ROS_3일차_보고서_20기_인턴_2025402040_컴퓨터정보공학부_노기문

토픽이해하기

ROS2와 토픽의 개념

ROS2는 여러 개의 노드가 협력하는 구조로 이루어진다. 각 노드는 데이터를 주고받으며, 이때 사용되는 핵심 메커니즘이 토픽(topic) 이다. 토픽은 publish과 subscribe의 개념을 따른다.

publisher는 특정 토픽 이름에 맞는 메시지를 계속 내보내고
subscriber는 해당 토픽을 구독하여 데이터를 받는다.

이 구조 덕분에 발행자와 구독자는 서로의 존재를 직접 알 필요가 없고, 단지 같은 토픽 이름과 메시지 타입만 일치하면 된다. 즉, 토픽은 일종의 비동기 데이터 버스 역할을 한다.

토픽 구조 시각화

rqt_graph 실행한뒤 GUI 기반 도구로, 현재 노드와 토픽 간 연결 관계를 시각적으로 확인 가능하다.
예시로 turtle_teleop_key 노드는 /turtle1/cmd_vel 토픽에 메시지를 발행하고, turtlesim_node가 같은 토픽을 구독하여 실제 거북이를 움직인다.

활용 의의

토픽은 ROS2 시스템의 기본 통신 단위다.
ros2 topic 계열 명령어를 활용하면 현재 노드들이 어떤 데이터를 주고받는지 실시간으로 파악할 수 있다.
이는 로봇 시스템의 동작을 이해하고 디버깅하며, 새로운 기능을 추가할 때 필수적인 과정이다.
특히 echo, info, pub, hz, bw 명령은 실제 메시지 흐름을 검증하는 데 핵심적이다.

정리

ROS2 토픽은 발행자와 구독자가 독립적으로 데이터를 주고받게 해주는 통신 채널이다.
튜토리얼을 통해 turtlesim을 예제로 사용하면서 토픽의 목록 확인, 메시지 구조 이해, 직접 발행 및 모니터링까지 경험할 수 있다.
결국 사용자는 토픽을 자유롭게 다루면서 시스템의 데이터 흐름을 조사하고 제어할 수 있는 능력을 얻게 된다.

서비스이해하기

ROS2 서비스 개념

ROS2에서 서비스는 request과 response 구조로 동작하는 통신 방식이다. 토픽이 발행과 구독을 통해 지속적으로 데이터를 주고받는 것과 달리, 서비스는 특정 시점에 요청을 보내고 그에 대한 응답을 받는 형태로 사용되다. 따라서 주기적인 데이터 전송보다는 필요할 때만 호출하는 기능에 적합하다.

서비스 다루기

서비스 목록 확인

: ros2 service list

현재 실행 중인 서비스들의 이름을 확인할 수 있다. 예로 /clear, /spawn, /reset 등이 있다.

서비스 타입 확인

: ros2 service type /clear

: ros2 service list -t

각 서비스가 어떤 타입으로 정의되어 있는지 확인할 수 있다. 예를 들어 /clear는 std_srvs/srv/Empty 타입이다.

특정 타입의 서비스 찾기

: ros2 service find std_srvs/srv/Empty

지정한 타입을 사용하는 서비스들을 찾을 수 있다.

서비스 인터페이스 구조 확인

: ros2 interface show turtlesim/srv/Spawn

요청과 응답 필드를 나누어 확인할 수 있다. 요청에는 x, y, theta, name이 있고 응답에는 name이 있다.

서비스 호출

: ros2 service call /clear std_srvs/srv/Empty

: ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: ''}"

CLI에서 직접 서비스를 호출할 수 있다. 요청 값이 필요한 경우 YAML 형식으로 입력한다.

핵심 정리

서비스는 요청할 때만 실행되는 통신 방식이다.
반복적이고 지속적인 데이터 교환은 토픽을 사용하고, 특정 요청에 대한 결과를 얻고자 할 때는 서비스를 사용한다.
ros2 service list, type, find, interface show, call 명령어로 서비스를 이해하고 활용할 수 있다.

