

[두산로보틱스] 지능형 로보틱스 엔지니어

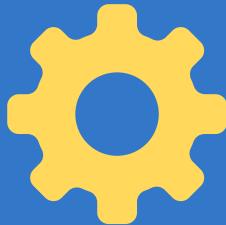
로봇 자동화 주차 시스템 (SLAM(위치추정 및 공간지도생성) 기반 자율주행 로봇 시스템 구현)

TEAM B-4 Async

[팀장] 이세현

[팀원] 강인우, 이형연

[멘토] 손미란 강사님



목 차

- 
- 01** 프로젝트 개요
 - 02** 프로젝트 팀 구성 및 역할
 - 03** 프로젝트 수행 절차 및 방법
 - 04** 프로젝트 수행 경과
 - 05** 자체 평가 의견

01 K-Digital Training

프로젝트 개요

1

프로젝트 주제 및 선정 배경, 기획의도

—
대한민국의 여러 종류의
자동차 수요 상승으로 인해
이를 주차할만한 장소와
수요가 부족한 상황,
완전 자율 주차장을 주제로
선정함.

2

프로젝트 구조

—
주차공간 인식
번호판 차종, ocr인식
해당 차종의 구역에 주차
해당 차종의 출차

3

활용 장비 및 재료

—
ros2
turtlebot4
roboflow, yolo
pyqt
influx db

4

프로젝트 내용

—
**-각 단위
-통합**

터틀봇4가 차량의
주차와 출차를 대신 해주는
컨셉

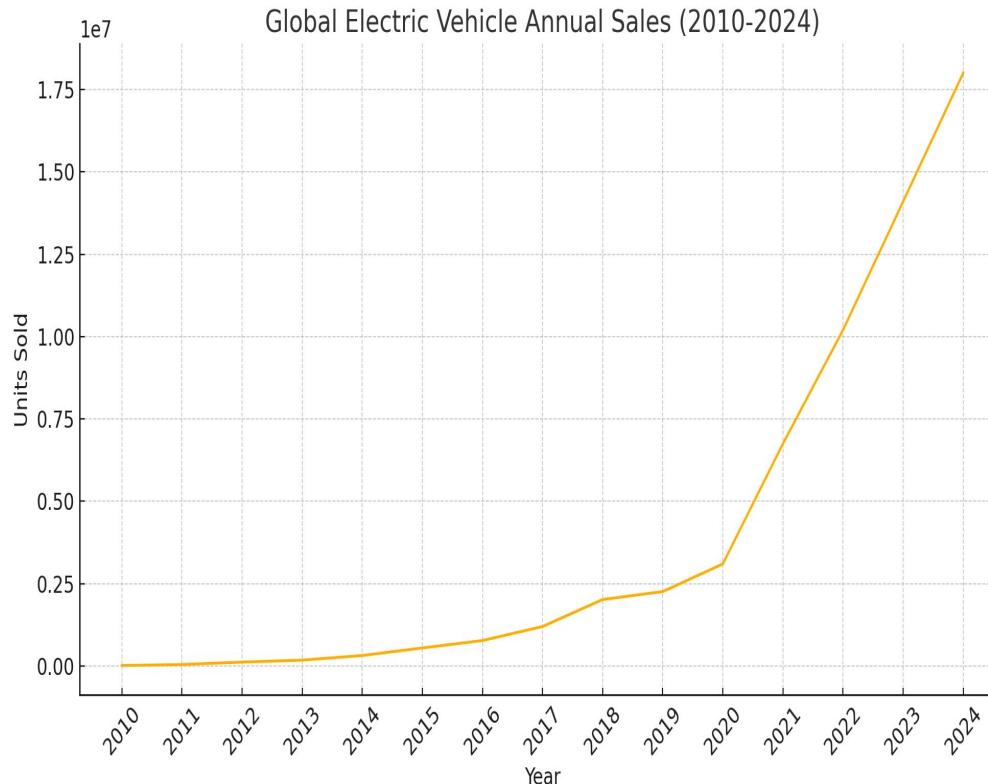
5

자체 평가 내역

—
완전 자율화이므로
노동안전사고도 없음.
정교한 설계로
자동차 접촉 사고도 없음.
프리미엄 서비스로 높은
수익과 고정된 수요가 예상

02 프로젝트 배경

▶ 증가하는 전기차수요



1. 전기차 판매, 새로운 시작

국제에너지기구(IEA)의 최신 보고서에 따르면, 2025년에 판매되는 자동차 4대 중 1대는 전기차(EV)가 될 전망이다. 이는 전 세계 자동차 판매의 약 25%를 차지하는 수치로, 2030년에는 전기차 비율이 40% 이상으로 증가할 가능성도 제기됨.

IEA의 ‘글로벌 EV 전망 보고서’에 따르면, 경제적 어려움에도 불구하고 전기차 시장은 지속적으로 성장하고 있음. 2024년 전기차 판매량은 1700만 대로 사상 최고치를 기록하며, 처음으로 글로벌 시장 점유율이 20%를 넘어섰습니다. 이러한 추세는 2025년 초에도 이어져, 1분기 전기차 판매가 전년 대비 35% 증가함.

02

프로젝트 배경

▶ 사용자 문제점



김포국제공항 실시간 주차장



실시간 주차 가능 대수는 제공처의 업데이트 시간차로 실제와 다를 수 있습니다.
요금은 공항의 할인 정책에 따라 다르게 적용될 수 있으므로 자세한 내용은 공식홈페이지에서 확인 바랍니다.

최종업데이트 2025.06.27. 14:23 • 제공 [한국공항공사](#)

주차요금

공식 주차대행

주차 예약

장소 국내선 2층 입구 공식 접수 구역

요금 일반 20,000원

예약 gmp.turuvalet.co.kr/main

전화 02-2660-4894

02

프로젝트 배경

▶ 안내/관리자 문제점



교보생명보험 주차관리요원이 1일 오후 서울 종로구 교보생명보험 본사에서 이동식에어컨 바람을 쐬며 출차하는 차량을 안내하고 있다. 사진=강민석 기자 kms@newsway.co.kr

**'2시간 작업 후 20분 휴식' 조항 빠진 채 여름 시작
열사병 첫 중대재해법 판결에도 제도는 제자리**

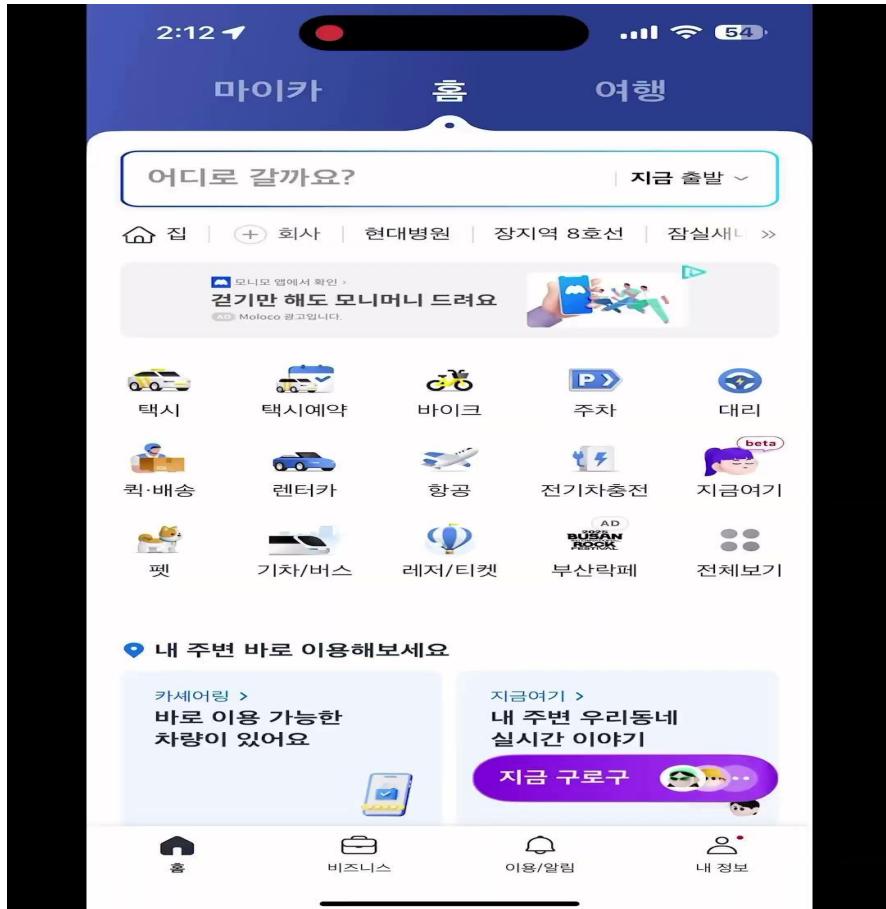


서울 성북구 장위4구역 주택정비사업 건설현장에서 폭염 대응 관련 안내문이 게시돼 있다. [뉴시스]

02

프로젝트 배경

▶ 벤치마킹 분석 - 입 출차 관리



- 전기차 충전소의 실시간 현황 파악 가능
- 완충시에도 주차공간 차지
- 일반차량의 주차공간 잔여 파악 불가
- 위치에 따른 독자적인 공간 확보가 아닌 건물 공간에 같이 서비스 제공 (전용주차장이 제공되는 지역도 있음)

02

K-Digital Training

프로젝트 배경

▶ 벤치마킹 분석 - 자율 주차

구분	핵심 포인트
공통 배경	해외에서도 다양한 주차 로봇 시스템이 상용화되어 주차 문화를 혁신 중
Lödige Industries (독일)	- Lift & Shuttle 방식 - 대규모·고밀도 시설 최적 - 덴마크 오르후스(100만 대 규모), 호주 시드니 레녹스(327대 규모) 등 설치
Serva Transport Systems (독일)	- 바닥 이동식 AGV로 기존 주차공간 최소 변경 활용 - 빠른 설치·유연한 확장 장점 - 유럽 다중 이용 시설에 확대 배치 중
Parkolay (스페인)	- AGV + 알고리즘으로 모듈 전체를 주차 공간화 - 제한된 공간에서 최대 주차량 확보 - 사용자 친화적 인터페이스 제공

경제

HL로보틱스, 프랑스 주차로봇 기업 '스탠리 로보틱스' 인수

수정 2024.10.08 15:12

이진주 기자



밸레파킹 중인 스탠리 로보틱스 주차로봇 '스탠리'. HL로보틱스 제공

https://www.hellot.net/data/photos/20250519/art_17467043855342_2c799c.gif

프로젝트 배경

▶ 자율 주차장



결제방식을 선택하고 충전구 위치를 선택하면 차량 충전구 위치에 따라 위에서 충전케이블이 내려오게 되는데, 충전케이블의 경우에는, 보통 무게가 무겁기 때문에, 이렇게 위에서 내려오는 방식은 충전을 더욱 편하고 깔끔하게 만들어준다. 무엇보다 케이블이 길고 땅에 끌리는 기존 방식과 달리 차량이 더러워지거나 긁힐 염려 등이 없어 만족스럽다. 충전기가 만족스러운 위치까지 내려오지 않았다면, 버튼을 이용해 세부 조정을 하면 되니, 크게 어려울 것이 없다.

프로젝트 팀 구성 및 역할

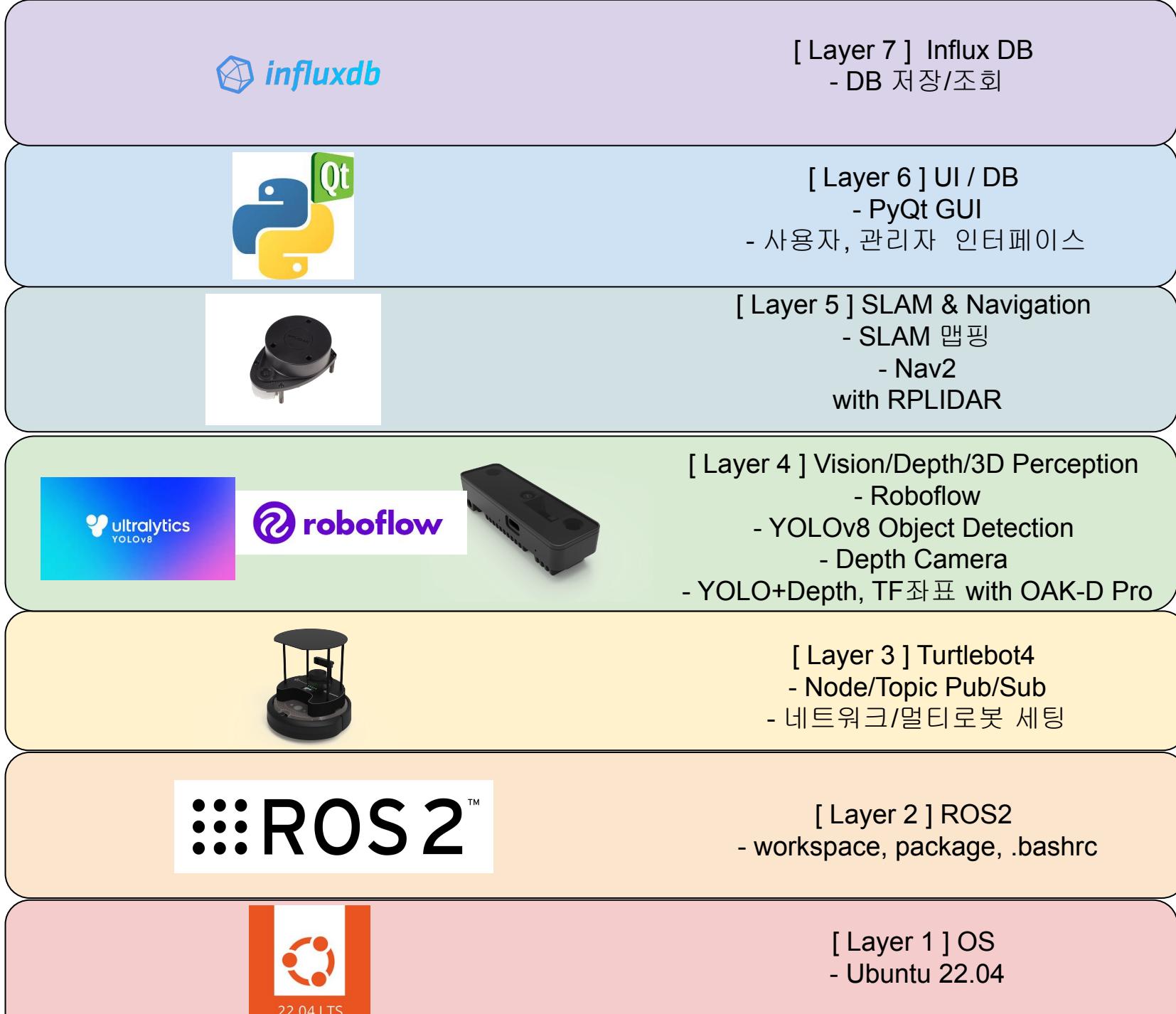
훈련생	역할	담당 업무
정서윤	팀장	SLAM 및 Navigation, 경고음 노드 연동, 통합 (gui-slam)
나승원	팀원	번호판 object detection
이동기	팀원	자료조사 및 차량 훈련 학습
홍진규	팀원	자료조사, 영상편집
이형연	팀원	GUI, 통합(번호판 object, ocr -gui), InfluxDB 설계
강인우	팀원	번호판 ocr, 주차공간 depth TF, 통합 (navi + tf)
이세현	팀원	주차공간 object detection, 주차공간 경고음 연동, 시스템 부하 측정 및 비교(경량화)

