ROBOT SPORT GAME TEAM

Team. RO:BIT | **ROS2**

18 기 정유정

ROS2

ROS : Robot Operating System 의 약자 . 로봇 응용 프로그램 개발을 위한 프레임워크

-Node
: 최소 단위의 실행 가능한 프로세스를 가리키는 용어로서 **하나의 실행 가능한 프로그램**

ROS 시스템을 위해서는 노드와 노드 사이에 입력과 출력 데이터를 서로 주고받게 설계해야만 함 . 여기서 주고받는 데이터를 메시지 (message) 라고 하고 주고받는 방식을 메시지 통신 (message communication) 이라고 함 . 메시지는 변수 형태 (int, float point, boolean, string) 로 구성함

각 노드들이 메시지 통신으로 서로 유기적으로 연결되어 작동함 .

통신 방법 : 토픽 (topic), 서비스 (service), 액션 (action), 파라미터 (parameter)

메시지 통신

-Topic(msg)

: 비동기식 단방향 메시지 송수신 방식

Publisher : 메시지를 발행
Subscriber : 발행된 메시지를 수신

- Service(srv)

: 양방향 메시지 송수신 방식
client 는 서비스를 요청 , server 는 서비스에 응답하고 요청받은 일을 수행한 뒤 결과값을 다시 전달

-Action

: 비동기식 + 동기식 양방향 메시지 송수신 방식
client 는 목표를 지정 , server 는 결과 전송 외에도 작업 수행 중 feedback 전송 가능

-Parameter

: 노드 내부 또는 외부에서 파라미터 설정 및 수정 가능

메시지 통신

- 현재 실행되고 있는 노드 , 토픽 – 서비스 – 액션

  $ ros2 node list

  $ ros2 topic list

  $ ros2 service list

  $ ros2 action list

- 노드 , 토픽 , 액션을 gui 로 확인 가능

  $ rqt_graph

- 토픽 내용을 실시간으로 확인 가능

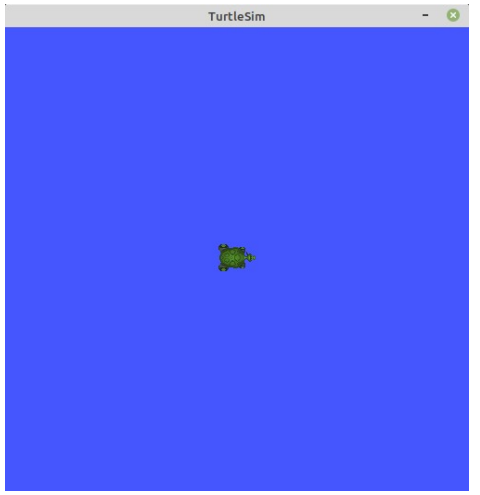
  $ ros2 topic echo [topic name]

  ex) ros2 topic echo /turtle1/cmd_vel

  $ rqt

  : 주기 (Hz), 대역폭 (bandwidth) 등 확인 가능

Turtlesim



- 패키지 설치

$ sudo apt update
$ sudo apt install ros-humble-turtlesim

$ ros2 pkg executables turtlesim : turtlesim 패키지에 포함되어 있는 노드 확인 가능

- draw_square : 사각형 모양으로 turtle 을 움직이게 하는 노드
- mimic : 유저가 지정한 토픽으로 동일 움직임의 turtlesim node 복수개 실행시킬 수 있는 노드
- turtle_teleop_key : turtle 을 움직이게 하는 노드 , 속도값을 퍼블리쉬
- turtlesim_node : teleop_key 노드로부터 속도값을 토픽으로 받아 움직이게 하는 2D 시뮬레이터 노드

- 노드 실행

$ ros2 run turtlesim turtlesim_node : ros2 run [package name] [node name]

$ ros2 run turtlesim turtle_teleop_key

- 토픽 발행

$ ros2 topic pub ⟨topic_name⟩ ⟨msg_type⟩ "⟨args⟩"
ex) $ 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}}"

C++ 패키지

- c++ 패키지 생성

$ ros2 pkg create [package 이름] --build-type [빌드 타입] --dependencies [의존하는패키지 1] [의존하는패키지 n]

$ cd ~/colcon_ws/src/

$ ros2 pkg create my_first_ros_rclcpp_pkg --build-type ament_cmake --dependencies rclcpp std_msgs

$ touch src/my_cpp_node.cpp : 파일 생성

```
.
├── include
│   └── my_first_ros_rclcpp_pkg
├── src
├── CMakeLists.txt
└── package.xml
```

