ROS(Robot Operation System) 설치

우선 ROS란? http://wiki.ros.org/ko/ROS/Introduction 참고

ROS는 Melodic 버전을 설치하였습니다. 다음의 공식가이드를 따라 설치하면 됩니다.
공식 가이드 : http://wiki.ros.org/melodic/Installation/Ubuntu

1. Setup your sources.list (pc가 packages.ros.org로부터 ROS관련 소프트웨어를 받을 수 있도록 설정합니다.)
-> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

2. Set up your keys (다운로드 받는 패키지가 신뢰 할 수 있다는 것을 확인하기 위해 key값을 설정해 줍니다.)
-> sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

3. ROS 툴 설치
-> sudo apt-get update
Desktop-Full Install: (Recommended) : ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators and 2D/3D perception
-> sudo apt install ros-melodic-desktop-full

4. Initialize rosdep (rosdep은 ROS 구동에 필요한 요소들을 쉽게 설치할 수 있도록 합니다. / ROS를 돌리기 위한 설정))
-> sudo rosdep init
-> rosdep update

5. Environment setup
-> echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
-> source ~/.bashrc

6. Dependencies for building packages
-> sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential

Lidar 설정

Lidar는 YDLIDAR의 X2모델을 사용했습니다. YDLIDAR 공식 홈페이지와 github에서 공식메뉴얼과 예제를 다운받아 확인 할 수 있습니다.

YDLIDAR 공식홈페이지 : http://www.ydlidar.com/download
github 페이지 : 0https://github.com/YDLIDAR/ydlidar_ros

YDLIDAR 공식 홈페이지에서 Ros.zip 파일 내의 pdf 파일을 통해 RosManual을 확인 할 수 있습니다.

1. ROS Workspace를 만들어 줍니다.
   (catkin을 사용하는데 이때 catkin은 ROS의 빌드 시스템을 말합니다. ROS빌드 시스템은 기본적으로 CMake(Cross Platform Make)를 이용하고 있어서 패키지 폴더에 CMakeList.txt 의 파일에 빌드 환경을 기술하고 있습니다.
   ROS에서는 CMake를 ROS에 맞도록 수정하여 ROS에 특화된 catkin 빌드 환경을 만들었습니다. catkin 빌드 시스템은 ROS와 관련된 빌드, 패키지 관리, 패키지간의 의존 관계 등을 편리하게 사용할 수 있습니다.)

-> mkdir -p ~/catkin_ws/src
-> cd ~/catkin_ws/src
-> catkin_init_workspace (작업공간을 만드는 명령어)
-> cd ~/catkin_ws && catkin_make (빌드 명령어)

2. catkin 환경변수를 추가해줍니다.

-> echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
-> source ~/.bashrc

3. 생성한 ~/catkin_ws/src 디렉토리로 이동하고 YDLIDAR의 공식 레포지토리를 clone 합니다

-> cd src
-> git clone https://github.com/YDLIDAR/ydlidar_ros

4. ydlidar_ros 폴더로 이동후 X2 제품에 해당하는 S2 브랜치로 이동합니다.(git 버전관리와 관련된 것으로 X2에 해당하는 버전인 S2로 설정시 이에 맞춰 레포지토리의 내용들이 모두 바뀜)

-> cd ydlidar_ros
-> git checkout s2

5. catkin_ws 폴더로 이동 후 catkin_make를 실행합니다. ydlidar_node와 ydlidar_client가 컴파일, 생성 됩니다.

-> cd ../..
-> catkin_make

6. ydlidar의 시리얼 포트 alias로 /dev/ydlidar 를 만듭니다(초기 환경설정 스크립트 실행, 스크립트 실행 후 lidar와 jetson을 연결한 usb포트를 한번 빼고 다시 연결 할 것)
-> roscd ydlidar/startup
-> sudo chmod 0777 *
-> sudo sh initenv.sh

Lidar 예제 실행

1. 터미널에서 거리 측정값 출력하기.

우선적으로 roscore를 먼저 실행한다. 이는 ROS를 사용할 때 제일 먼저 실행하게 됩니다.
- roscore : ROS 마스터를 실행하는 명령어(roscore)입니다. 같은 네트워크라면 다른 컴퓨터에서 실행해도 됩니다. 단 멀티 ROS코어를 지원하는 특수한 경우를 제외하고는 ROS코어는 동일 네트워크에서 하나만 구동되게 되어있습니다.
- node : ROS에서 최소 단위의 실행 프로세서를 말합니다. ROS에서는 하나의 기능을 하나의 노드에 구현하도록 권고하고 있습니다.
- Master : 노드와 노드 사이의 연결 및 메시지 통신을 위한 네임 서버입니다. 마스터가 없으면 ROS노드간 메시지, 토픽등의 통신을 할 수 없습니다.
- roslaunch : roslaunch는 한번에 복수개의 노드를 실행하는 명령어입니다.
- rosrun : rosrun은 ROS의 기본적인 실행 명령어입니다. 패키지에서 하나의 노드를 실행하는데 사용됩니다.

