HW2 보고서

로빛 20기 인턴 모시온

2025407006 로봇학부

목차

1. 개요

본 보고서는 이전 과제 1의 정보를 참고하여 각각
하나의 C++ 패키지 안에서의 발행 노드 및 구독 노드 토픽 통신
하나의 C++ 패키지 및 파이썬 패키지의 서로의 발행 노드 및 구독 노드 토픽 통신

(하나의 패키지 내의 C++ 소스코드 내의 발행 노드, 구독 노드 및 파이썬 소스코드 내의 발행 및 구독 노드 상호 통신 -> 해당 부분은 구현하지 못하였습니다.)

을 구현하여 직접적으로 토픽 이름을 지정하여 단일 발행 노드와 구독 노드 둘만의 상호통신을 구성할 수 있도록 실습하는 데에 목적을 두었습니다.

2. 코드

코드 설명에 앞서 본 프로젝트의 폴더 구성은 다음과 같습니다.
C++ 패키지 -> hw2
파이썬 패키지 -> hw2_py
C++와 파이썬 통합 패키지(미구현) -> cpp_py

또한 본 프로젝트는 hw2만을 구동하여 하나의 C++ 패키지 내의 발행 및 구독 노드간의 통신을 구현이 가능하며, 추가적으로 hw2_py를 구동하면 C++과 파이썬 간의 상호 토픽 통신이 가능하도록 구성하였습니다.

(hw2의 C++ 발행 노드가 hw2 내부 구독노드(Cpp_SUB2)와 hw2_py의 구독 노드 PY_SUB에 토픽 통신을, hw2_py의 발행 노드와 hw2의 C++ 구독 노드와 토픽 통신을 하는 구조입니다.)

2-1. P1(C++)

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>
```

우선 hw2의 P1(publisher) 코드입니다.
해당 코드 및 발행 노드의 코드(S1, S2) 또한 해당 모듈을 추가적으로 호출하였습니다.

각각 다음의 사용을 위해서 호출하였습니다.
chrono: 토픽을 전송할 시간 간격(초)를 표기/표시하기 위하여 선언
functional: 토픽을 다시 선언하기 위한 롤백 또는 바인딩을 위해 선언
예: create_wall_timer(1s, std::bind(&Publisher1::timer_callback, this));

memory: 포인터의 기능으로서 각각의 노드를 예제 표시와 달리

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

rclcpp::TimerBase::SharedPtr timer_;
rclcpp::Publisher<std_msgs::msg::String>::SharedPtr string_publisher_;
와 같이 포인터를 통해 지정하여 표기하기 위해 선언하였으며 또한 별도의 소멸자 선언없이 자동으로 메모리를 해제한다는 점에 의해 사용하였습니다.

```
class Publisher1 : public rclcpp::Node
{
  public:
    Publisher1()
    : Node("Cpp_PUB"), count_(0)
    {
      string_publisher_ = this->create_publisher<std_msgs::msg::String>("cpps", 10);
      int_publisher_ = this->create_publisher<std_msgs::msg::Int32>("cppi",10);
      float_publisher_ = this->create_publisher<std_msgs::msg::Float32>("cppf",10);
      timer_ = this->create_wall_timer(1s, std::bind(&Publisher1::timer_callback, this));
    }
```

발행 노드의 Class 구성입니다.
노드의 이름을 Cpp_PUB로서 명명하며, 공지 사항에 알맞게 서로 다른 3 종류 이상의 자료(정수, 실수, 문자열)를 각각 cpps(문자열), cppi(정수형), cppf(실수형)으로 각각의 토픽이름을 지정하여 이후 해당 토픽을 받을 Cpp_SUB2와 파이썬의 구독 노드에서 받을 수 있도록 하였습니다.

각각의 발행 노드를 create_publisher를 통해 생성하며, std_msgs의 사용 공지에 따라 각각의 자료형을 선언하였습니다.
(해당 토픽 3개는 1초 간격으로 전송됩니다.)