C++ 패키지

- CMakeLists

find_package(rclcpp REQUIRED) : std_msgs 등 패키지 의존성 설정

add_executable(my_cpp_node src/my_cpp_node.cpp)

ament_target_dependencies(my_cpp_node rclcpp)

install(TARGETS
my_cpp_node
DESTINATION lib/${PROJECT_NAME})

- package.xml

<build_depend>rclcpp</build_depend>
<exec_depend>rclcpp</exec_depend>

or

<depend>rclcpp</depend>

```
<buildtool_depend>ament_cmake</buildtool_depend>

<depend>rclcpp</depend>
<depend>std_msgs</depend>

<test_depend>ament_lint_auto</test_depend>
<test_depend>ament_lint_common</test_depend>
```

C++ 패키지

- 코드 작성

$ .vscode

$ gedit my_cpp_node.cpp

$ nano my_cpp_node.cpp

```cpp
#include "rclcpp/rclcpp.hpp"

class MyCppNode : public rclcpp::Node
{
public:
    MyCppNode() : Node("my_cpp_node")
    {
        RCLCPP_INFO(this->get_logger(), "Hello, ROS 2 C++ Node!");
    }
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<MyCppNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

```cpp
MyNode::MyNode() : Node("mynode")
{
  publisher = this->create_publisher<std_msgs::msg::String>("topicname",10);

}
MyNode::~MyNode()
{

}
```

Publisher, Subscriber 노드 작성

ROBIT ROBOT SPORT GAME TEAM

- publisher

.hpp

```
mycpp_publisher_ = this->create_publisher<std_msgs::msg::String>("topicname", 10);
timer_ = this->create_wall_timer(1s, std::bind(&MyCppNode::timer_callback,this));
```

```
private:
    rclcpp::TimerBase::SharedPtr timer_;
    void timer_callback();

    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr mycpp_publisher_;
```

.cpp

```
void MyCppNode::timer_callback()
  {
    auto msg = std_msgs::msg::String();
    msg.data = "Hello World: " + std::to_string(count_++);
    RCLCPP_INFO(this->get_logger(), "Published message: '%s'", msg.data.c_str());
    mycpp_publisher_->publish(msg);
  }
```

- subscriber

.hpp

```
mycpp_subscriber_ = this->create_subscription<std_msgs::msg::String>(
      "topicname",
      10,
      std::bind(&MyCppSubscriber::topic_callback, this, std::placeholders::_1));
```

```
private:
    void topic_callback(const std_msgs::msg::String::SharedPtr msg);
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr mycpp_subscriber_;
```

.cpp

```
void topic_callback(const std_msgs::msg::String::SharedPtr msg)
{
    RCLCPP_INFO(this->get_logger(), "Received message: '%s'", msg->data.c_str());
}
```

Python 패키지

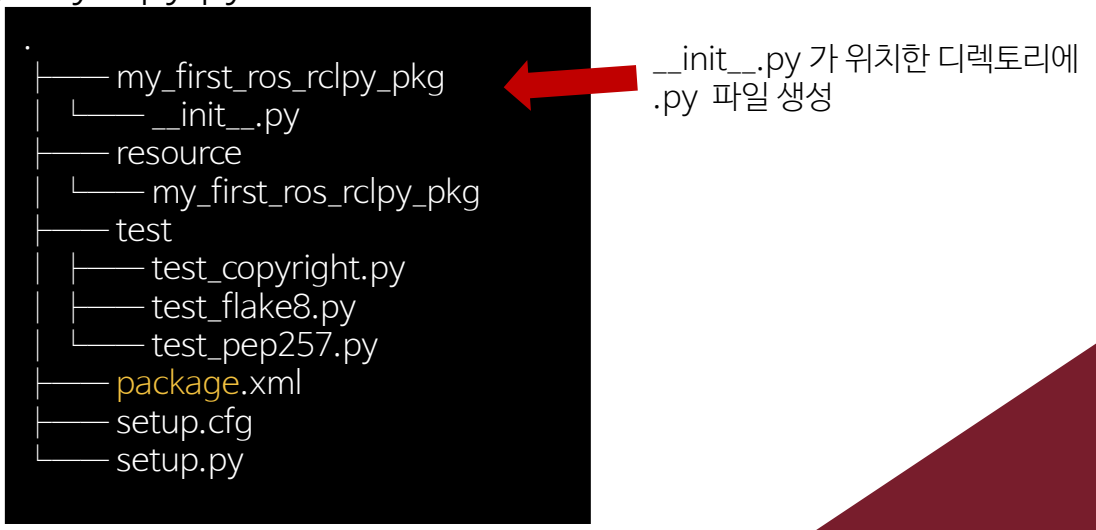
- python 패키지 생성

$ ros2 pkg create [package 이름] --build-type [빌드 타입] --dependencies [의존하는패키지 1] [의존하는패키지 n]

$ cd ~/colcon_ws/src/

$ ros2 pkg create my_first_ros_rclpy_pkg --build-type ament_python --dependencies rclpy std_msgs

$ touch my_first_ros_rclpy_pkg/my_first_ros_rclpy_pkg /myrclpy.py



Python 패키지

- package.xml

