

hw1 보고서

- 목차

1. 노드

- 1.1. ROS 2 그래프
- 1.2. ROS 2의 노드
- 1.3. 주요 명령어 (ros2 node list / info / remap)

2. 토픽

- 2.1. Setup (turtlesim + teleop)
- 2.2. rqt_graph 시각화
- 2.3. ros2 topic list / info / echo
- 2.4. ros2 interface show
- 2.5. ros2 topic pub (메시지 발행)
- 2.6. ros2 topic hz / bw
- 2.7. ros2 topic find
- 2.8. Clean up

3. 서비스

- 3.1. Setup
- 3.2. ros2 service list / type / find
- 3.3. ros2 interface show
- 3.4. ros2 service call (Empty / Spawn 예제)

4. 파라미터

- 4.1. Setup
- 4.2. ros2 param list / get / set
- 4.3. ros2 param dump / load
- 4.4. 실행 시 파라미터 파일 적용

5. 액션

- 5.1. Setup
- 5.2. Use actions (Goal, Feedback, Result, Cancel)
- 5.3. ros2 node info (Action Server / Client 확인)
- 5.4. ros2 action list / -t
- 5.5. ros2 action info
- 5.6. ros2 interface show (Goal/Result/Feedback 구조)
- 5.7. ros2 action send_goal (--feedback 옵션 포함)

6. 요약

7. Using parameters in a class

1. 노드

배경

- ROS 그래프는 여러 개의 ROS 2 요소(노드, 토픽, 서비스, 액션 등)가 동시에 데이터를 주고받으며 연결된 네트워크 구조를 시각화 해준다.

1.1. ROS 2 그래프

여러 실행 파일(프로그램)과 그 연결 관계를 시각적으로 표현한 것.

로봇 시스템 전체를 구성하는 요소들이 서로 데이터를 교환하며 동작.

1.2. ROS 2의 노드

로봇 시스템의 기본 단위.

각 노드는 하나의 모듈성 있는 기능만 담당 (예: 바퀴 제어, 센서 데이터 송신 등).

노드 간 통신 방법: 토픽, 서비스, 액션, 파라미터.

하나의 실행 파일 안에 여러 개의 노드가 포함될 수도 있음.

1.3 명령어

1) `ros2 node list`

현재 실행 중인 노드 이름을 확인하는 명령어:

2) 노드 이름 리매핑(Remapping)

노드 이름을 변경하여 실행:

ex)

```
ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle
```

결과:

/my_turtle

/turtlesim

/teleop_turtle

3) `ros2 node info`

특정 노드의 자세한 정보를 확인하는 명령어:

```
ros2 node info <노드이름>
```

ex)

```
ros2 node info /my_turtle
```

출력 내용:

Subscribers (구독 중인 토픽)

Publishers (발행하는 토픽)

Service Servers (제공하는 서비스)

Service Clients (연결하는 서비스)

Action Servers / Clients

이 명령어로 노드가 ROS 그래프 상에서 어떤 데이터 흐름을 주고받는지 확인할 수 있음.

2. ROS 2 토픽

배경

- ROS 2는 복잡한 시스템을 여러 노드로 나눔.
- 토픽은 노드 간 메시지를 교환하는 통신 채널(버스) 역할.
- 한 노드는 여러 토픽에 동시에 데이터를 발행하거나 구독할 수 있음.
- 토픽은 ROS 그래프의 핵심적인 데이터 흐름 요소.

2.1. Setup

turtlesim 실행: `ros2 run turtlesim turtlesim_node`

키보드 제어 실행: `ros2 run turtlesim turtle_teleop_key`

노드 이름은 `/turtlesim`, `/teleop_turtle`.

1.2. rqt_graph

노드와 토픽 연결을 시각화하는 GUI 도구.

실행: `ros2 run rqt_graph rqt_graph`

`/teleop_turtle` 노드가 `/turtle1/cmd_vel` 토픽에 publish, `/turtlesim` 노드는 이를 subscribe하는 구조 확인 가능.

