# Week5

## 17. ROS 2 도구와 CLI 명령어

ROS 2에는 다양한 개발 도구가 있다. ROS 2 도구에는 그 형태에 따라 4가지로 분류할 수 있다.

CLI 형태의 Command-Line Tools, GUI 형태의 RQt, 3차원 시각화툴인 RViz, 3차원 시뮬레이터 Gazebo이다.

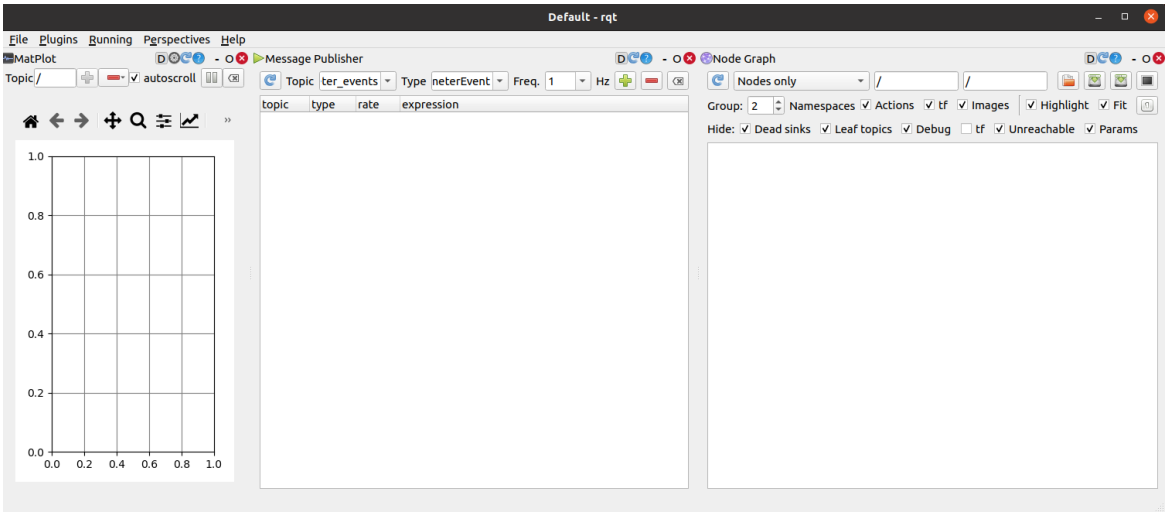
### CLI 기반 Command-Line Tools

- 명령어 기반의 툴로 로봇 엑세스 및 거의 모든 ROS 기능을 다룬다.
- 개발환경 및 빌드, 테스트 툴(colcon)
- 데이터를 기록, 재생, 관리하는 툴(ros2bag)
- 그 외 20여가지

```
$ ros2 extension_points -a
ros2action.verb: The extension point for 'action' verb extensions
ros2bag.verb: The extension point for 'bag' verb extensions
ros2cli.command: The extension point for 'command' extensions
ros2cli.daemon.verb: The extension point for 'daemon' verb extensions
ros2component.verb: The extension point for 'node' verb extensions
ros2doctor.verb: The extension point for 'doctor' verb extensions
ros2interface.verb: Extension point for 'interface' verb extensions
ros2lifecycle.verb: The extension point for 'lifecycle' verb extensions
ros2multicast.verb: The extension point for 'msg' verb extensions
ros2node.verb: The extension point for 'node' verb extensions
ros2param.verb: The extension point for 'param' verb extensions
ros2pkg.verb: The extension point for 'pkg' verb extensions
ros2service.verb: The extension point for 'service' verb extensions
ros2topic.verb: The extension point for 'topic' verb extensions
sros2.verb: The extension point for 'security' verb extensions
```

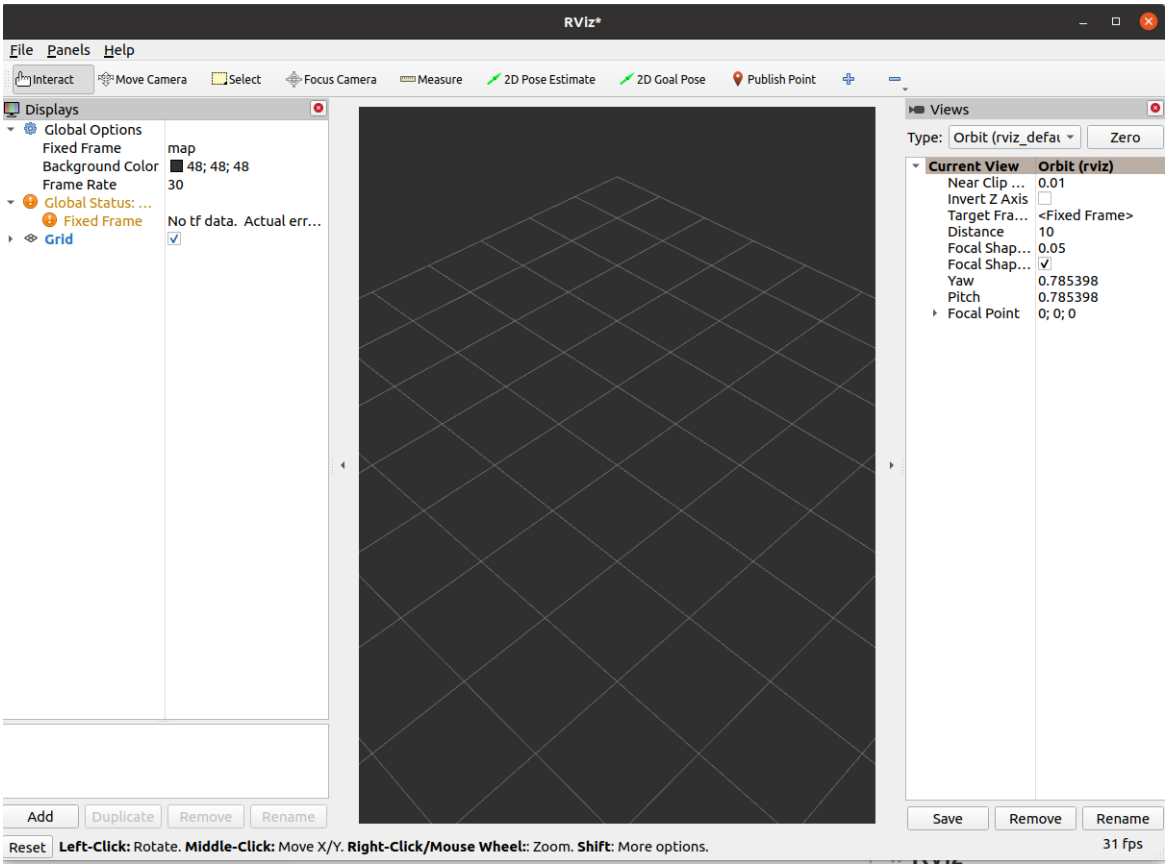
### GUI 기반 RQt

- 그래픽 인터페이스 개발을 위한 Qt 기반 프레임워크 제공
- 노드와 그들 사이의 연결 정보 표시(rqt_graph)
- 속도, 전압 등 시간이 지남에 따라 변화하는 데이터를 플로팅(rqt_plot)
- 그 외 30여가지