터미널에서 다음 명령어 실행(linear_node 실행)
-> roslaunch ydlidar lidar.launch

다른 터미널에서 다음 명령어 실행(lidar_client 실행)
-> rosrun ydlidar ydlidar_client

(lidar_node와 lidar_client의 소스코드는 ~/catkin_ws/src/ydlidar_ros/src 에 있습니다.)

2. rviz로 거리 측정 결과 시각화하기

우선적으로 roscore를 실행한다.

터미널에서 다음 명령어를 실행하면 rviz가 자동 실행되고 측정결과를 시각화하여 보여줍니다.
-> roslaunch ydlidar lidar_view.launch

Lidar 이용 Cartographer 실행

과정은 다음 사이트를 바탕으로 하였습니다.

https://github.com/msadowski/x2_cartographer

1. 이전과 같이 디렉토리를 임의로 만들어줍니다.

-> mkdir -p ~/catkin_ws/src

-> cd ~/catkin_ws/src

-> catkin_init_workspace **(작업공간을 만드는 명령어)**

-> cd ~/catkin_ws && catkin_make **(빌드 명령어)**

2. src 디렉토리에 다음 cartographer 레포지토리를 받아줍니다.

-> cd src

-> git clone https://github.com/msadowski/x2_cartographer.git

3. 다운받은 x2_cartographer 내에 ydlidar 레포지토리를 받아줍니다.

-> cd x2_cartographer

-> rm -rf ydlidar (기존 생성된 빈 폴더 삭제)

-> git clone https://github.com/YDLIDAR/ydlidar_ros.git ydlidar

4. 다운받은 ydlidar를 최신의 버전으로 바꿔줍니다.

-> cd ydlidar

-> git checkout s2

버전을 바꾸지 않을 경우 최종실행 시 다음과 같은 에러가 발생하게 됩니다.

5. 다음의 과정을 거쳐 실행합니다.

-> cd ../..

-> catkin_make (다운 파일들 build)

-> rosdep install --from-paths src --ignore-src --rosdistro melodic -r -y

(실행에 필요한 의존성 패키지를 설치해주며 필요로 하는 패키지는 package.yaml에서 확인이 가능하다.
이렇게 설치된 패키지는 /opt/ros/melodic/share 에서 확인이 가능하다.)

-> source devel/setup.bash

-> roslaunch carto_mapper mapper.launch (실행)

Cartographer, ROS 관련

1. Cartographer를 통해 생성한 이미지 저장

이때 map_server를 이용하게 되는데 관련 패키지가 없을 경우 설치해주어야 한다.

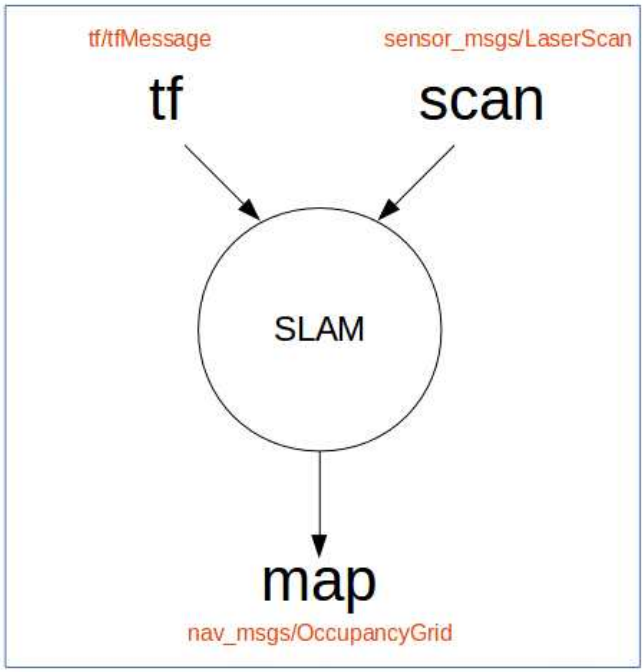
-> sudo apt-get install ros-melodic-map-server

-> rosrun map_server map_saver -f map

(map이라는 이름의 map.yaml(정보파일), map.pgm 두 개의 파일 현재 위치에 생성된다.)

2. 지도작성에는 다음 두 가지가 필요하다.

- 거리값 : ROS에서는 Scan이라는 이름으로 불림
- 위치값 : 이동량인 오도메트리(odometry)에 의존, 위치정보는 의존관계에 따라 바뀌기 때문에 tf라는 이름으로 불린다.