04

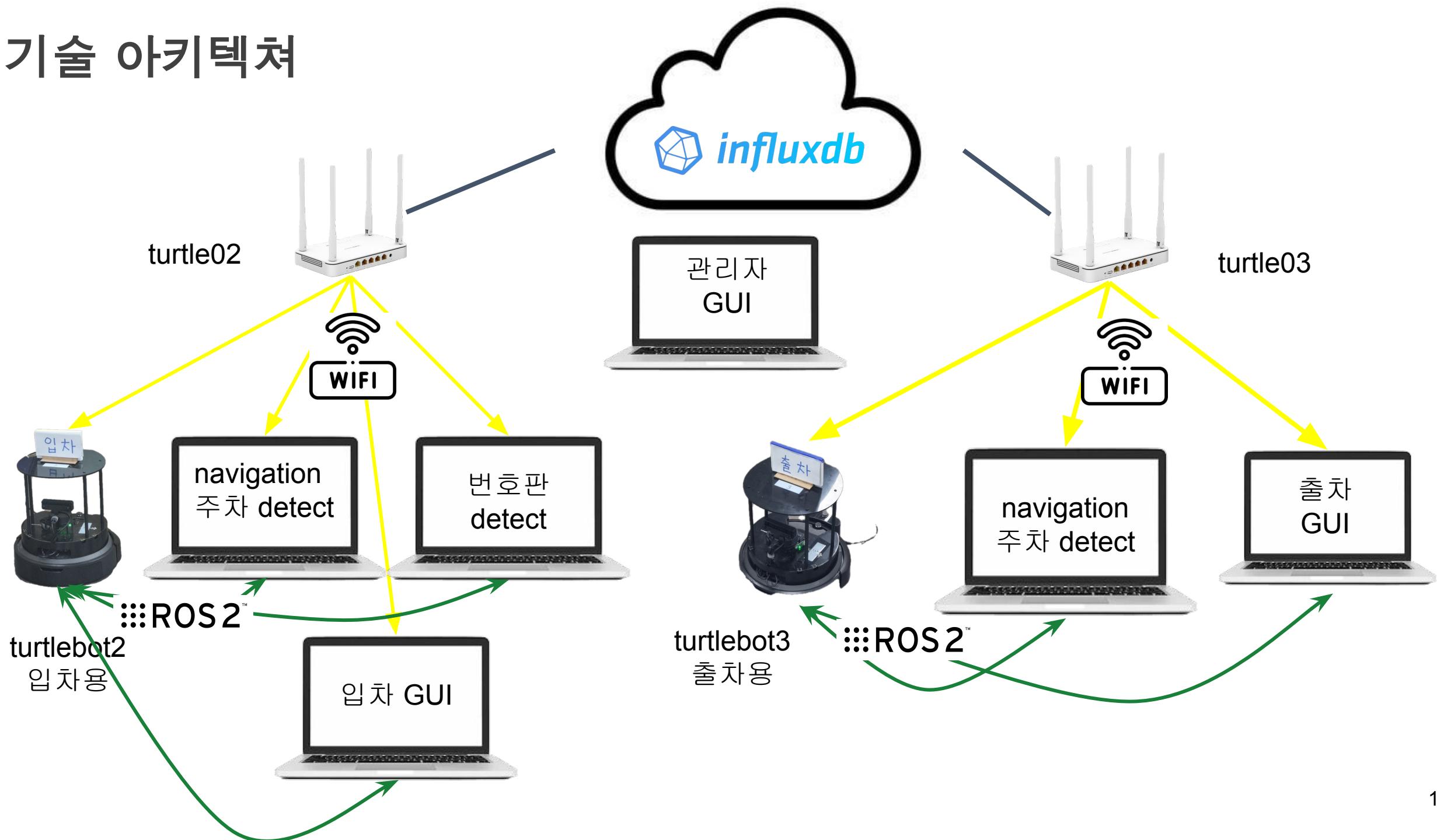
프로젝트 수행 절차 및 방법



기술 스택

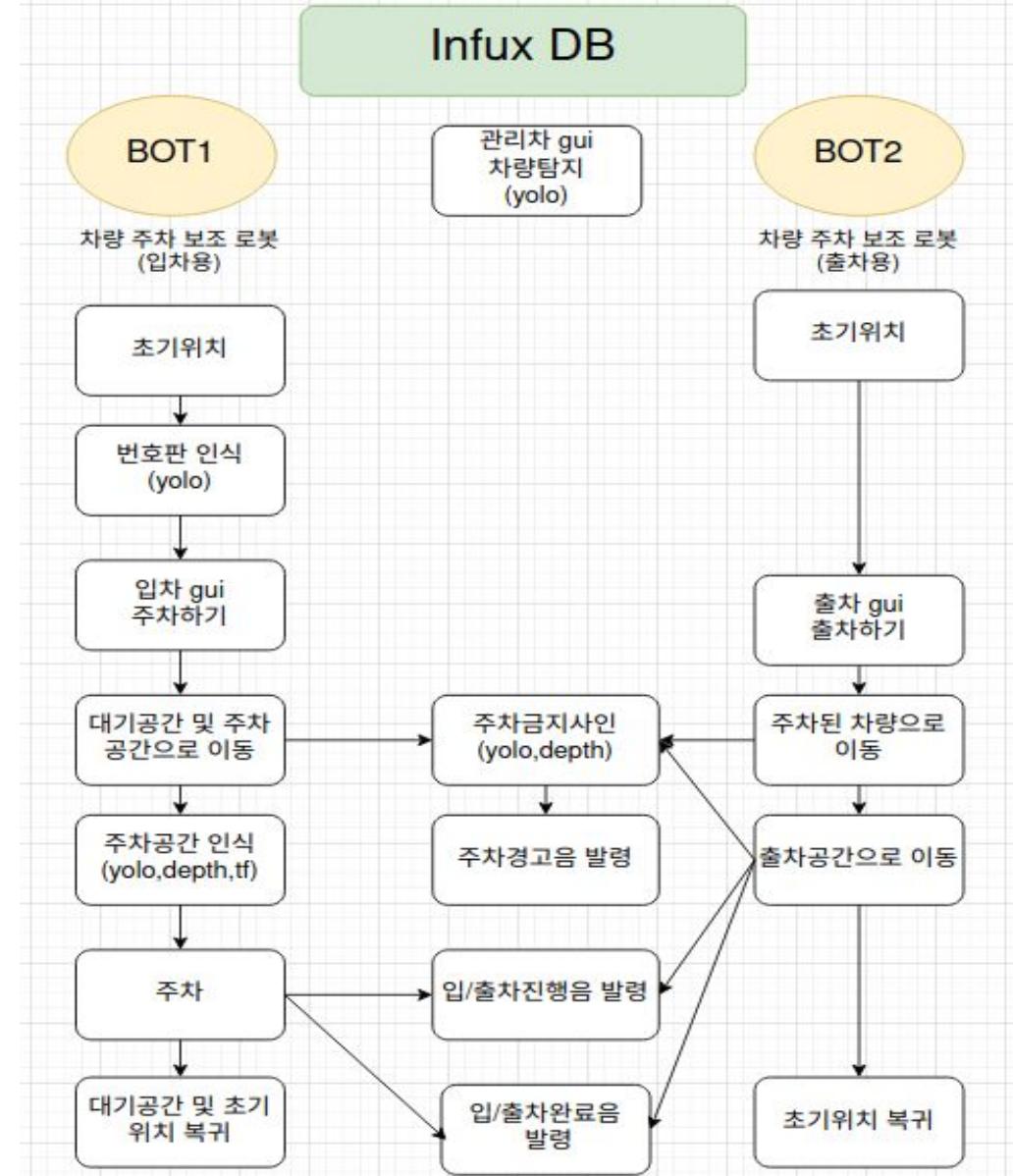


기술 아키텍쳐



Flow Chart

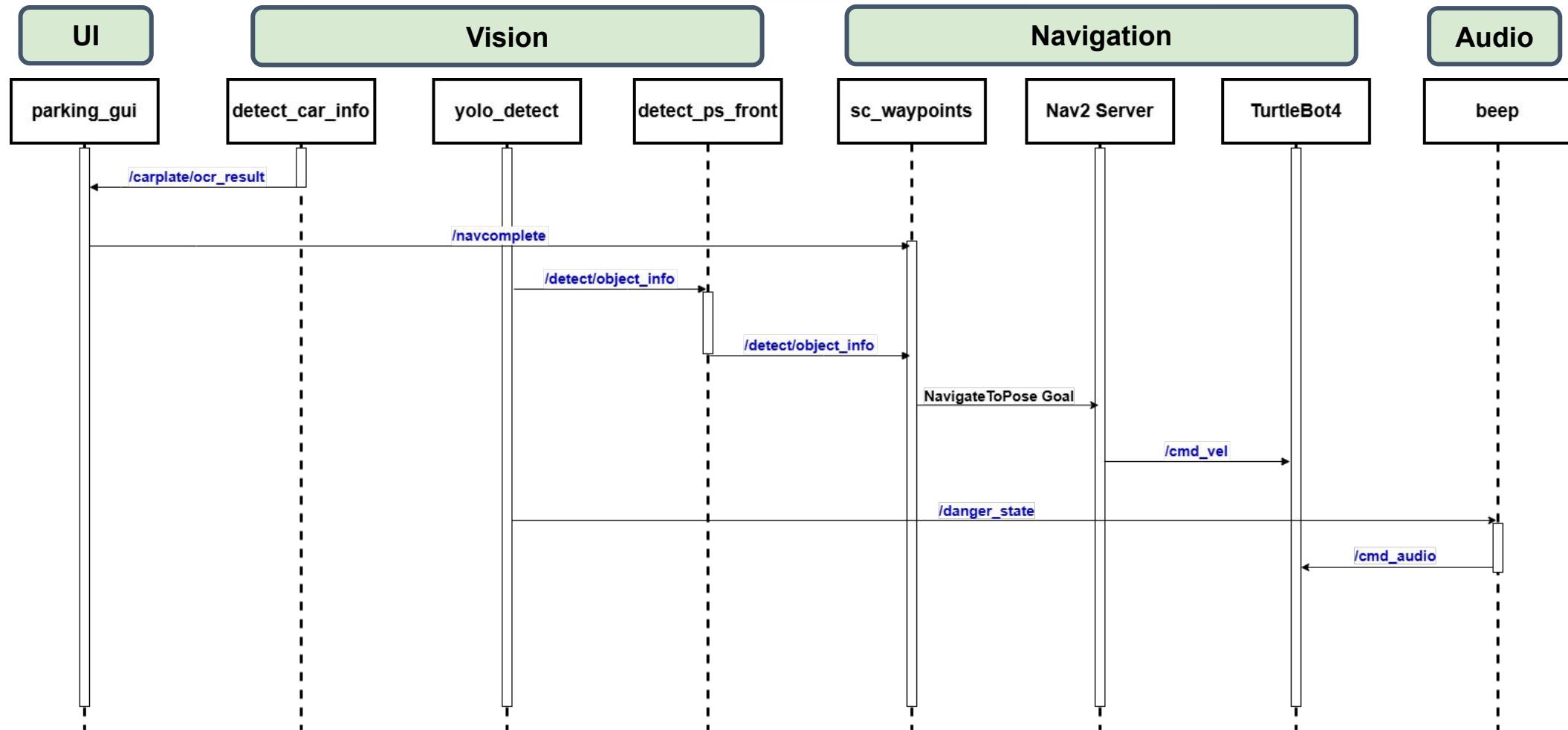
로봇 자동화 주차 시스템



05

프로젝트 수행 경과

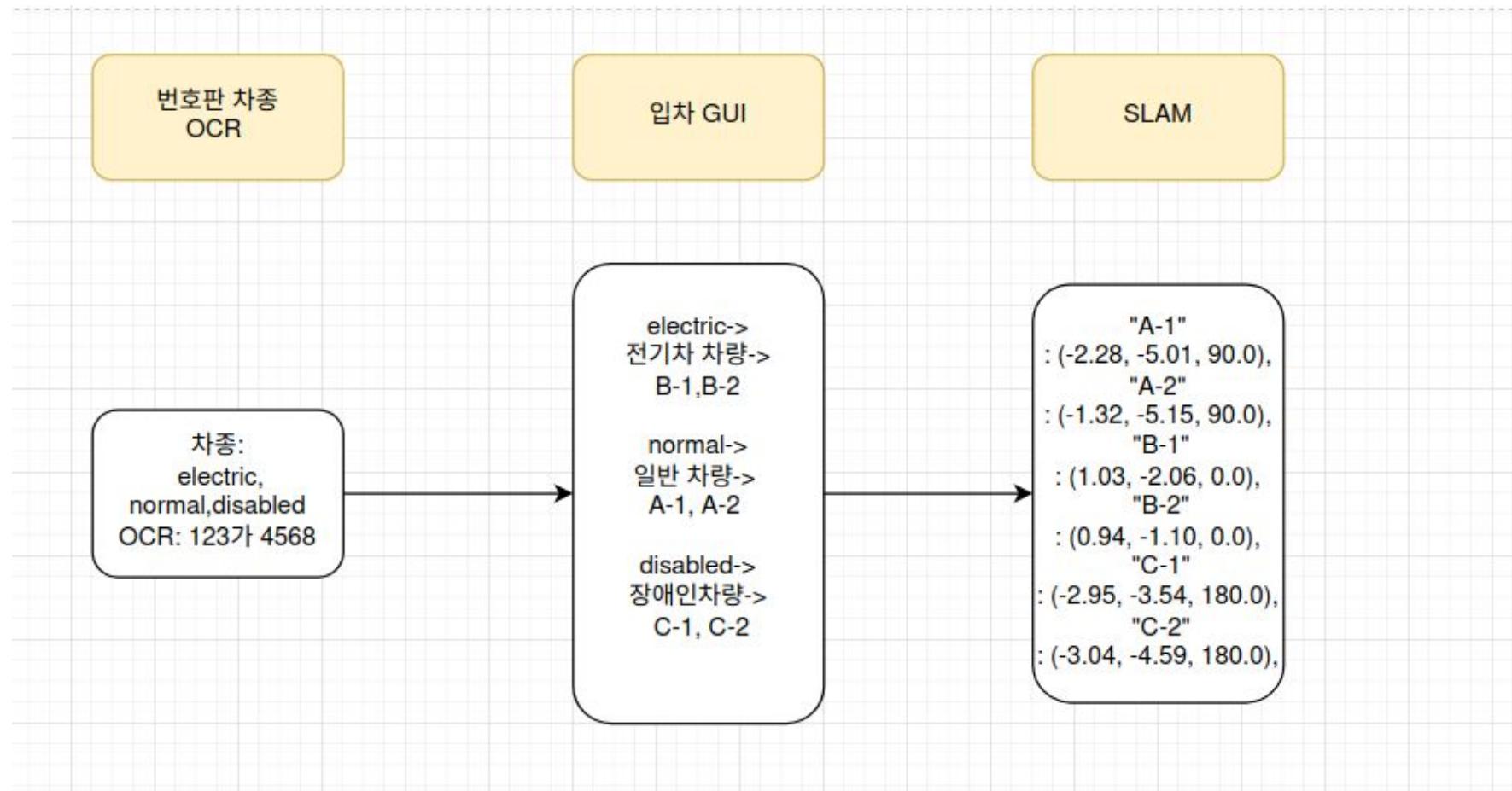
▶ 알고리즘 – 시퀀스 다이어그램



05

프로젝트 수행 경과

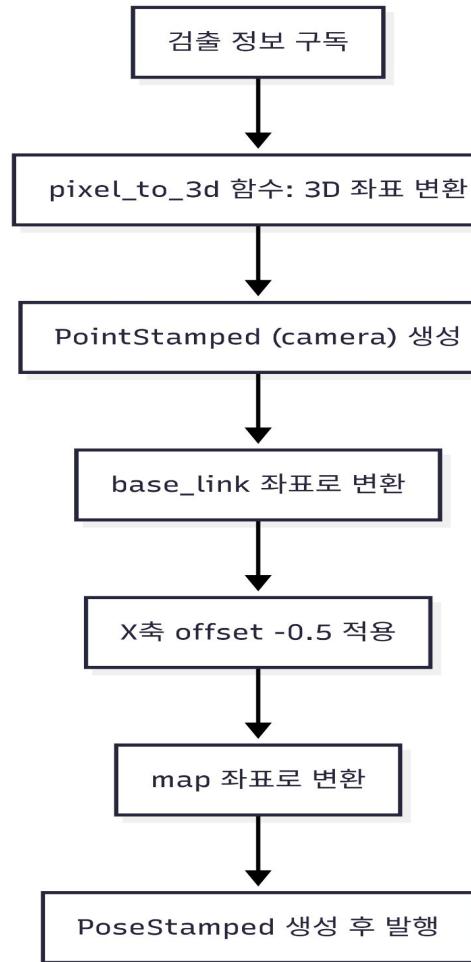
- ▶ **입차하기 :** 번호판 OCR-> GUI -> Navi



05

프로젝트 수행 경과

▶ 입차하기 : Depth-> TF



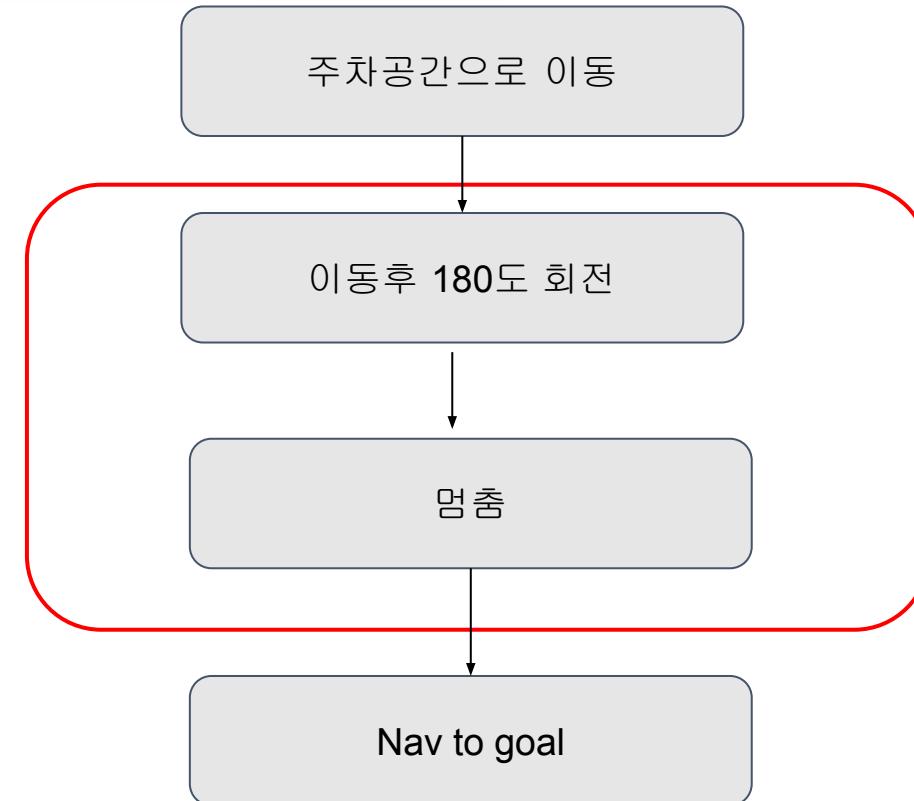
05

프로젝트 수행 경과

▶ 입차하기 : Navigation



클릭시 영상 재생



05

프로젝트 수행 결과

▶ 재료 및 물품



Turtlebot2



Turtlebot3



입차용 수레



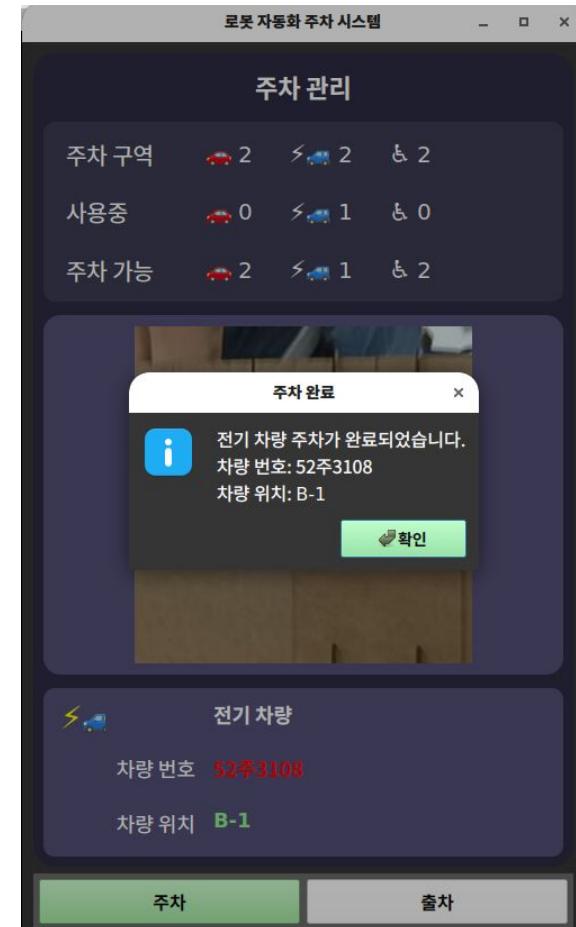
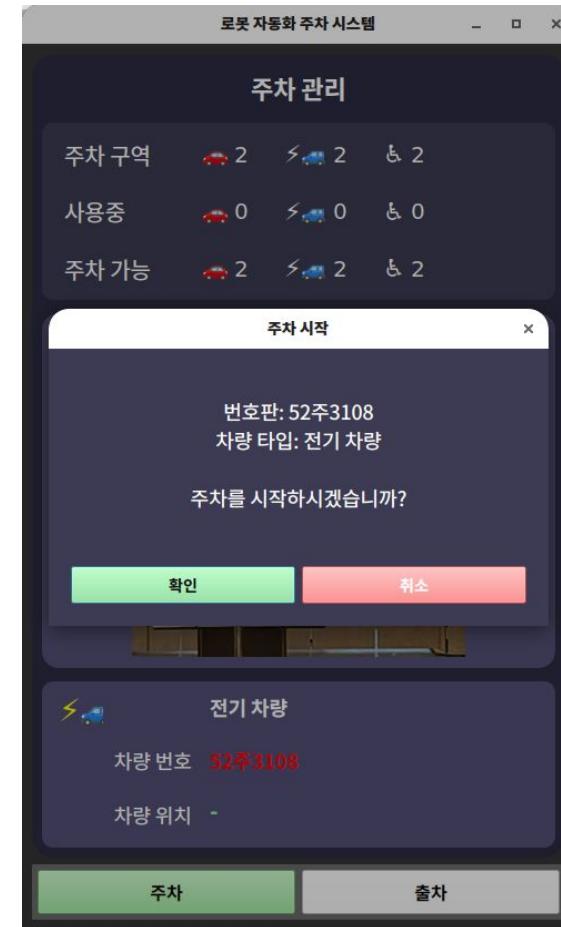
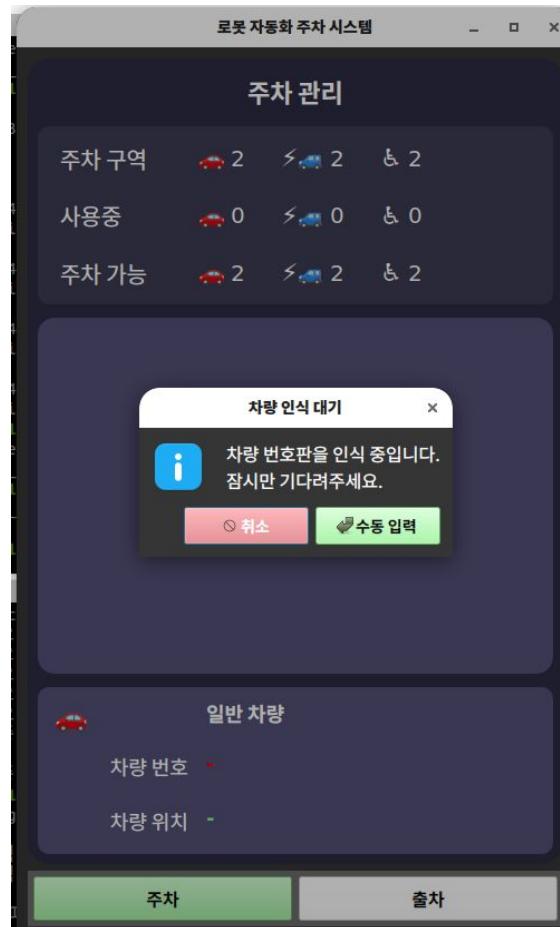
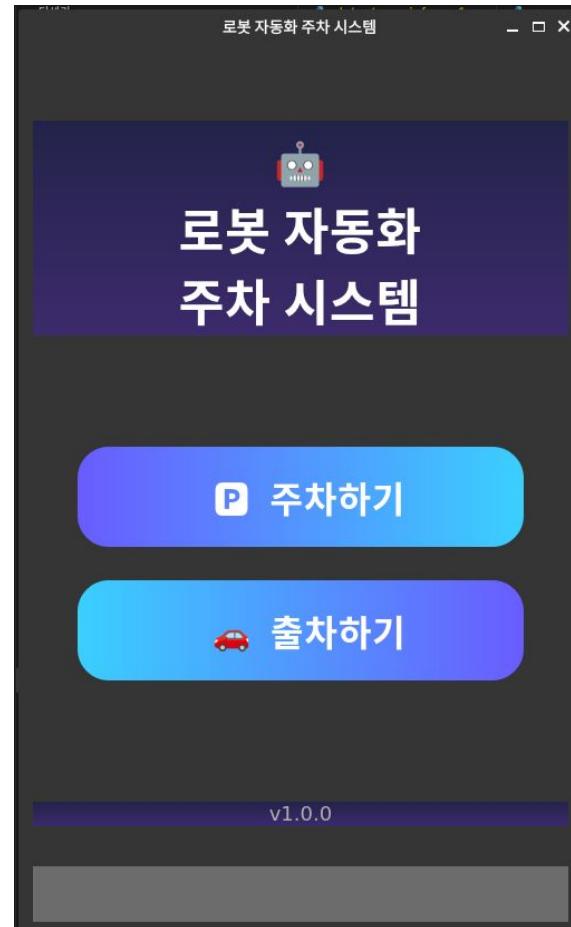
출차용 수레

