

스마트 모빌리티 프로그래밍

Ch 21. ROS(Robot Operating System)



김 영 탁

영남대학교 기계IT대학 정보통신공학과
(Tel : +82-53-810-3940; E-mail : yse09@ynu.ac.kr)

Outline

- ◆ 자율 주행 자동차의 기능 구조, 소프트웨어 플랫폼
- ◆ ROS (Robot Operating System) 란?
- ◆ ROS 구조와 개념
 - 파일 시스템 레벨
 - 연산 그래프 시스템
 - 커뮤니티 레벨
- ◆ ROS 설치
 - Ubuntu에 설치
- ◆ ROS 기반 분산 처리 기능 구현 실습

자율주행 자동차 (Autonomous Vehicle) 기능 구조 및 소프트웨어 플랫폼

자율주행 단계 6 등급

◆ 자율주행 단계

특징		내용
레벨 0	비자동 (No Automation)	운전자가 모든 조작을 담당. 전방충돌방지(FCA), 후측방 충돌경고(BCW) 같은 센서로 알림
레벨 1	운전자 지원 (Driver Assistance)	차로 유지보조(LFA), 스마트 크루즈컨트롤(SCC) 기능을 통해 주행을 보조
레벨 2	부분 자동화 (Partial Automation)	고속도로 주행보조(HDA) 기능으로 레벨 2보다 더 높은 수준의 보조
레벨 3	조건부 자율주행 (Conditional Automation)	특정 조건에서 자율주행 동작, 긴급상황에서는 운전자의 개입이 필요
레벨 4	고도 자율주행 (High Automation)	긴급상황에 대한 대처도 시스템이 담당하지만 필요시 운전자의 개입이 가능함
레벨 5	완전 자율주행 (Full Automation)	운전자가 필요 없음

자율주행 자동차

◆ NURO R2

- NURO Autonomous Delivery: <https://www.youtube.com/watch?v=VFOisGUSDGk>

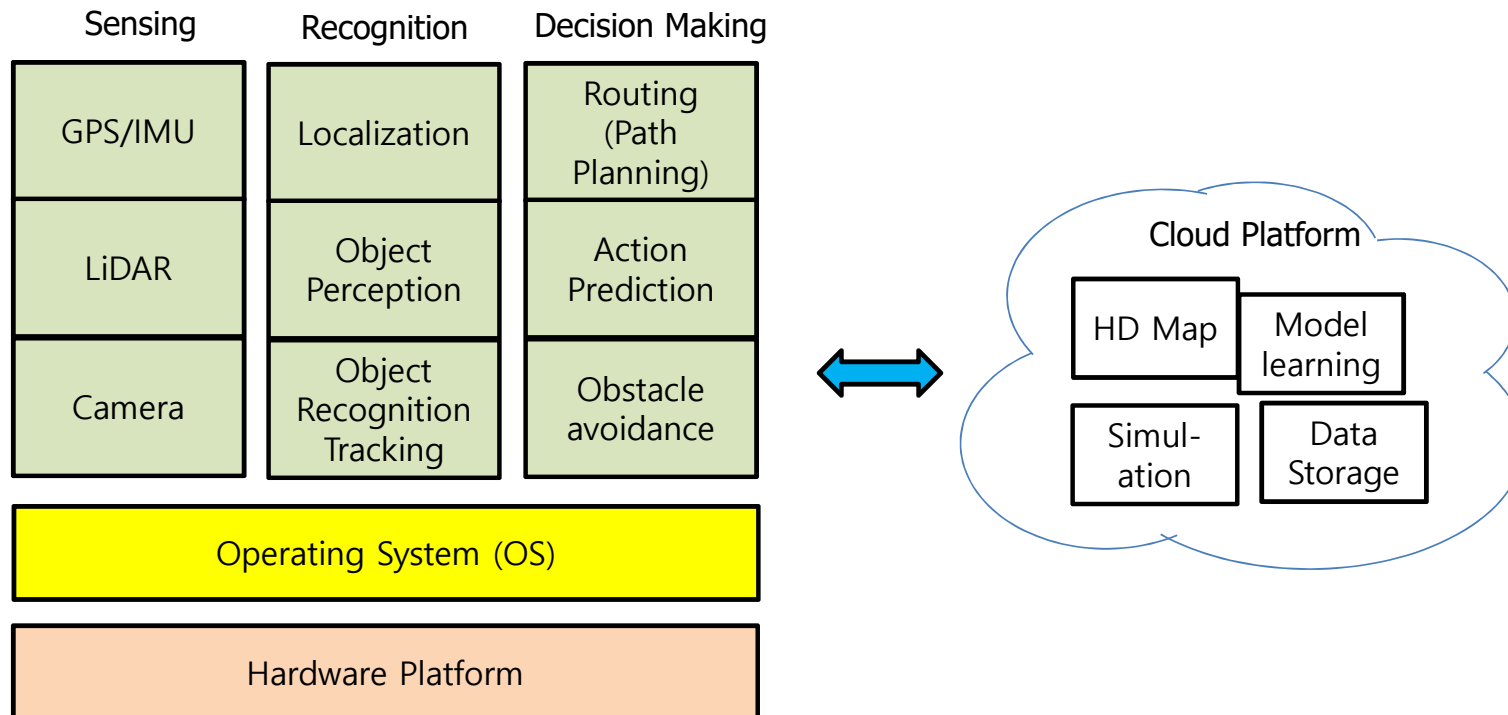
◆ Self-delivering autonomous robots

- https://www.youtube.com/watch?v=dagjQW_jgtE

◆ Uber



자율주행 자동차의 기본 기능 구조

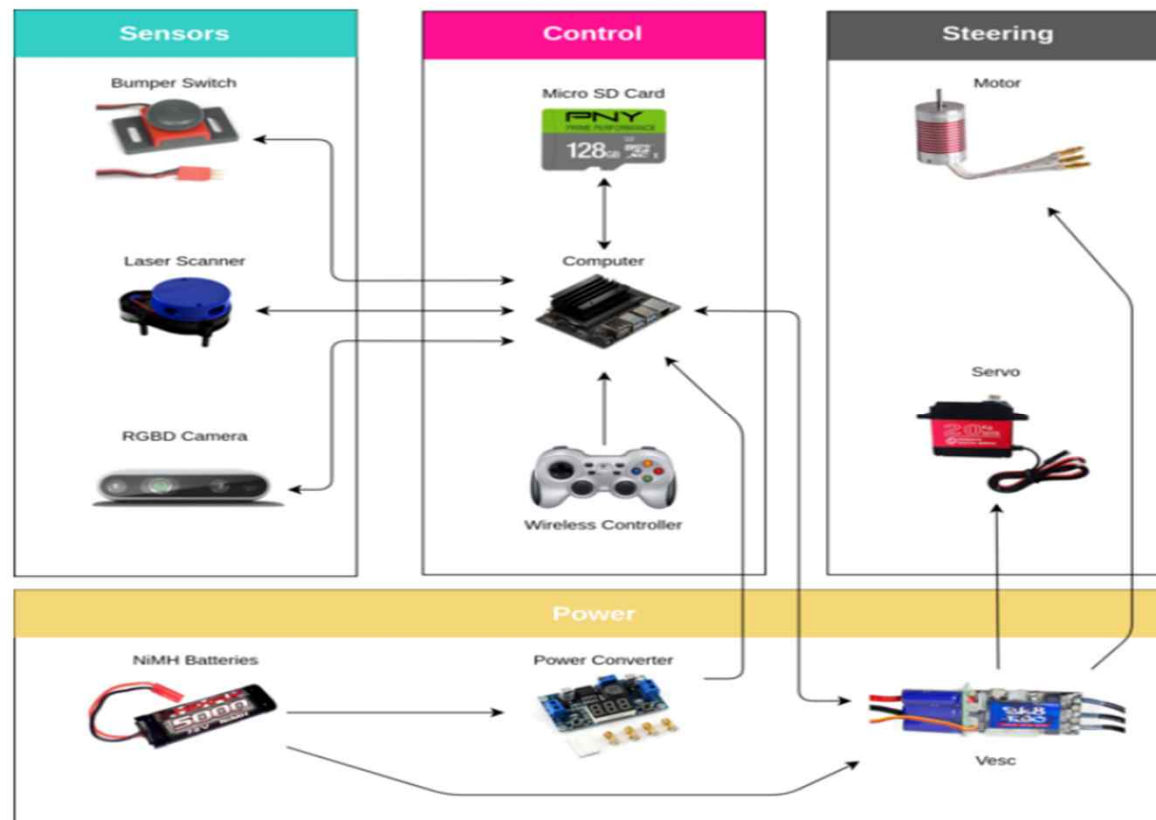


GPS (global positioning system)
 IMU (Inertial Movement Unit)
 LiDAR (Light Detection and Ranging)

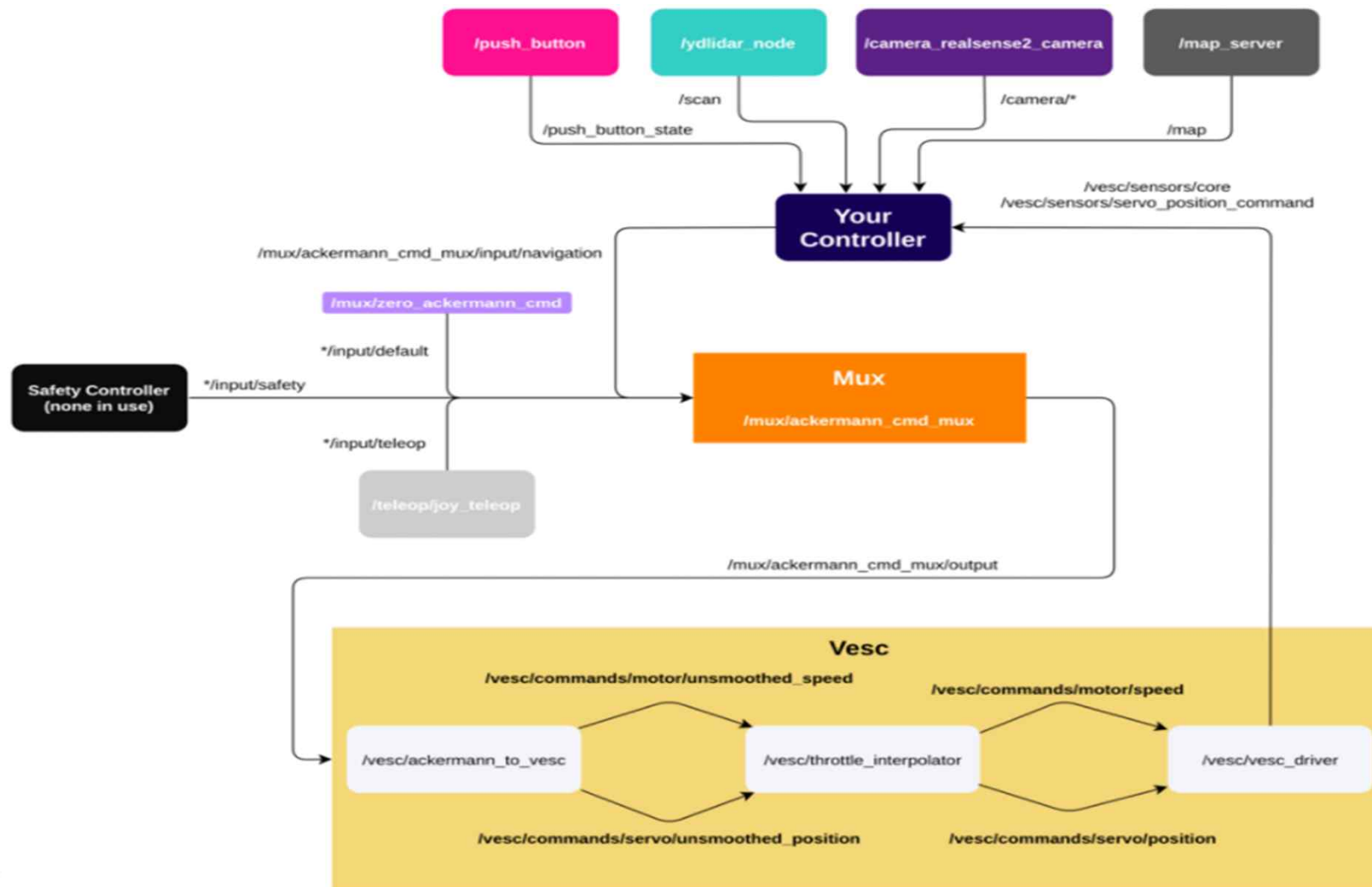
MuSHR

◆ MuSHR – The UW (Univ. of Washington) Open Racecar Project

- <https://mushr.io/tutorials/overview/>



MuSHR Functional Software Block Diagram



한백전자 LiDAR Steering Smart Car

◆ 한백전자 LiDAR Steering Smart Car

- 라이다 센서, 레이더 센서, 그리고 조향 시스템을 장착한 자율 주행 자동차로봇

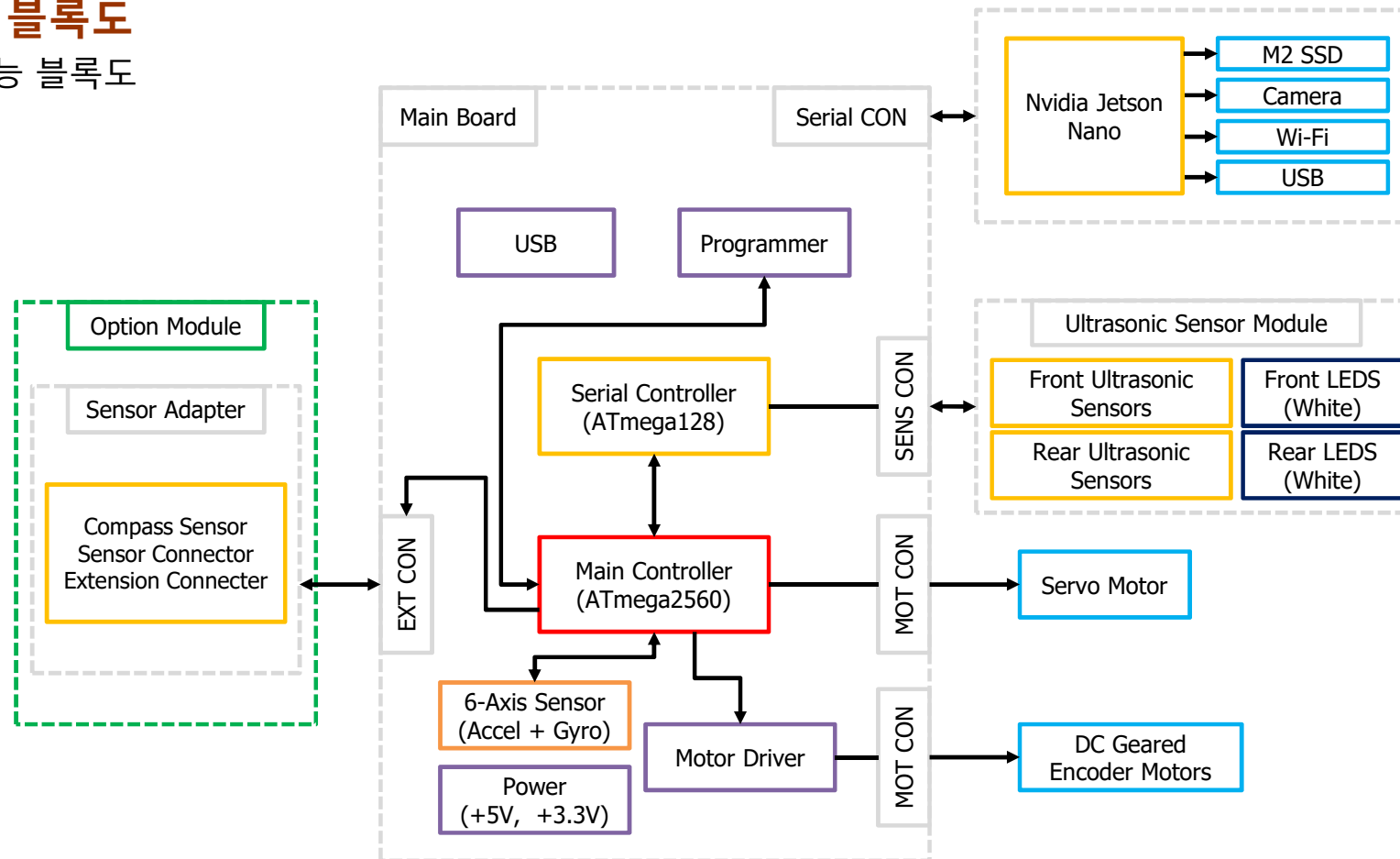


항목	규격, 모델
프로세서(Atmega 2560)	고성능 저전력 AVR 8bit Micro Controller 비 휘발성 프로그램과 데이터 메모리
ATmega128	
ATmega8	
MOTOR driver	L298P
DC MOTOR	IG32GM
Servo MOTOR	
Ultrasonic Sensor	MA40S4R/S
Acceleration & Gyro sensor	MPU-6050
Compass sensor	AK8975
LiDAR	RP Lidar A1M8
Encoder	05Type 1/4

한백 전자 LiDAR Steering Smart Car 기능 블록도

◆ 기능 블록도

- 기능 블록도



LiDAR Steering Smart Car 하드웨어 제원

◆ 속도 : 시속 15km (최대 0.32m/s)

◆ Atemga2560

- 센서 값을 읽어 현재 상태 파악, 정보를 통해 모터 구동, 장비의 동작

◆ 구동부 동작

- 4개의 바퀴를 1개의 DC모터가 제어하여 사륜구동으로 동작
- 엔코더가 내장된 모터, 동작 상태 감지 가능

◆ 조향 동작

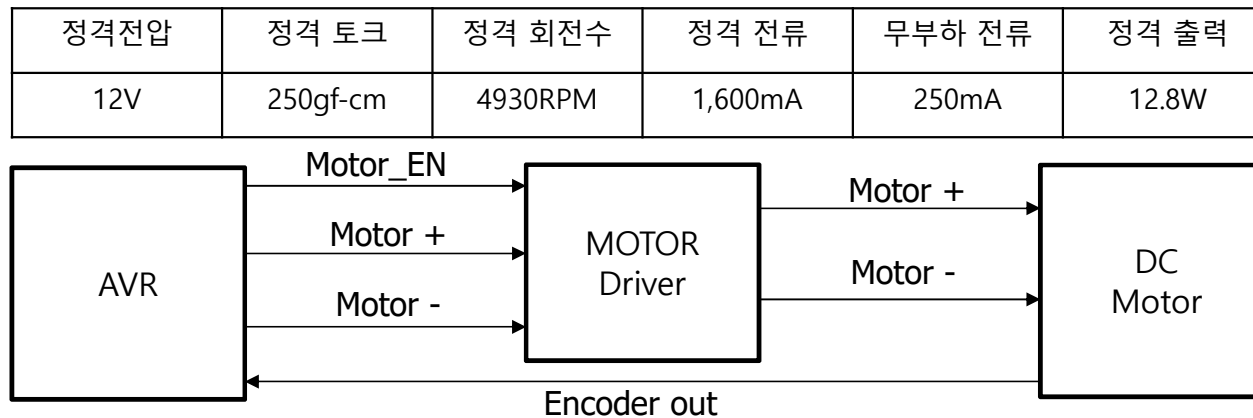
- 전면의 2개의 바퀴를 1개의 서보 모터가 제어



LiDAR Steering Smart Car의 DC 모터 특성

◆ SmartCar DC 모터 특성

- DC Geared 모터 사용
- ATmega2560칩으로 부터 PWM 제어신호를 DC모터 드라이버를 통해 모터를 제어
- 모터 왼쪽2개, 오른쪽 2개가 동시에 제어가 됨.
 - 전진, 후진, 우회전, 좌회전 구동 가능
 - 시계방향, 반시계방향 회전 제어
- 뒤쪽 2개의 모터는 엔코더가 내장
 - 어떤 방향으로 얼마나 이동한지 검증 가능

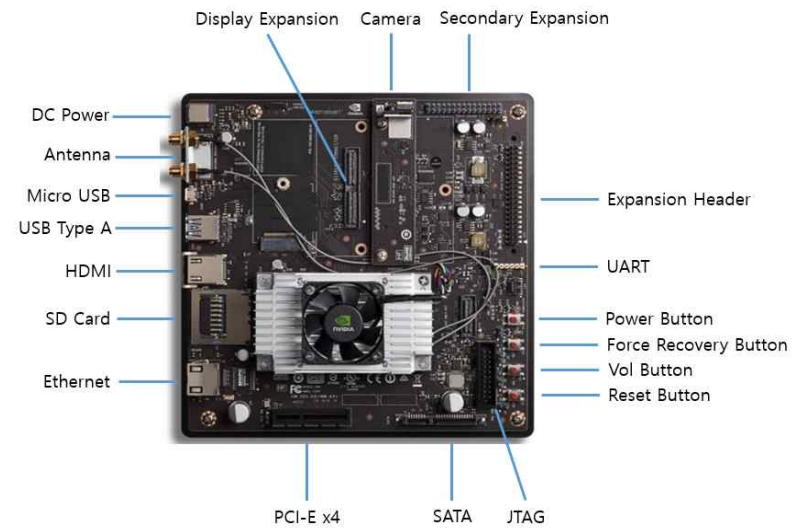


ch 21 - 12



휴인스 자율주행 자동차 (AI Lab-CAR)

