

ROS2 통신 - Topic

경희대학교 기계공학과
로봇 제어 지능 연구실
김상현 교수



01

ROS2 Topic

ROS2 Topic

- ◉ Topic이란 Node 간의 메시지를 주고받는 통신의 한 방법
- ◉ Topic을 주는 쪽을 Publisher, 받는 쪽을 Subscriber라고 한다.
- ◉ Topic의 주요 특징

- ◉ **이름**

- 각 Topic은 ROS 2 네임스페이스 내에서 고유한 이름을 가지며, 노드가 특정 Topic을 쉽게 찾고 통신할 수 있게 한다

- ◉ **데이터 유형**

- Topic은 데이터 형식이 정의되어 있어서 통신에서 일관성과 유형 안정성을 보장한다

- ◉ **Publisher**

- 데이터를 생성하고 Topic을 보내는 노드를 Publisher라고 한다.
 - Topic은 여러 Publisher 을 가질 수 있다.

- ◉ **Subscriber**

- Topic으로부터 데이터를 받고 처리하는 노드를 Subscriber라고 한다.
 - Topic은 여러 Subscriber 을 가질 수 있다.

- ◉ **비동기 통신**

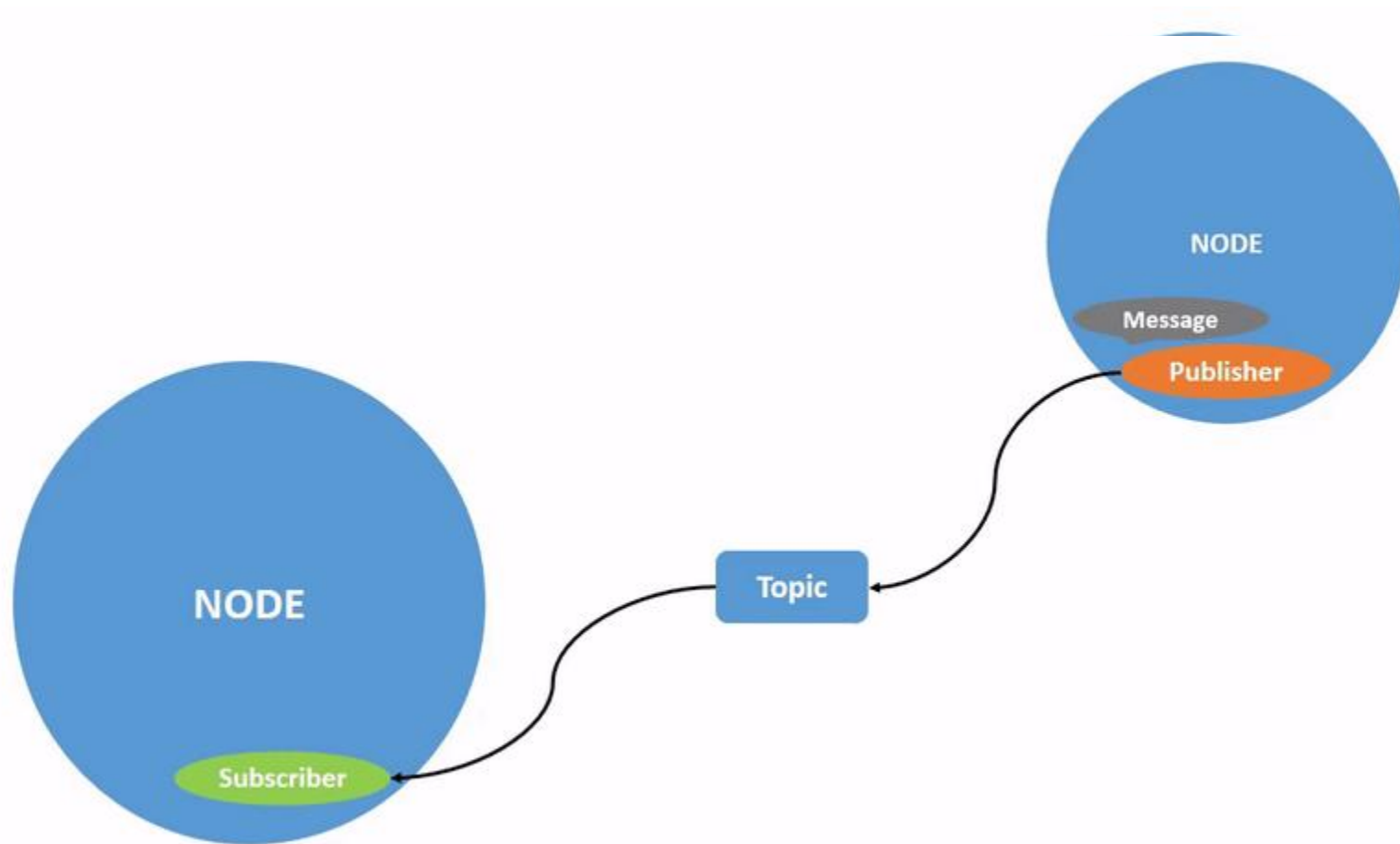
- Topic을 통한 통신은 비동기적으로 이루어지며, Publisher와 Subscriber가 동시에 활성화되지 않아도 된다.

- ◉ **메시지 큐**

- Topic은 수신 메시지를 저장하는 메시지 큐를 가질 수 있다. 큐 크기는 지정할 수 있고 용량을 초과하는 메시지는 삭제된다.

