<RO:BIT ROS 보고서>

18기 지능팀 인턴 선현빈

Chapter 1. 로봇 소프트웨어 플랫폼

1. 플랫폼의 구성 요소

Personal Computer와 Personal Phone의 공통점으로 OS가 존재하고, 응용 프로그램이 있는 등 여러 특징이 겹친다고 할 수 있지만, 가장 중요한 교집합은 '누구나 하나 정도는 보유하고 있는 대중화된 기기'라는 것이다. 그렇다면 PC와 스마트폰이라는 기기가 대중화될 수 있었던 이유이자 특징이 무엇인가?

우선, 두 기기는 모듈화된 하드웨어로 구성되어 있기 때문에 다양한 하드웨어의 결합이 가능하다는 특징이 있다.

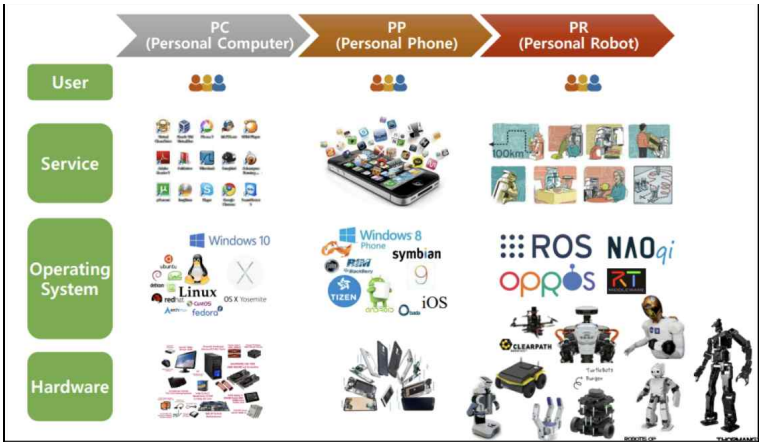
[수많은 모듈로 구성되어 있는 PC와 스마트폰]

두 번째로, 대중화된 기기에는 운영 체제와 애플리케이션(응용 프로그램)이 존재한다. 예를 들어, 현재 PC 운영 체제로써 Windows, Linux 등이 가장 널리 사용되고 있으며, 스마트폰 운영 체제로 ios, Android 등이 대부분의 스마트폰에 탑재되어 있다.

[오늘날 PC와 스마트폰에 탑재되는 다양한 운영체제들]

마지막으로 가장 중요한 특징은 그 기술의 시장이나 산업, 또는 분야에 대한 생태계가 존재한다. 특히 PC는 대중화되는 과정에서 하드웨어 모듈과 운영체제, 응용 프로그램(서비스), 그리고 유저로 구성되는 생태계를 구축하였고, 이러한 산업적 생태계가 조성되었기에 PC가 대중화될 수 있었다. 스마트폰 역시 PC와 마찬가지로, PC가 대중화되기 위해 지나쳤던 과정을 그대로 답습하여 대중화되었으며, 둘 모두 보이지 않는 생태계 속의 분업이 이루어지고 있다.

위와 같은 이유로 로봇 분야도 Personal Computer와 Personal Phone처럼 Personal Robot의 시장을 활성화하고 대중화하기 위해 같은 단계를 밟아갈 것으로 보인다. 그러나 현재 로봇 분야는 개인, 팀 혹은 회사 단위의 집단 모두 하나의 집단에서 영화 속 아이언맨의 모습처럼 납땜, OS, 각종 하드웨어, 소프트웨어, 통신, 바이오 등을 모두 다루어 로봇을 완성해야 하는 환경에서 완전히 벗어나지는 못한 상황에 놓여있으며, 이를 개선하기 위해 로봇 소프트웨어 플랫폼의 개발 및 발전이 활발히 이루어지고 있다.



[Personal 기기의 생태계 구조]

2. 로봇 소프트웨어 플랫폼

로봇 소프트웨어 플랫폼은 로봇 응용프로그램을 개발할 때에 필요한 하드웨어 추상화, 하위 디바이스 제어, 센싱, 인식, SLAM, 네비게이션, 매니퓰레이션 등의 기능을 구현하며, 패키지 관리 및 라이브러리, 개발/디버깅 도구를 제공하는 플랫폼이다.

1983년 Martin Cooper가 최초의 상용 핸드폰이라고 할 수 있는 모토로라 DynaTAC 8000을 시장에 선보였다. 이후에 소프트웨어 플랫폼에 대한 연구 개발이 이루어지면서, 아래와 같은 변화를 불러왔다.

- 하드웨어 인터페이스 통합

- 하드웨어 추상화, 규격화, 모듈화

- 가격 저하 + 성능 향상

- 하드웨어와 운영체제, 애플리케이션을 각각 분리

- 사용자 수요에 맞는 서비스에 집중(사용자의 니즈에 집중)

- 유저가 증가하여, 구매 및 피드백이 함께 증가 -> 생태계의 선순환 구조 형성

PC와 스마트폰의 운영체제는 현재 한 손에 꼽을 정도로 적은 숫자가 주류를 차지하고 있다. 반면에, 로봇 운영체제는 ROS를 필두로, 40여 개의 운영 체제

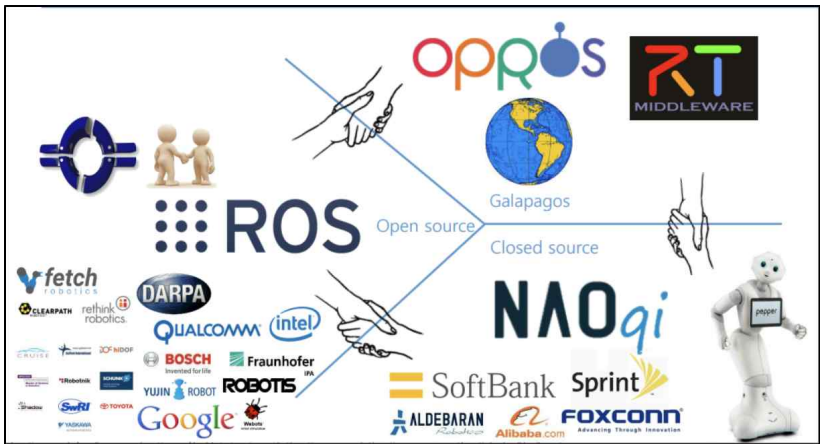
가 시장을 분할하고 있는 춘추전국 시대로 보여진다. 이러한 로봇 운영체제는 크게 세 가지로 분류할 수 있다.

- Open source : 안드로이드와 같은 오픈 소스 기반의 운영 체제로, ROS가 대표적이다.

- Closed source : IOS와 같은 폐쇄형 운영 체제로, NAOqi가 대표적이다.

- Galpagos : 보급이 많이 어려운 운영 체제로, OPROS, RTmiddleware 등이 있다.

초반에는 각기 다른 특징점이 존재했지만, 현재에는 각 운영 체제와 그 안에 존재하는 응용프로그램이 모두 비슷해졌다.



[주요 로봇 운영 체제]

3. 로봇 소프트웨어 플랫폼의 필요성

위에서 언급된 로봇 소프트웨어 플랫폼 중에서 어느 것이 가장 좋은지 우열을 가리고, 새로운 운영 체제를 더 개발하기 보다는, 로봇 소프트웨어 플랫폼이 개발되는 목적을 생각해볼 필요가 있다. 따라서 로보틱스 생태계 형성을 위해, 새로운 운동장을 만들려 하지 말고, ROS와 같은 로봇 소프트웨어 플랫폼을 사용하여 뛰어난 선수가 되는 것을 목표로 하는 것이 좋다.

4. 로봇 소프트웨어 플랫폼의 필요성이 가져올 미래

- 하드웨어 플랫폼과 소프트웨어 플랫폼 간의 인터페이스가 확립된다.

- 모듈형 하드웨어 플랫폼이 확산된다.

- 요구하는 선행 지식의 양이 대폭 감소하여 하드웨어에 대한 지식이 없어도 응용 프로그램을 작성할 수 있게 된다.

- 더 많은 소프트웨어 인력들이 로보틱스 분야로 진입할 수 있게 되면서, 로봇 제품에 대한 새로운 아이디어 및 다양한 시도가 이뤄지게 된다.

- 유저에게 제공할 서비스, 즉 유저의 니즈에 집중할 수 있게 된다.

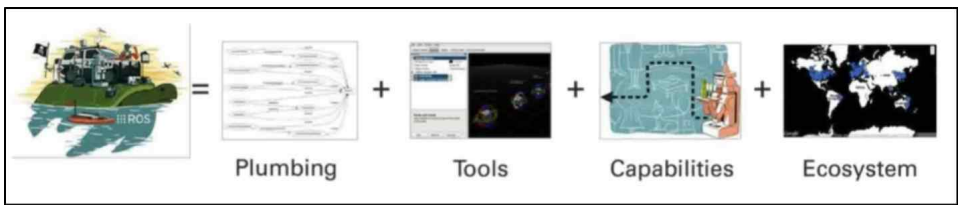
- 실수요가 있는 서비스를 제공하게 되어 유저 계층이 형성되고 피드백이 다양하고 많아지게 된다.

- 로봇 개발이 급속도로 발전할 수 있게 되는 계기가 된다.

Chapter 2. 로봇 운영체제 ROS

1. ROS란?

ROS는 현재 가장 많은 유저가 사용하고 있는 오픈 소스 기반이자, 로봇을 위한 메타 운영 체제이다. 엄밀히 따지면 OS는 아니며, 그렇기에 메타 운영체제라고 칭한다. 메타 운영 체제를 설명하기 전에, ROS는 큰 범주에서 로봇 소프트웨어 개발을 위한 소프트웨어 프레임워크이자, 미들웨어라고 할 수 있다.



[ROS의 소프트웨어 프레임워크적 구조]

2. 메타 운영 체제

메타 운영 체제는 정확히 정의되어 있는 용어는 아니지만, 응용 프로그램과 분산 컴퓨팅 자원 간의 가상화 레이어로 분산 컴퓨팅 자원을 활용하여, 스케

쥴링 및 로드 ,감시, 에러 처리 등을 실행하는 시스템을 말한다.

쉽게 말하자면, 윈도우나 리눅스, 안드로이드와 같은 전통적인 운영 체제는 아니며, 이런 기존의 운영 체제들을 로봇 응용 소프트웨어 개발을 위해 이용하는 '로봇 소프트웨어 프레임워크'라고 말할 수 있다. 따라서 로봇 개발을 위한 다양한 기능을 제공하는 tool box적 특징을 갖는다.

