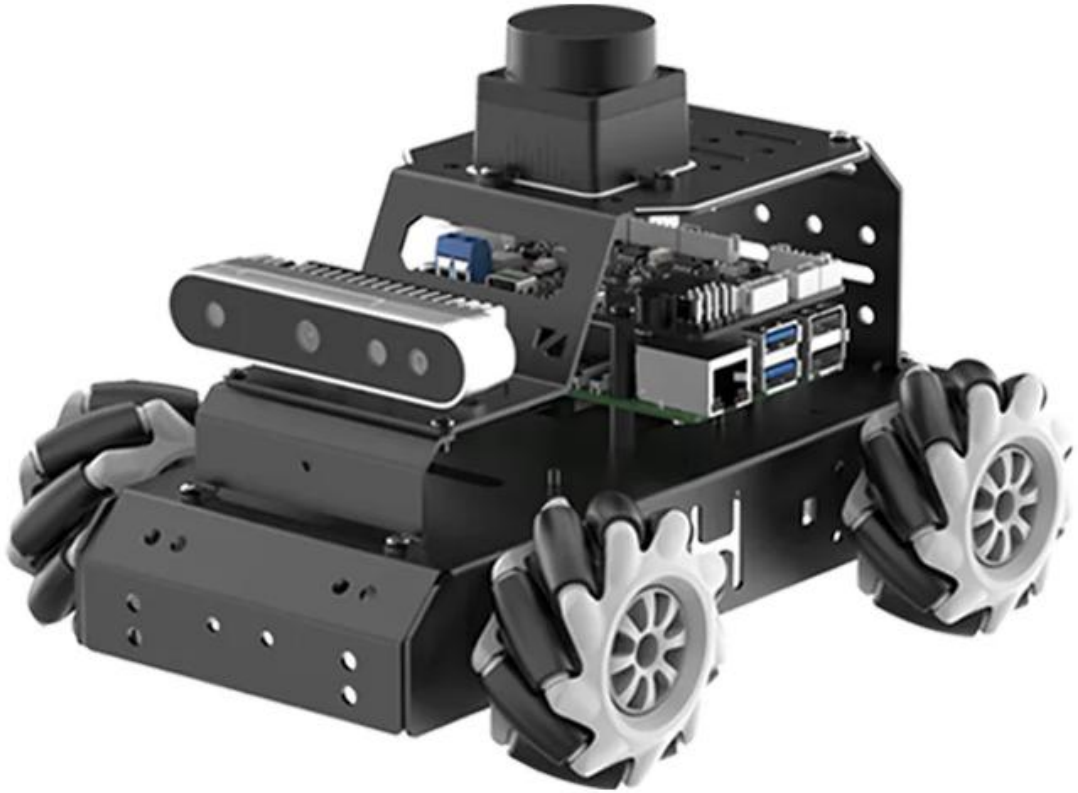
AI TRAINING SERIES

자율주행 Competition

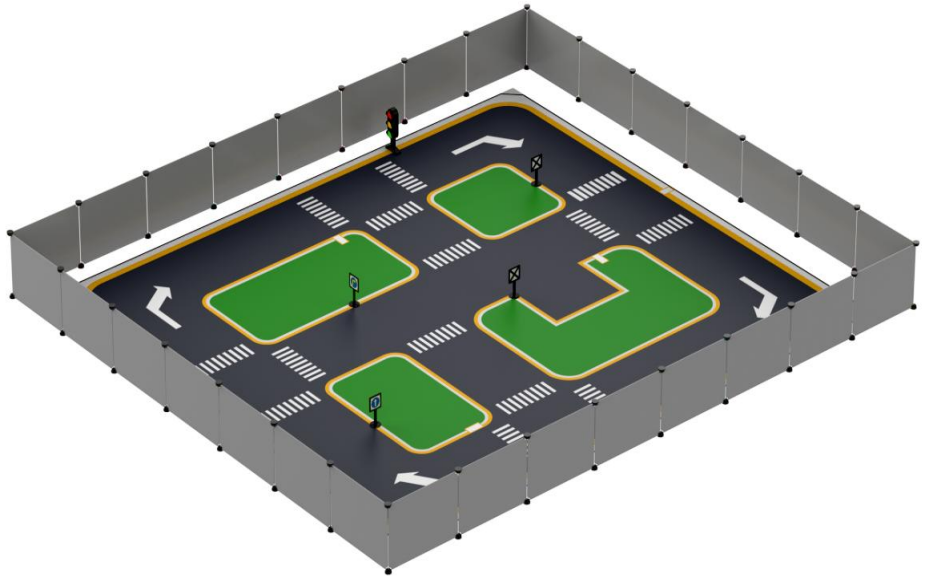
Contents