05

K-Digital Training

프로젝트 수행 경과

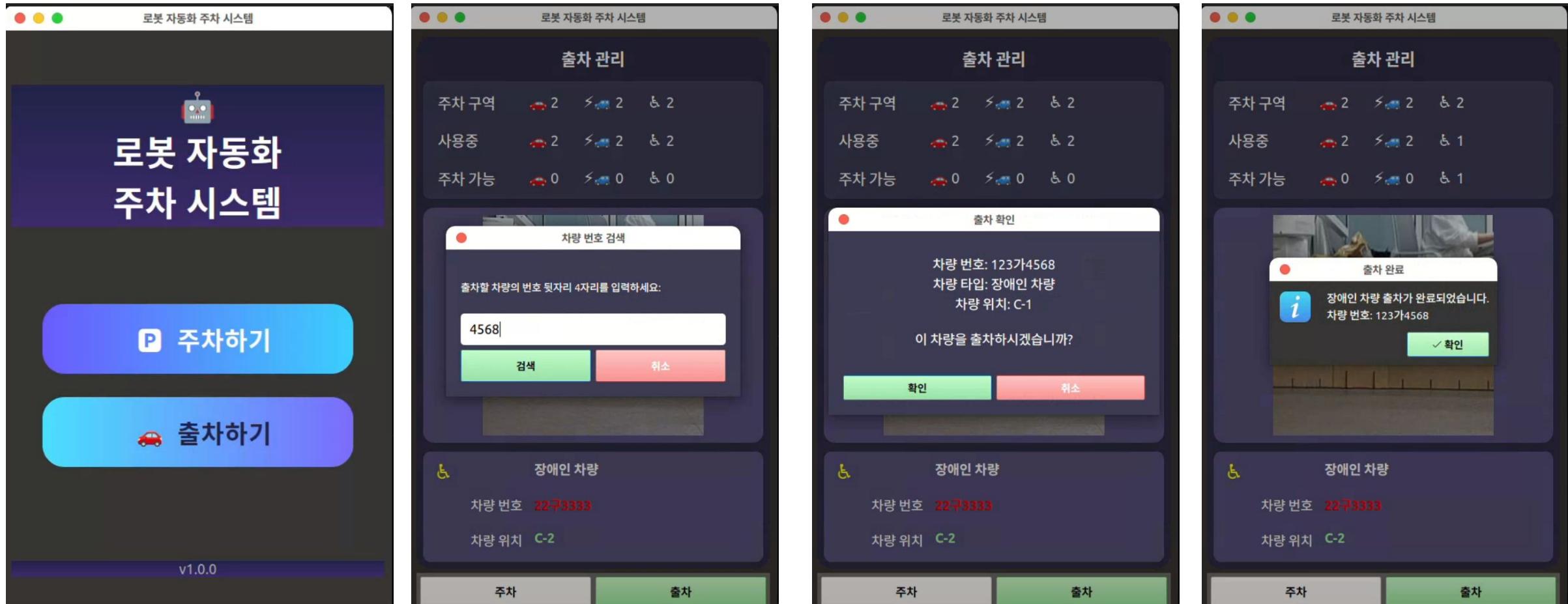
▶ GUI 사용자 – 주차하기



05

프로젝트 수행 경과

▶ GUI 사용자 -출차하기

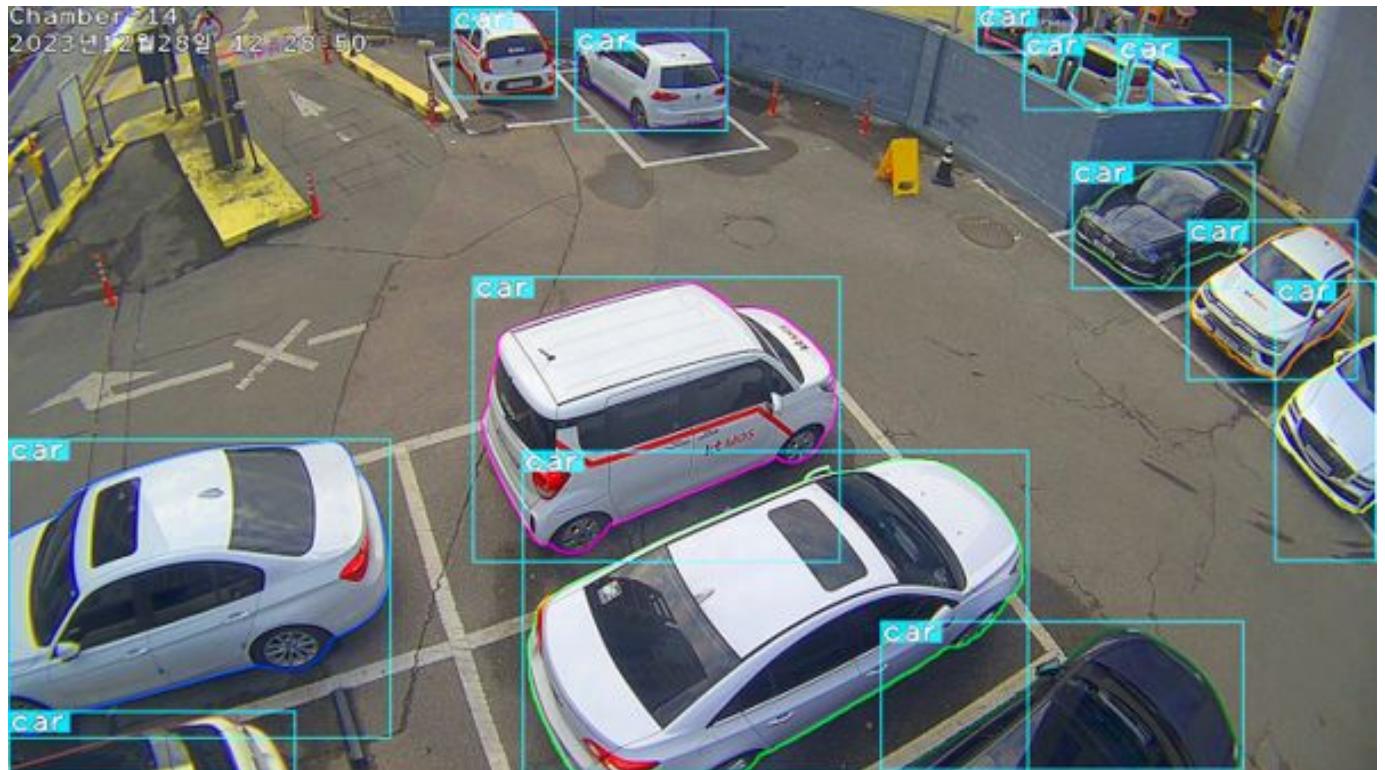


05 프로젝트 수행 경과

K-Digital Training

▶ GUI 관리자

- 자율화 주차장 안에서도 사람은 출입 가능.
그러므로 어떠한 경우의 사건사고의 가능성이 있음.
- 주차장 안에서 자동차를 객체인식하여 사고를 미연에 방지.
- 단순한 CCTV가 아닌 객체인식으로 차후에 보안 기능을 적용 가능.

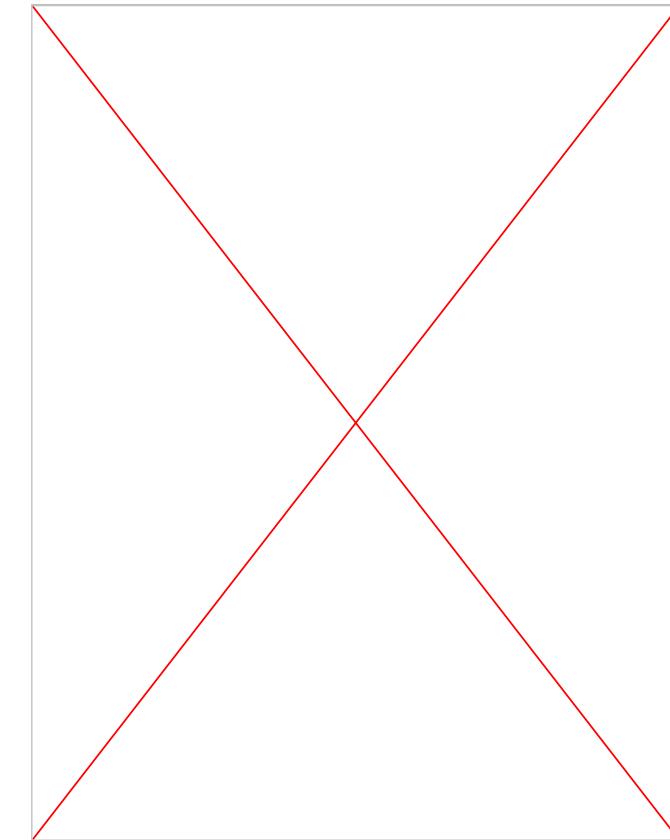
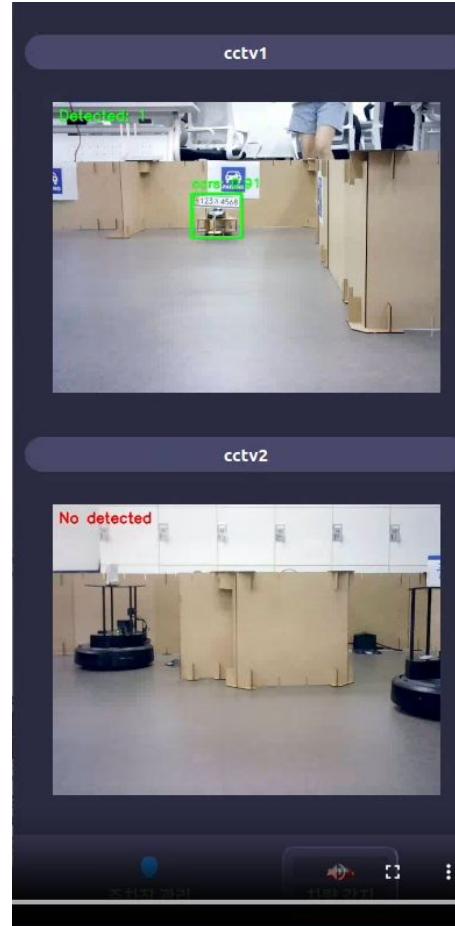
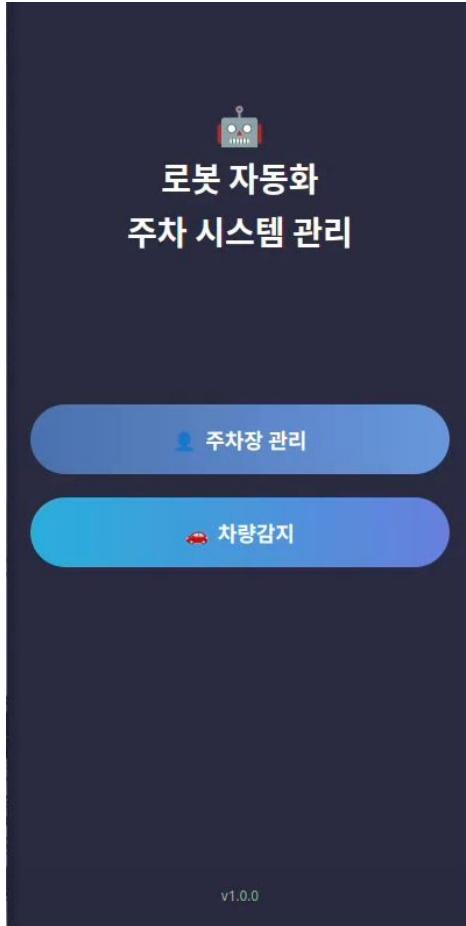


05

K-Digital Training

프로젝트 수행 경과

▶ GUI 관리자 – 차량 감지



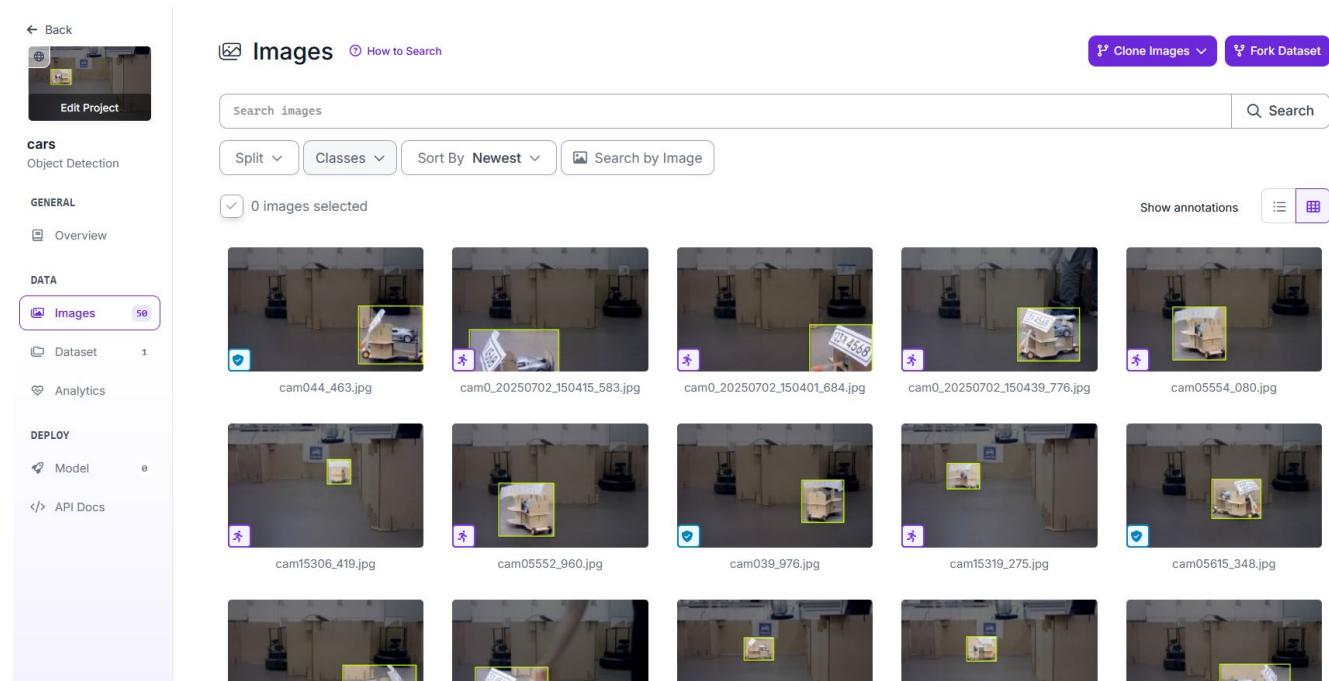
클릭시 영상 재생

05

프로젝트 수행 경과

▶ GUI 관리자– 차량 감지 dataset

- Model: YOLOv8n
- source_img : 50개(class 1개)
- Train / Valid / Test : 7:3:0
- Augmentation X
- epochs: 100, imgsz:320, batch: 16



Dataset Split

TRAIN SET

70%

35 Images

VALID SET

30%

15 Images

TEST SET

%

0 Images

05

K-Digital Training

프로젝트 수행 경과

- ▶ 주차공간분류(Disabled, Normal, EV)



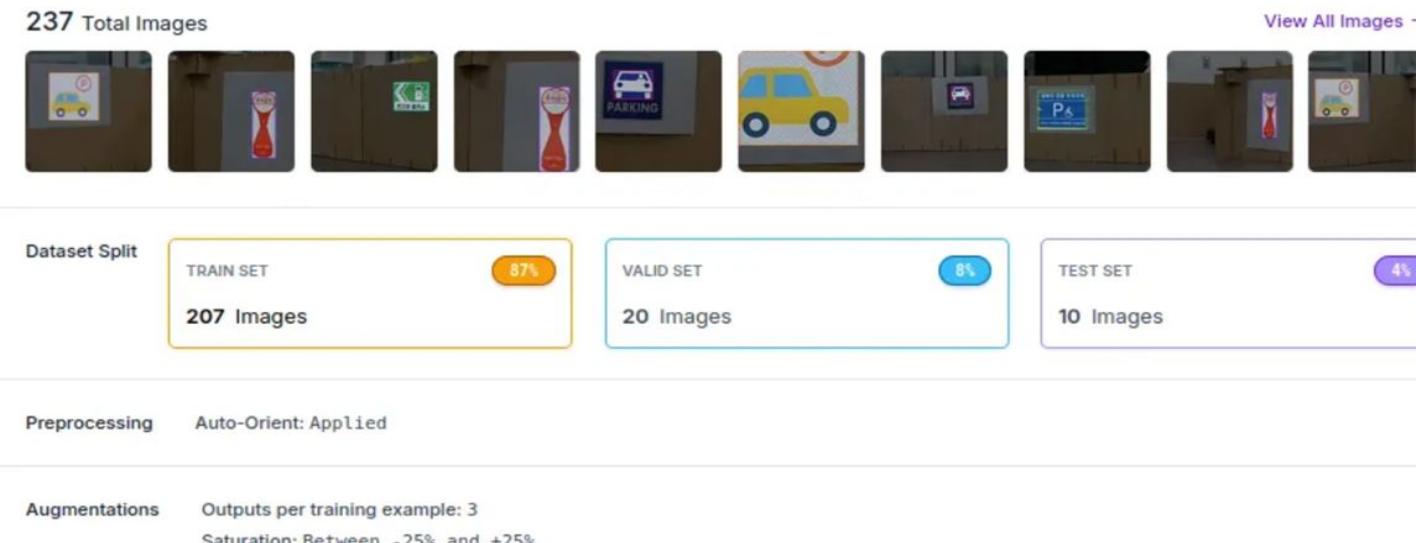
프로젝트 수행 경과

Vision – 주차공간 object detection

▶ 주차(Disabled, Normal, EV)

최종 데이터셋 구성

- model: YOLOv8n
- source_img : 110개(class 5개, park 30장, 이외 20장)
- Train / Valid / Test : 7:2:1
- Augmentation: Saturation (-25% ~ +25%),
training images 3 variants(total_img: 237개)
- epoch = 100
- imgsiz = 320
- batch_size = 16



05

K-Digital Training

프로젝트 수행 결과

- ▶ 탐지 결과(320x320)

Vision – 주차공간 object detection



프로젝트 수행 경과

Vision – 주차공간 object detection

▶ 주차(Disabled, Normal, EV) – 모델 선정 과정

- YOLO 모델 비교 과정

oakd_pro.yaml(params =>FPS:10, Resoultion: 720, img_sz=320)
yolo model(epoch=100, batch=16, img_sz=320)

- 이 측정은 훈련 중 측정값 (YOLO 모델별 학습 중 속도 측정 결과)

모델	Preprocess	Inference	Postprocess	Total (대략합)
YOLOv8n	0.0490	0.4676	0.9155	≈ 1.43 sec
YOLOv11n	0.0503	1.0518	1.2084	≈ 2.31 sec
YOLOv11s	0.0475	0.7127	0.8630	≈ 1.62 sec

모델별 훈련 중 측정값 데이터에서 **YOLOv8n** 이 가장 잘나왔지만 실제검증 필요

→ 학습 과정에서 나온 speed값들은 참고용, 실제 실행 환경의 부하를 판단하는 데는 불충분

05

K-Digital Training

프로젝트 수행 경과

Vision – 주차공간 object detection

- ▶ CPU, 네트워크 부하 – turtlebot4

htop - 실시간 CPU 사용량만을 보여주기 때문에 일관성 있는 비교가 어려움

```
ubuntu@ubuntu: ~ 192x40
[1] 100% Mem[3.70G/8.0K]
[2] 100% Swp[0K/0K]

      PID USER      PR  NI    VIRT    SWAP    RSS   CPU%    SENS%    TIME+  Command
  1141 ubuntu    20   0  4167M  191M 61764 S 26.9  5.1  1:05.83 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x --params-file /t
  1149 ubuntu    20   0  561M 27040 17300 S 19.7  0.7  2:01.71 /opt/ros/humble/lib/turtlebot4_node/turtlebot4_node --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x --params-file /t
  1153 ubuntu    20   0 1068M 30600 16904 S 14.4  0.8  1:37.53 /opt/ros/humble/lib/create3_republisher/create3_republisher --ros-args -r __ns:=/robot3 --params-file /opt/ros/humble/share/cre
  1151 ubuntu    20   0  556M 22644 15456 R 11.2  0.6  1:15.05 /opt/ros/humble/lib/turtlebot4_base/turtlebot4_base_node --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x
  1287 ubuntu    20   0 1467M 191M 61764 S 10.5  5.1  0:14.86 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1288 ubuntu    20   0 1467M 191M 61764 R  5.9  5.1  0:07.19 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1165 ubuntu    20   0  610M 57904 26948 S  3.9  1.5  0:23.35 /usr/bin/python3 /opt/ros/humble/lib/joint_state_publisher/joint_state_publisher --ros-args -r __node:=joint_state_publisher -r
  1251 ubuntu    20   0 1068M 30600 16904 S  3.9  0.8  0:19.42 /opt/ros/humble/lib/create3_republisher/create3_republisher --ros-args -r __ns:=/robot3 --params-file /opt/ros/humble/share/cre
  1277 ubuntu    20   0 1467M 191M 61764 S  3.9  5.1  0:24.18 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1250 ubuntu    20   0 1068M 30600 16904 S  3.3  0.8  0:19.25 /opt/ros/humble/lib/create3_republisher/create3_republisher --ros-args -r __ns:=/robot3 --params-file /opt/ros/humble/share/cre
  1659 ubuntu    20   0  8436  4572  2980 R  3.3  0.1  0:05.03 htop
  1252 ubuntu    20   0 1068M 30600 16904 S  2.6  0.8  0:19.12 /opt/ros/humble/lib/create3_republisher/create3_republisher --ros-args -r __ns:=/robot3 --params-file /opt/ros/humble/share/cre
  1155 ubuntu    20   0  556M 22388 15524 S  2.0  0.6  0:13.78 /opt/ros/humble/lib/joy_linux/joy_linux_node --ros-args -r __node:=joy_linux_node -r __ns:=/robot3 -r /diagnostics:=diagnostics
  1159 ubuntu    20   0  564M 22672 15460 S  2.0  0.6  0:17.81 /opt/ros/humble/lib/rplidar_ros/rplidar_composition --ros-args -r __node:=rplidar_composition -r __ns:=/robot3 --params-file /t
  1175 ubuntu    20   0  564M 22672 15460 S  2.0  0.6  0:10.10 /opt/ros/humble/lib/rplidar_ros/rplidar_composition --ros-args -r __node:=rplidar_composition -r __ns:=/robot3 --params-file /t
  1192 ubuntu    20   0  561M 27040 17300 S  2.0  0.7  0:13.20 /opt/ros/humble/lib/turtlebot4_node/turtlebot4_node --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x --params-file /t
  1194 ubuntu    20   0  561M 27040 17300 S  2.0  0.7  0:10.51 /opt/ros/humble/lib/turtlebot4_node/turtlebot4_node --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x --params-file /t
  1220 ubuntu    20   0 1068M 30600 16904 S  2.0  0.8  0:16.19 /opt/ros/humble/lib/create3_republisher/create3_republisher --ros-args -r __ns:=/robot3 --params-file /opt/ros/humble/share/cre
  1229 ubuntu    20   0 1467M 191M 61764 S  2.0  5.1  0:02.14 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1276 ubuntu    20   0 1467M 191M 61764 S  2.0  5.1  0:11.47 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1209 ubuntu    20   0  556M 22388 15524 S  1.3  0.6  0:10.87 /opt/ros/humble/lib/joy_linux/joy_linux_node --ros-args -r __node:=joy_linux_node -r __ns:=/robot3 -r /diagnostics:=diagnostics
  1228 ubuntu    20   0 1467M 191M 61764 S  1.3  5.1  0:02.19 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1141 ubuntu    20   0 1385M 50600 21504 S  0.7  1.3  0:00.71 /usr/bin/python3 /opt/ros/humble/bin/roslaunch /tmp/turtlebot4.launch.py
  1163 ubuntu    20   0  560M 25636 17464 S  0.7  0.7  0:08.76 /opt/ros/humble/lib/robot_state_publisher/robot_state_publisher --ros-args -r __node:=robot_state_publisher -r __ns:=/robot3 --
  1186 ubuntu    20   0  561M 27040 17300 S  0.7  0.7  0:04.78 /opt/ros/humble/lib/turtlebot4_node/turtlebot4_node --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x --params-file /t
  1210 ubuntu    20   0  556M 22644 15456 S  0.7  0.6  0:04.67 /opt/ros/humble/lib/turtlebot4_base/turtlebot4_base_node --ros-args -r __ns:=/robot3 --params-file /tmp/tmpprvrm9x
  1224 ubuntu    20   0 1467M 191M 61764 S  0.7  5.1  0:02.16 /opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3
  1   root     20   0 163M 11284  7408 S  0.0  0.3  0:02.84 /sbin/init fixrtc splash
  394 root     19   -1 48108 15560 14440 S  0.0  0.4  0:00.86 /lib/systemd/systemd-journald
  428 root     RT   0  282M 25676  7400 S  0.0  0.7  0:00.20 /sbin/multipathd -d -s
```