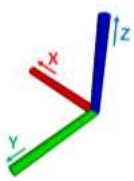


3. ROS 표준단위, 규약

다음 참고 : https://www.ros.org/reps/rep-0103.html

ROS 프로그래밍 전에 알아둬야 할 사항

- **표준 단위**
  - SI 단위 사용



Quantity	Unit
angle	radian
frequency	hertz
force	newton
power	watt
voltage	volt

Quantity	Unit
length	meter
mass	kilogram
time	second
current	ampere
temperature	celsius

- **좌표 표현 방식**
  - x: forward, y: left, z: up
  - 오른손 법칙

- **프로그래밍 규칙**

대상	명명 규칙	예제
패키지	under_scored	Ex) first_ros_package
토픽, 서비스	under_scored	Ex) raw_image
파일	under_scored	Ex) turtlebot3_fake.cpp
네임스페이스	under_scored	Ex) ros_awesome_package
변수	under_scored	Ex) string table_name;
타입	CamelCased	Ex) typedef int32_t PropertiesNumber;
클래스	CamelCased	Ex) class UrlTable
구조체	CamelCased	Ex) struct UrlTableProperties
열거형	CamelCased	Ex) enum ChoiceNumber
함수	camelCased	Ex) addTableEntry();
메소드	camelCased	Ex) void setNumEntries(int32_t num_entries)
상수	ALL_CAPITALS	Ex) const uint8_t DAYS_IN_A_WEEK = 7;
매크로	ALL_CAPITALS	Ex) #define PI_ROUNDED 3.0

4. ROS 기본

다음 참고 : https://github.com/robotpilot/ros-seminar

ROS 용어

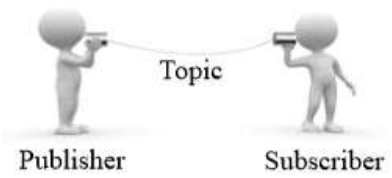
- Node
- 최소 단위의 실행 가능한 프로세스를 가리키는 용어로써 하나의 실행 가능한 프로그램으로 생각하면 된다. ROS 에서는 최소한의 실행단위로 프로그램을 나누어 작업하게 된다. 각 노드는 메시지 통신으로 데이터를 주고 받는다.

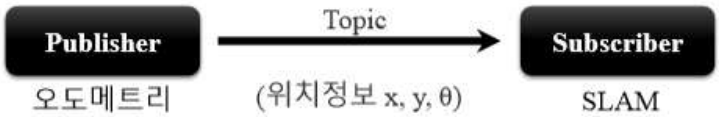
- Package
- 하나 이상의 노드, 노드 실행을 위한 정보 등을 묶어 놓은 것. 또한, 패키지의 묶음을 메타패키지라 하여 따로 분리한다.

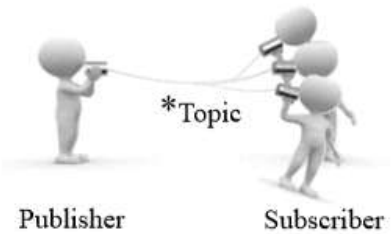
- Message
- 메시지를 통해 노드간의 데이터를 주고받게 된다. 메시지는 integer, floating point, boolean 와 같은 변수형태이다. 또한, 메시지 안에 메시지를 품고 있는 간단한 데이터 구조 및 메시지들의 배열과 같은 구조도 사용할 수 있다.

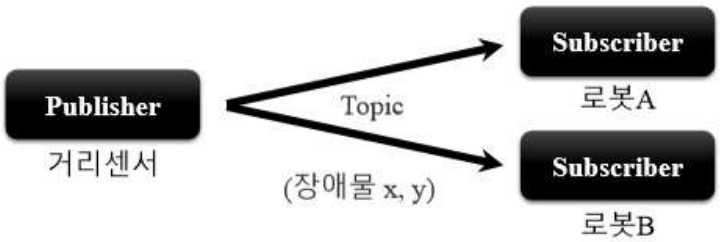
모든 프로그램은 노드 단위로 짠다.

ROS 용어 Topic, Publisher, Subscriber





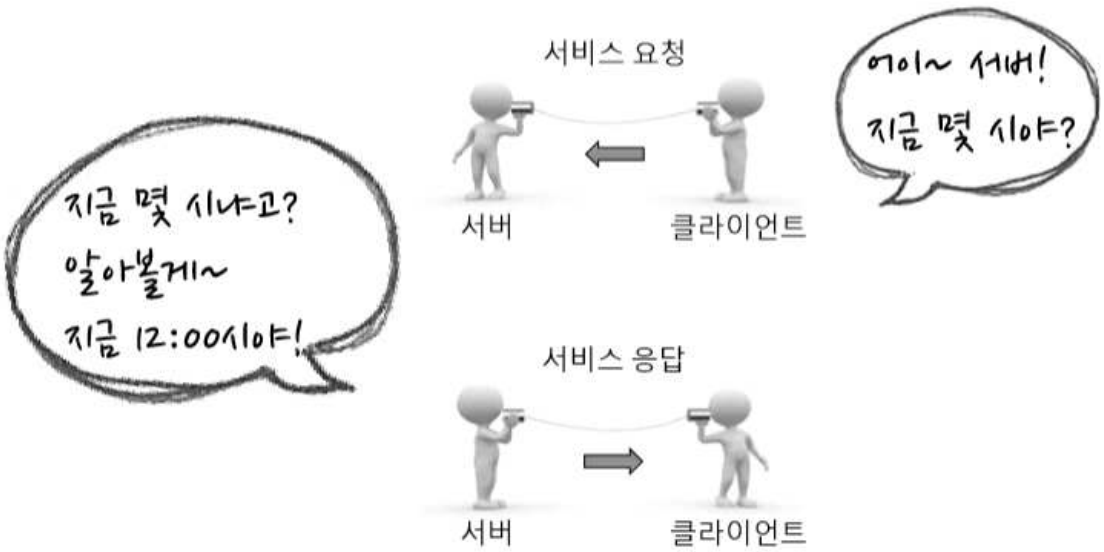




*Topic 에 대해 1:1의 Publisher, Subscriber 통신도 가능하며, 목적에 따라서 1:N, N:1, N:N 통신도 가능하다.

