[ROBIT ROS 보고서]

18기 지능팀 예비단원 주성민

1. 로봇 소프트웨어 플랫폼의 변화

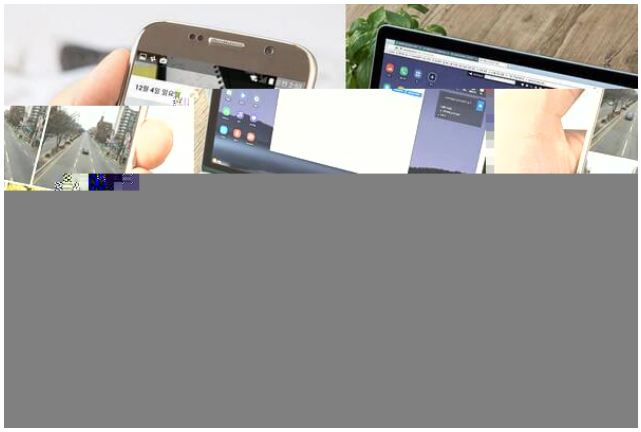
1) 컴퓨터와 휴대폰이 발전할 수 있었던 이유

- 컴퓨터와 휴대폰은 메인 보드, 회로 등 다양한 하드웨어의 결합이 가능한 하드웨어 모듈화가 되어있다.

- window, linux, Ubuntu 등의 운영체제가 존재하고 그 운영체제를 기반으로 한 어플리케이션이 탑재되어있다.

- 또한 하드웨어 모듈, 운영체제, 앱, 유저와 같이 네 가지 요소를 가진 에코 시스템이 형성되어있고 보이지 않는 생태계 속의 분업처럼 각 요소들이 맡은 역할을 수행한다

- 유저들이 사용하기 쉬운 환경이며 하드웨어 모듈, 운영체제, 앱을 이용하면서 피드백을 줄 수 있고 개발자들은 이 피드백을 받아들여 발전할 수 있었기에 현재와 같은 위상을 차지할 수 있었다.



2) 로봇 분야의 근황

(1) 주요 로봇 운영체제의 종류

- ROS(Robot Operating System): 로봇 응용 프로그램을 개발할 때 필요한 하드웨어, 기능의 구현, 메시지 패싱, 라이브러리 개발 및 디버깅 도구를 제공하는 로봇 개발을 위한 로봇 플랫폼

- RT-Middleware(RTM): 로봇 기능 요소(RT 기능 요소)의 소프트웨어 모듈을 복수 조합한 로봇 시스템(RT 시스템)을 구성하기 위한 소프트웨어 플랫폼

- OPRoS: 로봇의 주요 기능을 구현하는 응용 소프트웨어(컴포넌트), 이를 통합하는 표준 구조(프

레임워크), 개발 도구 등을 통칭하는 소프트웨어 플랫폼

2) 로봇 운영체제들의 특징

- 각각 다른 형태의 플랫폼을 개발(ROS: open source, OPRoS: Galapagos, NAO qi: closed source). 서로의 장점은 공유하고, 단점을 보완하면서 비슷한 환경으로 발전

- Window나 Mac OS처럼 로봇 분야에서 천하 통일할 정도로 독점하고 있는 운영체제가 없고 계속 발전 중, Personal Computer과 Personal Phone이 발전해왔던 절차 그대로 밟고 있음

3) 로봇 소프트웨어가 지향하는 방향성

(1) 로봇 소프트웨어 플랫폼이 가져올 미래

- 하드웨어 플랫폼과 소프트웨어 플랫폼 간의 인터페이스 확립

- 결합 가능한 모듈형 하드웨어 플랫폼의 확산

- 하드웨어에 대한 지식이 없어도 응용 프로그램 작성 가능(다른 분야에 종사하던 개발자의 참여 가능)

- 개발자의 능력에 따른 개발이 아닌, 유저의 수요에 따른 개발에 집중

- 실수요가 있는 서비스 제공으로 유저 계층 형성 및 피드백 활성화

=> Personal Computer, Personal Phone이 겪어왔던 과정, 역사 그대로 진행하고 발전

(2) 유저의 태도

어떠한 운영체제를 이용해야 하는가, 어떤 운영체제가 더 좋은가를 판단하는 것보다 어떤 운영체제를 사용하냐에 관계없이 해당 운영체제에 대한 역량을 기르는 것이 더 중요

2. ROS란 무엇인가?

1) 정의

- 로봇 소프트웨어 플랫폼

기계는 하드웨어와 이를 움직이는 소프트웨어로 구성되어있고 로봇을 구동하는 데에도 이 둘이 모두 필요하다. 로봇 소프트웨어 플랫폼은 로봇 응용 프로그램의 개발을 위한 *하드웨어 추상화, 하위 디바이스 제어와 센싱 및 인식, 위치 추정, 네비게이션 등의 기능. 패키지 관리, 라이브러리 개발 도구와 디버깅 도구를 모두 포함한다. 로봇의 소프트웨어와 하드웨어의 종류가 다양하지만 이로 인한 복잡성을 해결하기 위해 로봇 소프트웨어 플랫폼이 만들어졌고 비로소 협업이 가능해졌다.

*하드웨어 추상화: 프로그래밍 인터페이스를 통해 하드웨어 리소스에 접근할 수 있는 프로그램을 제공하는 소프트웨어 속 루틴의 집합. 쉽게 말해 하드웨어에 상관 없이 독립적으로 프로그래밍을 할 수 있도록 돕는다

- ROS란?

Robot Operating System의 약자로 Open Robotics에서 개발하고 있다. 오픈 소스로서 로봇의 메타 운영 시스템이다. 운영체제로서의 서비스를 기대할 수 있고, 하드웨어의 추상화, 디바이스의 쉬운 제어, 흔히 사용하는 기능의 실행, 프로세스 간의 메시지 패싱, 패키지 관리 등이 가능하다. Obtaining, Building, Writing을 위한 라이브러리와 도구를 제공하고 여러 대의 컴퓨터 간의 코드 실행이 가능하다. ROS는 로보틱스 소프트웨어 개발을 전세계 레벨에서 공동 작업이 가능하도록 생태계를 구축하는 것을 목표로 한다

- ROS의 필요성

여러 소프트웨어 플랫폼 중 하나인 ROS의 필요성은 다음과 같다

(1) 프로그램의 재사용성: 부분적으로 개발을 하고 다른 기능들은 필요한 패키지를 다운로드 받아 사용할 수 있고 공유가 가능하다.

(2) 통신 기반 프로그램: 하나의 서비스를 위해 하드웨어 드라이버, 센싱, 인식, 동작 등을 각각의 프로세서의 목적에 맞게 노드화를 하고 이 노드는 데이터를 플랫폼 상에서 주고 받는다. 따라서

하드웨어 의존성을 낮출 수 있으며, 이러한 네트워크 프로그래밍은 원격 제어를 용이하게 하고, 노드로 나누어진 덕분에 작은 단위로 디버깅을 할 수 있다.

(3) 개발 도구 지원: 디버깅, rqt, rviz 등을 지원하고 정해진 메시지 형식에 맞춰 이용하면 된다.

(4) 커뮤니티: 자발적으로 많은 패키지들이 개발 및 공유되고 있고 사용법이 적힌 wiki 페이지들이 있다.

(5) 로봇 생태계 구성: 하드웨어 기술을 통합할 운영체제가 필요하다. 다양한 소프트웨어가 필요하고 사용자도 이를 잘 이용해야한다. 모두가 따로 움직인다면 발전이 매우 더딜 것이므로 플랫폼은 개발 및 사용의 생태계를 잘 갖출 수 있도록 한다. 스마트폰의 하드웨어가 어떤 식으로 구성되어있는지 잘 알지 못해도 개발자들은 어플을 잘 개발한다. 이처럼 로봇 소프트웨어 플랫폼을 이용하면 로봇 하드웨어가 달라도 프로그래밍이 가능하다. 이러한 생태계를 잘 구성하고 관리하는 것이 ROS가 지향하는 바이다.

2) 특징

- ROS는 또 다른 하나의 운영체제인가?

전통적인 운영체제

(1) 범용 컴퓨터 기준으로 Windows(XP, 7, 8), Linux(Ubuntu, Redhat, Fedora 등), MAC 등이 있음

(2) 스마트폰 기준으로 Android, iOS 등의 운영체제가 존재한다

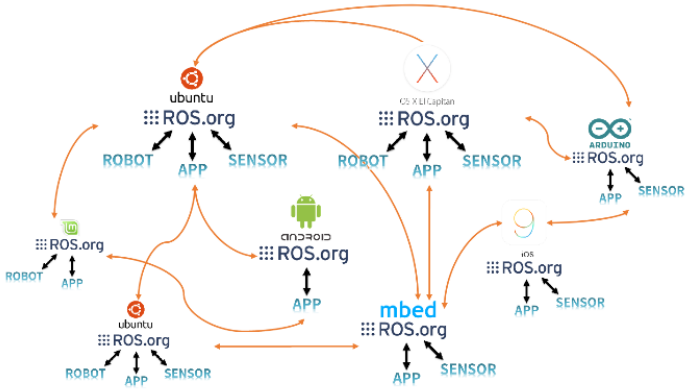
ROS가 새로운 운영체제라고 생각할 수도 있지만, 이는 메타 운영체제로 전통적인 운영체제 위에 설치되어 이들이 제공하는 기능들을 사용한다.