파라미터이해하기

ROS2 파라미터 개념

parameter는 노드의 configuration값이다. 노드는 정수, 실수, 불리언, 문자열, 리스트타입의 파라미터를 가질 수 있다. ROS2에서는 각 노드가 자체 파라미터를 유지한다.

Setup

두 개의 노드를 실행한다:

: ros2 run turtlesim turtlesim_node 실행하여 turtlesim 윈도우 띄운다

: ros2 run turtlesim turtle_teleop_key 실행하여 키보드로 제어 가능하게한다

ros2 param list

노드들이 가진 파라미터 목록을 본다.

예: /teleop_turtle 노드의 scale_angular, scale_linear, use_sim_time 등이 있고 /turtlesim 노드의 배경색 설정 파라미터들(background_r, background_g, background_b) 등이 있다.

ros2 param get

특정 노드의 특정 파라미터의 타입(type)과 현재 값(value)을 확인하다.

예: /turtlesim background_g 값이 정수(86)임을 확인하다.

ros2 param set

파라미터 값을 실행 중(runtime) 변경하다.

예: /turtlesim background_r 값을 150으로 바꿔 배경색이 변경됨.

단, 변경된 값은 현재 세션에만 유효하고, 재시작하면 초기 값으로 돌아간다.

ros2 param dump

노드의 현재 파라미터 값 전체를 출력하거나 파일로 저장한다.

예: /turtlesim의 파라미터들을 turtlesim.yaml 파일로 저장한다.

ros2 param load

저장된 파라미터 파일을 실행 중인 노드에 로드한다.

예: turtlesim.yaml 파일을 /turtlesim 노드에 불러들여 색상 등이 복원됨. 읽기 전용 파라미터는 변경 불가할 수 있음.

노드 시작 시 파라미터 파일 함께 로드

노드를 실행할 때 --ros-args --params-file <파일> 옵션을 붙여 저장된 설정을 불러들이다.
ROS
예: ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim.yaml 로 실행해 배경색 등이 미리 설정된 상태로 시작됨.

핵심 정리

서비스는 요청할 때만 실행되는 통신 방식이다.
반복적이고 지속적인 데이터 교환은 토픽을 사용하고, 특정 요청에 대한 결과를 얻고자 할 때는 서비스를 사용한다.
ros2 service list, type, find, interface show, call 명령어로 서비스를 이해하고 활용할 수 있다.

액션이해하기

Actions는 ROS2의 통신 방식 중 하나이다.
서비스와 비슷하지만 긴 시간 걸리는 작업을 처리할 때 사용된다.
Action은 세 부분(goal, feedback, result)으로 구성된다.
클라이언트(action client)가 목표(goal)를 서버(action server)에 보내면 서버는 목표를 인정하고 피드백을 지속적으로 보내다가 최종 결과를 반환한다.
클라이언트는 목표를 취소할 수 있다.

Setup
/turtlesim 노드와 /teleop_turtle 노드를 실행한다:
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
``` :contentReference[oaicite:4]{index=4}

Action 사용 관찰
teleop_turtle 노드를 띄우면 “G|B|V|C|D|E|R|T 키로 절대 방향을 회전” 등의 안내가 나온다. 이는 rotate_absolute 액션을 사용하는 기능이다.
회전 중에 ‘F’ 키를 누르면 목표가 취소됨(cancel됨) 메시지가 나타난다.
새로운 목표가 이전 목표가 완료되기 전에 들어오면 이전 목표가 ‘abort’됨(중단됨) 메시지가 나타난다.

ros2 node info
노드의 정보를 확인하여 어떤 action server와 action client가 있는지 본다.
예: /turtlesim 노드가 Action Servers: /turtle1/rotate_absolute 제공함을 확인하다. /teleop_turtle은 Action Clients로 그 액션에 목표(goal)를 보내는 노드임을 확인하다.

ros2 action list
현재 시스템에 존재하는 액션 이름 목록을 확인하다. 예: /turtle1/rotate_absolute만 있음.
-t 옵션을 붙이면 액션 타입(action type)도 같이 표시됨. 예: turtlesim/action/RotateAbsolute.

ros2 action info
특정 액션의 클라이언트 수(action clients)와 서버 수(action servers)를 확인하다. 예: /turtle1/rotate_absolute 액션에 /teleop_turtle이 클라이언트, /turtlesim이 서버임을 확인함.
```

ros2 interface show
액션 타입의 구조(goal, result, feedback) 확인한다.