05

프로젝트 수행 경과

Vision – 주차공간 object detection

▶ CPU, 네트워크 부하 – turtlebot4

측정 결과는 CSV 로그 저장, 성능을 객관적이고 정량적으로 비교 가능. (compressed_img 사용)

```
ubuntu@ubuntu:~$ python3 raspi_monitor.py
▶ 타겟 프로세스 찾음: PID=1156, CMD='/opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3'
ubuntu@ubuntu:~ 88x33
```

1

```
ubuntu@ubuntu:~$ python3 raspi_monitor.py
▶ 타겟 프로세스 찾음: PID=1156, CMD='/opt/ros/humble/lib/rclcpp_components/component_container --ros-args -r __node:=oakd_container -r __ns:=/robot3'
ubuntu@ubuntu:~ 88x33
[16:02:21] CPU: 27.9% | PID_CPU: 22.90% | MEM: 17.3% | TX: 53.75Mb | RX: 0.22Mb
[16:02:22] CPU: 26.0% | PID_CPU: 21.90% | MEM: 17.3% | TX: 53.23Mb | RX: 0.22Mb
[16:02:23] CPU: 28.5% | PID_CPU: 22.90% | MEM: 17.3% | TX: 53.96Mb | RX: 0.19Mb
[16:02:24] CPU: 28.2% | PID_CPU: 22.90% | MEM: 17.3% | TX: 54.06Mb | RX: 0.21Mb
^C중단 요청됨. 종료합니다.

종료 - 평균값 기록됨: CPU 27.01%, PID_CPU 23.06%, MEM 17.37%, TX 51.02Mb, RX 0.20Mb
ubuntu@ubuntu:~$ cat raspi_resource_log.csv
timestamp,CPU%,PID_CPU%,Mem%,TX_Mb,RX_Mb
16:02:04,24.9,11.0,17.4,12.53,0.12
16:02:05,27.0,16.0,17.4,22.57,0.18
16:02:06,24.0,12.0,17.4,17.74,0.15
16:02:07,28.0,32.9,17.4,38.21,0.13
16:02:08,33.0,35.9,17.3,99.80,0.31
16:02:09,29.8,27.9,17.4,69.97,0.22
16:02:10,26.3,23.9,17.4,55.14,0.21
16:02:11,28.1,24.9,17.4,53.35,0.20
16:02:12,26.4,22.9,17.4,55.05,0.21
16:02:13,25.8,22.9,17.4,52.95,0.23
16:02:14,25.3,22.9,17.4,54.31,0.20
16:02:15,25.7,22.9,17.4,53.84,0.20
16:02:16,25.6,22.9,17.4,54.21,0.23
16:02:17,26.2,23.9,17.4,54.31,0.20
16:02:18,27.4,23.9,17.3,53.19,0.20
16:02:19,27.4,23.9,17.3,54.39,0.23
16:02:20,25.8,22.9,17.4,54.84,0.19
16:02:21,27.9,22.9,17.3,53.75,0.22
16:02:22,26.0,21.9,17.3,53.23,0.22
16:02:23,28.5,22.9,17.3,53.96,0.19
16:02:24,28.2,22.9,17.3,54.06,0.21

AVERAGE,27.01,23.06,17.37,51.02,0.20
ubuntu@ubuntu:~$
```

프로젝트 수행 경과

Vision – 주차공간 object detection

▶ 주차(Disabled, Normal, EV) – 경량화 과정

- 실제 모델별 부하 측정

`oakd_pro.yaml(params => FPS:10, i_resolution: 720, i_size=640, i_enable_compression = False)`

모델	YOLO 프로세스 CPU (%)	TX 트래픽 (Mb)	RX 트래픽 (Mb)	Inference time	리소스 효율성 상대 평가
YOLOv8n	27.3	76.8	0.15	11.21ms	효율적
YOLOv11n	29.3	82.5	0.09	11.82ms	중간
YOLOv11s	27.4	77.4	0.17	12.04ms	무거움

프로젝트 수행 경과

Vision – 주차공간 object detection

▶ 주차(Disabled, Normal, EV) – 경량화 과정

- 실제 모델별 부하 측정

`oakd_pro.yaml(params => FPS:10, i_resoultion: 720, i_size=320, i_enable_compression = False)`

모델	전체 CPU 사용률 (%)	YOLO 프로세스 CPU (%)	메모리 사용률 (%)	TX 트래픽 (Mb)	RX 트래픽 (Mb)	Inference time	리소스 효율성 평가
YOLOv8n	21.91	26.90	17.59	74.96	0.07	10.26ms	효율적
YOLOv11n	22.87	26.94	17.70	76.65	0.06	10.44ms	중간
YOLOv11s	22.52	28.80	17.30	74.80	0.06	10.74ms	무거움

▣ 정리하면

- 모델이 성능에 큰 영향을 주지 않음 확인
- 즉, 지금 실험으로부터 모델 외부 요소에서 발생하는 병목을 해결해야함

그 결과:

💡 시스템 병목에 대한 경량화 절차는

모델이 아닌 FPS 제한, ROS 구조, 큐 처리, 쓰레드 설계, 이미지 전송 방식 등이 있음을 확인하였으며 이미지 전송 방식을 통해 시스템 병목현상을 완화하였음.

프로젝트 수행 경과

Vision – 주차공간 object detection

▶ 주차(Disabled, Normal, EV) – 경량화 과정

- 압축 이미지 전송

`oakd_pro.yaml(params => FPS:10, i_resoultion: 720, i_size=320, i_enable_compression = True)`

▶ Compressed image 사용 데이터 요약 (Model: YOLOv8n) + Depth

측정 항목	Turtlebot4 (Compressed)	Turtlebot4 (Raw)	User PC (Compressed)	User PC (Raw)
CPU%	24.80	20.89	2.78	2.19
PID CPU(YOLO)%	23.96	26.28	-	-
Mem%	17.16	17.25	27.79	27.55
TX (Mb)	50.04	71.87	0.08	0.14
RX (Mb)	0.05	0.10	32.44	45.60
FPS	-	-	9.99	9.98
추론 시간(ms)	-	-	8.39	7.86

프로젝트 수행 경과

Vision – 주차공간 object detection

▶ 주차(Disabled, Normal, EV) – 경량화 과정

- 압축 이미지 사용 효과 분석

1. CPU 사용률:

- Turtlebot4의 전체 CPU는 조금 상승($20.89 \rightarrow 24.80\%$)
- YOLO 노드 PID 기준 CPU 사용률은 오히려 감소($26.28 \rightarrow 23.96\%$)
👉 즉, 시스템 전체는 약간 더 일하지만, YOLO 처리 프로세스는 경량화됨

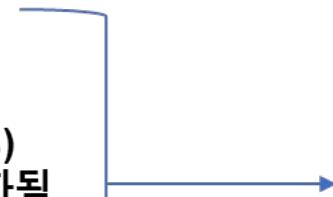
2. 네트워크 (TX, RX):

- TX (Turtlebot4 전송):
=> Raw: 71.87Mb → Compressed: 50.04Mb (약 30% 절감)
- RX (User PC 수신):
=> Raw: 45.60Mb → Compressed: 32.44Mb (약 28% 절감)

👉 네트워크 사용량이 눈에 띄게 줄었음. 특히 송신(TX) 최적화에 큰 효과

3. 추론 시간:

- 약간 증가 ($7.86\text{ms} \rightarrow 8.39\text{ms}$, $+0.5\text{ms}$)
👉 이는 압축 해제(decode)에 따른 비용으로 판단되며, 큰 차이는 아님.



요약:

- 압축 이미지 사용의 이점:
- PID 기준 연산 부하도 감소
 - 네트워크 전송량 25~30% 감소
 - 성능 손실 없이 추론 속도 거의 동일

05

프로젝트 수행 경과

Vision – 번호판 object detection

- ▶ Vision – 차량 번호판 object detection

electric ->



normal ->



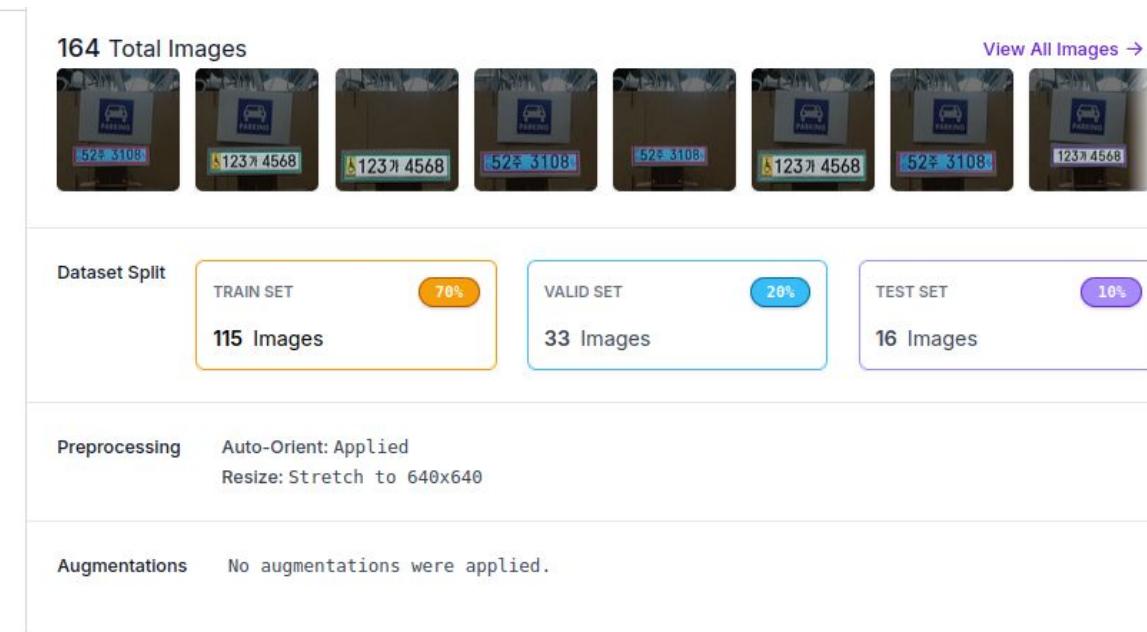
plate(disabled) ->



▶ Vision – 차량 번호판 object detection

최종 데이터셋 구성

- model: YOLOv8n
- source_img : 164개(class 4개)
- Train / Valid / Test : 7:2:1
- Augmentation: X(데이터셋 충분하다 판단)
- epoch = 50
- imgsz = 320
- batch_size = 16



05

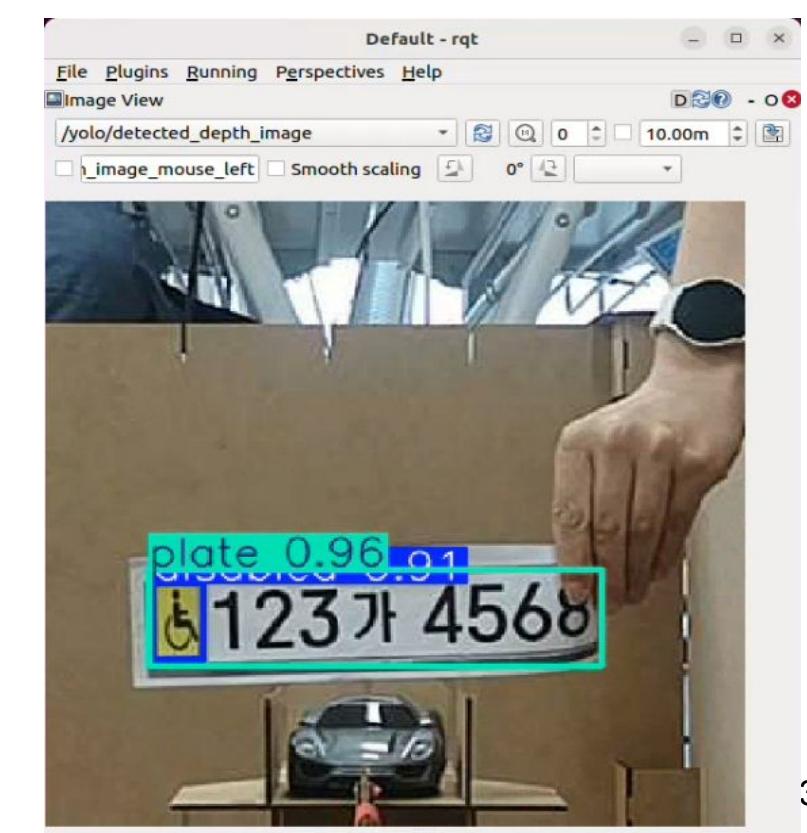
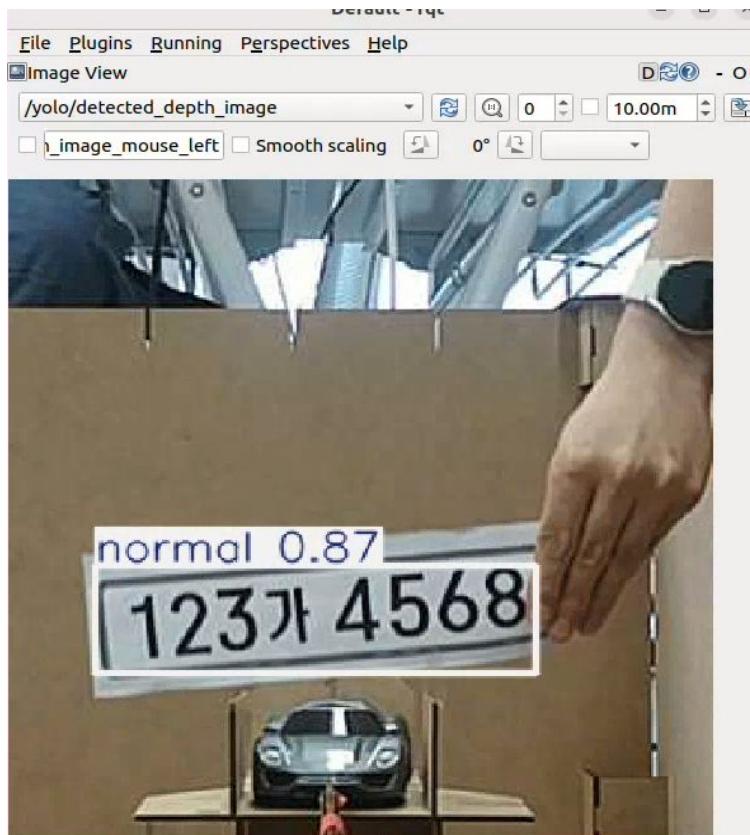
K-Digital Training

프로젝트 수행 경과

Vision – 번호판 object detection

▶ Vision – 차량 번호판 object detection

raw image => 탐지 결과(640x640)



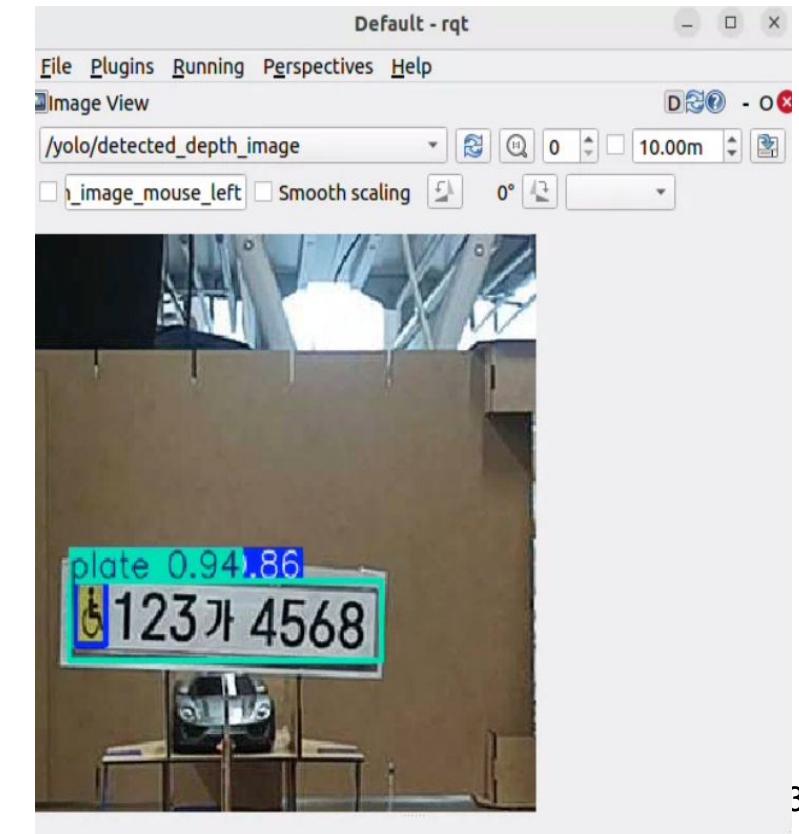
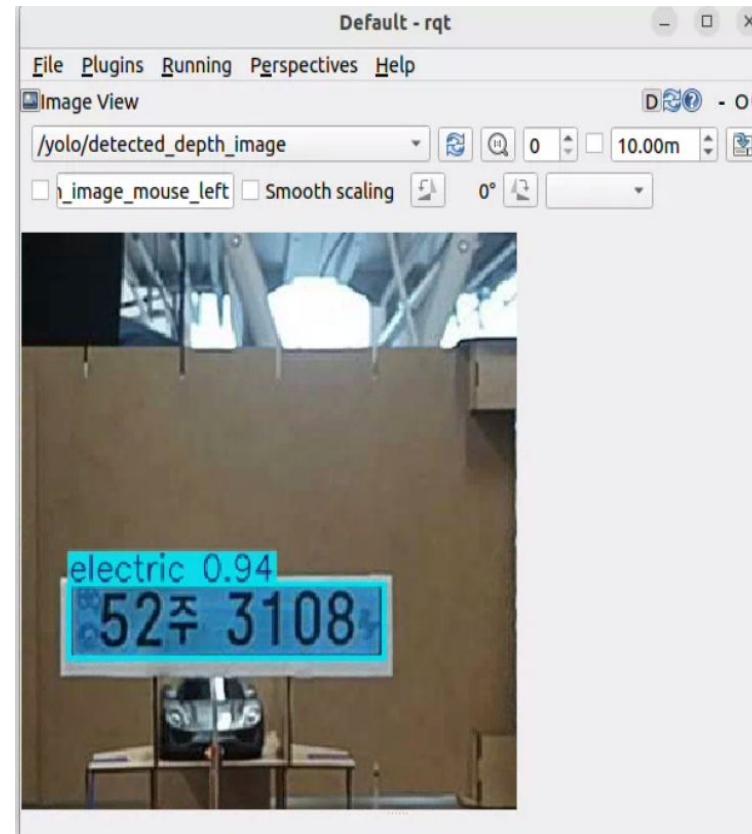
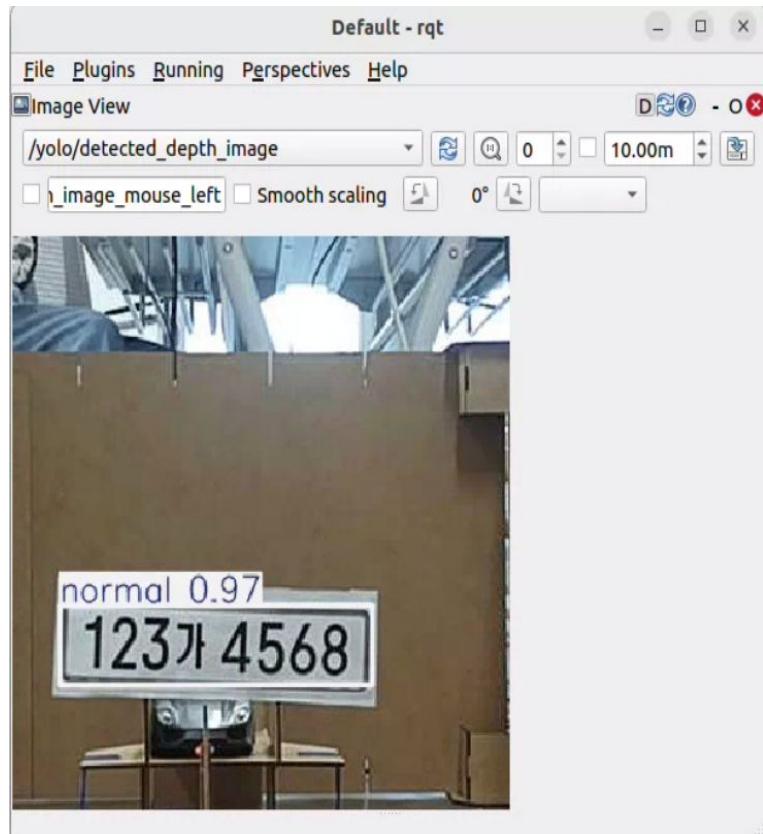
05

프로젝트 수행 경과

Vision – 번호판 object detection

▶ Vision – 차량 번호판 object detection

raw image => 탐지 결과(320x320)



프로젝트 수행 경과

Vision – 번호판 object detection

▶ Vision – 차량 번호판 object detection

raw image v.s. compressed image

- compressed image 전송: 주차 공간 경량화와 동일한 스펙 이용
- 기준 번호판은 normal / 각 파라미터 측정 시간: 2min으로 통일

측정 항목	Turtlebot4(Raw)	Turtlebot4(Compressed)	User PC(Raw)	User PC(Compressed)
CPU(%)	17.84	16.36	2.24	1.67
PID CPU(YOLO)(%)	17.02	15.68	-	-
Memory(%)	17.02	17.09	19.5	22.53
TX(Mb)	28.28	1.82	0.04	0.03
RX(Mb)	0.03	0.01	17.8	1.33
FPS	-	-	9.99	9.99
추론 시간(ms)	-	-	9.16	8.02

▶ Vision – 차량 번호판 object detection

compressed image 사용 효과 분석

1. CPU 사용률:

- Turtlebot4 전체 CPU 사용률 감소(17.84 >> 16.36)
- YOLO 노드 PID 기준 CPU 사용률 감소(17.02 >> 15.68)
-> YOLO 처리 프로세스 경량화 효과 확인

2. 네트워크 (TX, RX)

