ROS2_report_HaJaemin

ros2의 특징들을 먼저 간단히 알아보면

-platform : 3대 운영체제 모두 지원

-real_time 지원

-security : ros1에서의 ros master 보안 문제를 DDS를 도입하여 DDS보안 사양 적용

-communication : DDS 사용 → ros1과 비고하여 메시지 전달, 데이터 전송, 네트워크 대응, 보안 강화

-node_manage : ros master 삭제 →Dynamic Discovery 기능을 이용하여 DDS 미들웨어를 통해 직접 검색하여 노드를 연결 가능해짐

-language : c++ 14, python 3.7 → 최신식

-build system & build tool : [ament]라는 새로운 빌드시스템 사용, [colcon] 빌드 툴 사용

-multiple nodes : 컴포넌트 사용하여 동일한 실행 파일에서 복수의 노드 수행 가능해짐

-message : 컨셉은 변하지 않으나 사용 방법은 많이 바뀜(topic, service, action, ...)

DDS

로봇 운영체제 ros에서 중요시 여기는 메시지, 메시지 통신에 대해 먼저 알아보자. ros에서는 최소 단위의 실행 가능한 프로세서라고 정의하는 node단위의 프로그램 작성.

그리고 하나 이상의 노드 또는 노드 실행을 위한 정보 등을 묶어 놓은 것을 패키지라고 하고, 패키지의 묶음을 메타패키지라고 한다.

이제 노드와 노드 사이에 입출력 데이터를 서로 주고받게 해줘야하는데 주고 받는 방식을 메시지 통신이라고 한다. 이 통신 라이브러리는 DDS의 DDSI-RTPS를 사용하고 있다. 이를 사용함으로써 노드 간의 동적 검색 기능을 지원하여 ros master가 불필요하게 됨. 이 DDS 도입으로 송수신 관련 설정을 사용자가 직접 설정할 수 있게 되었다.

