

생성형 AI 사용 보고서

20기 예비단원 우정윤

강의 자료상에는 publisher와 subscriber 코드가 하나의 cpp 파일로 이루어져 있는 것처럼 보였으나, ROS 설치 페이지에 나와 있는 테스트 통신 코드를 보면,

Try some examples

Talker-listener

If you installed `ros-humble-desktop` above you can try some examples.

In one terminal, source the setup file and then run a C++ `talker`:

```
$ source /opt/ros/humble/setup.bash
$ ros2 run demo_nodes_cpp talker
```

In another terminal source the setup file and then run a Python `listener`:

```
$ source /opt/ros/humble/setup.bash
$ ros2 run demo_nodes_py listener
```

다음 사진과 같이 테스트 하는 것을 알 수 있고 이를 생각해 보았을 때에 publisher와 subscriber 노드가 각각 다른 파일에 저장되어야 하는 것이 아닌가 라는 의문이 들었고 이를 생성형 AI(chat gpt)를 통해 의문을 해결하였고 헤더파일과 소스파일 모두 publisher 노드와 subscriber 노드로 나누는 것이 맞다는 답변을 통해 과제의 구조를 정립할 수 있었다.

코드랑 헤더파일을 퍼블리셔와 섀스크라이버랑 나누는게 좋아 합치는데 좋아?
강의 자료에서는 한개로 이루고 있는데
과제할때 나누는게 유리한지 합치는데 유리한지

아주 좋은 질문이에요 🙌

"퍼블리셔/섀스크라이버를 한 파일에 합칠지, 각각 나눌지"는 상황에 따라 장단점이 달라집니다.

• 1. 한 파일(한 클래스/한 노드) 안에 합치는 경우

(강의 자료처럼 `my_cpp_node.cpp` 안에 퍼블리셔+섀스크라이버 기능을 모두 구현)

장점

- 코드가 간단하고 파일 수가 적음 → 과제 제출 시 관리가 편함
- 한 노드 안에서 `publish`와 `subscribe` 동작을 동시에 보여줄 수 있음
- 테스트할 때 터미널을 두 개 안 열어도 됨 (노드 하나만 띄워도 동작 가능)

단점

- 퍼블리셔/섀스크라이버 기능이 섞여 있어서 코드가 길어지고 복잡해짐
- 나중에 확장(예: 퍼블리셔만 따로 쓰고 싶거나, 섀스크라이버만 따로 띄우고 싶을 때)이 불편

• 2. 파일/노드를 분리하는 경우

(`publisher_node.cpp`, `subscriber_node.cpp` 따로 작성)

장점

- 역할이 분리되어 코드가 명확하고 깔끔함
- 실제 ROS2 프로젝트에서 권장되는 방식 (대부분 패키지에 여러 개의 실행파일을 둡니다)
- 독립적으로 실행 가능 → 테스트하기 편리 (예: C++ 퍼블리셔 ↔ Python 섀스크라이버 교차 통신 테스트)

단점

- 파일 수가 늘어나서 관리가 조금 번거로움
- 아주 단순한 과제에서는 "쓸데없이 쪼갬"은 느낌을 줄 수도 있음

• 과제 기준에서 추천

- 이번 과제는 "C++ ↔ Python 교차 통신"이 핵심 → `publisher`와 `Subscriber`를 분리하는 게 유리합니다.
- 그나마 C++ 퍼블리셔 ↔ Python 섀스크라이버, Python 퍼블리셔 ↔ C++ 섀스크라이버를 쉽게 시험할 수 있어요.