- TX(Turtlebot4 전송)
=> Raw: 28.3Mb -> Compressed: 1.82Mb (약 94% 절감)
- RX(User PC 수신)
=> Raw: 0.03Mb -> Compressed: 0.01Mb (약 69% 절감)

3. 추론 시간:

- 약간 감소(9.16ms -> 8.02ms / 1.14ms 감소)
-> 네트워크 사용량의 유의미한 감소로 인한 부가 효과로 예상됨

요약:

1. PID 기준 연산 부하 감소
2. 네트워크 전송량 ~ 94% 감소
3. 전반적으로 성능 향상 확인

▶ 차량 정보 확인(detect_car_info)

번호판을 감지하면 2가지 기능을 수행

1. YOLO모델을 기반으로 차종 분류, 어느 구역에 주차할지 결정
2. 번호판 영역을 OCR해서 감지한 차량 번호(ex.22가2222) 추출



프로젝트 수행 경과

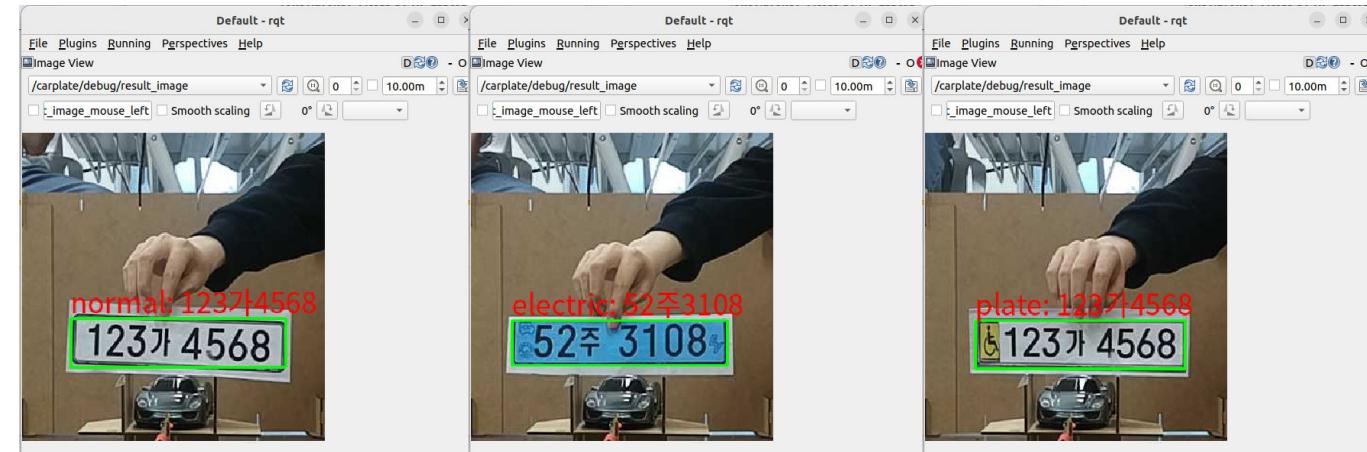
Vision – OCR

▶ 차량 번호판 OCR (EasyOCR)



- PyTorch 기반의 오픈소스 OCR 라이브러리
- CRAFT(텍스트 감지) + CRNN(텍스트 인식) 모델 구조
- 80개 이상 언어를 지원하며, 한글 인식도 가능
- 간단한 코드로 이미지에서 텍스트 추출 가능
(파인 티닝 불가)

<OCR 결과>



프로젝트 수행 경과

Vision – OCR

▶ 차량 번호판 OCR (EasyOCR)



테스트용 621 × 171 3채널 이미지

```
import easyocr
import time

# EasyOCR 리더 (GPU)
reader_gpu = easyocr.Reader(['ko'], gpu=True)

gpu_times = []

for i in range(10):
    start = time.time()
    result = reader_gpu.readtext(binary, detail=0)
    elapsed = time.time() - start
    gpu_times.append(elapsed)
    print(f"[GPU] [{i+1}] 결과: {result} | 시간: {elapsed:.3f}초")

print(f"GPU 평균 시간: {(sum(gpu_times)/len(gpu_times)):.3f}초")
```

Python

```
[GPU][1] 결과: ['123가 4568'] | 시간: 0.090초
[GPU][2] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][3] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][4] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][5] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][6] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][7] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][8] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][9] 결과: ['123가 4568'] | 시간: 0.085초
[GPU][10] 결과: ['123가 4568'] | 시간: 0.091초
GPU 평균 시간: 0.086초
```

추론 결과 비교

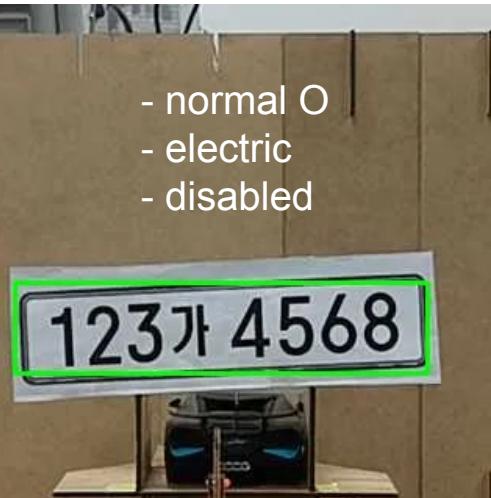
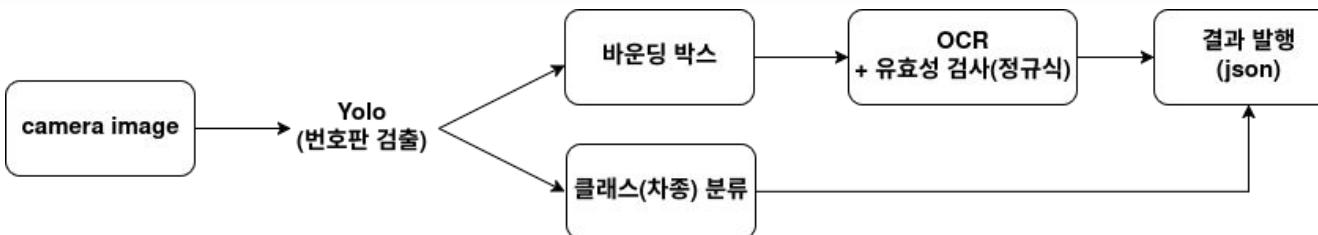
CPU/GPU	이진화 여부	추론 시간	OCR 결과
CPU	O	0.199s	123가 4567
	X	0.218s	1237 4567
GPU	O	0.086s	123가 4567
	X	0.097s	1237 4567

05

프로젝트 수행 경과

Vision – 번호판 & OCR

▶ 차량 정보 확인(detect_car_info)



1. 객체 검출, 차종
분류



2. 이미지 crop 및
이진화

123가 4567
1237 4567
123가 4567
123가 4567
123가 4567#
⋮

3. OCR

123가 4567
~~1237 4567~~
123가 4567
123가 4567
~~123가 4567#~~

\d{2,3}[가-힣]\d{4}

4. 유효성 검사

```
colcon build [3/3 done] [0 ongoing]
colcon build [3/3 done] [0 ongoing] 47x11

0: 640x640 1 normal, 3.6ms
[INFO] [1751533682.352524181] [carplate_ocr_node]: OCR 결과 (normal): 123가|4568
Speed: 3.9ms preprocess, 3.6ms inference, 0.7ms
postprocess per image at shape (1, 3, 640, 640)
)
[INFO] [1751533682.354098628] [carplate_ocr_node]: OCR JSON 결과: {"type": "normal", "car_plate": "123가|4568"}
```

5. 결과

▶ 차량 정보 확인(detect_car_info)

검출 결과 확인

```
"123가4568"

0: 640x640 1 disabled, 1 plate, 110.0ms
[INFO] [1751270394.157326934] [carplate_ocr_node]: OCR 결과 (plate): 123가4568
[INFO] [1751270394.161197988] [carplate_ocr_node]: 장애인 스티커 검출됨
Speed: 3.7ms preprocess, 110.0ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)
[INFO] [1751270394.163927194] [carplate_ocr_node]: OCR JSON 결과: {"type": "disabled", "car_plate": "123가4568"}  
[]  
田 /bin/bash 99x11  
data: '['{"type": "disabled", "car_plate": "123가4568"}'  
---  
]
```

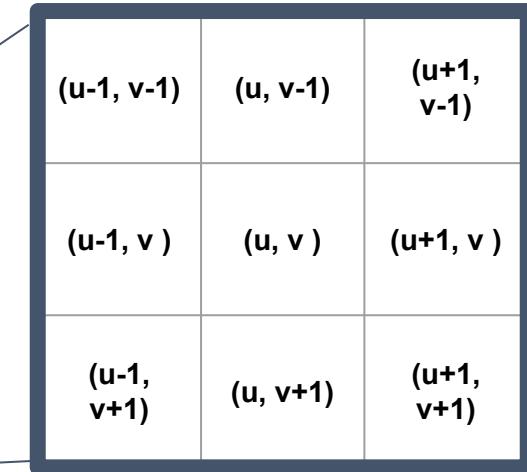
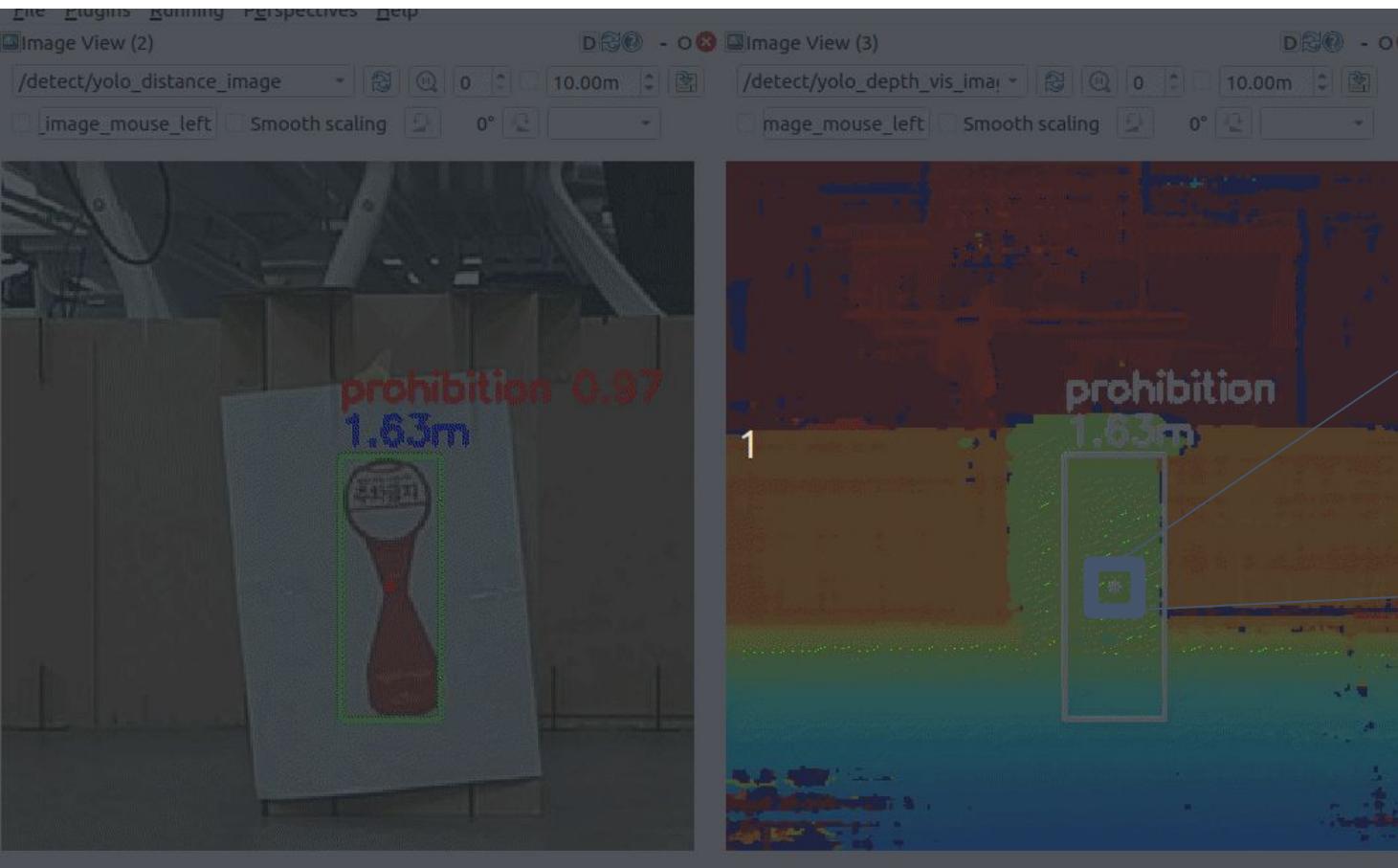
{
 "**type**": "**disabled**", → 차종
 "**car_plate**": "**123가4568**" → 차량번호
}

05

프로젝트 수행 경과

Slam, Navi –Depth, TF

▶ Depth 측정



객체의 중심 픽셀을 포함한 주변 3×3 영역의
깊이 값들 중 유효한 값들의 **중앙값**을
사용하여
해당 객체까지의 거리를 안정적으로 추정 46

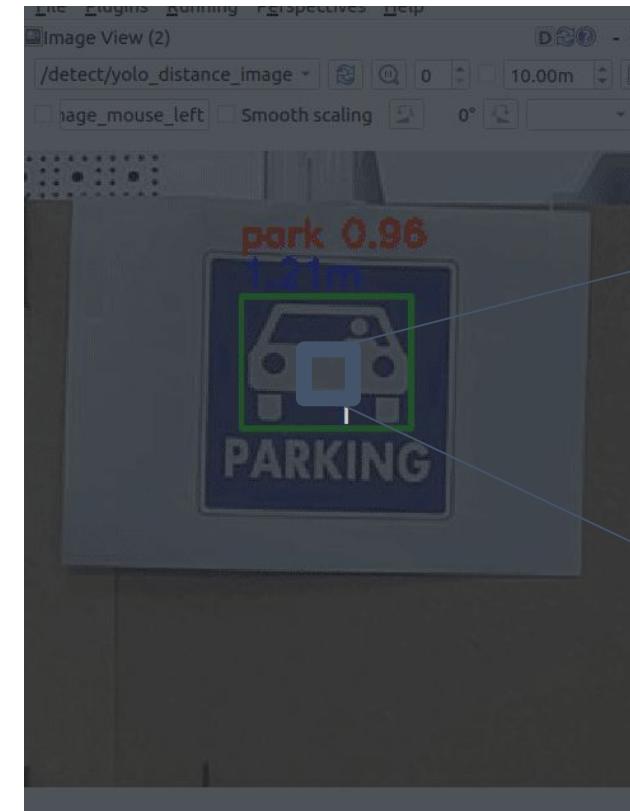
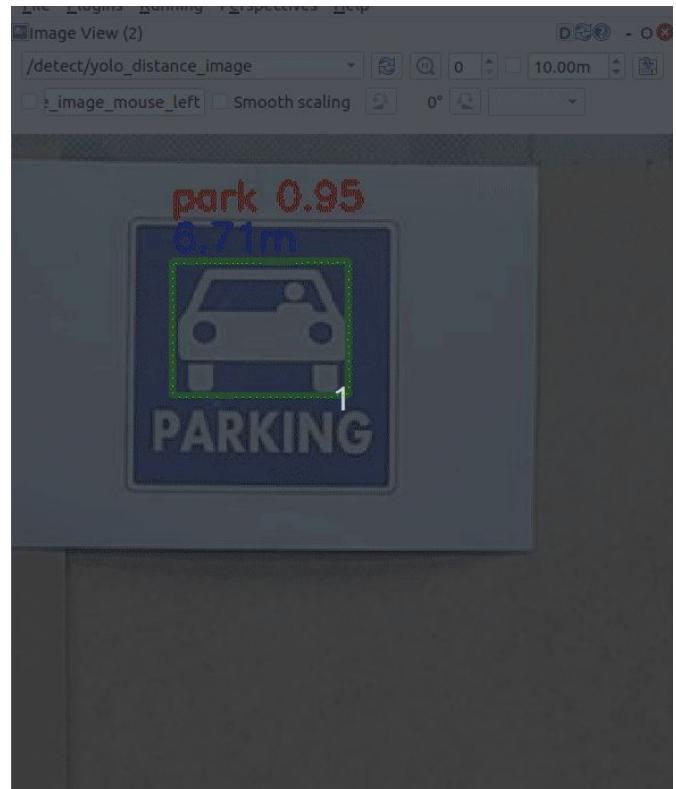
05

K-Digital Training

프로젝트 수행 경과

Slam, Navi –Depth, TF

▶ Depth 측정



(u-1, v-1)	(u, v-1)	(u+1, v-1)
1.26m	1.05m	1.10m
(u-1, v)	(u, v)	(u+1, v)
1.19m	2.50m	1.23m
(u-1, v+1)	(u, v+1)	(u+1, v+1)
0.0m	1.30m	1.15m