[ROS는 기존 운영체제를 이용한다]

3. ROS를 사용해야 하는 이유

a. 프로그램의 재사용성 : 유저는 자신이 개발하려는 부분에만 집중하고, 나머지 기능에 대해서는 관련 패키지를 다운로드하여 사용할 수 있다. 이와 같은 맥락으로, 자신이 개발한 프로그램을 다른 사람들이 사용할 수 있도록 공유하는 것도 가능하다. 예를 들어, NASA는 국제 우주 정거장에서 사용하는 Robonaut2의 제어를 위해 자체 프로그램 이외에도 다양한 드라이버를 제공하며, 멀티 플랫폼에서 사용 가능한 ROS와 실시간 제어 및 메시지 통신 복구, 신뢰성을 갖춘 OROCOS를 혼용해 임무를 수행했다.

b. 통신 기반 프로그램 : 노드 패키지화를 거쳐 최소 실행 단위로 나누어진 프로그램은 나누어진 노드끼리 데이터를 주고 받아야 하는데, 로봇 소프트웨어 플랫폼들은 이러한 데이터 통신에 대해 전반적 사항을 모두 갖추고 있다. 프로그램이 최소 실행 단위로 나누어 졌기에 디버깅에 유리하며, 각 노드가 하드웨어 의존성에서 벗어나 네트워크에서 통신을 제공하기 때문에 네트워크 프로그래밍이 가능하므로 원격 제어 및 IoT에 매우 유용하다.

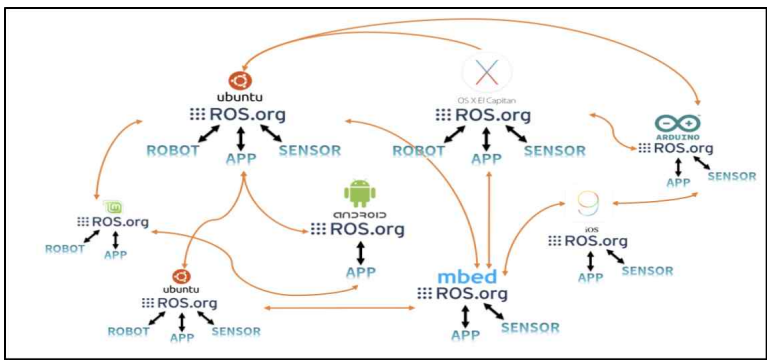
c. 개발 도구 지원 : ROS는 디버깅 관련 도구 및 2차원 플롯과 3차원 시각화에 대한 도구를 제공하므로 로봇 개발을 위한 도구를 직접 개발할 필요가 없다. 뿐만 아니라, 3차원 시뮬레이터도 제공하며, 3차원 거리 정보를 포인트 클라우드 형태로 변환하여 보여주는 등 센싱 및 시뮬레이션에서 다양한 상황에 대한 결과를 직관적으로 재현할 수 있다.

d. 생태계 조성 : 앞서 언급한 바와 동일하게, ROS는 로봇 소프트웨어 플랫폼으로써 범람하는 하드웨어 기술들을 통합할 운영 체제의 역할을 하며, 로보틱스 생태계의 틀을 갖춰나가고 있다.

e. 활성화되어 있는 커뮤니티 : 가장 중요하게 생각될 수 있는 부분으로, 여지껏 고립되어 있다고 볼 수 있었던 로봇 업계가 위에 언급한 다양한 기능으로 말미암아, 서로의 협업을 중시하는 방향으로 나아가고 있다. 일례로, ROS는 2015년 기준 자발적으로 5000개 이상의 패키지들이 개발 및 공유되고 있으며, 이에 대한 사용 방법을 설명한 위키 페이지가 16000페이지를 넘어서고 있다. 커뮤니티에서 가장 중요한 질의응답의 수는 24000건으로 로봇 업계의 소통 활성화가 로봇 소프트웨어 플랫폼을 통해 매우 긍정적인 방향으로 발전해가고 있다.

4. ROS의 목적

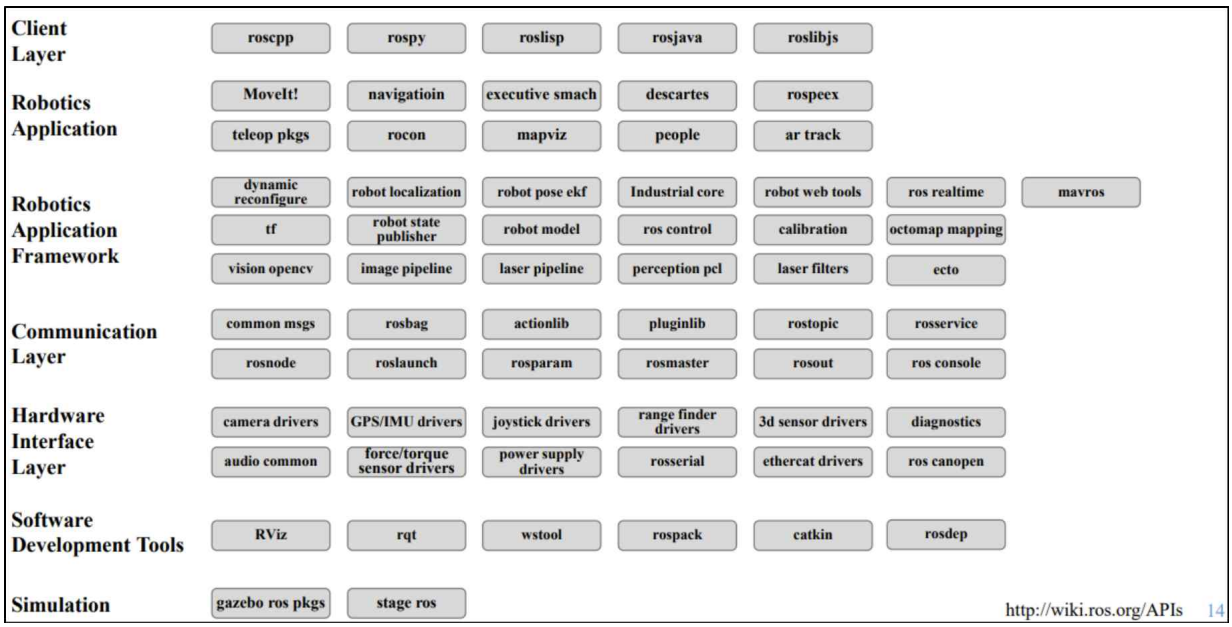
- ROS의 기능적 목적은 '이기종 디바이스 간의 통신 지원'의 지분이 가장 크다고 여겨진다. 이기종 디바이스 간의 통신 지원이란 아두이노, 안드로이드, 우분투, ios, mbed, ROS 등 서로 다른 운영 체제 및 응용 프로그램을 사용하고 있는 로봇끼리의 통신을 지원하는 것이다. 심지어, OS 탑재가 불가능한 MCU의 경우 시리얼 통신, 블루투스, LAN통신에 대한 라이브러리도 제공한다.



[이기종 디바이스 간의 통신 지원 조직도(예시)]

- ROS는 개발자가 어떠한 기능적 특징을 강조하기 보다는, 생태계 구축을 위하여 만들었다는 것을 강조할 정도로, 로보틱스 소프트웨어 개발을 전세계 단계에서 공동 작업이 가능하도록 **생태계를 구축하는 것이 진정한 목적**이라고 할 수 있다.

5. ROS의 구성 요소



[ROS의 구성 요소]

- **Client Layer** : 간단히 말해, 다양한 언어 지원에 대한 요소이다.

- **Robotics Application** : 네비게이션, 매니퓰레이션 등의 기능이 들어가 있다.

- **Framework** : maping, realtime, laser filter 등 각종 유용한 프로그램 지원에 대한 툴이 들어가 있다.

- **Communication Layer** : 토픽, 서비스 등 메시지 통신과 관련한 요소가 들어가 있다.

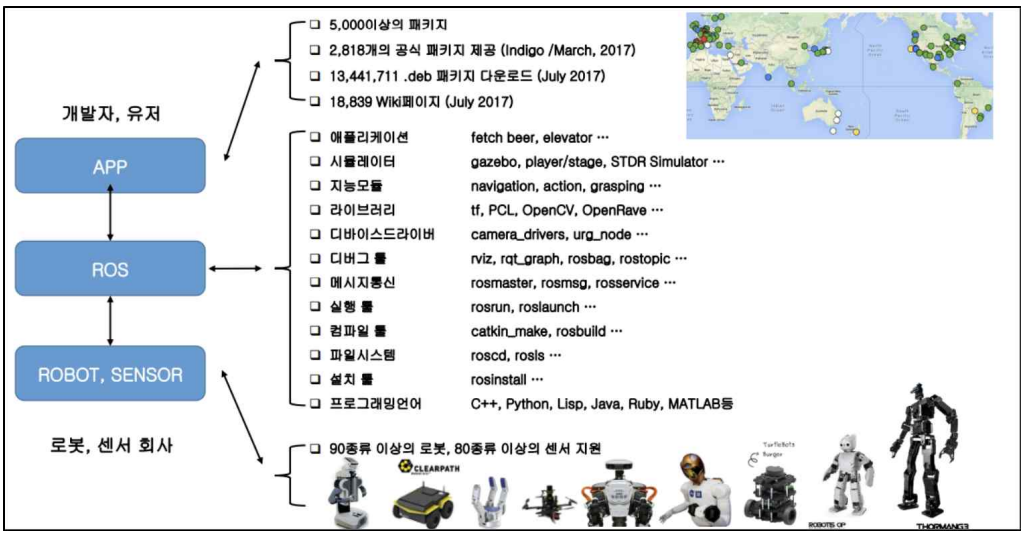
- **HI Layer** : 카메라, GPS, 모터 등에 대한 하드웨어 관련 인터페이스를 갖추고 있다.

- **Software Development Tools** : 시각화, GUI, 컴파일 등 소프트웨어 Develop에 대한 툴을 구성 요소로 갖추고 있다.

- **Simulation** : 2차원의 stage ros와 3차원의 gazebo를 지원하고 있다.

6. ROS의 생태계

2015-2017 기준 아래와 같은 생태계를 갖추었다.



[ROS의 생태계]

7. ROS의 특징

- **통신 기능** : 노드 간에 데이터 통신을 제공하여 메시지 교환을 통해 복잡한 프로그램을 잘게 나누어 공동 개발이 가능하며, 통상적으로 미들웨어로 지칭되는 메시지 전달 인터페이스를 지원한다.

  a. **메시지 파싱** : 로봇 개발 시 사용되는 통신 시스템 제공 - 센서 데이터를 어떻게 보낼지에 대한 규격화된 통신을 제공함.

  b. **메시지 기록 및 재생** : 송수신되는 메시지를 저장하고, 저장된 메시지를 나중에 사용할 수 있다. 저장된 메시지를 반복적 실험 및 알고리즘 개발에 용이하게 사용할 수 있다.

  c. **메시지 사용으로 다양한 프로그래밍 언어 사용 가능** : 노드 간의 데이터

교환은 메시지 형식으로 이뤄지기 때문에, 각각의 노드가 서로 다른 언어로 작성하는 것이 가능하다.(클라이언트 라이브러리 참고)

d. 분산 매개 변수 시스템 : 시스템 사용 변수를 글로벌 키 값으로 작성해 공유 및 수정하여, 외부에서 실시간으로 반영할 수 있다.

