<RO:BIT ROS2 - 1일차 과제3 보고서>

작성자: 박기홍

직책: 19기 예비 단원

소속: RO:BIT 지능 팀

작성일: 2024년 11월 4일

목차

1. 요약
2. 프로젝트 개요
3. 관련 이론 및 개념
    3.1 이론적 배경
    3.2 관련 개념
4. 개발 환경
5. 패키지 분석
6. 개발 과정
    6.1 패키지 설계 및 구조
    6.2 코드 및 주요 기능 설명
    6.3 실패 사례와 시도한 방법
7. 결과
8. 결론 및 향후 발전 방향
9. 참고 문헌
10. 부록

1. 요약

이 보고서는 chatter_cli 패키지 개발 과정과 결과에 대해 설명한다. chatter_cli 패키지는 ROS2에 내장된 talker와 listener를 레퍼런스로 똑같이 구현하는 내용의 패키지이다. talker에서 토픽을 퍼블리시하고, listner에서 서브스크라이브한 후 터미널로 출력하는 패키지이다.

본 패키지는 RO:BIT의 ROS2 - 1일차 과제3에 대한 패키지로 제시된 요구 사항을 충족하는 데 중점을 두었다. 이 보고서에서는 프로젝트의 개요, 관련 이론, 개발 환경, 개발 과정, 결과 등을 다루고 있다.

2. 프젝젝트 개요

이 프로젝트의 목표는 ROS2 환경에서 기본적인 퍼블리셔-서브스크라이버 통신을 구현하는 패키지인 chatter_cli를 개발하는 것이다. chatter_cli 패키지는 ROS2에 내장된 talker와 listener 노드를 기반으로 하여, talker에서 메시지를 퍼블리시하고, listener에서 해당 메시지를 구독해 터미널에 출력하도록 설계되었다. 이를 통해 ROS2의 핵심 통신 구조인 토픽 기반 메시지 전달 방식을 학습하고자 했다.

3. 관련 이론 및 개념

3.1 이론적 배경

이 프로젝트는 ROS2의 퍼블리셔-서브스크라이버 모델을 바탕으로 한다. ROS2 (Robot Operating System 2)는 로봇 소프트웨어 개발을 지원하는 프레임워크로, 노드(Node), 토픽(Topic), 메시지(Message), 퍼블리셔(Publisher), 서브스크라이버(Subscriber) 등의 개념을 활용해 노드 간의 통신을 구현한다. 이러한 구조를 통해 분산 시스템 내에서 모듈 간 데이터 교환을 효율적으로 처리할 수 있다.

퍼블리셔-서브스크라이버 모델: ROS2의 기본적인 통신 구조로, 데이터를 발신하는 노드(퍼블리셔)와 이를 수신하는 노드(서브스크라이버)로 구성된다. 퍼블리셔는 특정 토픽(Topic)으로 데이터를 발신하고, 해당 토픽을 구독한 서브스크라이버가 데이터를 수신한다. 이 프로젝트에서는 Talker가 퍼블리셔 역할을 수행하며, Listener가 서브스크라이버로서 퍼블리셔가 발신하는 메시지를 수신하여 터미널에 출력한다.

3.2 관련 개념

- **노드(Node)**: ROS2에서 독립적으로 실행되는 프로세스이다. 본 프로젝트에서는 Talker 노드와 Listener 노드가 각각 퍼블리셔와 서브스크라이버 역할을 수행한다. 두 노드는 독립적으로 실행되며, 서로 다른 터미널에서 구동된다.
- **토픽(Topic)**: 노드 간의 비동기적 데이터 송수신을 위한 통로이다. 특정 토픽으로 퍼블리시된 데이터는 해당 토픽을 구독하는 모든 노드로 전달된다. 본 프로젝트에서는 /chatter_cli와 /chatter_count 토픽을 사용해 메시지를 주고받는다. /chatter_cli 토픽은 Talker에서 퍼블리시한 사용자 입력 문자열을 전달하며, /chatter_count 토픽은 퍼블리시 횟수를 전달하는 데 사용되었다.
- **메시지(Message)**: 노드 간의 데이터 교환을 위해 사용하는 데이터 형식이다. ROS2는 다양한 기본 메시지 타입을 제공한다. 이 프로젝트에서는 std_msgs::msg::String 타입을 사용해 사용자로부터 입력받은 문자열을 전송하고, std_msgs::msg::Int64 타입을 사용해 퍼블리시 횟수를 전달했다.