Ⅰ. 팀별 자율 주행 미션 수행

Ⅰ. 팀별 자율 주행 미션 수행

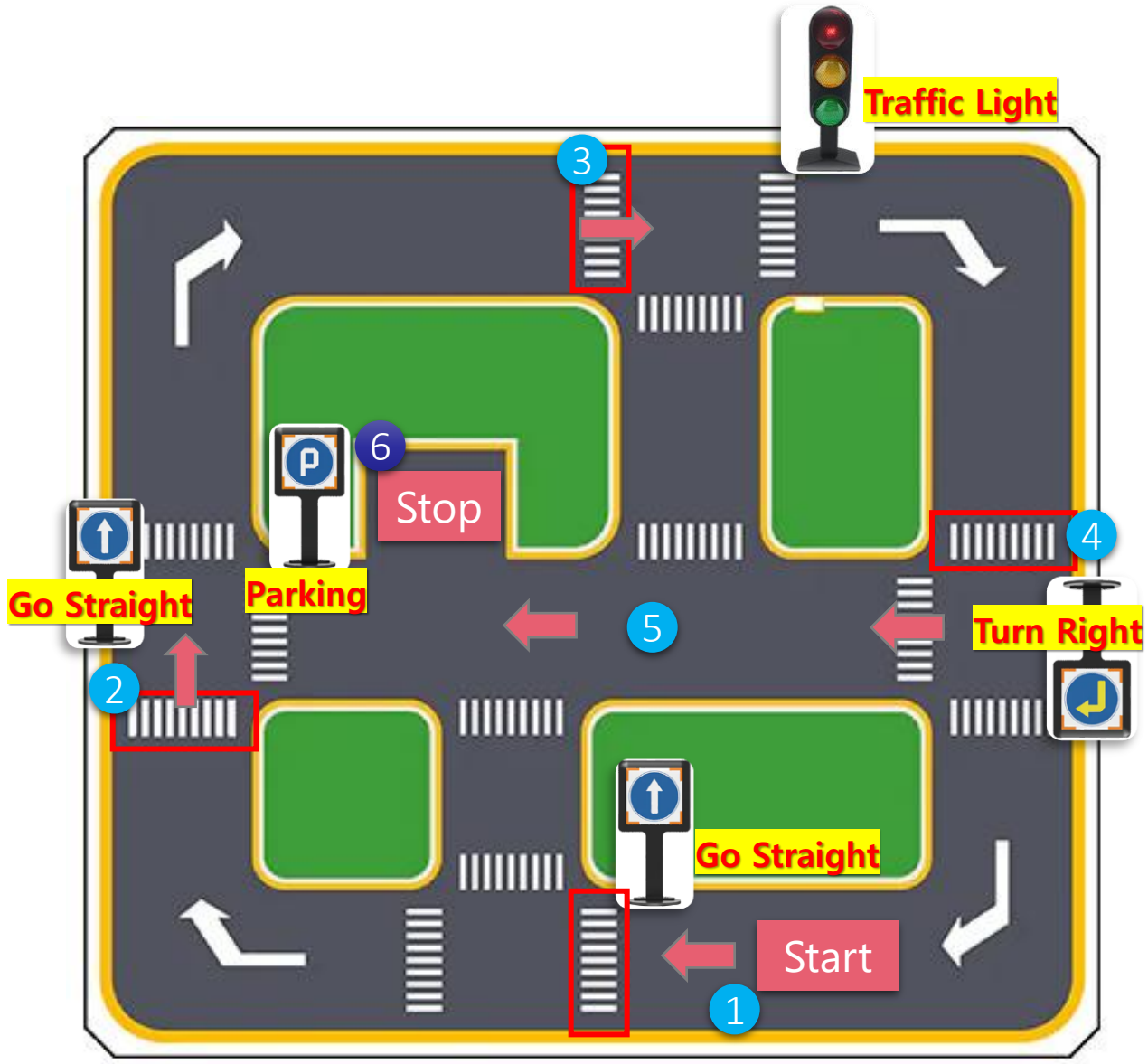
- 최종 프로젝트
  - 교육용 자동차 조립
  - 자율 주행 맵에서 미션 수행