예: turtlesim/action/RotateAbsolute 타입은
goal: float32 theta
result: float32 delta
feedback: float32 remaining 필드를 가짐.

ros2 action send_goal
명령줄에서 액션 목표(goal)를 직접 보낸다.
예:
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
목표가 수락됨(goal accepted), 결과(result)가 반환됨(success 상태) 예: delta 값.
--feedback 옵션을 쓰면 피드백 메시지(remaining 값 등)를 실시간으로 받는다.

요점 정리
Actions는 긴 동작을 제어하고 모니터링할 수 있게 한다.
목표(goal)를 보내고 진행상황(feedback)과 최종 결과(result)를 받는다.
목표를 취소하거나 새로운 목표로 이전 목표를 중단하게 할 수 있다.
로봇 내비게이션 같은 작업에 알맞다. 목표 위치로 이동하고 중간 상태를 알려주며 완료 상태를 리턴하는 유형이다.

퍼블리셔, 서스크라이버이해하기

ROS2에서 노드는 실행 가능한 프로세스로 다른 노드들과 통신한다 한다.
이 예제는 퍼블리셔(publisher)와 구독자(subscriber)를 만드는 간단한 “talker-listener” 시스템이다 한다. 퍼블리셔가 문자열 메시지를 토픽으로 발행하고 구독자가 그걸 받는다 한다.

패키지 생성
워크스페이스의 src 디렉터리로 이동한다.
ros2 pkg create --build-type ament_cmake cpp_pubsub 명령으로 패키지 cpp_pubsub 생성한다.

퍼블리셔 노드 작성
publisher_member_function.cpp 파일 만들고 rclcpp와 std_msgs/msg/string.hpp 헤더 포함한다.
클래스 MinimalPublisher 상속받아 rclcpp::Node 구현한다. 노드 이름은 minimal_publisher라 한다.
퍼블리셔 생성: 문자열 메시지 타입, 큐 크기(10), 토픽 이름 “topic” 사용한다.
타이머(timer) 설정: 주기 500밀리초마다 콜백함수 실행해 메시지 발행한다. 카운터(count_)로 메시지 내용 변경한다.
main 함수에서 rclcpp::init(), 노드 객체 spin, 종료 시 rclcpp::shutdown() 한다.

서스크라이버 노드 작성
subscriber_member_function.cpp 파일 만들고 같은 메시지 타입 std_msgs/msg/String 사용한다.
클래스 MinimalSubscriber 상속받아 rclcpp::Node 구현한다. 노드 이름 minimal_subscriber라 한다.
create_subscription 호출: 메시지 타입, 토픽 이름 “topic”, 큐 크기 10, 콜백 함수 지정한다. 콜백은 메시지 받았을 때 로그 출력한다.
main 함수 유사하게 초기화, spin, 종료 한다.

CMakeLists.txt 및 빌드 설정
CMakeLists.txt 에 퍼블리셔와 구독자 노드의 executable 설정 추가한다. ament_target_dependencies로 rclcpp, std_msgs 연결한다.

package.xml에도 의존성 명시한다.