- QoS

간단하게 설명하면 '데이터 통신의 옵션' →사용자가 직접 옵션 설정을 할 수 있게됨, ros1에서는 안됐음

-옵션의 종류들

Reliable : 데이터 손실을 방지함, 신뢰도 우선

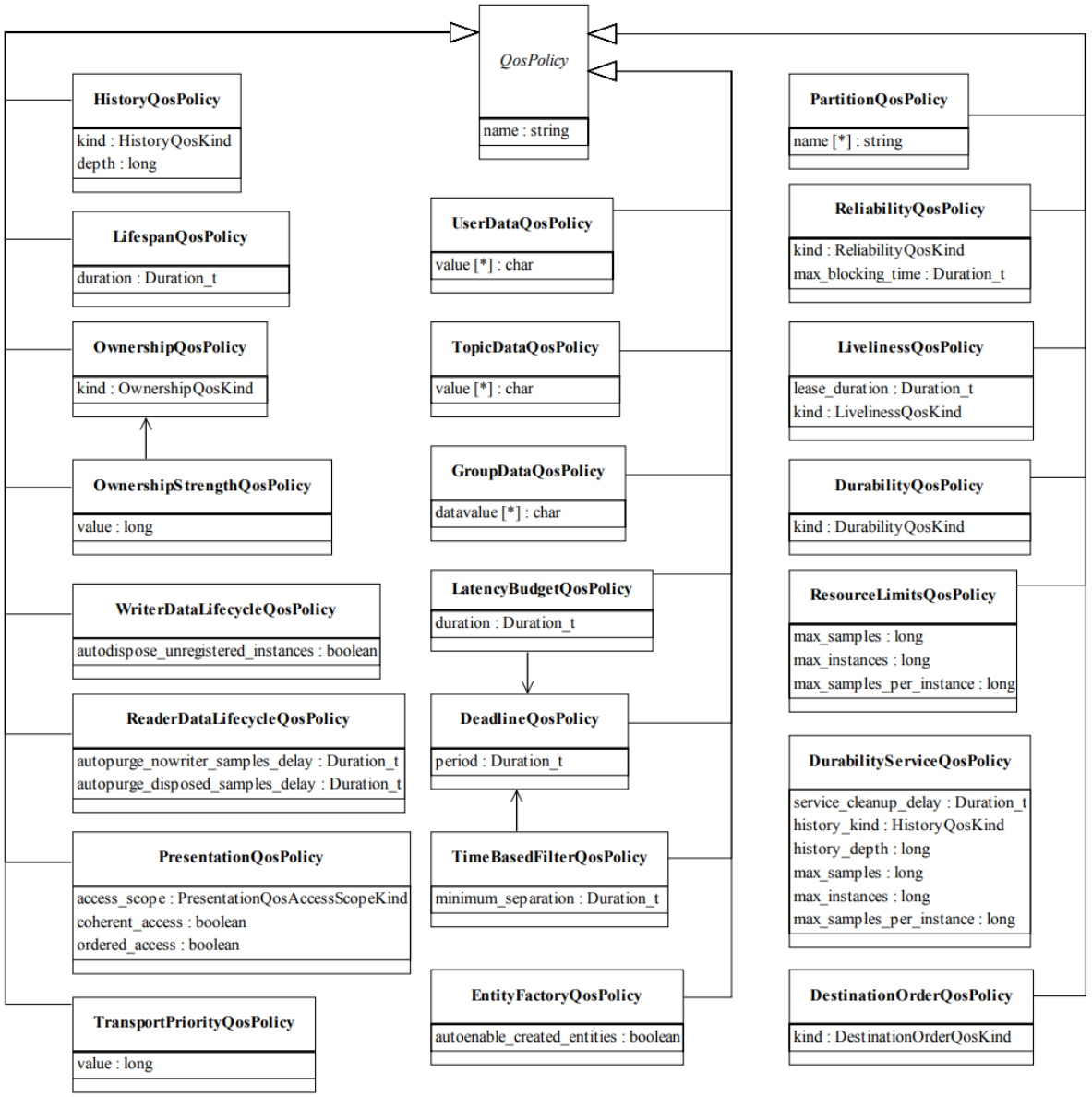
Reliability : 통신 속도를 최우선시 - best effort

History : 통신 상태에 따라 정해진 사이즈만큼의 데이터 보관

Durability : 데이터를 수신하는 sub가 생성되기 전의 데이터를 폐기할지에 대한 설정

Lifespan : 정해진 주기 안에서 수진되는 데이터만 유효 판정하고 그렇지 않은 데이터는 삭제

Liveliness : 정해진 주기 안에서 노드 or 토픽의 생가 확인



유저 Qos 프로파일

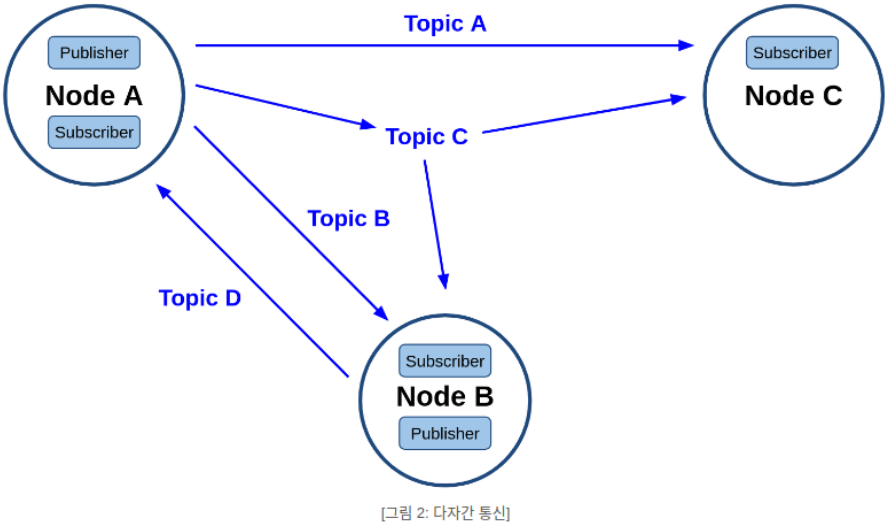
ex) reliability, history, durability, depth를 아래와 같이 원하는 옵션으로 커스텀하게 설정

```
from rclpy.qos import QoSDurabilityPolicy
from rclpy.qos import QoSHistoryPolicy
from rclpy.qos import QoSProfile
from rclpy.qos import QoSReliabilityPolicy
```

노드와 메시지 통신

- topic

비동기식 단방향 메시지 송수신 방식, 1:N, N:1, N:N 통신 가능



[그림 2: 다자간 통신]

-ros2 CLI툴로 topic 상태 확인 명령어들

ros2 topic info : 토픽의 pub/sub 확인

ros2 topic echo : 토픽 내용 확인(단위 →SI)

ros2 topic bw : 송수신 받는 토픽 메시지의 크기 확인

ros2 topic hz : 토픽 주기 확인

ros2 topic delay : RMW→지연시간 존재, 토픽 지연시간 확인

ros2 topic pub : 토픽을 발행

-ros2 bag record

발행하는 토픽을 파일 형태로 저장하고 필요할 때 저장된 토픽을 다시 불러와 동일 타이밍으로 재생할 수 있는 기능이 있다. 이를 rosbag이라고 한다. 이는 매우 유용한 ROS의 기능으로 디버깅에 큰 도움을 준다.

명령어

ros2 bag info : rosbag 파일 정보

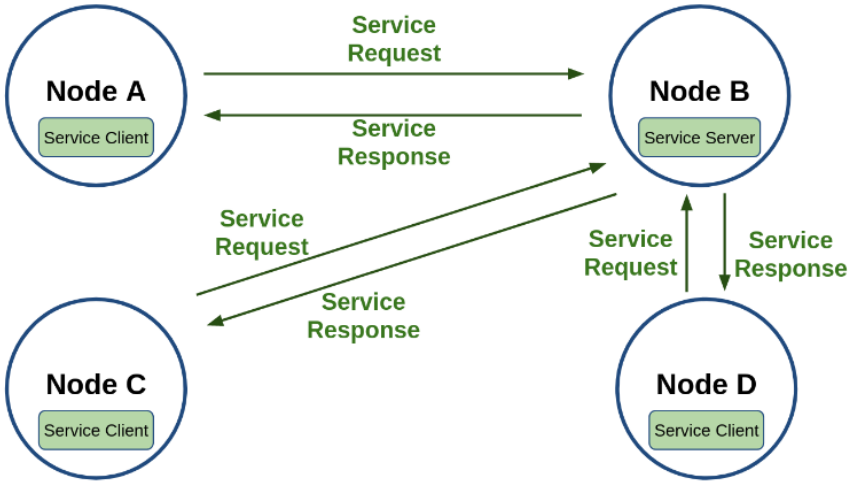
ros2 bag play : 저장한 rosbag 재생

- service

동기식 양방향 메시지 송수신 방식.

service client : 서비스의 요청을 하는 쪽

service server : 서비스의 응답을 하는 쪽, 결과값 전달



[그림 2: 서비스 서버와 클라이언트의 관계]