◆ AI Lab-Car



ch 21 - 13

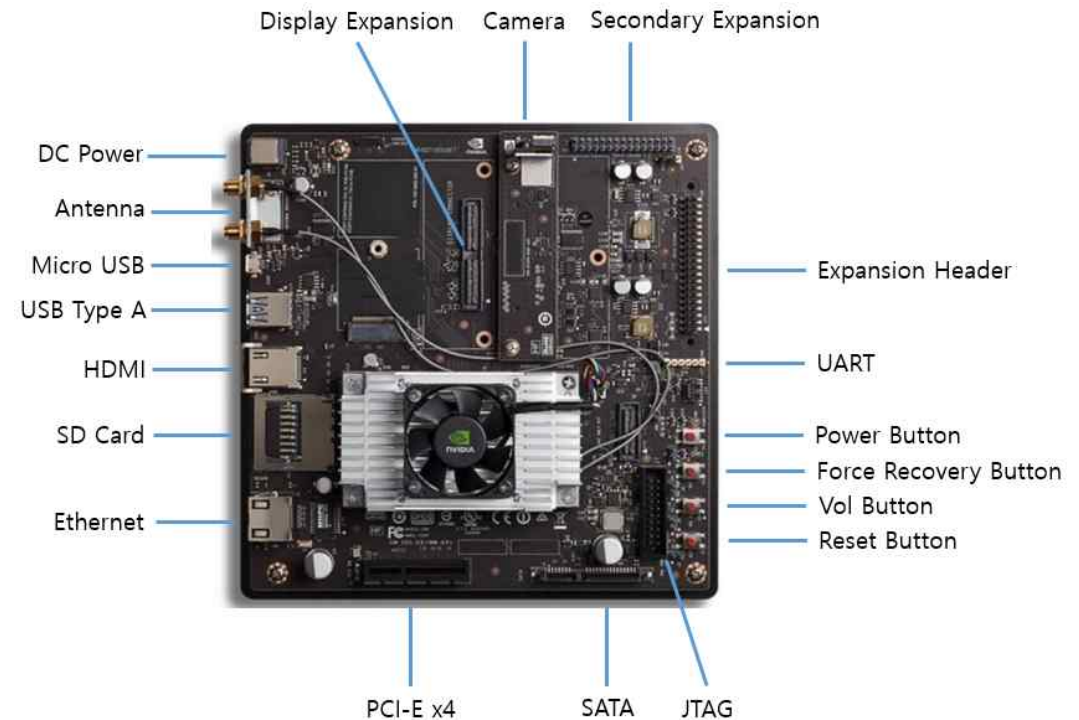
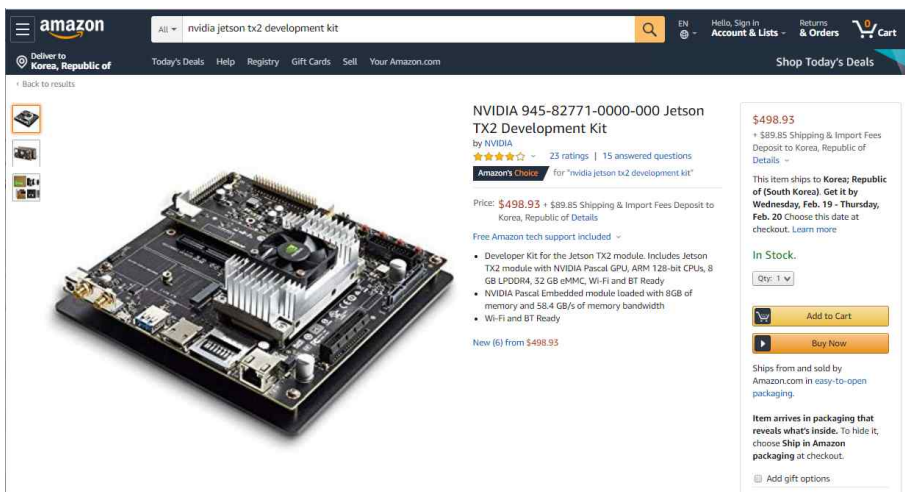


차세대 네트워킹 연구실
영남대학교 (YU-ANTL)

스마트 모빌리티 프로그래밍
교수 김 영 탁

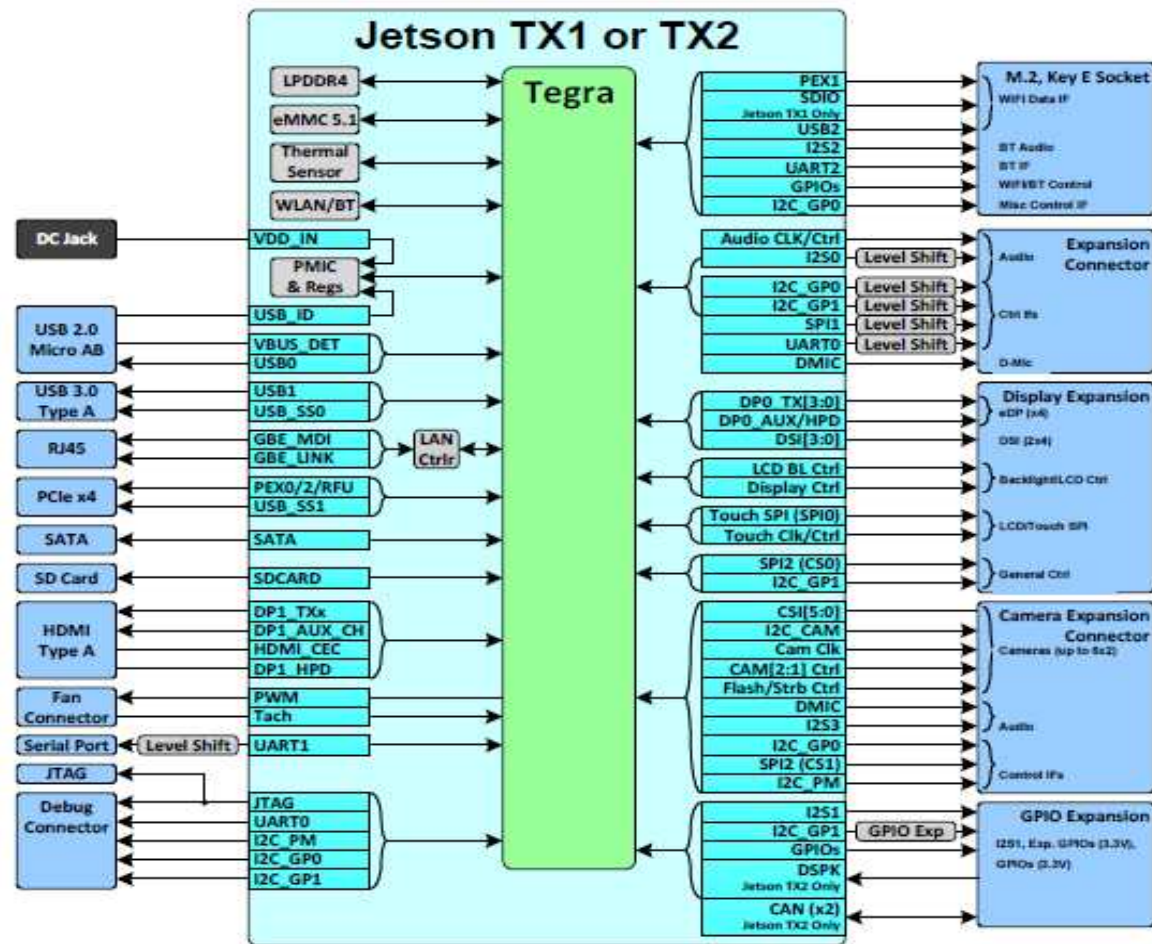
Nvidia Jetson TX2 Development Kit

◆ Nvidia Jetson TX2 Development Kit



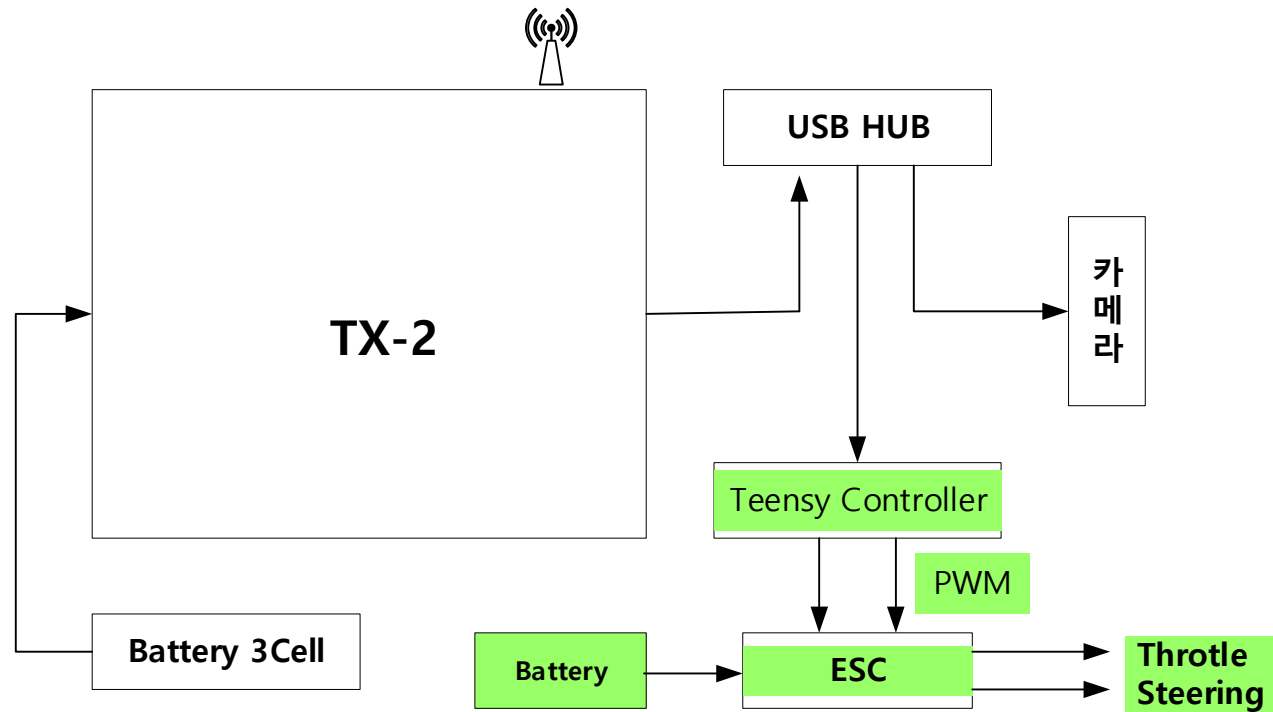
휴인스 AI Lab-Car 기능블록도

◆ AI Lab-Car 기능블록도



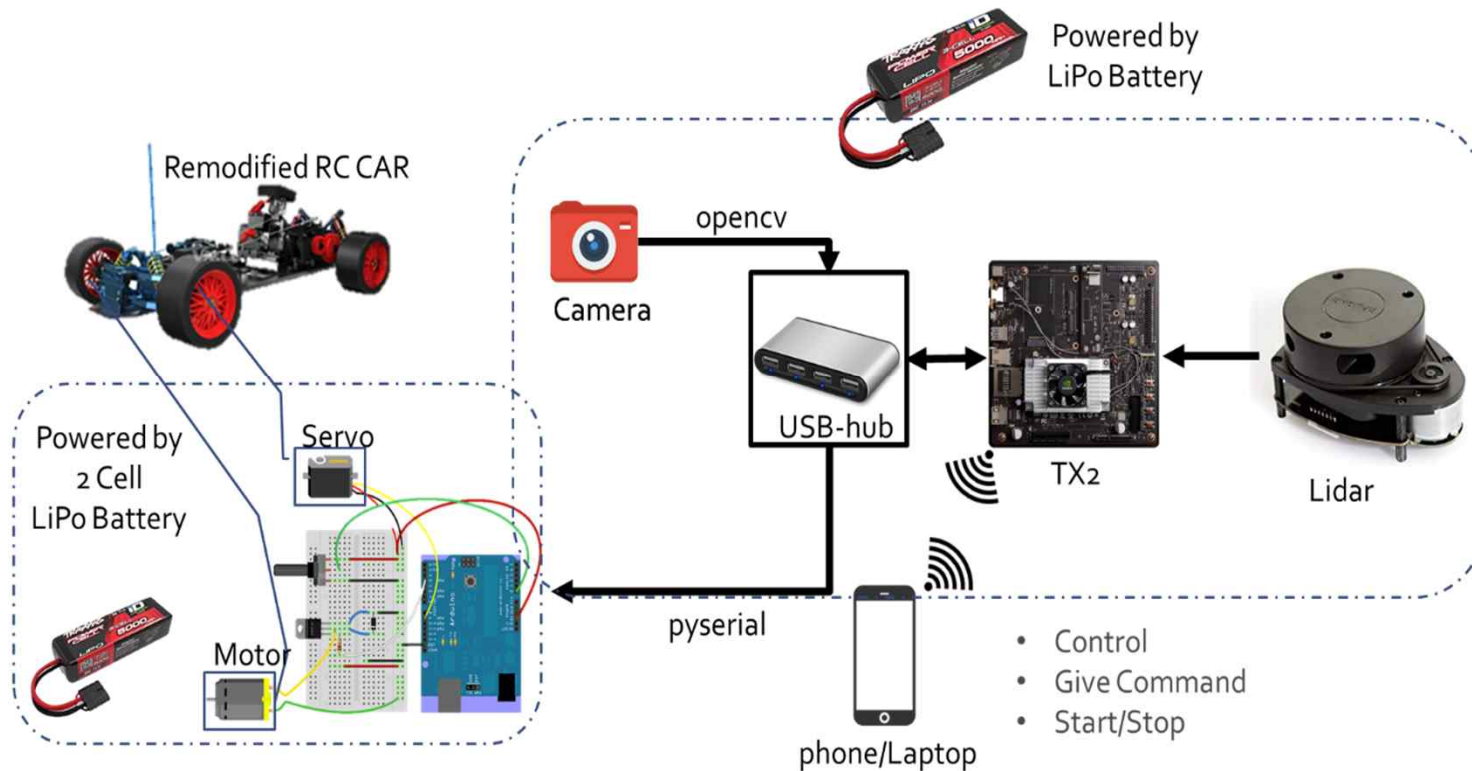
AI Lab-Car 기능 구성

◆ AI Lab-Car 기능 구성도



AI Lab-Car 기능 구성

◆ AI Lab-Car 기능 구성



차량 및 모터 제어부

- RC카
- 배터리(Lipo 2Cell)
- Teensy 3.2(Arduino)

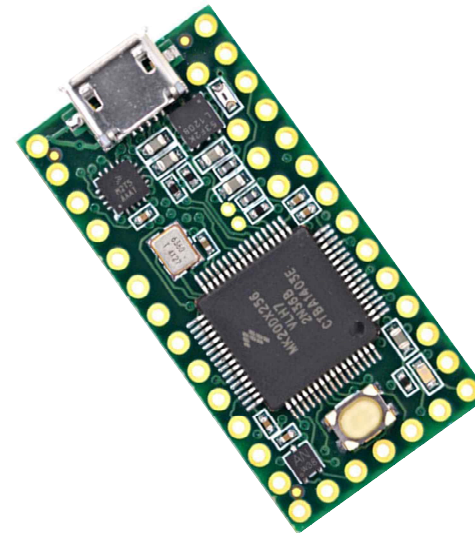
TX2

- Jetson TX2
- Lidar(RPLIDAR A1M8)
- Camera(Logitech c920)
- USB HUB
- 배터리(Lipo 3Cell)

AI Lab-Car Teensy Controller

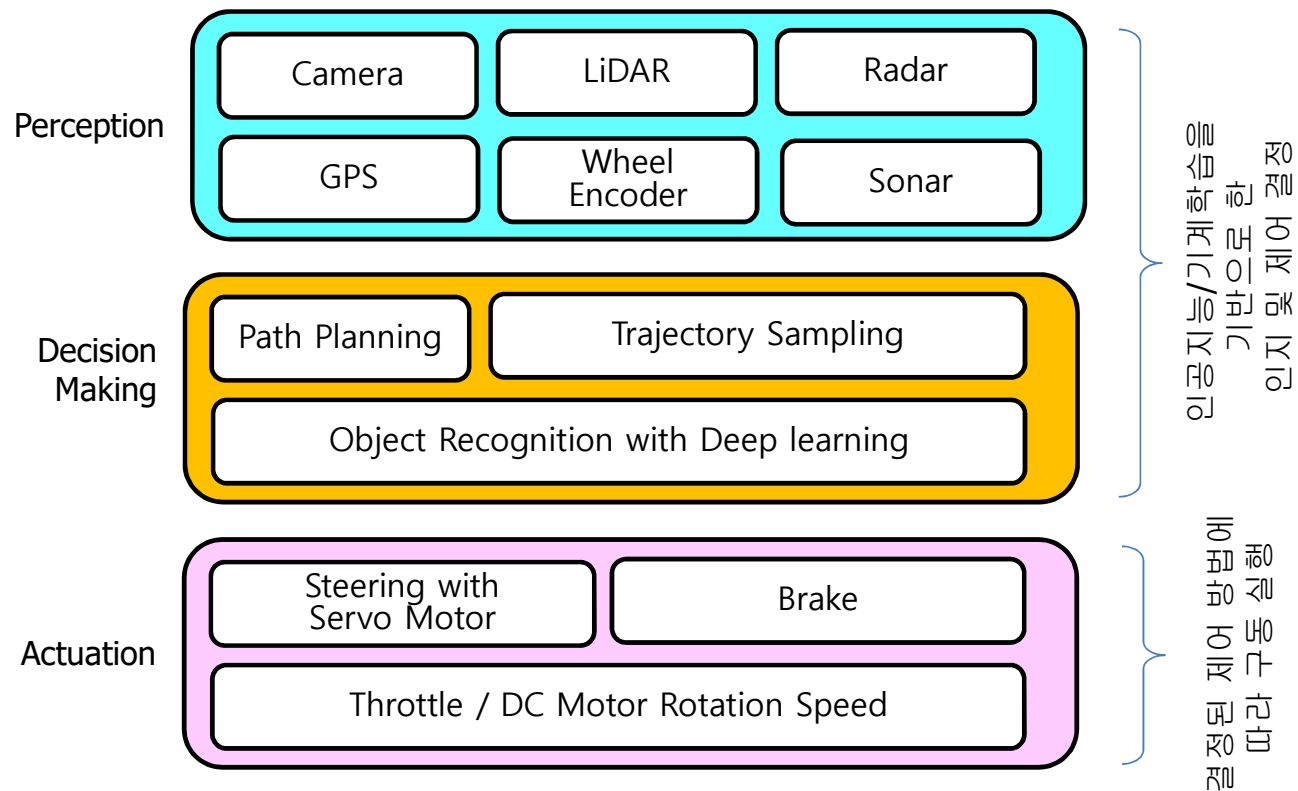
◆ Teensy 3.2

- USB 기반의 마이크로 컨트롤러 개발 시스템
- 높은 성능과 큰 메모리 용량, 풍부한 리소스를 제공
- 주요 특징
 - Arduino 소프트웨어와 라이브러리를 사용할 수 있음
 - 어떤 종류의 USB 디바이스 타입도 지원
 - 단일 푸시 버튼을 통해 프로그램 됨
 - 쉽게 사용할 수 있는 Teensy Loader Application
 - 무료로 사용할 수 있는 소프트웨어 개발 툴
 - Mac OS X, Linux, Windows 지원



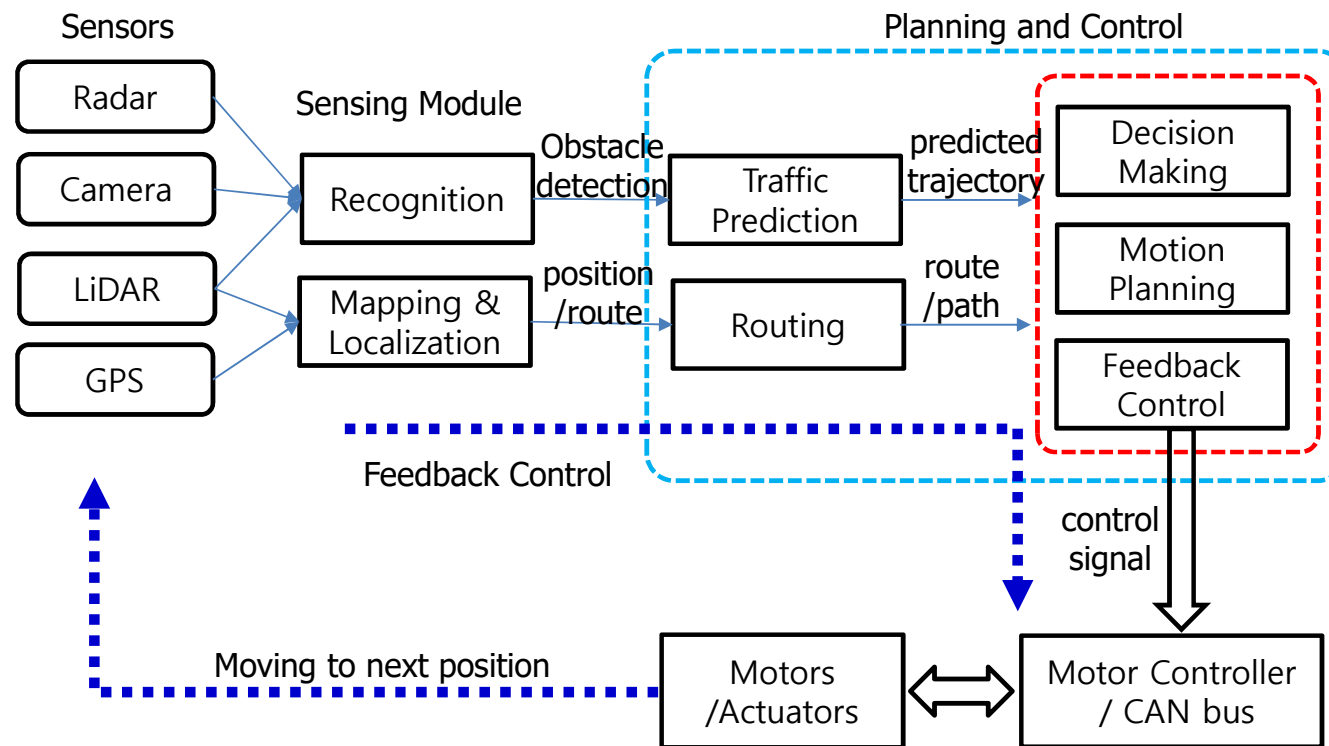
자율자동차의 인지, 제어 결정, 구동 제어

◆ 자율자동차의 개념적 기능 구분



Feedback Control in Autonomous Driving

◆ 자율 주행 자동차에서의 피드백 제어



스마트 모빌리티 소프트웨어를 위한 플랫폼

◆ Smart Mobility Software

- 다양한 센서들의 데이터 입력
- 실시간 영상 정보 입력, Deep Learning 기반 물체 인식
- 경로 계산, 차량 조향/구동 제어
- 차량 조향 제어 (steering control, servo motor)
- 차량 전후진 구동 (DC motor, step motor) 실시간 제어

◆ Smart Mobility Software 플랫폼의 핵심 기능

- 실시간으로 센서 모듈 데이터 처리
- 실시간으로 Deep Learning 기반 물체 인식 및 경로 제어
- 소프트웨어 모듈 간의 상호 통신 및 데이터 전달
- 하드웨어 및 구동부 제어 기능 모듈과 실시간 상호 연동
- Connected Car 기능
- 실시간 task scheduling
- 시스템 확장성, 융통성