- **퍼블리셔(Publisher)**: 노드가 특정 토픽으로 데이터를 전송하기 위해 사용하는 인터페이스이다. Talker 노드는 /chatter_cli와 /chatter_count 두 개의 퍼블리셔를 생성하여 각각 문자열 메시지와 퍼블리시 횟수를 발신한다.
- **서브스크라이버(Subscriber)**: 노드가 특정 토픽을 구독하여 데이터를 수신하기 위한 인터페이스이다. Listener 노드는 /chatter_cli와 /chatter_count 토픽을 각각 구독하여, 발신된 메시지와 메시지의 카운트를 수신하고 터미널에 출력한다.
- **콜백 함수(Callback Function)**: 서브스크라이버가 메시지를 수신할 때 실행되는 함수입니다. Listener 노드에서는 chatterCallback과 countCallback이라는 두 콜백 함수를 사용하여 각각 문자열 메시지와 퍼블리시 횟수를 처리했다.

4. 개발 환경

이 프로젝트는 Ubuntu 22.04에서 ROS2 Humble을 기반으로 수행되었다. 하드웨어는 Intel i9 프로세서와 32GB RAM을 갖춘 Lenovo Legion 노트북을 사용하하였다. 또한, ROS2 패키지 개발을 위해 Visual Studio Code와 Git을 사용하여 코드 작성과 버전 관리를 하였다.

소프트웨어 환경:

운영체제: Ubuntu 22.04 LTS

ROS2 버전: Humble Hawksbill

IDE 및 VCD: Visual Studio Code, Git

컴파일러: GCC (GNU Compiler Collection) 11.4.0

CMake: 프로젝트 빌드 및 패키지 관리를 위해 CMake를 사용하였으며, ROS2 패키지의 빌드 시스템인 ament_cmake를 사용하여 프로젝트를 구성하였다.

필수 패키지:

ament_cmake: ROS2 빌드 시스템을 지원하는 패키지

rclcpp: ROS2 C++ 클라이언트 라이브러리

std_srvs: 기본 서비스 메시지 패키지로, 배경색 변경 등의 다양한 서비스 구현을 위해 필요

5. 패키지 분석

이 프로젝트에서 개발한 chatter_cli 패키지는 ROS2에서 퍼블리시와 서브스크라이브의 개념을 이해하기 위해 설계되었다. ROS2 - demo_nodes_cpp의 talker와 listener 노드의 작동 방식을 참고하였으며, 퍼블리시와 서브스크라이브의 동작 방식을 이해하고 구현하는 것을 목표로 개발했다.

chatter_cli 패키지는 다음과 같은 주요 노드로 구성되어 있다:

- **talker**: 사용자로부터 입력 받은 메시지를 /chatter_cli 토픽으로 퍼블리시하며, 퍼블리시 횟수를 /chatter_count 토픽으로 전송한다.
- **listener**: /chatter_cli와 /chatter_count 토픽을 구독하며, 수신된 메시지를 터미널에 출력한다.

패키지의 각 기능은 ROS2의 노드로 구현되어 있으며, 노드 간의 통신을 통해 상호작용하도록 설계되었다. 이를 통해 ROS2의 퍼블리셔-서브스크라이버 모델의 실제 작동 방식을 실습하고 학습할 수 있었다.