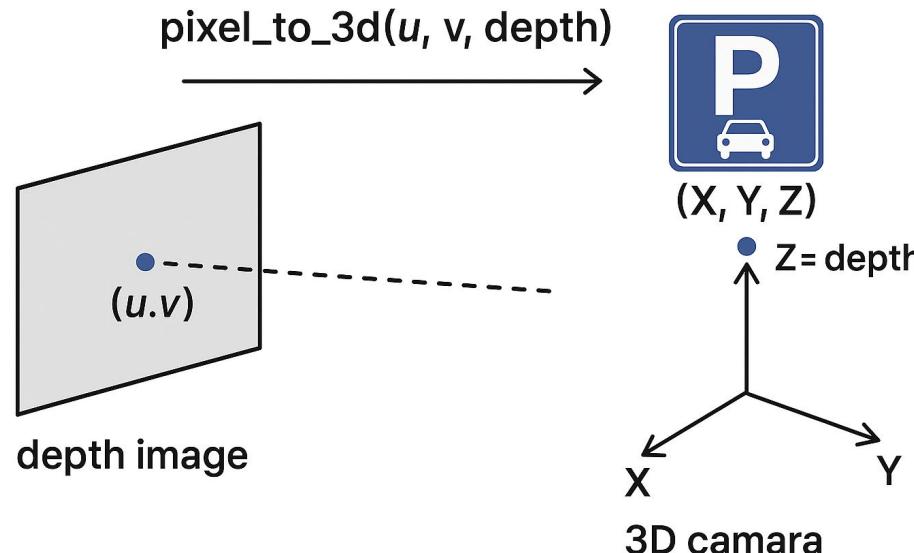
PARKING 사인 depth측정 개선

05

프로젝트 수행 경과

Slam, Navi –Depth, TF

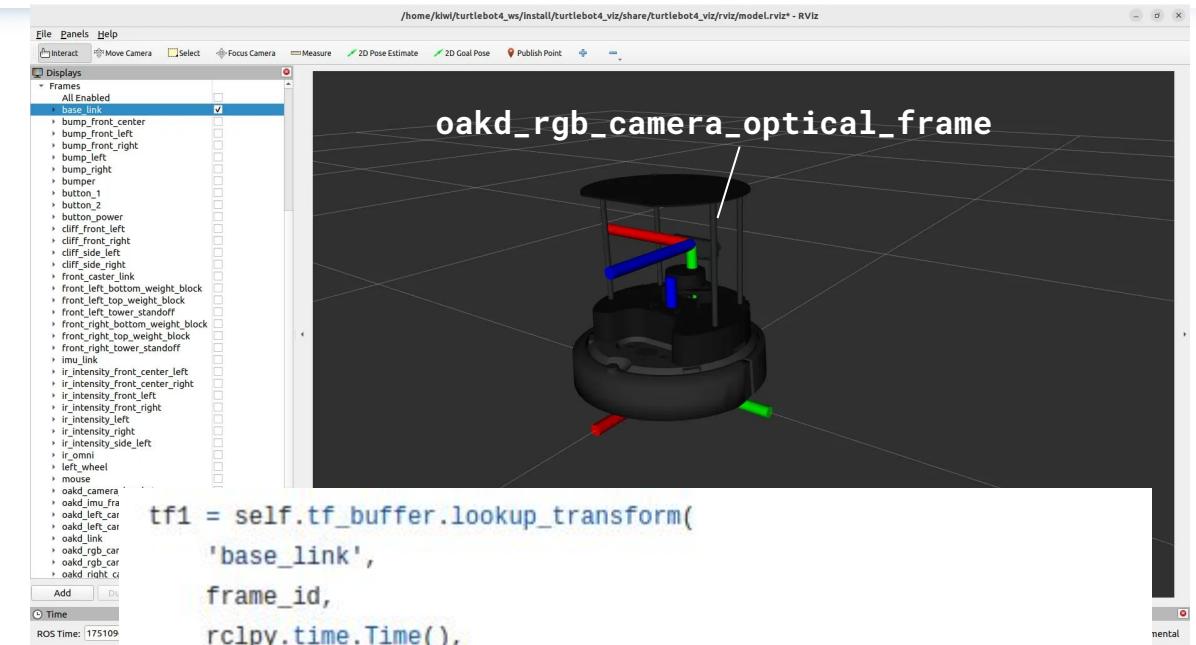
▶ 객체 검출 및 TF변환



$$\bullet \quad x = (u - cx) \times \frac{\text{depth}}{fx}$$

$$\bullet \quad y = (v - cy) \times \frac{\text{depth}}{fy}$$

$$\bullet \quad z = \text{depth}$$



검출된 객체 위치는 `oakd_rgb_camera_optical_frame`에서 시작해 목표 좌표계로 변환

`lookup_transform`으로 TF를 조회하고 `do_transform_point`로 좌표를 실제로 변환

05

K-Digital Training

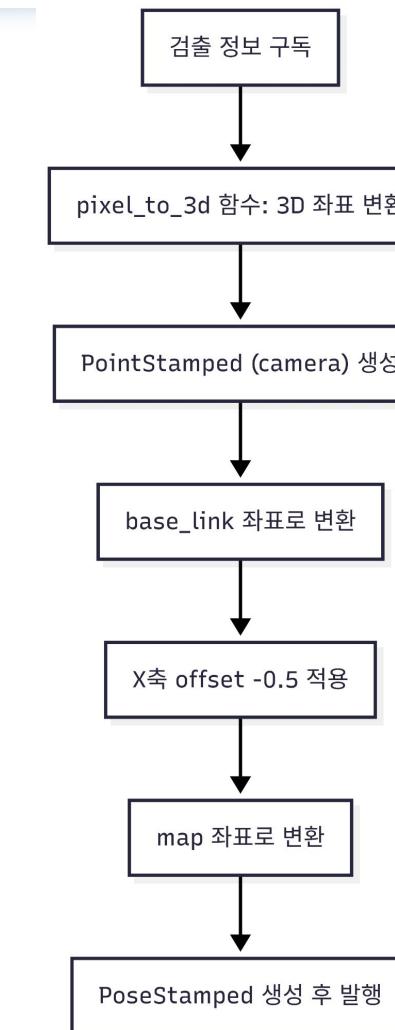
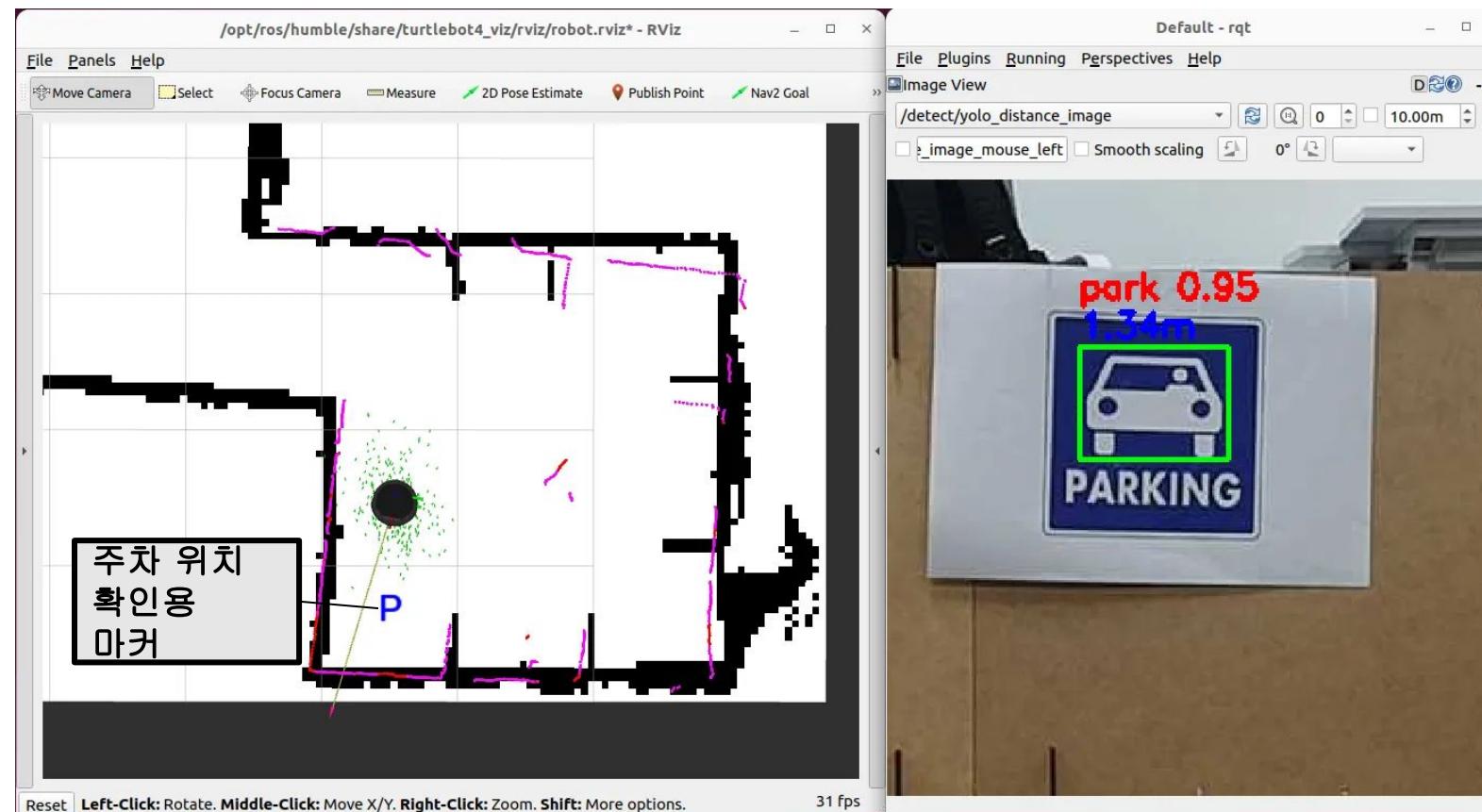
프로젝트 수행 경과

