
mean_trans_err = np.mean(trans_errors)

# _____
# 4) 결과 출력
# _____
print(f"샘플 수: {N}\n")
for i,(re,te) in enumerate(zip(rot_errors, trans_errors), start=1):
    print(f"--- 샘플 {i:2d} --- 회전 오차: {re:.4e} | 병진 오차: {te:.4f} mm")

print("\n✓ 평균 회전 오차 (Frobenius norm):", f"{mean_rot_err:.4e}")
print("✓ 평균 병진 오차:", f"{mean_trans_err:.4f} mm")

```

```

addinedu@addinedu-ASUS-TUF-Dash-F15-FX516PM-FX516PM:~/handeye_test$ ./eye_reprojection_error.py
샘플 수 : 10

--- 샘플 1 --- 회전 오차 : 2.7272e+00 | 병진 오차 : 0.2570 mm
--- 샘플 2 --- 회전 오차 : 2.6899e+00 | 병진 오차 : 0.2322 mm
--- 샘플 3 --- 회전 오차 : 2.6456e+00 | 병진 오차 : 0.3385 mm
--- 샘플 4 --- 회전 오차 : 2.8240e+00 | 병진 오차 : 0.3620 mm
--- 샘플 5 --- 회전 오차 : 2.7283e+00 | 병진 오차 : 0.3383 mm
--- 샘플 6 --- 회전 오차 : 2.6943e+00 | 병진 오차 : 0.2189 mm
--- 샘플 7 --- 회전 오차 : 2.7388e+00 | 병진 오차 : 0.2236 mm
--- 샘플 8 --- 회전 오차 : 2.7328e+00 | 병진 오차 : 0.2362 mm
--- 샘플 9 --- 회전 오차 : 2.7355e+00 | 병진 오차 : 0.2264 mm
--- 샘플 10 --- 회전 오차 : 2.7229e+00 | 병진 오차 : 0.1916 mm

✓ 평균 회전 오차 (Frobenius norm): 2.7239e+00
✓ 평균 병진 오차 : 0.2625 mm

```

3.4.7 타겟의 위치를 base 좌표계로 변환

3.4.7.1 send_coord() 사용법

`send_coord(id, coord, speed)`

- function: send one coord to robot arm
- Parameters:
 - id : send one coord to robot arm, 1-6 corresponds to [x, y, z, rx, ry, rz]
 - coord : coord value(float)

Coord Id	range
x	-281.45 ~ 281.45
y	-281.45 ~ 281.45
z	-70 ~ 412.67
rx	-180 ~ 180
ry	-180 ~ 180
rz	-180 ~ 180
- speed : (int) 1-100
- Return value:
 - 1 : complete

`send_coords(coords, speed, mode)`

- function: Send overall coordinates and posture to move the head of the robotic arm from its original point to your specified point
- Parameters:
 - coords: : a list of coords value [x,y,z,rx,ry,rz],length6
 - speed (int) : 1 ~ 100
 - mode: (int) 0 - angular, 1 - linear
- Return value:
 - 1 : complete

get_angles_plan() 함수

이 함수는 로봇의 움직임을 제어하거나 시뮬레이션할 때, 로봇이 특정 시점에 혹은 특정 동작 경로를 따라 어떤 조인트 각도를 가져야 하는지에 대한 계획된 각도 값들의 전체 목록을 제공하는 역할을 합니다. 예를 들어, 로봇이 A 지점에서 B 지점으로 이동하는 경로가 계획되어 있다면, 이 함수는 그 경로를 따라가기 위한 각 조인트들의 각도 시퀀스 중 현재 시점에서 필요하거나 다음 스텝에서 필요한 각도들을 반환할 수 있습니다.

3.MDI Mode and Operation

`get_angles()`

- **function:** get the degree of all joints
- **Return value:** list a float list of all degree

`get_angles_plan()`

- **Function:** Get the planned angles of all joints
- **Return value:** list List of floating points of all degrees

`send_angle(id, degree, speed)`

- **function:** send one degree of joint to robot arm

- **Parameters:**

- `id` : joint id(genre.Angle), range int 1-6
- `degree` : degree value(float)

Joint Id	range
1	-168 ~ 168
2	-135 ~ 135
3	-150 ~ 150
4	-145 ~ 145
5	-165 ~ 165
6	-180 ~ 180

- `speed` : the speed and range of the robotic arm's movement 1~100

- **Return value:**

- `1` : complete

3.4.7.2 base 좌표계 검증 3가지 방법

3.4.7.2.1 1. 고정된 물체로 테스트

1. QR 코드를 테이블 위 고정된 위치에 놓기
2. 로봇 팔을 여러 다른 위치로 움직이기 (5~6번)
3. 각 위치에서 사진 찍고 베이스 좌표 계산
4. 결과를 비교

만약 이 결과가 너무 다르게 나오면 좌표 변환에 문제가 있음을 예상

3.4.7.2.2 2. 물리적 측정으로 검증

1. QR 코드를 특정 위치에 놓기
2. 자로 실제 거리 측정: 로봇 베이스에서 QR 코드까지
3. 시스템이 계산한 좌표와 실제 측정값 비교

3.4.7.2.3 3. 로봇 실제 이동으로 검증

1. QR 코드를 테이블에 놓기
2. 카메라로 사진 찍어서 베이스 좌표 계산
3. 로봇을 그 좌표로 실제 이동시키기
4. 로봇이 QR 코드 근처에 정확히 도착하는지 확인

로봇이 QR 코드에서 5-10mm 이내에 도착하면 성공,
수 cm 이상 벗어나면 좌표 변환에 오류 있음

[Flask QR 코드 중점 서버 \(see page 74\)](#)

[로봇 좌표값 이동 코드 \(see page 79\)](#)

3.4.7.3 target → base 좌표계 변환 코드

target → camera
camera → gripper
gripper → base

3.4.7.3.1 단위: mm / radian

```
import numpy as np

#-----
## 1. 이미지의 좌표계 -> 카메라 좌표계

def pixel_to_normalized_coordinates(pixel_coords, camera_matrix):
```

```

"""
픽셀 좌표를 정규화된 이미지 좌표로 변환

Args:
    pixel_coords: [u, v] 픽셀 좌표 (이미지에서 클릭한 좌표)
    camera_matrix: 3x3 카메라 내부 파라미터 행렬 K

Returns:
    normalized_coords: [x_norm, y_norm, 1] 정규화된 이미지 좌표
"""
# 픽셀 좌표를 동차 좌표로 변환 [u, v, 1]
u, v = pixel_coords
pixel_homogeneous = np.array([u, v, 1.0])

# K의 역행렬 계산 (내부 파라미터의 역행렬)
K_inv = np.linalg.inv(camera_matrix)

# 정규화된 좌표 계산: [x_norm, y_norm, 1] = K^(-1) * [u, v, 1]
normalized_coords = K_inv @ pixel_homogeneous

return normalized_coords

def normalized_to_3d_camera_coordinates(normalized_coords, depth_z):
"""
정규화된 좌표를 실제 3D 카메라 좌표로 변환

Args:
    normalized_coords: [x_norm, y_norm, 1] 정규화된 이미지 좌표
    depth_z: 실제 깊이값 (자로 측정한 Z값)

Returns:
    camera_3d_coords: [X, Y, Z] 실제 3D 카메라 좌표
"""
# 깊이값을 곱해서 실제 3D 좌표 계산
camera_3d_coords = depth_z * normalized_coords

return camera_3d_coords

#-----
## 2. 카메라 좌표계 --> 그리퍼 좌표계

def camera_to_gripper_transform(camera_coords, X_matrix):
"""
Eye-in-hand 시스템에서 카메라 좌표계를 그리퍼 좌표계로 변환

Args:
    camera_coords: [X, Y, Z] 카메라 좌표계의 3D 점
    X_matrix: 4x4 Hand-eye calibration 변환 행렬 (카메라 → 그리퍼)

Returns:
    gripper_coords: [X, Y, Z] 그리퍼 좌표계의 3D 점
"""
# 카메라 좌표를 동차 좌표로 변환 [X, Y, Z, 1]
camera_homogeneous = np.array([camera_coords[0], camera_coords[1],
                                camera_coords[2], 1.0])

# X는 이미 카메라→그리퍼 변환이므로 직접 사용

```

```

# 그리퍼 좌표 계산: P_gripper = X * P_camera
gripper_homogeneous = X_matrix @ camera_homogeneous

# 동차 좌표에서 3D 좌표 추출
gripper_coords = gripper_homogeneous[:3]

return gripper_coords

def print_transformation_info(X_matrix):
    """변환 행렬 정보 출력"""
    print("== Hand-Eye Calibration 변환 행렬 X ==")
    print("(카메라 → 그리퍼 변환)")
    print(X_matrix)
    print()

    # 회전 행렬과 평행이동 벡터 추출
    R = X_matrix[:3, :3]
    t = X_matrix[:3, 3]

    print("회전 행렬 R (3x3):")
    print(R)
    print()
    print("평행이동 벡터 t:")
    print(t)
    print()

#-----#
## 3. 그리퍼 좌표계 -> 베이스 좌표계

def gripper_to_base_transform(gripper_coords, joint_angles, robot_dh_params):
    """
    그리퍼 좌표계를 베이스 좌표계로 변환

    Args:
        gripper_coords: [X, Y, Z] 그리퍼 좌표계의 3D 점
        joint_angles: 현재 로봇의 joint angles [ $\theta_1$ ,  $\theta_2$ , ...,  $\theta_n$ ]
        robot_dh_params: 로봇의 DH 파라미터 (또는 URDF 정보)

    Returns:
        base_coords: [X, Y, Z] 베이스 좌표계의 3D 점
    """
    # Forward kinematics를 통해 현재 joint angles로부터
    # 베이스→그리퍼 변환 행렬 계산
    T_base_to_gripper = forward_kinematics(joint_angles, robot_dh_params)

    # 그리퍼 좌표를 동차 좌표로 변환
    gripper_homogeneous = np.array([gripper_coords[0], gripper_coords[1],
                                    gripper_coords[2], 1.0])

    # 베이스 좌표 계산: P_base = T_base_gripper * P_gripper
    base_homogeneous = T_base_to_gripper @ gripper_homogeneous

    # 동차 좌표에서 3D 좌표 추출
    base_coords = base_homogeneous[:3]

return base_coords

```

```

def forward_kinematics(joint_angles, dh_params):
    """
    DH 파라미터를 이용한 Forward Kinematics

    Args:
        joint_angles: [θ1, θ2, ..., θn] 각 관절의 각도
        dh_params: DH 파라미터 리스트
            각 요소는 [a, alpha, d, theta_offset] 형태

    Returns:
        T: 4x4 베이스→그리퍼 변환 행렬
    """
    T = np.eye(4) # 단위 행렬로 초기화

    for i, (a, alpha, d, theta_offset) in enumerate(dh_params):
        # 현재 관절 각도 (입력 + 오프셋)
        theta = joint_angles[i] + theta_offset

        # DH 변환 행렬 계산
        T_i = dh_transformation_matrix(a, alpha, d, theta)

        # 누적 변환
        T = T @ T_i

    return T

def dh_transformation_matrix(a, alpha, d, theta):
    """
    DH 파라미터로부터 변환 행렬 계산

    Args:
        a: link length
        alpha: link twist
        d: link offset
        theta: joint angle

    Returns:
        T: 4x4 변환 행렬
    """
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cos_alpha = np.cos(alpha)
    sin_alpha = np.sin(alpha)

    T = np.array([
        [cos_theta, -sin_theta*cos_alpha, sin_theta*sin_alpha, a*cos_theta],
        [sin_theta, cos_theta*cos_alpha, -cos_theta*sin_alpha, a*sin_theta],
        [0, sin_alpha, cos_alpha, d],
        [0, 0, 0, 1]
    ])

    return T

def get_robot_dh_params():
    """
    실제 로봇의 DH 파라미터 반환

```

```

>Returns:
>Returns:
dh_params: DH 파라미터 리스트 [[a, alpha, d, theta_offset], ...]
"""

# 실제 로봇 DH 파라미터
d_vals      = [131.22,    0,      0,   63.4 , 75.05, 45.6 ]      # d_i [mm]
a_vals      = [ 0     , -110.4, -96,    0 ,  0   , 0   ]      # a_i [mm]
alpha_vals = [ 1.5708,    0,      0,   1.5708, -1.5708, 0 ]      # α_i [rad]
offsets    = [ 0     , -1.5708, 0,  -1.5708, 1.5708, 0 ]      # θ_offset_i [rad]

# [a, alpha, d, theta_offset] 형태로 조합 (mm 단위 유지)
dh_params = []
for i in range(6):
    dh_params.append([
        a_vals[i],          # a (mm)
        alpha_vals[i],      # alpha (rad)
        d_vals[i],          # d (mm)
        offsets[i]          # theta_offset (rad)
    ])

# # [a, alpha, d, theta_offset] 형태로 조합 (mm → m 변환)
# dh_params = []
# for i in range(6):
#     dh_params.append([
#         a_vals[i] / 1000.0,      # a (m)
#         alpha_vals[i],          # alpha (rad)
#         d_vals[i] / 1000.0,      # d (m)
#         offsets[i]              # theta_offset (rad)
#     ])

print("== 로봇 DH 파라미터 ==")
print("Joint | a_i | α_i | d_i | θ_offset_i")
print("-----|-----|-----|-----|-----")
for i, (a, alpha, d, offset) in enumerate(dh_params):
    print(f" {i+1} | {a:7.2f} | {alpha:7.4f} | {d:7.2f} | {offset:8.4f}")
print()

return dh_params

#-----
## main()

if __name__ == "__main__":
    # 1단계: 픽셀 좌표 → 정규화된 이미지 좌표
    print(f"✓ 1단계 - 픽셀 좌표 → 카메라 좌표")

    """
    [[986.93180332  0.          251.57937661]
     [ 0.           983.85767546 131.93997962]
     [ 0.           0.           1.          ]]
    """

    camera_matrix = np.array([
        [986.93180332,  0.0,      251.57937661],
        [ 0.0,          983.85767546, 131.93997962],
        [ 0.0,          0.0,       1.0]
    ])

```

```

])
# 이미지에서 클릭한 픽셀 좌표
pixel_point = [280, 260] # [u, v]

# 자로 측정한 깊이값 (단위: mm 또는 원하는 단위)
measured_depth = 170.0 # 170mm

# 1. 픽셀 좌표 → 정규화된 이미지 좌표
normalized_point = pixel_to_normalized_coordinates(pixel_point, camera_matrix)
print(f"픽셀 좌표: {pixel_point}")
print(f"정규화된 좌표: {normalized_point}")

# normalized_point[2] = measured_depth

# 2. 정규화된 좌표 → 실제 3D 카메라 좌표
camera_3d_point = normalized_to_3d_camera_coordinates(normalized_point,
measured_depth)
print(f"3D 카메라 좌표: {camera_3d_point}")

# # 결과 해석
print("\n== 결과 해석 ==")
print(f"X: {camera_3d_point[0]:.2f} (카메라 기준 좌우 방향)")
print(f"Y: {camera_3d_point[1]:.2f} (카메라 기준 상하 방향)")
print(f"Z: {camera_3d_point[2]:.2f} (카메라 기준 전후 방향, 측정값과 동일)\n\n")

#-----
## 2단계: 카메라 → 그리퍼 좌표
print(f"✓ 2단계 - 카메라 좌표 → 그리퍼 좌표")

# 주어진 X 행렬 (그리퍼 → 카메라 변환) / mm 단위
X_matrix = np.array([
    [-1.55740602e-01, -2.05727860e-01, -9.66137109e-01, 1.10446282e+02],
    [9.87208681e-01, -6.61964508e-02, -1.45041546e-01, 1.48750887e+01],
    [3.41157607e-02, 9.76367798e-01, -2.13405805e-01, 3.42848443e+02],
    [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00],
])
# gripper_3d = camera_to_gripper_transform(normalized_point, X_matrix)
gripper_3d = camera_to_gripper_transform(camera_3d_point, X_matrix)

print(f"최종 결과: 그리퍼 기준 타겟 위치")
print(f"X: {gripper_3d[0]:.2f} mm")
print(f"Y: {gripper_3d[1]:.2f} mm")
print(f"Z: {gripper_3d[2]:.2f} mm \n\n")

# print_transformation_info(X_matrix)

#-----
## 3단계: 그리퍼 → 베이스 좌표
print(f"✓ 3단계 - 그리퍼 좌표 → 베이스 좌표")

gripper_point = gripper_3d

# 조인트 각도 입력 (원본 값)

```

```

joint_angles_input = np.array([0.587, 1.873, -0.627, -1.0, -0.403, 1.187])

# 설정에 따른 단위 처리
JOINT_ANGLE_UNIT = "radian" # "radian" 또는 "degree" 중 선택

if JOINT_ANGLE_UNIT == "radian":
    joint_angles_rad = joint_angles_input
    joint_angles_deg = np.degrees(joint_angles_input)
    print(f"입력 단위: 라디안")
elif JOINT_ANGLE_UNIT == "degree":
    joint_angles_deg = joint_angles_input
    joint_angles_rad = np.radians(joint_angles_input)
    print(f"입력 단위: 도")
else:
    raise ValueError("JOINT_ANGLE_UNIT은 'radian' 또는 'degree'만 가능합니다.")

# 로봇 DH 파라미터
robot_dh = get_robot_dh_params()

print(f"그리퍼 좌표: {gripper_point} mm")
print(f"현재 Joint Angles: {joint_angles_deg} deg")
print(f"현재 Joint Angles: {joint_angles_rad} rad \n")

# 그리퍼 → 베이스 좌표 변환 (항상 라디안으로 전달)
base_point = gripper_to_base_transform(gripper_point, joint_angles_rad, robot_dh)
print(f"베이스 좌표: {base_point} mm")

```

3.4.7.4 Flask QR 코드 중점 서버

```

0
import cv2
import threading
import time
import json
from flask import Flask, Response, jsonify, render_template_string
app = Flask(__name__)
# 글로벌 변수
frame = None
frame_lock = threading.Lock()
latest_qr_data = {
    'detected': False,
    'center': None,
    'data': None,
    'timestamp': None
}
qr_data_lock = threading.Lock()
def capture_and_process_frames():
    """카메라에서 프레임을 캡처하고 QR 코드를 실시간으로 감지"""
    global frame, latest_qr_data
    # QR 코드 감지기 초기화
    qr_detector = cv2.QRCodeDetector()
    # 카메라 디바이스 열기
    camera = cv2.VideoCapture('/dev/jetcocam0')
    if not camera.isOpened():

```

```

print(":x: 카메라를 열 수 없습니다!")
return
print(":white_check_mark: 카메라 초기화 완료")
while True:
    success, img = camera.read()
    if not success:
        print(":x: 프레임을 읽을 수 없습니다.")
        time.sleep(0.1)
        continue
    # QR 코드 감지 수행
    data, points, _ = qr_detector.detectAndDecode(img)
    # QR 데이터 업데이트
    with qr_data_lock:
        if points is not None and len(points) > 0 and data:
            points = points[0].astype(int)
            # 중심점 계산
            center_x = int(points[:, 0].mean())
            center_y = int(points[:, 1].mean())
            center = (center_x, center_y)
            # QR 데이터 저장
            latest_qr_data = {
                'detected': True,
                'center': center,
                'data': data,
                'timestamp': time.time()
            }
            # 시각화
            # 중심점 표시
            cv2.circle(img, center, 8, (0, 0, 255), -1)
            cv2.putText(img, f"Center: {center}", (center_x + 15, center_y - 10),
                       cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
            # QR 데이터 표시
            cv2.putText(img, f"Data: {data}", (10, 30),
                       cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
            # QR 코드 윤곽선 표시
            for i in range(len(points)):
                pt1 = tuple(points[i])
                pt2 = tuple(points[(i + 1) % len(points)])
                cv2.line(img, pt1, pt2, (0, 255, 0), 3)
            # 콘솔에도 출력
            print(f":dart: [QR 감지] 중심: {center}, 데이터: '{data}'")
        else:
            # QR 코드 없음
            latest_qr_data = {
                'detected': False,
                'center': None,
                'data': None,
                'timestamp': time.time()
            }
            # "QR 코드 없음" 표시
            cv2.putText(img, "No QR Code Detected", (10, 30),
                       cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    # 프레임 인코딩 및 저장
    _, buffer = cv2.imencode('.jpg', img)
    with frame_lock:
        frame = buffer.tobytes()
    # 프레임 캡처 간격 (~30fps)

```

```

        time.sleep(0.03)
        camera.release()
    def generate_frames():
        """스트리밍용 프레임 생성"""
        global frame
        while True:
            with frame_lock:
                if frame is None:
                    time.sleep(0.1)
                    continue
                current_frame = frame
            yield (
                b'--frame\r\n'
                b'Content-Type: image/jpeg\r\n\r\n' + current_frame + b'\r\n'
            )
@app.route('/stream')
def video_feed():
    """비디오 스트림 엔드포인트"""
    return Response(
        generate_frames(),
        mimetype='multipart/x-mixed-replace; boundary=frame'
    )
@app.route('/qr_data')
def get_qr_data():
    """현재 QR 데이터 반환"""
    with qr_data_lock:
        return jsonify(latest_qr_data)
@app.route('/qr_center')
def get_qr_center():
    """QR 중심 좌표만 반환"""
    with qr_data_lock:
        if latest_qr_data['detected']:
            return jsonify({
                'success': True,
                'center': latest_qr_data['center'],
                'data': latest_qr_data['data']
            })
        else:
            return jsonify({
                'success': False,
                'message': 'QR 코드가 감지되지 않았습니다'
            })
@app.route('/')
def index():
    """메인 페이지"""
    html_template = """
    <!DOCTYPE html>
    <html>
    <head>
        <title>QR 코드 감지 시스템</title>
        <style>
            body { font-family: Arial, sans-serif; margin: 20px; }
            .container { max-width: 1200px; margin: 0 auto; }
            .video-container { text-align: center; margin: 20px 0; }
            .info-panel {
                background: #F5F5F5;
                padding: 20px;
    """

```

```

        border-radius: 8px;
        margin: 20px 0;
    }
    .qr-info {
        font-size: 18px;
        margin: 10px 0;
        padding: 10px;
        background: white;
        border-radius: 4px;
    }
    .detected { color: green; font-weight: bold; }
    .not-detected { color: red; }
    button {
        padding: 10px 20px;
        font-size: 16px;
        margin: 5px;
        cursor: pointer;
        background: #007BFF;
        color: white;
        border: none;
        border-radius: 4px;
    }
    button:hover { background: #0056B3; }
</style>
</head>
<body>
    <div class="container">
        <h1>:dart: QR 코드 감지 시스템</h1>
        <div class="video-container">
            <h2>:camera: 실시간 카메라</h2>
            
        </div>
        <div class="info-panel">
            <h2>:bar_chart: QR 감지 정보</h2>
            <div id="qr-status" class="qr-info not-detected">
                QR 코드 감지 대기 중...
            </div>
            <div id="qr-center" class="qr-info">
                중심 좌표: -
            </div>
            <div id="qr-data" class="qr-info">
                QR 데이터: -
            </div>
            <div id="timestamp" class="qr-info">
                마지막 업데이트: -
            </div>
            <button onclick="updateQRInfo()">:arrows_counterclockwise: 정보 새로고침</
button>
            <button onclick="getQRCenter()">:dart: 중심좌표 가져오기</button>
        </div>
        <div class="info-panel">
            <h2>:robot_face: 로봇 제어</h2>
            <button onclick="moveRobotToQR()">:arrow_right: QR 위치로 이동</button>
            <div id="robot-status" class="qr-info">
                로봇 상태: 대기 중
            </div>
        </div>
    </div>
</body>

```

```

        </div>
    </div>
    <script>
        // 자동으로 QR 정보 업데이트 (2초마다)
        setInterval(updateQRInfo, 2000);
        function updateQRInfo() {
            fetch('/qr_data')
                .then(response => response.json())
                .then(data => {
                    const statusDiv = document.getElementById('qr-status');
                    const centerDiv = document.getElementById('qr-center');
                    const dataDiv = document.getElementById('qr-data');
                    const timestampDiv = document.getElementById('timestamp');
                    if (data.detected) {
                        statusDiv.innerHTML = ':white_check_mark: QR 코드 감지됨';
                        statusDiv.className = 'qr-info detected';
                        centerDiv.innerHTML = `중심 좌표: (${data.center[0]}, ${data.center[1]})`;
                        dataDiv.innerHTML = `QR 데이터: ${data.data}`;
                    } else {
                        statusDiv.innerHTML = ':x: QR 코드 없음';
                        statusDiv.className = 'qr-info not-detected';
                        centerDiv.innerHTML = '중심 좌표: -';
                        dataDiv.innerHTML = 'QR 데이터: -';
                    }
                    const time = new Date(data.timestamp * 1000).toLocaleTimeString();
                    timestampDiv.innerHTML = `마지막 업데이트: ${time}`;
                })
                .catch(error => {
                    console.error('Error:', error);
                    document.getElementById('qr-status').innerHTML = ':x: 연결 오류';
                });
            }
            function getQRCenter() {
                fetch('/qr_center')
                    .then(response => response.json())
                    .then(data => {
                        if (data.success) {
                            alert(`QR 중심 좌표: (${data.center[0]}, ${data.center[1]})\nQR 데이터: ${data.data}`);
                        } else {
                            alert('QR 코드가 감지되지 않았습니다');
                        }
                    });
            }
            function moveRobotToQR() {
                fetch('/qr_center')
                    .then(response => response.json())
                    .then(data => {
                        if (data.success) {
                            document.getElementById('robot-status').innerHTML =
                                `:robot_face: QR 위치로 이동 중... 좌표: (${data.center[0]}, ${data.center[1]})`;
                            // 여기에 실제 로봇 이동 코드 추가
                            // 예: fetch('/move_robot', {method: 'POST', body: JSON.stringify(data.center)})
                        }
                    });
            }
        }
    </script>

```

```

        } else {
            alert('QR 코드를 먼저 감지해주세요');
        }
    });
}

// 페이지 로드 시 즉시 정보 업데이트
updateQRInfo();
</script>
</body>
</html>
"""

return render_template_string(html_template)
if __name__ == '__main__':
    print(":rocket: QR 감지 Flask 시작")
    # 카메라 캡처 및 QR 감지 스레드 시작
    capture_thread = threading.Thread(target=capture_and_process_frames)
    capture_thread.daemon = True
    capture_thread.start()
    print(":camera: 카메라 스레드 시작됨")
    print(":globe_with_meridians: 서버 주소: http://0.0.0.0:5000")
    print(":iphone: 로컬에서 접속: http://xn--ip-oj9ig8m:5000")
    # Flask 서버 시작
    app.run(host='0.0.0.0', port=5000, threaded=True, debug=False)

```

3.4.7.5 로봇 좌표값 이동 코드

```

# 로봇 좌표값 이동 코드
send_coorid =[100, 50, 100, 0, 0, 0]
re = mc.send_coords(send_coorid,10)
print(re)

```

3.4.8 joint 값 받는 코드

```

from pymycobot.mycobot import MyCobot
import time

# JetCobot 연결
mc = MyCobot('/dev/ttyJETCOBOT', 1000000)
mc.power_on()
mc.thread_lock = True
print("✓ JetCobot이 연결되었습니다.")

# 초기 위치 및 그리퍼 열기
initial_angles = [0, 0, 0, 0, 0, 40]

def move_home():
    mc.send_angles(initial_angles, speed=50)
    time.sleep(2)

# 초기 위치 이동

```

```

print("🏠 초기 위치로 이동 중...")
move_home()

# _____
# ● 수동 조작 위치 설정
# _____
print("\n🛠 좌표 설정 준비: 모터 OFF - 수동으로 위치 조작하세요 .")
mc.release_all_servos()

input("👉 원하는 위치로 조작 후 [엔터]를 눌러주세요 . . .")

# 현재 조인트 각도 및 좌표 읽기
angles = mc.get_angles()

print("\n✅ 수동 설정된 값:")
print(" - 조인트 각도 (degrees):", angles)

# 필요시 저장도 가능
import numpy as np
np.save("manual_angles.npy", angles)
print("\n💾 조인트 및 좌표 정보가 파일로 저장되었습니다 .")

```

```
angles = mc.get_angles()
```

3.4.8.1 get_angles()

- **function:** get the degree of all joints
- **Return value:** list a float list of all degree

Radian으로 변경할 필요가 있음

3.4.9 Hand Eye Calibration (Python code)

단위를 m와 radian을 사용하여 X행렬을 연산함.

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Hand-Eye Calibration 스크립트
- 카메라→그리퍼 변환 행렬을 구합니다.

## 주의할 점

```

- .png 파일의 이름은 Radian 값들임을 잊지 말 것.
- mm, m 단위를 혼용에서 사용하지 말 것.

만약 연산을 m로 하고 결과는 mm로 얻고 싶다면,
절대 연산 중간의 단위를 건들지 말고, 마지막의 결과를 변경해야 함.
(아마 지금까지 이런 실수를 계속 반복하지 않았을까 싶음)

** ☺ 이 코드는 m와 radian을 사용하여 연산이 진행됨 ☺ **

:::::

```
import os
import re
import numpy as np
import cv2

# =====#
# 초기 설정 TODO: 단위 신경쓰기
# =====#

# 1) 이미지가 저장된 폴더 경로
IMAGE_DIR = "C:/Users/USER/Desktop/study_python/pics"

# 2) 체스보드 패턴 (내부 코너 개수) 및 스퀘어 크기(m 단위)
PATTERN_SIZE = (6, 9)    # 가로 7, 세로 6 예시
SQUARE_SIZE = 0.020      # 체스보드 한 칸 크기 (m)

# 2.5) Joint 연산 단위 설정
JOINT_UNIT = "radian" # "radian" 또는 "degree" 중 선택

# 3) 카메라 내부 파라미터 & 왜곡 계수
cameraMatrix = np.array([
    [986.93180332, 0., 251.57937661],
    [0., 983.85767546, 131.93997962],
    [0., 0., 1.],
], dtype=np.float64)
distCoeffs = np.array([-2.77060722e-01, -1.73446994e+00, 4.61497572e-03,
4.60231034e-03, 6.43334573e+00], dtype=np.float64)

# 4) DH 파라미터: [(theta_offset(deg), d(m), a(m), alpha(deg)), ...]
#     MyCobot280 DH 파라미터
d_vals      = [131.22, 0, 0, 63.4, 75.05, 45.6]    # d_i [mm]
a_vals      = [0, -110.4, -96, 0, 0, 0]    # a_i [mm]
alpha_vals  = [1.5708, 0, 0, 1.5708, -1.5708, 0]    # α_i [rad]
offsets     = [0, -1.5708, 0, -1.5708, 1.5708, 0]    # θ_offset_i [rad]

# DH_PARAMS 리스트 자동 생성
DH_PARAMS = []
for d_mm, a_mm, alpha_rad, offset_rad in zip(d_vals, a_vals, alpha_vals, offsets):
    # mm → m
    d = d_mm / 1000.0
    a = a_mm / 1000.0

    DH_PARAMS.append((offset_rad, d, a, alpha_rad))
```

```

# =====
# 함수 정의
# =====

def fk_from_dh(joints_rad): ## Note: 연산 과정 이해 필요 && offset_deg를 theta_off_deg에 넣는 이유는? theta와 offset의 의미는?
    """
    DH 파라미터 기반 순방향 기구학(FK)
    joints_rad: [θ1, θ2, ..., θ6] in radians
    returns: R (3x3), t (3x1)
    """
    T = np.eye(4)
    for (theta_off_rad, d, a, alpha_off_rad), theta_rad in zip(DH_PARAMS, joints_rad):
        # 최종 각도 계산 (모두 라디안)
        θ = theta_rad + theta_off_rad
        α = alpha_off_rad

        # 삼각함수 계산 (라디안 입력)
        ct, st = np.cos(θ), np.sin(θ)
        ca, sa = np.cos(α), np.sin(α)

        # DH 변환행렬
        A = np.array([[ct, -st*ca, st*sa, a*ct],
                      [st, ct*ca, -ct*sa, a*st],
                      [0, sa, ca, d],
                      [0, 0, 0, 1]])
        T = T @ A

    R = T[:3, :3]
    t = T[:3, 3].reshape(3,1)
    return R, t

def parse_joints_from_name(filename):
    """
    파일명 '[0.014, -0.006, -0.009, -0.02, 0.005, 0.686].png' 형태에서
    6개의 조인트 값을 추출하여 라디안 단위면 그대로 이용
    아니라면 도 단위를 라디안 단위로 변경
    """
    # 1) 확장자 제거 + 양쪽 공백 제거
    name = os.path.splitext(os.path.basename(filename))[0].strip()
    # 2) 대괄호 안의 내용(소수, 음수, 공백, 쉼표)을 모두 잡는 패턴
    pattern = r"\[((-\d+\.\s,]+)\]"
    m = re.search(pattern, name)
    if not m:
        raise ValueError(f"파일명 형식 오류: {filename}")
    # 3) 쉼표로 분할, strip()으로 공백 제거, float 변환
    parts = [s.strip() for s in m.group(1).split(',')]
    joints_values = [float(x) for x in parts]

    # 4) 설정에 따른 단위 변환 (FK는 라디안을 기대)
    if JOINT_UNIT == "radian":
        # 라디안 그대로 사용
        print(f"라디안 사용: {joints_values}")
        return joints_values
    elif JOINT_UNIT == "degree":
        # 도 → 라디안 변환

```

```

joints_rad = [np.deg2rad(x) for x in joints_values]
print(f"도-라디안 변환: {joints_values} → {joints_rad}")
return joints_rad
else:
    raise ValueError(f"잘못된 JOINT_UNIT 설정: {JOINT_UNIT}. 'radian' 또는 'degree'를 사용하세요.")

```

```

def collect_data():
    """
    이미지 폴더를 순회하며 solvePnP와 FK를 통해
    Hand-Eye 보정 입력 리스트를 생성
    """
    # 3D object points (체스보드 코너 3D 좌표)
    objp = np.zeros((PATTERN_SIZE[1]*PATTERN_SIZE[0], 3), np.float32)
    objp[:, :2] = np.indices(PATTERN_SIZE).T.reshape(-1, 2) * SQUARE_SIZE

    R_g2b_list, t_g2b_list = [], []
    R_t2c_list, t_t2c_list = [], []

    for fname in sorted(os.listdir(IMAGE_DIR)):
        if not fname.lower().endswith((".png", ".jpg")):
            continue
        img_path = os.path.join(IMAGE_DIR, fname)
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # 체스보드 코너 검출
        ret, corners = cv2.findChessboardCorners(gray, PATTERN_SIZE, None)
        if not ret:
            print(f"코너 미검출: {fname}")
            continue
        cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1),
                         (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
                          0.001))

        # solvePnP -> target->camera
        ok, rvec, tvec = cv2.solvePnP(objp, corners, cameraMatrix, distCoeffs)
        if not ok:
            print(f"PnP 실패: {fname}")
            continue
        R_t2c, _ = cv2.Rodrigues(rvec)
        t_t2c = tvec.reshape(3, 1)

        # 파일명에서 조인트 값 추출 -> gripper->base via FK
        joints = parse_joints_from_name(fname)
        R_b2g, t_b2g = fk_from_dh(joints) # base->gripper 변환

        # FK 결과의 역행렬 계산하여 gripper-base 변환 구하기
        # 회전행렬의 역행렬은 전치행렬 ( $R^{-1} = R^T$ )
        R_g2b = R_b2g.T

        # 평행이동의 역변환:  $t_g2b = -R^T \times t_b2g$ 
        t_g2b = -R_g2b @ t_b2g

        R_g2b_list.append(R_g2b)
        t_g2b_list.append(t_g2b)
        R_t2c_list.append(R_t2c)

```

```

t_t2c_list.append(t_t2c)

print(f"수집 완료: {fname} -> joints={joints}")

return R_g2b_list, t_g2b_list, R_t2c_list, t_t2c_list

# =====
# 메인 루틴
# =====

if __name__ == "__main__":
    # 데이터 수집
    R_gripper2base, t_gripper2base, R_target2cam, t_target2cam = collect_data()

    # Hand-Eye 보정 수행
    R_cam2gripper, t_cam2gripper = cv2.calibrateHandEye(
        R_gripper2base, t_gripper2base,
        R_target2cam, t_target2cam,
        method=cv2.CALIB_HAND_EYE_PARK
    )

    # 결과 출력
    print("\n==== Hand-Eye 변환 결과 ===")
    print("R_cam2gripper:\n", R_cam2gripper)
    print("t_cam2gripper:\n", t_cam2gripper.flatten())

    # 동차 행렬 합성
    T = np.eye(4)
    T[:3,:3] = R_cam2gripper
    T[:3,3] = t_cam2gripper.flatten()
    ...
    ## ⓂR_cam2gripper (회전행렬)의 의미
    [[-155.74060161 -205.72786013 -966.13710858]
     [ 987.20868102 -66.19645083 -145.04154584]
     [ 34.11576075  976.36779825 -213.40580452]]

    # 첫 번째 열 [-0.156, 0.987, 0.034]
    # → 카메라의 X축이 그리퍼 좌표계에서 어느 방향을 향하는가?

    # 두 번째 열 [-0.206, -0.066, 0.976]
    # → 카메라의 Y축이 그리퍼 좌표계에서 어느 방향을 향하는가?

    # 세 번째 열 [-0.966, -0.145, -0.213]
    # → 카메라의 Z축(촬영 방향)이 그리퍼 좌표계에서 어느 방향을 향하는가?

    ## Ⓜt_cam2gripper (평행이동벡터)의 의미
    [110.44628245 14.87508869 342.84844308]

    # 카메라가 그리퍼 중심에서 얼마나 떨어져 있는가?
    # 카메라 위치는 그리퍼에서 110mm 앞, 15mm 오른쪽, 343mm 위 --> 값이 이상함
    .....

## Ⓜ예상되는 문제 (우선 단위는 제대로 맞춰다고 판단이 됨)
1. 체스보드 검출 과정에서 발생한 오차 (오차가 누적되어 큰 값으로 돌아옴)
```

2. 카메라 캘리브레이션이 잘못됨
 3. DH 파라미터의 문제 -> 이건 아닐거 같기는 함
 ...
`print("T_cam2gripper (4x4):\n", T)`

```
# mm 로 변환                                     ## Note:
R_cam2gripper_mm = R_cam2gripper             ## 회전 행렬은 단위를 수정할 필요가 없음
t_cam2gripper_mm = t_cam2gripper * 1000 ## 평행 이동 벡터만 단위를 m -> mm로 수정

T_mm = np.eye(4)
T_mm[:3,:3] = R_cam2gripper_mm
T_mm[:3,3] = t_cam2gripper_mm.flatten()

print("\n== Hand-Eye 변환 결과(mm) ==")
print("R_cam2gripper_mm:\n", R_cam2gripper_mm)
print("t_cam2gripper_mm:\n", t_cam2gripper_mm.flatten())
print("T_cam2gripper_mm (4x4):\n", T_mm)
```

3.5 관제 카메라 실행 코드 (bmp파일로 저장)

```
import cv2
import time

# 웹캠 연결
cap = cv2.VideoCapture(0)

# 웹캠이 정상적으로 열렸는지 확인
if not cap.isOpened():
    print("웹캠을 열 수 없습니다.")
    exit()

# 해상도 설정 (너비, 높이)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920) # 너비 설정
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080) # 높이 설정

# 이미지 번호
img_count = 0

while True:
    # 프레임 캡처
    ret, frame = cap.read()

    if not ret:
        print("프레임을 읽을 수 없습니다.")
        break

    # 이미지 출력
    height, width = frame.shape[:2] # 원본 이미지의 높이, 너비
    resized_frame = cv2.resize(frame, (width // 2, height // 2)) # 절반 크기로 리사이즈
    cv2.imshow('Webcam', resized_frame)
```

```

# 키보드 입력 대기
fps = cap.get(cv2.CAP_PROP_FPS)
wait_time = round(1000 / fps) # 밀리초로 변환
key = cv2.waitKey(wait_time) & 0xFF

# 's' 키를 누르면 이미지 저장
if key == ord('s'):
    # 타임스탬프나 번호를 이용해서 고유한 파일명 생성
    img_filename = f'/home/addinedu/workspace/Image/captured_image_{int(time.time())}.bmp'
    cv2.imwrite(img_filename, frame)
    print(f"이미지가 저장되었습니다: {img_filename}")

# 'q' 키를 누르면 종료
elif key == ord('q'):
    break

# 웹캠 해제
cap.release()
cv2.destroyAllWindows()

```

```

#실행 F5 // 파일저장 s // 종료 q
//쓰실분은 파일저장 위치만 수정해주면 될듯

```

3.5.1 관제 캠 (검은색) 캘리

```

import numpy as np
import cv2
import os
import argparse

def calibrate(dirpath, square_size, width, height, visualize=False):
    """ Apply camera calibration operation for images in the given directory path. """

    # termination criteria (최대 반복 횟수 30번, 오차가 0.001 이하)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # 3D 객체 점 생성: (0,0,0), (1,0,0), ..., (8,6,0)
    objp = np.zeros((height * width, 3), np.float32)
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)

    # 각 객체 점을 실제 크기(square_size로 크기 확장)
    objp = objp * square_size

    # 객체 점과 이미지 점을 저장할 리스트
    objpoints = [] # 3D 포인트 (실제 세계 좌표)
    imgpoints = [] # 2D 포인트 (이미지 좌표)

    images = os.listdir(dirpath)

    for fname in images:

```

```



```

```

if args["visualize"].lower() == "true":
    visualize = True
else:
    visualize = False

# 카메라 보정 함수 호출
ret, mtx, dist, rvecs, tvecs = calibrate(dirpath, square_size, visualize=visualize,
width=width, height=height)

# 결과 출력
print("카메라 행렬 (내부 파라미터):")
print(mtx)
print("왜곡 계수:")
print(dist)

# 저장 위치 변경: /home/addinedu/workspace/Image
save_path = "/home/addinedu/workspace/Image/"
np.save(os.path.join(save_path, "calibration_matrix"), mtx)
np.save(os.path.join(save_path, "distortion_coefficients"), dist)

print(f"카메라 캘리브레이션 완료! 저장 위치: {save_path}")

```

```

cd /home/addinedu/Downloads/cali_test/
실행코드 python3 LiveCali.py --dir /home/addinedu/workspace/Image --square_size 0.02

```

```

카메라 행렬 (내부 파라미터):
[[933.38893585  0.          972.927402   ]
 [ 0.           931.7890549  539.67101638]
 [ 0.           0.           1.          ]]
왜곡 계수:
[[-0.10640719  0.07541392 -0.00403532 -0.00425712 -0.04999974]]
평균 재투영 오차: 0.22118259619709824

```

3.5.2 관제 캠 (흰색) 캘리

```

import numpy as np
import cv2
import os
import argparse

def calibrate(dirpath, square_size, width, height, visualize=False):
    """ Apply camera calibration operation for images in the given directory path. """

    # termination criteria (최대 반복 횟수 30번, 오차가 0.001 이하)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # 3D 객체 점 생성: (0,0,0), (1,0,0), ..., (8,6,0)
    objp = np.zeros((height * width, 3), np.float32)
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)

```

```

# 각 객체 점을 실제 크기(square_size로 크기 확장)
objp = objp * square_size

# 객체 점과 이미지 점을 저장할 리스트
objpoints = [] # 3D 포인트 (실제 세계 좌표)
imgpoints = [] # 2D 포인트 (이미지 좌표)

images = os.listdir(dirpath)

for fname in images:
    img = cv2.imread(os.path.join(dirpath, fname))

    if img is None:
        print(f"[X] 이미지 불러오기 실패: {fname}")
        continue

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 체스보드 코너 찾기
    ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

    # 체스보드를 찾으면, 객체 점과 이미지 점을 추가
    if ret:
        print(f"[0] 체스보드 감지 성공: {fname}")
        objpoints.append(objp)

        # 코너를 정밀하게 보정
        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        imgpoints.append(corners2)

        # 이미지에 코너 그리기
        img = cv2.drawChessboardCorners(img, (width, height), corners2, ret)
    else:
        print(f"[X] 체스보드 감지 실패: {fname}")

    # 이미지 시각화 (선택 사항)
    if visualize:
        cv2.imshow('img', img)
        cv2.waitKey(0)

    # 카메라 보정 (내부 파라미터와 외곡 계수 계산)
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)

    return [ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints]

def calculate_reprojection_error(objpoints, imgpoints, rvecs, tvecs, mtx, dist):
    """ Calculate the reprojection error """
    total_error = 0
    for i in range(len(objpoints)):
        # 3D 객체 점을 2D 이미지로 투영
        imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)

        # 2D 이미지 점과 계산된 2D 이미지 점 사이의 차이 계산
        error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2) / len(imgpoints2)
        total_error += error

```

```

mean_error = total_error / len(objpoints)
return mean_error

if __name__ == '__main__':
    ap = argparse.ArgumentParser()
    ap.add_argument("-d", "--dir", required=True, help="Path to folder containing
checkerboard images for calibration")
    ap.add_argument("-w", "--width", type=int, help="Width of checkerboard
(default=9)", default=9)
    ap.add_argument("-t", "--height", type=int, help="Height of checkerboard
(default=6)", default=6)
    ap.add_argument("-s", "--square_size", type=float, default=0.02, help="Length of
one edge (in meters)")
    ap.add_argument("-v", "--visualize", type=str, default="False", help="To visualize
each checkerboard image")
    args = vars(ap.parse_args())

    dirpath = args['dir']
    square_size = args['square_size'] # 20mm = 0.02m

    width = args['width']
    height = args['height']

    if args["visualize"].lower() == "true":
        visualize = True
    else:
        visualize = False

    # 카메라 보정 함수 호출
    ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints = calibrate(dirpath,
square_size, visualize=visualize, width=width, height=height)

    # 결과 출력
    print("카메라 행렬 (내부 파라미터):")
    print(mtx)
    print("왜곡 계수:")
    print(dist)

    # Reprojection error 계산
    mean_error = calculate_reprojection_error(objpoints, imgpoints, rvecs, tvecs, mtx,
dist)
    print(f"평균 재투영 오차: {mean_error}")

    # 저장 위치 변경: /home/addinedu/workspace/Image2/
    save_path = "/home/addinedu/workspace/Image2/"
    np.save(os.path.join(save_path, "calibration_matrix"), mtx)
    np.save(os.path.join(save_path, "distortion_coefficients"), dist)

    print(f"카메라 캘리브레이션 완료! 저장 위치: {save_path}")

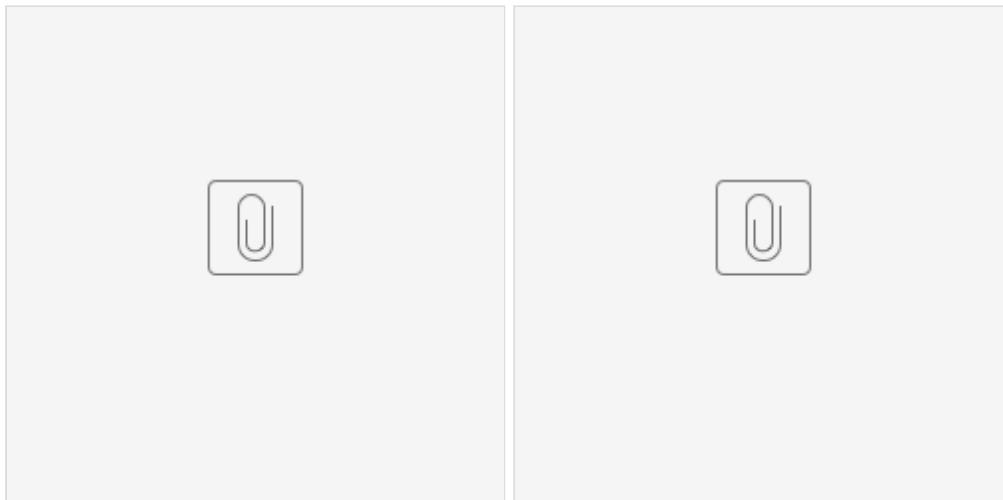
```

```
cd Downloads/cali_test (경로바꿔주고해야됨 중요)
#실행코드 python3 LiveCali.py --dir /home/addinedu/workspace/Image2 --square_size 0.02
```

카메라 행렬 (내부 파라미터) :

```
[[1.04884888e+03 0.00000000e+00 9.17719366e+02]
 [0.00000000e+00 1.03205061e+03 5.82551332e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
와곡 계수:
 [[ 0.01725646 -0.13491462  0.00208791 -0.00981669  0.10287163]]
 평균 재투영 오차: 0.1857427848950463
```

3.5.3 Numpy



3.6 Motion Planning

E Motion Planning은 로봇팔이 목표 위치까지 안전하고 효율적으로 이동할 수 있도록 경로를 계산해주는 과정
단순히 좌표만 보내는 방식은 장애물 충돌, 관절 제한, 부자연스러운 궤적 등 실제 작동에서 많은 문제가 발생할 수 있지만, Motion Planning을 사용하면 이러한 제약 조건을 모두 고려한 실제 가능한 궤적(trajectory)를 자동으로 생성해주기 때문에, 신뢰성 있는 로봇 동작을 구현가능
MoveIt2가 이 기능을 담당

<https://blog.naver.com/lion0634/223373004924>

<https://m.blog.naver.com/lion0634/223373632507>

<https://blog.naver.com/lion0634/223351637124>

3.6.1 전체 개념 요약

3.6.1.1 1. URDF (또는 Xacro)

- 로봇의 골격 구조를 표현한 파일
- 링크(link), 조인트(joint), 메쉬(mesh) 정보 등 포함
- 예: mycobot_280.urdf.xacro

3.6.1.2 2. MoveIt Setup Assistant

- URDF를 기반으로 해서 MoveIt이 사용할 수 있도록 필요한 설정 패키지 (`mycobot_moveit_config`)를 자동으로 생성
- 내부적으로는:
 - 플래닝 그룹(manipulator 등)
 - 자기 충돌 행렬(Self-Collision Matrix)
 - 컨트롤러 설정
 - RViz 시각화 설정
 - MoveGroup 실행 구성 등을 포함

3.6.1.3 3. MoveIt 실행 (Motion Planning Plugin in RViz)

- URDF 구조 + `moveit_config` 를 바탕으로:
 - 경로 계획(trajectory planning)
 - inverse kinematics
 - 충돌 회피
 - 목표 지점 도달 등 시뮬레이션 가능
- RViz 내에서 실제로 플래너가 만들어낸 “최적 경로”를 시각화

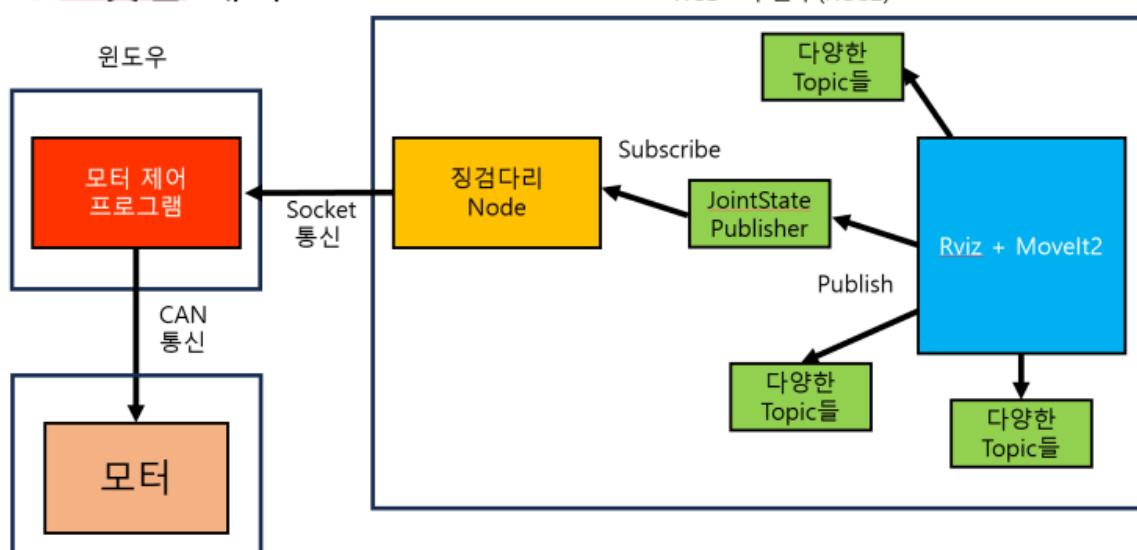
3.6.1.4 4. 실물 로봇 제어

- 시뮬레이션한 경로를 실제 로봇에 명령으로 전송
- 일반적으로는 ROS2 컨트롤러(`ros2_control`)가 연결되어 있어야 시뮬레이션 결과를 실제 모터 제어 명령으로 바꿔 전달 가능

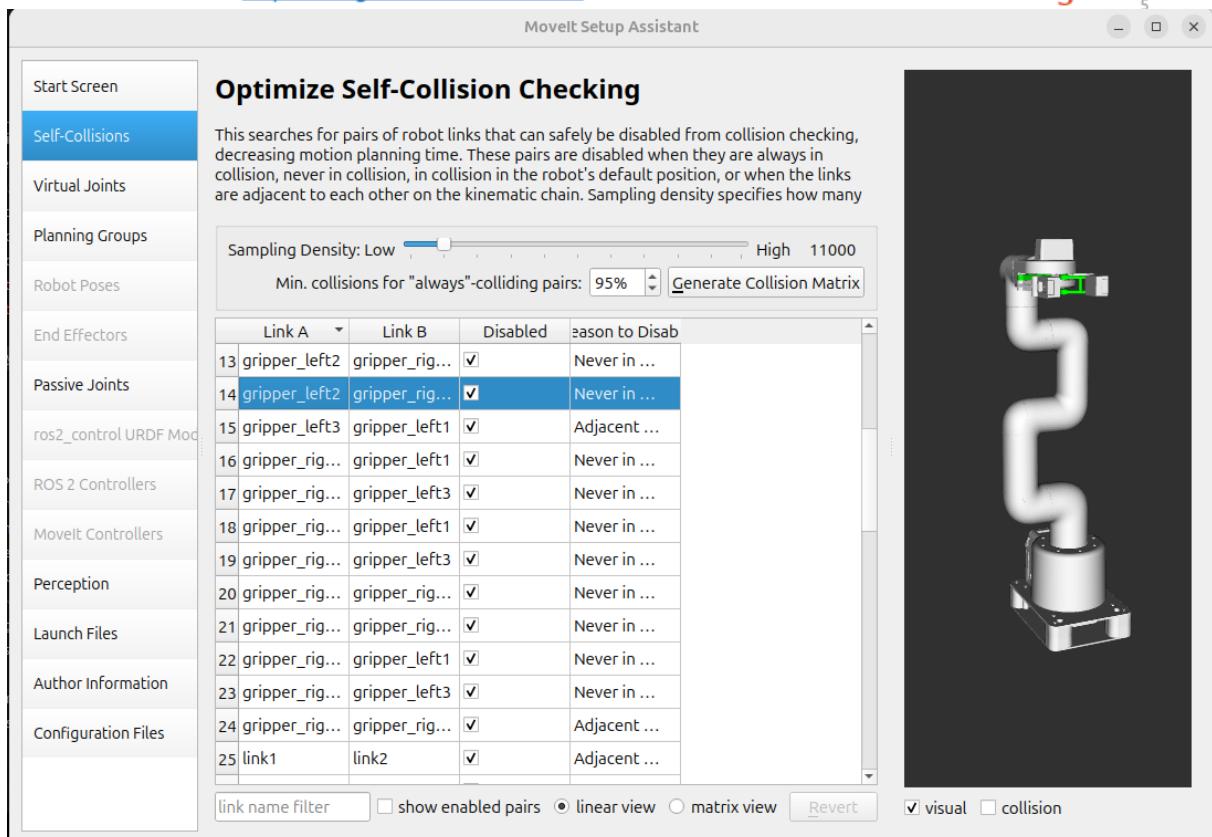
3.6.2 핵심 포인트

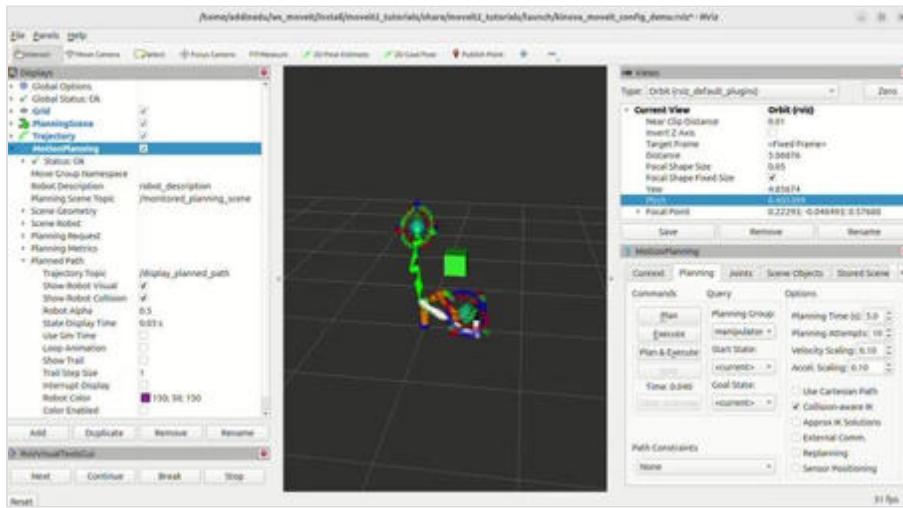
- URDF는 로봇의 모양을 설명
- MoveIt Setup Assistant는 로봇의 움직임 설정을 자동화
- MoveIt은 “목표로 가는 최적 경로”를 계산
- ROS2 Control이 연결되면 실제 로봇이 시뮬레이션대로 움직임

3.로봇팔 제어



실제 세계

<https://blog.naver.com/lion0634> 자세한 과정은 블로그 & 이전 발표자료



3.6.3 MyCobot Motion Planning

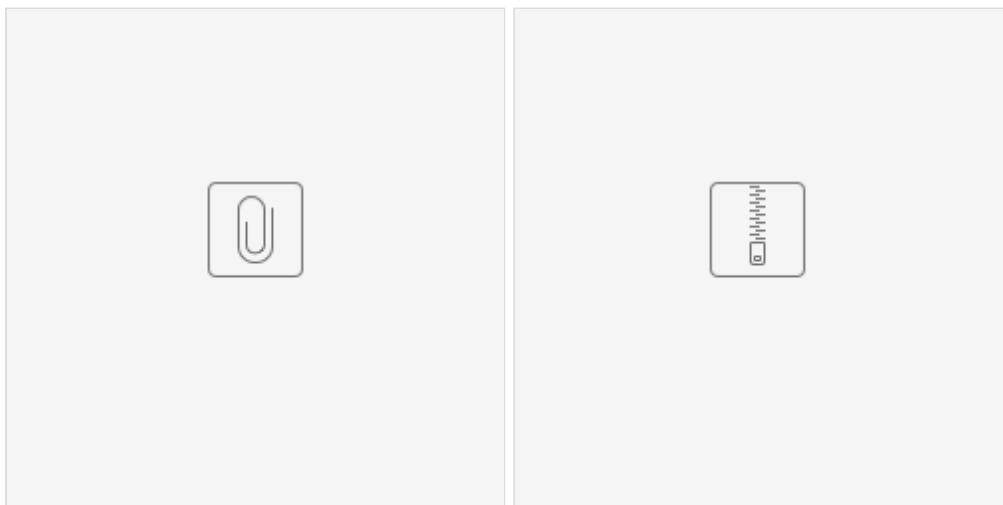
3.7 내부 파라미터 & 왜곡계수 & Reprojection error

▣ 체크보드 이미지를 이용해 우리 카메라의 **내부 파라미터**, **왜곡 계수**, 그리고 **Reprojection Error**를 구하기. 이 작업은 향후 비전 기반 제어를 위해 꼭 필요한 과정.

❖ 구한 값들 요약

- **내부 파라미터 (Camera Matrix)**
 - 카메라의 초점 거리와 이미지 중심 위치를 나타냄
 - 3D 좌표를 2D 이미지에 정확히 투영하는 데 필요함
- **왜곡 계수 (Distortion Coefficients)**
 - 실제 렌즈로 인해 발생하는 왜곡을 보정
 - 이 값이 있어야 실제 좌표와 이미지 좌표 간 오차를 줄일 수 있음
- **Reprojection Error (재투영 오차)**
 - 보정이 잘 되었는지를 나타내는 지표

<https://qkrtmddn0507.atlassian.net/wiki/spaces/196616/pages/edit-v2/7700770/> ---->>> (두번째 관제 캠)



[cam_matrix.py](#)²

[distortion_coefficients.npy](#)³

<https://darkpgmr.tistory.com/32> (무조오오오건 참고)

<https://github.com/GSNCodes/ArUCo-Markers-Pose-Estimation-Generation-Python>

3.7.1 카메라 Calibration으로 내부 파라미터와 왜곡 계수 구하기

```
'''  
Sample Usage:-  
python calibration.py --dir calibration_checkerboard/ --square_size 0.024  
'''  
  
import numpy as np  
import cv2  
import os  
import argparse  
  
def calibrate(dirpath, square_size, width, height, visualize=False):  
    """ Apply camera calibration operation for images in the given directory path. """  
  
    # termination criteria  
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)  
  
    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(8,6,0)  
    objp = np.zeros((height*width, 3), np.float32)  
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)  
  
    objp = objp * square_size
```

2. [https://qkrtdnn0507.atlassian.net/wiki/download/attachments/6619207/cam_matrix.py?
api=v2&cacheVersion=1&modificationDate=1752039655986&version=1](https://qkrtdnn0507.atlassian.net/wiki/download/attachments/6619207/cam_matrix.py?api=v2&cacheVersion=1&modificationDate=1752039655986&version=1)

3. [https://qkrtdnn0507.atlassian.net/wiki/download/attachments/6619207/distortion_coefficients.npy?
api=v2&cacheVersion=1&modificationDate=1752039655957&version=1](https://qkrtdnn0507.atlassian.net/wiki/download/attachments/6619207/distortion_coefficients.npy?api=v2&cacheVersion=1&modificationDate=1752039655957&version=1)

```

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = os.listdir(dirpath)

for fname in images:
    img = cv2.imread(os.path.join(dirpath, fname))

    if img is None:
        print(f"[X] 이미지 불러오기 실패: {fname}")
        continue

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

    # If found, add object points, image points (after refining them)
    if ret:
        print(f"[O] 체스보드 감지 성공: {fname}")
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (width, height), corners2, ret)
    else:
        print(f"[X] 체스보드 감지 실패: {fname}")

    if visualize:
        cv2.imshow('img',img)
        cv2.waitKey(0)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)

return [ret, mtx, dist, rvecs, tvecs]

if __name__ == '__main__':
    ap = argparse.ArgumentParser()
    ap.add_argument("-d", "--dir", required=True, help="Path to folder containing
checkerboard images for calibration")
    ap.add_argument("-w", "--width", type=int, help="Width of checkerboard
(default=9)", default=9)
    ap.add_argument("-t", "--height", type=int, help="Height of checkerboard
(default=6)", default=6)
    ap.add_argument("-s", "--square_size", type=float, default=1, help="Length of one
edge (in metres)")
    ap.add_argument("-v", "--visualize", type=str, default="False", help="To visualize
each checkerboard image")
    args = vars(ap.parse_args())

```

```
dirpath = args['dir']
# 2.4 cm == 0.024 m
# square_size = 0.024
square_size = args['square_size']

width = args['width']
height = args['height']

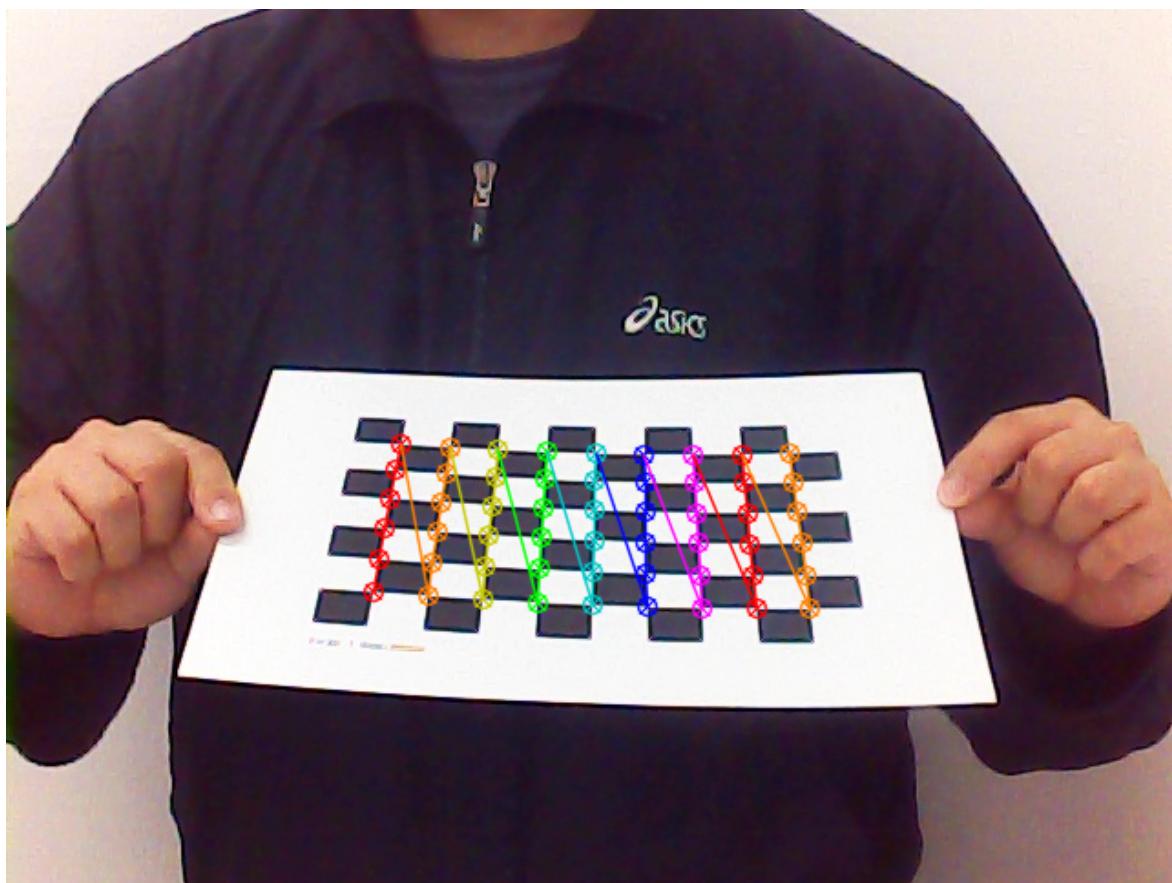
if args["visualize"].lower() == "true":
    visualize = True
else:
    visualize = False

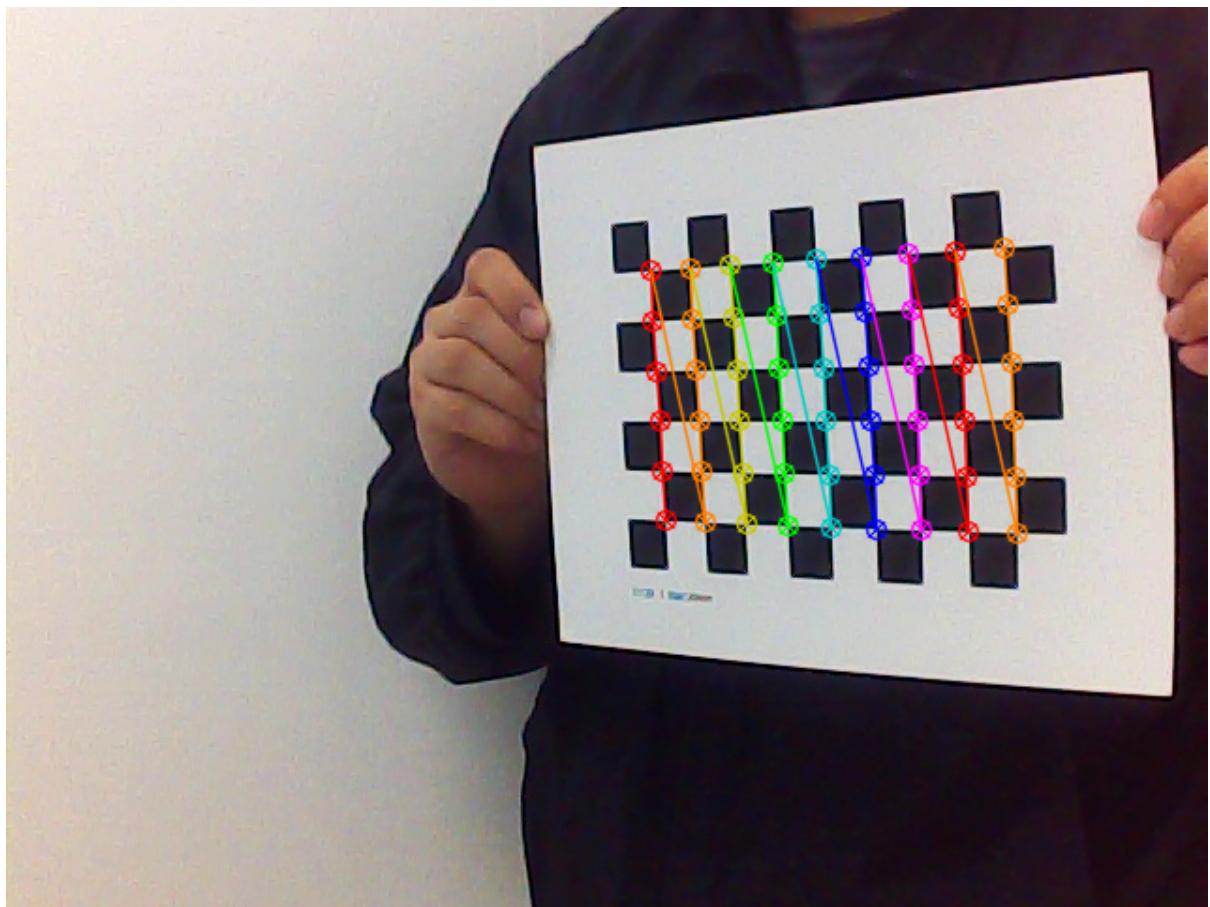
ret, mtx, dist, rvecs, tvecs = calibrate(dirpath, square_size, visualize=visualize,
width=width, height=height)

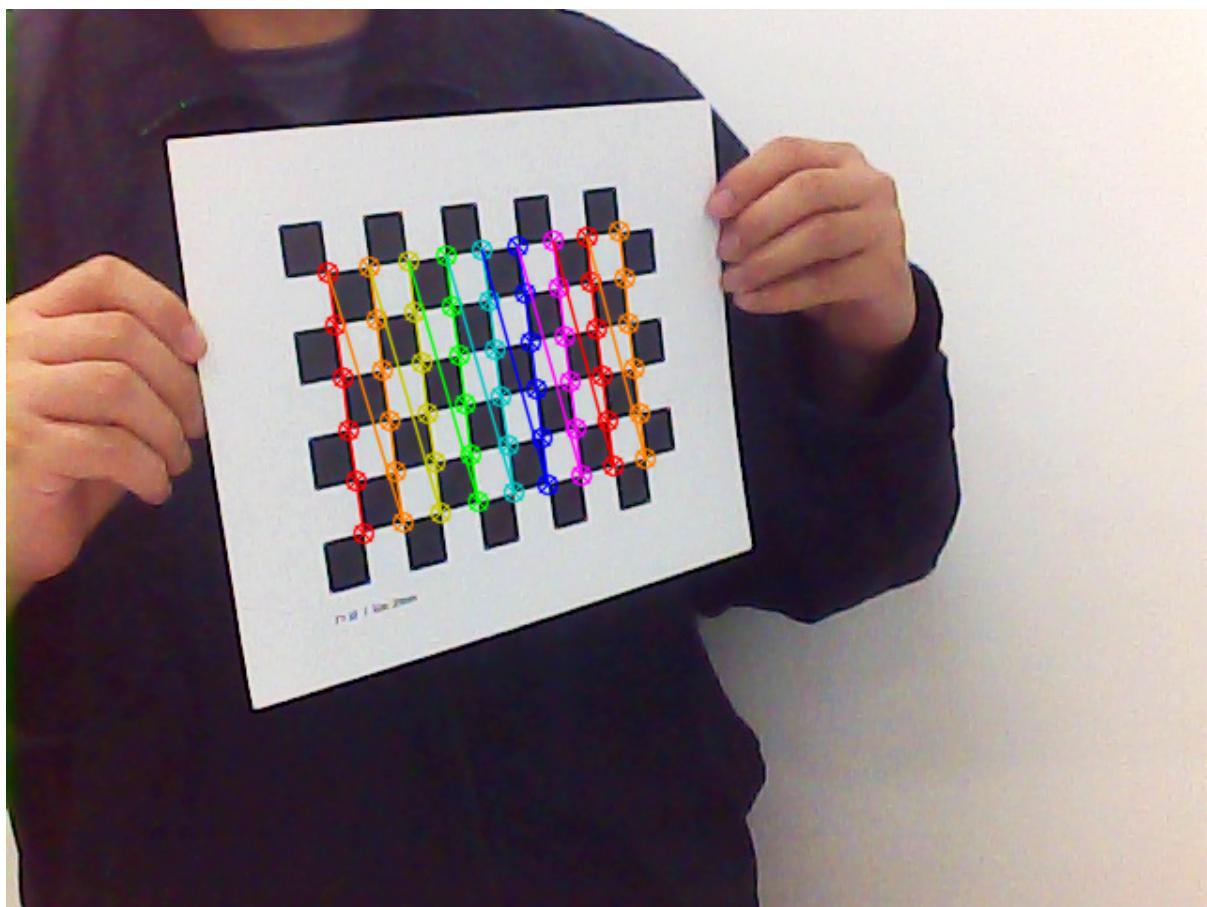
print(mtx)
print(dist)

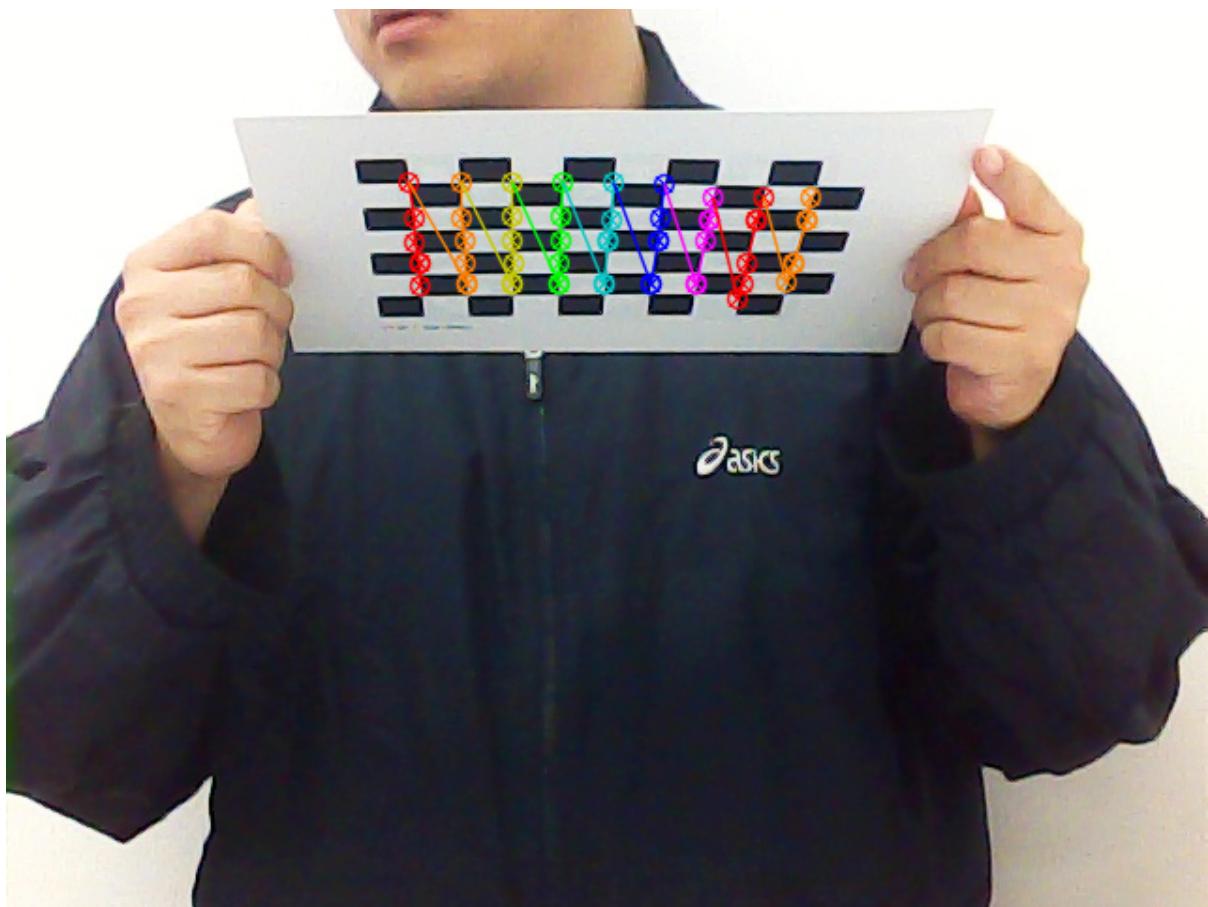
np.save("calibration_matrix", mtx)
np.save("distortion_coefficients", dist)
```