메타 운영체제란 애플리케이션과 분산 컴퓨팅자원간의 가상화 레이어로 분산 컴퓨팅 자원을 활용하여 스케줄링, 로드, 감시, 에러 처리 등을 실행하는 시스템이다. 즉, ROS는 윈도우, 리눅스와 같은 기존의 운영체제가 아니며 오히려 ROS는 기존의 전통적인 운영체제를 이용하고 있다. 기존 운영체제의 프로세스 관리 시스템, 파일 시스템, 유저 인터페이스, 프로그램 유틸 등을 사용하고 있다. 다수의 이기종 하드웨어간의 데이터 송수신, 스케쥴링, 에러 처리 등 로봇 응용 소프트웨어 개발을 위한 필수 기능들을 라이브러리 형태로 제공하고 있다. 또한 이러한 로봇 소프트웨어 프레임워크를 기반한 다양한 목적의 응용 프로그램을 개발, 관리, 제공하고 있으며 유저들이 개발한 패키지 또한 유통하는 생태계를 갖추고 있다.

- 이기종 디바이스간의 통신 지원

ROS가 메타운영체제를 택한 가장 큰 이유에는 이기종 디바이스간의 통신 지원이 가능하기 때문이다. 이기종 디바이스간의 통신 지원이란 어떤 운영체제를 사용하든 ROS를 사용하면 이기종 간의 시스템이더라도 통신을 지원해 로봇을 움직이거나 데이터를 받을 수 있게 된다. 세상에는 로봇의 하드웨어가 많고 다양하므로 하나의 운영체제(OS)로 모든 하드웨어를 다루기에는 한계가 존재한다. 따라서 로봇의 요구 조건을 하나의 하드웨어(OS)으로 충족을 시키지 못하므로 로봇에 맞는 운영체제를 사용하되, ROS를 설치하여 같이 사용하게 된다면 하드웨어(OS)가 달라도 통신이 가능하다는 것이다. 하나의 운영체제를 만드는 것보다 메타운영체제의 형태를 갖추면서 이기종 디바이스 간의 통신을 지원하는 것이 로봇을 개발하는데 있어서 유의미하다는 것이다.

이기종 디바이스간의 통신의 예로는 원격으로 이미지 전송하기, 안드로이드 스마트폰의 가속도 값을 pc에서 확인하기, 안드로이드 스마트 폰으로 turtle bot 제어하기 등이 있다. 이들은 모두 각각 다른 하드웨어를 이용하지만 이기종 디바이스간의 통신을 이용하여 제어하는 모습을 볼 수 있다.



- 통신 가능(통신 인프라)

  - 노드 간 데이터 통신을 제공

  - 통상적 미들웨어*로 지칭되는 메시지를 전달하는 인터페이스 지원

*미들웨어: 서로 다른 애플리케이션이 서로 통신하는데 사용되는 소프트웨어. 이는 빠르게 혁신할 수 있도록 애플리케이션을 지능적이고 효율적으로 연결하는 기능을 제공한다.

(1) 메시지 패싱 기능(message passing) *뒤에서 자세히 설명 예정

  - 로봇 개발 시에 빈번히 사용되는 통신 시스템 사용; 센서간의 데이터 전송이 가능

  - 캡슐화 및 코드 재사용을 촉진하는 노드들 간의 메시지를 전달하는 인터페이스 기능

(2) 메시지의 기록 및 재생

  - 로봇이 받고 있는 각종 센서, 디바이스의 상태를 기록할 수 있고 나중에 필요할 때 다시 재생할 수도 있다.

예) 내비게이션 알고리즘을 개발한다고 하면 코드가 작동이 되는지 수많은 디버깅이 필요함. 하지만 개발환경이 달라지거나 로봇의 하드웨어적 부분에서 변화가 생기면 기존 코드가 변화된 환경에서 제대로 동작하고 있는지 모름. 이때 환경을 고정시킨다면 알고리즘만 테스트할 수 있음. 로봇 작동 상태일 때의 값을 메시지로 주고 받는데 그 데이터를 저장하고 알고리즘을 수정한 뒤에 로봇을 돌리는 것이 아니라 저장된 값으로 돌리면 알고리즘의 향상 정도를 알 수 있음.

  => 노드 간 송/수신되는 데이터인 메시지를 저장하고 필요 시에 재사용 가능

  => 저장된 메시지를 기반으로 반복적인 실험 가능, 알고리즘 개발에 용이함

(3) 메시지 사용으로 인한 다양한 프로그래밍 언어를 사용 가능

  - 노드 간의 통신은 마스터를 기반으로 하기 때문에 각 노드에서 다른 프로그래밍 언어를 사용하더라도 통신이 되는 구조를 가진다.

  - 클라이언트 라이브러리: 클라이언트 라이브러리는 서비스를 요청하는 라이브러리로 roscpp, rospy, rosjava 등 다양한 프로그래밍 언어를 사용할 수 있다.

(4) 분산 매개 변수 시스템

- 시스템에서 사용되는 변수를 글로벌 키값으로 작성하여 공유 및 수정하여 실시간으로 반영, 예를 들면 프로그램을 돌리는데 외부에서 중요 변수를 글로벌 변수처럼 수정할 수 있음. 즉 프로그램 동작 프로세스를 외부에서 변화를 줄 수 있다는 것.

특징 2) 로봇 관련 다양한 기능

- 로봇에 대한 표준 메시지 정의: 카메라, IMU, 레이저 등의 센서/ 오도메트리, 경로 및 지도 등의 내비게이션 데이터 등의 표준 메시지를 정의하여 모듈화, 협업 작업을 유도, 효율성 향상

- 로봇 기하학 라이브러리: 로봇, 센서 등의 상대적 좌표를 트리화 시키는 TF 제공, 휴머노이드 로봇은 상대적 좌표로 표현 가능.

- 로봇 기술 언어: 로봇의 물리적 특성을 설명하는 XML 문서 기술 when modeling

- 진단 시스템: 로봇의 상태를 한눈에 파악할 수 있는 진단 시스템 제공. 예) 로봇의 베터리, 센서의 상태, 모터의 각도 등등을 진단할 수 있음

- 센싱/인식: 센서 드라이버, 센싱/인식 레벨의 라이브러리 제공

- 내비게이션: 로봇에서 많이 사용되는 로봇의 포즈 추정, 지도 내의 자기 위치 추정 제공. + 지도 작성에 필요한 SLAM, 작성된 지도 내에서 목적지를 찾아가는 navigation 라이브러리 제공. 예)내가 원하는 곳으로 로봇을 이동시키는 기술. 그 사이에 나오는 경로, 회피 기능을 구현할 수 있음.

- 매니퓰레이션: 로봇 앞에 사용되는 IK, FK는 물론 응용단의 pick and place를 지원하는 다양한 Manipulation 라이브러리 제공 + GUI 형태의 매니퓰레이션 TOOLS 제공(MoveIt).

특징 3) 다양한 개발 도구

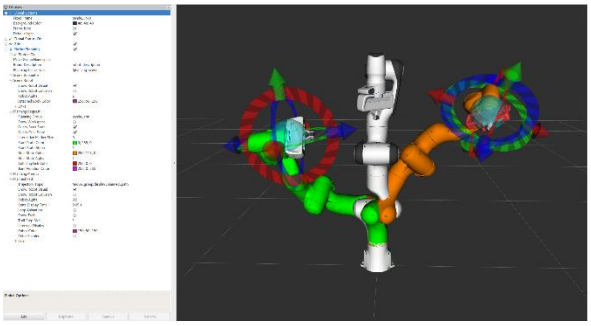
로봇 개발에 필요한 다양한 개발 도구를 제공하고 로봇 개발의 효율성을 향상시켜준다.

(1) Command-Line Tools: GUI 없이 ROS에서 제공되는 명령어로만 로봇 엑세스 및 거의 모든 로스 기능을 소화. *GUI란 Graphical User Interface로 사용자가 편리하게 사용할 수 있도록 입출력 등의 기능을 알기 쉬운 아이콘으로 나타내는 것

(2) RViz: 강력한 3D 시각화 툴을 제공, 레이저, 카메라 등의 센서 데이터를 시각화해주고, 로봇 외형과 계획된 동작을 표현해준다.