ROS2 Topic

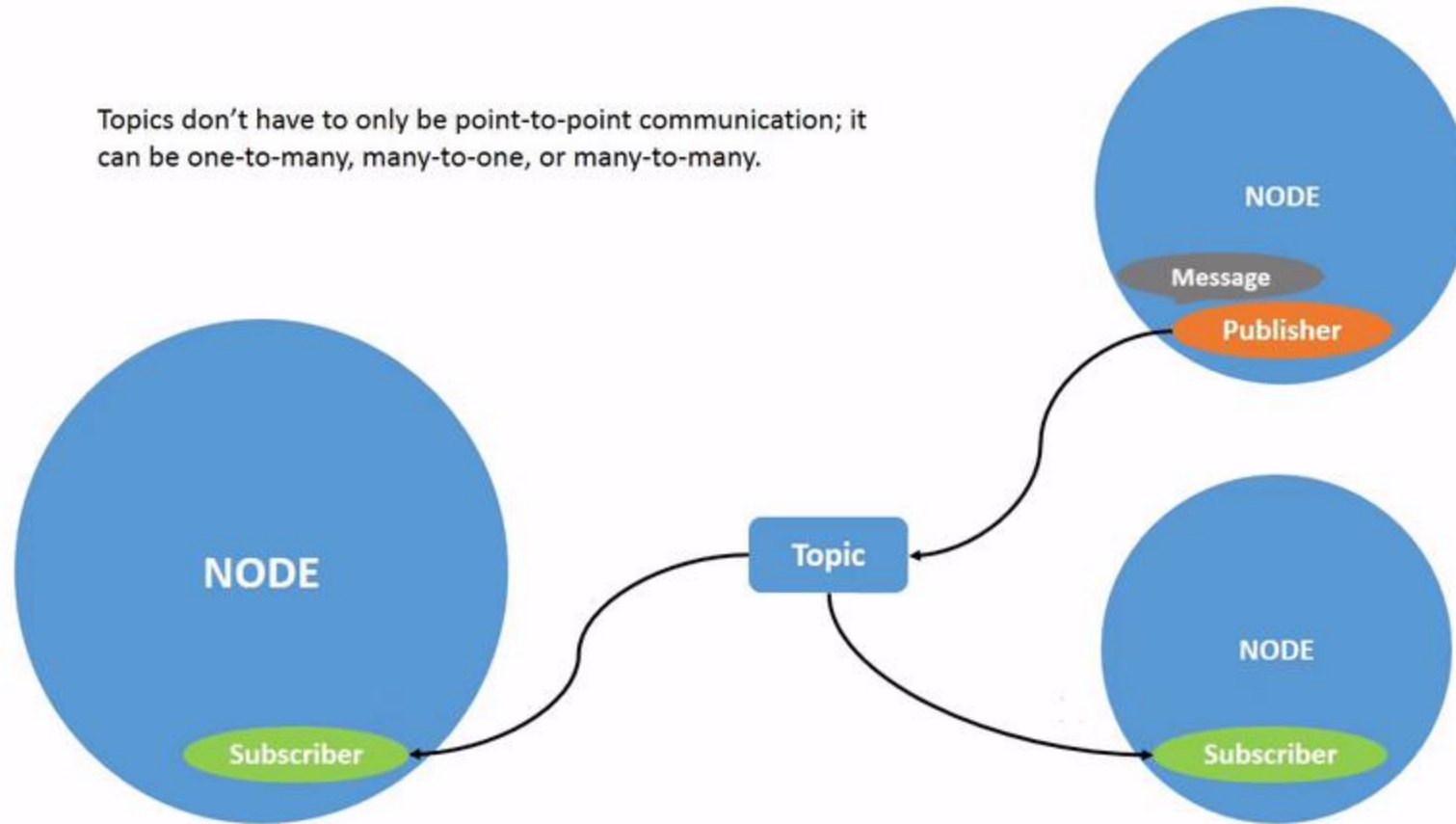
예시 – 1:1 통신



ROS2 Topic

예시 – N:N 통신

Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



ROS2 Topic

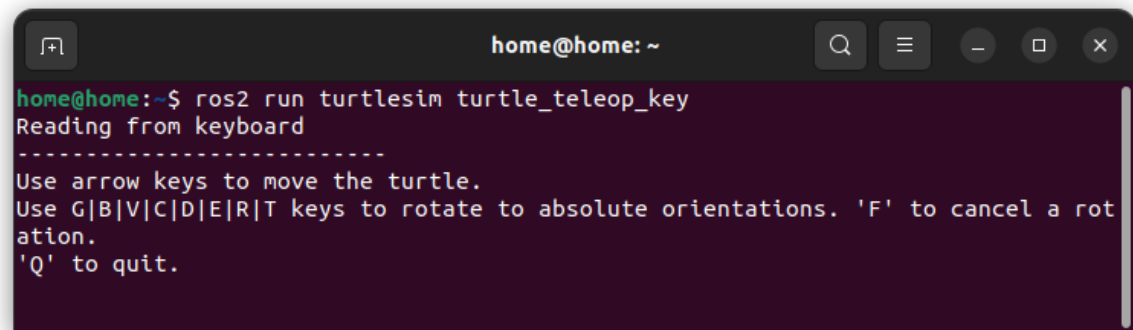
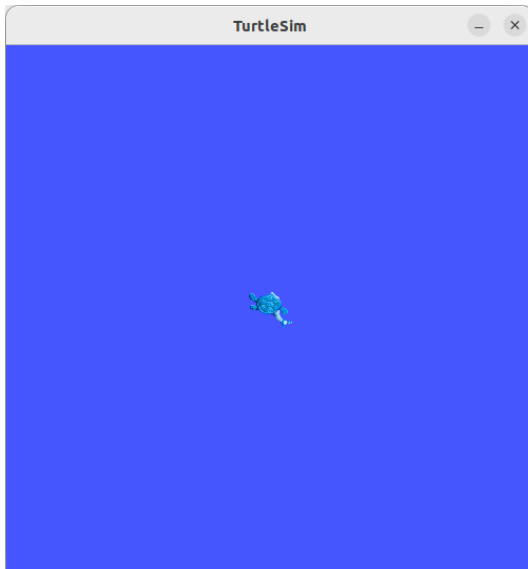
▶ ROS2 Topic 명령어