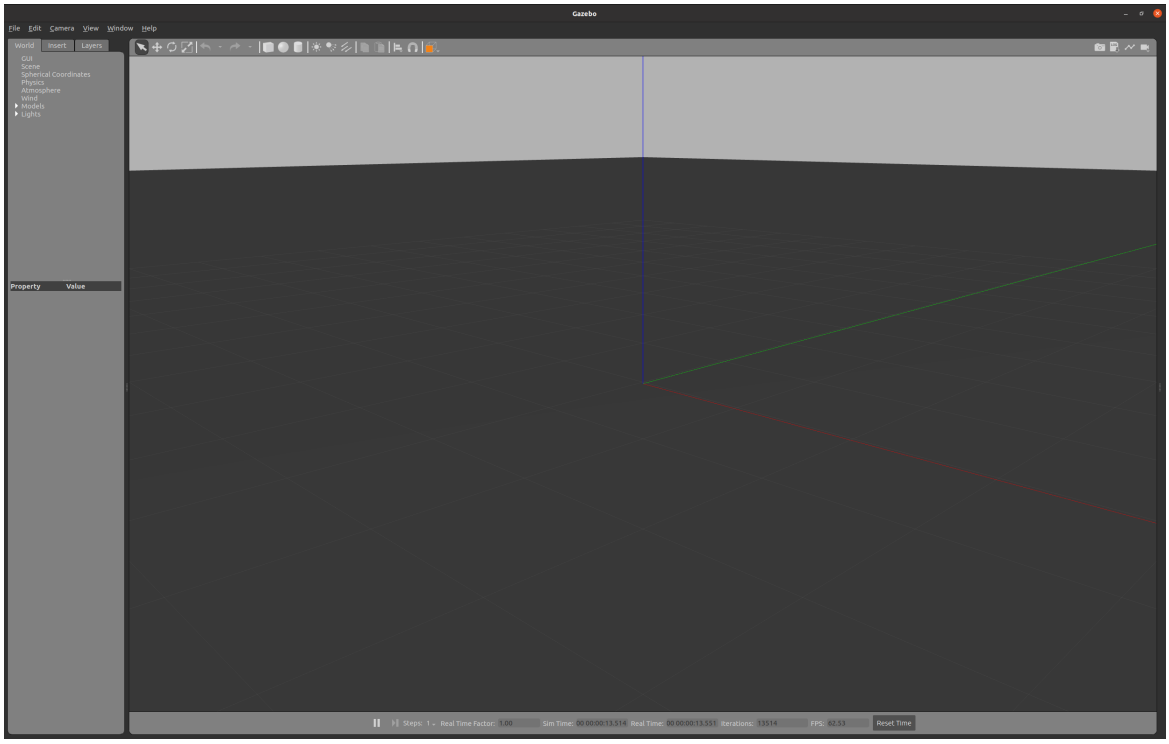

## RViz

- 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




### ROS 2 CLI 사용법

ROS 2 CLI 사용법은 기본적으로 터미널 창에서 각 명령어가 하나씩 수행되는 방식이다. 터미널 창에 다음과 같이 ros2를 시작으로 verbs, sub-verbs, options, arguments등을 입력해 실행시킨다.

```
$ ros2 [verbs] [sub-verbs] [options] [arguments]
```

ROS 2 CLI는 auto-completion을 지원하기 때문에 고유한 entry-point인 ros2를 입력하고 한칸의 스페이스 입력후 Tab을 누르면 사용가능한 명령어를 보여주어 선택할 수 있다. 또한 명령어를 다 입력하지 않고 Tab을 눌러주면 자동완성해주는 기능도 있기 때문에 적절히 사용하면 빠르게 명령어를 찾고 실행할 수 있다.

```
$ ros2 [Tab]
action          daemon          extensions      lifecycle       param           security        wtf
bag             doctor          interface       multicast       pkg             service
component       extension_points launch         node            run             topic
```

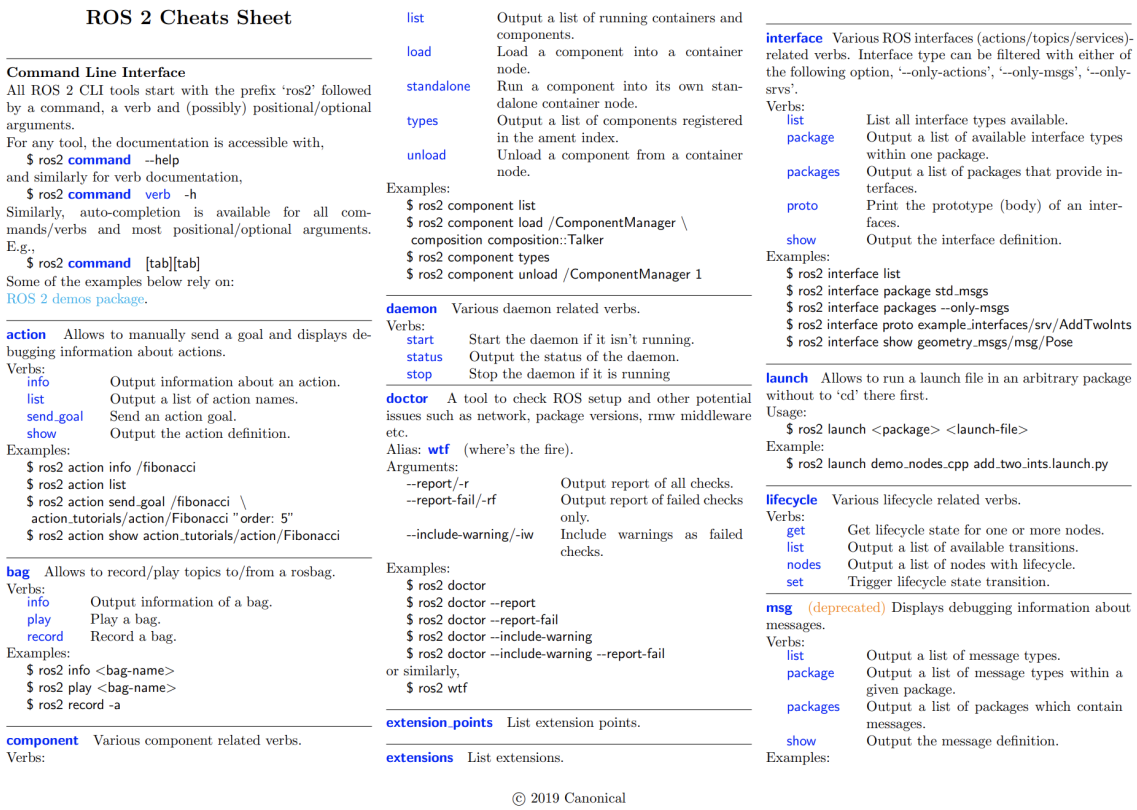
또한 -h(—help) 옵션을 사용하면 간단한 사용법도 나오니 참고하자

```
$ ros2 launch -h
usage: ros2 launch [-h] [-n] [-d] [-p | -s] [-a] package_name [launch_file_name] [launch_arguments [launch_arguments ...]]

Run a launch file

positional arguments:
  package_name          Name of the ROS package which contains the launch file
  launch_file_name      Name of the launch file
  launch_arguments      Arguments to the launch file; '<name>:=<value>' (for duplicates, last one wins)

optional arguments:
  -h, --help            show this help message and exit
  -n, --noninteractive  Run the launch system non-interactively, with no terminal associated
  -d, --debug           Put the launch system in debug mode, provides more verbose output.
  -p, --print, --print-description
                        Print the launch description to the console without launching it.
  -s, --show-args, --show-arguments
                        Show arguments that may be given to the launch file.
  -a, --show-all-subprocesses-output
                        Show all launched subprocesses' output by overriding their output configuration using the
                        OVERRIDE_LAUNCH_PROCESS_OUTPUT envvar.
```

## ROS 2 CLI 종류와 각 sub-verbs의 기능

20여가지의 ROS 2 CLI를 익히는 방법은 필요할 때 적절히 사용하면서 체득하는 것이라고 생각한다. 그래서 추후에 CLI가 등장할 때마다 사용하면서 체득하는 것으로 하고 일단 모든 CLI를 담고 있는 Cheat Sheet를 가져왔다.

cli_cheats_sheet.pdf



ROS 2 Cheats Sheet

**Command Line Interface**
All ROS 2 CLI tools start with the prefix 'ros2' followed by a command, a verb and (possibly) positional/optional arguments.
For any tool, the documentation is accessible with,
$ ros2 command --help
and similarly for verb documentation,
$ ros2 command verb -h
Similarly, auto-completion is available for all commands/verbs and most positional/optional arguments. E.g.,
$ ros2 command [tab][tab]
Some of the examples below rely on:
ROS 2 demos package.