- Ground Rules
  - 팀별 프로젝트를 위한 Github Repo 사용
    - README.md 작성
    - 기능별 폴더 나누기 (camera, motor, ...)
    - Branch 전략 및 PR
    - 가산점
      - CI/CD 구현
      - 기능별 Unittest 구현
      - HLD + UML 다이어그램 활용

Ⅰ. 팀별 자율 주행 미션 수행 (cont,)

- 로봇 규정
  1. 지정된 로봇 구성품 으로만 구현된 로봇으로 자율 주행을 수행 한다.
  2. 로봇은 경연중에 경연자 혹은 외부와 유, 무선을 이용한 모든 통신은 금지 된다.
- 주행 규정
  1. 정해진 도로 폭 내에서만 주행 해야 된다. **(-10점)**
  2. 출발 신호는 심판이 결정 하고, 신호 후 10초 동안 로봇의 자율적인 움직임이 없으면 감점 처리 된다. **(-10점)**
  3. 경연자는 로봇에 부착된 스위치를 이용하여 출발 신호를 로봇에게 전달 한다. **(-5점)**
  4. 로봇이 움직일 땐 녹색LED를 ON/빨간색 LED를 OFF 해야 하며, 로봇이 멈춰 섰을 땐, 녹색 LED를 OFF/빨간색 LED를 ON 해야 한다. **(-5점)**
  5. LED Control 을 빵판을 사용해서 Raspberry ROS <-> Host 통신으로 구현 **(+15점)**
  6. 횡단보도(빨간색 네모표시한 횡단보도만 해당) 앞에선 로봇은 반드시 정지 했다가 출발한다. **(-10점)**
  7. 도로에 적혀진 화살표를 인식 한 후 해당 방향의 노란색 LED 점멸을 반복 하여 해당 방향으로 주행 할 것임을 알린다. **(-10점)**
  8. 교차로에선 표지판을 인식하여 해당 방향으로 주행 해야 한다. **(-10점)**
  9. 신호등을 인식하면 우선 멈추고 해당 신호에 맞게 주행 한다. **(-10점)**
  10. 정지 및 주차 구간에선 정확한 라인 안에 주차가 되어야 하며 완료된 의미로 모든 LED를 점멸 한다. **(-10점)**
  11. Start - Stop 시간이 가장 빠른 팀에게 1등, 2등, 3등에게 등수별 차등 가산점 **(+15/+10/+5점)**