Turtlesim Node 실행 (Terminal 1)

```
$ ros2 run turtlesim turtlesim_node
```

Teleop Node 실행(Terminal 2)

```
$ ros2 run turtlesim turtle_teleop_key
```

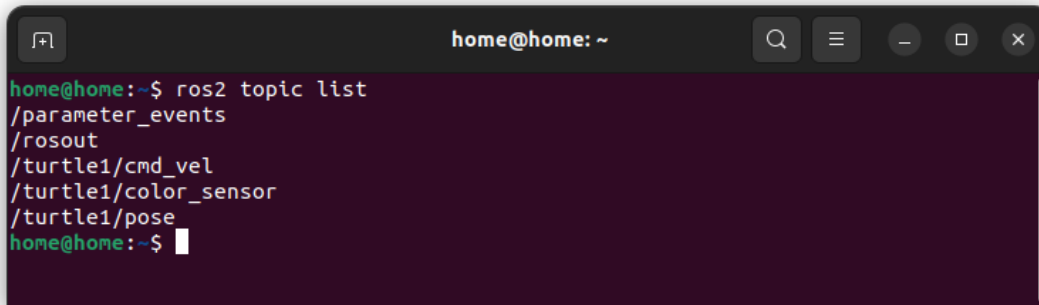
방향키와 문자들로 자유롭게 움직여보세요

ROS2 Topic

▶ ROS2 Topic 명령어

실행중인 Topic 목록 확인 (Terminal 3)

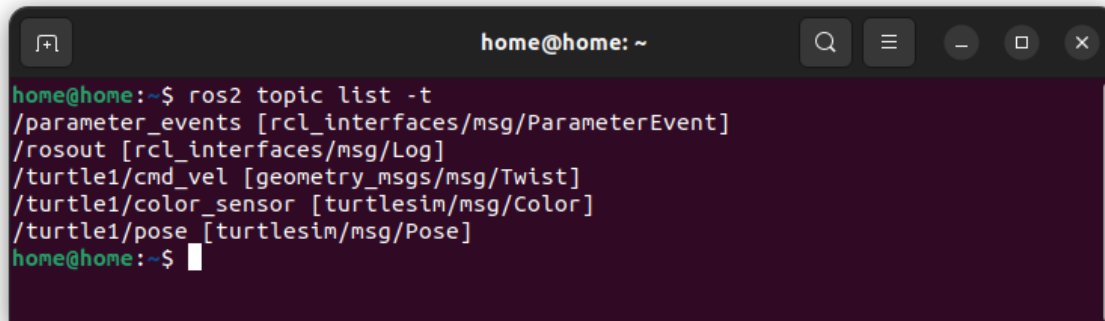
\$ ros2 topic list



```
home@home: ~  
home@home:~$ ros2 topic list  
/parameter_events  
/rosout  
/turtle1/cmd_vel  
/turtle1/color_sensor  
/turtle1/pose  
home@home:~$
```

메시지 타입을 확인하며 실행중인 Topic 목록 확인 (Terminal 3)

\$ ros2 topic list -t



```
home@home: ~  
home@home:~$ ros2 topic list -t  
/parameter_events [rcl_interfaces/msg/ParameterEvent]  
/rosout [rcl_interfaces/msg/Log]  
/turtle1/cmd_vel [geometry_msgs/msg/Twist]  
/turtle1/color_sensor [turtlesim/msg/Color]  
/turtle1/pose [turtlesim/msg/Pose]  
home@home:~$
```

ROS2 Topic

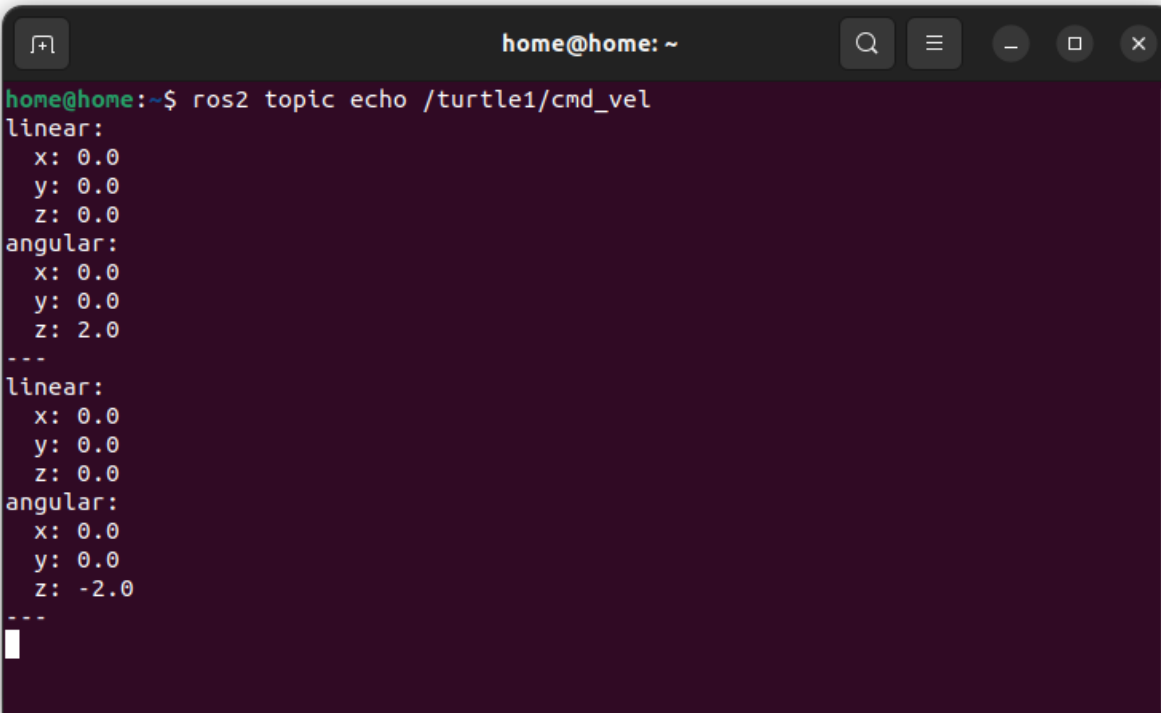
▶ ROS2 Topic 명령어

ROS2 Topic에서 발행되는 메시지 출력(Terminal 3)

```
$ ros2 topic echo /turtle1/cmd_vel
```

Topic 이름

Terminal 2에서 거북이를 움직여 Topic을 Publish하면 거북이의 속도값을 볼 수 있다.