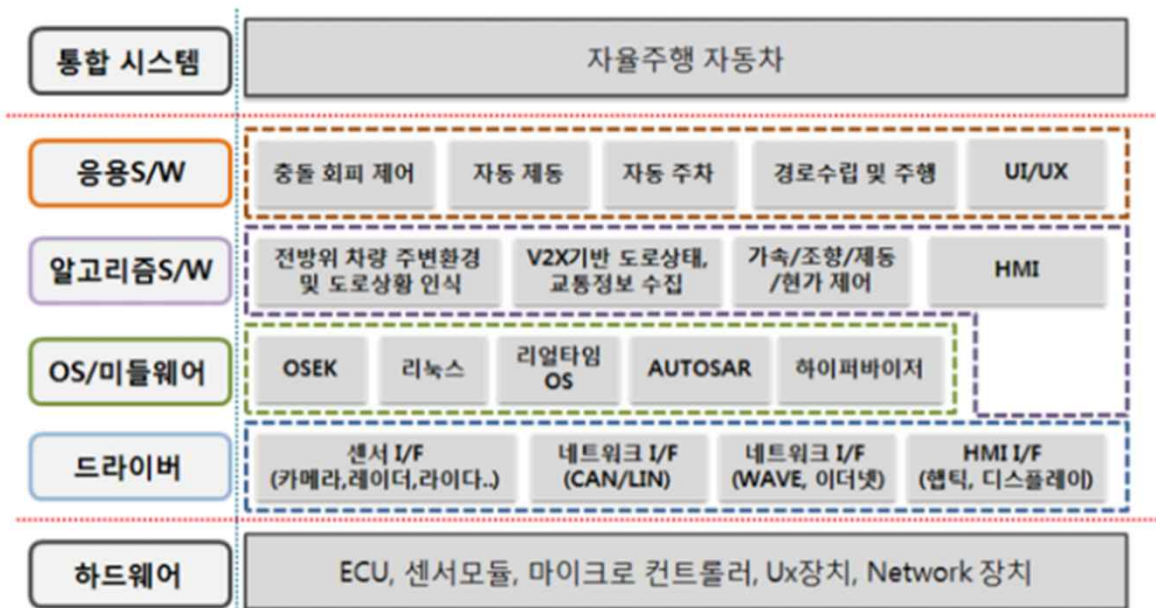
차량용 소프트웨어

◆ 차량용 소프트웨어 분류 및 주요 기술

분류	주요 기술 들
infotainment SW	<ul style="list-style-type: none"> 편의장치 등에 적용되는 인포테인먼트 영역 인포테인먼트 SW 플랫폼, 도구 등
전장 부품 통합 제어용 운영체제 (OS)	<ul style="list-style-type: none"> 차량의 안전장치에 적용되는 전자 장치 등의 SW 영역 안전 및 전장 부품용 SW 플랫폼, 도구 등 OSEK, RTOS 등 차량 통합 제어용 SW로서 운영체제 영역 <p>예: QNX RTOS, 현대차 ADAS 표준 SW 플랫폼 소프트웨어 플랫폼 (참고: https://blog.naver.com/shakey7/221798377322)</p>
외부 연계 및 통신, 서비스 SW	<ul style="list-style-type: none"> 차량 내외부 연결/통신에 사용되는 connected SW 영역 통신 및 빅데이터, 인공지능 및 IoT 연동, V2X 연동 등
개발 및 검증 SW	<ul style="list-style-type: none"> 차량 SW 개발 도구, 검증 도구, 검증 장치 구동 등

자율 주행 자동차 임베디드 SW 구조

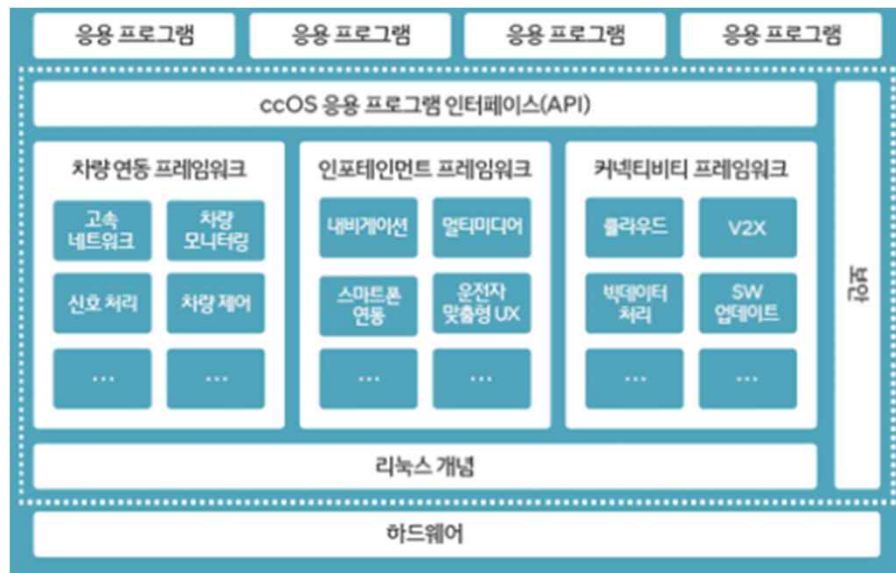
◆ 자율 주행 자동차 임베디드 SW 구조



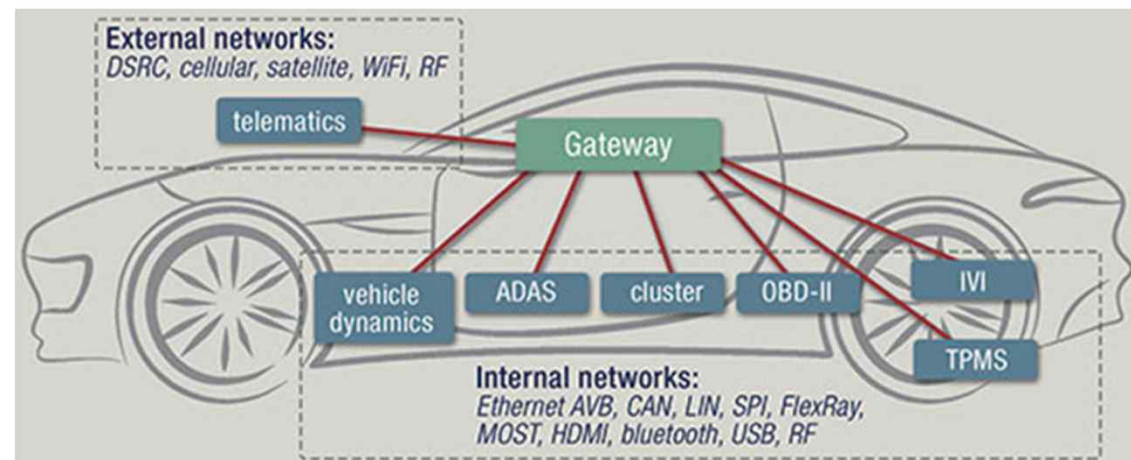
차량 내부 통신, 외부 연계 및 통신 소프트웨어

◆ 차량 내부 통신, 외부 연계 및 통신 소프트웨어

- IVN (in vehicle network): CAN, LIN에서 Ethernet 기반 연구가 활발히 진행 중
- OVN (outbound vehicle network): C-V2X



(ccOS: connected car OS)



ROS (Robot Operating System)

개요

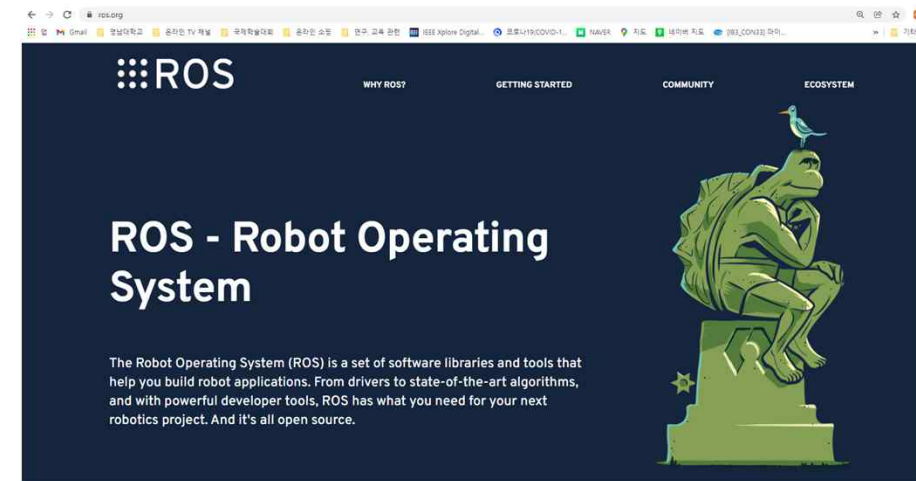
ROS 소개

◆ Robot Operating System (ROS)

- <https://www.ros.org/>
- <http://wiki.ros.org/ROS/Tutorials>
- <http://wiki.ros.org/catkin/Tutorials>
- 로봇을 위한 “메타” 오퍼레이팅 시스템
- 패키징, 빌드 툴을 포함
- 분산된 프로세스/머신 간의 통신 및 구성을 위한 아키텍처
- 시스템 런-타임과 데이터 분석을 위한 개발 도구
- 개발 언어에 독립된 아키텍처
 - C++, Python, Lisp, Java, 등.

◆ ROS는 아래 내용을 의미하지 않는다

- 실제 오퍼레이팅 시스템
- 프로그래밍 언어
- 프로그래밍 환경 / IDE
- Hard real-time 아키텍처



ROS 소개

◆ What is ROS (Robot Operating System)

- a software framework for programming robots
- prototypes originated from Stanford AI research, officially created and developed by Willow Garage (robotics research Lab. and incubator) starting in 2007
(source: https://en.wikipedia.org/wiki/Willow_Garage)
- Currently maintained by Open Source Robotics Foundation
- Consists of infrastructure, tools, capabilities, and ecosystem

Advantages and Disadvantages of ROS

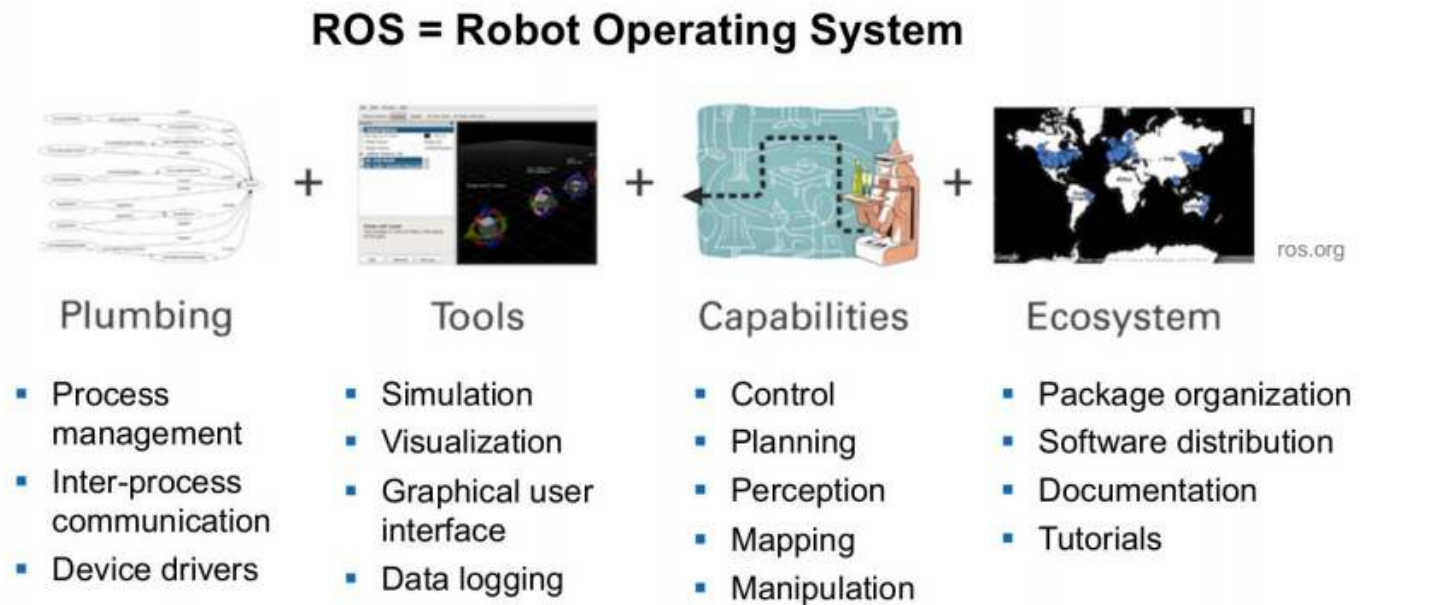
Advantages	Disadvantages
<ul style="list-style-type: none">▪ provides lots of infrastructure, tools, and capabilities▪ easy to try other people's work and share your own▪ large community▪ free, open source, BSD license	<ul style="list-style-type: none">▪ approaching maturity, but still changing▪ security and scalability are not first-class concerns▪ OSes other than Ubuntu Linux are not well supported
<ul style="list-style-type: none">▪ great for open-source and researchers	<ul style="list-style-type: none">▪ not great for mission-critical tasks



ROS 소개

◆ Robot Operating System (ROS)

- <https://www.ros.org/>



Slide Credit: Marco Hutter, ETH Zurich

ROS의 역사

- ◆ 2007년 스탠포드 인공지능 연구소에서 개발
- ◆ 2013년부터 OSRF에서 관리
- ◆ 오늘날, 많은 로봇, 대학, 회사에서 사용
- ◆ 로봇 프로그래밍의 준 표준



Meta-Operating System

◆ 메타운영체제(Meta-Operating System):

- 어플리케이션과 분산 컴퓨팅 자원 간의 가상화 레이어로 분산 컴퓨팅 자원을 활용하여, scheduling 및 로드, 감시, 에러 처리 등을 실행하는 시스템
- 즉, 윈도우, 리눅스, 안드로이드와 같은 전통적인 운영체제는 아니며, ROS는 기존의 전통적인 운영체제(리눅스, 윈도우, OS-X, 안드로이드)를 이용하고 있다.
- 기존 운영체제의 프로세스 관리 시스템, 파일 시스템, 유저 인터페이스, 프로그램 유틸(컴파일러, 스레드 모델 등)등을 사용
- 추가적으로 다수의 이기종 하드웨어 간의 데이터 송수신, scheduling, 에러 처리 등 로봇 응용 소프트웨어를 위한 필수 기능 들을 라이브러리 형태로 제공
- 기존 로봇 소프트웨어 프레임 워크를 기반으로 다양한 목적의 응용 프로그램을 개발, 관리, 제공하고 있으며 사용자 (user)들이 개발한 패키지 또한 유통하는 생태계(ecosystem)를 갖추고 있음

Meta-Operating System



Reference: <https://www.slideshare.net/yoonseokpyo/20160406-ros-1-for>

ROS(Robot Operating System)

◆ ROS Framework의 몇 가지 목표

- 가벼움: ROS는 가볍게 설계되어 ROS용으로 작성된 코드를 다른 로봇 소프트웨어 framework에 사용 가능
- ROS에 의존적이지 않은 라이브러리: 라이브러리를 개발할 때, 명확하고 기능적인 인터페이스를 갖는 ROS에 의존적이지 않도록 개발하는 것을 선호
- 언어 독립성: ROS framework은 최신의 어떤 언어로도 쉽게 구현할 수 있음
- 쉬운 테스트: ROS는 ros test라고 하는 단위/통합 테스트 framework 기반 위에 있는데, 설치 및 제거가 아주 쉬움
- 규모 적응성: ROS는 대규모 실행 시스템 및 개발 프로세스에도 적용 가능



ROS(Robot Operating System)

◆ 지원 운영체제

- ROS를 사용 가능한 운영체제(OS)로는 Ubuntu, OS X, Windows, Fedora, Gentoo, OpenSUSE, Debian, Raspbian, Arch, QNX Realtime OS 등이 있으나 기능 제한사항이 있을 수 있음
- 스마트폰 운영체제인 Android, iOS 의 경우, 부분적 사용 가능
- OS를 탑재할 수 없는 마이크로 컨트롤러 유닛(MCU)의 경우, Serial 통신, Bluetooth, LAN 경유로 통신할 수 있는 라이브러리 제공
- 기본적으로는 Ubuntu, OS X 에서 구동하는 것을 추천!

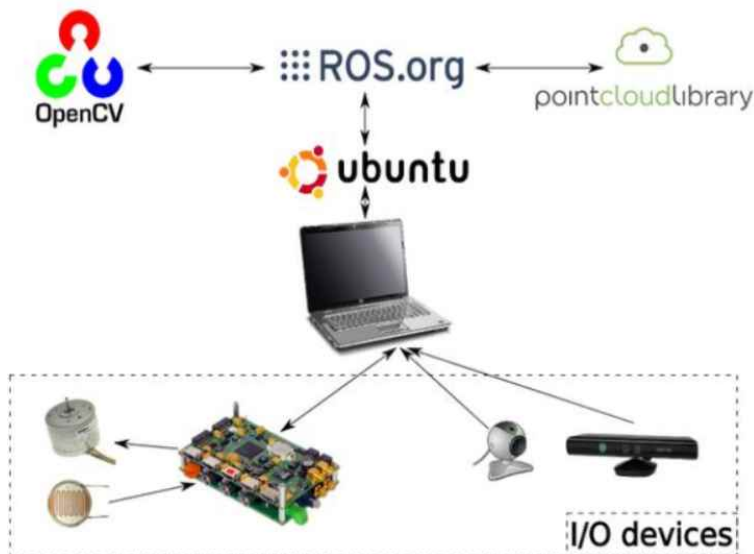


Ubuntu버전에 따른 ROS버전 선택

ROS Distribution	Supported OS	Experimental OS
Melodic Morenia	Ubuntu 18.04(LTS); Debian9	OS X(Homebrew), Gentoo, Ubuntu ARM, and OpenEmbedded/Yocto
Kinetic Kame	Ubuntu 16.34(LTS) and 15.10; Debian 8	OS X(Homebrew), Gentoo, Ubuntu ARM, and OpenEmbedded/Yocto
Jade Turtle	Ubuntu 15.04,14.10, and 14.04; Debian 8	OS X(Homebrew), Gentoo, Arch Linux, Android NDK, Ubuntu ARM, OpenEmbedded/Yocto
Indigo Igloo (LTS)	Ubuntu 14.04(LTS) and 13.10; Debian 7	OS X(Homebrew), Gentoo, Arch Linux, Android NDK, Ubuntu ARM, OpenEmbedded/Yocto

ROS를 사용하는 이유

- ◆ 소프트웨어 측면의 많은 문제들을 해결하기 위해 설계
- ◆ 광범위한 센서, 액추에이터를 지원하는 일련의 라이브러리를 포함
- ◆ 쉬운 하드웨어 인터페이스 제공



ROS 설치 및 ROS 노드간 기능 시험

ROS 설치

◆ ROS 및 관련 패키지 설치

- Jetson Nano : ROS Melodic
- Raspberry Pi : ROS noetic



ROS 설치

◆ ROS 설치

- <https://www.ros.org/>



- Raspberry Pi 4에 ROS noetic 설치 guideline:
<https://varhowto.com/install-ros-noetic-raspberry-pi-4/>
- Raspberry Pi 4에 ROS melodic 설치 guideline:
<https://www.seeedstudio.com/blog/2019/08/01/installing-ros-melodic-on-raspberry-pi-4-and-rplidar-a1m8/>

ROS 다운로드

◆ ROS Base

- \$ sudo apt install ros-melodic-ros-base

◆ ROS Desktop

- \$ sudo apt install ros-melodic-desktop

◆ ROS Full

- \$ sudo apt install ros-melodic-desktop-full

◆ ROS Packge(선택)

- \$ sudo apt-get install ros-melodic-(PACKAGE NAME)



ROS설치를 위한 저장소, 키 지정

◆ ROS 저장소를 sources.list.d에 추가하기

- `$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`

◆ Access key 추가하기

- `$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654`

```
iron@antl:~$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --r
Executing: /tmp/apt-key-gpghome.jWMkrtLF8t/gpg.1.sh --keyserver hkp://keyserv
gpg: key F42ED6FBAB17C654: "Open Robotics <info@osrfoundation.org>" not chang
gpg: Total number processed: 1
gpg:                unchanged: 1
```

◆ 패키지 목록 업데이트

- `$ sudo apt update`

Install ROS(1)

◆ ROS 설치를 위해 저장소 등록과 인증키 다운로드

- `sudo apt update`
- `sudo apt upgrade`

◆ use 'sudo apt autoremove' to remove them

- 패지지에 사용되었던 임시 파일을 삭제
- `sudo apt autoremove`

◆ <http://packages.ros.org>에 접근할 수 있도록 `source.list.d`에 저장소(Repository)를 추가.