Ⅱ. 우승팀 발표 및 강사 총평

Ⅱ. 우승팀 발표 및 강사 총평

Ⅲ. 교육 내용 종합 정리

Ⅲ. 교육 내용 종합 정리

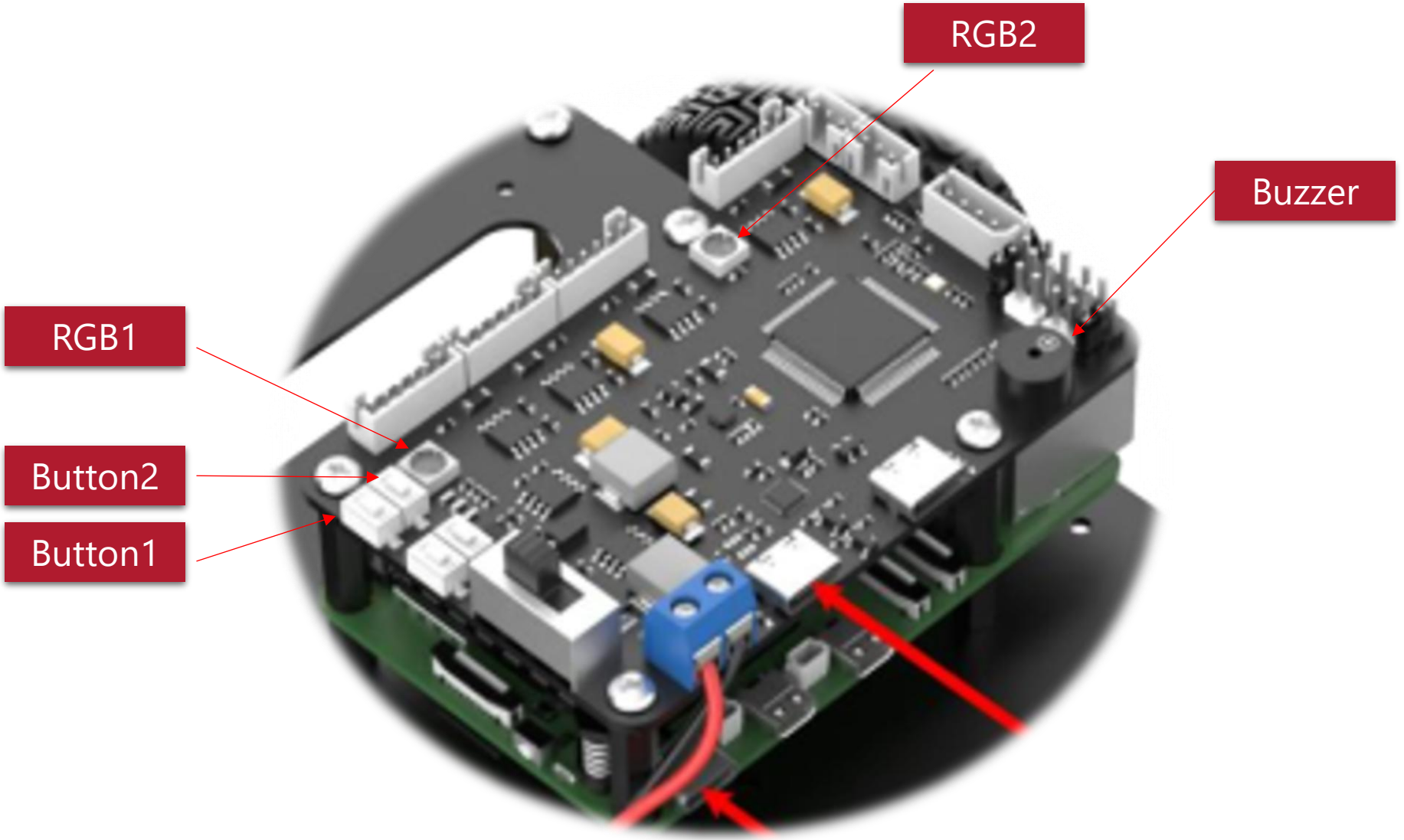
- Week 1: Embedded 시스템 기초 및 개발 보드 활용 입문
  - 라즈베리파이 기본 설정 및 Linux와 친숙해지기
  - Python으로 GPIO를 제어(LED ON/OFF) 하고 오실로스코프를 활용한 파형 분석

- Week 2: 센서 데이터 처리 및 ROS2 활용
  - ROS2 란 무엇인가와 Docker 기반의 환경 이해
  - Depth Camera, LiDAR, Servo Motor, OpenCV, ML 등을 응용한 다양한 경험

- Week 3: 자율 주행 Competition
  - Lane Keeping/Road Sign Detection/Traffic Light Recognition 등 어플리케이션 활용
  - 팀 프로젝트 수행