-CLI 툴 명령어

ros2 service list : 서비스 목록 확인

ros2 service type : 서비스 형태 확인

ros2 service find : 서비스 찾기

ros2 service call : 서비스 요청 → 결과 도출

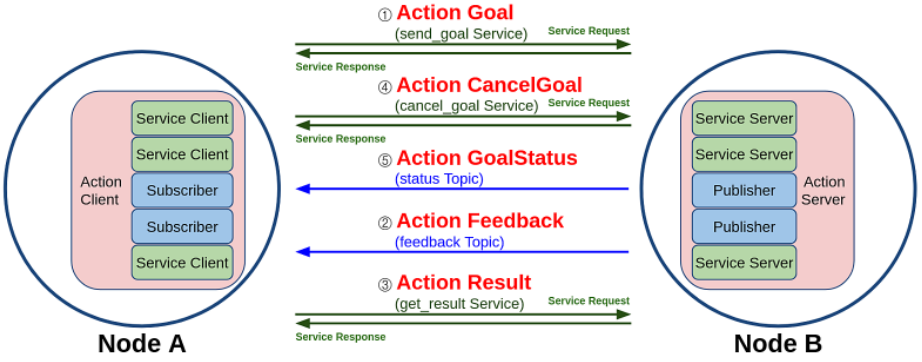
-service Interface

메시지 인터페이스(message interface, msg)에 대해 알아보았다. 서비스 또한 토픽과 마찬가지로 별도의 인터페이스를 가지고 있는데 이를 서비스 인터페이스라 부르며, 파일로는 srv 파일을 가르킨다.

- action

비동기식 + 동기식 양방향 메시지 송수신 방식

액션 목표 Goal를 지정하는 Action client과 액션 목표를 받아 특정 태스크를 수행하면서 중간 결괏값에 해당되는 액션 피드백(Feedback)과 최종 결괏값에 해당되는 액션 결과(Result)를 전송하는 Action server 간의 통신



[그림 2: 액션 목표, 피드백, 결과]

ROS 2에서는 토픽과 서비스 방식을 혼합하여 사용하였다. 그 이유로 토픽으로만 액션을 구성하였을 때 토픽의 특징인 비동기식 방식을 사용하게 되어 ROS 2 액션[10]에서 새롭게 선보이는 목표 전달(send_goal), 목표 취소 (cancel_goal), 결과 받기 (get_result)를 동기식인 서비스를 사용하기 위해서이다. 이런 비동기 방식을 이용하다보면 원하는 타이밍에 적절한 액션을 수행하기 어려운데 이를 원할히 구현하기 위하여 목표 상태(goal_state)라는 것이 ROS 2에서 새롭게 선보였다. 목표 상태는 목표 값을 전달 한 후의 상태 머신을 구동하여 액션의 프로스세를 쫓는 것이다.

-CLI 툴 명령어들

ros2 action list -t : 액션 목록 확인

ros2 action info : 액션 정보

ros2 action send_goal : 액션 목표를 전달 ←피드백이 여기에 포함됨

-액션 Interface

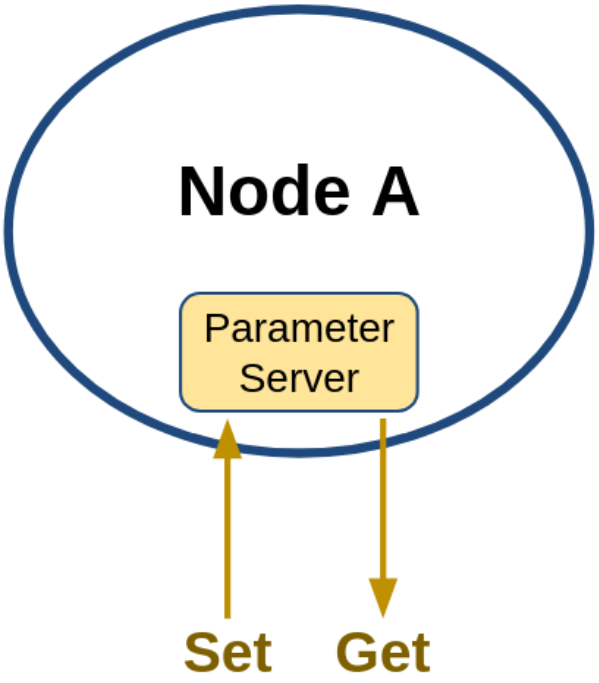
여기서 다룬 액션 또한 토픽, 서비스와 마찬가지로 별도의 인터페이스를 가지고 있는데 이를 액션 인터페이스라 부르며, 파일로는 action 파일을 가르킨다. 액션 인터페이스는 메시지 및 서비스 인터페이스의 확장형이다.

