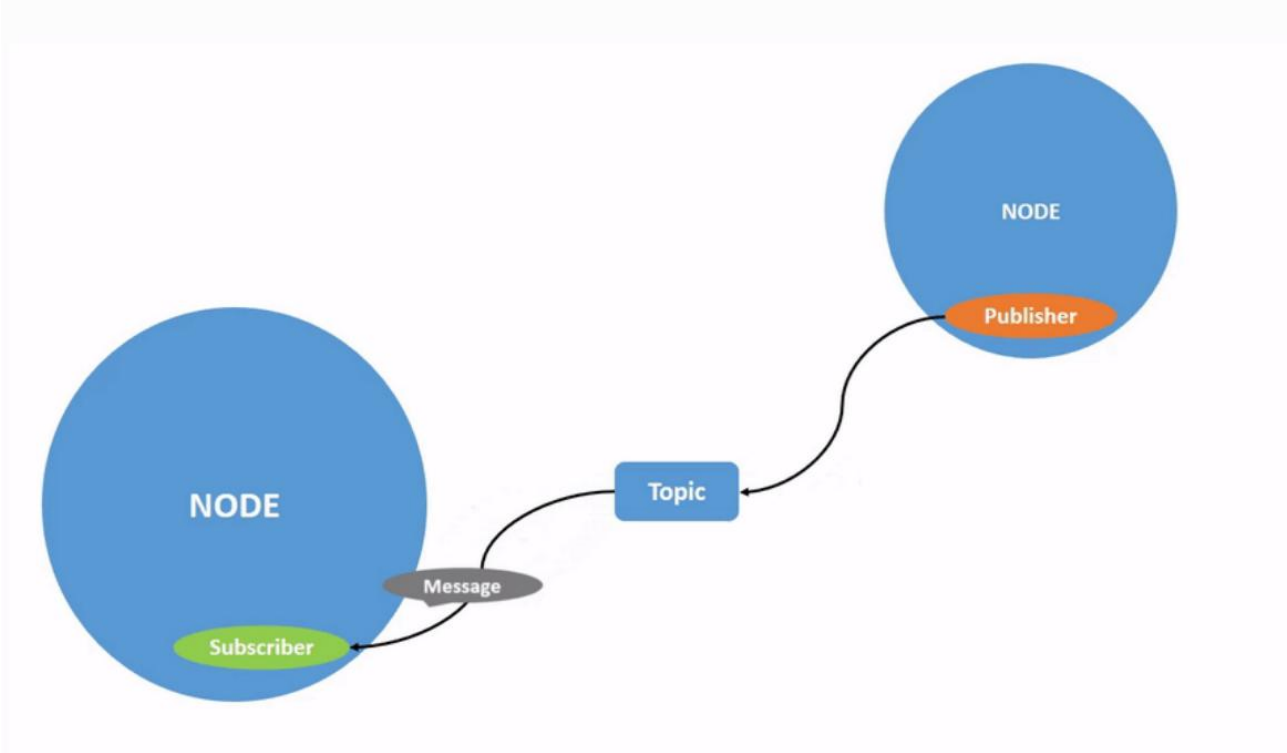
ROS2 day3 hw1 학습 보고서

2025407012/로봇학부/송연우

목차

1. topic

1) 개요: 토픽은 복잡한 시스템을 여러 모듈형 노드로 나눈 ros2에서의 노드 간 메시지 교환을 위한 중간 다리 역할을 하는 핵심 요소입니다. 시스템은 여러 노드로 나누어지며, 노드끼리 다수의 주제, 데이터를 게시할 수 있고 동시에 다수의 주제를 구독할 수 있습니다. 토픽은 시스템의 여러 부분 간 데이터를 이동시키는 주요 방법 중 하나입니다.