(3) RQT: 그래픽 인터페이스 개발을 위한 GUI 기반의 툴 box인 Qt 기반 프레임 워크를 제공하고 노드와 그들 사이의 연결 정보를 표시한다(rqt_gragh). 인코더, 전압 또는 시간이 지남에 따라 변화하는 숫자를 플로팅한다(rqt_plot). 또한 데이터를 메시지 형태로 기록하고 재생하는 rqt_bag도 제공한다.

(4) Gazebo: 물리엔진을 탑재하고, 로봇, 센서, 환경 모델 등을 지원하고, 3차원 시뮬레이터로서 작동한다. 또한 ROS와도 높은 호환성을 가진다.


<RViz>

3) 명령어 및 단축키

ROS를 이용하는데 있어서 특정 기능을 실행시키는 단축키와 명령어가 존재한다. Package, topic, message, node, 실행, bag 관련 명령어가 존재한다. 위의 용어에 관한 내용은 아래에 설명할 예정이다.

(1) PACKAGE 관련

명령어	설명
$ catkin_create_pkg <패키지 이름> <의존성>	새로운 패키지를 생성
$ rospack list	시스템에 설치되어 있는 패키지를 확인
$ rospack find <패키지 이름>	설치된 패키지의 위치를 확인
$ rospack depends <패키지 이름>	패키지와 의존성있는 패키지를 확인
$ roscd <위치 이름>	패키지가 존재하는 디렉토리로 이동

(2) TOPIC 관련

명령어	설명
$ rostopic list	시스템에서 사용중인 토픽을 확인한다
$ rostopic type <토픽 명>	토픽의 타입을 확인한다
$ rostopic info <토픽 명>	토픽의 타입과 더불어 현재 해당 토픽을 구독하고 발행하는 노드의 정보를 확인한다
$ rostopic echo <토픽 명>	전달 중인 토픽의 메시지 내용을 확인한다

(3) MESSAGE 관련

명령어	설명
$ rosmsg show <토픽명>	토픽 안의 메시지 데이터의 타입을 확인한다

(4) NODE 관련

명령어	설명
$ rosnode list	시스템에 실행중인 node의 리스트 확인
$ rosnode info <노드 명>	특정 node의 정보를 확인
$ rqt_gragh	노드들 간의 연결된 메시지 관계를 시각화

(5) 실행 관련

명령어	설명
$ rosrun <패키지 명> <코드 파일 이름>	작성한 소스코드를 실행
$ roslaunch <패키지 명> <런치 파일 이름>	Rosrun은 노드 하나를 실행시킬 때 사용하는 명령어이다. 여러 개의 노드를 실행시키기 위해서는 rosrun을 실행하는 노드 개수만큼 켜야 한다. Roslaunch는 Rosrun 명령을 쉽게 사용하기 위한 명령으로 여러 개의 노드를 한번에 실행시킬 때 사용된다. 단 이 노드는 반드시 패키지에 포함되어 있는 런치 파일이어야한다
$roscore	마스터 노드 실행
$rosclean	ROS 로그 파일 검사 및 삭제
$ catkin_make	캐킨 빌드 시스템에 기반을 둔 빌드

3. 중요 개념

1) 노드

노드(node)란, 최소 단위의 실행 가능한 프로세스를 가리키는 용어로써 하나의 실행 가능한 프로그램으로 생각하면 된다. ROS에서는 최소한의 실행 단위로 프로그램을 나누어 작업하게 된다, 각 노드는 메시지 통신으로 데이터를 주고 받는다.

예) 얼굴인식

(1) 카메라 디바이스로 raw data 받아오기 <노드 1>

(2) 얼굴만 찾아내거나 얼굴을 뺀 나머지 배경은 지우는 등의 필터 과정 <노드 2>

(3) 직원의 얼굴 데이터와 가공된 데이터와 매칭하기 <노드 3>

(4) 판단하기 <노드 4>

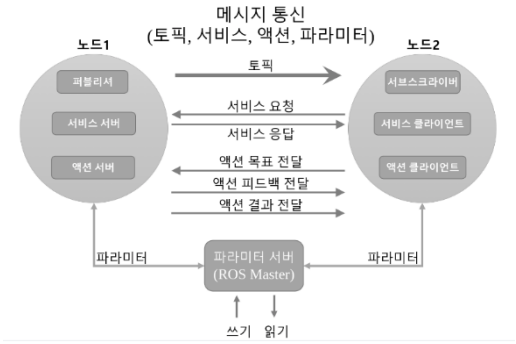
(5) 문이 열리는 등의 하드웨어 제어 <노드 5>

간단하게 5개의 과정을 거쳐 만들어지는데 5개를 각각 하나의 노드화 해서 5개의 프로그램을 개발 후에 합치는 방식이다. 이렇게 프로그램을 노드화해서 개발하면 그에 따른 장점은 유닛테스트가 가능하다는 것이다. 어느 부분에서 작동이 안되는지도 알 수 있고 노드끼리의 통신이 문제인지 노드 자체가 문제인지 등 문제가 발생하였을 때 원인을 쉽게 찾을 수 있다. ROS의 목적은 로보틱스 생태계를 구성하여 전세계적으로 로봇 개발을 해보는 것이다. 각각 노드화를 시켜놓고 공개하면 여러 사람들이 보고 이용할 수 있다. 이를 이용 후에 해당 노드를 발전시켜 다시 공유할 수 있는 등 개발자들 사이에서의 상호작용이 활발해질 수 있다

2) 패키지

패키지(Package)란 하나 이상의 노드, 노드 실행을 위한 정보 등을 묶어 놓은 것으로 위에서 말한 5개의 과정을 하나의 패키지로 묶어 얼굴인식 프로그램이 되는 것이다.

3) Message: 메시지를 통해 노드 간의 데이터를 주고 받게 된다. 메시지는 integer, floating point, boolean와 같은 변수 형태이다. 또한, 메시지 안에 메시지를 품고 있는 간단한 데이터 구조 및 메시지들의 배열과 같은 구조도 사용할 수 있다. 목적과 방식에 따라 (1)topic (2) service (3) action (4)parameter가 있다.



<ROS 메시지>

*파라미터 서버: ROS의 마스터 기능, parameter도 관리, 글로벌 변수로 외부에서 수정 가능

(1) Topic: 통신의 90%는 topic을 사용할 정도로 많이 사용되는 방식이다.

Publisher: 데이터를 보내는 측(=publish node)

Subscriber: 데이터를 받는 측(=subscribe node)

- 단뱡향, 연속적인 통신 방법

- 일방적인 데이터를 계속 보내야하는 상황일 때 사용 (예: 센서 데이터)

- Topic에 대해 1:1의 publisher, subscriber 통신도 가능하며, 목적에 따라서 1:N, N:1, N:N 통신도 가능

*오도메트리: 로봇의 위치를 추정하는 노드

<publisher, subscriber의 1:1 topic 통신>

<publisher, subscriber의 1:N topic 통신>

<topic 실습>

-Catkin_ws의 src 폴더로 이동하여 패키지 생성

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg ros_tutorials_topic message_generation std_msgs roscpp
```

- 생성한 패키지로 이동하여 다음과 같은 내용을 작성

```
$ cd ros_tutorials_topic
$ ls
```

```
include
src
CMakeLists.txt
package.xml
```

헤더 파일 폴더, 소스 코드 폴더, 빌드 설정 파일. 패키지 설정 파일

- 패키지 설정 파일 수정: ROS의 필수 설정 파일 중 하나인 package.xml은 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 기술하고 있다

패키지 설정 파일 안에는 버전, 패키지 이름, 어떤 패키지인가에 대한 설명, 적용한 라이선스, 저자(메일과 이름), maintainer(저자와 동일할 필요는 없다 연락처로서의 역할을 한다), bug tracker, 레포지토리, 빌드 툴, 사용 언어, 메시지 형태 등을 기입한다.

```
$ gedit package.xml
```

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_tutorials_topic</name>
  <version>0.1.0</version>
  <description>ROS turtorial package to learn the topic</description>
  <license>Apache 2.0</license>
  <author email="pyo@robotis.com">Yoonseok Pyo</author>
  <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer>
  <url type="website">http://www.robotis.com</url>
  <url type="repository">https://github.com/ROBOTIS-GIT/ros_tutorials.git</url>
  <url type="bugtracker">https://github.com/ROBOTIS-GIT/ros_tutorials/issues</url >
  <buildtool_depend>catkin</buildtool_depend>
  <depend>roscpp</depend>
  <depend>std_msgs</depend>
  <depend>message_generation</depend>
  <export></export >
</package>
```

- 빌드 설정 파일(CMakeLists.txt) 수정

```
$ gedit CMakeLists.txt
```

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_topic)
```

캐킨 빌드를 할 때 요구되는 구성요소 패키지이다. 의존성 패키지로 message_generation,

std_msgs, roscpp이며 이 패키지들이 존재하지 않으면 빌드 도중에 에러가 난다.

```
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs roscpp)
```

메시지 선언: MsgTutorial.msg

```
add_message_files(FILES MsgTutorial.msg)
```

의존하는 메시지를 설정하는 옵션이다.

std_msgs가 설치되어 있지 않다면 빌드 도중에 에러가 난다.