- 로봇 관련 다양한 기능 : 로보틱스에서 많이 사용되는 모델링, 센싱, 인식, 네비게이션, 매니퓰레이션 등의 기능을 지원한다.

a. 로봇에 대한 표준 메시지 정의 : 커뮤니티 내에서 각각의 센서에 대한 표준 통신 방식, 메시지를 정의하고 있다.

b. 로봇 기하학 라이브러리 : 로봇 및 센서의 상대적인 좌표를 트리화하는 TF를 제공한다.

c. 로봇 기술 언어 : 로봇의 물리적 특성을 설명하는 XML 문서를 기술할 수 있다.

d. 로봇 진단 시스템 : 로봇 상태를 직관적으로, 한눈에 파악할 수 있는 진단 시스템을 제공한다.

e. 센싱 및 인식 : 센서 드라이버와 센싱 및 인식 레벨의 라이브러리를 제공한다.

f. 네비게이션 : 로봇의 위치 및 자세 추정, SLAM, 네비게이션 라이브러리를 제공한다.

g. 매니퓰레이션 : 로봇 암에 사용되는 IK, FK 등의 라이브러리를 제공하며, GUI 형태의 매니퓰레이션 Tool인 MoveIt!을 제공한다.

- 여러 개발 도구 : 명령어 도구, 시각화 도구 Rviz, GUI 도구 모음 rqt, 3차원 시뮬레이터 Gazebo를 지원한다.

a. 명령어 라인 도구 : GUI 없이 ROS에서 제공되는 명령어로만 로봇 엑세스 및 대부분의 ROS 기능을 소화할 수 있다.

b. Rviz : 3D 시각화 및 센서 데이터 시각화를 제공하고, 로봇의 외형과 계획

되어 있는 동작을 시각적으로 표현한다.

c. **rqt** : 그래픽 인터페이스 개발을 위해 Qt기반의 프레임 워크를 제공한다.

d. **gazebo** : 물리 엔진을 탑재하여 로봇과 센서, 환경 모델 등을 지원하는 3차원 시뮬레이터이다. ROS와의 호환성이 높다.

8. ROS 버전 선택

- **리눅스** : 5년간 기술 지원이 약속되어 있는 최신 LTS 버전의 우분투를 지원한다. 2년마다 매년 4월에 LTS 버전을 릴리즈한다. 릴리즈 3개월 이후에 사용하는 것을 권한다.

- **ROS** : 2년마다 매년 5월에 LTS 버전을 릴리즈한다. 릴리즈 3개월 이후에 사용하는 것을 추천한다.

- **Gazebo** : "gazebosim.org"에서 ROS 호환성 정보 검토 후 사용한다.

Chapter 3. ROS 개발환경 구축

1. ROS 설치

- 한 줄 설치 코드

```
wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_kinetic.sh
&& chmod 755 ./install_ros_kinetic.sh && bash ./install_ros_kinetic.sh
```

[ROS 한 줄 설치 코드]

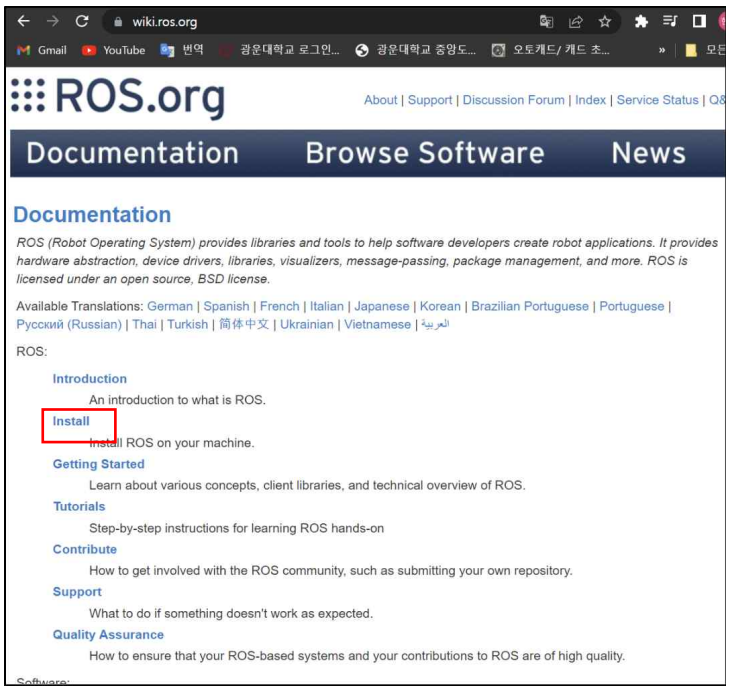
wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_kinetic.sh

&& chmod 755 ./install_ros_kinetic.sh && bash ./install_ros_kinetic.sh

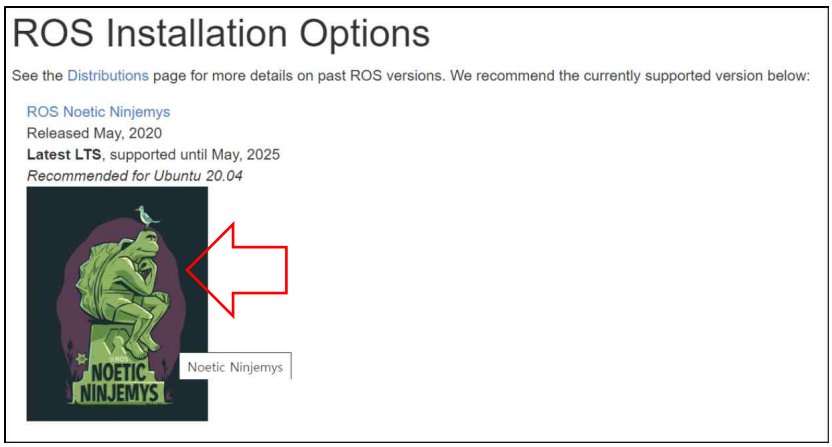
- 수동 설치 방법

a. ROS.org, 로스 위키에 접속한다.

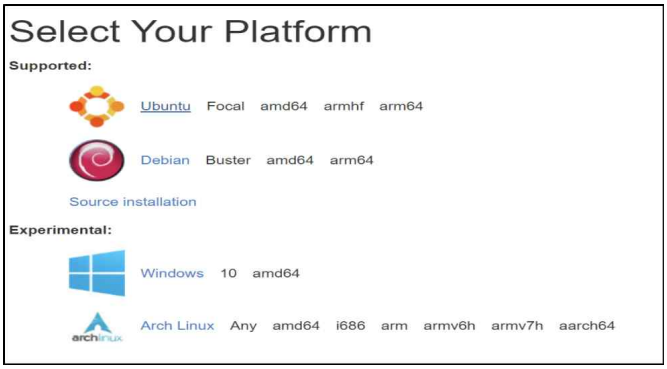
b. Install을 클릭한다.



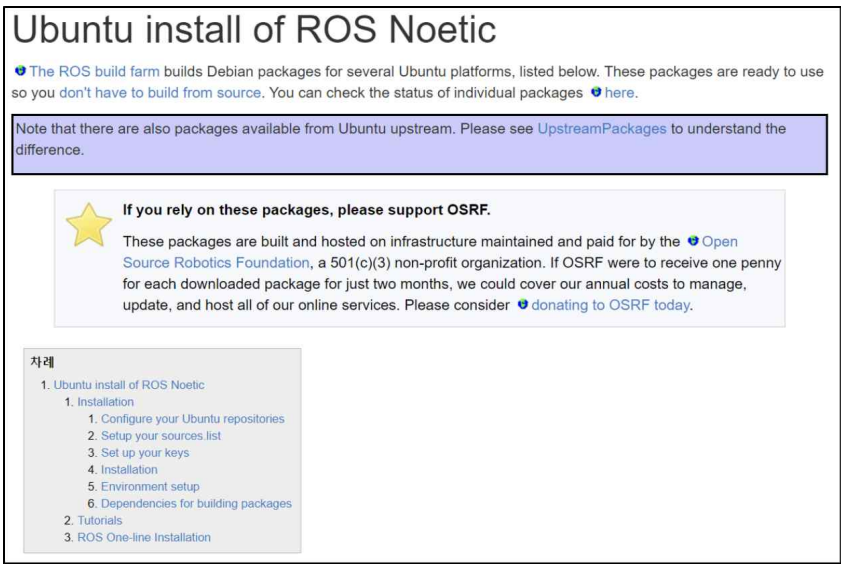
c. Noetic을 선택한다.



d. 사용하고 있는 OS를 선택한다.



e. 매뉴얼에 따라 설치한다.



2. ROS 환경 설정

환경 설정 wiki 사이트 :

https://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

```
$ gedit ~/.bashrc 또는 $ eb

alias eb ='nano ~/.bashrc'
alias sb ='source ~/.bashrc'
alias cw ='cd ~/catkin_ws'
alias cs ='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'

source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://localhost:11311
export ROS_HOSTNAME=localhost

#export ROS_MASTER_URI=http://192.168.1.100:11311
#export ROS_HOSTNAME=192.168.1.100
```

[ROS 네트워크, 단축키 등 주요 환경 설정]

3. ROS 동작 테스트

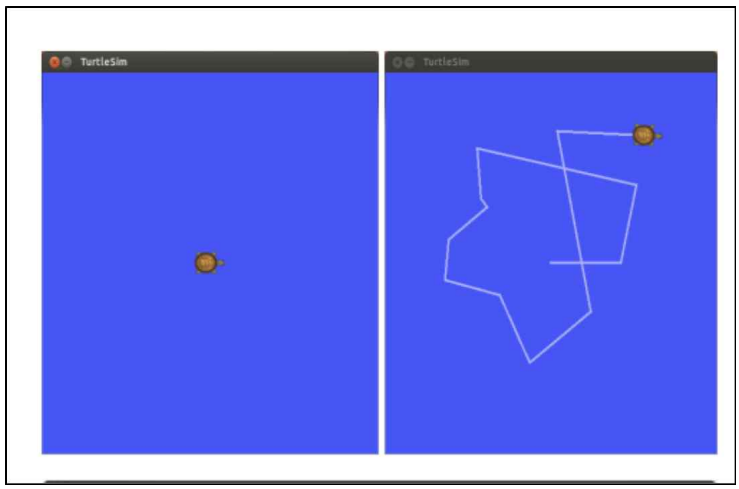
turtlesim 패키지로 ROS가 제대로 설치되었고, 동작하는지 테스트할 수 있다.

- roscore

- rosrun turtlesim turtlesim_node

- rosrun turtlesim turtle_teleop_key

위의 세 가지 명령어를 각각 별도의 터미널 창을 생성하여 입력하여 준 뒤, teleop_key 창에서 방향키를 조작한다.