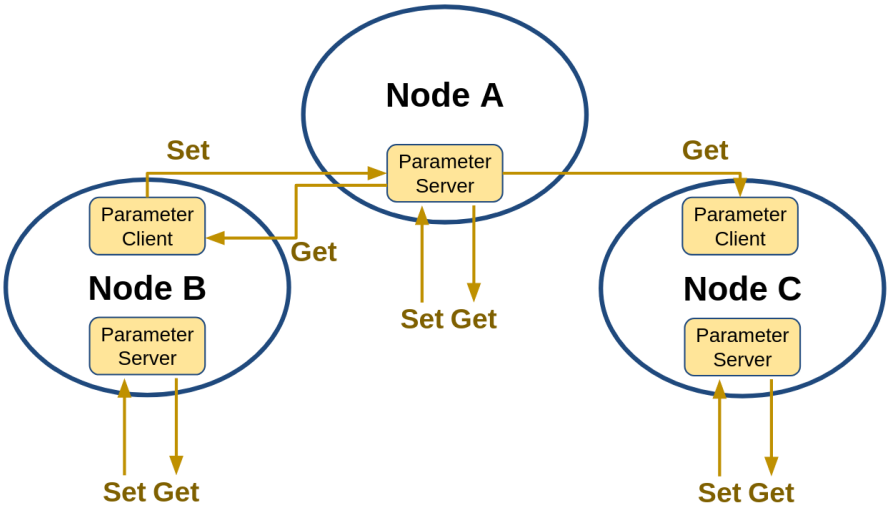
- Interface

노드 간에 데이터를 주고받을 때에는 토픽, 서비스, 액션이 사용되는데 이 때 사용되는 데이터의 형태를 ROS 인터페이스(interface) [7]라고 한다. ROS 인터페이스에는 ROS 2에 새롭게 추가된 IDL(interface definition language)과 ROS 1부터 ROS 2까지 널리 사용 중인 msg, srv, action 이 있다. 토픽, 서비스, 액션은 각각 msg, srv, action interface를 사용하고 있으며 정수, 부동 소수점, 불리언과 같은 단순 자료형을 기본으로 하여 메시지 안에 메시지를 품고 있는 간단한 데이터 구조 및 메시지들이 나열된 배열과 같은 구조도 사용할 수 있다.

- parameter

각 노드에 파라미터 관련 Parameter server를 실행시켜 외부의 Parameter client 간의 통신으로 파라미터를 변경하는 것으로 서비스와 동일하다고 볼 수 있다. 단 노드 내 매개변수 또는 글로벌 매개변수를 서비스 메시지 통신 방법을 사용하여 노드 내부 또는 외부에서 쉽게 지정(Set) 하거나 변경할 수 있고, 쉽게 가져(Get)와서 사용할 수 있게 하는 점에서 목적이 다르다고 볼 수 있다.





-CLI 툴 명령어들

ros2 param list : 파라미터 목록 확인

ros2 param describe : 내용 확인

ros2 param get : 파라미터 읽어오기

ros2 param set : 파라미터 사용

ros2 param dump : 파라미터 저장

ros2 param delete : 파라미터 삭제

ROS2 도구와 CLI 명령어

- CLI

-명령어 기반의 툴로 로봇 억세스 및 거의 모든 ROS 기능을 다룬다.

-개발환경 및 빌드, 테스트 툴(colcon)

-데이터를 기록, 재생, 관리하는 툴(ros2bag)

-colcon, ros2bag 외 20여 가지

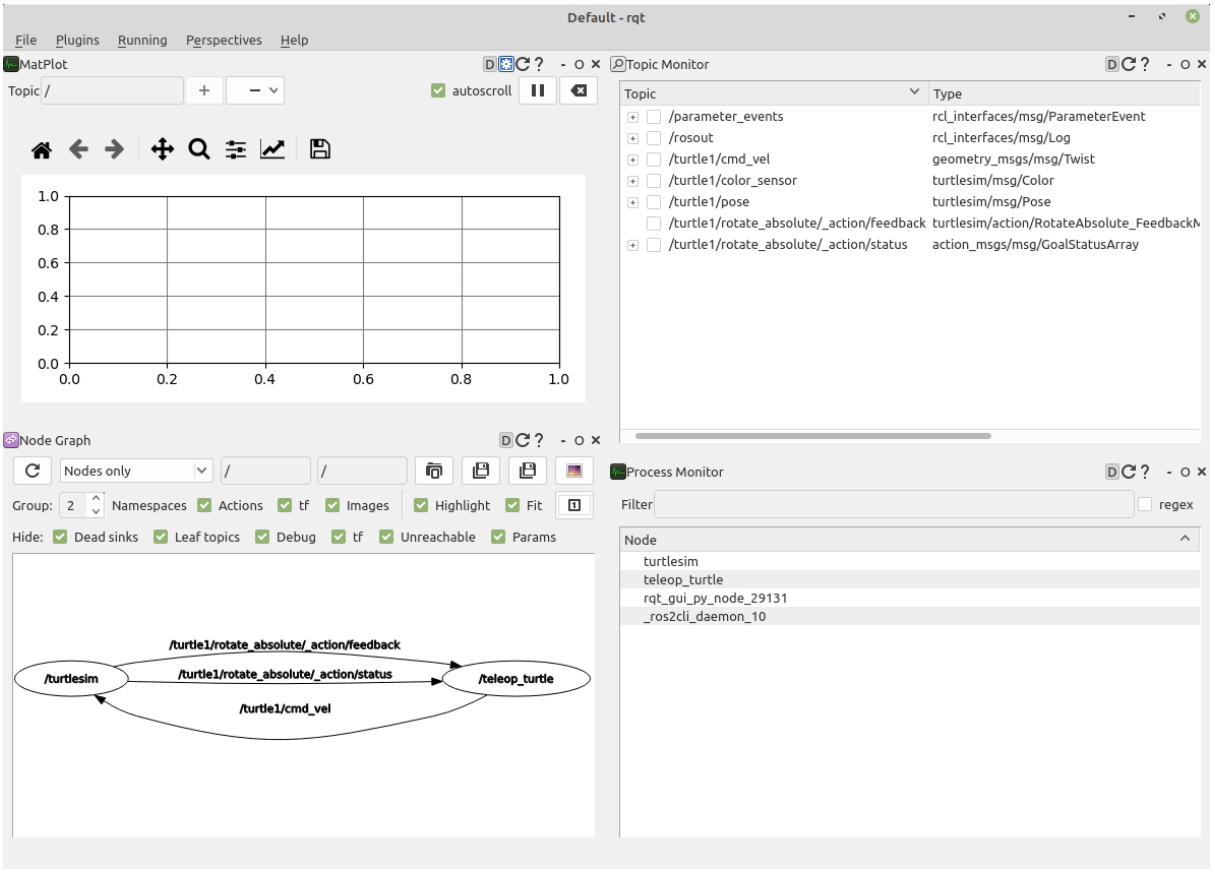
- GUI(RQt)