Slam, Navi –Depth, TF

▶ 객체 검출 및 TF변환

`detect_ps_front.py`

검출된 PARKING 사인으로부터 주차할 위치 계산



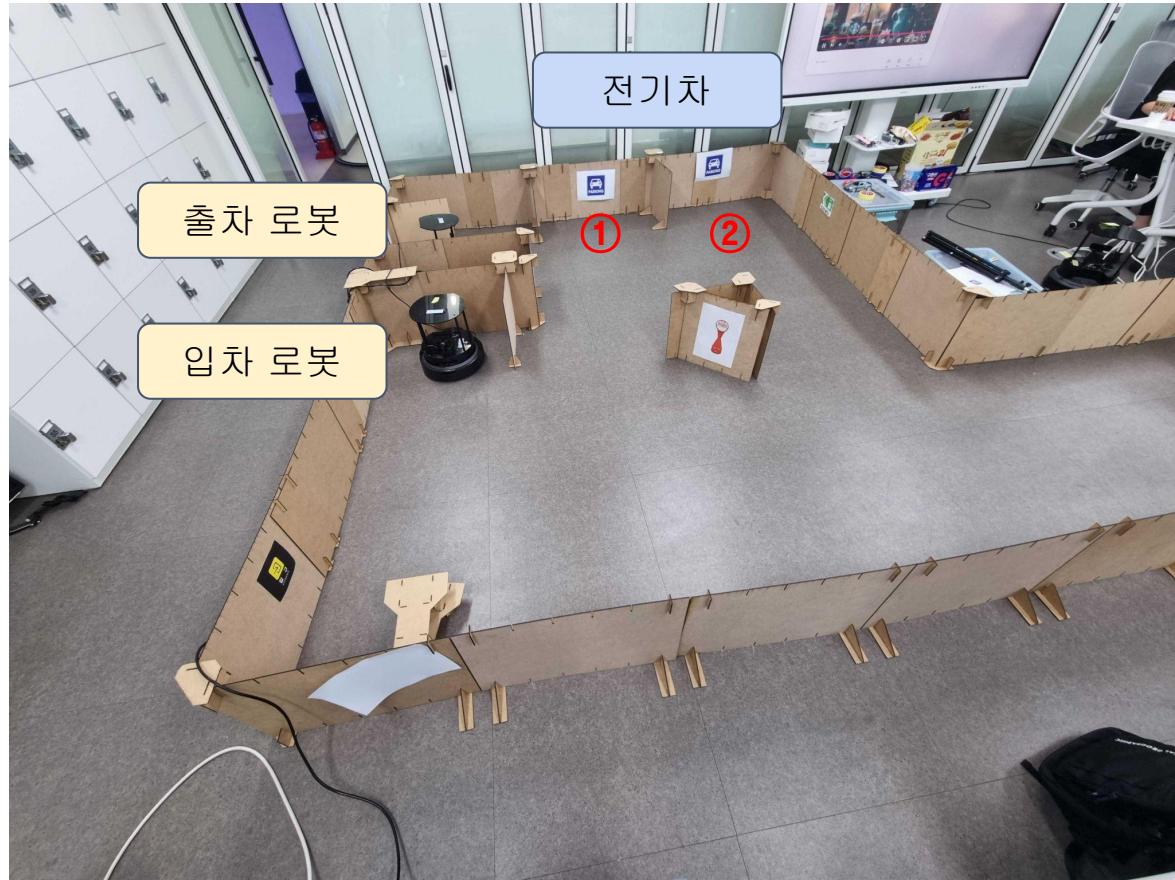
05

K-Digital Training

프로젝트 수행 경과

Slam, Navi

▶ 맵

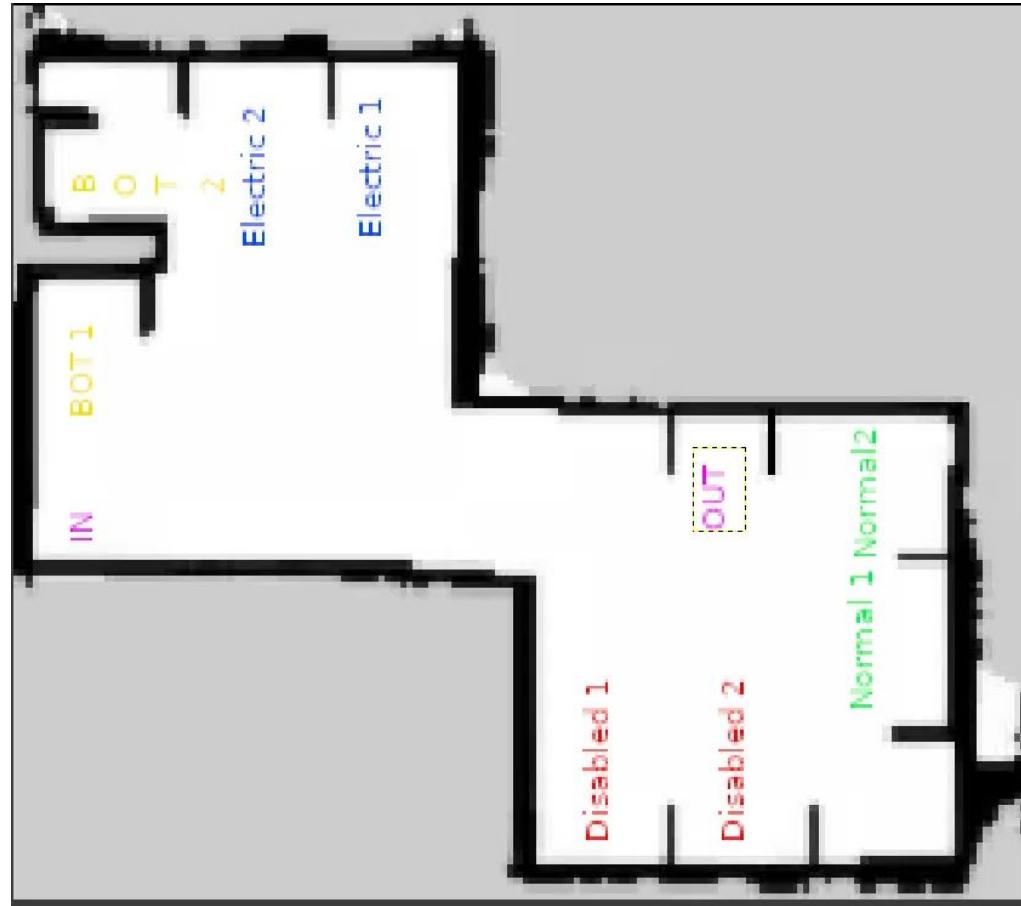


05

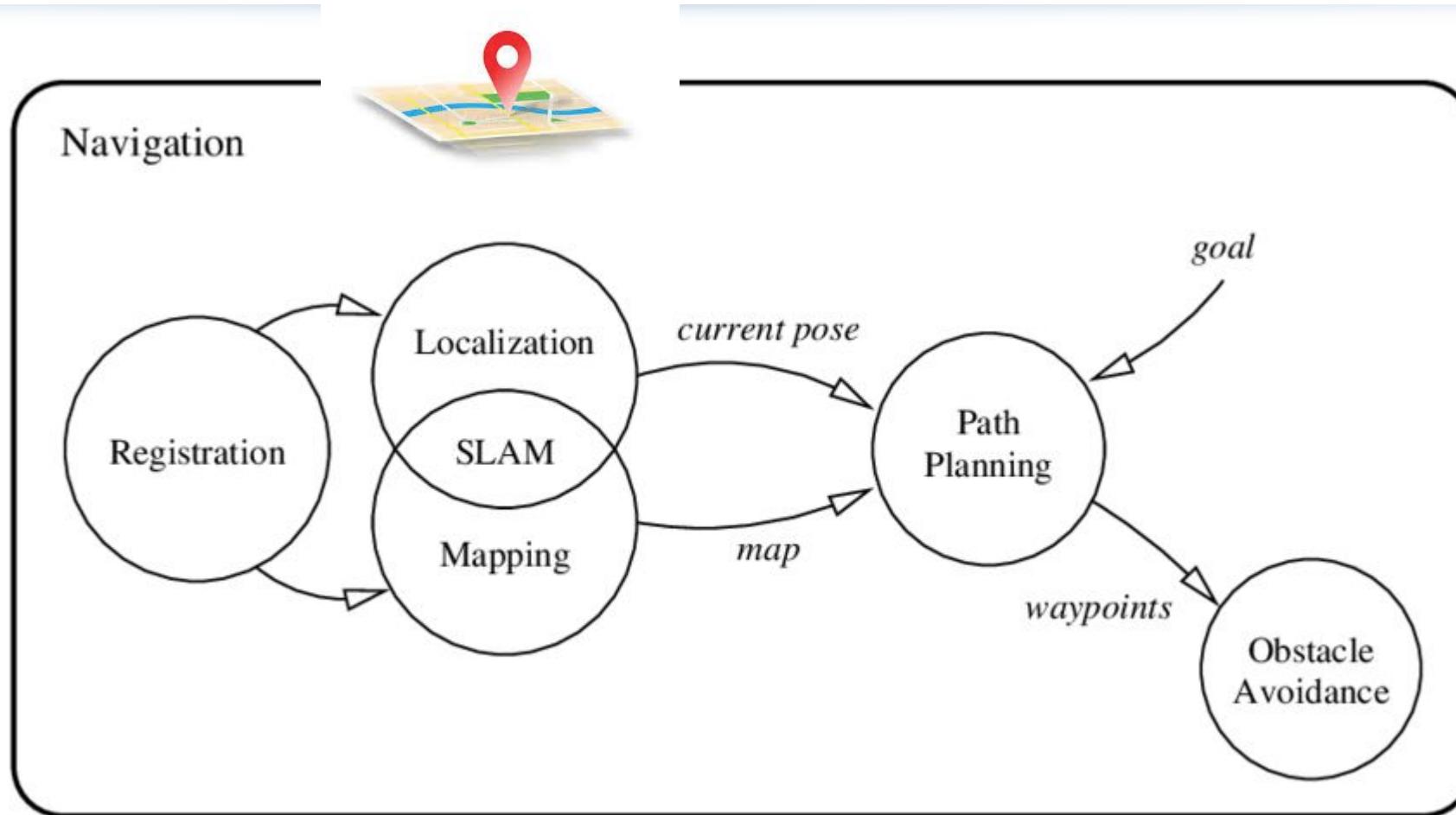
K-Digital Training

프로젝트 수행 경과

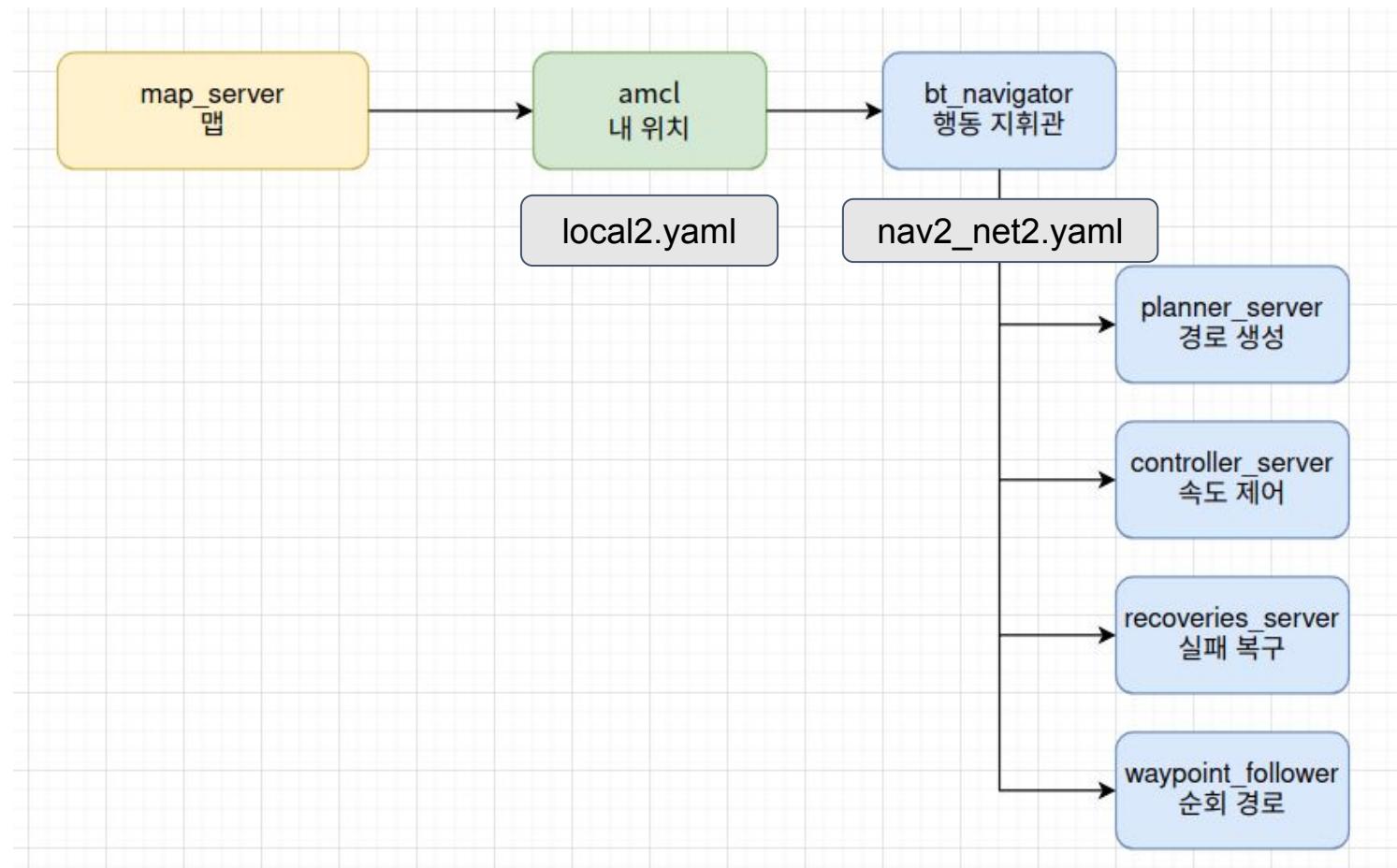
Slam, Navi



▶ Slam & navigation



▶ Slam & navigation

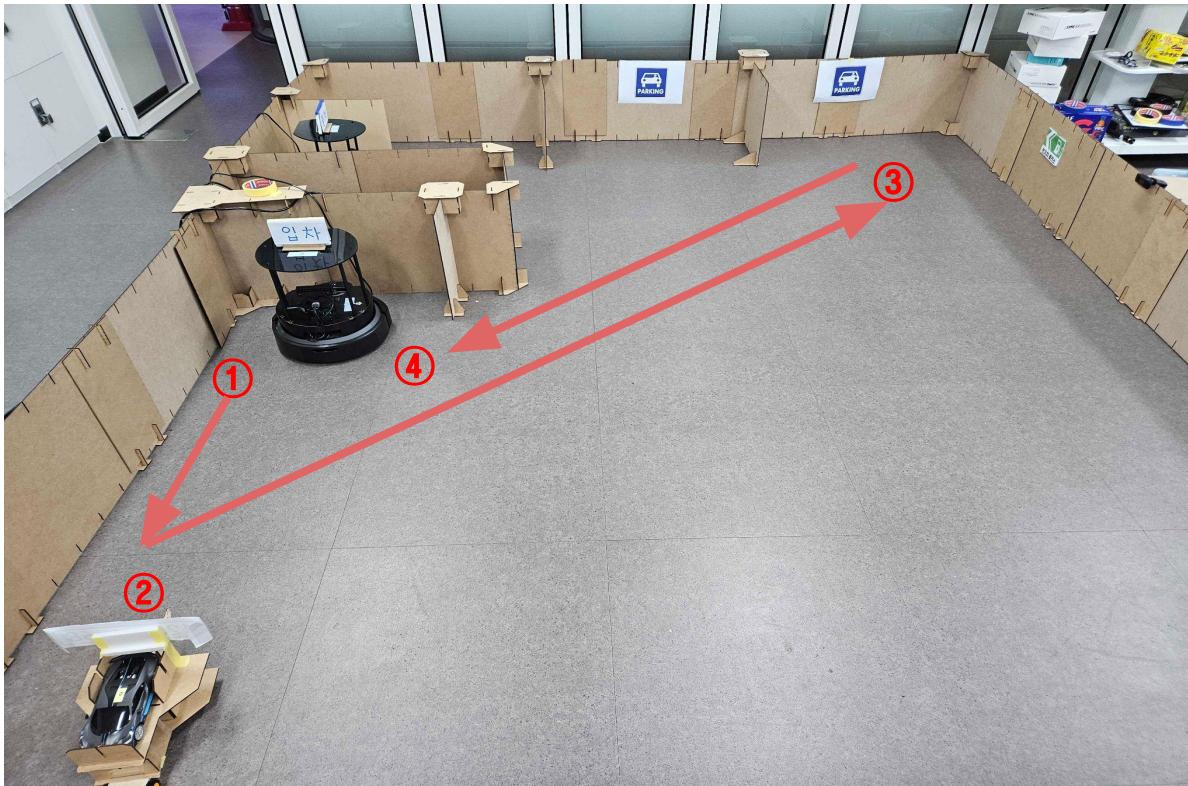


05

프로젝트 수행 경과

▶ 시나리오

입차하기



출차하기



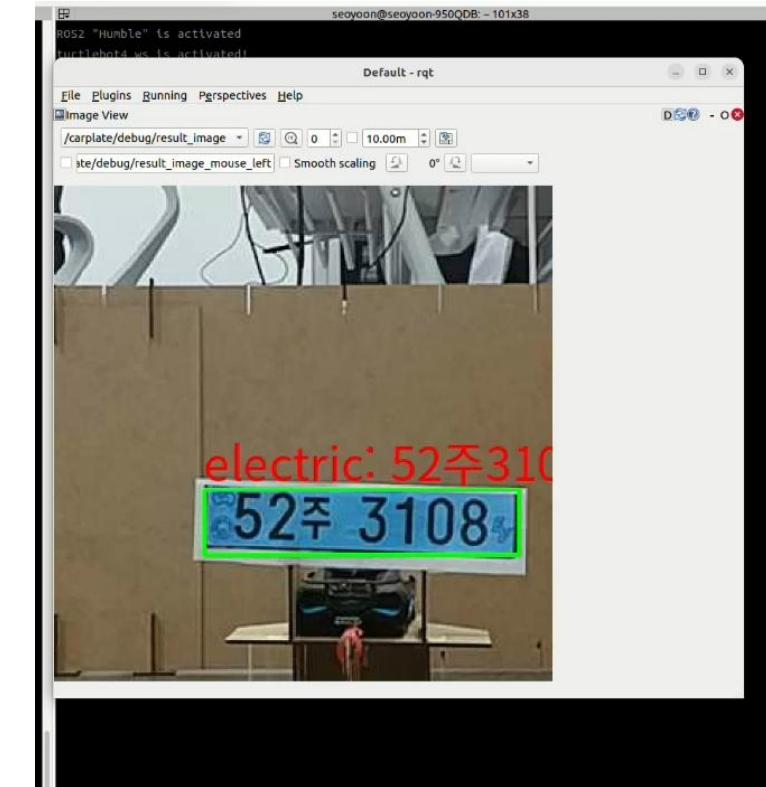
프로젝트 수행 경과

▶ 입차하기

1. nav 초기위치에서 undock



2. yolo 번호판 차종인식 +ocr



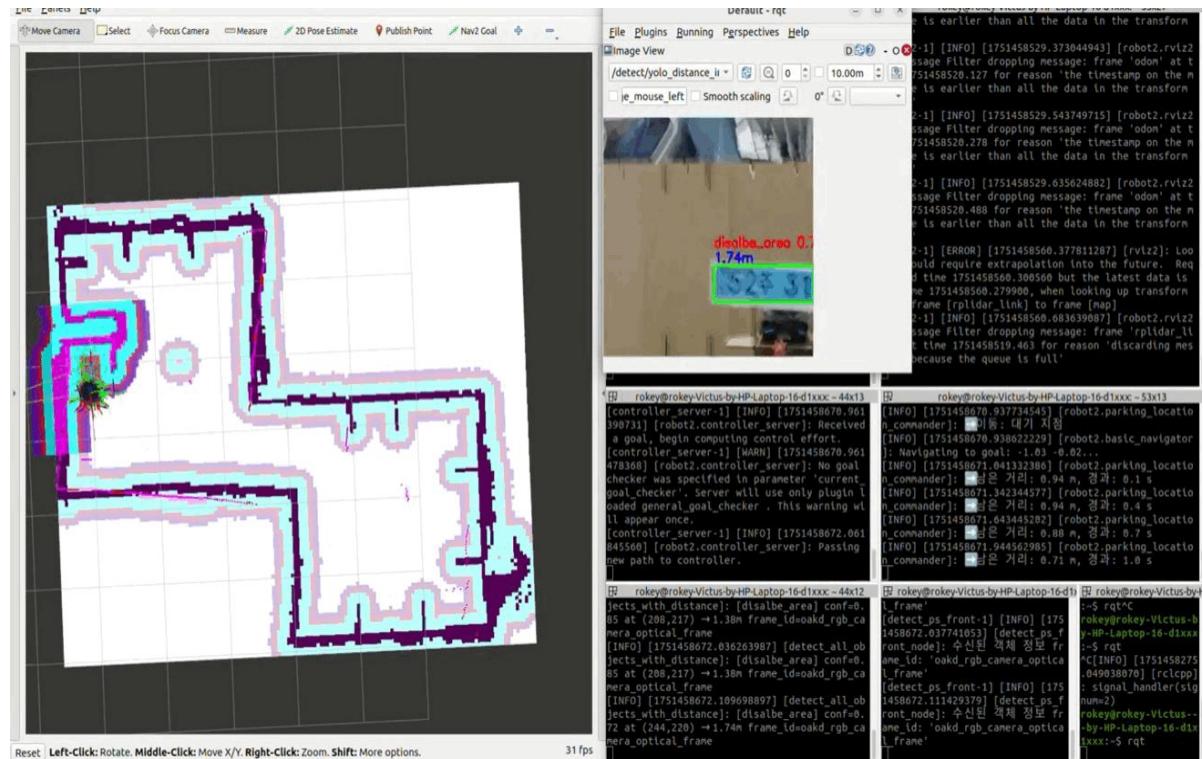
05 K-Digital Training 프로젝트 수행 경과

▶ 입차하기

3. gui 주차하기 클릭



4. 대기위치로 이동

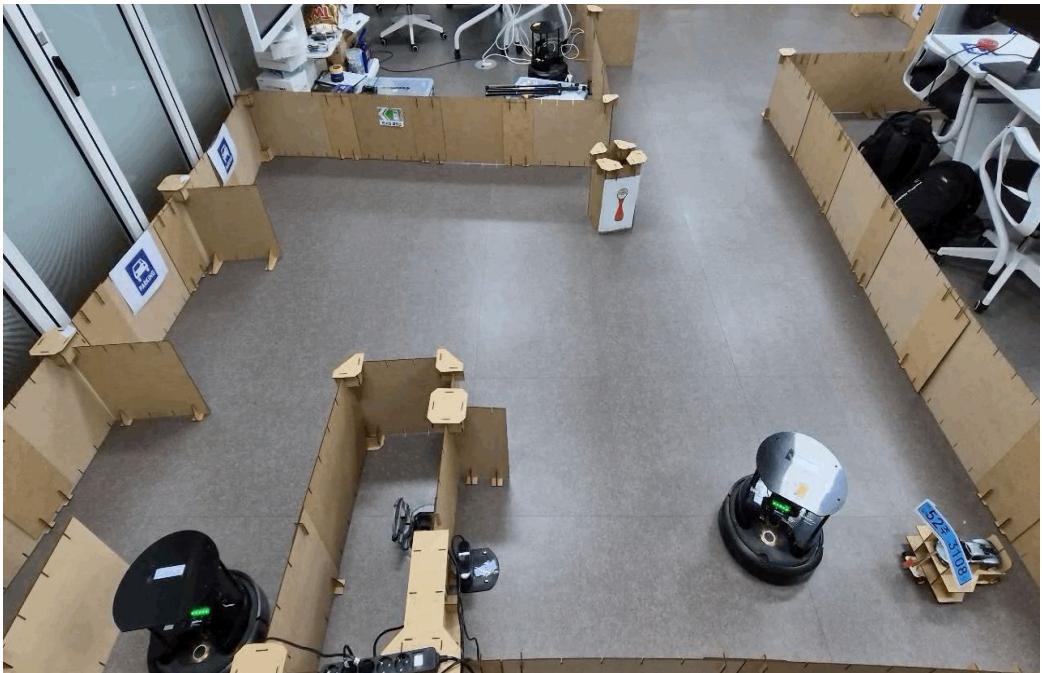


05

프로젝트 수행 경과

▶ 입차하기

5. 차 수레 걸어주기



6. 주차공간 이동



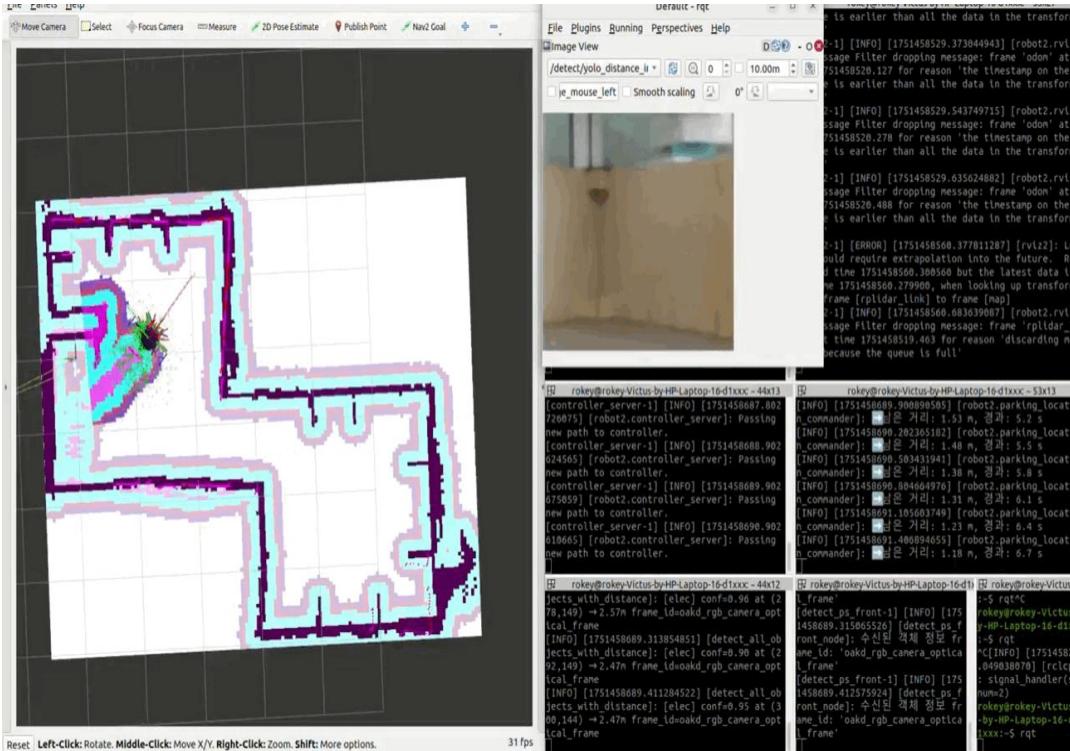
05

K-Digital Training

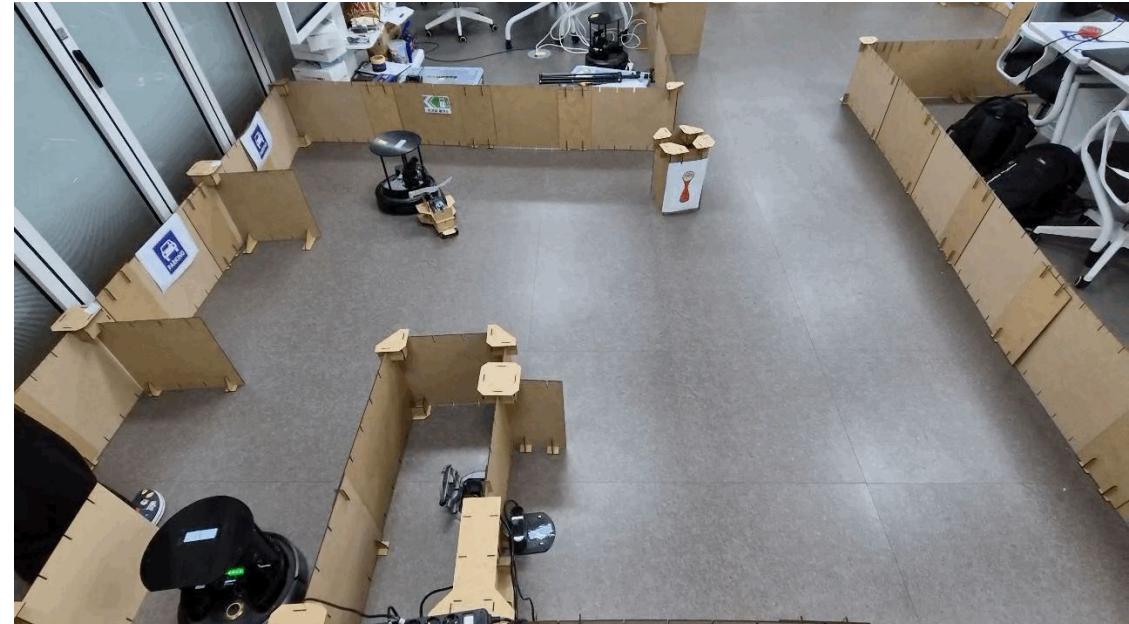
프로젝트 수행 경과

▶ 입차하기

7. depth, tf 발행



8. 앞으로 전진 (baselink 좌표 기준 tf으로부터 0.5m)



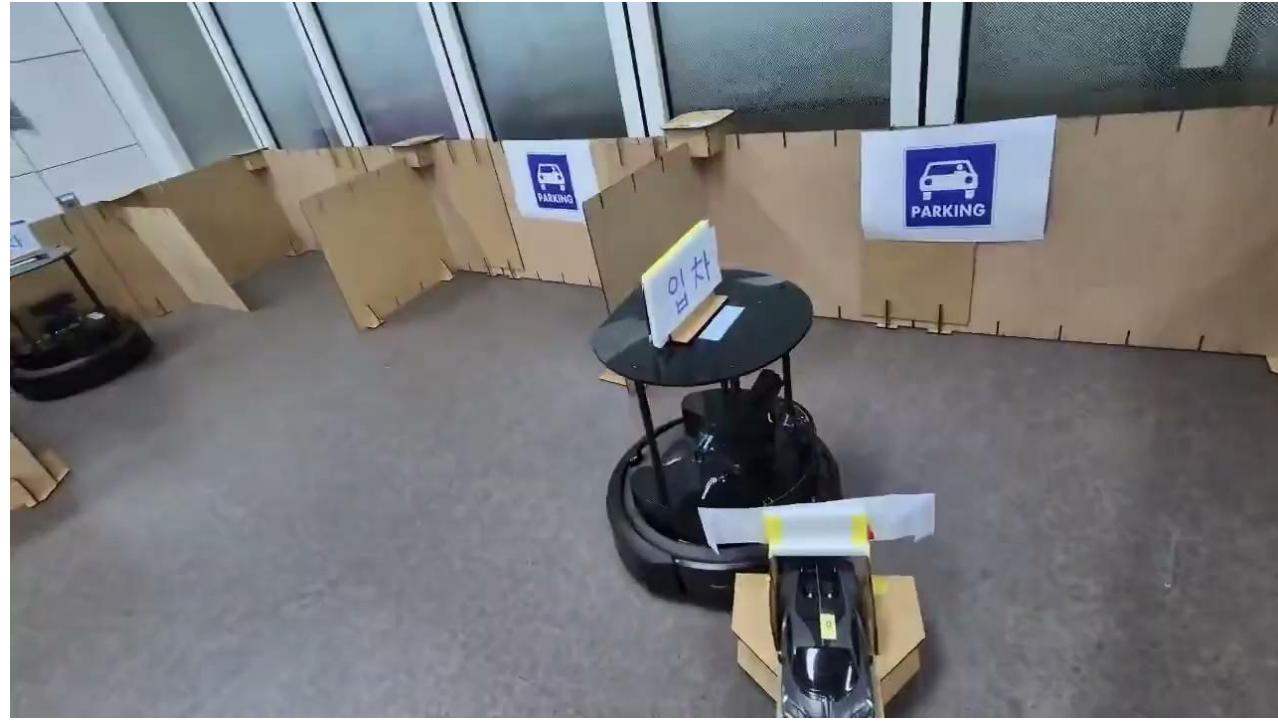
프로젝트 수행 경과

▶ 입차하기 (클릭시 영상 재생)

9. 180도 회전 -> 주차진행음 발령



10. 주차완료 -> 주차완료음 발령

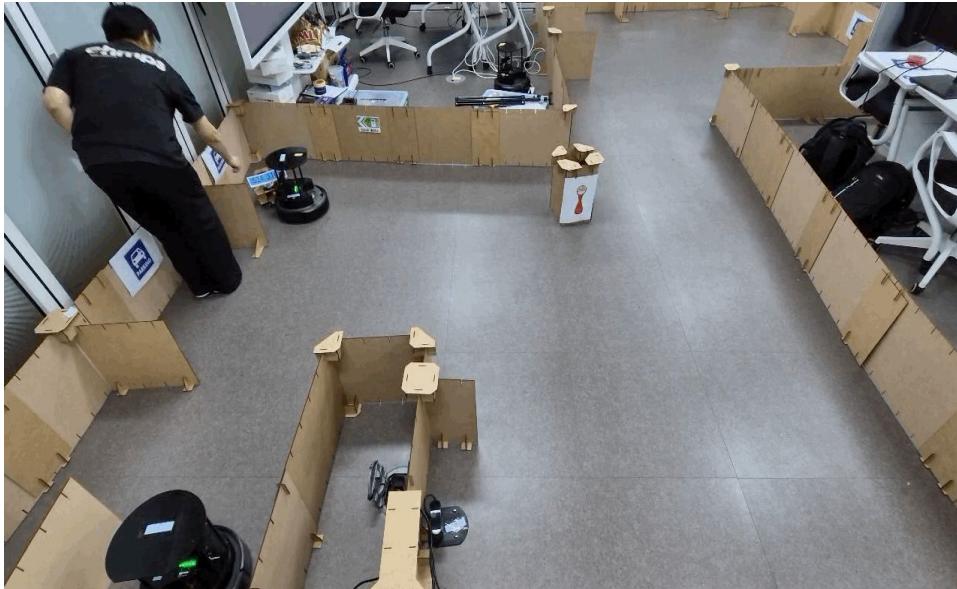


05

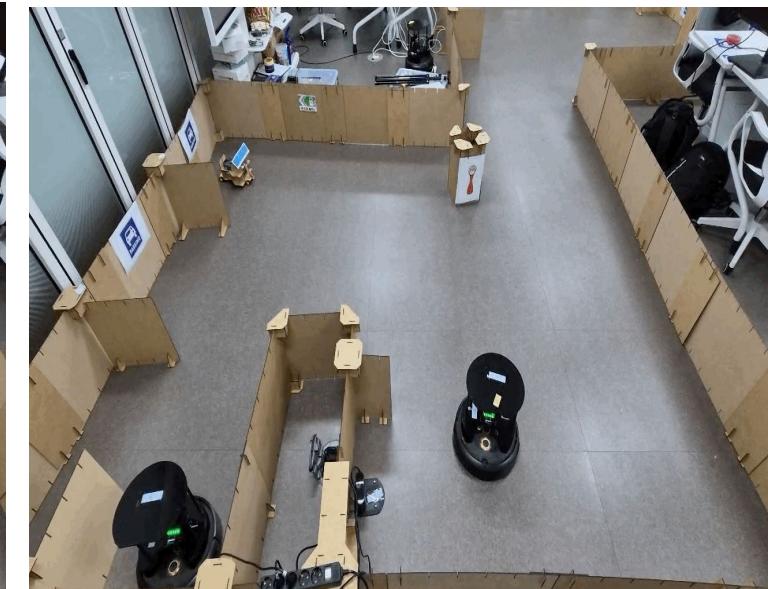
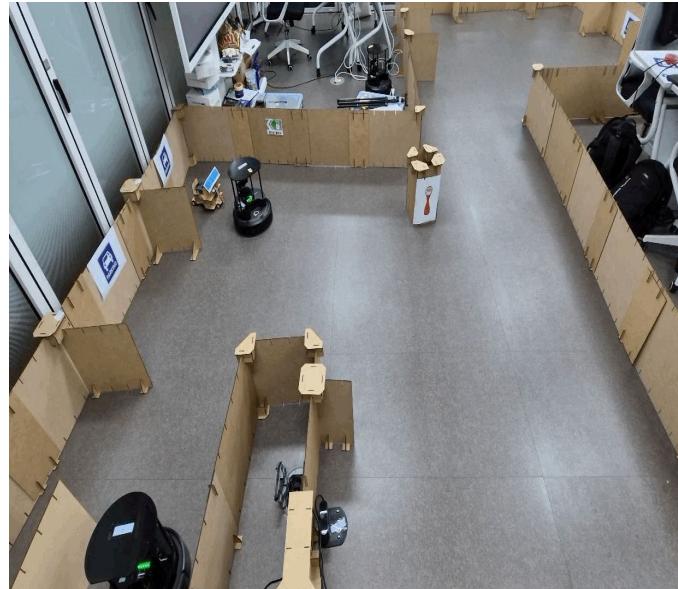
프로젝트 수행 경과

▶ 입차하기

11. 차 수레 빼주기



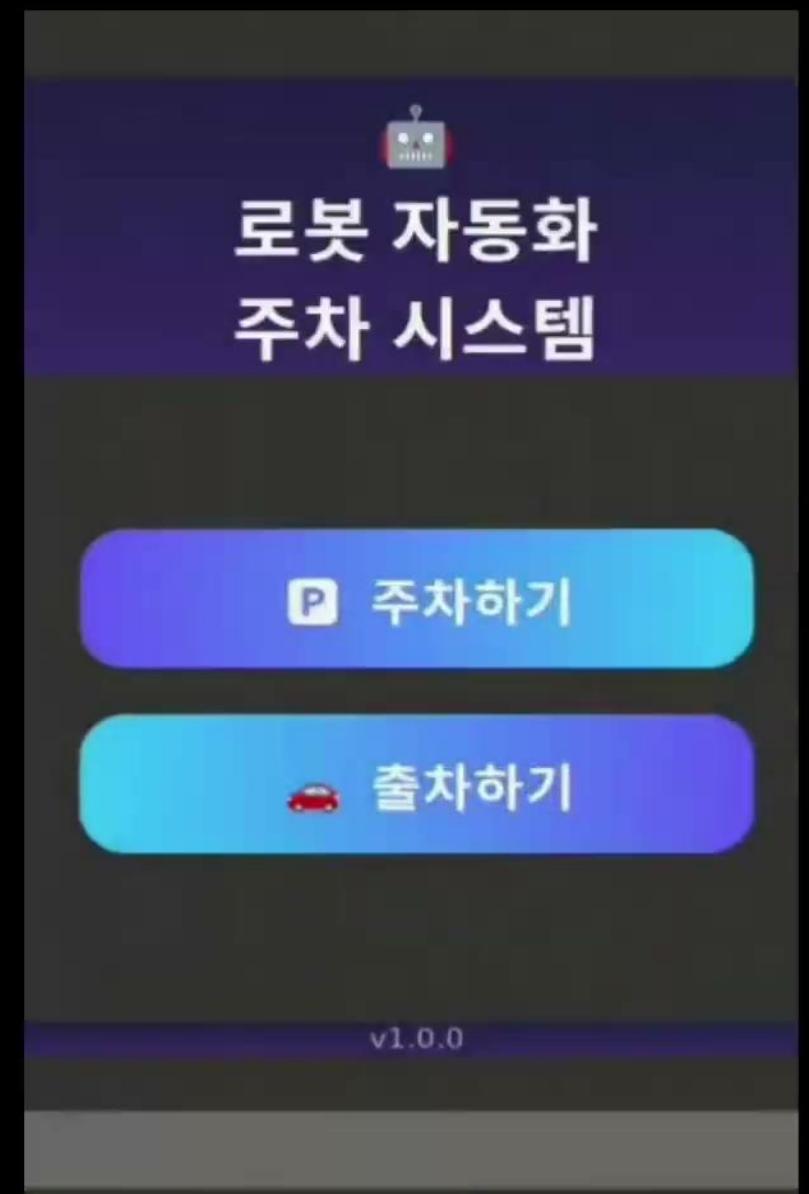
12. 대기 위치, 초기위치 복귀 , dock



▶ 입차 – 로봇 자동화 주차 시스템 (클릭시 영상 재생)