A terminal window titled 'home@home: ~' with standard window controls. It shows the command 'ros2 topic echo /turtle1/cmd_vel' being executed. The output displays two sets of velocity data for the 'linear' and 'angular' components, separated by dashed lines. The first set shows linear velocity at (0.0, 0.0, 0.0) and angular velocity at (0.0, 0.0, 2.0). The second set shows linear velocity at (0.0, 0.0, 0.0) and angular velocity at (0.0, 0.0, -2.0).

```
home@home:~$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 2.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -2.0
---
```

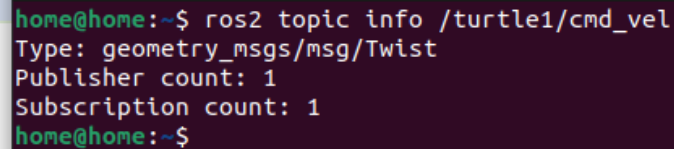

ROS2 Topic

▶ ROS2 Topic 명령어

토픽의 통신 구조 파악 (Terminal 3)

```
$ ros2 topic info /turtle1/cmd_vel
```

Topic 이름



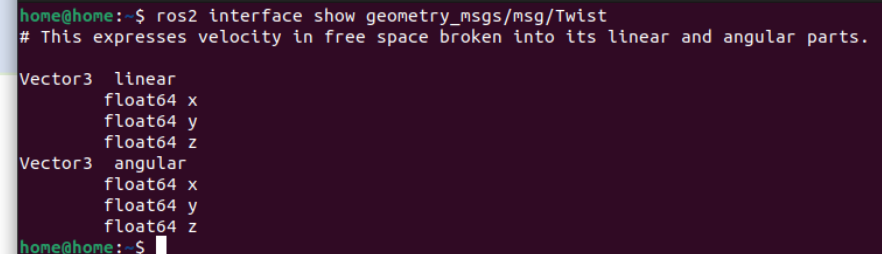
```
home@home: ~  
home@home:~$ ros2 topic info /turtle1/cmd_vel  
Type: geometry_msgs/msg/Twist  
Publisher count: 1  
Subscription count: 1  
home@home:~$
```

토픽 메시지 구조 파악 (Terminal 3)

```
$ ros2 interface show geometry_msgs/msg/Twist
```

Topic 이름

메시지의 데이터 구조에 대한 자세한 정보를 확인할 수 있다.



```
home@home: ~  
home@home:~$ ros2 interface show geometry_msgs/msg/Twist  
# This expresses velocity in free space broken into its linear and angular parts.  
Vector3 linear  
  float64 x  
  float64 y  
  float64 z  
Vector3 angular  
  float64 x  
  float64 y  
  float64 z  
home@home:~$
```

ROS2 Topic

▶ ROS2 Topic 명령어

Terminal에서 Topic Publish 하는법

```
$ ros2 topic pub <Topic 이름> <메시지 타입> “<실제 데이터>”
```

Topic 한번만 Pub(Terminal 3)

```
$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0},angular:{x: 0.0, y: 0.0, z: 1.8}}"
```

Terminal에서 입력할 때 어느정도 입력하고 **Tab키**를 적극활용해서 입력에러를 줄일 수 있다

Topic 여러번 Pub(Terminal 3)

```
$ ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0},angular:{x: 0.0, y: 0.0, z: 1.8}}"
```

rate 명령어는 Hz, 즉 초당 몇회의 publish를 결정하는 변수

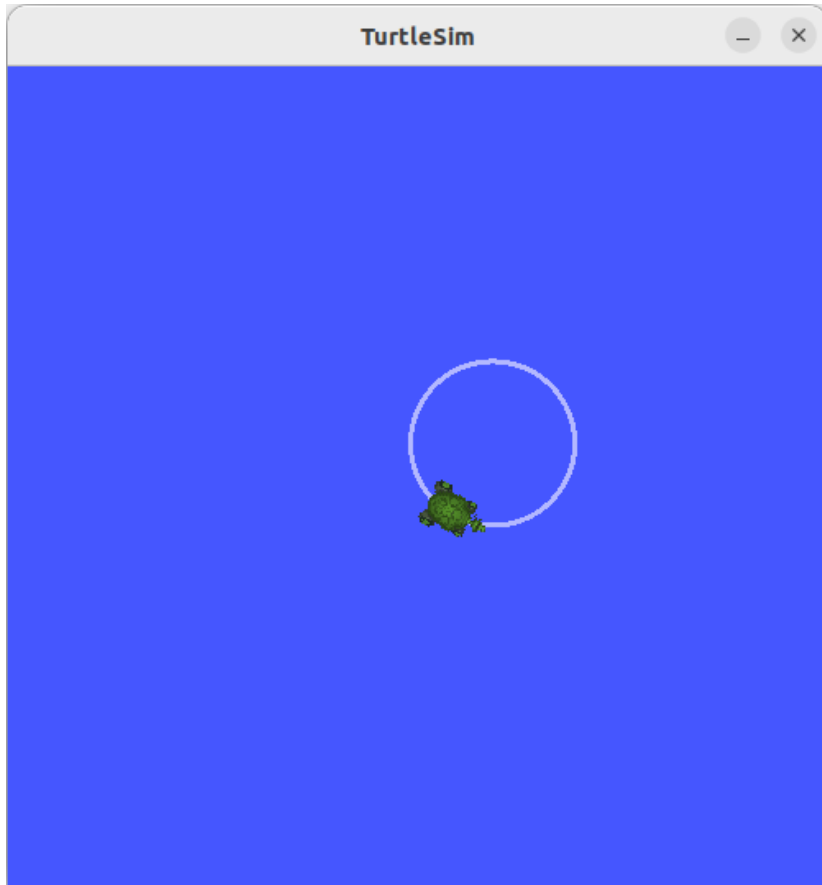
발행된 토픽의 비율 확인 (Terminal 4)

```
$ ros2 topic hz /turtle1/cmd_vel
```