-그래픽 인터페이스 개발을 위한 Qt 기반 프레임워크 제공

-노드와 그들 사이의 연결 정보 표시(rqt_graph)

-속도, 전압 또는 시간이 지남에 따라 변화하는 데이터를 플로팅(rqt_plot)

-rqt_graph, rqt_plot 외 30여 가지



-플러그인 종류

1. 액션 (Actions)

- Action Type Browser: Action 타입의 데이터 구조를 확인

2. 구성 (Configuration)

- Dynamic Reconfigure: 노드들에서 제공하는 파라미터 값 확인 및 변경
- Launch: roslaunch 의 GUI 버전

3. 내성 (Introspection)

- Node Graph: 실행 중인 노드들의 관계 및 토픽을 확인 가능한 그래프 뷰
- Package Graph: 노드의 의존 관계를 표시하는 그래프 뷰

- Process Monitor: 실행 중인 노드들의 CPU 사용률, 메모리 사용율, 스레드 수 등을 확인

4. 로깅 (Logging)

- Bag: ROS 데이터 로깅
- Console: 노드들에서 발생되는 경고(Warning), 에러(Error) 등의 메시지를 확인
- Logger Level: ROS의 Debug, Info, Warn, Error, Fatal 로거 정보를 선택하여 표시

5. 다양한 툴 (Miscellaneous Tools)

- Python Console: 파이썬 콘솔 화면
- Shell: 쉘(shell)을 구동
- Web: 웹 브라우저를 구동

6. 로봇 (Robot)

- 사용하는 로봇에 따라 계기판(dashboard) 등의 플러그인을 이곳에 추가

7. 로봇툴 (Robot Tools)

- Controller Manager: 컨트롤러 관리에 필요한 플러그인
- Diagnostic Viewer: 로봇 디바이스의 경고 및 에러 확인
- Moveit! Monitor: 로봇 매니퓰레이터 툴인 Moveit! 데이터 확인
- Robot Steering: 로봇에게 병진 속도와 회전 속도를 토픽으로 발행하는 GUI 툴
- Runtime Monitor: 실시간으로 노드들에서 발생되는 에러 및 경고를 확인

8. 서비스 (Services)

- Service Caller: 실행 중인 서비스 서버에 접속하여 서비스를 요청
- Service Type Browser: 서비스 타입의 데이터 구조를 확인

9. 토픽 (Topics)

- Message Publisher: 메시지 발행
- Message Type Browser: 메시지 타입의 데이터 구조 확인
- Topic Monitor: 토픽 목록 확인 및 사용자가 선택한 토픽의 정보를 확인

10. 시각화 (Visualization)

- Image View: 카메라의 영상 데이터를 확인
- Navigation Viewer: 로봇 네비게이션의 위치 및 목표지점 확인
- Plot: 2차원 데이터 플롯 GUI 플러그인, 2차원 데이터의 도식화
- Pose View: 현재 TF의 위치 및 모델의 위치 표시
- RViz: 3차원 시각화 툴인 RViz 플러그인
- TF Tree: tf 관계를 트리로 나타내는 그래프 뷰

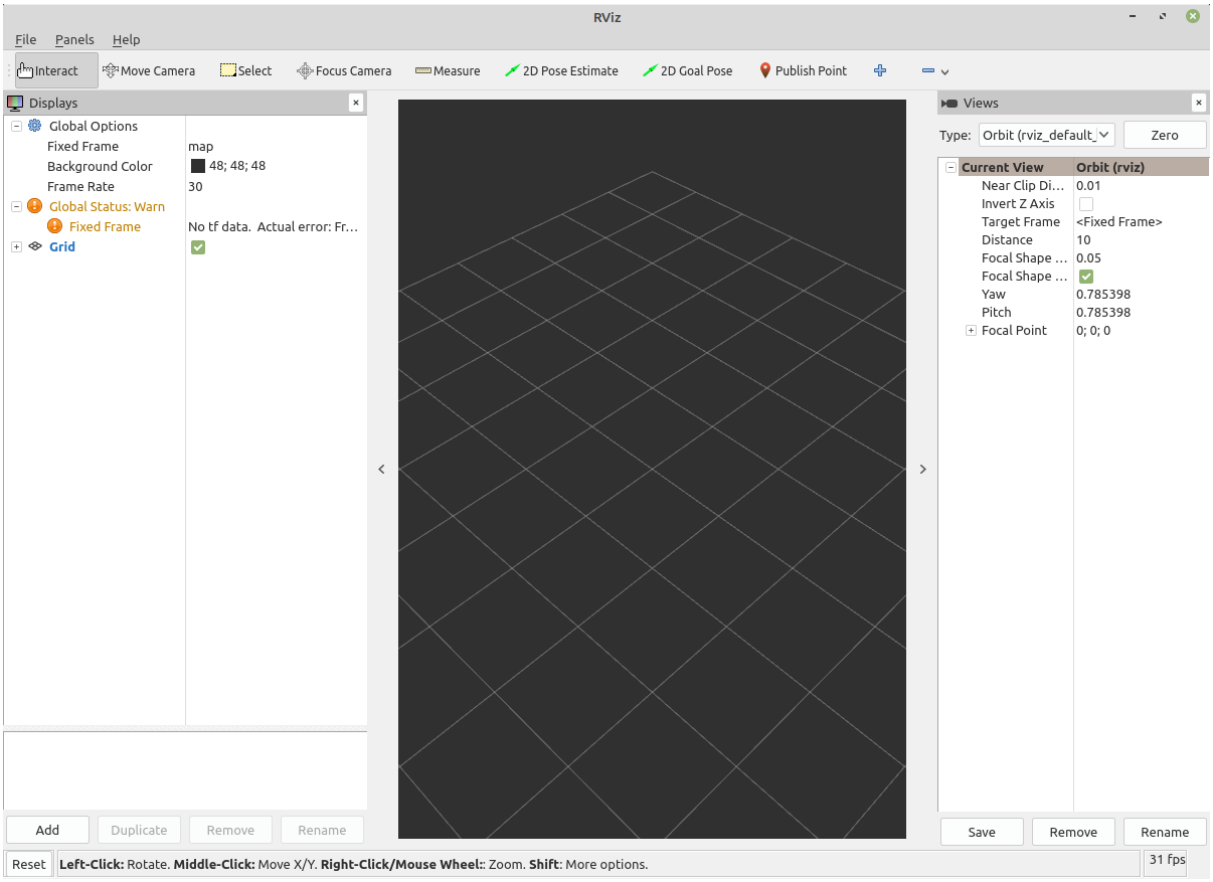
.