**action** Allows to manually send a goal and displays debugging information about actions.
Verbs:
info — Output information about an action.
list — Output a list of action names.
send_goal — Send an action goal.
show — Output the action definition.
Examples:
$ ros2 action info /fibonacci
$ ros2 action list
$ ros2 action send_goal /fibonacci \
action_tutorials/action/Fibonacci "order: 5"
$ ros2 action show action_tutorials/action/Fibonacci

**bag** Allows to record/play topics to/from a rosbag.
Verbs:
info — Output information of a bag.
play — Play a bag.
record — Record a bag.
Examples:
$ ros2 info <bag-name>
$ ros2 play <bag-name>
$ ros2 record -a

**component** Various component related verbs.
Verbs:
list — Output a list of running containers and components.
load — Load a component into a container node.
standalone — Run a component into its own standalone container node.
types — Output a list of components registered in the ament index.
unload — Unload a component from a container node.
Examples:
$ ros2 component list
$ ros2 component load /ComponentManager \
composition composition::Talker
$ ros2 component types
$ ros2 component unload /ComponentManager 1

**daemon** Various daemon related verbs.
Verbs:
start — Start the daemon if it isn't running.
status — Output the status of the daemon.
stop — Stop the daemon if it is running

**doctor** A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.
Alias: **wtf** (where's the fire).
Arguments:
--report/-r — Output report of all checks.
--report-fail/-rf — Output report of failed checks only.
--include-warning/-iw — Include warnings as failed checks.
Examples:
$ ros2 doctor
$ ros2 doctor --report
$ ros2 doctor --report-fail
$ ros2 doctor --include-warning
$ ros2 doctor --include-warning --report-fail
or similarly,
$ ros2 wtf

**extension_points** List extension points.

**extensions** List extensions.

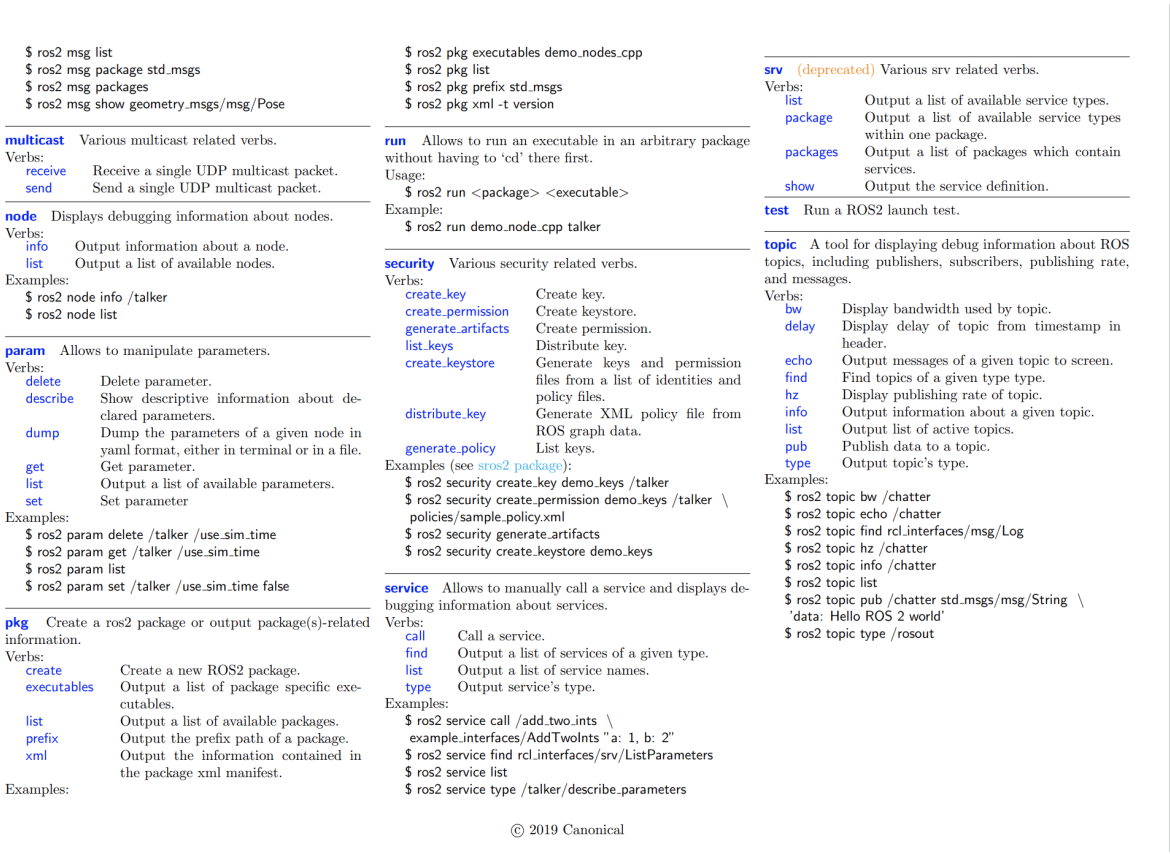
**interface** Various ROS interfaces (actions/topics/services)-related verbs. Interface type can be filtered with either of the following option, '--only-actions', '--only-msgs', '--only-srvs'.
Verbs:
list — List all interface types available.
package — Output a list of available interface types within one package.
packages — Output a list of packages that provide interfaces.
proto — Print the prototype (body) of an interfaces.
show — Output the interface definition.
Examples:
$ ros2 interface list
$ ros2 interface package std_msgs
$ ros2 interface packages --only-msgs
$ ros2 interface proto example_interfaces/srv/AddTwoInts
$ ros2 interface show geometry_msgs/msg/Pose

**launch** Allows to run a launch file in an arbitrary package without to 'cd' there first.
Usage:
$ ros2 launch <package> <launch-file>
Example:
$ ros2 launch demo_nodes_cpp add_two_ints.launch.py

**lifecycle** Various lifecycle related verbs.
Verbs:
get — Get lifecycle state for one or more nodes.
list — Output a list of available transitions.
nodes — Output a list of nodes with lifecycle.
set — Trigger lifecycle state transition.

**msg** (deprecated) Displays debugging information about messages.
Verbs:
list — Output a list of message types.
package — Output a list of message types within a given package.
packages — Output a list of packages which contain messages.
show — Output the message definition.
Examples:

© 2019 Canonical



```
$ ros2 msg list
$ ros2 msg package std_msgs
$ ros2 msg packages
$ ros2 msg show geometry_msgs/msg/Pose
```

**multicast** Various multicast related verbs.
Verbs:
- receive — Receive a single UDP multicast packet.
- send — Send a single UDP multicast packet.

**node** Displays debugging information about nodes.
Verbs:
- info — Output information about a node.
- list — Output a list of available nodes.