6. 개발 과정

6.1. 패키지 설계 및 구조

패키지와 노드 구조를 설계 할 때 노드를 독립적으로 구현하는 것에 초점을 맞추어 개발을 진행하는 것이 중요하다. 그러나, 이번 과제에서는 main.cpp파일을 만들지 않고, talker.cpp와 listener.cpp에 내장시켰다. talker와 listener는 서로 다른 노드로 독립되어야 하며, 각 터미널에서 각각의 노드를 활성화 해야 하는 것으로 이해하였다. 이는 spin해야 하는 타겟 노드가 서로 달라야 하기에 main_talker.cpp와 main_listener.cpp를 생성하고 각 구현부에 main() 함수를 내장해야 한다. 이번 패키지에서는 깔끔하게 talker.cpp 하나와 listener.cpp 하나로 구성하기 위해 각 구현부에 main() 함수를 개별적으로 내장 시켜 구현하였다.

과제에서 제시된 주요 기능은 다음과 같다:

1. talker 노드에서 터미널로 퍼블리시 할 std_msgs/String의 값 입력 받고 퍼블리시하기.
2. std_msgs/Int64를 사용해서 몇 번째 퍼블리시 인지 개별적으로 퍼블리시하기.
3. listener 노드에서 talker 노드에서 퍼블리시한 토픽 두 개를 서브 스크라이브한 후, 터미널로 출력하기. e.g.,) hello, 1 -> Subscribed "hello", "1"
4. 토픽명은 /chatter_cli와 /chatter_count로 고정하기.

각 기능은 분산된 구조로 제어하기 위해 독립적인 노드로 설계하였다.

- talker: 사용자로부터 내용을 입력 받고, 입력 값과 카운트를 listener로 퍼블리시하도록 설계하였다.
- listener: talker로부터 전달 받은 입력 값과 카운트를 터미널로 출력하도록 설계하였다.

6.2. 코드 및 주요 기능 설명

6.2.1. talker 노드

talker노드는 사용자가 입력한 메시지를 퍼블리시하고, 퍼블리시 횟수를 카운트한 다음에 listener 노드로 전송하는 역할을 한다.

6.2.1.1. Talker() 생성자

```
Talker::Talker() : Node("talker")
{
    // 퍼블리셔 초기화
    publisher_ =
this->create_publisher<std_msgs::msg::String>("/chatter_cli", 10);
    count_publisher_ =
this->create_publisher<std_msgs::msg::Int64>("/chatter_count", 10);
    RCLCPP_INFO(this->get_logger(), "Talker Node 초기화 완료");
}
```

/chatter_cli 토픽에 std_msgs::msg::String 타입으로 사용자 입력 메시지를 퍼블리시하는 publisher_와 /chatter_count 토픽에 퍼블리시 횟수를 std_msgs::msg::Int64 타입으로 전송하는 count_publisher_를 생성했다. 생성을 완료했으면, 디버깅을 위해 초기화 완료 문구를 출력하도록 설계하였다.

6.2.1.2. run() 메서드

```
void Talker::run()
{
    // rclcpp::Rate rate(1); // 1초에 한 번 루프 반복
    while (rclcpp::ok())
    {
        // 사용자로부터 문자열 입력 받기
        std::string input;
        std::cout << "메시지 입력: ";
        std::getline(std::cin, input); // getline(): C언의 gets() 같은
함수임

        // 문자열 메시지 퍼블리시
        auto string_message = std_msgs::msg::String();
        string_message.data = input;

        publisher_->publish(string_message);

        message_count_++;
        auto count_message = std_msgs::msg::Int64();
        count_message.data = message_count_;
```

```
        count_publisher_->publish(count_message);

        // ROS 스핀 및 루프 속도 유지
        rclcpp::spin_some(this->get_node_base_interface());
    }
}
```

run() 메서드 내부에서 talker의 핵심 기능을 구현했다. while()문 내에서 사용자로부터 문자열을 입력 받고, std_msgs::msg::String 타입 객체에 저장한 후에 /chatter_cli 토픽으로 퍼블리시한다.