rate 명령어는 Hz, 즉 초당 몇회의 publish가 이뤄지는지 확인

ROS2 Topic

▶ 실행 결과



```
home@home: ~  
home@home:~$ ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "linear:  
x: 2.0  
y: 0.0  
z: 0.0  
angular:  
x: 0.0  
y: 0.0  
z: 1.8 "
```

```
home@home: ~  
home@home:~$ ros2 topic hz /turtle1/cmd_vel  
average rate: 1.000  
  min: 1.000s max: 1.000s std dev: 0.00006s window: 2  
average rate: 1.000  
  min: 1.000s max: 1.001s std dev: 0.00034s window: 3  
average rate: 1.000  
  min: 0.999s max: 1.001s std dev: 0.00047s window: 5  
average rate: 1.000  
  min: 0.999s max: 1.001s std dev: 0.00040s window: 7  
average rate: 1.000  
  min: 0.999s max: 1.001s std dev: 0.00038s window: 8  
average rate: 1.000  
  min: 0.999s max: 1.001s std dev: 0.00035s window: 10  
home@home:~$
```

02

ROS2 Topic (실습)

ROS2 Topic 실습 (Python)

- ▶ (실습1) topic_tutorial_py Package를 만들어, velocity를 publish하고 pose를 subscribe하는 node들을 만들어 본다.

topic_tutorial_py 패키지 만들기

```
$ ros2 pkg create topic_tutorial_py --build-type ament_python --dependencies turtlesim std_msgs geometry_msgs rclpy  
# ros2_ws/src폴더 안에 만들 것
```

Setup.py 수정 (2차시 예제 참조)

```
entry_points={  
    'console_scripts': [  
        'pub_vel_node = topic_tutorial_py.pubvel:main',  
        'sub_pose_node = topic_tutorial_py.subpose:main',  
    ],  
}
```

ROS2 Topic 실습 (Python)

- (실습1) topic_tutorial_py Package를 만들어, velocity를 publish하고 pose를 subscribe하는 node들을 만들어 본다.

pubvel.py 만들기 (2차시 예제 참조)

```
import rclpy
import random
from rclpy.node import Node
from geometry_msgs.msg import Twist

class TurtlesimVelNode(Node):
    def __init__(self):
        super().__init__("turtlesim_pub_vel_node")
        self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
        self.msg_ = Twist()
        self.timer_ = self.create_timer(0.1, self.timer_callback)

    def timer_callback(self):
        self.msg_.linear.x = (random.random() - 0.5)
        self.msg_.angular.z = 2*(random.random())-1
        self.publisher_.publish(self.msg_)

        self.get_logger().info(f"Twist Message - Linear X: {self.msg_.linear.x}, Angular Z: {self.msg_.angular.z}")

def main(args=None):
    rclpy.init(args=args)
    pub_vel_node = TurtlesimVelNode()
    rclpy.spin(pub_vel_node)
    pub_vel_node.destroy_node()
    rclpy.shutdown()

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

turtle1/cmd_vel topic에 Twist 메시지 전달할 publisher 생성

Twist타입의 메시지 생성

생성된 twist_msg에 임의의 속도값을 설정

Publisher 객체에 속도값을 넣은 twist_msg를 넣어 전달

ROS2 Topic 실습 (Python)

▶ 코드 설명

– 퍼블리싱 메시지

메시지 타입: 각 토픽들은 정해진 메시지 규격을 따른다.

1) `from geometry_msgs.msg import Twist`

퍼블리싱 오브젝트: 각 메시지들은 지정된 토픽을 통해 “발간” 된다.

1) `self.publisher_ = self.create_publisher(<message_type>, <topic_name>, <queue_size>)`

– `node.create_publisher`: an object of class `node`

– `message_type`: — formally called the template parameter — is the data type for the messages(본 예제에서는 `geometry_msgs` 의 `Twist`)

– `topic_name`: a string containing the name of the topic on which we want to publish (본 예제에서는 `turtle1/cmd_vel`)

– `queue_size`: the size of the queue where messages are stored.

메시지 오브젝트: msg를 `geometry_msgs/Twist`로 생성하고, msg안에 다양한 값들을 넣어준다.

퍼블리싱 메시지: 마지막으로 메시지 오브젝트를 퍼블리싱 오브젝트에 삽입하여, 발간

1) `publisher_.publish(msg);`

ROS Topic 실습 (Python)

- ▶ (실습1) topic_tutorial_py Package를 만들어, velocity를 publish하고 pose를 subscribe하는 node들을 만들어 본다.

subpose.py 만들기 (2차시 예제 참조)

```
import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose

class SubPoseNode(Node):
    def __init__(self):
        super().__init__("subscribe_pose_node")
        self.subscription_ = self.create_subscription(
            Pose,
            'turtle1/pose',
            self.pose_callback,
            10
        )

    def pose_callback(self, msg):
        x = msg.x
        y = msg.y
        theta = msg.theta

        self.get_logger().info(f"Turtlebot Pose -X: {x}, -Y: {y}, -Theta: {theta}")

def main(args=None):
    rclpy.init(args=args)
    sub_pose_node = SubPoseNode()
    rclpy.spin(sub_pose_node)
    rclpy.shutdown()

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

subscribe_pose_node를 생성

turtle1/pose를 pose_callback함수를 통해 Subscribe한다고 설정

Pose msg를 수신받을 때마다 호출되는 Callback함수 생성

Robot의 위치를 x,y,theta로 받아 출력

ROS2 Topic 실습 (Python)

▶ 코드 설명

- Callback 함수 : 얻은 메시지를 통해 어떤 작업을 수행할 것인가?