Examples:
```
$ ros2 node info /talker
$ ros2 node list
```

**param** Allows to manipulate parameters.
Verbs:
- delete — Delete parameter.
- describe — Show descriptive information about declared parameters.
- dump — Dump the parameters of a given node in yaml format, either in terminal or in a file.
- get — Get parameter.
- list — Output a list of available parameters.
- set — Set parameter

Examples:
```
$ ros2 param delete /talker /use_sim_time
$ ros2 param get /talker /use_sim_time
$ ros2 param list
$ ros2 param set /talker /use_sim_time false
```

**pkg** Create a ros2 package or output package(s)-related information.
Verbs:
- create — Create a new ROS2 package.
- executables — Output a list of package specific executables.
- list — Output a list of available packages.
- prefix — Output the prefix path of a package.
- xml — Output the information contained in the package xml manifest.

Examples:
```
$ ros2 pkg executables demo_nodes_cpp
$ ros2 pkg list
$ ros2 pkg prefix std_msgs
$ ros2 pkg xml -t version
```

**run** Allows to run an executable in an arbitrary package without having to 'cd' there first.
Usage:
```
$ ros2 run <package> <executable>
```
Example:
```
$ ros2 run demo_node_cpp talker
```

**security** Various security related verbs.
Verbs:
- create_key — Create key.
- create_permission — Create keystore.
- generate_artifacts — Create permission.
- list_keys — Distribute key.
- create_keystore — Generate keys and permission files from a list of identities and policy files.
- distribute_key — Generate XML policy file from ROS graph data.
- generate_policy — List keys.

Examples (see sros2 package):
```
$ ros2 security create_key demo_keys /talker
$ ros2 security create_permission demo_keys /talker \
 policies/sample_policy.xml
$ ros2 security generate_artifacts
$ ros2 security create_keystore demo_keys
```

**service** Allows to manually call a service and displays debugging information about services.
Verbs:
- call — Call a service.
- find — Output a list of services of a given type.
- list — Output a list of service names.
- type — Output service's type.

Examples:
```
$ ros2 service call /add_two_ints \
 example_interfaces/AddTwoInts "a: 1, b: 2"
$ ros2 service find rcl_interfaces/srv/ListParameters
$ ros2 service list
$ ros2 service type /talker/describe_parameters
```



**srv** (deprecated) Various srv related verbs.
Verbs:
- list — Output a list of available service types.
- package — Output a list of available service types within one package.
- packages — Output a list of packages which contain services.
- show — Output the service definition.

**test** Run a ROS2 launch test.

**topic** A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.
Verbs:
- bw — Display bandwidth used by topic.
- delay — Display delay of topic from timestamp in header.
- echo — Output messages of a given topic to screen.
- find — Find topics of a given type type.
- hz — Display publishing rate of topic.
- info — Output information about a given topic.
- list — Output list of active topics.
- pub — Publish data to a topic.
- type — Output topic's type.

Examples:
```
$ ros2 topic bw /chatter
$ ros2 topic echo /chatter
$ ros2 topic find rcl_interfaces/msg/Log
$ ros2 topic hz /chatter
$ ros2 topic info /chatter
$ ros2 topic list
$ ros2 topic pub /chatter std_msgs/msg/String \
 'data: Hello ROS 2 world'
$ ros2 topic type /rosout
```

## 18. ROS 2 GUI 개발을 위한 RQt

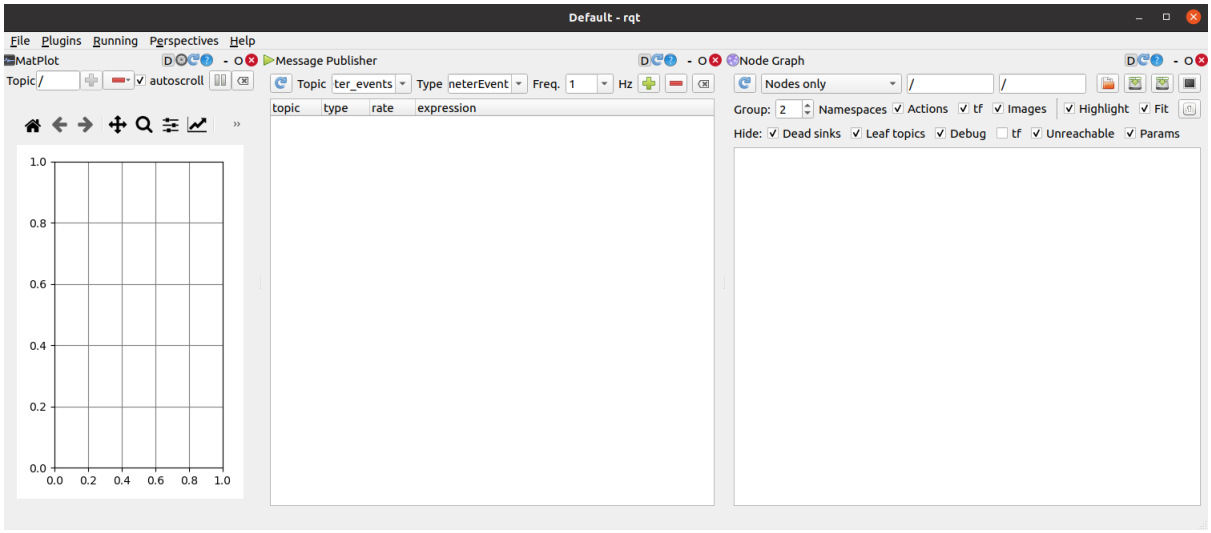
RQt는 플러그인 형태로 다양한 도구와 인터페이스를 구현할 수 있는 그래픽 사용자 인터페이스 프레임워크이며 다양한 목적의 GUI툴을 모아둔 ROS의 종합 GUI 툴박스이다.

RQt의 실행방법은 3가지다.

첫 번째 실행 방법은 RQt를 실행하여 메뉴에서 원하는 플러그인을 선택해 실행하는 것이다.

터미널 창에 rqt를 입력해 실행하는 GUI창이 실행되는데 Plugins 메뉴에서 원하는 플러그인을 선택하면 해당 플러그인이 화면에 표시된다.

```
$ rqt
```




두 번째  실행 방법은 ros2 run 명령어를 사용해 각 rqt 관련 패키지들의 노드들을 하나씩 실행시키는 방법이다.

```
$ ros2 run rqt_msg rqt_msg
```