빌드 및 실행

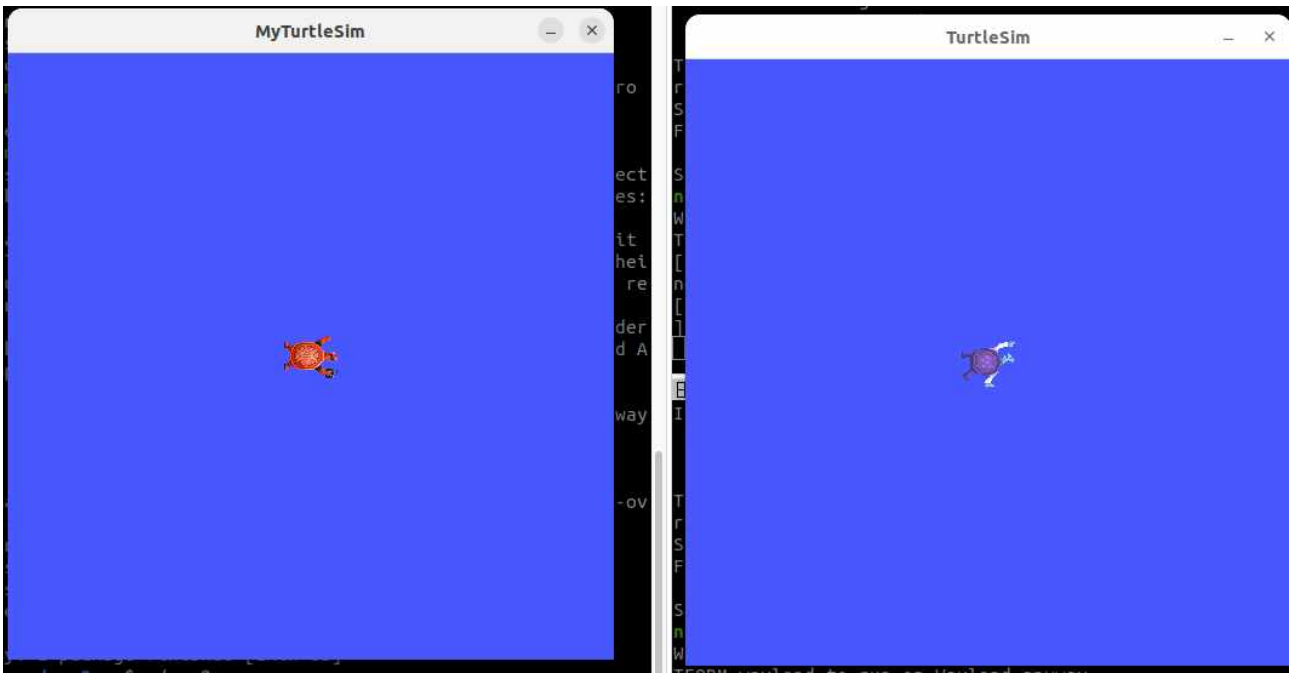
워크스페이스 루트에서 colcon build --packages-select cpp_pubsub 실행한다.
새 터미널에서 source install/setup.bash (또는 적절한 setup 스크립트) 실행한다.
퍼블리셔 노드 실행: ros2 run cpp_pubsub talker → 0.5초 주기로 메시지 발행 시작한다.
구독자 노드 실행: ros2 run cpp_pubsub listener → 메시지가 출력됨 ("I heard: …") 한다.