THANK YOU

Tips



Tips - RGB LEG

ROS2 docker -> /home/ubuntu/RRCLite_demo/rgb_controller_node.py

```python
class RGBControllerNode(Node):
    def __init__(self):
        super().__init__('rgb_controller')
        self.publisher_ = self.create_publisher(RGBStates, '/ros_robot_controller/set_rgb', 10)
        self.timer = self.create_timer(2.0, self.timer_callback)
        self.get_logger().info('RGB Controller Node has been started.')
        self.current_color_index = 0

    def timer_callback(self):
        # Define a set of colors
        colors = [
            (255, 0, 0),    # Red
            (0, 255, 0),    # Green
            (0, 0, 255),    # Blue
            (255, 255, 0),  # Yellow
            (0, 255, 255),  # Cyan
            (255, 0, 255)   # Magenta
        ]

        # Change the color for both LEDs
        color = colors[self.current_color_index % len(colors)]
        self.current_color_index += 1

        msg = RGBStates()
        msg.states = [
            RGBState(index=1, red=color[0], green=color[1], blue=color[2]),
            RGBState(index=2, red=color[0], green=color[1], blue=color[2])
        ]

        self.publisher_.publish(msg)
```

Tips - Button

ROS2 docker -> /home/ubuntu/RRCLite_demo/button_node.py

```python
class ButtonPressReceiver(Node):

    def __init__(self, name):
        super().__init__(name)
        self.create_subscription(ButtonState, '/ros_robot_controller/button', self.button_callback, 10)
        self.get_logger().info('ButtonPressReceiver node started')

    def button_callback(self, msg):
        if msg.id == 1:
            self.process_button_press('Button 1', msg.state)
        elif msg.id == 2:
            self.process_button_press('Button 2', msg.state)

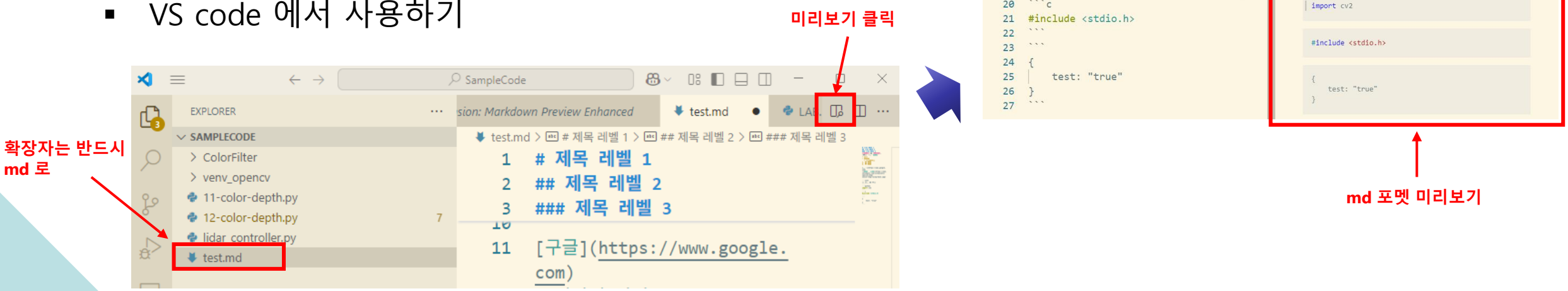
    def process_button_press(self, button_name, state):
        if state == 1:
            self.get_logger().info(f'{button_name} short press detected')
            # You can add additional logic here for short press
        elif state == 2:
            self.get_logger().info(f'{button_name} long press detected')
            # You can add additional logic here for long press
```

Tips - Markdown?

- What is Markdown?
  - 2004년경 블로그 에디터들이 HTML 태그 작성이 번거로워 읽기와 쓰기 모두 편한 포멧이 필요했고 이를 위한 가독성 높은 평문 구조로 작성하면서 HTML 로 쉽게 변환이 가능한 포멧을 만듬

- 어디에서 사용?
  - 블로그 포스트
  - 깃 허브 README, 오픈소스 문서화
  - 논문 초안, 기술문서 (Jupyter Notebook 등)

- VS code 에서 사용하기



Tips - Markdown Syntax

- 제목 (Heading)

```
# 제목 레벨 1
## 제목 레벨 2
### 제목 레벨 3
#### 제목 레벨 4
```

- 강조(Emphasis)

```
**두껍게(Bold)**
*기울임(Italic)*
~~취소선(Strikethrough)~~
```

- 목록(List)
  - 순서 없는 목록: -, *, +
  - 순서 있는 목록: 숫자 + 점

```
- 사과
- 바나나
  - 서브 아이템
1. 첫 번째
2. 두 번째
```