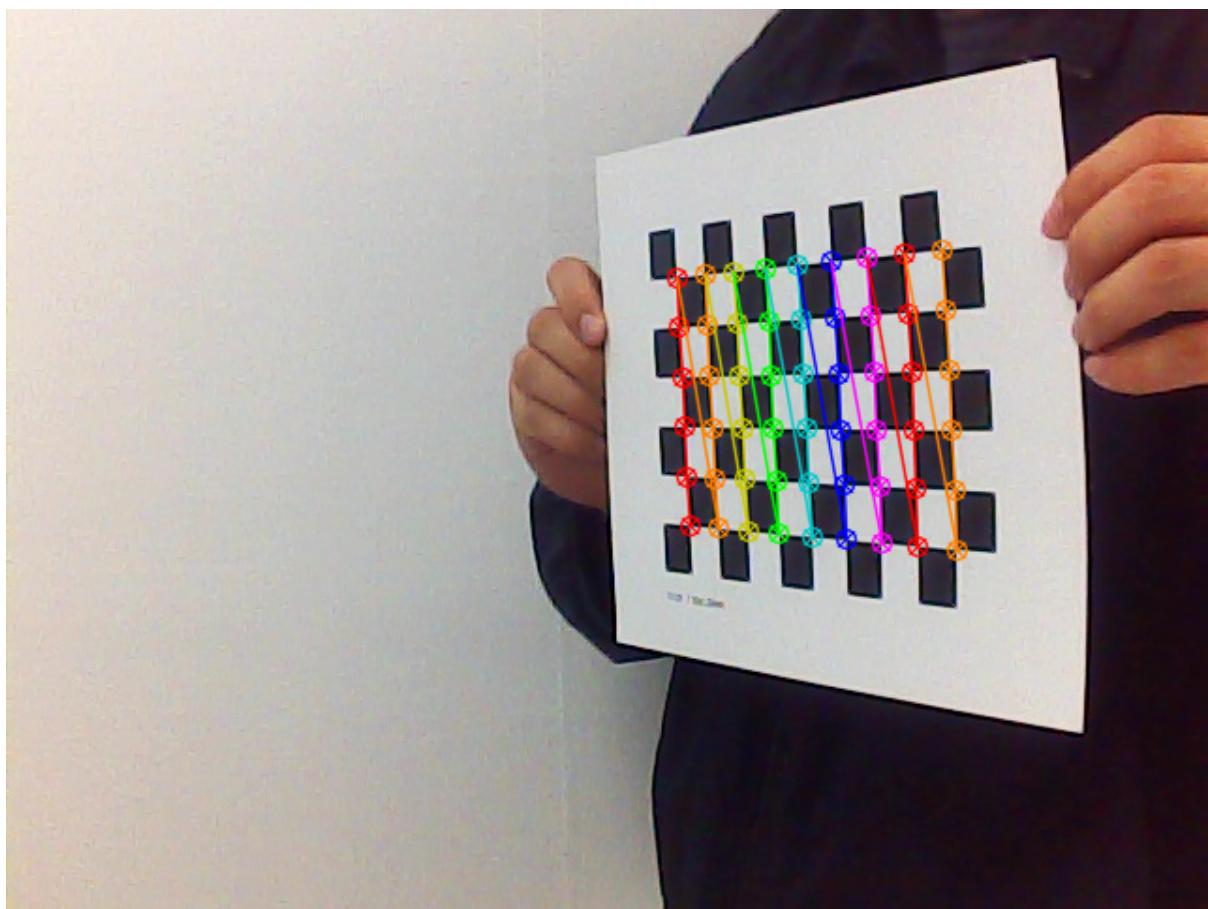
```
$ python3 calibration.py --dir /home/addinedu/cali_test/pics --width 6 --height 9 --
square_size 0.02 --visualize True
# 실행코드
```

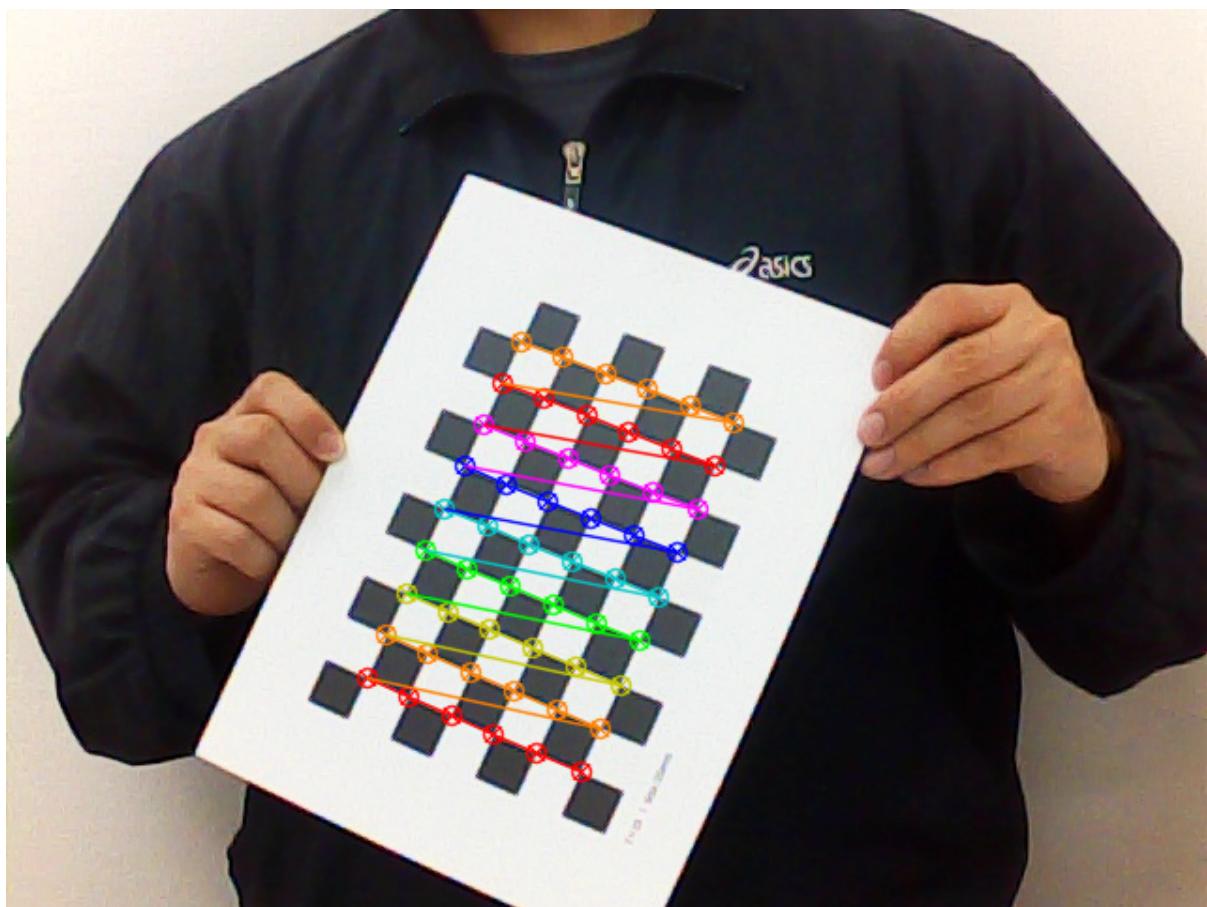


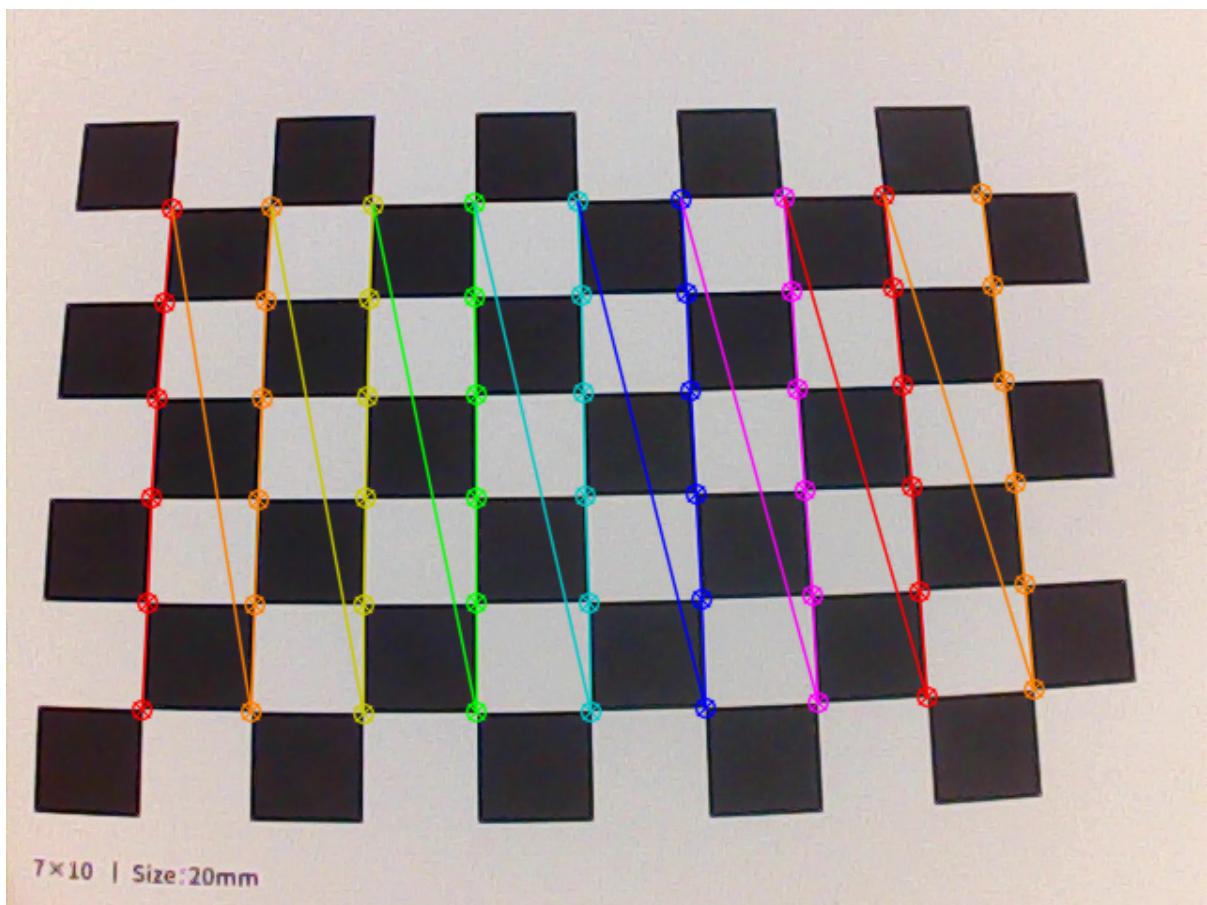


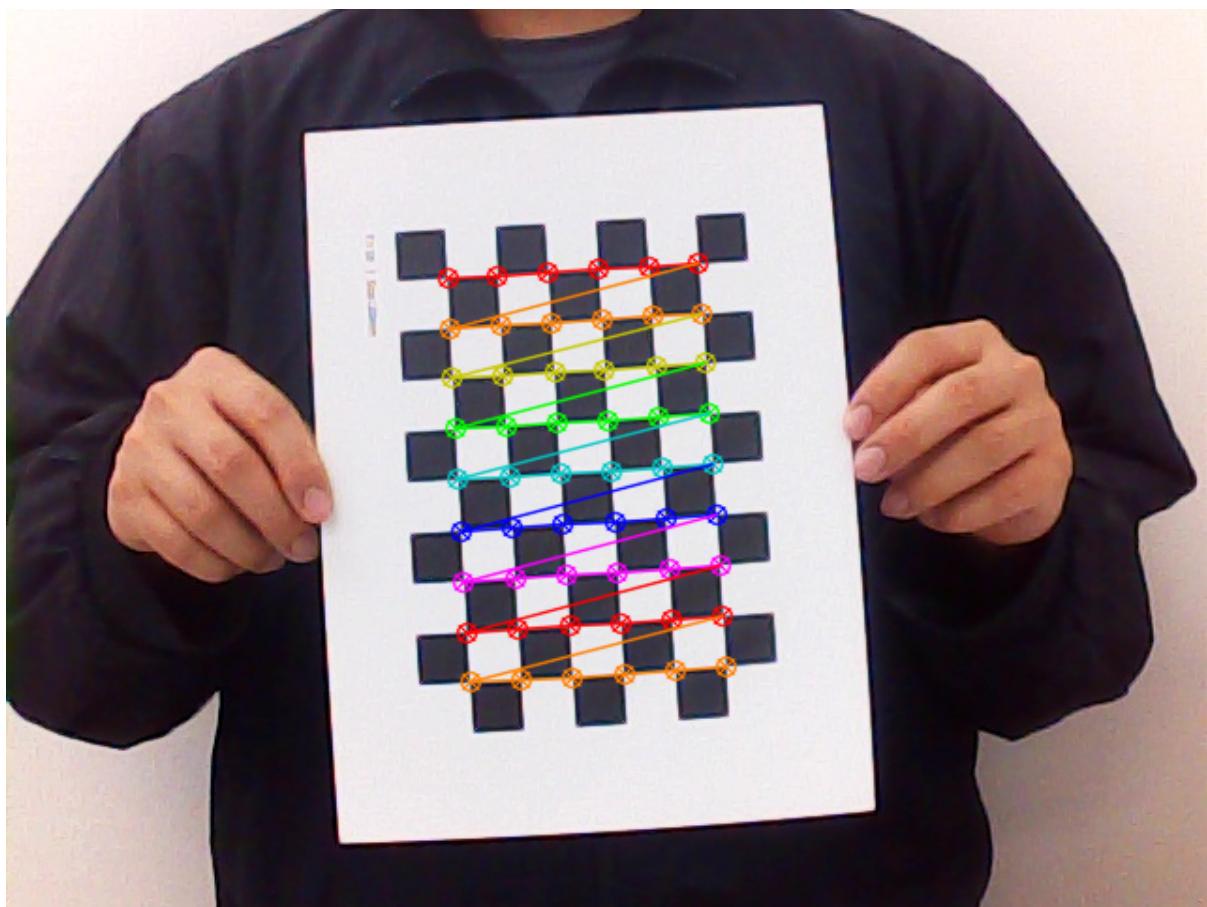


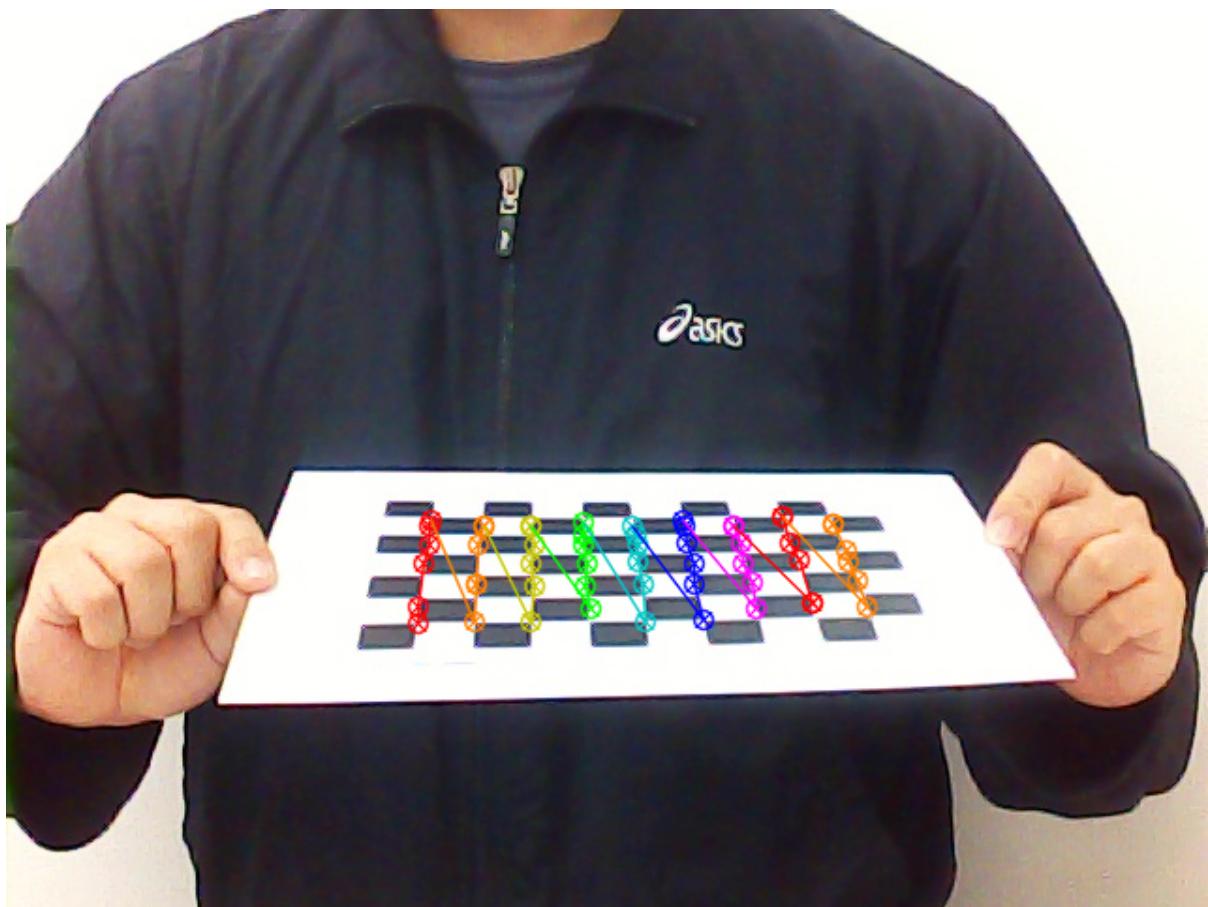


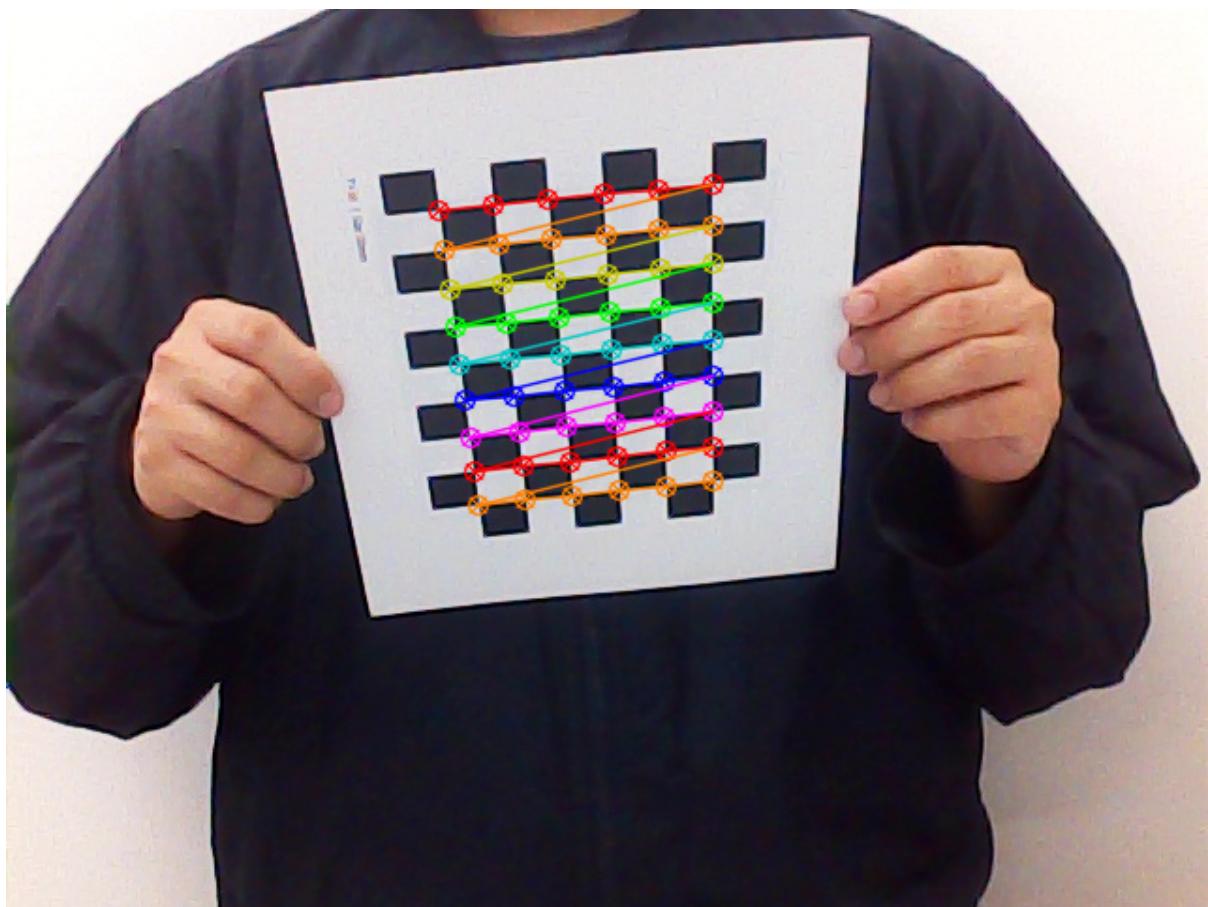


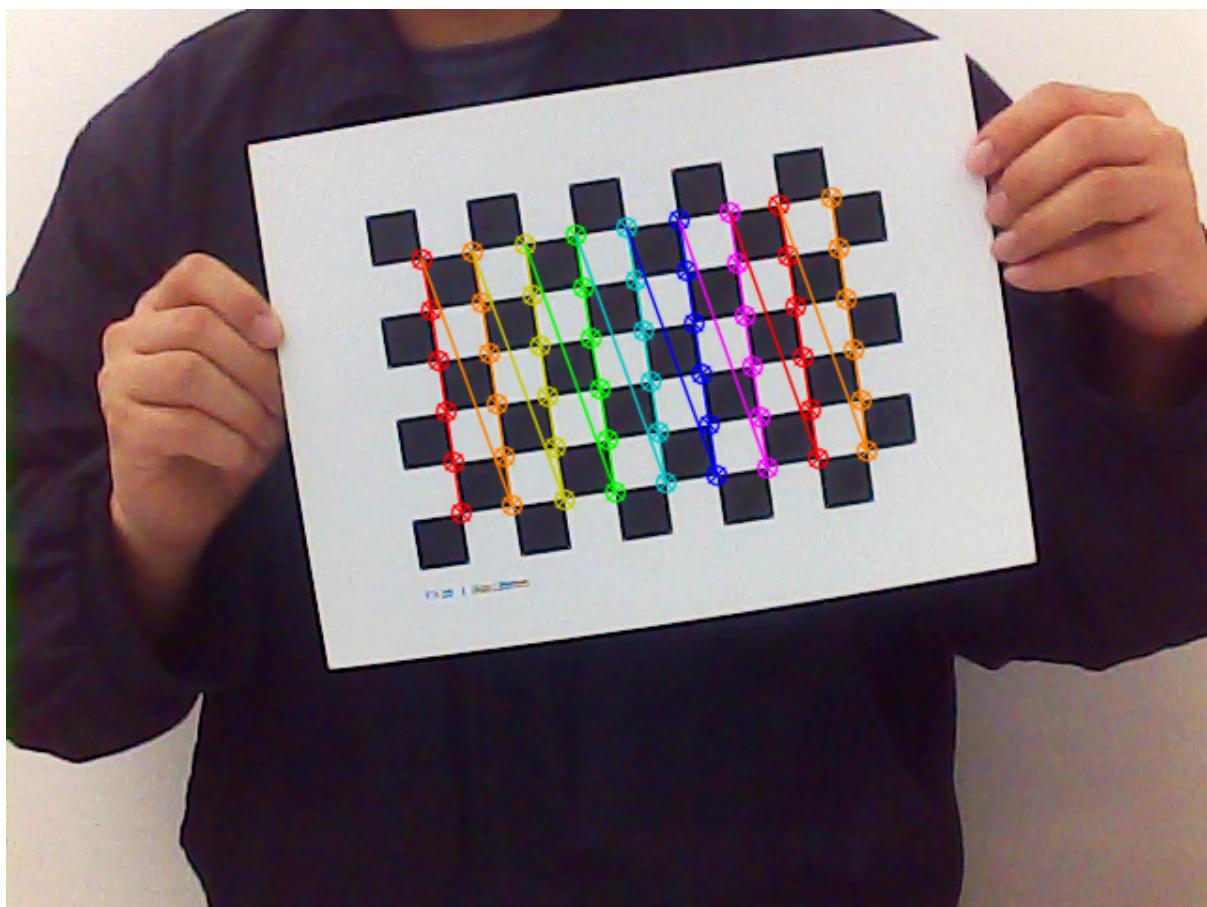


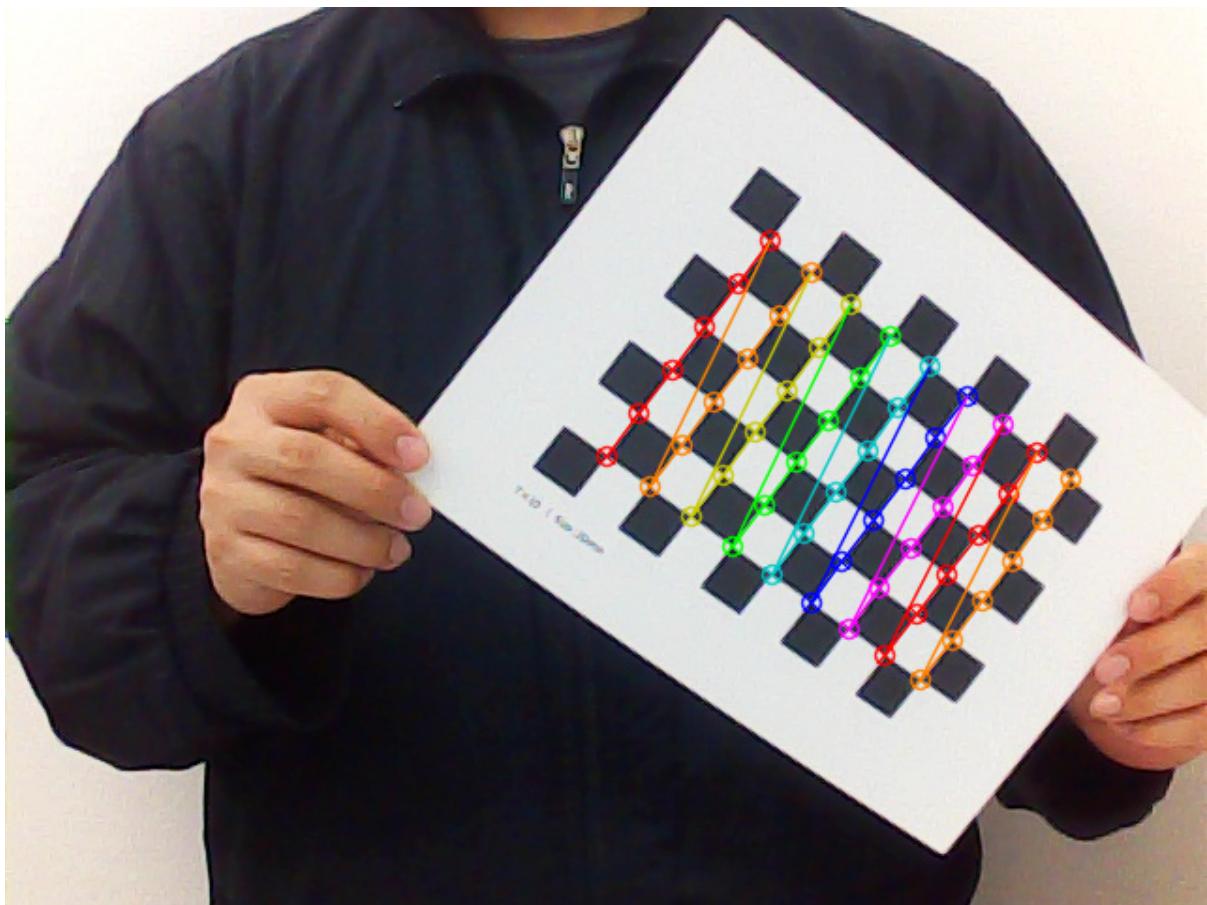












```
## 출력결과

[晌] 내부 파라미터 (Camera Matrix):
[[986.93180332  0.          251.57937661]
 [ 0.           983.85767546 131.93997962]
 [ 0.           0.           1.          ]]

[晌] 왜곡 계수 (Distortion Coefficients):
 [[-2.77060722e-01 -1.73446994e+00  4.61497572e-03  4.60231034e-03
   6.43334573e+00]]
```

3.7.1.1 Reprojection error 구하기 (보충 설명 필요)

```
import cv2
import numpy as np
import os
import glob

# === 사용자 설정 ===
```

```

image_dir = "./pics" # 체스보드 이미지 폴더 경로
width = 6             # 체스보드 가로 코너 수
height = 9            # 체스보드 세로 코너 수
square_size = 0.02    # 체스보드 한 칸 길이 (미터)
visualize = True      # 코너 시각화 여부

# === 체스보드 3D 좌표 준비 ===
objp = np.zeros((width * height, 3), np.float32)
objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)
objp *= square_size

# === 포인트 저장 변수 ===
objpoints = [] # 실제 3D 포인트
imgpoints = [] # 이미지상 2D 포인트

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

images = glob.glob(os.path.join(image_dir, "*.png"))

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

    if ret:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners2)

        if visualize:
            vis = cv2.drawChessboardCorners(img, (width, height), corners2, ret)
            cv2.imshow("Corners", vis)
            cv2.waitKey(100)
    else:
        print(f"[X] 체스보드 감지 실패: {fname}")

cv2.destroyAllWindows()

# === 카메라 보정 ===
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
    objpoints, imgpoints, gray.shape[::-1], None, None)

# === 리프로젝션 에러 계산 ===
total_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2) / len(imgpoints2)
    total_error += error

mean_error = total_error / len(objpoints)

# === 결과 출력 ===
print("\n☞ 내부 파라미터 (Camera Matrix):\n", mtx)
print("\n☞ 왜곡 계수 (Distortion Coefficients):\n", dist.ravel())
print(f"\n☞ 평균 리프로젝션 에러: {mean_error:.4f} 픽셀")

# === 결과 저장 ===

```

```
np.save("calibration_matrix.npy", mtx)
np.save("distortion_coefficients.npy", dist)
```

```
## 출력결과

❖ 내부 파라미터 (Camera Matrix):
[[986.93180332  0.          251.57937661]
 [ 0.          983.85767546 131.93997962]
 [ 0.          0.          1.          ]]

❖ 왜곡 계수 (Distortion Coefficients):
[-2.77060722e-01 -1.73446994e+00  4.61497572e-03  4.60231034e-03
 6.43334573e+00]

❖ 평균 리프로젝션 에러: 0.1658 픽셀
(중간에
imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist) dist 값을
none으로)
```

```
...
Sample Usage:-  

python calibration.py --dir calibration_checkerboard/ --square_size 0.024  

...  

import numpy as np
import cv2
import os
import argparse

def calibrate(dirpath, square_size, width, height, visualize=False):
    """ Apply camera calibration operation for images in the given directory path. """

    # termination criteria
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(8,6,0)
    objp = np.zeros((height*width, 3), np.float32)
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)

    objp = objp * square_size

    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d point in real world space
    imgpoints = [] # 2d points in image plane.

    images = os.listdir(dirpath)
```

```

for fname in images:
    img = cv2.imread(os.path.join(dirpath, fname))

    if img is None:
        print(f"[X] 이미지 불러오기 실패: {fname}")
        continue

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

    # If found, add object points, image points (after refining them)
    if ret:
        print(f"[O] 체스보드 감지 성공: {fname}")
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (width, height), corners2, ret)
    else:
        print(f"[X] 체스보드 감지 실패: {fname}")

    if visualize:
        cv2.imshow('img',img)
        cv2.waitKey(0)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)

return [ret, mtx, dist, rvecs, tvecs]

if __name__ == '__main__':
    ap = argparse.ArgumentParser()
    ap.add_argument("-d", "--dir", required=True, help="Path to folder containing
checkerboard images for calibration")
    ap.add_argument("-w", "--width", type=int, help="Width of checkerboard
(default=9)", default=9)
    ap.add_argument("-t", "--height", type=int, help="Height of checkerboard
(default=6)", default=6)
    ap.add_argument("-s", "--square_size", type=float, default=1, help="Length of one
edge (in metres)")
    ap.add_argument("-v", "--visualize", type=str, default="False", help="To visualize
each checkerboard image")
    args = vars(ap.parse_args())

    dirpath = args['dir']
    # 2.4 cm == 0.024 m
    # square_size = 0.024
    square_size = args['square_size']

    width = args['width']
    height = args['height']

```

```

if args["visualize"].lower() == "true":
    visualize = True
else:
    visualize = False

ret, mtx, dist, rvecs, tvecs = calibrate(dirpath, square_size, visualize=visualize,
width=width, height=height)

print(mtx)
print(dist)

np.save("calibration_matrix", mtx)
np.save("distortion_coefficients", dist)

"""

$ python3 calibration.py --dir /home/addinedu/cali_test/pics --width 6 --height 9
--square_size 0.02 --visualize True
"""

```

3.7.2 (두번째 관제 캠)내부 파라미터 & 왜곡계수 & Reprojection error

3.7.2.1 카메라 Calibration으로 내부 파라미터와 왜곡 계수 구하기

```

"""

Sample Usage:-
python calibration.py --dir calibration_checkerboard/ --square_size 0.024
"""

import numpy as np
import cv2
import os
import argparse

def calibrate(dirpath, square_size, width, height, visualize=False):
    """ Apply camera calibration operation for images in the given directory path. """

    # termination criteria
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(8,6,0)
    objp = np.zeros((height*width, 3), np.float32)
    objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)

    objp = objp * square_size

    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d point in real world space
    imgpoints = [] # 2d points in image plane.

    images = os.listdir(dirpath)

```

```

for fname in images:
    img = cv2.imread(os.path.join(dirpath, fname))

    if img is None:
        print(f"[X] 이미지 불러오기 실패: {fname}")
        continue

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

    # If found, add object points, image points (after refining them)
    if ret:
        print(f"[O] 체스보드 감지 성공: {fname}")
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (width, height), corners2, ret)
    else:
        print(f"[X] 체스보드 감지 실패: {fname}")

    if visualize:
        cv2.imshow('img',img)
        cv2.waitKey(0)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)

return [ret, mtx, dist, rvecs, tvecs]

if __name__ == '__main__':
    ap = argparse.ArgumentParser()
    ap.add_argument("-d", "--dir", required=True, help="Path to folder containing
checkerboard images for calibration")
    ap.add_argument("-w", "--width", type=int, help="Width of checkerboard
(default=9)", default=9)
    ap.add_argument("-t", "--height", type=int, help="Height of checkerboard
(default=6)", default=6)
    ap.add_argument("-s", "--square_size", type=float, default=1, help="Length of one
edge (in metres)")
    ap.add_argument("-v", "--visualize", type=str, default="False", help="To visualize
each checkerboard image")
    args = vars(ap.parse_args())

    dirpath = args['dir']
    # 2.4 cm == 0.024 m
    # square_size = 0.024
    square_size = args['square_size']

    width = args['width']

```

```

height = args['height']

if args["visualize"].lower() == "true":
    visualize = True
else:
    visualize = False

ret, mtx, dist, rvecs, tvecs = calibrate(dirpath, square_size, visualize=visualize,
width=width, height=height)

print(mtx)
print(dist)

np.save("calibration_matrix", mtx)
np.save("distortion_coefficients", dist)

```

```

python3 /home/addinedu/Downloads/cali_test/calibration.py --dir /home/addinedu/
Downloads/cali_test/second\ pics/ --square_size 0.02
#실행코드(2번째관제캠)

```

```

## 출력결과

[내부 파라미터 (Camera Matrix):
 [[1.06251780e+03 0.00000000e+00 9.54998938e+02]
 [0.00000000e+00 1.06496688e+03 5.57370338e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

[외곡 계수 (Distortion Coefficients):
 [[ 0.04556705 -0.24528447 -0.00753058 -0.00275635  0.22951817]]]
```

3.7.2.1.1 Reprojection error 구하기

```

import cv2
import numpy as np
import os
import glob

# === 사용자 설정 ===
image_dir = "./pics" # 체스보드 이미지 폴더 경로
width = 6 # 체스보드 가로 코너 수
height = 9 # 체스보드 세로 코너 수
square_size = 0.02 # 체스보드 한 칸 길이 (미터)
visualize = True # 코너 시각화 여부

# === 체스보드 3D 좌표 준비 ===

```

```

objp = np.zeros((width * height, 3), np.float32)
objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2)
objp *= square_size

# === 포인트 저장 변수 ===
objpoints = [] # 실제 3D 포인트
imgpoints = [] # 이미지상 2D 포인트

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

images = glob.glob(os.path.join(image_dir, "*.png"))

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

    if ret:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners2)

        if visualize:
            vis = cv2.drawChessboardCorners(img, (width, height), corners2, ret)
            cv2.imshow("Corners", vis)
            cv2.waitKey(100)
    else:
        print(f"[X] 체스보드 감지 실패: {fname}")

cv2.destroyAllWindows()

# === 카메라 보정 ===
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
    objpoints, imgpoints, gray.shape[::-1], None, None)

# === 리프로젝션 에러 계산 ===
total_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2) / len(imgpoints2)
    total_error += error

mean_error = total_error / len(objpoints)

# === 결과 출력 ===
print("\n☞ 내부 파라미터 (Camera Matrix):\n", mtx)
print("\n☞ 왜곡 계수 (Distortion Coefficients):\n", dist.ravel())
print(f"\n☞ 평균 리프로젝션 에러: {mean_error:.4f} 픽셀")

# === 결과 저장 ===
np.save("calibration_matrix.npy", mtx)
np.save("distortion_coefficients.npy", dist)

```

3.8 다중 로봇 경로 계획 정리본

[feat.2대장님의 다정한 명령⁴](#)

[robot initialpose가 어디에 적용 되는지⁵](#)

3.9 음성 인식, LLM 관련



이찬종/관련 Confluence 링크 정리

[스터디] 디렉토리 내부의 페이지 링크로 정리

[음성 명령 with SST, GPT API \(작성 예정 아마 ~ 7/13?\) \(see page 219\)](#)

3.10 이미지 처리, 카메라 관련



이찬종/관련 Confluence 링크 정리

[스터디] 디렉토리 내부의 페이지 링크로 정리

[Visual C++, about 비트맵 \(see page 214\)](#)

[ArUco marker + RPi 카메라 + Flask \(작성중\) \(see page 205\)](#)

[YOLOv8 + ROS2 \(Python ver.\) \(see page 197\)](#)

3.11 다중 주행

HBRS-SDP

다중로봇이 존재하는 경우 중앙 dispatcher 가 로봇의 상태를 보고 어떤 로봇이 어떤 작업을 맡을지 정해주는 시스템을

ros2로 구현한것

4. https://www.notion.so/2-2251adbda1df80959eedefdd85308c63?source=copy_link

5. https://www.notion.so/robot-initialpoes-22f1adbda1df80ae9386cd724fd8acaa?source=copy_link

호출 수신 : dispatcher로부터 task를 개별로봇이 받음

즉 각로봇별로 dispatcher가 해당로봇(가장가깝거나 일을 안하고있는)에게만 task를 publish하는 방식

따라서 로봇별로 assign_task토픽을 만들어서 publish하는 구조

```
[dispatcher_node] ← 한 개
|
|← goal → /pinky1/navigate_to_pose
|← goal → /pinky2/navigate_to_pose
|
[핑키1] → bt_navigator → [BT Tree 실행: NavigateToPose, Spin, Wait 등]
[핑키2] → bt_navigator → [BT Tree 실행: NavigateThroughPoses, Recovery 등]
```

pinky1/ |— bt_navigator |— controller_server |— planner_server |— behavior_tree xml |— costmap

멀티 핑키 제어 로직

1. 사용자가 호출을 요청(앱이나 UI)
2. dispatcher가 호출순서를 선택(우선순위 판단해 적절한 핑키 선택)
3. 선택된 핑키에 goal위치를 전송(NavigateToPose)

: dispatcher와 핑키의 통신방식 ⇒ /pinkyN/navigate_to_pose action으로 목표 위치이동

1. 핑키들이 경로탐색 및 서로 장애물로 인식해 협조
2. 핑키는 경로를 따라 주행하며 주기적으로 경로 재계산
3. 도착 후 상태를 보고
4. dispatcher는 다음 호출을 처리

4,5에 대한 로직:

1. 각로봇이 라이다 기반으로 앞에 움직이는 물체를 감지해 장애물로판단 그러나 라이다로는

동적장애물로판단을 못함

⇒ 상대로봇의 위치를 토픽으로 받아 costmap에 직접 반영

각 핑키는 /tf, /odom, /robot_pose등 위치정보를 publish하고 다른 핑키는 이를 가짜센서처럼 obstacle layer에 추가

ex. /pinky2_pose → /pinky1/costmap/obstacle_layer가 수신

1. 경로를 따라 주행하면서 주기적으로 경로를 재계산하는 방식

알고리즘 : TEB (time elastic band)

⇒ 로봇의 동역학, 장애물, 시간 정보까지 고려해서 경로를 재계산하는 local planner

실시간으로 trajectory를 최적화해서 가는

흐름 : 전역경로는 global_planner로 만들고, local planner로 teb를 설정해서 최적화된 경로 따라가게 하고

다른 펑키위치를 동적인 장애물로 costmap에 넣으면 자율주행

3.11.1 다중 로봇 경로 조사

https://lee-jaewon.github.io/multi_robot/Multi_Robot_SLAM_overview/

3.11.2 MAPF-Multi Agent Path Finding (A* 확장)

로봇이 맵을 공유해야함. 즉 맵에서 Global 위치를 공유해야하고 서로의 Goal point도 공유해야함.

경로 계획을 할 때 서로의 경로가 겹치는 부분이 있는지 확인하고 겹치는 부분이 있다면 충돌 시간을 계산해서 속도 및 경로를 조절하는 방식으로 진행해야함.

3.11.2.1 Path finding

1. Local Repair A*(LRA*)
2. Cooperative A*(CA*)
3. Hierarchical CA*(HCA*)
4. Windowed HCA*(WHCA*)

3.11.2.1.1 LRA*

1. 각 에이전트들은 다른 에이전트와 인접한 경우를 제외하고는 타 에이전트들을 무시하고 경로를 계획합니다.
2. 에이전트들은 계획된 경로대로 이동하다가, 다른 에이전트와 충돌이 발생하게 될 경우에만 경로를 재계획합니다.
 - LRA* 알고리즘은 빠르고, 간단하다는 장점이 있지만 복잡한 환경에서 여러 에이전트를 다룰 수 없다는 명확한 한계가 존재합니다.
 - 좁은 구간에 여러 에이전트가 몰릴 경우, 무차별적인 경로 재계획이 발생하면서 오히려 LRA* 때문에 cyclic conflict가 발생할 수 있습니다.

3.11.2.1.1.1 설명:

- 각 에이전트는 자신의 현재 위치에서 A*로 1스텝만 계획하고 이동한다.
- 충돌이 발생하면 그때마다 다시 A*를 수행해 회피한다.
- 완전히 로컬한 지식만 사용하며, 협조는 없다.

3.11.2.1.1.2 수도 코드:

```

for each agent:
    initialize current position
    while not at goal:
        next_move = A*_plan(current_position, goal, obstacles)
        if next_move is occupied by another agent:
            wait or replan
    
```

```

else:
    move to next_move

```

3.11.2.2 CA*

1. Agents={A1,A2,... An}, (reservation table = 시간 t에 대해 좌표가 점유되어 있는지 여부를 관리하는 3차원)
 2. A =Agents.pop();
 3. 에이전트 A에 대해서 reservation table에서 점유된 좌표들을 피하며 시간-경로를 계획한다.
 4. 3에서 계획한 시간-경로를 reservation table에 표시한다.
 5. Agents에 에이전트가 남아있으면 2로 돌아간다.
- 참고: 최적의 경로를 미리 계산하여 에이전트들에게 할당하는 decoupled greedy 알고리즘들은 한 에이전트를 path가 다른 에이전트의 path를 침범하면 문제를 해결할 수 없다.

3.11.2.2.1 ☰ 설명:

- 각 에이전트는 A*를 사용하되, **예약 테이블(reservation table)**을 참조하여 다른 에이전트의 경로를 회피하도록 계획한다.
- 시간-공간(time-space) 충돌을 고려하며, 완전히 중앙집중식 방식으로 동작한다.

3.11.2.2.2 ☰ 수도 코드:

```

initialize reservation_table = {}

for each agent in order:
    path = A*_plan(start, goal, reservation_table)
    reserve path in reservation_table

```

3.11.2.3 HCA*

일반적으로 abstract space에서 heuristic 함수의 성능을 높이는 방법은 2가지가 있다.

1. abstract space에서의 최적 경로를 미리 계산해두는 것
2. hierarchical abstraction을 적용하는 것

1의 경우에는 사용 가능한 경로가 동적으로 변하는 복잡한 멀티 에이전트 환경에서 적합하지 않다.

2의 경우에는 1) 좋은 hierarchical abstraction을 사용하고, 2) 각 abstraction에서 계산된 값들을 빠르게 재활용해야 한다.

1. 각 에이전트에 대해서는 다른 에이전트들을 무시하고 abstract space의 최적 경로를 저장하는 2차원 맵을 이용하여 heuristic 함수를 사용한다.
2. Reversed Resumable A* (RRA*)를 이용하여 abstraction에서 계산된 값들을 재활용한다.

→ 사용하기 적합하다.

▣ 설명:

- CA*와 비슷하지만, 다른 에이전트의 미래 행동까지 전체적으로 고려하지 않는다.
- 경로는 먼저 계산하되, 현재 노드와 다음 노드의 충돌만 피하면서 경로를 수정한다.

```
initialize reservation_table = {}

for each agent in order:
    low_level_path = A*_plan(start, goal) # ignore reservations
    for each step in low_level_path:
        if step is reserved in reservation_table:
            replan local detour
            reserve step in reservation_table
```

3.11.2.3.1 ▣ 수도 코드:

하지만 HCA*도 문제들이 있다.

1. 미션이 종료된 에이전트들이 다른 에이전트의 경로를 막아버리는 경우 valid solution을 찾을 수 없다.
2. 속도를 향상시키기 위해서 매우 많은 메모리를 사용하게 된다.

3.11.2.3.2 WHCA*

논문에서는 각 에이전트들이 목적지까지의 경로를 탐색하는 것이 아니라, 주어진 윈도우 사이즈만큼만 경로를 탐색하는 것으로 문제를 해결했다.

에이전트들은 경로를 어느 정도 진행했을 때, 다시 윈도우 사이즈만큼만 경로를 탐색한다.

모든 에이전트가 목적지에 도착할 때까지 경로 탐색이 반복된다.

처음부터 모든 경로를 계획하는 것이 아니기 때문에 많은 문제가 해결된다는 것이 특징이다.

윈도우 사이즈를 적절히 설정하는 것으로 서로의 상태를 고려할 수 있게 되는 것이다.

그리고 시간이 지날 때마다 에이전트들의 경로를 재탐색하기 때문에 HCA*에서 지적되었던 1,2도 해결된다. 모든 경로를 계획하는 것도 아니기 때문에 문제 3도 해결된다.

3.11.2.3.2.1 ▣ 설명:

- HCA*를 시간적으로 **고정된 윈도우(window size)**만큼만 고려한다.
- 예: 다음 5스텝까지만 다른 에이전트의 예약을 반영하여 경로를 계산한다.
- 윈도우가 지나면 다시 경로 재계산을 한다.

3.11.2.3.2.2 ▣ 수도 코드:

```
initialize reservation_table = {}
```

```
for each agent in order:
    windowed_path = A*_plan(start, goal, reservation_table, window_size)
    reserve windowed_path in reservation_table

while not all agents at goal:
    for each agent:
        follow planned path for one step
        if window expired:
            windowed_path = A*_plan(current_position, goal, reservation_table,
window_size)
            reserve new path in reservation_table
```

출처

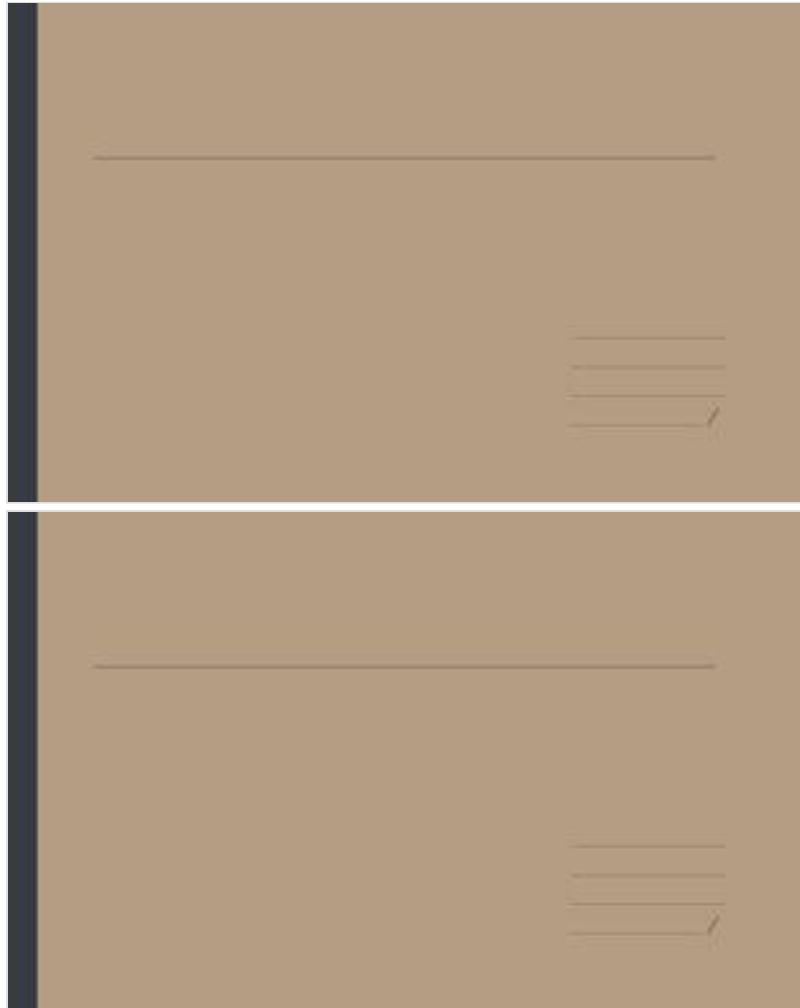
<https://goodahn.tistory.com/92>

<https://github.com/igrek51/coop-pathfinder>

4 스터디

4.1 선형대수

4.1.1 선형대수-(기본+심화)



4.2 관제

4.2.1 다중 로봇 경로 조사 (MAPF-Multi Agent Path Finding)

로봇이 맵을 공유하고 Global 위치를 공유하고, 서로의 Goal point도 공유한다.

경로 계획을 할 때 서로의 경로가 겹치는 부분이 있는지 확인하고 겹치는 부분이 있다면 충돌 시간을 계산해서 속도 및 경로를 조절하는 방식으로 진행해야함.

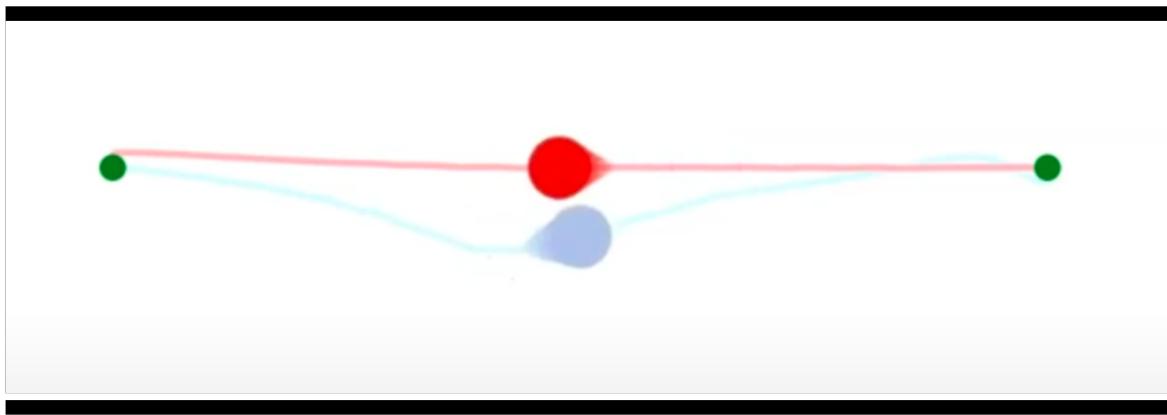
4.2.1.1 1.로봇간에 이동 통로를 완전히 분리한다.(협로 주행)

- 통로가 엄청 좁은 경우(로봇만 이동하는 통로)
- <https://jiaoyangli.me/research/mapf/>



4.2.1.2 2. (큰 통로) 동일한 통로를 이동하되 서로 부딪히지 않는 local plan을 수행한다.

- 이동 통로가 로봇 두대 혹은 로봇과 사람이 이동하기 충분히 큰 경우
- 통로의 크기를 고려해서 **우측 주행**한다.
- 사람과 함께 주행하는 경우 로봇의 이동속도를 낮추어 주행한다.

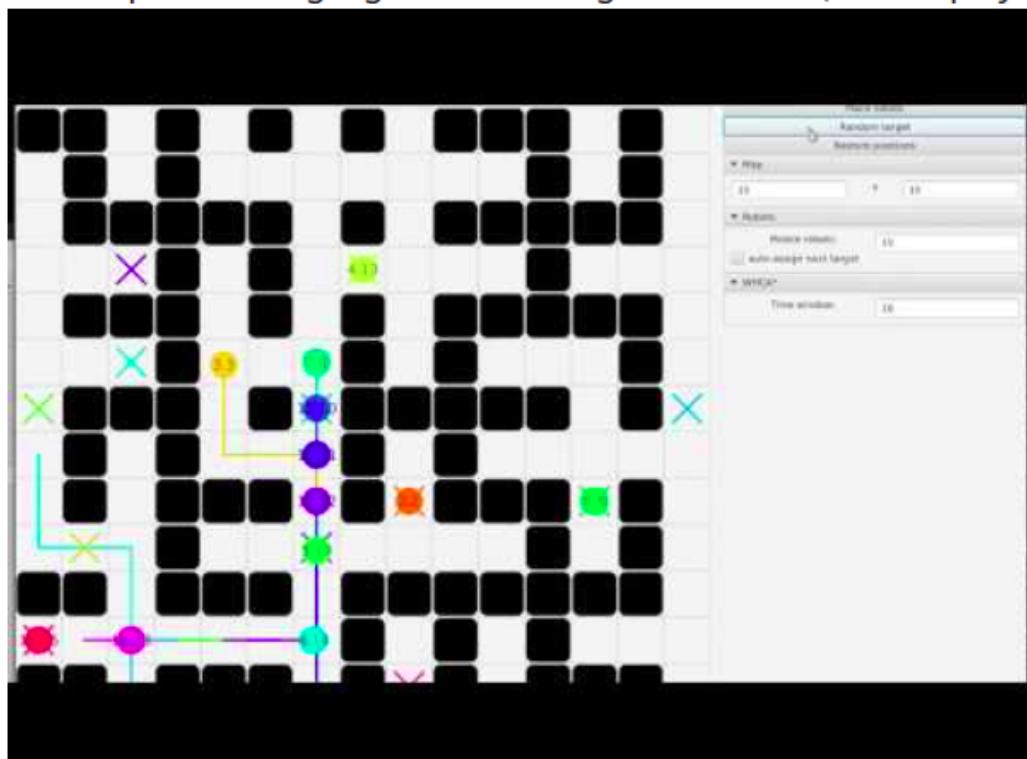


4.2.1.3 Path finding 기법(A* 개선버전)

<https://github.com/jcontrerasleyton/MAPF>

<https://github.com/igrek51/coop-pathfinder>

WHCA*-based path-finding algorithm solving bottlenecks (click to play)



video):

4.2.1.3.1 Path finding

1. Local Repair A*(LRA*)

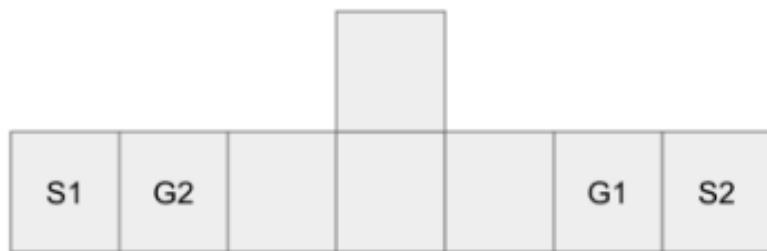
2. Cooperative A*(CA*)
3. Hierarchical CA*(HCA*)
4. Windowed HCA*(WHCA*)

4.2.1.3.1.1 LRA*

1. 각 에이전트들은 다른 에이전트와 인접한 경우를 제외하고는 타 에이전트들을 무시하고 경로를 계획한다.
2. 에이전트들은 계획된 경로대로 이동하다가, 다른 에이전트와 충돌이 발생하게 될 경우에만 경로를 재계획 한다.
 - LRA*알고리즘은 빠르고, 간단하다는 장점이 있지만 복잡한 환경에서 여러 에이전트를 다룰 수 없다는 명확한 한계가 존재한다.
 - 좁은 구간에 여러 에이전트가 몰릴 경우, 무차별적인 경로 재계획이 발생하면서 오히려 LRA* 때문에 cyclic conflict가 발생할 수 있다.

4.2.1.3.2 CA*

1. Agents={A1,A2,... An}, (reservation table = 시간 t에 대해 좌표가 점유되어 있는지 여부를 관리하는 3차원)
 2. A =Agents.pop();
 3. 에이전트 A에 대해서 reservation table에서 점유된 좌표들을 피하며 시간-경로를 계획한다.
 4. 3에서 계획한 시간-경로를 reservation table에 표시한다.
 5. Agents에 에이전트가 남아있으면 2로 돌아간다.
- 참고: 최적의 경로를 미리 계산하여 에이전트들에게 할당하는 decoupled greedy 알고리즘들은 한 에이전트를 path가 다른 에이전트의 path를 침범하면 문제를 해결할 수 없다.



greedy, decoupled 알고리즘으로 이거 풀기 어렵죠 ㅜ

4.2.1.3.3 HCA*

일반적으로 abstract space에서 heuristic 함수의 성능을 높이는 방법은 2가지가 있다.

1. abstract space에서의 최적 경로를 미리 계산해두는 것
2. hierarchical abstraction을 적용하는 것

1의 경우에는 사용 가능한 경로가 동적으로 변하는 복잡한 멀티 에이전트 환경에서 적합하지 않다.

2의 경우에는 1) 좋은 hierarchical abstraction을 사용하고, 2) 각 abstraction에서 계산된 값을 빠르게 재활용해야 한다.

1. 각 에이전트에 대해서는 다른 에이전트들을 무시하고 abstract space의 최적 경로를 저장하는 2차원 맵을 이용하여 heuristic 함수를 사용한다.

2. Reversed Resumable A* (RRA*)를 이용하여 abstraction에서 계산된 값을 재활용한다.

→ 사용하기 적합하다.

하지만 HCA*도 문제들이 있다.

1. 미션이 종료된 에이전트들이 다른 에이전트의 경로를 막아버리는 경우 valid solution을 찾을 수 없다.
2. 속도를 향상시키기 위해서 매우 많은 메모리를 사용하게 된다.

WHCA*

논문에서는 각 에이전트들이 목적지까지의 경로를 탐색하는 것이 아니라, 주어진 윈도우 사이즈만큼만 경로를 탐색하는 것으로 문제를 해결했다.

에이전트들은 경로를 어느 정도 진행했을 때, 다시 윈도우 사이즈만큼만 경로를 탐색한다.

모든 에이전트가 목적지에 도착할 때까지 경로 탐색이 반복된다.

처음부터 모든 경로를 계획하는 것이 아니기 때문에 많은 문제가 해결된다는 것이 특징이다.

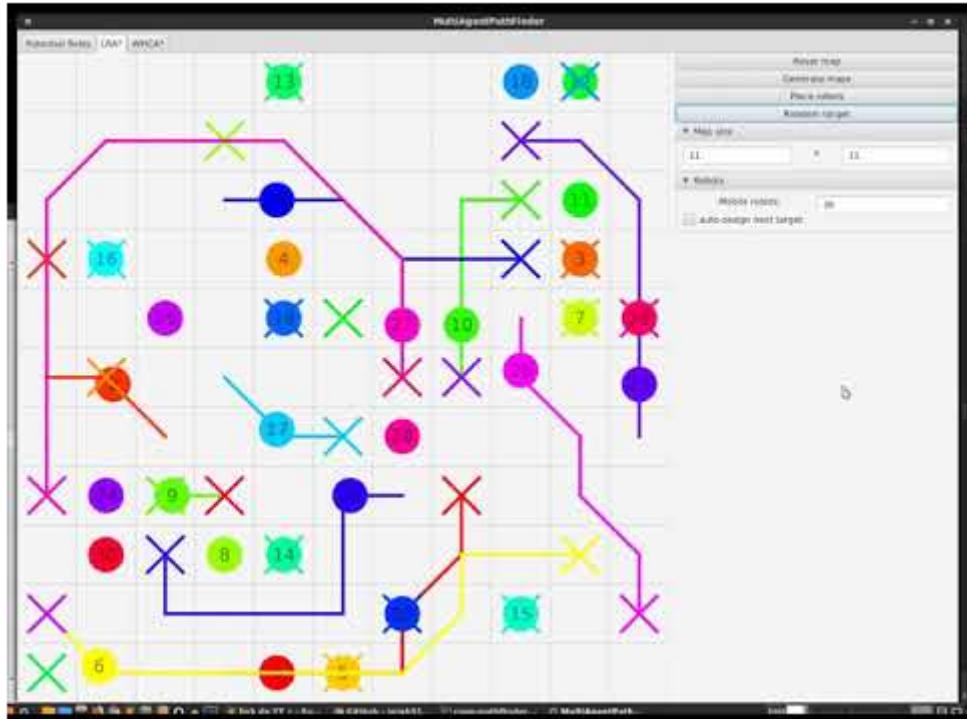
윈도우 사이즈를 적절히 설정하는 것으로 서로의 상태를 고려할 수 있게 되는 것이다.

그리고 시간이 지날 때마다 에이전트들의 경로를 재탐색하기 때문에 HCA*에서 지적되었던 1,2도 해결된다. 모든 경로를 계획하는 것도 아니기 때문에 문제 3도 해결된다.

출처

<https://goodahn.tistory.com/92>

Local-Repair A* path-finding algorithm simulation (click to play video):



Local-Repair A* (LRA*)는 동적 환경에서 효율적인 경로 계획을 위해 개발된 알고리즘입니다. 기존의 A* 알고리즘을 확장하여 환경 변화에 대응할 수 있도록 설계되었습니다.

4.2.1.3.3.1 기본 개념

LRA*는 환경에 변화가 발생했을 때 전체 경로를 다시 계산하는 대신, 영향을 받는 부분만 국부적으로 수정하는 접근 방식을 사용합니다. 이를 통해 계산 효율성을 크게 향상시킵니다.

4.2.1.3.3.2 주요 특징

증분적 재계획: 환경 변화가 감지되면 전체 경로를 재계산하지 않고 변화된 부분과 그 주변 영역만 수정합니다. 이는 특히 작은 변화가 자주 발생하는 환경에서 유용합니다.

일관성 유지: 알고리즘은 휴리스틱 함수의 일관성을 유지하면서 최적성을 보장합니다. 경로 수정 과정에서도 A*의 최적성 보장 특성을 유지합니다.

효율적인 메모리 사용: 이전 계산 결과를 재사용하여 메모리와 계산 시간을 절약합니다.

4.2.1.3.3.3 알고리즘 작동 원리

LRA*는 다음과 같은 단계로 작동합니다:

1. **초기 경로 계산:** 표준 A* 알고리즘을 사용하여 초기 최적 경로를 계산합니다.
2. **변화 감지:** 환경에서 장애물의 추가, 제거, 이동 등의 변화를 감지합니다.
3. **영향 영역 분석:** 변화가 현재 경로에 미치는 영향을 분석하고 수정이 필요한 영역을 식별합니다.
4. **국부적 수정:** 영향을 받는 부분만 재계산하여 경로를 수정합니다. 이 과정에서 기존 계산 결과를 최대한 활용합니다.

5. **최적성 검증:** 수정된 경로가 여전히 최적인지 확인하고 필요시 추가 조정을 수행합니다.

4.2.1.3.3.4 적용 분야

LRA*는 다음과 같은 영역에서 특히 유용합니다:

- **로봇 내비게이션:** 동적 환경에서 움직이는 로봇의 실시간 경로 계획
- **게임 AI:** 실시간 전략 게임에서 유닛의 경로 계획
- **자율주행:** 교통 상황 변화에 따른 경로 재계획
- **네트워크 라우팅:** 네트워크 토플로지 변화에 따른 경로 재계산

4.2.1.3.3.5 장점과 한계

장점:

- 전체 재계산 대비 현저히 빠른 처리 속도
- 메모리 효율성
- 실시간 응용에 적합
- 기존 A*의 최적성 보장

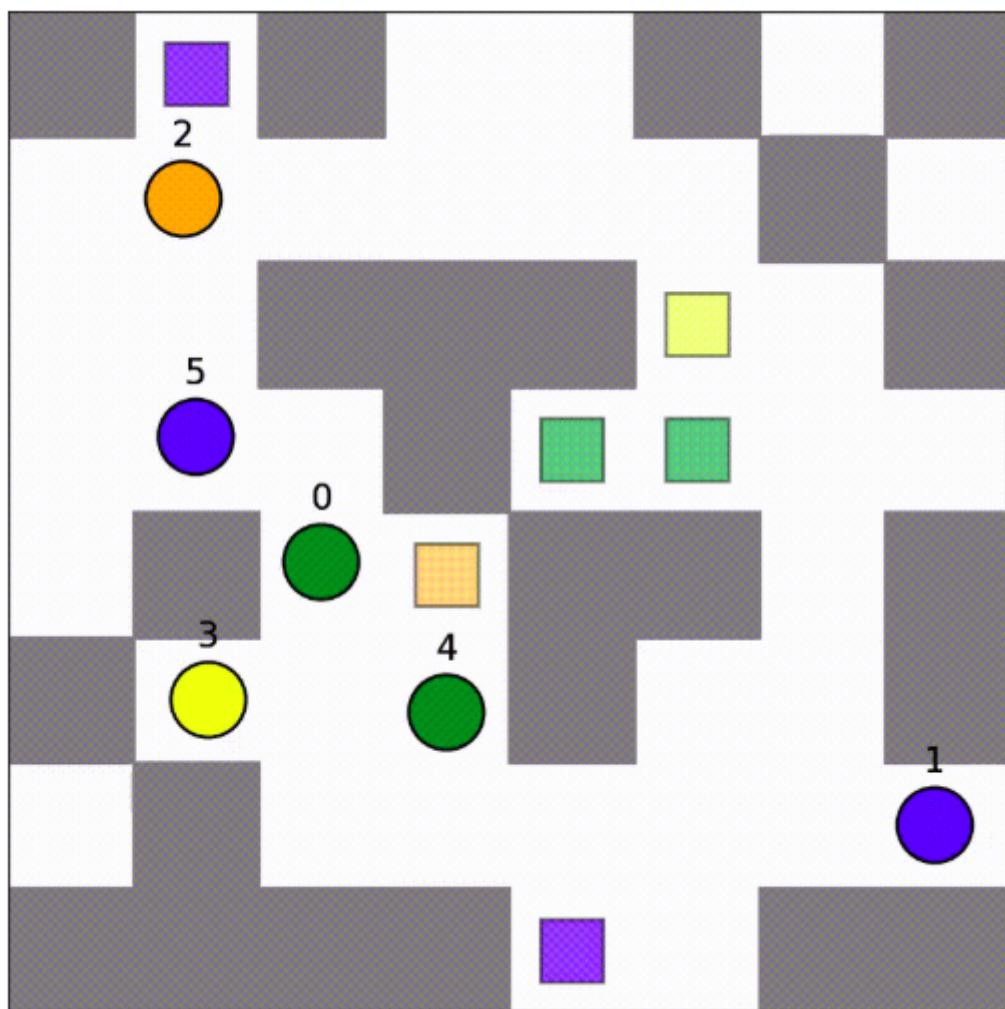
한계:

- 구현 복잡성이 높음
- 대규모 환경 변화 시 성능 저하 가능
- 초기 구현 비용이 높음

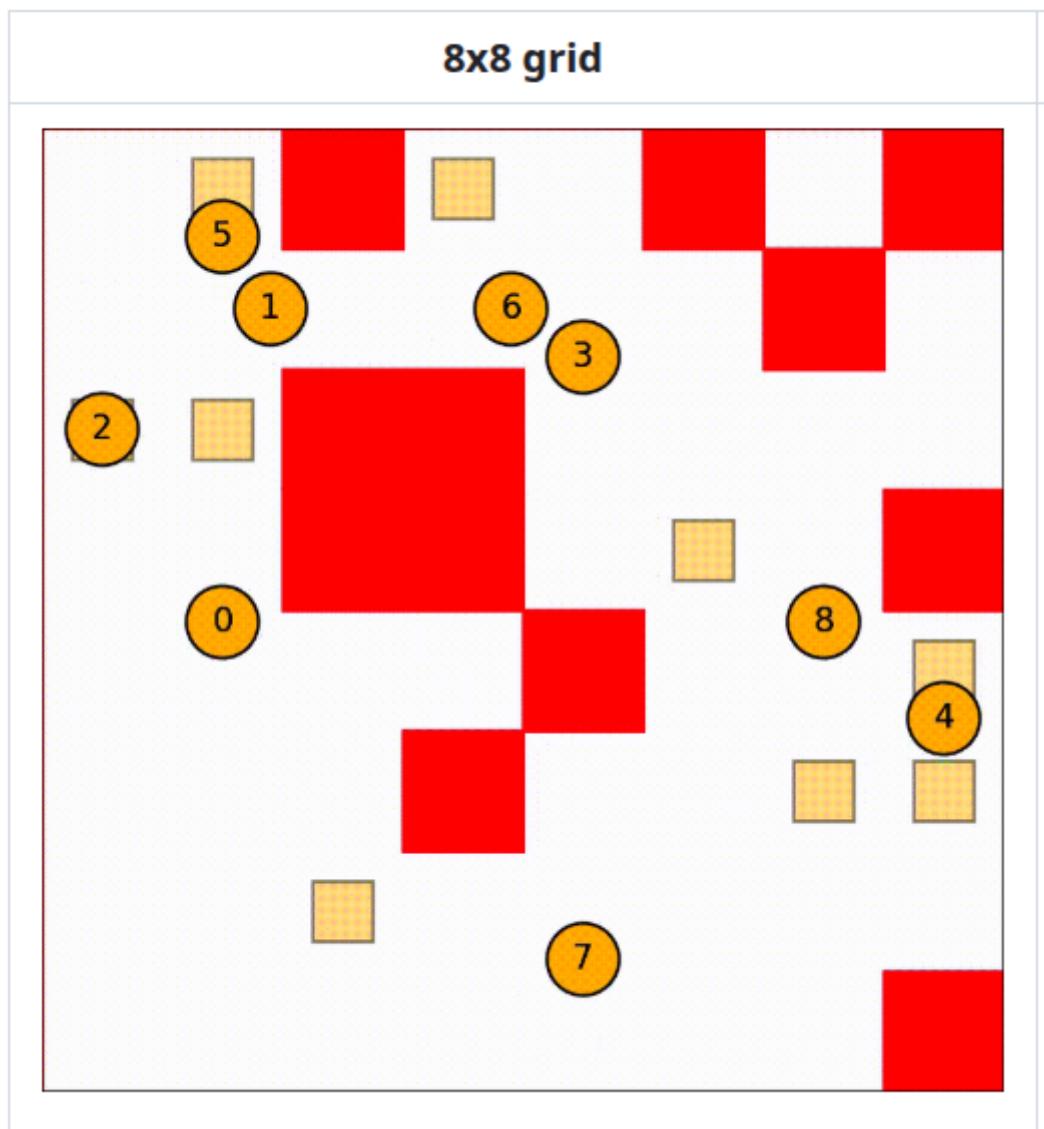
LRA*는 동적 환경에서의 경로 계획 문제를 해결하는 강력한 도구로, 실시간 제약이 있는 시스템에서 특히 가치가 큽니다.

4.2.1.4 Multi_agent_path_planning

<https://github.com/nicofretti/MAPF/tree/master>



https://github.com/atb033/multi_agent_path_planning?tab=readme-ov-file#centralized-solutions



4.2.1.5 Navigation

<https://docs.nav2.org/concepts/index.html>

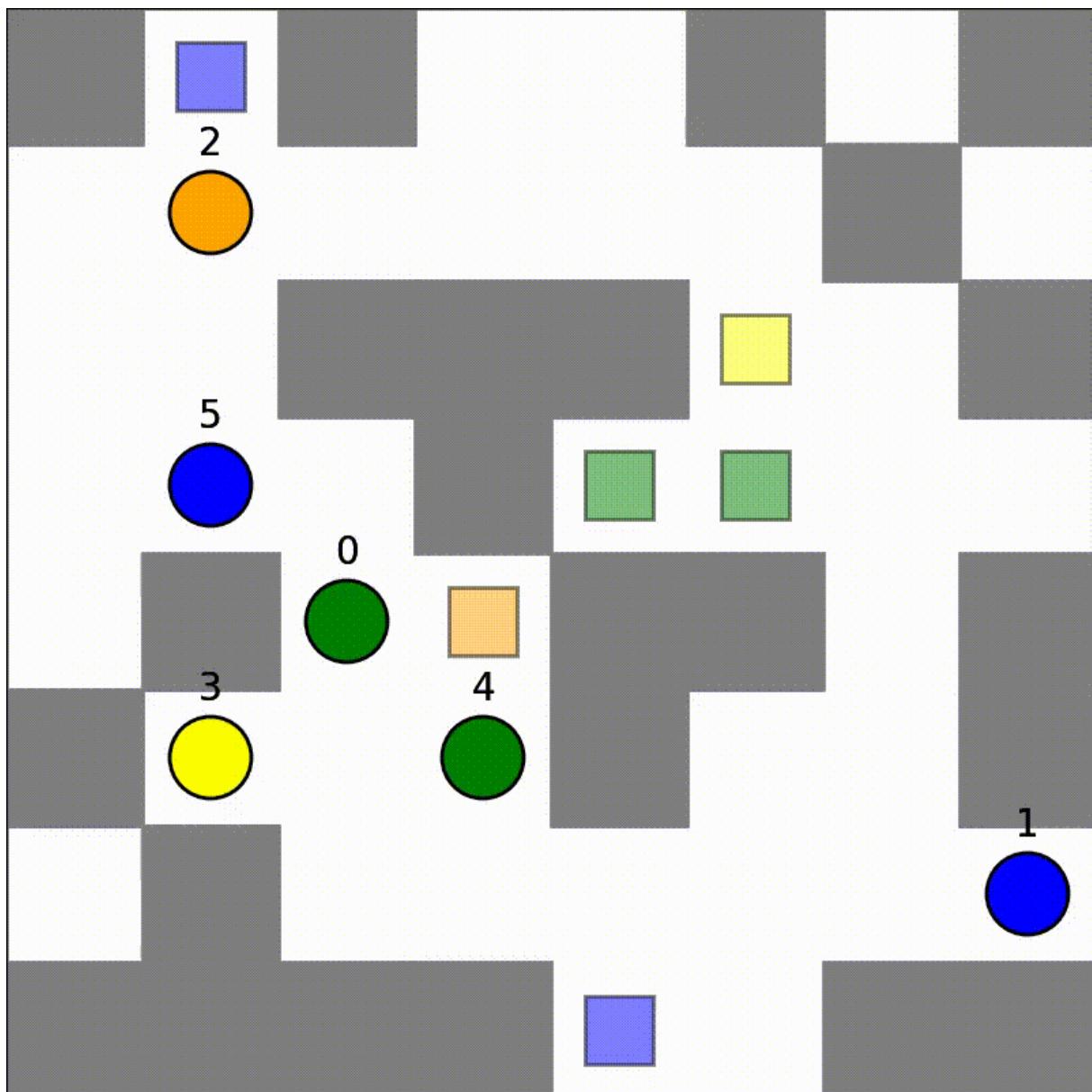
4.2.1.6 2대장님(제우님)의 다정한 명령(다중 로봇 경로 조사)

4.2.1.6.1 다중 로봇 경로 기술 조사 (MAPF-Multi Agent Path Finding), 코드분석

<https://github.com/nicofretti/MAPF>



python run_experiments.py --disjoint --random --solver CBS (주요코드)



4.2.1.6.2 관련 링크

<https://goodahn.tistory.com/130> : Task and Path Planning For Multi Agent Pickup and Delivery

작업 배정, 경로 계획을 분리할 때 참고 링크(알고리즘 / Assignment Problem)

4.2.1.6.2.1 TA-Prioritized(Prioritize Planning을 이용해서 경로 계획)

- 각 agent(로봇)들의 남은 작업 수행 시간이 큰 순서대로 우선 순위를 정합니다.
- 작업을 배정 받지 못한 agent들이 모두 작업을 배정 받을 때 까지 진행

```
{
    a1 : [(t1, p1, 10, d1, 10), (t2, p2, 9, d2, 9)],
    a2 : [(t3, p3, 10, d3, 9), (t4, p4, 9, d4, 8)],
    a3 : [(t5, p5, 10, d5, 8), (t6, p6, 9, d6, 7)]
}
```

	a1	a2	a3	경로 계획 결과
남은 작업 수행 시간 1	38	36	34	a1의 p1 경로 계획
남은 작업 수행 시간 2	28	36	34	a2의 p3 경로 계획
남은 작업 수행 시간 3	28	26	34	a3의 p5 경로 계획

4.2.1.6.2.2 1.상황설명

- 3대의 로봇 존재
- 각 로봇은 여러 작업을 받아서 픽업 → 딜리버리 하는 경로를 계획 해야 한다.
- 순서대로 a1 → a2 → a3 순서로 경로를 계산.

4.2.1.6.2.3 2.문제상황

ex) a3의 경로를 계획할 때

- 이미 a1,a2는 자신의 경로를 다 계획해서 고정된 위치에 있을 거라고 생각함
- 그런데 이 위치들이 a3가 지나가야 하는 길을 막고 있는 것

→a3은 경로를 찾지 못할 수도 있음(dead lock)에 빠지게 됨

4.2.1.6.2.4 3.해결책: Reserving Dummy Path

로봇이 멈춰 있다기 보다는 작업을 끝낸 뒤 자기 자리로 빠진다고 가정하고 경로를 계산.

- a1은 실제로 p1에서 멈춰서 있겠지만, 계산할 때 a1이 조용히 옆으로 빠져서 자기 주차장으로 간다고 가정
- a2도 p3에 있는게 아니라, 계산할 때 p3 작업이 끝나면 자리를 비운다고 가정

→ 결과 = a3는 막혀 있는 곳 없이 경로를 만들 수 있음

추가 정리⁶

6. <https://www.notion.so/2251adbda1df8002885deb8f050d453a?pvs=21>

4.2.1.6.3 요약:

“Reserving Dummy Path란, 앞선 로봇들이 실제 위치에 멈춰 있다고 가정하지 않고, 작업이 끝나면 주차 위치로 이동한다고 ‘가정’하는 기법입니다. 이렇게 하면 뒤따라오는 로봇들이 막히지 않고 경로를 계획할 수 있게 됩니다. 이는 실제로 로봇이 이동하는 건 아니며, 단지 경로 탐색 시 충돌 방지를 위해 사용되는 가상의 예약 방식입니다.”

4.2.1.6.4 TA-Hybrid 개요

TA-Hybrid는 MAPD 문제를 두 단계로 나눠 해결하는 접근법입니다.