2.3. ros2 topic list

현재 실행 중인 모든 토픽 확인: `ros2 topic list`

토픽과 메시지 타입 함께 확인: `ros2 topic list -t`

출력:

`/turtle1/cmd_vel [geometry_msgs/msg/Twist]`

`/turtle1/pose [turtlesim/msg/Pose]`

2.4. ros2 topic echo

특정 토픽의 메시지 내용 실시간 확인: `ros2 topic echo /turtle1/cmd_vel`

-> 거북이를 움직이면 Twist 메시지가 출력됨.

2.5. ros2 topic info

토픽에 연결된 발행자/구독자 수와 메시지 타입 확인: `ros2 topic info /turtle1/cmd_vel`

2.6. ros2 interface show

토픽 메시지 타입의 내부 구조 확인: `ros2 interface show geometry_msgs/msg/Twist`
-> linear, angular 벡터 구조 확인 가능.

2.7. ros2 topic pub

토픽에 직접 메시지 발행하기:

```
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

기본적으로 1Hz로 지속 발행.

한 번만 발행: `ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{...}"`

2.8. ros2 topic hz

토픽 발행 주기(Hz) 확인: `ros2 topic hz /turtle1/pose`

2.9. ros2 topic bw

토픽의 대역폭(데이터 전송량) 확인: `ros2 topic bw /turtle1/pose`

2.10. ros2 topic find

특정 메시지 타입을 사용하는 토픽 찾기: `ros2 topic find geometry_msgs/msg/Twist`

출력:

```
/turtle1/cmd_vel
```

2.11. Clean up

모든 실행 중인 노드를 종료하려면 각 터미널에서 `Ctrl + C`.

종료 시에 까먹지 말고 할 것

3. ROS 2 서비스

배경

서비스는 ROS 그래프에서 사용하는 또 다른 통신 방법.

토픽과 달리 요청-응답 모델 기반.

토픽: 지속적으로 메시지를 발행/구독.

서비스: 클라이언트가 요청했을 때만 서버가 응답.

3.1. Setup

터틀심 노드 실행: `ros2 run turtlesim turtlesim_node`

키보드 제어 실행: `ros2 run turtlesim turtle_teleop_key`

3.2. ros2 service list

현재 실행 중인 모든 서비스 목록 확인: `ros2 service list`

출력:

```
/clear  
/kill  
/reset  
/spawn  
/turtle1/set_pen  
/turtle1/teleport_absolute  
/turtle1/teleport_relative
```

-> 대부분의 노드에는 공통적으로 파라미터 관련 서비스도 존재.

3.3. ros2 service type

특정 서비스의 타입 확인: `ros2 service type /clear`

출력:

```
std_srvs/srv/Empty
```

-> 요청과 응답 데이터가 없는 서비스.

서비스 목록 전체와 타입 함께 보기: `ros2 service list -t`

3.4. ros2 service find

특정 타입의 서비스를 검색: `ros2 service find std_srvs/srv/Empty`

출력:

```
/clear  
/reset
```

3.5. ros2 interface show

서비스 요청(Request)/응답(Response) 구조 확인:

`ros2 interface show turtlesim/srv/Spawn`

출력:

```
float32 x  
float32 y  
float32 theta  
string name  
----
```

```
string name
```

-> 요청 시 x, y, theta 위치와 각도를 지정, name은 선택적 입력.

-> 응답(Response)으로 생성된 거북이 이름 반환.

3.6. ros2 service call

서비스 호출: `ros2 service call <service_name> <service_type> <arguments>`

인자가 없는 서비스 호출: `ros2 service call /clear std_srvs/srv/Empty`

-> 화면에 그려진 선이 모두 지워짐.

인자가 있는 서비스 호출 (거북이 생성):

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

응답:

response:

`turtlesim.srv.Spawn_Response(name='turtle2')`

-> 새로운 거북이 turtle2가 생성됨.

4. ROS 2 파라미터

배경

파라미터(Parameter)는 노드의 설정값.

노드별로 독립적으로 관리되며, 정수(int), 실수(float), 불리언(bool), 문자열(string), 리스트(list) 값을 가질 수 있음.

런타임에서 변경 가능하며, 저장/로드를 통해 재사용할 수도 있음.

4.1. Setup

노드 실행:

`ros2 run turtlesim turtlesim_node`

`ros2 run turtlesim turtle_teleop_key`

4.2. ros2 param list

현재 노드들의 파라미터 목록 확인: `ros2 param list`

출력:

/turtlesim:

background_r

background_g

background_b

use_sim_time

-> turtlesim 노드에는 배경색을 제어하는 RGB 파라미터가 있음.

4.3. ros2 param get

특정 파라미터의 타입과 현재 값 확인: `ros2 param get /turtlesim background_g`

출력:

Integer value is: 86

4.4. ros2 param set

파라미터 값 변경: `ros2 param set /turtlesim background_r 150`

-> turtlesim 창 배경색이 바로 변경됨.

! 이 방법은 현재 세션에서만 적용되고, 노드 재시작 시 초기화됨.

4.5. ros2 param dump

현재 파라미터 값을 파일로 저장: `ros2 param dump /turtlesim > turtlesim.yaml`

결과:

/turtlesim:

```
ros__parameters:
  background_b: 255
  background_g: 86
  background_r: 150
  use_sim_time: false
```

4.6. ros2 param load

파일에서 파라미터 불러오기: `ros2 param load /turtlesim turtlesim.yaml`

일부 read-only 파라미터(qos_overrides 등)는 런타임에서 수정 불가 -> 경고 발생.

4.7. 노드 시작 시 파라미터 파일 불러오기

저장된 파라미터 파일을 적용하며 노드 실행:

`ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim.yaml`

-> 실행 시 배경색이 YAML에 저장된 값으로 설정됨.

5. ROS 2 액션

배경

액션은 장시간 실행되는 작업을 위한 통신 방식.

구성 요소: Goal(목표), Feedback(중간 피드백), Result(결과).

서비스와 차이점:

서비스: 요청 -> 단일 응답.

액션: 요청 -> 실행 중 피드백 -> 최종 결과 -> 취소 가능.

액션은 토픽 + 서비스 위에서 동작하며, 클라이언트-서버 모델을 사용.

5.1. Setup

```
ros2 run turtlesim turtlesim_node  
ros2 run turtlesim turtle_teleop_key
```

5.2. Use actions

방향 키: cmd_vel 토픽을 통해 거북이 이동.

문자 키(G, B, V, C, D, E, R, T): rotate_absolute 액션 실행.

F 키: 현재 실행 중인 목표(goal) 취소.

새로운 goal이 들어오면 서버가 이전 goal을 abort할 수도 있음.

5.3. ros2 node info

turtlesim 노드 정보 확인: `ros2 node info /turtlesim`

출력:

Action Server: /turtle1/rotate_absolute

-> /turtlesim 노드는 액션 서버 역할 수행.

teleop_turtle 노드 확인: `ros2 node info /teleop_turtle`

출력: Action Client: /turtle1/rotate_absolute

-> teleop_turtle 노드는 액션 클라이언트 역할 수행.

5.4. ros2 action list

현재 ROS 그래프의 액션 확인: `ros2 action list`

출력:

/turtle1/rotate_absolute

액션 타입까지 보기: `ros2 action list -t`

출력:

/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]

5.5. ros2 action info

특정 액션의 클라이언트/서버 정보 확인: `ros2 action info /turtle1/rotate_absolute`

출력:

Action clients: /teleop_turtle

Action servers: /turtlesim

5.6. ros2 interface show

액션 타입 구조 확인: `ros2 interface show turtlesim/action/RotateAbsolute`

출력:

```
float32 theta      # Goal (목표)
---
float32 delta      # Result (결과)
---
float32 remaining  # Feedback (실시간 피드백)
```

5.7. ros2 action send_goal

액션 직접 실행:

```
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
```

출력:

```
Goal accepted with ID: ...
Result:
  delta: -1.568
Goal finished with status: SUCCEEDED
-> 거북이가 90도 회전.
```

피드백 받기:

```
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: -1.57}" --feedback
```

출력:

```
Feedback:
  remaining: -3.11
...
Result:
  delta: 3.12
-> 실행 중 회전이 얼마나 남았는지 실시간으로 확인 가능.
```

6. 요약

- 노드: ROS 2 시스템의 최소 단위. 각각 독립된 기능을 담당하며 토픽, 서비스, 액션, 파라미터를 통해 통신.
- 토픽: 노드 간 메시지 스트리밍 방식 통신. 발행과 구독. `ros2 topic pub`, `echo`, `hz`, `bw` 등으로 확인 가능.
- 서비스: 요청과 응답 기반 통신. `ros2 service call`로 직접 호출 가능. 짧은 작업에 적합.

- 파라미터: 노드 설정값. 런타임에서 수정 가능하며 YAML 파일로 저장/로드 가능.
- 액션: 장시간 실행되는 작업용 통신. 목표 - 피드백 - 결과 흐름. ros2 action send_goal로 목표를 실행하고, 필요시 취소 가능.

7. Using parameters in a class

- 노드에서 파라미터(Parameter)를 선언하면 런타임 중 변경 가능.
- 콘솔/런치파일을 통해 값 제어 가능.
- C++ 클래스 내부에서 파라미터 선언 → 활용 → 수정까지 가능.

7.1 C++ 노드 코드 분석

- src/cpp_parameters_node.cpp

#include <chrono> : 타이머 시간 단위(예: 1000ms)

#include <functional>: std::bind 사용을 위함

#include <string> : 문자열 처리를 위함

#include <rclcpp/rclcpp.hpp> : ROS2 C++ 클라이언트 라이브러리

using namespace std::chrono_literals;

"1000ms" 같은 표현을 가능하게 함

MinimalParam() : Node("minimal_param_node")

부모 Node 생성자 호출 → 노드 이름: minimal_param_node

this->declare_parameter("my_parameter", "world");

파라미터 "my_parameter" 선언, 기본값 world

timer_ = this->create_wall_timer(1000ms, std::bind(&MinimalParam::timer_callback, this));

1초마다 timer_callback 실행

std::string my_param = this->get_parameter("my_parameter").as_string();

현재 파라미터 값 읽어오기 (string 변환)

RCLCPP_INFO(this->get_logger(), "Hello %s!", my_param.c_str());

로그 출력: "Hello {파라미터값}!"

this->set_parameters(reset_param);

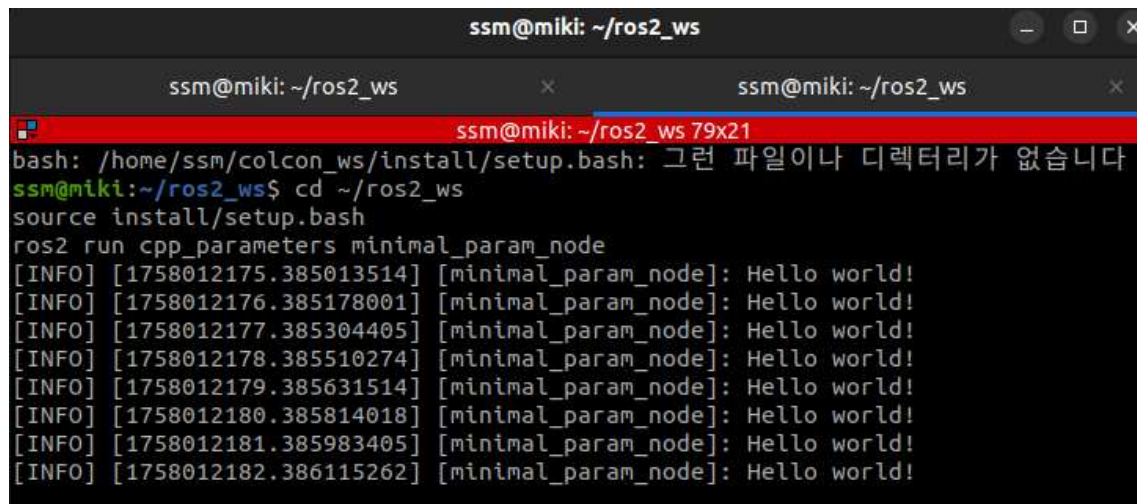
파라미터를 매번 "world"로 재설정