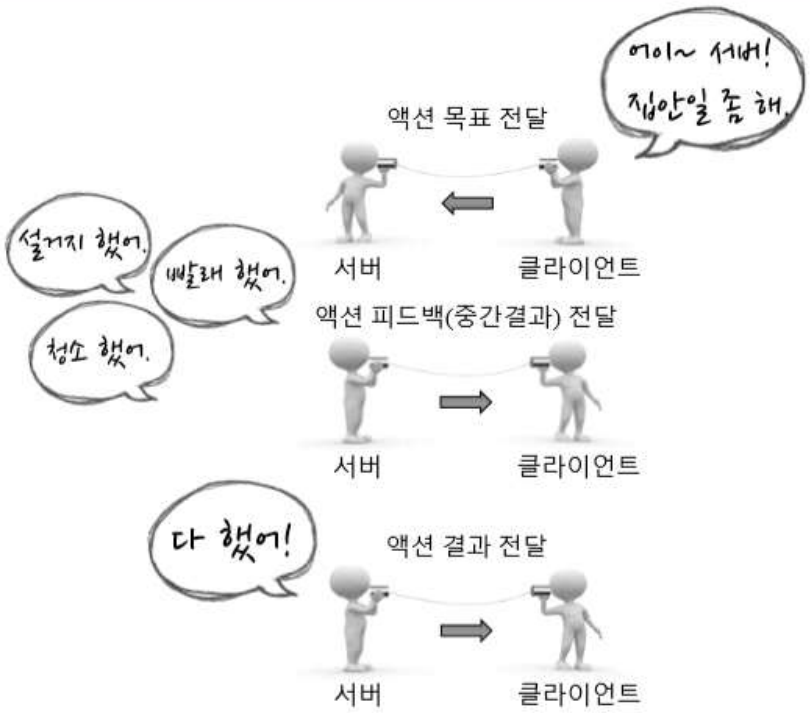
단방향, 연속성 / 제일 많이 쓰임 (90%)

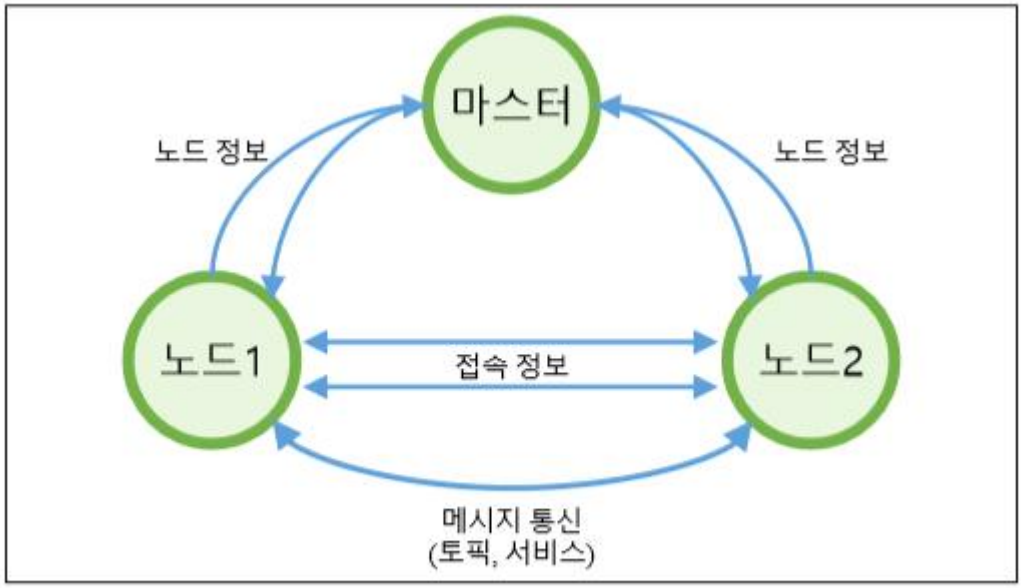
ROS 용어 Service, Service server, Service client



양방향, 일회성 / 이후 접속 끊김

ROS 용어 Action, Action server, Action client





1. 마스터구동 (노드정보관리, 노드 간 통신)
-> roscore
2, 노드2 구동(서브스크라이버 노드 구동)
-> rosrun 패키지이름 노드이름
마스터에 노드이름 / 토픽이름 / 메시지 형태 / IP정보 / port번호 전달
3. 노드1 구동(퍼블리셔 노드 구동)
-> rosrun 패키지이름 노드이름
마스터에 노드2와 같이 정보전달
4. 마스터가 노드2(서브스크라이버 노드)에 새로운 퍼블리셔 정보 전달
5. 퍼블리셔 노드에 접속 요청(마스터로부터 받은 퍼블리셔 정보를 이용하여 TCPROS 접속을 요청)
6. 서브스크라이버 노드에 접속 응답(접속응답에 해당되는 TCP URI 주소와 포트번호 전송)
7. 이후에는 마스터 상관없이 TCP ROS를 이용하여 직접 연결
8. 메시지 통신