- `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
${lsb_release -sc} main" > /etc/apt/sources.list.d/ros-
latest.list'`

◆ 만약에 bad substitution이란 치환 오류가 발생하면 `lsb_release`를 bionic으로 변경할 것

```
다음 패키지가 자동으로 설치되었지만 더 이상 필요하지 않습니다:
apt-clone archdetect-deb bogl-bterm busybox-static cryptsetup-bin
dpkg-repack gir1.2-timezonemap-1.0 gir1.2-xml-1.0 grub-common
kde-window-manager kpackagetool5 kwayland-data kwin-common kwin-data
kwin-x11 libdebian-installer4 libdecorations2-5v5
libdecorations2private5v5 libkf5activities5 libkf5declarative-data
libkf5declarative5 libkf5globalaccelprivate5 libkf5idle5
libkf5kcmutils-data libkf5kcmutils5 libkf5package-data libkf5package5
libkf5plasma5 libkf5quickaddons5 libkf5waylandclients libkf5waylandserver5
libkscreenlocker5 libkwin4-effect-builtins1 libkwineffects11
libkwineffects11 libkwinrenderutils11 libkgsttools-p1 libqt5multimedia5
libqt5multimedia5-plugins libqt5multimediaquick-p5 libqt5multimediawidgets5
libxcb-composite0 libxcb-cursor0 libxcb-damage0 os-prober
python3-dbus.mainloop.pyqt5 python3-icu python3-pam python3-pyqt5
python3-pyqt5.qtsvg python3-pyqt5.qtwebkit python3-sip
qml-module-org-kde-quickcontrolsaddons qml-module-qtmultimedia
qml-module-qtquick2 rotate-taskset taskset-data
Use 'sudo apt autoremove' to remove them.
```

Install ROS(2)

◆ 패키지를 다운로드를 위한 인증키 등록

- `sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654`
- 다음 메시지가 출력되면 정상적으로 인증키를 가져온 것임

```
gpg: Total number processed: 1
gpg:             imported: 1
```

- `sudo apt update`

Install ROS(3)

◆ ROS Melodic 설치

- `sudo apt install ros-melodic-desktop-full`
- `sudo apt-get install python-pip`
- `sudo pip install -U rosdep`
- `sudo rosdep init`

◆ rosdep

- rosdep은 뒤에 적어준 패키지의 의존성 파일을 찾아서 설치해주는 명령어
(위치 : `/etc/ros/rosdep/sources.list.d`)

```
yasun95@yasun95-desktop:~$ sudo rosdep init
[sudo] yasun95의 암호:
Wrote /etc/ros/rosdep/sources.list.d/20-default.list
Recommended: please run

rosdep update
```

- `rosdep update`
- `echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc`
- `source ~/.bashrc`

Install ROS(4)

◆의존성 빌드 패키지 다운로드

- `sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool`
- `mkdir -p ~/catkin_ws/src`
- `cd ~/catkin_ws/src`
- `catkin_init_workspace`
- `cd ..`
- `catkin_make`

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ysj/catkin_ws/build
```

- **성공적으로 `catkin_make` 진행됨을 확인**
- `echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc`
- `source ~/.bashrc`

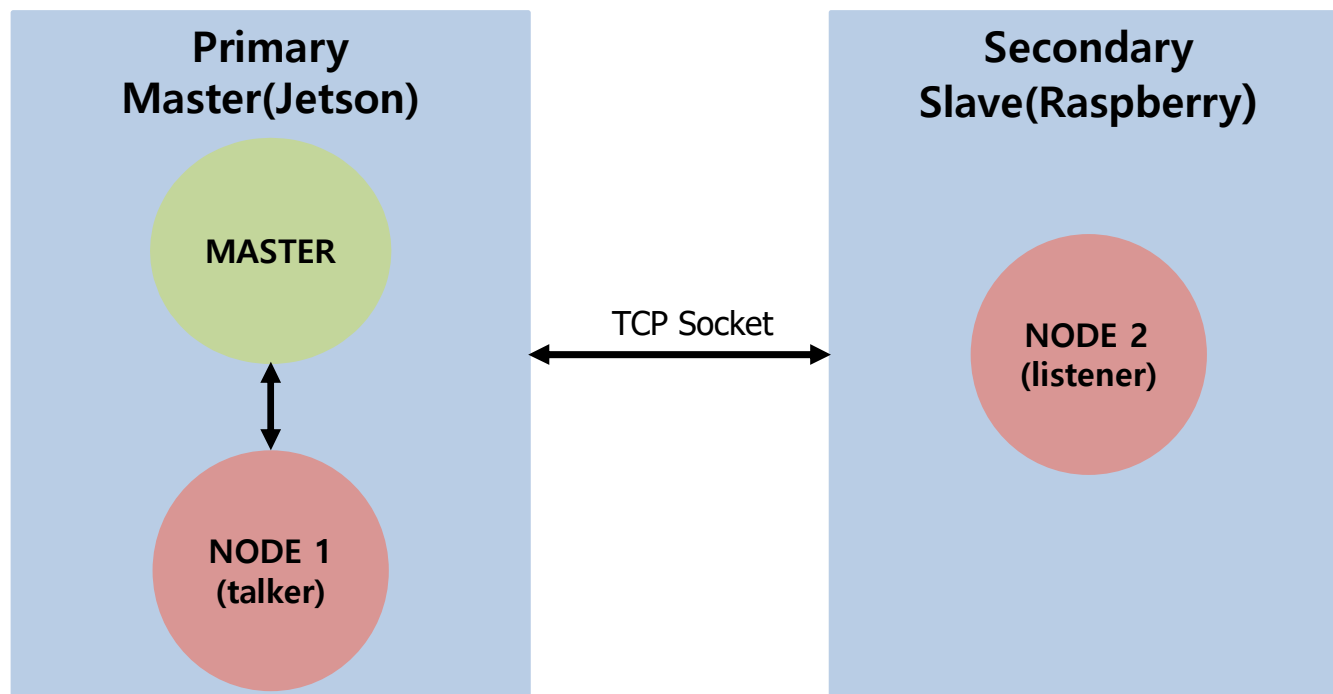
ROS를 사용하는 파이썬 프로그래밍

◆ ROS 환경에서의 파이썬 프로그래밍

- 참고 : Fundamental Robot Operating System (ROS) concepts using MuSHR, <https://mushr.io/tutorials/intro-to-ros/>



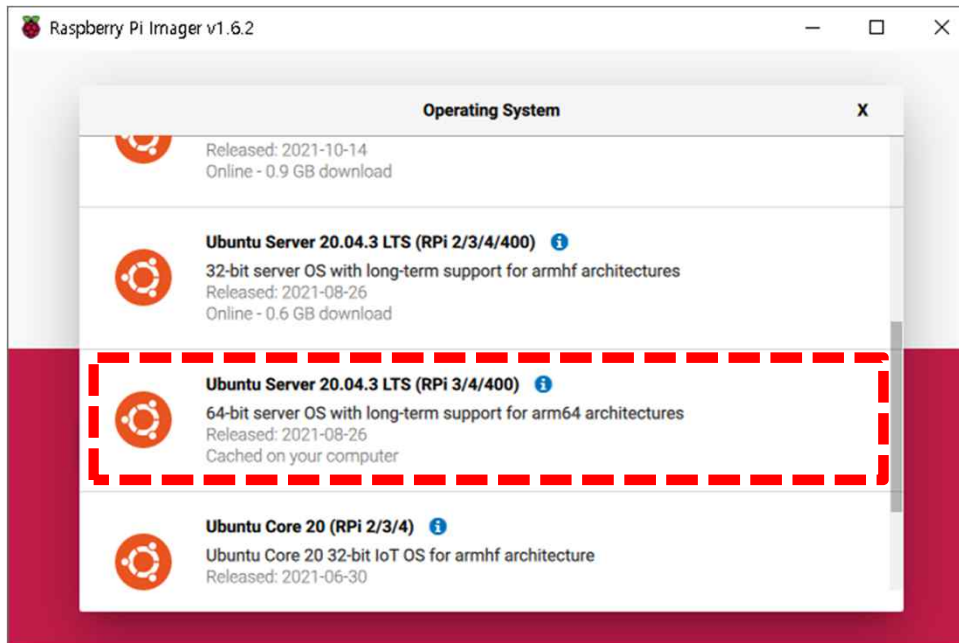
Jetson Nano와 Raspberry Pi에 설치된 ROS 노드간의 통신 (1)



Jetson Nano와 Raspberry Pi에 설치된 ROS 노드간의 통신 (2)

◆ Install Ubuntu on Raspberry

- Ubuntu Server 20.04 설치



Jetson Nano와 Raspberry Pi에 설치된 ROS 노드간의 통신 (3)

◆ Raspberry Pi 4에 ROS noetic 설치

- 참고: <http://wiki.ros.org/noetic/Installation/Ubuntu>
- 주의: **Ubuntu 20.04**는 **ROS melodic** 지원하지 않으므로 **ROS noetic**을 설치

- `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`
- `sudo apt install curl`
- `curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -`
- `sudo apt update`
- `sudo apt install ros-noetic-desktop`
- `source /opt/ros/noetic/setup.bash`
- `echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc`
- `source ~/.bashrc`
- `sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential`
- `sudo apt install python3-rosdep`
- `sudo rosdep init`
- `rosdep update`

Jetson Nano와 Raspberry Pi에 설치된 ROS 노드간의 통신 (4)

◆ Create package on ROS noetic

- Catkin_ws, Package, node 생성 과정은 melodic과 동일
- Package와 node 생성 후 CMakeList.txt 파일을 아래와 같이 수정해야 함

CMakeList.txt

```
catkin_install_python(PROGRAMS package 아래 경로명/노드명(1) package 아래 경로명/노드명(2)
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Jetson Nano와 Raspberry Pi에 설치된 ROS 노드간의 통신 (5)

◆ 기기간 연결을 위한 환경 변수 설정

- ROS_IP, ROS_MASTER_URI, ROS_HOSTNAME 3개의 환경변수를 설정해 주어야함

vi ~/.bashrc (**master**와 **slave** 각각 수정)

◆ Master

```
export ROS_IP={IP_of_master}  
export ROS_MASTER_URI=http://localhost:11311  
export ROS_HOSTNAME=$ROS_IP
```

◆ Slave

```
export ROS_MASTER_URI=http://{IP_of_master}:11311  
export ROS_HOSTNAME={IP_of_slave}
```

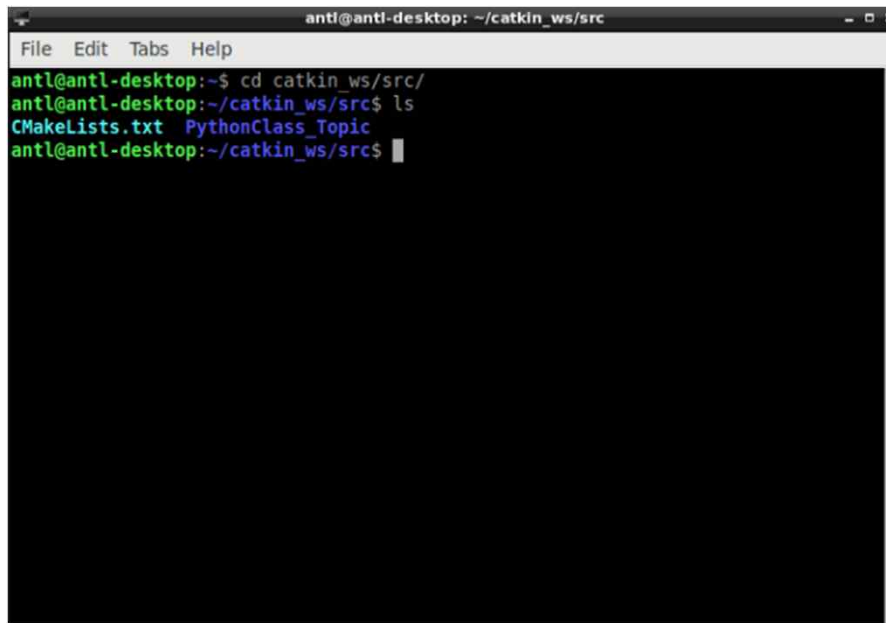
◆ Master와 Slave 각각 실행

source ~/.bashrc (**master**와 **slave** 각각 실행)

Python Programming in ROS(1)

◆ 패키지 만들기

- `cd ~/catkin_ws/src`
- `catkin_create_pkg pythonclass_topic std_msgs rospy`
- 이때 패키지 패키지명에는 대문자 사용불가



```
anti@antl-desktop: ~/catkin_ws/src
File Edit Tabs Help
anti@antl-desktop:~$ cd catkin_ws/src/
anti@antl-desktop:~/catkin_ws/src$ ls
CMakeLists.txt PythonClass_Topic
anti@antl-desktop:~/catkin_ws/src$
```

Python Programming in ROS(2)

◆ Publisher (talker) 생성

- vi talker.py
- 단, 앞서 생성한 pythonclass_topic/src 경로에 파일을 생성하여야 함
- chmod +x talker.py

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def talker():
    print("ready for talk")
    while not rospy.is_shutdown():
        talker_str = raw_input("input message:")
        print("talker_str is :", talker_str)
        pub.publish(talker_str)
        rate.sleep()

if __name__ == "__main__":
    pub = rospy.Publisher('pythonclass', String, queue_size=10)
    rospy.init_node('talker', anonymous = True, disable_signals = True)
    rate = rospy.Rate(10)
    try:
        talker()
    except KeyboardInterrupt:
        pass
```



Python Programming in ROS(3)

◆ Subscriber (listener) 생성

- vi listener.py
- **주의:** 앞서 생성한 pythonclass_topic/src 경로에 파일을 생성하여야 함
- chmod +x listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    listen_data = data.data
    print(listen_data)

def listener():
    print("ready to listen")
    rospy.Subscriber('pythonclass', String, callback)

    rospy.spin()

if __name__ == "__main__":
    rospy.init_node('listener', anonymous = True)
    listener()
```



Jetson Nano와 Raspberry Pi에 설치된 ROS 노드간의 통신 (4)

◆ 터미널 창 3개에서 실행

- roscore (master)
- rosrunk package명 node명 (master)
- rosrunk package명 node명 (slave)

```
auto-starting new master
process[master]: started with pid [4316]
ROS_MASTER_URI=http://165.229.185.182:11311/

setting /run_id to 4453b16e-7e9f-11ec-be83-48b02d2e2901
process[rosout-1]: started with pid [4331]
started core service [/rosout]
```

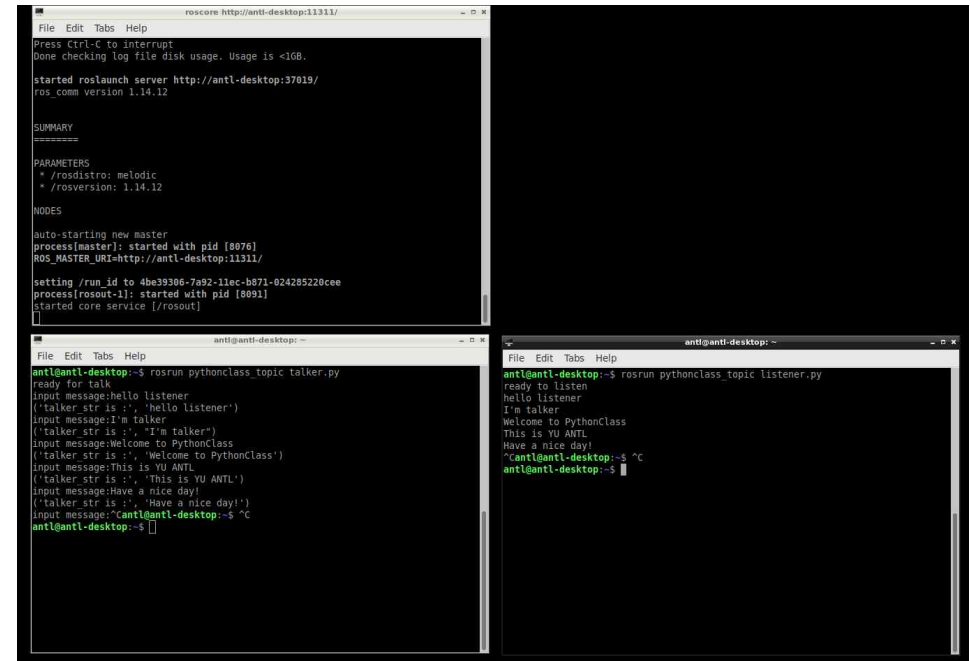
master

```
antl@antl-desktop:~$ rosrunk pythonclass_topic talker.py
ready for talk
input message:asdf
('talker_str is :', 'asdf')
```

master

```
ubuntu@ubuntu:~$ rosrunk test listener.py
ready to listen
asdf
```

slave



```
roscore http://antl-desktop:11311/
File Edit Tabs Help
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
started roslaunch server http://antl-desktop:37019/
ros_comm version 1.14.12

SUMMARY
=====
PARAMETERS
 * /roslistro: melodic
 * /rosversion: 1.14.12

NODES
auto-starting new master
process[master]: started with pid [8076]
ROS_MASTER_URI=http://antl-desktop:11311/

setting /run_id to 4be39306-7a92-11ec-b871-024285220cee
process[rosout-1]: started with pid [8091]
started core service [/rosout]

antl@antl-desktop:~$ rosrunk pythonclass_topic talker.py
ready for talk
input message:hello listener
('talker_str is :', 'hello listener')
input message:I'm talker
('talker_str is :', 'I'm talker')
input message:Welcome to PythonClass
('talker_str is :', 'Welcome to PythonClass')
input message:This is YU ANTL
('talker_str is :', 'This is YU ANTL')
input message:Have a nice day!
('talker_str is :', 'Have a nice day!')
input message:"antl@antl-desktop:~$ ^C
antl@antl-desktop:~$

antl@antl-desktop:~$ rosrunk pythonclass_topic listener.py
ready to listen
hello listener
I'm talker
Welcome to PythonClass
This is YU ANTL
Have a nice day!
"antl@antl-desktop:~$ ^C
antl@antl-desktop:~$
```

ROS 종속성과 환경 구성

◆ **rosdep** 명령은 컴파일하거나 설치하고자 하는 소스 코드의 시스템 의존성을 설치하는 것을 도와준다.

- `$ sudo rosdep init`
- `$ rosdep update`

◆ **ROS 환경구성**

- `$ source /opt/ros/melodic/setup.bash`
- 터미널을 열때마다 앞의 명령을 입력하는 것을 피하기 위해 아래 명령으로 자동으로 실행되게 등록한다.
- `$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc`

ROS 작동 테스트

◆ \$ source ~/.bashrc

◆ \$ roscore

```
iron@antl:~$ roscore
... logging to /home/iron/.ros/log/72bcb058-3da3-11ea-a509-f46d04e470e5/roslaunch-antl-32279.log
checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://antl:35531/
ros_comm version 1.14.3

SUMMARY
=====

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.3

NODES

auto-starting new master
process[master]: started with pid [32290]
ROS_MASTER_URI=http://antl:11311/

setting /run_id to 72bcb058-3da3-11ea-a509-f46d04e470e5
process[rosout-1]: started with pid [32301]
started core service [/rosout]
```

◆ roscore를 처리하는 동안 다른 것을 실행할 수 없으므로 새 터미널을 연다.



roinstall 얻기

◆ \$ roscore list

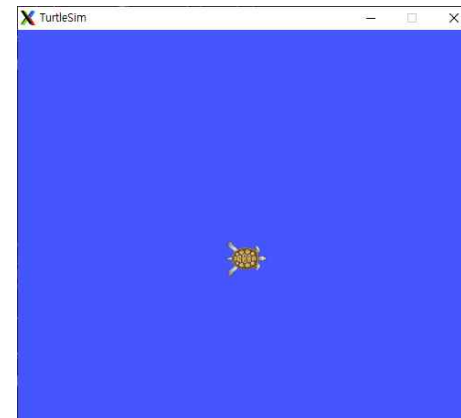
```
iron@antl:~$ roscore list  
/roscout
```

◆ 다른 패키지를 설치하는데 도움을 주는 명령 툴 설치

- \$ sudo apt-get install python-roinstall

◆ Roscore와 turtlesim이 모두 동작하는지 확인

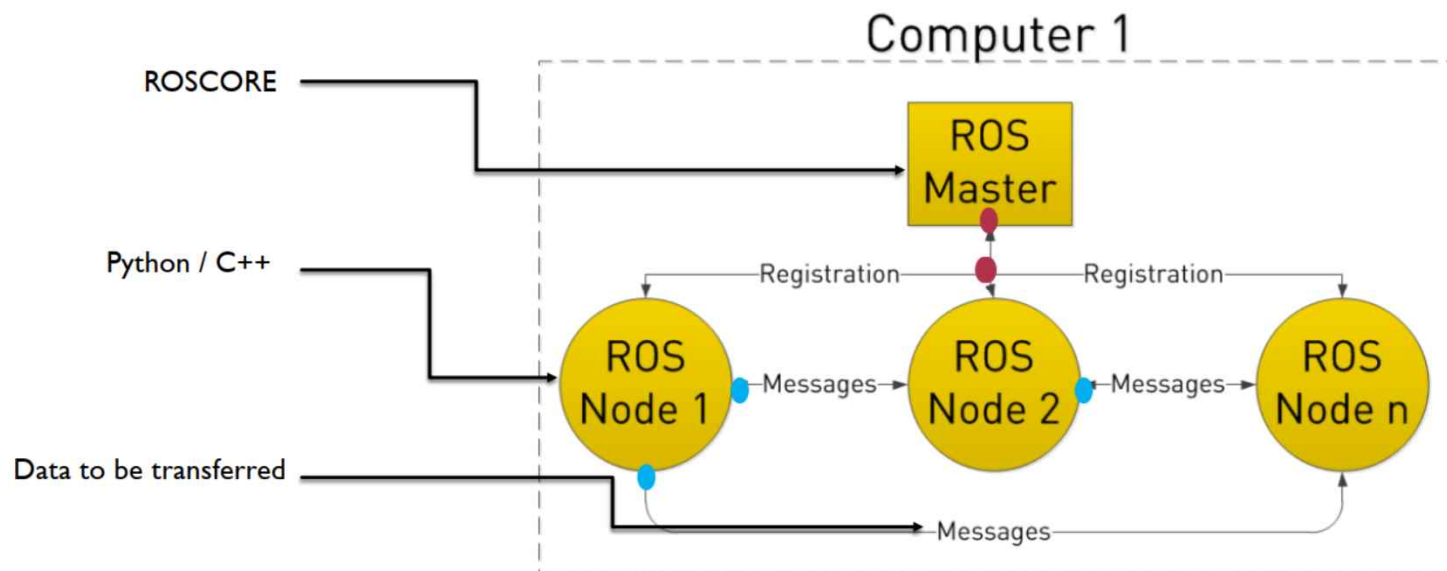
- \$ roscore
- \$ rosrunc turtlesim turtlesim_node
- 잘 동작한다면 옆의 화면이 뜨는 것을 확인할 수 있다.