그리고 메시지 카운트 변수인 message_count_ 의 값도 후위 연산으로 1씩 증가 시키고, std_msgs::msg::Int64 타입 객체에 저장한 다음 /chatter_count 토픽으로 퍼블리시 했다.

rclcpp::spin_some() 함수를 사용해서 ROS2 이벤트 처리를 진행하면서 루프를 지속시켰다. rclcpp::spin_some()는 ROS2에서 사용되는 spin 함수 중 하나로, 특정 노드에서 발생하는 콜백을 비동기처리 하는 데 사용 된다는 것을 알게 되었다. 특징은 루프() 내에서 사용되고, 다른 작업을 중단하지 않고도 콜백을 비동기적으로 처리하도록 설계 되었다는 것이다.

6.2.1.3. main() 함수

```
int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<Talker>();
    node->run(); // run 함수 호출
    rclcpp::shutdown();
    return 0;
}
```

main() 함수에서 Talker 노드의 인스턴스를 생성하고, run() 함수를 호출해 노드를 실행하도록 처리했다.

6.2.2. listener 노드

listener 노드는 talker 노드가 퍼블리시한 메시지와 퍼블리시 횟수를 서브스크라이브하여 터미널에 출력하는 역할을 한다.

6.2.2.1. Listener() 생성자

```
Listener::Listener() : Node("listener")
{
    // 문자열 메시지 서브스크라이버 초기화
    subscription_ =
this->create_subscription<std_msgs::msg::String>("/chatter_cli", 10,
std::bind(&Listener::chatterCallback, this, std::placeholders::_1));
    // 카운트 메시지 서브스크라이버 초기화
    count_subscription_ =
this->create_subscription<std_msgs::msg::Int64>("/chatter_count", 10,
std::bind(&Listener::countCallback, this, std::placeholders::_1));
    RCLCPP_INFO(this->get_logger(), "Listener Node 초기화 완료");
}
```

Listener 생성자는 사용자로부터 입력 받은 문자열과 메시지 카운트를 talker로부터 받아 출력해야 하므로, 두 개의 서브스크라이버를 설정한다. subscription_은 /chatter_cli 토픽을 구독하고, std_msgs::msg::String 타입의 메시지를 수신한다. count_subscription_은 /chatter_count 토픽을 구독하고, std_msgs::msg::Int64 타입의 퍼블리시 횟수를 수신한다.

두 개의 서브스크라이버 설정이 완료되면, 디버깅을 위해 초기화 완료를 출력하도록 했다.

6.2.2.2. chatterCallback() 메서드

```
void Listener::chatterCallback(const std_msgs::msg::String::SharedPtr
msg)
{
    RCLCPP_INFO(this->get_logger(), "퍼블리시된 메시지: '%s'",
msg->data.c_str());
}
```

countCallback() 메서드는 /chatter_count 초픽으로부터 퍼블리시 횟수를 수신하는 콜백 메서드이다. listener 노드는 수신한 퍼블리시 횟수를 터미널에 출력하고, 출력 형식은 "퍼블리시된 메시지: {메시지 내용}"이다.

6.2.2.3. countCallback() 메서드

```
void Listener::countCallback(const std_msgs::msg::Int64::SharedPtr msg)
{
    RCLCPP_INFO(this->get_logger(), "퍼블리시된 횟수: %ld", msg->data);
}
```

countCallback() 메서드는 /chatter_count 토픽으로부터 퍼블리시 횟수를 수신하는 콜백 메서드이다. listener 노드는 수신한 퍼블리시 횟수를 터미널에 출력하고, 이때 출력 형식은 "퍼블리시된 횟수: {횟수}"이다.

