

2023 국제 로봇 콘테스트 스팀컵 오토레이스 전기적 자동차시점 팀 메뉴얼

최원준, 양윤희

December 15, 2023



Introduction

2023 국제 로봇 콘테스트에서 참가한 경험과 함께 Steamcup Autorace 부문에서의 개발 과정을 공유하고자 합니다¹. 우리 팀은 Steamcup Autorace 부문에서 참가하여 많은 도전과 경험을 얻었습니다. 경쟁이 치열한 만큼, 우리는 효과적인 전략 수립의 중요성을 배웠고 이후의 참가자들을 위한 메뉴얼을 작성했습니다. 이 메뉴얼은 해당 대회 참가를 위한 개발의 기초부터 고급 기술까지 모든 적용된 기술을 다루며, 효과적인 차량 설계와 프로그래밍에 대한 가이드를 제공합니다. 또한, 우리가 마주한 어려움과 극복한 해결책들도 공유하여 미래의 참가자들이 더욱 빠르게 발전할 수 있도록 돋고자 합니다. 이 메뉴얼은 로봇 콘테스트 참가자들에게 소중한 참고 자료가 될 것이며, 더 나아가 로봇 기술 분야에 새로운 도약을 이끌어낼 수 있는 계기가 될 것입니다.

¹이 프로젝트는 부산대학교 전기공학과의 정한유 교수님의 지도와 지원으로 진행되었습니다. 정한유 교수님의 지원 아래, 우리 팀은 성공적인 프로젝트를 성취할 수 있었습니다. 감사의 인사를 전합니다.

Contents

1 대회 개요	3
1.1 로봇 규정	3
1.1.1 로봇 규격	3
1.1.2 미션 수행에 제한 사항	3
1.1.3 제한 사항	4
1.2 경기 규정	4
1.2.1 경기 방법	4
1.2.2 심사 기준	4
1.3 미션	5
1.3.1 신호등 구간	5
1.3.2 갈림길 구간	6
1.3.3 공사 구간	6
1.3.4 주차 구간	6
1.3.5 차단바 구간	7
1.3.6 터널 구간	7
1.4 실제 경기장	8
2 H/W Setup	11
2.1 Robot Configuration	11
2.2 Camera Setup	12
2.3 Port Setup	12
2.4 Battery Setup	13
2.5 OpenCR Setup & Hardware Assembly	13
2.6 Etc.	13
3 S/W Setup	14
3.1 개발 환경	14
3.1.1 Operating System	14
3.1.2 ROS	14
3.1.3 Utility	14
3.2 Code Setup	14
3.3 기본 명령어	14
3.4 ECP Package	17
3.4.1 ecp_sensor	17
3.4.2 ecp_preproc	18
3.4.3 ecp_description	22
3.4.4 ecp_slam	23
3.4.5 ecp_navigation	23
3.4.6 ecp_control	24
3.4.7 ecp_bringup	26
3.5 Demo Scenario	27
3.5.1 실제 환경 Launch 파일 실행 순서	27
3.5.2 Gazebo 환경 Launch 파일 실행 순서	27
3.5.3 Debug 적용하여 rqt_reconfigure 사용을 원할 때	27

1 대회 개요

본 대회는 2017년부터 7년간 진행되어온 대회로써 ROBOTIS에서 판매하는 Turtlebot 기반의 로봇으로 진행되어 왔으나 2023년 대회는 ROS(Robot Operating System)을 운용 가능한 플랫폼을 활용하여 참가하면 되는 규정으로 변경되었다. 참가 대상은 고등학생 이상의 2인 4인으로 구성된 팀 단위로 참가 가능하다.

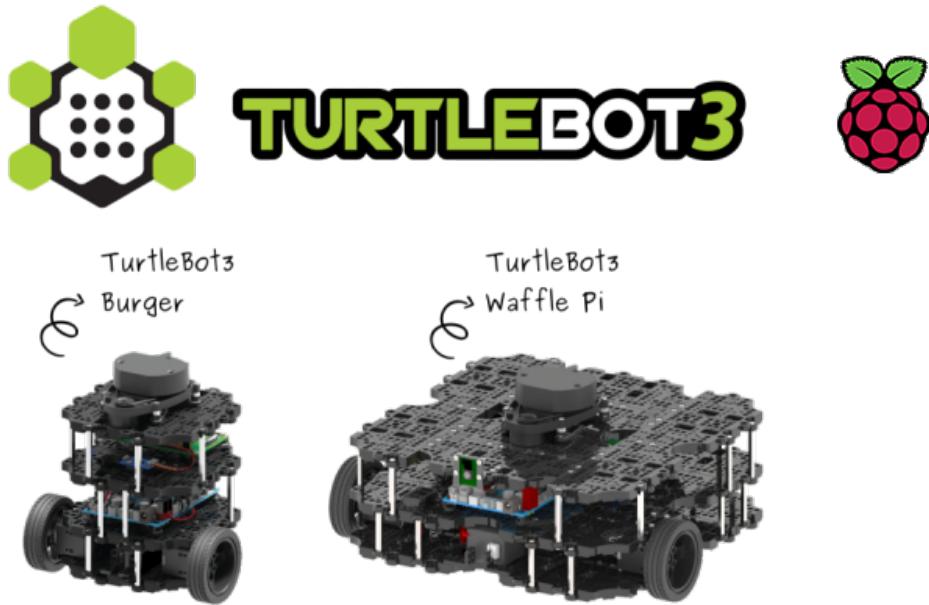


Figure 1: Turtlebot Device.

1.1 로봇 규정

아래는 2023년도의 로봇 규정에 대한 사항이며 자세한 내용은 2023 Autorace 규정집에서 확인할 수 있다.

1.1.1 로봇 규격

1. 로봇은 ROS 공식 플랫폼을 기본으로 하며 SBC(Single Board Computing), 센서 등을 자유롭게 추가 및 변형할 수 있다.
2. 구동 장치의 중간 제어기는 OpenCR을 사용해야 함.
3. 구동 장치는 플라스틱 기어를 사용하며 통신 방식은 TTL 방식, 전압은 최대 12V까지 허용함.
4. 구동 장치의 크기는 25x45x30 (가로x세로x깊이) 이상, 30x47x35 (가로x세로x깊이) 미만으로 제한함.

1.1.2 미션 수행에 제한 사항

1. 주행로 폭 및 터널 구간 천장 높이를 참고하여 변경해야 함.

1.1.3 제한 사항

- 로봇의 크기, 무게, 부품의 추가 및 변형에 제한은 없으나 경기장을 손상시킬 가능성이 있거나 위험 요소가 있는 경우 심판은 소속 팀에게 위험 요소 제거를 명령할 수 있음.
- 로봇은 외부 조작 없이 인식 시스템 등을 이용한 완전 자율 동작만으로 미션을 수행해야 함.
- 각 팀은 개발 및 처리용 오퍼레이팅 PC를 사용할 수 있으며, 로봇과의 통신 방법은 각 팀이 마련해야 함.

1.2 경기 규정

1.2.1 경기 방법

- 모든 팀은 1회의 주행 기회가 주어지며 그 결과로 순위를 정한다.
- 경기 시작 이후로 로봇 또는 PC를 접촉 시 회당 5점 감점한다.
- 로봇이 동작하지 않고 30초 이상 정지돼 있는 경우 동작 불응 상태로 보고 경기를 종료한다.
- 팀당 최대 10분의 시간이 주어진다. (준비 시간 5분, 미션 수행 제한 시간 5분)

1.2.2 심사 기준

- 미션 점수

Traffic Light	2 way Intersection	Construction Site	Parking	Level Crossing	Tunnel	Time Score
20 point	20 point	20 point	20 point	20 point	20 point	20 point

Figure 2: 미션 점수 표.

- 점수는 미션 점수와 시간 점수, 감점을 합산/감산하여 결정된다.
- 동점자는 아래 순서대로 비교하여 순위를 정한다.
 - 미션 구간의 시도 횟수가 높은 순
 - 감점 점수가 적은 순
 - 시간 점수가 높은 순

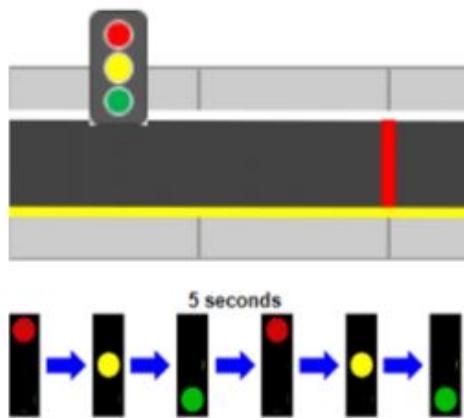
1.3 미션

대회 진행 동안에 점수를 부여 받을 수 있는 미션은 6개이고 기본적인 표지판 인식과 차선 주행은 완료 가능해야 한다.



Figure 3: 경기장 도식.

1.3.1 신호등 구간



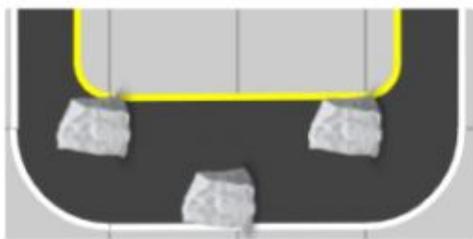
1. 로봇은 순차적으로 등장하는 신호등의 색을 스스로 인식하고 외부의 조작 없이 출발해야 한다.
2. 신호등은 빨간색, 노란색, 녹색으로 순환하며 녹색 불은 5초간 점등된다.
3. 로봇의 출발 여부와 상관 없이 첫 번째 녹색 불이 점등되는 순간부터 미션 시간이 자동으로 시작된다.
4. 로봇은 녹색 신호일 때 출발선을 통과해야 하며, 그 외의 신호에서 출발 시 미션 실패로 판정된다.

1.3.2 갈림길 구간



1. 로봇이 미션을 시작한 이후 랜덤으로 표시되는 좌회전 또는 우회전 이정표의 방향으로 로봇이 이동하면 된다.
2. 이정표 신호와 다른 방향으로 진행시 미션 실패로 판정한다.