```
generate_messages(DEPENDENCIES std_msgs)
```

캐킨 패키지 옵션으로 라이브러리, 캐킨 빌드 의존성, 시스템 의존 패키지를 기술한다.

```
catkin_package(
  LIBRARIES ros_tutorials_topic
  CATKIN_DEPENDS std_msgs roscpp
)
```

인클루드 디렉터리를 설정한다

```
include_directories(${catkin_INCLUDE_DIRS})
```

topic_publisher 노드에 대한 빌드 옵션이다.

실행 파일, 타깃 링크 라이브러리, 추가 의존성 등을 설정한다.

```
add_executable(topic_publisher src/topic_publisher.cpp)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})
```

topic_subscriber 노드에 대한 빌드 옵션이다.

```
add_executable(topic_subscriber src/topic_subscriber.cpp)
add_dependencies(topic_subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_subscriber ${catkin_LIBRARIES})
```

- 메시지 파일 작성: 노드에서 사용할 메시지인 MsgTutorial.msg를 빌드할 때 포함, Time (메시지 형식), stamp (메시지 이름), int32 (메시지 형식), data (메시지 이름), 메시지 타입은 time과 int32 이외에도 bool, int8, int16, float32, string, time, duration 등의 메시지 기본, 타입과 ROS 에서 많이 사용되는 메시지를 모아놓은 common_msgs 등도 있다.

```
$ mkdir msg                → ros_tutorials_topic 패키지에 msg라는 메시지 폴더를 신규 작성
$ cd msg                   → 작성한 msg 폴더로 이동
$ gedit MsgTutorial.msg    → MsgTutorial.msg 파일 신규 작성 및 내용 수정
$ cd ..                    → ros_tutorials_topic 패키지 폴더로 이동
```

- 퍼블리셔 노드 작성

ros_tutorials_topic 패키지의 소스 폴더인 src 폴더로 이동, 소스 파일 신규 작성 및 내용 수정

```
$ cd src
$ gedit topic_publisher.cpp
```

ROS 기본 헤더파일, MsgTutorial 메시지 파일 헤더, 노드 메인 함수, 노드명 초기화, ROS 시스템과 통신을 위한 노드 핸들 선언

```
#include "ros/ros.h"
#include "ros_tutorials_topic/MsgTutorial.h"

int main(int argc, char **argv)
{
  ros::init(argc, argv, "topic_publisher");
  ros::NodeHandle nh;
```

퍼블리셔 선언, ros_tutorials_topic 패키지의 MsgTutorial 메시지 파일을 이용한 퍼블리셔 ros_tutorial_pub 를 작성한다. 토픽명은 "ros_tutorial_msg" 이며, 퍼블리셔 큐(queue) 사이즈를

100개로 설정한다는 것이다

```
ros::Publisher ros_tutorial_pub = nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_tutorial_msg", 100);
```

루프 주기를 설정한다. "10" 이라는 것은 10Hz를 말하는 것으로 0.1초 간격으로 반복된다

```
ros::Rate loop_rate(10);
```

MsgTutorial 메시지 파일 형식으로 msg 라는 메시지를 선언

```
ros_tutorials_topic::MsgTutorial msg;
```

메시지에 사용될 변수 선언

```
int count = 0;
```

현재 시간을 msg의 하위 stamp 메시지에 담기 -> count라는 변수 값을 msg의 하위 data 메시지에 담기 -> stamp.sec 메시지를 표시 -> stamp.nsec 메시지를 표시, data 메시지를 표시, 메시지를 발행하기, 위에서 정한 루프 주기에 따라 슬립에 들어가기 -> count 변수 1씩 증가

```
while (ros::ok())
{
 msg.stamp = ros::Time::now();
 msg.data  = count;
```

```
  ROS_INFO("send msg = %d", msg.stamp.sec);
  ROS_INFO("send msg = %d", msg.stamp.nsec);
  ROS_INFO("send msg = %d", msg.data);

  ros_tutorial_pub.publish(msg);

  loop_rate.sleep();

  ++count;
 }

 return 0;
}
```

- 서브스크라이브 노드 작성

패키지의 소스 폴더인 src 폴더로 이동, 소스 파일 신규 작성 및 내용 수정

```
$ roscd ros_tutorials_topic/src
$ gedit topic_subscriber.cpp
```

ROS 기본 헤더파일, MsgTutorial 메시지 파일 헤더 -> msgCallback 함수: 메시지 콜백 함수로서 밑에서 설정한 ros_tutorial_msg라는 이름의 토픽, 메시지를 수신하였을 때 동작하는 함수이다. 입력 메시지로는 ros_tutorials_topic 패키지의 MsgTutorial 메시지를 받도록 되어있다

stamp.sec 메시지를 표시, stamp.nsec 메시지를 표시, data 메시지를 표시한다

```
#include "ros/ros.h"
#include "ros_tutorials_topic/MsgTutorial.h"

void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{
  ROS_INFO("recieve msg = %d", msg->stamp.sec);
  ROS_INFO("recieve msg = %d", msg->stamp.nsec);
  ROS_INFO("recieve msg = %d", msg->data);
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "topic_subscriber");

  ros::NodeHandle nh;

ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);

  ros::spin();

  return 0;
}
```

- ROS 노드 빌드

```
$ cd ~/catkin_ws          → catkin 폴더로 이동
$ catkin_make             → catkin 빌드 실행
```

- 퍼블리셔 실행