6.2.2.4. main() 함수

```
int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<Listener>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

main() 함수에서 Listener 노드의 인스턴스를 생성하고, run() 함수를 호출해 노드를 실행하도록 처리했다.

*Talker와 Listener에 main() 함수가 각각 존재하는 이유는 두 노드를 서로 독립적인 프로세스로 실행하기 위함이다.

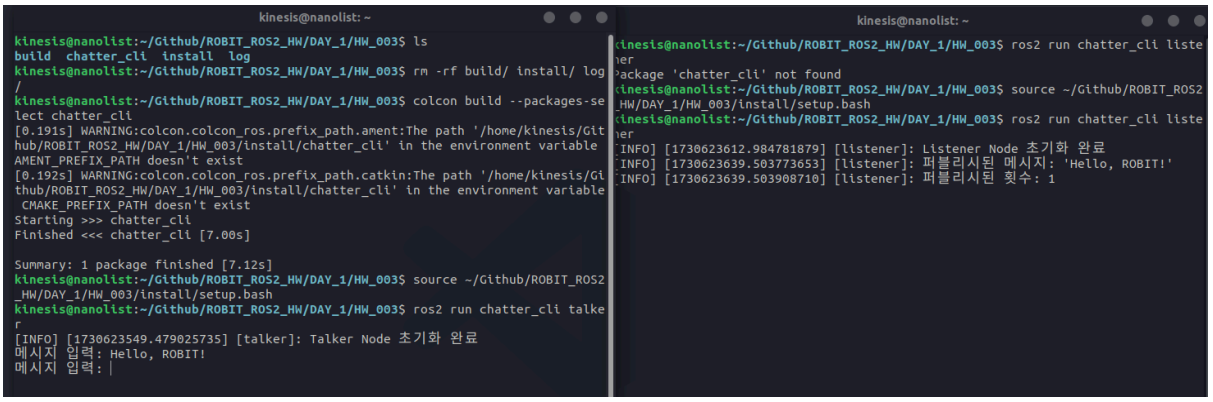
6.3.실패 사례와 시도한 방법

우선 과제를 이해하는 데 있어 시간이 오래 걸렸다. ROS2에 대한 개념이 부족한 백그라운드에서 해당 과제를 이해는 데 시간이 오래 걸렸으며, 구현을 할 때도 시간이 오래 걸렸다.

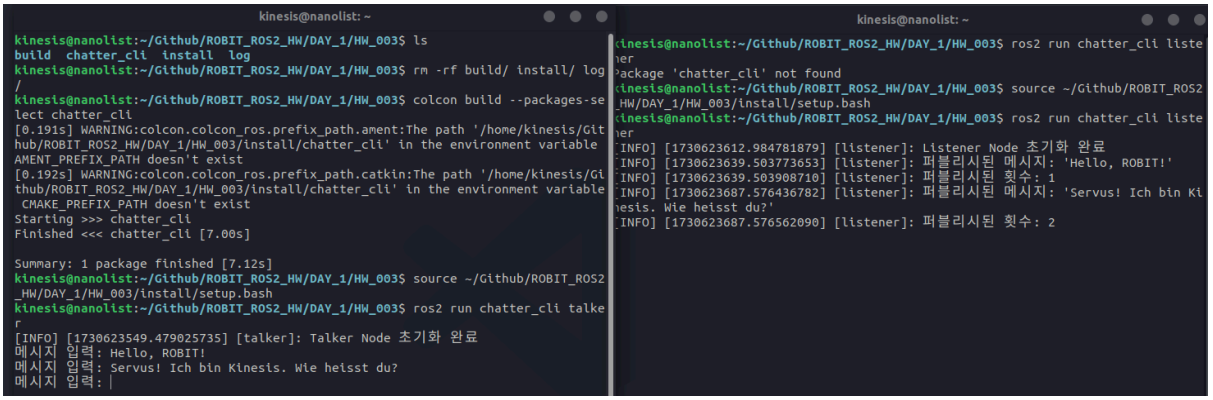
talker와 listener의 개념을 먼저 이해하기 위해 자료를 찾아 보았다. rqt를 실행하면 어떻게 구성되어 있는지도 알 수 있다. 그래도 이해가 안 되고 어떤 부분을 놓쳤는지 알고 싶을 때는 velog에 있는 I ROBO U님의 자료와 오로카에서 여러 글을 찾아 보았다. 그래도 모르겠을 때는 GPT를 사용하였으며, 제대로 이해하고 있는 게 맞은지 검토하기 위해서도 적극 사용하였다.

현재 가장 큰 약점은 과제를 이해하고 수행하는 능력과 ROS2에 대한 기본적인 백그라운드가 부족하다는 것이다.

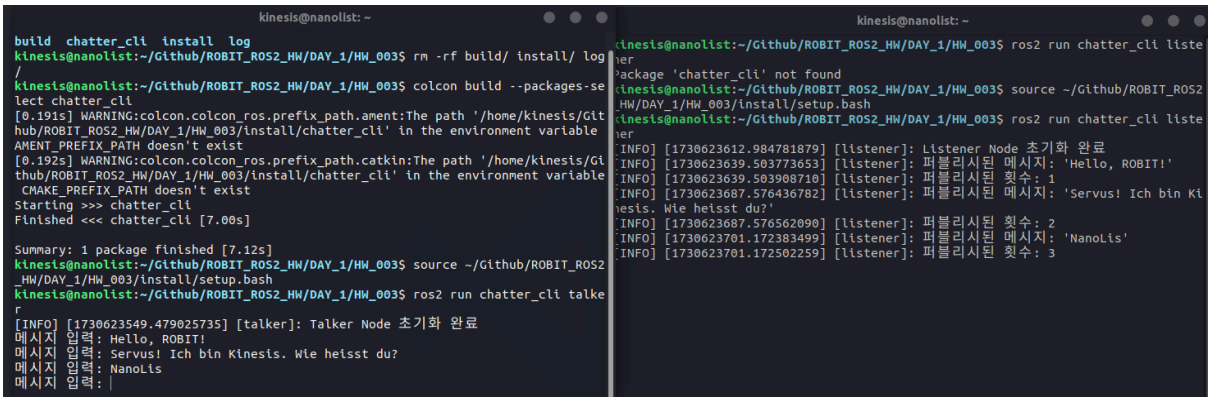
7. 결과



<chatter_cli 실행 결과 1>



<chatter_cli 실행 결과 2>



<chatter_cli 실행 결과 3>

8. 결론 및 향후 발전 방향

이번 프로젝트는 ROS2의 퍼블리셔-서브스크라이버 모델을 이해하고 이를 구현하기 위해 chatter_cli 패키지 개발이 과제로 진행 되었다. talker 노드를 통해 사용자로부터 내용을 입력 받고 받은 내용과 횟수를 퍼블리시 하고, listener 노드에서 이를 서브스크라이브하여 터미널에 출력하는 과정을 실습해 보면서 퍼블리셔-서브스크라이버에 대한 개념을 새로 알고 어떻게 사용하는 것인지 배울 수 있었다.