```
private:
  void timer_callback()
  {
    auto string1 = std_msgs::msg::String();
    string1.data = "From_Cpp";
    RCLCPP_INFO(this->get_logger(), "OUTPUT(S): '%s'", string1.data.c_str());
    string_publisher_->publish(string1);

    auto int1 = std_msgs::msg::Int32();
    int int1data = count_++;
    int1.data = int1data;
    RCLCPP_INFO(this->get_logger(), "OUTPUT(I): '%d'", int1.data);
    int_publisher_->publish(int1);

    auto float1 = std_msgs::msg::Float32();
    float1.data = 3.14f;
    RCLCPP_INFO(this->get_logger(), "OUTPUT(F): '%f'", float1.data);
    float_publisher_ ->publish(float1);
  }
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr string_publisher_;
  rclcpp::Publisher<std_msgs::msg::Int32>::SharedPtr int_publisher_;
  rclcpp::Publisher<std_msgs::msg::Float32>::SharedPtr float_publisher_;
  size_t count_;
```

타이머 콜백 함수 정의

1초 간격의 토픽 발행을 위한 함수 timer_callback의 정의입니다.

예제 표기에 따라 자료형을 자동으로 조정하는 auto를 통해 각각의 전송할 정보를 string1, int1, float1 변수로 저장하여 각각의 생성된 객체(각각의 publisher)로 전송되어 발행될 수 있도록 하였습니다.

RCLCPP_INFO의 사용은 각 발행할 정보들을 터미널에 출력하여 시각화할 수 있도록 하기 위한 사용이며, 예제 코드를 참고하였습니다.

2-2. S1(C++)

```
class Subscriber1 : public rclcpp::Node

public:
  Subscriber1()
  : Node("Cpp_SUB")
  {
    string_subscription_ = this->create_subscription<std_msgs::msg::String>("pys", 10, std::bind(&Subscriber1::string_callback, this, _1));
    int_subscription_ = this->create_subscription<std_msgs::msg::Int32>("pyi", 10, std::bind(&Subscriber1::int_callback,this,_1));
    float_subscription_ = this->create_subscription<std_msgs::msg::Float32>("pyf", 10, std::bind(&Subscriber1::float_callback, this, _1));
  }
```

파이썬의 발행 노드 토픽 통신을 받아 C++의 구독 노드로 받는 S1의 Class 입니다. 노드 명칭을 Cpp_SUB로, 각각 전달받을 자료형들을 파이썬에서 앞으로 전달할 토픽 이름 pys(문자열), pyi(정수형), pyf(실수형)으로 받도록 지정하였으며 이에 따른 구독 메시지를 생성하였습니다.

```
private:
  void string_callback(const std_msgs::msg::String & msg) const
  {
    RCLCPP_INFO(this->get_logger(), "From_PYS: '%s'", msg.data.c_str());
  }
  void int_callback(const std_msgs::msg::Int32 & msg) const
  {
    RCLCPP_INFO(this->get_logger(), "From_PYI: '%d'", msg.data);
  }
  void float_callback(const std_msgs::msg::Float32 & msg) const
  {
    RCLCPP_INFO(this->get_logger(), "From_PYF: '%f'", msg.data);
  }
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr string_subscription_;
  rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr int_subscription_;
  rclcpp::Subscription<std_msgs::msg::Float32>::SharedPtr float_subscription_;
```

해당 함수 또한 토픽 통신으로 받게되는 정보를 처리하기 위하여 각각의 자료형에 따른 함수 (자료형 종류)_callback을 정의하여 사용하였습니다.

전달받게 되는 메시지 종류를 순서대로 String, Int32, Float32형태로 해석하도록 하였으며 받은 정보를 RCLCPP_INFO를 통해 포매팅을 하여 터미널 로그에 출력되도록 하였습니다.

2-3. S2(C++)

```
class Subscriber1 : public rclcpp::Node
{
  public:
    Subscriber1()
    : Node("Cpp_SUB2")
    {
      string_subscription_ = this->create_subscription<std_msgs::msg::String>("cpps", 10, std::bind(&Subscriber1::string_callback, this, _1));
      int_subscription_ = this->create_subscription<std_msgs::msg::Int32>("cppi", 10, std::bind(&Subscriber1::int_callback,this,_1));
      float_subscription_ = this->create_subscription<std_msgs::msg::Float32>("cppf", 10, std::bind(&Subscriber1::float_callback, this, _1));
    }
```

파이썬의 발행 노드의 토픽 통신이 아닌 내부 동일 패키지의 P1(C++)의 발행 노드에 따른 구독 노드 코드입니다.

cpp의 발행 노드를 통해 값을 받기 위하여 cpps, cppi, cppf의 토픽들을 받도록 하였습니다.