- RViz

-3차원 시각화툴

-레이저, 카메라 등의 센서 데이터를 시각화

-로봇 외형과 계획된 동작을 표현

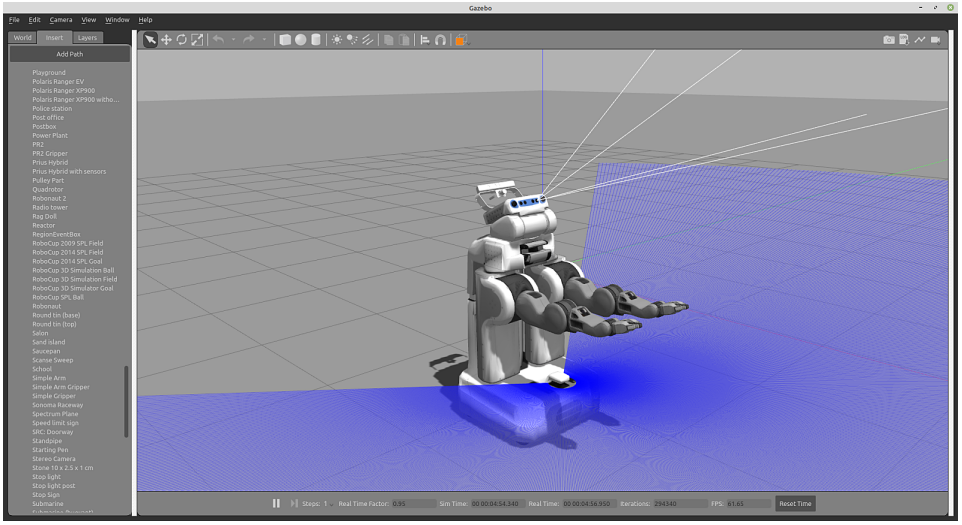


- Gazebo

-3차원 시뮬레이터

-물리 엔진을 탑재, 로봇, 센서, 환경 모델 등을 지원

-타 시뮬레이터 대비 ROS와의 높은 호환성



ros2의 파일 시스템

ROS 2의 기본적인 폴더 및 파일 구성에 대해서 알아보자. ROS 2에서 소프트웨어 구성을 위한 기본 단위가 패키지(package)로써 ROS의 응용프로그램은 패키지 단위로 개발되고 관리된다. 패키지는 ROS의 최소 단위의 실행 프로세서인 노드(node)를 하나 이상 포함하거나 다른 노드를 실행하기 위한 런치(launch)와 같은 실행 및 설정 파일들을 포함하게 된다.

ROS 패키지 설치는 바이너리 형태로 제공되어 별도의 빌드과정 없이 바로 실행하는 방법과 해당 패키지의 소스 코드를 직접 다운로드 한 후 사용자가 빌드하여 사용하는 방법이 있다. 이는 패키지 사용 목적에 따라 달리 사용된다. 만약, 해당 패키지를 수정하여 사용하고자 할 경우나 소스 코드 내용을 확인할 필요가 있다면 후자의 설치 방법을 이용하면 된다.

ROS 1에서는 `catkin_ws`와 같이 특정 워크스페이스를 확보하고 하나의 워크스페이스에서 모든 작업을 다 했는데 ROS 2에서는 복수의 독릭된 워크스페이스를 사용할 수 있어서 작업 목적 및 패키지 종류별로 관리할 수 있게 되었다.

ROS 1에서의 catkin은 패키지를 빌드 한 후 devel 이라는 폴더에 코드를 저장한다. 이 폴더는 패키지를 설치할 필요없이 패키지를 사용할 수 있는 환경을 제공한다. 이를 통해 파일 복사를 피하면서 사용자는 파이썬 코드를 편집하고 즉시 코드 실행할 수 있었다. 단 이러한 기능은 매우 편리한 기능이지만 패키지를 관리하는 측면에서 복잡성을 크게 증가시켰다. 이에 ROS 2에서는 패키지를 빌드 한 후 설치해야 패키지를 사용할 수 있도록 바뀌었다.

ros2의 빌드 시스템과 빌드 툴

우선, 빌드 시스템(build system)과 빌드 툴 (build tools)을 나누어 비교하고 ROS에서는 어떻게 역할 분담이 되어 있는지 알아보자. 결론부터 말하자면 빌드 시스템과 빌드 툴의 큰 차이는 단일 패키지를 대상으로 하느냐 전체 패키지를 대상으로 하느냐이다. 즉, 빌드 시스템은 단일 패키지를 대상으로 하며, 빌드 툴은 시스템 전체를 대상으로 한다.