```
rclcpp::init(argc, argv);
ROS2 초기화
rclcpp::spin(std::make_shared<MinimalParam>());
노드 실행 (콜백 처리)
rclcpp::shutdown();
종료 처리
```

7.2 실행 결과

(1) 기본 실행

ros2 run cpp_parameters minimal_param_node

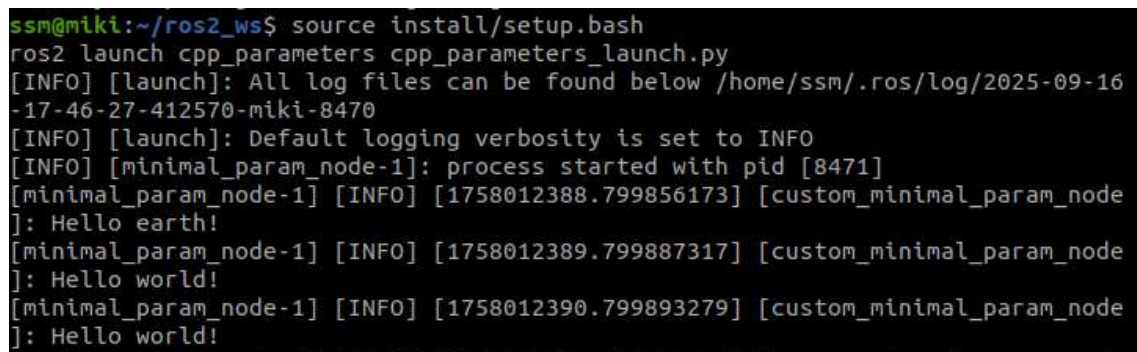


```
ssm@miki: ~/ros2_ws
bash: /home/ssm/colcon_ws/install/setup.bash: 그런 파일이나 디렉터리가 없습니다
ssm@miki:~/ros2_ws$ cd ~/ros2_ws
source install/setup.bash
ros2 run cpp_parameters minimal_param_node
[INFO] [1758012175.385013514] [minimal_param_node]: Hello world!
[INFO] [1758012176.385178001] [minimal_param_node]: Hello world!
[INFO] [1758012177.385304405] [minimal_param_node]: Hello world!
[INFO] [1758012178.385510274] [minimal_param_node]: Hello world!
[INFO] [1758012179.385631514] [minimal_param_node]: Hello world!
[INFO] [1758012180.385814018] [minimal_param_node]: Hello world!
[INFO] [1758012181.385983405] [minimal_param_node]: Hello world!
[INFO] [1758012182.386115262] [minimal_param_node]: Hello world!
```

(2) 런치파일에서 파라미터 변경

launch/cpp_parameters_launch.py 변경

ros2 launch cpp_parameters cpp_parameters_launch.py



```
ssm@miki:~/ros2_ws$ source install/setup.bash
ros2 launch cpp_parameters cpp_parameters_launch.py
[INFO] [launch]: All log files can be found below /home/ssm/.ros/log/2025-09-16-17-46-27-412570-miki-8470
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [minimal_param_node-1]: process started with pid [8471]
[minimal_param_node-1] [INFO] [1758012388.799856173] [custom_minimal_param_node]: Hello earth!
[minimal_param_node-1] [INFO] [1758012389.799887317] [custom_minimal_param_node]: Hello world!
[minimal_param_node-1] [INFO] [1758012390.799893279] [custom_minimal_param_node]: Hello world!
```