```
  <depend>rclpy</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

Python 패키지

- setup.py 작성

entry_points 옵션의 console_scripts 키를 사용한 실행 파일 설정

[script name] = [pkg name].[node name]:main

실행 시
$ ros2 run [node_name] [script_name] : $ ros2 run my_first_ros_rclpy_pkg myrclpy_node

```
entry_points={
    'console_scripts': [
        'myrclpy_node = my_first_ros_rclpy_pkg.helloworld_publisher:main',
    ],
},
```

Python 패키지

- setup.cfg 작성

'my_first_ros_rclpy_pkg' 와 같이 패키지 이름을 기재
colcon 를 이용하여 빌드 시 지정 폴더에 실행 파일이 생성 (/home/username/colcon_ws/install/my_first_ros_rclpy_pkg/lib/my_first_ros_rclpy_pkg)

```
[develop]
script-dir=$base/lib/my_first_ros_rclpy_pkg
[install]
install-scripts=$base/lib/my_first_ros_rclpy_pkg
```

Publisher, Subscriber 노드 작성

ROBIT ROBOT SPORT GAME TEAM

- publisher

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class HelloworldPublisher(Node):

    def __init__(self):
        super().__init__('helloworld_publisher')
        self.helloworld_publisher = self.create_publisher(String, 'helloworld', 10)
        self.timer = self.create_timer(1, self.publish_helloworld_msg)
        self.count = 0

    def publish_helloworld_msg(self):
        msg = String()
        msg.data = 'Hello World: {0}'.format(self.count)
        self.helloworld_publisher.publish(msg)
        self.get_logger().info('Published message: {0}'.format(msg.data))
        self.count += 1


def main(args=None):
    rclpy.init(args=args)
    node = HelloworldPublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

- subscriber

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class HelloworldSubscriber(Node):

    def __init__(self):
        super().__init__('Helloworld_subscriber')
        self.helloworld_subscriber = self.create_subscription(
            String,
            'helloworld',
            self.subscribe_topic_message,
            10)

    def subscribe_topic_message(self, msg):
        self.get_logger().info('Received message: {0}'.format(msg.data))


def main(args=None):
    rclpy.init(args=args)
    node = HelloworldSubscriber()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Keyboard Interrupt (SIGINT)')
    finally:
        node.destroy_node()
        rclpy.shutdown()


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

빌드 및 실행

```
$ cd ~/colcon_ws
$ colcon build --packages-select my_cpp_pkg

$ source ~/colcon_ws/install/setup.bash
$ ros2 run my_cpp_pkg my_cpp_node
```

: 워크스페이스의 my_cpp_pkg 만 빌드

: 워크스페이스 환경 설정

: my_cpp_node 실행

과제 1

- 터미널에서 토픽 발행을 통해 삼각형 , 원 , 사각형 그리기

turtlesim 실행해보면서 토픽 통신이 어떤 식으로 이루어지는지 보고서로 자세히 작성

* 각 과제별 보고서 작성 필수
* 소스 파일과 헤더 파일 분리하여 작성

과제 2

- Topic 통신 구현

c++ 패키지 제작하여 publisher, subscriber 노드 작성
python 패키지 제작하여 publisher, subscriber 노드 작성

콜백 받은 데이터를 터미널에 출력
토픽 통신 시 메시지 형태는 int, string, float 등 최소 3 개 이상 주고 받을 것

- 하나의 패키지 내에서 2 개의 노드를 통해 토픽 통신하는 과정
-std_msgs 를 활용해 c++ 패키지와 python 패키지가 서로 토픽 통신하는 과정
위 두 과정 모두 구현할 것

* 각 과제별 보고서 작성 필수
* 소스 파일과 헤더 파일 분리하여 작성

과제 3

- turtlesim 패키지를 활용해 그림 그리기

Turtlesim 에서 제공하는 토픽 메시지를 활용해 그림 그리는 패키지 제작

ros2 run 을 통해 제작한 패키지 실행 시 turtlesim 2d simulator 에 그림이 그려지도록 작성

WASD 키보드 입력받아 도형 그리기
삼각형 , 사각형 , 원 등 그리도록 작성
도형마다 굵기 , 색상 다르게 설정

패키지는 C++, python 중 택 1

* 각 과제별 보고서 작성 필수
* 소스 파일과 헤더 파일 분리하여 작성