단순히 생각하면 빌드 시스템만을 이용하면 ROS의 패키지들을 사용할 수 있는 것처럼 보인다. 하지만 ROS에서는 수많은 패키지가 함께 빌드하여 실행시키는 구조이기 때문에 각 패키지별로 서로 다른 빌드 시스템을 호출하고 패키지들의 종속성은 매우 얽혀있는 경우에는 이 얽혀있는 의존성 실타래를 풀고 토폴로지 순서대로 빌드 해야만 한다. 이때에 사용되는 것이 ROS의 빌드 툴이다. ROS 빌드 툴은 각 패키지에 기술되어 있는 종속성 그래프를 해석하고 토폴로지 순서로 각 패키지에 대한 특정 빌드 시스템을 호출한다. 이러한 빌드 툴은 ROS의

개발 환경을 설정하고 빌드 시스템을 호출하고 빌드 된 패키지를 사용하도록 실행 환경을 구성하게 된다.

ROS 2에서는 새로운 빌드 시스템인 'ament'을 사용한다. ament도 크게는 두 가지인데 그 중 가장 많이 사용되는 ament_cmake는 ROS 1에서 사용되는 빌드 시스템인 catkin[4]의 업그레이드 버전으로 CMake의 빌드 설정 파일인 CMakeList.txt에 기술된 빌드 설정을 기반으로 빌드를 수행하게 된다.

ROS 2 패키지를 생성하는 방법으로는 두 가지인데 하나는 직접 패키지 폴더를 만들고 그 안에 파일 시스템에 필수적인 `package.xml`이나 `CMakeLists.txt` 또는 `setup.py` 등을 포함시켜주고 소스 코드를 작성하는 것과 ros2cli [7] 명령어를 이용하는 것이다.

-ros2 pkg create : 패키지 생성 명령어

-빌드 명령어

```
$ cd ~/robot_ws && colcon build --symlink-install --packages-
```

물리량, 좌표, 시간 표현

-물리량

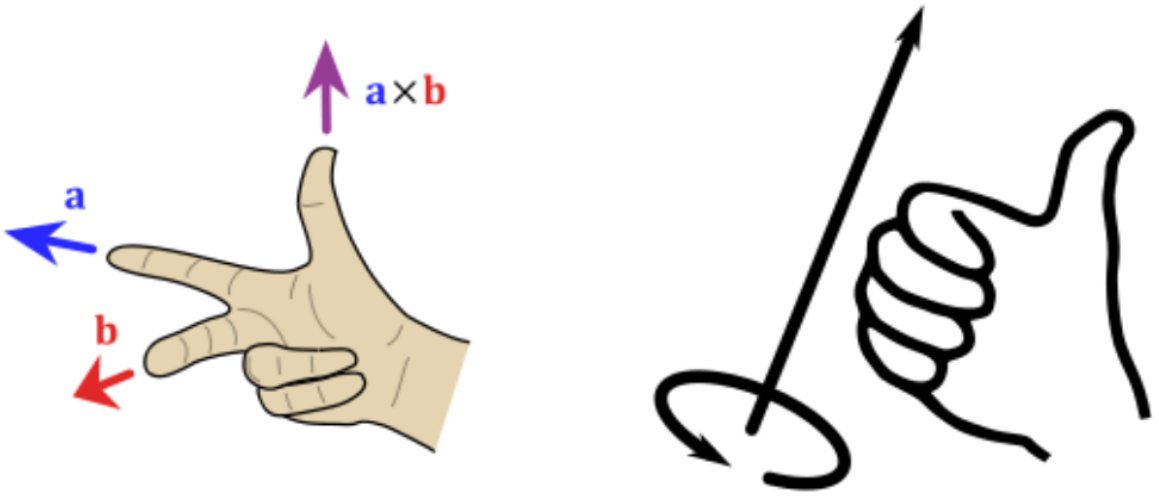
msg, service, action 인터페이스는 노드 간에 데이터를 주고받는데 사용된다. 그 내용에는 어떤 `**이름**`으로 어떠한 `**자료형**`으로 데이터를 보내느냐가 기술되어 있다.

ROS 커뮤니티에서는 ROS 프로그래밍에 사용하는 표준 단위로 세계적으로 가장 널리 사용되고 있는 국제단위계인 SI 단위

물리량	단위 (SI unit)	물리량	단위 (SI derived unit)
length (길이)	meter (m)	angle (평면각)	radian (rad)
mass (질량)	kilogram (kg)	frequency (주파수)	hertz (Hz)
time (시간)	second (s)	force (힘)	newton (N)
current (전류)	ampere (A)	power (일률)	watt (W)
		voltage (전압)	volt (V)
		temperature (온도)	celsius (°C)
		magnetism (자기장)	tesla (T)