세 번째 실행 방법은 지정된 단축 명령어를 사용하는 것이다. 단 많지는 않다.

```
$ rqt_graph
$ rqt_topic
```

## RQt 플러그인의 종류 및 사용예시

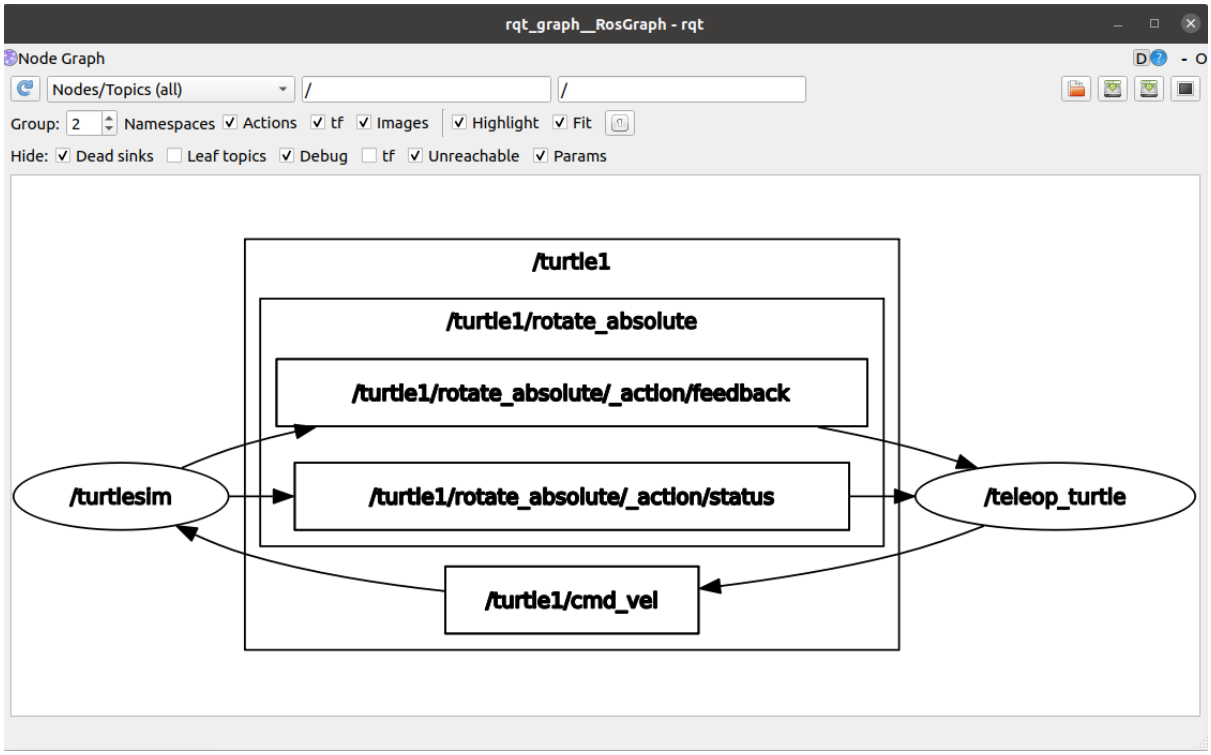
이번 장에서 모든 플러그인을 알아보기에는 내용이 너무 많으므로 사용 빈도가 높고 중요한 플러그인만 알아보자.

### 1. Node Graph

- 메뉴: Plugins → Introspection → Node Graph

현재 개발환경에서 실행한 노드들의 관계를 그래프 형태로 표시하는 RQt 플러그인이다.

rqt_graph로 실행하는 것이 더 빠르다.

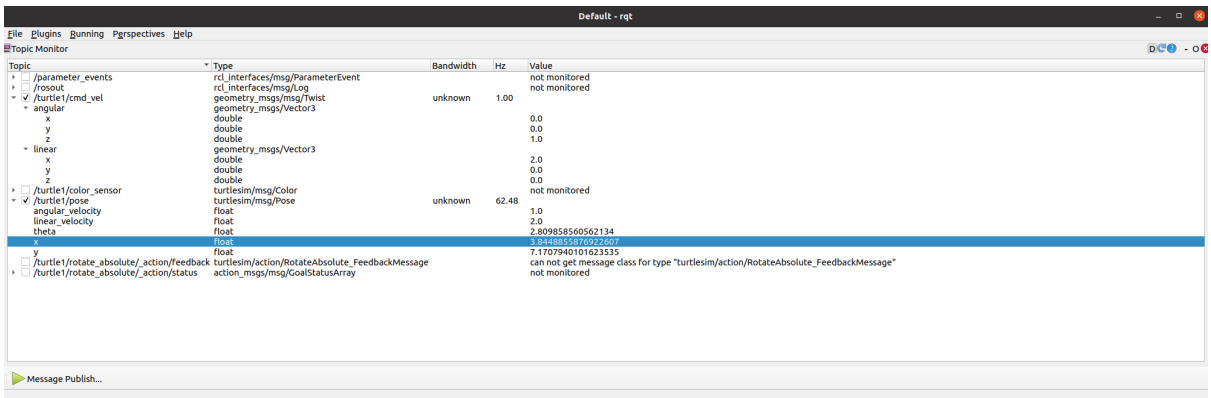



## 2. Topic Monitor

- 메뉴: Plugins → Topics → Topic Monitor

현재 개발환경에서 실행한 노드가 사용하는 토픽의 목록을 확인할 수 있고, 사용자가 선택한 토픽의 토피이름, 타입, 대역폭, 퍼블리시 주기값과 같은 토픽 정보를 확인할 수 있는 RQt 플러그인이다.

ros2 topic 명령어의 GUI 버전이다.



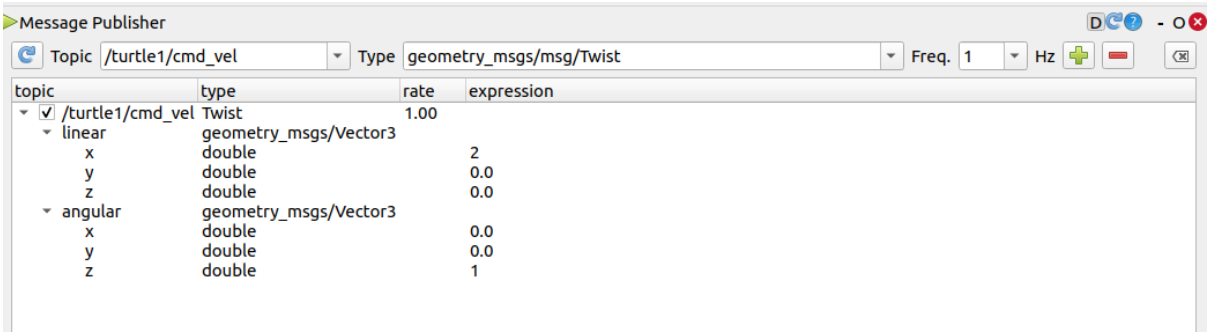

## 3. Message Publisher

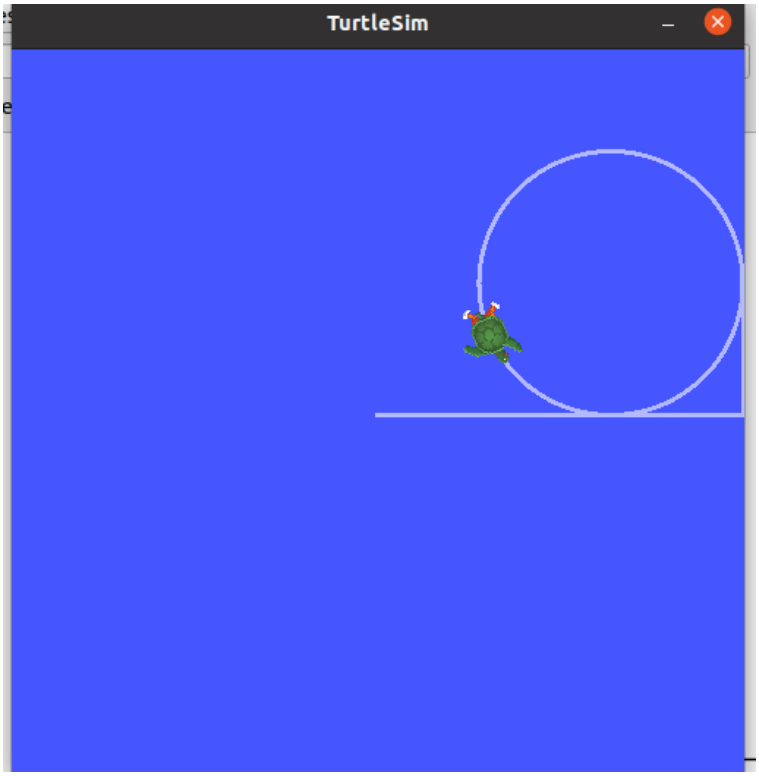
- 메뉴: Plugins → Topics → Message Publisher

특정 토픽 이름으로 특정 타입의 토픽을 퍼블리시하는 토픽 퍼블리셔 역할을 하는 RQt 플러그인이다.

ros2 topic pub 명령어의 GUI 버전이다.