- **MAPF**: 현재 할 일을 수행할 로봇들의 경로 계획
- **AMAPF**: 이후 할 일을 위해 작업을 바꿔야 하는 로봇들의 경로 계획 (작업 교환 포함)

용어 정의 (선행 이해 필요)

- **pickup location**: 로봇이 물건을 받는 위치
- **delivery location**: 로봇이 물건을 전달해야 하는 위치
- **MAPF**: 경로만 계획 (목표는 고정됨)
- **AMAPF**: 경로 + 작업 교환까지 포함 (목표가 바뀔 수 있음)
- **A1**: 현재 작업을 수행해야 할 로봇 집합
- **A2**: 새로운 작업을 맡을 예정인 로봇 집합
- **constraint**: 이미 결정된 경로 (다른 로봇이 사용하는 경로) → 경로 계산 시 피해야 함

4.2.1.6.5 TA-Hybrid 절차 요약

문제해결 단계⁷

4.2.1.6.6 두 알고리즘의 사용처 및 목적

항목	TA-Prioritized	TA-Hybrid
⽬적	빠르게 경로를 생성하고 deadlock을 방지	작업 교환(task exchange)까지 고려해 더 유연하게 경로와 작업을 최적화
핵심 전략	우선순위 기반 경로 계획 + dummy path 사용	CBS + 작업 교환(min-cost max-flow) + dummy path 사용
작업 교환	✗ 없음 (로봇이 자신의 작업만 수행)	✓ 있음 (로봇끼리 작업을 바꾸며 유연하게 대응)

7. <https://www.notion.so/2251adbda1df80e0801ae17863129ce2?pvs=21>

항목	TA-Prioritized	TA-Hybrid
❖ 계산 비용	비교적 낮음 (단순 구조)	더 높음 (subgroup, flow 최적화 필요)
❖ 사용처	로봇이 고정된 역할을 수행하는 곳 (e.g. 창고)	작업 교환이 필요한 유연한 환경 (e.g. 라스트마일 배송, 드론 관리 등)

알고리즘	핵심 목표	적합 환경
TA-Prioritized	빠른 경로 계획, 충돌 회피	고정 작업, 단순 경로, 창고형 물류
TA-Hybrid	경로 + 작업 최적화 (교환 포함)	복잡한 환경, 실시간 task 재배정이 필요한 곳

최종 요약

4.2.1.6.7 작업 배정과 경로 계획을 동시에 할 때

<https://goodahn.tistory.com/128> : Conflict-Based Search with Optimal Task Assignment

작업 배정과 경로 계획을 동시에 할 때(알고리즘 / Path Finding)

4.2.1.6.7.1 TA-Prioritized: MAPD(Multi-Agent and Delivery) 문제를 해결하기 위한 알고리즘, 작업 할당(Task Allocation)과 우선순위 기반 경로 계획(Prioritized Path Planning)을 결합 한 구조

- 로봇이 하나의 목표만 가는게 아니라,
- pickup지점에서 물건을 받아,
- delivery 지점까지 가져가야 한다.
- pickup → delivery쌍이 반복되며 task가 발생하는 것이 MAPD의 핵심이다.

4.2.1.6.7.2 ❖ TA-Prioritized 핵심 알고리즘 구성

Task Assignment (TA 단계)

- 여러 개의 task들을 현재 가능한 agent들에게 할당함
- 할당 기준:
 - agent가 task를 언제 시작할 수 있는지
 - 예상 작업 시간
- task를 할당받은 agent들마다 남은 작업 수행 시간 계산

예: $a1 \circ (p1 \rightarrow d1), (p2 \rightarrow d2)$ 수행 예정이면, 총 수행 시간이 38이라면 $a1$ 의 우선순위가 가장 높음

우선순위 계산

- 모든 agent에 대해 남은 작업 시간을 기준으로 우선순위 결정
- 순위가 높은 agent부터 먼저 경로를 짜
- 후순위 에이전트는 앞선 에이전트의 경로를 피하면서 경로를 계산해야 함

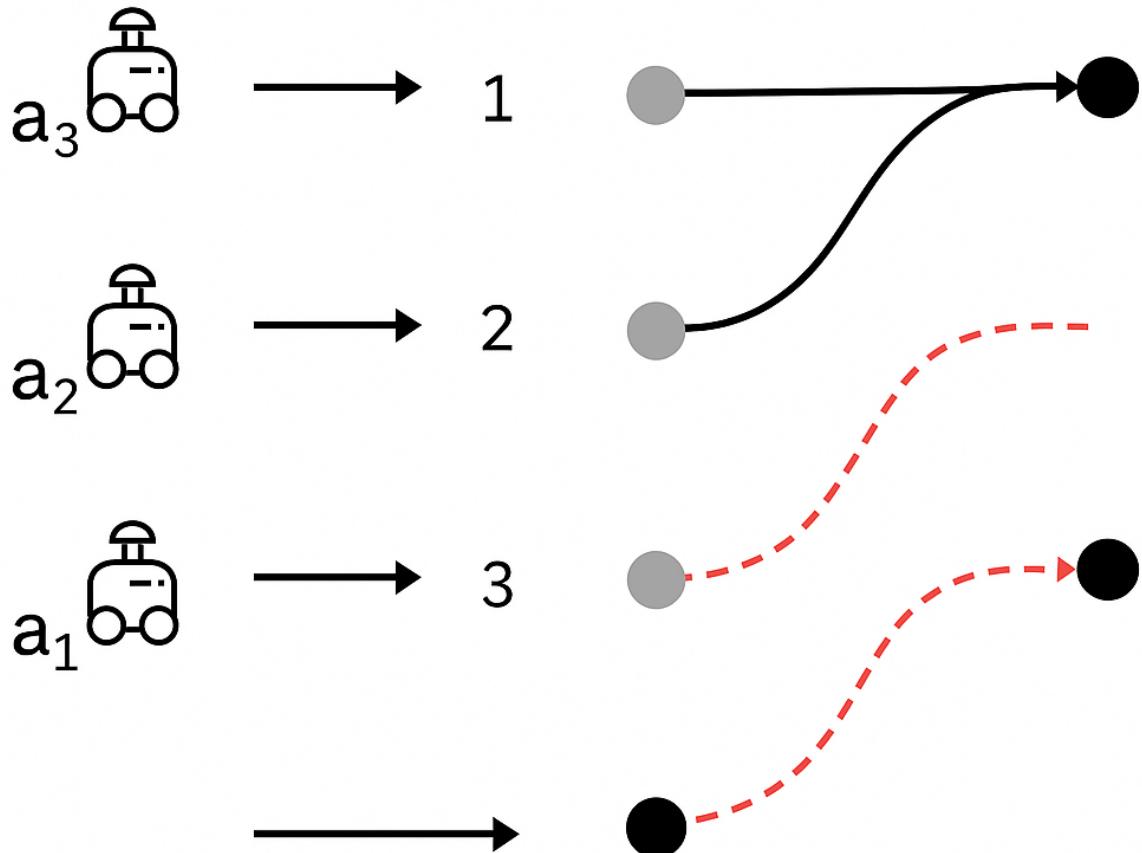
경로 계획 (Prioritized Planning 단계)

경로를 계산할 때는 pickup → delivery 두 부분으로 나누고,

각 부분은 A 탐색 알고리즘*으로 계획됨

- 각 agent에 대해:
 - 현재 위치 → pickup 지점 A*
 - pickup 지점 → delivery 지점 A*
- 이 때, 이미 경로가 고정된 다른 에이전트의 경로를 constraint로 포함

Prioritized Planning



reserving dummy path (핵심 기법)

이 알고리즘의 가장 중요한 기술이 바로 이거야.

경로를 짤 때, 앞선 로봇들이 거기에 그대로 멈춰 있다고 가정하면,

뒤따르는 로봇은 갈 수 있는 길이 없어져 경로 계획 실패(dealock)가 생길 수 있다.

그래서 해결책은:

“앞선 에이전트가 작업을 마친 후 자리를 비운다”고 가정하고 경로를 짠다.

즉:

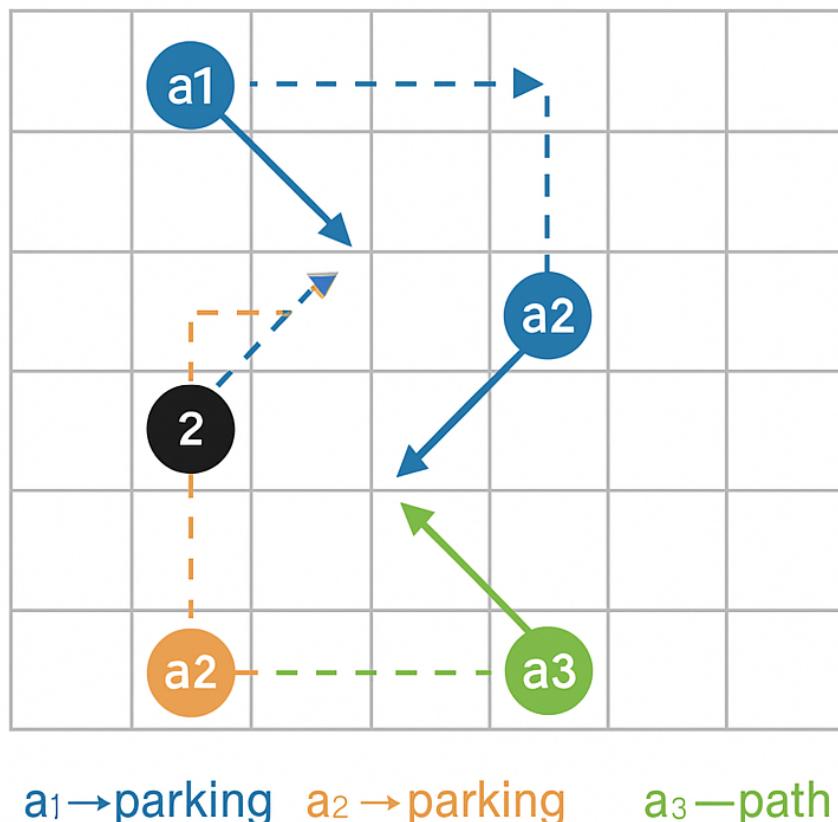
- 실제로 로봇은 움직이지 않아도,
- 경로 탐색 시에는 마치 작업이 끝나면 주차 위치(parking location)로 빠질 것처럼 dummy path를 예약 해두는 것처럼 처리함

이렇게 하면 뒤따르는 로봇도 막힘 없이 경로를 계획할 수 있다.

4.2.1.6.7.3 알고리즘 흐름 요약 (구조적으로)

1. 모든 agent의 남은 작업 시간 계산 → 우선순위 정렬
2. 순서대로 agent에 대해:
 - a. pickup 경로 A* 계산 (dummy path 포함)
 - b. delivery 경로 A* 계산 (dummy path 포함)
3. 각 agent는 이전 agent 경로를 constraint로 고려
4. 전체 경로가 충돌 없이 확보되면 실행

After completing its task, each agent is assumed to move to a parking location. This allows subsequent agents to plan a path without encountering a deadlock.



4.2.1.7 Multi-Agent Path Finding (MAPF) in ROS +Conflict-Based Search (CBS)

https://github.com/speedzjy/mapf_ros/tree/humble

https://github.com/speedzjy/ridgeback_mapf/tree/humble



ROS humble로 구현 되어 있음...

- Conflict-Based Search (CBS)
- Enhanced Conflict-Based Search (ECBS)

- Prioritized Planning using SIPP(**example code** for SIPP, the code to check swap has not been written yet)

4.2.1.7.1 Conflict-Based Search (CBS)

- 최적해 보장
- 두 단계 알고리즘으로 되어 있다.
- 높은 레벨의 서치는 constraint tree(CT)에 구현되어 있음
CT는 하나의 에이전트의 위치와 시간의 제약조건을 포함한 노드를 갖음
- 낮은 레벨에서는 A*를 사용해서 각각의 경로를 찾음

4.2.1.7.2 Enhanced Conflict-Based Search (ECBS)

- 다중 로봇 경로 탐색에서 근사 최적해를 제공 (최적해 되신 빠른 해를 제공)

4.2.1.7.3 Prioritized Planning using SIPP

- Safe Interval Path Planning(SIPP) : 단일 로봇 local planner, 비충돌 보장, 정적&동적 장애물 고려한 경로 계획 수행
- 우선순위가 있는 경로계획의 경우에 다른 로봇을 동적 장애물로 인식
- 먼저 계획을 세우는 에이전트의 궤적은 나중에 계획을 세우는 에이전트의 제약 조건에 붙은 동적 장애물 궤적으로 간주된다.

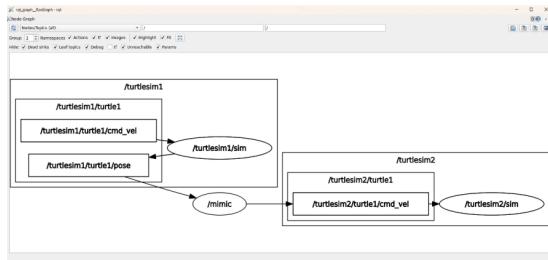
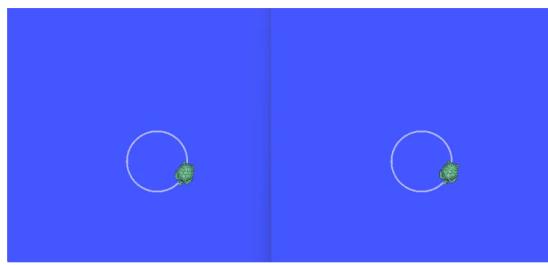
4.2.1.8 다중 로봇 경로계획

4.2.1.8.1 Stage 1

- 초기 위치 고정 후 각 로봇암 위치로 이동.
 - 짐 먼저 다 싣을 로봇 이동

필요 기능

- 맵 공유
- 로봇마다 네임스페이스 분류



- 관제 시스템 : 로봇 상태 표시 (대기(0)/ 수하물 장소 입구(1)/ 첫번째 로봇암(2)/ 두번째 로봇암(3))
- 전역 경로 알고리즘 A*
- 제어 DWB(파라미터 강화학습으로 최적화)

4.2.1.8.2 Stage 2

- 초기 위치에서 모두 동시에 출발하여 로봇암 위치로 이동(스케줄링 필요)

추가 필요 기능

- 회피주행 알고리즘(Local planner)
- 겹치는 경로 해결

4.2.1.8.3 Stage 3

- 사람 호출 받았을 때 사람의 위치로 이동 한 뒤 짐을 싣고 로봇암 위치로 이동(스케줄링 필요)

추가 필요 기능

4.2.1.8.4 Stage 4

- 호출을 두번 이상 받았을 때 각각의 위치로 이동하고 짐을 싣고 로봇암으로 이동(짐 두개)
 - 호출한 사람에게 가장 빨리 갈 수 있는 로봇이 이동.

추가 필요 기능

4.2.2 behavior tree(구조도)

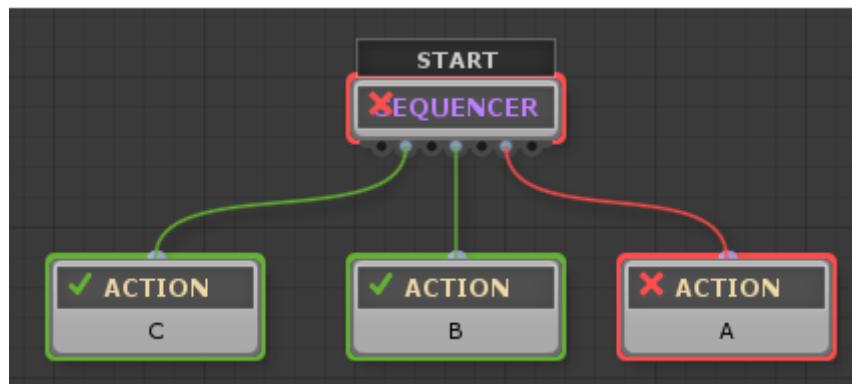


4.2.2.1 behavior tree: 행동을 결정하는 계획표/로직을 트리처럼 구조화/framework

4.2.2.1.1 즉, 노드끼리 유기적인 연결을 바탕으로 효과적인 내부 체계를 다질 수 있는 구조도!

4.2.2.1.2 우리가 최종적으로 구현해야하는 구조?

4.2.2.1.3 ros2의 action client node와 Behavior tree내의 action노드와의 연계



3 <https://lifeisforu.tistory.com/327>

behavior tree는 필요한 node간의 연결로 로봇의 내부 구조를 다지는 역할 그러나 여기서(BT)의 node는 ros2의 노드와 같은개념은 아님 ; 어떤것이 다른가?

Ros2 노드 : 특정작업 수행 및 다른 노드와의 통신 / BT노드: 로봇의 행동을 제어하는 큰 틀

4.2.2.1.4 최종 목표인 BT 에서의 action과 ros2의 action client를 연결하는 과정은 후에 다루겠음

Q. 왜 굳이 behavior tree를 만들/사용해야 하는가?

A. 물론 cpp나 python로 내부 함수에서 class, def함수를 사용해 main으로 마무한 후

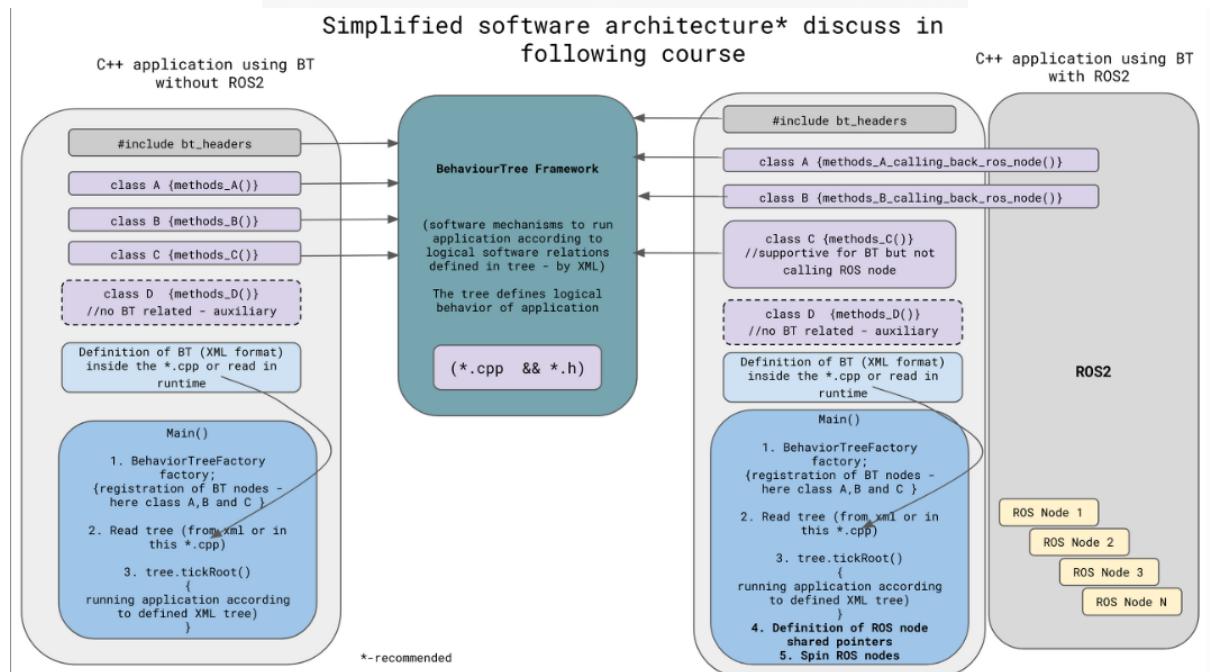
if와 else문으로 노드끼리 연결은 할 수 있으나, 구조의 복잡도가 높아지면 코드끼리 꼬일수 있고 유지보수에 어려움이 생기기에 Behavior tree를 사용하는편이 유리

게다가 행동 단위를 독립적으로 분리, 디버깅 가능, 노드간 재사용 가능하기도함

4.2.2.1.5 BT의 내부

TICK : 행동 하나씩 실행하는 신호 (위의 사진참고. action들 사이와 start와 연결된 줄을 tick이라는 조건으로 내부에서 조건을 주고받음) 임. 위에서 아래로 실행 신호를 보낸다고 생각하면됨 하나의 상위 노드내에서 조건을 만족시켜 loop를 돈다고 생각하면 편함 / 조건을 만족시키면 옆의 노드 실행 or 상위 노드로 이동해 최상위 노드로 까지 이동 , 조건 불만족시 이행될때까지 loop를 running 시킴

NODE : 트리의 한 조각 / 노드는 아래의 사진과 같이 4개로 이루어져있음



4.2.2.2 ros2와의 연결성

4.2.2.2.1 class A, B,C => 각각 하나의 BT노드 즉 기능을 수행하는 단위 ex. assignTask....

behaviorTreefactory 로 클래스를 등록시키고 xml로 트리를 읽어서 tick으로 실행

4.2.2.2.2 하지만 ros2가 있어야 센서데이터를 수신하고 명령을 퍼블리시하고 액션을 호출할수있음

어떻게?

클래스(비티노드)가 Ros2 노드와 상호작용해야됨

main 함수에서 Ros Node관리

```
rclcpp::init(argc, argv);
auto node = std::make_shared<rclcpp::Node>("bt_node");
rclcpp::spin(node);
```

=> BT실행과 ROS 콜백을 동시에 실행 : 센서 데이터나 메시지를 받는건 ROS2의 Callback이 담당하고 이 콜백을 작동시키려면 rclcpp::spin(node)를 호출해야됨

시각화

XML트리 → behaviorTreefactory에서 등록된 ros내장된 통신을 진행

→ 클래스들 ros2: publisher, subscriber, Actionclient등을 포함한 클래스 불러오기

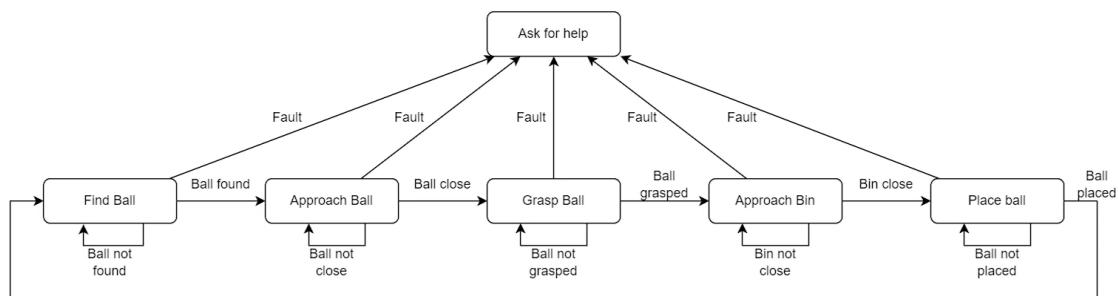
→ tickRoot()실행 → 트리가 tick 될때마다 Ros2노드가 실제 동작 수행

4.2.2.3 Behavior Tree 추가 자료

4.2.2.3.1 Behavior Tree

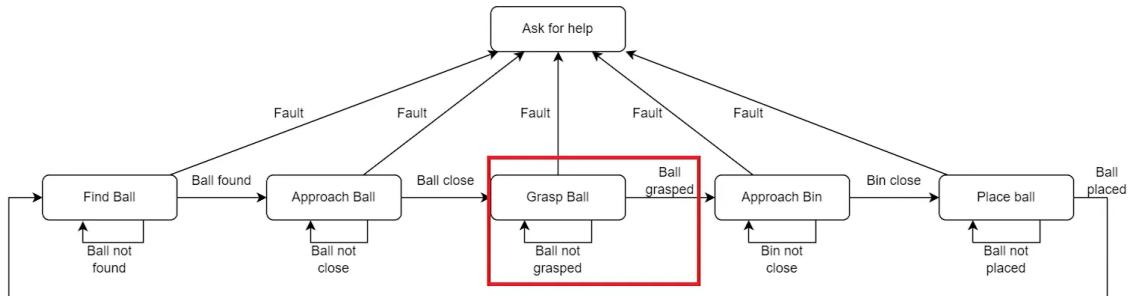
4.2.2.3.2 Finite state machine

- 순차적 방식으로 작동
- 상태를 입력으로 받음



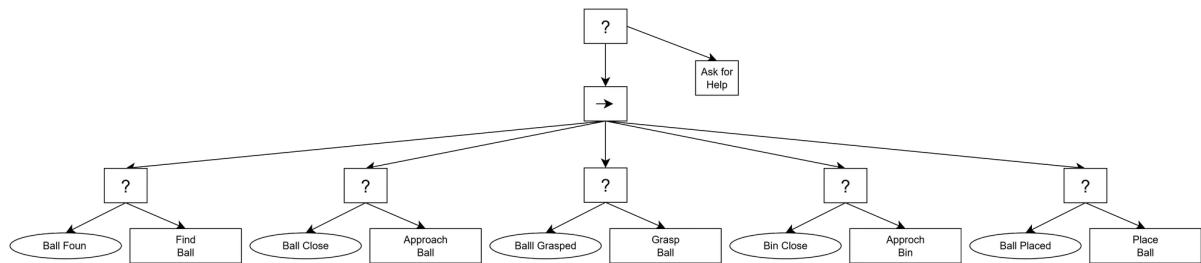
단점

- Reactivity (재사용성 떨어짐= 기능 추가가 힘듬; 복잡성이 증가했을 때 취약)
- Modularity (앞뒤의 내용과 연관되어서 모듈화 하기 힘듦)



*FSM을 쓰는 경우는 매우 단순한 기능을 수행할 때 유용함.

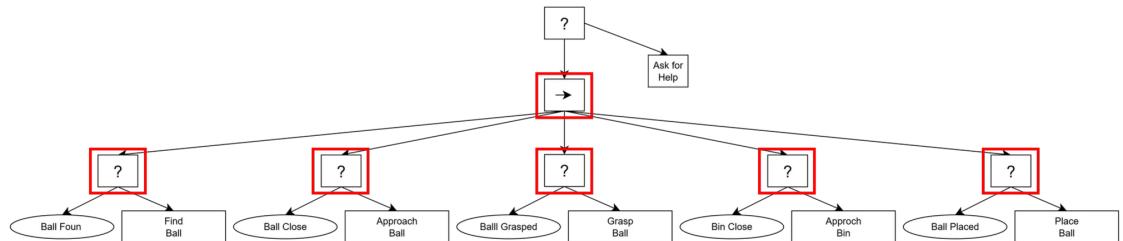
4.2.2.3.3 Behavior Tree



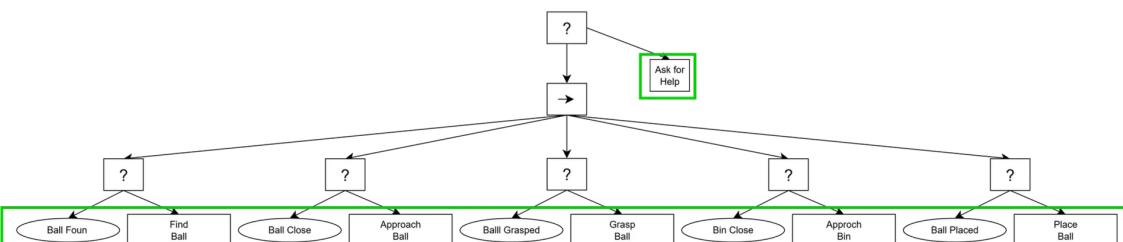
4.2.2.3.3.1 행동 트리 기본 구조

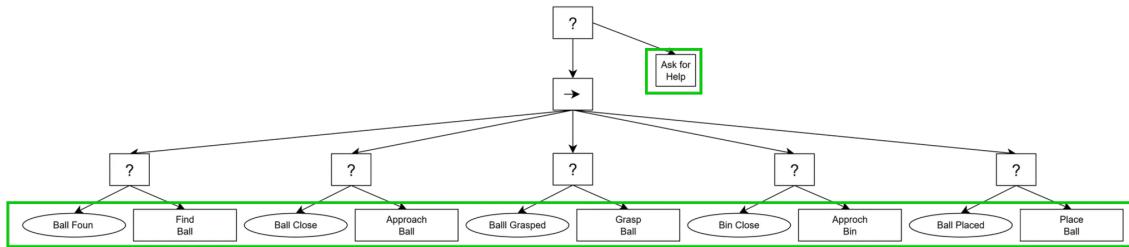
행동 트리는 방향성이 있는 루트 트리로:

- 내부 노드: 제어 흐름 노드(Control Flow Nodes)



- 잎 노드: 실행 노드(Execution Nodes)

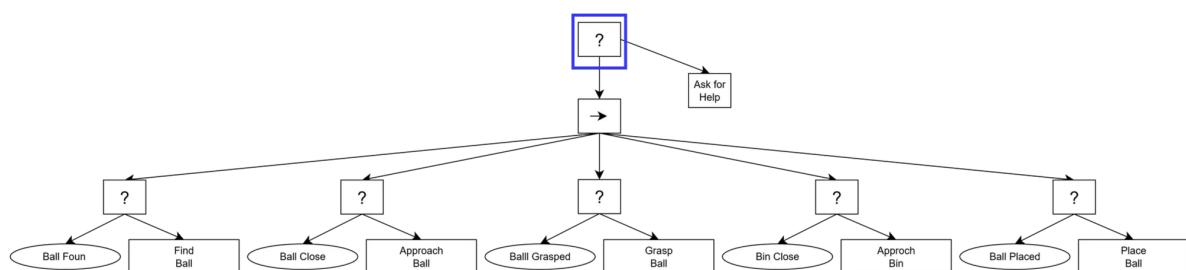




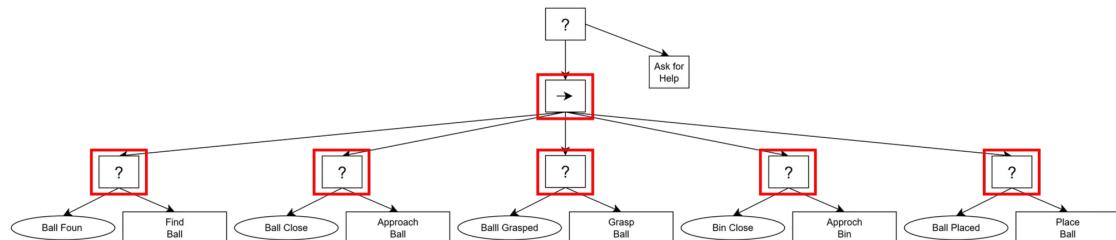
4.2.2.3.3.2 틱(Tick) 시스템

틱은 노드를 실행시키는 신호입니다:

- 루트 노드에서 시작하여 하위 노드들로 전파



- 노드가 활성 상태일 때만 실행됨
- 매 프레임마다 또는 특정 조건에서 발생
- 하위 내부 노드를 실행시키고 단계적으로 전파



4.2.2.3.3.3 실행 순서와 규칙

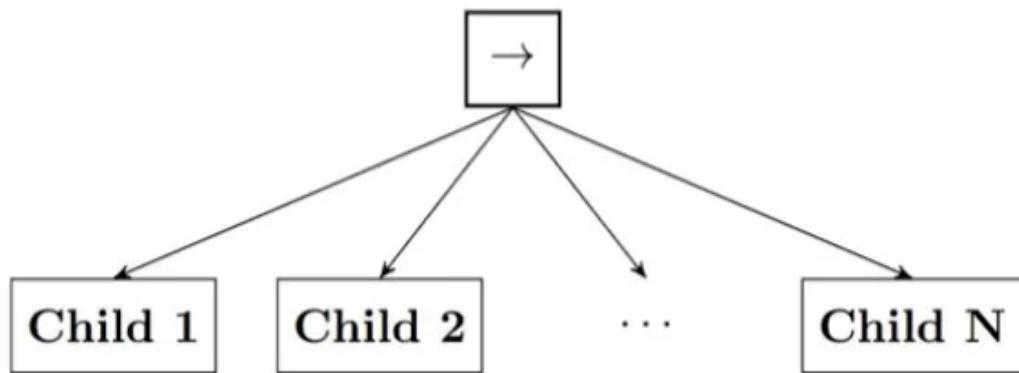
기본 실행 순서:

- 왼쪽에서 오른쪽으로 진행
- 부모 노드의 타입에 따라 자식 노드 실행 방식이 달라짐

4.2.2.3.3.4 주요 제어 노드 타입

1. Sequence (순서)

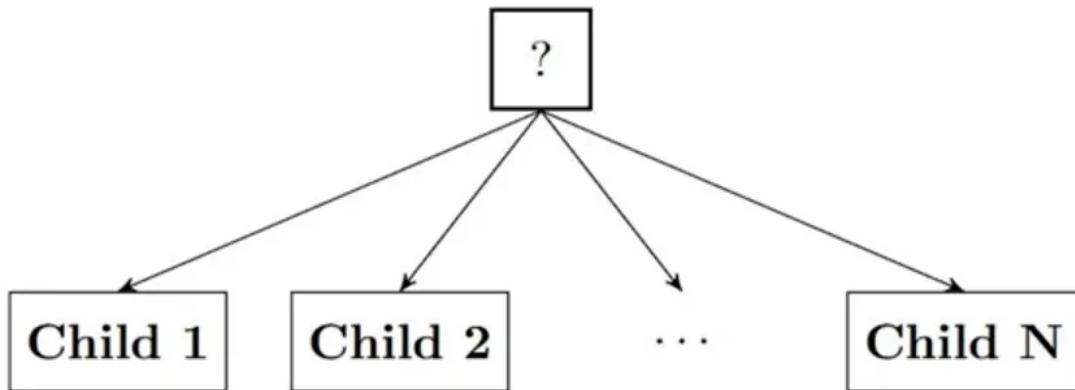
- 모든 자식을 왼쪽부터 순서대로 실행
- 하나라도 실패하면 전체 실패



Graphical representation of a Sequence node with N children.

2. Selector/Fallback (선택)

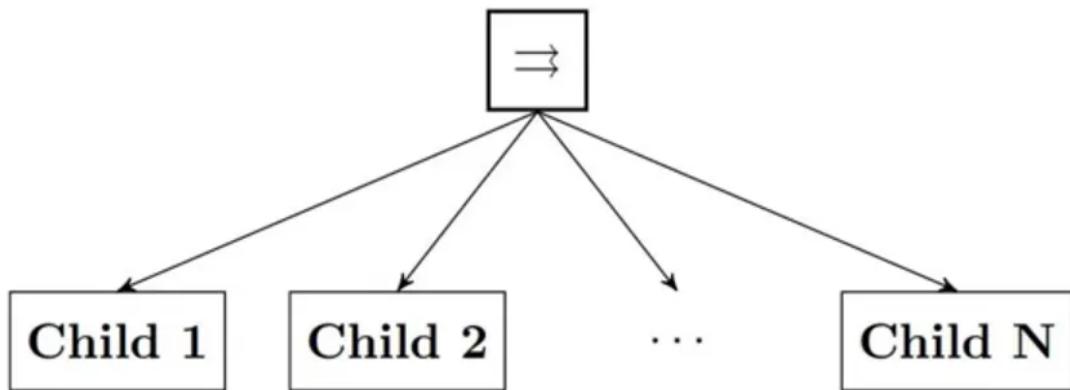
- 왼쪽부터 실행하되, 하나가 성공하면 전체 성공
- 모두 실패해야 전체 실패



Graphical representation of a Fallback node with N children.

3. Parallel (병렬)

- 모든 자식을 동시에 실행



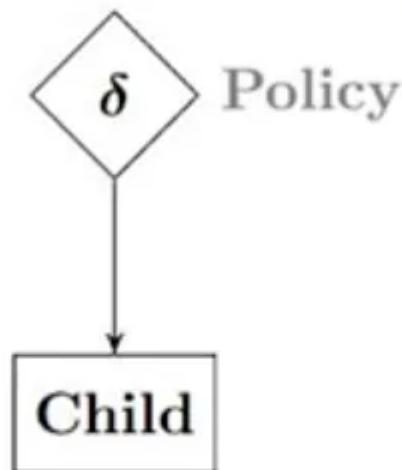
Graphical representation of a Parallel node with N children.

Decorator Node는 행동 트리에서 단일 자식 노드의 동작을 수정하거나 확장하는 특별한 제어 노드입니다.

4.2.2.3.3.5 Decorator의 기본 특징

구조:

- 하나의 자식 노드만 가짐
- 자식 노드의 결과를 받아서 변형하거나 조건을 추가
- 자식 노드를 감싸는 래퍼(Wrapper) 역할



Decorator node. The label describes the user defined policy.



4.2.2.3.3.6 1. Action Node (행동 노드)

기능:

- 실제 작업을 수행하는 노드
- 게임에서 캐릭터 움직임, 공격, 애니메이션 등
- 로봇에서 모터 제어, 센서 읽기 등

반환값:

- **SUCCESS**: 작업 완료
- **FAILURE**: 작업 실패
- **RUNNING**: 작업 진행 중 (여러 틱에 걸쳐 실행)

예시:

```
Action Examples:
├─ Move to Target
├─ Play Animation
├─ Fire Weapon
├─ Open Door
└─ Pick Up Item
└─ Send Message
```

구현 예:

```
cpp
class MoveToTarget : public ActionNode {
    Status execute() {
        if (reached_target()) {
            return SUCCESS;
        }
        if (path_blocked()) {
            return FAILURE;
        }
        move_towards_target();
        return RUNNING; // 계속 이동 중
    }
}
```

4.2.2.3.3.7 2. Condition Node (조건 노드)

기능:

- 조건을 확인하는 노드
- 환경 상태, 변수 값, 센서 데이터 등을 체크
- 즉시 실행되어 바로 결과 반환 (RUNNING 없음)

반환값:

- **SUCCESS:** 조건 만족
- **FAILURE:** 조건 불만족
- **RUNNING:** 일반적으로 사용하지 않음

예시:

```
Condition Examples:
├─ Is Enemy Visible?
├─ Health > 50%?
├─ Is Door Open?
├─ Has Item?
├─ Is Target in Range?
└─ Battery Level OK?
```

구현 예:

```
cpp
class IsEnemyVisible : public ConditionNode {
    Status execute() {
        if (can_see_enemy()) {
            return SUCCESS;
        } else {
            return FAILURE;
        }
    }
}
```

4.2.2.3.3.8 실제 사용 패턴

전형적인 조합:

```
Sequence
├─ Is Enemy in Range? (Condition)
├─ Face Enemy (Action)
├─ Has Ammo? (Condition)
└─ Fire Weapon (Action)
```

Guard 패턴:

```
Fallback
├─ Sequence
|   ├─ Health Low? (Condition)
|   └─ Find Health Pack (Action)
└─ Continue Fighting (Action)
```

주요 차이점:

- **Action:** 세계를 변화시킴 (부작용 있음)
- **Condition:** 세계 상태만 확인 (부작용 없음)

이 두 타입의 노드가 조합되어 복잡한 AI 행동을 만들어냅니다!

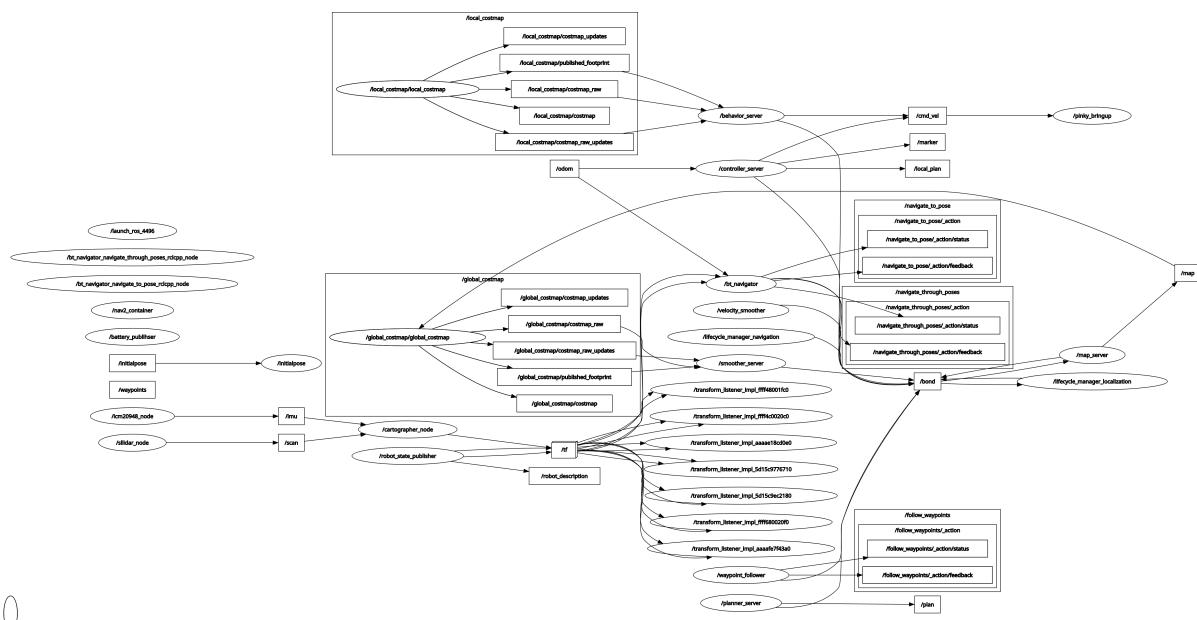
4.3 AMR

4.3.1 PID 개념 및 코드 정리

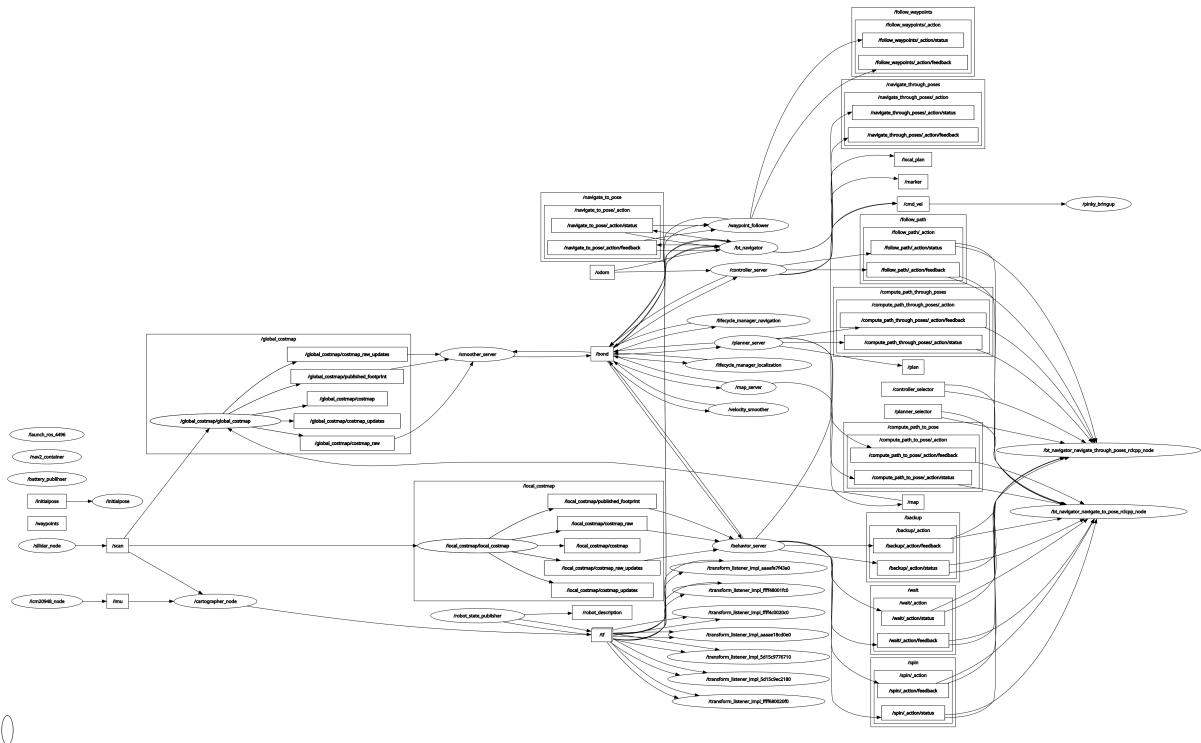
<https://www.notion.so/ROS2-2331adbda1df801db0d1de2b7c70b788>

실전 클릭 => 4강 클릭

4.3.2 pinky nav2 공부



○



4.3.2.1 Nav2 controllers

	DWB	MPPI	Regulated Pure Pursuit	(Rotation Shim Controller)
중심 개념	가능한 속도 조합들을 모두 평가	가능한 속도 조합들을 모두 평가 (DWB보다 범위가 더 넓음)	경로의 한 점 (lookahead) 따라감	main planner 보조 해서 제자리 회전
계산 비용	low	very high	low	low
곡선 처리	각진 경우도 있음	각진 경우도 있음	부드러움	
제자리 회전	o	x	x	o

- global planner
 - 시작 위치 → 목표 위치까지 경로(global path) 생성

- 현재 로봇이 향하고 있는 방향과 상관없이 현재 위치에서 목표 지점까지 가는 경로만을 생성함
- 초기 방향을 맞추는 것은 local planner의 역할
- static 장애물만 고려하고 dynamic 장애물은 고려하지 않고 경로 생성
-
- local planner:
 - global path를 따라가기 위한 속도 명령을 실시간으로 계산
 - global path를 가까이 유지하려고 하지만 주변에 장애물이 있다면 잠시 우회해서 회피하고 원래 전역 경로를 재접근하려 시도
- planner server:
 - 전역 경로 생성 > 알고리즘 플러그인을 로드 > 경로 생성
- controller server:
 - 전역 경로 기반으로 실시간 장애물을 고려한 로컬 경로 생성

<https://github.com/ros-navigation/navigation2/tree/jazzy>

4.3.2.1.1 1. DWB(Dynamic Window Approach) Controller

[동자 원리]

(1) 로봇의 현재 속도, 가속도 제한 등을 기준으로 당장 몇 초 내에 낼 수 있는 속도(vx, vy, w) 조합을 탐색

현재 상태와 제한을 고려하여 샘플링된 속도 조합들: Dynamic Window (동적 창)

```
vx_samples: 20
vy_samples: 5
vtheta_samples: 20
```

ex) $vx \in [0, 0.26], vtheta \in [-1.0, 1.0]$ 이면

- vx는 20단계 → 0.0, 0.013, 0.026, ..., 0.26
- vtheta는 20단계 → -1.0, -0.894, ..., 1.0

위 방식으로 모든 속도 조합을 만들고 아래 조건을 동시에 만족하는 영역만 탐색

=> DWA(Dynamic Window Approach)

(1) kinematic constraint: max_vel_x, max_vel_theta 같은 기계적 한계

(2) dynamic constraint: 가속도 제한에 따라 지금 순간에서 가능한 속도 변화

(3) collision-free constraint: 장애물에 부딪히지 않는 속도 조합만 남김

(2) 각 속도 조합들에 대해:

- 일정 시간 동안 시뮬레이션으로 로봇의 궤적 예측
- 장애물과 충돌 여부 확인
- 여러 critics들로 비용 평가

가장 cost가 낮은 속도(가장 좋은 궤적)를 선택해서 cmd_vel publish

- costmap 이용 > 각 trajectory마다 costmap 충돌 여부 평가

Costmap: 2D 격자 지도에 위험도(cost)를 저장한 지도

장점:

- 계산량이 적어 작은 컴퓨팅 파워에도 실시간 성 확보에 유리

실시간 성 확보는 로봇이 주행할 때 동적으로 변화하는 주변 환경에 대응하도록 해줘 주행 성능에 중요하다

- 제자리 회전 가능

단점: 당장 도달 가능한 속도들은 아래 제약들로 인해 제약이 생겨 전체 공간을 충분히 탐색하지 못한다

- 속도 샘플링 개수 제한

- 시뮬레이션 시간 제한
- 속도 제한 범위 설정

그로인해 local minimum이 빠질 가능성 있다?????

local minimum: 전체적으로 최적이 아니지만, 현재 선택할 수 있는 후보 중에서의 최적의 선택지

ex)

만약 로봇 앞에 장애물이 있다면

- $vx>0$ 후보 대부분이 충돌로 제거됨
- 실제로 DWA의 목표는 “목표를 향해서 가면서 충돌하지 않는 것”이지 “경로선을 따라가는 것”은 부가적인 요소다 → 충돌 회피 후 경로선에서 벗어난 운항을 할 수 있다
- Trajectory Generator Plugin
 - (1) StandardTrajectoryGenerator: 기본 궤적 생성 방식. 현재 속도를 기준으로 샘플링
 - (2) LimitedAccelGenerator: 가속도 제한을 고려한 궤적 생성기
- Trajectory Critics: 장애물 회피

각각 궤적에 대해 cost를 계산해 최적 궤적을 선택하는 데 사용

- (1) BaseObstacleCritic: 장애물과의 충돌 여부 평가
- (2) GoalAlignCritic: 궤적이 목표를 향하고 있는 정도 평가
- (3) PathAlignCritic: Global Path와 궤적의 정렬 정도
- (4) GoalDistCritic: 목표 지점과 얼마나 가까운지
- (5) PathDistCritic: Path와 얼마나 떨어졌는지
- (6) RotateToGoalCritic: 회전이 필요한 경우 판단
- (7) TwirlingCritic: 제자리 회전 같은 불필요한 동작 억제
- (8) OscillationCritic: 진동, 왕복 움직임 억제
- (9) PreferForwardCritic: 후진보다는 전진을 선호하도록 유도

4.3.2.1.2 2. Graceful Controller

[동작 원리]

1. 전역 경로에서 일정 거리 떨어진 “목표 지점”을 설정하고 이 지점을 따라가면 부드러운 경로 생성
2. Lyapunov 기반 제어 법칙 기반으로 부드럽게 속도/방향 제어

Lyapunov 기반 제어 법칙

(r, φ, δ) 라는 극좌표 기반 오차로 제어함:

- r : 현재와 목표 지점 간 거리
- φ : 로봇이 바라보는 방향과 목표점 사이의 각도
- δ : 로봇의 회전 방향과 φ 간의 차이 (정렬 상태)

r 이 작아질수록 목표에 가까워짐,

φ, δ 가 0에 가까워질수록 목표 방향과 정렬됨

- k_{phi} : 방향 조절 속도 제어
- k_{delta} : 정렬 속도 제어
- beta : 곡률이 클 때 선속도 감소 정도
- λ : 거리 멀수록 선속도 증가(감속 보정 계수)

$$\begin{aligned} v &= \lambda * r * \cos(\delta) \\ \omega &= k_{\text{phi}} * \varphi + k_{\text{delta}} * \delta \end{aligned}$$

- v : 선속도(linear)
- ω : 각속도(angular)

- 목표점에서 멀 때는 빠르게 접근 ($v \uparrow$)
- 가까워질 수록 속도 감소
- 방향 틀어졌을 때 회전 중심으로 움직여서 정렬

장점:

- 부드럽고 자연스러운 궤적 생성(진동 최소화)

부르러운 제어란?

단점:

- 복잡한 설정 필요(k_{ϕ} , β 등 튜닝 어려움)
- 직접적인 장애물 회피 기능은 포함하지 않음 > dwb처럼 경로를 costmap에 쏘보고 평가하는 방식x
- 목표만 보고 제어를 생성하기 때문에 전역 경로 상에 장애물이 있다면 회피하지 않고 속도 감소 or 정지
 - 장애물 근처 목표 재설정을 하지 않는다

** 부드럽고 매끄러운 움직임에 특화된 컨트롤러 > 위험 회피, 동적 장애물 대응에는 약한 설계

4.3.2.1.3 3. MPPI(Model Predictive Path Integral) Controller

[동작 원리]

- 이전 제어 결과를 기반으로
- Gaussian 분포에서 여러 제어 perturbation(속도 약간 변화된 궤적)을 샘플링
- 각 궤적을 **forward simulate**하여 예측
- **plugin-based critic**으로 개별 궤적 평가 (장애물 회피, 경로 추종, 부드러움 등)
- 각 궤적의 cost를 softmax 방식으로 조합해 최적 제어 추정
- 반복(iteration)하며 점진적으로 해법 수렴 → 다음 제어 입력 결정

1. 이전 제어 입력의 Gaussian 분포에서 제어값을 무작위 샘플링

=> 현재 속도의 Gaussian 분포를 기준으로 랜덤하게 속도 조합을 만든다

ex) 현재 상태(위치, 속도)를 기준으로

현재: $vx=0.2, wz=0.0$

→ Sample 1: $\Delta vx = -0.05, \Delta wz = 0.1 \rightarrow new: vx=0.15, wz=0.1$

2. 제어 입력을 적용하여 n초 동안의 궤적을 시뮬레이션하여 궤적을 만든다
3. 예측 궤적을 costmap에 대입해서 충돌 여부 판단
4. 충돌하지 않는 궤적 중에서 critics 기반으로 cost 부여
5. 각 궤적의 cost를 softmax 방식로 확률처럼 변환

Softmax란?

: 값들을 확률처럼 만들어주는 함수

$$\text{Softmax}(x_i) = \frac{e^{-x_i/\lambda}}{\sum_j e^{-x_j/\lambda}}$$

5. 모든 샘플 궤적을 weighted average 해서 최종 제어 명령 계산 (비용 적은 것에 더 많은 weight 부여)

→ average하는 이유: 최소 cost 하나만 고르면 → 작은 오차나 노이즈에도 선택이 바뀜

1. 제어 주기마다(예: 20Hz → 0.05초마다 한 번) 제어 명령 생성

장점:

- 무작위 궤적을 수천 개 생성 → dwb와 다르게 궤적의 수가 몇 백 개로 제한되지 않는다
- 예측 기반 제어(Predictive MPC) → 향후 궤적까지 고려

단점:

- 계산 복잡도 매우 높음(대규모 샘플 사용) → 메모리 사용 증가
- 튜닝 어려움

4.3.2.1.4 4. Regulated Pure Pursuit

- lookahead point 따라가는 방식
- navigation2의 RegulatedPurePursuitController은 속도, 방향 둘 다 계산

장점:

- 튜닝 난이도/계산 비용 낮음

단점:

- 경로 이탈 시 취약, 시작 지점에서 제자리 회전 불가

4.3.2.1.5 5. Rotation Shim Controller

main local planner을 보조해서 제자리 회전을 전담

4.3.2.2 Costmap

[인플레이션(Inflation)]

로봇의 코스트맵에서 장애물 주변의 코스트 값을 확장하여, 로봇이 장애물에 더 가까이 접근하는 것을 방지하는 과정을 의미합니다. 이는 로봇이 안전하게 이동할 수 있도록 경로 계획 및 충돌 회피에 중요한 역할을 합니다.

ROS (Robot Operating System)에서, 특히 코스트맵(costmap)을 사용하는 내비게이션 시스템에서 인플레이션은 중요한 개념입니다. 코스트맵은 로봇 주변의 환경을 나타내는 그리드 기반의 맵으로, 각 셀은 특정 위치의 장애물 존재 여부와 정도를 나타내는 코스트 값을 가집니다.

[cost map 설명]

<https://velog.io/@cherry0319/Nav2-Nav2-%EA%B5%AC%EC%84%B1-%EC%9A%94%EC%86%8C#:~:text=Costmap%20Layer%20costmap%EC%9D%80%20%EC%97%AC%EB%9F%AC%20%EA%B0%9C%EC%9D%98%20layer%EB%A5%BC%20%EC%A1%B0%ED%95%A9%ED%95%98%EC%97%AC,%EA%B2%B0%ED%95%A9%EB%90%98%EB%A9%B4%20%EB%A1%9C%EB%B4%87%EC%9D%B4%20%E>

C%9D%B4%EB%8F%99%ED%95%A0%20%EC%88%98%20%EC%9E%88%EB%8A%94%20%EB%B9%84%
 EC%9A%A9%EC%A7%80%EB%8F%84(Costmap)
 %EA%B0%80%20%EC%99%84%EC%84%B1%EB%90%9C%EB%8B%A4.

- **global costmap**은 static map으로부터 만들어집니다
 - static_layer : /map 의 OccupancyGrid 를 costmap으로 변환
 - obstacle_layer : 센서 /scan 등을 이용한 동적 장애물 처리
 - inflation_layer : 장애물 주변에 **buffer (cost spread)** 를 주는 레이어
- **local costmap**은 센서 값으로부터 만들어집니다
 - voxel_layer : 3D 센서 (예: depth camera, 3D LiDAR) 데이터를 사용해 **2D 평면에 장애물로 투영**
 - obstacle_layer 의 상위 버전이라 할 수 있음
 - raytracing 도 가능 (장애물 사라짐 감지)
 - inflation_layer : 마찬가지로 장애물 주변 buffer 처리

4.3.3 Nav2 패키지

4.3.3.1 Pinky_Navigation

bringup_launch.xml

```
<?xml version='1.0'?>
<launch>
  <set_env name="RCUTILS_LOGGING_BUFFERED_STREAM" value="1"/>

  <arg name="namespace" default="" />
  <arg name="map" default="$(find-pkg-share pinky_navigation)/map/pinklab.yaml"/>
  <arg name="use_sim_time" default="False"/>
  <arg name="params_file" default="$(find-pkg-share pinky_navigation)/params/nav2_params.yaml"/>
  <arg name="params_cartographer" default="nav2_cartographer_params.lua"/>
  <arg name="autostart" default="True"/>
  <arg name="container_name" default="nav2_container"/>
  <arg name="use_composition" default="True"/>
  <arg name="use_respawn" default="False"/>
  <arg name="log_level" default="info"/>
  <arg name="lifecycle_nodes_local" default="['map_server']"/>
  <arg name="lifecycle_nodes_nav" default="['controller_server',
    'smoother_server',
    'planner_server',
    'behavior_server',
    'bt_navigator',
    'waypoint_follower',
```

```

'velocity_smoothener']"/>

<group>
    <push-ros-namespace namespace="$(var namespace)"/>
    <node pkg="rclcpp_components" exec="component_container_isolated" name='$(var
container_name)' output="screen" args="--ros-args --log-level $(var log_level)">
        <param from="$(var params_file)"/>
        <param name='map' value="$(var map)"/>
        <param name='autostart' value="$(var autostart)"/>
        <param name='use_sim_time' value="$(var use_sim_time)"/>

        <remap from="/tf" to="tf"/>
        <remap from="/tf_static" to="tf_static"/>
    </node>

    <include file="$(find-pkg-share pinky_cartographer)/launch/
cartographer.launch.xml">
        <arg name="configuration_basename" value="$(var params_cartographer)"/>
        <arg name="use_nav2" value="True"/>
    </include>

    <include file="$(find-pkg-share pinky_navigation)/launch/
localization_launch.xml">
        <arg name="params_file" value="$(var params_file)"/>
        <arg name="map" value="$(var map)"/>
        <arg name="use_sim_time" value="$(var use_sim_time)"/>
        <arg name="autostart" value="$(var autostart)"/>
        <arg name="use_composition" value="$(var use_composition)"/>
        <arg name="use_respawn" value="$(var use_respawn)"/>
        <arg name="container_name" value="$(var container_name)"/>
        <arg name='lifecycle_nodes' value="$(var lifecycle_nodes_local)"/>
    </include>

    <include file="$(find-pkg-share pinky_navigation)/launch/
navigation_launch.xml">
        <arg name="params_file" value="$(var params_file)"/>
        <arg name="use_sim_time" value="$(var use_sim_time)"/>
        <arg name="autostart" value="$(var autostart)"/>
        <arg name="use_composition" value="$(var use_composition)"/>
        <arg name="use_respawn" value="$(var use_respawn)"/>
        <arg name="container_name" value="$(var container_name)"/>
        <arg name='lifecycle_nodes' value="$(var lifecycle_nodes_nav)"/>
    </include>
</group>

</launch>

```

- Cartographer 실행
- Localization 실행
- Navigation 실행

AMCL 적용

4.3.4 Cartographer 빌드

https://github.com/ros2/cartographer_ros

cartographer를 pinky home에 만든다.

```
mkdir -p cartographer_ws/src
cd cartographer_ws/src

git clone https://github.com/ros2/cartographer_ros.git

# 1) 소스 디렉토리로 이동
cd ~/cartographer_ws/src/cartographer_ros

# 2) 'ros2' 브랜치 체크아웃
git checkout ros2

# 3) 워크스페이스 최상위로 이동
cd ~/cartographer_ws

# 4) 의존성 설치
rosdep update
rosdep install --from-paths src --ignore-src -r -y

# 5) 빌드
colcon build

# 6) 설치된 환경 로드
source install/setup.bash
```

빌드 로그를 보면 `cartographer_rviz` 패키지에서 Boost iostreams 라이브러리를 못 찾았다는 오류가 발생했습니다:

```
The link interface of target "cartographer" contains:
  Boost::iostreams
but the target was not found.
```

이는 `libboost-iostreams-dev` 개발 패키지가 설치되어 있지 않기 때문에 생기는 문제입니다. 아래처럼 설치하고 다시 빌드해 보세요.

```
# 1. Boost iostreams 개발 패키지 설치
```

```

sudo apt update
sudo apt install libboost-iostreams-dev

# 2. (선택) 다른 누락 의존성도 rosdep으로 한 번에 설치
cd ~/cartographer_ws
rosdep install --from-paths src --ignore-src -r -y

# 3. 워크스페이스 루트에서 재빌드
cd ~/cartographer_ws

colcon build --allow- overriding cartographer_ros

```

오류2

```

# 1) Boost iostreams 개발 헤더 설치
sudo apt update
sudo apt install libboost-iostreams-dev

# 2) (혹시 누락된 의존성이 더 있을 경우 대비)
cd ~/cartographer_ws
rosdep install --from-paths src --ignore-src -r -y

# 3) 워크스페이스 루트에서 재빌드
cd ~/cartographer_ws

# /opt/ros/jazzy 밑에 설치된 cartographer_ros 대신 소스 쪽 버전을 반드시 우선 적용하고 싶다면:
colcon build --allow- overriding cartographer_ros

```

오류3 해결

```

sudo apt update
sudo apt install nano
nano ~/cartographer_ws/src/cartographer_ros/cartographer_rviz/CMakeLists.txt

```

4.3.4.1 편집 방법

- 화살표 키 $\uparrow\downarrow\leftarrow\rightarrow$ 로 커서를 이동합니다.
- `find_package(cartographer REQUIRED)` 라인을 찾으세요.
- 그 바로 위나 아래 빈 줄에 다음을 추가하세요:

```
find_package(Boost REQUIRED COMPONENTS iostreams)
```

4.3.4.2 저장 & 종료

- 저장: `Ctrl + S` → `Enter`
- 종료: `Ctrl + X`

4.3.4.3 다시 빌드

```
cd ~/cartographer_ws
colcon build --allow-override cartographer_ros
source install/setup.bash
```

4.3.5 pinky_gazebo

4.3.5.1 오류 해결

`~/pinky_violet/src/pinky_violet/pinky_description/urdf/imu.xacro`

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" >
  <gazebo reference="imu_link">
    <sensor name="tb3_imu" type="imu">
      <always_on>true</always_on>
      <update_rate>200</update_rate>
      <topic>imu</topic>
      <gz_frame_id>imu_link</gz_frame_id>
      <imu>
        <angular_velocity>
          <x>
            <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>2e-4</stddev>
            </noise>
          </x>
          <y>
            <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>2e-4</stddev>
            </noise>
          </y>
          <z>
            <noise type="gaussian">
              <mean>0.0</mean>
              <stddev>2e-4</stddev>
            </noise>
          </z>
        </angular_velocity>
        <linear_acceleration>
          <x>
            <noise type="gaussian">
```

```

<mean>0.0</mean>
<stddev>1.7e-2</stddev>
</noise>
</x>
<y>
<noise type="gaussian">
<mean>0.0</mean>
<stddev>1.7e-2</stddev>
</noise>
</y>
<z>
<noise type="gaussian">
<mean>0.0</mean>
<stddev>1.7e-2</stddev>
</noise>
</z>
</linear_acceleration>
</imu>
</sensor>
</gazebo>
</robot>

```

imu.xacro에서 <gz_frame_id>imu_link</gz_frame_id> 를 추가

4.3.5.2 SLAM 실행

```

ros2 launch pinky_gazebo launch_sim.launch.xml

ros2 launch pinky_cartographer cartographer.launch.xml use_sim_time:=True

ros2 launch pinky_cartographer map_view.launch.xml

ros2 run teleop_twist_keyboard teleop_twist_keyboard

ros2 run nav2_map_server map_saver_cli -f gazebo_map

```

4.3.5.3 Naviagtion 실행

```

ros2 launch pinky_gazebo launch_sim.launch.xml

ros2 launch pinky_navigation initialpose_bringup_launch.xml map:=gazebo_map.yaml
use_sim_time:=True

```

```
ros2 launch pinky_navigation nav2_view.launch.xml
```

4.3.5.4 Domain bridge을 이용해서 goal point 입력

4.4 ROBOT ARM

4.4.1 카메라로부터 위치와 자세 추출:

만약 카메라를 사용하여 로봇의 자세나 위치를 추정하려면, **카메라 캘리브레이션**과 **3D 복원** 기법을 사용할 수 있습니다. OpenCV를 사용하여 카메라에서 **3D 좌표**를 추출하거나, 카메라의 자세를 추정할 수 있습니다.

4.4.1.1 예시: 카메라에서 3D 좌표 추출 (OpenCV 활용)

```
import cv2
import numpy as np

# 카메라의 내부 파라미터 (예시, 실제로는 캘리브레이션 데이터를 사용해야 함)
camera_matrix = np.array([[800, 0, 640], [0, 800, 360], [0, 0, 1]])

# 왜곡 계수 (예시, 실제로는 캘리브레이션 데이터 사용)
dist_coeffs = np.array([0, 0, 0, 0])

# 3D 세계 좌표 (예시)
object_points = np.array([[0, 0, 0], [1, 0, 0], [0, 1, 0]], dtype=np.float32)

# 2D 이미지 점 (예시)
image_points = np.array([[640, 360], [700, 360], [640, 400]], dtype=np.float32)

# PnP 문제 해결 (Perspective-n-Point)
success, rotation_vector, translation_vector = cv2.solvePnP(object_points,
    image_points, camera_matrix, dist_coeffs)

# 회전 행렬로 변환
rotation_matrix, _ = cv2.Rodrigues(rotation_vector)

# 회전 행렬에서 Euler 각도 (roll, pitch, yaw) 추출
from scipy.spatial.transform import Rotation as R
r = R.from_matrix(rotation_matrix)
roll, pitch, yaw = r.as_euler('xyz', degrees=True)

# 결과 출력
print(f"Position: x={translation_vector[0]}, y={translation_vector[1]}, z={translation_vector[2]}")
print(f"Orientation: roll={roll}, pitch={pitch}, yaw={yaw}")
```

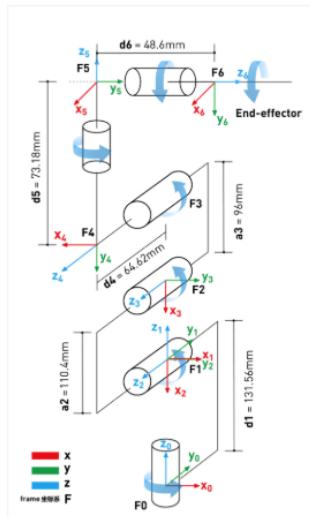
4.4.1.2 결론:

- **3D 위치(x, y, z)**와 **자세(roll, pitch, yaw)**는 주로 로봇 제어 시스템 또는 SLAM 기술을 통해 추출됩니다.
- 카메라로부터 **3D 좌표와 자세**를 추출하려면 OpenCV와 같은 라이브러리나 SLAM 시스템을 활용해야 합니다.
- ROS 2와 MoveIt2**를 사용하면, 로봇의 위치와 자세를 쉽게 추적하고 추출할 수 있습니다.

4.4.2 DH Parameter & 변환행렬 // OpenCV로 QR인식

4.4.2.1 코드 보실 분들은 핑크색 하이라이트만 보세용_ 나머지는 과정 or 부분 코드

3 DH Parameters



SDH :

mycobot280:: 6 axis, RRRRRR, stdDH, slowRNE					
j	theta	d	a	alpha	offset
1	q1	131.22	0	1.5708	0
2	q2	0	-110.4	0	-1.5708
3	q3	0	-96	0	0
4	q4	63.4	0	1.5708	-1.5708
5	q5	75.05	0	-1.5708	1.5708
6	q6	45.6	0	0	0

```
mycobot280:: 6 axis, RRRRRR, stdDH, slowRNE+-----+
| j | theta | d | a | alpha | offset |
+---+-----+---+---+-----+
| 1 | q1 | 131.22 | 0 | 1.5708 | 0 |
| 2 | q2 | 0 | -110.4 | 0 | -1.5708 |
| 3 | q3 | 0 | -96 | 0 | 0 |
| 4 | q4 | 63.4 | 0 | 1.5708 | -1.5708 |
| 5 | q5 | 75.05 | 0 | -1.5708 | 1.5708 |
| 6 | q6 | 45.6 | 0 | 0 | 0 |
```

	6		q6		45.6		0		0		0		
+	-	+	-	+	-	+	-	+	-	+	-	+	-

```
# θ: 실시간으로 얻음
angles_deg = mc.get_angles()
angles_rad = np.radians(angles_deg)

# d, a, α: 미리 정의된 값 사용 (예시는 가상의 값)
DH_table = [
    [angles_rad[0], 131.22, 0.0, np.pi/2],
    [angles_rad[1], 0.0, -110.4, 0.0],
    [angles_rad[2], 0.0, -96, 0.0],
    [angles_rad[3], 63.4, 0.0, np.pi/2],
    [angles_rad[4], 75.05, 0.0, -np.pi/2],
    [angles_rad[5], 45.6, 0.0, 0.0],
]
```

```
##경유지 설정해서 움직이는 코드 (좌표 2개 설정)

from pymycobot.mycobot import MyCobot
import time

# JetCobot 초기 설정
mc = MyCobot('/dev/ttyJETCOBOT', 1000000)
mc.power_on()
mc.thread_lock = True
print("✓ JetCobot이 연결되었습니다.")

# 초기 위치 정의 (그리퍼 open 포함)
initial_angles = [0, 0, 0, 0, 0, 40]

def move_home():
    mc.send_angles(initial_angles, speed=50)
    time.sleep(2)

# 초기 위치로 이동
print("🏠 초기 위치로 이동")
move_home()

# 좌표1 수동 설정
print("\n🛠 좌표1 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("❗ 좌표1 위치 설정 후 엔터를 누르세요...")
coord1 = mc.get_coords()
print("✓ 좌표1 저장됨:", coord1)

# 초기 위치 복귀
mc.power_on()
time.sleep(1)
move_home()
```

```

# 좌표2 수동 설정
print("\n▣ 좌표2 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("▶ 좌표2 위치 설정 후 엔터를 누르세요...")
coord2 = mc.get_coords()
print("✓ 좌표2 저장됨:", coord2)

# 초기 위치 복귀
mc.power_on()
ime.sleep(1)
move_home()

# 2회 반복: 좌표1 → 좌표2 → 초기 위치
for i in range(2):
    print(f"\n▣ 반복 {i+1}/2: 좌표1 이동")
    mc.send_coords(coord1, speed=50, mode=0)
    time.sleep(2)

    print("▶ 좌표2 이동")
    mc.send_coords(coord2, speed=50, mode=0)
    time.sleep(2)

    print("▣ 초기 위치 복귀")
    move_home()

print("\n✓ 모든 작업 완료")

```

```

##첫번째 좌표에서 상자 집고 2번째 좌표에서 상자 놓는 코드 (그리퍼 길이가 있어서 좀 높게 위치를 조정))

from pymycobot.mycobot import MyCobot
import time

# JetCobot 초기 설정
mc = MyCobot('/dev/ttyJETCOBOT', 1000000)
mc.power_on()
mc.thread_lock = True
print("✓ JetCobot이 연결되었습니다.")

# 초기 위치 정의 (그리퍼 open 포함)
initial_angles = [0, 0, 0, 0, 0, 40]

def move_home():
    mc.send_angles(initial_angles, speed=50)
    time.sleep(2)

def open_gripper():
    mc.set_gripper_value(100, 50) # 최대 열기
    time.sleep(1)

def close_gripper():
    mc.set_gripper_value(0, 50) # 최대 닫기
    time.sleep(1)

```

```

# 초기 위치로 이동 및 그리퍼 열기
print("🏠 초기 위치로 이동")
move_home()
open_gripper()

# 좌표1 수동 설정 (집기 위치)
print("\n🛠 좌표1 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("💡 좌표1 (상자 pick 위치) 설정 후 엔터를 누르세요...")
coord1 = mc.get_coords()
print("✅ 좌표1 저장됨:", coord1)

# 초기 위치 복귀
mc.power_on()
time.sleep(1)
move_home()

# 좌표2 수동 설정 (놓기 위치)
print("\n🛠 좌표2 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("💡 좌표2 (상자 drop 위치) 설정 후 엔터를 누르세" \
"요...")
coord2 = mc.get_coords()
print("✅ 좌표2 저장됨:", coord2)

# 초기 위치 복귀
mc.power_on()
time.sleep(1)
move_home()
open_gripper()

# -----
# 2회 반복: pick → 초기 → drop → 초기
for i in range(2):
    print(f"\n🔁 반복 {i+1}/2 시작")

    # 좌표1 이동 후 물건 집기
    print("📦 좌표1로 이동 (집기)")
    mc.send_coords(coord1, speed=50, mode=0)
    time.sleep(2)
    close_gripper()
    time.sleep(1)

    # 초기 위치로 복귀
    print("➡ 초기 위치로 복귀 (물건 든 상태)")
    move_home()

    # 좌표2로 이동 후 물건 놓기
    print("📤 좌표2로 이동 (놓기)")
    mc.send_coords(coord2, speed=50, mode=0)
    time.sleep(2)
    open_gripper()
    time.sleep(1)

    # 다시 초기 위치로 복귀
    print("➡ 초기 위치로 복귀")
    move_home()

```

```

open_gripper()

print("\n✓ 모든 작업 완료")

```

아래 예시는 **myCobot 280** 의 표준-DH 테이블을 그대로 코드에 옮겨 — 원하는 조인트 각(q1 ~ q6, rad)만 바꿔 넣으면

1. **Base** → **End-Effector** 변환 행렬 **T_BE**
2. 그 역행렬인 **EE** → **Base** 변환 **T_EB**
3. 회전 행렬 **R**, 위치 벡터 **p**

를 한번에 구하도록 구성한 스켈레톤입니다.

**자주 하는 실수 체크리스트

1. 라디안/도 혼동 → `np.deg2rad()`로 변환 후 입력.
2. 오프셋 빼먹기 → 항상 $\theta = q + \text{offset}$.
3. 단위 일관성 → d, a 모두 mm 혹은 m로 통일.
4. 역행렬을 일반 `np.linalg.inv()`로 바로 구하면 $\text{SE}(3)$ 구조를 깨는 수치 오차가 생길 수 있으니
 $\text{R}^T, -\text{R}^T p$ 공식을 쓰는 편이 안전.

이제 원하는 조인트 각을 `q_example` 대신 넣어서 돌려 보시면 **Base** → **EE** 위치·자세와 그 역행렬, 회전/변위 벡터를 바로 확인할 수 있습니다!

4.4.2.2 로봇 DH 파라미터를 사용한 회전 및 이동 행렬 계산 code

```

import numpy as np

# _____
# 1) DH 1-프레임 변환
# _____
def dh_transform(theta, d, a, alpha):
    """std-DH 4x4 homogeneous transform A_i-1^i"""
    ct, st = np.cos(theta), np.sin(theta)
    ca, sa = np.cos(alpha), np.sin(alpha)
    #DH파라미터를 4x4 동차변환 행렬로 변환해
    return np.array([
        [ct, -st * ca, st * sa, a * ct],
        [st, ct * ca, -ct * sa, a * st],
        [0, sa, ca, d],
        [0, 0, 0, 1]
    ])

# _____
# 2) myCobot 280 DH 상수 (단위: mm, rad), offsets은 물리적 θ각과 DH 각도 차이를 보정해
# _____
d_vals      = [131.22,     0,       0,   63.4 ,  75.05,  45.6 ]    # d_i

```

```

a_vals      = [    0     , -110.4, -96,   0   ,   0   ]    # a_i
alpha_vals = [ 1.5708,   0,      0,  1.5708, -1.5708, 0]    # α_i
offsets    = [    0     , -1.5708, 0, -1.5708,  1.5708, 0]    # θ_offset_i

# _____
# 3) 전진 기구학 함수 (조인트값 -> 변환 행렬 )
# _____
def forward_kinematics(q):
    """
    q : 길이 6 리스트/배열, [q1..q6] (rad)
    반환: (T_BE 4x4, R 3x3, p 3x1)
    """
    if len(q) != 6:
        raise ValueError("6-axis 로봇이므로 q 길이는 6이어야 합니다.")

    # 전체 동차변환 누적
    T = np.eye(4) #누적 변환 초기값= 단위행렬
    for i in range(6): #6개의 조인트 반복
        theta_i = q[i] + offsets[i]           # θ = q + offset / 실측 + 오프셋
        A_i     = dh_transform(theta_i,
                               d_vals[i],
                               a_vals[i],
                               alpha_vals[i])
        T = T @ A_i                         # 누적 곱

    R = T[:3, :3]    # 회전 성분
    p = T[:3, 3]     # 변위(위치) 성분
    return T, R, p #최종 EE자세를 T,R,p로 쪼개 리턴

# _____
##메모란 !!!!

###**SE(3)**는 3차원 공간의 "자세 + 위치"를 한 번에 다루는 4x4 행렬 집합이고,
##빠른 역행렬 공식과 안정된 누적곱 계산이 가능해서 로봇 제어·비전 등에서 기본 표현 방식이야.
###회전행렬 R의 특성
#직교행렬 (Orthogonal): R.T @ R = I
##행렬식(det R) = +1
##→ 그래서 R의 역행렬은 전치(R.T)로 빠르게 계산 가능!
###T의 역행렬 공식 (SE(3) 고유 특성)
#아래 공식으로 역변환이 빠르게 가능함:
#T_inv = [ R.T      -R.T @ p ]
#[ 0         1      ]
##구성 닫힘
#SE(3) 두 개를 곱해도 SE(3)이다.
##→ T1 @ T2도 여전히 SE(3)이다.
#그래서 누적변환(FK) 할 수 있어!
#좌표변환에 바로 사용 가능
#어떤 3D 점 x 를 위치시키고 싶으면:x' = T @ [x; 1]

# _____
#4) 역행렬(End-Effector → Base), SE(3) 특성을 활용해서 np.linalg.inv 안쓰고 직접 계산

def inverse_transform(T):
    """SE(3) 행렬의 역행렬(빠른 방법)"""
    R = T[:3, :3]
    p = T[:3, 3]

```

```

T_inv      = np.eye(4)
T_inv[:3,:3] = R.T    #  $R^{-1} = R^T$ 
T_inv[3, 3] = -R.T @ p  #  $-R^T \cdot p$ 

return T_inv

# _____
# 5) 사용 예시
#
if __name__ == "__main__":
    # 예: 모든 조인트 0 rad (home posture)
    q_example = [0, 0, 0, 0, 0, 40]

    T_BE, R, p = forward_kinematics(q_example)
    T_EB       = inverse_transform(T_BE)

    np.set_printoptions(precision=6, suppress=True)
    print("Base_T_EE (T_BE) =\n", T_BE)
    print("\nRotation R =\n", R)
    print("\nTranslation p (mm) =\n", p)
    print("\nEE_T_Base (T_EB) =\n", T_EB)

```

#검증함수

```

import numpy as np
from numpy.linalg import norm

def fk_inverse_consistency(q, n_points=10, scale=100):
    """T_BE, T_EB 왕복 오차(RMS) 계산
    q        : 6-DoF 조인트각(rad)
    n_points : EE 기준 무작위 점 개수
    scale    : 점 생성 범위(±scale mm)
    """
    T_BE, _, _ = forward_kinematics(q)
    T_EB       = inverse_transform(T_BE)    # EE→Base 의 역

    # 1) EE 좌표계에 무작위 3-D 점 n개 생성
    np.random.seed(42)                      # 재현 가능
    pts_EE = np.random.uniform(-scale, scale, (n_points, 3))
    pts_EE_h = np.hstack([pts_EE, np.ones((n_points, 1))]).T  # (4,n)

    # 2) EE → Base 변환 // 행렬 곱
    pts_B  = T_BE @ pts_EE_h                # (4,n)

    # 3) 다시 Base → EE 변환(역행렬) #변환된 좌표의 상위 3개 행(row)만 가져옴
    #→ 즉, x, y, z 위치만 추출 (4번째 행은 1이라서 보통 생략)
    pts_EE_rec = (T_EB @ pts_B)[:3].T       # (n,3)

    # 4) 오차 계산
    errs   = norm(pts_EE_rec - pts_EE, axis=1)  # per-point(mm)
    rms    = np.sqrt((errs**2).mean())
    return rms, errs

# — 사용 예시 ——————

```

```

if __name__ == "__main__":
    q_home = [0, 0, 0, 0, 0, 0]
    rms, each = fk_inverse_consistency(q_home, n_points=10, scale=100)

    np.set_printoptions(precision=16, suppress=True) #precision 은 정밀도, 자릿수
    print("개별 오차(mm) =", each)
    print("RMS 오차(mm) =", rms)

```

개별 오차(mm) = [4.44e-16 8.88e-16 5.33e-16 ...]

RMS 오차(mm) = 6.17e-16

1e-15~1e-12 수준이면 알고리즘·오프셋·단위 모두 OK

```

174     np.set_printoptions(precision=0, suppress=True)
175     print("Base_T_EE (T_BE) =\n", T_BE)
176     print("\nRotation R =\n", R)
177     print("\nTranslation p (mm) =\n", p)
178     print("EE_T_Base (T_EB) =\n", T_EB)

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS

① (jetcobot) jetcobot@raspi:~$ /home/jetcobot/venv/jetcobot/bin/python3 /home/jetcobot/.jupyter/dev/practice1.py
Note: This class is no longer maintained since v3.6.0, please refer to the project documentation: https://github.com/e
✓ JetCobot0| 연결되었습니다.
📍 초기 위치로 이동

🛠 좌표1 설정 : 모터 OFF - 수동 조작
❶ 좌표1 (상자 pick 위치) 설정 후 엔터를 누르세요...
File "/home/jetcobot/.jupyter/dev/practice1.py", line 33, in <module>
    # input("❶ 좌표1 (상자 pick 위치) 설정 후 엔터를 누르세요...")