[방향키 조작에 따라 화면의 거북이가 움직이고, 경로가 남는다]

- 추가적으로 cm을 입력하여 컴파일을 확인한다.

4. ROS에서 사용 가능한 IDE

- > https://wiki.ros.org/IDEs 에서 사용 가능한 IDE를 확인할 수 있다.

- 추천1 : **Qt creator + Qt Creator Plugin for ROS** - GUI 기반으로 rqt를 사용할 때, 유용하게 사용할 수 있다. 또한, CmakeLists.txt를 그대로 사용할 수 있다.

  설치 : sudo apt-get install qtcreator

- 추천2 : **VS Code + ROS Extension** - HW_001_001 참조, 간단한 텍스트 편집기이며 빠르다. 비슷한 계열로 Atom, Clion, Sublime Text 등도 좋다.

- 추천3 : **Eclipse** - 많은 사람들이 사용하는 익숙한 IDE이지만, 무겁다.

- 클라우드 환경 : **ROS Develoment Studio / AWS RoboMaker** - 클라우드에 접속하여 브라우저에서 동작하는 형태이므로 별도 설치가 필요하지 않다. 더불어, 윈도우, 리눅스, macOS와 무관하게 브라우저만 있으면 된다.

Chapter 4. ROS의 중요 컨셉

1. ROS 기본 용어 및 메시지 종류 정리

기본	Node	최소 단위의 실행 가능한 프로세스로, 하나의 실행 가능한 프로그램을 생각하면 된다. ROS에서는 Node 단위로 작업하게 된다. 노드와 패키지를 나누는 것은 개발자에게 달려 있다.
	Package	하나 이상의 노드, 노드 실행을 위한 정보 등을 묶어 놓은 것, 패키지의 묶음은 메타 패키지라 하여 따로 분리한다. ROS는 5000개 이상의 패키지가 있다.
	Message	메시지를 통해 노드간의 데이터를 주고 받게 된다. 메시지는 int, float, point, boolean과 같은 변수 형태이며, 메시지가 메시지를 품고 있는 데이터 구조, 메시지의 배열 구조 등도 사용할 수 있다.
토픽	Topic	단방향이자, 연속성을 갖는 메시지 통신 방법이다. 목적에 따라서 1대1, 1대N, N대1, N대N 모두 가능하다.
	Publisher	메시지를 보내는 대상
	Subscriber	메시지를 받는 대상
서비스	Service	양방향이자, 일회성인 메시지 통신 방식이다. 다시 하기 위해서는 재접속을 해야 한다.
	Service server	서비스 클라이언트의 요청에 대한 응답을 하는 대상
	Service	서비스 서버에게 서비스를 요청하여 서비스의 응답

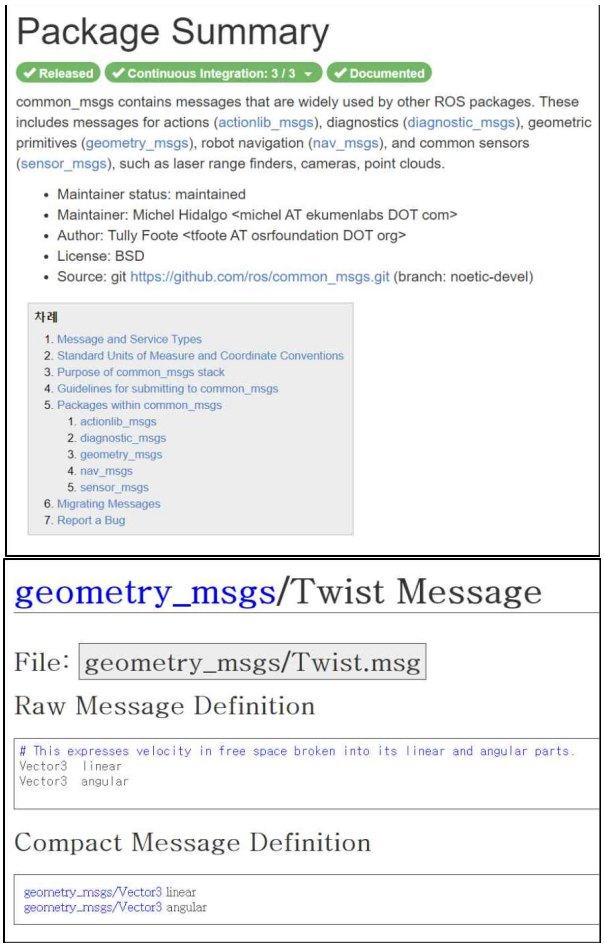
	client	을 받는 대상
액션	Action	액션은 서비스보다 복잡하거나 중간에 피드백이 필요할 때 사용한다.
	Action server	전달 받은 액션 목표에 대한 액션 피드백(중간 결과)를 클라이언트에게 전송하면서, 최종적으로 액션 결과를 전달한다.
	Action client	액션 서버에게 액션 목표를 전달한다.

- Node는 예를 들어, 안면 인식 시스템을 만든다고 할 때, 카메라가 로우 데이터를 받는 프로세스, 받은 데이터가 거치게 되는 여러 개의 필터, 결과 값을 측정하는 프로세스, 결과를 보안 시스템에 보내는 프로세스, 받은 결과를 토대로 문을 열거나 경고음을 알리는 프로세스 등 여럿으로 나눌 수 있을 것이다. 이런 각각의 프로세스들, 각각의 필터가 하나 하나의 노드라고 할 수 있다.

- Node는 각각의 프로세스를 세분화함으로써, 오픈 소스를 사용하는 입장에서 필요한 부분만 가져다 사용하는 것이 편리하다는 장점이 있으나, 노드1에서 노드2로, 노드2에서 노드3으로, 차례차례 통신한다는 단점이 있다.

- ROS에서 메시지 통신을 할 때, 9할 이상은 토픽을 사용하며, 나머지 10%는 대다수의 경우 서비스를 사용한다고 할 수 있다. 로봇이 많이 복잡해진 경우에만 액션을 사용한다.

- Message는 ROS에서 제공하고 있는 메시지 이외에도, 직접 만들어서 사용하는 것이 가능하다.



Package Summary

Released | Continuous Integration: 3 / 3 | Documented

common_msgs contains messages that are widely used by other ROS packages. These includes messages for actions (actionlib_msgs), diagnostics (diagnostic_msgs), geometric primitives (geometry_msgs), robot navigation (nav_msgs), and common sensors (sensor_msgs), such as laser range finders, cameras, point clouds.

- Maintainer status: maintained
- Maintainer: Michel Hidalgo <michel AT ekumenlabs DOT com>
- Author: Tully Foote <tfoote AT osrfoundation DOT org>
- License: BSD
- Source: git https://github.com/ros/common_msgs.git (branch: noetic-devel)

차례

1. Message and Service Types
2. Standard Units of Measure and Coordinate Conventions
3. Purpose of common_msgs stack
4. Guidelines for submitting to common_msgs
5. Packages within common_msgs
    1. actionlib_msgs
    2. diagnostic_msgs
    3. geometry_msgs
    4. nav_msgs
    5. sensor_msgs
6. Migrating Messages
7. Report a Bug