아래와 같이 토픽을 퍼블리시하면 turtlesim의 turtle이 움직이는 것을 볼 수 있다.



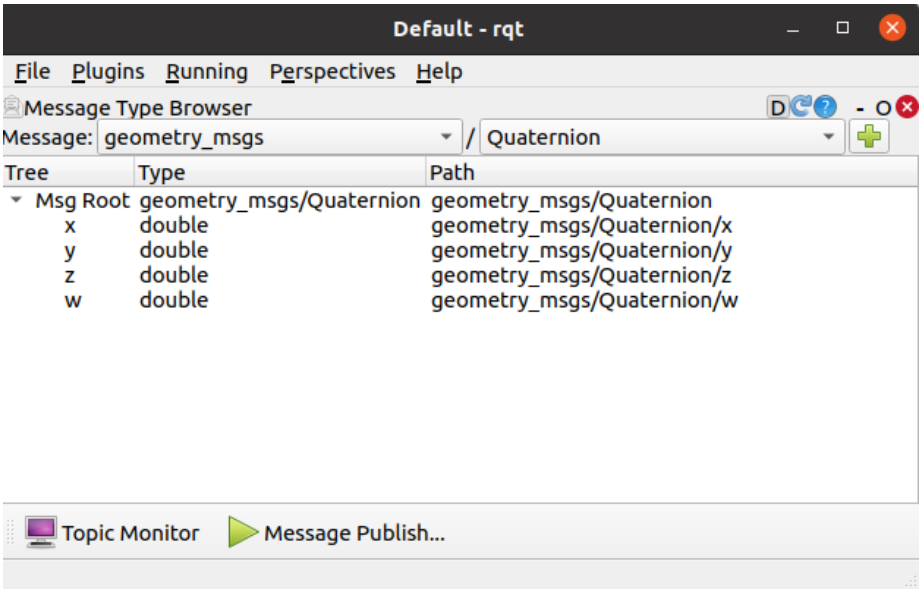

## 4. Message Type Browser

- 메뉴: Plugins → Topics → Message Type Browser

특정 토픽의 타입을 확인하는 RQt 플러그인이다.

ros2 interface 명령어의 GUI 버전이다.

아래와 같이 geometry_msgs의 Quaternion을 확인해보면 double형의 x,y,z,w가 들어있음을 확인 할 수 있다.



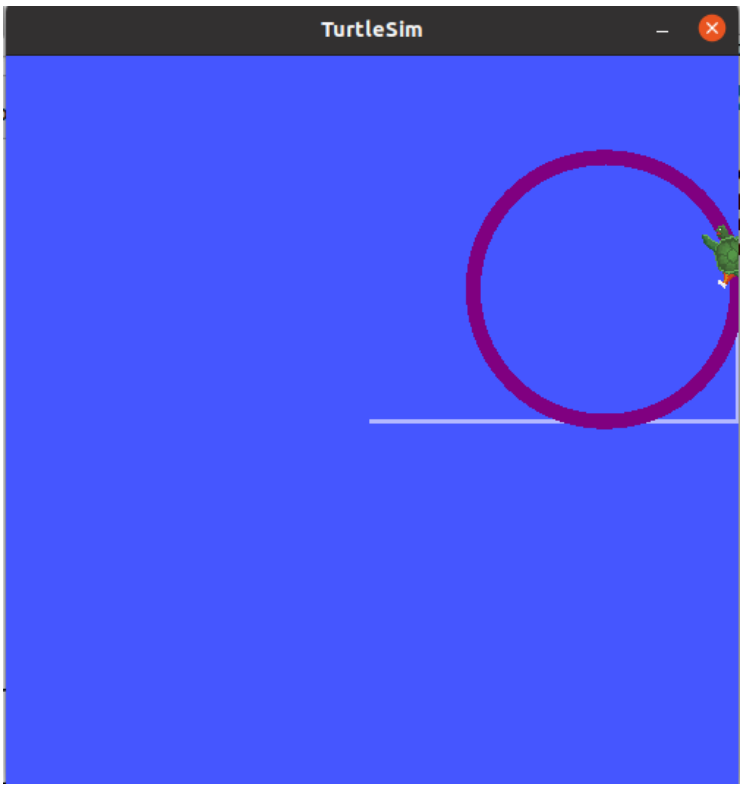

## 5. Service Caller

- 메뉴: Plugins → Services → Service Caller

현재 개발환경에서 실행 중인 서비스 서버에 서비스를 요청하는 RQt 플러그인이다.

ros2 service call 명령어의 GUI 버전이다.

turtle의 경로를 보라색으로 바꾸고선 크기를 10으로 설정했다



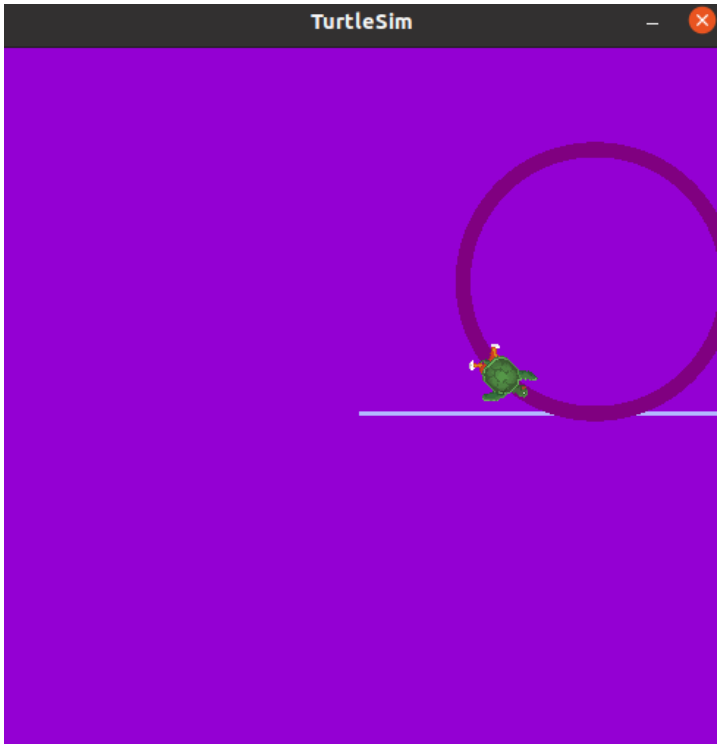

## 6. Parameter Reconfigure

- 메뉴: Plugins → Configuration → Dynamic Reconfigure

노드들에서 제공하는 파라미터 값을 확인하고 변경할 수 있는 RQt 플러그인이다.

ros2 param 명령어의 GUI 버전이다.

/turtlesim 노드의 background_r,g,b 값을 변경하여 배경색을 바꿔보았다.




## 7. Plot

- 메뉴: Plugins → Visualization → Plot

2차원 데이터 플롯 기능을 갖춘 RQt 플러그인으로 2차원 데이터를 도식화 해준다.

## 8. Image View

- 메뉴: Plugins → Visualization → Image View

카메라의 영상 데이터를 확인할 수 있는 RQt 플러그이다.