KeyboardInterrupt

● (jetcobot) jetcobot@raspi:~$ /home/jetcobot/venv/jetcobot/bin/python3 /home/jetcobot/.jupyter/dev/practice1.py
Note: This class is no longer maintained since v3.6.0, please refer to the project documentation: https://github.com/e
✓ JetCobot0| 연결되었습니다.
📍 초기 위치로 이동
● (jetcobot) jetcobot@raspi:~$ /home/jetcobot/venv/jetcobot/bin/python3 /home/jetcobot/.jupyter/dev/practice1.py
Note: This class is no longer maintained since v3.6.0, please refer to the project documentation: https://github.com/e
✓ JetCobot0| 연결되었습니다.
📍 초기 위치로 이동
Base_T_EE (T_BE) =
[[ 0.000004 -0.000011  1.        45.601309]
 [ -1.         0.        0.000004 -63.400591]
 [ -0.         -1.       -0.000011 412.669265]
 [  0.          0.          0.          1.        ]]

Rotation R =
[[ 0.000004 -0.000011  1.        ]
 [-1.          0.        0.000004]
 [-0.          -1.       -0.000011]]

Translation p (mm) =
[ 45.601309 -63.400591 412.669265]

EE_T_Base (T_EB) =
[[ 0.000004 -1.        -0.        -63.400758]
 [ -0.000011  -0.        -1.        412.669767]
 [  1.          0.000004 -0.000011 -45.596529]
 [  0.          0.          0.          1.        ]]

```

```

169     q_example = [[0, 0, 0, 20, 0, 0]]
170
171     T_BE, R, p = forward_kinematics(q_example)
172     T_EB        = inverse_transform(T_BE)
173
174     np.set_printoptions(precision=6, suppress=True)
175     print("Base_T_EE (T_BE) =\n", T_BE)
176     print("\nRotation R =\n", R)
177     print("\nTranslation p (mm) =\n", p)
178     print("\nEE_T_Base (T_EB) =\n", T_EB)
179
180
filebrowser...
$                                         
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
[ 45.601309 -63.400591 412.669265]

EE_T_Base (T_EB) =
[[ 0.000004 -1.         -0.         -63.400758]
 [ -0.000011  0.         -1.         412.669767]
 [  1.         0.000004 -0.000011 -45.596529]
 [  0.         0.         0.         1.        ]]
● (jetcobot) jetcobot@raspi:~$ /home/jetcobot/venv/jetcobot/bin/python3 /home/jetcobot/.jupyter/dev/practice1.py
Note: This class is no longer maintained since v3.6.0, please refer to the project documentation: https://github.com/e
✓ JetCobot0| 연결되었습니다.
⚠ 초기 위치로 이동
Base_T_EE (T_BE) =
[[ -0.988032  0.14082   0.062949 -70.339937]
 [ -0.15425   -0.902015 -0.403208  39.529625]
 [  0.000001  -0.408092  0.912941 409.876928]
 [  0.         0.         0.         1.        ]]
● (jetcobot) jetcobot@raspi:~$ 
Rotation R =
[[ -0.988032  0.14082   0.062949]
 [ -0.15425   -0.902015 -0.403208]
 [  0.000001  -0.408092  0.912941]]
Translation p (mm) =
[-70.339937  39.529625 409.876928]

EE_T_Base (T_EB) =
[[ -0.988032  -0.15425    0.000001  -63.401156]
 [  0.14082    -0.902015  -0.408092  212.829137]
 [  0.062949   -0.403208   0.912941  -353.826858]
 [  0.         0.         0.         1.        ]]
○ (jetcobot) jetcobot@raspi:~$ []

```

4.4.2.2.1 #QR 코드 인식 후 정보 읽기

QR코드 생성 <https://me-qr.com/ko/qr-code-generator/qr>

```

import cv2
import requests
from bs4 import BeautifulSoup #BeautifulSoup 은 HTML 파싱해서 텍스트 추출용

qr_detector = cv2.QRCodeDetector() #OpenCV에서 제공하는 QR 코드 인식기 생성
cap = cv2.VideoCapture(0) #0.1.2..중 택1

if not cap.isOpened():
    print("카메라를 열 수 없습니다.")
    exit()

prev_data = "" # 중복 방지, 이전에 인식한 QR 코드 내용을 기억해두고, 같은 값이면 중복 출력 방지
#근데 중복 방지를 하면 안될거같음 !! (수정 고려)

#실시간으로 카메라에서 프레임(영상 한 장) 받아오기
while True:
    ret, frame = cap.read()
    if not ret:
        print("프레임 읽기 실패")

```

```

break

    data, points, _ = qr_detector.detectAndDecode(frame) # QR코드에 들어있는 문자열 (URL), 큐
    알4개꼭짓점, 큐알 원본이미지(사용 안 함)

    # QR 영역이 인식되고, 유효한 좌표인지 확인
    if points is not None and len(points) > 0 and data:
        points = points[0].astype(int)

        # QR 코드 테두리 그리기 꼭짓점 좌표를 int로 변환하여 openCV에서 그리기 가능하게 해
        for i in range(len(points)):
            pt1 = tuple(points[i])
            pt2 = tuple(points[(i+1) % len(points)])
            cv2.line(frame, pt1, pt2, (0, 255, 0), 2) #line함수로 네개 꼭짓점 연결해서 초록색 테두리 그
        립

        # 중복 방지 및 디코딩된 텍스트 처리_ 같은 큐알 코드가 계속 감지되어도 콘솔에 한번만 출력 되도록 중복 필터링
        if data != prev_data:
            prev_data = data
            print(f"[QR 코드 URL] {data}")

        # 웹 페이지 내용 가져오기 // 웹페이지 전체를 파싱해서 <body> 영역의 텍스트만 추출
        try:
            response = requests.get(data, timeout=5) #웹페이지의 HTML 전체를 문자열로받아
            soup = BeautifulSoup(response.text, 'html.parser')

            # 전체 body 텍스트 추출
            body_text = soup.body.get_text(strip=True)
            print(f"[페이지 내용] {body_text}")
        except Exception as e:
            print(f"[에러] 웹페이지 요청 실패: {e}") #예외처리

        # 프레임에 텍스트 표시
        cv2.putText(frame, data, (points[0][0], points[0][1] - 10),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)

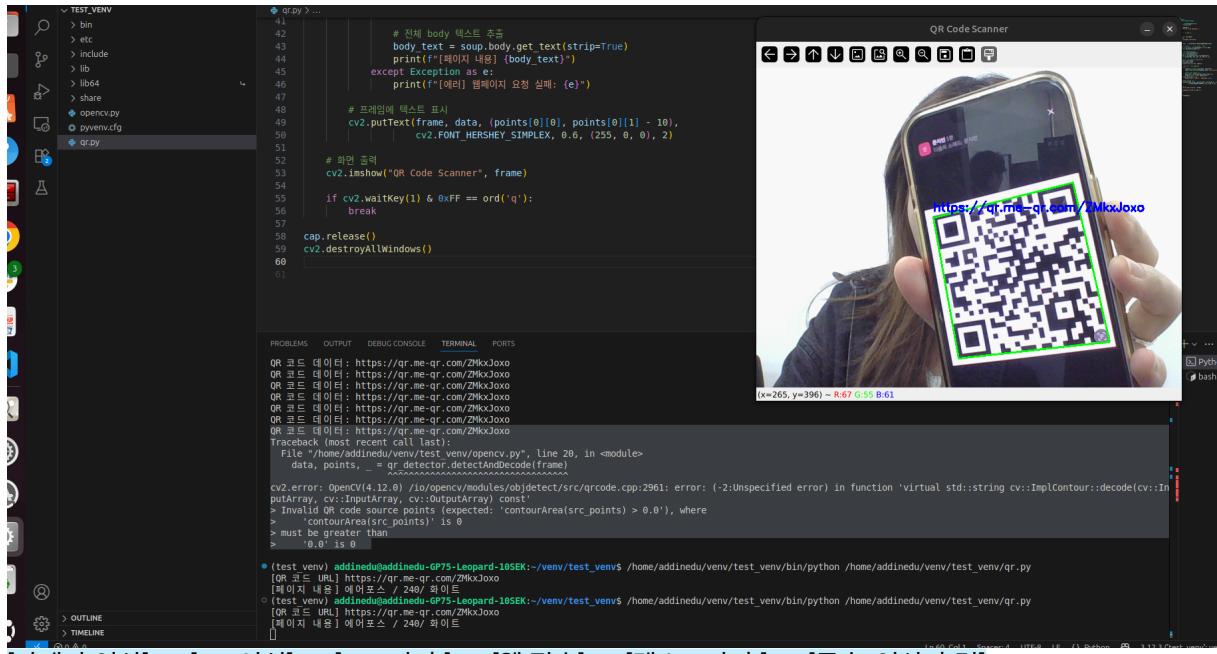
        # 화면 출력
        cv2.imshow("QR Code Scanner", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()

```

아래처럼 출력 에어포스/240/화이트



[카메라 영상] → [QR 인식] → [URL 추출] → [웹 접속] → [텍스트 추출] → [콘솔/영상 출력]

```
from flask import Flask
from pyngrok import ngrok # 추가

app = Flask(__name__)

@app.route("/product/<name>/<size>/<color>")
def show_product(name, size, color):
    return f"상품명: {name}, 크기: {size}, 색상: {color}"

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=True)
```

4.4.2.3 Flask 서버에서 HTML 형식으로 응답 주기 (위 코드 수정 버전)

```
from flask import Flask
from pyngrok import ngrok # 필요 없으면 생략 가능

app = Flask(__name__)

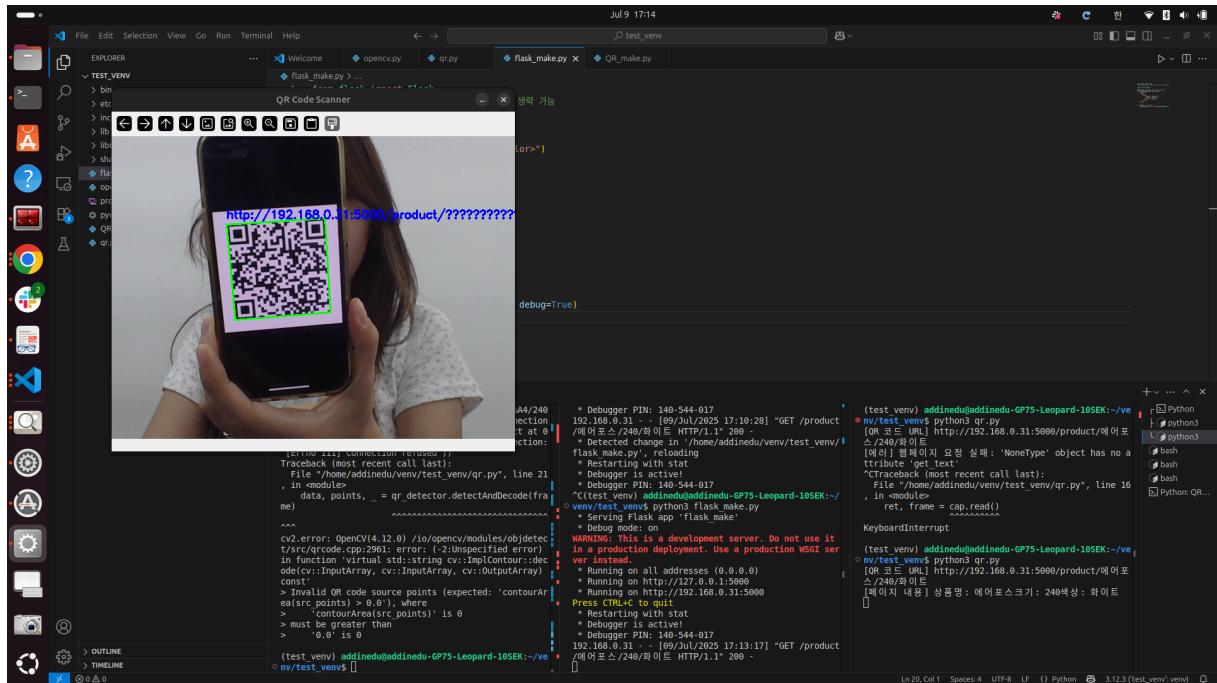
@app.route("/product/<name>/<size>/<color>")
def show_product(name, size, color):
    return """
<html>
<body>
    <p>상품명: {name}</p>
    <p>사이즈: {size}</p>
    <p>색상: {color}</p>
</body>
</html>
    """
```

```

        </body>
    </html>
"""

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=True)

```



```

#QR생성
import qrcode

# Flask 서버 주소에 맞게 변경
url = "http://192.168.0.31:5000/product/에어포스/240/화이트"

# QR 코드 생성
qr = qrcode.QRCode(version=1, box_size=10, border=4)
qr.add_data(url)
qr.make(fit=True)

img = qr.make_image(fill="black", back_color="white")
img.save("product_qr.png") # 파일로 저장
img.show()

```

터미널 켜서 두개 동시 실행

```

import cv2
import requests
from bs4 import BeautifulSoup

```

```

#사이트 통해서 만든 QR을 인식함
qr_detector = cv2.QRCodeDetector()
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("카메라를 열 수 없습니다.")
    exit()

prev_data = "" # 종복 방지

while True:
    ret, frame = cap.read()
    if not ret:
        print("프레임 읽기 실패")
        break

    data, points, _ = qr_detector.detectAndDecode(frame)

    # QR 영역이 인식되고, 유효한 좌표인지 확인
    if points is not None and len(points) > 0 and data:
        points = points[0].astype(int)

        # QR 코드 테두리 그리기
        for i in range(len(points)):
            pt1 = tuple(points[i])
            pt2 = tuple(points[(i+1) % len(points)])
            cv2.line(frame, pt1, pt2, (0, 255, 0), 2)

        # 종복 방지 및 디코딩된 텍스트 처리
        if data != prev_data:
            prev_data = data
            print(f"[QR 코드 URL] {data}")

        # 웹 페이지 내용 가져오기
        try:
            response = requests.get(data, timeout=5)
            soup = BeautifulSoup(response.text, 'html.parser')

            # 전체 body 텍스트 추출
            body_text = soup.body.get_text(strip=True)
            print(f"[페이지 내용] {body_text}")
        except Exception as e:
            print(f"[에러] 웹페이지 요청 실패: {e}")

        # 프레임에 텍스트 표시
        cv2.putText(frame, data, (points[0][0], points[0][1] - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)

        # 화면 출력
        cv2.imshow("QR Code Scanner", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()

```

4.4.2.4 QR 코드 인식(OpenCV) + Flask 서버 실행을 한 파일

```

from flask import Flask
import threading
import cv2
import requests
from bs4 import BeautifulSoup
import time

# --- Flask 서버구성 ---
app = Flask(__name__)

@app.route("/product/<name>/<size>/<color>")
def show_product(name, size, color):
    return f"""
<html>
    <body>
        <p>상품명: {name}</p>
        <p>사이즈: {size}</p>
        <p>색상: {color}</p>
    </body>
</html>
"""

# --- Flask 서버를 스레드에서 실행 ---
def run_flask():
    app.run(host='0.0.0.0', port=5000, debug=False)

# --- QR 코드 인식 함수 ---
def detect_qr():
    qr_detector = cv2.QRCodeDetector()
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("카메라를 열 수 없습니다.")
        return

    prev_data = ""

    while True:
        ret, frame = cap.read()
        if not ret:
            print("프레임 읽기 실패")
            break

        data, points, _ = qr_detector.detectAndDecode(frame)

        if points is not None and len(points) > 0 and data:
            points = points[0].astype(int)
            for i in range(len(points)):
                pt1 = tuple(points[i])
                pt2 = tuple(points[(i+1) % len(points)])
                cv2.line(frame, pt1, pt2, (0, 255, 0), 2)

            if data != prev_data:
                prev_data = data

```

```

print(f"[QR 코드 URL] {data}")

try:
    response = requests.get(data, timeout=5)
    soup = BeautifulSoup(response.text, 'html.parser')
    body_text = soup.body.get_text(strip=True)
    print(f"[페이지 내용] {body_text}")
except Exception as e:
    print(f"[에러] 웹페이지 요청 실패: {e}")

cv2.putText(frame, data, (points[0][0], points[0][1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)

cv2.imshow("QR Code Scanner", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()

# --- 메인 시작점 ---
if __name__ == "__main__":
    # 1. Flask 서버를 백그라운드 스레드로 시작
    flask_thread = threading.Thread(target=run_flask)
    flask_thread.daemon = True # 메인 종료되면 같이 종료됨
    flask_thread.start()

    # 2. Flask 서버가 커질 시간 기다린 후 카메라 시작
    time.sleep(2) # 2초 대기 (서버 부팅 시간)
    detect_qr()

```

4.4.2.5 #QR코드 생성 + 인식 + flask서버 생성 !!!!

동일한 네트워크가 아니더라도 큐알인식 가능

 외부 기기로 접속시 visit site 누르면 정보 나옴

```

from flask import Flask
import threading
import cv2
import requests
from bs4 import BeautifulSoup
import time
from pyngrok import ngrok # Ngrok 추가
import qrcode

# --- Flask 서버 구성 ---
app = Flask(__name__)

```

```

@app.route("/product/<name>/<size>/<color>")
def show_product(name, size, color):
    return f"""
<html>
    <body>
        <p>상품명: {name}</p>
        <p>사이즈: {size}</p>
        <p>색상: {color}</p>
    </body>
</html>
"""

# --- Flask 서버를 스레드에서 실행 ---
def run_flask():
    # ngrok으로 외부 URL 연결
    public_url = ngrok.connect(5000).public_url
    print(f"[Ngrok 외부 접속 URL] {public_url}")

    # QR 코드 생성 (예시 상품)
    generate_qr(public_url, "에어포스", "250", "화이트")

    app.run(host='0.0.0.0', port=5000, debug=False)

# --- QR 코드 생성 함수 ---
def generate_qr(base_url, name, size, color):
    full_url = f"{base_url}/product/{name}/{size}/{color}"
    print(f"[QR 코드 생성용 URL] {full_url}")

    qr = qrcode.QRCode(version=1, box_size=10, border=4)
    qr.add_data(full_url)
    qr.make(fit=True)

    img = qr.make_image(fill="black", back_color="white")
    img.save("product_qr.png")
    img.show()

# --- QR 코드 인식 함수 ---
def detect_qr():
    qr_detector = cv2.QRCodeDetector()
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("카메라를 열 수 없습니다.")
        return

    prev_data = ""

    while True:
        ret, frame = cap.read()
        if not ret:
            print("프레임 읽기 실패")
            break

        data, points, _ = qr_detector.detectAndDecode(frame)

        if points is not None and len(points) > 0 and data:

```

```

points = points[0].astype(int)
for i in range(len(points)):
    pt1 = tuple(points[i])
    pt2 = tuple(points[(i+1) % len(points)])
    cv2.line(frame, pt1, pt2, (0, 255, 0), 2)

if data != prev_data:
    prev_data = data
    print(f"[QR 코드 URL] {data}")

try:
    response = requests.get(data, timeout=5)
    soup = BeautifulSoup(response.text, 'html.parser')
    body_text = soup.body.get_text(strip=True)
    print(f"[페이지 내용] {body_text}")
except Exception as e:
    print(f"[에러] 웹페이지 요청 실패: {e}")

cv2.putText(frame, data, (points[0][0], points[0][1] - 10),
           cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)

cv2.imshow("QR Code Scanner", frame)

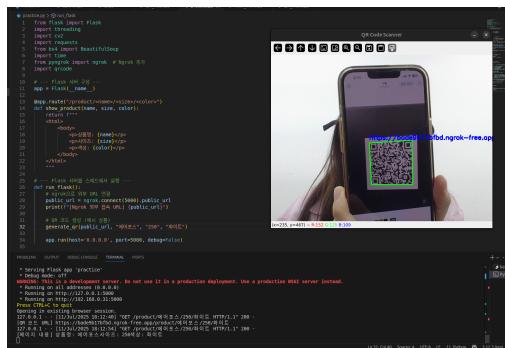
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()

# --- 메인 시작점 ---
if __name__ == "__main__":
    flask_thread = threading.Thread(target=run_flask)
    flask_thread.daemon = True
    flask_thread.start()

time.sleep(3) # Flask 서버 및 ngrok 준비 대기
detect_qr()

```



카메라 띄워서 이미지 출력하는 코드

```

import cv2
import numpy as np
import requests

```

```

import threading
import time
import os
class VideoStreamClient:
    def __init__(self, server_url):
        self.server_url = server_url
        self.stream_active = False
        self.stream_thread = None
        self.latest_frame = None
        self.frame_lock = threading.Lock()
    def start_stream(self):
        if self.stream_active:
            print("이미 스트림이 활성화되어 있습니다.")
            return
        self.stream_active = True
        self.stream_thread = threading.Thread(target=self._receive_stream)
        self.stream_thread.daemon = True
        self.stream_thread.start()
    def stop_stream(self):
        self.stream_active = False
        if self.stream_thread:
            self.stream_thread.join(timeout=1.0)
    def get_latest_frame(self):
        with self.frame_lock:
            if self.latest_frame is None:
                return None
            return self.latest_frame.copy()
    def _receive_stream(self):
        try:
            response = requests.get(self.server_url, stream=True)
            if response.status_code != 200:
                print(f"서버 연결 실패: {response.status_code}")
                self.stream_active = False
                return
            bytes_data = bytes()
            for chunk in response.iter_content(chunk_size=1024):
                bytes_data += chunk
                a = bytes_data.find(b'\xff\xd8')
                b = bytes_data.find(b'\xff\xd9')
                if a != -1 and b != -1:
                    jpg = bytes_data[a:b+2]
                    bytes_data = bytes_data[b+2:]
                    frame = cv2.imdecode(np.frombuffer(jpg, dtype=np.uint8),
cv2.IMREAD_COLOR)
                    with self.frame_lock:
                        self.latest_frame = frame
                    if not self.stream_active:
                        break
        except Exception as e:
            print(f"스트림 수신 중 오류 발생: {e}")
        finally:
            self.stream_active = False
    if __name__ == "__main__":
        server_url = "http://192.168.5.1:5000/stream"
        save_dir = "/home/addinedu/cali_test" # 원하는 저장 디렉토리
        os.makedirs(save_dir, exist_ok=True)
        client = VideoStreamClient(server_url)

```

```

client.start_stream()
image_count = 0 # 저장된 이미지 번호
try:
    while True:
        frame = client.get_latest_frame()
        if frame is not None:move_group.launch.py
            cv2.imshow("Jetcobot Camera Stream", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
        elif key == ord('c'):
            filename = os.path.join(save_dir, f"capture_{image_count:03d}.png")
            if frame is not None:
                cv2.imwrite(filename, frame)
                print(f"이미지 저장됨: {filename}")
                image_count += 1
except KeyboardInterrupt:
    print("종료 중...")
finally:
    client.stop_stream()
    cv2.destroyAllWindows()

```

4.4.2.5.1 #카메라 연결

```

import numpy as np
import requests
import threading
import time
import os
class VideoStreamClient:
    def __init__(self, server_url):
        self.server_url = server_url
        self.stream_active = False
        self.stream_thread = None
        self.latest_frame = None
        self.frame_lock = threading.Lock()
    def start_stream(self):
        if self.stream_active:
            print("이미 스트림이 활성화되어 있습니다.")
            return
        self.stream_active = True
        self.stream_thread = threading.Thread(target=self._receive_stream)
        self.stream_thread.daemon = True
        self.stream_thread.start()
    def stop_stream(self):
        self.stream_active = False
        if self.stream_thread:
            self.stream_thread.join(timeout=1.0)
    def get_latest_frame(self):
        with self.frame_lock:
            if self.latest_frame is None:
                return None

```

```

        return self.latest_frame.copy()
def _receive_stream(self):
    try:
        response = requests.get(self.server_url, stream=True)
        if response.status_code != 200:
            print(f"서버 연결 실패: {response.status_code}")
            self.stream_active = False
            return
        bytes_data = bytes()
        for chunk in response.iter_content(chunk_size=1024):
            bytes_data += chunk
            a = bytes_data.find(b'\xff\xd8')
            b = bytes_data.find(b'\xff\xd9')
            if a != -1 and b != -1:
                jpg = bytes_data[a:b+2]
                bytes_data = bytes_data[b+2:]
                frame = cv2.imdecode(np.frombuffer(jpg, dtype=np.uint8),
cv2.IMREAD_COLOR)
                with self.frame_lock:
                    self.latest_frame = frame
                if not self.stream_active:
                    break
    except Exception as e:
        print(f"스트림 수신 중 오류 발생: {e}")
    finally:
        self.stream_active = False
if __name__ == "__main__":
    server_url = "http://192.168.5.1:5000/stream"
    save_dir = "/home/addinedu/cali_test" # 원하는 저장 디렉토리
    os.makedirs(save_dir, exist_ok=True)
    client = VideoStreamClient(server_url)
    client.start_stream()
    image_count = 0 # 저장된 이미지 번호
    try:
        while True:
            frame = client.get_latest_frame()
            if frame is not None:
                cv2.imshow("Jetcobot Camera Stream", frame)
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                break
            elif key == ord('c'):
                filename = os.path.join(save_dir, f"capture_{image_count:03d}.png")
                if frame is not None:
                    cv2.imwrite(filename, frame)
                    print(f"이미지 저장됨: {filename}")
                    image_count += 1
    except KeyboardInterrupt:
        print("종료 중...")
    finally:
        client.stop_stream()
        cv2.destroyAllWindows()

```

4.4.3 로봇팔 임의 좌표

```
# [28.21, -48.42, -46.23, 5.36, 6.59, 37.7]
```

```
from pymycobot.mycobot import MyCobot
import time

# JetCobot 초기 설정
mc = MyCobot('/dev/ttyJETCOBOT', 1000000)
mc.power_on()
mc.thread_lock = True
print("✓ JetCobot이 연결되었습니다.")

# 초기 위치로 이동 (기본 자세 + 그리퍼 열기)
initial_angles = [0, 0, 0, 0, 0, 40]
mc.send_angles(initial_angles, speed=50)
time.sleep(2)

# 모터 OFF → 수동 조작 허용
print("⚙ 모터 OFF - 수동으로 원하는 위치로 이동하세요.")
mc.release_all_servos()
input("👉 위치 조정 후 엔터를 누르세요...")

24
# 현재 좌표 읽기 (사용자가 이동한 위치)
target_coords = mc.get_coords()
print("📍 저장된 위치 좌표:", target_coords)

# 모터 ON → 초기 위치 복귀
mc.power_on()
time.sleep(1)
mc.send_angles(initial_angles, speed=50)
time.sleep(2)

# 저장된 좌표로 3회 반복 이동
for i in range(3):
    print(f"\n➡️ 반복 {i+1}/3: 저장된 위치로 이동 중...")
    mc.send_coords(target_coords, speed=50, mode=0) # mode=0: 상대좌표 X, 절대좌표
    time.sleep(3)
    print("➡️ 다시 초기 위치로 복귀")
    mc.send_angles(initial_angles, speed=50)
    time.sleep(2)

print("✓ 모든 작업 완료")
```

```
##경유지 설정해서 움직이는 코드 (좌표 2개 설정)
```

```
from pymycobot.mycobot import MyCobot
import time
```

```

# JetCobot 초기 설정
mc = MyCobot('/dev/ttyJETCOBOT', 1000000)
mc.power_on()
mc.thread_lock = True
print("✓ JetCobot이 연결되었습니다.")

# 초기 위치 정의 (그리퍼 open 포함)
initial_angles = [0, 0, 0, 0, 0, 40]

def move_home():
    mc.send_angles(initial_angles, speed=50)
    time.sleep(2)

# 초기 위치로 이동
print("🏠 초기 위치로 이동")
move_home()

# 좌표1 수동 설정
print("\n🛠 좌표1 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("❗ 좌표1 위치 설정 후 엔터를 누르세요...")
coord1 = mc.get_coords()
print("✓ 좌표1 저장됨:", coord1)

# 초기 위치 복귀
mc.power_on()
time.sleep(1)
move_home()

# 좌표2 수동 설정
print("\n🛠 좌표2 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("❗ 좌표2 위치 설정 후 엔터를 누르세요...")
coord2 = mc.get_coords()
print("✓ 좌표2 저장됨:", coord2)

# 초기 위치 복귀
mc.power_on()
ime.sleep(1)
move_home()

# 2회 반복: 좌표1 → 좌표2 → 초기 위치
for i in range(2):
    print(f"\n➡ 반복 {i+1}/2: 좌표1 이동")
    mc.send_coords(coord1, speed=50, mode=0)
    time.sleep(2)

    print("➡ 좌표2 이동")
    mc.send_coords(coord2, speed=50, mode=0)
    time.sleep(2)

    print("➡ 초기 위치 복귀")
    move_home()

print("\n✓ 모든 작업 완료")

```

```

##첫번째 좌표에서 상자 집고 2번째 좌표에서 상자 놓는 코드 (그리퍼 길이가 있어서 좀 높게 위치를 조정)

from pymycobot.mycobot import MyCobot
import time

# JetCobot 초기 설정
mc = MyCobot('/dev/ttyJETCOBOT', 1000000)
mc.power_on()
mc.thread_lock = True
print("✓ JetCobot이 연결되었습니다.")

# 초기 위치 정의 (그리퍼 open 포함)
initial_angles = [0, 0, 0, 0, 0, 40]

def move_home():
    mc.send_angles(initial_angles, speed=50)
    time.sleep(2)

def open_gripper():
    mc.set_gripper_value(100, 50) # 최대 열기
    time.sleep(1)

def close_gripper():
    mc.set_gripper_value(0, 50) # 최대 닫기
    time.sleep(1)

# 초기 위치로 이동 및 그리퍼 열기
print("🏠 초기 위치로 이동")
move_home()
open_gripper()

# 좌표1 수동 설정 (집기 위치)
print("\n🛠 좌표1 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("👉 좌표1 (상자 pick 위치) 설정 후 엔터를 누르세요...")
coord1 = mc.get_coords()
print("✓ 좌표1 저장됨:", coord1)

# 초기 위치 복귀
mc.power_on()
time.sleep(1)
move_home()

# 좌표2 수동 설정 (놓기 위치)
print("\n🛠 좌표2 설정: 모터 OFF - 수동 조작")
mc.release_all_servos()
input("👉 좌표2 (상자 drop 위치) 설정 후 엔터를 누르세" \
"요...")
coord2 = mc.get_coords()
print("✓ 좌표2 저장됨:", coord2)

# 초기 위치 복귀
mc.power_on()
time.sleep(1)

```

```

move_home()
open_gripper()

# -----
# 2회 반복: pick → 초기 → drop → 초기
for i in range(2):
    print(f"\n▣ 반복 {i+1}/2 시작"

    # 좌표1 이동 후 물건 집기
    print("⬇ 좌표1로 이동 (집기)")
    mc.send_coords(coord1, speed=50, mode=0)
    time.sleep(2)
    close_gripper()
    time.sleep(1)

    # 초기 위치로 복귀
    print("▣ 초기 위치로 복귀 (물건 든 상태)")
    move_home()

    # 좌표2로 이동 후 물건 놓기
    print("⬆ 좌표2로 이동 (놓기)")
    mc.send_coords(coord2, speed=50, mode=0)
    time.sleep(2)
    open_gripper()
    time.sleep(1)

    # 다시 초기 위치로 복귀
    print("▣ 초기 위치로 복귀")
    move_home()
    open_gripper()

print("\n✓ 모든 작업 완료")

```

4.5 VISION

4.5.1 Homography

https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html

<https://visionbook.mit.edu/homography.html>

[https://en.wikipedia.org/wiki/Homography_\(computer_vision\)](https://en.wikipedia.org/wiki/Homography_(computer_vision))

<https://www.geeksforgeeks.org/computer-vision/what-is-homography-how-to-estimate-homography-between-two-images/>

[Homography.pdf⁸](#)

8. <https://qkrtmddn0507.atlassian.net/wiki/download/attachments/8552502/Homography.pdf?api=v2&cacheVersion=1&modificationDate=1752420578281&version=1>



4.5.2 ROS2 OpenCV c++ build



작성자 : 박승우 - ubuntu 및 c++ 환경에서 opencv build 하는 방법

4.5.2.1 OpenCV Linux

- opencv Linux download Link : https://docs.opencv.org/4.x/d7/d9f/tutorial_linux_install.html

```
# Install minimal prerequisites (Ubuntu 18.04 as reference)
sudo apt update && sudo apt install -y cmake g++ wget unzip

# Download and unpack sources
wget -O opencv.zip https://github.com/opencv/opencv/archive/4.x.zip
unzip opencv.zip

# Create build directory
mkdir -p build && cd build

# Configure
cmake ..../opencv-4.x

# Build
cmake --build .
```

위 프로세스를 완성하면 build 폴더에 여러 폴더중 “lib” 폴더 내부에 **.o .so 파일** 생성됨.

4.5.2.2 vscode에서 외부 라이브러리 사용하는 방법 (cmake)

- 리눅스 환경에서 cmake 와 make를 사용한 빌드 프로세스
 - https://www.youtube.com/watch?v=QhKkZ_7Wgfg&t=724s
- opencv PC 버전 build 파일 (디버그 빌드)
 - [lib_pc.zip⁹](#)
- opencv 라즈베리파일 build 파일 (디버그 빌드)
 - [lib_Raspberry_PI.zip¹⁰](#)



CPU 아키텍처에 따라 빌드했으므로 아키텍처에 맞춰서 다운 받아야 함

- build file 구조
 - lib
 - include : opencv 헤더파일
 - bin : opencv .o .so 라이브러리 파일



해당 파일을 다운받아 적절한 경로에 첨부하고 프로젝트에서 경로를 통해 접근하면 됨.

4.5.2.2.1 vscode에서 외부라이브러리 링크하는 방법

- 예제 코드 폴더 구조
 - opencv_test
 - main.cpp : 예제코드
 - CmakeList.txt : cmake 빌드용 파일
 - build 폴더 : 빌드 후 실행파일(.out)
 - lib 폴더 : 라이브러리 폴더
 - opencv
 - include : opencv 헤더파일
 - bin : opencv .o .so 라이브러리 파일

-
- main.cpp 예제코드

9. https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/720925/lib_pc.zip?api=v2&cacheVersion=1&modificationDate=1751594628548&version=2

10. https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/720925/lib_Raspberry_PI.zip?api=v2&cacheVersion=1&modificationDate=1751594631066&version=2

```
#include <iostream>
#include "opencv2/opencv.hpp" //opencv 헤더

using namespace std;
using namespace cv;

int main()
{
    std::cout << cv::getBuildInformation() << std::endl;
    Mat image = imread("download.jpg");
    if(image.empty())
    {
        printf("Image Load fail");
        return -1;
    }

    imshow("download.jpg",image);
    waitKey(0);
}
```

- c_cpp_properties.json (vscode에서 IntelliSense를 사용하기 위해 설정)

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "${workspaceFolder}/lib/include"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "c17",
            "cppStandard": "gnu++17",
            "intelliSenseMode": "linux-gcc-x64",
            "configurationProvider": "ms-vscode.cmake-tools"
        }
    ],
    "version": 4
}
```



includePath 내부에 opencv 헤더파일 폴더 경로 삽입

- launch.json (디버깅을 사용하기 위해 필요한 파일)

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug OpenCV Program",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/build/opencv_Test",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerPath": "/usr/bin/gdb",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ],
      "preLaunchTask": "build",
      "internalConsoleOptions": "openOnSessionStart"
    }
  ]
}
```



"program": "\${workspaceFolder}/build/ 빌드후 실행파일 이름" ← 작성

4.5.2.2.1.1 QnA

- 위 방법으로 vscode를 설정하는 이유는?
 - vscode는 코드 편집기로 visual studio와 다르게 컴파일러가 존재하지 않는다, 물론 gcc or g++를 사용해서 빌드 할수는 있지만 ROS 환경에서 빌드 하기 위해 cmake로 빌드하는 방법으로 통일 하였다.
- build 파일을 따로 압축해서 올린 이유는?
 - 설치자마다 opencv 패키지의 설치 경로를 다르게 할수 있어, 외부라이브러리를 사용할때 필요한 헤더 파일과 라이브러리 파일만 추출하여 상대경로로 사용할수 있도록 구성함.

4.5.2.2.2 CmakeList.txt file 작성 요령

```
cmake_minimum_required(VERSION 3.10)
```

```

project(OpenCV_Test)

set(CMAKE_CXX_STANDARD 17)

# 경로 설정
set(OpenCV_INCLUDE_DIR "${CMAKE_SOURCE_DIR}/lib/include")
set(OpenCV_LIB_DIR "${CMAKE_SOURCE_DIR}/lib/bin")

# include 경로 등록
include_directories(${OpenCV_INCLUDE_DIR})

# 라이브러리 경로 등록
link_directories(${OpenCV_LIB_DIR})

# 실행 파일 생성
add_executable(OpenCV_Test main.cpp)

# 필요한 OpenCV 모듈 수동 연결 (필요한 .a/.so 파일 이름만 작성)
target_link_libraries(OpenCV_Test
    opencv_core
    opencv_imgproc
    opencv_highgui
    opencv_imgcodecs
)

```

- 1 : 프로젝트에서 사용할 최소 Cmake 버전 (3.10 버전 보다 낮으면 에러 발생)
- 2 : Cmake 프로젝트 이름 - 빌드후 생성되는 파이너리(실행파일)의 이름으로 사용됨.
- 4 : C++ 표준은 C++17로 설정
- 7 ~ 8 : OpenCV_INCLUDE_DIR, OpenCV_LIB_DIR에 각각 경로를 설정함
 - CMAKE_SOURCE_DIR : CMakeLists.txt 가 위치한 최상위 디렉토리
- 11 : 컴파일러가 참조할 헤더파일 경로
- 15 : 링커가 .so or .a 라이브러리를 찾을수 있도록 경로 설정
- 18 : main.cpp 파일을 기반으로 OpenCV_Test라는 이름의 **실행 파일 생성**
- 21 ~ 25 : OpenCV_Test 실행 파일에 필요한 **OpenCV 라이브러리를 연결**
 - 각 이름은 libopencv_core.so, libopencv_imgproc.so 등의 파일명에서 lib 접두어와 .so 확장자를 제외한 이름
 - link_directories()에서 등록한 경로에서 해당 .so 또는 .a 파일을 찾는다.



Cmake에서 디버깅 모드로 빌드한 경우 vscode에서 launch.json 파일만 설정하면 디버깅 가능

4.5.3 YOLOv8 + ROS2 (Python ver.)



/이찬종/ YOLO를 ros2와 연동해 object detection 하는 법

- 일단 해 보고는 있지만 여기 패키지들 그대로 쓰기보단 참고해서
- 그냥 'Flask 등으로 camera 송출 → YOLO로 판단 → 위치 정보만 ROS2 통신'으로

전송하는 방식이 나을 거 같다는 생각 중. 물론 중간에 생각 또 바뀔 수 있음

1. ROS2와 YOLO 연동
2. YOLO custom dataset 생성/학습
3. 노트북에 GPU 환경 설정 (이건 강의장에서 가능할듯)
- (4. 모터 제어하기 ?)

4.5.3.1 1. ROS2와 YOLO 연동

상단 카메라 높이 대략 147cm

```
addinedu@addinedu-Bravo-17-D7VF:~$ ls /dev/vid*
/dev/video0  /dev/video1
addinedu@addinedu-Bravo-17-D7VF:~$ ls /dev/vid*
/dev/video0  /dev/video1  /dev/video2  /dev/video3
addinedu@addinedu-Bravo-17-D7VF:~$ █
```

노트북에 상단 카메라 연결 전/후 화면. /dev/video2 로 가져와서 하면 될듯함

(집 컴퓨터의 경우 /dev/video0가 떠서)

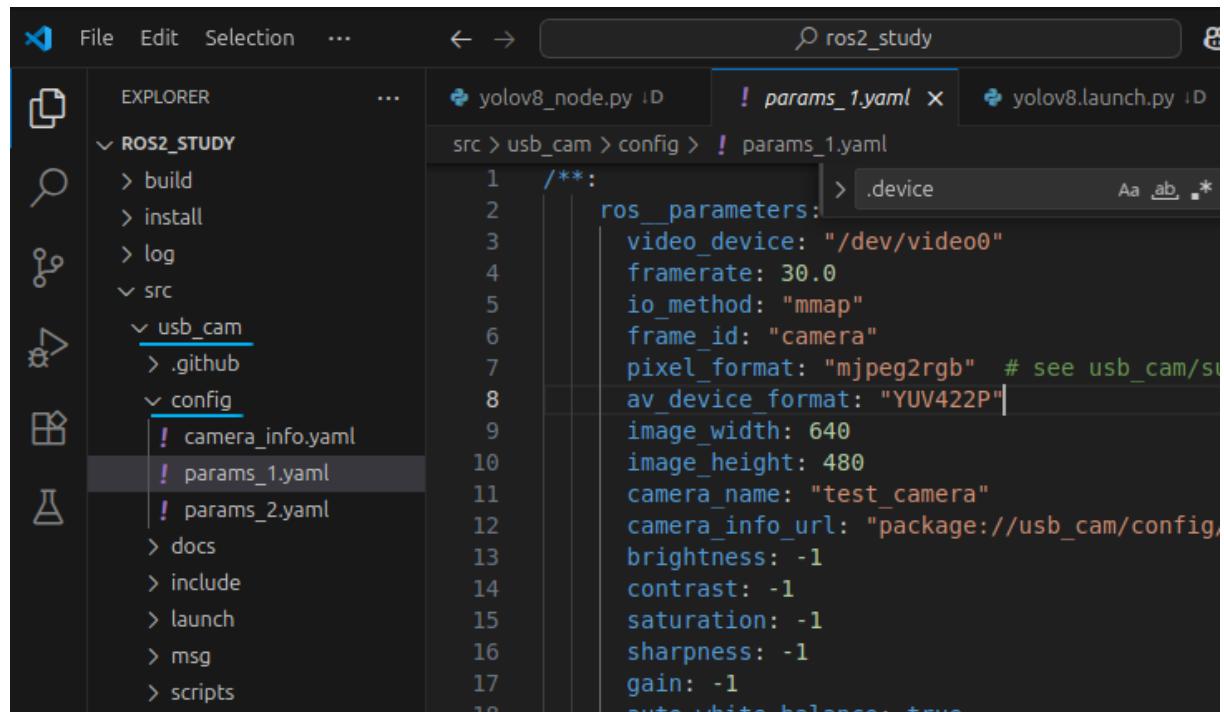
v4l2-ctl -d /dev/video0 --list-formats-ext 명령어를 치면 이런 식으로 화면이 나오는데

```
j@ros2: ~ v4l2-ctl -d /dev/video0 --list-formats-ext
ioctl: VIDIOC_ENUM_FMT
Type: Video Capture

[0]: 'MJPG' (Motion-JPEG, compressed)
    Size: Discrete 1280x720
        Interval: Discrete 0.033s (30.000 fps)
        Interval: Discrete 0.040s (25.000 fps)
        Interval: Discrete 0.050s (20.000 fps)
        Interval: Discrete 0.067s (15.000 fps)
        Interval: Discrete 0.100s (10.000 fps)
        Interval: Discrete 0.200s (5.000 fps)
    Size: Discrete 800x600
        Interval: Discrete 0.033s (30.000 fps)
        Interval: Discrete 0.040s (25.000 fps)
        Interval: Discrete 0.050s (20.000 fps)
        Interval: Discrete 0.067s (15.000 fps)
        Interval: Discrete 0.100s (10.000 fps)
        Interval: Discrete 0.200s (5.000 fps)
    Size: Discrete 640x480
        Interval: Discrete 0.033s (30.000 fps)
        Interval: Discrete 0.040s (25.000 fps)
        Interval: Discrete 0.050s (20.000 fps)
        Interval: Discrete 0.067s (15.000 fps)
        Interval: Discrete 0.100s (10.000 fps)
        Interval: Discrete 0.200s (5.000 fps)
    Size: Discrete 640x360
        Interval: Discrete 0.033s (30.000 fps)
```

컴퓨터에 연결된 카메라가 지원하는 픽셀 포맷(MJPEG 등)이나 해상도(640x480 등)를 보여주는 것인데 이 값이 아래 usb_cam 패키지를 쓸 때 config를 바꿔야 할 때 필요.

'launch/camera.launch.py' 를 보면 'config/params_1.yaml' 를 참조함.



암튼 개요는 ROS2의 1) **usb_cam** 패키지와

https://github.com/ros-drivers/usb_cam

2) **yolo_ros** 패키지를 함께 사용한다는 것이고

https://github.com/mgonzs13/yolo_ros

1) 에서 카메라로 찍는 이미지를 publish하면 2) 에서 subscribe 해서 detection 하는 방식

```
# 설치부터. ros2_ws 를 workspace라 했을 때

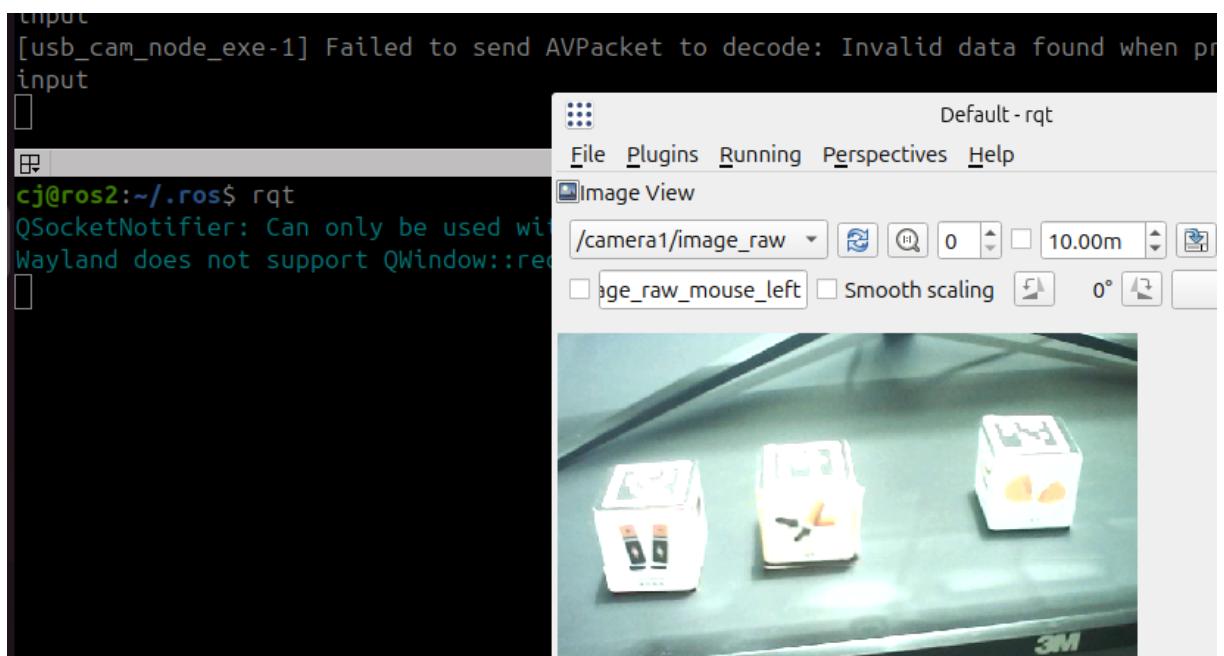
# 코드 가져오기
$ cd ros2_ws/src
$ git clone https://github.com/ros-drivers/usb_cam/
$ git clone https://github.com/mgonzs13/yolo_ros

# 의존성 설치
$ cd workspace
$ rosdep install --from-paths src --ignore-src -y -r

# 빌드 및 setup
$ cd workspace
$ colcon build
$ source workspace/install/setup.bash
```

- **usb_cam** 테스트를 위해 아래와 같이 launch 후 rqt로 image_raw를 보면

```
ros2 launch usb_cam camera.launch.py
```



(VMware에서 우분투를 돌린 거라 그런가? 심하게 버벅임. 강의장 컴퓨터는 잘 될 수도)

- 다음으로 YOLO를 쓰려면 ultralytics를 설치해야 하는데, 가상 환경에서 설치하면 ROS2가 인식을 잘 못하므로 밖에서 `pip install ultralytics` 를 해야 하는 걸로 보임.

- 근데 pip별로 요구하는 버전이 특정 돼 있으므로 그냥 다음 명령어를 실행하자

```
# requirements.txt 파일 안에 있는거 보고 pip 설치한다는 의미
pip3 install -r yolo_ros/requirements.txt
```

yolo_ros / requirements.txt

mgonzs13 upgrading lap dependency

Code	Blame	5 lines (5 loc) · 88 Bytes
1 numpy<2		
2 opencv-python>=4.8.1.78		
3 typing-extensions>=4.4.0		
4 ultralytics==8.3.91		
5 lap>=0.5.12		

- 집 컴퓨터는 GPU가 없는 관계로 device의 default 값 수정, 그리고 topic 이름 맞추기 위해 input_image_topic 부분은 /camera1/image_raw로

The screenshot shows two terminal windows side-by-side. Both windows are navigating through the directory structure of the YOLO ROS package, specifically within the 'yolo_ros' folder.

Left Terminal:

```
... > yolo_ros > yolo.launch.py > yolo_node.py > yoloLaunch.py > __init__.py
src > yolo_msgs > yolo.msg > yolo_node.py > yoloLaunch.py > __init__.py
41 from yolo_msgs.msg import Mask
42 from yolo_msgs.msg import KeyPoint2D
43 from yolo_msgs.msg import Detection
44 from yolo_msgs.msg import DetectionArray
45 from yolo_msgs.msg import SetClasses
from yolo_msgs.srv import SetClasses

class YoloNode(LifecycleNode):
    def __init__(self) -> None:
        super().__init__("yolo_node")
        # params
        self.declare_parameter("model_type", "YOLO")
        self.declare_parameter("model_file", "yolov8n.pt")
        self.declare_parameter("device", "cuda")
        self.declare_parameter("yolo encoder", "bg8")
        self.declare_parameter("enable", True)
        self.declare_parameter("image_reliability", 0.05)
        self.declare_parameter("threshold", 0.5)
```

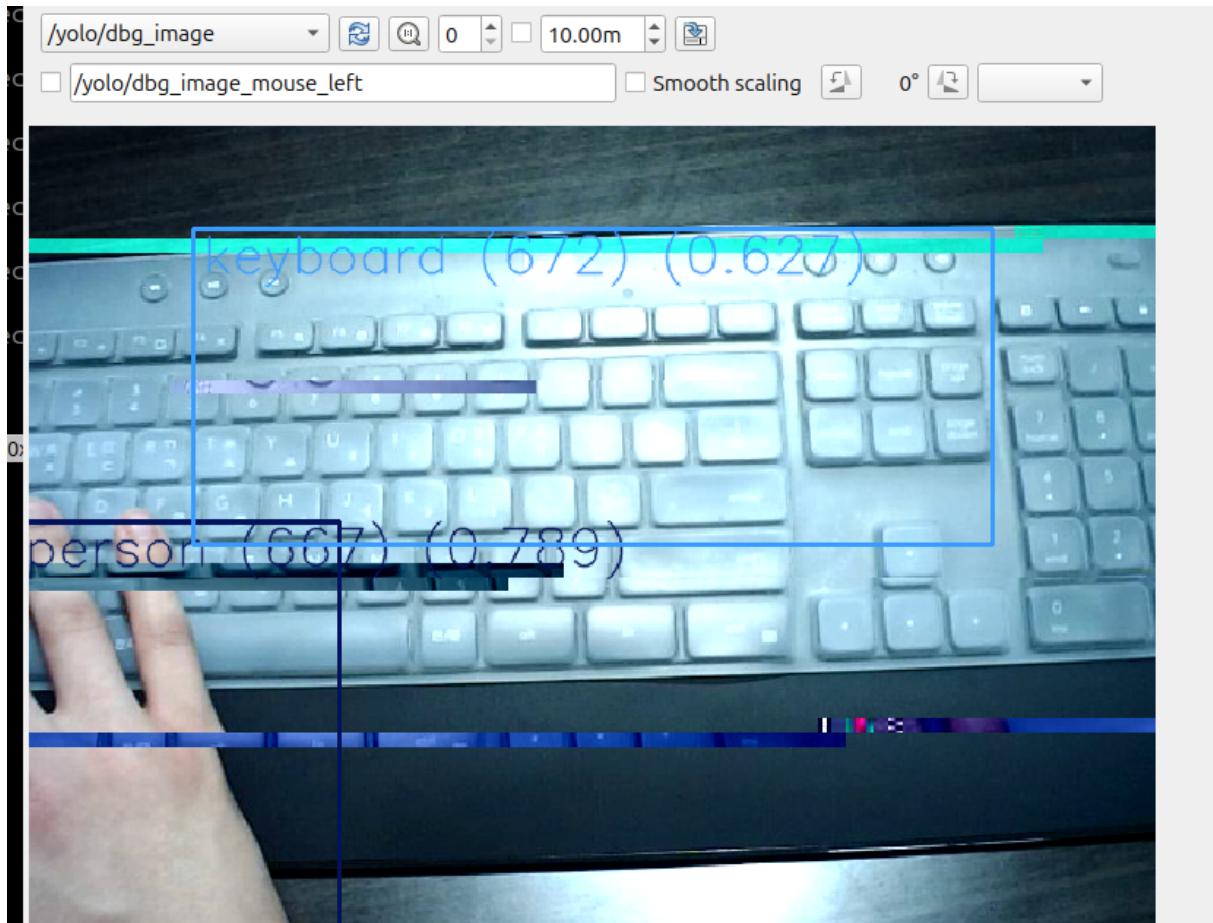
Right Terminal:

```
... > yolo.launch.py > yolo_node.py > yoloLaunch.py > generate_launch_description
src > yolo_ros > yolo_bridge > launch > yoloLaunch.py > __init__.py
30     def generate_launch_description() -> LaunchDescription:
31         return launch.actions.PythonAction(
32             name="yolo_launch",
33             python_executable="python",
34             python_file="yolo.launch.py",
35             arguments=[],
36         )
37         launch_arguments=[
38             LaunchConfiguration("model", default="yolov8n.pt"),
39             LaunchConfiguration("tracker", default="byteTracker"),
40             LaunchConfiguration("encoder", default="bg8"),
41             LaunchConfiguration("device", default="cuda"),
42             LaunchConfiguration("enable", default=True),
43             LaunchConfiguration("reliability", default=0.5),
44             LaunchConfiguration("topic", default="yolo"),
45             LaunchConfiguration("input_image_topic", default="/camera1/image_raw"),
46             LaunchConfiguration("output_image_topic", default="/camera1/image_raw"),
47         ),
48         ],
49         image_reliability=LaunchConfiguration("image_reliability", default="1"),
50         namespace=LaunchConfiguration("namespace", default="yolo"),
51     ).items()
```

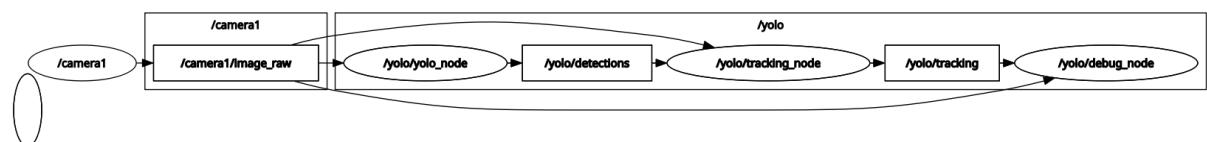
- 후에 colcon build, setup.bash 수행 후 usb_cam 해놓은 상태로 다음 실행하면

```
ros2 launch yolo_bringup yolov8.launch.py
```

rqt를 보면 손을 보고 person, 키보드 보고 keyboard라고 실시간 탐지하는 화면은 나옴



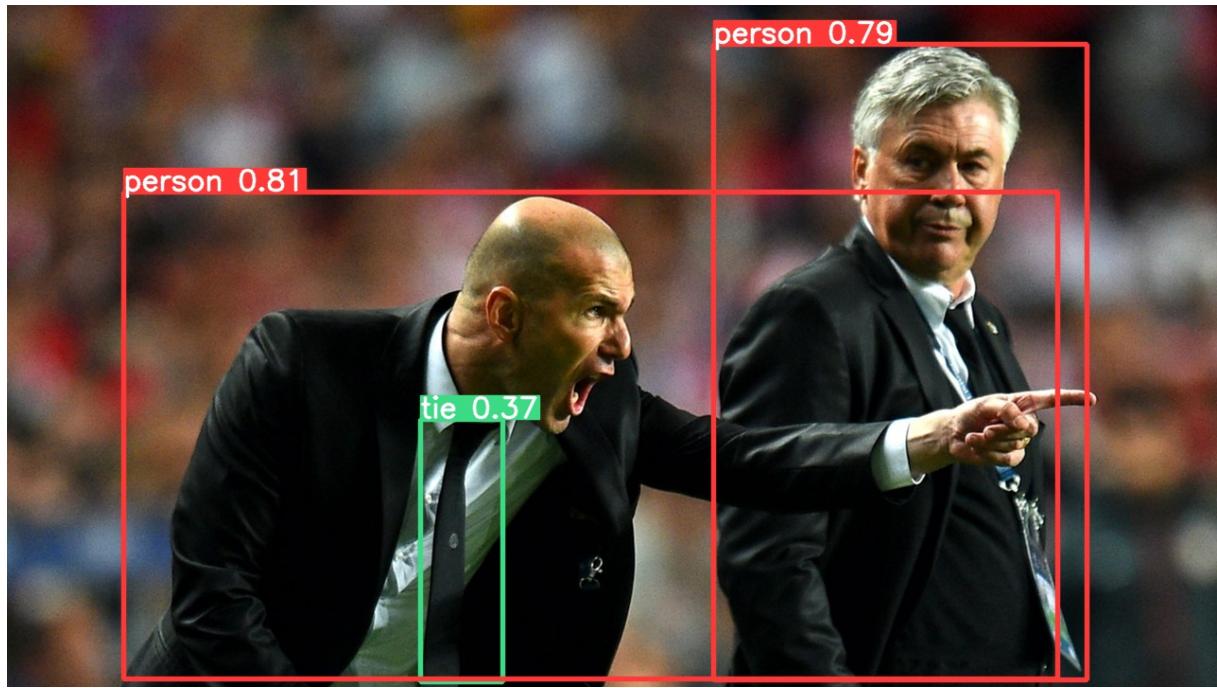
rqt_graph는 이런 상태



4.5.3.2 2. YOLO custom dataset 생성/학습

그리고 Yolo를 custom dataset으로 학습 시키는 방법도 알아둘 필요가 있음

우리의 경우는 아마 Pinky나, 매장 바닥에 떨어진 신발 박스 이런 걸로.



4 YOLO로 image detection 을 한 예시

이미지를 고르고 여기에 YOLO에서 박스 쳐 주는 것처럼 labeling 을 해야 하는데

여기선 **labelimg** 라는 툴이 대표적으로 쓰인다고 함

<https://github.com/HumanSignal/labelImg>

Albumentation 이라고 데이터 증식하는 툴도 있다고 함

https://github.com/venture21/yolov8_drone?tab=readme-ov-file

4.5.3.3 3. 노트북에 GPU 환경 설정

또 하나 지급받은 노트북에서 딥러닝용 환경 설정 필요해보임. 왜냐면

1 YOLO는 CNN 기반이라 계산량이 많음

- 추론도 convolution 연산이 대부분
 - CPU로는 1장 처리할 때 수백~수천 ms 소요됨
 - GPU는 같은 작업을 병렬로 처리해서 10~50배 빠름
-

2 실시간 처리하려면 FPS 확보가 필수

예를 들어 YOLOv5n 모델 기준으로 비교해볼게:

모델	해상도	CPU 추론 속도	GPU 추론 속도
YOLOv5n	640x480	200-500ms per frame (25 FPS)	~10-30ms per frame (30+ FPS)
YOLOv8n	640x480	~300-600ms per frame	~10-40ms per frame

- CPU만 쓰면 3 FPS 근처
 - GPU 쓰면 30 FPS 이상 가능
-

3 카메라 스트리밍에서는 프레임 지연 누적

- CPU만 쓰면 지연이 쌓여 “딜레이 카메라”처럼 밀림
- GPU는 거의 실시간으로 처리 가능

nvidia-driver - CUDA - cuDNN 가 세팅되어야 하고 서로간에 버전 설정도 중요한 부분인데,

방법은 아래 강의에서 확인 가능

https://www.youtube.com/watch?v=00u2Jj96nEA&list=PL0xYz_4oqpvffUTYPP09I3wjmczCAV9S&index=3

결론: 지금 받은 노트북에 이미 세팅되어 있을 것으로 예상. 아래 명령어를 치면 드라이버 버전이나 CUDA 버전, GPU 메모리 크기 등을 알 수 있다

```
addinedu@addinedu-Bravo-17-D7VF:/etc/modprobe.d$ nvidia-smi
Tue Jul  8 10:06:50 2025
+-----+
| NVIDIA-SMI 570.133.07      Driver Version: 570.133.07    CUDA Version: 12.8 |
+-----+
| GPU  Name           Persistence-M | Bus-Id     Disp.A  | Volatile Uncorr. ECC | | | |
| Fan  Temp   Perf   Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
| |          |          |              |             | MIG M. |
+-----+
| 0  NVIDIA GeForce RTX 4060 ... Off  00000000:01:00.0 On   N/A |
| N/A   51C   P8          5W /  80W |      70MiB /  8188MiB |  34%     Default |
| |          |          |              |             | N/A |
+-----+
+-----+
| Processes:
| GPU  GI CI          PID  Type  Process name          GPU Memory |
| ID   ID             ID   ID               Usage          |
+-----+
| 0  N/A N/A          2678   G  /usr/lib/xorg/Xorg          59MiB |
+-----+
```

다음 스크립트로도 확인 가능. 물론 (가상환경이 안전은 함) PyTorch를 pip로 설치해야 한다

```
import torch

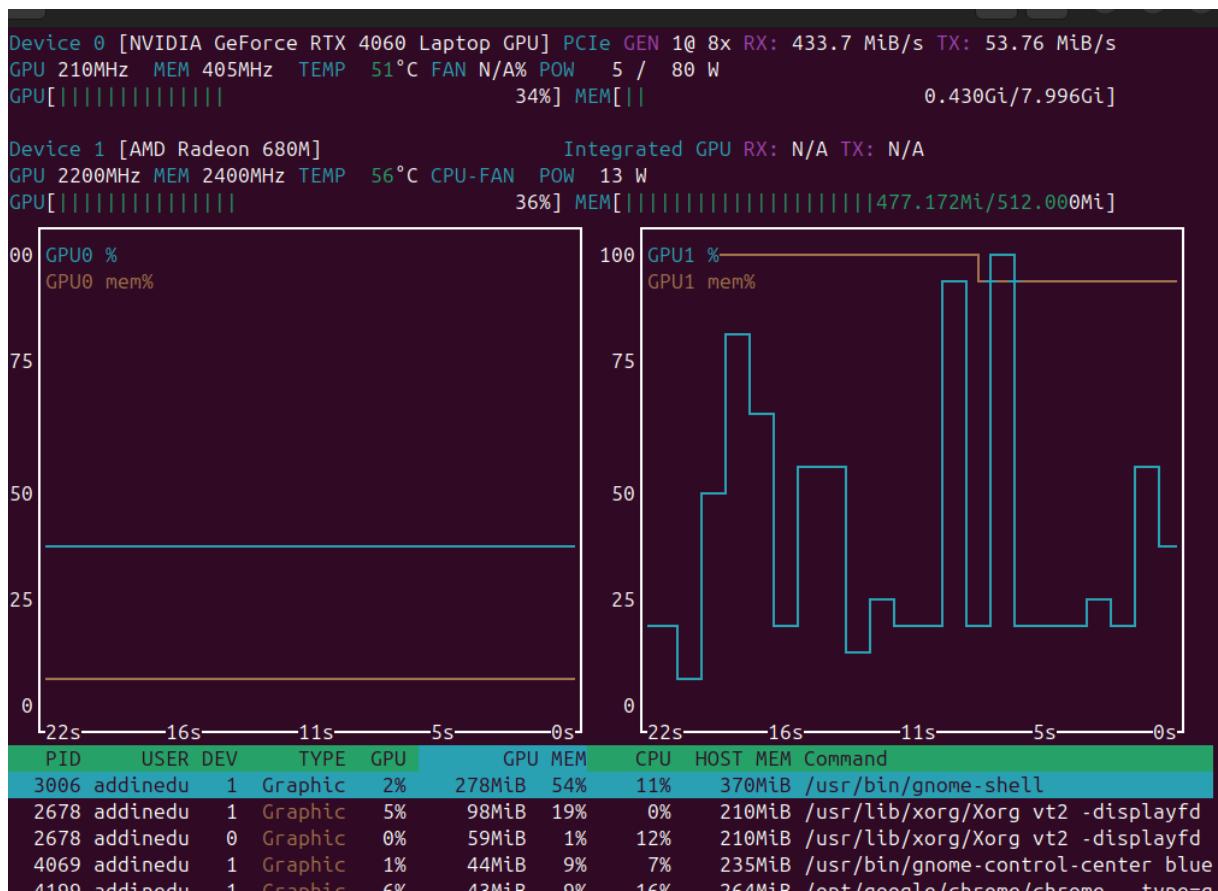
print("CUDA available:", torch.cuda.is_available())
print("cuDNN version:", torch.backends.cudnn.version())
print("cuDNN enabled:", torch.backends.cudnn.is_available())
```

- 하지만 그래도 세팅 과정을 한 번 요약해보자면

nvidia 공식 드라이버를 써야 하므로

1. nouveau라 하는 리눅스 제공 드라이버를 비활성화
2. 초기 파일 시스템 업데이트 후 재부팅
3. 기존 **nvidia 드라이버** 지우고 새로 **설치**
4. sudo apt-mark hold 명령어로 **특정 드라이버 업데이트 막아두기**
5. **CUDA, cuDNN 설치**

정도가 되겠다. 강의 말미에 알았는데 nvtop 이란 명령어도 있음



4.5.3.4 (4. 모터 연동해서 제어하기?)

4.5.4 ArUco marker + RPi 카메라 + Flask (작성중)



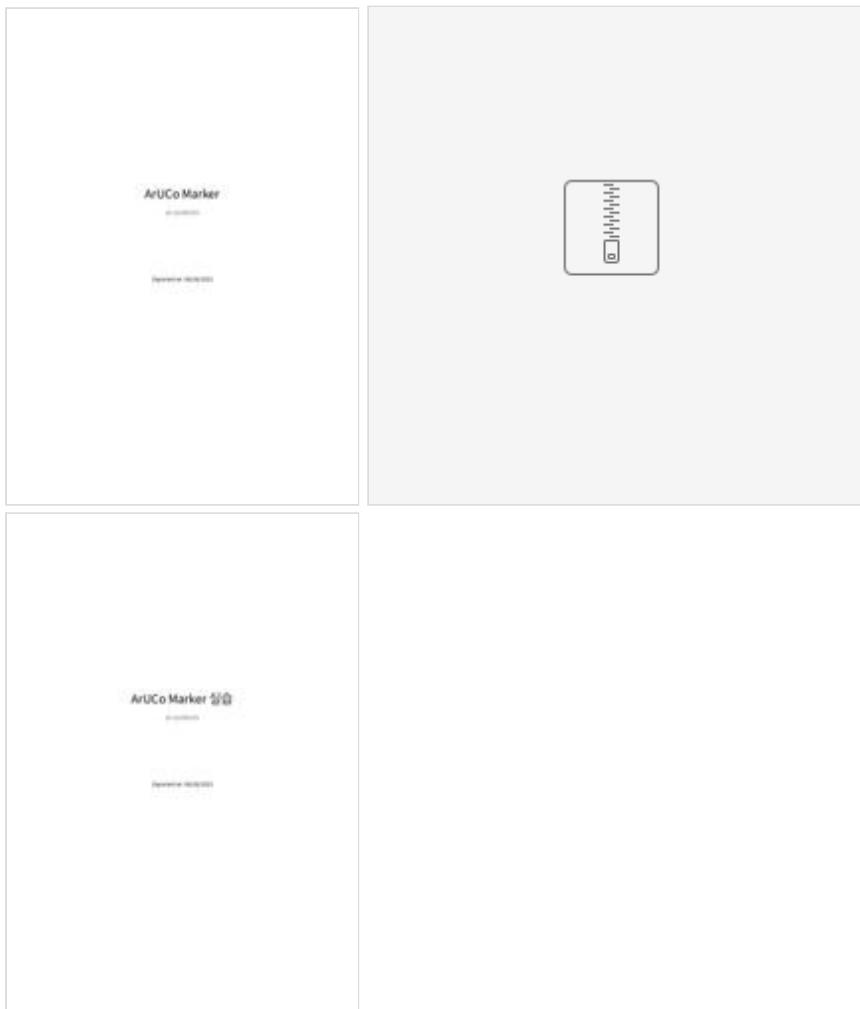
이찬종/ 라즈베리파이 상 카메라, Flask, GPIO 등

- ArUco Marker

파이썬 버전 변화에 따라 함수 달라진 부분 유의해야 함

pinkwink 블로그

<https://pinkwink.kr/1427> pdf도 있음



<https://github.com/GSNCodes/ArUCo-Markers-Pose-Estimation-Generation-Python> 여긴 calibration 위한 intrinsic matrixs, distortion coefficients 구해주는 코드도 제공

<https://iambeginnerdeveloper.tistory.com/231>

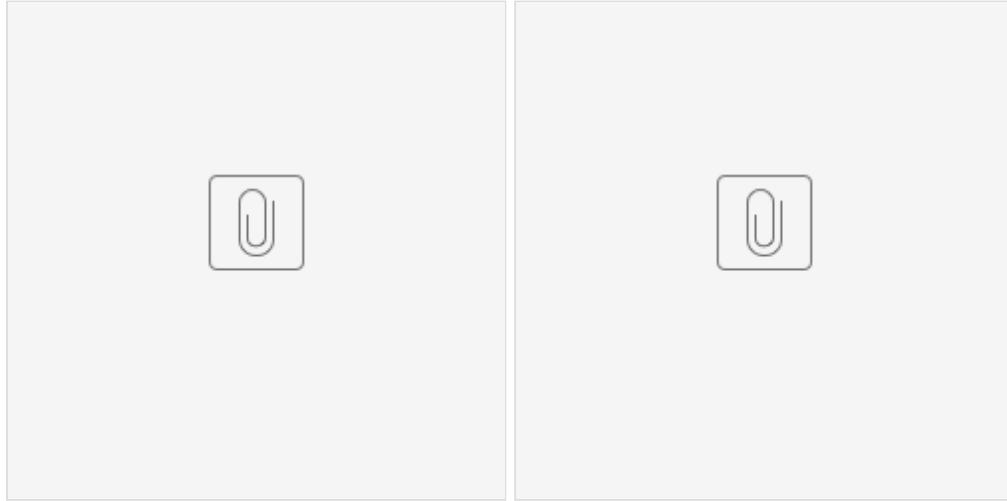
<https://robogony.tistory.com/entry/OpenCV-2-%EC%95%84%EB%A3%A8%EC%BD%94%EB%A7%88%EC%BB%A4-%EC%9D%B8%EC%8B%9D%ED%95%98%EA%B8%B0-ArUco-marker>

<https://yehjin00.tistory.com/entry/%EB%A7%88%EC%BB%A4%EC%9D%B8%EC%8B%9D%ED%95%B4%EC%84%9C-%EB%A1%9C%EB%B4%87-%EC%9C%84%EC%B9%98-%EC%A1%B0%EC%A0%95%ED%95%98%EA%B8%B0>

- 이건 더 자세히 알고 싶을 경우

https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

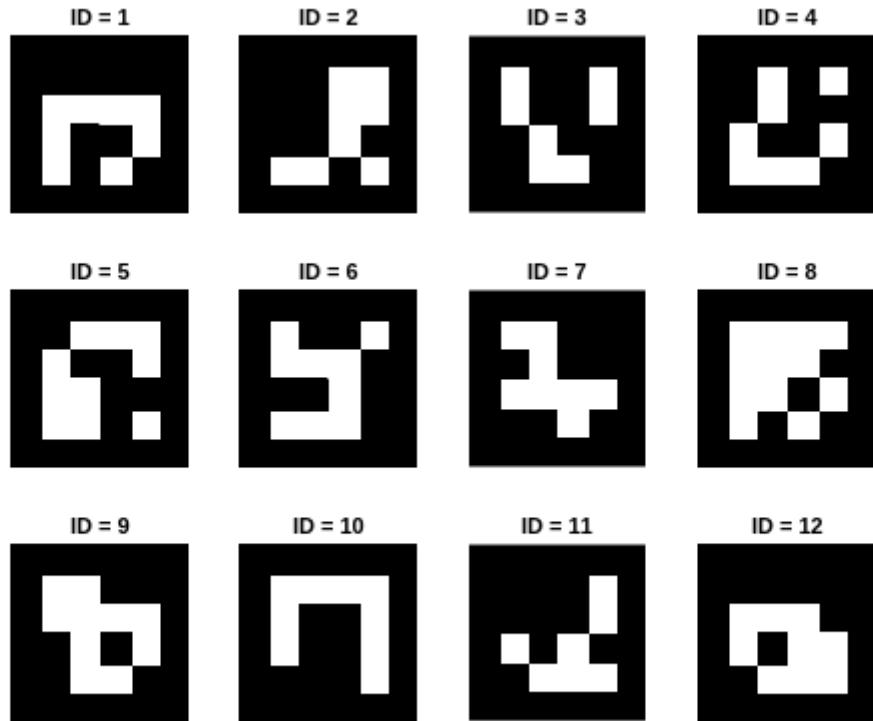
코드는 파일로 올리고 (여기 .npy 파일 의미음)



[practice.ipynb¹¹](#)

- 마커 이미지 생성 및 검출

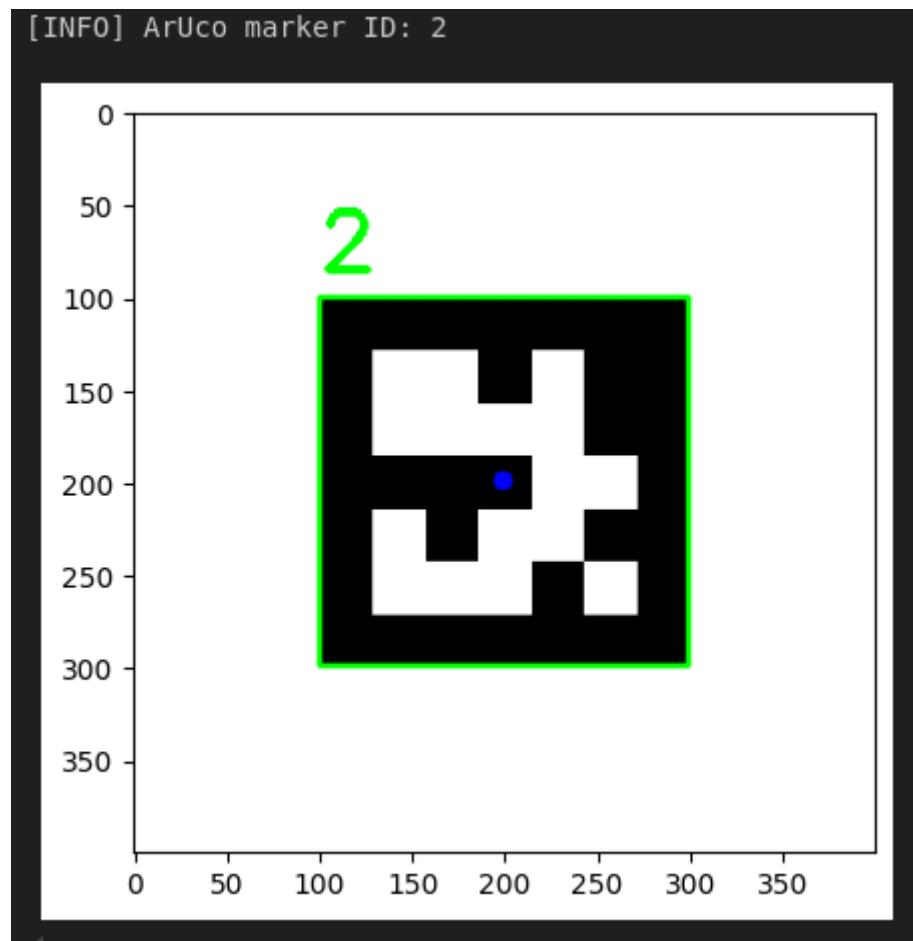
11. [https://qkrtmddn0507.atlassian.net/wiki/download/attachments/3735909/practice.ipynb?
api=v2&cacheVersion=1&modificationDate=1751903680197&version=1](https://qkrtmddn0507.atlassian.net/wiki/download/attachments/3735909/practice.ipynb?api=v2&cacheVersion=1&modificationDate=1751903680197&version=1)



5 아루코 마커란? DICT_4X4_50 or 100 or ...

어떤 마커인지를 잘 detection이 되려면 아래처럼 여백이 있는 이미지를 주어야 함

여백 없이 생성한 마커 자체를 읽으면 탐지 못 함 (이것 땜에 뺄짓을 몇 시간...)



6 Type = 'DICT_5X5_100'에서 2번 ID 가진 marker 검출 화면

- Calibration도, 어떤 type인지도 모른 채로 해서 그런지 뭐가 뜨진 않는데

```

# 마커의 3D 위치 표시
pos_x = tvecs[i][0][0]
pos_y = tvecs[i][0][1]
pos_z = tvecs[i][0][2]

# 회전 벡터를 오일러 각도로 변환
rot_matrix, _ = cv2.Rodrigues(rvecs[i])
euler_angles = cv2.RQDecomp3x3(rot_matrix)[0]

# 마커 정보 표시
corner = corners[i][0]
center_x = int(np.mean(corner[:, 0]))
center_y = int(np.mean(corner[:, 1]))

cv2.putText(frame_undistorted,
            f"ID: {ids[i][0]}",
            (center_x, center_y - 40),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5, (0, 0, 0), 2)

cv2.putText(frame_undistorted,
            f"Pos: ({pos_x:.2f}, {pos_y:.2f}, {pos_z:.2f})m",
            (center_x, center_y),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5, (0, 0, 0), 2)

cv2.putText(frame_undistorted,
            f"Rot: ({euler_angles[0]:.3f}, {euler_angles[1]:.3f}, {euler_angles[2]:.3f})deg",
            (center_x, center_y + 40),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5, (0, 0, 0), 2)

```

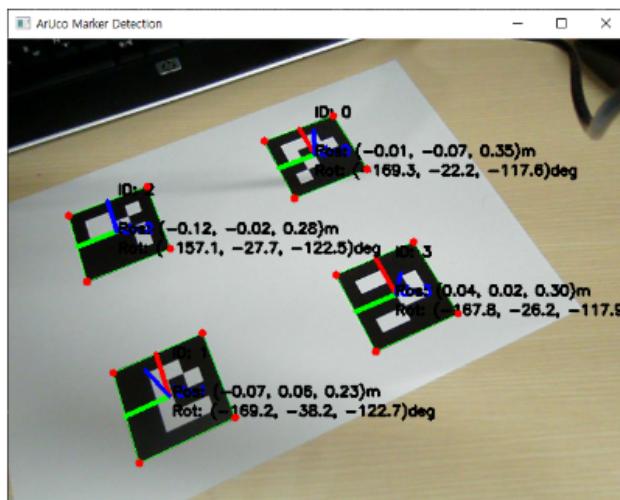
7 그래도 YOLO 때와 달리 VMware 밖에서 하니 화면은 잘 나옴

- 원래는 마커 한 변의 길이도 집어넣을 수 있기 때문에 카메라로부터의 거리(cm) 및 각도(degree) 도 구할 수 있는 것으로 보임

robogony.tistory.com/entry/OpenCV-2-아루코마커-인식하기-ArUco-marker

아루코마커를 통한 위치 추정

앞서 말했듯이 우리는 아루코마커를 통해 위치와 각도를 추정할 수 있다. 인식한 아루코마커의 위치와 각도에 따라서 x, y, z축 방향으로의 위치와 회전 각도를 계산할 수 있다.



아루코마커를 통한 위치 추정

여기는 필요하면 나중에 작성 다시...?

- Pinky 카메라 송출 ([pinky_dfc6에서 진행](#))

https://www.youtube.com/watch?v=t0QByjex3b8&list=PL0xYz_4oqpvidCm57CYbWlj-hFbMdJuhi&index=13

https://github.com/PinkWink/for_ROS2_study/blob/main/RPi%20Study/pycam_install.md

우선 위 영상대로 하는 것이 안전해보임

하지만 사실 세팅은 이미 되어있었다..! ← 첫 주차 Pink06 LCD 관련 자료 참고

- Flask 사용
- 간단 실습

서버에서 (이 경우는 pinky_20f0) **flask_test.py** 실행 후 클라이언트 (노트북)에서 **flash_practice.ipynb** 실행하면 아래와 같은 화면. ip 주소는 pinky에서 ifconfig 후 wlan0:에서 나오는 주소를 사용

The screenshot shows a terminal window at the top displaying a sequence of 'Counter updated' messages from 981 to 1004. Below it is a Jupyter Notebook interface. The notebook has a single cell containing Python code. The output of this cell shows four requests to a URL, each printing the current value of the counter (534, 535, 536, 539) before sleeping for one second.