```
$ rosrun ros_tutorials_topic topic_publisher
```

- 서브스크라이버 실행

```
$ rosrun ros_tutorials_topic topic_subscriber
```

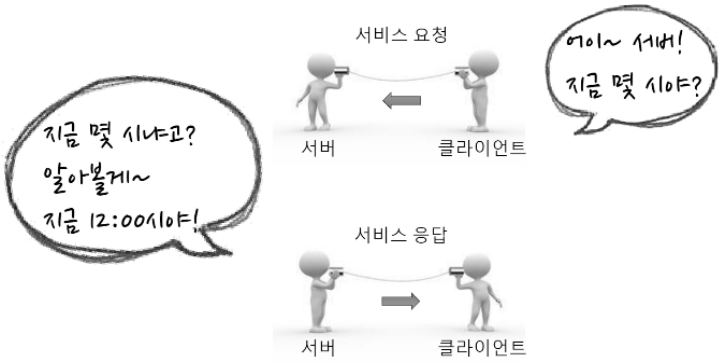
(2) Service: topic 다음으로 종종 사용되는 방식으로 5% 이상의 사용 비율을 차지한다.

클라이언트: 서비스를 요청하는 측

서버: 서비스를 응답하는 측

- 양방향, 일회성(여러 번 통신하려면 재접속 필요)

예) 클라이언트가 서버에게 로봇의 위치 명령을 내림 -> 로봇의 위치가 이동 -> 로봇(서버)이 클라이언트에게 도착했음을 알림



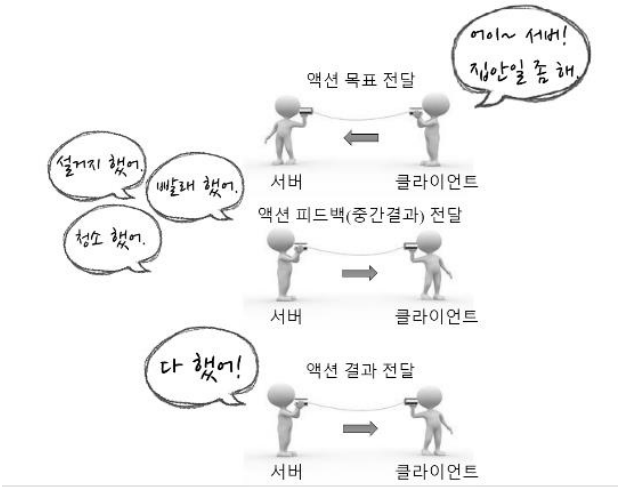
<Service 통신의 예>

(3) Action: 5% 이하의 사용 비율을 가지며 task 프로그램일수록 action을 사용한다

- 클라이언트가 서버에게 액션 명령을 전달

- 다른 통신 방식과 다르게 액션은 각 단계가 진행될 때마다 중간결과를 클라이언트에게 전달

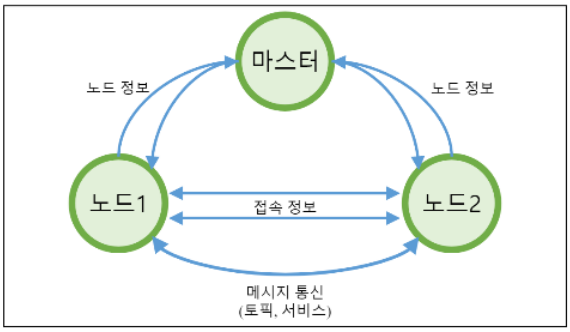
- 모든 명령을 다 수행한 후 서버가 클라이언트에게 액션 결과를 전달



<Action 통신>

8. 메시지 통신

ROS에서 가장 기본이 되는 기술적 포인트: 노드간의 메시지 통신



1) 마스터 구동: XMLRPC(XML-Remote Procedure Call)

- $ roscore: 네트워크 끄기 전까지 roscore를 활성화하여 마스터를 구동

- 마스터는 노드 정보를 관리하는데 노드간의 통신을 연결시켜주는 매개체 역할

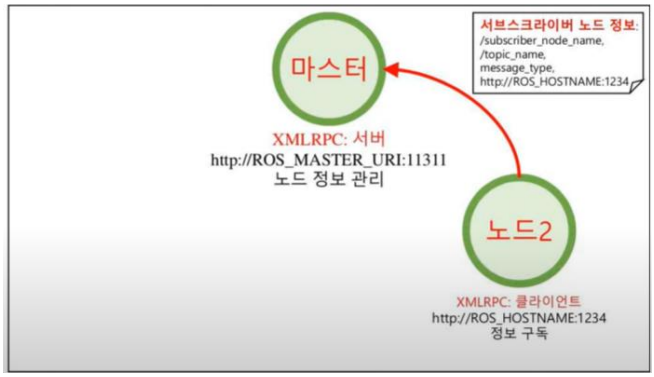
2) Subscriber node 구동

- $ rosrun 패키지이름 노드이름

- subscriber node 정보를 마스터에게 전달

- 이때 정보는 노드 이름, 토픽 이름, 메시지 타입, IP번호와 port 번호를 전달

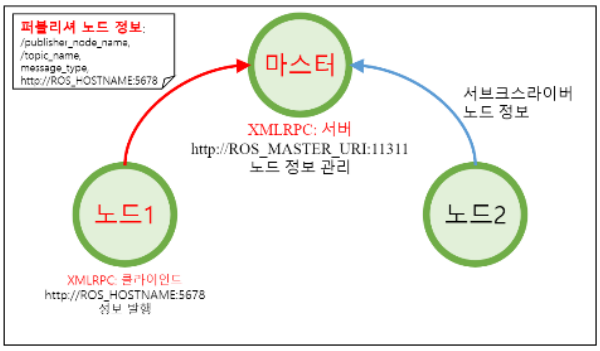
- IP 번호와 port 번호는 다른 노드들과 접속시키기 위함



3) Publisher node 구동

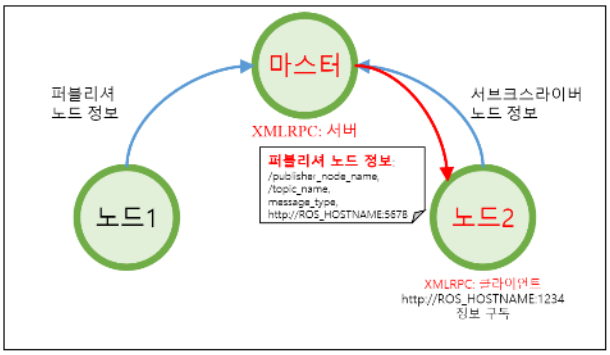
- $ rosrun 패키지이름 노드이름

- 마스터는 노드 1로부터 topic 형식, message 형식을 받고 노드 2로부터 받은 정보와 비교하여 정보가 같으면 메시지 전달



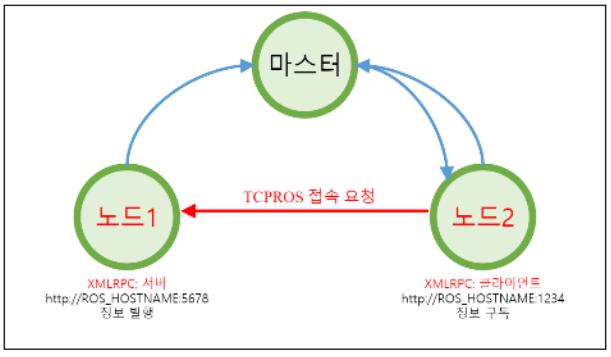
4) publisher의 정보 알림

- 마스터는 subscriber node에게 새로운 publisher 정보를 알린다



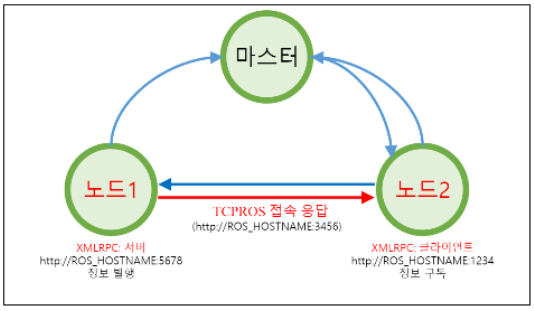
5) publisher node에 접속 요청

마스터로부터 받은 publisher 정보를 이용하여 TCPROS에 접속을 요청한다. 해당 서버에 접속을 요청하면 마스터를 매개로 통신하는 것이 아닌 노드간의 직접적인 통신이 가능해진다.



6) Subscriber node에 접속 응답

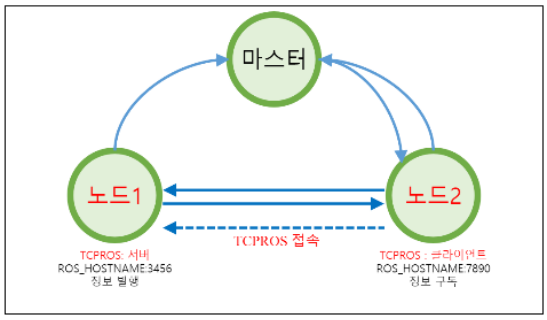
접속 응답에 해당되는 자신의 TCP URI 주소와 포트 번호를 전송한다



7) TCP 접속

TCPROS를 이용하여 publisher node와 직접 연결한다

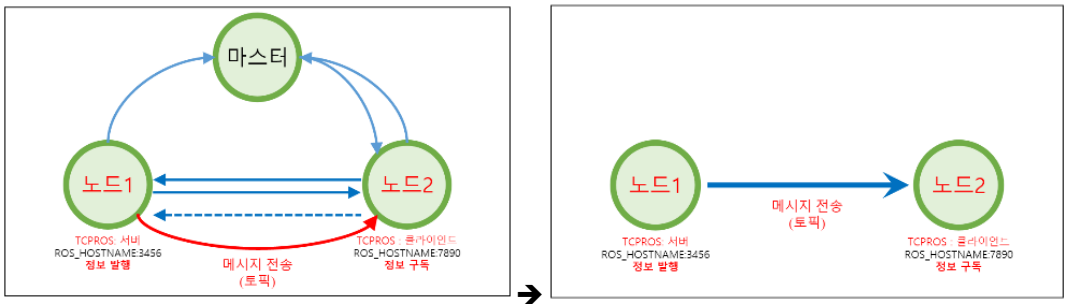
*서버의 이름을 주목하여 보면 XMLRPC에서 TCPROS로 변경되었고 마스터를 통한 통신이 아닌 직접 통신으로 바뀜.



8-1) 메시지 전송 (Topic)

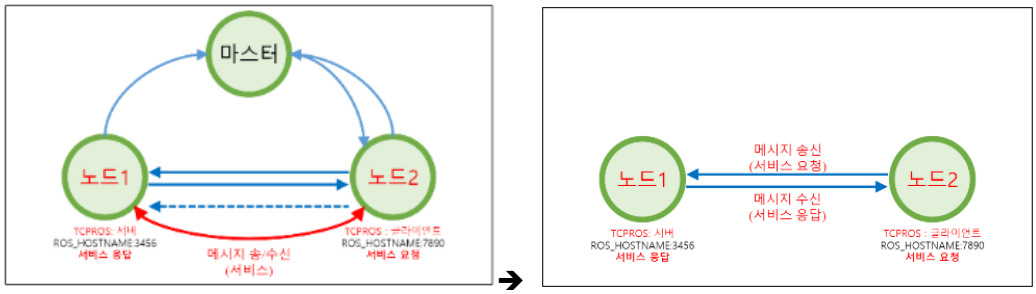
발행자 노드는 subscriber node에게 메시지 전송

접속이 끊기지 않는 이상 연속적으로 메시지를 전송한다(Topic 의 특징)



8-2) 메시지 전송 (Service의 요청 및 응답)

- 서비스 요청 및 서비스 응답이 1회 수행되고 서로간의 접속을 끊는다, 일회성



9) 메시지 통신을 이해한 상태에서 6-3의 turtlesim을 적용해보자

(1) roscore 실행

(2) rosrun turtlesim(패키지 이름) turtlesim_node(노드 이름)

Subscriber node 정보 전달

(3) rosrun turtlesim(패키지 이름) turtle_teleop_key(노드 이름)

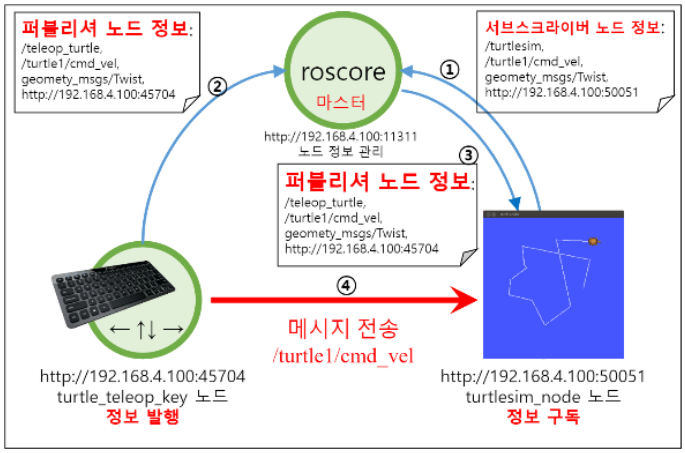
Publisher node 정보 전달

(4) 마스터는 정보 매칭 후 Subscriber node에게 Publisher 정보를 전달 후 접속 요청

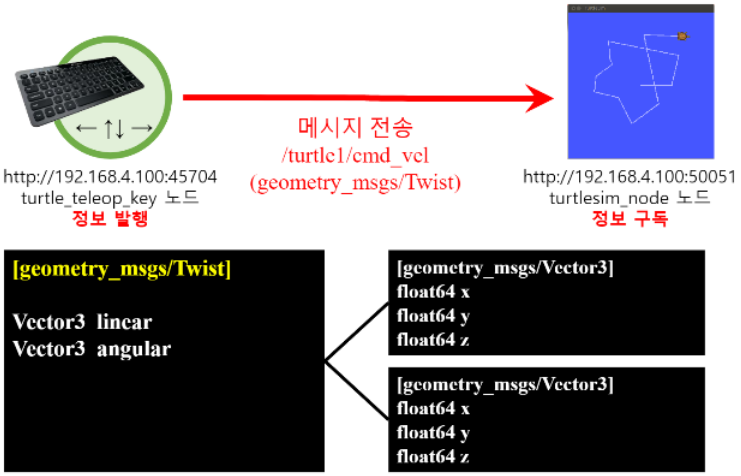
(5) 메시지 전송

Publisher node의 키보드 값이 전달되는 것이 아니라 해당되는 회전속도, 병진속도가 geometry_msgs/Twist의 형태로 메시지 전송

전송 받은 데이터에 맞게 프로세싱하게 됨



<ROS 메시지 구조>



< ROS 메시지 (예: geometry_msgs/Twist)>

* 네임(name)

하나의 프로그램을 만들기 위해서는 여러 노드가 필요하다. 하나의 OS로 같은 로봇 2대를 제어하게 되는 경우, 동일한 노드를 2번 실행하게 된다. 하지만 같은 노드 2개를 동시에 실행하지 못하기 때문에 네임을 통해서 같은 노드라도 여러 번 실행할 수 있게 한다.