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

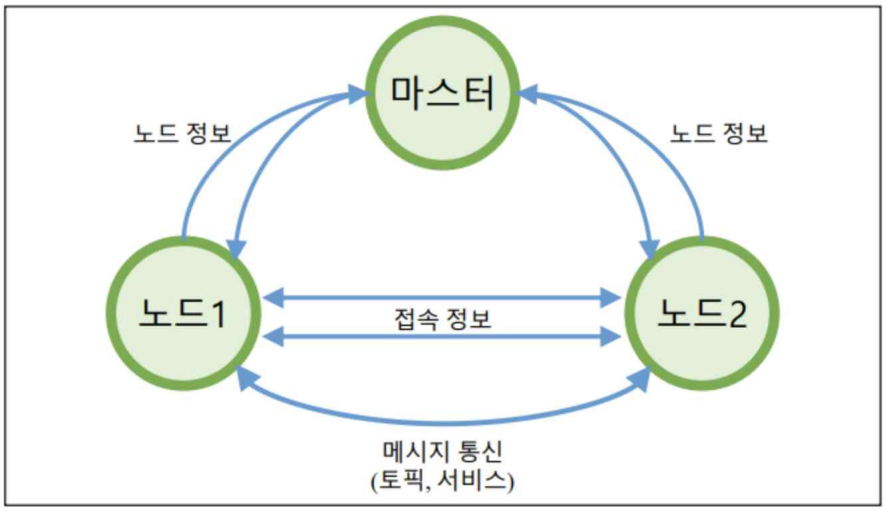
```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

[ROS 위키의 해당 페이지에서 제공 메시지 목록을 확인할 수 있다(위), turtlesim프로그램에서 사용한 Twist 메시지이다-메시지가 메시지를 감싸고 있는 구조(아래)]

2. 메시지 통신의 개념

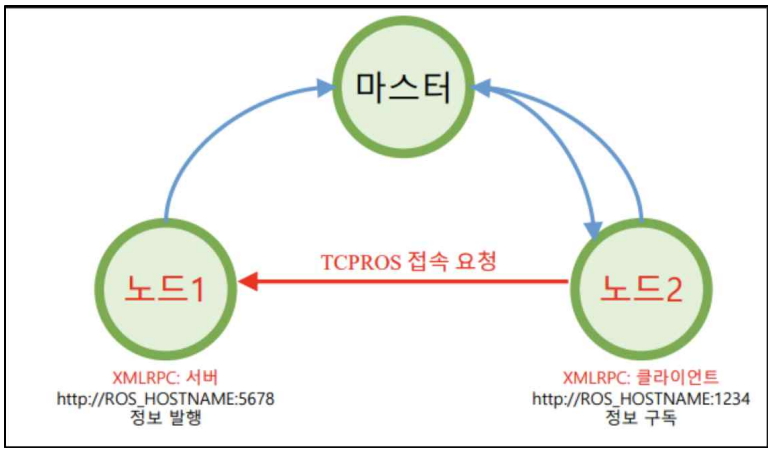


[메시지 통신의 과정]

- 마스터 : $roscore을 통해 마스터를 구동한다. XMLRPC(XML-Remote Procedure Call)이라는 아주 간단한 서버를 구현하여, 노드 정보를 관리하는 역할을 한다. 두 종류의 노드로부터 전달받은 정보가 서로 같은지 매칭해보고, 일치한다면 상대 노드에게 정보를 전달한다.

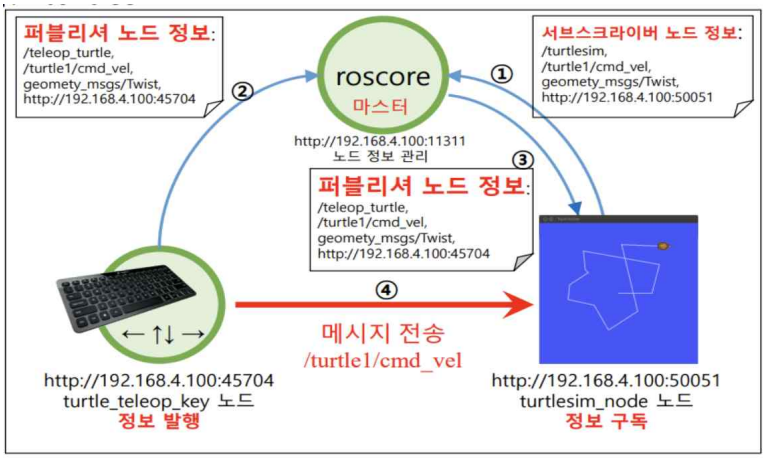
- 서브스크라이버 노드 구동 : $rosrun 패키지이름 노드이름 으로 실행한다. 자신의 정보를 마스터에게 전달하는데, 노드명, 토픽명, 메시지의 형태, IP 정보와 포트 번호를 마스터에 전송한다.

- 퍼블리셔 노드 구동 : $rosrun 패키지이름 노드이름 으로 실행한다. 서브스크라이버 노드와 마찬가지로 마스터에게 정보를 전달한다.



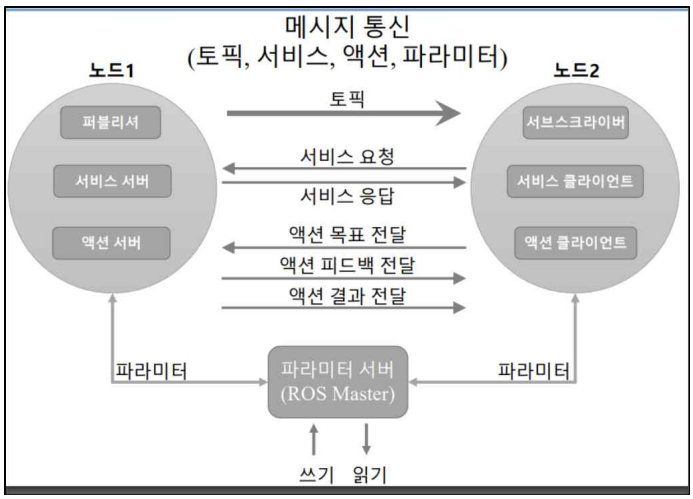
[서브스크라이버 노드가 상대의 정보를 전달받은 이후]

- 노드1과 노드2 사이의 TCPROS 통신으로 바뀌어 서브스크라이버 노드가 퍼블리셔 노드에게 접속 요청을 하고, 퍼블리셔 노드가 접속에 응답하면 자신의 TCP URL 주소와 포트 번호를 전송한다. TCPROS로 두 노드는 직접 연결된다. 이후 퍼블리셔 노드는 서브스크라이버 노드에게 메시지를 전송한다.(토픽)



[turtlesim 예제를 메시지 통신 개념에 적용한 사진]

3. 파라미터



[파라미터 및 메시지 통신 과정의 구조도(모든 방법을 표시)]

파라미터는 어떤 글로벌 변수를 네트워크에 지정하여, 외부 노드에서 변경시키고 임의의 프로세스를 바꿀 수 있다. ROS Master의 기능 중 일부라고 볼 수 있으며, getParam(매개변수 읽기), setParam(매개변수 초기설정) 등을 통해 사용한다. 파라미터는 int, float, boolean, string, list 등으로 설정할 수

있다.

4. 네임

네임은 노드, 메시지(토픽, 서비스, 액션, 파라미터)가 가지는 고유의 식별자이며, 노드나 메세지의 이름을 변경시키고 싶을 때 사용한다.

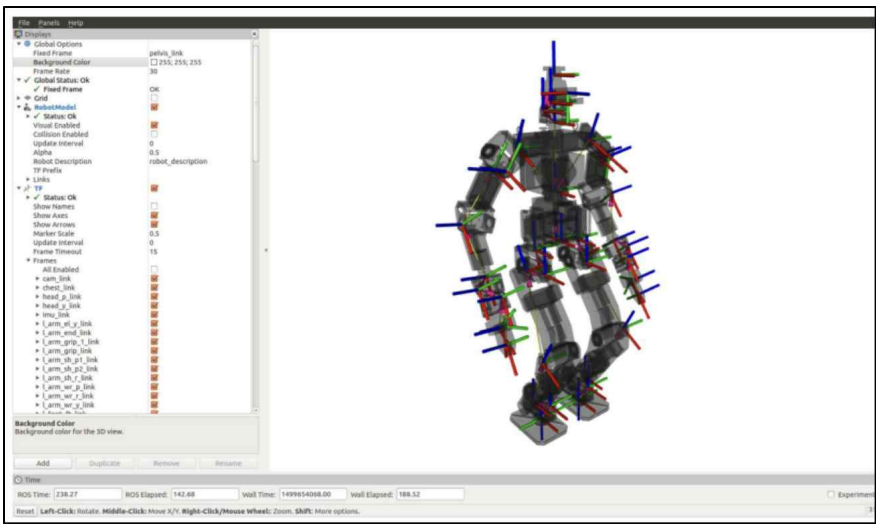
- 글로벌 : 문자 없이 네임을 바로 작성하거나 네임 앞에 슬래쉬(/)를 붙인다.
- 프라이빗 : 네임 앞에 틸트(~)를 붙인다.

Node	Relative (default)	Global	Private
/node1	bar → /bar	/bar → /bar	~bar → /node1/bar
/wg/node2	bar → /wg/bar	/bar → /bar	~bar → /wg/node2/bar
/wg/node3	foo/bar → /wg/foo/bar	/foo/bar → /foo/bar	~foo/bar → /wg/node3/foo/bar

[네임 작성 예시]

5. 좌표변환(TF)

좌표변환(tranform)은 로봇의 각 조인트, 즉 관절들의 상대 좌표를 변환하여 트리 형태로 여러 조인트들 사이의 관계도를 표시하는 기능 및 프로그램이다.



[좌표 변환 기능을 실행한 화면]

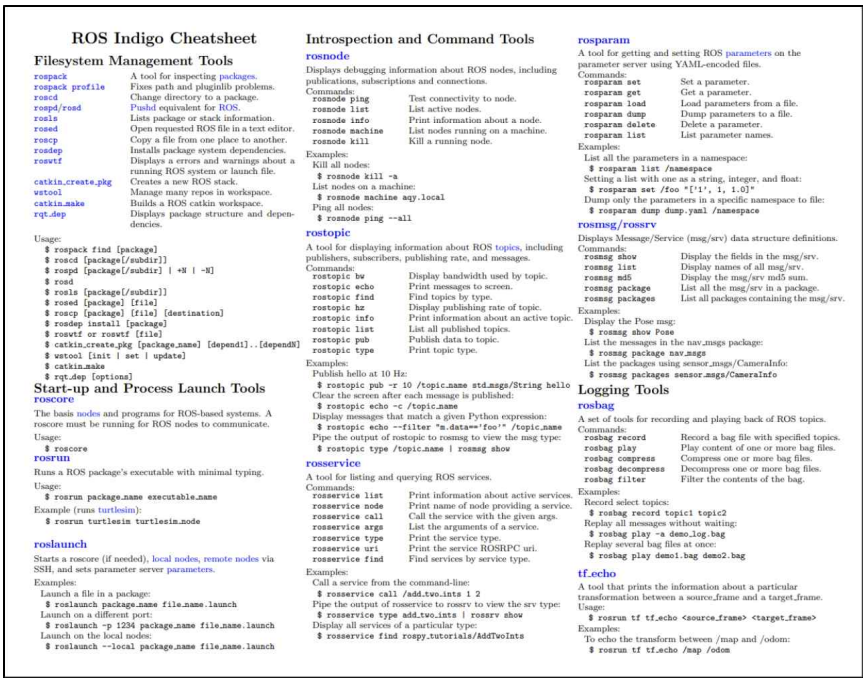
Chapter 5. ROS 명령어

1. 기본 명령어(커맨드 라인 툴)

rospack, roscd, rospd, rosls, rosed, roscp, rosdep,
roswf, catkin_create_pkg, wstool, catkin_make,
roscore, rosrun, roslaunch, rosnode, rostopic,
rosservice, rosparam, rosmsg, rossrv, rosbag, tf_echo

[커맨드 라인 툴]

▶https://github.com/oroca/oroca_ros_tutorials/raw/master/ROScheatsheet_indigo_catkin.pdf에서 위 커맨드 라인 툴의 명령어 설명과 그 외 많은 명령어에 대한 설명을 알 수 있다.



ROS Indigo Cheatsheet

Filesystem Management Tools

rospack — A tool for inspecting packages.
rospack profile — Fixes path and pluginlib problems.
roscd — Change directory to a package.
rospd/rosd — Pushd equivalent for ROS.
rosls — Lists package or stack information.
rosed — Open requested ROS file in a text editor.
roscp — Copy a file from one place to another.
rosdep — Installs package system dependencies.
roswtf — Displays a errors and warnings about a running ROS system or launch file.
catkin_create_pkg — Creates a new ROS stack.
wstool — Manage many repos in workspace.
catkin_make — Builds a ROS catkin workspace.
rqt_dep — Displays package structure and dependencies.

Usage:
$ rospack find [package]
$ roscd [package[/subdir]]
$ rospd [package[/subdir] | +N | -N]
$ rosd
$ rosls [package[/subdir]]
$ rosed [package] [file]
$ roscp [package] [file] [destination]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ catkin_create_pkg [package_name] [depend1]..[dependN]
$ wstool [init | set | update]
$ catkin_make
$ rqt_dep [options]

Start-up and Process Launch Tools

roscore
The basis nodes and programs for ROS-based systems. A roscore must be running for ROS nodes to communicate.
Usage:
$ roscore

rosrun