```

[2]: import requests
[6]: server_ip = '192.168.0.222'
      url = f'http://[{server_ip}]:5000/counter'
[8]: url
[8]: 'http://192.168.0.222:5000/counter'
[7]: import time
for _ in range(4):
    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    print('현재 카운터 값:', data.get('counter'))
    time.sleep(1)

```

현재 카운터 값: 534
현재 카운터 값: 535
현재 카운터 값: 536
현재 카운터 값: 539

코드 설명은 아래 참조 (설명 미완)

```

# flask_test.py 이 경우 pinky에서 실행, 서버

from flask import Flask, jsonify
import threading
import time

```

```

app = Flask(__name__)
counter = 0

def counter_updater():
    global counter
    while True:
        time.sleep(1)
        counter += 1
        print(f'Counter updated: {counter}')


@app.route('/counter')
def get_counter():

    return jsonify({'counter': counter})

if __name__ == '__main__':
    updater_thread = threading.Thread(target=counter_updater, daemon=True)
    updater_thread.start()

    app.run(host = '0.0.0.0', port=5000)

```

```

# flask_practice.ipynb 노트북에서 실행, 클라이언트

import requests
import time

server_ip = '192.168.0.222'
url = f'http://{server_ip}:5000/counter'

for _ in range(4):
    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    print('현재 카운터 값:', data.get('counter'))

    time.sleep(1)

```

- GPIO로 초음파 센서 값 받아오기

4.5.5 Visual C++, about 비트맵



이찬종/영상 처리 관련

- 비트맵 (bitmap)은 곧 비트로 그린 맵.

컴퓨터에서 영상을 표현하는 대표적 방법으로 특정 비트 조합이 특정 색상에 매핑되는 방식

3 3 3 3 3 3 3 3 3		0 000000	█
0 1 4 1 4 1 4 0		1 FF0000	█
0 4 1 4 1 4 1 0		2 00FF00	█
0 5 5 5 5 5 5 0		3 0000FF	█
0 5 5 5 5 5 5 0		4 FFFFFF	█
0 1 4 1 4 1 4 0		5 FFFF00	█
0 4 1 4 1 4 1 0		6 FF00FF	█
2 2 2 2 2 2 2 2		7 00FFFF	█
		8 FF0080	█
		9 FF8040	█
		A 804000	█
		B 008080	█
		C 800000	█
		D 800080	█
		E 8080FF	█



8 <https://learn.microsoft.com/en-us/windows/win32/gdiplus-gdiplus-types-of-bitmaps-about>

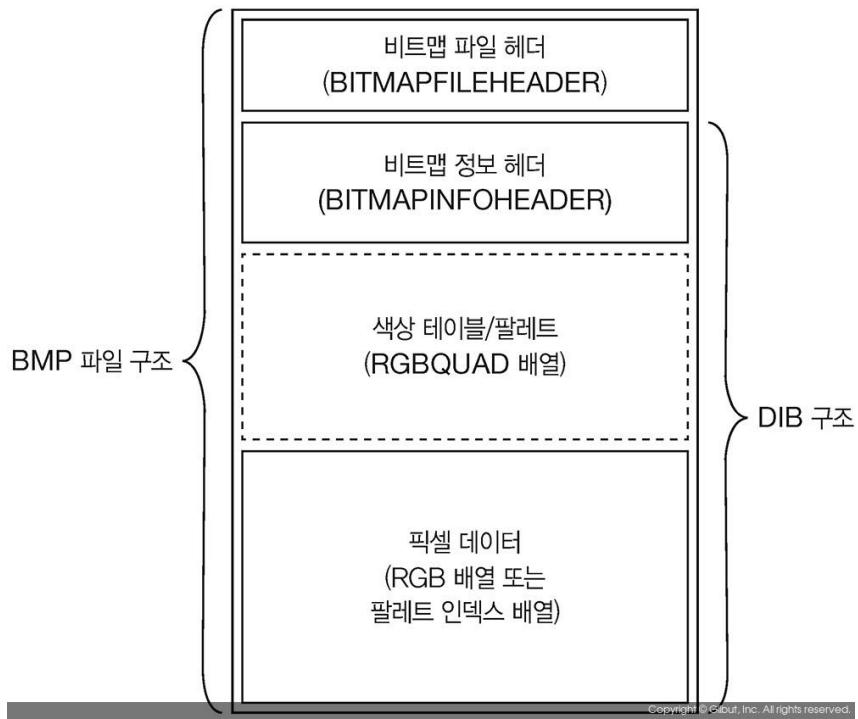
압축이 없기 때문에 분석이 용이하지만 메모리 용량을 많이 차지

비트맵 방식으로 DIB(Device Independent Bitmap)이 있는데 모니터와 같은

출력 장치에 상관없이 비트맵이 제대로 출력되도록 할 수 있음.

널리 알려진 JPG, GIF, PNG 등의 포맷도 화면에 출력되기 위해 DIB로 전환해야 함.

- BMP 파일의 구조는 다음과 같다



⁹ <https://thebook.io/006796/0052/>

맨 위의 file header는 파일의 크기와 같은 파일 자체 정보를,

```
typedef struct tagBITMAPFILEHEADER {
    WORD      bfType;
    DWORD     bfSize;
    WORD      bfReserved1;
    WORD      bfReserved2;
    DWORD     bfOffBits;
} BITMAPFILEHEADER, FAR *LPBITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

10 여기서 bfOffBits는 픽셀 데이터 시작 위치 오프셋

다음 info header는 해상도, 압축 방법 등 이미지/영상 관련 정보를 뜻함.

```

typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER, FAR *LPBITMAPINFOHEADER, *PBITMAPINFOHEADER;

```

11 여기서 biWidth, biHeight는 픽셀 단위. biHeight의 경우 양수면 뒤집힌 상태(180도 회전 말고) 인데 일반적으로 이 상태로 저장됨

팔레트(=color table)은 방식에 따라 없을 수도 있는데 트루컬러 영상을 포함하는 BMP 파일은 팔레트 없이 각 픽셀에 대해 BGR 색상값이 3바이트로 나타나는 방식

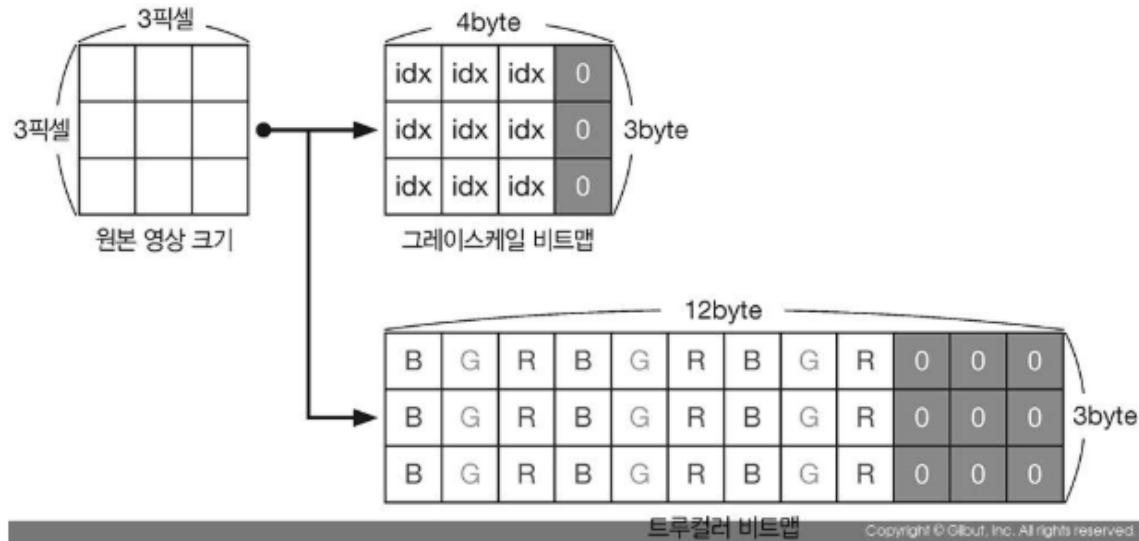
```

typedef struct tagRGBQUAD {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;

```

픽셀 데이터 영역에 저장될 때는 효율적인 데이터 관리를 위해 가로 크기 4의 배수 유지

▼ 그림 3-3 3x3 크기 영상의 비트맵 픽셀 데이터 정렬 방식



12 꼭 0은 아니고 안 쓰이는 임의의 값이 할당

4.5.6 Anaonda 사용법

(중간에 시행착오가 담겨 있어서 한 번 짹 읽고 따라하는 것을 추천)

- venv, virtualenv랑 비슷하게 가상환경 만들고 패키지 설치 편하게 해주는 도구. ML, 과학 계산에 많이 쓰이고 의존성 관리가 잘 되어있다는 것, 또 C/C++ 라이브러리도 설치 관리가 가능하다는 게 특징.
- lscpu 명령어로 본인 컴퓨터의 cpu를 확인 (x86인지 ARM64인지 이런 거) 후 설치
<https://www.anaconda.com/download/success>

기존 설치된 파이썬과의 충돌 우려도 있고 해서 간단히 사용법 등을 정리해보고자 함

- 설치하다 보면 이런 게 뜨는데, 뭔 소리냐면 터미널 열 때마다 자동으로 conda 가상환경 켜게끔 설정할 거냐 묻는 것이고 우리 환경에선 그냥 필요할 때 'conda init'하는 게 편할 거 같아서 no로 진행

```
Do you wish to update your shell profile to automatically initialize conda?
This will activate conda on startup and change the command prompt when activated.
If you'd prefer that conda's base environment not be activated on startup,
run the following command when conda is activated:

conda config --set auto_activate_base false

You can undo this by running `conda init --reverse $SHELL`? [yes|no]
[no] >>> [ ]
```

✓ 3 macOS / Linux에서는

- 설치 시 `conda init` 이 헬 설정 파일(`.bashrc` , `.zshrc`)을 수정
 - 로그인 시 `conda base` 환경이 자동 활성화되도록
 - `python` → Anaconda base의 파이썬이 됨
- 기존 Python이랑 PATH 충돌 가능:
 - 기본 `python` (`/usr/bin/python`, `brew` 등) 대신 `conda base` 환경의 `python`이 잡힘
- 해결:
 - `.bashrc` 나 `.zshrc` 에서 `conda init` 지우면 PATH 영향 제거
 - 필요할 때만 `conda activate`

그런데 `conda` 명령어 자체를 인식 못해서.. `~/.bashrc` 에 다음 커맨드 추가

```
source ~/anaconda3/etc/profile.d/conda.sh
```

한 다음 `conda init`을 하면 결국 아래 스크립트가 `.bashrc`에 추가돼버려서

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup=$(('/home/addinedu/anaconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)
if [ $? -eq 0 ]; then
  eval "$__conda_setup"
else
  if [ -f "/home/addinedu/anaconda3/etc/profile.d/conda.sh" ]; then
    . "/home/addinedu/anaconda3/etc/profile.d/conda.sh"
  else
    export PATH="/home/addinedu/anaconda3/bin:$PATH"
  fi
fi
unset __conda_setup
# <<< conda initialize <<<

conda deactivate
```

`conda deactivate` 를 막줄에 추가했음.

`conda` 명령어 자주 안 쓸거면 위 스크립트는 지워도 될듯함 (왜냐면 터미널이 좀 늦게 떠서)

- 간단 사용법

- 원하는 가상환경 켜기/끄기 : `conda activate [가상환경이름]` / `conda deactivate`
- 가상환경 만들기 : `conda create -n [환경이름] python=3.xx`

- 가상환경 삭제 : conda remove -n [환경이름] --all
- 만들어 둔 패키지/환경 목록 보기 : conda list / conda env list
- 패키지 설치/삭제/업그레이드 : conda install [패키지명]=[버전.xx] / conda remove [패키지명]
conda update [패키지명]

- 설치할 패키지를 가져올 channel을 지정하는 것이 가능

- conda install -c [채널명] [패키지명]

주요 채널로 defaults, conda-forge, bioconda 등이 있음

4.6 서비스 기능

4.6.1 음성 명령 with SST, GPT API (작성 예정 아마 ~ 7/13?)

아래와 같은 흐름으로 로봇에게 음성 명령이 가능해짐

1. 음성을 텍스트로 변환
2. 텍스트를 LLM에 넘겨 액션 시퀀스로 변환 (json 형태)
3. 생성된 액션 시퀀스로 실제로 로봇을 동작

- Speech To Text 는 말 그대로 음성 → 텍스트

온라인 오프라인 STT 모델이 몇 개 있음.

만약 싶는다면 그 대상은 Pinky일테니 오프라인으로 생각하고...

이전 기수 RoboDine 의 경우 오프라인 OpenAI Whisper 사용. Vosk라는 속도가 좀 더 빠른 모델도 있지만 정확도 측면에서 **Whisper**가 우위

'pip install openai-whisper sounddevice'로 시작

```
ROS2 workspace/src로 가서
$ ros2 pkg create stt_whisper --build-type ament_python
$ cd workspace
$ colcon build --packages-select stt_whisper
```

후에 workspace/src/stt_whisper/stt_whisper.py라는 파일을 만들고

다음 내용이 포함된 코드를 작성

```

import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import whisper

class WhisperNode(Node):
    ...

    def myFun(self):
        # whisper 모델 불러오기.
        self.model = whisper.load_model("small")

        # audio_file은 음성 녹음 파일
        result = self.model.transcribe(audio_file, language='ko')['text']
        ...

        # 퍼블리시
        msg = String()
        msg.data = result
        self.publisher_.publish(msg)
        ...
    ...

```

다음 명령어들로 .wav 파일로 녹음 및 실행이 가능, audio_file = "/home/cj/stt_test.wav" 임

```

(stt) cj@ros2:~$ arecord -f S16_LE -r 16000 -c 1 ~/stt_test.wav
Recording WAVE '/home/cj/stt_test.wav' : Signed 16 bit Little Endian, Rate 16000
Hz, Mono
^CAborted by signal Interrupt...
(stt) cj@ros2:~$ aplay stt_test.wav
Playing WAVE 'stt_test.wav' : Signed 16 bit Little Endian, Rate 16000 Hz, Mono

```

setup.py 고쳐 주고, colcon build 그리고 setup.bash

The screenshot shows the VS Code interface with the Explorer and Editor tabs. The Explorer tab displays a file tree for 'ROS2_STUDY' containing '.vscode', 'build', 'install', 'log', 'src' (which contains 'stt_whisper' with files '_init_.py', 'my_whisper.py', 'resource', 'test', 'package.xml', 'setup.cfg', and 'setup.py'), 'usb_cam', 'yolo_ros', and 'yolov8m.pt'. The Editor tab shows the 'setup.py' file with the following code:

```

1  from setuptools import find_packages, setup
2
3  package_name = 'stt_whisper'
4
5  setup(
6      name=package_name,
7      version='0.0.0',
8      packages=find_packages(exclude=['test']),
9      data_files=[
10          ('share/ament_index/resource_index/packages',
11           ['resource/' + package_name]),
12          ('share/' + package_name, ['package.xml']),
13      ],
14      install_requires=['setuptools'],
15      zip_safe=True,
16      maintainer='cj',
17      maintainer_email='cj@todo.todo',
18      description='TODO: Package description',
19      license='TODO: License declaration',
20      tests_require=['pytest'],
21      entry_points={
22          'console_scripts': [
23              'my_whisper = stt_whisper.my_whisper:main',
24          ],
25      },
26 )

```

가상 환경 끄고 해야 됐음. 밑줄 친 내용이 녹음한 음성 (정확히는 1,2,3 말고 하나,둘,셋)

아마 CPU로 해서 더 오래 걸리지 않았나 싶다

```

cj@ros2:~/ros2_study$ ros2 run stt_whisper my_whisper
100% [██████████] | 461M/461M [00:14<00:00, 34.0MiB/s]
[INFO] [1751811576.347844743] [whisper_node]: Whisper 모델 로드 완료!
/home/cj/.local/lib/python3.12/site-packages/whisper/transcribe.py:132: UserWarning: FP16 is not supported on CPU; using FP32 instead
    warnings.warn("FP16 is not supported on CPU; using FP32 instead")
[INFO] [1751811579.717715801] [whisper_node]: Whisper: 마이크 테스트 1.2.3
[INFO] [1751811579.718093480] [whisper_node]: Whisper 변환 시간: 3.37초

```

- 다음은 LLM에게 부탁하여 액션 sequence로 바꾸기

- 마지막으로 액션 시퀀스에 따라 로봇 움직이기

4.7 강의자료/튜토리얼

4.7.1 Jecobot Rviz 환경구축

<https://automaticaddison.com/create-and-visualize-a-robotic-arm-with-urdf-ros-2-jazzy/>

https://github.com/automaticaddison/mycobot_ros2/blob/jazzy/mycobot_description/urdf/sensors/intel_rgbd_cam_d435.urdf.xacro

```
ros2 launch urdf_tutorial display.launch.py model:=/home/addinedu/mycobot_ws/src/
mycobot_ross/mycobot_description/urdf/robots/mycobot_280.urdf.xacro
```

```
source install/setup.bash
```

4.7.2 ROS2 humble 공부 자료

<https://jungwonpark-robotics.notion.site/9cae9481f0024fc788872c8beea7f169>

4.7.3 Tutorials - ROS Jazzy

https://www.notion.so/turtlesim-node-222149491b9280818857eb441100b6d8?source=copy_link

https://www.notion.so/Client-libraries-222149491b9280ffba84c818de6f6eba?source=copy_link

```
import cv2
import numpy as np
import requests
import threading
import time

class VideoStreamClient:
    def __init__(self, server_url):
        """
        비디오 스트림 클라이언트 초기화
        Args:
            server_url (str): 서버 스트림 URL (예: 'http://192.168.5.1:5000/stream')
        """
        self.server_url = server_url
        self.stream_active = False
        self.stream_thread = None
        self.latest_frame = None
        self.frame_lock = threading.Lock()
```

```

def start_stream(self):
    """스트림 수신 시작"""
    if self.stream_active:
        print("이미 스트림이 활성화되어 있습니다.")
        return
    self.stream_active = True
    self.stream_thread = threading.Thread(target=self._receive_stream)
    self.stream_thread.daemon = True
    self.stream_thread.start()

def stop_stream(self):
    """스트림 수신 중지"""
    self.stream_active = False
    if self.stream_thread:
        self.stream_thread.join(timeout=1.0)

def get_latest_frame(self):
    """최신 수신된 프레임 반환"""
    with self.frame_lock:
        if self.latest_frame is None:
            return None
        return self.latest_frame.copy()

def _receive_stream(self):
    """비디오 스트림 수신 및 처리하는 내부 메서드"""
    try:
        # 스트림에 연결
        response = requests.get(self.server_url, stream=True)
        if response.status_code != 200:
            print(f"서버 연결 실패: {response.status_code}")
            self.stream_active = False
            return

        # MJPEG 스트림의 바이트 데이터
        bytes_data = bytes()

        # 스트림 처리 루프
        for chunk in response.iter_content(chunk_size=1024):
            bytes_data += chunk
            a = bytes_data.find(b'\xff\xd8')  # JPEG 시작 바이트
            b = bytes_data.find(b'\xff\xd9')  # JPEG 끝 바이트
            if a != -1 and b != -1:
                # 완전한 JPEG 프레임 찾음
                jpg = bytes_data[a:b+2]
                bytes_data = bytes_data[b+2:]
                # JPEG 바이트 데이터를 이미지로 디코딩
                frame = cv2.imdecode(np.frombuffer(jpg, dtype=np.uint8),
cv2.IMREAD_COLOR)
                # 최신 프레임 저장
                with self.frame_lock:
                    self.latest_frame = frame

                # 스트림이 비활성화되면 종료
                if not self.stream_active:
                    break
    
```

```

        except Exception as e:
            print(f"스트림 수신 중 오류 발생: {e}")
    finally:
        self.stream_active = False

# 사용 예시
if __name__ == "__main__":
    # 서버 URL (서버의 IP 주소로 변경 필요)
    server_url = "http://192.168.5.1:5000/stream"
    # 클라이언트 생성 및 시작
    client = VideoStreamClient(server_url)
    client.start_stream()

    # 프레임 표시 루프
    try:
        while True:
            frame = client.get_latest_frame()
            if frame is not None:
                cv2.imshow("Jetcobot Camera Stream", frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
    except KeyboardInterrupt:
        print("종료 중...")
    finally:
        client.stop_stream()
        cv2.destroyAllWindows()

```

4.7.4 핑크랩 강의목록

https://www.youtube.com/@pinklab_studio/playlists

이하는 예상 습득 기술

0. ROS2 입문 과정

- 각종 환경 세팅
- topic, service, action 을 turtle로 실습 (+ rclpy)

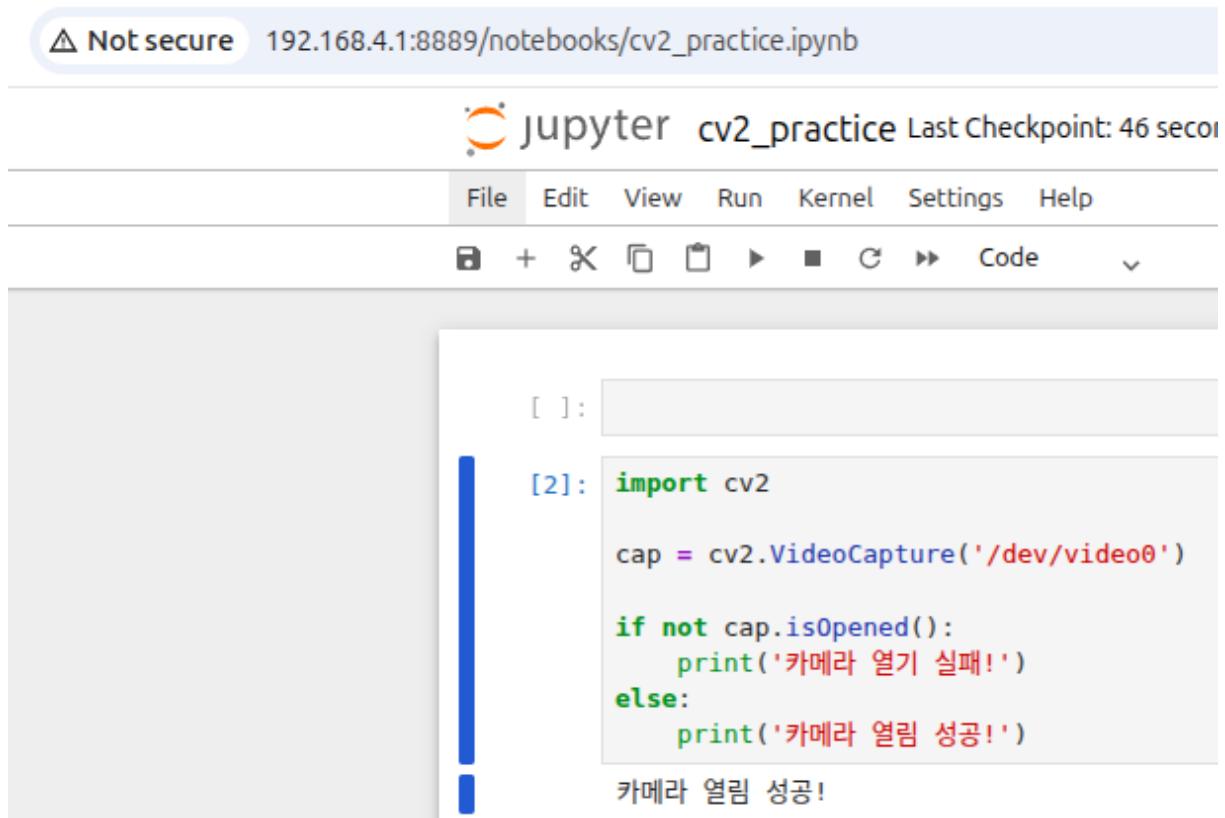
1. ROS2 응용 과정

- Package, Node 만들기
- 서비스/액션 서버 응용, 멀티 쓰레딩 사용
- 파라미터 다루기, launch, log, RQT

2. ROS2 실전 과정

- 터미널 더 편하게 쓰기. ex) 터미널마다 ROS_DOMAIN_ID 띄워주기

OpenCV 토픽 발행



The screenshot shows a Jupyter Notebook interface with the title 'cv2_practice.ipynb'. The code cell [2] contains the following Python script:

```

[2]: import cv2
cap = cv2.VideoCapture('/dev/video0')
if not cap.isOpened():
    print('카메라 열기 실패!')
else:
    print('카메라 열림 성공!')

```

The output of the code is displayed below the cell:

```

카메라 열림 성공!

```

jetcobot 뿐만 아니라 pinky_dfc6 의 경우도 위와 같이 접근 가능..한데 cap.read()는 바로 안 됨

https://blog.naver.com/idea_robot/223469882209

여기처럼 libcamera 라이브러리를 사용하거나

<https://forums.raspberrypi.com/viewtopic.php?t=381571>

처럼 Picamera2 같은 걸 중간에 써야 할 듯 한데 일단 보류.

- TF 브로드캐스팅
- ROS DOMAIN BRIDGE 사용
- PID 제어, QT로 제어기 모니터링

3. 공짜술

- YOLO, ChatGPT API
- Ollama 로 LLM
- GPU 환경 설정 (CUDA/cuDNN)

4. 산딸기 기초 정복

- Flask(서버), GPIO (, serial 은 좀 덜 중요할 수도?)

등등

5 Sprint 기록

5.1 <RT-18> <곽제우> <traffic manager(멀티 로봇 경로 계획) 설계> <7/15>

<https://qkrtmddn0507.atlassian.net/wiki/spaces/196616/pages/edit-v2/9666602/>

5.1.1 멀티로봇 동시에 NAV2 수행

5.1.1.1 코드

1. domain bridge 설치

```
sudo apt install ros-jazzy-domain-bridge
```

https://github.com/ros2/domain_bridge

2. 관련 코드

https://github.com/jewookwak/ROS2_Jazzy_Study

5.1.2 Terminal

[로봇] pinky bringup 실행

```
ros2 launch pinky_bringup bringup.launch.xml
```

[로봇] 내비게이션 실행

```
ros2 launch pinky_navigation initialpose_bringup_launch.xml map:=map.yaml
```

[로봇] goalpose listener 실행

```
ros2 run pinky_rmf goal_pose_listener
```

[PC] nav2_view.rivz 실행

```
ros2 launch pinky_navigation nav2_view.launch.xml
```

[PC] domain bridge 실행

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_goalpose.yaml
```

[PC] pinky1 run goalpose 실행

```
ros2 run pinky_mapf goal_point1
```

[PC] pinky2 run goalpose 실행

```
ros2 run pinky_mapf goal_point2
```

5.1.3 MAPF: 멀티 로봇 경로 계획

<https://github.com/nicofretti/MAPF>

<aside> 🎧

제약 사항 A*를 바꾸는 형태이기 때문에 map 자체를 받아서 사용해야한다. 따라서 waypoints 만 지정해서 네비 게이션하는 방식과 호환이 잘 안될 수 있다. 가상의 격자를 만들어서 새로운 Grid map을 만들어서 사용해야 할 것이다.

</aside>

5.1.3.1 CBS(Conflict-Based Search)

CBS는 positvie constraint와 negative constraint로 특정 시간에 agent가 특정 위치에 도달할 수 있도록 한다.

5.1.3.2 ⚡ 1 Task 1: Space-Time A*

기존의 A*는 공간(cell)만 고려했지만, 이제는 시간(time step)도 함께 고려해서 충돌 없는 경로를 탐색해야 해요.

5.1.3.2.1 ◆ 1.1 Space-Time 도메인 탐색

목표: `single_agent_planner.py` 의 `a_star()` 함수를 수정해서 시간까지 고려하는 A* 구현

5.1.3.2.2 ☰ 변경 사항 요약

1. 노드에 시간 정보 추가

```
root = {
    'location': start,
```

```

    'g_val': 0,
    'h_val': h_val,
    'timestep': 0,
    'parent': None
}

```

2. closed_list 구조 변경

기존: `closed_list[cell] = g_val`

변경: `closed_list[(cell, timestep)] = g_val`

5.1.3.2.3 ◇ 1.2 Vertex Constraint 처리

목표: 어떤 에이전트가 특정 시간에 특정 셀에 있을 수 없도록 제한하기

5.1.3.2.4 예시 (agent 2가 시간 5에 (3, 4)에 있지 못함):

```
{'agent': 2, 'loc': [(3,4)], 'timestep': 5}
```

5.1.3.2.5 구현 단계

1. `constraints` 리스트를 `a_star()`에 넘겨줌

2. 제약 테이블 생성
 - `build_constraint_table(constraints, agent_id)`
 - 시간 단위로 딕셔너리 생성

3. 제약 위반 검사
 - `is_constrained(curr_loc, next_loc, timestep, constraint_table)`

📝 테스트

```
python3 run_experiments.py --instance instances/exp1.txt --solver Prioritized
```

예: agent 0/o goal (1,5)o 시간 4에 도착하지 못하도록 제한 → agent 0이 잠시 기다리게 됨

5.1.3.2.6 ◇ 1.3 Edge Constraint 처리

목표: 특정 시간에 특정 경로를 통해 이동하는 것을 막기

5.1.3.2.7 예시:

agent 2가 시간 4→5에 (1,1) → (1,2) 이동 금지

```
{'agent': 2, 'loc': [(1,1), (1,2)], 'timestep': 5}
```

- `is_constrained()` 함수에 edge constraint도 고려되도록 수정
- 이전 위치 + 현재 위치 + 시간 step으로 검사

테스트

```
# agent 1이 시간 0~1에 (1,2) → (1,3) 금지
python3 run_experiments.py --instance instances/exp1.txt --solver Prioritized
```

5.1.3.2.8 ◇ 1.4 Goal Constraints 처리

문제: 어떤 에이전트가 goal에 도달했지만 그 위치에 계속 머물러 있으면 나중에 충돌 가능

5.1.3.2.9 예시:

agent 0이 시간 10에 goal (1,5)에 있으면 안 됨

- 기존 goal 도달 조건: `current['location'] == goal`
- 수정 필요: goal 도달 여부는 해당 위치에 도달하고, 그 시간도 제약을 만족해야 함
- goal에 도착하더라도 기다릴 수 있게 허용

5.1.3.2.10 ◇ 1.5 선택 과제: 수작업 제약 조건 설계

- 직접 constraints를 설계하여 충돌 없는 최단 경로 만들기
- 테스트: 여러 constraint 적용 후 실행 결과 확인
- 출력 결과에 충 경로 길이 합도 출력됨

5.1.3.2.11 요약 테스트 명령어들

```
# 기본 테스트 (동작 유지)
python3 run_experiments.py --instance instances/exp1.txt --solver Independent

# 제약 조건 테스트 (Prioritized 사용)
python3 run_experiments.py --instance instances/exp1.txt --solver Prioritized
```

5.1.3.3 2 Task 2: Prioritized Planning

에이전트별 우선순위를 정해서 순차적으로 경로를 계획하는 방법입니다.

앞선 에이전트의 경로를 기반으로 뒤따르는 에이전트에 제약을 걸어 충돌을 방지합니다.

5.1.3.3.1 ◦ 기존 Independent Solver vs Prioritized Solver

항목	Independent	Prioritized
충돌 방지 대상	환경만	환경 + 다른 에이전트
계획 순서	동시에	순차적으로 (우선순위 사용)
사용 제약	없음	앞선 에이전트의 경로로부터 유도된 제약

5.1.3.3.2 ☰ 구현 파일: prioritized.py

5.1.3.3.3 ◦ 2.1 Vertex Constraint 추가

앞선 에이전트가 점유한 cell에 대해 뒤따르는 모든 에이전트에게 vertex 부여

5.1.3.3.4 구현 예시:

```
for t in range(len(path)):
    for other_agent in range(i + 1, num_of_agents):
        constraints.append({
            'agent': other_agent,
            'loc': [t],
            'timestep': t
        })
```

📎 테스트

```
python3 run_experiments.py --instance instances/exp2_1.txt --solver Prioritized
```

예상 결과: agent 2는 goal에 도착하더라도 그대로 머무를 수 없음 → 충돌 방지 필요

5.1.3.3.5 ◆ 2.2 Edge Constraint 추가

두 에이전트가 동시에 서로의 위치로 교차 이동하는 경우를 방지

5.1.3.3.6 구현 예시:

```
for t in range(1, len(path)):
    prev = path[t - 1]
    curr = path[t]
    for other_agent in range(i + 1, num_of_agents):
        constraints.append({
            'agent': other_agent,
            'loc': [curr, prev],
            'timestep': t
        })
```

📎 다시 테스트 (위와 동일한 명령어):

```
python3 run_experiments.py --instance instances/exp2_1.txt --solver Prioritized
```

이제 모든 충돌이 해결되어야 합니다.

5.1.3.3.7 ◆ 2.3 (선택) Goal 유지 방지 제약 추가

어떤 에이전트가 goal에 도달한 후 그 자리에 계속 머무르면 다른 에이전트와 충돌할 수 있음

5.1.3.3.8 해결 방법:

- 도달 이후 시간부터 무한히 goal 위치에 머무는 것을 다른 에이전트에 대해 금지
- 예시:

```
python3
복사편집
for t in range(goal_time, max_time):
    constraints.append({
        'agent': other_agent,
        'loc': [goal],
        'timestep': t
    })
```

테스트:

```
python3 run_experiments.py --instance instances/exp2_2.txt --solver Prioritized
```

확인: agent 0이 goal에 도달한 후, agent 1이 그 위치로 가면 안 됨

5.1.3.3.9 ◆ 2.4 (선택) 실패 처리

어떤 우선순위에서는 해가 없을 수도 있음

5.1.3.3.10 테스트 인스턴스:

```
python3 run_experiments.py --instance instances/exp2_3.txt --solver Prioritized
```

- 예상 문제: 무한 루프 또는 무한 경로 생성

5.1.3.3.11 해결 방법:

- 최대 탐색 시간(`max_timestep`)을 설정
 - 앞선 에이전트들의 경로 길이 합 + 환경 크기 정도로 제한
-

5.1.3.3.12 ◆ 2.5 (선택) 우선순위 한계 실험

다음 조건을 만족하는 MAPF 인스턴스를 직접 만들거나 테스트해 볼 수 있어요:

- 어떤 우선순위에서는 충돌 없는 해가 없음
- 어떤 우선순위에서는 최적이 아님
- 모든 우선순위에서도 해가 없는 경우

실험 예시:

- `instances/exp2_1.txt ~ exp2_3.txt`
- 또는 직접 `.txt` 인스턴스를 만들어 테스트 가능

5.1.3.4 ✓ 정리

구현 단계	내용
2.1	앞 에이전트 경로를 기반으로 vertex constraint 추가
2.2	edge constraint 추가로 교차 이동 방지
2.3	goal에 도달한 후 충돌 방지를 위한 제약 추가
2.4	실패 방지를 위한 max timestep 설정
2.5	우선순위의 불완전성과 비최적성 확인 실험

5.1.3.5 ✎ 3 Task 3: Conflict-Based Search (CBS)

CBS는 우선순위 기반 알고리즘보다 느리지만,

항상 해를 찾고(optimal), 충돌 없이 정확하게 경로를 생성할 수 있는 강력한 방법입니다.

5.1.3.5.1 🔔 핵심 아이디어 요약

- CBS는 두 계층 구조로 동작합니다:
 - **High-Level** (상위): 충돌이 있는 경로를 찾아 제약을 추가하며 **Constraint Tree(CT)**를 확장
 - **Low-Level** (하위): 주어진 제약을 만족하는 단일 에이전트 A* 실행

5.1.3.6 🔔 구현 파일: cbs.py

5.1.3.6.1 ◈ 3.1 충돌 탐지: detect_collisions()

에이전트들 경로 간의 충돌을 탐지합니다.

5.1.3.6.2 ♪ 충돌 종류:

1. **Vertex Collision**: 두 에이전트가 같은 위치에 같은 시간에 존재
2. **Edge Collision**: 서로 서로의 위치를 교환 (예: agent 0@| A → B, agent 1@| B → A)

5.1.3.6.3 🔔 함수 구현

- `get_location(path, t)` 를 사용해서 시간 t의 위치를 가져옴
- `detect_collision(path1, path2)` → 한 쌍의 에이전트 경로 비교
- `detect_collisions(paths)` → 전체 에이전트 쌍 반복

5.1.3.6.4 🔎 출력 예시:

```
[{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 3}]
```

📎 테스트

```
python3 run_experiments.py --instance instances/exp3_1.txt --solver CBS
```

5.1.3.6.5 ◆ 3.2 충돌을 제약으로 변환: `standard_splitting()`

충돌이 발견되면, 해당 충돌을 해결하기 위해 제약을 나누어 두 개의 자식 노드를 생성합니다.

5.1.3.6.6 예시 (`vertex collision`):

```
collision = {'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 3}
```

→ 다음 제약 2개 생성:

```
[
  {'agent': 0, 'loc': [(1, 4)], 'timestep': 3},
  {'agent': 1, 'loc': [(1, 4)], 'timestep': 3}
]
```

5.1.3.6.7 `edge collision`의 경우:

```
{'loc': [(x1, y1), (x2, y2)], 'timestep': t}
```

→ 두 에이전트에 대해 각각 방향 반대의 `edge` 제약 부여

5.1.3.6.8 ◆ 3.3 CBS High-Level 탐색: `find_solution()`

충돌 없는 최적 경로가 나올 때까지 제약을 계속 추가하며 탐색합니다.

5.1.3.6.9 알고리즘 흐름 (Pseudocode 요약)

1. 초기 루트 노드 생성 (제약 없음, 독립적으로 경로 탐색)
2. OPEN 리스트에 추가
3. **while** OPEN 리스트가 비어있지 않으면:
 - 비용이 가장 적은 노드 P를 꺼냄
 - 만약 P에 충돌이 없다면 → 해결 완료
 - 충돌이 있다면 → 하나 골라서 `standard_splitting`으로 나눔
 - 각각의 제약을 반영한 자식 노드 Q 생성

- 해당 agent에 대해 A* 실행
- 실패 시 무시
- 경로 갱신 + 충돌 탐지 + cost 계산
- OPEN 리스트에 다시 추가

5.1.3.6.10 💡 도움 함수:

- `push_node(OPEN, node)`
- `pop_node(OPEN)`

5.1.3.6.11 ◆ 3.4 테스트 및 결과 확인

```
python3 run_experiments.py --instance "instances/test_*" --solver CBS --batch
```

- 실행 결과가 `results.csv`로 저장됨
- `instances/min-sum-of-cost.csv`와 비교 가능

5.1.3.7 ✅ Task 3 요약

단계	구현 항목
3.1	충돌 탐지 (vertex/edge)
3.2	충돌 → 제약 조건으로 변환
3.3	상위 탐색 알고리즘 (<code>find_solution</code>) 완성
3.4	결과 확인 및 테스트

5.1.3.8 🧠 4 Task 4: CBS with Disjoint Splitting

기본 CBS는 충돌 시 두 에이전트 모두에게 부정적인 제약(negative constraint)을 줍니다.

*Disjoint Splitting*은 이 중 하나를 긍정적인 제약(positive constraint)으로 바꿔서

다른 에이전트는 모두 자동으로 해당 셀/이동을 피하도록 만들기 때문에 성능이 향상됩니다.

5.1.3.8.1 🔎 예시: vertex 충돌

- agent A와 B가 시간 t에 cell x에서 충돌
- 기존 CBS:**

```
{'agent': A, 'loc': [x], 'timestep': t}    # agent A 금지
{'agent': B, 'loc': [x], 'timestep': t}    # agent B 금지
```

- Disjoint Splitting:**

```
{'agent': A, 'loc': [x], 'timestep': t, 'positive': True}    # A 반드시 있어야 함
```

→ 이렇게 하면 A는 해당 위치에 있어야 하고, 나머지 에이전트는 모두 자동적으로 금지됨.

5.1.3.9 🔐 구현 항목 요약

5.1.3.9.1 ◦ 4.1 Positive Constraint 지원

파일: `single_agent_planner.py`

함수들 설정:

- `build_constraint_table(constraints, agent)`
- `is_constrained(loc1, loc2, time, constraint_table)`

5.1.3.9.2 변경 사항:

- constraint 딕셔너리에 `'positive': True` 키가 있을 수 있음
- 제약 테이블에 positive 제약 따로 기록
- 각 노드 확장 시:
 - 해당 에이전트가 positive constraint를 만족하지 않으면 → **prune**

5.1.3.9.3 예시:

```
{
    'agent': 1,
    'loc': [(2, 2)],
    'timestep': 5,
    'positive': True
}
```

5.1.3.9.4 ◆ 4.2 Disjoint Splitting 구현

파일: cbs.py

함수 추가: disjoint_splitting(collision)

- 랜덤하게 agent 1 또는 2 중 하나 선택
- 선택된 agent에는 **positive constraint** 부여
- 나머지는 기존 CBS처럼 negative constraint 부여

```
import random
if random.randint(0,1) == 0:
    # agent1 → positive constraint
    # agent2 → negative constraint
else:
    # agent2 → positive constraint
    # agent1 → negative constraint
```

5.1.3.9.5 ◆ 4.3 High-Level 탐색 수정

파일: cbs.py

함수 수정: find_solution(disjoint=True) 또는 disjoint_cbs_main() 형태

5.1.3.9.6 변경 사항:

- Positive constraint가 걸렸을 때는 해당 에이전트뿐 아니라 다른 에이전트들도 재탐색 필요
- paths_violate_constraint(constraint, paths) 함수로 확인

```
violators = paths_violate_constraint(constraint, paths)
for agent_id in violators:
    path = a_star(...)
    if path is None:
```

```

skip_node = True
break
update_paths(...)
```

5.1.3.9.7 ✎ 테스트

```
python3 run_experiments.py --instance instances/exp4.txt --solver CBSDisjoint
```

→ 결과: 기본 CBS는 약 11개의 노드를 확장,

CBS with disjoint splitting은 약 8개 정도로 줄어듦 (컴퓨터/랜덤에 따라 조금 다름)

5.1.3.10 ✓ Task 4 요약

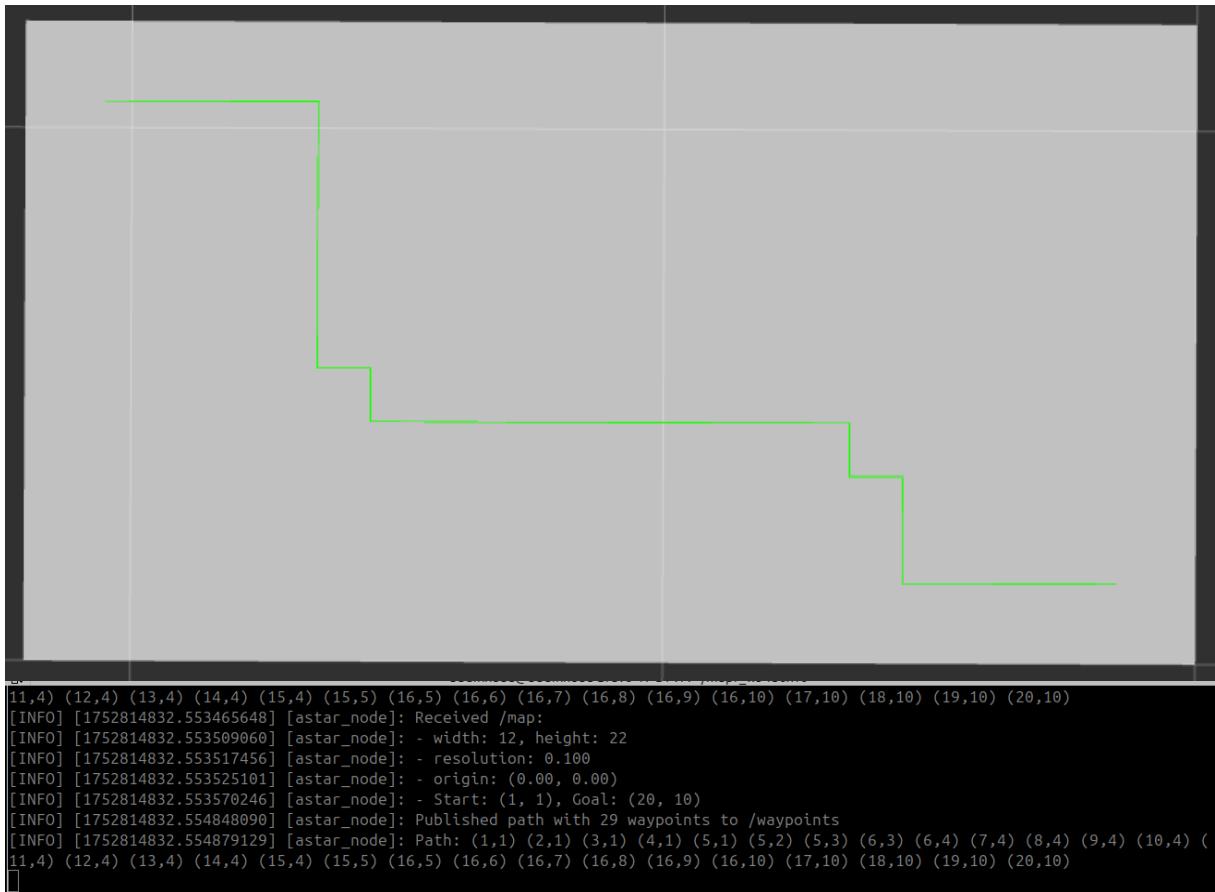
단계	구현 항목
4.1	positive constraint 처리
4.2	disjoint splitting 함수 구현
4.3	High-level search 수정 (positive constraint가 영향을 주는 모든 agent 경로 재탐색)
4.4	테스트: exp4.txt 등에서 node 확장 수 비교

5.1.4 구현

5.1.5 1단계: waypoints 기준 A*

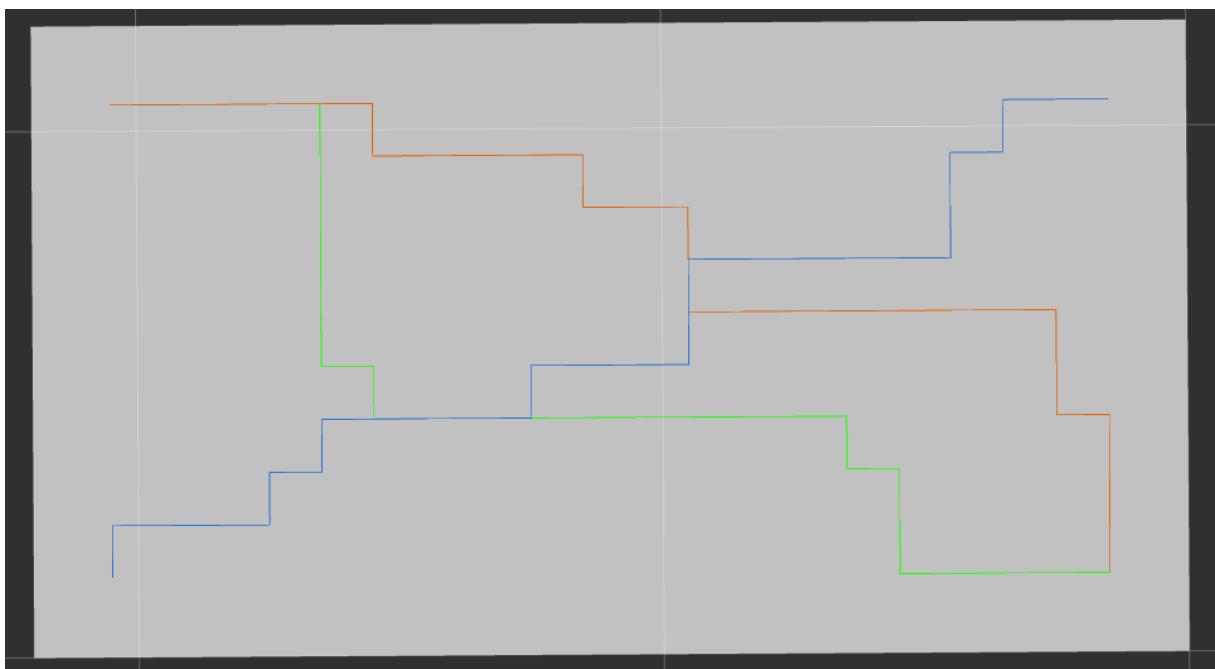
- 관제에서 Occupancy_grid map을 준다는 가정
- agent 1개의 path를 출력

결과 시각화



5.1.5.1 2 단계 구현

prioritized_planner_node.cpp



```
agent_starts_ = { {1, 1}, {1, 10}, {20, 10} }; // 예시: 3개
agent_goals_ = { {20, 10}, {20, 1}, {1, 1} };
```

5.1.5.2 코드 간략한 설명

기존 A*에서 추가된 기능

space-time A*

agent 1에 대한 경로를 만들 때 각 그리드에서 agent1의 timestep이 추가된다.

5.1.5.2.1 vertex constraint

```
{'agent': 1, 'loc': [(4,5)], 'timestep': 7, 'final': False} # 1번 에이전트는 7
스텝에 (4,5)에 있으면 안 됨
```

5.1.5.2.2 edge constraint

```
{'agent': 0, 'loc': [(3,4), (3,5)], 'timestep': 9, 'final': False} # 0번 에이
전트는 9 스텝에 (3,4)→(3,5)로 이동하면 안 됨
```

5.1.5.2.3 positive constraint (Disjoint-CBS용)

```
{'agent': 2, 'loc': [(1,1)], 'timestep': 2, 'positive': True, 'final':
False} # 2번 에이전트는 2스텝에 반드시 (1,1)에 있어야 함 (나머지 agent는 가지 마라)
```

추가 설명

ROS의 `nav_msgs/Path` 메시지에는 기본적으로 각 waypoint(=PoseStamped)에 시간 정보를 담을 수 있습니다.

바로 각 포즈의 `header.stamp` 필드에 **timestep(스텝 단위, 혹은 초/밀리초 변환)**을 추가할 수 있습니다.

5.1.5.3 수정 예시: 각 waypoint에 timestep 반영

1. timestep 기준:

- 각 경로의 0, 1, 2, ... 번째 포인트에 대해

- `header.stamp = now + timestep * dt` 형식으로 설정할 수 있습니다.
- `dt`는 예를 들어 1초(1.0) 또는 원하는 시간 간격을 의미합니다.

• **핵심:**

```
pose.header.stamp = start_time + rclcpp::Duration::from_seconds(dt *
t);
```

→ 각 waypoint가 자신의 timestep(경로 내 순번)에 맞게 시간정보를 가짐

5.1.5.4 요약

- 각 waypoint에 `header.stamp`로 timestep을 포함시키려면 위 방식대로 적용
 - RViz나 다른 노드에서 이 시간 정보를 읽어 시뮬레이션, 동기화 등에 쓸 수 있음
-

5.1.5.4.1 필요하면 전체 코드 수정본도 만들어드릴 수 있습니다!

혹시 전체 우아하게 고친 예시가 필요하면 말씀해 주세요.

질문 주신 방식 그대로, Path의 모든 pose에 "timestep"을 header.stamp로 넣어주는 방식입니다!

5.1.5.4.1.1 나의 말:

`pose.header.stamp`가 뭐야?

ChatGPT의 말:

좋은 질문입니다!

5.1.5.5 `pose.header.stamp` 란?

- `geometry_msgs/msg/PoseStamped` 메시지는 ROS의 표준 메시지 타입입니다.
- 이 메시지의 구조는 아래와 같습니다:

```
std_msgs/Header header geometry_msgs/Pose pose
```

- 여기서 **`header`**는 ROS 메시지의 표준 메타데이터 구조로,
 - `stamp` (시간 정보, ROS time)
 - `frame_id` (좌표계 frame 이름)
를 포함합니다.
-

5.1.5.5.1 1. header.stamp 의 역할

- 0| Pose(waypoint)가 언제 기준의 위치인지를 나타내는 시간 정보
 - ROS 시스템에서 타임스탬프(`rclcpp::Time`, 초/나노초 등)로 표현
 - Path 재생, 동기화, 시뮬레이션, 재현 등에 꼭 필요
-

5.1.5.5.2 2. Path 메시지에 어떻게 들어가나?

- `nav_msgs/Path` 는 내부적으로 `std::vector<PoseStamped>` `poses` 멤버를 가짐
 - 각 `PoseStamped`마다 `header.stamp` 이 있어 "몇 번째(혹은 언제) 위치"라는 정보를 포함
-

5.1.5.5.3 3. 사용 예시

- 실제 로봇 주행 기록/재현(Playback)
 - 시뮬레이터와의 시간 동기화
 - 여러 경로의 충돌 체크 시 "동일 시점" 비교
-

5.1.5.5.4 4. 정리

- `pose.header.stamp` 은
"이 위치 정보가 어느 시간(timestep, ROS time)에 해당하는지"를 담는 필드입니다.
 - 각 `waypoint`(경로점)에 시간 정보를 넣고 싶다면,
바로 이 `header.stamp` 에 원하는 ROS 시간 값을 넣으면 됩니다.
-

더 궁금한 예시, ROS 메시지 실제 구조 등 원하시면 언제든 설명드릴 수 있습니다!

5.1.5.5.4.1 나의 말:

처음 시작 시점을 기준으로 3초마다 이동하는 한칸씩 이동하는 것으로 만들려고 해 경로의 constraint 조건에 `timestep`이 들어가는데 이것을 `pose.header.stamp`에 넣어서 전송해도 괜찮은 것 맞아?

ChatGPT의 말:

정확하게 이해하셨습니다!

네, `constraint`의 `timestep`과 각 `waypoint`의 `pose.header.stamp` 를 맞춰서
"처음 시작 시점 + 3초 × timestep"으로 설정하는 것은
ROS 표준에 완벽히 부합하며, 가장 깔끔하고 통신 친화적인 방법입니다.

5.1.5.6 왜 이 방법이 적합한가?

1. 경로의 각 waypoint에 '실제 시간 정보'를 직접 포함
 - 시간에 따라 어디에 있어야 하는지 다른 노드(로봇, 시뮬레이터, 동기화 노드 등)에서 명확하게 알 수 있음
 2. constraint의 timestep과 ROS 메시지의 시간 필드를 '동일하게' 활용
 - 예:
 - 0번 점: start_time + 0*3초
 - 1번 점: start_time + 1*3초
 - 2번 점: start_time + 2*3초
 - ...
 - 이런 식으로 ROS Time(Unix Epoch/ROS 시간)에 맞게 전달
 3. ROS에서 공식적으로 지원하는 방식
 - 모든 메시지는 header.stamp 를 기준으로 타임싱크, 트랜스폼, 재현 등에 사용됨
-
-

5.1.5.7 론

constraint에 timestep이 들어가면, 그 시간값을 pose.header.stamp로 옮겨주는 방식이 ROS multi-agent, 동기화/재생/분산 제어 모두에 적합합니다.

- 표준 ROS 패키지와 완벽 호환
 - 시뮬레이터/실로봇 모두 동일한 구조로 재사용 가능
 - 동기화가 필요한 복수 로봇 제어에서도 권장
-

마음 놓고 pose.header.stamp에 “시작 시각 + 3초 × timestep”을 넣으세요!

(이게 ROS Way입니다.)

추가로 각종 constraint-check/동기화에 어떻게 활용할지 궁금하면 구체 예시도 만들어드릴 수 있습니다.

5.1.5.7.1 나의 말:

그러면 해당 내용이 반영된 prioritized_planner_node.cpp 코드를 작성해줘.

5.1.5.7.1.1 ChatGPT의 말:

네!

각 waypoint의 pose.header.stamp 가 "시작 시각 + 3초 × timestep" 이 되도록 전체 구조를 반영한 prioritized_planner_node.cpp 코드를 아래에 제공합니다.

5.1.5.8 포인트 요약

- 각 waypoint의 header.stamp = 시작시각 + 3초 × timestep
- constraint와 ROS 메시지의 시간 정보가 완벽히 매핑
- 퍼블리셔는 생성자에서 한 번만 생성
- RViz, echo에서 토픽을 언제든 볼 수 있음

이대로 적용하시면 timestep에 맞는 시간 정보가 각 path에 포함됩니다!

혹시 추가적으로 constraint와 시간정보 활용, 동기화 제어 예시가 필요하면 언제든 말씀해 주세요.

5.1.5.8.1 나의 말:

궁금한게 실제 로봇에 이렇게 적용하면 로봇이 3초 이내에 경로점에 도착하면 거기서 멈추고 경로점에 도달했다는 것을 관제 서버에 publish할 수 있는 걸까?

5.1.5.8.1.1 ChatGPT의 말:

네, 바로 그 구조가 가능합니다!

ROS의 시간(timestep) 정보를 각 waypoint에 넣는 방식은 **실제 로봇과 분산 시스템에서 “동기화된 행동/보고”에 매우 적합합니다.**

5.1.5.9 1. 동작 흐름 (실제 로봇에 적용 시)

1. 관제(서버, 마스터)

- 경로(path)의 각 waypoint에 "언제(=header.stamp)" 도달해야 하는지 정보 포함
- 이 경로를 로봇에게 전송

2. 로봇(클라이언트) 제어 노드

- 자신의 현재 시간과, 다음 waypoint의 도달 시간(header.stamp)을 비교
- 예를 들어 "3초마다 한 칸 이동"이 목표라면
 - 아직 3초가 안 지났다면, 목표점에 미리 도달하더라도 멈춘 채 대기
 - header.stamp에 딱 맞게 다음 waypoint로 출발/도달을 맞춤

3. 도달 보고(관제에 publish)

- 로봇이 실제로 waypoint에 도달한 순간,
(혹은 header.stamp 시간이 됐을 때)
/reached_waypoint 등으로 “몇 번째 waypoint 도달 완료!”를 publish
- 관제 서버에서 이 도달 메시지를 수신,
예상 시간과 실제 도달 시간 비교 및 동기화 가능

5.1.5.10 2. 구현 요령

- 로봇이 목표점에 미리 도달하면,
 - header.stamp 시각까지 그 자리에서 멈추고 대기
 - header.stamp와 실제 시간(now())가 같아지거나 약간 지났을 때
 - 다음 waypoint로 진행
 - 도달 시점(혹은 대기 시작 시점)에 관제 서버로 메시지 publish
 - 예: /agent1/reached_waypoint (std_msgs/Int32 또는 커스텀 메시지)
-

5.1.5.11 4. 장점

- 동기화된 다중 로봇 시뮬레이션/운영이 쉬워짐
 - 실제 도달/출발 시각을 관제와 항상 맞출 수 있음
 - 과거 재생(playback)이나 "시간 기준 평가"도 매우 쉬워짐
 - 실제 물리 속도와 경로 계획 시간(timestep)을 일치시켜 행동 패턴을 만들 수 있음
-

5.1.5.12 5. 요약 답변

네!

각 waypoint에 header.stamp로 도달해야 할 "계획 시간"을 넣고,
로봇이 미리 도착했으면 그 자리에서 대기,
실제 도달했거나 header.stamp 시간이 되면 관제에 보고하는 시스템은
실제 로봇과 분산 시스템, 시뮬레이션에서 매우 효과적이고 표준적인 방식입니다.

-
- ROS 공식 패턴에도 부합
 - 대형 관제/다중 로봇 프로젝트에서도 이런 구조를 많이 사용합니다.
-

추가로,

- 도달 확인 메시지 포맷
 - 실제 ROS2 퍼블리셔/구독자 예제
 - 제어 노드 로직
- 이 필요하시면 언제든 요청해 주세요!

5.2 <RT-8> <문지언> <로봇팔_카메라 캘리브레이션> <7/11 내용>

5.2.1 DH Parameter를 사용한 카메라와 엔드이펙터 사이의 변환 행렬 구하기

- mycobot의 파라미터를 인터넷 서칭해서 구함

```
# θ: 실시간으로 얻음
angles_deg = mc.get_angles()
angles_rad = np.radians(angles_deg)

# d, a, α: 미리 정의된 값 사용 (예시는 가상의 값)
DH_table = [
    [angles_rad[0], 131.22, 0.0, np.pi/2],
    [angles_rad[1], 0.0, -110.4, 0.0],
    [angles_rad[2], 0.0, -96, 0.0],
    [angles_rad[3], 63.4, 0.0, np.pi/2],
    [angles_rad[4], 75.05, 0.0, -np.pi/2],
    [angles_rad[5], 45.6, 0.0, 0.0],
]
```

사진 첨부

5.2.1.1 # DH파라미터를 사용한 회전 및 이동 행렬 계산 코드

- DH 파라미터 정의 > 전진기구학 함수로 T 계산 > R과 p를 분리 > 역변환 계산도 빠르게 구현 > 예시로 조인트 각도 입력 > 최종 위치, 자세 출력

```
import numpy as np

# _____
# 1) DH 1-프레임 변환
# _____
def dh_transform(theta, d, a, alpha):
    """std-DH 4x4 homogeneous transform A_i-1^i"""
    ct, st = np.cos(theta), np.sin(theta)
    ca, sa = np.cos(alpha), np.sin(alpha)
    return np.array([
        [ct, -st * ca, st * sa, a * ct],
        [st, ct * ca, -ct * sa, a * st],
        [0, sa, ca, d],
        [0, 0, 0, 1]
    ])
# _____
```

```

# 2) myCobot 280 DH 상수 (단위: mm, rad)
#
d_vals      = [131.22,    0,      0,   63.4 ,  75.05,  45.6 ]    # d_i
a_vals      = [    0, -110.4, -96,     0 ,     0 ,     0 ]    # a_i
alpha_vals = [ 1.5708,    0,      0,  1.5708, -1.5708,  0]    # α_i
offsets    = [    0, -1.5708,  0, -1.5708,  1.5708,  0]    # θ_offset_i

#
# 3) 전진 기구학 함수 (조인트 → 엔드아이펙터 자세)
#
def forward_kinematics(q):
    """
    q : 길이 6 리스트/배열, [q1..q6] (rad)
    반환: (T_BE 4×4, R 3×3, p 3×1)
    """
    if len(q) != 6:
        raise ValueError("6-axis 로봇이므로 q 길이는 6이어야 합니다.")

    T = np.eye(4)
    for i in range(6):
        theta_i = q[i] + offsets[i] #실제 조인트 값에 오프셋 보정
        A_i = dh_transform(theta_i, d_vals[i], a_vals[i], alpha_vals[i]) #여기서
transformation matrix 생성
    T = T @ A_i #누적 곱

    R = T[:3, :3]
    p = T[:3, 3]
    return T, R, p

#
# 4) 역변환 함수 (SE(3)의 특성을 활용한 역행렬)
#
def inverse_transform(T):
    """SE(3) 행렬의 역행렬(빠른 방법)"""
    R = T[:3, :3]
    p = T[:3, 3]
    T_inv = np.eye(4)
    T_inv[:3, :3] = R.T
    T_inv[:3, 3] = -R.T @ p
    return T_inv

#
# 5) 사용 예시
#
if __name__ == "__main__":
    q_example = [0, 0, 0, 0, 0, 40]
    T_BE, R, p = forward_kinematics(q_example)
    T_EB = inverse_transform(T_BE)

    np.set_printoptions(precision=6, suppress=True)
    print("Base_T_EE (T_BE) =\n", T_BE)
    print("\nRotation R =\n", R)
    print("\nTranslation p (mm) =\n", p)
    print("\nEE_T_Base (T_EB) =\n", T_EB)

```

출력

```

Base_T_EE (T_BE) =
[[ -0.988032   0.14082   0.062949 -70.339937]
 [ -0.15425   -0.902015  -0.403208  39.529625]
 [ 0.000001  -0.408092   0.912941 409.876928]
 [ 0.          0.          0.          1.        ]]

Rotation R =
[[-0.988032  0.14082  0.062949]
 [-0.15425  -0.902015 -0.403208]
 [ 0.000001  -0.408092  0.912941]]

Translation p (mm) =
[-70.339937  39.529625 409.876928]

EE_T_Base (T_EB) =
[[ -0.988032   -0.15425   0.000001  -63.401156]
 [ 0.14082    -0.902015  -0.408092  212.829137]
 [ 0.062949   -0.403208   0.912941 -353.826858]
 [ 0.          0.          0.          1.        ]]

```

5.2.2 메모 및 개념 정리 (전반 흐름 설명 포함)

5.2.2.1 ◆ 1. SE(3)란?

- **SE(3):** 3D 공간의 회전(R) + 이동(p) 정보를 함께 표현하는 **4×4 동차변환 행렬(homogeneous transform)** 집합.
 - 기호로는 T 또는 T_BE, T_EB 처럼 쓰임. (Base ↔ End-Effector 변환)
-

5.2.2.2 ◆ 2. SE(3)의 수학적 특성

5.2.2.2.1 ✓ 회전행렬 R 의 성질

- 2. SE(3)의 수학적 특성

회전행렬 R 의 성질

- 직교 행렬: $R^T \cdot R = I$
- 행렬식 $\det(R) = +1$
- \Rightarrow 역행렬은 단순히 전치행렬: $R^{-1} = R^T$

SE(3) 전체 행렬의 역행렬 공식

다음 공식을 이용하면 일반적인 `np.linalg.inv()` 없이도 효율적이고 안정적으로 계산 가능:

$$T^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix}$$

5.2.2.3 ◆ 3. 누적 변환 (Forward Kinematics) 4. 좌표 변환 활용

- 3. 누적 변환 (Forward Kinematics)

- DH 파라미터를 이용하여, 각 조인트의 상대변환 A_{i-1}^i 를 계산하고,
- 이를 연속으로 곱해서 T_0^6 (즉 Base \rightarrow EE 변환) 구함.
- 회전 행렬 R , 위치 벡터 p 도 이로부터 분리해 얻을 수 있음.

- 4. 좌표 변환 활용

어떤 3D 점 x 를 엔드 이펙터 기준으로 표현하고, Base 기준 좌표계로 변환하고 싶을 때:

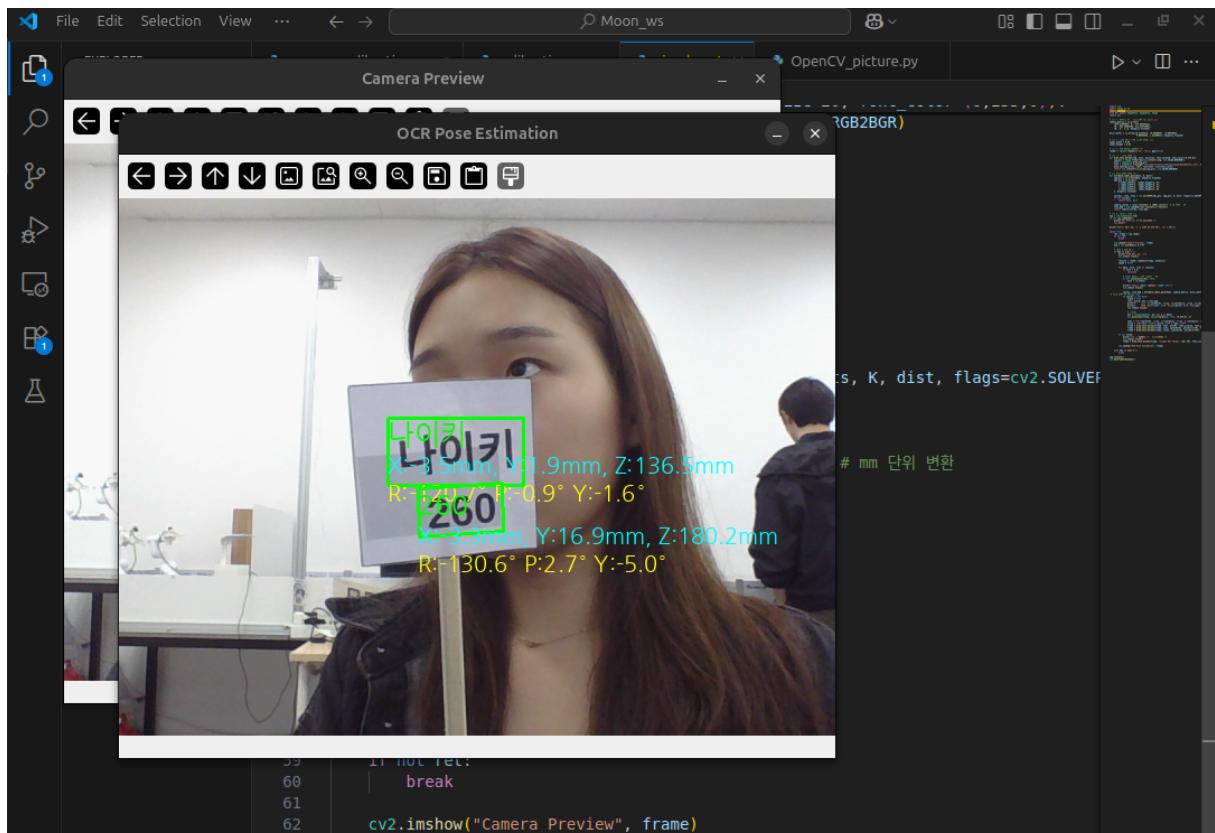
$$x' = T \cdot \begin{bmatrix} x \\ 1 \end{bmatrix}$$

이는 로봇 제어, 비전, 경로 계획 등 다양한 곳에서 핵심적인 개념임.

5.3 <RT-37> <이찬종> <OCR & Detection> <7/31>



https://drive.google.com/file/d/1QYWPTu7DND6irP_eYblu5H55CNbTGQGh/view?resourcekey



```

63     key = cv2.waitKey(1) & 0xFF
64
65     # 캡처 & OCR 실행
66     if key == ord('c'):
67         print("[OCR 실행 중...]")
68         sys.stdout.flush()
69
70     results = reader.readtext(frame, detail=1)
71     found = False
72
73     for bbox, text, conf in results:
74         if conf < 0.6:
75             continue
76
77         # numpy 타입일 경우 문자열 변환
78         if not isinstance(text, str):
79             text = str(text)
80
81         print(f"[인식] {text} (신뢰도: {conf:.2f})")
82         sys.stdout.flush()
83
84     # Pose 계산 후 터미널 출력
85     if coords is not None:
86         found = True
87         roll, pitch, yaw = rvec_deg
88         print(f"좌표: X={coords[0]:.1f}mm, Y={coords[1]:.1f}mm, Z={coords[2]:.1f}mm")
89         print(f"방향: Roll={roll:.1f}°, Pitch={pitch:.1f}°, Yaw={yaw:.1f}°")
90         sys.stdout.flush()
91
92     # 시각화
93     pts = [tuple(map(int, p)) for p in bbox]
94     cv2.polylines(frame, [np.array(pts)], True, (0,255,0), 2)
95
96     info = f"X:{coords[0]:.1f}mm, Y:{coords[1]:.1f}mm, Z:{coords[2]:.1f}mm"
97     info2 = f"R:{roll:.1f}° P:{pitch:.1f}° Y:{yaw:.1f}°"
98     frame = draw_text_korean(frame, text, pts[0], font_size=25, font_color=(0,0,255))
99

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Using CPU. Note: This module is much faster with a GPU.
카메라 실행 중: 'c' → 캡처 후 OCR 실행, 'q' → 종료
[OCR 실행 중...]

- (test_venv) addinedu@addinedu-GP75-Leopard-10SEK:~/Moon_ws\$ python /home/addinedu/Moon_ws/simul.py

Using CPU. Note: This module is much faster with a GPU.
카메라 실행 중: 'c' → 캡처 후 OCR 실행, 'q' → 종료
[OCR 실행 중...]
[인식] 나이키 (신뢰도: 1.00)
좌표: X=8.3mm, Y=-8.1mm, Z=80.0mm
[인식] 260 (신뢰도: 0.76)
좌표: X=12.8mm, Y=0.7mm, Z=113.5mm

- (test_venv) addinedu@addinedu-GP75-Leopard-10SEK:~/Moon_ws\$ python /home/addinedu/Moon_ws/simul.py

Using CPU. Note: This module is much faster with a GPU.
카메라 실행 중: 'c' → 캡처 후 OCR 실행, 'q' → 종료
[OCR 실행 중...]
[인식] 나이키 (신뢰도: 1.00)
좌표: X=-3.5mm, Y=1.9mm, Z=136.5mm
방향: Roll=-120.7°, Pitch=-0.9°, Yaw=-1.6°
[인식] 260 (신뢰도: 1.00)
좌표: X=-3.3mm, Y=16.9mm, Z=180.2mm
방향: Roll=-130.6°, Pitch=2.7°, Yaw=-5.0°

```

import cv2
import numpy as np
import easyocr
from PIL import ImageFont, ImageDraw, Image
import sys

# === 1. 카메라 내부 파라미터 (K, dist) ===
camera_matrix = np.array([
    [679.37865654, 0., 317.98359742],
    [0., 681.58665788, 212.92172904],
    [0., 0., 1.]], dtype=np.float32)

```

```

dist_coeffs = np.array([0.12239146, -0.96600967, -0.00930853,
                      0.00392854, 1.63260631], dtype=np.float64)

# === 2. 라벨 실제 크기 (미터 단위) ===
label_width = 0.04
label_height = 0.04

# === 3. OCR Reader 초기화 ===
reader = easyocr.Reader(['en', 'ko'], gpu=False)

# === 한글 출력 함수 ===
def draw_text_korean(img, text, position, font_size=20, font_color=(0,255,0)):
    img_pil = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    draw = ImageDraw.Draw(img_pil)
    font = ImageFont.truetype("/usr/share/fonts/truetype/nanum/NanumGothic.ttf",
                             font_size)
    draw.text(position, text, font=font, fill=font_color)
    return cv2.cvtColor(np.array(img_pil), cv2.COLOR_RGB2BGR)

# === Pose 계산 함수 ===
def estimate_label_pose(bbox, K, dist):
    img_pts = np.array(bbox, dtype=np.float32)
    obj_pts = np.array([
        [-label_width/2, -label_height/2, 0],
        [label_width/2, -label_height/2, 0],
        [label_width/2, label_height/2, 0],
        [-label_width/2, label_height/2, 0]
    ], dtype=np.float32)

    success, rvec, tvec = cv2.solvePnP(obj_pts, img_pts, K, dist,
                                         flags=cv2.SOLVEPNP_IPPE_SQUARE)
    if not success:
        return None, None

    camera_coords = (tvec.flatten() * 1000).tolist() # mm 단위 변환
    rvec_deg = (np.rad2deg(rvec.flatten())).tolist()
    return camera_coords, rvec_deg

# === 5. 카메라 열기 ===
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("[X] 카메라를 열 수 없습니다.")
    sys.exit()

print("카메라 실행 중: 'c' → 캡처 후 OCR 실행, 'q' → 종료")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    cv2.imshow("Camera Preview", frame)
    key = cv2.waitKey(1) & 0xFF

    # 캡처 & OCR 실행
    if key == ord('c'):

```

```

print("[OCR 실행 중...]")
sys.stdout.flush()

results = reader.readtext(frame, detail=1)
found = False

for bbox, text, conf in results:
    if conf < 0.6:
        continue

    # numpy 타입일 경우 문자열 변환
    if not isinstance(text, str):
        text = str(text)

    print(f"[인식] {text} (신뢰도: {conf:.2f})")
    sys.stdout.flush()

    coords, rvec_deg = estimate_label_pose(bbox, camera_matrix, dist_coeffs)
    # Pose 계산 후 터미널 출력
    if coords is not None:
        found = True
        roll, pitch, yaw = rvec_deg
        print(f"    좌표: X={coords[0]:.1f}mm, Y={coords[1]:.1f}mm,
Z={coords[2]:.1f}mm")
        print(f"    방향: Roll={roll:.1f}°, Pitch={pitch:.1f}°, Yaw={yaw:.1f}°")
        sys.stdout.flush()

        # 시각화
        pts = [tuple(map(int, p)) for p in bbox]
        cv2.polyline(frame, [np.array(pts)], True, (0,255,0), 2)

        info = f"X:{coords[0]:.1f}mm, Y:{coords[1]:.1f}mm, Z:{coords[2]:.1f}mm"
        info2 = f"R:{roll:.1f}° P:{pitch:.1f}° Y:{yaw:.1f}°"
        frame = draw_text_korean(frame, text, pts[0], font_size=25,
font_color=(0,255,0))
        frame = draw_text_korean(frame, info, (pts[0][0], pts[0][1]+30),
font_size=20, font_color=(0,255,255))
        frame = draw_text_korean(frame, info2, (pts[0][0], pts[0][1]+55),
font_size=20, font_color=(255,255,0))

    if not found:
        print("[경고] 타겟을 찾지 못했습니다.")
        sys.stdout.flush()
        frame = draw_text_korean(frame, "Target Not Found", (20, 40), font_size=25,
font_color=(255,0,0))

    cv2.imshow("OCR Pose Estimation", frame)

elif key == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()

```

5.4 <RT-17> <이창연> <멀티 로봇 위치 ROS2 연동(Nav2 package)> <7/16>

test_map.pgm

https://www.notion.so/Grid-feat-tracked_pose-ROS2-Nav2-package-2321adbda1df80e3910be89b01c0065b?source=copy_link



13 22x12

(0.1 ,0.1)

(0.1 , 1)

(2 , 0.1)

(2, 1)

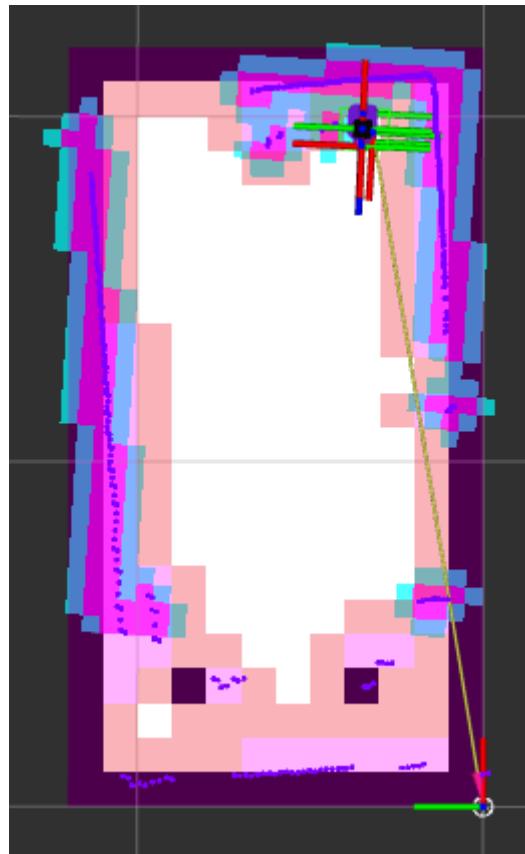
test_map.yaml

```
image: test_map.pgm
mode: trinary
resolution: 0.1
origin: [0, 0, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

실제 맵의 크기는 2mx1m 이다

pgm 이미지를 22x12로 설정했고 벽면을 제외하면 20x10이다.