Topic이 도착할 때 마다 수행되는 함수.

```
1) def pose_callback(self,msg)
```

Subscriber 오브젝트: 어떤 토픽을 받을 것인지 생성해야하는 오브젝트.

```
self.subscription_ = self.create_subscription(<message_type>,<topic_name>,<callback_function>,<Queue size>)
```

- node->create_subscription : an object of class node
- topic_name: a string containing the name of the topic on which we want to publish (본 예제에서는 turtle1/pose)
- callback_function is the pointer to the callback function, &poseMessageReceived in the example

rcipy.spin(node)

- 1) rcipy.spin(node)은 ROS에게 노드가 종료될 때까지 콜백을 기다리고 실행하도록 요청합니다. 즉, 대략 다음 루프와 동일합니다.
while rcipy.ok():
 {Callback function}

ROS2 Topic 실습 (Python)

- (실습2) 만든 node들을 실행하고, turtle1/pose 의 정보를 ros2 topic info를 이용하여, 파악해 봅시다.

빌드

```
colcon build #ros2_ws에서
```

- (실습2) 만든 node들을 실행하고, turtle1/pose 의 정보를 ros2 topic info를 이용하여, 파악해 봅시다.

Terminal1

```
$ ros2 run topic_tutorial_py pub_vel_node
```

Terminal2

```
$ ros2 run topic_tutorial_py sub_pose_node
```

- (실습3) rqt_graph를 활용하여 노드들간의 관계를 파악해 봅시다.

Terminal3

```
rqt_graph
```

ROS2 Topic 실습 (C++)

- (실습1) topic_tutorial_cpp Package를 만들어, velocity를 publish하고 pose를 subscribe하는 node들을 만들어 본다.

topic_tutorial_cpp 패키지 만들기

```
$ ros2 pkg create topic_tutorial_cpp --build-type ament_cmake --dependencies turtlesim std_msgs geometry_msgs rclcpp  
# ros2_ws/src폴더 안에 만들 것
```

Cmakelists.txt 수정 (2차시 예제 참조)

```
add_executable(pub_vel_node src/pubvel.cpp)  
ament_target_dependencies(pub_vel_node rclcpp geometry_msgs)
```

```
install(TARGETS  
  pub_vel_node  
  DESTINATION lib/${PROJECT_NAME})
```

```
add_executable(sub_pose_node src/subpose.cpp)  
ament_target_dependencies(sub_pose_node rclcpp turtlesim)
```

```
install(TARGETS  
  sub_pose_node  
  DESTINATION lib/${PROJECT_NAME})
```

ROS2 Topic 실습 (C++)

- (실습1) topic_tutorial_cpp Package를 만들어, velocity를 publish하고 pose를 subscribe하는 node들을 만들어 본다.

pubvel.cpp 만들기 (2차시 예제 참조)

```
#include "rclcpp/rclcpp.hpp"
#include "geometry_msgs/msg/twist.hpp"
#include <chrono>
#include <functional>
#include <stdlib.h>

class TurtlesimVelNode : public rclcpp::Node
{
public:
    TurtlesimVelNode() : Node("turtlesim_pub_vel_node")
    {
        publisher_ = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel",10);
        timer_ = this->create_wall_timer(std::chrono::milliseconds(100),std::bind(&TurtlesimVelNode::timer_callback,this));
        msg_ = std::make_shared<geometry_msgs::msg::Twist>();
    }
private:
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr publisher_;
    geometry_msgs::msg::Twist::SharedPtr msg_;

    void timer_callback()
    {
        msg_ -> linear.x = double(rand() -double(RAND_MAX)/2.0)/double(RAND_MAX);
        msg_ -> angular.z = 2 * double(rand())/double(RAND_MAX)-1;
        publisher_ ->publish(*msg_);
        RCLCPP_INFO(this->get_logger(),"Twist Message - Linear X: %f Angular Z: %f",
            msg_ -> linear.x, msg_ ->angular.z);
    }
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc,argv);
    auto node = std::make_shared<TurtlesimVelNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
}
```

turtle1/cmd_vel topic에 Twist 메시지 전달할 publisher 생성

Twist타입의 메시지 생성

생성된 twist_msg에 임의의 속도값을 설정

Publisher 객체에 속도값을 넣은 twist_msg를 넣어 전달

ROS2 Topic 실습 (C++)

▶ 코드 설명

– 퍼블리싱 메시지

메시지 타입: 각 토픽들은 정해진 메시지 규격을 따른다.

1) `#include "geometry_msgs/msg/twist.hpp"`

퍼블리싱 오브젝트: 각 메시지들은 지정된 토픽을 통해 “발간” 된다.

1) `publisher_ = this->create_publisher<message_type>("topic_name", "queue size")`

– `this->create_publisher`: an object of class `node`

– `message_type`: — formally called the template parameter — is the data type for the messages(본 예제에서는 `geometry_msgs` 의 `Twist`)

– `topic_name`: a string containing the name of the topic on which we want to publish (본 예제에서는 `turtle1/cmd_vel`)

– `queue_size`: the size of the queue where messages are stored.

메시지 오브젝트: msg를 `geometry_msgs::msg::Twist`로 생성하고, msg안에 다양한 값들을 넣어준다.

퍼블리싱 메시지: 마지막으로 메시지 오브젝트를 퍼블리싱 오브젝트에 삽입하여, 발간

1) `publisher_>publish(msg);`

ROS Topic 실습 (C++)

- (실습1) topic_tutorial_cpp Package를 만들어, velocity를 publish하고 pose를 subscribe하는 node들을 만들어 본다.

subpose.cpp 만들기 (2차시 예제 참조)

```
#include "rclcpp/rclcpp.hpp"
#include "turtlesim/msg/pose.hpp"

#include "memory"

using std::placeholders::_1;
class SubPoseNode : public rclcpp::Node
{
public:
    SubPoseNode() : Node("subscribe_pose_node")
    {
        subscription_ = this->create_subscription<turtlesim::msg::Pose>
            ("turtle1/pose",10,std::bind(&SubPoseNode::pose_callback,this,_1));
    }
private:
    void pose_callback(turtlesim::msg::Pose::SharedPtr msg)
    {
        double x = msg->x;
        double y = msg->y;
        double theta = msg->theta;

        RCLCPP_INFO(this->get_logger(),"Turtlebot Pose -X: %f, Y: %f, Theta: %f",x,y,theta);
    }

    rclcpp::Subscription<turtlesim::msg::Pose>::SharedPtr subscription_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<SubPoseNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
}
```

subscribe_pose_node를 생성

turtle1/pose를 pose_callback함수를 통해 Subscribe한다고 설정

Pose msg를 수신받을 때마다 호출되는 Callback함수 생성

Robot의 위치를 x,y,theta로 받아 출력

ROS2 Topic 실습 (C++)

▶ 코드 설명

- Callback 함수 : 얻은 메시지를 통해 어떤 작업을 수행할 것인가?