- 링크 및 이미지

```
[구글로 이동](https://www.google.com)
![대체 텍스트 - 강아지](https://example.com/dog.jpg)
```

- 코드 표현
  - 인라인 코드: `코드`
  - 블록 코드:

```
```언어
// 여러 줄 코드 작성
```

````
# 제목 레벨 1
## 제목 레벨 2
### 제목 레벨 3
**강조** 와 *기울임*, ~~삭제선~~,
- 사과
- 바나나
  - 서브아이템
1. 첫 번째
2. 두 번째

[구글링크](https://www.google.com)

![강아지
사진](https://www.vets4pets.com/siteassets/species/dog/senior-dog-lying-down.jpg)

한줄코드 `export TEST`

```언어
// 여러 줄 코드
```

```python
import cv2
```

```c
#include <stdio.h>
```

```json
{
    test: "true"
}
```
````

제목 레벨 1

제목 레벨 2

제목 레벨 3

강조 와 기울임, 삭제선,

- 사과
- 바나나
  - 서브아이템

1. 첫 번째
2. 두 번째

구글링크

한줄코드 `export TEST`

```
// 여러 줄 코드
```

```
import cv2
```

```
#include <stdio.h>
```

```
{
    test: "true"
}
```
```

Tips - README.md Sample

```
# 🚗 AutoDrive-X
[![License](https://img.shields.io/badge/license-MIT-blue)]()

> **한 줄 소개**: 자율주행 핵심 기능(인식·제어·경로 계획)을 통합한 팀 프로젝트

---

## 📋 목차
1. [프로젝트 개요](#프로젝트-개요)
2. [주요 기능](#주요-기능)
3. [아키텍처](#아키텍처)
4. [설치 및 실행](#설치-및-실행)
5. [예시 사용법](#예시-사용법)
6. [데이터 & 모델](#데이터--모델)
7. [팀원](#팀원)
8. [라이선스](#라이선스)

---

## 📝 프로젝트 개요
- **목표**: ROS 기반 자율주행 파이프라인 설계
- **스택**: Python, ROS2, OpenCV, YOLO, Docker

---

## 🚀 주요 기능
- **차선 인식**: OpenCV + Depth Map
- **객체 감지**: YOLOv5

---

## 🏗️ 아키텍처
![Architecture Diagram](./docs/architecture.png)
1. 센서 드라이버 (RGB 카메라, Depth 카메라)
2. 인식 노드 → 토픽 퍼블리시
3. 제어 노드 → 액추에이터 명령

---

## ⚙️ 설치 및 실행

### 1) 사전 준비
- Raspberrypi 5 + raspberrypi OS
- Docker + ROS2 humble

### 2) 실행 방법
```bash
# How to run ?
```
```