어디서 부족하고, 어떤 능력이 부족한지 잘 알고 있다고 생각한다. ROS2에 대한 기본적인 백그라운드를 향상 시키기 위해 과제를 다 끝냈으면 ROS2 관련 자료를 찾아보고 깃블로그에 정리하는 것이 절실하다. 구선생님의 실습 영상을 참고하여 구현하는 것도 큰 도움이 될 것이라 생각한다.

9. 참고 문헌

- 표윤석. (2020.08.18.). "000 로봇 운영체제 ROS 강좌 목차". https://cafe.naver.com/openrt/24070
- 표윤석. (2024.09.05.). "로봇 운영체제 ROS 2 특강 발표 자료", https://docs.google.com/presentation/d/1QvUHj8iZCWIGiWRCgHFG-DLAmoUK6khO9xBXCzjPTTs/edit
- i_robo_u. (n.d.). "인공지능 로봇 개발". Velog. https://velog.io/@i_robo_u/series/%EC%9D%B8%EA%B3%B5%EC%A7%80%EB%8A%A5%EB%A1%9C%EB%B4%87%EA%B0%9C%EB%B0%9C

10. 부록

10.1 레파지토리 링크

ROBIT_ROS2_HW: https://github.com/kinesis19/ROBIT_ROS2_HW

10.2 과제 Wiki

ROS2_HW_DAY_1: https://github.com/kinesis19/ROBIT_ROS2_HW/wiki/ROS2_HW_DAY1

10.3 패키지 구조

패키지의 구조는 다음과 같다:

```
chatter_cli
├── CMakeLists.txt
├── include
│   └── chatter_cli
│       ├── listener.hpp
│       └── talker.hpp
├── package.xml
└── src
    ├── listener.cpp
    └── talker.cpp
```