사용 방법은 왼쪽 상단에 토픽 이름(ex)/image)만 적어주면 되는데 이를 위해서는 이미지 데이터가 sensor_msgs/msg/Image 형태로 퍼블리시 되고 있어야 한다.

image_tools 패키지의 cam2image 노드를 실행시켜 테스트 해보았다.

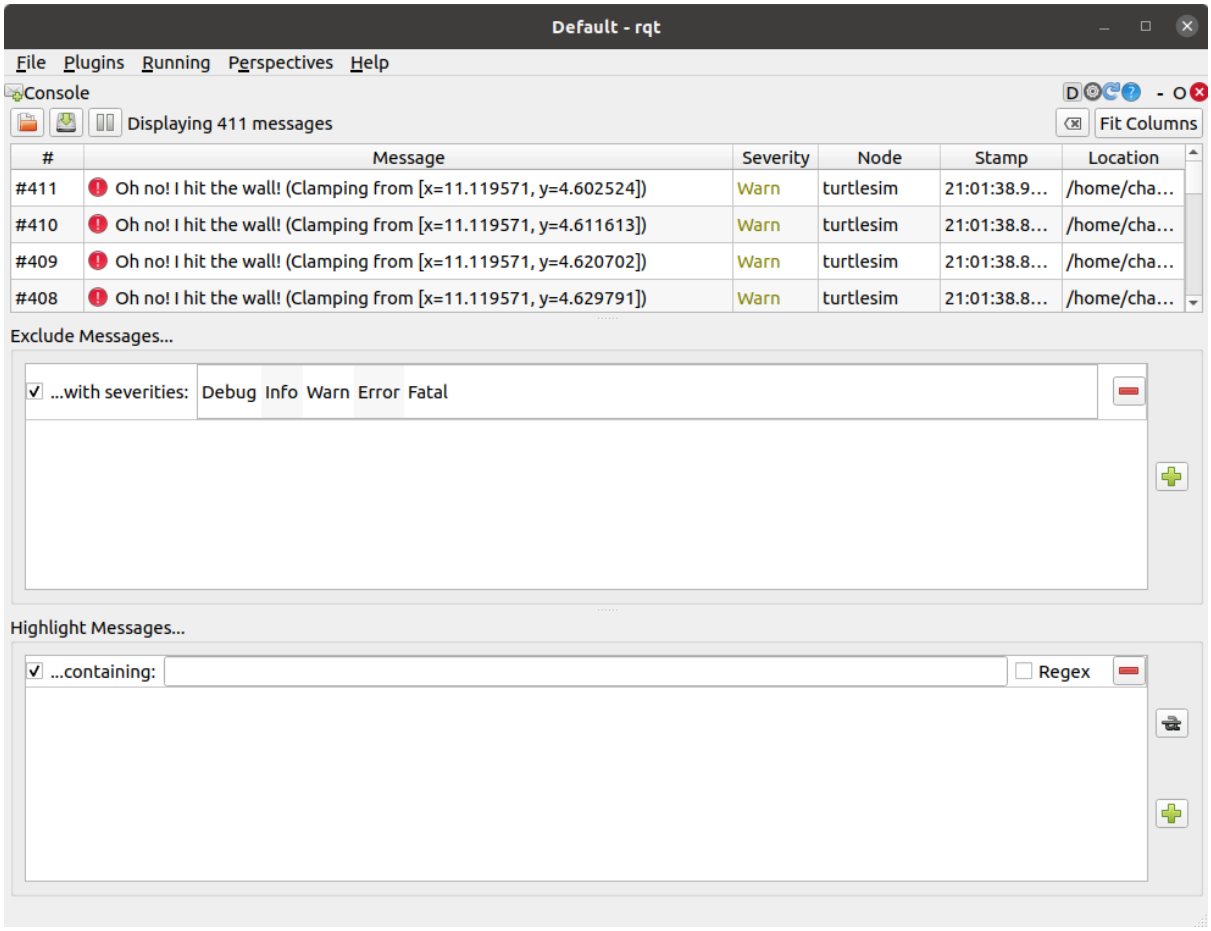
```
$ sudo apt install ros-foxy-image-tools
```

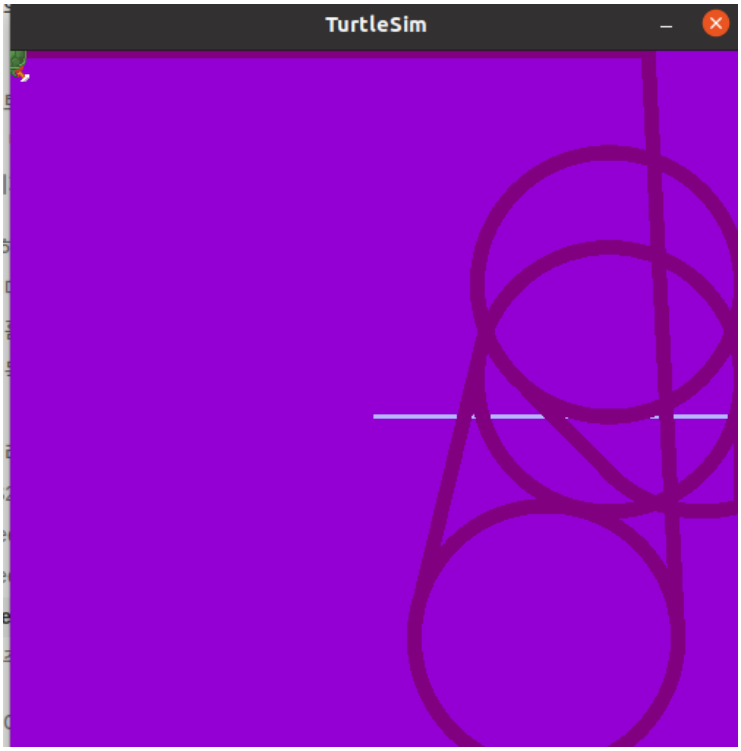
```
$ ros2 run image_tools cam2image
```

### 9. Console

- 메뉴: Plugins → Logging → Console

노드들에서 발생되는 정보(info), 경고(warning), 에러(error) 등의 rosout 데이터를 확인할 수 있는 RQt 플러그인이다.
/turtlesim 노드에서 turtle을 제어하였을 때 벽에 부딪치는 상태에서 나오는 Warn을 확인해보았다.




## 19. ROS 2의 표준 단위

ROS 커뮤니티에서는 ROS 프로그래밍에 사용되는 표준단위로 세계적으로 가장 널리 사용되고 있는 국제단위계인 SI unit과 국제단위계의 7개의 기본 단위를 조합해 만들어진 SI 유도 단위(SI derived unit)를 표준 단위로 정하였다.