따라서 각 픽셀 좌표가 waypoint로 사용할 수 있다.



5.5 <RT-22> <문지언> <로봇팔 - 비전 캘리브레이션> <7/11 >

5.5.1 X 는 해(solution)지 정답과 비교할 수 있는 GT(Ground Truth)가 없음

- 정답에서 몇 퍼센트 틀렸다"는 오차율을 정의할 수 없음.
- 따라서 아래와 같은 간접적 오차 측정 지표들을 사용해서 X행렬을 평가하자

정리		
항목	의미	설명
Reprojection Error	카메라 파라미터 정확도 척도	실제 검출된 2D 점과 예측된 2D 점의 거리 (픽셀 단위)
왜곡 보정 전/후 이미지 오차	왜곡 보정 효과 측정	단순히 시각적 차이거나 보정 좌표간의 거리 차이
보정 잘 됐는지 확인	Reprojection error를 통해 판단	0.1 ~ 1.0 픽셀 이면 우수

✓ 그래서 채택되는 전략은?

측정 방식	목적	대표 지표
Residual error $\ AX - XB\ $	수학적으로 얼마나 잘 풀렸는가	L2 norm
Reprojection error	실제 로봇과 카메라 정렬 정확도	픽셀 단위 or mm 단위 오차
Pose error (if GT X exists)	GT와 얼마나 비슷한가	회전각 오차 (deg), 위치 오차 (mm)

Hand-Eye Calibration은 비선형임 => why? X행렬은 4x4행렬이다.

회전행렬은 $RT^*R=I$, $\det(R)=1$ 을 만족해야해, 쿼터니언 $\|q\|=1$ 을 만족해야해

==> *간접 지표들(L2 오차, 회전각 오차, reprojection error 등)**로 정밀도를 평가함

```
import cv2

def detect_qr_center():
    qr_detector = cv2.QRCodeDetector()
    cap = cv2.VideoCapture(2)

    if not cap.isOpened():
        print("X 카메라를 열 수 없습니다.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            print("X 프레임 읽기 실패")
            break

        data, points, _ = qr_detector.detectAndDecode(frame)

        if points is not None and len(points) > 0 and data:
            points = points[0].astype(int)

            # 중심점 계산
            center_x = int(points[:, 0].mean())
            center_y = int(points[:, 1].mean())
            center = (center_x, center_y)

            # 중심점 시각화
            cv2.circle(frame, center, 6, (0, 0, 255), -1)
            cv2.putText(frame, f"Center: {center}", (center_x + 10, center_y),
                       cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

            # 디버그 출력
            print(f"[QR 중심 좌표] {center}")
            print(f"[QR 데이터] {data}")

    # 윤곽선 표시
```

```

for i in range(len(points)):
    pt1 = tuple(points[i])
    pt2 = tuple(points[(i + 1) % len(points)])
    cv2.line(frame, pt1, pt2, (0, 255, 0), 2)

cv2.imshow("QR Scanner", frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()

# 실행
if __name__ == "__main__":
    detect_qr_center()

```

<<7/17자 할 일들>>

1. 실제 잡고 놓고 좌표 변환 했을때 얼마나 차이나는지 육안으로 확인하기
 2. 검증 방식 3가지에 대해서 간단히 조사
 3. 위 보라색 형관펜의 의미 파악하기
-
- #사진 찍을때의 조인트 값을 받기 위해 사용된 코드 (수동조작 유\무 코드 2개)
 - [picture_joint_check.py](#)¹²
 - R_target2cam, t_target2cam, joint_angels 구하는 코드
 - [target2cam.py](#)¹³
 - R_target2cam, t_target2cam, R_gripper2base, t_gripper2base 행렬 출력 코드=확인용
 - [all_matrix.py](#)¹⁴
 - Hand-Eye Calibration → Reprojection error 검출 코드 ::실제 검출된 2D 점과 예측된 2D 점의 거리 (픽셀 단위)
 -  결과 침부
 - [reprojection_error.py](#)¹⁵
 - gripper2base code
 - [gripper2base_matrix.py](#)¹⁶
 - Hand in eye calibration code

12. [https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/picture_joint_check.py?
api=v2&cacheVersion=1&modificationDate=1752722044946&version=1](https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/picture_joint_check.py?api=v2&cacheVersion=1&modificationDate=1752722044946&version=1)

13. [https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/target2cam.py?
api=v2&cacheVersion=1&modificationDate=1752721551026&version=2](https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/target2cam.py?api=v2&cacheVersion=1&modificationDate=1752721551026&version=2)

14. [https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/all_matrix.py?
api=v2&cacheVersion=1&modificationDate=1752722267911&version=1](https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/all_matrix.py?api=v2&cacheVersion=1&modificationDate=1752722267911&version=1)

15. [https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/reprojection_error.py?
api=v2&cacheVersion=1&modificationDate=1752721784964&version=1](https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/reprojection_error.py?api=v2&cacheVersion=1&modificationDate=1752721784964&version=1)

16. [https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/gripper2base_matrix.py?
api=v2&cacheVersion=1&modificationDate=1752721582577&version=2](https://qkrtdmddn0507.atlassian.net/wiki/download/attachments/11141121/gripper2base_matrix.py?api=v2&cacheVersion=1&modificationDate=1752721582577&version=2)

- [calibrateHandEye.py¹⁷](#)

```
Python

# base to gripper 변환 계산
T_base2gripper, _, _ = forward_kinematics(q)

# gripper to base 변환 계산 (역변환)
T_gripper2base = inverse_transform(T_base2gripper)
```

5.6 <RT-19> <이은혜> <SW 구체적 설계 (통신, 구조)> <25.07.18>

- 사용할 토픽 메시지, 노드, 클래스, 상태 등을 정리한 그림판입니다.

https://www.canva.com/design/DAGtNE3k2mQ/yDDKfWcUoxt0t4AjpN9g3g/edit?utm_content=DAGtNE3k2mQ&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

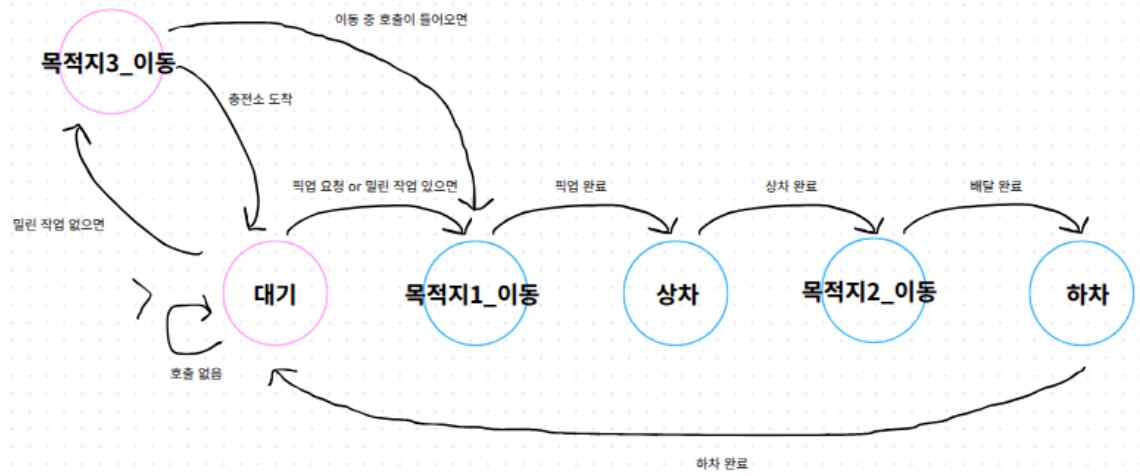
- 시퀀스 다이어그램 - Rotus - Confluence (see page 36) 위 그림판을 정리한 시퀀스 다이어그램입니다.

[예외 상황을 제외한 Flow]



[상태 변환 그래프]

17. <https://qkrtdnn0507.atlassian.net/wiki/download/attachments/11141121/calibrateHandEye.py?api=v2&cacheVersion=1&modificationDate=1752721599641&version=2>



[용어 정리]

Robot_state

- working
- idle
- stop

job_state

- 대기
- 퍽업 이동
- 상차 대기
- 배달 이동
- 하차 대기
- 충전 이동

task_name

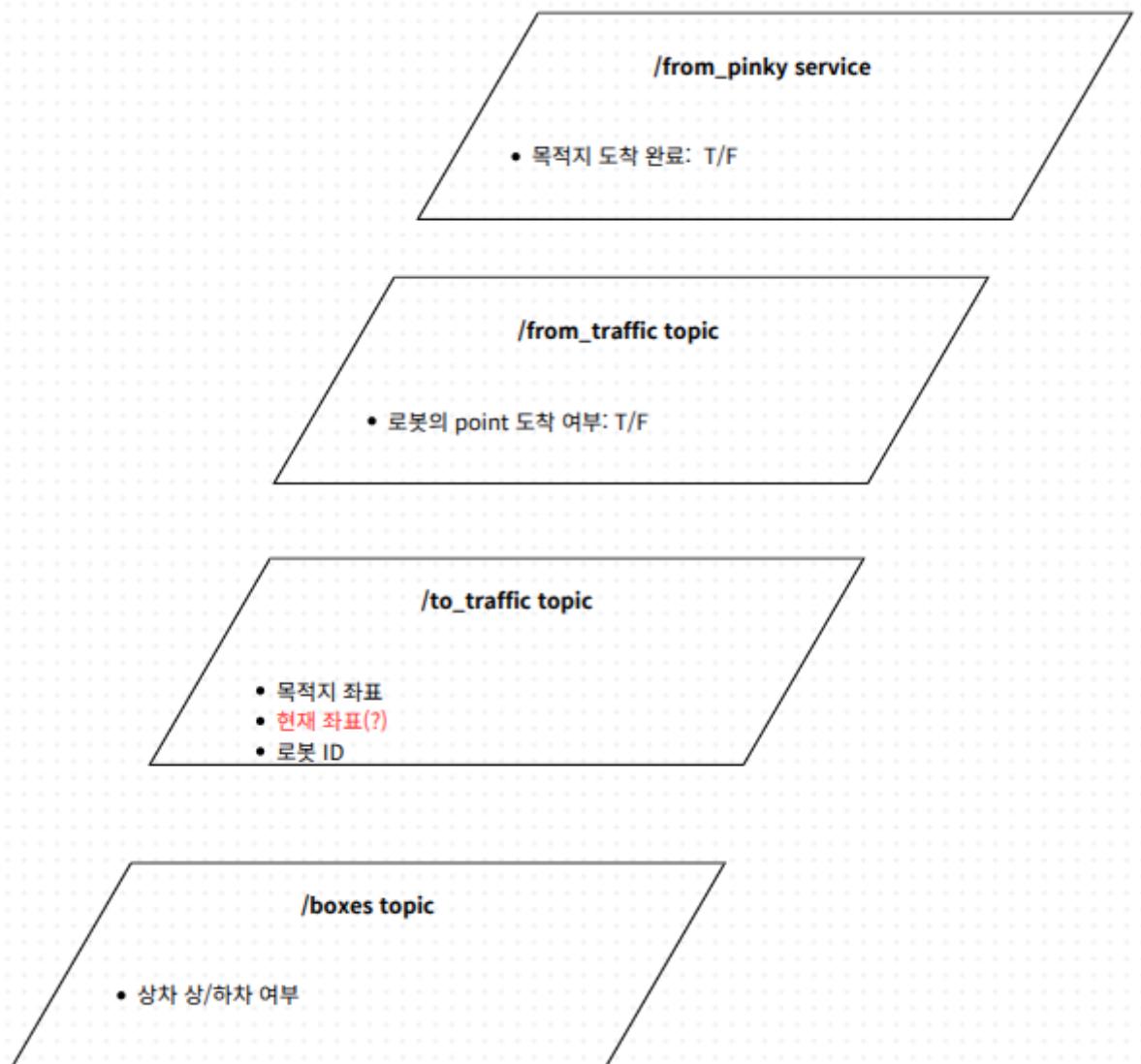
- 배달
- 수거

- State: 각 로봇이 갖는 구조체

struct{

- robot_ID
- robot_state
- job_state
- 배터리
- 신발 제품명
- 로봇 위치
- 목적지1(창고/수거존)
- 목적지2(배달지/창고)
- 목적지3(충전)
- task_name
- 고객 닉네임

[사용할 토픽 및 서비스]



5.7 <RT-27><황다연><PID제어>

패키지 생성 PINKY_VIOLET 의 SRC 내부

```
ros2 pkg create --build-type ament_python pid_controller_node
```

pid 노드 내부 코드

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
```

```

from geometry_msgs.msg import Twist, PoseStamped
import math

class PIDVelocityFilterNode(Node):
    def __init__(self):
        super().__init__('pid_velocity_filter_node')

    # PID 파라미터
    self.declare_parameter("kp", 0.1)
    self.declare_parameter("ki", 0.0)
    self.declare_parameter("kd", 0.0)
    self.declare_parameter("output_limit", 0.3)

    self.kp = self.get_parameter("kp").value
    self.ki = self.get_parameter("ki").value
    self.kd = self.get_parameter("kd").value
    self.output_limit = self.get_parameter("output_limit").value

    # 구독자
    self.cmd_vel_sub = self.create_subscription(
        Twist,
        '/cmd_vel',
        self.cmd_vel_callback,
        10
    )

    self.pose_sub = self.create_subscription(
        PoseStamped,
        '/tracked_pose',
        self.pose_callback,
        10
    )

    # 발행자
    self.publisher = self.create_publisher(
        Twist,
        '/filtered_cmd_vel',
        10
    )

    # 내부 상태
    self.last_error = 0.0
    self.integral = 0.0
    self.actual_velocity = 0.0

    self.prev_position = None
    self.prev_time = self.get_clock().now()

    self.get_logger().info("PIDVelocityFilterNode started (using /tracked_pose for velocity)")

def pose_callback(self, msg):
    now = self.get_clock().now()
    pos = msg.pose.position

    if self.prev_position is not None:
        dt = (now - self.prev_time).nanoseconds * 1e-9

```

```

if dt > 0:
    dx = pos.x - self.prev_position.x
    dy = pos.y - self.prev_position.y
    velocity = math.sqrt(dx**2 + dy**2) / dt
    self.actual_velocity = velocity

    self.get_logger().debug(
        f"[POSE] dt={dt:.3f}, dx={dx:.3f}, dy={dy:.3f},
velocity={velocity:.3f}"
    )

self.prev_position = pos
self.prev_time = now

def cmd_vel_callback(self, msg):
    target = msg.linear.x
    actual = self.actual_velocity
    error = target - actual

    self.integral += error
    self.integral = max(min(self.integral, 10.0), -10.0) # anti-windup

    derivative = error - self.last_error
    self.last_error = error

    # PID 연산
    output = self.kp * error + self.ki * self.integral + self.kd * derivative
    output = max(min(output, self.output_limit), -self.output_limit)

    self.get_logger().info(
        f"[PID] Target={target:.3f}, Actual={actual:.3f}, Error={error:.3f},
Output={output:.3f}"
    )

    filtered = Twist()
    filtered.linear.x = output
    filtered.linear.y = msg.linear.y
    filtered.linear.z = msg.linear.z
    filtered.angular = msg.angular

    self.publisher.publish(filtered)
def main(args=None):
    rclpy.init(args=args)
    node = PIDVelocityFilterNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if name == 'main':
    main()

```

5.7.1 setup.py

```
from setuptools import find_packages, setup

package_name = 'pid_controller_node'

setup(
    name=package_name,
    version='0.0.1',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
         ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pinky',
    maintainer_email='pinky@todo.todo',
    description='PID velocity filtering node using odometry feedback',
    license='MIT',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'pid_velocity_filter_node = pid_controller_node.pid_velocity_filter_node:main',
        ],
    },
)
```

launch

```
<?xml version='1.0' ?>

<launch>
    <arg name="use_sim_time" default="False"/>
    <arg name="motor_ratio" default="1.0"/>

    <include file="$(find-pkg-share pinky_description)/launch/upload.launch.xml">
        <arg name="use_sim_time" value="$(var use_sim_time)"/>
    </include>

    <include file="$(find-pkg-share sllidar_ros2)/launch/sllidar_c1_launch.py">
    </include>

    <include file="$(find-pkg-share ros2_icm20948)/icm20948_node_launch.py">
    </include>

    <node pkg="pinkyBringup" exec="pinkyBringup">
        <param name="motor_ratio" value="$(var motor_ratio)"/>
        <!--  Remap /cmd_vel to /filtered_cmd_vel -->
        <remap from="/cmd_vel" to="/filtered_cmd_vel"/>
```

```

</node>

<!-- <node pkg="pinky_bringup" exec="camera">
</node> -->

<node pkg="pinky_bringup" exec="pinky_battery_publiser">
</node>

</launch>

```

5.8 <RT-27> <곽제우> <PID제어 적용> <2025.7.22>

5.8.1 로봇 속도를 0.6배로 감속

1. pinky_violet 접속
 - a. pinky_bringup/pinkylib/motor


```
self.min_speed = 20/2 으로 최소 속도 변경
```
 - b. pinky_bringup/pinky_bringup.py


```
self.pinky.move(set_l*0.6, set_r*0.6) 으로 변경
```
2. 만약 pinky가 직진으로 가지 않으면
`pinky_bringup/launch/bringup.launch.xml`에서


```
<arg name="motor_ratio" default="1.055"/>
```

 변경

5.8.2 PID 대신 nav2_params.yaml 설정

1. xy_goal_tolerance= 0.05
2. yaw_goal_tolerance= 0.05
3.

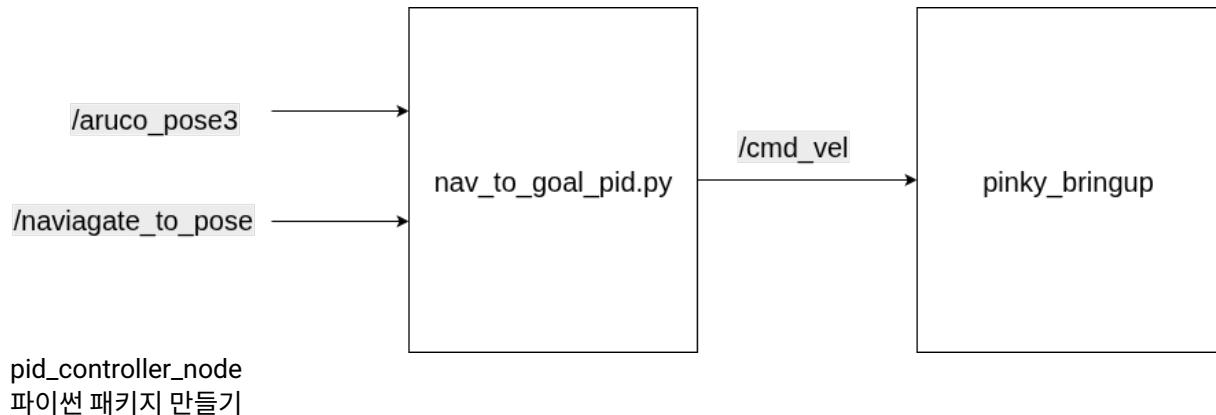
```
progress_checker:
  plugin: "nav2_controller::SimpleProgressChecker"
  required_movement_radius: 0.05 -> 0.2
  movement_time_allowance: 10.0 -> 3.0
```

5.8.2.1 ⚡ progress_checker.required_movement_radius

- **역할:**
로봇이 "진행(progress)하고 있다"라고 판단하는 최소 이동 거리를 의미함.
- **구체적으로:**
이 값은 일정 시간(예: 10초, **movement_time_allowance**) 동안
로봇이 목표 위치로부터 이 거리 이상 움직이지 않으면,

Nav2는 "로봇이 멈췄다(정체/감힘/문제 발생)"고 판단하고 **recovery 행동**(예: 후진, 재탐색, 회전 등)을 시작하게 됨.

5.8.3 aruco maker 기준으로 P제어 적용



5.8.4 nav_to_goal_pid.py

```

#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist, PoseStamped
from nav2_msgs.action import NavigateToPose
from rclpy.action import ActionServer
import math

def quaternion_to_yaw(q):
    siny_cosp = 2.0 * (q.w * q.z + q.x * q.y)
    cosy_cosp = 1.0 - 2.0 * (q.y * q.y + q.z * q.z)
    return math.atan2(siny_cosp, cosy_cosp)

def normalize_angle(angle):
    while angle > math.pi:
        angle -= 2.0 * math.pi
    while angle < -math.pi:
        angle += 2.0 * math.pi
    return angle

class PControllerNode(Node):
    def __init__(self):
        super().__init__('p_controller_node')

```

```

self.goal_pose = None
self.threshold = 0.05 # [m] 도달 오차 허용치
self.arrived = False

self.declare_parameter('kp_linear', 0.2)
self.declare_parameter('kp_angular', 0.1)
self.declare_parameter('max_linear', 0.5)
self.declare_parameter('max_angular', 1.0)
self.declare_parameter('min_linear', 0.05)    # 최소 선속도
self.declare_parameter('min_angular', 0.05)    # 최소 각속도

self.kp_linear = self.get_parameter('kp_linear').value
self.kp_angular = self.get_parameter('kp_angular').value
self.max_linear = self.get_parameter('max_linear').value
self.max_angular = self.get_parameter('max_angular').value
self.min_linear = self.get_parameter('min_linear').value
self.min_angular = self.get_parameter('min_angular').value

self.pose_sub = self.create_subscription(PoseStamped, '/aruco_pose3',
self.pose_callback, 10)
self.cmd_pub = self.create_publisher(Twist, '/cmd_vel', 10)

self.action_server = ActionServer(
    self,
    NavigateToPose,
    '/navigate_to_pose',
    self.execute_callback
)

self.get_logger().info(
    f"Simple PControllerNode started with /aruco_pose3 → /cmd_vel, "
    f"max_linear={self.max_linear}, max_angular={self.max_angular}, "
    f"min_linear={self.min_linear}, min_angular={self.min_angular}"
)

def pose_callback(self, msg):
    if self.goal_pose is None:
        return

    dx = self.goal_pose.position.x - msg.pose.position.x
    dy = self.goal_pose.position.y - msg.pose.position.y
    distance_error = math.hypot(dx, dy)

    robot_yaw = quaternion_to_yaw(msg.pose.orientation)
    target_yaw = math.atan2(dy, dx)
    angular_error = normalize_angle(target_yaw - robot_yaw)

    # --- 도달 체크 ---
    if distance_error < self.threshold:
        linear_v = 0.0
        angular_v = 0.0
        if not self.arrived:
            self.get_logger().info(f"[P] 도착! e={distance_error:.3f}, 목표 제어 멈춤.")
            self.arrived = True
    else:
        linear_v = self.kp_linear * distance_error

```

```

angular_v = self.kp_angular * angular_error

# --- 최소 속도 적용 (최대 속도 제한 전) ---
if 0 < abs(linear_v) < self.min_linear:
    linear_v = self.min_linear * (1 if linear_v > 0 else -1)
if 0 < abs(angular_v) < self.min_angular:
    angular_v = self.min_angular * (1 if angular_v > 0 else -1)

linear_v = max(min(linear_v, self.max_linear), -self.max_linear)
angular_v = max(min(angular_v, self.max_angular), -self.max_angular)
self.arrived = False

twist = Twist()
twist.linear.x = linear_v
twist.angular.z = angular_v
self.cmd_pub.publish(twist)

self.get_logger().info(
    f"[P] e={distance_error:.3f}, yaw_err={math.degrees(angular_error):.1f}°, "
    f"cmd={({linear_v:.3f}, {angular_v:.3f})}"
)

async def execute_callback(self, goal_handle):
    self.get_logger().info("Received NavigateToPose goal")
    self.goal_pose = goal_handle.request.pose.pose
    self.arrived = False # 새로운 goal 지정 시 도착상태 해제
    goal_handle.succeed()
    result = NavigateToPose.Result()
    return result

def main(args=None):
    rclpy.init(args=args)
    node = PControllerNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

5.8.4.1 동작 구조 요약

5.8.4.1.1 노드 초기화

- 노드 이름: p_controller_node
- 파라미터 선언:
 - kp_linear, kp_angular (P 개인)
 - max_linear, max_angular (최대 선속/각속도)
 - min_linear, min_angular (최소 선속/각속도)
- 구독: /aruco_pose3 (로봇 실위치, PoseStamped)

- 퍼블리시: /cmd_vel (Twist, 로봇 구동 명령)
- 액션서버: /navigate_to_pose (Nav2의 goal 수신)

5.8.4.1.2 2. goal 명령 처리

- /navigate_to_pose 액션 goal을 받으면,
goal_pose에 목표 위치를 저장
(execute_callback 함수)

5.8.4.1.3 3. 위치 콜백 (제어의 핵심)

- /aruco_pose3 를 받으면:
 - 목표(goal_pose)가 없으면 아무것도 안 함
 - 목표와의 거리, 각도 오차 계산
 - distance_error : 목표와 현위치 사이의 거리
 - angular_error : 현 방향에서 목표방향까지 회전해야 할 각도 (라디안)
 - 목표에 도달하면
 - 선속도, 각속도 모두 0으로 (정지)
 - 최초 도달 시 “도착!” 로그
 - 목표에 도달하지 않았으면
 - P제어로 선속도, 각속도 계산


```
linear_v = kp_linear * distance_error
angular_v = kp_angular * angular_error
```
 - 최소/최대 속도 보정
 - 최소 속도: 0보다 크고 min보다 작으면 min으로 올림
(슬금슬금 안 가고 뚜벅뚜벅 가게!)
 - 최대 속도: max값으로 제한
 - /cmd_vel로 publish
 - linear.x, angular.z에 각각 할당 후 퍼블리시
 - 상태/제어값을 info 로그로 남김
 - 거리오차, 각도오차(도 단위), 명령 속도 등

5.8.4.1.4 4. 정상 동작 예시 로그

```
[P] e=0.330, yaw_err=166.0°, cmd=(0.300, 0.300)
```

- e: 거리 오차
- yaw_err: 각도 오차(도)
- cmd: 선속도, 각속도 (m/s, rad/s)

5.8.4.2 주요 특징

- 최소/최대 속도 모두 파라미터화 (`ros2 param set`으로 런타임 조정 가능)
 - 목표 도달 자동 정지
 - 액션(goal) 재지정 시 자동으로 다시 추적 시작
 - 각도 오차는 항상 [-pi, pi] 범위
 - yaw는 내부적으로 라디안, 로그만 도 단위
-

5.8.4.3 실제 사용 팁

- PID가 아니라 P제어 (I, D는 없음)
 - 동작중에 파라미터 변경해서 실험 가능
 - /aruco_pose3 토픽만 잘 들어오면 별도 오도메트리/SLAM 필요 없음
 - 최소 속도 세팅을 너무 높이면, 도달 후 멈추지 않고 '진동'할 수도 있음
-

5.8.5 코드 실행 Terminal

```

20f0:
ssh pinky@192.168.0.222
export ROS_DOMAIN_ID=52

1e3e:
ssh pinky@192.168.0.215
export ROS_DOMAIN_ID=53

dfc6:
ssh pinky@192.168.0.219
export ROS_DOMAIN_ID=51

[로봇] pinky bringup 실행

ros2 launch pinky_bringup bringup.launch.xml

[로봇] 내비게이션 실행

ros2 launch pinky_navigation initialpose_bringup_launch.xml

[PC] nav2_view.rivz 실행

pc랑 로봇이랑 도메인 아이디 일치시키기!!

```

```

ros2 launch pinky_navigation nav2_view.launch.xml

[로봇] nav_to_goal_pid 실행
ros2 run pid_controller_package nav_to_goal_pid

ros2 run pid_controller_node nav_to_goal_pid

[PC] 관제캡
ros2 run aruco_marker_pkg aruco_path_tracker

[PC] cbs(경로생성)
ros2 run mapf_planner cbs_planner_node_test2

```

5.8.6 setup.py

```

from setuptools import find_packages, setup

package_name = 'pid_controller_node'

setup(
    name=package_name,
    version='0.0.1',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pinky',
    maintainer_email='pinky@todo.todo',
    description='PID velocity filtering node using odometry feedback',
    license='MIT',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'nav_to_goal_pid = pid_controller_node.nav_to_goal_pid:main',
            'pid_velocity_filter_node =
pid_controller_node.pid_velocity_filter_node:main',
        ],
    },
)

```

package.xml

```
<?xml version="1.0"?>
```

```

<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
  schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>pid_controller_node</name>
  <version>0.0.0</version>
  <description>PID velocity filter node</description>
  <maintainer email="pinky@todo.todo">pinky</maintainer>
  <license>MIT</license>

  <!-- 실행 의존성 -->
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>nav_msgs</exec_depend>
  <exec_depend>nav2_msgs</exec_depend>

  <!-- 테스트 의존성 -->
  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>

```

5.8.7 localization_launch.2xml

```

<?xml version='1.0'?>
<launch>
  <set_env name="RCUTILS_LOGGING_BUFFERED_STREAM" value="1"/>

  <arg name="namespace" default="" />
  <arg name="yaml_filename" default="" />
  <arg name="map" default="" />
  <arg name="use_sim_time" default="false" />
  <arg name="params_file" default="$(find-pkg-share nav2_bringup)/params/
nav2_params.yaml" />
  <arg name="autostart" default="true" />
  <arg name="container_name" default="nav2_container" />
  <arg name="use_composition" default="True" />
  <arg name="use_respawn" default="False" />
  <arg name="log_level" default="info" />
  <arg name="lifecycle_nodes" default="['map_server']" />

  <load_composable_node target="$(var container_name)" if="$(var use_composition)">
    <composable_node pkg="nav2_map_server" plugin="nav2_map_server::MapServer"
      name="map_server">
      <param from="$(var params_file)" />
      <param name='use_sim_time' value="$(var use_sim_time)" />
      <param name='yaml_filename' value="$(var map)" />

      <remap from="/tf" to="tf" />
    </composable_node>
  </load_composable_node>
</launch>

```

```

        <remap from="/tf_static" to="tf_static"/>
    </composable_node>

    <composable_node pkg="nav2_lifecycle_manager"
plugin="nav2.lifecycle_manager::LifecycleManager"
name="lifecycle_manager_localization">
        <param name='use_sim_time' value="$(var use_sim_time)"/>
        <param name='autostart' value="$(var autostart)"/>
        <param name='node_names' value="$(var lifecycle_nodes)"/>
    </composable_node>
</load_composable_node>

<group unless="$(var use_composition)">
    <push-ros-namespace namespace="$(var namespace)"/>
    <node pkg="nav2_map_server" exec="map_server" name='map_server'
output="screen">
        <param from="$(var params_file)"/>
        <param name='yaml_filename' value="$(var map)"/>
        <param name='respawn' value="$(var use_respawn)"/>
        <param name='respawn_delay' value="2.0"/>
        <param name='log_level' value="$(var log_level)"/>
        <param name='use_sim_time' value="$(var use_sim_time)"/>

        <remap from="/tf" to="tf"/>
        <remap from="/tf_static" to="tf_static"/>
    </node>

    <node pkg="nav2_lifecycle_manager" exec="lifecycle_manager"
name="lifecycle_manager_navigation" output="screen">
        <param name='use_sim_time' value="$(var use_sim_time)"/>
        <param name='autostart' value="$(var autostart)"/>
        <param name='node_names' value="$(var lifecycle_nodes)"/>
    </node>
</group>

</launch>

```

5.8.8 initialpose_bringup_launch2.xml

```

<?xml version='1.0'?>
<launch>
    <set_env name="RCUTILS_LOGGING_BUFFERED_STREAM" value="1"/>

    <arg name="namespace" default="" />
    <arg name="map" default="$(find-pkg-share pinky_navigation)/map/test_map01.yaml"/>
    <arg name="use_sim_time" default="False"/>
    <arg name="params_file" default="$(find-pkg-share pinky_navigation)/params/
nav2_params.yaml"/>
    <arg name="params_cartographer" default="initial_pose.lua"/>
    <arg name="autostart" default="True"/>
    <arg name="container_name" default="nav2_container"/>
    <arg name="use_composition" default="True"/>
    <arg name="use_respawn" default="False"/>

```

```

<arg name="log_level" default="info"/>
<arg name="lifecycle_nodes_local" default="['map_server']"/>
<arg name="lifecycle_nodes_nav" default="['controller_server',
    'smoother_server',
    'planner_server',
    'behavior_server',
    'bt_navigator',
    'waypoint_follower',
    'velocity_smoothener']"/>

<group>
    <push-ros-namespace namespace="$(var namespace)"/>
    <node pkg="rclcpp_components" exec="component_container_isolated" name='$(var
container_name)' output="screen" args="--ros-args --log-level $(var log_level)">
        <param from="$(var params_file)"/>
        <param name='map' value="$(var map)"/>
        <param name='autostart' value="$(var autostart)"/>
        <param name='use_sim_time' value="$(var use_sim_time)"/>

        <remap from="/tf" to="tf"/>
        <remap from="/tf_static" to="tf_static"/>
    </node>

    <include file="$(find-pkg-share pinky_cartographer)/launch/
cartographer.launch.xml">
        <arg name="configuration_basename" value="$(var params_cartographer)"/>
        <arg name="use_nav2" value="True"/>
    </include>

    <include file="$(find-pkg-share pinky_navigation)/launch/
localization_launch2.xml">
        <arg name="params_file" value="$(var params_file)"/>
        <arg name="map" value="$(var map)"/>
        <arg name="use_sim_time" value="$(var use_sim_time)"/>
        <arg name="autostart" value="$(var autostart)"/>
        <arg name="use_composition" value="$(var use_composition)"/>
        <arg name="use_respawn" value="$(var use_respawn)"/>
        <arg name="container_name" value="$(var container_name)"/>
        <arg name='lifecycle_nodes' value="$(var lifecycle_nodes_local)"/>
    </include>

</group>
</launch>

```

5.9 <RT-31> <곽제우> <로봇 waypoint 추종> <2025.7.22>

5.9.1 MAPF 실행

```
git clone -b dev_TrafficManager https://github.com/addinedu-roscamp-5th/roscamp-
repo-3.git
```

colcon build 및 source install/setup.bash 실행

```
ros2 run mapf_planner cbs_planner_node
```

5.9.1.1 경로 생성 후 방향 결정 & /navigation_to_pose pose 및 orientation 정보 넣어서 넘겨주기

5.9.1.1.1 Map에서 동서남북

5.9.1.1.2 1. 북쪽 방향의 quaternion 값

방향	yaw(라디안)	quaternion (x, y, z, w)
동쪽 →	0	(0, 0, 0, 1)
북쪽 ↑	$\pi/2$	(0, 0, 0.7071, 0.7071)
서쪽 ←	π	(0, 0, 1, 0)
남쪽 ↓	$-\pi/2$	(0, 0, -0.7071, 0.7071)

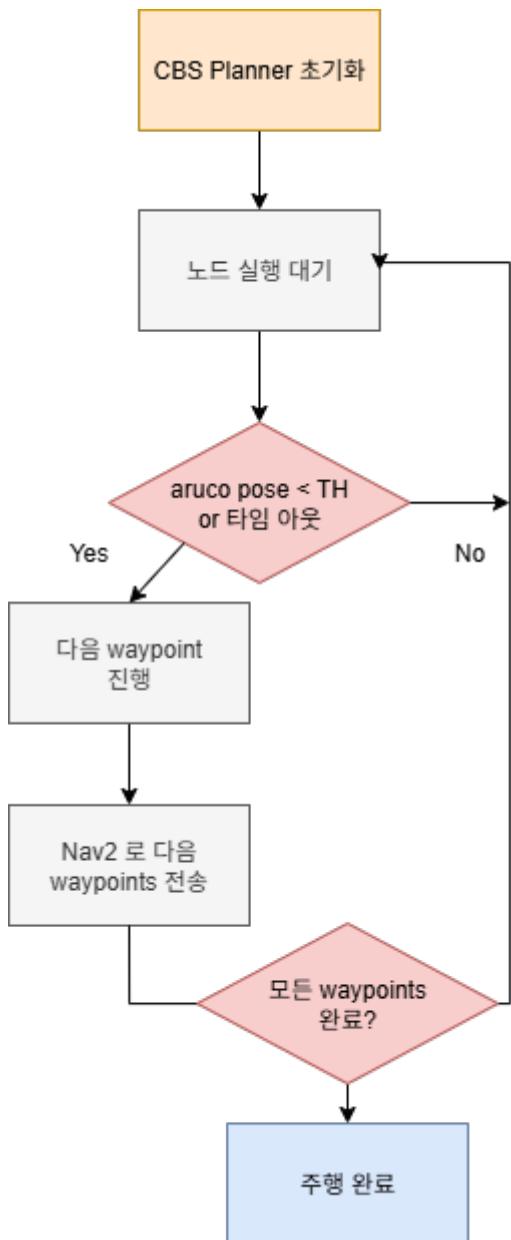
5.9.1.2 전체 코드 예시 (방향 포함)

```
void send_goal(double x, double y, double yaw_rad) {
    if (!nav_client_->wait_for_action_server(1s)) {
```

```
RCLCPP_WARN(this->get_logger(), "Nav2 action server not available!");
return;
}
NavigateToPose::Goal goal_msg;
goal_msg.pose.header.frame_id = "map";
goal_msg.pose.header.stamp = this->now();
goal_msg.pose.pose.position.x = x;
goal_msg.pose.pose.position.y = y;

tf2::Quaternion q;
q.setRPY(0, 0, yaw_rad); // yaw_rad: 원하는 각도(라디안)
goal_msg.pose.pose.orientation.x = q.x();
goal_msg.pose.pose.orientation.y = q.y();
goal_msg.pose.pose.orientation.z = q.z();
goal_msg.pose.pose.orientation.w = q.w();

nav_client_->async_send_goal(goal_msg);
RCLCPP_INFO(this->get_logger(), "Goal sent to Nav2: (%.2f, %.2f, %.2f rad)", x, y,
yaw_rad);
}
```



CBS path Planner 코드

```

#include "rclcpp/rclcpp.hpp"
#include "geometry_msgs/msg/pose_stamped.hpp"
#include "nav2_msgs/action/navigate_to_pose.hpp"
#include "rclcpp_action/rclcpp_action.hpp"
#include "mapf_planner/cbs.hpp"
#include "visualization_msgs/msg/marker_array.hpp"

#include <vector>
#include <cmath>

using NavigateToPose = nav2_msgs::action::NavigateToPose;
using namespace std::chrono_literals;
  
```

```

struct WaypointWithYaw {
    double x, y, yaw;
};

struct RobotData {
    std::vector<WaypointWithYaw> waypoints;
    size_t waypoint_index{0};
    bool goal_sent{false};
    geometry_msgs::msg::PoseStamped current_pose;
    rclcpp_action::Client<NavigateToPose>::SharedPtr action_client;
    rclcpp::Publisher<visualization_msgs::msg::MarkerArray>::SharedPtr marker_pub;
    rclcpp::Subscription<geometry_msgs::msg::PoseStamped>::SharedPtr pose_sub;
    rclcpp::TimerBase::SharedPtr timeout_timer; // ← 타임아웃 타이머

    static constexpr double THRESHOLD = 0.04;      // 도달 판단 기준 [m]
    static constexpr double THRESH_SQ = THRESHOLD * THRESHOLD;
};

class CBSPlannerNode : public rclcpp::Node {
public:
    CBSPlannerNode()
        : Node("cbs_planner_node_aruco3_timeout")
    {
        RCLCPP_INFO(get_logger(), "CBS Planner Node (aruco_pose3 with timeout) Start");

        // 1) 맵 & CBS 경로 계산
        std::vector<std::vector<bool>> map = {
            {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}, // 19x19 grid
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
            {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
        };
        std::vector<Position> starts = {{4,19},{3,19},{2,19}};
        std::vector<Position> goals = {{5,13},{5,12},{5,11}};
        CBSSolver solver(map, starts, goals);
        auto paths = solver.findSolution(true);
        if (paths.size() < 3) {
            RCLCPP_ERROR(get_logger(), "경로 계산 실패 또는 3번 로봇 경로 없음");
            return;
        }

        constexpr double RES = 0.1;
        // 2) robot3_ 초기화
        robot3_.action_client =
            rclcpp_action::create_client<NavigateToPose>(&this, "/navigate_to_pose3");
        robot3_.marker_pub =
            create_publisher<visualization_msgs::msg::MarkerArray>(
                "/waypoints_marker3", 10);
    }
}

```

```

robot3_.pose_sub =
    create_subscription<geometry_msgs::msg::PoseStamped>(
        "/aruco_pose3", rclcpp::QoS(10),
        [this](auto msg){ this->handle_pose(msg); });

// paths[2] (3번 로봇) → 실제 좌표 변환
for (auto &cell : paths[2]) {
    double x = (cell.second - 1) * RES + RES/2.0;
    double y = (cell.first - 1) * RES + RES/2.0;
    double yaw = M_PI/2.0; // 원하는 yaw
    robot3_.waypoints.push_back({x,y,yaw});
}

// 3) 마커 퍼블리시 타이머
marker_timer_ = create_wall_timer(
    200ms, [this]() { publish_markers(); });
}

private:
// /aruco_pose3 콜백: 도달 or 타임아웃 처리
void handle_pose(const geometry_msgs::msg::PoseStamped::SharedPtr msg) {
    auto &rd = robot3_;
    rd.current_pose = *msg;

    // 1) 처음 목표 전송
    if (!rd.goal_sent && rd.waypoint_index < rd.waypoints.size()) {
        send_goal(rd.waypoints[rd.waypoint_index]);
        rd.goal_sent = true;
        start_timeout(); // 타임아웃 타이머 시작
        return;
    }

    // 2) 거리 제곱으로 도달 체크
    const auto &wp = rd.waypoints[rd.waypoint_index];
    double dx = rd.current_pose.pose.position.x - wp.x;
    double dy = rd.current_pose.pose.position.y - wp.y;
    if (dx*dx + dy*dy < RobotData::THRESH_SQ) {
        RCLCPP_INFO(get_logger(),
                    "aruco_pose3 reached wp[%zu] (%.2f, %.2f), dist=%.3f",
                    rd.waypoint_index, wp.x, wp.y, std::sqrt(dx*dx+dy*dy));
        rd.timeout_timer->cancel(); // 도달했으니 타이머 취소
        advance_waypoint();
    }
}

// Nav2에 goal 전송
void send_goal(const WaypointWithYaw &wp) {
    auto &rd = robot3_;
    if (!rd.action_client->wait_for_action_server(1s)) {
        RCLCPP_WARN(get_logger(), "[3] action server not available");
        return;
    }
    NavigateToPose::Goal goal;
    goal.pose.header.frame_id = "map";
    goal.pose.header.stamp = now();
    goal.pose.pose.position.x = wp.x;
    goal.pose.pose.position.y = wp.y;
}

```

```

goal.pose.pose.orientation.z = std::sin(wp.yaw/2.0);
goal.pose.pose.orientation.w = std::cos(wp.yaw/2.0);

auto opts = rclcpp_action::Client<NavigateToPose>::SendGoalOptions();
opts.goal_response_callback =
    [this](auto handle){
        if (!handle) RCLCPP_ERROR(get_logger(),"[3] goal rejected");
    };
robot3_.action_client->async_send_goal(goal, opts);

RCLCPP_INFO(get_logger(),
            "[3] Sent goal → (%.2f, %.2f)", wp.x, wp.y);
}

// 타임아웃 타이머 시작
void start_timeout() {
    auto &rd = robot3_;
    // 이전 타이머가 있으면 취소
    if (rd.timeout_timer && rd.timeout_timer->is_oneshot() == false) {
        rd.timeout_timer->cancel();
    }
    // 5초 후 호출, one-shot 효과를 위해 콜백 내에서 cancel
    rd.timeout_timer = create_wall_timer(
        5s,
        [this]() {
            auto &rd = robot3_;
            RCLCPP_WARN(get_logger(), "[3] waypoint[%zu] timeout after 5s",
                        rd.waypoint_index);
            rd.timeout_timer->cancel();
            advance_waypoint();
        });
}
}

// 다음 웨이포인트로 진행
void advance_waypoint() {
    auto &rd = robot3_;
    rd.waypoint_index++;
    rd.goal_sent = false;
    if (rd.waypoint_index < rd.waypoints.size()) {
        send_goal(rd.waypoints[rd.waypoint_index]);
        rd.goal_sent = true;
        start_timeout();
    } else {
        RCLCPP_INFO(get_logger(), "[3] all waypoints done");
    }
}

// Marker publish
void publish_markers() {
    visualization_msgs::msg::MarkerArray arr;
    int cnt = 0;
    for (size_t i = robot3_.waypoint_index;
         i < robot3_.waypoints.size(); ++i)
    {
        const auto &wp = robot3_.waypoints[i];
        visualization_msgs::msg::Marker m;
        m.header.frame_id = "map";

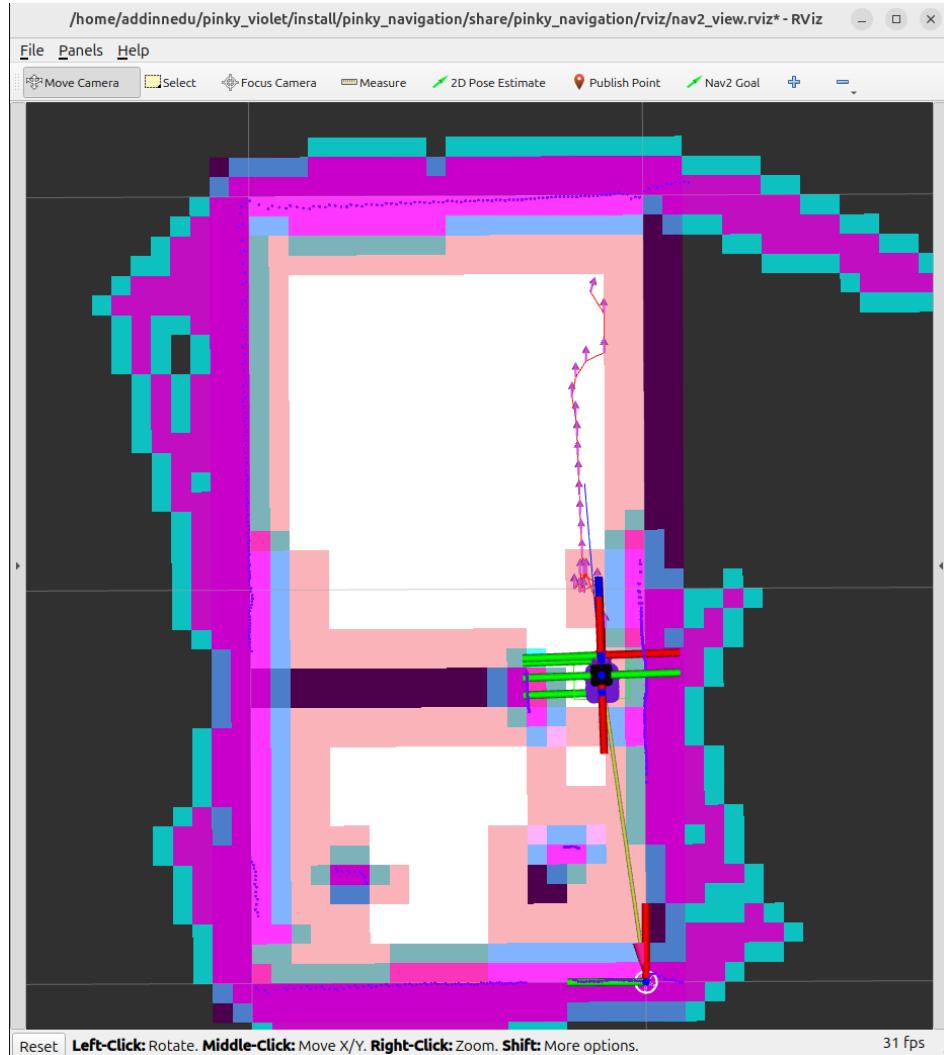
```

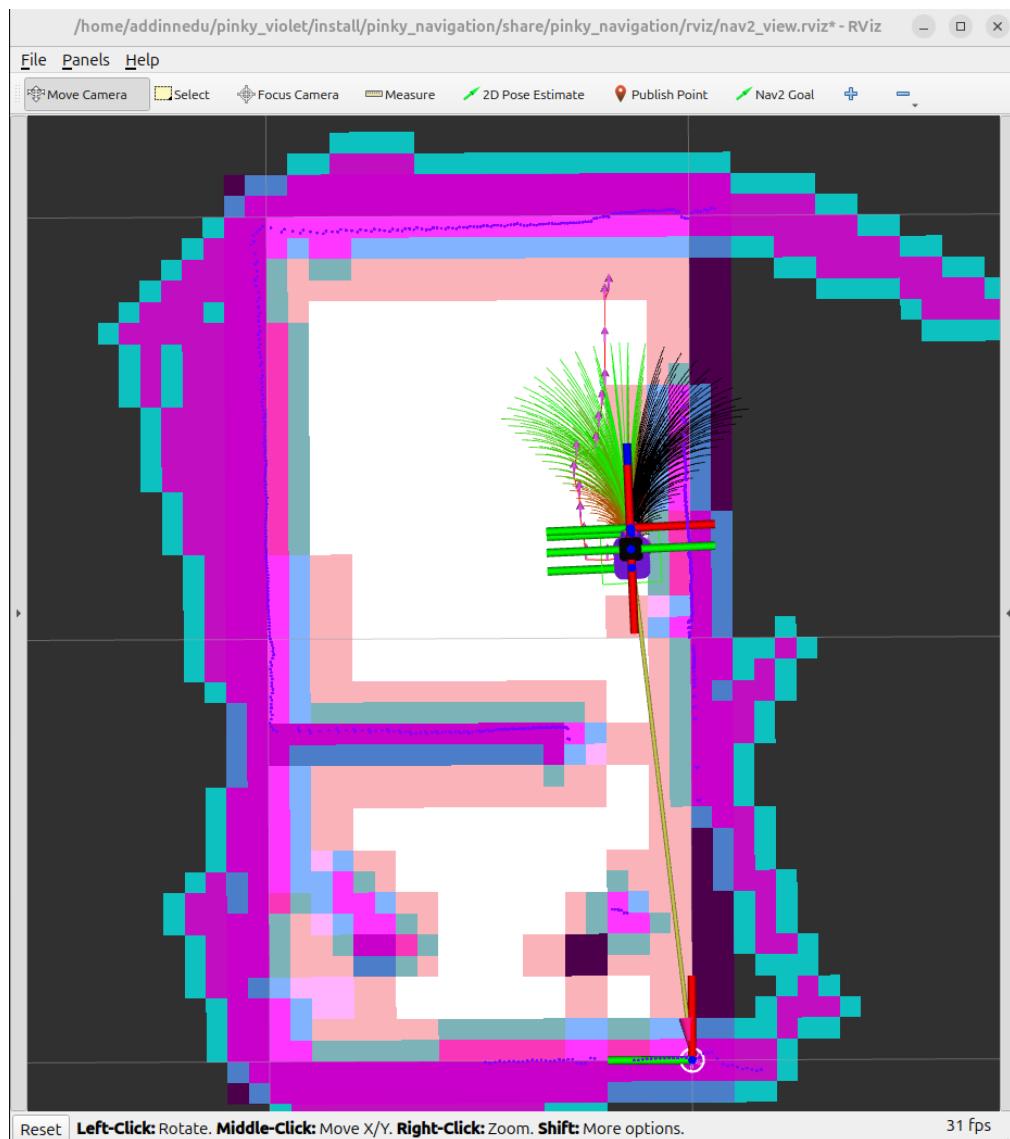
```
m.header.stamp = now();
m.ns = "wp_3";
m.id = cnt++;
m.type = visualization_msgs::msg::Marker::SPHERE;
m.pose.position.x = wp.x;
m.pose.position.y = wp.y;
m.pose.position.z = 0.05;
m.scale.x = m.scale.y = m.scale.z = 0.05;
m.color.r = 1.0f; m.color.g = 0.5f;
m.color.b = 0.0f; m.color.a = 1.0f;
arr.markers.push_back(m);
}
robot3_.marker_pub->publish(arr);
}

RobotData robot3_;
rclcpp::TimerBase::SharedPtr marker_timer_;
};

int main(int argc, char **argv) {
rclcpp::init(argc, argv);
rclcpp::spin(std::make_shared<CBSPlannerNode>());
rclcpp::shutdown();
return 0;
}
```

5.9.2 Global path 지그재그 현상



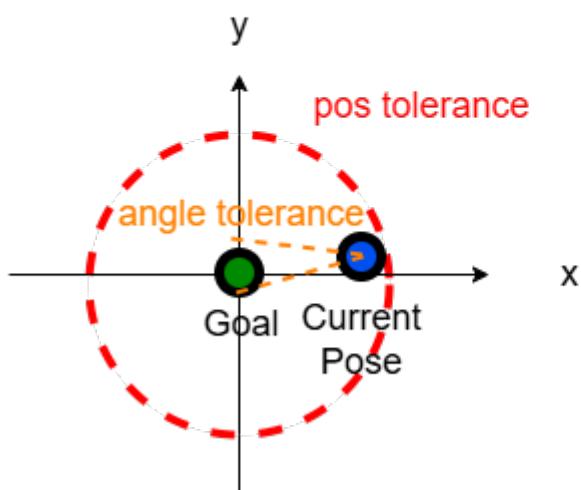
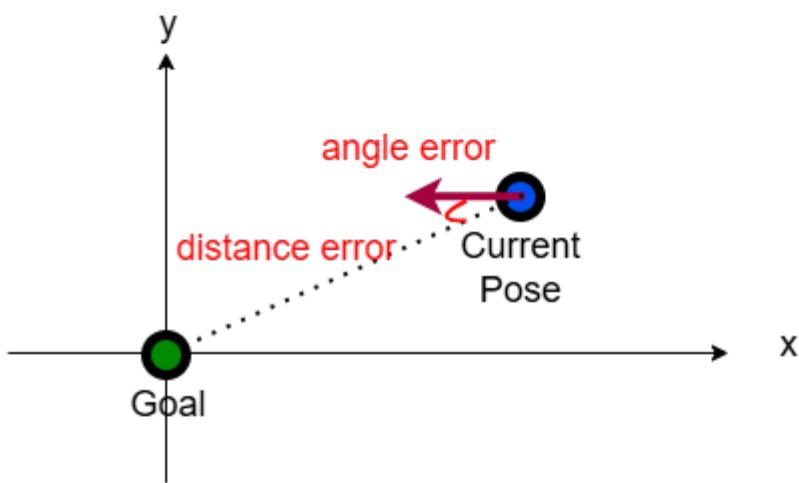
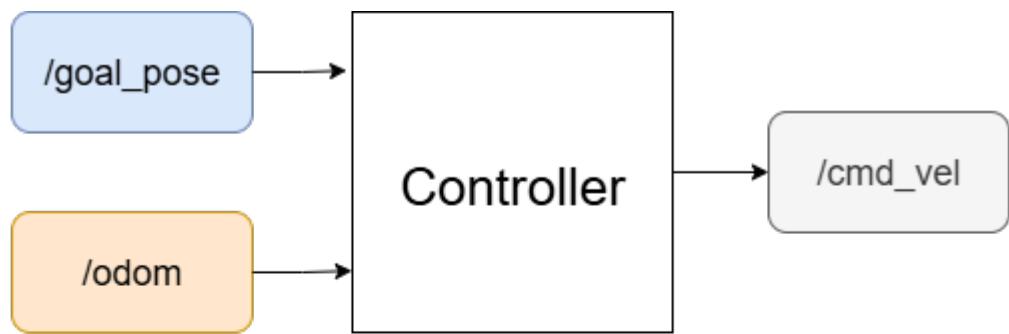


<https://youtu.be/KUHdw2tP2Nc>

<https://youtu.be/k0v4tgKJ3Qs>

5.9.3 GoalPoint Approaching Control

목표 위치까지 이동하는 제어기



5.10 <RT-17> <곽제우> <멀티 로봇 위치 ROS2 연동(Nav2 package)> <2025.7.22>

5.10.1 ROS 버전 Aruco Maker 실행 방법

```
ros2 run aruco_marker_pkg aruco_path_tracker
```

5.10.2 ARUCO MAKER 설정

udev rule 적용

```
ls -l /dev/video*
#video2에 대한 설정 변경
udevadm info --name=/dev/video2 --attribute-walk

#내용 입력
sudo gedit /etc/udev/rules.d/99-camera.rules
SUBSYSTEM=="video4linux", ATTR{name}=="Lenovo FHD Webcam Audio: Lenovo",
ATTRS{interface}=="Lenovo FHD Webcam", SYMLINK+="video_cam"
```

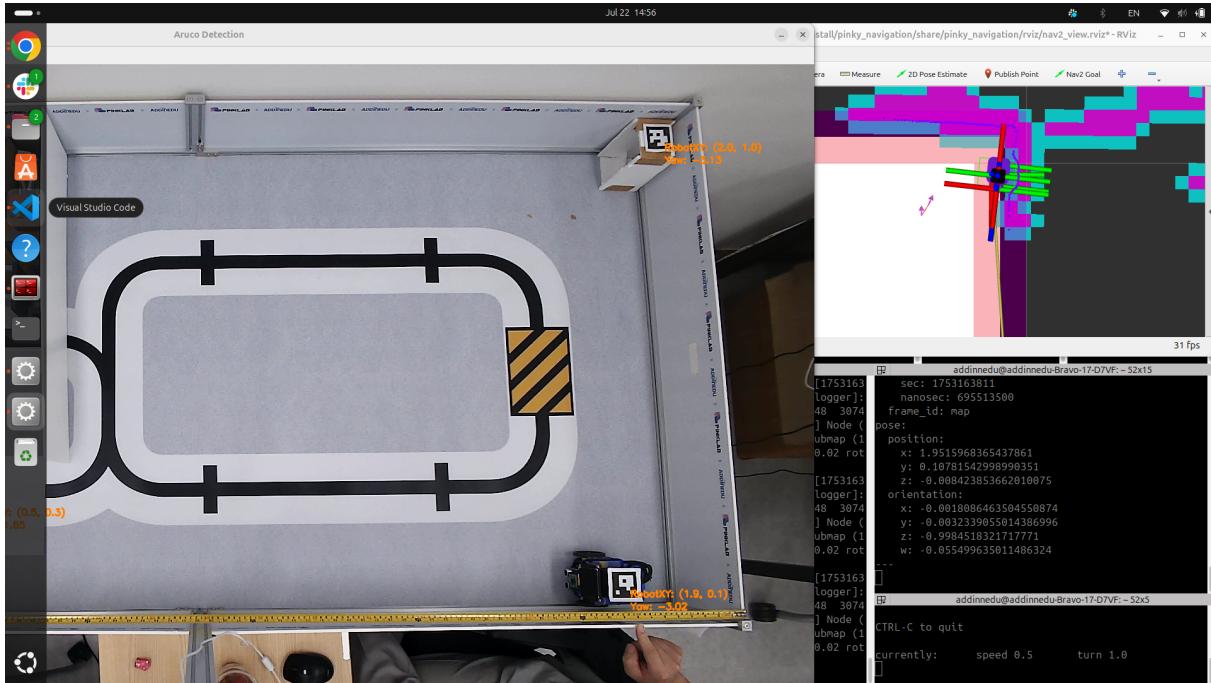
```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

Lenovo FHD Webcam Audio: Lenovo 의 장치 이름은 “video_cam”이 된다.

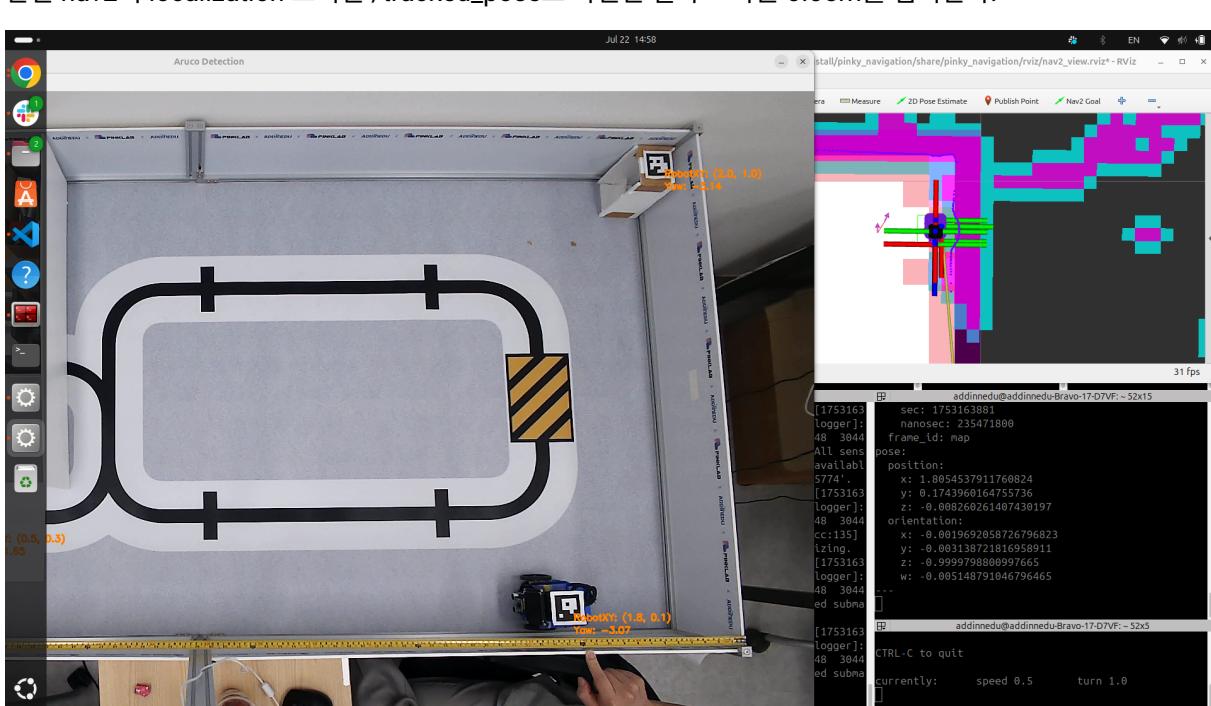
aruco code에서

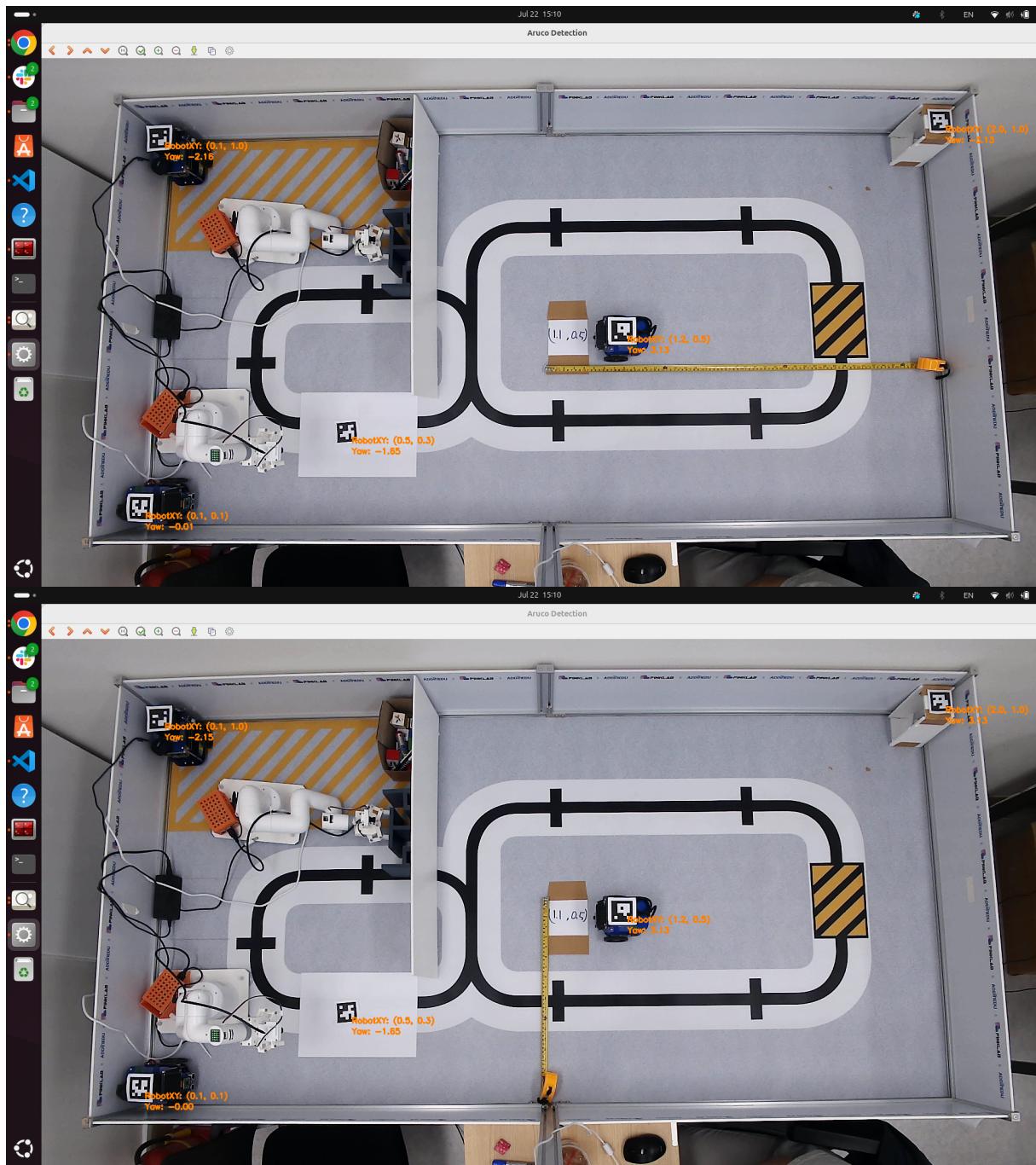
```
self.cap = cv2.VideoCapture(2)
->
self.cap = cv2.VideoCapture("/dev/video_cam")
으로 변경한다.
```

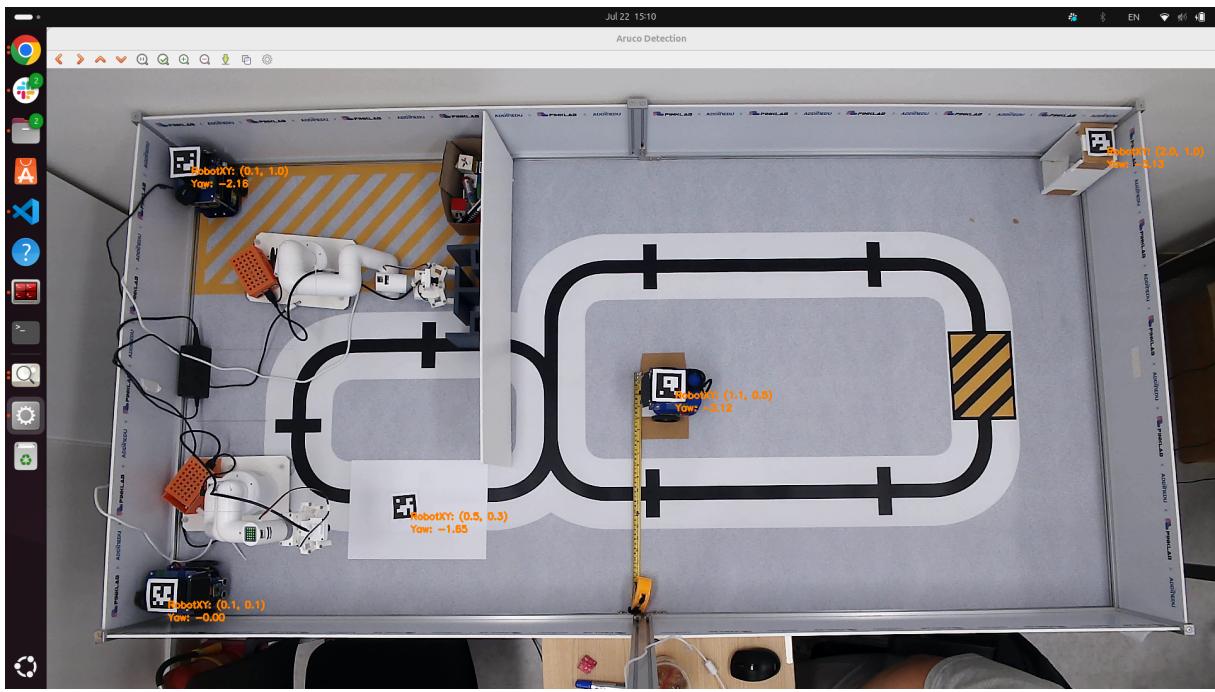
5.10.2.1 위치 정확성 테스트



반면 nav2의 localization 토픽인 /tracked_pose로 확인한 결과 오차는 0.05m를 넘어간다.







다음으로 확인 할 것은 waypoint를 따서 $(2, 0.1) \rightarrow (1.1, 0.5)$ 위치로 이동하는데 astar로 경로 계획을 하고 3초 간격으로 waypoint를 주면 $(1.1, 0.5)$ 로 이동하도록 한다.

1. 헤더 파일에서 test_map01.yaml과 test_map.pgm을 받아서 map 배열을 astar에게 넘겨준다.
2. aruco_detector.py에서 로봇의 위치를 publish하면 waypoint_arrival.cpp에서 로봇이 waypoint에 도달 했는지 여부를 확인하고 도달하면 waypoint_arrived를 true로 출력하고 3초가 지났는지 여부를 확인하고 3초가 지났으면 다음 waypoint로 넘어간다.
3. 마지막 waypoint(Goal point)에 도달하면 goal_arrived 를 publish한다.

5.10.3 Tracked_pose VS 관제 카메라 Robot_pose

https://github.com/addinedu-roscamp-5th/roscamp-repo-3/tree/dev_mapf

dev_mapf branch로 깃 클론 실행한다.