Runs a ROS package's executable with minimal typing.
Usage:
$ rosrun package_name executable_name
Example (runs turtlesim):
$ rosrun turtlesim turtlesim_node

roslaunch
Starts a roscore (if needed), local nodes, remote nodes via SSH, and sets parameter server parameters.
Examples:
Launch a file in a package:
$ roslaunch package_name file_name.launch
Launch on a different port:
$ roslaunch -p 1234 package_name file_name.launch
Launch on the local nodes:
$ roslaunch --local package_name file_name.launch

Introspection and Command Tools

rosnode
Displays debugging information about ROS nodes, including publications, subscriptions and connections.
Commands:
rosnode ping — Test connectivity to node.
rosnode list — List active nodes.
rosnode info — Print information about a node.
rosnode machine — List nodes running on a machine.
rosnode kill — Kill a running node.
Examples:
Kill all nodes:
$ rosnode kill -a
List nodes on a machine:
$ rosnode machine aqy.local
Ping all nodes:
$ rosnode ping --all

rostopic
A tool for displaying information about ROS topics, including publishers, subscribers, publishing rate, and messages.
Commands:
rostopic bw — Display bandwidth used by topic.
rostopic echo — Print messages to screen.
rostopic find — Find topics by type.
rostopic hz — Display publishing rate of topic.
rostopic info — Print information about an active topic.
rostopic list — List all published topics.
rostopic pub — Publish data to topic.
rostopic type — Print topic type.
Examples:
Publish hello at 10 Hz:
$ rostopic pub -r 10 /topic_name std_msgs/String hello
Clear the screen after each message is published:
$ rostopic echo -c /topic_name
Display messages that match a given Python expression:
$ rostopic echo --filter "m.data=='foo'" /topic_name
Pipe the output of rostopic to rosmsg to view the msg type:
$ rostopic type /topic_name | rosmsg show

rosservice
A tool for listing and querying ROS services.
Commands:
rosservice list — Print information about active services.
rosservice node — Print name of node providing a service.
rosservice call — Call the service with the given args.
rosservice args — List the arguments of a service.
rosservice type — Print the service type.
rosservice uri — Print the service ROSRPC uri.
rosservice find — Find services by service type.
Examples:
Call a service from the command-line:
$ rosservice call /add_two_ints 1 2
Pipe the output of rosservice to rossrv to view the srv type:
$ rosservice type add_two_ints | rossrv show
Display all services of a particular type:
$ rosservice find rospy_tutorials/AddTwoInts

rosparam
A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.
Commands:
rosparam set — Set a parameter.
rosparam get — Get a parameter.
rosparam load — Load parameters from a file.
rosparam dump — Dump parameters to a file.
rosparam delete — Delete a parameter.
rosparam list — List parameter names.
Examples:
List all the parameters in a namespace:
$ rosparam list /namespace
Setting a list with one as a string, integer, and float:
$ rosparam set /foo "['1', 1, 1.0]"
Dump only the parameters in a specific namespace to file:
$ rosparam dump dump.yaml /namespace

rosmsg/rossrv
Displays Message/Service (msg/srv) data structure definitions.
Commands:
rosmsg show — Display the fields in the msg/srv.
rosmsg list — Display names of all msg/srv.
rosmsg md5 — Display the msg/srv md5 sum.
rosmsg package — List all the msg/srv in a package.
rosmsg packages — List all packages containing the msg/srv.
Examples:
Display the Pose msg:
$ rosmsg show Pose