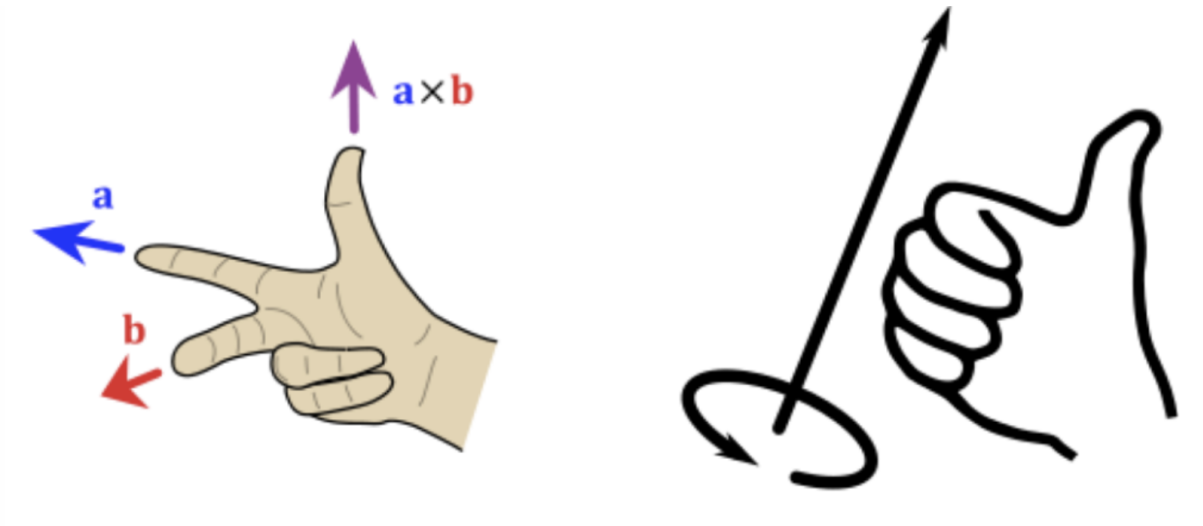
| 물리량 | 단위(SI unit) | 물리량 | 단위(SI unit) |
|---|---|---|---|
| Length(길이) | Meter(m) | Angle(평면각) | Radian(rad) |
| Mass(질량) | Kilogram(kg) | Frequency(주파수) | Hertz(Hz) |
| Time(시간) | Second(s) | Force(힘) | Newton(N) |
| Current(전류) | Ampere(A) | Voltage(전압) | Volt(V) |
| | | Temperature(온도) | Celsius(C) |
| | | Magnetism(자기장) | Tesla(T) |

또한 ROS 커뮤니티에서는 노드간에 주고받는 데이터의 이름과 자료형을 ROS 2 인터페이스에서 규정하고 있으며 그 단위는 REP-103에서 표준 단위로 정해놓았다.

## 20. ROS 2의 좌표 표현

ROS 2에서는 서로 다른 좌표 표현을 사용할 경우 발생하는 좌표계 불일치를 사전에 막기위해 좌표표현에 대한 규칙을 만들어 놓았다.

ROS 2 커뮤니티에서는 모든 좌표계를 삼차원 벡터 표기 관습중 오른손 법칙에 따라 표현한다. 기본적으로 회전축은 아래의 그림과 같이 검지, 중지, 엄지를 축으로 사용하며 오른손의 손가락을 감는 반시계 방향이 정회전 + 방향이다.



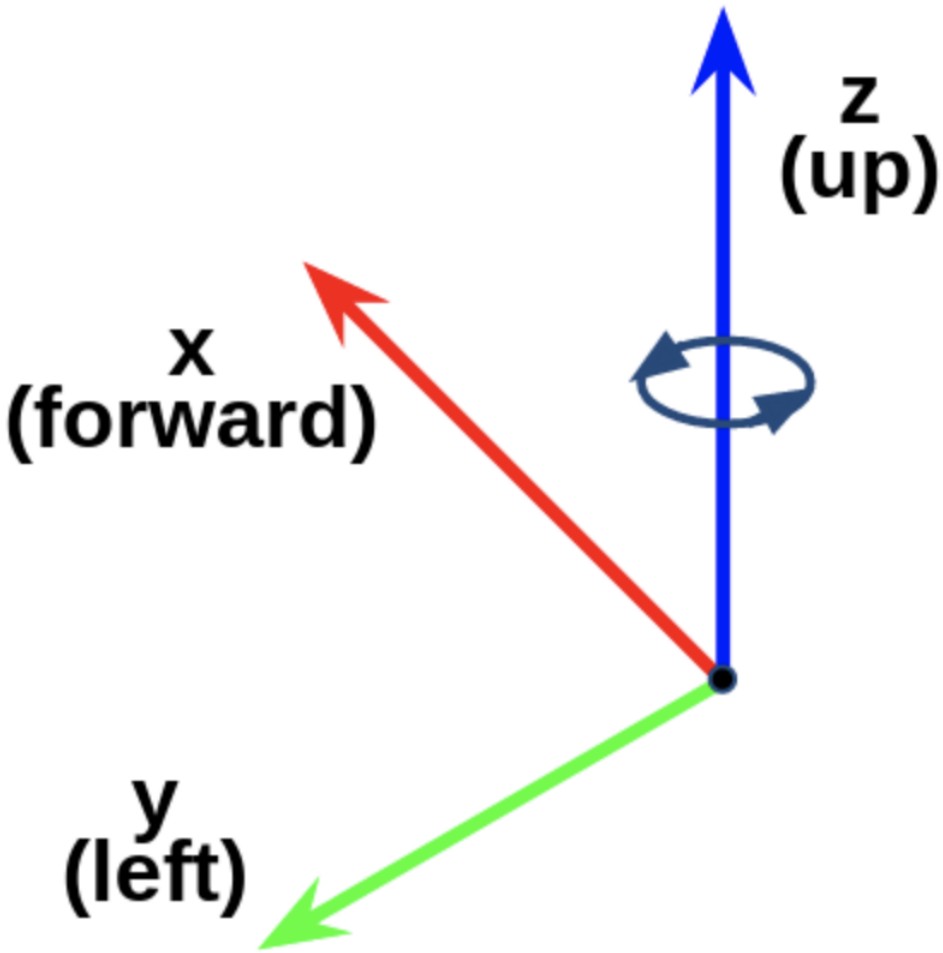

### 좌표 표현의 축 방향(Axis Orientation) 규칙

1. 기본 3축

   축 방향(Axis Orientation)으로 x forward, y left, z up을 사용한다.

   Rviz, Gazebo에서 기본 3축의 표현을 RGB 원색으로 표현하는데 순서대로

   x Red, y Green, z Blue 이다.




2. ENU 좌표

   지리적 위치(Geographic location)의 단거리 데카르트 표현의 경우 ENU(East North Up) 규칙을 사용한다.

3. 접미사 프레임 사용(Suffix Frames)

   앞에서 언급한 기본 3축 및 ENU 좌표에서 벗어나는 경우 접미사 프레임을 사용하여 기본 좌표계와 구별하여 사용한다.

   a. _optical 접미사

      컴퓨터 비전 분야의 경우, 카메라 좌표계로 많이 사용되는 z forward, x right, y down 을 사용하는데 이럴 경우 카메라 센서의 메시지에 _optical 접미사를 붙여 구분한다. 이때에는 카메라 좌표계와 로봇 좌표계 간의 TF(Transform)가 필요하다.

   b. _ned 접미사

      실외에서 동작하는 시스템의 경우, 사용하는 센서 및 지도에 띠라 ENU 가 아닌 NED(North East Down) 좌표계를 사용하야 할 때가 있다. 이때에는 _ned 접미사를 붙여 구분한다.

## 좌표 표현의 회전 표현(Rotation Representation) 규칙

회전의 경우 좌표계가 같아도 목적 및 계산 방법에 따라 표현하는 방법이 다양한데 대표적으로 사용하는 회전 표현 방식은 다음과 같다.

1. 쿼터니언(Quraternion)

- 간결한 표현 방식으로 가장 널리 사용된(x, y, z, w)
- 특이점 없음(No singularities)

2. 회전 매트릭스(Rotation Matrix)
   - 특이점 없음(No singularities)
3. 고정축 roll, pitch, yaw(Fixed axis XYZ)
   - 각 속도에 사용
4. 오일러 각도 yaw, pitch, roll(Euler angle ZYX)
   - 전역 좌표계에서 회전이 발생하기 때문에 짐벌락현상 발생
   - 사용을 권장하지 않는다.