1.3.3 공사 구간



1. 장애물을 피해 주행로를 통과하면 성공하는 미션이다.
2. 장애물은 바닥에 고정되어 있다.
3. 미션 중 로봇이 장애물에 접촉 여부는 상관없다.

1.3.4 주차 구간



1. 하우스 로봇이 주차되어 있지 않은 구역에 로봇이 완전히 진입한 후 빠져 나오면 성공하는 미션이다.

2. 하우스 로봇이 주차된 공간에 진입 시 미션 실패로 판정한다. 단, 미션 성공 이후 진입은 무방하다.
3. 하우스 로봇이 주차된 위치는 바뀔 수 있다.

1.3.5 차단바 구간



1. 1번 센서 감지시 차단바가 내려오면 정지하고 차단바가 개방되면 통과한다.
2. 차단바가 내려온 상태에서 2번 센서에 로봇이 감지될 경우 미션실패로 판정한다.
3. 2번 센서의 위치는 차단바로부터 6cm 떨어져 있으며, 1번 센서의 위치는 임의의 위치에 놓일 수 있다.

1.3.6 터널 구간



1. 조명과 이정표 등이 없는 암전구간을 통과하면 된다.
2. 로봇이 터널에서 나오지 못할 경우 미션 실패로 판정한다.
3. 내부에는 임의의 장애물들이 설치되어 있다. (형태, 크기 랜덤)
4. 터널 구간 내부의 크기는 약 1.8m x 1.8m이며, 높이는 24cm이다.
5. 입구/출구의 폭은 주행로와 같으며, 높이는 24cm이다.

1.4 실제 경기장

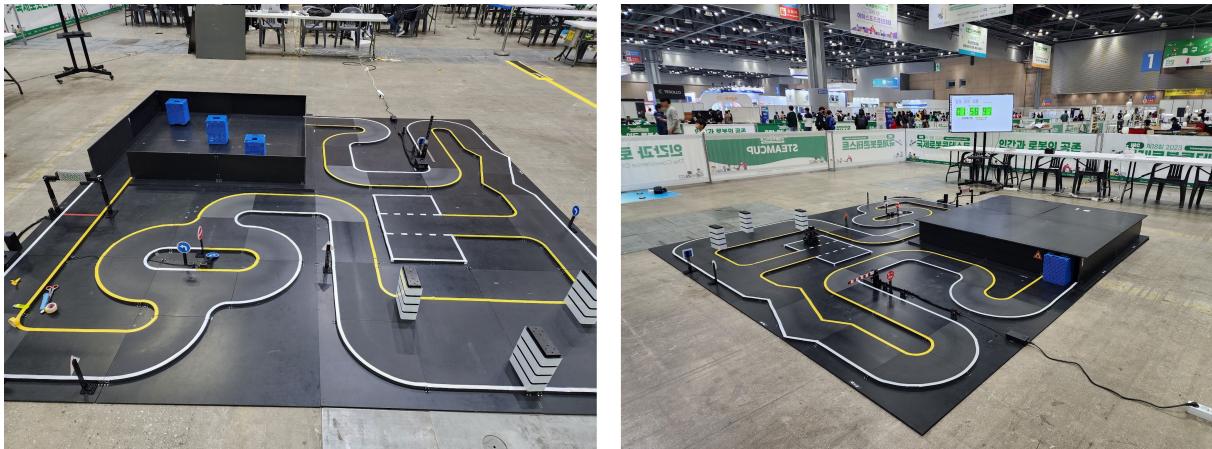


Figure 4: 실제 대회 경기장.

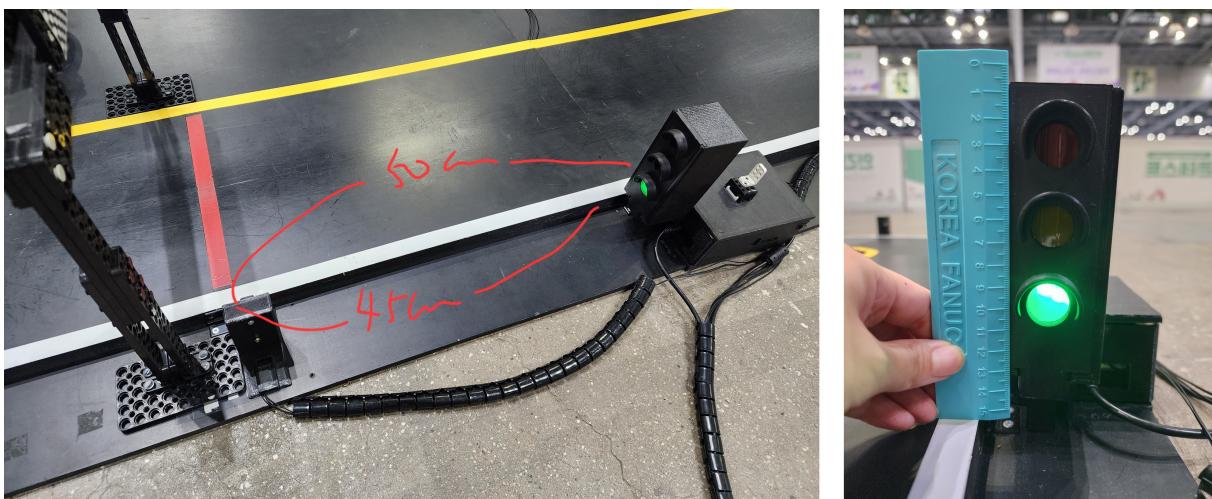


Figure 5: 신호등 규격.



Figure 6: 신호등 변경 모습.



Figure 7: 실제 표지판 규격.

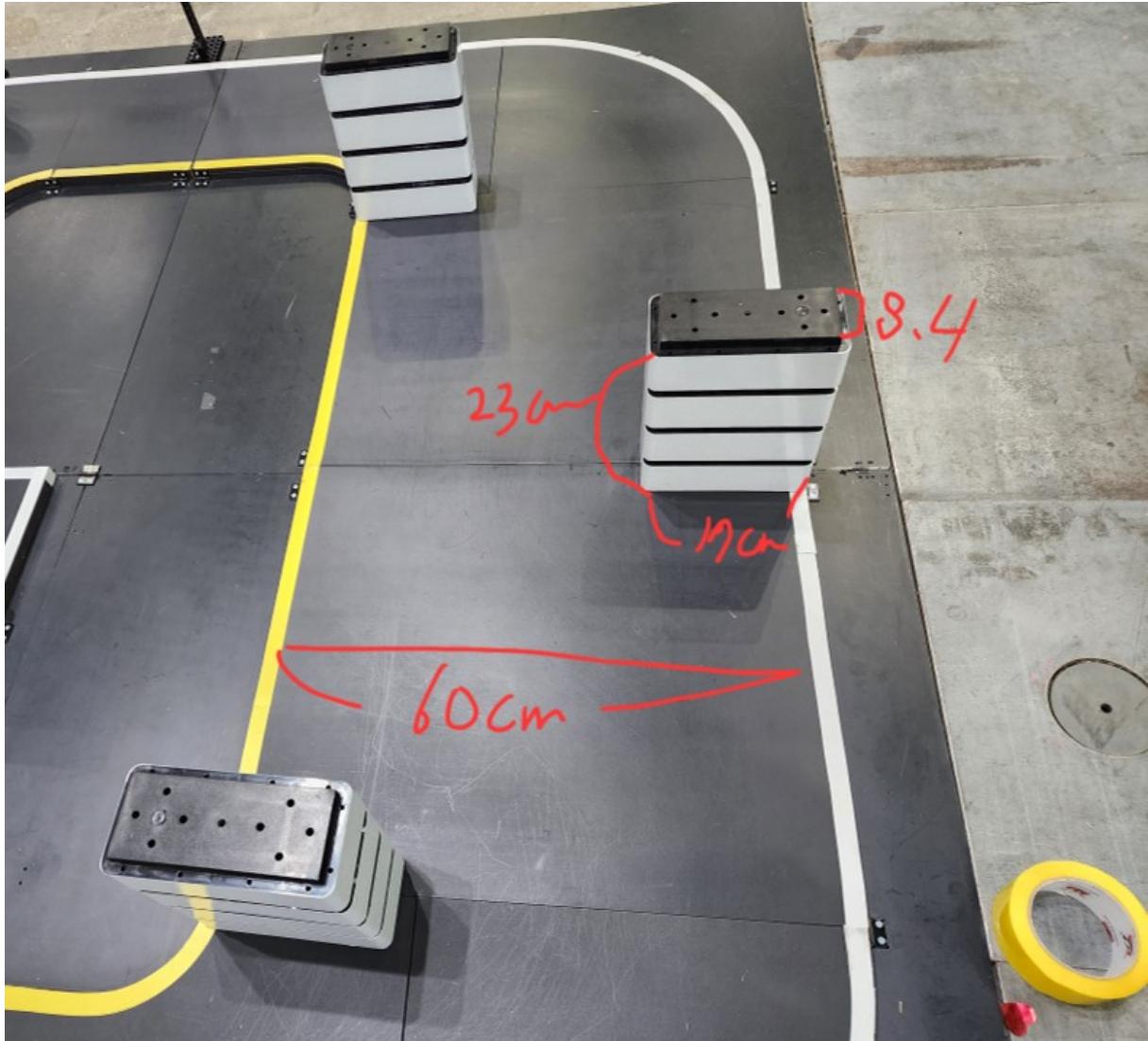


Figure 8: 실제 장애물 규격.