ROS구조와 동작

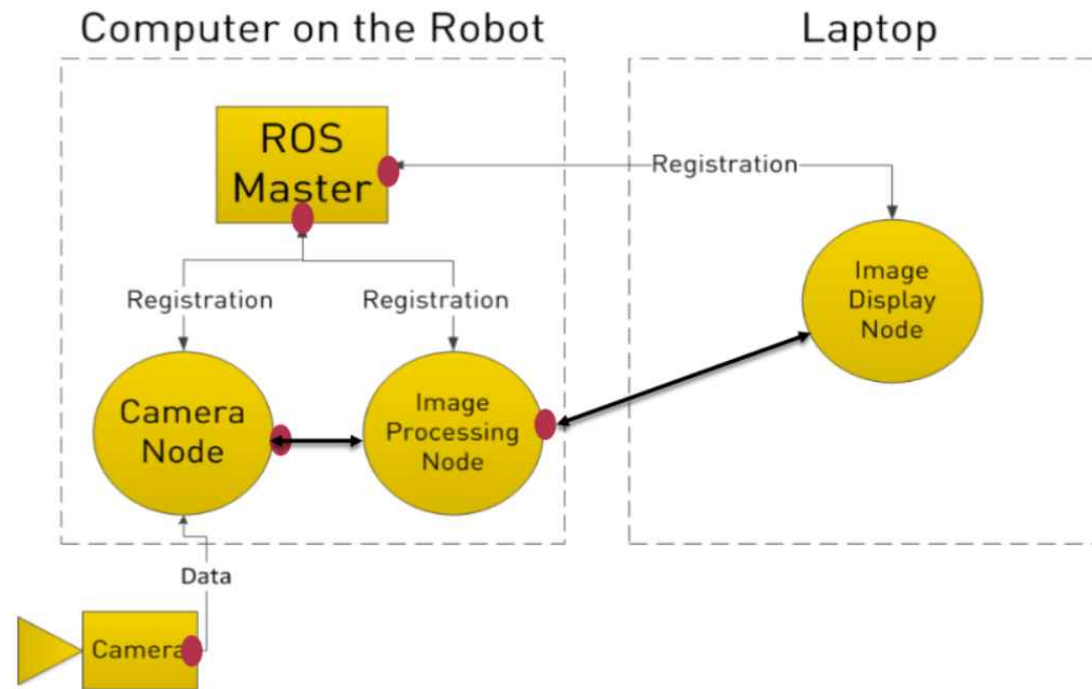
ROS 기본 개념

◆ ROS Node간 연결 및 상호 작용

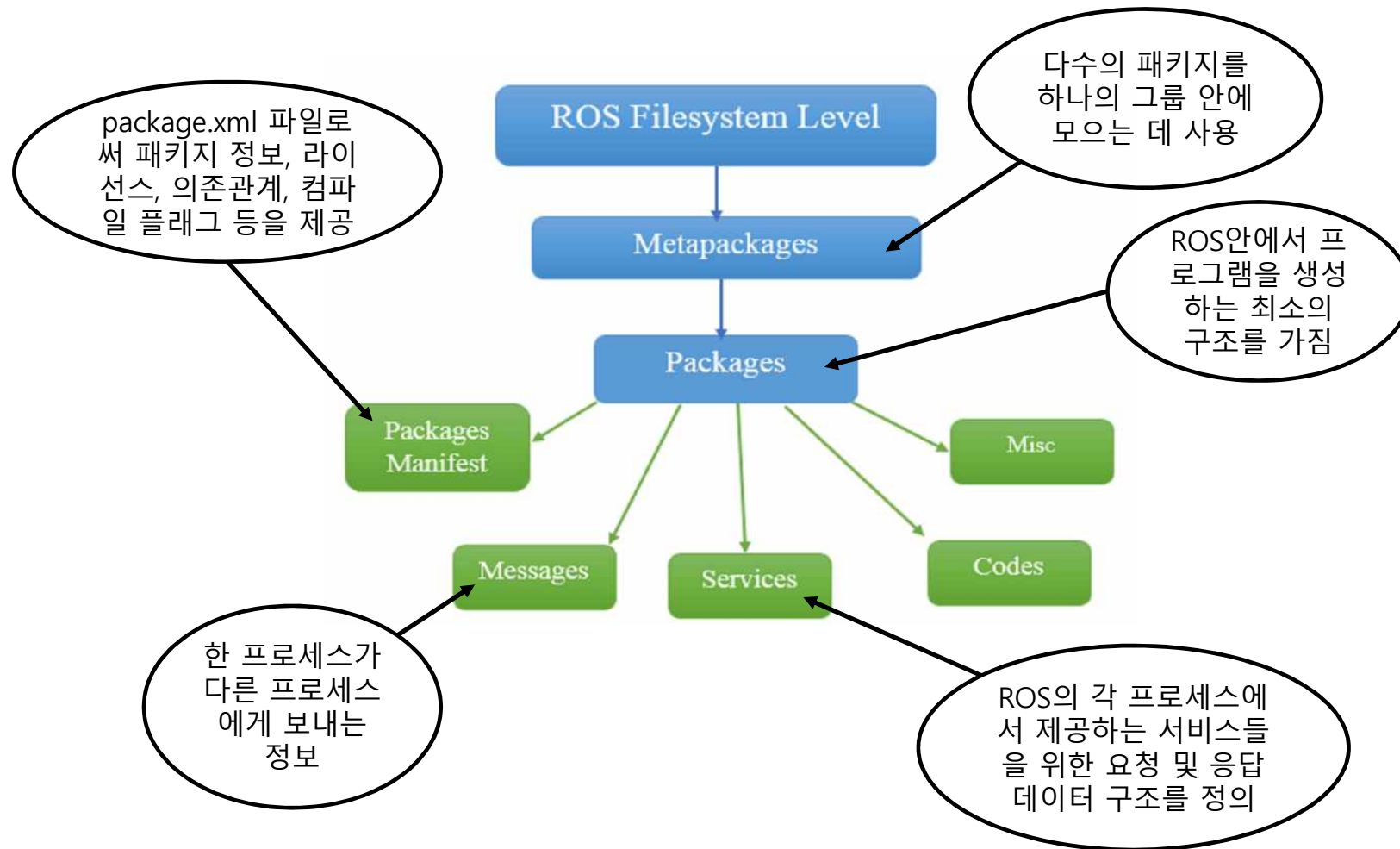


ROS 기본 개념

◆ 서로 다른 컴퓨터 상의 ROS 노드들 간의 접속



ROS 파일 시스템 레벨의 이해



Master, Node, Package, Meta Package, Message

◆ Master

- 노드와 노드 사이의 연결 및 메시지 통신을 위한 네임 서버이다.
- 마스터가 없으면 ROS 노드간 메시지, 토픽 등의 통신을 할 수 없다.
- 마스터는 Slave들과의 접속 상태를 유지하지 않는 HTTP기반의 프로토콜은 XMLRPC를 이용하여 Slave들과 통신한다.

◆ Node

- 최소 단위의 실행 가능한 프로세서
- 하나의 실행 가능한 프로그램으로 생각하면 된다.
- 각 노드는 메시지 통신으로 데이터를 주고 받는다.

◆ Package

- 하나 이상의 노드 또는 노드 실행을 위한 정보 등을 묶어 놓은 것.

◆ Meta Package

- 다수의 패키지를 하나의 그룹 안에 모으는 데 사용하는 것.

◆ Message

- 메시지를 통해 노드 간의 데이터를 주고 받는다.
- 메시지는 integer, floating, point, boolean과 같은 변수형태이다.



ROS의 구조

◆ ROS의 세 가지 레벨

Level	설 명
파일시스템 레벨	▪ ROS의 내부 구성, 폴더 구조, ROS가 동작하기 위해 필요한 최소한의 파일을 설명하기 위하여 한 그룹의 개념들이 사용됨
연산 그래프 레벨	▪ 프로세스와 시스템 간에 통신하는 곳으로 시스템을 구성하고 모든 프로세스를 관리하며 하나 이상의 컴퓨터에서 통신을 하는 등 모든 개념과 시스템을 볼 수 있음
커뮤니티 레벨	▪ 모든 개발자들이 자신의 지식과 알고리즘 그리고 코드를 서로 공유하게 하는 도구와 개념들이 있음

ROS 파일 시스템 레벨의 이해

◆ Example turtlesim package

- Turtlesim 패키지의 tree를 확인하기 위하여 tree 명령어 설치
- \$ sudo apt-get install tree
- 해당하는 디렉토리에서 tree 명령어를 실행하면
옆의 화면과 같이 하위 파일 들을 보여 줌.

메세지(msg) 타입

패키지 매니페스트

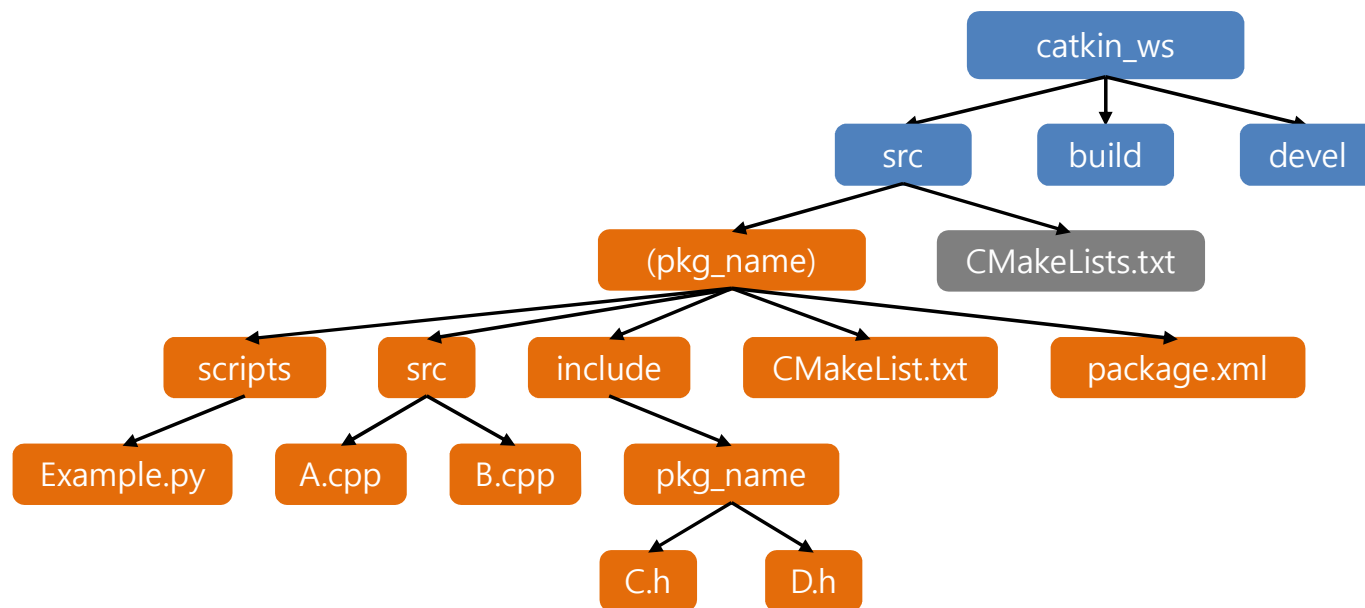
서비스(srv) 타입

```
iron@antl:/opt/ros/melodic/share/turtlesim$ tree
.
├── cmake
│   ├── turtlesimConfig.cmake
│   ├── turtlesimConfig-version.cmake
│   ├── turtlesim-msg-extras.cmake
│   └── turtlesim-msg-paths.cmake
├── images
│   ├── box-turtle.png
│   ├── diamondback.png
│   ├── electric.png
│   ├── fuerte.png
│   ├── groovy.png
│   ├── hydro.png
│   ├── hydro.svg
│   ├── indigo.png
│   ├── indigo.svg
│   ├── jade.png
│   ├── kinetic.png
│   ├── kinetic.svg
│   ├── lunar.png
│   ├── lunar.svg
│   ├── melodic.png
│   ├── palette.png
│   ├── robot-turtle.png
│   ├── sea-turtle.png
│   └── turtle.png
├── msg
│   ├── Color.msg
│   ├── Pose.msg
│   └── package.xml
└── srv
    ├── Kill.srv
    ├── SetPen.srv
    ├── Spawn.srv
    ├── TeleportAbsolute.srv
    └── TeleportRelative.srv
```

ROS Package 구조

◆ Package의 구조

- scripts/: Bash, Python 또는 다른 스크립트 언어에 있는 실행가능 스크립트들이다.
- src/: 우리가 작성한 소스 파일들이 있는 곳.
- include/(package_name)/: 필요로 하는 라이브러리의 헤더들을 포함
- CMakeLists.txt: Cmake빌드 파일
- package.xml: 패키지 매니페스트



ch 21 - 66

ROS Package.xml

◆ package.xml

- 패키지의 정보를 규정, 해당 패키지의 이름, 의존관계 등의 정보들을 보여준다.
이 모든 것이 패키지들의 설치와 배포를 쉽게 한다.
- package.xml에 사용되는 전형적인 두 개의 태그: <build_depend>와 <run_depend>
- <build_depend>
이 태그는 현재 패키지를 설치하기 전에 어떤 패키지가 설치되어야 하는지를 보여준다.
- <run_depend>
이 태그는 패키지의 코드를 실행하는데 필요한 패키지들을 보여준다.



ROS Package 생성 및 수정 명령어

◆ Package를 생성하고 수정하기 위한 명령어

- rospack: 시스템에서 패키지의 정보를 얻거나 패키지를 검색
- catkin_create_pkg: 새로운 패키지를 생성
- catkin_make: 작업공간을 컴파일하는 명령어
- rosdep: 패키지의 시스템 의존 관계 설치
- rqt_dep: 패키지의 의존관계들을 그래프로 보여주는 명령어

ROS Package 이동 명령어 제공 패키지 - rosbash

◆ 패키지와 폴더 및 파일 간에 쉽게 이동할 수 있도록 하는 명령어를 제공하는 **rosbash**라는 패키지를 제공한다.

- rosbash package 경로: /opt/ros/melodic/share/rosbash
- roscd: 디렉터리를 변경할 때 사용, 리눅스의 cd와 같은 역할
- rosed: 파일을 편집할 때 사용
- roscp: 패키지로 부터 파일을 복사할 때 사용
- rosd: 패키지의 디렉터리들을 나열
- rosls: 패키지의 파일들을 나열, 리눅스의 ls와 같은 역할

→ 대체적으로 리눅스 명령어와 매우 유사하다.



MetaPackage

- ◆ **Meta-package**는 오직 하나의 파일만 들어 있는 특별한 패키지 이다.
오직 **package.xml**파일만 들어 있다.

```
iron@antl:/opt/ros/melodic/share/ros_core$ ls  
package.xml
```

- 일반적으로 특징적 기능을 따라 그룹화된 다른 패키지들을 포함한다.

- ◆ **Meta-package 관련 명령어**

- \$ rosstack find (metapkg_name)

```
iron@antl:~$ rosstack find ros_core  
/opt/ros/melodic/share/ros_core
```

- \$ vim /opt/ros/melodic/share/(metapkg_name)/package.xml

```
<?xml version="1.0"?>  
<?xml-model  
  href="http://download.ros.org/schema/package_format2.xsd"  
  schematypens="http://www.w3.org/2001/XMLSchema"?>  
<package format="2">  
  <name>ros_core</name>  
  <version>1.4.1</version>  
  <description>A metapackage to aggregate the packages required to use publish / subscribe, services, launch files, and other core ROS concepts.</description>  
  <maintainer email="mikael@osrfoundation.org">Mikael Arguedas</maintainer>  
  <license>BSD</license>
```

ROS Communication Layer: ROS Core, ROS Master

◆ ROS Master

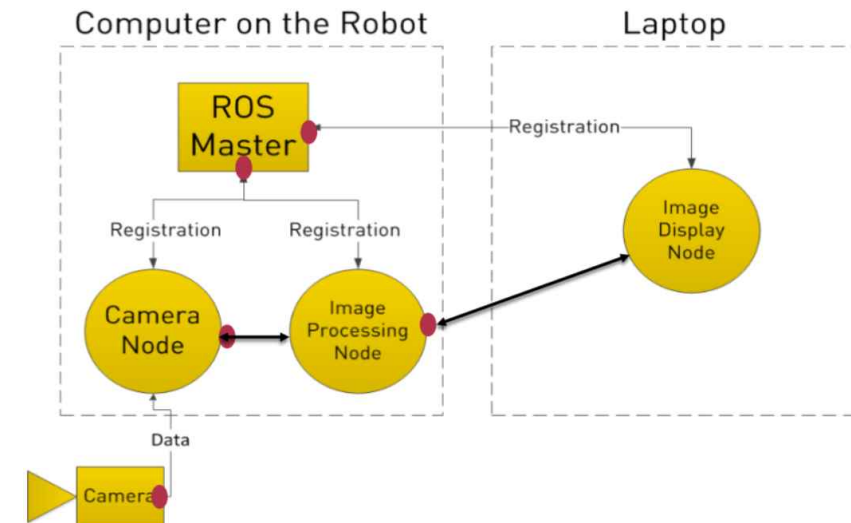
- XML과 RPC 에 기반한 중앙 집중식 서버 (central server)
- 통신 연결을 중재
- ROS 그래프 리소스에 대한 이름 등록과 검색

◆ Parameter Server

- 영구 구성 매개 변수와 기타 임의 데이터 저장

◆ 'rosout'

- 사람이 읽을 수 있는 메시지를 위한 네트워크 기반의 "stdout"



ROS Communication Layer: Graph Resources

◆ Node

- 네트워크에 분산되어 있는 프로세스
- 네트워크를 통해 전송된 데이터의 Source와 Sink 역할 수행

◆ Parameters

- 구성 및 초기화 설정과 같은 영구 데이터로 parameter server에 저장된 데이터 (e.g., camera configuration)

◆ Topics

- 비동기식 다-대-다 통신 (asynchronous many-to-many communication) 흐름

◆ Services

- 동기식 일-대-다 네트워크 (synchronous one-to-many network, multicasting) 기반 서비스



ROS Communication Protocols: Node 연결

◆ ROS Topics

- 비동기식 stream 방식 통신
- ROS.msg 사양의 타입
- 하나 이상의 발행자를 가질 수 있다
- 하나 이상의 구독자를 가질 수 있다

◆ ROS Services

- 동기식 기능 호출 방식의 통신
- ROS.srv 사양의 타입
- 하나의 서버만 가질 수 있다
- 하나 이상의 클라이언트를 가질 수 있다

◆ Actions

- Topics 상위에 빌드 됨
- 긴 시간 실행되는 프로세스
- 취소 기능



msg(Message)

◆ 각 노드들은 메시지를 통해 노드 간 데이터를 주고받는다.

- 메시지는 integer, floating, point, boolean과 같은 변수 형태이다.
- 간단한 데이터 구조나 배열과 같은 구조도 사용할 수 있다.

기본 타입	직렬화(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	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	std::string	string
time	secs/nsecs signed 32-bit ints	<u>ros::Time</u>	<u>rospy.Time</u>
duration	secs/nsecs signed 32-bit ints	<u>ros::Duration</u>	<u>rospy.Duration</u>

msg(Message)

◆ Header Type

- 시간과 프레임, 시퀀스 번호를 추가하는 데 사용된다.
- 메시지에 숫자를 넣고, 누가 메시지를 보내는지 볼 수 있게 하고, 사용자가 볼 수 있는 많은 함수들을 사용할 수 있게 한다.
- 이 명령어로 헤더의 구조를 알 수 있다.
\$ rosmmsg show std_msgs/Header

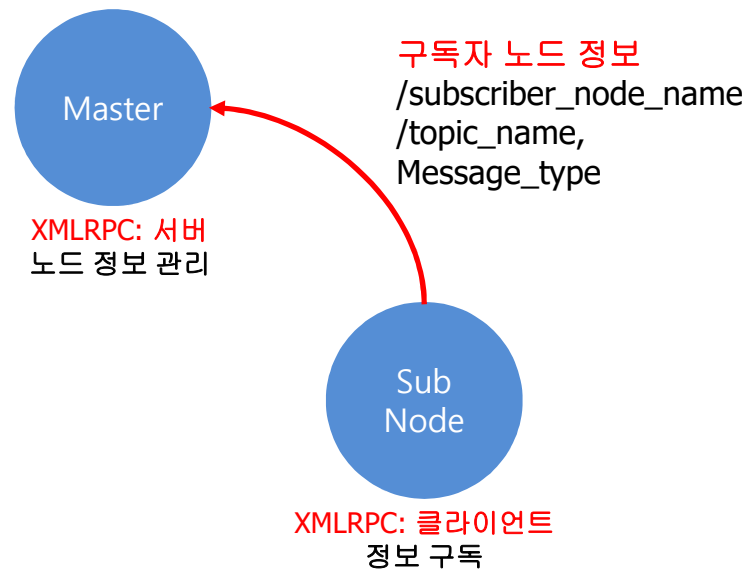
```
iron@antl:~/catkin_ws/src/chapter2_tutorials$ rosmmsg show std_msgs/Header
uint32 seq
time stamp
string frame_id
```

- rosmmsg툴은 메시지 정의에 대한 정보를 출력하고 특정 메시지 타입을 사용하는 소스 파일을 찾아준다.
- 자세한 생성 방법에 대해서는 뒤에서 다루도록 한다.

msg(Message)

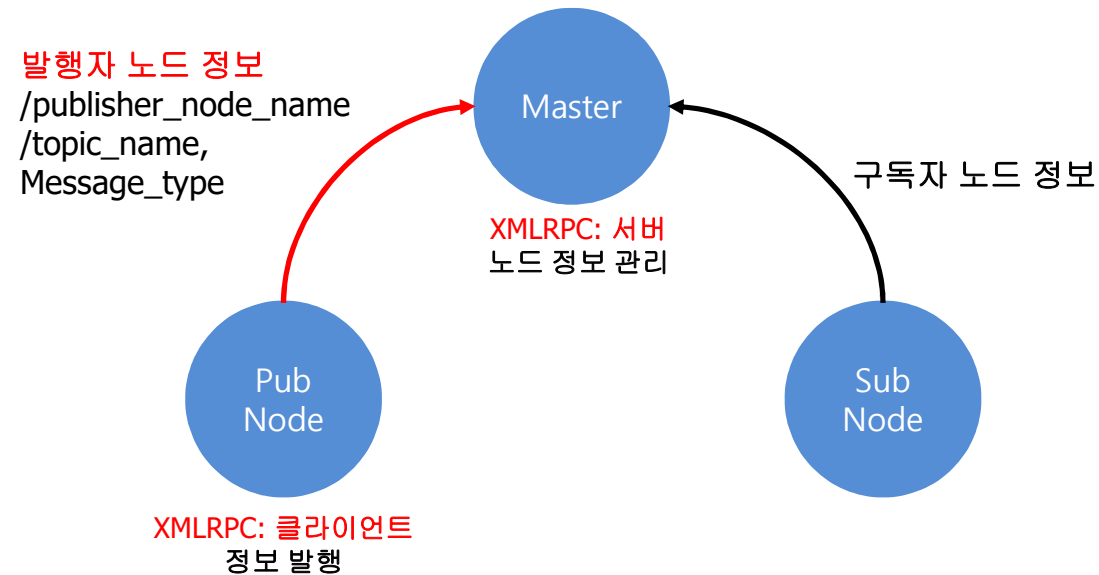
◆ 메시지 통신 과정

1. 마스터 구동: XMLRPC(XML-Remote Procedure Call)
\$ roscore
2. 구독자(Subscriber) 노드 구동
\$ rosrn (sub_pkgnode_name)



msg(Message)

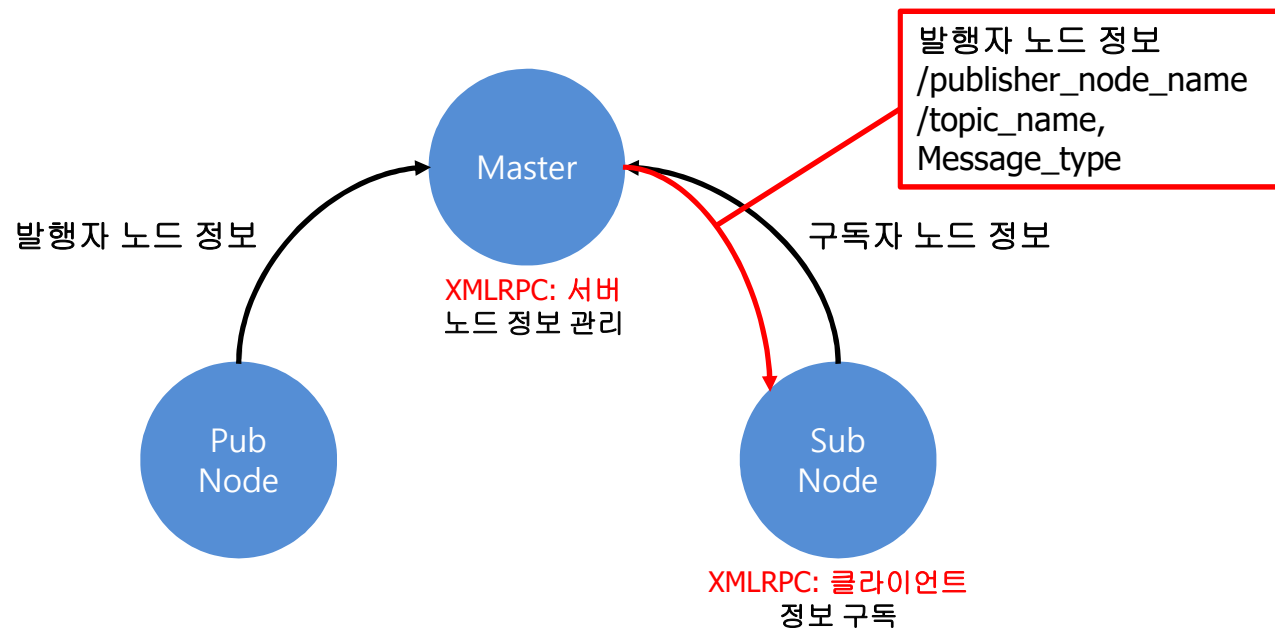
3. 발행(Publisher) 노드 구동
\$ rosrn (pub_pkgnode_name)



msg(Message)