**로봇 자동화 주차 시스템
(입차)**



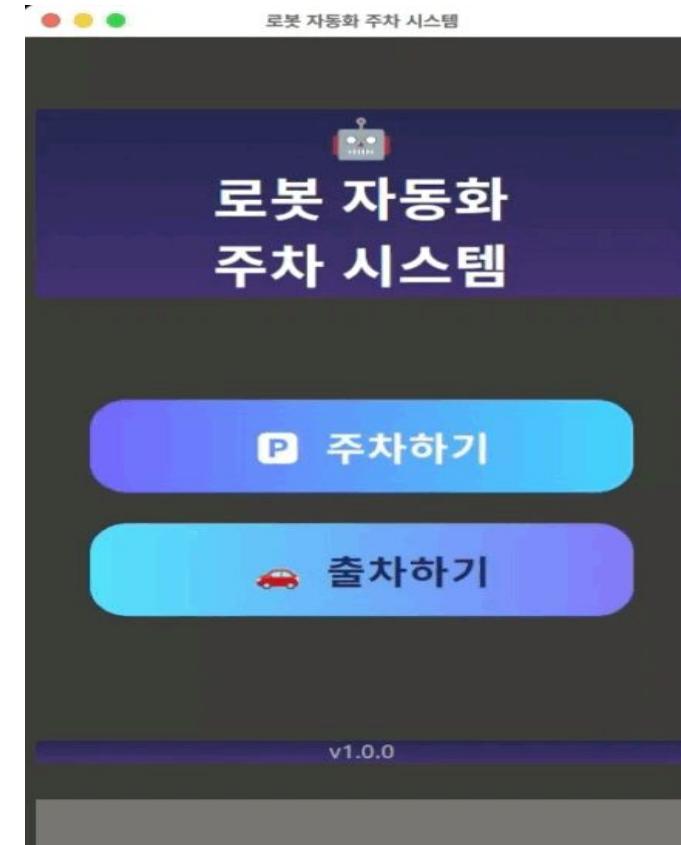
프로젝트 수행 경과

▶ 출차하기

1. nav 초기위치에서 undock



2. gui 출차하기 클릭



05

프로젝트 수행 경과

▶ 출차하기

3. 주차공간으로 이동



4. 차 수레 걸어주기



05 K-Digital Training

프로젝트 수행 경과

▶ 출차하기

5. 출차공간으로 이동



6. 차 수레 빼주기



프로젝트 수행 경과

▶ 출차하기

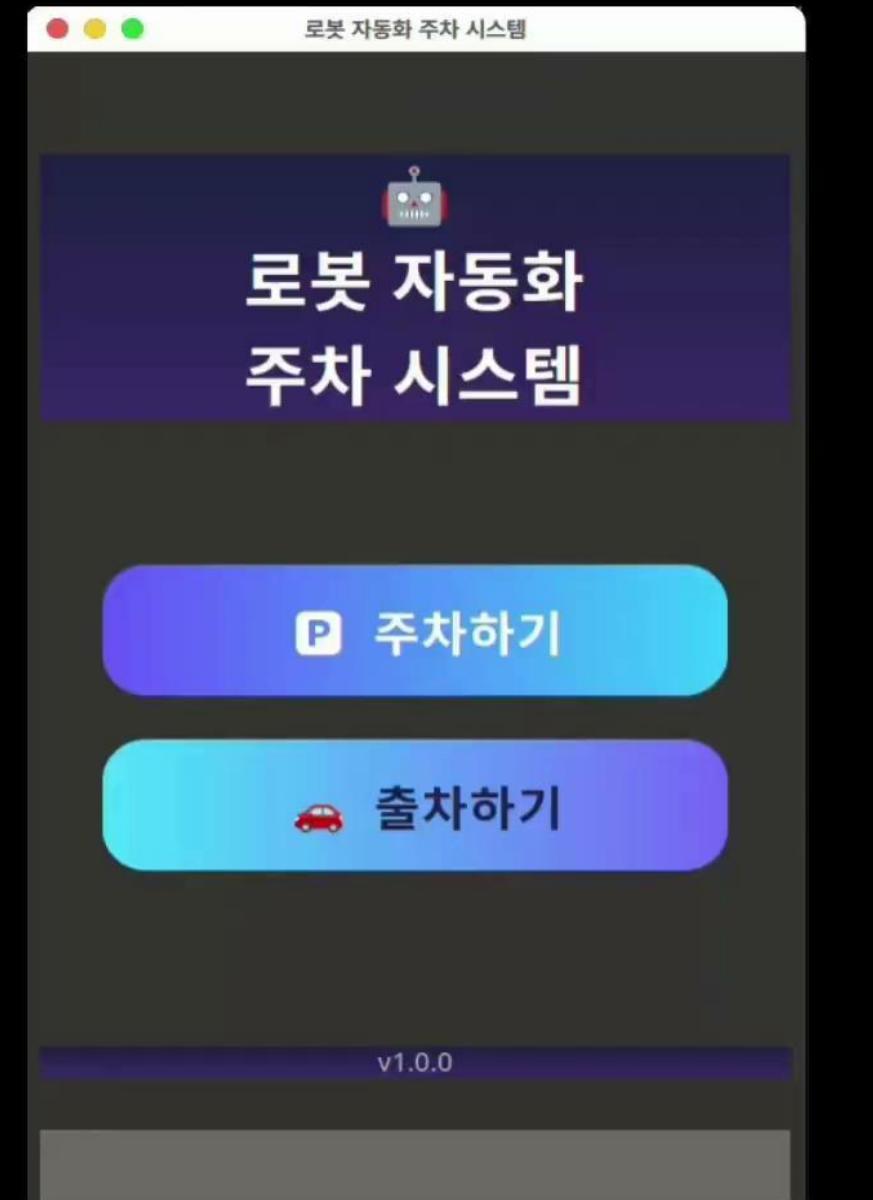
7. 초기위치 이동, dock



▶ 출차 – 로봇 자동화 주차 시스템 (클릭시 영상 재생)



**로봇 자동화 주차 시스템
(출차)**



05

K-Digital Training

프로젝트 수행 경과

▶ DB

- **Influx Database**를 사용하여 자동차
입,출차를 자동 기록
- 실시간 데이터 수집 및 접근성
- 데이터 보안 및 무결성
- 클라우드로 로컬 공유기 서버 과부하
방지

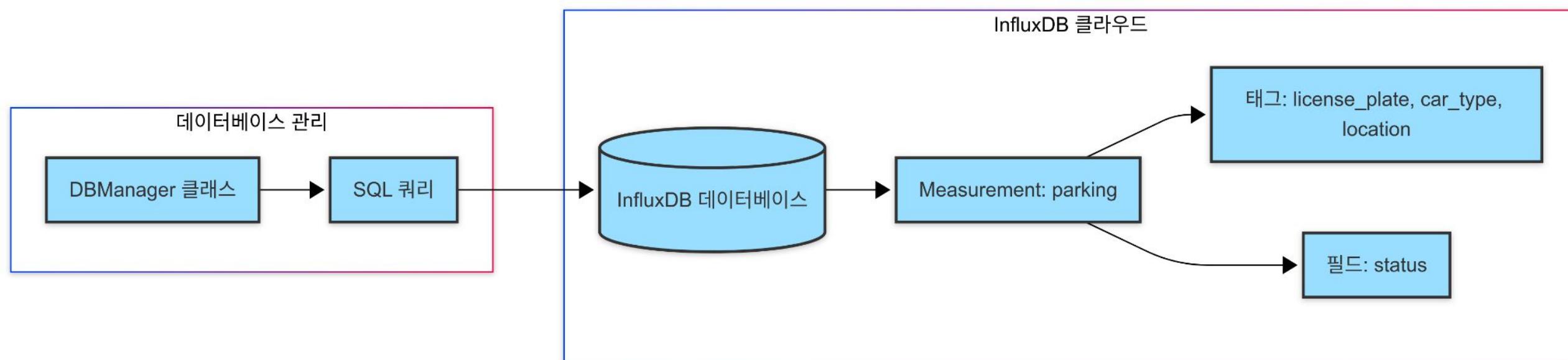
The screenshot shows the official website for InfluxDB. At the top, there's a navigation bar with links for Products, Use Cases, Developers, Pricing, Contact Us, Sign In, and a prominent pink "Start Now" button. Below the navigation is a search icon. The main headline reads "Time series starts with InfluxDB". A sub-headline says "Join the millions of developers using InfluxDB to predict, respond, and adapt in real-time. InfluxDB 3 is now generally available for production use cases." A purple button labeled "Try InfluxDB" is visible. To the right, a large screenshot of the InfluxDB 3 Query Data interface is displayed, featuring a sidebar with various database management options like Configs, Servers, Databases, and a central area for running SQL or Natural Language queries. Below this is a detailed time series chart showing multiple data series (e.g., Ge, Humidity, Status, Temperature) over a period from May 28 to June 1.

05

프로젝트 수행 경과

▶ DB

- Architecture



05

프로젝트 수행 경과

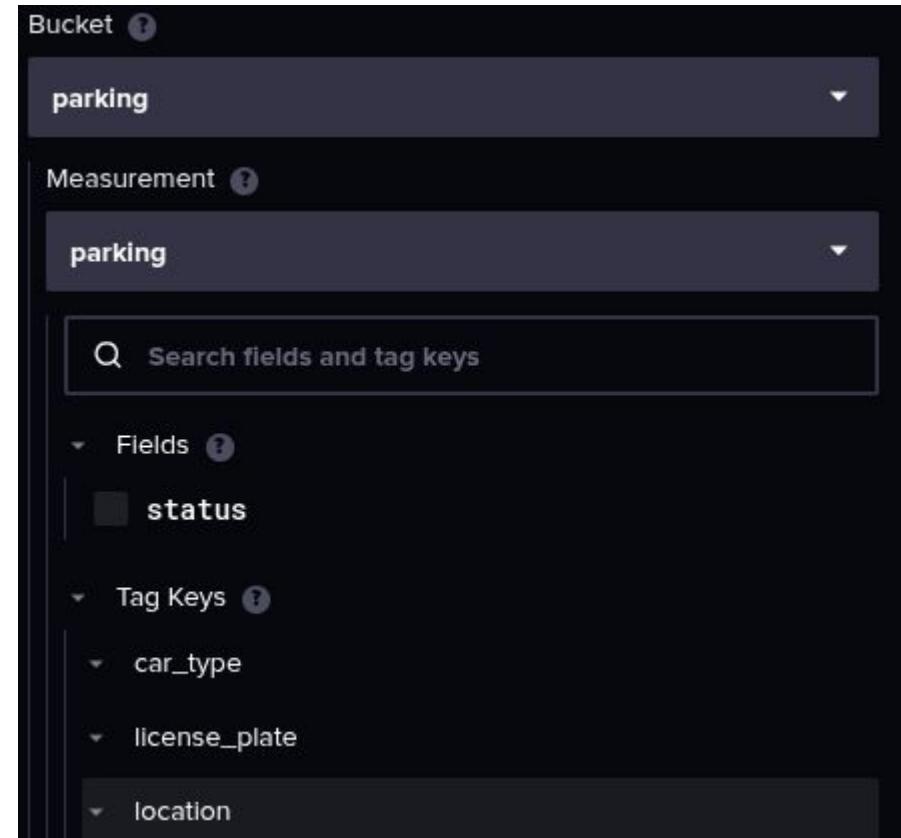
▶ DB

- **Structure**

- DataBase Name : parking
- Measurement : parking (주차 데이터 저장)

- **Schema**

- Tag Keys : car_type (차량 종류), license_plate (차량 번호),
location (차량 위치)
- Status : parked (입차), exit (출차)
- TimeStamp : 입차 및 출차 시간



프로젝트 수행 경과

▶ DB

- **Parking Location (limited)**

- normal : A-1, A-2

- ev : B-1, B-2

- disabled : C-1, C-2

car_type	license_plate	location	status	time
no group string	no group string	no group string	no group string	no group dateTime:RFC3339
disabled	123가4568	C-1	exit	2025-07-02T12:43:44.000Z
disabled	123가4568	C-1	parked	2025-07-02T12:44:45.000Z
disabled	123가4568	C-1	exit	2025-07-02T12:47:02.000Z

- **Data Flow**

- 입차 : 차량 번호 & 타입, 차량 위치, status : parked, 입차 시간 기록

- 출차 : 차량 번호 & 타입, 차량 위치, status : exit, 출차 시간 기록

- 시간 순으로 입차와 출차 기록, 관리 용이

프로젝트 수행 경과

▶ DB

- **DB Initialize**

- 클라이언트 및 관리자 초기화
- 데이터 로드 및 조회
- 최근 30일이내 주차 및 출차 이력 조회

```
2025-07-03 11:35:42,527 - INFO - InfluxDB 클라이언트 초기화 완료
2025-07-03 11:35:42,527 - INFO - DB 관리자 초기화 완료
2025-07-03 11:35:42,527 - INFO - 데이터베이스에서 데이터 로드 중...
2025-07-03 11:35:42,527 - INFO - 현재 주차된 차량 정보 조회 중...
2025-07-03 11:35:43,527 - INFO - 현재 주차된 차량이 없습니다.
2025-07-03 11:35:43,527 - INFO - 모든 주차 데이터 조회 중 (최근 30일)...
2025-07-03 11:35:43,739 - INFO - 총 46개의 데이터를 조회했습니다.
2025-07-03 11:35:43,741 - INFO - 데이터 로드 완료: 0대 주차 중
```

- **DB Synchronize**

- 5초 주기로 DB 갱신
- 주차된 차량 정보 조회
- 주차된 차량 정보 갱신

```
2025-07-03 12:00:34,671 - INFO - 데이터베이스에서 데이터 갱신 중...
2025-07-03 12:00:34,671 - INFO - 현재 주차된 차량 정보 조회 중...
2025-07-03 12:00:34,908 - INFO - 현재 주차된 차량: 1대
2025-07-03 12:00:34,908 - INFO - 데이터 갱신 완료: 1대 주차 중
2025-07-03 12:00:39,500 - INFO - 데이터베이스에서 데이터 갱신 중...
2025-07-03 12:00:39,500 - INFO - 현재 주차된 차량 정보 조회 중...
2025-07-03 12:00:39,737 - INFO - 현재 주차된 차량: 1대
2025-07-03 12:00:39,738 - INFO - 데이터 갱신 완료: 1대 주차 중
```

프로젝트 수행 경과

▶ DB

- 주차된 차량 정보 조회

1. WITH

- (1) latest_status 임시 테이블 생성
- (2) 각 차량 번호 별 가장 최근 시간 탐색
- (3) 최근 30일 이내 데이터 대상 탐색
- (4) 차량 번호 별로 그룹화

2. Main Query

- (1) p(parking table) & ls(latest_status table) JOIN
- (2) JOIN ON절(조인 조건) : 차량 번호 일치 & 최신 시간 일치
- (3) 조회 결과를 최신순(내림차순) 정렬

3. Search Result

→ license_plate(차량 번호), car_type(차량 타입), location(차량 위치),

```
WITH latest_status AS (
    SELECT license_plate, MAX(time) as latest_time
    FROM parking
    WHERE time >= now() - INTERVAL '30 days'
    GROUP BY license_plate
)
SELECT p.license_plate, p.car_type, p.location, p.status, p.time
FROM parking p
JOIN latest_status ls ON p.license_plate = ls.license_plate AND p.time = ls.latest_time
WHERE p.status = 'parked'
ORDER BY p.time DESC
```

프로젝트 수행 경과

▶ DB

- **출차 기록 조회 (days = 7, limit = 20)**

- 차량 번호, 타입, 위치, 주차 및 출차 여부, 작업 시간 선택
- 최근 7일 이내 데이터 대상 탐색
- exit(출차) 상태인 데이터만 추출
- 작업 시간 기준 최신순(내림차순) 정렬
- 최근 20개의 레코드만 반환

```
SELECT license_plate, car_type, location, status, time  
FROM parking  
WHERE time >= now() - INTERVAL '{days} days'  
AND status = 'exit'  
ORDER BY time DESC  
LIMIT {limit}
```

- **주차장 주차 및 출차 기록 조회 (days = 30, limit = 100)**

- 차량 번호, 타입, 위치, 주차 및 출차 여부, 작업 시간 선택
- 최근 30일 이내 데이터 대상 탐색
- 작업 시간 기준 최신순(내림차순) 정렬
- 최근 100개의 레코드만 반환

```
SELECT license_plate, car_type, location, status, time  
FROM parking  
WHERE time >= now() - INTERVAL '{days} days'  
ORDER BY time DESC  
LIMIT {limit}
```

프로젝트 수행 경과

▶ DB

- 차량 번호로 차량 조회

- 차량 번호가 동일하고 주차 상태인 레코드 반환
- 작업 시간 기준 최신순(내림차순) 정렬
- 반환 레코드는 1개(차량 1대)

```
SELECT license_plate, car_type, location, status, time
FROM parking
WHERE license_plate = '{license_plate}'
AND status = 'parked'
ORDER BY time DESC
LIMIT 1
```

- 위치 기반 차량 조회

- 차량 위치가 동일하고 주차 상태인 레코드 반환
- 작업 시간 기준 최신순(내림차순) 정렬
- 반환 레코드는 1개(차량 1대)

```
SELECT license_plate, car_type, location, status, time
FROM parking
WHERE location = '{location}'
AND status = 'parked'
ORDER BY time DESC
LIMIT 1
```

프로젝트 수행 경과

▶ DB

- 차량 번호 뒷자리 4자리 숫자로 차량 검색
 - 주차 상태이면서 검색한 4자리 숫자를 포함한 레코드 반환
 - 작업 시간 기준 최신순(내림차순) 정렬

```
SELECT license_plate, car_type, location, status, time
FROM parking
WHERE status = 'parked'
AND license_plate LIKE '%{partial_plate}'
ORDER BY time DESC
```

프로젝트 수행 경과

▶ DB

- 주차 및 출차 이벤트로 인한 데이터 추가
 - InfluxDB는 관계형 데이터베이스가 아닌 시계열 데이터베이스(비정형), SQL의 **INSERT** 대신 직접 데이터베이스에 작성
 - line_protocol 형식(InfluxDB에서 요구하는 형식)으로 차량 정보(차량 번호, 타입, 위치, 상태, 시간)를 작성
 - 작성한 내용을 InfluxDB에 작성

```
# 현재 시간 (RFC3339 형식)
timestamp = int(datetime.now().timestamp())

# Line Protocol 형식으로 데이터 생성
# measurement, tag1=value1, tag2=value2 field1=value1, field2=value2 timestamp
line_protocol = f"parking,license_plate={license_plate},car_type={car_type},location={location} status=\"{status}\" {timestamp}"

# InfluxDB에 데이터 쓰기
self.client.write([line_protocol], write_precision='s')
```

▶ 입·출차 – 로봇 자동화 주차 시스템 (클릭시 영상 재생)



로봇 자동화 주차 시스템 (입차 출차)



자체 평가 의견

완성도 평가

10/10

개선점이나 보완할 점

- GUI를 안드로이드 앱으로 만들어 유저 친화력 향상
- 주차공간으로 들어갈때의 로봇 자세의 정확성

평가 항목	세부 기준	배점
1. AI Vision 객체 인식 정확도 및 Depth 거리 계산이 이루어지는가 ?	정확도 90~95, depth 거리 계산 3*3으로 개선	10점
2. TF Transform 좌표 변환을 이용하여 목표 위치로 이동(Nav2)을 수행하는가 ?	tf 좌표 기준 -> baselink-> x축 offser-> map 목표 위치 설정 후 이동	10점
3. 로봇 간 협업이 이루어졌는가 ?	입차용 로봇, 출차용 로봇 db 연동	10점
4. 전체 시스템이 통합되어 One Take로 동작할 수 있는가 ?	입차, 출차 ONE TAKE 영상	5점
5. 비즈니스 요구사항, 시스템 요구사항, 시스템 설계, 노드 구성도가 논리적이고 구체적인가 ?	시스템 flow, 아키텍쳐, 노드 구성도 작성	10점
6. 기능별 flow chart, 세부 설명(노드, 토픽 등), 영상으로 설명되었는가 ?	기능별 flow, 세부 설명 시나리오 작성	10점
7. PPT 및 발표가 제시한 문서 구성과 시간을 준수했는가 ?	각 단위기능 부터 통합기능, 최종영상	10점
8. 챌린지 기능을 추가하였는가 ?	추가 기능(gui, 경고음, db)	5점
총점		70점

자체 평가 의견

잘한 부분

- YOLO 객체인식, Depth, TF, Navigation장애물 회피 등 수업에서 배운 내용 모두 수행
- GUI로 유저친화적인 요소 추가
- DB를 사용하여 데이터 무결성 유지

아쉬운 점

- 통합 부분에서 많은 시간 소비
- 경량화를 진행했음에도 네트워크 사용자가 너무 많아 터틀봇4의 임무 수행 버거움

느낀 점이나 경험한 성과

- 개발 과정보다 네트워크 품질에 의해 프로젝트의 성과 변동
- 클라우드 DB를 사용하여 데이터 관리 용이 및 보안 향상



깃허브

<https://github.com/rokeycb4/turtlebot4-slam-nav>

소스코드

[https://github.com/rokeycb4/turtlebot4-slam-nav/tree/main/
rokey_ws/src/rokey_pjt/rokey_pjt](https://github.com/rokeycb4/turtlebot4-slam-nav/tree/main/rokey_ws/src/rokey_pjt/rokey_pjt)