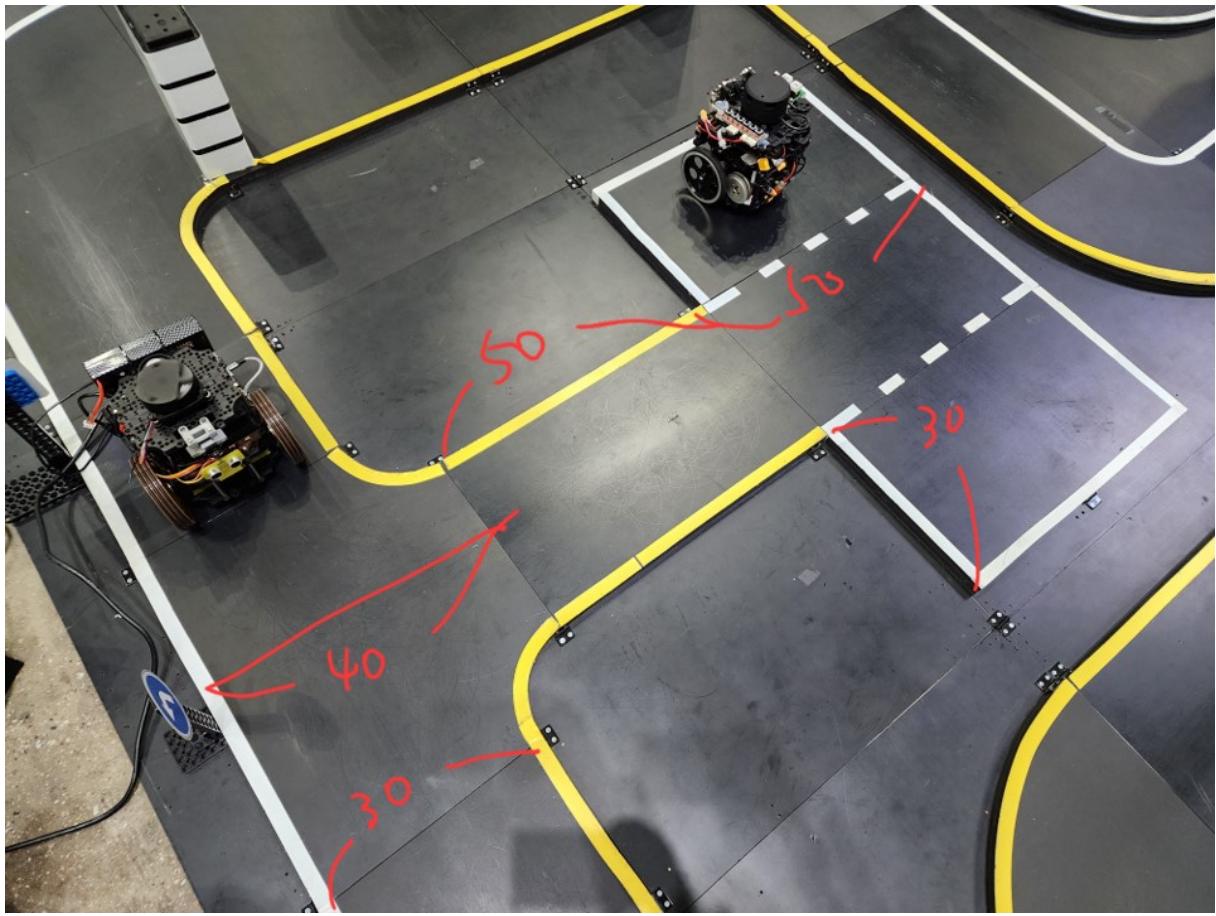


Figure 9: 실제 주차장 규격.

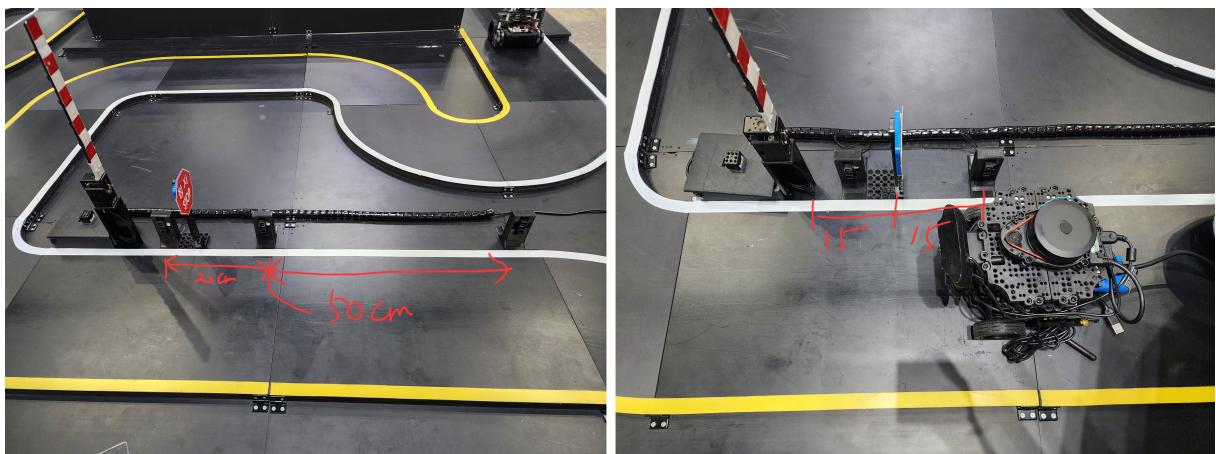


Figure 10: 실제 차단바 규격.

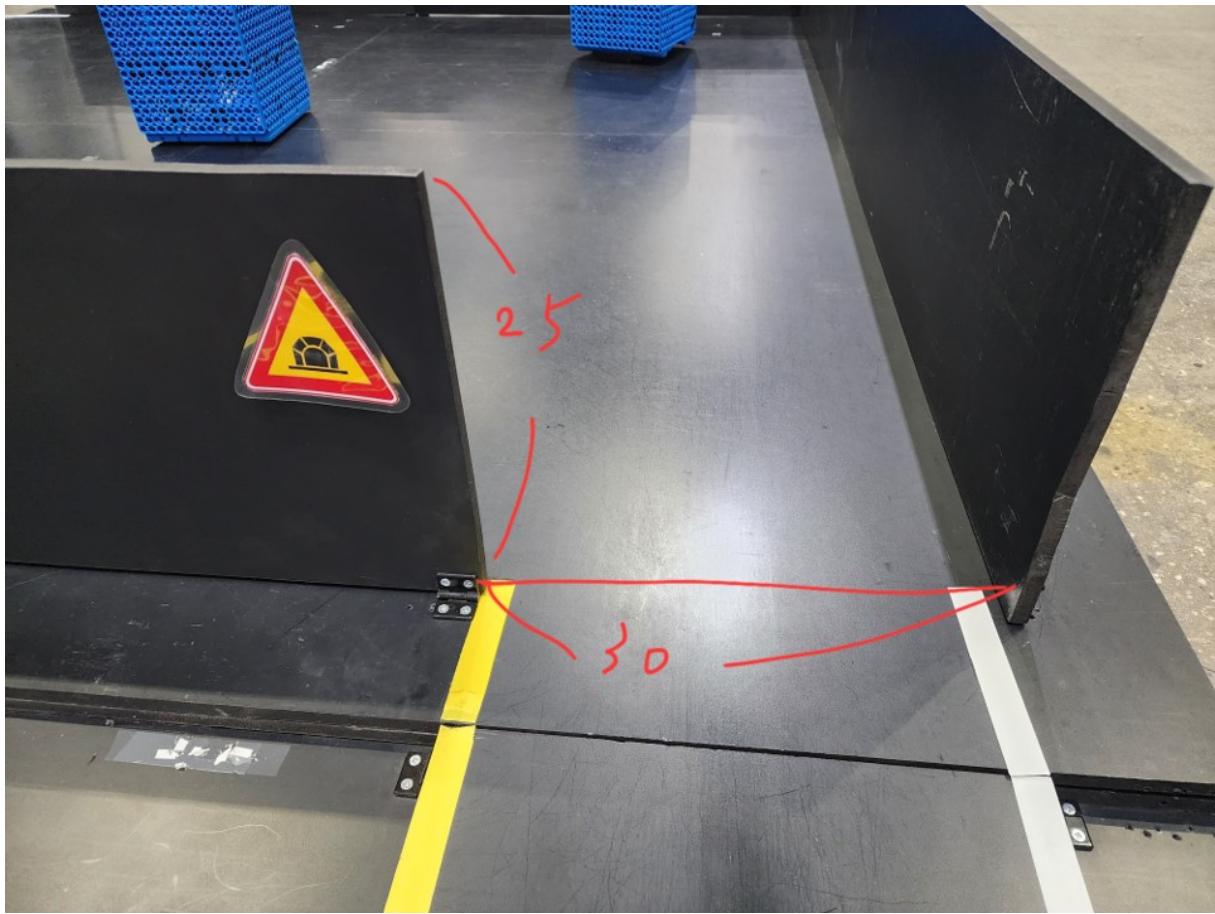


Figure 11: 실제 터널 입구 규격.

2 H/W Setup

2.1 Robot Configuration



Figure 12: 프로젝트에 사용 된 로봇 및 센서.

기존 Turtlebot3에 라즈베리파이 대신 젯슨 나노(+와이파이 모듈 추가)와 카메라 2개를 추가하여 구성 하였음. 우리는 최소한의 센서를 추가하여 비전 기반으로 모든 미션을 수행 하려고 Figure 12와 같이 제작 하였음.

2.2 Camera Setup

[turtlebot3]의 3층과 4층 팬의 전방 부분에 하단의 카메라는 뒤집고 상단의 카메라와 맞물리 게(Figure 13의 파랑 박스) 끼워서 고정. 상단의 카메라는 최대한 아랫쪽을 향할 수 있도록 각도를 조정, 하단의 카메라는 지면과 90도가 되도록 각도를 조정 (Figure 13의 빨강 박스).

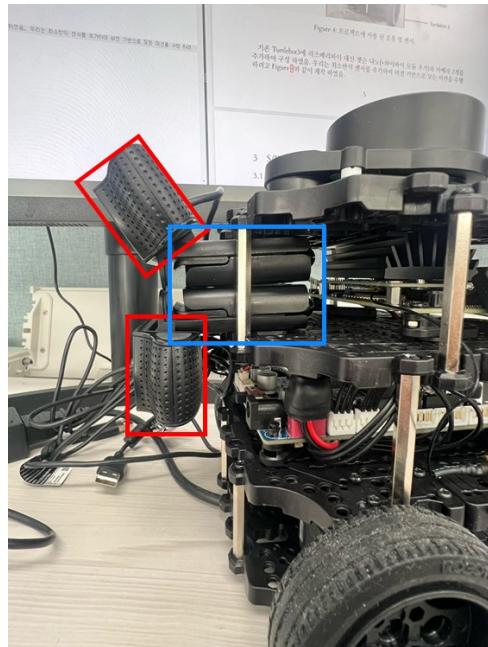


Figure 13: 카메라 세팅 방법.