Topic이 도착할 때 마다 수행되는 함수.

```
1) void pose_callback(turtlesim::msg::Pose::SharedPtr msg)
```

Subscriber 오브젝트: 어떤 토픽을 받을 것인지 생성해야하는 오브젝트.

```
subscription_ = this->create_subscription<message type>("topic_name", "queuesize", "callback function")
```

- node->create_subscription : an object of class node
- topic_name: a string containing the name of the topic on which we want to publish (본 예제에서는 turtle1/pose)
- callback_function is the pointer to the callback function, &poseMessageReceived in the example

rcpp::spin(node)

- 1) rcpp::spin(node)은 ROS에게 노드가 종료될 때까지 콜백을 기다리고 실행하도록 요청합니다. 즉, 대략 다음 루프와 동일합니다.
While (rcpp.ok())
{Callback function;}

ROS2 Topic 실습 (C++)

- (실습2) 만든 node들을 실행하고, turtle1/pose 의 정보를 ros2 topic info를 이용하여, 파악해 봅시다.

빌드

```
colcon build #ros2_ws에서
```

- (실습2) 만든 node들을 실행하고, turtle1/pose 의 정보를 ros2 topic info를 이용하여, 파악해 봅시다.

Terminal1

```
$ ros2 run topic_tutorial_cpp pub_vel_node
```

Terminal2

```
$ ros2 run topic_tutorial_cpp sub_pose_node
```

- (실습3) rqt_graph를 활용하여 노드들간의 관계를 파악해 봅시다.

Terminal3

```
rqt_graph
```

02

ROS Message

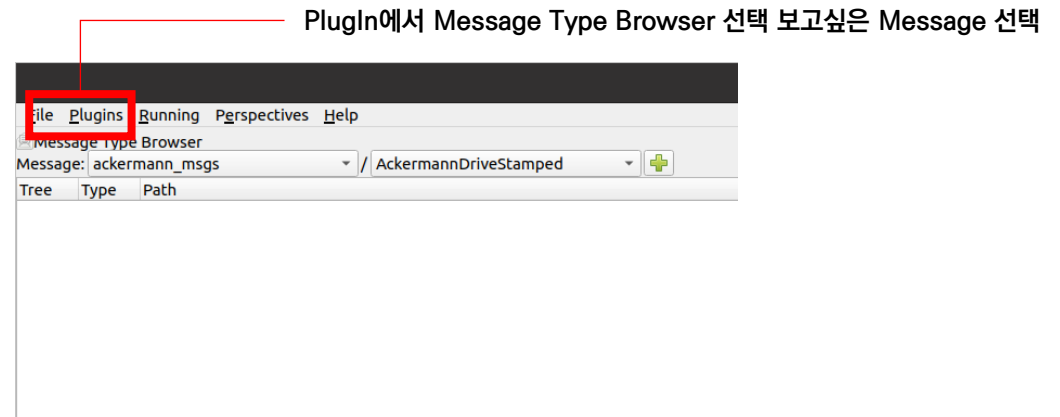
ROS Message

▶ 스탠다드 메시지 (std_msgs)

– http://wiki.ros.org/std_msgs

- Bool
- Byte
- ByteMultiArray
- Char
- ColorRGBA
- Duration
- Empty
- Float32
- Float32MultiArray
- Float64
- Float64MultiArray
- HeaderCheck your run_id folder and browse the rosout.log file
- Int16
- Int16MultiArray
- Int32
- Int32MultiArray
- Int64
- Int64MultiArray
- Int8
- Int8MultiArray
- MultiArrayDimension
- MultiArrayLayout
- String
- Time
- UInt16
- UInt16MultiArray
- UInt32
- UInt32MultiArray
- UInt64
- UInt64MultiArray
- UInt8
- UInt8MultiArray

▶ (실습4) rqt를 활용하여 원하는 메시지의 형태를 살펴봅니다.



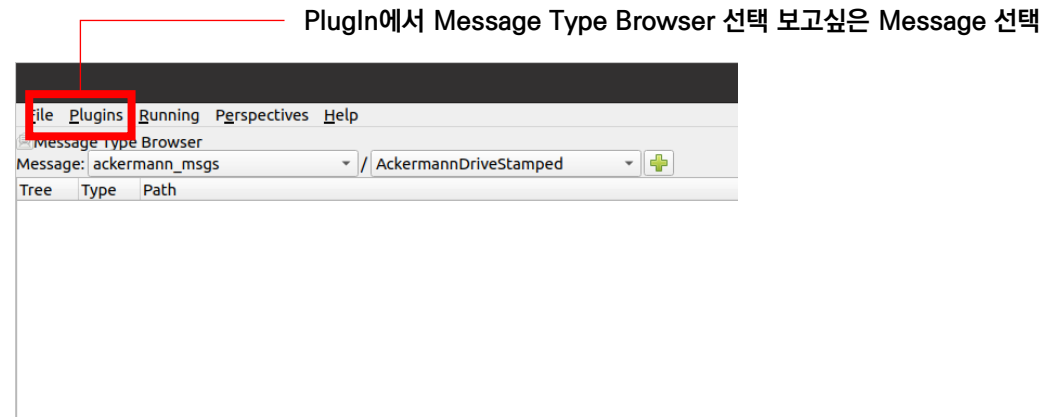
ROS Message

Common 메시지

http://wiki.ros.org/sensor_msgs, http://wiki.ros.org/geometry_msgs 등

- Accel
- AccelStamped
- AccelWithCovariance
- AccelWithCovarianceStampedmessage
- Inertia
- InertiaStamped
- Point
- Point32
- PointStamped
- Polygon
- PolygonStamped
- Pose
- Pose2D
- PoseArray
- PoseStamped
- PoseWithCovariance
- PoseWithCovarianceStamped
- Quaternion
- QuaternionStamped
- Transform
- TransformStamped
- Twist
- TwistStamped
- TwistWithCovariance
- TwistWithCovarianceStamped
- Vector3
- Vector3Stamped
- Wrench
- WrenchStamped