5. 패키지 직접 만들기(C++)

1) 패키지 생성

$ cd ~/catkin_ws/src
$ catkin_create_pkg ros_tutorials_topic message_generation std_msgs roscpp
(ros_turorials_topic 이라는 패키지를 만든다. message_generation, std_msgs, roscpp 라는 3개의 의존성 패키지 추가)

2) 패키지 설정 파일(package.xml) 수정

ROS의 필수 설정 파일 중 하나인 package.xml은 패키지 정보를 담은 XML 파일로서 패키지 이름, 저작자, 라이선스, 의존성 패키지 등을 기술하고 있다
$ gedit package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_tutorials_topic</name> (패키지 이름, 만들 때 정해준 이름)
  <version>0.1.0</version> (소프트웨어 버전, 개인적으로 사용할 때는 상관없으나 공식 패키지에는 반드시)
  <description>ROS turtorial package to learn the topic</description> (패키지에 대한 간략한 설명, 공식패키키지에서는 검색어로 쓰이기 때문에 공식에서는 중요)
```

```
 <license>Apache 2.0</license> (개발에 쓰이는 라이센스)
 <author email="pyo@robotis.com">Yoonseok Pyo</author> (저자)
 <maintainer email="pyo@robotis.com">Yoonseok Pyo</maintainer> (패지키 관리자, 초반에는 저자랑 동일인물, 이후에는 변경가능)
 <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>
 <build_depend>roscpp</build_depend> (패키지를 빌드할 때 의존하는 패키지명을 적어준다)
 <build_depend>std_msgs</build_depend>
 <build_depend>message_generation</build_depend>
 <exec_depend>roscpp</exec_depend> (패키지를 실행할 때 의존하는 패키지명을 적어준다)
 <exec_depend>std_msgs</exec_depend>
 <exec_depend>message_runtime</exec_depend>
 <export></export > (로스에서 명시하지 않은 태그를 사용할때 쓰인다. 일반적인 경우 쓸일이 없다.)
</package>
```

3) 빌드 설정 파일(CMakeLiust.txt) 수정

$ gedit CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3) (cmake 최소 요구사항)
project(ros_tutorials_topic) (package.xml에서 입력한 패키지 이름을 그대로 사용한다.)

## 캐킨빌드를 할 때 요구되는 구성요소 패키지이다.
## 의존성패키지로 message_generation, std_msgs, roscpp이며 이 패키지들이 존재하지 않으면 빌드도중에 에러가난다.
find_package(catkin REQUIRED COMPONENTS message_generation std_msgs roscpp)

## 메시지선언: MsgTutorial.msg
## 메시지 파일을 추가하는 옵션이며, 여기서 사용한 메시지는 뒤에 직접 작성 할 것)
add_message_files(FILES MsgTutorial.msg)

## 의존하는 메시지를 설정하는 옵션이다.
## std_msgs가 설치되어 있지않다면 빌드도중에 에러가난다.
## 추가한 메시지 파일에 필요한 의존성
generate_messages(DEPENDENCIES std_msgs)

## 캐킨패키지 옵션으로 라이브러리, 캐킨빌드 의존성, 시스템의존패키지를 기술한다.
catkin_package(
  LIBRARIES ros_tutorials_topic (라이브러리는 패키지 이름을 사용)
  CATKIN_DEPENDS std_msgs roscpp (의존성)
)

## 인클루드 디렉터리를 설정한다.
## catkin_INCLUDE_DIRS은 catkin의 전체적인 디렉터리의 'include' 폴더를 의미함
include_directories(${catkin_INCLUDE_DIRS})

## topic_publisher노드에대한빌드옵션이다.
## 실행파일, 타깃링크 라이브러리, 추가 의존성등을 설정한다.
```

add_executable(topic_publisher src/topic_publisher.cpp) (src/topic_publisher.cpp파일을 참조하여 topic_publisher라는 실행파일을 생성한다)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS} (topic_publisher를 만들 때 뒤에 해당하는 의존성을 참조한다, 대표적으로는 우선 메시지파일을 빌드하고 이후 topic_publisher를 빌드)
${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})

topic_subscriber노드에 대한 빌드옵션이다.
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})

4) 메시지 파일 작성

• 앞서 CMakeLists.txt 파일에 다음과 같은 옵션을 넣었다.