List the messages in the nav_msgs package:
$ rosmsg package nav_msgs
List the packages using sensor_msgs/CameraInfo:
$ rosmsg packages sensor_msgs/CameraInfo

Logging Tools

rosbag
A set of tools for recording and playing back of ROS topics.
Commands:
rosbag record — Record a bag file with specified topics.
rosbag play — Play content of one or more bag files.
rosbag compress — Compress one or more bag files.
rosbag decompress — Decompress one or more bag files.
rosbag filter — Filter the contents of the bag.
Examples:
Record select topics:
$ rosbag record topic1 topic2
Replay all messages without waiting:
$ rosbag play -a demo_log.bag
Replay several bag files at once:
$ rosbag play demo1.bag demo2.bag

tf_echo
A tool that prints the information about a particular transformation between a source_frame and a target_frame.
Usage:
$ rosrun tf tf_echo <source_frame> <target_frame>
Examples:
To echo the transform between /map and /odom:
$ rosrun tf tf_echo /map /odom

[위 링크의 ROS cheatsheet의 일부]

아래의 명령어들 이외에도, 새로운 명령어를 확인하고 싶거나 필요한 명령어가 있는 경우, ROS wiki 메인에서 Software: Common Tools를 선택하여 Command-line tools에 대한 정보를 확인할 수 있다.

▶ https://wiki.ros.org/ROS/CommandLineTools

2. 쉘 명령어

명령어	중요도	명령어 풀이	세부 설명
roscd	★★★	ros+cd(changes directory)	지정한 ROS 패키지의 디렉터리로 이동
rosls	★☆☆	ros+ls(lists files)	ROS 패키지의 파일 목록 확인
rosed	★☆☆	ros+ed(editor)	ROS 패키지의 파일 편집
roscp	★☆☆	ros+cp(copies files)	ROS 패키지의 파일 복사
rospd	☆☆☆	ros+pushd	ROS 디렉터리 인덱스에 디렉터리 추가
rosd	☆☆☆	ros+directory	ROS 디렉터리 인덱스 확인

[ROS 쉘 관련 주요 명령어]

- roscd는 cd명령어와 같이 지정한 ROS 패키지의 디렉터리로 이동할 때 사용한다.

3. 실행 명령어

명령어	중요도	명령어 풀이	세부 설명
roscore	★★★	ros+core	- master(ROS 네임 서비스) - rosout(로그 기록) - parameter server(파라미터 관리)
rosrun	★★★	ros+run	노드 실행
roslaunch	★★★	ros+launch	노드를 여러 개 실행 및 실행 옵션 설정
rosclean	★★☆	ros+clean	ROS 로그 파일을 검사하거나 삭제

[ROS 실행 관련 주요 명령어]

- roscore는 하나의 마스터를 실행시킬 때 사용하며, 위 설명과 같이 세 가지 기능을 수행한다. 멀티 마스터를 사용하는 경우도 있지만 드물기 때문에 매우 많이 사용하게 될 명령어이다.

4. 정보 명령어

명령어	중요도	명령어 풀이	세부 설명
rostopic	★★★	ros+topic	ROS 토픽 정보 확인
rosservice	★★★	ros+service	ROS 서비스 정보 확인
rosnode	★★★	ros+node	ROS 노드 정보 확인
rosparam	★★★	ros+param(parameter)	ROS 파라미터 정보 확인, 수정
rosbag	★★★	ros+bag	ROS 메시지 기록, 재생
rosmsg	★★☆	ros+msg	ROS 메시지 정보 확인
rossrv	★★☆	ros+srv	ROS 서비스 정보 확인
rosversion	★☆☆	ros+version	ROS 패키지 및 배포 릴리즈 버전 정보 확인
roswtf	☆☆☆	ros+wtf	ROS 시스템 검사

[ROS 정보 관련 주요 명령어]

5. catkin 명령어

명령어	중요도	세부 설명
catkin_create_pkg	★★★	패키지 자동 생성
catkin_make	★★★	캐킨 빌드 시스템에 기반을 둔 빌드
catkin_eclipse	★★☆	캐킨 빌드 시스템으로 생성한 패키지를 이클립스에서 사용할 수 있게 변경
catkin_prepare_release	★★☆	릴리즈할 때 사용되는 로그 정리 및 버전 태깅
catkin_generate_changelog	★★☆	릴리즈할 때 CHANGELOG.rst 파일 생성 또는 업데이트
catkin_init_workspace	★★☆	캐킨 빌드 시스템의 작업 폴더 초기화
catkin_find	★☆☆	캐킨 검색

[ROS catkin 관련 주요 명령어]

- catkin 명령어는 ROS의 컴파일이나 빌드와 관련한 기능을 수행하는 명령어이다.

6. 패키지 명령어

명령어	중요도	명령어 풀이	세부 설명
rospack	★★★	ros+pack(age)	ROS 패키지와 관련된 정보 보기
rosinstall	★★☆	ros+install	ROS 추가 패키지 설치
rosdep	★★☆	ros+dep(endencies)	해당 패키지의 의존성 파일 설치
roslocate	☆☆☆	ros+locate	ROS 패키지 정보 관련 명령어
roscreate-pkg	☆☆☆	ros+create-pkg	ROS 패키지 자동 생성(구 rosbuild 시스템에서 사용)
rosmake	☆☆☆	ros+make	ROS 패키지를 빌드(구 rosbuild 시스템에서 사용)

[ROS 패키지 관련 주요 명령어]

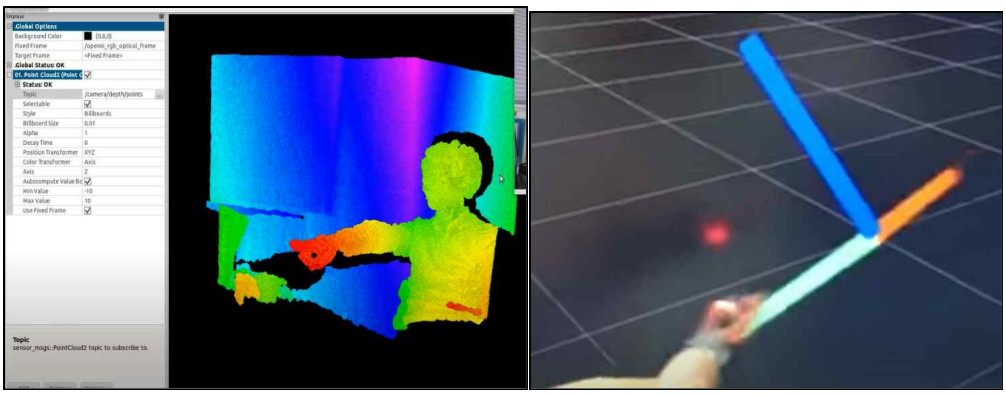
- 패키지 명령어는 설치에 관한 기능이나, 파일의 의존성 등을 확인할 때 사용하는 명령어이다.

Chapter 6. ROS 도구

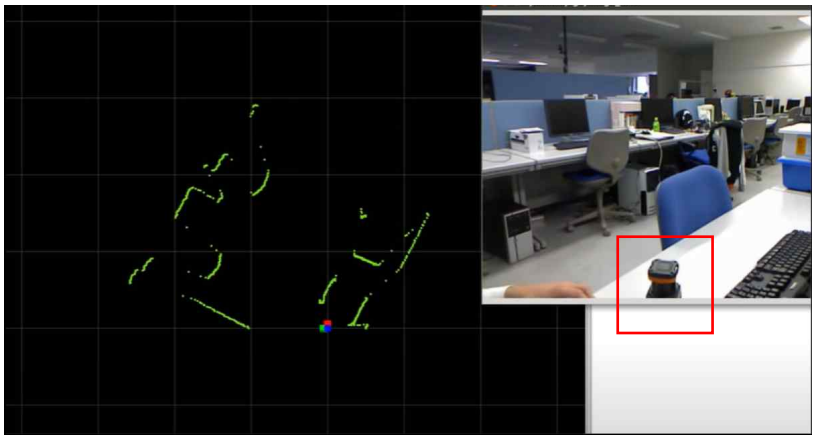
1. Rviz

ROS의 3차원 시각화 도구로, 센서 데이터를 시각화하거나, 로봇 외형 및 동작 표현, 내비게이션, 매니퓰레이션, 원격 제어와 같은 기능을 포함하고 있다.

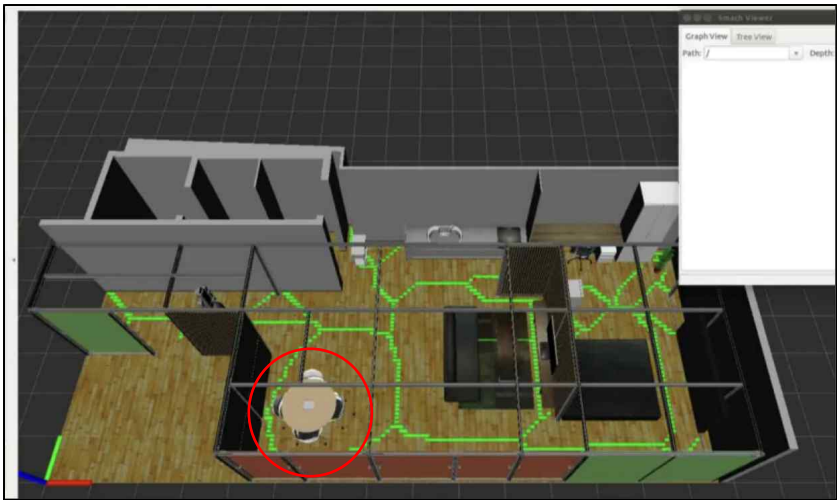
센서 데이터 시각화의 예) 레이저 거리 센서의 거리 데이터 표기, Depth 카메라의 데이터를 포인트 클라우드 형태로 시각화, 카메라의 영상 데이터나 IMU 센서의 관성 데이터 등을 시각화.