◉ (실습4) rqt를 활용하여 Twist 메시지의 형태를 살펴봅니다.



ROS Topic 실습 (Python)

- (실습4) direction 노드를 만들어, turtle1/cmd_vel의 linear.x가 음수 일 때, std_msgs/Bool을 이용하여, “Bool을 False” 반대의 경우 True로 퍼블리시하는 publish-subscriber 노드를 만듭시다.

direction.py 만들기 (2차시 예제 참조)

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Bool
from geometry_msgs.msg import Twist

class DirectionCheck(Node):

    def __init__(self):
        super().__init__("check_direction_node")
        self.subscription_ = self.create_subscription(
            Twist,
            'turtle1/cmd_vel',
            self.twistCallback,
            10
        )
        self.publisher_ = self.create_publisher(Bool, 'linear_x_sign', 10)
        self.bool_msg = Bool()

    def twistCallback(self, msg):
        self.bool_msg.data = [msg.linear.x >= 0]
        if(self.bool_msg.data):
            self.get_logger().info("Bool Linear_x: True")
        else:
            self.get_logger().info("Bool Linear_x: False")
        self.publisher_.publish(self.bool_msg)

def main(args=None):
    rclpy.init(args=args)
    check_direction_node = DirectionCheck()
    rclpy.spin(check_direction_node)

    check_direction_node.destroy_node()
    rclpy.shutdown()

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

check_direction_node를 생성
publisher, subscriber을 node에서 생성 이를 callback함수로 전달

Callback함수를 통해 Message를 /turtle1/cmd_vel을 subscribe
했을 때 동작을 설명

ROS2 Topic 실습 (Python)

- ▶ (실습4) direction 노드를 만들어, turtle1/cmd_vel의 linear.x가 음수 일 때, std_msgs/Bool을 이용하여, “Bool을 False” 반대의 경우 True로 퍼블리시하는 publish-subscriber 노드를 만듭시다.

Setup.py 수정 (2차시 예제 참조)

```
entry_points={
    'console_scripts': [
        'pub_vel_node = topic_tutorial_py.pubvel:main',
        'sub_pose_node = topic_tutorial_py.subpose:main',
        'check_direction_node = topic_tutorial_py.direction:main'
    ],
}
```

빌드

colcon build #ros2_ws에서

실행

\$ ros2 run topic_tutorial_py check_direction_node

ROS Topic 실습 (C++)

- ◉ (실습4) direction 노드를 만들어, turtle1/cmd_vel의 linear.x가 음수 일 때, std_msgs/Bool을 이용하여, “Bool을 False” 반대의 경우 True로 퍼블리시하는 publish-subscriber 노드를 만듭시다.

direction.py 만들기 (2차시 예제 참조)

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/bool.hpp"
#include "geometry_msgs/msg/twist.hpp"

class DirectionCheck : public rclcpp::Node
{
public:
    DirectionCheck() : Node("check_direction_node")
    {
        subscription_ = this->create_subscription<geometry_msgs::msg::Twist>(
            "turtle1/cmd_vel", 10, std::bind(&DirectionCheck::twistCallback, this, std::placeholders::_1));

        publisher_ = this->create_publisher<std_msgs::msg::Bool>("linear_x_sign", 10);

        RCLCPP_INFO(this->get_logger(), "DirectionCheck node has been started.");
    }

private:
    void twistCallback(const geometry_msgs::msg::Twist::SharedPtr msg)
    {
        std_msgs::msg::Bool bool_msg;
        bool_msg.data = (msg->linear.x >= 0);

        if (bool_msg.data)
        {
            RCLCPP_INFO(this->get_logger(), "Bool Linear_x: True");
        }
        else
        {
            RCLCPP_INFO(this->get_logger(), "Bool Linear_x: False");
        }

        publisher_>publish(bool_msg);
    }

    rclcpp::Subscription<geometry_msgs::msg::Twist::SharedPtr> subscription_;
    rclcpp::Publisher<std_msgs::msg::Bool::SharedPtr> publisher_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);

    auto node = std::make_shared<DirectionCheck>();
    rclcpp::spin(node);

    rclcpp::shutdown();
    return 0;
}
```

check_direction_node를 생성
publisher, subscriber을 node에서 생성 이를 callback함수로 전달

Callback함수를 통해 Message를 /turtle1/cmd_vel을 subscribe
했을 때 동작을 설명

ROS2 Topic 실습 (C++)

- (실습4) direction 노드를 만들어, turtle1/cmd_vel의 linear.x가 음수 일 때, std_msgs/Bool을 이용하여, “Bool을 False” 반대의 경우 True로 퍼블리시하는 publish-subscriber 노드를 만듭시다.

Cmakelists.txt 수정 (2차시 예제 참조)

```
add_executable(check_direction_node src/direction.cpp)
ament_target_dependencies(check_direction_node rclcpp turtlesim
geometry_msgs std_msgs)
```

```
install(TARGETS
check_direction_node
DESTINATION lib/${PROJECT_NAME})
```

빌드

```
colcon build #ros2_ws에서
```

실행

```
$ ros2 run topic_tutorial_cpp check_direction_node
```

감사합니다