2.3 Port Setup

[Jetson Nano]는 2개의 Display(상: DP, 하: HDMI) 포트(Figure 14 빨강), 4개의 USB-A 포트(Figure 14의 노랑)와 1개의 LAN 포트(Figure 14의 초록)로 구성되어 있음. USB-A 포트는 로봇의 Lidar 센서, OpenCR과 연결, 두 개의 카메라로 사용 됨.

이 중 OpenCR과 Lidar 센서의 USB-A to micro 5-pin 케이블은 90도 꺾여있기 때문에 좌측에만 연결할 수 있고 해당 포트를 사용하면 display 포트가 사용 불가함. USB-A to micro 5-pin 케이블을 90도 꺾이지 않은 케이블을 추가 구매하는 것을 추천(케이블 중에는 데이터 전송이 불가능하고 충전만 가능한 케이블이 있으니 확인 후 구매). USB 포트를 더 사용하고 싶다면 USB Hub를 구매하여 장착하여 사용(단, USB Hub 사용 시 카메라 입력에 딜레이가 발생 가능성 있으므로 카메라는 직접적으로 연결 바람).

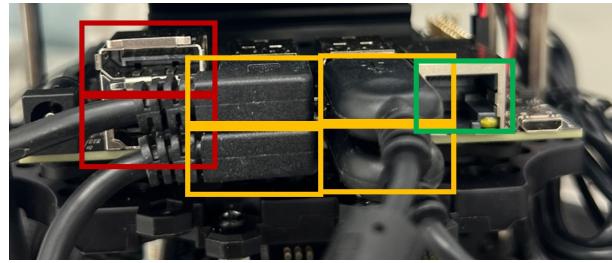


Figure 14: Jetson Nano Port Setup.

2.4 Battery Setup

로봇의 배터리를 풀 충전하여 사용하게 되면 센서 사용 및 연산 정도에 따라 약 2~3시간 정도 사용 가능. 배터리가 부족하게 되면 OpenCR에서 부저음이 들리고 불이 들어와도 로봇이 작동하지 않음. 여러분의 배터리를 충전하면서 사용하길 권장. 로봇의 배터리 분리는 Figure 15(a)의 초록 박스의 고정 부품을 제거하고 배터리를 꺼내어 Figure 15(b)과 같이 배터리를 분리할 수 있음.

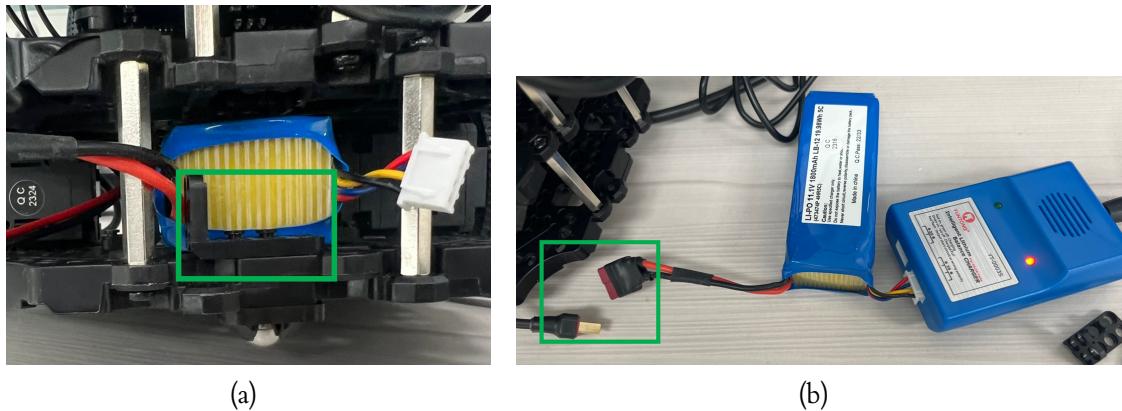


Figure 15: (a) 로봇의 배터리 고정 방법. (b) 배터리 교체를 위한 분리 방법.

2.5 OpenCR Setup & Hardware Assembly

모터 제어를 위한 OpenCR 내부의 코드 업로드는 ROBOTIS e-Manual에서 제공하는 자료를 따라 설치하였음(link). 하드웨어 조립은 **turtlebot** 책자를 통해 제작 하였으며, **Jetson Nano**로 변경하기 위해 Power 공급을 위해서는 OpenCR의 5V 포트와 **Jetson Nano** pin 보드 중 3, 5번 포트의 옆에 있는 5V, GND 포트를 각각 빨강, 검정 포트로 연결해야함.

2.6 Etc.

SBC와 PC간 원격을 하기 위해서는 와이파이 모듈이 반드시 필요함. 또한 키보드나 마우스를 사용하기 위해 Bluetooth 모듈이 필요함. USB-A 포트가 한정되어 있기 때문에 동글이 아닌 Bluetooth 기반의 키보드 및 마우스 이용하는 것을 권장. 딜레이 없이 실시간으로 **Jetson Nano**의 상황을 확인하고 싶다면 휴대용 모니터를 구매하여 사용하는 것을 추천.

WiFi의 아이피 설정은 로봇과 원격 PC간 Topic을 주고 받기 위해 반드시 설정 해야하기 때문에 반드시 다음 링크를 확인하고 진행하길 권장. (설명 Link).

3 S/W Setup

3.1 개발 환경

3.1.1 Operating System

`Jetson Nano 4GB`와 `Ubuntu` PC에서 개발을 진행하였음. `Jetson Nano`와 PC 모두 `Ubuntu`는 `18.04 LTS` 버전을 설치 하였으며, `Jetson Nano` 설치는 64GB SD 카드에 진행하였다 (설치 Link).

3.1.2 ROS

`Ubuntu 18.04 LTS`를 지원하는 ROS의 버전은 `melodic`이므로 `ros-melodic`을 설치하여 진행하였다 (설치 Link).

3.1.3 Utility

VSCode에서 python extension을 설치하여 개발을 진행하는 것을 권장한다. PC와 `Jetson Nano`의 코드를 공유고 버전관리를 하며 작업하기 위해서는 `git`을 활용하는 것을 추천하며, `Jetson Nano`의 코드를 직접 수정하고 싶다면 VScode의 ssh extension을 활용하여 원격으로 접속하여 작업하는 것을 권장한다. 여러 launch 파일들을 실행하기 위해서 `Terminator`를 설치하여 작업하는 것을 추천한다 (설치 Link).

3.2 Code Setup

우리가 개발한 코드는 모두 github <https://github.com/asd147asd147/2023-Autorace>에 업로드 되어있습니다. `catkin_ws`로 이동하여 아래의 명령어를 통해 모든 코드를 다운로드 하시길 바랍니다. 해당 명령어를 사용하기 위해서 `git`을 설치해야 합니다 (설치 Link. 프로젝트 진행 시 이해가 안되거나 문제가 발생하면 `github` 내의 `Issue` 란에 문제점을 작성해주시면 알려드리도록 하겠습니다. 또한 해당 대회를 참가하기 전에 참여했던 개발 문서를 작성중에 있으니 참고하시기 바랍니다 (Blog Link).

```
1 $ git clone https://github.com/asd147asd147/2023-Autorace.git
```

3.3 기본 명령어

- 새로운 패키지를 설치하였을 때 `catkin_ws`에서 반드시 실행해야 하는 명령어:

```
1 $ catkin_make
```

- ROS 명령어를 실행하기 위한 터미널에 반드시 한 번 실행 해야하는 명령어 :

```
1 $ roscore
```

- 특정 Launch 파일을 실행하기 위한 명령어(한 줄에 입력) :

```
1 $ roslaunch {package_name} {launch_file_name}.launch
```

- Gazebo 환경에서 시뮬레이션을 위한 명령어(한 줄에 입력) :

```
$ roslaunch turtlebot3_gazebo
turtlebot3_autorace_2020.launch
```

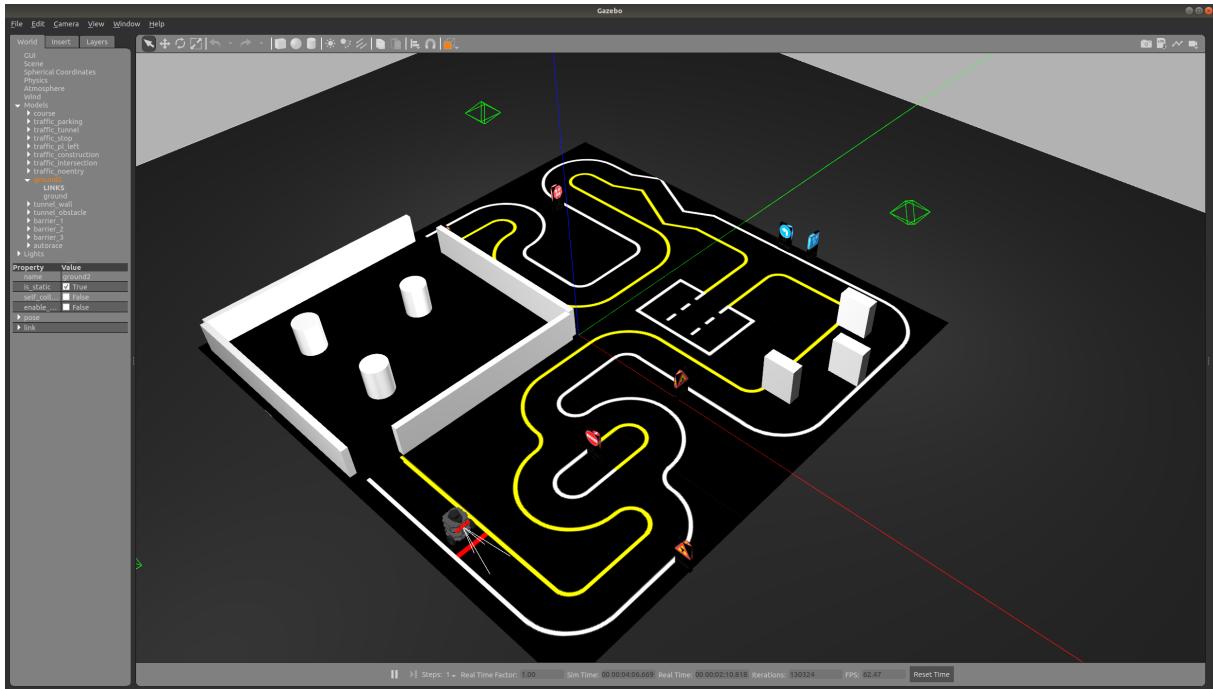


Figure 16: Gazebo 초기 실행 화면.