🚗 AutoDrive-X

license MIT

> **한 줄 소개**: 자율주행 핵심 기능(인식·제어·경로 계획)을 통합한 팀 프로젝트

📋 목차

1. 프로젝트 개요
2. 주요 기능
3. 아키텍처
4. 설치 및 실행
5. 예시 사용법
6. 데이터 & 모델
7. 팀원
8. 라이선스

📝 프로젝트 개요

- **목표**: ROS 기반 자율주행 파이프라인 설계
- **스택**: Python, ROS2, OpenCV, YOLO, Docker

🚀 주요 기능

- **차선 인식**: OpenCV + Depth Map
- **객체 감지**: YOLOv5

🏗️ 아키텍처

Architecture Diagram

1. 센서 드라이버 (RGB 카메라, Depth 카메라)
2. 인식 노드 → 토픽 퍼블리시
3. 제어 노드 → 액추에이터 명령

⚙️ 설치 및 실행

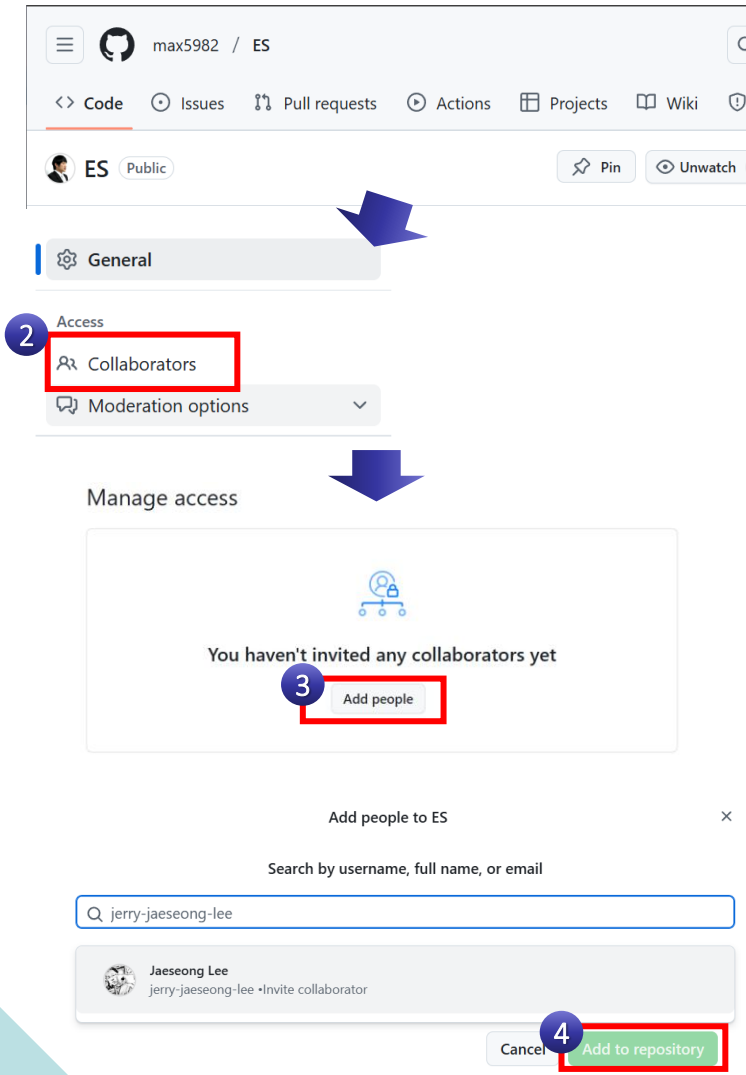
1) 사전 준비

- Raspberrypi 5 + raspberrypi OS
- Docker + ROS2 humble

2) 실행 방법

```bash
# How to run ?
```