4. 발행자 정보 알림

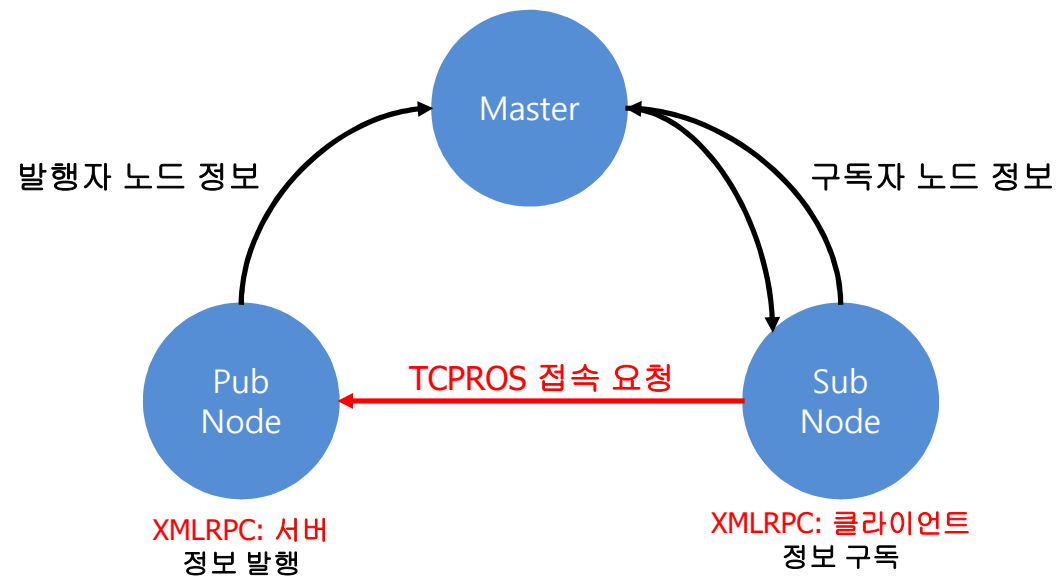
- 마스터는 구독자 노드에게 새 발행자 정보를 알린다.



msg(Message)

5. 발행자 노드에 접속 요청

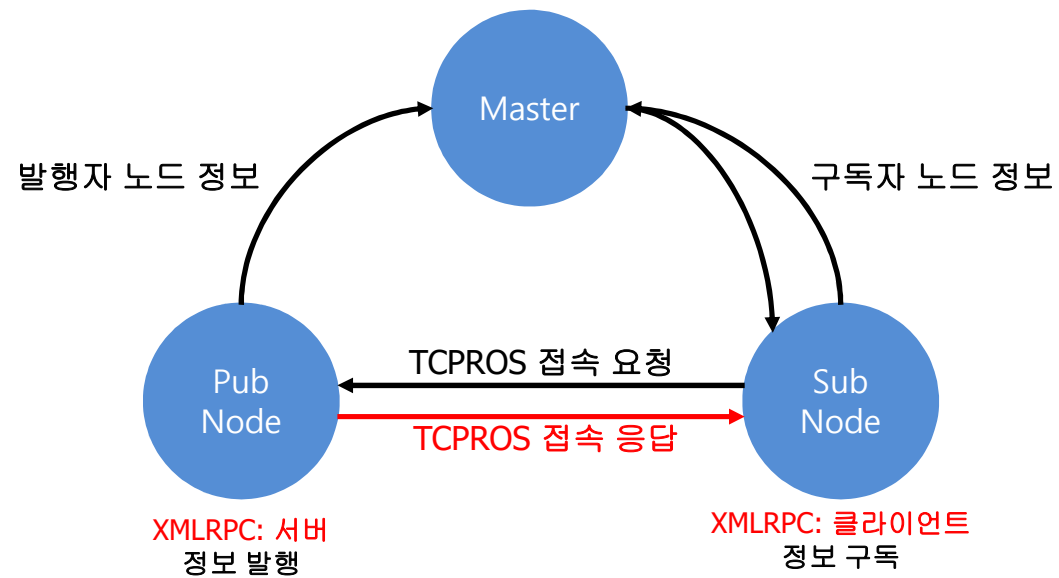
- 마스터로부터 받은 발행자 정보를 이용해 TCPROS 접속을 요청



msg(Message)

6. 구독자 노드에 접속 응답

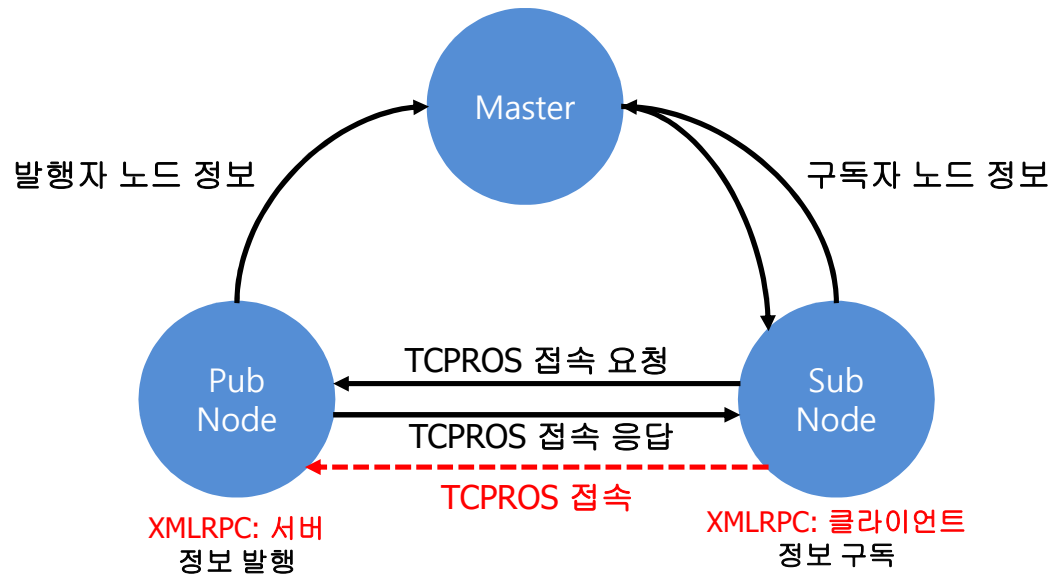
- 접속 응답에 해당되는 자신의 TCP URL 주소와 포트번호 전송



msg(Message)

7. TCP 접속

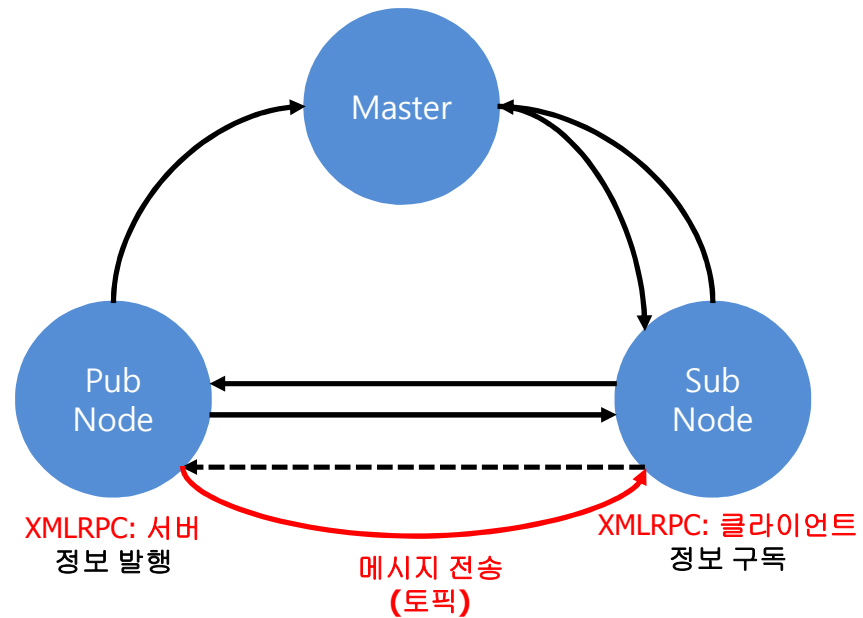
- TCPROS를 이용하여 발행자 노드와 직접 연결한다.



msg(Message)

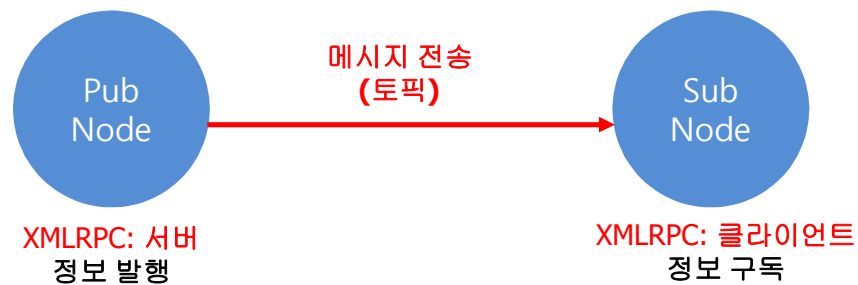
8. 메시지 전송

- 발행자 노드는 구독자 노드에게 메시지를 전송(토픽)



msg(Message)

- 토픽방식에서는 접속을 끊지 않는 이상 지속적으로 메시지를 전송 (연속성)

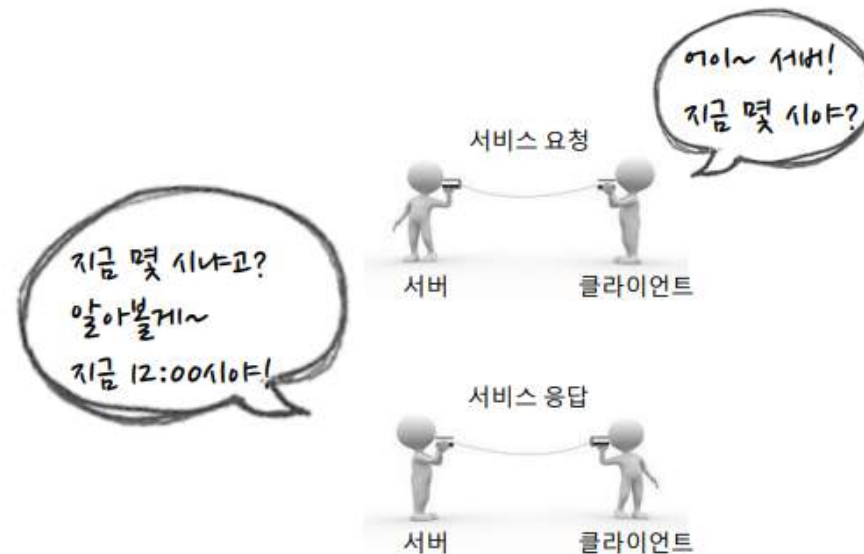


ROS srv(Service)

◆ ROS service (srv)

- ROS에서 서비스란 1:1 또는 1:N 노드 간에 메시지를 주고받는 양방향 통신으로 데이터를 요청하는 Client와 이를 수행하는 Server로 구성된다.

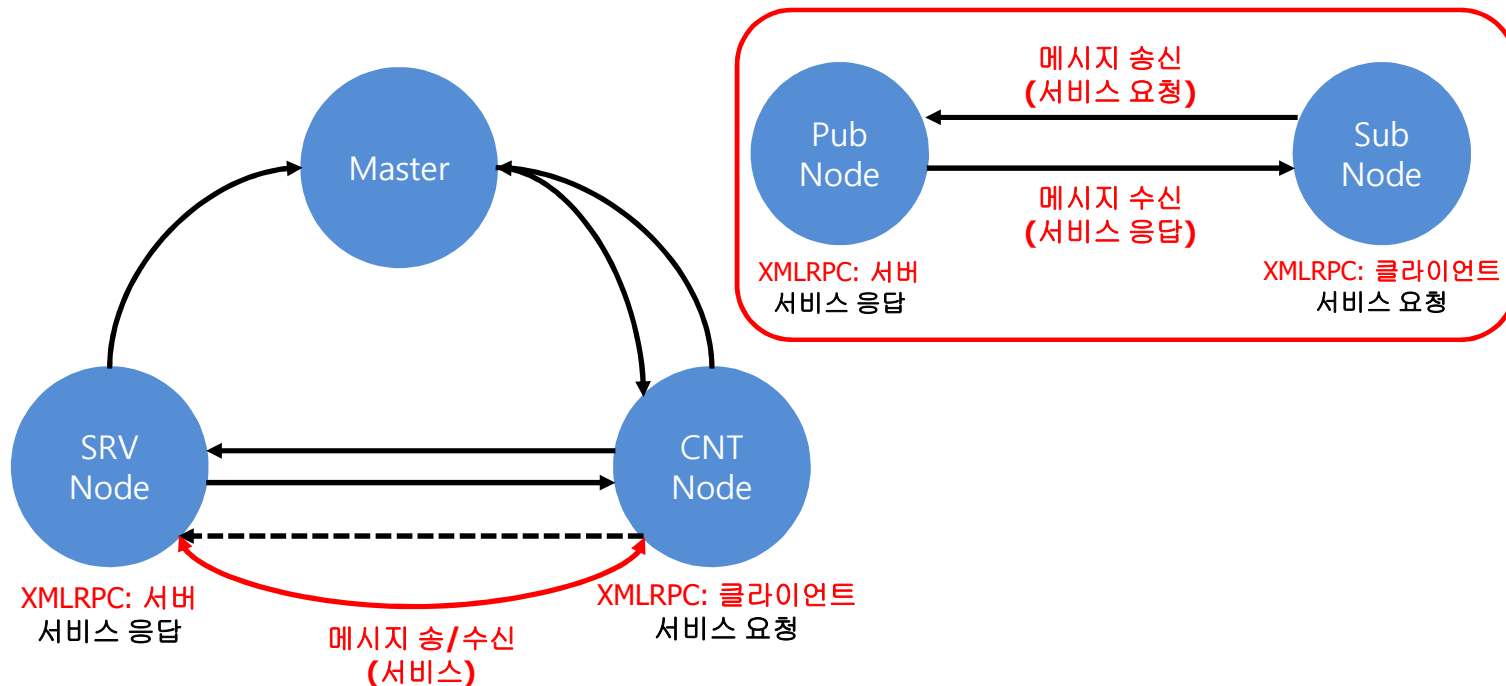
Service, Service server, Service client



Reference: <https://www.slideshare.net/yoonseokpyo/20160406-ros-1-for>

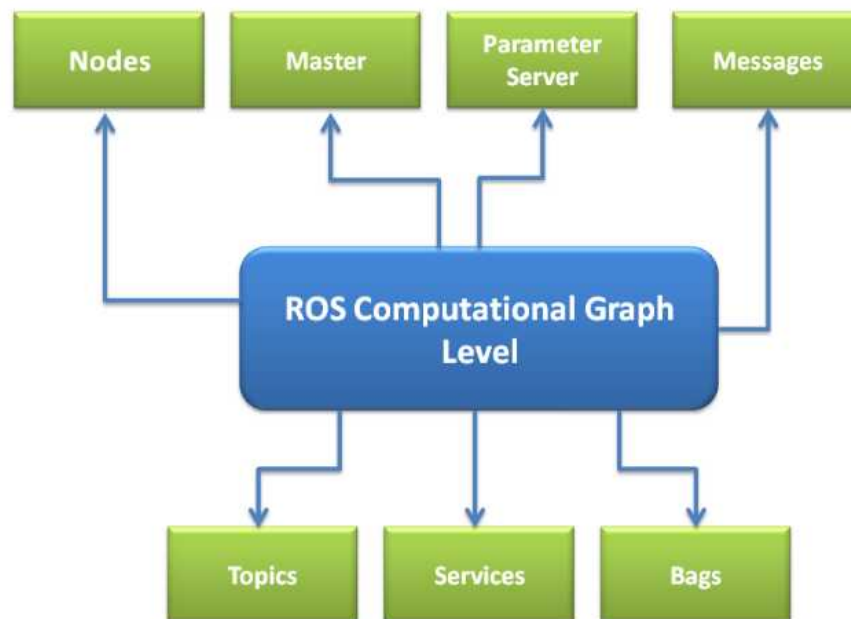
ROS srv(Service)

- ◆ 서비스는 토픽과 달리 1회 한해 접속하고 요청과 응답이 끝나면 서로 간의 접속을 종료 (1회성)



ROS Computational Graph Level (연산 그래프 레벨)

- ◆ ROS는 모든 프로세스들이 연결되는 네트워크를 생성한다.
그림에서 보듯이 노드들이 서로 상호작용한다.
 - ◆ 이 레벨의 기본적인 개념은 노드, 마스터, 파라미터 서버, 메시지, 서비스, 토픽 그리고 bag, 이들은 각기 다른 방식으로 그래프에 데이터를 제공한다.
- 파라미터 서버: 패키지에서 매
개변수를 사용할 때 각 매개변
수를 등록하는 서버를 말한다.
 - Bag: ROS 메시지 데이터를 저
장하고 재생하기 위한 포맷이다.



ROS Community Level (커뮤니티 레벨)

◆ 별개의 커뮤니티들이 소프트웨어와 지식을 교환할 수 있게 하는 ROS 자원.

- **배포판:** 사용자가 설치할 수 있는, 버전이 있는 메타패키지의 집합
- **저장소:** ROS는 코드 저장소의 연합된 네트워크에 의존하는데, 여기에서 서로 다른 기관들이 그들만의 로봇 소프트웨어 구성요소들을 개발하고 공개할 수 있다.
- **ROS Wiki:** ROS에 대한 정보를 작성하는 주요 포럼, 누구나 계정을 만들고 작성할 수 있다.
- **버그 티켓 시스템:** 문제를 발견했거나 새로운 특징을 제안하기 원한다면, ROS는 이를 위한 자원으로 버그 티켓 시스템이 있다.
- **메일링 리스트:** ROS에 관한 질문을 하는 포럼일 뿐만 아니라 ROS의 새 업데이트에 관한 기본적인 통신 채널이다.
- **ROS 대답:** 사용자는 이 자원을 이용해서 포럼에서 질문할 수 있다.
- **블로그:** <http://www.ros.org/news>

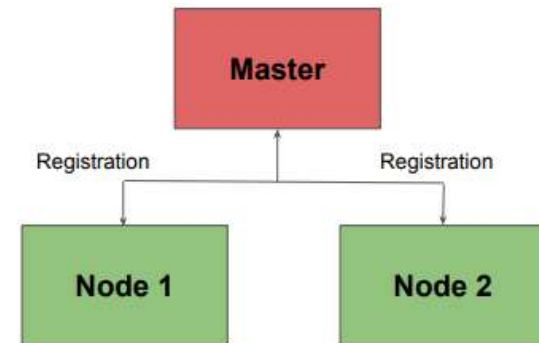
ROS Master

- ◆ Note들 간의 통신을 관리
- ◆ 모든 node는 시작 시에 Master에 등록
- ◆ Master 아래 명령으로 시작
 - \$ roscore



ROS Nodes

- ◆ 한 가지 목적을 가지는 실행 가능한 프로그램
- ◆ 개별적으로 컴파일 되고, 실행되며, 관리된다
- ◆ 패키지로 구성
- ◆ 아래 명령으로 **Node** 실행
 - `$ rosrun <패키지 이름> <노드 이름>`
- ◆ 활성화된 노드는 다음 명령으로 확인
 - `$ rostopic list`



ROS Topics

◆ Node들은 topics를 통해 메시지 통신

- Node는 topic을 구독하거나 발행할 수 있다
- 일반적으로 게시(발행)자 1명과 구독자는 n명

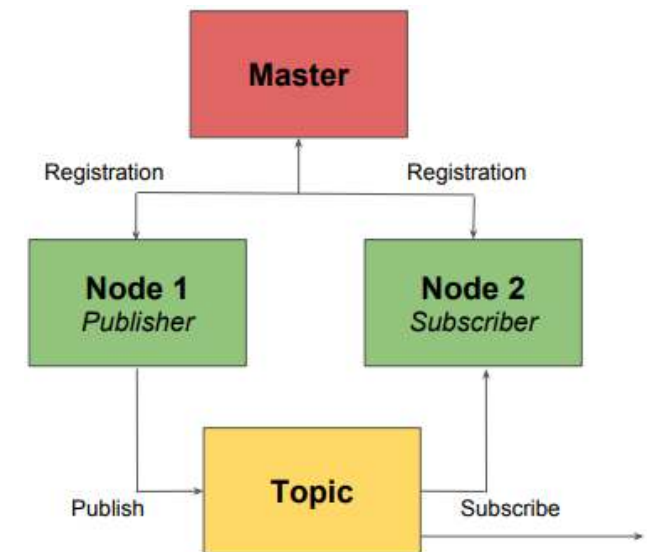
◆ Topic은 메시지 스트림의 이름

◆ 활성화된 topic을 아래 명령으로 리스트

- \$ rostopic list

◆ 아래 명령으로 발행되는 topic 메시지를 출력

- \$ rostopic echo /topic



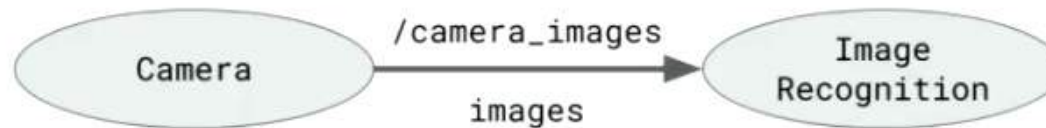
Topics

◆ ROS Topics

ROS Topics

ROS nodes communicate with each other over **topics**

- If you want to send messages, you **publish** to a topic
- If you want to receive messages, you **subscribe** to a topic



Topic, Publisher, Subscriber

◆ Topic

- 메시지의 이름, 주제를 나타냄



◆ Publish & Publisher(발행 & 발행자)

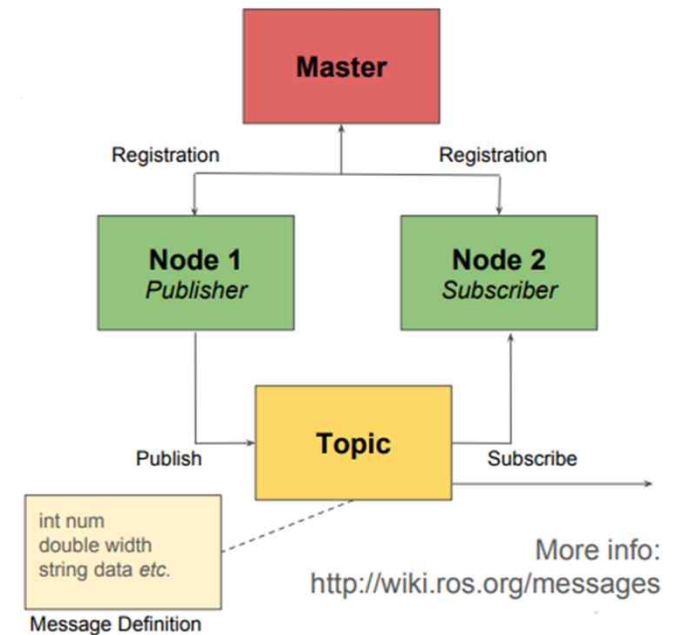
- 토픽의 내용에 해당하는 메시지 형태의 데이터를 송신하는 것을 의미한다. 발행자 노드는 발행을 수행하기 위해 토픽을 포함한 자신의 정보를 마스터에 등록하고 구독을 원하는 노드에게 메시지를 보낸다. 발행자는 복수개로 등록이 가능하다.