네임

- 노드, 메시지(토픽, 서비스, 액션, 파라미터)가 가지는 고유의 식별자

- ROS는 그래프(graph)라는 추상 데이터 형태(abstract data type)를 지원

- Global: 문자 없이 네임을 바로 쓰거나 네임 앞에 슬래쉬(/)를 붙임

- Private: 네임 앞에 틸트(~)를 붙임

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

* 좌표 변환(TF, transform)

- 각 조인트(joint)들의 상대 좌표 변환

트리(tree)의 형태로 조인트들간의 관계도를 표시함

*조인트(joint): 로봇을 구성하는 하나의 구성요소를 링크라고 하고, 링크와 링크를 연결하는 연결부를 조인트라고 함.

* 클라이언트 라이브러리(client library)

다양한 프로그래밍 언어 지원

- roscpp, rospy, roslisp

- rosjava, roscs, roseus, rosgo, roshask, rosnodejs, RobotOS.jl, roslua, PhaROS, rosR, rosruby, Unreal-Ros-Plugin

- MATLAB for ROS

- LabVIEW for ROS

10. ROS 도구

1) RViz (ROS Visualization Tool)

ROS의 3D 시각화툴로 센서 데이터의 시각화, 레이저 거리 센서의 거리 데이터, RealSense, Kinect, Xtion 등의 Depth Camera의 포인트 클라우드 데이터, 카메라의 영상 데이터, IMU 센서의 관성 데이터 등을 시각적으로 나타낼 수 있다

로봇이 이동할 맵(환경)을 띄울 수 있고 로봇을 위치시켜 도착지점까지의 경로, 속도, 등등을 확인해보는 툴로써 작용할 수 있다

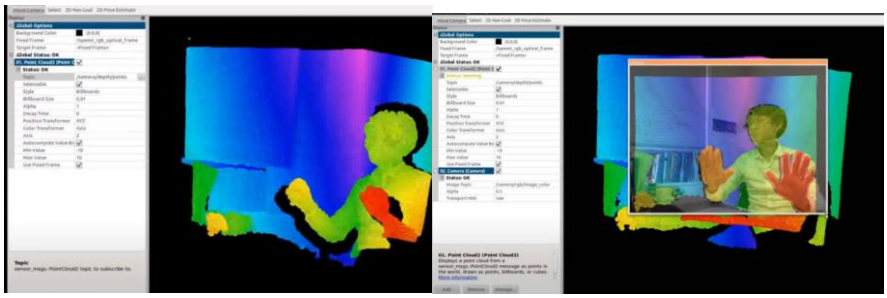
로봇암을 RViz에 올려놓고 잘 만들어졌는지. 경로대로 로봇을 움직이는 것까지 확인 가능하다

휴머노이드 로봇의 경우 보행, 다음 디딤발의 위치까지 계산 가능하다

(1) Depth 정보

- RViz를 통해 확인이 가능하다. 빨간색은 거리상 가까울수록 뜨고, 파란색은 거리상 멀수록 뜬다.

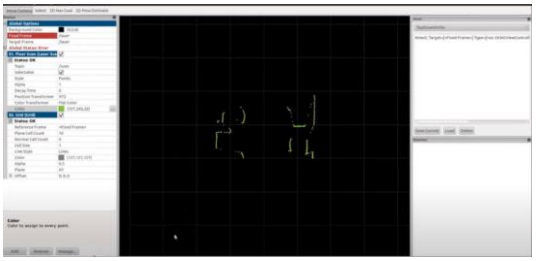
- 또한 RGB데이터와 Depth의 합성어로 RGBD데이터도 존재하여 거리와 형태를 모두 표현한다.



<RGB, RGBD 데이터의 예시>

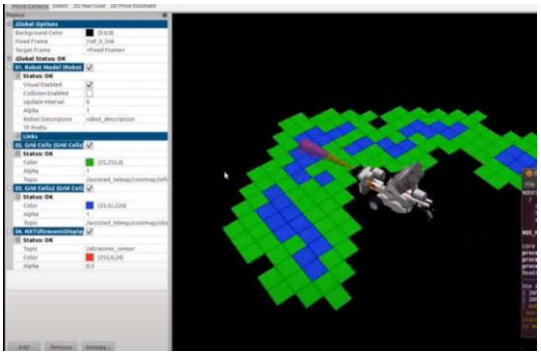
(2) 레이저 기반 센서의 거리측정

- 레이저 기반의 센서를 사용하므로 물체의 정확한 거리와 위치를 파악할 수 있다



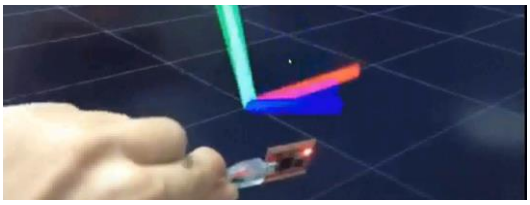
(3) 초음파 센서의 거리측정

- 초음파 센서를 사용하기 때문에 정확한 거리 측정보다는 해당 영역을 확인하는데 이용된다



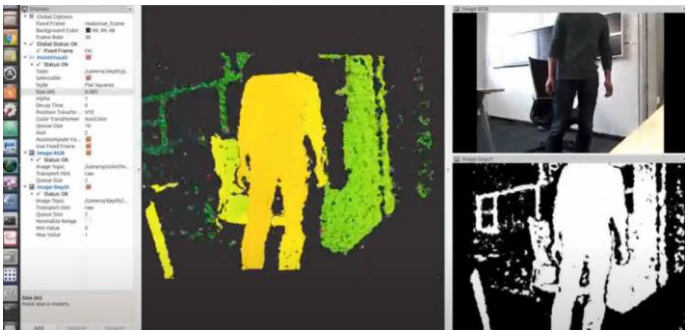
(4) IMU센서의 관성값

- 드론 등의 로봇의 자세나 균형을 잡기 위해 사용된다



(5) Real Sense의 point cloud와 color, depth 영상

- 영상을 통해 ir, color, Depth 정보를 확인할 수 있다

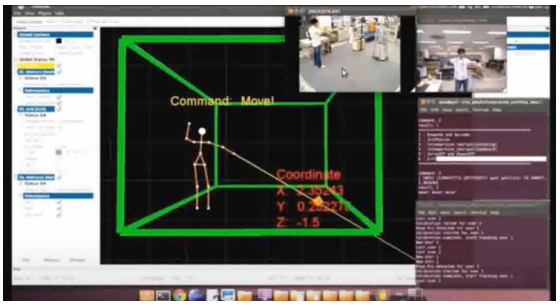


<depth 정보와 color 정보, 적외선 센서를 이용한 거리 정보>

(6) 사람의 골격과 지시 방향 표시

- raw data를 이용해 2차적인 가공을 하여 결과값을 도출할 수 있고 골격을 통해서 사람이 가리키

고 있는 방향을 계산하여 구할 수 있다.



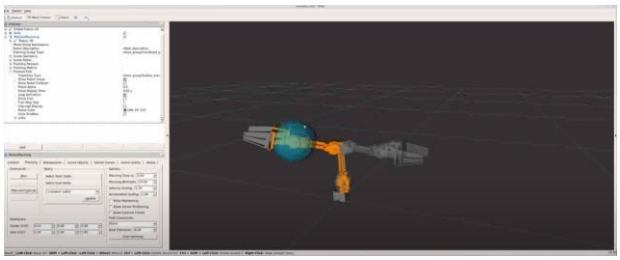
<좌표와 선을 이용한 사람의 형태를 선으로 표현>

(7) 환경 모델, 로봇 모델, 경로까지 RViz에서 구현 가능

- 환경을 모델링하여 실제로 로봇이 처할 환경까지 구현이 가능하고 환경과 로봇을 이용하여 경로, 네비게이션까지 시뮬레이션을 할 수 있다

(8) 매니퓰레이션

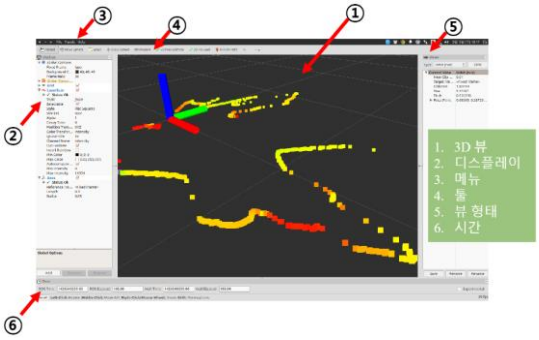
- 매니퓰레이션을 RViz상에서 시뮬레이션을 할 수 있고 실제로 로봇이 움직이는 것을 볼 수 있다



* RViz 설치: $ sudo apt install ros-kinetic-rviz

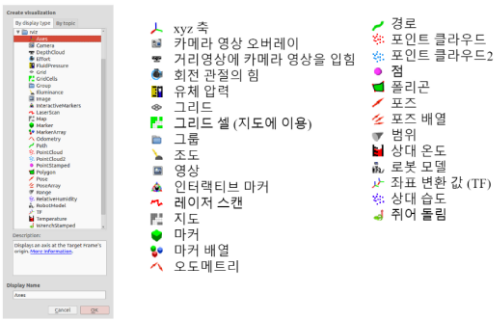
RViz 실행: $ rosrun rviz rviz 또는 $ rviz

* RViz의 화면 구성



* RViz의 디스플레이 종류

- 2번의 add를 클릭하여 원하는 정보를 추가하여 얻을 수 있다



2) RQT (GUI tool box); 플러그인 방식의 ROS의 종합 GUI 툴

- rqt_bag, rqt_plot, rqt_graph 등을 플러그 인으로 하는 ROS의 종합 GUI 툴로써 사용가능

- rqt는 Qt로 개발되어 있어 유저들이 자유롭게 플러그인하여 개발을 추가 가능

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

* rqt 실행: $ rqt

* rqt 플러그인

(1) 액션: action type browser (action 타입의 데이터 구조 확인)

(2) 구성: dynamic reconfigure (노드들에서 제공하는 설정값 변경을 위한 GUI 설정값 변경), launch (roslaunch의 gui 버전)