해당 명령어를 실행하면 `ecp_description` 패키지의 `urdf` 폴더의 `ecp.gazebo.xacro` 터틀봇 모델을 불러옴.

- ROS 내부에서 `node` 사이의 `topic`을 주고 받는 그래프를 확인하는 명령어 :

```
$ rqt_graph
```

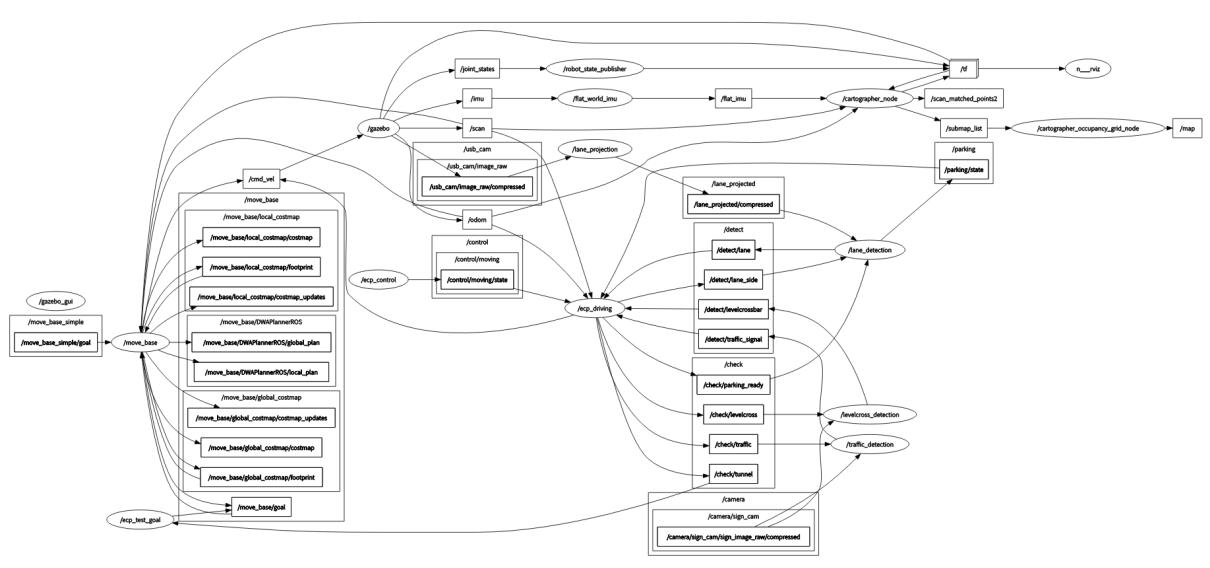


Figure 17: rqt_graph 실행 화면.

- 발행 되는 topic의 정보를 확인할 수 있는 창의 띄우는 명령어 :

```
$ rqt
```

plugin 탭에서 **topic** 포맷에 맞는 플러그인을 선택하고 확인하고 싶은 topic을 설정하면 실시간으로 해당 topic 정보를 확인 할 수 있음.

- 실시간으로 파라미터 정보를 수정하고 싶을 경우 사용하는 명령어 :

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

단, 변경하고자 하는 코드(ex. ecp_proproc의 lane_projection)의 **dynamic_reconfigure.server**의 **Server**가 지정되어 있어야함 (설명 Link).

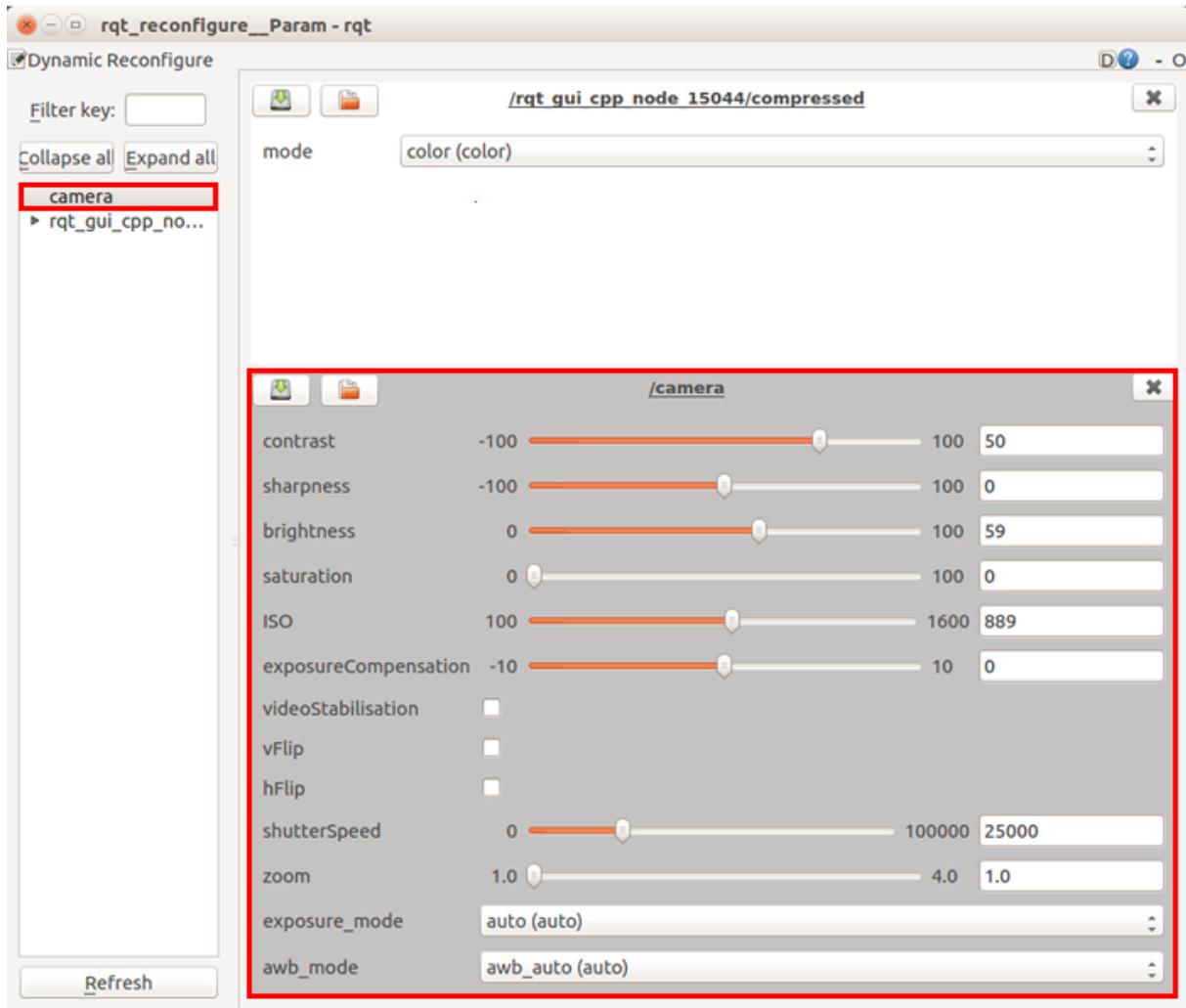


Figure 18: rqt reconfigure 실행 화면.

3.4 ECP Package

3.4.1 ecp_sensor

시뮬레이션이 아닌 Real World의 로봇에 장착된 2개의 전방 카메라와 라이다 센서의 속성과 실행을 위한 패키지. 해당 패키지 실행을 위해서는 LD-08 패키지와 usb_cam 패키지를 반드시 설치해야만 함.

패키지 내의 `config` 폴더에는 차선 인식을 위한 `lane_camera`와 이외의 전방 인식을 위한 `sign_camera`의 `yaml` 파일이 있음. `lane_camera`와 `sign_camera`는 각각 `/dev/video0`과 `/dev/video1`의 포트로 설정되어 있음.

`pixel_format`은 가장 적은 용량을 사용하는 `mjpeg` 포맷을 사용하며 너비x높이x프레임은 `320x180x10`으로 설정. 해당 설정은 JetsonNano 4GB 보드 설정이며, 보드 컴퓨터의 성능이 좋다면 해상도와 프레임을 높일 수록 좋음.

`intrinsic_controls`의 속성 중 `focus_auto`와 `exposure_auto_priority`는 모두 `false`로 설정해야 카메라 딜레이가 줄어듬. `white_balance_temperature_auto`는 `true`로 설정해야 주변 환경 조도에

따른 색 보정이 변하지 않고 일관성을 유지함.

launch 파일 실행 방법 :

```
$ rosrun ecp_sensor ecp_sensor.launch
```

[ecp_sensor.launch] 파일은 실제 카메라와 Lidar 센서의 입력을 받아들이기 때문에 원격 PC에서 실행하면 오류가 발생할 수 밖에 없음. 반드시 [SBC(Jetson Nano)]의 터미널에서 실행해야 함.

3.4.2 ecp_preproc

이 패키지는 카메라로 수신되는 영상을 각 미션에 맞게 활용하기 위해 컴퓨터 비전을 기반으로 하는 전처리를 모두 수행하는 패키지임. 차선인식([lane_cam]), 차단바([level_cross]), 표지판인식([sign_cam]), 신호등인식([traffic])을 위한 4개의 launch 파일과 해당 laucnh 파일을 한번에 실행할 수 있는 [core] 파일로 구성되어 있음.