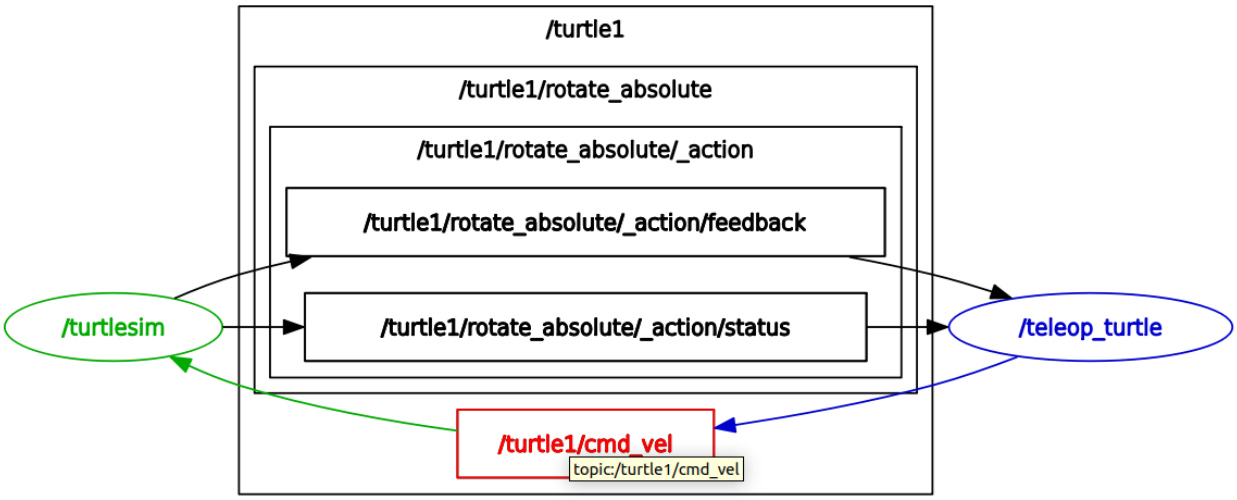


2) 작동 구조: 토픽은 publisher 과 subscriber 이 함께 메시지를 주고받는 연결 지점이며, publisher 이 발행하고 있는 토픽을 subscriber 가 구독하면 메시지를 받을 수 있습니다. 이때 한 토픽을 중심으로 일대다, 다대일, 다대다 노드들이 연결될 수 있으며 복잡한 프로그램에서는 여러 토픽들이 각 노드를 연결하며 시스템이 동작하도록 합니다.

3) 관련 명령어들:

- rqt_graph: 터미널에 ros2 run rqt_graph rqt_graph 를 입력하면 현재 실행 중인 노드, 액션, 토픽, 서비스 등등 메시지의 통신 과정을 시각적으로 볼 수 있습니다.

다음은 turtlesim 의 노드들을 실행하는 중 rqt_graph 를 실행시킨 사진으로,



마우스를 가져다 대면 토픽이 빨갛게 변하면서 해당 토픽으로 통신하는 노드를 강조합니다.

-ros2 topic list: 새 터미널에서 명령을 실행하면 현재 시스템에서 활성화 중인 모든 토픽 목록을 반환합니다.
여기에 -t 를 붙이면 토픽의 메시지 유형도 함께 반환합니다.

-ros2 topic echo: 현재 발행되고 있는 토픽을 터미널창에 출력합니다.