(3) 내성: node gragh (구동중인 노드들의 관계도 및 메시지의 흐름을 확인하기위한 그래프 뷰), package graph (노드의 의존 관계를 표시하는 그래프 뷰), process monitor (실행중인 노드들의 cpu 사용률, 메모리 사용률, 스레드수 등을 확인)

(4) 로깅: bag(ROS 데이터 로깅), console(노드들에서 발생되는 경고, 에러 등의 메시지 확인), looger level(ROS의 debug, info. Warn, error 로거 정보를 선택하여 표시)

(5) 다양한 툴: python console, shell, web

(6) 로봇: 사용하는 로봇에 따라 계기판 등의 플러그인을 추가 가능

(7) 로봇툴: controller manager(컨트롤러 제어에 필요한 플러그인), diagnostic viewer(로봇 디바이스 및 에러 확인), moveit! Monitor(로봇 팔 계획에 사용되는 moveit! 데이터 확인), robot steering(로봇 조정 gui 툴, 원격 조정에서 이 gui툴을 이용하여 로봇 조종), runtime monitor(실시간으로 노드들에서 발생되는 에러 및 경고를 확인)

(8) 서비스: service caller(구동중인 서비스 서버에 접속하여 서비스를 요청), service type browser(서비스 타입의 데이터 구조 확인)

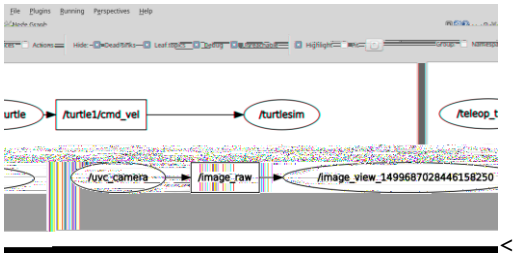
(9) 토픽: easy message publisher(토픽을 gui 환경에서 발행), topic publisher(토픽을 생성하여 발행), topic type browser(토픽 타입의 데이터 구조 확인), topic monitor(사용자가 선택한 토픽의 정보를 확인)

(10) 시각화: image view(카메라의 영상 데이터를 확인), navigation viewer(로봇 네비게이션의 위치 및 목표지점 확인), plot(2차원 데이터 플롯 gui 플러그인, 2차원 데이터의 도식화), pose view(현재 TF의 위치 및 모델의 위치 표시), RViz(3차원 시각화 툴인 RViz 플러그인), TF tree(tf 관계를 트리로 나타내는 그래프 뷰)

* rqt 실습

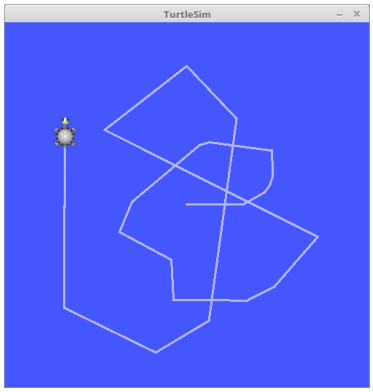
(1) $ rosrun uvc_camara uvc_camera_node -> $ rqt_image_view: 카메라를 이용한 대상의 이미지 정보를 rqt에서 확인할 수 있다

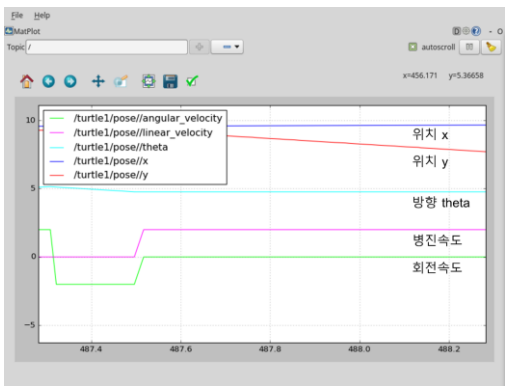
(2) $ rosrun turtlesim turtlesim_node -> $ rosrun turtlesim turtle_teleop_key -> $ rosrun uvc_camara uvc_camera_node -> $ rosrun image_view image_view image:=image_raw -> $ rqt_graph: 그래프를 이용하여 정보가 어떻게 전달이 되는지 그 경로를 알 수 있다.



<그래프를 이용한 정보 이동 경로 표시>

(3) $ rosrun turtlesim turtlesim_node -> $ rosrun turtlesim turtle_teleop_key -> $ rqt_plot /turtle1/pose: 축마다의 위치 정보, 방향, 병진속도, 회전속도를 실시간으로 볼 수 있다





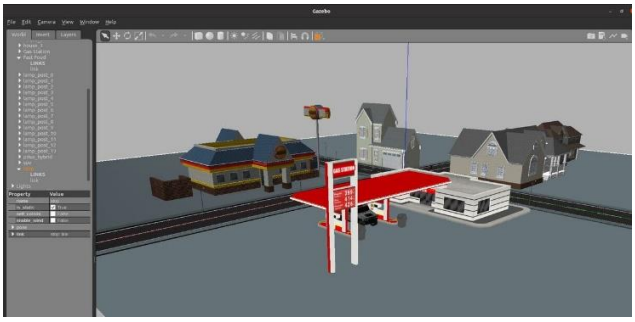
(4) $ rosrun uvc_camera uvc_camera_node -> $ rosbag record/image_raw -> $ rqt_bag: 메시지 녹화, 저장 가능, bag로 재생 가능

3) Gazebo

- Gazebo는 로봇 개발에 필요한 3차우너 시뮬레이션을 위한 로봇, 센서, 환경 모델 등을 지원하고 물리 엔진을 탑재하여 실제와 근사한 결과를 얻을 수 있는 3차원 시뮬레이터이다.

- 이는 최근에 나온 오픈 진영 시뮬레이터 중 가장 좋은 평가를 받고 있고, 미국 DARPA Robotics Challenge의 공식 시뮬레이터로 선정되어 개발에 더욱 박차를 가하고 있는 상황이다.

- ROS에서는 그 태생이 player/stage, gazebo를 기본 시뮬레이터로 사용하여서 ROS와의 호환성도 좋다



<환경 모델의 3차원 시뮬레이터>

10. roslaunch 사용법

- rosrun: 하나의 노드를 실행하는 명령어

- roslaunch: 하나 이상의 정해진 노드를 실행시킬 수 있다.

- 노드를 실행할 때 패키지의 매개변수나 노드 이름 변경, 노드 네임 스페이스 설정, ROS_ROOT 및 ROS_PACKAGE_PATH 설정, 환경 변수 설정 등의 옵션을 붙일 수 있는 명령어

- roslaunch는 '*.launch'라는 파일을 사용하여 실행 노드를 설정하는데 이는 XML 기반이며, 태그별 옵션을 제공한다

- 실행 명령어는 "roslaunch [패키지명] [roslaunch 파일]"이다.

1) 활용: 이전에 작성한 topic_publisher와 topic_subscriber 노드의 이름을 바꾸어서 실행. (1) *.launch 파일 작성. Roslaunch에 사용되는 파일은 *.launch 파일명을 가지며 해당 폴더에 launch 라는 폴더에 생성. 다음 명령어 대로 폴더 생성 후 union.launch라는 파일을 생성해보자