1. 신호등인식(traffic)

- [nodes/traffic_detection] 코드로 실행.
- [/check/traffic] topic을 subscribe하며, 1이라는 정보를 받게 되면 신호등인식을 시작함.
- [cbImage] 함수에서 Figure 19(a)의 이미지 중 우측에서 40x35(WxH) 영역을 추출하여 Figure 19(b)의 형태로 변환.
- [fn_traffic_count_fixed_light] 함수에서 빨, 노, 초 색상을 필터링하고 원형 contour를 추출 함 (Figure 20(a)).
- 사전에 지정한 R,G,B 영역에 contour가 감지되면 해당 신호로 인지.
- [rqt]의 [image_view]를 통해 [/detect/traffic/compressed] topic을 확인하면 Figure 20(b)처럼 확인 가능.
- 초록 신호가 3번 이상 감지되면 [/detect/traffic_signal] topic을 publish.

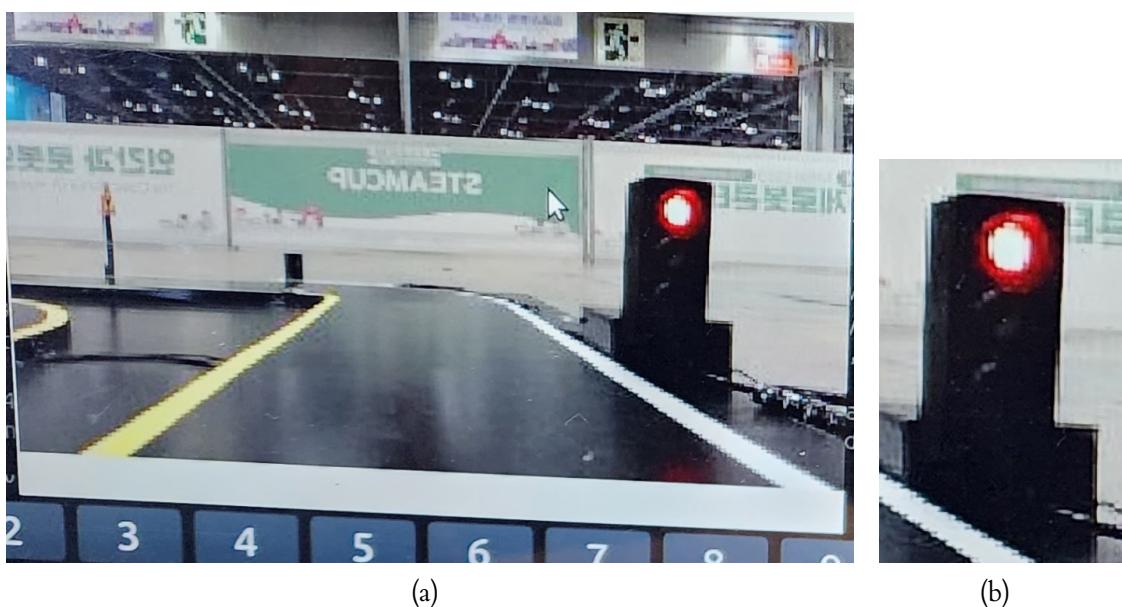


Figure 19: (a) 카메라로 입력 받은 원본 이미지. (b) cbImage에서 신호등 ROI 영역만 추출한 이미지.

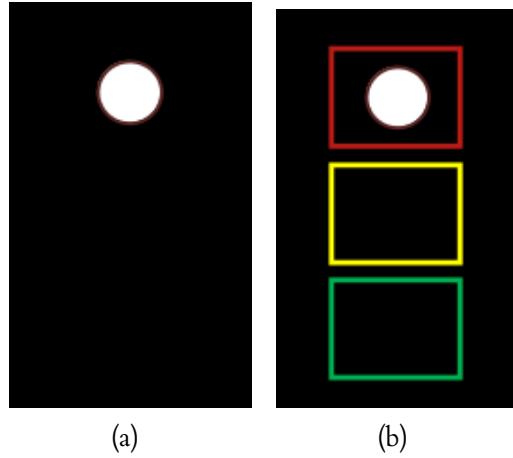


Figure 20: (a) 빨,노,초 색상 필터링 후 이미지를 흑백으로 변경. (b) 사전에 지정한 R,G,B 영역으로 디버깅 이미지 publishing.

launch 파일 실행 방법 :

```
$ roslaunch ecp_preproc ecp_preproc_traffic.launch
```

2. 차선인식(lane_cam)

- `nodes/lane_projection`과 `nodes/lane_detection` 코드로 실행.
- `lane_projection`은 `lane_camera`로 받은 이미지를 Bird-Eye-View로 변환해줌.
- calibration 폴더의 `lane_camera`의 `projection.yaml` 값을 이용하여 변환함.
- ROBOTIS e-Manual에서 수행 된 방법을 기반으로 작성 되었음 (링크 클릭 후 Kinetic 버튼 클릭).
- launch 파일 실행 시 마지막에 `mode:=debug`를 추가하면 calibration 적용 이미지도 topic으로 확인 가능 (Figure 21).

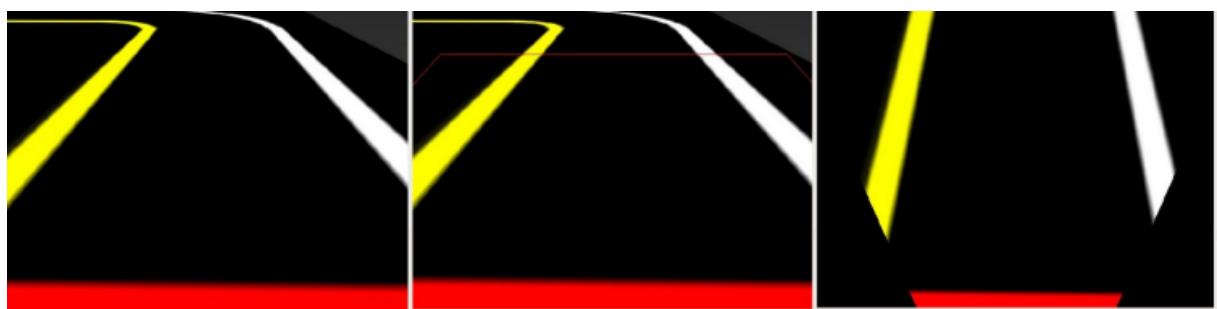


Figure 21: Lane projection 결과 rqt topic 화면. 좌측부터 원본, debug 모드, Bire-Eye-View

- `lane_detection`은 `lane_projection`으로 수행 된 Bird-Eye-View 이미지에서 차선의 중앙을 검출.
- 이미지를 grayscale 형태로 변환하고 슬라이딩 윈도우 방식으로 이미지를 위에서 부터 높이가 4인 크기로 `CHAIN_APPROX_SIMPLE` 방식의 contour를 추출함 (Figure 22(b)).
- 추출한 contour의 중심을 계산하고 해당 자리에 circle mark를 생성하고 중심 차선을

계산 (Figure 22(a)).

- 한 쪽 차선만 검출 되었을 경우 중앙으로 사전에 정의된 값만큼 shift 적용.
- 양쪽 모두 검출 되지 않으면 이전 프레임의 에러를 사용.
- 최종적으로 `ecp_prepoc`의 `msg` 폴더에 정의된 `DetectLaneInfo` 타입으로 검출한 중앙선의 아래에서 1/4 지점의 x좌표와 발견한 차선(양쪽:0, 왼쪽:1, 오른쪽:2)으로 `/detect/lane` topic publish.

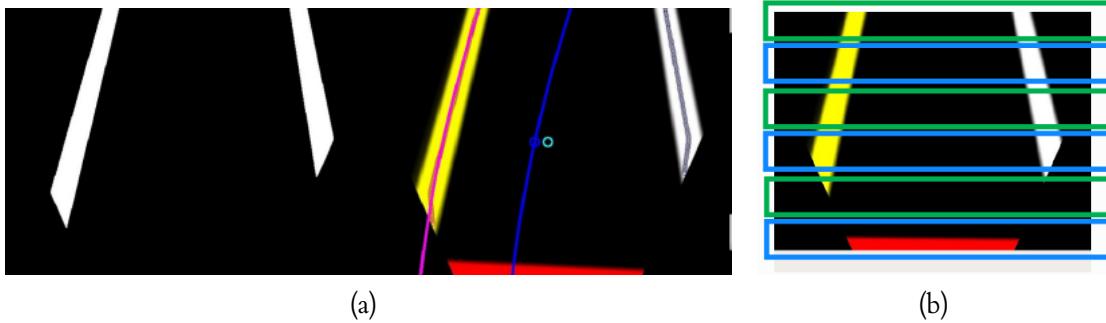


Figure 22: (a) Grayscale 변경후 중심 차선 추출 결과 rqt 확인. (b) Sliding window 방식으로 contour를 추출하는 개념(실제로 더 잘게 나누어서 추출함).

launch 파일 실행 방법 :

```
1 $ roslaunch ecp_prepoc ecp_prepoc_lane_cam.launch
```

3. 표지판 인식(sign_cam)

- `src/sign_detection.py` 코드로 실행.
- `traffic_image`에 저장된 표지판 이미지 불러오기.
- 불러온 이미지를 모두 grayscale로 변환하고 400x400 크기로 변환.
- erosion을 적용하여 노이즈 제거.
- 각 표지판마다 5도씩 최종 360까지 회전하여 HOG를 계산하여 저장.
- `K-NeighborsClassifier` 모델로 학습. - 정면 카메라로 부터 입력 받은 영상에서 빨강, 노랑, 파랑 영역을 추출.
- Contour로 영역 추출 및 너비/높이 비율이 0.8~1.2 사이인 후보군 선택.
- 선택된 후보군을 앞서 학습한 모델에 입력으로하여 표지판 인식 (Figure 23).
- 인식된 표지판은 `ecp_sign_state.py`의 상태로 `/detect/traffic` topic으로 publish.