```
git clone -b dev_mapf https://github.com/addinedu-roscamp-5th/roscamp-repo-3.git
```

해당 패키지를 colcon build 및 source install/setup.bash 한다.

</tracked_pose를 path로 출력하는 코드>

```
ros2 launch mapf_planner multi_node_launch.py
```

실행하면 (1.1,0.5) => (2,0.1)을 왕복 5회 하는 코드가 실행되고
/tracked_pose를 nav_msgs/msg/Path 타입으로 누적해서 rviz상에 띄운다.

</aruco_tracked_pose1 를 path로 출력하는 코드>

```
ros2 run aruco_marker_pkg aruco_path_tracker
```

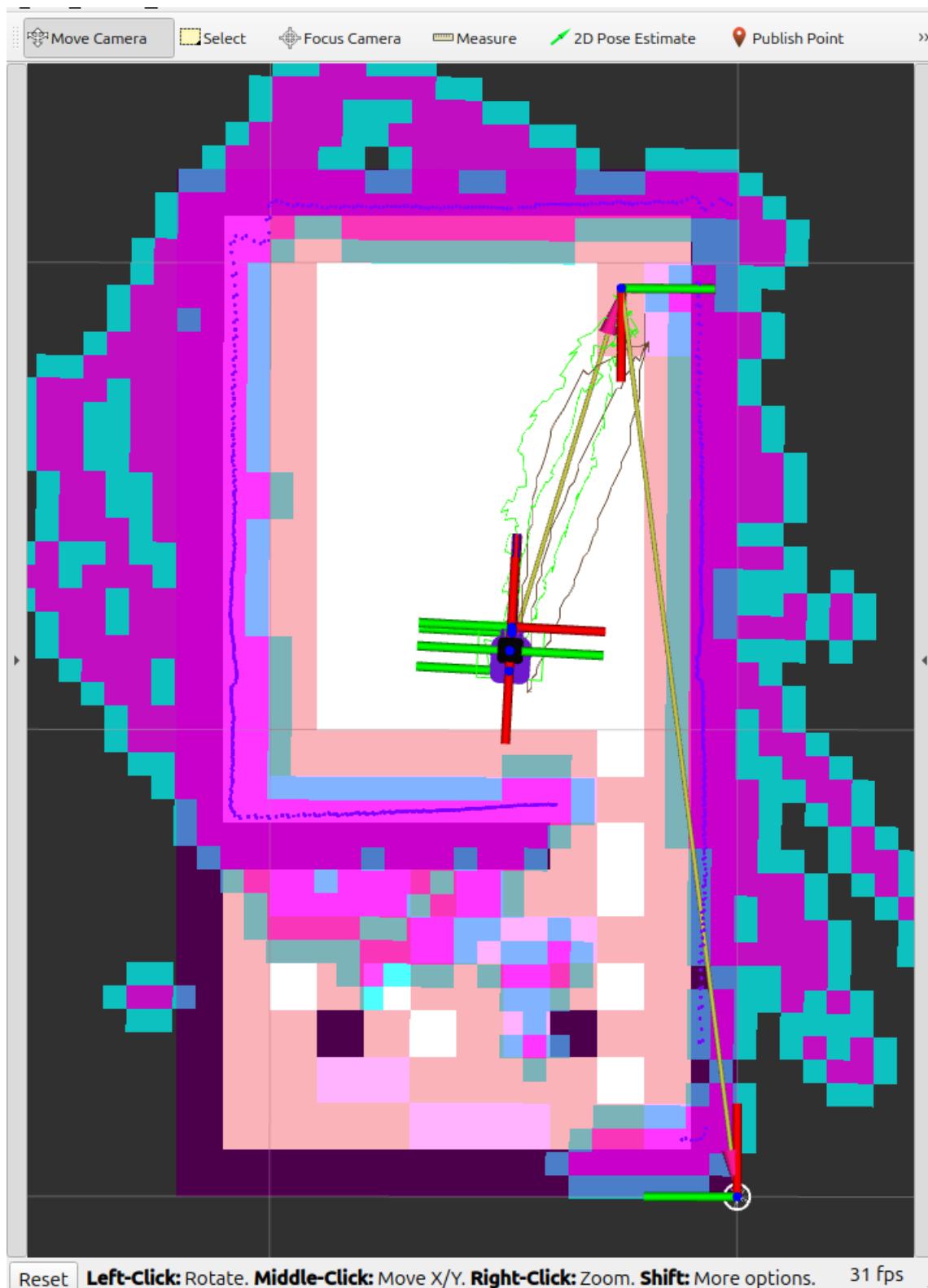
ArUco 마커 위치 (robot_xy)를 PoseStamped 로 변환

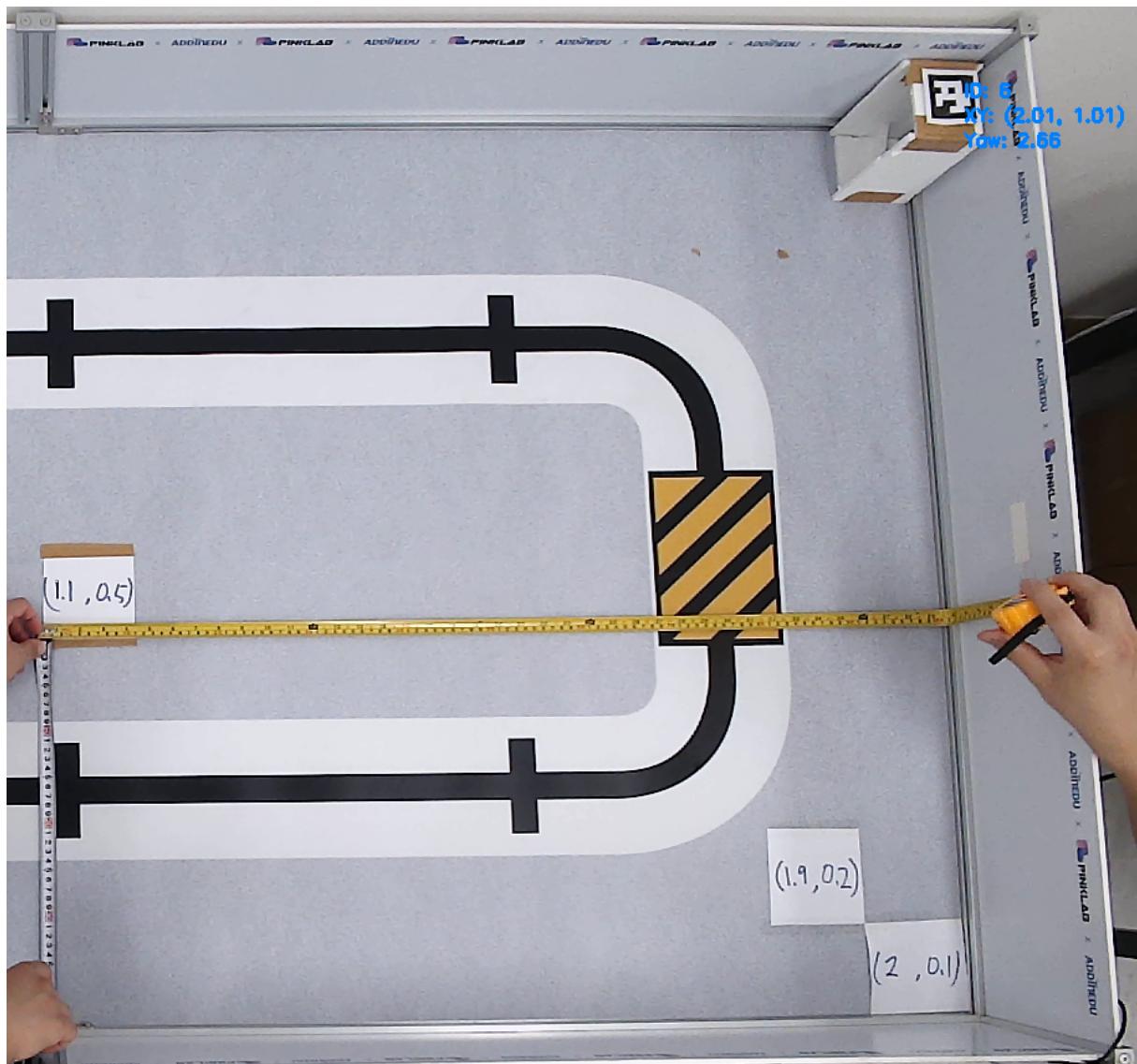
nav_msgs/msg/Path 타입으로 누적해서 /aruco_tracked_pose 에 퍼블리시

RViz에서 Path로 경로 확인

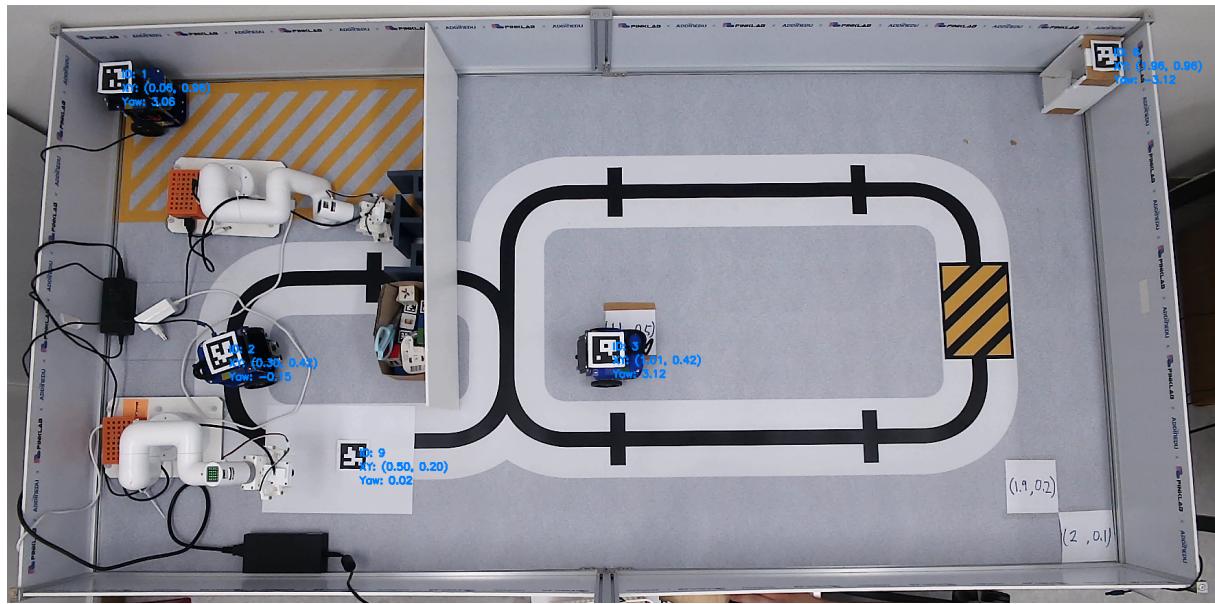
초록색 : /tracked_pose

갈색 : /aruco_tracked_pose1





14 실제 위치 (1,0.4)



15 aruco marker (1.01, 0.42)오차 (x: 0.01, y:0.02)

```

---  

header:  

  stamp:  

    sec: 1753248501  

    nanosec: 672405800  

  frame_id: map  

pose:  

  position:  

    x: 1.1695884329171766  

    y: 0.49128664450893583  

    z: -0.003327645760657183  

  orientation:  

    x: -0.005348607012017992  

    y: 0.003632383828654541  

    z: 0.7103303319895871  

    w: 0.7038387724801153
---
```

16 cartographer의 tracked_pose(1.16,0.49)오차 (x: 0.16, y:0.09)

GT	/tracked_pose	Error	/aruco_tracked_pose	Error
(1.0,0.4)	(1.16,0.49)	(0.16, 0.09)	(1.01, 0.42)	(0.01, 0.02)
(1.9,0.2)	(1.82, 0.31)	(0.08, 0.11)	(1.85, 0.15)	(0.05, 0.05)

GT	/tracked_pose	Error	/aruco_tracked_pose	Error
(0.8,0.6)	(0.93, 0.74)	(0.13, 0.14)	(0.82, 0.62)	(0.02, 0.02)
(1.3,0.3)	(1.39, 0.46)	(0.09, 0.16)	(1.33, 0.3)	(0.03, 0.00)
(1.1, 0.5)	(1.20, 0.50)	(0.10, 0.0)	(1.10, 0.50)	(0.00, 0.00)

아루코 마커 yaw 반영

```

import cv2
import numpy as np
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
from geometry_msgs.msg import PoseStamped
from nav_msgs.msg import Path
import json
import math

def yaw_to_quaternion(yaw):
    """Z축 회전(yaw, radian)을 쿼터니언(x, y, z, w)으로 변환."""
    q = {}
    q['x'] = 0.0
    q['y'] = 0.0
    q['z'] = math.sin(yaw / 2.0)
    q['w'] = math.cos(yaw / 2.0)
    return q

class ArucoDetector:
    def __init__(self, camera_matrix, dist_coeffs, marker_length=0.05):
        self.aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_5X5_250)
        self.aruco_params = cv2.aruco.DetectorParameters_create()
        self.camera_matrix = camera_matrix
        self.dist_coeffs = dist_coeffs
        self.marker_length = marker_length

        self.src_pts = np.array([
            [-0.96,  0.41],
            [-0.98, -0.49],
            [ 0.97,  0.39],
            [ 0.99, -0.53]
        ], dtype=np.float32)
        self.dst_pts = np.array([
            [0.05,  0.05],
            [0.05,  0.95],
            [1.95,  0.05],
            [1.95,  0.95]
        ], dtype=np.float32)
        self.H = cv2.getPerspectiveTransform(self.src_pts, self.dst_pts)

    def transform_point(self, pt):

```

```

src = np.array([pt[0], pt[1], 1.0], dtype=np.float32)
dst = self.H @ src
dst = dst / dst[2]
return float(dst[0]), float(dst[1])

def detect_markers(self, frame):
    corners, ids, _ = cv2.aruco.detectMarkers(frame, self.aruco_dict,
parameters=self.aruco_params)
    detected = []
    if ids is not None:
        rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(
            corners, self.marker_length, self.camera_matrix, self.dist_coeffs)
        for i in range(len(ids)):
            marker_id = int(ids[i][0])
            tvec = tvecs[i][0]
            rvec = rvecs[i][0]
            x, y = self.transform_point((tvec[0], tvec[1]))
            x, y = round(x, 2), round(y, 2)

            rot_mat, _ = cv2.Rodrigues(rvec)
            yaw = np.arctan2(rot_mat[1, 0], rot_mat[0, 0]) - np.pi / 2
            yaw = (yaw + np.pi) % (2 * np.pi) - np.pi
            yaw = round(float(yaw), 3)

            corners_reshaped = corners[i].reshape((4, 2))
            center_x = int(np.mean(corners_reshaped[:, 0]))
            center_y = int(np.mean(corners_reshaped[:, 1]))

            detected.append({
                "id": marker_id,
                "robot_xy": [x, y],
                "yaw": yaw,
                "text_pos": (center_x, center_y)
            })
    return detected

class ArucoPathTracker(Node):
    def __init__(self, camera_matrix, dist_coeffs):
        super().__init__('aruco_path_tracker_id')
        self.detector = ArucoDetector(camera_matrix, dist_coeffs)
        self.cap = cv2.VideoCapture("/dev/video_cam")
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

        self.json_pub = self.create_publisher(String, 'detected_markers', 10)
        self.path_publishers = {} # id -> publisher for Path
        self.pose_publishers = {} # id -> publisher for PoseStamped
        self.paths = {} # id -> Path()

    def process_frame(self):
        ret, frame = self.cap.read()
        if not ret:
            self.get_logger().warn("Camera frame not received!")
            return

        detected = self.detector.detect_markers(frame)

```

```

        self.json_pub.publish(String(data=json.dumps([
            {"id": d["id"], "robot_xy": d["robot_xy"], "yaw": d["yaw"]} for d in
detected
])))

for marker in detected:
    marker_id = marker["id"]
    x, y = marker["robot_xy"]
    yaw = marker["yaw"]
    cx, cy = marker["text_pos"]

    # 화면 표시
    cv2.putText(frame, f"ID: {marker_id}", (cx + 10, cy),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 128, 0), 2)
    cv2.putText(frame, f"XY: {(x:.2f}, {y:.2f})", (cx + 10, cy + 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 128, 0), 2)
    cv2.putText(frame, f"Yaw: {yaw:.2f}", (cx + 10, cy + 40),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 128, 0), 2)

    # Path publisher
    path_topic = f"/aruco_path{marker_id}"
    if marker_id not in self.path_publishers:
        self.path_publishers[marker_id] = self.create_publisher(Path,
path_topic, 10)
        path = Path()
        path.header.frame_id = "map"
        self.paths[marker_id] = path

    # PoseStamped publisher
    pose_topic = f"/aruco_pose{marker_id}"
    if marker_id not in self.pose_publishers:
        self.pose_publishers[marker_id] = self.create_publisher(PoseStamped,
pose_topic, 10)

    # 현재 pose
    pose = PoseStamped()
    pose.header.stamp = self.get_clock().now().to_msg()
    pose.header.frame_id = "map"
    pose.pose.position.x = x
    pose.pose.position.y = y
    pose.pose.position.z = 0.0

    # 쿼터니언 적용
    q = yaw_to_quaternion(yaw)
    pose.pose.orientation.x = q['x']
    pose.pose.orientation.y = q['y']
    pose.pose.orientation.z = q['z']
    pose.pose.orientation.w = q['w']

    # Path 누적 + publish
    self.paths[marker_id].header.stamp = self.get_clock().now().to_msg()
    self.paths[marker_id].poses.append(pose)
    self.path_publishers[marker_id].publish(self.paths[marker_id])

    # PoseStamped publish
    self.pose_publishers[marker_id].publish(pose)

```

```
cv2.imshow("Aruco Path Tracker by ID", frame)
cv2.waitKey(1)

def destroy_node(self):
    self.cap.release()
    cv2.destroyAllWindows()
    super().destroy_node()

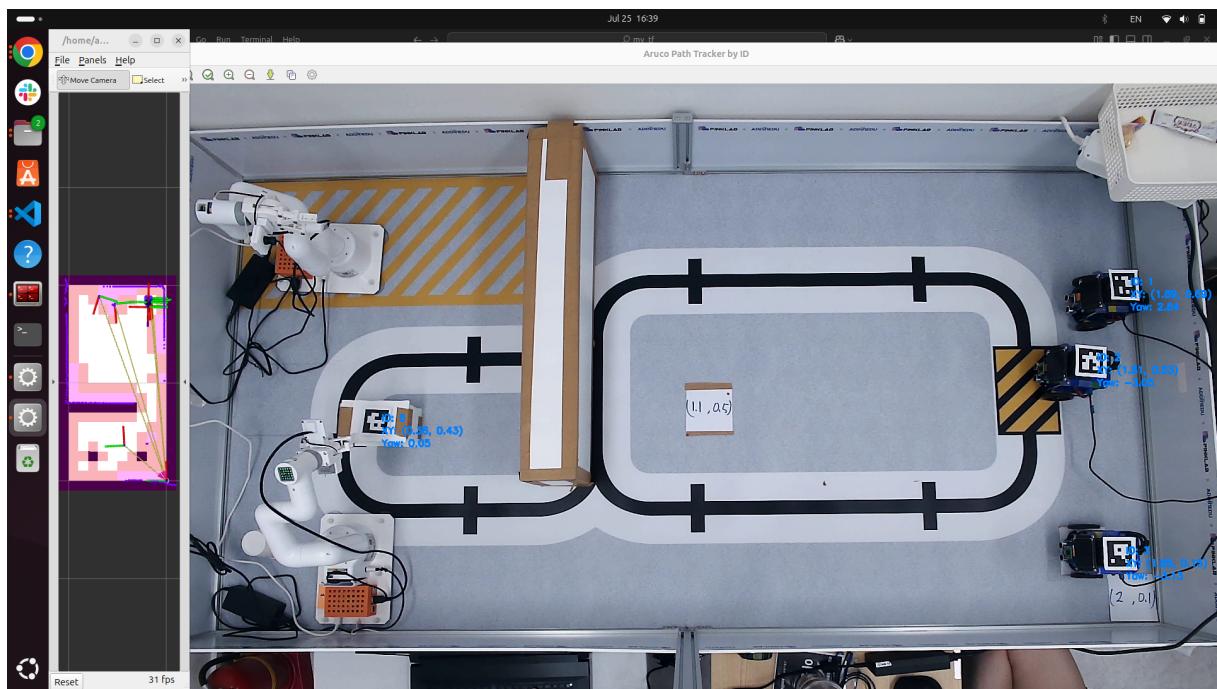
def main():
    rclpy.init()
    base_path = '/home/addinenedu/monitoring_camera_ws/src/aruco_marker_pkg/include/'
    camera_matrix = np.load(base_path + 'camera_matrix.npy')
    dist_coeffs = np.load(base_path + 'dist_coeffs.npy')

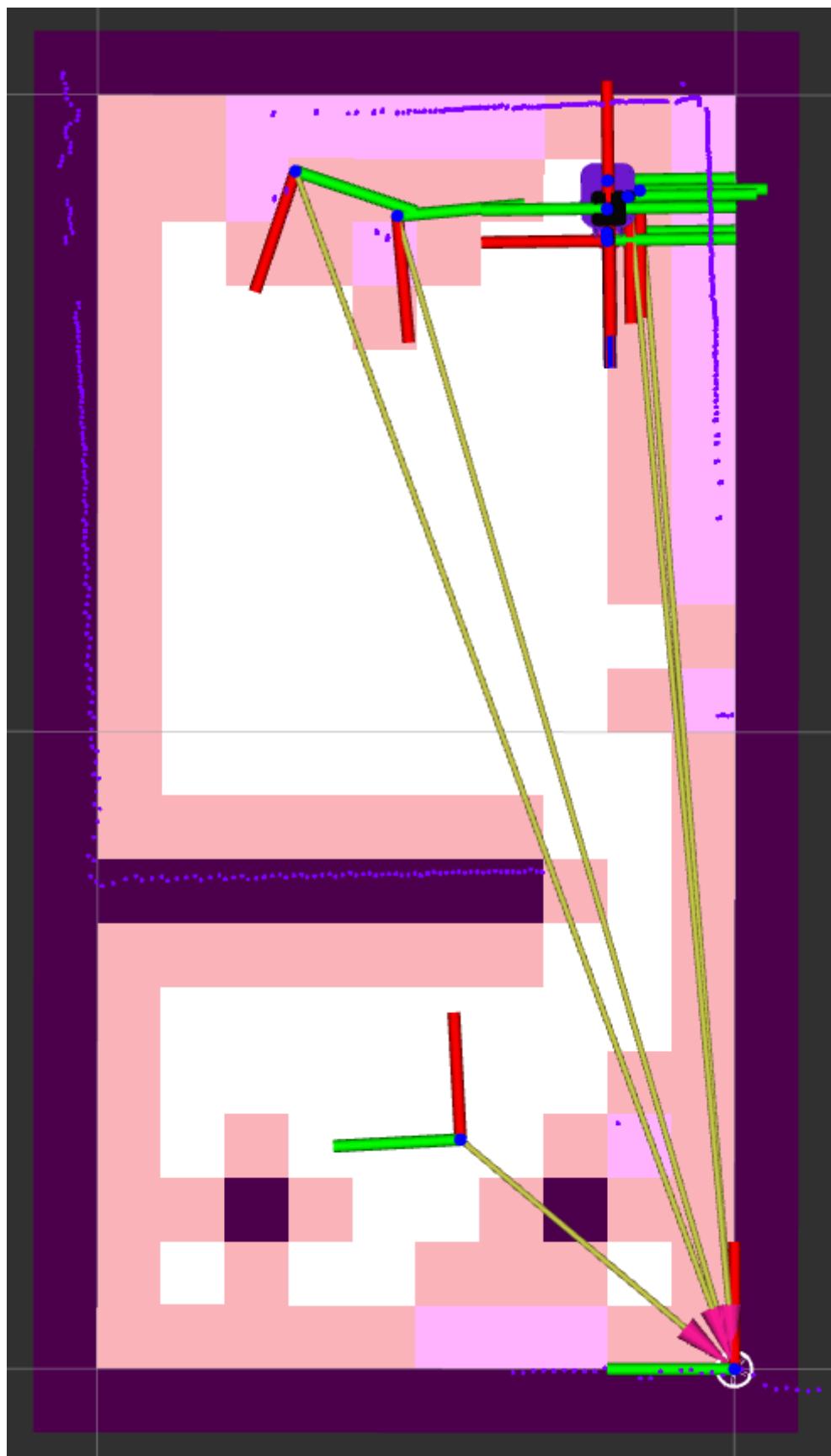
    node = ArucoPathTracker(camera_matrix, dist_coeffs)
    try:
        while rclpy.ok():
            node.process_frame()
            rclpy.spin_once(node, timeout_sec=0.01)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

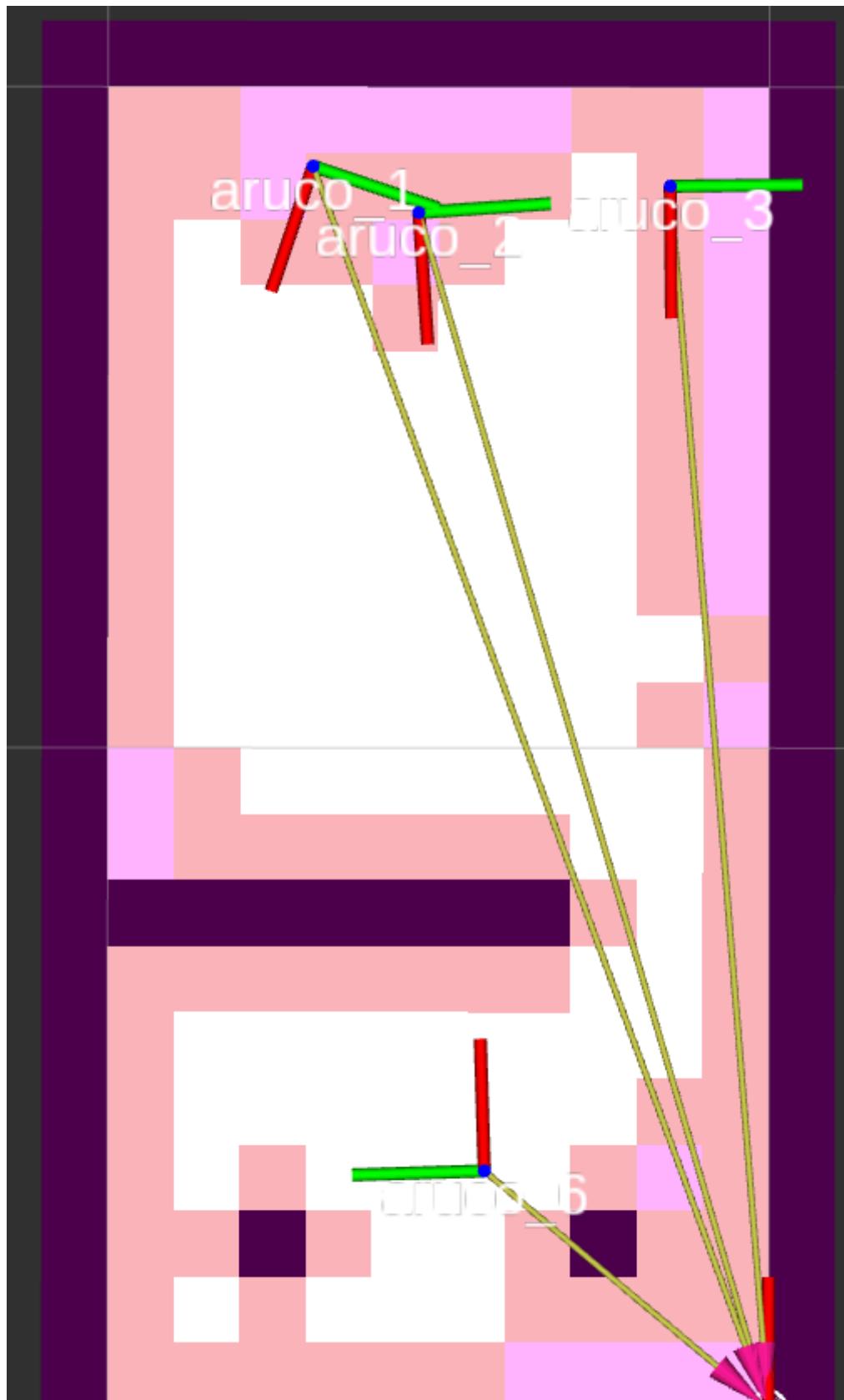
if __name__ == '__main__':
    main()
```

5.10.3.1 Aruco maker TF

```
ros2 run aruco_marker_pkg aruco_TF_broadcaster
```









아르코

5.10.3.2 Amcl launch 실행

참고 주소: https://docs.nav2.org/tutorials/docs/navigation2_on_real_turtlebot3.html

<https://github.com/ros-navigation/navigation2/blob/main/nav2 Bringup/launch/bringup.launch.py>

에서

nav2 Bringup/launch에서

bringup.launch.py

localization.launch.py

참고해서

nav2_amcl을 실행한다.

5.10.4 TF이란?

- **Transform: 좌표계 변환의 약자**

5.10.4.1 base_link와 base_footprint의 차이

base_footprint:

- 로봇이 지면에 투영된 점을 기준으로 하는 좌표계
- 일반적으로 지면(바닥) 높이에 위치 ($Z=0$)
- 로봇의 "발자국"이나 지면 접촉점을 나타냄
- 내비게이션이나 맵핑에서 주로 사용
- 지면을 기준으로 한 2D 경로 계획에 유용

base_link:

- 로봇의 실제 물리적 중심이나 주요 구조체를 기준으로 하는 좌표계
- 보통 로봇의 중앙 부분에 위치 (지면보다 높은 곳)
- 로봇의 실제 기하학적 중심점
- 센서 데이터 처리나 조작기 제어에서 주로 사용
- 로봇의 물리적 구조를 반영

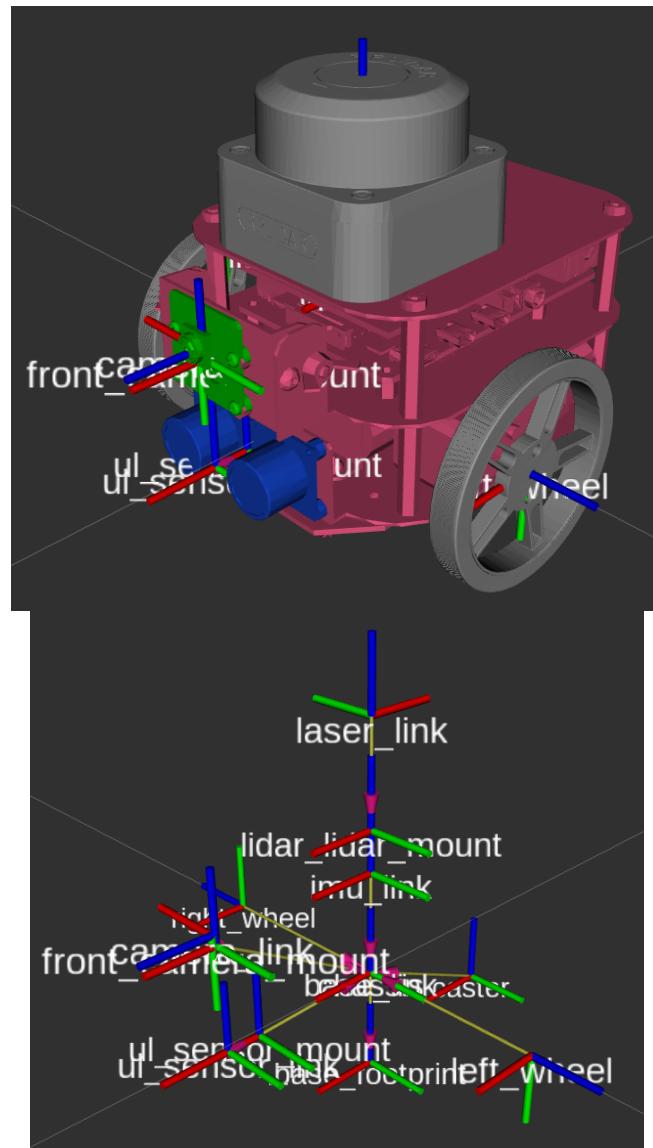
관계:

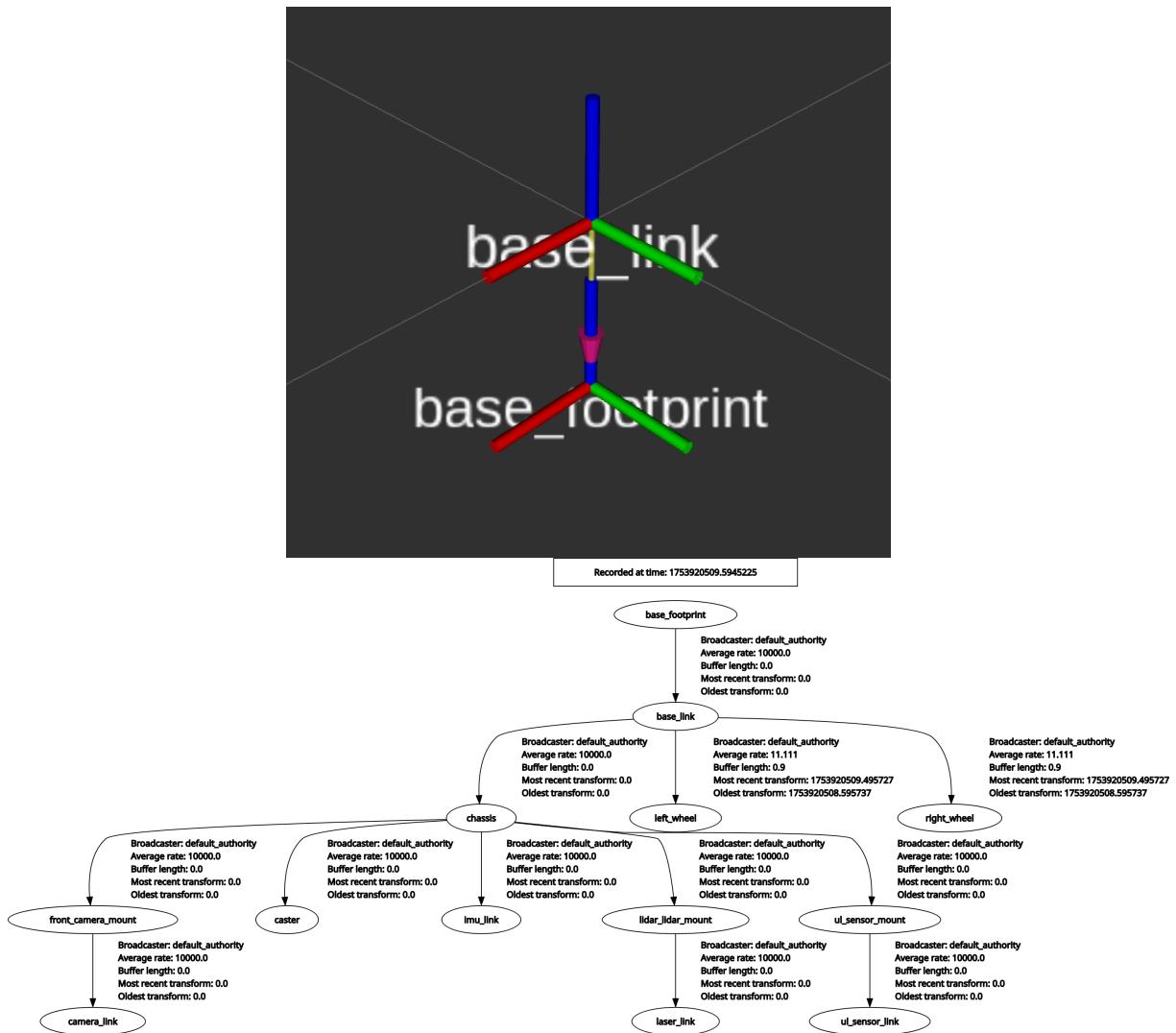
- base_footprint는 보통 base_link를 지면에 수직으로 투영한 위치
- 두 프레임 간에는 주로 Z축(높이) 방향으로만 차이가 남

- base_footprint → base_link 변환은 일반적으로 (0, 0, 로봇높이)의 translation만 포함

이렇게 구분하는 이유는 내비게이션은 2D 평면에서, 센서나 조작은 3D 공간에서 이루어지기 때문입니다.

5.10.4.2 기본 구조





5.10.4.3 odom & TF 교체

- Nav2에서 catographer의 odom과 TF를 관제 카메라의 aruco odom과 TF로 대체한다.
- aruco_odom_publihser.py 에서 odom과 aruco의 TF를 publish한다.
- Aruco 마커와 모바일 로봇의 `base_footprint` 를 정적 변환으로 연결하려면, ArUco 마커가 고정된 환경의 기준 좌표이고, `base_footprint` 는 로봇의 기준 좌표라면, 두 프레임 간의 상대 위치 관계를 수동으로 지정해야 합니다. 이를 위해 ROS2에서는 `static_transform_publisher` 를 사용할 수 있다.

```
ros2 run aruco_marker_pkg aruco_odom_publisher
```

```
ros2 run tf2_ros static_transform_publisher 0.5 0.0 0.0 0 0 1.57 odom
base_footprint
```

5.10.4.3.1 Launch 설정

cartographer.launch.xml

```
<?xml version='1.0' ?>
<launch>
    <arg name="use_sim_time" default="false"/>
    <arg name="cartographer_prefix" default="$(find-pkg-share pinky_cartographer)"/>
    <arg name="cartographer_config_dir" default="$(var cartographer_prefix)/params"/>
    <arg name="configuration_basename" default="cartographer_params.lua"/>
    <arg name="resolution" default="0.05"/>
    <arg name="publish_period_sec" default="1.0"/>

    <arg name="use_nav2" default="False"/>

    <node pkg="cartographer_ros" exec="cartographer_node" name='cartographer_node'
    args="--configuration_directory $(var cartographer_config_dir) -configuration_basename
    $(var configuration_basename)">
        <param name='use_sim_time' value="$(var use_sim_time)"/>
    </node>

    <node pkg="cartographer_ros" exec="cartographer_occupancy_grid_node"
    name='cartographer_occupancy_grid_node' args="--resolution $(var resolution)
    --publish_period_sec $(var publish_period_sec)" unless="$(var use_nav2)">
        <param name='use_sim_time' value="$(var use_sim_time)"/>
    </node>
    <!-- Static TF: aruco_3 → base_footprint -->
    <node pkg="tf2_ros" exec="static_transform_publisher" name="aruco_to_basefootprint"
    output="screen"
        args="0.1 0.2 0 0 0 1.57 odom base_footprint"/>
    </node>
</launch>
```

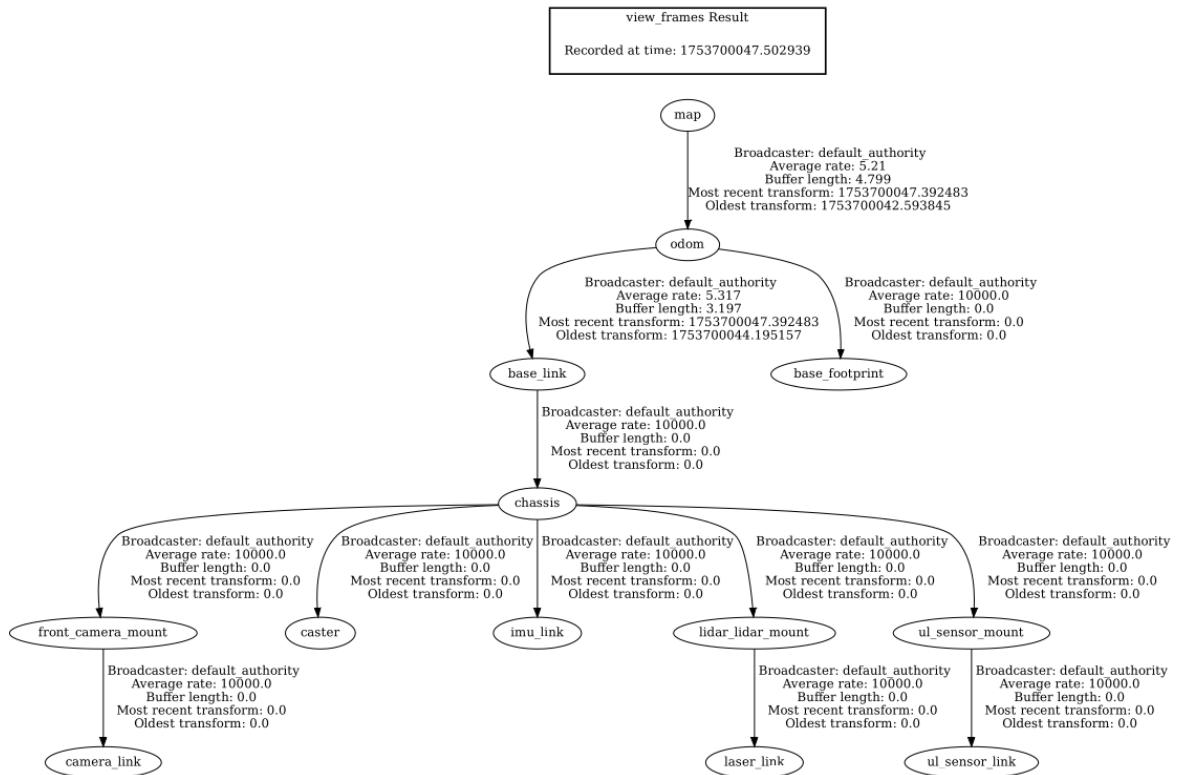
5.10.4.3.2 TF 확인

```
ros2 run tf2_tools view_frames
```

```
ros2 run tf2_ros tf2_echo aruco_1 base_footprint
```

- 이 방식은 `base_footprint` 가 ArUco 마커로부터 상대적으로 항상 고정된 위치에 있을 때만 유효합니다.

- 만약 로봇이 움직이며 ArUco를 보고 localization하는 구조라면, 정적 변환 대신 동적 TF(`tf2_ros::TransformBroadcaster`)를 사용해야 합니다.



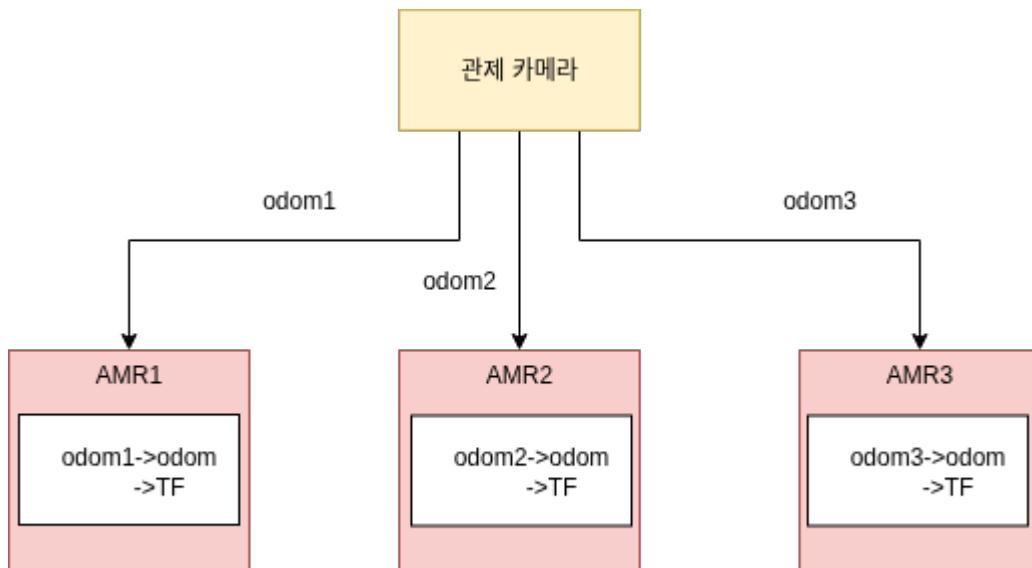
5.10.5 결과 확인!

- aruco maker도 x,y각각에 대해서 0.05m 오차를 갖는다.

https://www.youtube.com/watch?v=NNNUgljsV_4

5.10.5.1 개별 모바일 로봇에서 TF를 받는 방법

- 관제 PC에서 aruco maker에게 위치 속도에 대한 정보와 odom에 각 모바일 로봇의 ID를 붙여서 토픽으로 넘겨준다.
- 각 모바일 로봇은 개별적으로 `odom_{id}`를 `odom`과 TF으로 변환해 broadcast하는 과정을 거친다.
-> 이렇게 하면 Nav2를 직접적으로 수정하지 않고 개별 모바일 로봇에 TF를 전달할 수 있다.



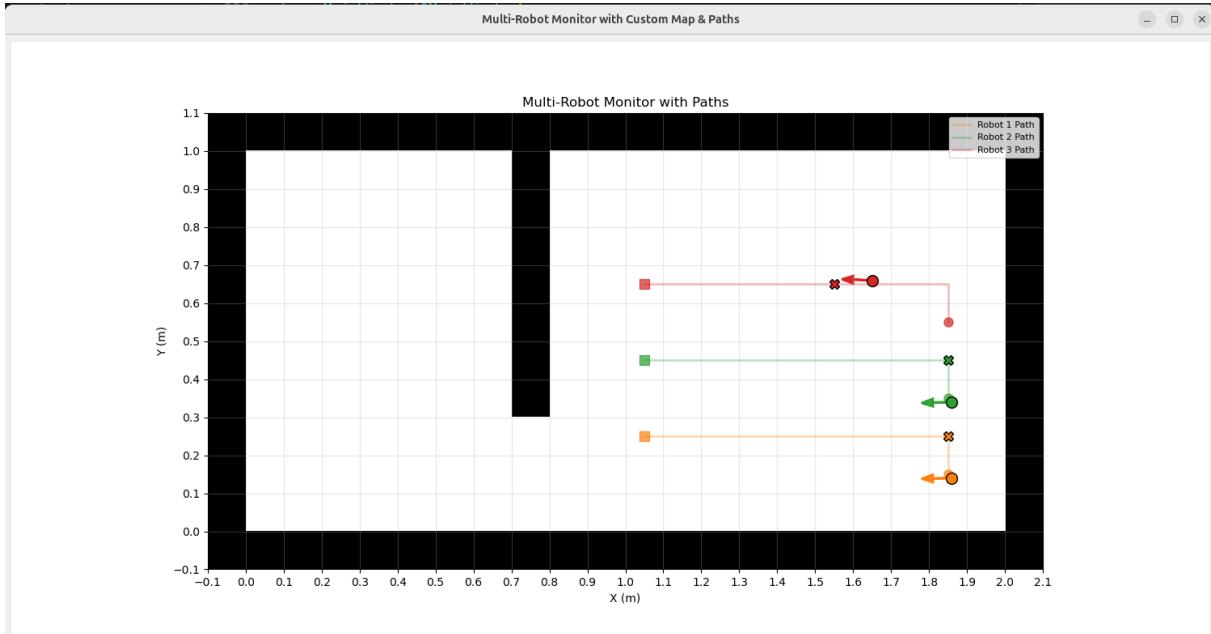
TF(odom->base_link)

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_monitoring_TF.yaml
```

domain_bridge_monitoring_TF.yaml

```
# Name of the domain bridge, used for node naming and logging
name: my_bridge
topics:
  odom_1:
    type: nav_msgs/msg/Odometry
    from_domain: 59
    to_domain: 51
  odom_2:
    type: nav_msgs/msg/Odometry
    from_domain: 59
    to_domain: 52
  odom_3:
    type: nav_msgs/msg/Odometry
    from_domain: 59
    to_domain: 53
```

5.10.5.2 PYQT를 이용해서 맵상에서 로봇의 위치를 GUI로 표현



5.11 <RT-23> <박민수> <로봇팔_Moveit> <7/28>

<https://automaticaddison.com/add-and-plan-around-objects-using-moveit-2-ros-2-jazzy/>

https://moveit.picknik.ai/main/doc/tutorials/your_first_project/your_first_project.html

https://moveit.picknik.ai/main/doc/examples/setup_assistant/setup_assistant_tutorial.html

왼쪽 하단 보면 code 라 하고 다운받는게 있음

visual tracking course

32.Jetson Orin SUPER - AI Visual Capture Course

33.Jetson Orin SUPER - Robotic Arm URDF Model Course

34.Jetson Orin SUPER - Moveit advanced kinematics course

35.FAQ

Download

- Instruction_Manual
- SystemFile_for_NANO4GB
- SystemFile_For_OrinNANO
- SystemFile_For_OrinNX
- VM_File
- Code
- 3D Model File
- Jetson_Boot_USB.tar.gz

JetCobot
7-Axis Visual Collaborative Robotic Arm

For Jetson NANO 4GB | For Jetson Orin NANO 8GB | For Jetson Orin NX 16GB

ROS system | Python programming
Inverse kinematics analysis | Moveit simulation
Deep learning | Mediapipe development

YAHBOOM

공유 문서함 > 5.JetCobot_Code

유형

이름

For_JetPack6.2_ORIN-SUPER

For_JetPack6.2_ORIN-SUPER 공유 폴더

For_JetsonNANOBO1_ORIN

소유자

yahboomsmart01

yahboomsmart01

이거 써서 urdf 불러온걸 그대로 저기에 넣어주면 됨 (넣어줄때도 colcon build 하고 source install/local.setup.bash 해주면 된다)

튜토리얼 쭉 해주면 됨

/home/addinedu/jetcobot_colcon_ws/src/jetcobot_moveit/config 이 위치에 있는 joint_limits.yaml 이걸 수정함

```

# joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed

# For beginners, we downscale velocity and acceleration limits.
# You can always specify higher scaling factors (<= 1.0) in your motion requests. # Increase the values below to 1.0 to always move at maximum speed
default_velocity_scaling_factor: 0.1
default_acceleration_scaling_factor: 0.1

# Specific joint properties can be changed with the keys [max_position, min_position, max_velocity, max_acceleration]
# Joint limits can be turned off with [has_velocity_limits, has_acceleration_llimits]

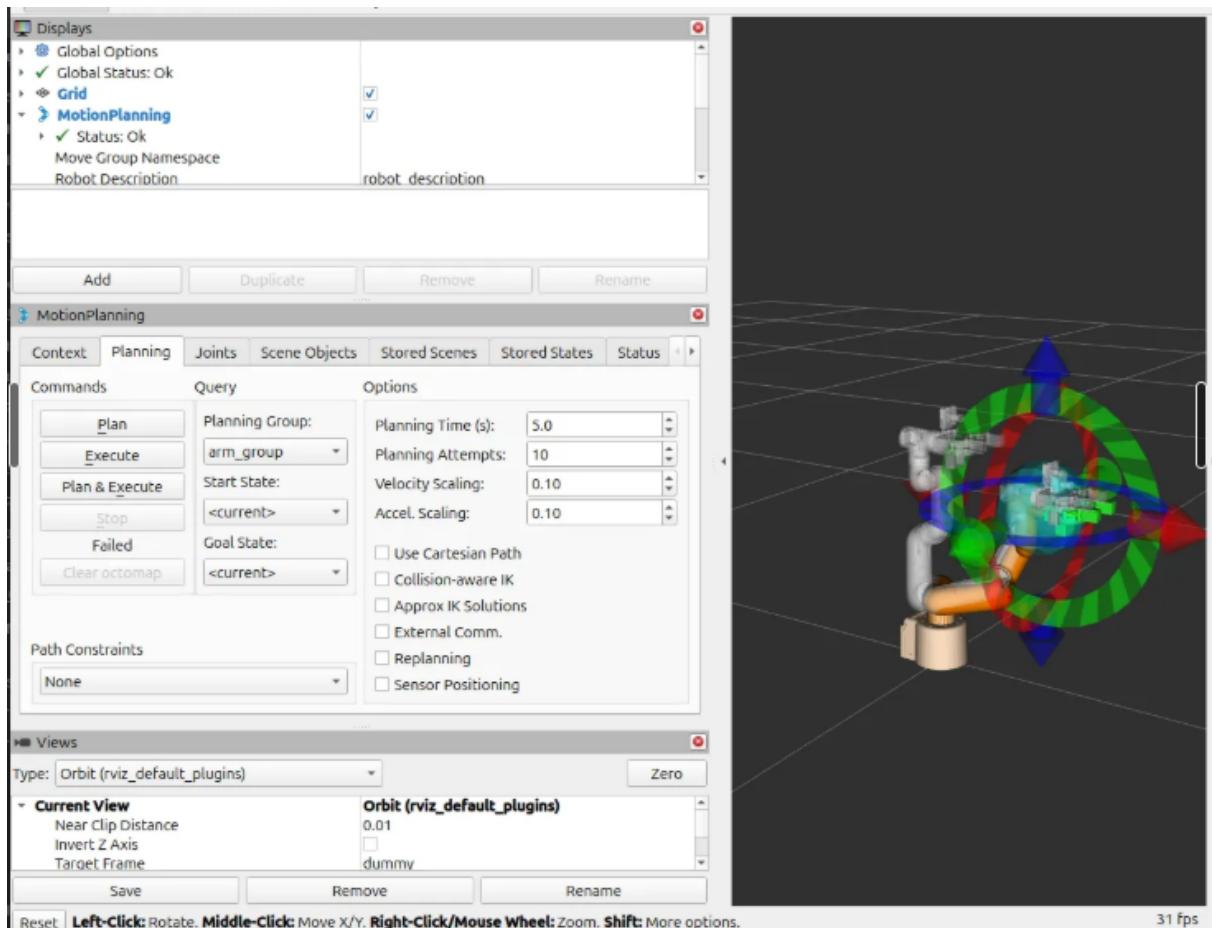
joint_limits:
  1_Joint:
    has_velocity_limits: true
    max_velocity: 2.0
    has_acceleration_limits: true
    max_acceleration: 2.0
  2_Joint:
    has_velocity_limits: true
    max_velocity: 2.0
    has_acceleration_limits: true
    max_acceleration: 2.0
  3_Joint:
    has_velocity_limits: true
    max_velocity: 2.0
    has_acceleration_limits: true
    max_acceleration: 2.0
  4_Joint:
    has_velocity_limits: true
    max_velocity: 2.0
    has_acceleration_limits: true
    max_acceleration: 2.0
  5_Joint:
    has_velocity_limits: true
    max_velocity: 2.0
    has_acceleration_limits: true
    max_acceleration: 2.0
  6_Joint:
    has_velocity_limits: true
    max_velocity: 2.0
    has_acceleration_limits: true
    max_acceleration: 2.0

```

(수정 내용 false로 되어있는것들 다 true로 바꾸고 2.0으로 수정함 처음에 2로 바꿨었는데 무슨 int로 인식한다고 해가지고 뒤에 .0을 붙여야함)

(음 위에 0.1?이랑 2.0랑 곱해서 속도가? 정해진다)

예를 들면 $0.1 \times 2.0 = 0.2$)



5.12 <RT-35> <문지언> <jetcobot 소프트웨어 개발> <7/31>

<RT-35><선태욱> 추가 정리 (see page 313)



5.12.1 코드 플로우 -pick and place

flask 서버 → AprilTag 데이터 전송 → 로봇팔이 데이터를 받음 → AprilTag 데이터로 camera pose → base pose 연산 → 로봇팔 이동

5.12.1.1 <1>

5.12.1.2 문제상황

- base 좌표계 기준 타겟의 위치로 보냈을 때 오차가 큼
 - 그리퍼의 길이를 고려하지 않아 마지막 링크의 중점이 타겟의 위치로 보내지는게 확인됨

5.12.1.3 해결방안

1. dh 파라미터 수정 - 마지막 링크 길이를 그리퍼 포함된 길이로 늘리기
2. dh 파라미터 수정 - 마지막 모터에 링크를 하나 더 추가 (그리퍼)
3. 움직이는 동작이 정해져있으므로 각 동작마다 x,y,z축을 각각 보정해줌

=> FK,IK,파라미터 설정 등 번거로움을 덜어내기 위해 3번 선택

```
# ✓ z축 보정 (방법 1)
position_after = position_before.copy()
position_after[2] += 65 # z축 방향으로 +65mm , 다른 축으로 보정하고 싶으면 인덱스 수정하면 됨
```

5.12.1.4 <2>

5.12.1.5 문제상황

- 베이스 좌표계에서 타겟의 좌표계로 가는 로테이션 행렬의 영향으로 그리퍼가 계속 돌아가 물체를 비스듬히 잡게됨 (이때, 그리퍼의 방향이 바닥과 수평일때 문제, 수직일때는 영향을 끼치지 않아 위 영상처럼 잘 작동함)

5.12.1.6 해결방안

- 1번 로봇이 물건을 꽂힐때는 그리퍼가 바닥과 수평을 이뤄야하므로, 로테이션 행렬을 (-90,40,-90)으로 고정한 채로 이동함 => 잘 이동함
 - **추후에** 무브잇이 가능하게 되면 타겟의 위치를 단순히 점으로 취급하지 말고
그리퍼를 사각형 or box로 모델링해서 충돌 체크 가능하게 해줘야 정확함
 - 이렇게 하면 MoveIt이 “어라? 그리퍼 몸체가 선반에 닿겠네?” 하고 경로를 수정함

5.12.1.7 <3>

5.12.1.8 문제상황

- 그리퍼 구조 때문에 선반에서 타겟이 조금 안쪽으로 들어가있으면 그리퍼 아래쪽에 부딪힘

5.12.1.9 해결방안

- 선반 바깥쪽으로 물건을 진열함
- ...ing
 - 추후에 무브잇 고려

5.12.1.10 <4>

5.12.1.11 문제상황

- 로봇팔이 가지 못하는 base 좌표가 존재
- 로봇팔의 이동 경로를 예측하지 못하여 충돌 문제 & 타겟을 pick하지 못하는 문제가 발생

5.12.1.12 해결방안

- ing..
-

5.12.2

로직

신발이 나가는 경우(서버)

역할	노드 이름	서비스 이름	동작하는 행위	받는 데이터 정보	응답	호출함수명
로1		arm1_service	선반에서 물건을 인식하고 pick	고객ID, 신발 정보 (제품명, 사이즈, 색상), 핑키번호, 신발 위치	성공 여부	arm1_shelf_to_buffer()
			버퍼의 올바른 위치에 물건 place	위치, 신발 정보, 조인트값, 핑키 번호	성공 여부	
로2		amr2_service	버퍼 위 물건을 pick	도착한 핑키 번호, 버퍼 상태	성공 여부	arm2_buffer_to_pinky
			핑키에게 물건 place	핑키 위치	성공 여부 + 핑키 상태 업데이트 함수 호출	

신발이 들어오는 경우 (서버)

역할	노드 이름	서비스 이름	동작하는 행위	받는 데이터 정보	응답	호출함수명
로1		arm1_service	핑키에게 물건을 pick	핑키 도착	성공 여부 + 핑키 상태 업데이트 함수 호출	arm1_buffer_to_shelf()

			버퍼에 물건을 place	버퍼 상태	성공 여부	
로2		amr2_service	버퍼 위 물건을 인식하고 pick	로1의 성공 여부, 버퍼 상태	성공 여부	arm2_pinky_to_buffer()
			선반에게 물건 place	x	성공 여부	

5.12.2.1 추가 요소

- 버퍼 상태를 업데이트 함. 따라서 버퍼도 존을 나눠야함. 그냥 통일시키는게 좋을듯, 버퍼 또한 핑키마다 구역

고려요소

- 우리는 언제 실행해? 핑키처럼 함수를 호출하는 방식으로 진행해야하나?

5.13 <RT-35><선태욱> 추가 정리

확인해볼것

- * 로봇팔에 부착되어 있는 카메라로부터 이미지를 불러오는 것이 로봇 안에서만 사용될 것이면 굳이 udp와 같은 통신이 필요할까?
- * 로봇팔이 이미지 정보를 AI SERVER에 보내야한다면 UDP를 사용하는 것이 맞을거 같다.
- * 그렇다면 다른 팀은 어떻게 하지?
- * 캠 UI를 띄우고 싶다면 어떻게 하면 되지? → 파이썬에서 카메라 자원을 가져와서 파이썬 api로 시각화를 하면 될까?

현재 우리 로봇팔의 로직은 내부에서 촬영하는 이미지를 자체적으로 에이프릴 테그를 감지해서 정보를 연산하고 좌표 변환 연산을 하고 혼자서 움직이는 거잖아. 너무 많은 연산을 주는건가?

그렇다면 로봇팔 제어 노드는 무조건 로봇팔 안에 있어야 하니, 연산량을 덜어주려면 카메라의 정보를 다른 컴퓨터에 UDP로 쏘고, 컴퓨터에서 에이프릴테그를 감지하고, 베이스 좌표를 연산해서 다시 로봇팔한테 보내주는 방식을 써야할까?

만약 여기에 딥러닝이 추가 된다면 무조건 이렇게 해야하지 않을까?

그렇다면 현재 구현을 같은 방식으로 해놔야 편하지 않을까?

그러면 이미지를 UDP로 쓸 수 있는 코드를 만들어 놓을까? 임시방편으로?

로봇팔 내부에서 카메라를 실행시켜도 캠을 볼 수 없음 → GUI 디스플레이가 없음

그러면 이미지를 UDP로 받아서 실행시켜야 하나?

- 로봇팔에 부착된 카메라에서 찍히는 이미지를 UDP로 보내는 코드
- 이미지에서 에이프릴 테그를 감지하고 camera의 coords와 deg를 구하는 코드
- camera의 coords와 deg 그리고 current_joints 값을 활용하여 base 좌표를 구하는 코드

5.13.1 1번 문제 - 카메라 버퍼 문제

5.13.1.1 ⚡ 문제 상황 설명

Robot Arm Service Server는 외부로부터 특정 동작을 요청받는 메시지를 수신합니다. 예를 들어, 메시지에 `buffer_to_pinky`라는 문자열이 포함되어 있다면, 로봇팔은 버퍼에서 Pinky로봇으로 타겟을 pick-and-place하는 동작을 수행해야 합니다.

이때 로봇팔은 타겟의 정확한 위치를 파악하기 위해 카메라로부터 실시간 이미지를 캡처하고, 해당 이미지에서 AprilTag를 감지하여 좌표를 계산합니다.

5.13.1.1.1 ! 그러나 다음과 같은 문제가 발생합니다:

ROS 노드에서 `cv2.VideoCapture()`로 이미지를 가져오는 과정에서, 카메라의 내부 버퍼에 이전 프레임들이 쌓이고 있어, 실시간성이 보장되지 않습니다.

그 결과, AprilTag이 시야에서 이미 사라졌음에도 불구하고, 버퍼에 남아 있던 과거 프레임이 사용되어 잘못된 감지가 발생합니다.

예시:

- 첫 번째 서비스 호출에서는 정상적으로 AprilTag을 감지하고 pick-and-place 동작이 수행됩니다.
- 이후 AprilTag을 치운 뒤 두 번째 서비스를 호출하면, 버퍼에 남아있던 이전 이미지에서 여전히 AprilTag이 탐지되어 실제와 맞지 않는 잘못된 동작이 이루어집니다.

```

==== April Tag 좌표 정보 ====
카메라 기준 좌표: [-6.105250639107791, 8.079935682381349, 194.9
9912103421678]
회전 벡터 (도): [2.1861971378667877, 1.8960765148228649, -6.682
391416767992]

==== Base 좌표계로 변환 중... ====
베이스 좌표 [x, y, z, roll, pitch, yaw]: [55.81784541285408, 19
9.44435517873399, 48.18518997156943, -177.0, 2.0, -52.0]
1차 Joint 접근 목표: [45.81784541285408, 199.44435517873399, 68
.18518997156943, -177.0, 2.0, -52.0]
로봇 이동(2단계) 명령을 전송했습니다.
이동 완료까지 대기 중...
이동 후 현재 좌표: [45.8, 207.6, 84.1, -172.31, 6.87, -51.1]
그리퍼를 완전히 닫습니다.
[INFO] [1754282772.044533927] [robot2_control_node]: 버퍼 → 핑
키: 1, 신발: nike_air
[INFO] [1754282879.602836617] [robot2_control_node]: [서비스 요청]
action: buffer_to_pinky, shelf_num: 3, pinky_num: 1, shoe_i
nfo: nike_air
라디안: [1.673, -0.537, -1.526, 0.514, -0.028, 1.018]

==== April Tag 좌표 정보 ====
카메라 기준 좌표: [-6.117838891379874, 8.082161794531283, 194.9
542740935778]
회전 벡터 (도): [0.7694137903874144, 3.2262081568389913, -6.746
284522519978]

==== Base 좌표계로 변환 중... ====

```

5.13.1.2 해결을 위해 고려할 수 있는 방법

1. 카메라 버퍼를 비우는 코드 추가

`cap.read()` 를 여러 번 호출하여 버퍼를 비우고 마지막 프레임만 사용하는 방식.

2. GStreamer 또는 V4L2 설정으로 버퍼링 최소화

예: `cv2.VideoCapture("v4l2src num-buffers=1 ! videoconvert ! appsink", cv2.CAP_GSTREAMER)`

3. 스레드 기반의 실시간 카메라 스트리밍 처리

항상 최신 프레임을 유지하는 전용 스레드를 돌리며, 요청이 들어오면 가장 최신 프레임만 반환.

5.13.2 1번 문제 해결

스레드 기반의 실시간 카메라 스트리밍 처리를 했다.

영상을 다루는 전용 스레드를 만들어, 실시간으로 이미지를 찍고 있다가, 프레임 요청이 들어오면 가장 최신 프레임을 반환해주는 방식을 채택했다.

```

import cv2
import threading
import time

```

```

class CameraManager:
    def __init__(self, device='/dev/jetcocam0'):
        self.cap = cv2.VideoCapture(device)
        if not self.cap.isOpened():
            raise RuntimeError(f"카메라를 열 수 없습니다: {device}")

        self.latest_frame = None
        self.running = True
        self.lock = threading.Lock()
        self.thread = threading.Thread(target=self._update_frame, daemon=True)
        self.thread.start()

    def _update_frame(self):
        """카메라로부터 프레임을 계속 읽는 백그라운드 루프"""
        while self.running:
            ret, frame = self.cap.read()
            if ret:
                with self.lock:
                    self.latest_frame = frame
            time.sleep(0.03) # 30 FPS 가정 (너무 높으면 부하 발생)

    def get_frame(self):
        """가장 최근 프레임을 반환"""
        with self.lock:
            if self.latest_frame is None:
                raise RuntimeError("아직 프레임이 준비되지 않았습니다.")
            return self.latest_frame.copy()

    def release(self):
        self.running = False
        self.thread.join()
        self.cap.release()

```

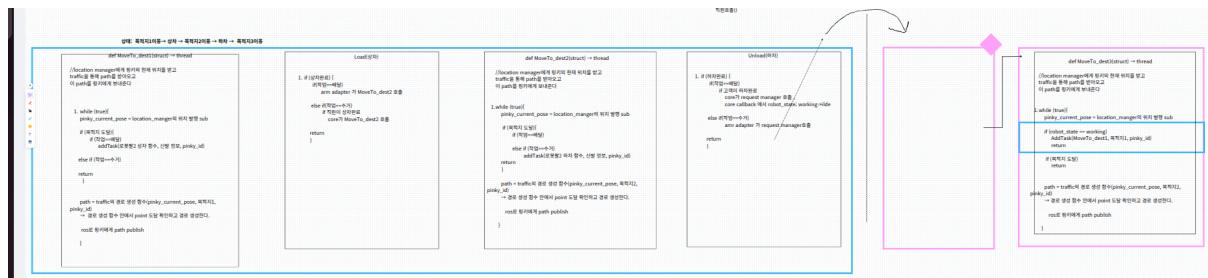
5.13.2.1 Service Call 명령어

```

ros2 service call /arm2_service custom_messeage/srv/RobotArmRequest "{shelf_num: 3,
pinky_num: 1, action: 'buffer_to_pinky', shoe_info: 'nike_air'}"

```

5.14 <RT-36> <황다연> <AMR-ADAPTER>



amr adpater 내 시나리오 작성 및 코드 완성

Thread Pool 시나리오

```

1. 고객 request가 ROS로 들어온 이 정의(고객 위치, 친환경정보)가 request 구조체로 들어온 > 로봇이 물체를 고려해 요청 큐에 적당 request 구조체로 넣어줌 > callback 함수 먼저 request manager의 정의처럼 참조하는 함수 호출(* request manager 기능) > arm adapter의 request 구조체 전달
2. 물체지정함수에서 robot_state 가 idle이며 배터리 30% 이상인 물체 찾아간다(들어온 core에 있는 request queue pop 함수(*request queue가 not empty) * request queue는 구조체 안에 있는 정보박스로 core에서 request manager의 정의처럼 참조하는 함수 호출(* request manager 기능) * request queue는 구조체로 core에서 def MoveTo_dest1(로봇 아이디) 호출
3. amr adapter가 LMI에 traffic에게 실시간 위치를 ros로 sub > amr adapter 가 정기 토큰트로 당시 core함수에 traffic solver들을 준비 > traffic이 solver들을 통해 dest1도착 완료 > amr adapter 가 traffic의 return(def dest1로의 도착완료)을 core를 통해 핸드off > amr adapter 가 로봇을 상자하는 함수 호출(task queue에 넣어줌)
4. 4.1. (예상) 로봇완료 상자 함수 thread에서 실행 > 로봇의 상태로 > 로봇명은 def MoveTo_dest2(dest2, 로봇 아이디) 호출 (task queue에 넣어줌)
5-2. (수거) traffic의 경로상자를 위해 core가 def MoveTo_dest2(dest2, 로봇 아이디) 호출 > amr adapter가 LMI에 traffic에게 실시간 위치를 ros로 sub > amr adapter 가 정기 토큰트로 당시 core함수에 traffic solver들을 준비 > traffic이 solver들을 통해 dest2도착 완료 > amr adapter 가 traffic의 return(def dest2로의 도착완료)을 core를 통해 핸드off
5-2. (수거) amr adapter가 LMI에 traffic에게 실시간 위치를 ros로 sub > amr adapter 가 정기 토큰트로 당시 core함수에 traffic solver들을 준비 > traffic이 solver들을 통해 dest2도착 완료 > amr adapter 가 traffic의 return(def dest2로의 도착완료)을 core를 통해 핸드off
6-1. (예상) (gut상에서 고객이 판매되는 누름/아차판표) > gut에서 core로 정보가 callback함수에서 직관적 사용인지 확인 (작동하면 def MoveTo_dest3(로봇 아이디) (* amr adapter 기능) 호출
6-2. (수거) 로봇완료 상자 함수 thread에서 실행 > 로봇이 차운 관리를 > 로봇을 향수 가 request manager 함수 호출
7. core에서 request manager의 로봇상태를 주면 > request queue가 empty인경우 > empty(예상) > traffic에게 경로상자를 위해 def MoveTo_dest3(로봇 아이디) (* amr adapter 기능) 호출
8-1. (이용증) <request>가 들어와 템카가 지정되는 경우 L2 가 8이랑 많으면 thread에서 실행 > 로봇_state가 working으로 바뀌면 3번 실행
8-2. (이용증) <로봇> <로봇> 카드 X-traffic이 특정한 디자인(보안도장)에 로봇을 경로를 생성 > 템카 출력됨 > dest3도착

```

5.15 <RT-XX><이창연><PID_GUI.origin.ver맵>

https://www.notion.so/PID_GUI-2401adbda1df80c581f3cafdcb34585c?source=copy_link

5.16 <RT-29> <이찬종> <UI 및 서버 개발> <7/31>

6 User-system Interaction via RoboCallee Server

번호	제품	가격	재고	작성일시
1	나이키_250	120000	2	July 26, 2025, 11:44 a.m.
2	나이키_260	120000	2	July 26, 2025, 11:52 a.m.

User Interface 구성

- 사용자는 QR 등으로 접속
상품 조회, 배송/안내 요청 가능
- 반응형 웹으로 제작하여
스마트폰을 통한 접속 용이

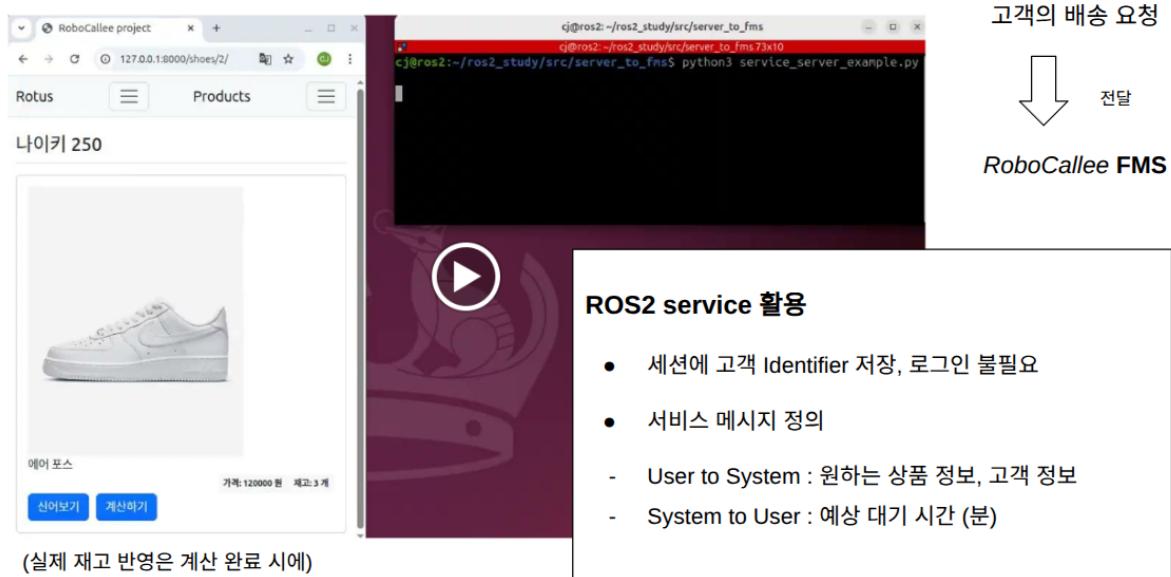
구현 기능

- 제품명 기반 검색
- 상품 정보 제공
- 버튼 클릭을 통한 ROS2 통신
(다음 슬라이드)

TODO

- 통신 내용 구체화
- 디자인 개선
- 관리자 페이지 내용 추가
 - Jetcobot cam 영상
 - Pinky 수거 요청 버튼

8 User-system Interaction via RoboCallee Server



https://drive.google.com/file/d/1fXyNlwxi48EP_EeVPLJqiDmmBPhfkHw3/view?usp=drive_link

5.17 <RT-39> <곽제우> <Traffic Planner 개발> <8/4>

5.18 <RT-38> <이찬종> <Robot Arm Adapter 개발> <8/5>

fms RobotArmAdapter 기능 우선 dummy 수준으로 구현

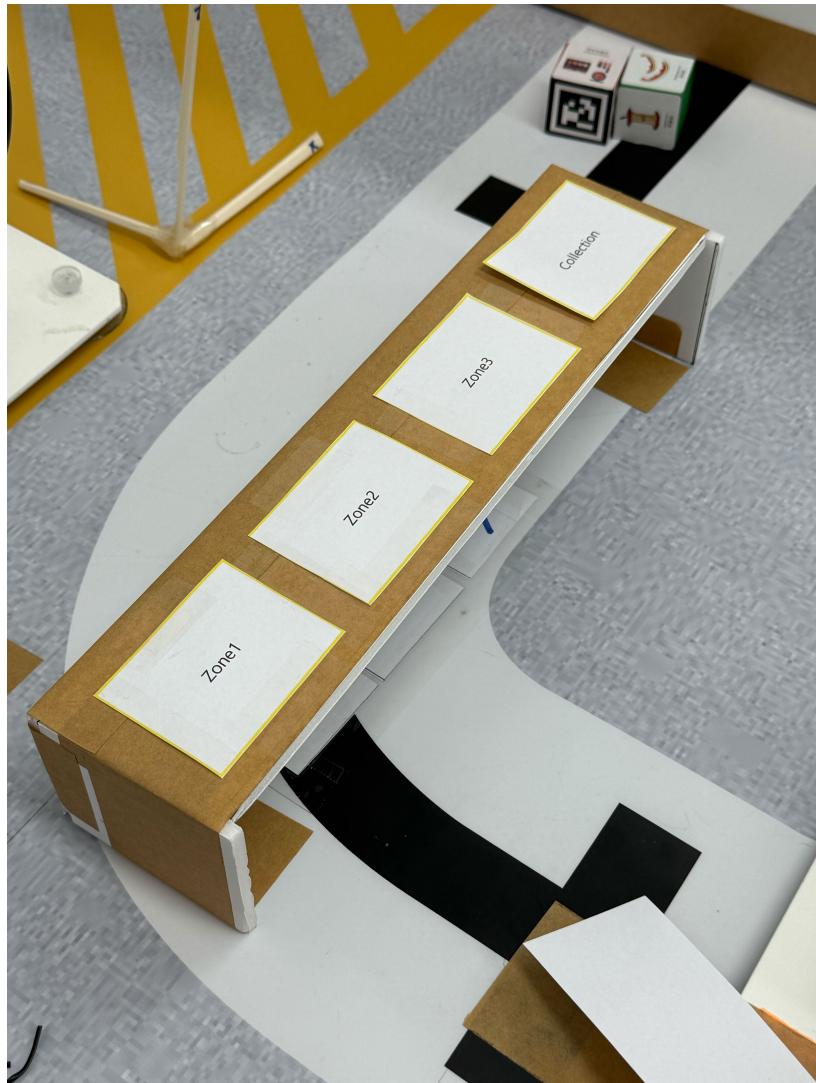
OCR 진행 중

5.19 <RT-41> <문지언> <robot arm 연동 테스트> <8/4>

5.19.1 로봇팔 환경 구성

5.19.1.1 버퍼 환경 구축

5.19.1.1.1 버퍼 크기 (가로 30cm, 세로 7cm)

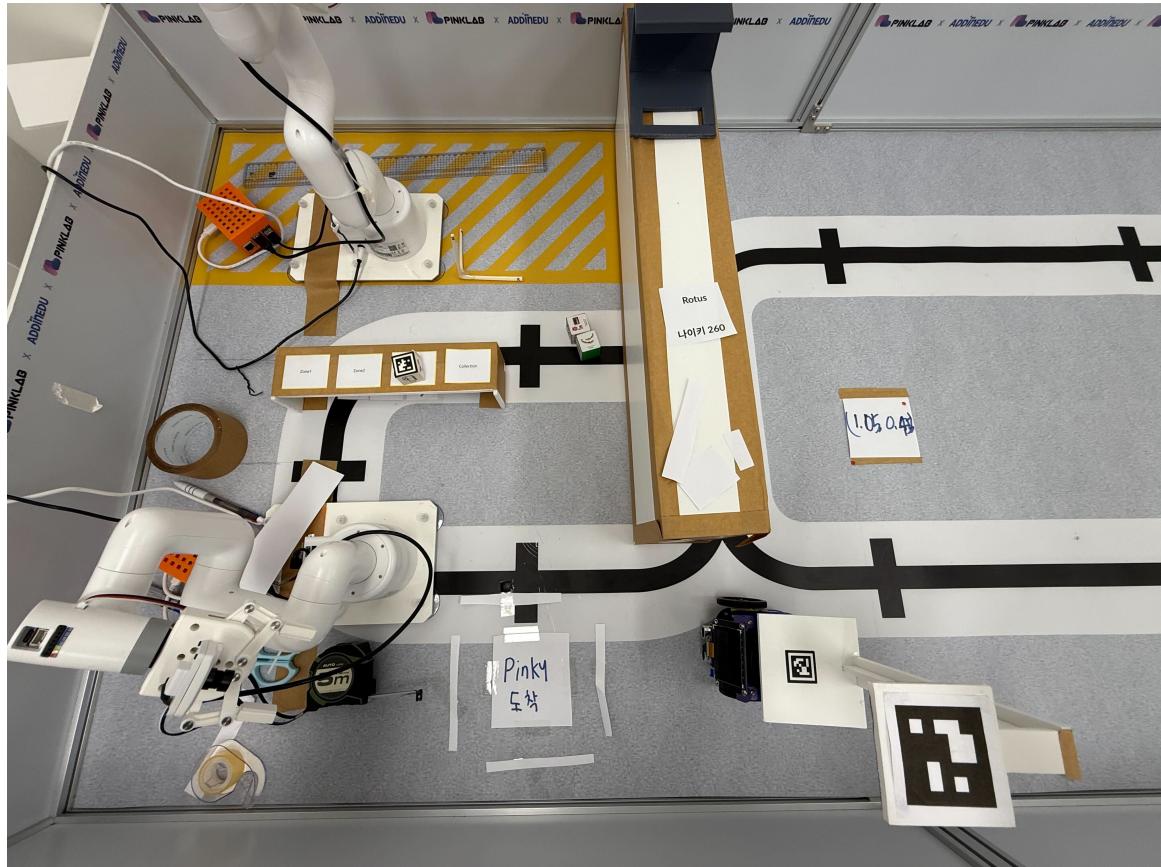


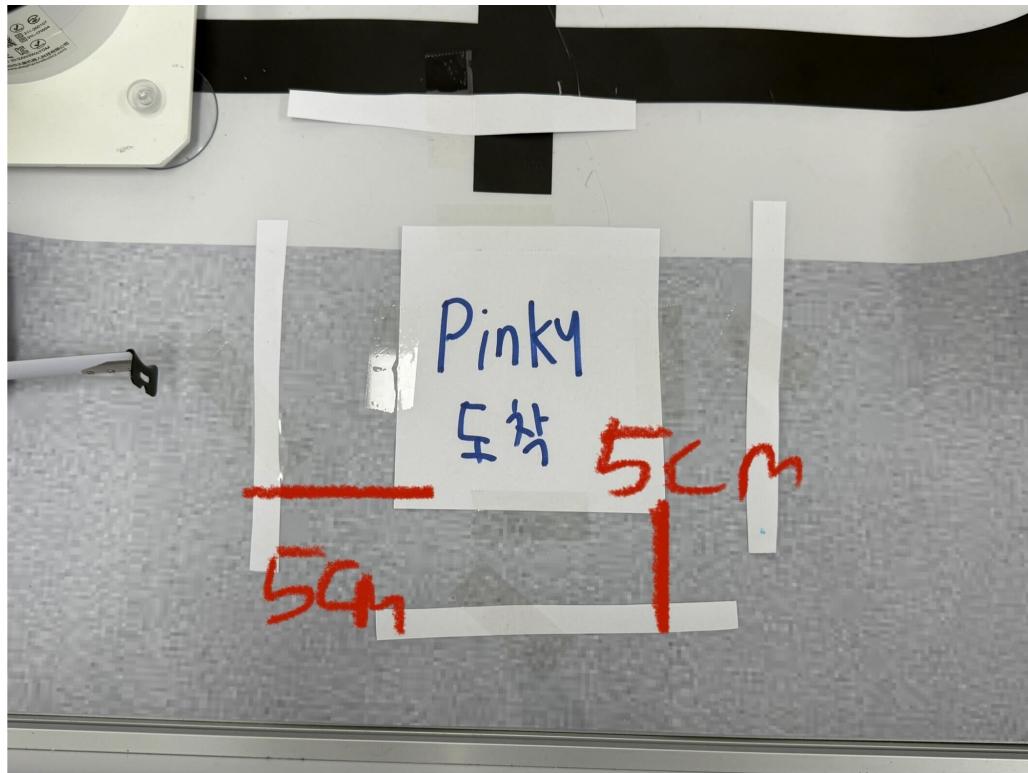
5.19.1.1.2 버퍼 zone 크기 (가로 6cm, 세로 4.5cm)



5.19.1.2 핑키 도착 범위 설정

5.19.1.2.1 오차 범위 5cm





5.19.1.3 시연 영상



5.19.2 버퍼에 따른 로봇팔 보정값

5.19.2.1 초기 자세 통일할 경우 보정값

- 현재 조인트값 : [88.5, 14.58, -1.66, -79.27, 7.03, 43.76]
 - 구역존 이름 : zone 1
 - 보정값 :
 $\text{approach}[0] -= 25$
 $\text{approach}[1] -= 10$
 $\text{approach}[2] += 90$
-

- 현재 조인트값 : [88.5, 14.58, -1.66, -79.27, 7.03, 43.76]
- 구역존 이름 : zone 2
- 보정값 :
 $\text{approach}[0] -= 25$ $\text{approach}[1] -= 0$ $\text{approach}[2] += 80$

-
- 현재 조인트값 : [88.5, 14.58, -1.66, -79.27, 7.03, 43.76]
 - 구역존 이름 : zone 3
 - 보정값 :


```
approach[0] -= 15
approach[1] -= 12
approach[2] += 90
```
-

5.19.2.2 각 버퍼에 따른 초기 자세 수정 및 보정값

- 현재 조인트값 : `mc.send_angles([137.1, -9.84, -31.28, -30.84, -3.69, 91.14], 20)`
 - 구역존 이름 : zone 1
 - 보정값 :


```
approach[0] -= 10
approach[1] -= 7
approach[2] += 70
```
 - 현재 조인트값 : `mc.send_angles([119.0, -12.04, -32.34, -36.12, -2.1, 69.78], 20)`
 - 구역존 이름 : zone 2
 - 보정값 :


```
approach[0] -= 5
approach[1] -= 10
approach[2] += 70
```
 - 현재 조인트값 : `mc.send_angles([119.0, -12.04, -32.34, -36.12, -2.1, 69.78], 20)`
 - 구역존 이름 : zone 3
 - 보정값 :


```
approach[0] -= 10
approach[1] -= 10
approach[2] += 70
```
-

5.19.2.3 로봇팔2에게 물건을 받기 위한 핑키의 범위

`[-26.8, 84.19, -48.07, -59.23, 7.91, 39.02]`

```
[ -14.67, 91.58, -87.62, -37.79, -6.67, 44.2] # 확정 - 조인트
```

5.19.2.4 핑키에게 운반되어 오는 target을 그리퍼로 잡는 영상 (그리퍼 수평)

핑키 바라보는 각도 :

```
mc.send_angles([-14.23, 44.56, -23.55, -49.65, -0.61, 35.59], 20)
```

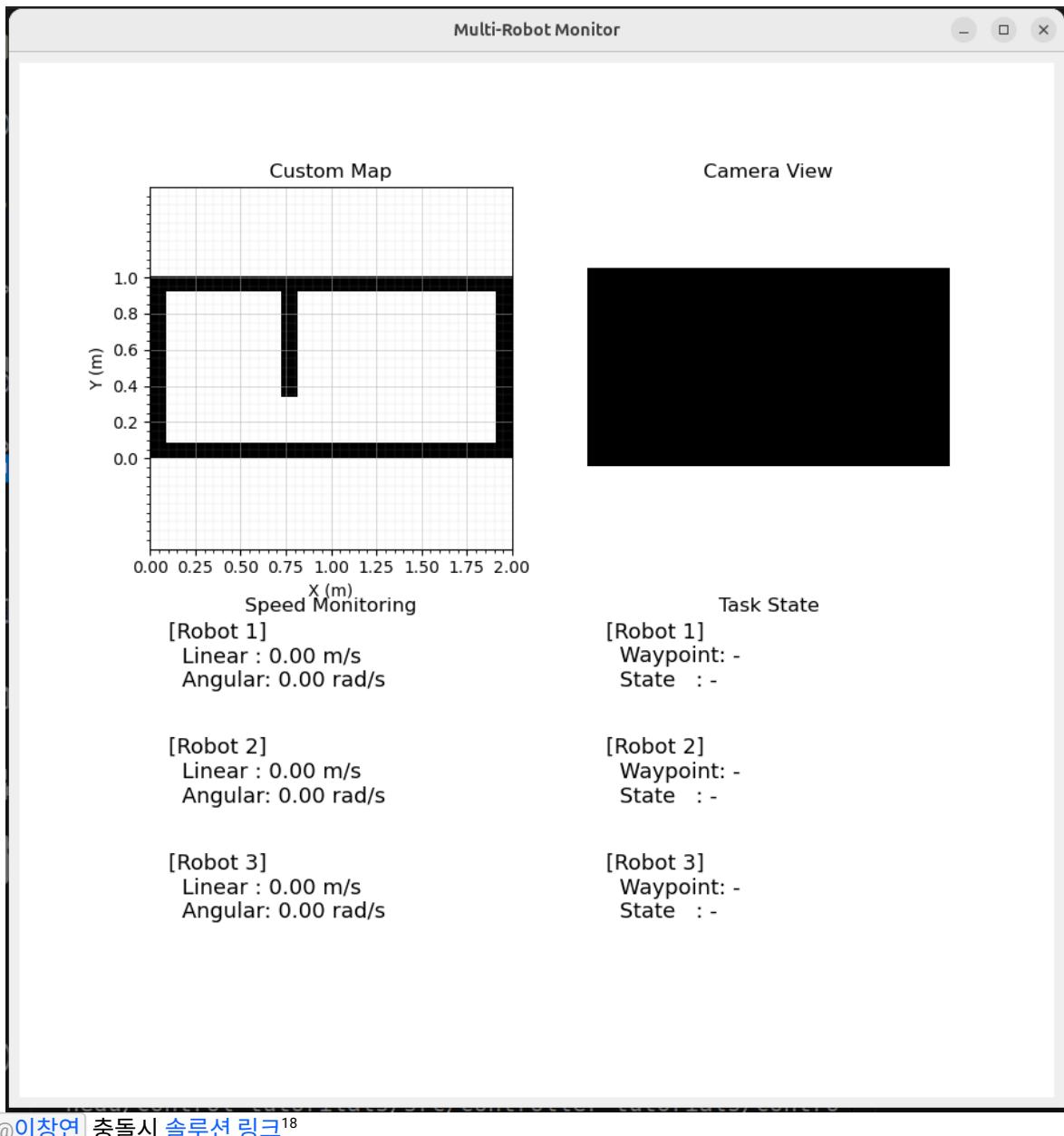
 그리퍼 각도와 보정값 사진 첨부

현재 핑키에서 9cm위에서 집음 ← robot arm의 하드웨어 한계로 너무 낮으면 물건을 수평으로 잡지 못함



5.20 <RT-34><이창연><PID_GUI맵 subplot.ver>

https://www.notion.so/5-8-5-2451adbda1df809087e9f5aca17a74ea?source=copy_link [@이창연] 경로:
제일 최근 코드 찾아가시면 됩니다!!