```
$ roscd ros_tutorials_topic
$ mkdir launch
$ cd launch
$ gedit union.launch
```

2) 사용법: <launch> 태그 안에는 명령어로 노드를 실행할 때 필요한 태그들이 기술된다. <node>는 roslaunch로 실행할 노드를 기술하게 된다. 옵션으로는 pkg(패키지 이름), type(실제 실행할 노드의 이름), name(위 type에 해당하는 노드가 실행될 때 붙여지는 이름, 일반적으로는 type과 같게 설정하지만 필요에 따라 실행할 때 이름을 변경하도록 설정 가능)이 있다.

```
<launch>
  <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher1"/>
  <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber1"/>
  <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher2"/>
  <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber2"/>
</launch>
```

Union.launch 파일 실행

```
$ roslaunch ros_tutorials_topic union.launch --screen
```

실행 결과

```
$ rosnode list
/topic_publisher1
/topic_publisher2
/topic_subscriber1
/topic_subscriber2
/rosout
```

결과적으로는 topic_publisher 노드가 topic_publisher1과 topic_publisher2로 이름이 바뀌어 두 개의 노드가 실행되었고 topic_subscriber 노드도 마찬가지이다.

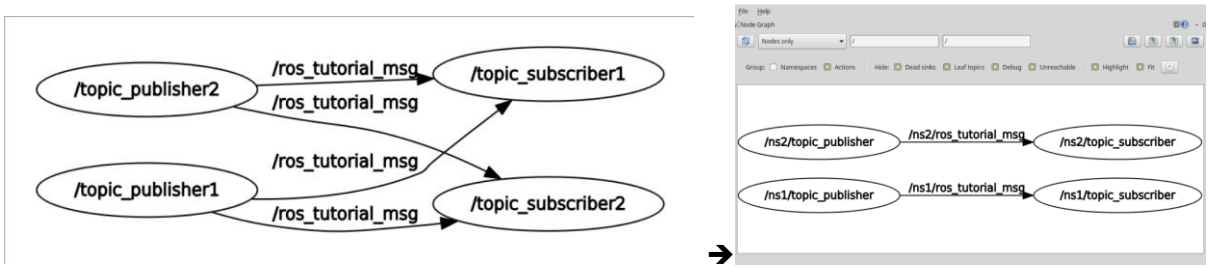
문제는 "퍼블리쉬 노드와 서브스크라이버 노드를 각각 두 개씩 구동하여 서로 별도의 메시지 통신하게 한다"는 처음 의도와는 다르게 rqt_graph를 통해 보면 서로의 메시지를 모두 서브스크라이브하고 있다는 것이다. 이는 단순히 실행되는 노드의 이름만을 변경해주었을 뿐 사용되는 메시지의 이름을 바꿔주지 않았기 때문이다. 문제를 다른 roslaunch 네임스페이스 태그를 사용하여 해결해보자

Union.launch를 수정해주면

```
$ roscd ros_tutorials_service/launch
$ gedit union.launch
```

```
<launch>
  <group ns="ns1">
    <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
    <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber"/>
  </group>
  <group ns="ns2">
    <node pkg="ros_tutorials_topic" type="topic_publisher" name="topic_publisher"/>
    <node pkg="ros_tutorials_topic" type="topic_subscriber" name="topic_subscriber"/>
  </group>
</launch>
```

변경된 노드들의 모습을 볼 수 있다.



3) launch 태그

<launch> roslaunch 구문의 시작과 끝을 알림

<node> 노드 실행에 대한 태그로 패키지, 노드명, 실행명을 변경가능

<machine> 노드를 실행하는 pc이름, address, ros-root, ros-package-path 등을 설정

<include> 다른 패키지나 같은 패키지에 속해 있는 다른 launch를 불러와 하나의 launch파일처럼 실행

<remap> 노드 이름, 토픽 이름 등의 노드에서 사용 중인 ROS 변수의 이름을 변경

<param> 매개변수 이름, 타이프, 값 등을 설정

<arg> launch 파일 내에 변수를 정의할 수 있어서 아래와 같이 실행할 때 매개변수를 변경

<launch>

<arg name="update_ period" default="10" />

<param name="timing" value="$(arg update_ period)"/>

</launch>

roslaunch my_package my_package.launch update_ period:=30

6. ROS 개발환경 구축

1) ROS 설치 방법

http://wiki.ros.org/kinetic/Installation/Ubuntu

2) ROS 환경 설정

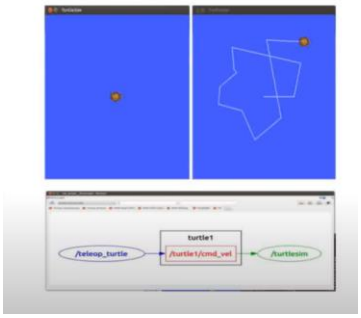
http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

```
$ gedit ~/.bashrc 또는 $ eb

alias eb ='gedit ~/.bashrc'
alias sb ='source ~/.bashrc'
alias agi='sudo apt-get install'
alias gs ='git status'
alias gp ='git pull'
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
```

3) ROS 동작 테스트

- turtlesim 패키지

- roscore ->rosrun turtlesim turtlesim_node -> rosrun turtlesim turtle_teleop_key -> rosrun rqt_gragh



<실행결과>

4) ROS에서 사용 가능한 통합개발환경

- Qtcreator: rqt 플러그인 및 GUI 개발 용이

- Visual Studio Code: 간단한 텍스트 편집기 지향, 빠름

- Eclipse: 많은 사람들이 사용하는 익숙한 통합개발환경

5) 환경설정

환경변수 alias는 주로 명령어나 경로 등을 더 간단하게 표현하고 사용하기 위해 환경 변수에 별명을 지정하는 것을 의미한다. 이를 통해 사용자는 긴 경로나 복잡한 명령어를 간략하게 입력하여 사용할 수 있다.

.bashrc 환경 변수 파일 수정

$ sudo gedit ~/.bashrc

가장 아래쪽에 다음과 같이 입력

```
alias gb='cd && gedit .bashrc'
alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gb='gedit ~/.bashrc'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'
alias cb='source devel/setup.bash'
alias depinstall='rosdep update && rosdep install --from-paths . --ignore-src -r -y'
alias rn='sudo systemctl restart NetworkManager'
alias symlink='sudo udevadm control --reload-rules && sudo udevadm trigger'
alias killg='killall -9 gzserver && killall -9 gzclient && killall -9 rosmaster'
```