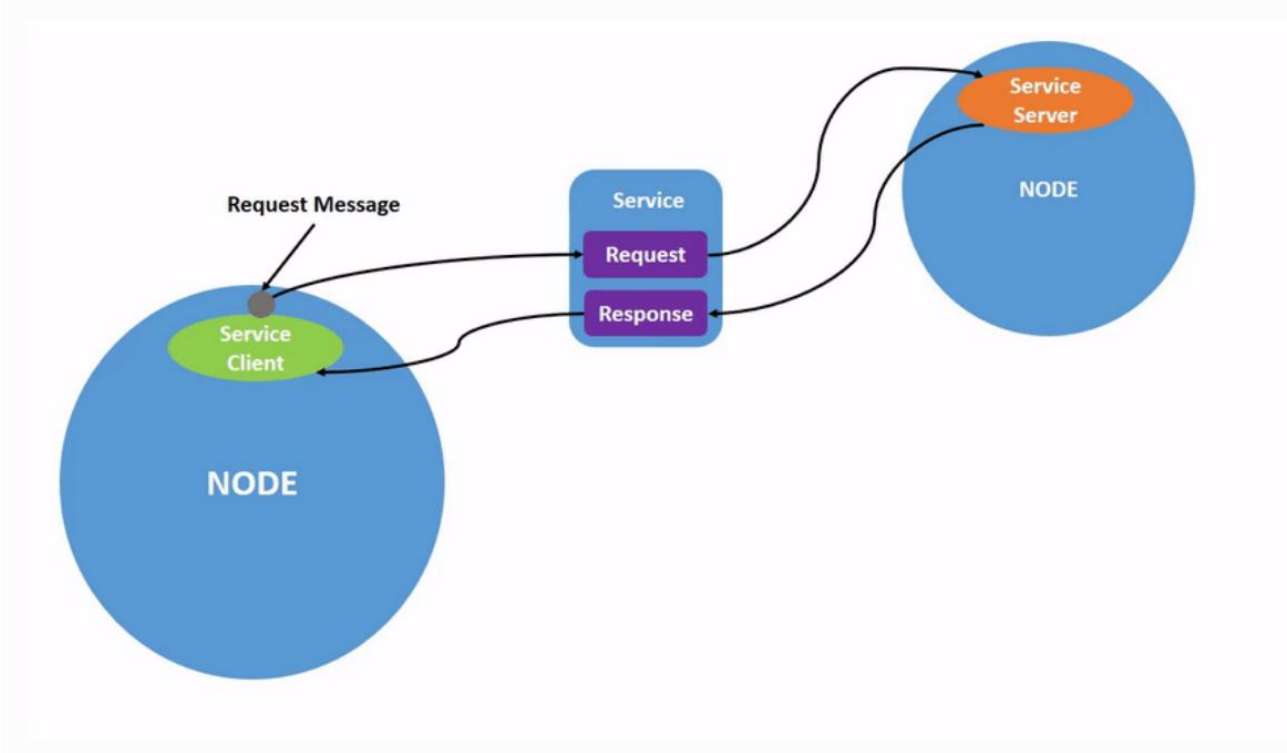
-ros2 topic info: 이 명령어를 터미널창에 입력하면 해당 토픽에서의 통신 상대와 통신 주체들의 수를 반환합니다.

-ros2 interface show: 뒤에 찾고 싶은 메시지의 자료형을 입력하면 해당 메시지가 요구하는 구체적인 데이터 구조를 확인할 수 있습니다.

-ros2 topic pub: 명령어 뒤에 토픽 이름, 메시지 타입, 명령어를 입력하여 직접 topic 에 메시지를 발행할 수 있습니다.

2. service

1) 개요: 서비스는 ros 에서 노드 간의 또다른 통신 방법입니다. 토픽에서는 노드가 토픽을 구독하면 얼마든지 업데이트된 메시지를 받을 수 있었지만, 서비스에서는 다릅니다. 서비스는 호출-응답 모델을 기반으로 하며, 클라이언트가 구체적으로 호출해 메시지를 요구할 때만 데이터를 제공합니다.
이때 다대다, 일대다, 다대일 통신이 모두 가능했던 토픽과는 달리, service 통신에서의 서버 노드는 단 하나만이 존재하며, 클라이언트 노드는 다수로 존재할 수 있습니다.



2) 작동 구조: 서비스는 토픽과 비슷하게 service client 와 service server 노드 중간에서 메시지를 전달하는 다리 역할을 하며, client 노드가 service 에 request 를 보내면 server 에서 그에 대한 답변으로 response 를 보내는 방식으로 작동합니다. 따라서 지속적인 통신이 가능한 topic 과 반대로 일시적이고 제한적인 통신만이 가능하기에 지속적 움직임을 나타내는 데이터보다는 일시적으로 설정을 바꾸는 등의 데이터 운송에 매치되는 경우가 많습니다.

서비스에는 또한 두 가지 유형이 존재합니다. 한 가지는 요청 메시지이고, 다른 하나는 응답 메시지입니다. 요청 메시지는 client 가, 응답 메시지는 server 가 보냅니다.

3) 관련 명령어들:
- ros2 service list: 터미널에서 명령을 실행하면 현재 시스템에서 활성화된 모든 서비스 목록이 반환됩니다.
명령어 뒤에 -t 를 붙이면 topic 에서와 마찬가지로 활성 서비스의 유형이 함께 표시됩니다.

-ros2 service type: 명령어 뒤에 service 의 이름을 입력하면 해당 서비스의 유형을 알 수 있습니다. 예를 들어, trutlesim 의 clear 서비스의 유형은 std_srvs/srv/Empty 임을 알 수 있는데, Empty 유형은 서비스 호출이 요청할 때 데이터를 보내지 않고 응답받을 때 데이터를 받지 않는다는 것을 의미합니다.