@이창연 충돌시 솔루션 링크¹⁸

5.21 <RT-40> <0|은혜> <Vision 통한 회피주행> <25.08.06>

5.21.1 [ZMQ 설치]

1. 가상 환경 생성
 - `python3 -m venv venv_zmq`
2. 가상 환경 활성화

18. https://www.notion.so/2471adbda1df807ab25bfda4ac06e383?source=copy_link

- source venv_zmq/bin/activate
3. ZMQ 설치
- pip install pyzmq opencv-python
 - pip install opencv-python pyzmq pillow

5.21.2 [YOLO 설치]

1. 가상 환경 안에서
- pip install typeguard
 - pip install ultralytics

5.21.3 [UDP, ZMQ를 사용한 이미지 전달]

5.21.3.1 왜 ZMQ를 선택했을까? (UDP와의 비교)

5.21.3.1.1 1. 데이터 유실 문제 (신뢰성)

- **UDP** ☠: "그냥 보내고 잊어버리는" 방식입니다. 데이터를 보내긴 하지만 중간에 유실되거나 순서가 뒤바뀌어 도착해도 전혀 책임지지 않습니다. 이미지처럼 용량이 큰 데이터를 보낼 때 패킷 일부가 유실되면, 받은 쪽에서는 **깨진 이미지 파일을 받게 되어 YOLO 분석 자체가 불가능해집니다.**
- **ZMQ** ☠: (내부적으로 TCP를 사용할 경우) 데이터 전송을 보장합니다. 마치 등기우편처럼 데이터가 제대로 도착했는지 확인하고, 유실되면 알아서 다시 보내줍니다. 따라서 **언제나 온전한 이미지 파일을 수신할 수 있습니다.**

5.21.3.1.2 2. 메시지 경계 문제 (개발 편의성)

- **UDP** ☠: 데이터의 시작과 끝을 구분하지 못합니다. 연속으로 이미지 2장을 보내면, 받는 쪽에서는 어디까지가 첫 번째 이미지이고 어디부터가 두 번째 이미지인지 알기 어렵습니다. 이를 해결하려면 개발자가 직접 데이터 앞에 크기 정보나 구분자를 붙이는 복잡한 코드를 추가해야 합니다.
- **ZMQ** ☠: ZMQ는 데이터를 '메시지'라는 원자적 단위로 다룹니다. 보내는 쪽에서 `send()`로 보낸 데이터 덩어리 전체를 받는 쪽에서 `recv()` 한 번으로 온전히 받을 수 있습니다. **이미지 한 장이 하나의 메시지로 깔끔하게 분리되기 때문에 개발이 매우 편리해집니다.**

5.21.3.2 UDP, ZMQ를 이용한 객체탐지 속도 비교

[udp]

0: 480x640 1 tv, 13.5ms

Speed: 2.4ms preprocess, 13.5ms inference, 1.4ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 tv, 12.4ms

Speed: 1.9ms preprocess, 12.4ms inference, 1.3ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 tv, 12.5ms

Speed: 2.3ms preprocess, 12.5ms inference, 1.4ms postprocess per image at shape (1, 3, 480, 640)

[zmq]

0: 640x640 1 person, 12.1ms

Speed: 2.6ms preprocess, 12.1ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 person, 11.8ms

Speed: 2.7ms preprocess, 11.8ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 person, 11.8ms

Speed: 2.9ms preprocess, 11.8ms inference, 1.4ms postprocess per image at shape (1, 3, 640, 640)

5.21.3.3 정리 비교

항목 UDP ZMQ

이미지 해상도 480x640 640x640

탐지 객체 tv person

추론 시간 12.4–13.5ms 11.8–12.1ms

전처리 + 후처리 ~3.8ms ~4.0ms

전체 속도 ZMQ가 약간 빠름 UDP가 조금 느림

5.21.3.4 결론

ZMQ 쪽이 안정적으로 빠른 추론 성능을 보임

UDP는 전송 중 손실이 있었거나 해상도가 낮게 유지되어 정확도에 영향 줬을 가능성 있음

만약 정확한 탐지와 안정성이 중요하다면, 현재 결과로는 ZMQ가 유리

결론

ZMQ가 더 해상도가 높았음에도 전체 속도가 빠른 것은, 단순히 YOLO가 빠르기 때문만이 아니라 ZMQ의 전송이 UDP보다 더 안정적이고 빠르게 동작했을 가능성이 크다는 것을 의미합니다.

5.21.3.5 [회피 위한 Flow]

1. zmq 수신

2. yolo로 탐지

- 사물의 bbox 너비 계산

3. 사람 추적

- 사람: pid로 /cmd_vel publish
- bbox의 중심과 이미지의 중심의 오차를 사용하여 vel, steering 계산
- 최근에 측정한 사람의 상대 위치 업데이트

4. 사물 회피

- nav2로 회피
- 사람의 위치 = goal
 - Nav2 스택에게 goal pose 전송

Nav2 스택에게 goal pose 전송

- Nav2 스택 내에 **BT Navigator**라는 이름의 노드가 운영하는 **Maps_to_pose** 액션 서버에게 전송
- 이는 RViz에서 'Nav2 Goal' 버튼과 같은 원리
- nav2_bringup의 launch 파일에 아래처럼 토픽을 액션으로 remap하는 부분이 있다

```
<remap from="/goal_pose" to="/navigate_to_pose"/>
```

- 카메라 기준 상대 좌표를 TF 통해 map 기준 좌표로 변환

TF 변환

1. 계산: `estimate_distance`, `estimate_angle` 을 사용해 카메라 기준의 3D 위치를 계산
 - `estimate_distance`, `estimate_angle`: 코드 내 구현된 함수
2. 1차 변환: 카메라 기준 좌표 → 로봇(`base_link`) 기준 좌표로 TF 변환

3. 2차 변환: 로봇 기준 좌표 → 지도(map) 기준 좌표로 TF 변환하여 Nav2에 전송

5. lidar에 장애물 반영

- bbox → 거리 추정 → 각도 범위 산출 → lidar에 해당 장애물 반영 → scan pub

0. 각도폭 계산

각도폭이란(Angular Width)?

- 가상 장애물이 LiDAR scan에서 차지할 영역

1. 물체의 중심 각도(이미지에서 물체가 위치한 각도)를 배열의 인덱스로 변환
2. `scan.ranges` 배열의 몇 번째 인덱스에 해당하는지 계산
3. 가상 장애물의 폭을 인덱스 개수로 변환 (폭은 임의로 정해 놓았다)
4. 가상 장애물이 차지할 모든 인덱스 범위를 순회
5. 해당 각도에서 LiDAR가 실제로 측정한 거리 값과 카메라로 물체까지의 추정 거리 중 더 작은 값 선택(값이 작음== 더 가깝다 == 장애물이 존재한다)

핀홀 카메라 모델에서 비례식을 이용하여 물체의 거리 추정

비례식: $W/Z = w/f$

- W: 물체 실제 크기 (m)
- Z: 물체까지의 거리 (m)
- w: 이미지에서 물체가 차지하는 크기 (px)
- f: 카메라 초점거리 (px 단위)

빛은 직선으로 진행하기 때문에, 실제 물체와 이미지 평면에서의 상은 **삼각형의 닮음 관계**를 이루게 된다.

5.21.3.5.1 [유사 초점 거리 계산]

0. 초점 거리란?

- 이미지의 크기와 원근감을 결정
- 초점 거리(Focal Length)는 지도의 축척(Scale)과 같습니다. 지도에 "1cm = 1km"라고 쓰여있는 것과 같습니다. 이 축척이 있어야 지도 위의 거리를 실제 거리로 환산할 수 있습니다. 즉, 초점 거리는 이미지 속 픽셀 단위를 실제 세계의 미터 단위로 번역하는 데 필요한 핵심 정보

1. 유사 초점 거리(Pseudo Focal Length)란?

공식적인 캘리브레이션을 통해 얻은 카메라의 광학적 초점 거리(f_x, f_y)가 아니라, **경험적인 측정을 통해 역으로 계산해낸 '실용적인 상수'**

이 값은 실제 렌즈의 초점 거리와는 다를 수 있지만, 특정 조건 하에서 "픽셀 단위를 미터 단위로 변환해주는 환산 계수" 역할을 효과적으로 수행

- 진짜 초점 거리(from calibration): 렌즈 왜곡 등 모든 변수를 고려한 '과학적인' 값

5.21.3.5.1.1 1단계: 사전 측정 (단 한 번만 수행)

1. **실제 너비(W_{real})**를 아는 물체를 준비합니다. (예: 너비 30cm = 0.3m 짜리 책)
2. 카메라로부터 **실제 거리(D_{known})**를 정확히 아는 위치에 물체를 둡니다. (예: 정확히 2m 떨어진 곳)
3. 해당 물체를 카메라로 찍어 **바운딩 박스의 픽셀 너비(P_{known})**를 측정합니다. (예: 85 픽셀)

5.21.3.5.1.2 2단계: '유사 초점 거리(F)' 계산

- 거리=(실제 너비×초점 거리)/픽셀 너비
- 초점 거리(F)=(거리×픽셀 너비)/실제 너비

$$F = \frac{D_{\text{known}} \times P_{\text{known}}}{W_{\text{real}}} = \frac{2.0\text{m} \times 85\text{px}}{0.3\text{m}} \approx 567$$

- 이제 우리 카메라의 **유사 초점 거리 값 $F = 567$ **
- 이 값은 픽셀 단위를 미터 단위로 변환해주는 마법의 상수 역할

5.21.3.5.1.3 3단계: 실시간 거리 추정, 방향 추정

이제 로봇이 동작하는 동안에는, 이 F 값을 이용해 실시간으로 거리를 추정합니다.

- 입력: YOLO로 감지한 물체의 실시간 픽셀 너비(P_{runtime})
- 계산:

$$D_{\text{estimated}} = \frac{W_{\text{real}} \times F}{P_{\text{runtime}}}$$

- 예시: 만약 움직이는 사람(너비 0.5m)의 픽셀 너비가 110px로 측정되었다면,

$$D_{\text{estimated}} = \frac{0.5\text{m} \times 567}{110\text{px}} \approx 2.58\text{m}$$

6. 거리-픽셀 로그

csv에 bbox_pixel_width, estimated_distance로 쌓인다 -> 회귀 통해 보정

- ZMQ 이미지 수신 + YOLO 추적은 스레드로 분리 → 멀티스레드로 실시간성 높임
-

5.21.3.5.2 ## YOLOv8n vs YOLOv10n 주요 차이점

두 모델의 가장 큰 차이점은 **추론 후처리** 과정에 있습니다.

5.21.3.5.2.1 🧠 YOLOv8n

- **구조:** 수년간 검증된 안정적인 아키텍처를 가집니다. 객체를 탐지한 후, 겹치는 여러 개의 후보 박스 (Bounding Box) 중에서 가장 정확한 박스 하나만 남기는 **NMS(Non-Maximum Suppression)**라는 후처리 과정이 필수적입니다.
- **성능:** 속도와 정확도 사이의 균형이 매우 뛰어납니다. 현재까지도 수많은 프로젝트에서 사용되는 강력한 기준 모델입니다.

5.21.3.5.2.2 🕵️ YOLOv10n

- **구조:** 가장 큰 혁신은 '**NMS-Free**', 즉 **후처리가 필요 없는 구조**라는 점입니다. 모델 학습 단계에서부터 중복된 후보를 만들지 않도록 설계되어, 추론 시 NMS 과정이 생략됩니다.

- 성능: NMS 과정이 없어지면서 추론 속도가 더 빨라지고 지연 시간이 줄어듭니다. 같은 'n'(nano) 모델끼리 비교했을 때, 일반적으로 YOLOv8n보다 더 적은 연산량으로 더 높은 정확도(mAP)를 보여줍니다.

쉽게 비유하자면, YOLOv8은 여러 후보를 제시한 뒤 가장 좋은 하나를 고르는 '후보 추천 후 최종 선택' 방식이라면, YOLOv10은 처음부터 확신에 찬 '단일 최종 후보'를 바로 제시하는 방식에 가깝습니다.

5.21.3.5.3 ## 어떤 모델을 선택해야 할까요?

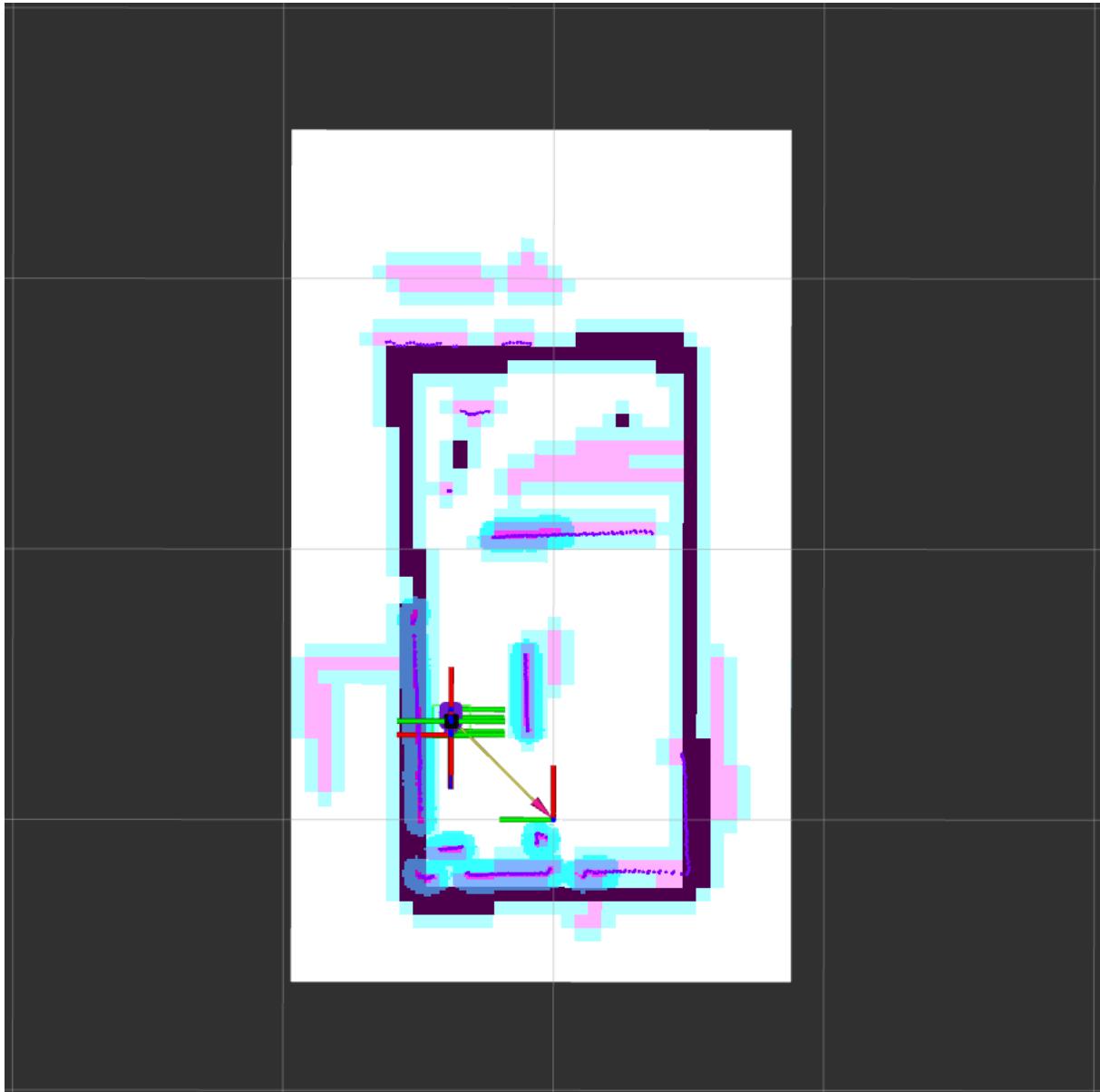
사용자의 목표인 '사람 추종과 객체 인식'은 로봇이나 드론과 같은 환경에서 실시간으로 빠르게 처리하는 것이 중요합니다.

- YOLOv10n을 선택해야 하는 경우:
 - 최고의 성능이 필요할 때: 더 빠른 속도와 더 높은 정확도를 원할 때 가장 좋은 선택입니다.
 - 자원이 제한된 환경일 때: 로봇의 내장 컴퓨터(임베디드 시스템)와 같이 컴퓨팅 파워가 부족한 환경에서 지연 시간을 최소화하고 싶을 때 매우 유리합니다.
 - 새로운 프로젝트를 시작할 때.
- YOLOv8n을 선택해도 괜찮은 경우:
 - 최고의 안정성이 필요할 때: 이미 검증된 수많은 자료와 커뮤니티 지원이 필요할 때 좋습니다.
 - 기존 YOLOv8 기반 프로젝트를 수정하거나 확장할 때.

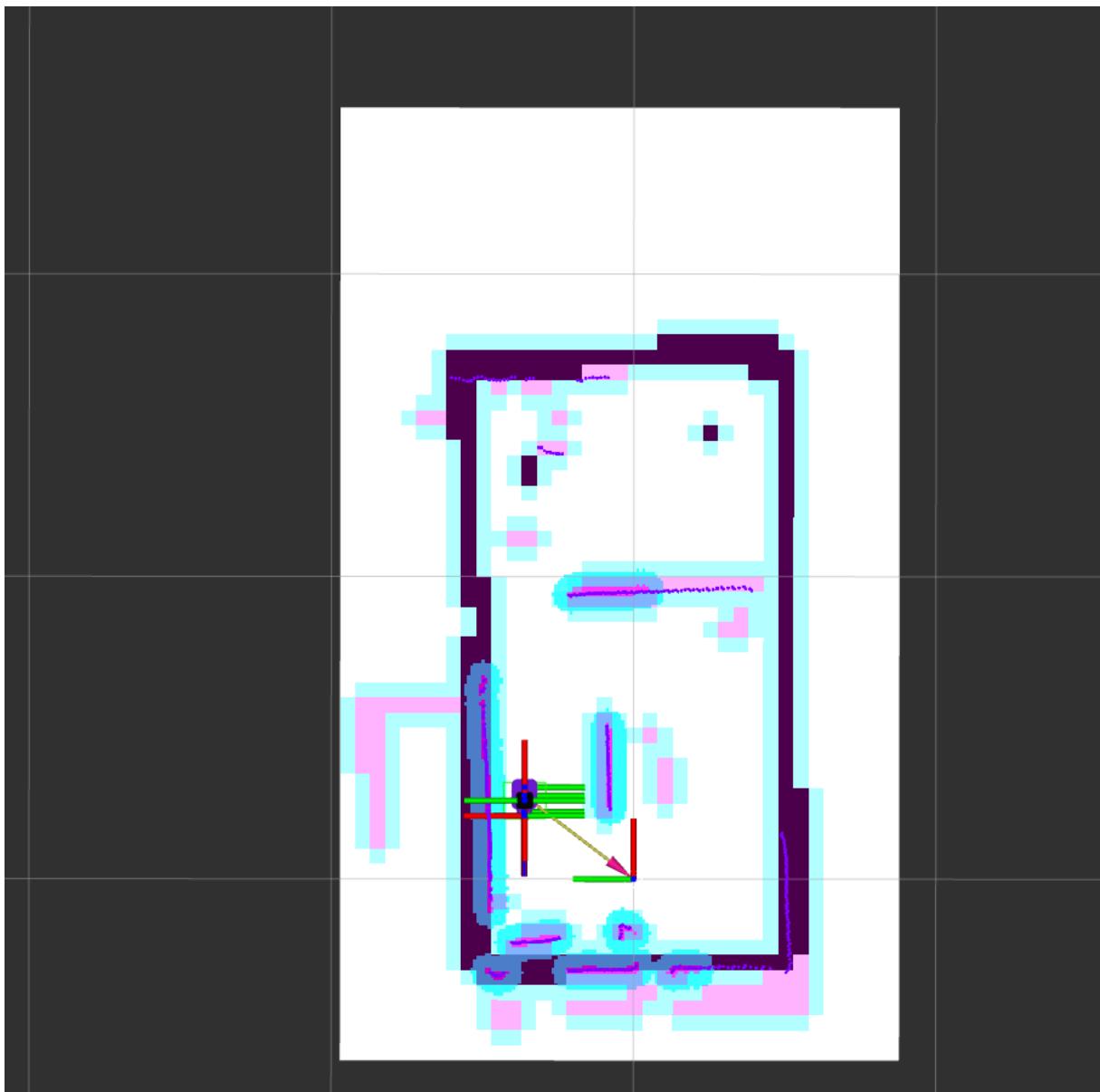
5.21.3.5.4 최종 결론

새로운 프로젝트를 시작하고, 특히 로봇 위에서 실시간 추종 성능을 최대한으로 끌어올리고 싶다면 YOLOv10n을 사용하시는 것을 강력히 추천합니다. 더 낮은 지연 시간과 개선된 정확도는 사람을 부드럽고 정확하게 따라가는 데 큰 이점이 될 것입니다.

5.21.3.6 8/16- 라이다 데이터에 가상 장애물 추가한 결과



17 원본 /scan 데이터



18 가상 장애물을 추가한 /new_scan

[주행 결과]

- 전보다는 더 우회해서 돌아가지만 아직은 완벽한 회피 주행이 되지 않는다.

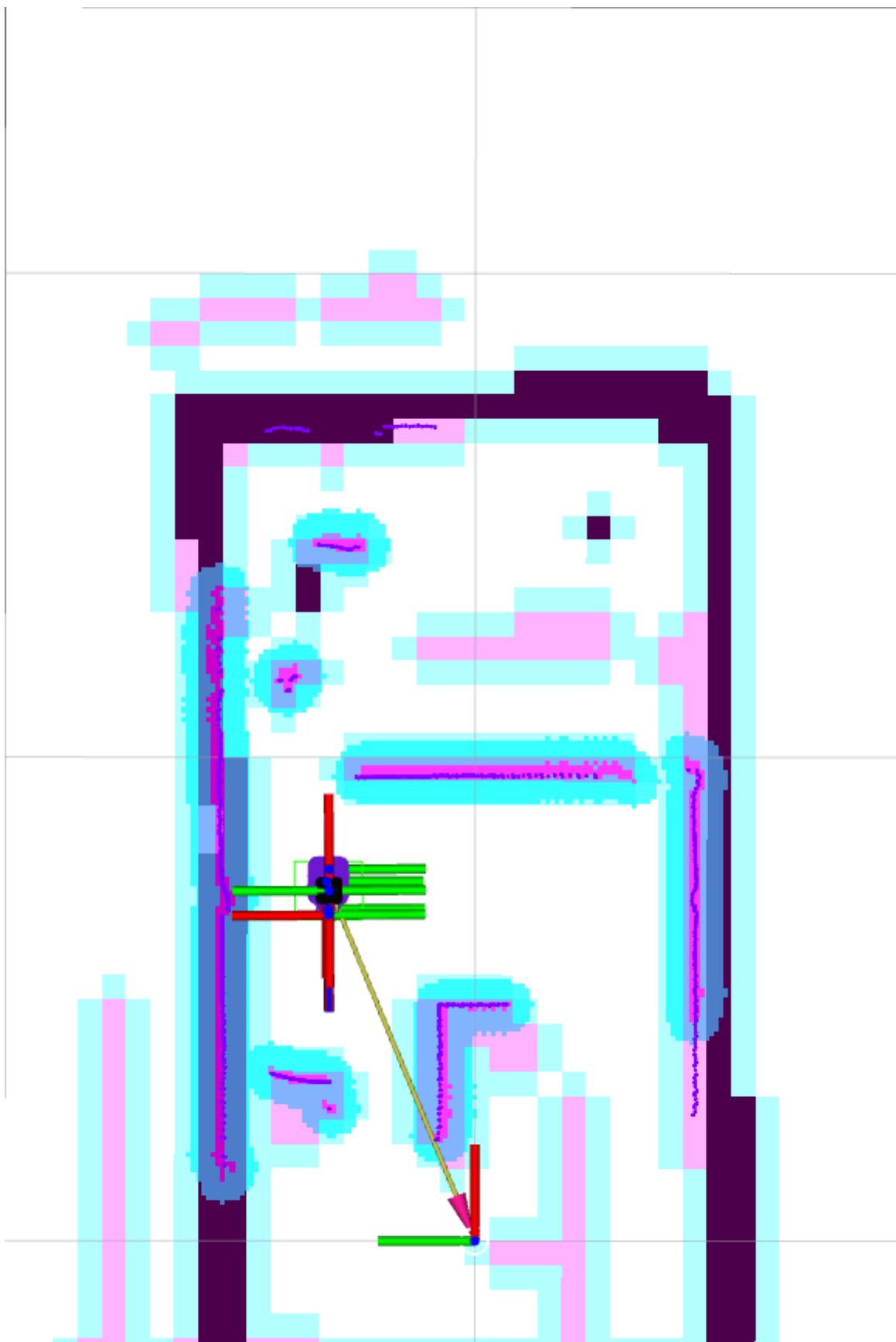
[원인 분석]

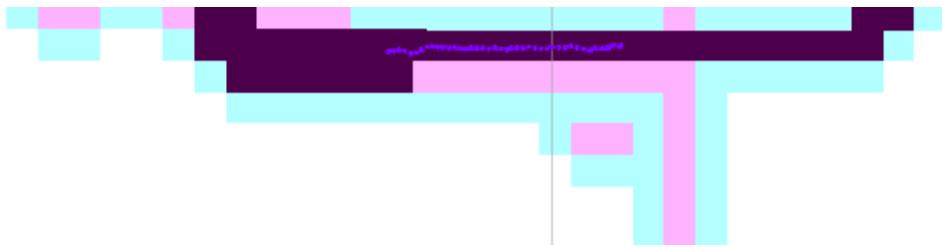
- 카메라 시야에서 장애물이 사라지면 추가한 가상 장애물이 사라져 원본 /scan으로 돌아온다.

5.21.3.7 8/18 - 코드 수정



19 원본





20 가상 장애물 추가

[원인]

원본 /scan은 publish 되지 않고 장애물을 탐지했을 때만 publish되는 /new_scan만 publish 되고 있었다. 실시간 라이다 데이터 반영이 되지 않고 있었다.

[문제]

추가한 장애물이 왜 왼쪽에 치우쳐 있나?

[강사님 feedback]

- 맵에 있는 다른 물체들도 가상 장애물 추가하면 어떻게 하나?
- 구한 각도에 있는 장애물의 index를 늘리는 방법
- 핑키를 yolo 모델에서 인식하지 못해서 → 핑키의 카메라, 모니터 등만 인식하고 전체를 인식하지 못하는 경우도 있을 수도 있어서 전체 너비를 인식하지 못하는 경우가 있을 수도 있음

→ 시나리오 변경: 비전을 사용하는 현재 시나리오를 유지하려면 14cm보다 작지만 이미 학습이 된 물체를 사용하자

e.g.) 우리의 소파 → 소파라고 인식을 못하면 소파라고 인식하는 그림을 붙이자

5.21.3.8 카메라 캘리브레이션

Calibrating using 9 valid images...

```
[[749.04619185 0. 301.91118398]
 [ 0. 749.2763205 324.9294889 ]
 [ 0. 0. 1. ]]
```

```
[[ 1.60446384e-01 -1.13984452e+00 -1.91316889e-03 1.35314278e-03
 2.51874695e+00]]
```



21 원본 /scan

Sprint 기록 – 338



22 가상 장애물 추가한

회피 성공!

장애물을 cache로 저장했지만, 물체와 카메라 간의 거리를 저장해서 인지 로봇이 움직이면 일정 거리를 유지한 채 장애물도 같이 움직인다.

5.22 <RT-41> <문지언> <robot arm 연동 테스트> <8/14>

5.22.1 로봇팔2 동작 영상

5.22.1.1 buffer1 → pinky



5.22.1.2 buffer2 → pinky



5.22.1.3 buffer3 → pinky



5.22.1.4 pinky → buffer



5.22.2 로봇팔1 동작 영상

5.22.2.1 collection to shelf(2)



5.22.2.2 collection to shelf(1)



5.22.2.3 shelf(1) to buffer(1)



5.22.2.4 shelf(2) to buffer(2)



6 Robot 실행 메뉴얼

6.1 AMR(pinky) 실행 코드 정리

공유기 ip 주소를 이용해서 SSH로 pinky 연결

```
dfc6:  
ssh pinky@192.168.0.219  
export ROS_DOMAIN_ID=51  
  
20f0:  
ssh pinky@192.168.0.222  
export ROS_DOMAIN_ID=52  
  
1e3e:  
ssh pinky@192.168.0.215  
export ROS_DOMAIN_ID=53
```

6.1.1 SLAM

6.1.1.1 Terminal

[로봇] pinky bringup 실행

```
ros2 launch pinky_bringup bringup.launch.xml
```

[로봇] cartographer 실행

```
ros2 launch pinky_cartographer map_view.launch.xml
```

[로봇] map server로 맵 저장

```
nav2_map_server map_saver_cli -f map1
```

6.1.2 Navigation

6.1.2.1 Terminal

[로봇] pinky bringup 실행

```
ros2 launch pinky_bringup bringup.launch.xml
```

[로봇] 내비게이션 실행

```
ros2 launch pinky_navigation initialpose_bringup_launch.xml map:=map001.yaml
```

[PC] nav2_view.rviz 실행

pc랑 로봇이랑 도메인 아이디 일치시키기!!

```
ros2 launch pinky_navigation nav2_view.launch.xml
```

[로봇] pinky teleop 실행

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

[로봇] pid 실행

```
ros2 run pid_controller_node pid_velocity_filter_node
```

[PC] 관제 카메라 실행: Aruco TF 전달

```
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

```
ros2 run aruco_marker_pkg aruco_odom_publisher
```

[로봇] 관제 노드 launch 실행TF 및 goalpose 노드 실행

```
ros2 launch pinky_rmf rmf_nodes_launch.py
```

[로봇] Aruco odom_{id}을 odom과 TF로 변환

```
ros2 run aruco_marker_pkg monitoring_map_guiros2 run pinky_rmf aruco_TF_node
```

[PC] TF static_transform

```
ros2 run tf2_ros static_transform_publisher 0.5 0.0 0.0 0 0 1.57 odom base_link
```

6.1.2.2 관제 카메라 연결

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

[관제 PC] Domain Bridge 관제 cam cbs_planner 실행

```
export ROS_DOMAIN_ID=59
ros2 run domain_bridge domain_bridge ~/domain_bridge_monitoring_TF.yaml
```

```
export ROS_DOMAIN_ID=59
ros2 run domain_bridge domain_bridge ~/domain_bridge_goalpose.yaml
```

```
export ROS_DOMAIN_ID=59
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

```
export ROS_DOMAIN_ID=59
ros2 run mapf_planner cbs_planner_node_pose_pub
```

[관제 PC] Monitoring Map 0.2m grid 실행

```
ros2 run aruco_marker_pkg monitoring_map_gui02
```

[관제 PC] Traffic planner 0.2m grid 실행

```
ros2 run traffic_planner traffic_path_planner02 --ros-args -p num_robots:=2 -p start_positions:="[1.0,10.0,2.0,10.0]" -p goal_positions:="[1.0,3.0,3.0,5.0]"
```

6.1.3 P 제어 이동

ID :53

[로봇] bringup

```
ros2 launch pinkyBringup bringup.launch.xml
```

[로봇] move to goal

```
ros2 run pid_controller_node move_to_goal
```

[PC] Domain bridge: 관제 aruco maker

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_monitoring_TF.yaml
```

[PC] 관제 goalpose

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_goalpose.yaml
```

[PC] 관제 카메라 실행: Aruco TF 전달

```
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

[PC] GUI

```
ros2 run aruco_marker_pkg monitoring_map_gui
```

[PC] MAPF 실행

```
ros2 run mapf_planner cbs_planner_node_pose_pub

ros2 run mapf_planner cbs_planner_node_pose_pub_n_robots --ros-args -p
num_robots:=2 -p start_positions:=[2.0,19.0,4.0,19.0] -p
goal_positions:=[8.0,10.0,2.0,10.0]
ros2 run mapf_planner cbs_planner_node_pose_pub_n_robots --ros-args \
-p num_robots:=3 \
-p start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] \
-p goal_positions:=[1.0,6.0,1.0,7.0,1.0,8.0,]

ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=3 -p
start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] -p
goal_positions:=[1.0,6.0,1.0,7.0,1.0,8.0]
ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=1 -p
start_positions:=[2.0,19.0] -p goal_positions:=[2.0,7.0]
ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=3 -p
start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] -p
goal_positions:=[1.0,4.0,1.0,5.0,1.0,6.0]

ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=2 -p
start_positions:=[2.0,19.0,4.0,19.0] -p goal_positions:=[1.0,4.0,1.0,5.0]
ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=2 -p
start_positions:=[2.0,19.0,5.0,19.0] -p goal_positions:=[1.0,4.0,5.0,9.0]
```

물류 창고 이동

1

충전소 및 신발 위치로 이동

```
ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=3 -p
start_positions:=[2.0,4.0,2.0,6.0,2.0,8.0] -p
goal_positions:=[5.0,17.0,8.0,13.0,5.0,9.0]
```

6.1.3.1 P 개인 튜닝

[PC] 로봇과 Domain_ID 일치 시키기

[로봇] bringup

```
ros2 launch pinkyBringup bringup.launch.xml
```

[로봇] move to goal tuning

```
ros2 run pid_controller_node move_to_goal_tuning
```

[PC] aruco camera

```
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

[PC] GUI

```
ros2 run aruco_marker_pkg monitoring_map_gui
ros2 run aruco_marker_pkg monitoring_gui
ros2 run monitoring_pkg integrated_aruco_monitor
```

[PC] Traffic Path Planner

```
ros2 run mapf_planner cbs_planner_node_pose_pub_n_robots --ros-args -p
num_robots:=3 -p start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] -p
goal_positions:=[8.0,10.0,2.0,10.0,2.0,0.0,4.0,0.0]
ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=3 -p
start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] -p
goal_positions:=[1.0,6.0,1.0,7.0,1.0,8.0]

ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=3 -p
start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] -p
goal_positions:=[2.0,7.0,2.0,10.0,2.0,13.0]
```

6.1.3.2 관제 카메라 연결

```
sudo udevadm control --reload-rules
sudo udevadm trigger
```

6.1.3.3 PID 개인 튜닝

[로봇] bringup

```
ros2 launch pinkyBringup bringup.launch.xml
```

[로봇] move to goal tuning

```
ros2 run pid_controller_node move_to_goal_pid
```

[PC] aruco camera

```
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

[PC] 관제 goalpose

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_goalpose.yaml
```

[PC] GUI

```
ros2 run aruco_marker_pkg qmonitor_state_machine
```

[PC] move to goal

```
ros2 run traffic_planner traffic_path_planner --ros-args -p num_robots:=3 -p start_positions:=[2.0,19.0,4.0,19.0,6.0,19.0] -p goal_positions:=[2.0,7.0,2.0,10.0,2.0,13.0]
```

6.1.3.4 DWB

[로봇] bringup

```
ros2 launch pinkyBringup bringup.launch.xml
```

[로봇] move to goal tuning

```
ros2 run pid_controller_node dwb_planner
```

6.1.3.5 관제 카메라 연결

```
sudo udevadm control --reload-rules  
sudo udevadm trigger
```

[PC] aruco camera

```
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

[PC] 관제 goalpose

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_goalpose.yaml
```

[PC] GUI

```
ros2 run aruco_marker_pkg qmonitor_state_machine
```

6.1.3.6 Map 띄우기**[PC] 관제 카메라 및 TF 브로드캐스트**

```
ros2 run aruco_marker_pkg multi_aruco_odom_publisher
```

[PC]rviz2 실행

```
rviz2
```

[PC] map server 실행

```
ros2 run nav2_map_server map_server --ros-args -p yaml_filename:=/home/addininedu/map001.yaml
```

[PC] map 서버 lifecycle 실행

```
ros2 lifecycle set /map_server configure
ros2 lifecycle set /map_server activate
```

6.1.3.7 마지막 PID controller + GUI (다중연결)**[PC] 관제**

domain_id=59

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_server_amr.yaml
```

[PC] 관제 GUI

```
ros2 run aruco_marker_pkg monitoring_gui_intergrated3
```

[AMR] bring up

```
ros2 launch pinkyBringup bringup.launch.xml
```

[AMR] move_to_goal_pid_controller

```
ros2 run pid_controller_node move_to_goal_pid_controller
# 로봇별 PID 제어기 실행 (robot_id 파라미터로 구분)
ros2 run pid_controller_node move_to_goal_pid_controller --ros-args -p robot_id:=1
ros2 run pid_controller_node move_to_goal_pid_controller --ros-args -p robot_id:=2
ros2 run pid_controller_node move_to_goal_pid_controller --ros-args -p robot_id:=3
```

6.1.3.8 Aruco 위치 Calibration

```
ros2 run aruco_marker_pkg aruco_test
```

6.1.3.9 강화학습 DQN

[AMR] bring up

```
ros2 launch pinky_bringup bringup.launch.xml
```

[AMR] front scan

```
ros2 run local_planner front_scan
```

[AMR] rl environment

```
ros2 run local_planner rl_environment
```

[AMR] rl dqn

```
# 활성화
source ~/turtlebot3_ml_env/bin/activate
# 가상환경이 활성화된 상태에서
pip install --upgrade pip
pip install tensorflow keras numpy scipy pyqtgraph matplotlib
# 가상환경의 패키지 경로 확인
python -c "import site; print(site.getsitepackages()[0])"

# PYTHONPATH에 추가
export PYTHONPATH="$PYTHONPATH:$python -c 'import site; print(site.getsitepackages()[0])'"

# 확인
echo $PYTHONPATH
```

```
ros2 run local_planner dqn 1 600
```

```
ros2 run tf2_ros static_transform_publisher 1 0.2 4 0 3.14159 0 map surveillance_camera
rviz2
```

6.1.4 Domain Bridge 예제

https://github.com/jewookwak/ROS2_Jazzy_Study

```
#ROS_BRIDGE STUDY
```

방법 1 listener 쪽에서 토픽명 remapping

```
export ROS_DOMAIN_ID=1
```

```
ros2 run demo_nodes_py listener --ros-args -r chatter:=/topic_to_domain1
```

```
export ROS_DOMAIN_ID=2
```

```
ros2 run demo_nodes_py listener --ros-args -r chatter:=/topic_to_domain2
```

```
export ROS_DOMAIN_ID=3
```

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_test.yaml
```

```
export ROS_DOMAIN_ID=3
```

```
ros2 run my_demo_pkg talker1
```

```
export ROS_DOMAIN_ID=3
```

```
ros2 run domain_bridge domain_bridge ~/domain_bridge_test2.yaml
```

```
export ROS_DOMAIN_ID=3
```

```
ros2 run my_demo_pkg talker2
```

방법 2 yaml에서 topic명 remapping

```
-----
```

```
export ROS_DOMAIN_ID=1
ros2 run demo_nodes_py listener
```

```
export ROS_DOMAIN_ID=2
ros2 run demo_nodes_py listener
```

```
export ROS_DOMAIN_ID=3
ros2 run domain_bridge domain_bridge ~/domain_bridge_remapped_topic.yaml
```

```
export ROS_DOMAIN_ID=3
ros2 run my_demo_pkg talker1
```

```
export ROS_DOMAIN_ID=3
ros2 run my_demo_pkg talker2
```

```
jewoo@jewoo:~$ export ROS_DOMAIN_ID=3
jewoo@jewoo:~$ ros2 run domain_bridge domain_bridge ~/domain_bridge_waypoints.yaml
domain_bridge_waypoints.yaml
```

```
name: send_waypoints
topics:
  way_points1:
    remap: way_points
    type: std_msgs/msg/String
    from_domain: 3
    to_domain: 1
  way_points2:
    remap: way_points
    type: std_msgs/msg/String
    from_domain: 3
    to_domain: 2
```

6.2 ROS_DOMAIN_ID

로봇 : ROS_DOMAIN

(pinky@192.168.4.1¹⁹⁾

```
pinky_dfc6 : 51
pinky_20f0 : 52
pinky_1e3e : 53

(jetcobot@192.168.5.120)

jetcobot_aa85 : 61 (문에 가까운)
jetcobot_aa3b : 62
```

6.3 통합 코드 빌드 & Git 명령어

6.3.1 통합 코드 빌드

디버깅 모드로 빌드

```
/roscamp-repo-3/Service/ROS_Task_Manager/robocallee_fms$ <- 이 위치에서 빌드/ vscode 열어야 함
.
colcon build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

vscode 도 **robocallee_fms**에서 열어야 함.

```
git checkout -b <로컬브랜치명> origin/<원격브랜치명>

addinunedu@:/~/Rotus_team_traffic/roscamp-repo-3$ git branch
  Traffic
* dev_temp
  main
```

주의 branch check out 전에 commit이나 stash를 해놓고 branch

```
git status
git stash
git pull origin dev
git stash pop
git add.
git commit -m "~~~"
git config --global user.name "jewookwak"
git config --global user.email "jewoo2963@naver.com"
git commit -m "~~~"
git push origin dev
```

19. mailto:pinky@192.168.4.1

20. mailto:jetcobot@192.168.5.1

6.3.1.1 1. git status

```
git status
```

- 현재 Git 저장소의 상태를 확인합니다.
- 변경된 파일, 추적되지 않은 파일, 커밋 대기 중인 파일 등을 보여줍니다.
- 주로 커밋 전에 어떤 변경 사항이 있는지 확인할 때 사용합니다.

6.3.1.2 2. git stash

```
git stash
```

- 작업 중이던 변경 사항(추가·수정된 파일)을 임시로 저장하고, 워킹 디렉토리를 마지막 커밋 상태로 되돌립니다.
- 주로 브랜치를 변경해야 하는데 현재 변경 사항을 아직 커밋하고 싶지 않을 때 사용합니다.
- 변경 사항은 "스택" 형태로 저장됩니다.

6.3.1.3 3. git pull origin dev

```
git pull origin dev
```

- 원격 저장소(`origin`)의 `dev` 브랜치에서 최신 변경 사항을 가져와 현재 브랜치에 **병합(merge)** 합니다.
- `git fetch + git merge` 과 같은 동작을 합니다.

6.3.1.4 4. git stash pop

```
git stash pop
```

- `git stash`로 저장해둔 가장 최근 변경 사항을 다시 적용합니다.
- 동시에 스택에서 해당 `stash` 항목을 삭제합니다.
- 만약 적용 중에 충돌(conflict)이 발생할 수 있습니다.

6.3.1.5 5. git add .

- 현재 디렉토리(.) 내 변경된 모든 파일을 **스테이징 영역**에 추가합니다.
 - `git commit` 명령을 실행하면 스테이징된 파일만 커밋됩니다.
 - . 대신 특정 파일 이름을 지정할 수도 있습니다.
-

6.3.1.6 6. git commit -m "~~"

```
git commit -m "~~"
```

- 스테이징된 파일들을 커밋하고, "~~" 부분에 커밋 메시지를 기록합니다.
 - 메시지는 변경 내용을 간단하고 명확하게 적는 것이 좋습니다.
-

6.3.1.7 7.

```
git config --global user.name "jewookwak"  
git config --global user.email "jewoo2963@naver.com"
```

- Git 사용자 이름과 이메일을 전역(global) 설정에 저장합니다.
 - 전역 설정이므로, 해당 컴퓨터의 모든 저장소에서 동일한 이름·이메일이 사용됩니다.
 - 커밋 로그에 이 정보가 기록됩니다.
-

6.3.1.8 8. git commit -m "~~"

```
git commit -m "~~"
```

- (다시) 스테이징된 변경 사항을 커밋합니다.
 - 앞에서 한 commit이 잘못되었거나 추가 변경이 있을 때 새로 실행할 수 있습니다.
-

6.3.1.9 9. git push origin dev

```
git push origin dev
```

- 현재 브랜치의 커밋을 원격 저장소(`origin`)의 `dev` 브랜치에 업로드합니다.
- 원격 저장소와 로컬 저장소를 동기화하는 과정입니다.

<통신통합코드>

6.3.1.10 패키지 노드 실행

```
■ cd roscamp-repo-3/Service/ROS_Task_Manager/robocallee_fms
source install/setup.bash
ros2 run robocallee_fms robocalle_fms_node
```

6.3.1.11 핑키 주행노드실행

6.3.1.12 들어오는지 확인

```
■ ros2 topic info -v /odom_2 # Pub=robocallee_fms, Sub=goal_mover
ros2 topic echo -once /odom_2
ros2 topic hz /odom_2 # 주기 나오면 OK
```

6.3.2 오동이 계속 들어오는지(ROS IF 변환 결과)

`ros2 topic hz /odom_2`

6.3.3 goal_mover가 속도 내는지

ros2 topic echo /cmd_vel

6.3.3.1 goalpose 토픽 확인

```
☒ ros2 topic list | grep goalpose
ros2 topic info -v /goalpose2 # Pub=robocallee_fms, Sub=goal_mover 여야 함
ros2 topic echo --once /goalpose2 # 한 건 짹히는지
```

핑키내부에서

ros2 run pid_controller_node move_to_goal_tuning --ros-args -r /odom:=/odom_1 -r /goal_pose:=/goalpose1 -r /cmd_vel:=/cmd_vel

+ 브링업 실행

6.4 Django Server 실행 명령어

전체적인 설명은 여기를 참조하면 됨

<https://wikidocs.net/70588>

우선 아래 내용과 직접 연관된 부분들은 1-03, 1-04, 2-02, 2-03

6.4.1 [Git clone]

1. Robocallee_Server를 git clone 한다.

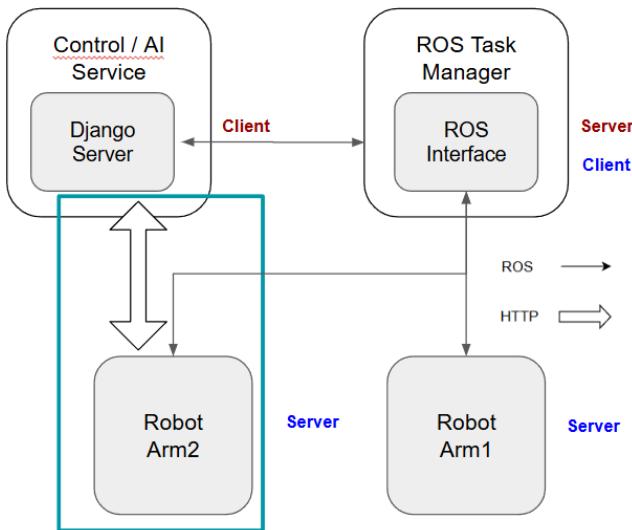
https://github.com/AndRobotics323/RoboCallee_Server.git

2. 아래 파일을 /Robocallee_Server 안에 추가해준다.



→ 꼭 /Robocallee_Server 안은 아니어도 되고 본인 편한 곳에다 저장하면 된다. 암튼 터미널끼리
ROS_DOMAIN_ID는 맞춰놔야 함

Robot Arm 통신



- ROS Service Client / Server
- Django Server / ROS Interface
- ROS Interface / RobotArm1, 2
→ request로 명령, response로 수행 결과 전달
- AI Service 와의 HTTP 통신 방법 구축
- RobotArm2에서 카메라 송출 화면에 대한 OCR inference 요청
Ex) 짜이한 상자 정보 파악 or 상자 놓을 버퍼 위치 구분
- Django Server는 POST request를 받은 후,
탐지된 ‘단어’ 및 ‘픽셀 좌표값’ response
→ 상자까지의 depth 추정 및 참고 보관 정보
업데이트 등에 활용 가능

4

여기서 RobotArm 1, 2에 해당하는 **ROS service server**를 열어두기 위함이다. FMS의 RosInterface.cpp에서 클래스 init 하는 부분을 보면 알 수 있듯 'arm1_service', 'arm2_service'에 해당하는 걸 실행시켜놔야

fms도 작동하기 때문에 하는 일. 그리고 Django Server는 fms의 **ROS service client**이므로 fms를 실행시켜놔야 아래에서 설명할 'python manage.py 어쩌구'가 잘 작동함

6.4.2 [가상환경 설정]

1. 가상환경 생성: \$ python3 -m venv <가상환경이름>

- e.g.,) \$ python3 -m venv myvenv
- 2. 가상환경 실행: \$ source <가상환경이름>/bin/activate
- e.g.,) \$source myvenv/bin/activate

6.4.3 [장고 환경 설정]

**** 가상 환경 안에서 ****



\$ pip install -r requirements.txt

1. pip install --upgrade pip
2. pip install django
3. pip install numpy
4. pip install opencv-python
5. pip install posix_ipc
6. pip install Pillow
7. pip install pyyaml (이거 뭐 빨간 글씨 좀 뜰 텐데 설치 된 거임)

외에도 본인 환경에서 부족한 pip가 있다면 알잘딱깔센으로 설치해두도록 하자

6.4.4 [서버 실행 방법]

1. robocallee_fms의 setup.bash sourcing 하기
-> \$ source /<폴더가_있는_경로>/roscamp-repo-3/Service/ROS_Task_Manager/robocallee_fms/install/setup.bash

- ~/.bashrc에 포함시켜 놓으면 편하다.

물론 colcon build 잘 돼서 install 안에 local_setup.bash가 생성된 상태에서 해야 한다.

```
source /opt/ros/jazzy/setup.bash
source ~/pinky_violet/install/local_setup.bash
source /home/addinedu/dev_temp/roscamp-repo-3/Service/ROS_Task_Manager/robocallee_fms/install/setup.bash
```

2. (terminal1)\$ **ros2 run robocallee_fms robocallee_fms_node**

**** 가상환경 안에서, /RoboCalle_Server 안에서 ****

sourcing이 필수고 가상환경은 아님. 우리가 만든 custom msg 이용하기 위한 작업으로 파일 내부의
from robocallee_fms.srv import RobotArmRequest 등을 성공적으로 하기 위함.

3. (terminal2) \$ **python3 jetco1_service_server.py**

→ 만약 robocallee_fms.srv not found와 비슷한 메시지가 출력된다면 robocallee_fms의 setup.bash
sourcing이 안 된 것인니 다시 sourcing 한다.

4. (terminal3) \$ **python3 jetco2_service_server.py**

jetcobot server는 가상환경은 안 해도 되는데 여기서부터 manage.py 류 실행할 때는 <가상환경이름>/bin/
activate 필수

이건 DB 관련된 명령어이긴 한데

python manage.py makemigrations

python manage.py migrate

도 일단 해주도록 하자.

6.4.5 계정 생성???

뭔 소리냐면 아래 9. 를 위한 관리자 계정을 만드는 과정. 위의 custom msg 불러오는 sourcing을 여기 터미널에
서도 한 다음, 가상 환경에서

python manage.py createsuperuser

The screenshot shows a sidebar with links like '2-02 모델', '2-03 장고 관리자' (highlighted in blue), '2-04 조회와 템플릿', etc. The main content area has a title '슈퍼유저' and a note about creating a superuser. Below is a terminal window showing the command execution:

```
(mysite) c:\projects\mysite> python manage.py createsuperuser
사용자 이름 (leave blank to use 'pahke'): admin
이메일 주소: admin@mysite.com
Password:
Password (again):
비밀번호가 너무 짧습니다. 최소 8 문자를 포함해야 합니다.
비밀번호가 너무 일상적인 단어입니다.
비밀번호가 전부 숫자로 되어 있습니다.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

(mysite) c:\projects\mysite>
```

이메일은 딱히 상관없을 거 같긴 한데.. 그냥 통일성을 위해 ID와 비번은 아래와 같이 설정했음

- Username: Rotus
- Password: RoboCallee

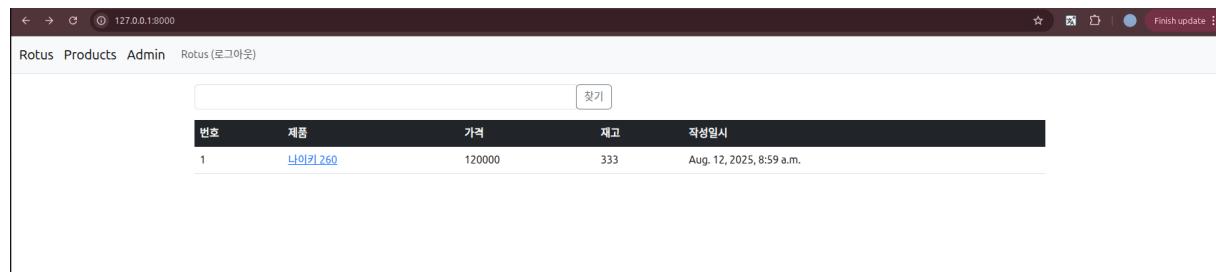
(위 캡처에서 Password: 다음에 키보드 입력해도 화면에는 뭔가 안 뜨는 게 정상이다)

5. (terminal4) \$ python manage.py runserver 물론 여기는 가상환경 안에서

7. 웹 사이트 접속

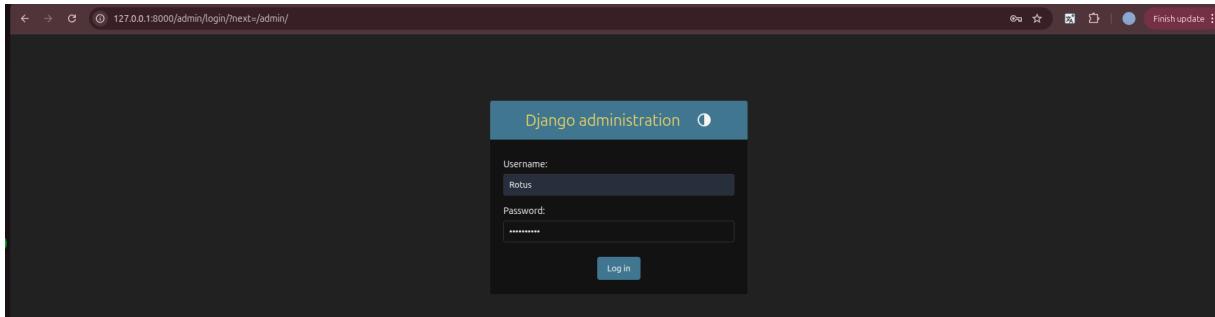
→ 웹 사이트 주소: <http://127.0.0.1:8000/>

8. 처음에는 아래 페이지와 달리 아무것도 안 뜬다

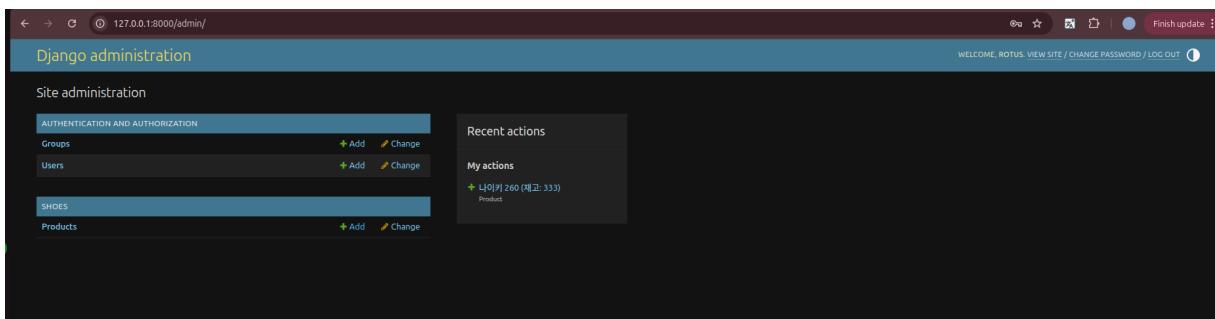


9. 관리자 페이지로 접속한다. <http://127.0.0.1:8000/admin>

- Username: Rotus
- Password: RoboCallee



9. SHOES에서 Add를 클릭한다.



10. 아래와 같이 정보를 입력해준다.

- a. Model, Size, Color, X, Y, Fitting area, Photo, Price, Stock
- b. Photo에는 아래와 같이 Screenshot한 사진을 사용하면 된다.



[127.0.0.1:8000/admin/shoes/product/add/](#)

Please correct the errors below.

Modify date:	Date: 2025-08-19 Today
Time:	16:00:00 Now
Note: You are 9 hours ahead of server time.	
Model:	Nike2
Size:	260
Color:	White
X:	1.0
Y:	2.0
Fitting area:	1번 소매
Photo:	<input type="button" value="Choose File"/> No file chosen
Description:	This field is required. shoes
Create date:	Date: 2025-08-12 Today Time: 09:53:30 Now
Note: You are 9 hours ahead of server time.	
Price:	1200
Stock:	333

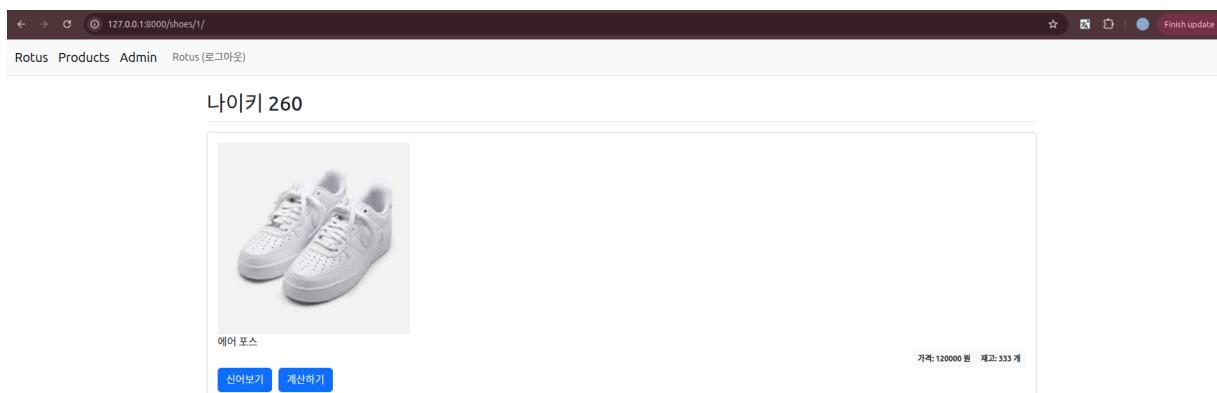
c. 정보를 다 입력한 후 맨 아래에 있는 SAVE를 누르면 된다.

11. <http://127.0.0.1:8000/>로 돌아가면 신발이 추가된 것을 확인할 수 있다.

[127.0.0.1:8000](#)

번호	제품	가격	재고	작성일시
1	나이키 260	120000	333	Aug. 12, 2025, 8:59 a.m.
2	Nike2 260	1200	333	Aug. 12, 2025, 9:53 a.m.

12. 제품명을 클릭하면 아래 페이지로 넘어간다.



13. “신어보기”를 클릭하면, 아래와 같이 FMS로 신발 요청이 전송된 것을 확인할 수 있다.

```
[2025-08-12 19:38:21] [INFO] [virtual int core::Core::RequestCallback(const Commandefine::GUIRequest&)] Request received: 나이키
[2025-08-12 19:38:21] [INFO] [virtual int core::Core::RequestCallback(const Commandefine::GUIRequest&)] Request from: customer
[2025-08-12 19:38:21] [INFO] [int Manager::RequestManager::EnqueueRequest(const Commandefine::GUIRequest&)] EnqueueRequest 완료
[2025-08-12 19:38:21] [INFO] [int Manager::RequestManager::EnqueueRequest(const Commandefine::GUIRequest&)] wait_list: 0
[2025-08-12 19:38:21] [INFO] [void Manager::RequestManager::BestRobotSelector()] 선택된 AMR: AMR0
[2025-08-12 19:38:21] [INFO] [void Manager::RequestManager::PopRequest(Commandefine::GUIRequest&)] PopRequest 완료
[2025-08-12 19:38:21] [INFO] [void Adapter::AmrAdapter::SetTaskInfo(const Commandefine::GUIRequest&)] color = White, model = 나이키 size = 260, robot_id = 0
```

6.4.6 Django 서버와 로봇팔 연동 후 OCR 연산

OCR 경로: RobotArm(Flask 스트림) → Django(직접 HTTP) → (결과 회신)

로봇팔은 **Django OCR API**를 POST로 호출하고,
Django는 **Flask 스트림에서 프레임 하나 뽑아 EasyOCR 후,**
Base64 이미지 + 텍스트/좌표를 JSON으로 응답,
로봇팔은 필요한 정보만 추출해서 바로 사용

6.4.6.1 # 로봇팔 ↔ 장고 통신

6.4.6.1.1) 로봇: Django에 OCR 요청 & 결과 활용

```
# robot_side_ocr_client.py
import requests, base64, cv2, numpy as np, re

DJANGO_URL = 'http://<django_ip>:8000/gwanje/ocr_from_flask_stream/'

DEFINED_MODELS = ['나이키', '아디다스', '뉴발란스', '반스', '컨버스', '푸마']
DEFINED_COLORS =
['white', 'black', 'red', 'blue', 'green', 'yellow', 'gray', 'brown', 'pink', 'purple']
DEFINED_SIZES =
['230', '235', '240', '245', '250', '255', '260', '265', '270', '275', '280', '285', '290']

def ask_django_ocr(url=DJANGO_URL, mode='get_shoe_info', timeout=(2, 8)):
    # 필요시 stream_url을 body로 넘기도록 확장 가능:
    # resp = requests.post(url, json={"stream_url": "<http://robot:5000/stream>"}, timeout=timeout)
    resp = requests.post(url, timeout=timeout)
    resp.raise_for_status()
    data = resp.json()

    # 빈 응답 처리
    if not data or 'results' not in data:
        return None

    word_coords = data['results']

    if mode == 'get_coords':
        return word_coords

    elif mode == 'get_shoe_info':
        info = {'model': 'unknown', 'color': 'unknown', 'size': -1}
        for item in word_coords:
```

```

text = item.get('text', '')
if re.fullmatch(r'[A-Za-z]+', text) and text in DEFINED_COLORS:
    info['color'] = text
elif re.fullmatch(r'[가-힣]+', text) and text in DEFINED_MODELS:
    info['model'] = text
elif re.fullmatch(r'\\d{2,3}', text) and text in DEFINED_SIZES:
    info['size'] = int(text)
return [info]

elif mode == 'show_image':
    img_b64 = data.get('image')
    if not img_b64:
        return None
    img = cv2.imdecode(np.frombuffer(base64.b64decode(img_b64), np.uint8),
cv2.IMREAD_COLOR)
    cv2.imshow('OCR Result', img)
    while True:
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()
    return None

if __name__ == '__main__':
    print(ask_django_ocr(mode='get_shoe_info'))

```

- resp = requests.post(url, timeout=timeout) 로 타임아웃 설정
- 결과가 없을 때 None 처리
- 세 가지 모드: 좌표만, 신발정보만, 결과이미지 보기

6.4.6.1.2 2) 장고: URL 라우팅

```

# gwanje/urls.py
from django.urls import path
from . import views

app_name = 'gwanje'

urlpatterns = [
    path('ocr_from_flask_stream/', views.ocr_from_flask_stream,
name='ocr_from_flask_stream'),
    # (참고) cam, video_feed, come_here, done_employee 등 다른 엔드포인트도 존재
]

```

이 라인이 `/gwanje/ocr_from_flask_stream/` URL로 들어온 요청을
`views.ocr_from_flask_stream()`에 연결.

6.4.6.1.3 3) 장고: OCR 뷰 (Flask 스트림 → EasyOCR → JSON)

```
# gwanje/views.py (핵심 부분만)
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from .camera_jetco import run_ocr_from_flask

# Create your views here.
@csrf_exempt
def ocr_from_flask_stream(request):
    # result_img_b64, text = run_ocr_from_flask() # base64 인코딩된 이미지 + OCR 결과
    result_img_b64, word_coords = run_ocr_from_flask()

    return JsonResponse({
        'image': result_img_b64,
        'results': word_coords # 리스트: [[(x1,y1), (x2,y2), ...], [...], ...]
    })
```

`ocr_from_flask_stream()` 를 실행하고 인코딩된 이미지와 OCR 결과를 JSON 형식으로 반환

6.4.6.1.4 4) 장고: 실제 OCR 로직 (MJPEG에서 프레임 1장 뽑고 EasyOCR)

```
# gwanje/camera_jetco.py (요지: 네 코드 기반)
import requests, cv2, numpy as np, easyocr, base64

reader = easyocr.Reader(['ko', 'en'], gpu=False)

def run_ocr_from_flask():
    stream = requests.get("<http://192.168.0.168:5000/stream>", stream=True)
    byte_data = b''

    for chunk in stream.iter_content(chunk_size=1024):
        byte_data += chunk
        start = byte_data.find(b'\xff\xd8')
        end = byte_data.find(b'\xff\xd9')
        if start != -1 and end != -1:
            jpg = byte_data[start:end+2]
            npimg = np.frombuffer(jpg, np.uint8)
            frame = cv2.imdecode(npimg, cv2.IMREAD_COLOR)
            break

    result = reader.readtext(frame)

    word_coords = []

    for (bbox, text, conf) in result:
        # (tl, tr, br, bl) = bbox
        # tl = tuple(map(int, tl))
        # br = tuple(map(int, br))
        # cv2.rectangle(img, tl, br, (0, 255, 0), 2)
        # cv2.putText(img, text, tl, cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 1)
```

```

pts = np.array(bbox).astype(int)

coord_list = []
for (x, y) in pts:
    x, y = int(x), int(y)
    coord_list.append((x, y))
    cv2.circle(frame, (x, y), 3, (255, 255, 0), -1) # 초록 점 찍기

word_coords.append({'text': text, 'coords': coord_list})

# 이미지 base64 인코딩
_, buf = cv2.imencode('.jpg', frame)
img_b64 = base64.b64encode(buf).decode('utf-8')

# return img_b64, [r[1] for r in result]
return img_b64, word_coords

```

- MJPEG 스트림에서 JPEG 한 장만 추출
- EasyOCR로 (텍스트, 네 꼭짓점 좌표, 신뢰도) 추출
- 결과 이미지를 Base64로 반환해 웹/모바일에서 즉시 미리보기 가능

6.4.6.2 # 문제 상황

로봇에서 Django 서버의 `/gwanje/ocr_from_flask_stream/` 엔드포인트로 `requests.post()` 요청 시:

```

ConnectionError: HTTPConnectionPool(host='192.168.5.17', port=8000)...
(Caused by NewConnectionError: [Errno 111] Connection refused)

```

의미: TCP 연결 자체가 거절됨 → 해당 IP:PORT에서 서버가 리스닝 중이지 않음.

6.4.6.2.1 원인

- `python manage.py runserver` 기본 실행 시, **127.0.0.1:8000**로만 바인딩됨.
- 이렇게 되면 Django 서버는 서버 자신만 접근 가능하고, 같은 네트워크의 다른 장치(예: 로봇)는 접속 불가.

6.4.6.2.2 해결 방법

1. 외부 접속 허용을 위해 **0.0.0.0**으로 바인딩:

```

python manage.py runserver 0.0.0.0:8000

```

- 0.0.0.0 : 서버의 모든 네트워크 인터페이스에서 접속 허용
- :8000 : 8000번 포트로 서비스

1. 서버 IP 확인

서버 쪽에서:

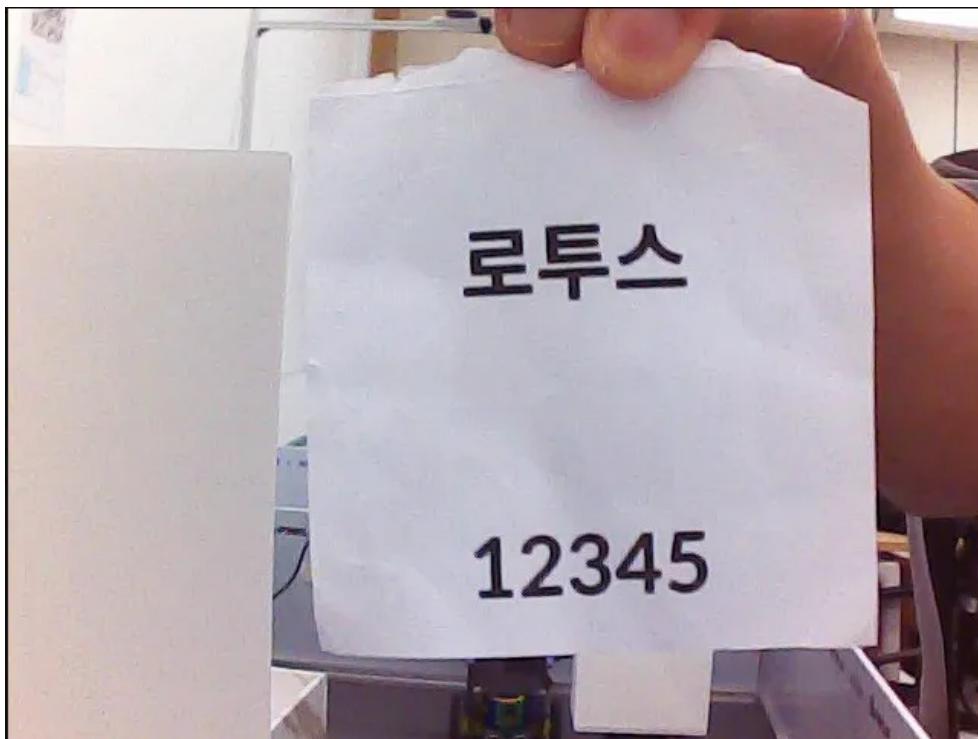
```
hostname -I
```

예) 192.168.5.17

1. 로봇에서 요청 URL 설정:

```
url = "<http://192.168.5.17:8000/gwanje/ocr_from_flask_stream/>"  
requests.post(url)
```

6.4.6.3 # 실행 결과



```
jetcobot@raspi:~/CJ_ws$ /usr/bin/python3 /home/jetcobot/CJ_ws/http_request.py  
200  
[{'text': '로투스', 'coords': [[376, 134], [556, 134], [556, 212], [376, 212]]}, {'text': '12345',  
'coords': [[377, 362], [570, 362], [570, 434], [377, 434]]}]
```

6.4.6.4 장고에서 로봇팔2 기준 OCR 연산 결과 받아오기

```
# 장고에 OCR 결과 받기-----
time.sleep(3)
print("이미지 찍는 중")
django_url = 'http://192.168.5.17:8000/gwanje/ocr_from_flask_stream/'
shoe_info = ask_django_ocr(django_url, 'get_shoe_info')

response.model = shoe_info['model']
response.color = shoe_info['color']
response.size = shoe_info['size']
self.get_logger().info(f'[서비스 요청] model: {response.model}, color: {response.color}, size: {response.size}')
# -----
```



6.4.6.4.1 로봇팔2 service call

```
jetcobot@raspi:~/sun_ws/ros2_ws$ ros2 service call /arm2_service robocallee_fms/srv/RobotArmRequest "{amr_id: 3 , action: 'pinky_to_buffer', shelf_num: 2}"
requester: making request: robocallee_fms.srv.RobotArmRequest_Request(amr_id=3, action='pinky_to_buffer', shelf_num=2)

response:
robocallee_fms.srv.RobotArmRequest_Response(robot_id=2, amr_id=3, action='에 이 프릴 테 그 인식 실패', model='나이키', size=260, color='None', success=False)
```

6.4.6.4.2 로봇팔2 service server

```
192.168.5.17 - - [15/Aug/2025 16:16:42] "GET /stream HTTP/1.1" 200 -
[INFO] [1755242204.149745508] [robot2_control_node]: [서비스 요청] model: 나이키, color: None, size: 260
```

6.5 Robot Arm Service 실행

robot arm1 ssh 접속

```
ssh jetcobot@192.168.0.167
```

robot arm2 ssh 접속

```
ssh jetcobot@192.168.0.168
```

```
jetcobot@raspi:~
```

```
addinedu@addinedu-ASUS-TUF-Dash-F15-FX516PM-FX516PM:~$ ssh jetcobot@192.168.0.167
jetcobot@192.168.0.167's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1032-raspi aarch64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

47 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

50 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon Aug 18 19:13:50 2025 from 192.168.5.17
jetcobot@raspi:~$
```



```
jetcobot@raspi:~
```

```
addinedu@addinedu-ASUS-TUF-Dash-F15-FX516PM-FX516PM:~$ ssh jetcobot@192.168.0.168
jetcobot@192.168.0.168's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1032-raspi aarch64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

314 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

50 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon Aug 18 18:36:01 2025 from 192.168.0.167
not found: "/home/jetcobot/CJ_ws/ros_ws/src/robocallee_fms/install/local_setup.bash"
not found: "/home/jetcobot/CJ_ws/ros_ws/src/robocallee_fms/install/local_setup.sh"
jetcobot@raspi:~$
```

6.5.1 <생략 가능한 부분>

로봇팔 1,2 각각에 자신의 ROS2 워크스페이스 폴더 만들고 아래 깃 주소 clone 하기

https://github.com/SunFlower2819/ROS2_myrobot_pick_and_place.git

빌드하기

```
colcon build
```

6.5.2 이미 있는 환경 이용하기

robot arm1 환경 경로

```
cd venv/mycobot/dev/moon_ws/ros2_ws
```

robot amr2 환경 경로

```
cd sun_ws/ros2_ws
```

실행 전에 패키지 불러오기 (로봇팔1,2 둘다 해줘야 함)

```
source install/setup.bash
```

로봇팔1 서비스 서버 실행시키기

```
ros2 run pick_and_place robot1_control_node
```

로봇팔2 서비스 서버 실행시키기

```
ros2 run pick_and_place robot2_control_node
```

서비스 콜 해보기

```
ros2 service call /arm1_service robocallee_fms/srv/RobotArmRequest "{amr_id: 3, action: 'shelf_to_buffer', shelf_num: 1}"
```

```
ros2 service call /arm2_service robocallee_fms/srv/RobotArmRequest "{amr_id: 3 , action: 'pinky_to_buffer', shelf_num: 2}"
```