```
private:
  void string_callback(const std_msgs::msg::String & msg) const
  {
    RCLCPP_INFO(this->get_logger(), "From_CppS: '%s'", msg.data.c_str());
  }
  void int_callback(const std_msgs::msg::Int32 & msg) const
  {
    RCLCPP_INFO(this->get_logger(), "From_CppI: '%d'", msg.data);
  }
  void float_callback(const std_msgs::msg::Float32 & msg) const
  {
    RCLCPP_INFO(this->get_logger(), "From_CppF: '%f'", msg.data);
  }
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr string_subscription_;
  rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr int_subscription_;
  rclcpp::Subscription<std_msgs::msg::Float32>::SharedPtr float_subscription_;
```

각각의 자료형에 따른 정보를 터미널에 출력하기 위한 각각의 콜백함수입니다.

구성의 이전의 S1과 동일합니다.

터미널의 동시 출력 시 이를 구분 지을 수 있도록 표시 정보를 다르게 변경하였습니다.(From_Cpp_)

2-4. P1(Python)

```
class py_string(Node):
    def __init__(self):
        super().__init__('PY_PUB')
        self.string_publisher = self.create_publisher(String, 'pys', 10)
        self.int_publisher = self.create_publisher(Int32, 'pyi', 10)
        self.float_publisher = self.create_publisher(Float32, 'pyf',10)
        self.timer = self.create_timer(1, self.publish_string_msg)
        self.count = 0
```

이전 S1에서 받을 토픽명을 해당 코드에서는 발행하므로 publisher 생성을 각각 pys, pyi, pyf를 통해 각각의 publisher 메시지로 생성하였습니다.

(py_string의 class 명칭은 해당 코드 완성 이전 helloworld 출력 발행 노드 예제 코드를 기반으로 완성하여 이에 따른 종속성을 명칭을 수정함으로서 빌드 오류 및 기타 오류를 발생하지 않도록 그대로 유지하였습니다.)

```
def publish_string_msg(self):
    msg_s = String()
    msg_s.data = 'From_PY'