launch 파일 실행 방법 :

```
1 $ roslaunch ecp_prepoc ecp_prepoc_sign_cam.launch
```

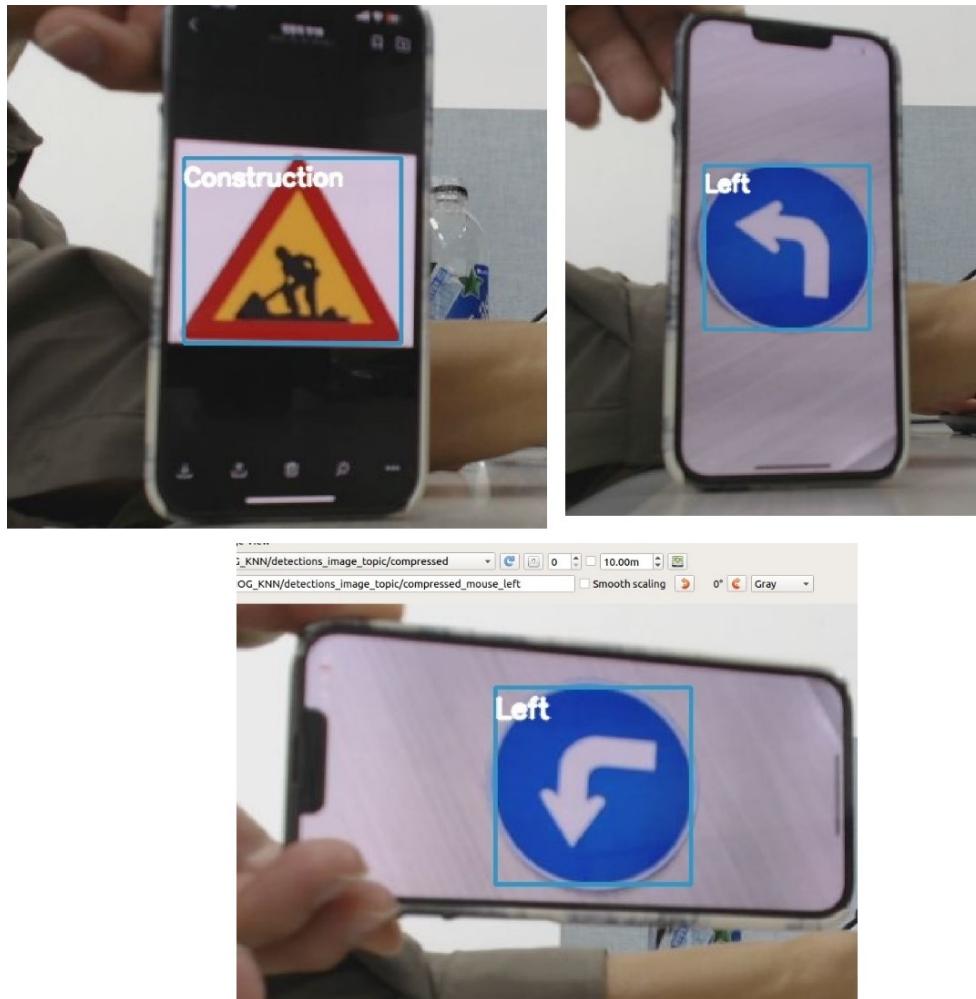
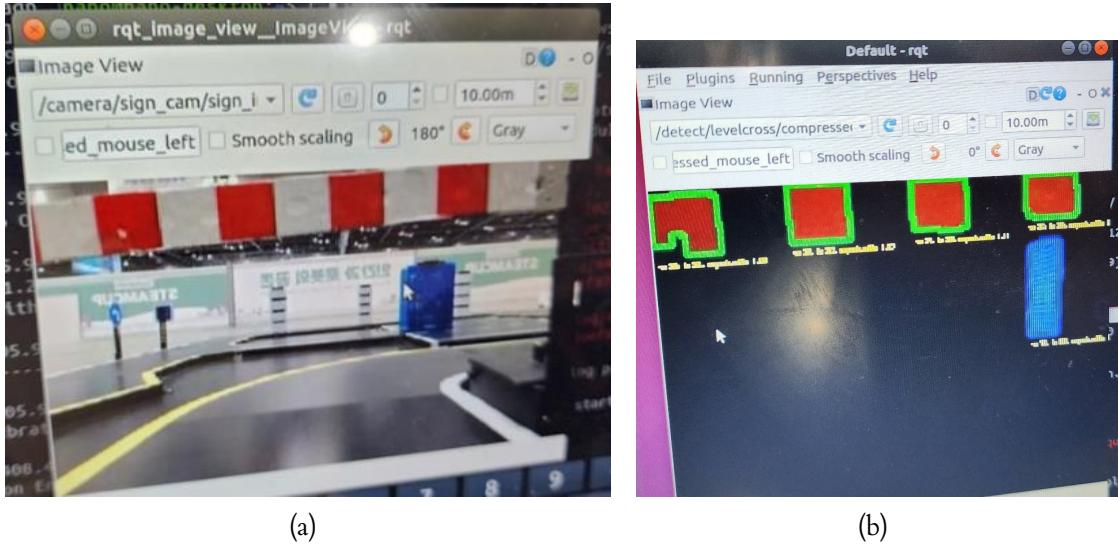


Figure 23: 표지판 인식 결과.

4. 차단바 인식(levelcross)

- `nodes/levelcross_detection` 코드로 실행.
- `/check/levelcross` topic을 subscribe하며 값이 1일 때 차단바 인식 알고리즘을 시작.
- 처리 속도를 높이기 위해 차단바 영역의 ROI를 추출.
- 빨강 영역을 추출하고 contour로 후보군 선택.
- 높이와 너비가 30 100, 너비/높이 비율이 0.5 1.5 사이인 후보군만 선택.
- 차단바 발견 시 `/detect/levelcrossbar` topic을 `True`로 publish.
- 차단바 소멸 시 `/detect/levelcrossbar` topic을 `False`로 publish.



(a)

(b)

Figure 24: (a) 카메라 입력으로 받은 차단바 원본 이미지. (b) 차단바 추출 알고리즘으로 차단바 영역 추출 결과 이미지.

launch 파일 실행 방법 :

```
1 $ roslaunch ecp_preproc ecp_preproc_levelcross.launch
```

3.4.3 ecp_description

이 패키지는 **Unified Robot Description Format(URDF)**의 파일을 저장하고 있으며, Gazebo, RViz 등에서 로봇의 각 센서 및 부품들의 모델링 구조를 추가하거나 수정할 수 있음. 우리는 Real World의 RViz에서 사용하기 위한 **ecp.urdf.xacro**와 Gazebo의 시뮬레이션 환경에서 사용하기 위한 **ecp.gazebo.xacro** 두개의 파일이 있음. 로봇의 모델링 파일은 ROBOTIS에서 제공하는 **turtlebot3_description**에서 정보를 가져오므로 해당 패키지를 지우면 작동하지 않음.

ROBOTIS에서 제공하는 **urdf** 파일은 카메라 센서를 제공하지 않기 때문에 Gazebo 환경에서 사용하기 위해서는 필요한 카메라 센서를 urdf에서 수정하여 추가해야함. 우리는 아래와 같이 차선 주행을 위한 카메라만 추가 하였음. 각 태크의 값을 조절하여 원하는 카메라 센서로 변경할 수 있음.

```

1 <gazebo reference="usb_cam_link">
2   <material>Gazebo/Red</material>
3   <sensor name="camera" type="camera">
4     <pose>0 0 0 0 0</pose>
5     <visualize>true</visualize>
6     <update_rate>30</update_rate>
7     <camera>
8       <horizontal_fov>1.089</horizontal_fov>
9     </camera>
10    <image>
11      <format>MJPG</format>
12      <width>640</width>
13      <height>480</height>
14    </image>
```

```

14         <clip>
15             <near>0.05</near>
16             <far>8.0</far>
17         </clip>
18     </camera>
19     <plugin name="camera_controller" filename
20      ="libgazebo_ros_camera.so">
21         <cameraName>usb_cam</cameraName>
22         <imageTopicName>image_raw</imageTopicName
23     >
24         <cameraInfoTopicName>camera_info</
25         cameraInfoTopicName>
26             <frame_name>camera_link_optical</
27             frame_name>
28             <min_depth>0.1</min_depth>
29             <max_depth>10.0</max_depth>
30         </plugin>
31     </sensor>
32 </gazebo>

```

3.4.4 ecp_slam

이 패키지는 Lidar 센서로 Simultaneous Localization and Mapping(SLAM)으로 지도를 생성함. 우리는 Gmapping, Cartographer 두 가지 방법을 사용해 보았으며, cartographer가 성능이 월등히 높아 기본값으로 설정해두었음. ROBOTIS에서는 karto, hector, frontier_exploration과 같은 다른 방법도 제공하고 있음(Link).

- nodes/ecp_test_goal의 코드로 작동함.
- Gmapping은 내장되어 있기 때문에 바로 실행 가능하지만 Cartographer는 추가적인 설치 이후에 사용 가능(설치 Link).
- /check/tunnel topic을 subscribe하며, topic을 callback 받게 되면 가고자 하는 위치 및 로봇의 방향을 MoveBaseActionGoal msg 포맷으로 publish.
- 로봇은 SLAM을 가장 처음 실행 시킨 위치를 (x:0,y:0,z:0)로 저장하고 있음.
- 위치 값은 실행 시킨 초기 위치를 기준으로 상대적인 값을 가지게 됨.
- ecp_slam.launch 파일의 rviz 하단의 코드를 주석해제하면 launch 파일을 실행함과 동시에 설정된 Rviz로 자동으로 실행.
- 단, 모니터가 없는 (ex. Jetson Nano를 화면과 연결하지 않은 경우) 상황에서는 RViz를 실행하면 오류가 발생되어 자동으로 종료됨.

```
1 $ rosrun ecp_slam ecp_slam.launch
```

3.4.5 ecp_navigation

이 패키지는 SLAM을 통해 생성된 map을 기준으로 로봇이 입력 받은 목적지 까지 자동으로 장애물을 회피하며 길 찾기를 수행함. param 폴더의 각 yaml 파라미터를 변경함으로써 알고리즘을 최적화 할 수 있음. 우리는 global_planner로 D* Lite 알고리즘(설치 Link)을 사용하였음 (Figure 25). 실행하기 위한 별도의 명령어는 존재하지 않음.