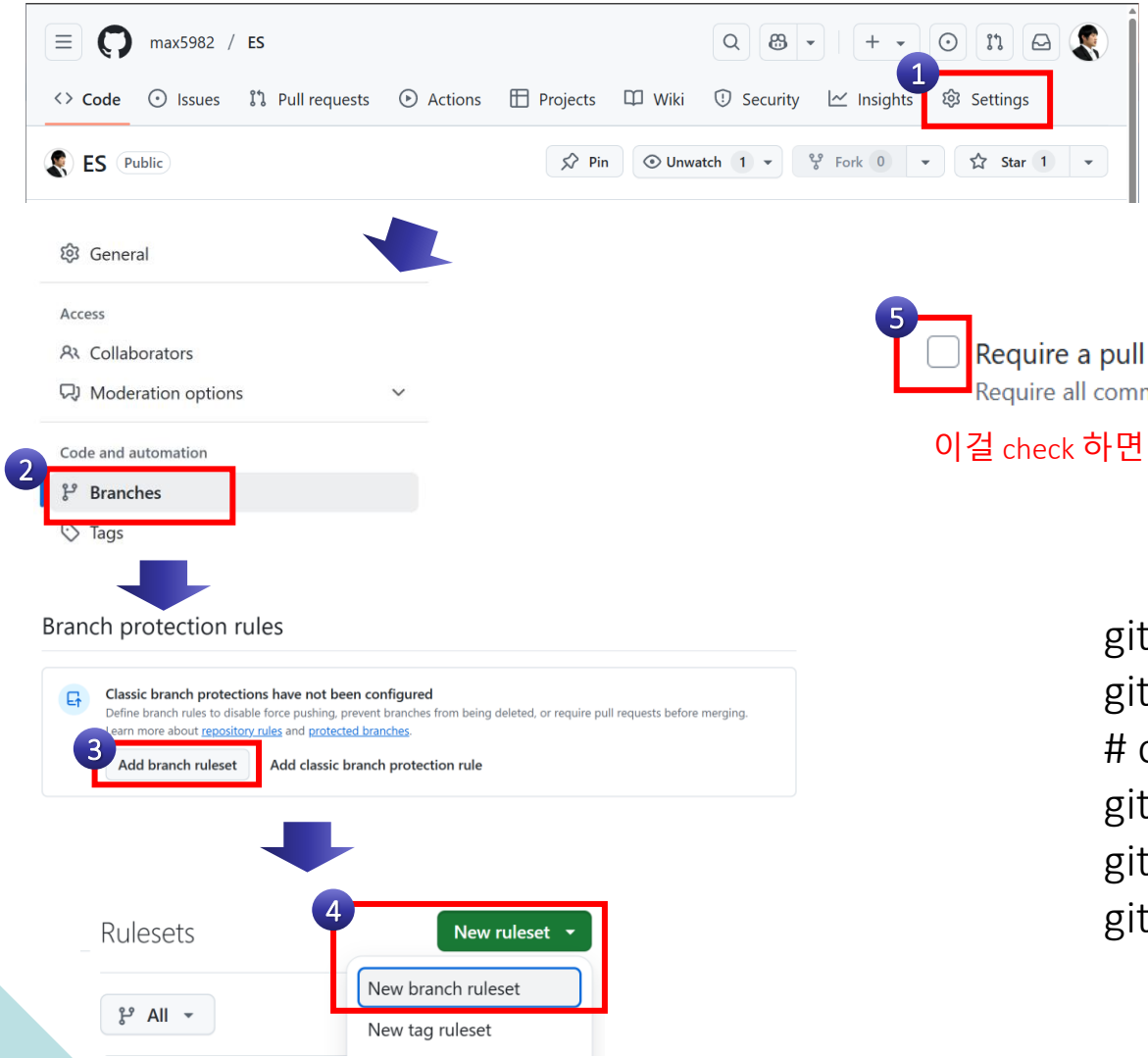
Tips - Github Collaborator 추가하기

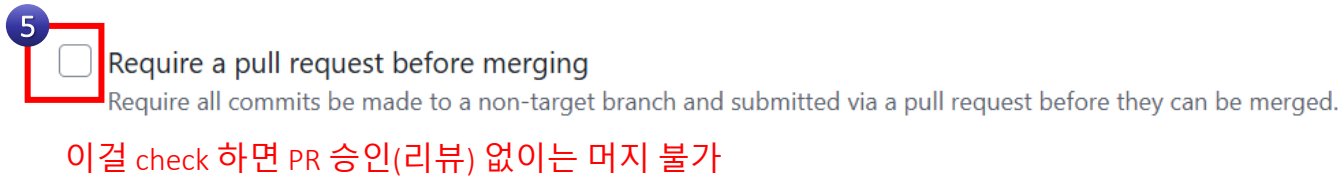


Note: github 유료 사용인 경우 아래와 같이 사용자별 권한 설정 가능

Role	설명	사용 예시
Read	코드 보기·이슈 열람만 가능	문서 리뷰어, QA 담당자
Triage	이슈/PR 분류·레이블링 가능	프로젝트 매니저
Write	코드 푸시·PR 생성·머지 가능	일반 개발자
Maintain	브랜치 보호·설정 변경 가능	시니어 개발자, 섹션 리더
Admin	레포 설정·협업자 관리·삭제 등 모든 권한	프로젝트 오너

Tips - Github Branch Protection





이걸 check 하면 PR 승인(리뷰) 없이는 머지 불가

```
git branch new-branch-name
git checkout new-branch-name
# code change
git add .
git commit -m “comment….”
git push -u origin new-branch-name
```