    msg_i = Int32()
    msg_i.data = self.count

    msg_f = Float32()
    msg_f.data = 2.414

    self.string_publisher.publish(msg_s)
    self.int_publisher.publish(msg_i)
    self.float_publisher.publish(msg_f)
    self.get_logger().info('{0}'.format(msg_s.data))
    self.get_logger().info('{0}'.format(msg_i.data))
    self.get_logger().info('{0}'.format(msg_f.data))
    self.count += 1
```

각 자료형의 정보를 저장할 변수들을 지정함과 각각 지정된 값을 저장하도록 하였습니다.

msg_s = From_PY - 송신자

msg_i.data = self.count() - 송신 횟수(32비트형에 따라 .data를 통해 포매팅을 하여 형식(비트 단위)의 차이에 따른 오류를 일으키지 않도록 하였습니다.)

msg_f = 2.414 - 루트2 값

해당 정보들이 발행될 수 있도록 생성된 각 자료형에 알맞은 토픽 메시지를 발행하도록 하였습니다.

2-5. S1(Python)

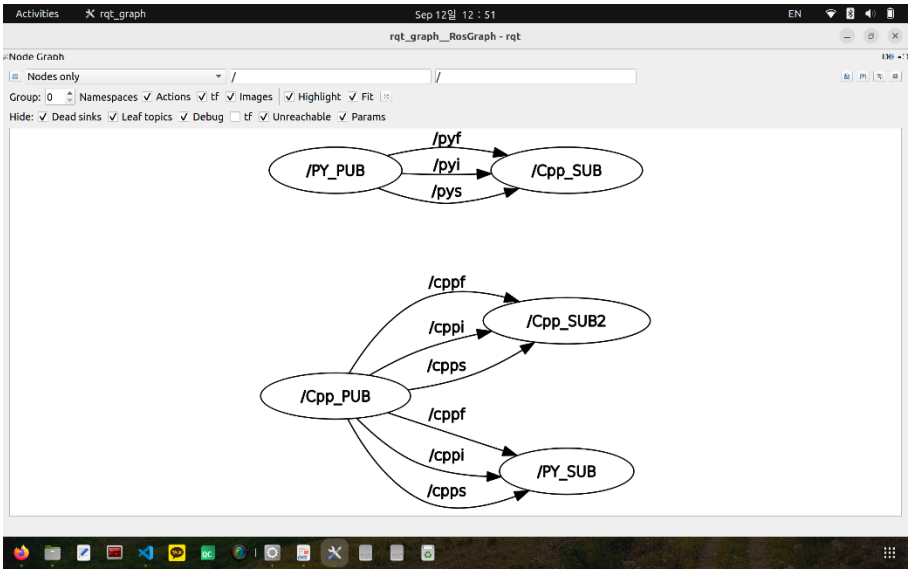
```
class PY_SUB(Node):
    def __init__(self):
        super().__init__('PY_SUB')
        self.string_subscriber = self.create_subscription(String, 'cpps', self.string_message, 10)