-ros2 service : 명령어 뒤에 유형의 이름을 쓰면 해당 유형의 모든 서비스를 찾을 수 있습니다.

-ros2 interface show: 뒤에 찾고 싶은 메시지의 자료형을 입력하면 해당 메시지가 요구하는 구체적인 데이터 구조를 확인할 수 있습니다.

-ros2 service call: 명령어 뒤에 서비스 이름, 유형, 문장을 작성하여 직접 서비스를 호출할 수 있습니다.

3. parameter
1) 개요: parameter 는 노드의 설정 값입니다. 노드는 int, float, bool, arg 등등 다양한 parameter 를 저장할 수 있고, 각 노드가 각각의 parameter 를 관리합니다.

2) 관련 명령어들:
- ros2 param list: 터미널에서 명령을 실행하면 노드와 그에 속한 매개변수를 확인할 수 있습니다.

-ros2 param get: parameter 의 유형과 현재 값을 표시합니다.

-ros2 param set: 노드가 실행 중인 상태에서 parameter 값을 변경하기 위해서 사용합니다.

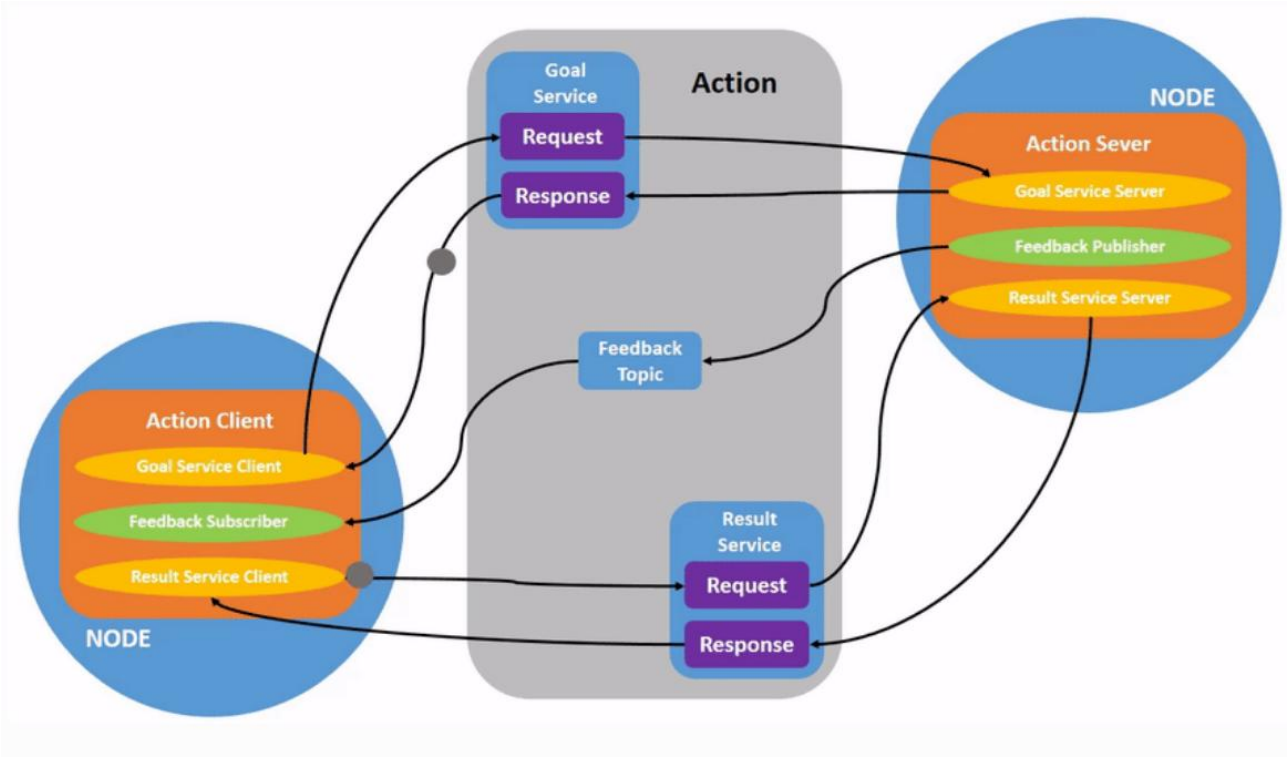
-ros2 param dump: 이 명령어를 터미널창에 입력하면 해당 토픽에서의 통신 상대와 통신 주체들의 수를 반환합니다.