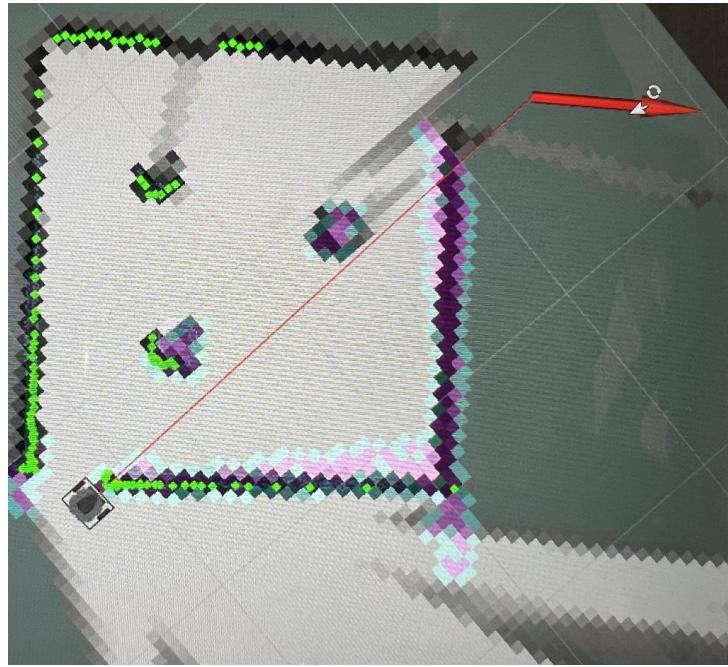


Figure 25: Gazebo에서 SLAM 실행 후 RViz에서 Navigation 실행 화면.

3.4.6 ecp_control

이 패키지는 앞서 언급한 모든 패키지들을 관리하고 로봇의 상태 변수들을 제어함. `src` 폴더 내부의 `ecp_control.py`는 표지판을 인식을 통한 미션과 로봇의 상태를 통해 현재 상태를 결정함. `ecp_driving.py`는 `ecp_control.py`에서 받은 상태를 통해 `Finite State Machine(FSM)` 기반으로 미션을 수행함. `ecp_state.py`와 `ecp_sign_state.py`는 `ecp_control.py`에서 사용되는 미션과 로봇의 상태 변수를 enum 형태로 저장한 파일.

1. ecp_sign_state.py

- Tabel 1과 같이 총 9가지의 상태로 구분 됨.
- `ecp_control.py`에서 `/detect/traffic_sign` topic을 통해 `sign_state`가 변경 됨.

SIGN_STATE	표지판	값
CONSTRUCTION	장애물	0
FORBID	접근금지	1
LEFT	좌회전	2
PARKING	주차	3
RIGHT	우회전	4
STOP	정지	5
TUNNEL	터널	6
INTERSECTION	교차로	7
NONE	기본값	8

Table 1: ecp_sign_state.py

2. ecp_state.py

- Tabel 2와 같이 총 8가지의 상태로 구분 됨.
- `ecp_control.py`에서 `sign_state`와 현재 미션 완료 상태를 고려하여 `ecp_state`가 변경 됨.

STATE	의미	값
NORMAL	기본값(일반주행)	0
INTERSECTION	교차로(방향 확인 전)	1
INTERSECTION_LEFT	교차로(좌회전)	2
INTERSECTION_RIGHT	교차로(우회전)	3
OBSTACLE	장애물 미션	4
PARKING	주차 미션	5
LEVELCROSS	차단바 미션	6
TUNNEL	터널 미션	7

Table 2: ecp_state.py

3. ecp_driving.py

- 각 미션별 상태 천이도는 Figure 26와 같다.

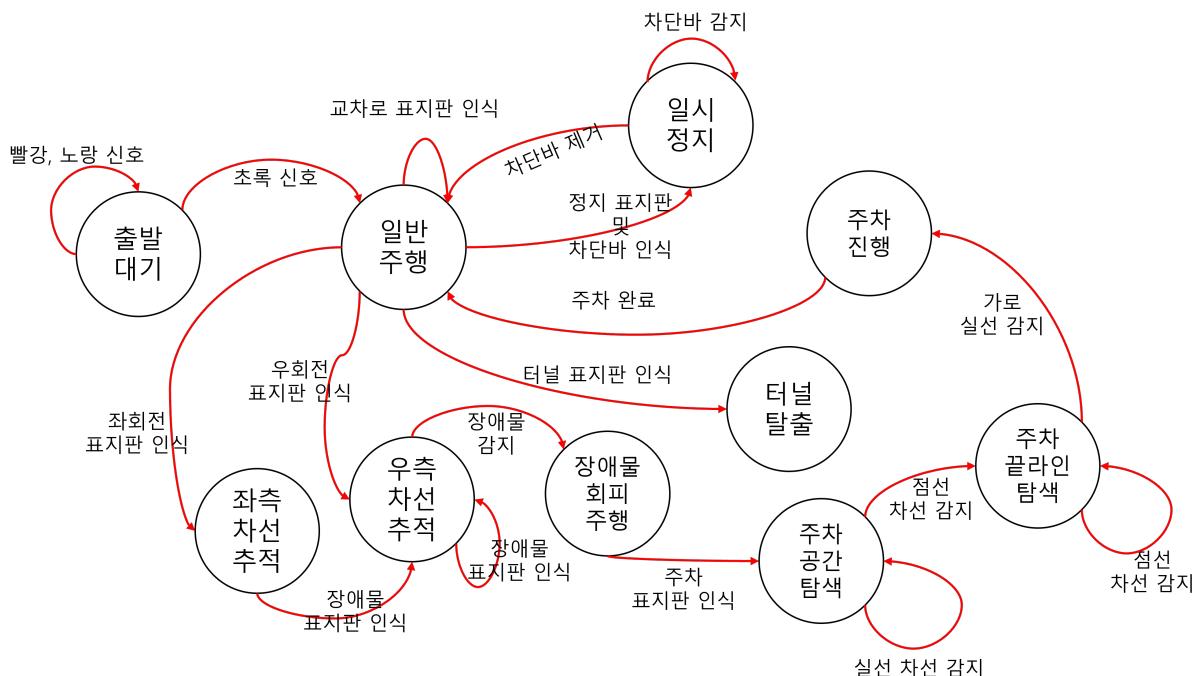


Figure 26: 로봇 미션 수행 FSM 구조.

4. NORMAL

- 출발 준비 상태일 경우 초록 불이 감지 되기전까지 대기.
- 출발 후 PD Control 기반의 차선 주행 진행.

5. INTERSECTION

- 교차로 표지판 인식 후 일반 PD Control 주행 진행.

6. INTERSECTION LEFT or RIGHT

- 좌/우회전 표지판 발견시 해당 표지판에 맞는 차선을 따라 추적 주행.
- 차선이 보이지 않을 시 좌/우 표지판에 따라 해당 방향으로 궤도 주행.

7. OBSTACLE

- 장애물 표지판 발견시 우측 차선 추적 주행.
- 장애물 감지시 순서에 따라 장애물 회피 후 차선 추적 주행.
- 장애물 회피를 위해서 Lidar를 통해 입력 받은 360도의 거리 값 중 전방 -30 30 사이의 scan 값 중 5cm 이하의 데이터 개수 측정.
- 5cm 이하의 데이터가 12~28개의 경우 현재 추적하는 차선과 반대반향으로 장애물 기준으로 궤도 주행.

8. PARKING

- 주차 표지판 인식시 좌측 차선 추적 주행 및 `/check/parking_ready` topic publish.
- `ecp_preproc`의 `nodes/lane_detection`에서 점선 및 주차 끌라인 찾는 알고리즘 수행 및 topic publish.
- 점선 차선 발견시 주차 끌라인 발견할 때 까지 주행.
- 빈 주차공간 탐지 후 주차 수행 및 주차 완료 후 일반 주행.
- Lidar를 통해 좌/우측의 3cm 이하의 데이터 개수 측정.
- 빈 주차 공간 앞에서 0.4초 회전 후 0.8초 직진.
- 0.1초 정지 후 0.8초 후진 및 동일한 방향으로 0.4초 회전.
- 주차 완료 `/parking/state` topic publish.

9. LEVEL CROSS

- 차단바 인식 시 일시 정지.
- 차단바 제거 시 일반 주행.

10. TUNNEL

- 터널 표지판 발견 시 터널 탈출 진행.
- Navigation 알고리즘을 통해 출발 지점으로 탈출 진행.

launch 파일 실행 방법 :

```
$ roslaunch ecp_control ecp_control.launch
```

3.4.7 ecp_bringup

이 패키지는 `description`에서 로봇에 실행되기 위해 필요한 패키지로 수정할 필요 없음.

3.5 Demo Scenario

3.5.1 실제 환경 Launch 파일 실행 순서

SBC(Jetson Nano)에서 모든 파일을 실행하고자 할 때 터미널 1 (SBC) :

```
1 $ roscore
```

터미널 2 (SBC) :

```
1 $ rosrun ecp_sensor ecp_sensor.launch
```

터미널 3 (SBC) :

```
1 $ rosrun ecp_preproc ecp_preproc_core.launch
```

터미널 4 (SBC) :

```
1 $ rosrun ecp_slam ecp_slam.launch
```

터미널 5 (SBC) :

```
1 $ rosrun ecp_control ecp_control.launch
```

Remote PC 터미널 1 (Optional, RViz 확인) :

```
1 $ rosrun ecp_slam ecp_slam.launch
```

3.5.2 Gazebo 환경 Launch 파일 실행 순서

.bashrc 파일의 wifi 아이피 설정 주의 터미널 1 (Remote PC) :

```
1 $ roscore
```

터미널 2 (Remote PC) :

```
1 $ rosrun turtlebot3_gazebo  
turtlebot3_autorace_2020.launch
```

터미널 3 (Remote PC) :

```
1 $ rosrun ecp_preproc ecp_preproc_core.launch  
gazebo:=on
```

터미널 4 (Remote PC) :

```
1 $ rosrun ecp_slam ecp_slam.launch gazebo:=on
```

터미널 5 (Remote PC) :

```
1 $ rosrun ecp_control ecp_control.launch
```

3.5.3 Debug 적용하여 rqt_reconfigure 사용을 원할 때

```
1 $ rosrun ecp_preproc ecp_preproc_core.launch  
debug:=on
```