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

• 노드에서 사용할 메시지인 MsgTutorial.msg를 빌드할 때 포함하라는 이야기

```
$ mkdir msg      → ros_tutorials_topic패키지에 msg라는 메시지 폴더를 신규작성
$ cd msg         → 작성한 msg 폴더로이동
$ gedit MsgTutorial.msg     → MsgTutorial.msg 파일 신규작성 및 내용수정
```

$ cd .. → ros_tutorials_topic 패키지폴더로 이동

• 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. 여기서는 간단한 예제를 만들어 보기 위한 것으로 time과 int32를 이용하였다. (부록C 및 http://wiki.ros.org/msg 를 참고 할 것!)

```
time stamp   (현재 시간 사용하기 위함)
int32 data
```

5) 퍼블리셔 노드 작성

• 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다

```
add_executable(topic_publisher src/topic_publisher.cpp)
```

• src 폴더의 topic_publisher.cpp라는 파일을 빌드하여 topic_publisher라는 실행 파일을 만들라는 이야기

```
$ cd src     → ros_tutorials_topic 패키지의 소스폴더인 src 폴더로이동
$ gedit topic_publisher.cpp     → 소스파일 신규작성 및 내용수정
```

```
#include "ros/ros.h"              // ROS 기본헤더파일
#include "ros_tutorials_topic/MsgTutorial.h"      // MsgTutorial메시지파일헤더(빌드후자동생성됨)
int main(int argc, char **argv)                   // 노드메인함수
{
    ros::init(argc, argv, "topic_publisher");      // 노드명초기화
    ros::NodeHandle nh;                           // ROS 시스템과통신을위한노드핸들선언

   // 퍼블리셔선언, ros_tutorials_topic패키지의 MsgTutorial메시지파일을이용한
   // 퍼블리셔ros_tutorial_pub를작성한다. 토픽명은 "ros_tutorial_msg" 이며,
   // 퍼블리셔큐(queue) 사이즈를 100개로설정한다는 것이다
```

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

// 루프주기를설정한다. "10" 이라는것은 10Hz를말하는것으로 0.1초간격으로반복된다
    ros::Rate loop_rate(10);
// MsgTutorial메시지파일형식으로 msg 라는메시지를선언
    ros_tutorials_topic::MsgTutorial msg;
// 메시지에사용될변수선언
    int count = 0;

    while (ros::ok())
    {
        msg.stamp = ros::Time::now();     // 현재시간을 msg의하위 stamp 메시지에담는다
        msg.data = count;          // count라는변수값을 msg의하위 data 메시지에담는다

        ROS_INFO("send msg = %d", msg.stamp.sec);  // stamp.sec 메시지를표시한다, printf 같은 의미
        ROS_INFO("send msg = %d", msg.stamp.nsec); // stamp.nsec 메시지를표시한다
        ROS_INFO("send msg = %d", msg.data);          // data 메시지를표시한다

        ros_tutorial_pub.publish(msg);     // 메시지를발행한다

        loop_rate.sleep(); // 위에서정한루프주기에따라슬립에들어간다

        ++count; // count 변수 1씩증가
    }
    return 0;
}
```

6) 서브스크라이버 노드 작성

• 앞서 CMakeLists.txt 파일에 다음과 같은 실행 파일을 생성하는 옵션을 주었다

```
add_executable(topic_publisher src/topic_publisher.cpp)
```

• src 폴더의 topic_publisher.cpp라는 파일을 빌드하여 topic_publisher라는 실행 파일을 만들라는 이야기

```
$ roscd ros_tutorials_topic/src → 패키지의 소스폴더인 src 폴더로이동
$ gedit topic_subscriber.cpp → 소스파일 신규작성 및 내용수정
```

```
#include "ros/ros.h"        // ROS 기본헤더파일
#include "ros_tutorials_topic/MsgTutorial.h"     // MsgTutorial 메시지파일헤더 (빌드 후 자동생성됨)
// 메시지콜백함수로써, 밑에서설정한 ros_tutorial_msg라는이름의토픽
// 메시지를수신하였을때동작하는함수이다
// 입력메시지로는 ros_tutorials_topic 패키지의 MsgTutorial 메시지를받도록되어있다
void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{
    ROS_INFO("recieve msg = %d", msg->stamp.sec); // stamp.sec 메시지를표시한다
    ROS_INFO("recieve msg = %d", msg->stamp.nsec); // stamp.nsec 메시지를표시한다
    ROS_INFO("recieve msg = %d", msg->data); // data 메시지를표시한다
}
int main(int argc, char **argv)       // 노드메인함수
```

```
{
    ros::init(argc, argv, "topic_subscriber"); // 노드명초기화
    ros::NodeHandle nh;                        // ROS 시스템과 통신을 위한 노드핸들선언

    // 서브스크라이버선언, ros_tutorials_topic패키지의 MsgTutorial메시지파일을이용한
   // 서브스크라이버ros_tutorial_sub를작성한다. 토픽명은 "ros_tutorial_msg" 이며, (동일하게)
   // 서브스크라이버큐(queue) 사이즈를 100개로설정한다는 것이다
   ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg", 100, msgCallback);

   // 콜백함수호출을위한함수로써, 메시지가수신되기를대기,
   // 수신되었을경우콜백함수를실행한다
   ros::spin(); //메시지가 도착하기 전까지 아무동작 안함
   return 0;
}
```

7) ROS 노드 빌드

• 다음 명령어로 ros_tutorials_topic 패키지의 메시지 파일, 퍼블리셔 노드, 서브스 크라이버 노드를 빌드하자