◆ Subscribe & Subscriber(구독 & 구독자)

- 토픽의 내용에 해당되는 메시지 형태의 데이터를 수신하는 것을 말한다. 구독자 노드는 토픽을 포함한 자신의 정보들을 마스터에 등록하고, 구독하고자 하는 토픽을 발행하는 발행자 노드의 정보를 마스터로 부터 받는다. 구독자는 복수개로 등록 가능하다.

ROS Message

- ◆ Topic의 타입을 정의하는 데이터 구조
- ◆ 정수, 실수, 불리언, 문자열 등의 중첩 구조와 객체 배열로 구성
- ◆ *.msg 파일에 정의
- ◆ Topic의 타입은 아래 명령으로 볼 수 있다
 - \$ rostopic type /topic
- ◆ Topic으로 메시지 발행
 - \$ rostopic pub /topic type args



ROS Messages

◆ ROS Messages

geometry_msgs/Point.msg

```
float64 x  
float64 y  
float64 z
```

sensor_msgs/Image.msg

```
std_msgs/Header header  
uint32 seq  
time stamp  
string frame_id  
uint32 height  
uint32 width  
string encoding  
uint8 is_bigendian  
uint32 step  
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header  
uint32 seq  
time stamp  
string frame_id  
geometry_msgs/Pose pose  
→ geometry_msgs/Point position  
float64 x  
float64 y  
float64 z  
geometry_msgs/Quaternion  
orientation  
float64 x  
float64 y  
float64 z  
float64 w
```

ROS Service

◆ 서비스를 통해 Node 간의 요청/응답 통신 구현

- 서비스 서버가 서비스를 공고하면
- 서비스 클라이언트가 이 서비스에 액세스

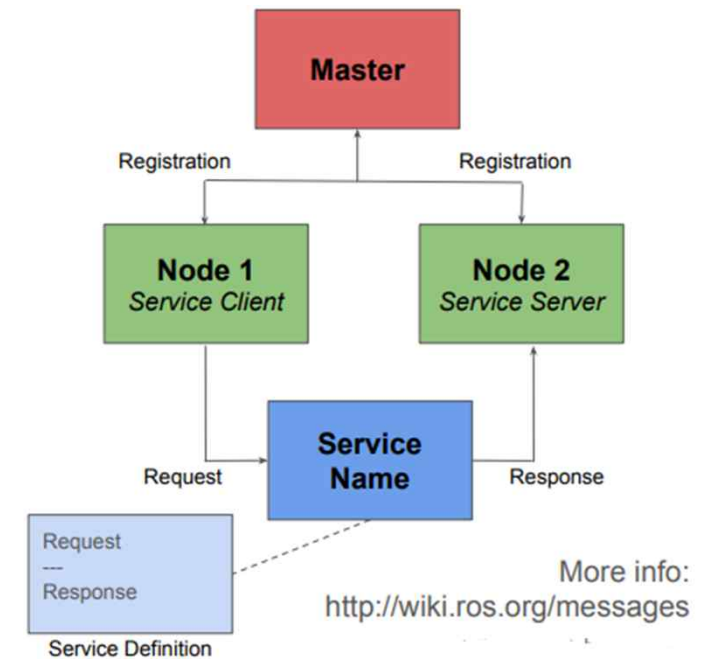
◆ 메시지와 유사한 구조로 서비스는 *.srv 파일에 정의

◆ 아래 명령으로 이용할 수 있는 서비스를 리스트

- `$ rosservice list`

◆ 서비스 타입은 아래 명령으로 확인

- `$ rosservice type /service_name`



Service

◆ Service

- 메시지는 비동기 방식이다. 하지만 요청과 응답이 함께 사용되는 동기 방식의 메시지 교환 방식도 필요하다. 이에 따라, ROS에는 서비스라는 이름으로 메시지 동기 방식을 제공하고 있다.

◆ Service Server

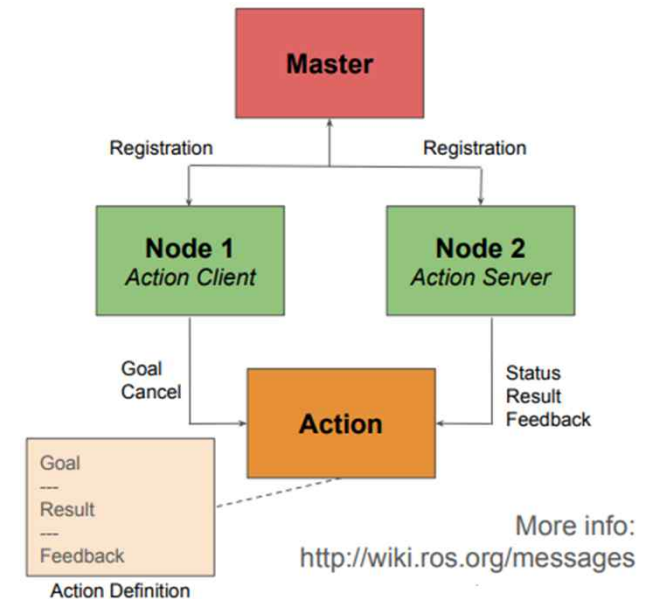
- 서비스 서버는 요청을 입력으로 받고, 응답을 출력으로 하는 서비스 메시지 통신의 서버 역할을 말한다. 요청과 응답은 모두 메시지로 되어 있으며, 서비스 요청에 의하여 주어진 서비스를 수행 후에 그 결과를 서비스 클라이언트에 전달한다.

◆ Service Client

- 서비스 클라이언트는 요청을 출력으로 하고, 응답을 입력으로 받는 서비스 메시지 통신의 클라이언트 역할을 말한다. 요청과 응답은 모두 메시지로 되어 있으며, 서비스 요청을 서비스 서버에 전달하고 그 결과 값을 서비스 서버로부터 받는다.

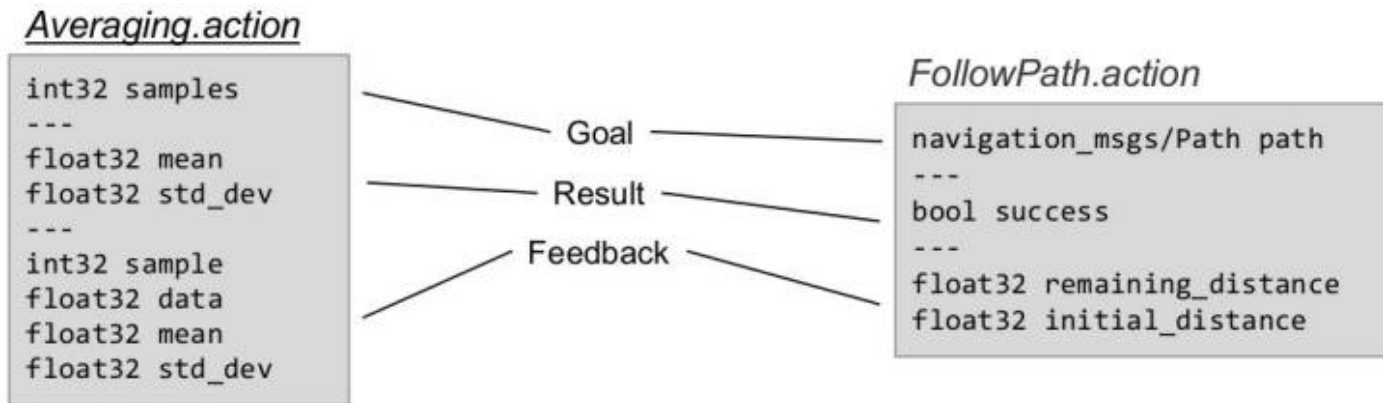
ROS Action

- ◆ 서비스 호출과 유사하지만 다음에 대한 가능성을 제공한다
 - 작업 취소 (preempt)
 - 진행사항에 대한 피드백 수신
- ◆ 시간적으로 확장되고 목적 지향적인 행동에 대한 인터페이스 구현에 최상의 방법
- ◆ 서비스 구조와 유사하며, *.action 파일에 정의
- ◆ 내부적으로는 topic의 집합으로 구현된다



ROS Action

◆ ROS Action



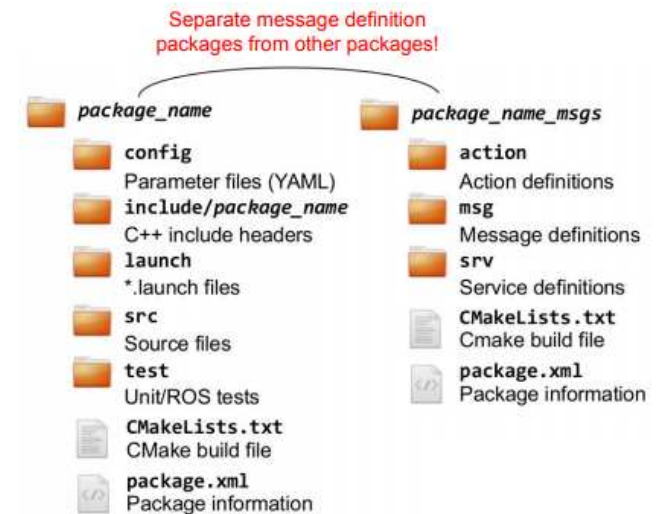
ROS Bags

- ◆ Bag은 메시지 데이터를 저장하는 형식
- ◆ *.bag 확장자를 가지는 바이너리 포맷
- ◆ 시각화와 분석을 위해 데이터셋을 로깅하고 기록하는데 적합
- ◆ Bag에 모든 topics를 기록
 - \$ rosbag record --all
- ◆ 주어진 topics를 기록
 - \$ rosbag record topic_1 topic_2 topic_3
- ◆ Bag에 대한 정보 표시
 - \$ rosbag info bag_name.bag
- ◆ 주어진 topics를 기록
 - \$ rosbag play [options] bag_name.bag

--rate=factor	Publish rate factor
--clock	Publish the clock time (set param use_sim_time to true)
--loop	Loop playback

ROS Packages

- ◆ ROS 소프트웨어는 소스 코드, launch 파일, configuration 파일, message definitions, data, 그리고 문서를 포함할 수 있는 패키지로 구성된다.
- ◆ 다른 패키지를 빌드하거나 필요로 하는 (예를 들어, 메시지 정의) 패키지는 이들을 **dependencies**로 선언한다
- ◆ 새로운 패키지를 만들 때 아래와 같이 한다.
 - `$ catkin_create_pkg package_name {dependencies}`



ROS ecosystem에서 코드 구성 방법

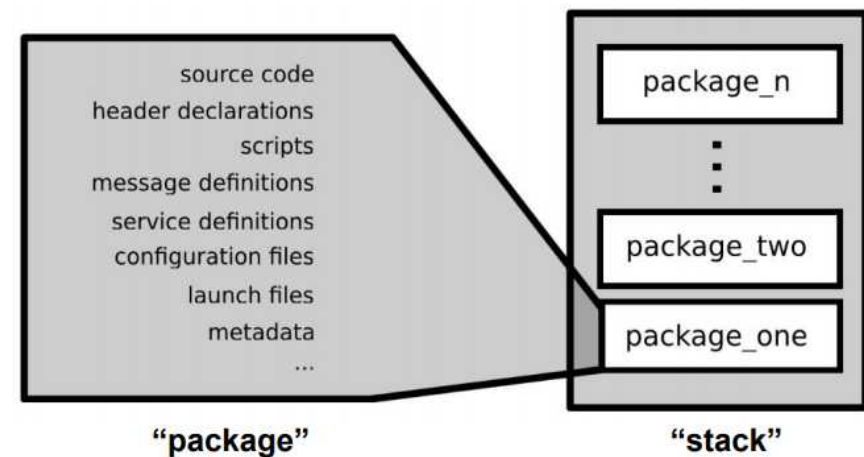
◆ ROS 코드는 두 가지 다른 레벨로 그룹화 된다

◆ Packages:

- ROS 빌드 시스템에서 atomic dependency로 취급되고 빌드 되는 이름이 부여된 소프트웨어 모음

◆ Stacks:

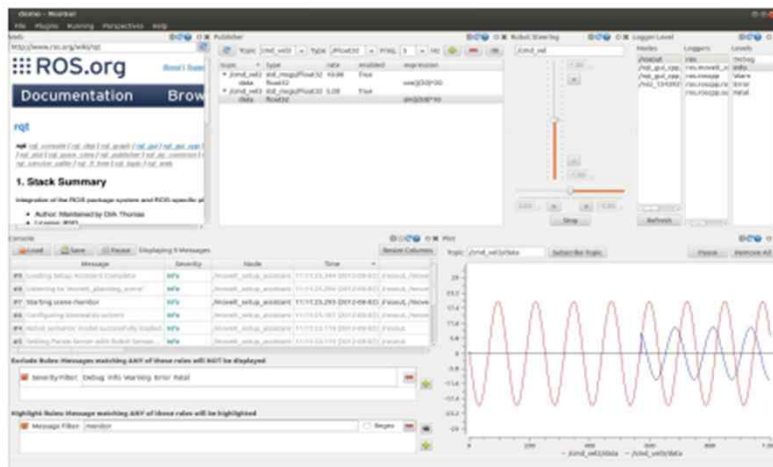
- 배포를 위해 명명된 패키지 모음



ROS GUI Tools

◆ ROS에서 사용가능한 GUI Tools

rqt : A QT based GUI developed for ROS



rviz : Powerful tool for 3D Visualization



ROS와 함께 사용할 수 있는 라이브러리/툴

◆ ROS와 함께 사용할 수 있는 libraries/tools



ROS Catkin 기반 패키지 생성 및 분산처리 기능 구현

Catkin - Build System

◆ Catkin

- Catkin은 실행파일, 라이브러리, 그리고 인터페이스를 만드는 ROS 빌드 시스템이다
- Catkin은 제공되는 설치 환경에 catkin command line 툴이 사전 설치된다

◆ Catkin workspace로 이동해서 아래 명령으로 패키지를 빌드 한다.

- `$ cd ~/catkin_ws`
- `$ catkin_make --package package_name`

◆ 새로운 패키지를 빌드 할 때마다 환경을 업데이트 한다

- `$ source devel/setup.bash`



Catkin Build System

◆ Catkin workspace

Work here



src

The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch



build

The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch



devel

The development (devel) space is where built targets are placed (prior to being installed).

Catkin

◆ What is Catkin

- http://wiki.ros.org/catkin/conceptual_overview
- catkin is **the official build system of ROS** and the successor to the original ROS build system, rosbuilt.
- [catkin](#) combines [CMake](#) macros and Python scripts to provide some functionality on top of CMake's normal workflow.
- [catkin](#) was designed to be more conventional than [rosbuild](#), allowing for better distribution of packages, better cross-compiling support, and better portability.
- [catkin](#)'s workflow is very similar to [CMake](#)'s but adds support for automatic 'find package' infrastructure and building multiple, dependent projects at the same time.



Catkin

◆ Installation of Catkin

- https://wiki.ros.org/catkin#Installing_catkin
- `sudo apt-get install ros-noetic-catkin`
- Catkin has the following dependencies:
 - [CMake](#) - A cross-platform, open-source build system.
 - [Python](#) - Python is a general-purpose, interpreted high-level programming language. Version 2.7 is required ([reference](#)).
 - [catkin_pkg](#) - A Python runtime library for catkin.
 - [empy](#) - A Python template library.
 - [nose](#) - A Python testing framework.
 - [GTest](#) - A cpp unittest framework from Google.
 - [GNU C++ Compiler \(g++\)](#) - The GNU C++ compiler
- `sudo apt-get install cmake python-catkin-pkg python-empy python-nose python-setuptools libgtest-dev build-essential`



Catkin, Workspace

◆ catkin

- ROS의 빌드 시스템을 말한다.
- ROS 빌드 시스템은 기본적으로 CMake(Cross Platform Make)를 이용하고 있어서 패키지 폴더에 CMakeList.txt 의 파일에 빌드 환경을 기술하고 있다.
- ROS에서는 CMake를 ROS에 맞도록 수정하여 ROS에 특화된 catkin 빌드 환경을 만들었다.
- catkin 빌드 시스템은 ROS와 관련된 빌드, 패키지 관리, 패키지 간의 의존 관계 등을 편리하게 사용할 수 있다.

◆ ROSBuild

- catkin 빌드 시스템 이전에 사용했던 빌드 시스템이다.
- 현재도 호환성 때문에 사용할 수 있지만 추천하지 않는다.

◆ workspace

- 소스파일과 환경이 들어 있고, 패키지를 컴파일하는 방법을 제공한다.
- 다양한 패키지를 동시에 컴파일하려고 할 때 편리하다.



Workspace 공간 별 용도

◆ src 공간

- 패키지, 프로젝트 등을 넣는 곳이다.
- 이 영역의 파일 중에서 가장 중요한 파일은 CMakeList.txt이다.
- Workspace에서 패키지를 구성할 때 cmake에 의해 src폴더가 컴파일되기 때문이다.

◆ build 공간

- Cmake와 catkin이 패키지와 프로젝트에 대한 캐시 정보, 설정, 그리고 다른 매개 파일들을 보관한다.

◆ devel 공간

- 컴파일된 프로그램들을 보관하는 공간이다.
- 프로그램을 테스트할 때 사용되며, 테스트가 끝나면 다른 개발자들과 공유하기 위해 패키지를 저장하거나 보낼 수 있다.

ROSCore, ROSrun, ROSlaunch

◆ ROSCore

- ROS 마스터를 실행하는 명령어(roscore)다.
- 같은 네트워크라면 다른 컴퓨터에서 실행해도 된다.
- 단 멀티 ROS코어를 지원하는 특수한 경우를 제외하고는 ROS코어는 동일 네트워크에서 하나만 구동되게 되어있다.

◆ ROSrun

- rosrun은 ROS의 기본적인 실행 명령어다. 패키지에서 하나의 노드를 실행하는데 사용된다.

◆ ROSlaunch

- roslaunch는 한번에 복수 개의 노드를 실행하는 명령어다.
- 그 외의 기능으로 실행 시 패키지의 매개변수를 변경, 노드 명의 변경, 노드 네임 스페이스 설정, ROS_ROOT 및 ROS_PACKAGE_PATH설정, 이름 변경, 환경 변수 변경 등의 실행 시 변경 할 수 있는 많은 옵션들을 가진 노드 실행에 특화된 ROS명령어다.

ROS Launch

◆ ROS Launch

- Launch는 여러 개의 노드와 설정 파라미터를 실행하는 툴이다
- *.launch 파일로 XML로 작성된다
- Roscore가 실행 중이 아니라면, launch는 roscore를 자동으로 실행한다
- 아래 명령으로 패키지의 launch 파일을 실행한다
 - `$ roslaunch package_name file_name.launch`



ROS Parameter Server

◆ ROS Parameter Server

- Node는 런-타임 시에 parameter server를 사용하여 매개변수를 검색하고 저장
- 구성 매개변수와 같은 정적 데이터에 가장 적합
- 매개변수는 launch 파일이나 별도의 YAML 파일에 저장될 수 있다
- 모든 매개변수는 다음 명령으로 리스트 한다
 - `$ rosparam list`



ROS Time

◆ ROS Time

- ROS는 일반적으로 PC 시스템의 클럭을 타임 소스로 사용한다
- 기록된 데이터의 시뮬레이션이나 재생의 경우, 시뮬레이션 된 시간(일시 정지, 슬로우 다운 등)으로 작업하는 것이 편리하다
- 시뮬레이션 된 시간으로 작업하기 위해,
 - /use_sim_time 매개변수를 설정
 - \$ rosparam set use_sim_time true
 - 다음으로부터 topic/clock에 시간을 게시
 - Gazebo (디폴트로 활성화 됨)
 - ROS bag (--clock 옵션 사용)
- 시뮬레이터 된 시간의 이점을 취하기 위해 항상 ROS Time API를 사용

```
ros::Time  
ros::Time begin = ros::Time::now();  
double secs = begin.toSec();
```

```
ros::Duration  
ros::Duration duration(0.5); // 0.5s
```

ROS 실습

◆ ROS 작업공간 (Workspace) 생성

```
// 시스템 레벨의 ROS 설정. .bashrc 파일에 추가
$ source /opt/ros/melodic/setup.bash

// 작업공간의 생성과 초기화
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace // 이 폴더에 CMakeLists.txt 파일 생성

// 아래 명령의 실행으로 build와 devel 폴더 그리고 많은 결과물 생성
// build: C++ 사용시에 라이브러리와 실행 프로그램과 같은 일부 결과물을 저장.
// devel: 환경 설정파일들을 저장.
$ cd ~/catkin_ws
$ catkin_make

// 시스템이 현재 작업공간과 그 안에 포함된 코드를 사용하도록 설정
$ source devel/setup.bash
```



ROS 실습

◆ 패키지 파일 (beginner) 생성과 노드 실행

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg beginner rospy

$ cd ~/catkin_ws/src/beginner
$ mkdir scripts && cd scripts // C++ 소스인 경우, src 폴더로 이동
$ vi publisher.py
$ vi subscriber.py

$ cd ~/catkin_ws && catkin_make
$ roscore
$ rosrun beginner publisher.py
$ rosrun beginner subscriber.py
$ rqt_graph
```



ROS Package 내용

```
cais@cais-pc: ~/Downloads/HUINS-11_to_19/ros_ws/catkin_ws/src/beginner
cais@cais-pc: ~/Downloads/HUINS-11_to_19/ros_ws/catkin_ws/src/beginner$ tree -L 1
.
├── CMakeLists.txt
├── include
├── package.xml
├── scripts
└── src

3 directories, 2 files
```

these files indicate that the folder we are in is a ROS package.

Python scripts

C++ source files

ROS node/topic

◆ Python으로 ROS publisher 생성

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4
5 def talker():
6     pub = rospy.Publisher('publisher_topic', String, queue_size=10)
7     rospy.init_node('publisher_node', anonymous=True)
8     rate = rospy.Rate(10) # 10hz
9     while not rospy.is_shutdown():
10         hello_str = "welcome to CAIAS (^_^) %s" % rospy.get_time()
11         rospy.loginfo(hello_str)
12         pub.publish(hello_str)
13         rate.sleep()
14
15 if __name__ == '__main__':
16     try:
17         talker()
18     except rospy.ROSInterruptException:
19         pass
```



ROS node/topic

◆ Python으로 ROS subscriber 생성

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import String
4
5 def callback(data):
6     rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
7
8 def listener():
9     rospy.init_node('listener_node', anonymous=True)
10
11     rospy.Subscriber("publisher_topic", String, callback)
12
13     # spin() simply keeps python from exiting until this node is stopped
14     rospy.spin()
15
16 if __name__ == '__main__':
17     listener()
```

Catkin Workspace 생성

◆ Catkin_ws 생성(1)

- source 명령어는 스크립트 파일을 수정한 후에 수정된 값을 바로 적용하기 위해 사용하는 명령어이다.
- \$ cd
- \$ source /opt/ros/melodic/setup.bash
- \$ mkdir -p ~/catkin_ws/src
- \$ cd ~/catkin_ws/
- \$ catkin_make

```
iron@antl:~/catkin_ws$ catkin_make
Base path: /home/iron/catkin_ws
Source space: /home/iron/catkin_ws/src
Build space: /home/iron/catkin_ws/build
Devel space: /home/iron/catkin_ws/devel
Install space: /home/iron/catkin_ws/install
Creating symlink "/home/iron/catkin_ws/src/CMakeLists.txt" pointing to "/opt/ros/melodic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /home/iron/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/iron/catkin_ws/devel -DCMAKE_INSTALL_PREFIX=
####
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
```



Catkin Workspace 생성

◆ Catkin_ws 생성(2)

```
####  
#### Running command: "make -j4 -l4" in "/home/iron/catkin_ws/build"  
####
```

- 이러한 문구가 뜬다면 해당 경로로 이동해 위의 명령어를 실행한다.