-ros2 param load: 파일에서 현재 실행 중인 노드로 매개변수를 로드합니다.

-ros2 run <package_name> <executable_name> --ros-args --params-file <file_name>: 저장한 매개변수 값을 사용하여 동일한 노드를 run 할 때 사용할 수 있습니다.

4. action

1) 개요: action 은 ros 의 또다른 통신 유형으로, 장기적으로 실행되는 작업을 위한 것입니다. action 은 goal, feedback, result 로 구성되며 topic, service 를 기반으로 합니다. action 은 service 와 비슷한 기능을 하지만 action 을 취소할 수 있습니다. action 은 또한 topic 과도 비슷한 특징이 있는데 service 와 달리 지속적인 피드백을 제공합니다.



2) 작동 구조: action 의 경우, publisher-subscriber 모델과 유사한 client-server 모델을 사용하며, action client 노드가 action server 노드로 goal 을 전송하고, action server 노드는 받은 goal 을 확인하고 feedback 과 result 를 반환합니다.

3) 관련 명령어들:
- ros2 action list: 터미널에서 명령을 실행하면 현재 시스템에서 활성화된 모든 동작이 반환됩니다.
명령어 뒤에 -t 를 붙이면 action 유형을 함께 반환합니다.

-ros2 action info: 명령을 실행하면 각 노드에서 실행되며 학습된 내용을 반환합니다.

-ros2 interface show: action 유형과 함께 입력하면 데이터 유형을 반환합니다.

-ros2 action send_goal: 터미널에서 직접 goal 을 내보낼 수 있습니다.

5. 실습

Using parameters in class (C++)

C++클래스에서 파라미터를 생성하고 사용하는 실습입니다. 작업에서 사용할 워크스페이스로 ros2_ws 를 사용하였고, 해당 디렉토리의 src 디렉토리 내부에 새 패키지를 생성합니다.

```
yu@yu:~$ cd ros2_ws/src
yu@yu:~/ros2_ws/src$ ros2 pkg create --build-type ament_cmake --license Apache-2
.0 cpp_parameters --dependencies rclcpp
going to create a new package
package name: cpp_parameters
destination directory: /home/yu/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['yu <yeonu0070@kw.ac.kr>']
licenses: ['Apache-2.0']
build type: ament_cmake
dependencies: ['rclcpp']
creating folder ./cpp_parameters
creating ./cpp_parameters/package.xml
creating source and include folder
creating folder ./cpp_parameters/src
creating folder ./cpp_parameters/include/cpp_parameters
creating ./cpp_parameters/CMakeLists.txt
yu@yu:~/ros2_ws/src$
```

패키지를 생성하고 나면 위 사진과 같이 생성된 폴더와 파일들이 터미널에 출력됩니다.

처음 패키지를 생성할 때 rclcpp 를 종속성으로 사용했으므로 CmakLists.txt 파일에서 관련 코드를 추가해 줍니다.

1.

```
add_executable(minimal_param_node src/cpp_parameters_node.cpp)
```

add_executable 에서 ros2 run 명령어로 노드를 실행할 시에 참고할 파일을 설정해 줍니다.

2.

```
ament_target_dependencies(minimal_param_node rclcpp)
```

또한 노드를 실행할 때의 종속성을 설정해 줍니다.

3.

```
install(TARGETS
```

install 과 밑의 4번 코드로 실행할 노드의 이름을 적습니다.

4.

```
    minimal_param_node
```

다음으로 cpp 파일의 코드를 작성합니다.

ros2_ws/src/cpp_parameters/src 에서 cpp_parameters_node.cpp 파일을 생성 후, 다음과 같이 작성했습니다.

먼저 종속성 추가 부분입니다. 종속성에서는 chrono, functional, string, rclcpp/rclcpp.hpp 를 추가합니다.

그리고 클래스 MinimalParam 을 선언합니다. 해당 클래스는 rclcpp::Node 를 부모 클래스로 가집니다. 아래의 코드는 public 부분입니다.

1.

```
public:
  MinimalParam()
```

생성자를 선언합니다.

2.

```
  : Node("minimal_param_node")
```

부모 노드 초기화에서 실행할 때 노드의 이름을 정해줍니다.