```
$ cd ~/catkin_ws    → catkin 폴더로이동
$ catkin_make    → catkin 빌드실행
```

8) 퍼블리셔 실행 [참고로 노드 실행에 앞서 roscore를 실행하는 것을 잊지 말자]

• ROS 노드 실행 명령어인 rosrun을 이용하여, ros_tutorials_topic 패키지의 topic_publisher 노드를 구동하라는 명령어

```
$ rosrun ros_tutorials_topic topic_publisher
```

9) 서브스크라이버 실행

• ROS 노드 실행 명령어인 rosrun을 이용하여, ros_tutorials_topic 패키지의 topic_subscriber 노드를 구동하라는 명령어

```
$ rosrun ros_tutorials_topic topic_subscriber
```

9) 실행된 노드들의 통신 상태 확인

```
$ rqt_graph
$ rqt [플러그인(Plugins)] → [인트로스펙션(Introspection)] → [노드그래프(Node Graph)]
```

• [참고] rostopic

• rostopic 명령어를 이용하여 현재 ROS 네트워크에서 사용 중인 토픽 목록, 주기, 데이터 대역폭, 내용 확인 등이 가능하다.

```
$ rostopic list
/ros_tutorial_msg
/rosout          //시스템상 기본
/rosout_agg      //시스템상 기본
```

```
$ rostopic info /ros_tutorial_msg    //타입, 퍼블리셔, 서브스크라이버 확인가능
```

5. roslaunch

• rosrun이 하나의 노드를 실행하는 명령어하는 명령어 이다. • roslaunch는 하나 이상의 정해진 노드를 실행시킬 수 있다.
• 그 밖의 기능으로 노드를 실행할 때 패키지의 매개변수나 노드 이름 변경, 노드 네임스페이스 설정, ROS_ROOT 및 ROS_PACKAGE_PATH 설정, 환경 변수 변경 등의 옵션을 붙일 수 있는 ROS 명령어이다.
• roslaunch는 '*.launch'라는 파일을 사용하여 실행 노드를 설정하는데 이는 XML 기반이며, 태그별 옵션을 제공한다.
• 실행 명령어는 "roslaunch [패키지명] [roslaunch 파일]"이다.

1) roslaunch의 활용

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

```
<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>
```

• <launch> 태그 안에는 roslaunch 명령어로 노드를 실행할 때 필요 한 태그들이 기술된다. <node>는 roslaunch로 실행할 노드를 기술 하게 된다. 옵션으로는 pkg, type, name이 있다.
 • pkg 패키지의 이름
 • type 실제 실행할 노드의 이름(노드명)
 • name 위 type에 해당하는 노드가 실행될 때 붙여지는 이름(실행명), 일반 적으로는 type과 같게 설정하지만 필요에 따라 실행할 때 이름을 변경하도록 설정할 수 있다.

• roslaunch 파일을 작성하였으면 다음처럼 union.launch를 실행하자.

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

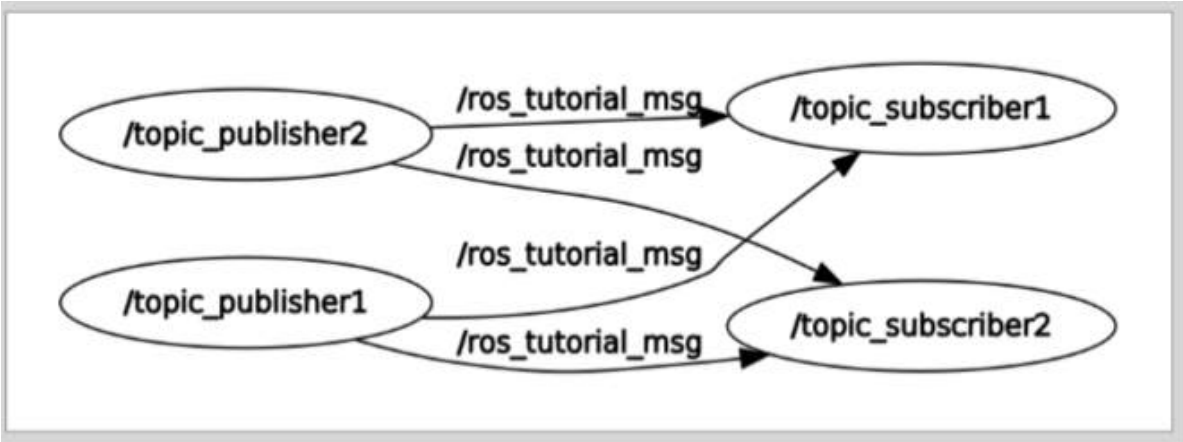
• [참고] roslaunch 사용시에 상태 출력 방법
 • roslaunch 명령어로여럿의 노드가 실행될 때는 실행되는 노드들의 출력(info, error 등)이 터미널 스크린에 표시되지 않아 디버깅하기 어렵게 된다. 이때에 --screen 옵션을 추가해주면 해당 터미널에 실행되는 모든 노드들의 출력들 이 터미널 스크린에 표시된다.

• 실행 결과?

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

• 결과적으로 topic_publisher 노드가 topic_publisher1과 topic_publisher2로 이름이 바뀌어 두 개의 노드가 실행되었다.
• topic_subscriber 노드도 topic_subscriber1과 topic_subscriber2로 이 름이 바뀌어 실행되었다.

- 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 이 름을 바꿔주지 않았기 때문이다.
- 이 문제를 다른 roslaunch 네임스페이스 태그를 사용하여 해결해보자.



- union.launch을 수정하자. (group으로 묶어서)

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