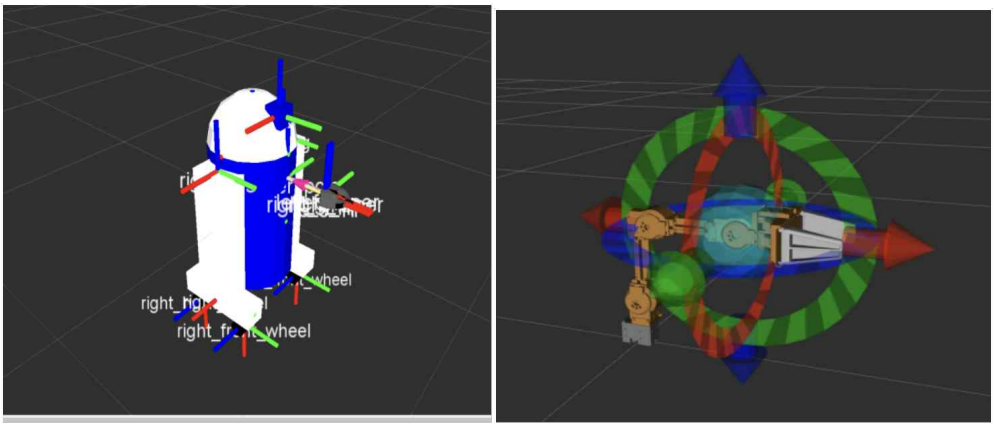
[Kinect 포인트 클라우드 데이터(좌), IMU센서의 관성 데이터(우) 시각화]



[레이저 거리 센서의 거리 값을 시각화한 화면]



[환경, 로봇, 경로를 시각화한 화면(녹색 선이 경로, 붉은 원 안의 물체가 로봇이다)]



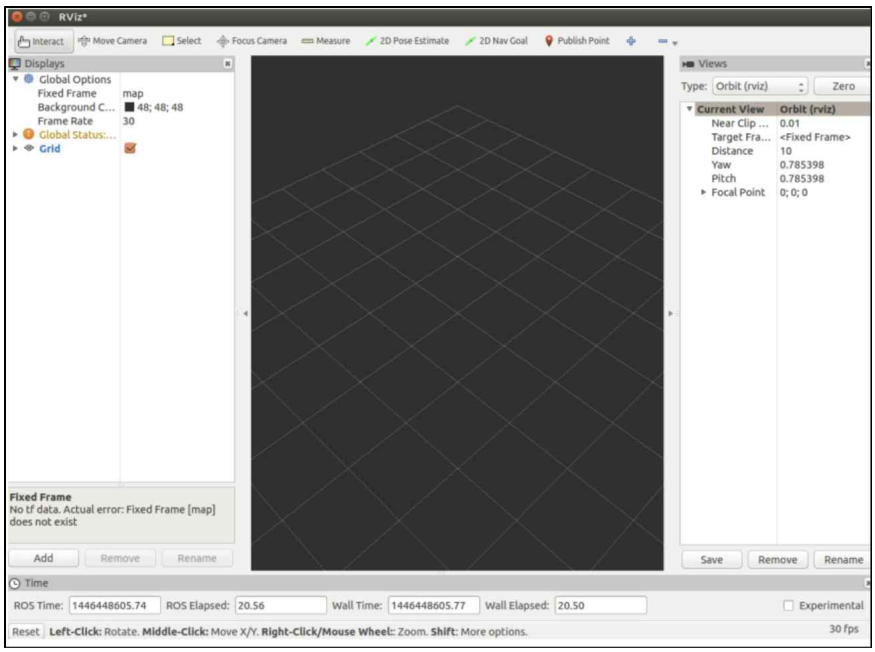
[R2-D2 로봇 모델(좌), 인터렉티브 마커를 활용한 IK 목표 위치와 경로 표시(우)]

RViz 설치 및 실행 과정

a. **설치** : $ sudo apt install ros-kinetic-rviz(ros kinetic 기준)

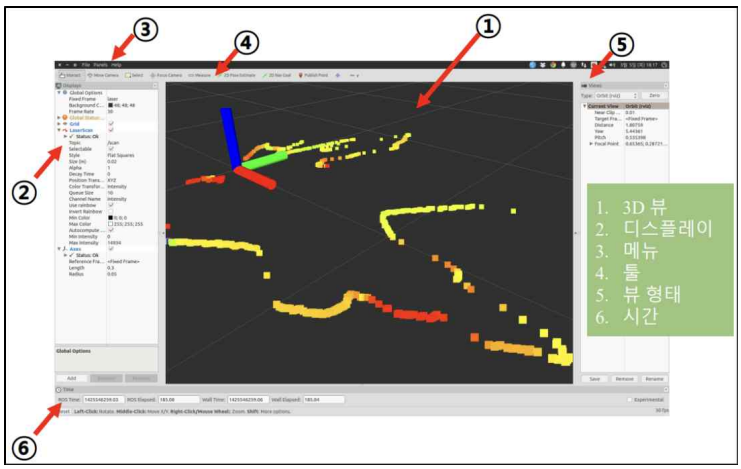
b. **실행** : $ rosrun rviz rviz 또는 $ rviz

c. **초기 화면**



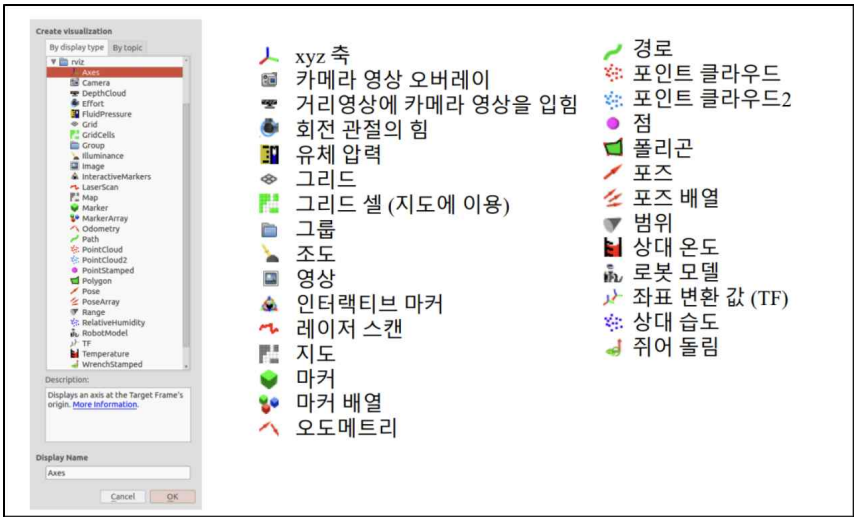
[RViz를 처음 실행했을 때의 화면(미설정)]

RViz 화면 구성



[레이저 거리 센서의 데이터를 시각화한 화면]

RViz 디스플레이 종류



[디스플레이의 ADD를 클릭하면 디스플레이 종류를 선택할 수 있다]

2. rqt

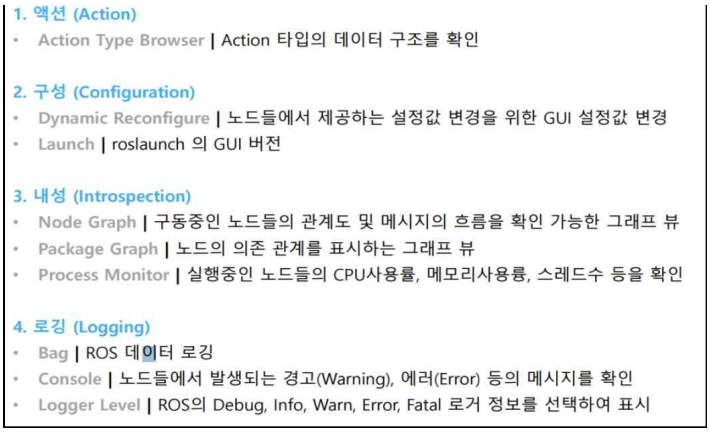
rqt는 rqt_bag, rqt_plot, rqt_graph 등을 플러그인으로 하는 ROS의 종합 GUI 툴로, Qt로 개발되어 있어서 사용자들이 자유롭게 플러그인을 개발하여 추가할 수 있다. 아래는 대표적인 플러그인들이다.

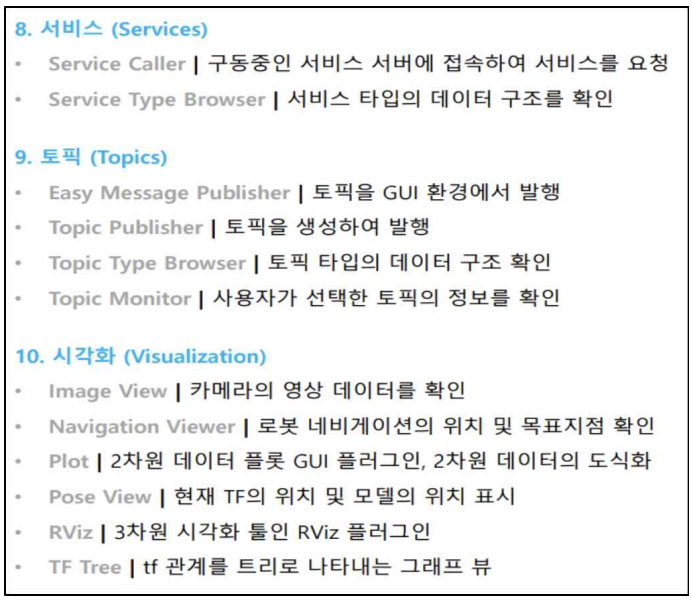
- rqt_graph : 노드와 그들 사이의 연결 정보 표시

- rqt_plot : 인코더, 전압, 또는 시간이 지남에 따라 변화하는 숫자를 플로팅

- rqt_bag : 데이터를 메시지 형태로 기록하고 재생

- rqt_image_view : 카메라의 영상 데이터를 확인



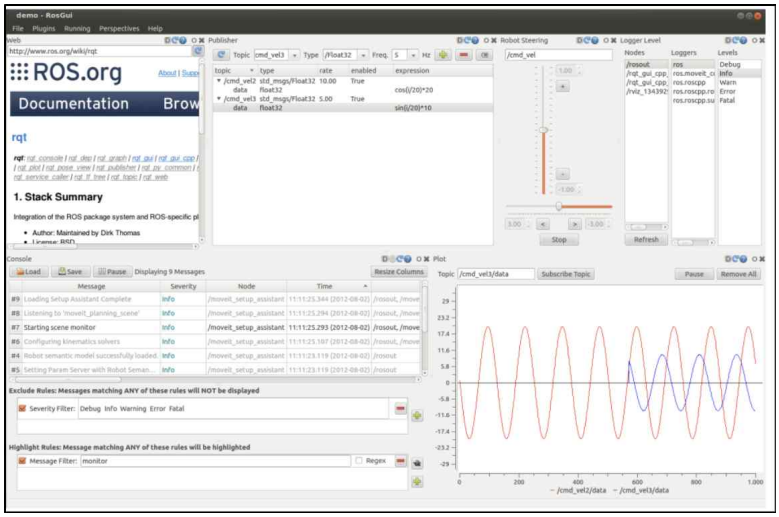


rqt 설치 : $ sudo apt install ros-kinetic-rqt ros-kinetic-rqt-common-plugins

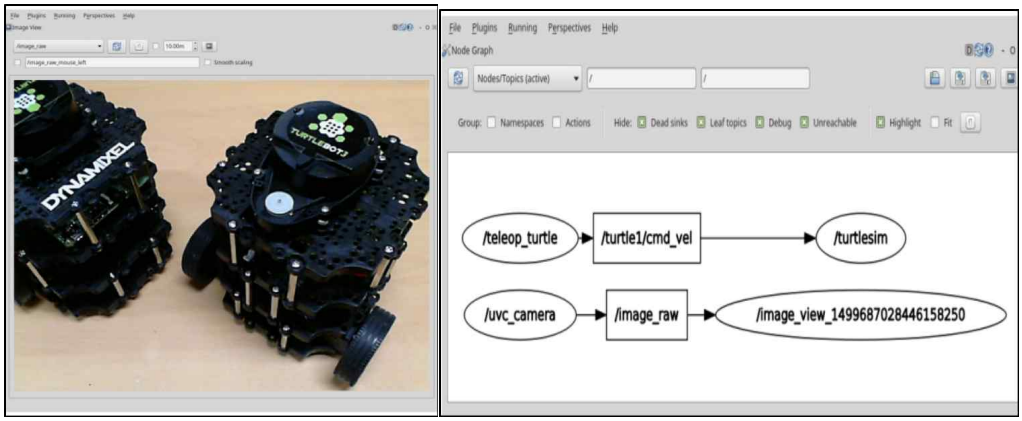
rqt 실행 : $ rqt

[rqt 초기 화면]

rqt 사용 예시



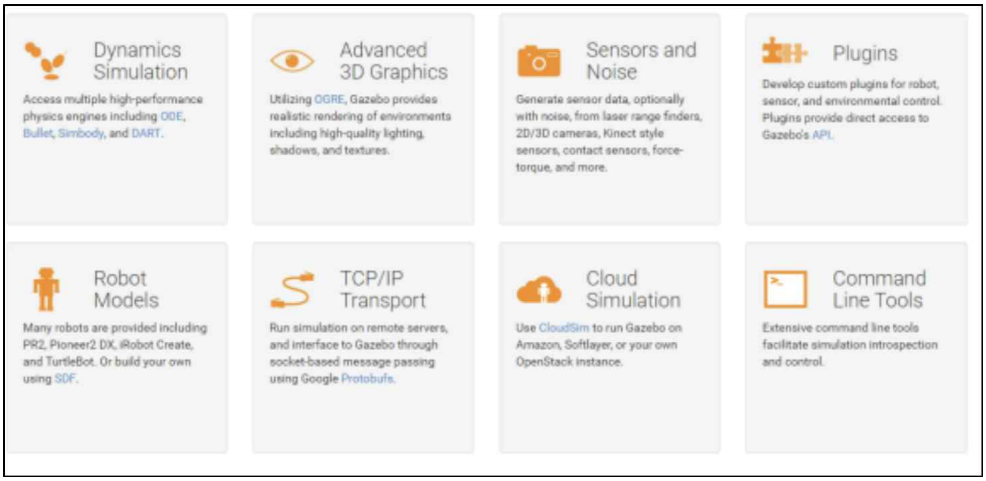
[example 1]



[example 2]

3. Gazebo

가제보는 로봇 개발을 위한 3차원 시뮬레이터에 대한 로봇, 센서, 환경 모델을 지원하고, 무엇보다도 물리 엔진이 탑재되어 있기 때문에 실제와 유사한 결과를 시뮬레이션할 수 있다.



[가제보의 기능들]

Chapter 7. ROS 기본 프로그래밍

1. ROS 프로그래밍 이전에 알아둬야 할 사항

- 표준 단위 : ROS에서는 회전축에 대해서는 오른손 법칙을 기준으로 하며, x축은 forward, y축은 left, z축은 up 방향을 가리킨다. 다른 물리량의 표준 단위에 대해서는 SI 단위계를 사용한다.

- ROS 프로그래밍 규칙에 대해서는 ROS 위키의 ROS C++ style guide를 참조할 수 있다.

- ROS에서 사용하는 변수는 int형, uint형, float형, string형, bool형을 비롯해 time형이나 duration과 같은 처음 접하는 변수형도 사용한다. 자세한 사항은 아래의 표나 ROS위키 msg(메시지) 항목을 참조할 수 있다.

Primitive Type	Serialization	C++	Python
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string (4)	std::string	str
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

Array Type	Serialization	C++	Python
fixed-length	no extra serialization	boost::array, std::vector	tuple
variable-length	uint32 length prefix	std::vector	tuple
uint8[]			bytes
bool[]		std::vector<uint8_t>	list of bool

[ROS에서 사용하는 데이터 타입에 대한 표]

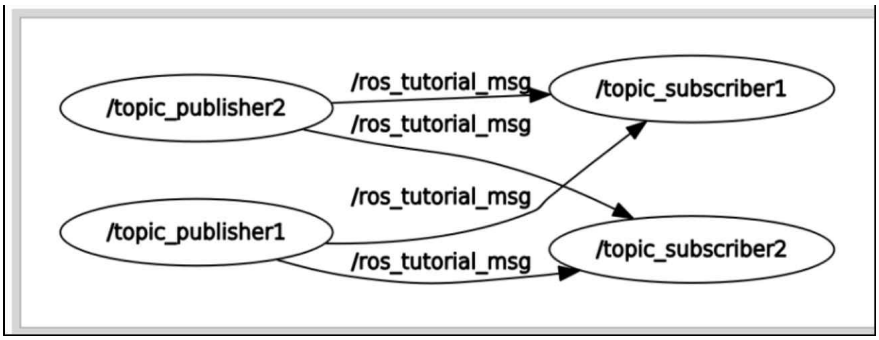
2. roslaunch 사용법

기본적으로 rosrun이 하나의 노드를 실행하는 명령어라면, roslaunch는 하나 이상의 정해진 노드를 실행할 수 있는 명령어이다.주 기능이 아닌, 다른 옵션을 부여하여 실행할 때에는 파라미터나 노드의 이름 변경, 환경 변수 변경, ROS_ROOT 및 ROS_PACKAGE_PATH 설정 등의 다양한 기능을 수행할 수 있다.

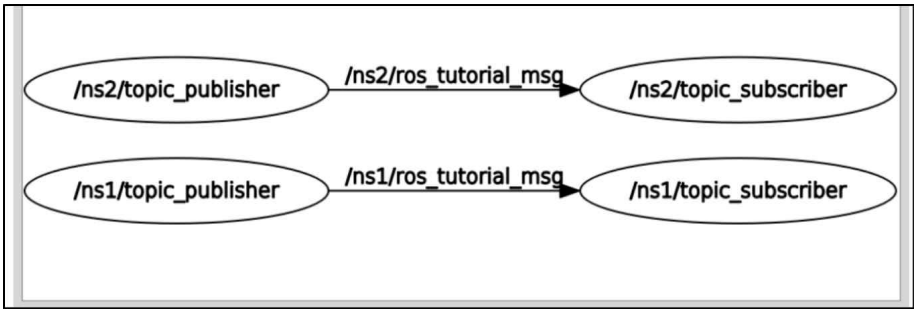
roslaunch는 '*.launch'라는 파일을 사용하여 실행 노드를 설정하고, 이는 XML 기반이다. 추가적으로 태그별 옵션도 제공한다.

- 실행 명령어 : roslaunch <패키지 명> <roslaunch 파일>

rosluanch를 사용하면 퍼블리셔 노드와 서브스크라이버 노드를 각각 두 개씩 구동하면서 서로 별도의 메시지를 통신하게 할 수 있다.



[roslaunch 네임스페이스 태그 사용 전▲]



[roslaunch 네임스페이스 태그 사용 후▲]

▶ launch의 태그 목록

<launch>	roslaunch 구문의 시작과 끝을 가리킨다.
<node>	노드 실행에 대한 태그이다. 패키지, 노드명, 실행명을 변경할 수 있다.
<machine>	노드를 실행하는 PC의 이름, address, ros-root, ros-package-path 등을 설정할 수 있다.
<include>	다른 패키지나 같은 패키지에 속해 있는 다른 launch를 불러와 하나의 launch파일처럼 실행할 수 있다.
<remap>	노드 이름, 토픽 이름 등의 노드에서 사용 중인 ROS 변수의 이름을 변경할 수 있다.
<env>	경로, IP 등의 환경 변수를 설정한다. (거의 안쓰임)
<param>	매개변수 이름, 타이프, 값 등을 설정한다
<rosparam>	rosparam 명령어 처럼 load, dump, delete 등 매개변수 정보의 확인 및 수정한다.
<group>	실행되는 노드를 그룹화할 때 사용한다.
<test>	노드를 테스트할 때 사용한다. <node>와 비슷하지만 테스트에 사용할 수 있는 옵션들이 추가되어 있다.
<arg>	launch 파일 내에 변수를 정의할 수 있어서 아래와 같이 실행할 때 매개변수를 변경 시킬 수도 있다.

```
<launch>
  <arg name="update_ period" default="10" />
  <param name="timing" value="$(arg update_ period)"/>
</launch>

roslaunch my_package my_package.launch update_ period:=30
```