3.

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

파라미터를 생성하고 이름과 초기값을 지정해 줍니다.

4.

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

계속해서 메시지를 발행하기 위해 타이머 또한 설정합니다. bind()함수를 사용해 타이머의 1초간격 타임아웃 시마다 메시지를 발행하도록 설정합니다.
여기까지가 생성자 코드입니다.

5.

```
  void timer_callback()
```

일정 시간마다 동작하는 timer_callback()함수입니다.

6.

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

노드로부터 생성자에서 생성된 파라미터를 가져와 문자열로 변환 후 my_param 에 저장합니다.

7.

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

저장한 값을 터미널에 띄워 코드의 작동 여부를 알립니다.

8.

```
    std::vector<rclcpp::Parameter> all_new_parameters{rclcpp::Parameter("my_parameter", "w
    this->set_parameters(all_new_parameters);
```

다음으로 파라미터를 변경 한 후 다시 원래대로 돌려놓습니다.

9. 클래스 선언 중 마지막인 private 부분입니다. 타이머를 생성합니다.

```
private:
  rclcpp::TimerBase::SharedPtr timer_;
```

10. main 함수 부분 코드입니다. 첫번째로 초기화를 합니다.

```
  rclcpp::init(argc, argv);
```

11.

```
  rclcpp::spin(std::make_shared<MinimalParam>());
```

spin()함수가 노드를 실행시킵니다.

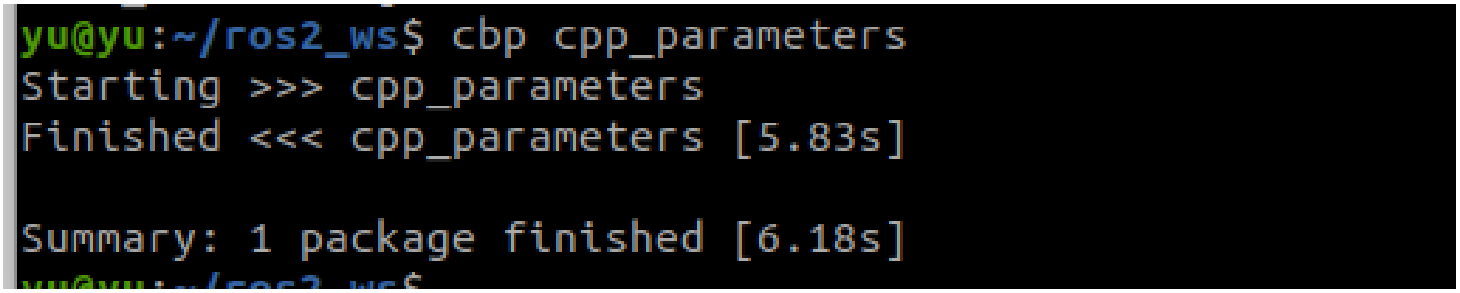
12.

cpp 파일의 마지막 부분입니다. shutdown()함수를 호출 후 종료합니다.

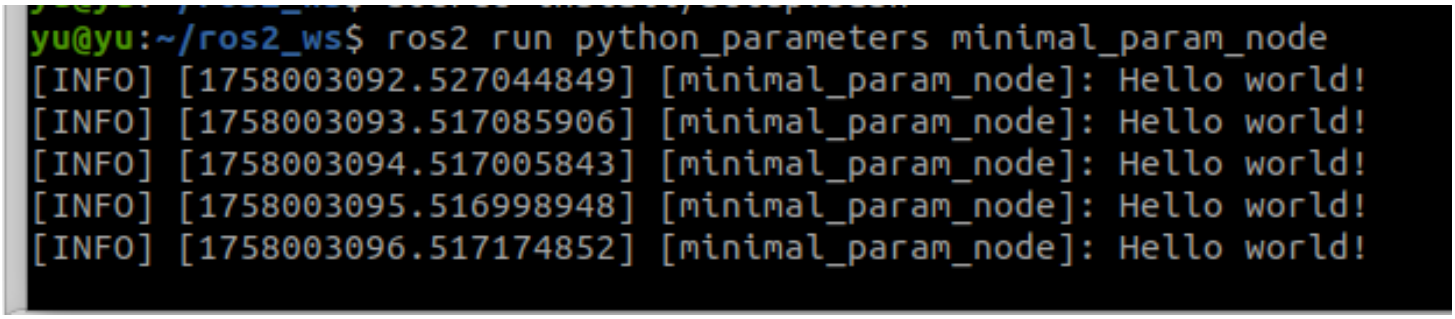
```
rclcpp::shutdown();
return 0;
```

다음으로 패키지를 빌드해 실행하는 부분입니다.