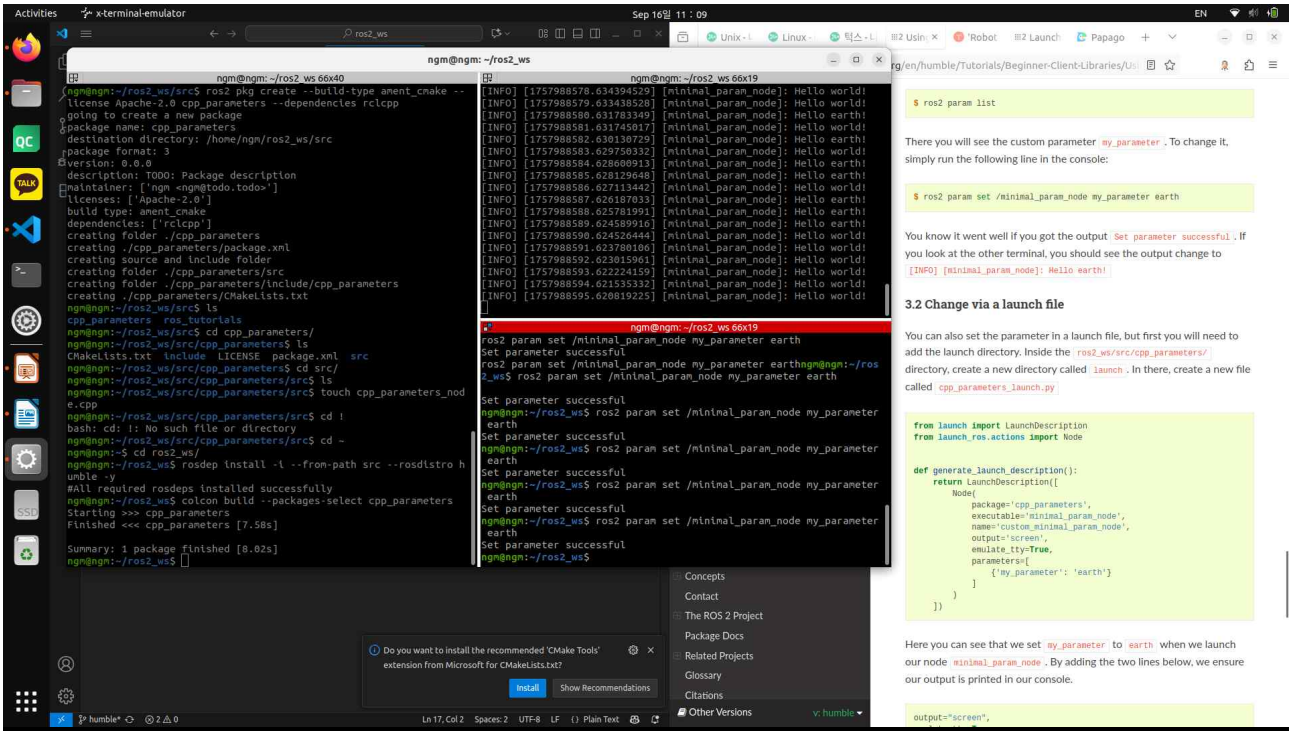
요점

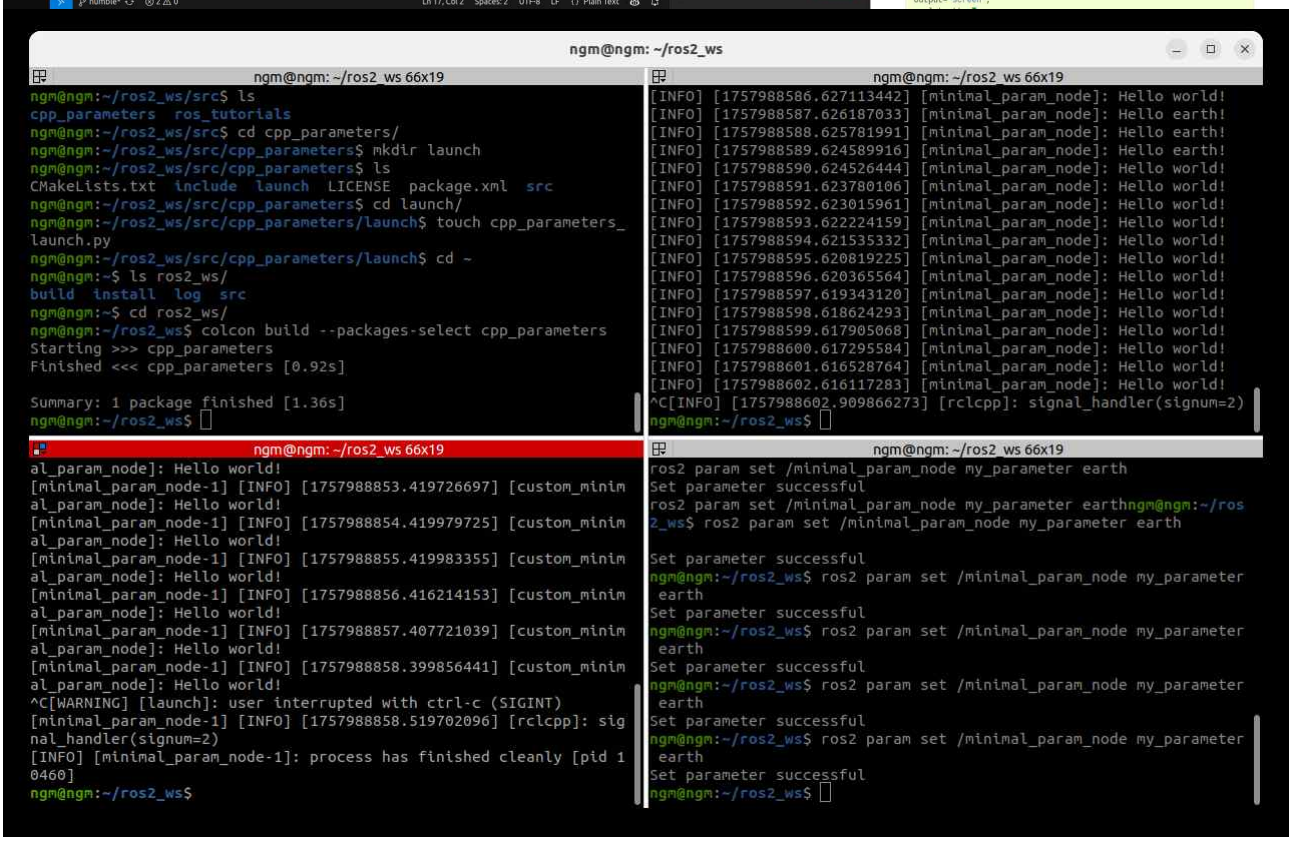
퍼블리셔와 구독자는 같은 메시지 타입과 토픽 이름을 사용해야 통신 가능하다 한다.
퍼블리셔 쪽은 타이머 콜백으로 주기 발행, 구독자는 메시지 도착 시 실행되는 콜백만 있으면 된다.
만들어진 노드들을 빌드 전에 설정 파일(CMakeLists, package.xml)에 정확한 의존성 및 실행 파일 정의가 필요하다.

파라미터구현하기

Node 내부 설정(configuration)을 실행 시나 launch 파일을 통해 바꿀 수 있게 하려면 파라미터(parameter)를 클래스 내부에서 선언(declare)하고 사용하는 방식이 필요하다 한다. 이 튜토리얼은 C++에서 클래스 내부에 파라미터를 선언하고, 콘솔 또는 launch 파일로 값 변경하는 방법을 보여준다.




ngm@ngm: ~/ros2_ws
ngm@ngm: ~/ros2_ws 66x40
ngm@ngm: ~/ros2_ws 66x19
ngm@ngm: ~/ros2_ws 66x19
$ ros2 param list
There you will see the custom parameter my_parameter . To change it, simply run the following line in the console:
$ ros2 param set /minimal_param_node my_parameter earth
You know it went well if you got the output Set parameter successful . If you look at the other terminal, you should see the output change to [INFO] [minimal_param_node]: Hello earth!
3.2 Change via a launch file
You can also set the parameter in a launch file, but first you will need to add the launch directory. Inside the ros2_ws/src/cpp_parameters/ directory, create a new directory called launch . In there, create a new file called cpp_parameters_launch.py
Here you can see that we set my_parameter to earth when we launch our node minimal_param_node . By adding the two lines below, we ensure our output is printed in our console.


ngm@ngm: ~/ros2_ws
ngm@ngm: ~/ros2_ws 66x19
ngm@ngm: ~/ros2_ws 66x19
ngm@ngm: ~/ros2_ws 66x19
ngm@ngm: ~/ros2_ws 66x19