```
$ source devel/setup.bash  
$ echo $ROS_PACKAGE_PATH
```

```
iron@antl:~/catkin_ws$ echo $ROS_PACKAGE_PATH  
/home/iron/catkin_ws/src:/opt/ros/melodic/share
```

- 아래의 경로에서 \$ tree 를 실행시켜보면 workspace가 생성된 것을 볼 수 있다.

```
iron@antl:~/catkin_ws$ tree
```

```
build  
├── atomic_configure  
│   ├── env.sh  
│   ├── local_setup.bash  
│   ├── local_setup.sh  
│   ├── local_setup.zsh  
│   ├── setup.bash  
│   ├── setup.sh  
│   └── _setup_util.py  
└── setup.zsh
```

```
devel  
├── cmake.lock  
├── env.sh  
├── lib  
├── local_setup.bash  
├── local_setup.sh  
├── local_setup.zsh  
├── setup.bash  
├── setup.sh  
├── _setup_util.py  
└── setup.zsh  
src  
└── CMakeLists.txt -> /opt/ros/melodic/share/catkin/cmake/toplevel.cmake
```

Multiple workspace Example

◆ 첫 번째 catkin 작업 공간 만들기

- `$ mkdir -p ~/catkin_ws/src`
- `$ cd ~/catkin_ws/`
- `$ catkin_make`
- 아래의 명령어로 catkin_ws가 자동으로 작동하게 한다.
- `$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc`
- 이제 .bashrc에는 다음이 포함된다.
참고로 두 줄의 순서는 매우 중요하다.
무조건 catkin_ws 전에 global ROS install이 있어야 한다.

```
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

- catkin작업 공간 생성은 두 단계로 이루어지는 것을 알 수 있다.
 1. 내가 작업공간을 만들고자 하는 경로에서 catkin_make로 작업공간 생성
 2. 내가 만든 catkin 작업 공간을 source하여 사용

Multiple workspace Example

◆ 다른 catkin 작업 공간 만들기

- 이미 가지고 있는 workspace외에 다른 workspace를 생성 (Home폴더가 아닌 원하는 위치에 만들 수도 있다.)
- `$ mkdir -p ~/test_d/catkin_ws2/src`
- `$ cd ~/test_d/catkin_ws2/`
- `$ catkin_make`
- `$ echo "source ~/test_d/catkin_ws2/devel/setup.bash" >> ~/.bashrc`
- 이제 .bashrc에는 다음의 줄들이 있다.

```
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
source ~/test_d/catkin_ws2/devel/setup.bash
```

- 첫 번째 줄은 컴퓨터에서 ROS가 설치된 위치를 찾아 실행한다.
- 두 번째 줄은 처음으로 만든 작업 공간인 catkin_ws를 활성화 한다.
- 세 번째 줄은 그 다음 만든 작업 공간인 catkin_ws2를 활성화하고, 이전에 활성화 되었던 catkin_ws는 비활성화 한다.

→ 이처럼 동일한 세션에서 작업 공간을 여러 개 만들 수 있지만 여러 개를 동시에 실행 시킬 수는 없다.



Multiple workspace Example

◆ 여러 catkin 작업 공간 간의 전환

- Catkin workspace에서 다른 workspace로 전환하려면 활성화하려는 workspace를 source명령어로 실행하면 된다.
- 터미널을 열게 되면 .bashrc에 마지막 줄의 workspace가 활성화 된다.
예를 들어 보면 밑의 사진에서 보이는 catkin_ws2가 활성화 되는 것이다.

```
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
source ~/test_d/catkin_ws2/devel/setup.bash
```

- 여기서 catkin_ws로 전환하고 싶다면 터미널 창에
\$ source ~/catkin_ws/devel/setup.bash를 치면 된다.
- 또는 .bashrc에서 줄의 순서를 변경하거나 필요 없는 줄에 주석을 달아서 다수의 workspace를 관리 할 수 있다.

catkin workspace에서 package 생성

◆ 수작업으로 패키지 만들기

- `$ cd ~/catkin_ws/src`
- `$ catkin_create_pkg chapter2_tutorials std_msgs roscpp`
- 이 명령의 포맷은 패키지 이름과 이 패키지가 가질 의존관계를 포함한다.
- `$ catkin_create_pkg [package_name][종속성1] ... [종속성2]`

- 아래는 예시로 `std_msgs`와 `roscpp`를 포함하였다.
- `std_msgs`: 근본적인 데이터 타입과 메시지 구조를 나타내는 공통 메시지 타입을 포함.
- `roscpp`: ROS를 C++로 구현한 코드. 이는 C++ 프로그래머들이 ROS 토픽, 서비스, 파라미터들과 빠르게 인터페이스를 할 수 있게 도와주는 클라이언트 라이브러리를 제공한다.

```
iron@antl:~/catkin_ws/src$ catkin_create_pkg chapter2_tutorials std_msgs roscpp
Created file chapter2_tutorials/CMakeLists.txt
Created file chapter2_tutorials/package.xml
Created folder chapter2_tutorials/include/chapter2_tutorials
Created folder chapter2_tutorials/src
Successfully created files in /home/iron/catkin_ws/src/chapter2_tutorials. Please adjust the values in package.xml.
iron@antl:~/catkin_ws/src$ ls
chapter2_tutorials  CMakeLists.txt
```

Package build

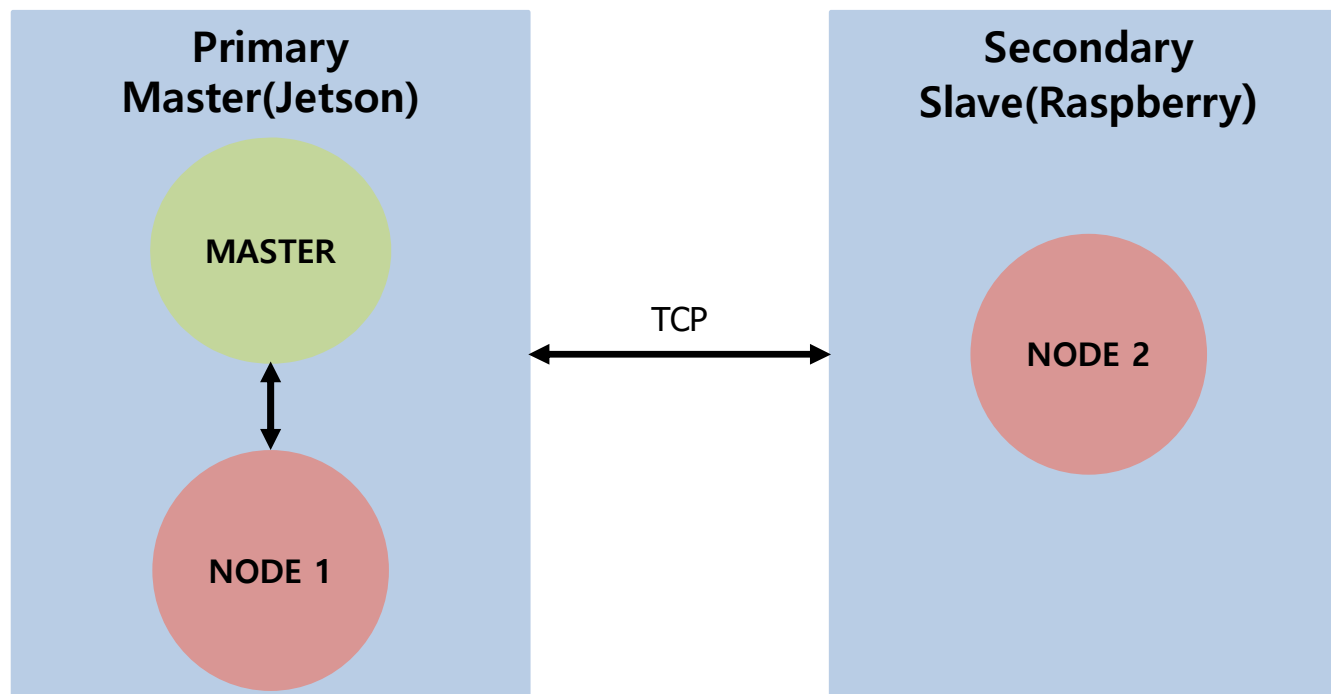
◆ Workspace에서 패키지 빌드

- \$ cd ~/catkin_ws/
- \$ catkin_make
- 잘 컴파일 되었다면 이러한 화면을 보게 된다.

```
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~
-- ~~~~ traversing 1 packages in topological order:
-- ~~~~ - chapter2_tutorials
-- ~~~~
-- +++ processing catkin package: 'chapter2_tutorials'
-- ==> add_subdirectory(chapter2_tutorials)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/iron/catkin_ws/build
```

- catkin_make는 반드시 workspace폴더에서 실행되어야 한다.
- 만약 단일 패키지를 컴파일하고 싶다면
\$ catkin_make -pkg <package name> 를 실행하면 된다.

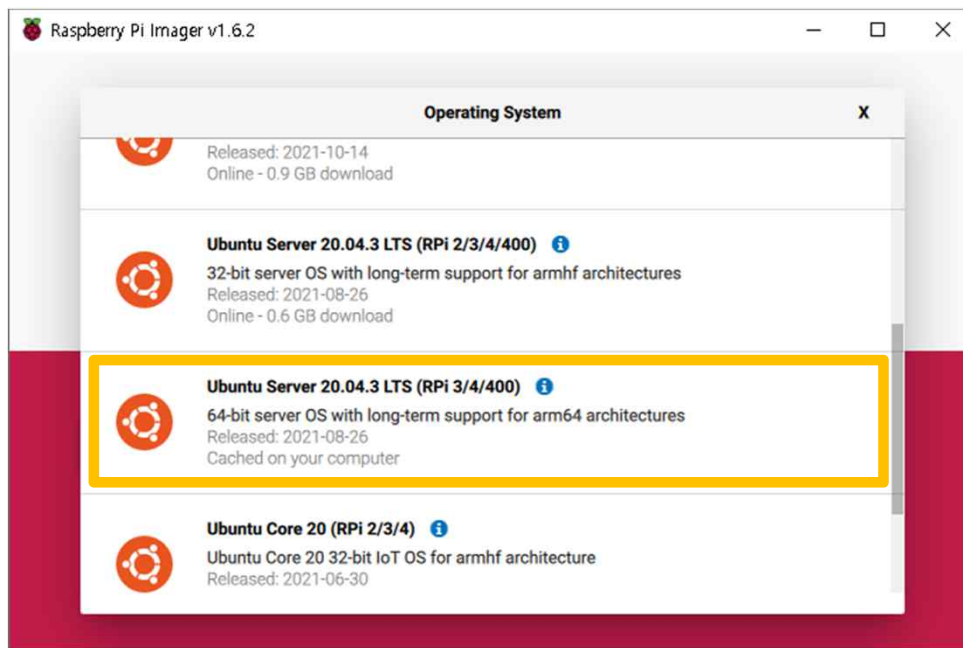
Communication over TCP/IP on ROS(1)



Communication over TCP/ IP on ROS(2)

◆ Install Ubuntu on Raspberry

- Ubuntu Server 20.04 설치



Communication over TCP/IP on ROS(3)

◆ Install ROS noetic on Raspberry

참고: <http://wiki.ros.org/noetic/Installation/Ubuntu>

Ubuntu 20.04는 ROS melodic 지원하지 않으므로 ROS noetic install

- `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`
- `sudo apt install curl`
- `curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -`
- `sudo apt update`
- `sudo apt install ros-noetic-desktop`
- `source /opt/ros/noetic/setup.bash`
- `echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc`
- `source ~/.bashrc`
- `sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential`
- `sudo apt install python3-rosdep`
- `sudo rosdep init`
- `rosdep update`



Communication over TCP/IP on ROS(4)

◆ Create package on ROS noetic

- Catkin_ws, Package, node 생성 과정은 melodic과 동일
- // 단, Package와 node 생성 후 CMakeList.txt 파일을 아래와 같이 수정해야 함

CMakeList.txt

```
catkin_install_python(PROGRAMS package 아래 경로명/노드명(1) package 아래 경로명/노드명(2)  
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}  
)
```

Communication over TCP/IP on ROS(5)

◆ 기기간 연결을 위한 환경 변수 설정

- ROS_IP, ROS_MASTER_URI, ROS_HOSTNAME 3개의 환경변수를 설정해주어야 함
- vi ~/.bashrc (**master와 slave 각각 수정**)

master

```
export ROS_IP={IP_of_master}  
export ROS_MASTER_URI=http://localhost:11311  
export ROS_HOSTNAME=$ROS_IP
```

slave

```
export ROS_MASTER_URI=http://{IP_of_master}:11311  
export ROS_HOSTNAME={IP_of_slave}
```

- source ~/.bashrc (**master와 slave 각각 실행**)

Communication over TCP/IP on ROS(6)

● 실행 결과

- 터미널 창 3개 필요
- roscore (master)
- rosrun package명 node명 (master)
- rosrun package명 node명 (slave)

```
auto-starting new master
process[master]: started with pid [4316]
ROS_MASTER_URI=http://165.229.185.182:11311/

setting /run_id to 4453b16e-7e9f-11ec-be83-48b02d2e2901
process[rosout-1]: started with pid [4331]
started core service [/rosout]
```

master

```
antl@antl-desktop:~$ rosrun pythonclass_topic talker.py
ready for talk
input message:asdf
('talker_str is :', 'asdf')
```

master

```
ubuntu@ubuntu:~$ rosrun test listener.py
ready to listen
asdf
```

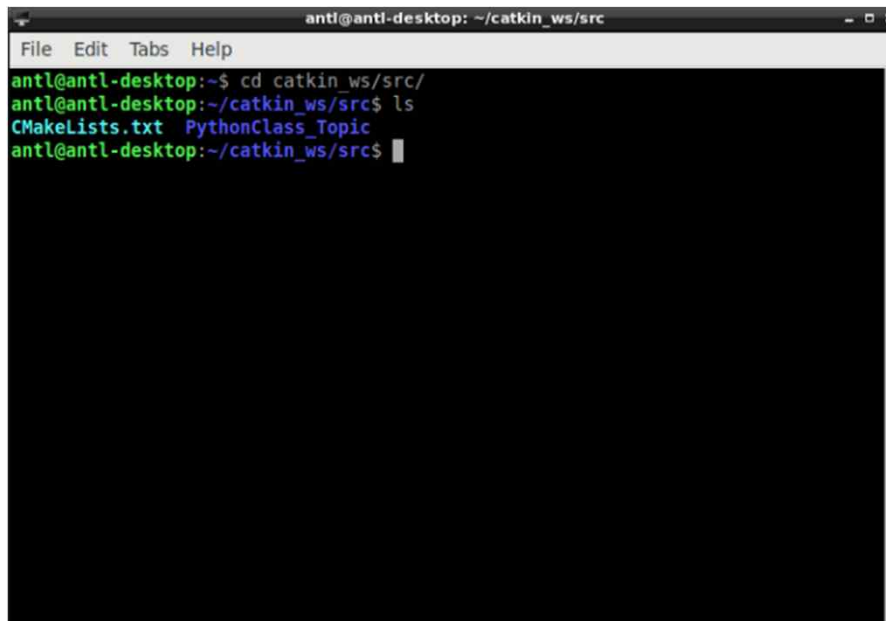
slave



Python Programming in ROS(1)

◆ 패키지 만들기

- `cd ~/catkin_ws/src`
- `catkin_create_pkg pythonclass_topic std_msgs rospy`
//이때 패키지 이름에는 대문자 사용불가



```
anti@antl-desktop: ~/catkin_ws/src
File Edit Tabs Help
antl@antl-desktop:~$ cd catkin_ws/src/
antl@antl-desktop:~/catkin_ws/src$ ls
CMakeLists.txt PythonClass_Topic
antl@antl-desktop:~/catkin_ws/src$
```

Python Programming in ROS(2)

◆ Publisher 생성

- `vi talker.py` // 단, 앞서 생성한 `pythonclass_topic/src` 경로에 파일을 생성해야 함
- `chmod +x talker.py`



◆ Source Code

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def talker():
    print("ready for talk")
    while not rospy.is_shutdown():
        talker_str = raw_input("input message:")
        print("talker_str is :", talker_str)
        pub.publish(talker_str)
        rate.sleep()

if __name__ == "__main__":
    pub = rospy.Publisher('pythonclass', String, queue_size=10)
    rospy.init_node('talker', anonymous = True, disable_signals = True)
    rate = rospy.Rate(10)
    try:
        talker()
    except KeyboardInterrupt:
        pass
```



Python Programming in ROS(3)

◆ Subscriber 생성

- `vi listner.py` // 단, 앞서 생성한 `pythonclass_topic/src` 경로에 파일을 생성해야 함
- `chmod +x talker.py`



◆ Source Code

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    listen_data = data.data
    print(listen_data)

def listener():
    print("ready to listen")
    rospy.Subscriber('pythonclass', String, callback)

    rospy.spin()

if __name__ == "__main__":
    rospy.init_node('listener', anonymous = True)
    listener()
```

Python Programming in ROS(4)

◆ 생성한 node 실행

- 터미널 창 3개준비
- 첫번째 터미널에 roscore 입력 (master실행)
- 두번째 터미널에 rosruntime pythonclass_topic talker.py (node 실행)
- 세번째 터미널에 rosruntime pythonclass_topic listener.py (node 실행)

The image displays three terminal windows illustrating the ROS setup and node execution:

- Top Window (roscore):** Shows the ROS master starting. It includes a summary of parameters (roscdistro: melodic, rosversion: 1.14.12) and a list of nodes (auto-starting new master, process[master]: started with pid [8076], ROS_MASTER_URI=http://antl-desktop:11311/).
- Bottom Left Window (talker):** Shows the execution of the talker node. It receives input messages like "hello listener", "I'm talker", "Welcome to PythonClass", "This is YU ANTL", and "Have a nice day!".
- Bottom Right Window (listener):** Shows the execution of the listener node. It receives input messages like "hello listener", "I'm talker", "Welcome to PythonClass", "This is YU ANTL", and "Have a nice day!".

ROS의 부족한 점

◆ 신뢰성 부족

- ROS에서 마스터는 하나만 존재하며, 장애가 발생한 노드를 복구하기 위한 모니터가 없음

◆ 성능 저하 가능성

- 메시지를 방송 (broadcasting) 방식으로 보내는 과정에서 동일한 메시지가 여러 번 중복될 수 있으며, 이로 인해 성능 저하 발생 가능

◆ 보안 (인증, 암호화) 기능 부족

- 인증 (authentication)과 암호화 (encryption) 메커니즘이 없음

Homework 21

Homework 20

21.1 Raspberry Pi 환경에 ROS를 설치하고, 노드간 통신 기능 구현

- Raspberry Pi에 ROS를 설치하라.
- 설치된 ROS에서 하나의 publisher (event_generator)와 하나의 subscriber (event_processor)가 통신할 수 있도록 topic (event)를 구성하라.
- publisher인 event_generator가 10개의 event 메시지를 생성하여 subscriber인 event_processor에게 전달하는 기능을 구현하라.

21.2 두대의 Raspberry Pi 환경에 ROS를 설치하고, 서로 다른 호스트에 구성된 노드간 통신 기능 구현

- ROS가 설치된 Raspberry Pi 2대를 준비하라.
- 설치된 ROS에서 하나의 publisher (event_generator)와 하나의 subscriber (event_processor)를 서로 다른 Raspberry Pi에 각각 구성하고, 서로 통신할 수 있도록 topic (event)를 구성하라.
- publisher인 event_generator가 10개의 event 메시지를 생성하여 subscriber인 event_processor에게 전달하는 기능을 구현하라.



Reference

- [1] <http://wiki.ros.org/ko/ROS/Introduction>.
- [2] ROS tutorial #1: Introduction, Installing ROS, and running the Turtlebot simulator, <https://www.youtube.com/watch?v=9U6GDonGFHw>
- [3] <http://wiki.ros.org/ROS/Tutorials>.
- [4] Introduction to the Robot Operating System (ROS) Middleware - Mike Anderson, The PTR Group, Inc., <https://www.youtube.com/watch?v=yWtGUk3PBms>.
- [5] TurtleBot, <http://wiki.ros.org/Robots/TurtleBot>.
- [6] (PACKT)ROS로 효과적인 로봇 프로그래밍하기 - 저자 Anil Machtani, Luis Sanchez, Enrique Fernandez, Aaron Martinez.
- [7] http://wiki.ros.org/catkin/Tutorials/create_a_workspace
- [8] <https://roboticsbackend.com/ros-multiple-catkin-workspaces/>
- [9] http://wiki.ros.org/catkin/Tutorials/using_a_workspace
- [10] <https://www.slideshare.net/yoonseokpyo/20160406-ros-1-for>
- [11] <https://jungmonster.tistory.com/154>.
- [12] <https://www.youtube.com/watch?v=0BxVPCInS3M>.
- [13] Introduction to the Robot Operating System (ROS) Middleware - Mike Anderson, The PTR Group, Inc., <https://www.youtube.com/watch?v=yWtGUk3PBms>.
- [14] ROS and Raspberry Pi for Beginners | Tutorial #0 - Topics Packages RosMaster, https://www.youtube.com/watch?v=iLiI_IRedhI.
- [15] <https://www.theconstructsim.com/how-to-install-ros-on-ubuntu/>.
- [16] Dongkeycar with Raspberry Pi, <https://www.donkeycar.com/>.
- [17] <https://mushr.io/tutorials/intro-to-ros/>.
- [18] ROS melodic install(1), <https://wiki.ros.org/melodic>
- [19] ROS melodic install(2), <https://95mkr.tistory.com/entry/ROS3>