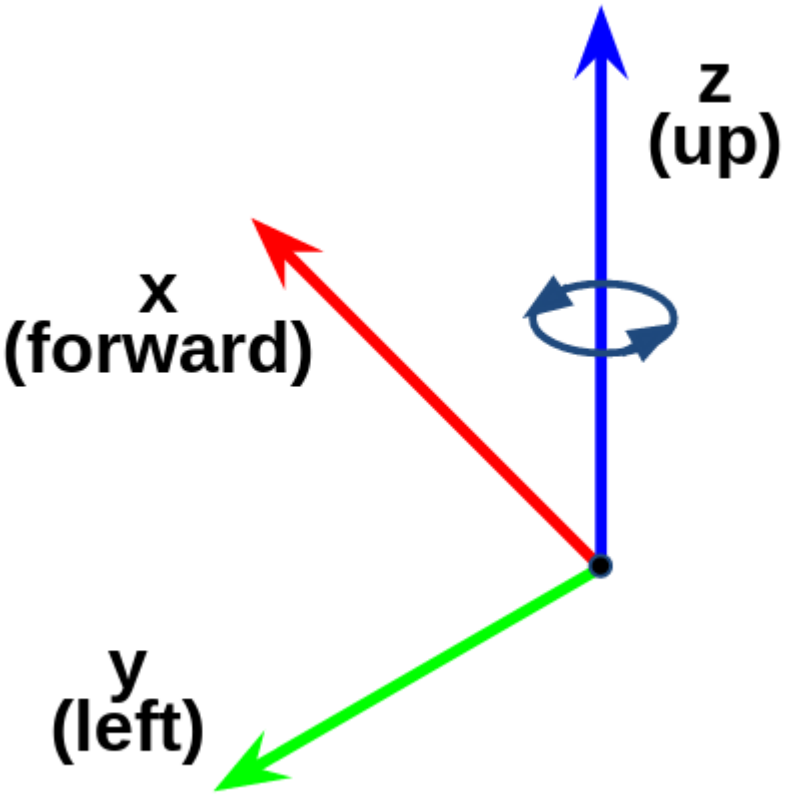
-좌표

ROS 커뮤니티에서는 모든 좌표계를 삼차원 벡터 표기관습을 이해하기 위한 일반적인 기억법인 오른손 법칙 에 따라 표현한다. 기본적으로 회전 축의 경우에는 그림과 같이 검지, 중지, 엄지를 축을 사용하며 오른손의 손가락을 감는 방향이 정회전 + 방향이다.



-축

ROS 커뮤니티에서는 그림과 같이 축 방향 (Axis Orientation)으로 x forward, y left, z up 을 사용한다. 시각화 툴 RViz나 3차원 시뮬레이터 Gazebo에서 이러한 기본 3축의 표현에 있어서 헤깔리지 않도록 RGB의 원색으로 표현하는데 순서대로 Red는 x 축, Green은 y 축, Blue는 z축을 의미한다.



-시계와 시간

다수의 센서를 사용하는 로봇은 시간에 따른 각 센서 값의 변화량과 그 센서들 간의 시간 동기화가 매우 중요하다. ROS 2는 여러 노드들이 서로 통신하며 다양한 정보(센서 값, 알고리즘을 수행한 결괏값 등)들을 주고 받기 때문에 해당 정보들이 발간된 정확한 시간이 필수적이다.

ROS 2에서 사용하는 **기본 시계는 System Clock**

이며 rclcpp 에서는 std::chrono라이브러리를 rclpy 는 time 모듈을 캡슐화하여 사용하고 있다.

-시간 추상화

ROS 2 에서는 기본 시계 이외에도 타임머신처럼 동작하는 시계도 사용이 가능하다. 타임머신 처럼 동작하는 시계는 과거 어느 시점으로 시간을 돌려 줄 수도 있고, 시간을 더 빠르게 흘

러가게 할 수도 있으며 또 시간을 멈출 수 있는 기능을 가지고 있다. 이 시계는 과거에 기록한 데이터를 다룰 때(ros2bag)나 로봇 시뮬레이션(gazebo, ignition)에서 사용할 수 있고, 이를 통해 사용자는 개발된 알고리즘을 보다 효율적으로 디버깅할 수 있다.

- **System Time**

System Clock을 사용한 시간을 말한다. 이는 단조 증가하지만 타임서버와의 동기화를 통해 시간이 거꾸로 가는 경우도 있다. 예를 들면 server pc와 remote pc 간의 데이터 통신을 원활히 하기 위해서는 시간을 동기화 시켜야만 하는데, 아래 명령어를 각 pc에 입력하여 특정 서버의 시간으로 동기화 할 수 있다.

- **ROS Time**

보통 시뮬레이션 환경에서 시간을 조절하기 위해 많이 사용한다. 노드가 생성되기 전에 노드가 기본으로 가지고 있는 파라미터 중의 하나인 use_sim_time 을 통해 사용할 수 있으며, use_sim_time 이 True 로 설정된 노드는 /clock 토픽을 구독할 때까지 시간을 0으로 초기화 한다.

- **Steady Time**

Hardware timeouts 를 사용한 시간을 말한다. 위에서 알아본 시간들과 달리 무조건 단조 증가(monotonic) 한다는 특성을 가진다.

-time API

- **Time**

Time 클래스는 시간을 다룰 수 있는 오퍼레이터를 제공하며 그 결과를 seconds 혹은 nanoseconds 단위로 반환해준다. seconds 는 double 형이고, nanoseconds 는 unsinged int의 64비트형을 가지며 nanoseconds 가 seconds 보다 더 정확한 시간을 반환한다.

- **Duration**

Duration[14] 클래스는 순간의 시간(timestamp, 5시 30분 29초)이 아닌 기간(3시간 후, 1시간 전)을 다룰 수 있는 오퍼레이터를 제공하며 그 결과를 seconds 혹은 nanoseconds 단위로 반환해준다. Duration 은 이전 시간을 이야기할 수 있고 이는 음수로 표기된다.

Duration은 Time 과의 연산이 가능하여 보다 직관적으로 시간을 다룰 수 있도록 한다. 예를 들면, 아래 예제 코드와 같이 실제 시간보다 1초 느린 값을 간단히 계산할 수 있다.

- **Rate**

Rate클래스는 반복문에서 특정 주기를 유지시켜 주는 API를 제공한다. 아래 예시를 보면 WallRate 클래스의 생성자에 헤르츠 단위로 주기를 설정한 후, 반복문 가장 아래 줄에 sleep() 함수로 주기를 맞춰 주는 것을 확인할 수 있다. (Rate는 System Clock을 사용하고 WallRate는 Steady Clock을 사용해서 시간을 확인한다.) 하지만 ROS 2 에서는 콜백 함수를 사용하는 Timer[16] API를 제공하고 있기에 이를 사용하는 것을 추천한다.