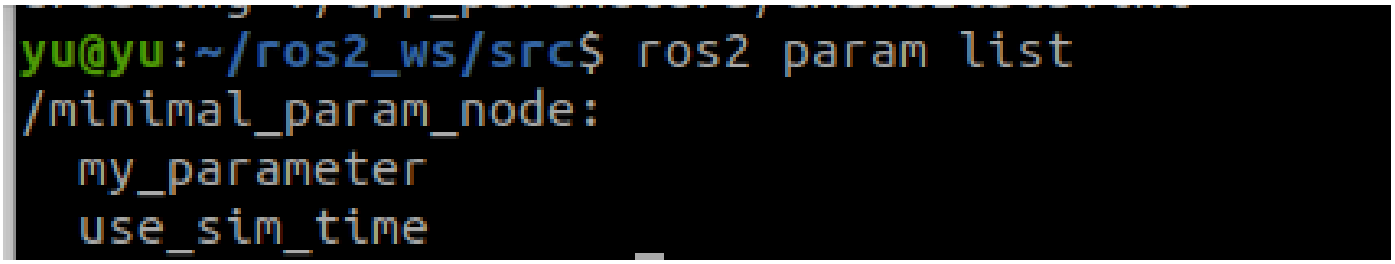
alias 로 colcon build –packages-select 명령어를 cbp 로 저장해 두었으므로 cbp cpp_parameters 로 패키지를 빌드합니다.



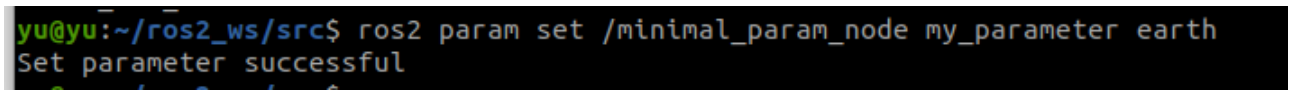
source 로 경로 설정 후 run 해줍니다.

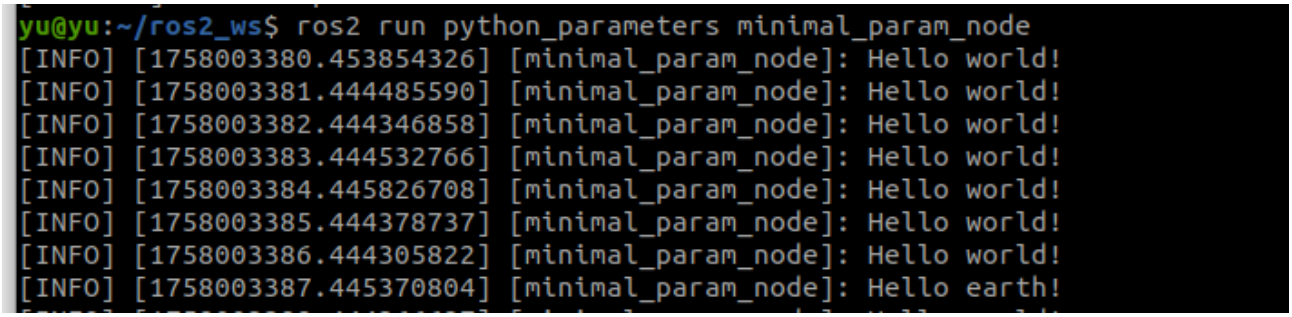


위 사진과 같이 정상적으로 작동하는 모습입니다. 노드가 실행되는 동안 param list 명령어를 터미널에 입력하면 아래 사진과 같이 설정한 파라미터가 표시됩니다.



다음으로 파라미터가 실행 중인 때 다른 터미널에서 set 명령어를 사용해 직접 파라미터를 설정했습니다.





위 사진의 마지막 줄에서 정상적으로 파라미터가 set 되었습니다.
1번 set 된 후 다시 원래 상태로 돌아오도록 코드를 작성했으므로 원상태인 world 를 출력하는 것을 볼 수 있습니다.