        self.int_subscriber = self.create_subscription(Int32, 'cppi', self.int_message, 10)
        self.float_subscriber = self.create_subscription(Float32, 'cppf', self.float_message, 10)
```

각 전달받는 자료형의 정보에 따라 각각의 토픽을 구독하고 메시지가 해석되도록 하였습니다. P1(C++)의 발행 토픽 명에 알맞게 각각의 subcription을 지정하여 생성하였습니다.

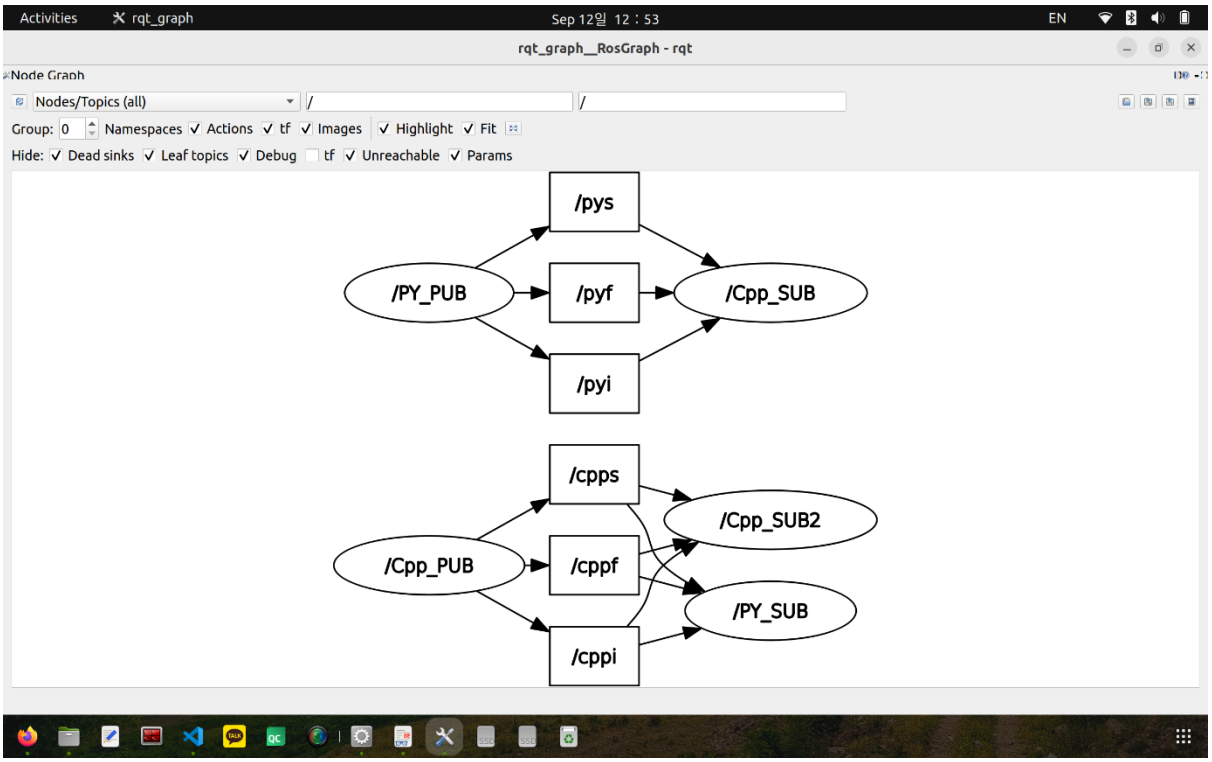
```
def string_message(self,msg):
    self.get_logger().info('{0}'.format(msg.data))
def int_message(self, msg):
    self.get_logger().info('{0}'.format(msg.data))
def float_message(self, msg):
    self.get_logger().info('{0}'.format(msg.data))
```

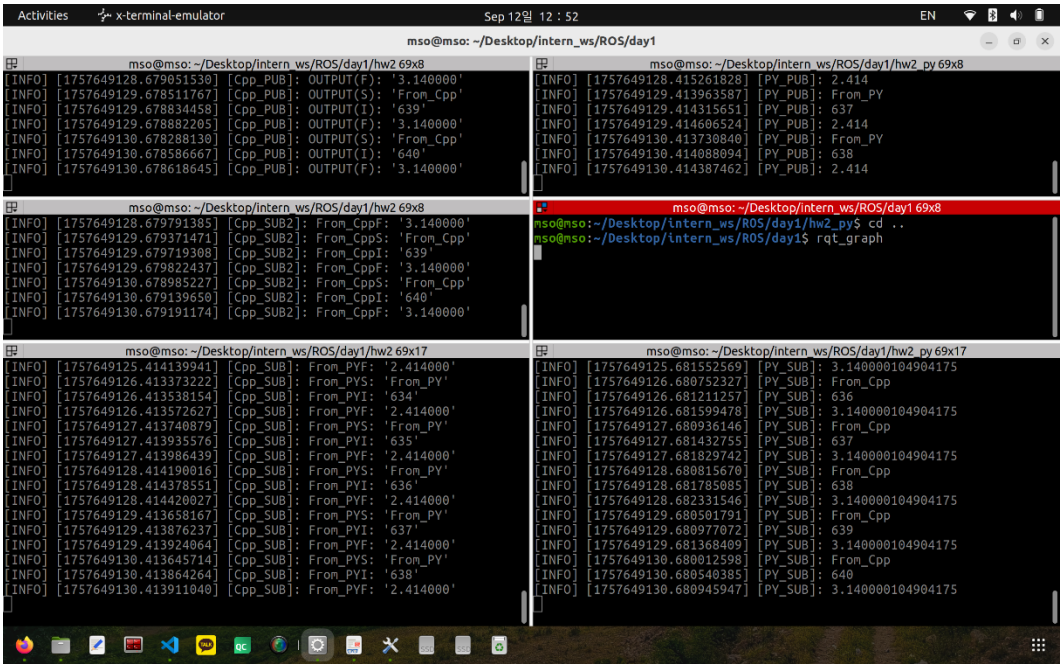
토픽의 구독에 따라 전달받는 정보를 토대로 터미널의 로그에 포매팅하여 출력되도록 하였습니다.

3. 결과



C++의 발행 노드에 따른 각 자료형의 토픽이 각각 C++ 패키지 내의 구독 노드(Cpp_SUB2)와 파이썬 패키지 내의 구독 노드(S1)에 전달되며, 반대로 파이썬의 발행 노드(P1)의 각 토픽이 C++내의 구독 노드 (Cpp_SUB)에 전달되는 과정입니다.





실제 동작입니다.