```
<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>
```

- 변경 후의 실행된 노드들의 모습

2) launch 태그

<launch> roslaunch 구문의 시작과 끝을 가리킨다.

<node> 노드 실행에 대한 태그이다. 패키지, 노드명, 실행명을 변경할 수 있다.

<machine> 노드를 실행하는 PC의 이름, address, ros-root, ros-package-path 등을 설정할 수 있다.

<include> 다른 패키지나 같은 패키지에 속해 있는 다른 launch를 불러와 하나의 launch파일처럼 실행할 수 있다.

<remap> 노드 이름, 토픽 이름 등의 노드에서 사용 중인 ROS 변수의 이름을 변경할 수 있다.

<env> 경로, IP 등의 환경 변수를 설정한다. (거의 안쓰임)

<param> 매개변수 이름, 타이프, 값 등을 설정한다

<rosparam> rosparam 명령어 처럼 load, dump, delete 등 매개변수 정보의 확인 및 수정한다.

<group> 실행되는 노드를 그룹화할 때 사용한다.

<test> 노드를 테스트할 때 사용한다. <node>와 비슷하지만 테스트에 사용할 수 있는 옵션들이 추가되어 있다.

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

6. Python 패키지 직접 만들기

다음 참고 : https://htsstory.tistory.com/entry/ROS-python%ED%8C%8C%EC%9D%B4%EC%8D%AC-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-1-%ED%86%A0%ED%94%BD-%EB%A9%94%EC%8B%9C%EC%A7%80-%ED%86%B5%EC%8B%A0

1) 기존의 C++ 의 과정과 동일하다.
   다만 패키지를 생성시 의존성 패키지 rospy를 생성해준다.
-> catkin_create_pkg my_first_python std_msgs rospy

2) 메시지 파일 생성
-> roscd my_first_python
-> mkdir msg
-> echo "int64 num" > msg/Num.msg

3) 퍼블리셔 생성
-> cd src
-> gedit talker.py

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

import rospy
from std_msgs.msg import String

def talker():
        pub = rospy.Publisher('chatter',String,queue_size=10)
        rospy.init_node('talker', anonymous = True)
        rate = rospy.Rate(10) #10hz
        while not rospy.is_shutdown():
                hello_str = "hello world %s" % rospy.get_time()
                rospy.loginfo(hello_str)
                pub.publish(hello_str)
                rate.sleep()

if __name__=='__main__':
        try:
                talker()
      except rospy.ROSInterruptException:
                pass
```

```
chmod +x talker.py
```

4) 서브스크라이버 생성
-> gedit listener.py

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

import rospy
from std_msgs.msg import String
```

```
def callback(data):
        rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():
        rospy.init_node('listener',anonymous=True)

        rospy.Subscriber("chatter", String, callback)

        rospy.spin()

if __name__=='__main__':
        listener()
```

```
chmod +x listener.py
```

5) 실행

-> roscore

-> rosrun my_first_python talker.py

-> rosrun my_first_python listener.py

7. c++ publisher 와 Python subscriber 연결확인

다음과정은 기존에 만든 파일을 바탕으로 서로 연결이 되도록 수정하였으며 가능한 하나의 패키지에서 만들 것을 권장한다. (실행하기까지 여러 오류가 발생하는데 같은 메시지 파일을 공유하게 하는 것이 중요? 다른 패키지에서 노드들간 통신하려면? 뭔소리지? 암튼 하나의 패키지에서 하는 것이 편함)

1) 기존의 c++ publisher 일부 수정
-> msg 파일의 time stamp 제거 및 publisher 의 time 관련 문장 제거

2) python의 subscriber를 다음과 같이 수정

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

import rospy
from std_msgs.msg import Int32

def callback(data):
        rospy.loginfo("I heard %s", data.data)

def listener():
        rospy.init_node('listener',anonymous=True)

        rospy.Subscriber("ros_tutorial_msg", Int32, callback)

        rospy.spin()

if __name__=='__main__':
        listener()
```

3) 각 노드를 실행시켜 연결되는지 확인

8. import 문제

jetbot 동작을 위한 모듈을 받아 import시 모듈을 찾지 못하는 에러가 발생했다

에러 해결은 다음과 같다. 기존의 파이썬 코드는 #!/usr/bin/env python 와 같이 기본 파이썬을 python2를 사용하는데 문제를 해결하려면 python3가 필요하므로 #!/usr/bin/env python3 와 같이 환경을 바꾸어준다.

이후 python2에만 있는 rospkg를 찾지 못하므로 python3에도 설치해준다.

-> sudo apt-get install python3-pip python3-yaml

-> sudo pip3 install rospkg catkin_pkg