ROS2 day4 hw1 ros2 lifecycle + Qos 과제 보고서

로빛 20기 인턴 현창석

목차

- Lifecycle Node란?

Lifecycle Node (생명주기 노드) 는 정의된 상태 머신(STM)을 갖춘 관리형 노드이다. 일반 노드와 달리, 라이프사이클 노드는 명시적으로 상태와 전환을 정의하여 초기화, 실행, 종료 중에 동작을 세밀하게 제어할 수 있다.
처음에 구성되면, 구성되지 않은 노드는 비활성 상태로 전환된다. 다시 말해서 노드는 작동할 준비가 되어 있지만 콜백 함수 및 퍼블리셔는 작동하지 않는다. 값을 통해 동작을 수행하지 않고 이전의 동작을 이어서 한다. 매개 변수를 변경하고, 퍼블리시와 섭스크라이버 등을 추가하고 또는 제거하는 노드를 재구성할 수 있다.

이러한 기능은 로봇의 에너지 절약 측면에서 효과적이다 . 로봇 청소기가 방을 청소하기위해 방의 지도를 그리는 과정은 청소를 하기 전 단 한번만 필요하다 . 그 이후로는 맵을 만드는 노드를 off 시켜 주어야한다. 이렇게 해야 로봇의 에너지를 절약할 수 있다.

그럼 이제 노드 상태를 나타내는 용어를 살펴보겠다.

Unconfigured : node가 객체화(instantiated)된 직후의 상태이다. node class의 생성자가 호출됐다고 생각하면 된다. 이 상태에서는 변수 등의 선언만 형식적으로 되어 있다

nactive : node내 parameter를 변경하거나, 특정 topic의 publication/subscription을 생성/삭제 하거나, service를 생성/삭제 하는 등 system구성이 이루어진 상태이다. 다만 생성/삭제만 됐을 뿐 실제로 데이터를 구독하거나 service를 처리하진 않는 상태이다.

Active : 실제로 node가 기능을 수행할 때 상태이다. 기능을 수행하기 위해 데이터를 읽고 처리하거나, service 요청을 처리, output을 만들어낸다.

Finalized : node가 완전히 없어지기 직전 상태이다. 이 상태는 "shutdown"을 통해서 도달할 수도 있지만, error가 발생해서도 도달 할 수 있는 상태로써 error등을 잡는 debugging 용도로 사용되기 위한 상태이다.

- QoS란?

ROS2 QoS(Quality of Service)는 DDS 미들웨어를 통해 노드 간 데이터 통신의 신뢰성, 내구성을 조절하여 로봇 및 시스템의 안정성과 효율성을 높이는 데 활용된다.
Q0s는 각 상황마다 사용되는 분야가 다르다 . 안정적인 통신을 원한다면 Reliable을 사용한다.
데이터가 유실되는 것을 허용하면서 빠른 통신 속도를 중요시 하는 상황에서는
Best-effort을 사용한다. 주로 실시간 영상을 전송할 때 사용한다.

데이터를 유지해야하는 상황이라면 Data-local-durable을사용한다. 노드가 시작하기 전 데이터까지 서브스크라이버에게 전송한다. 주로 처음 시작 노드에 과거의 데이터를 전송할 때 사용한다

시간과 관련한 분야도 있다 .

첫 번째는 History이다. 이것은 수신자가 버퍼에 저장되는 데이터의 개수를 지정한다. 두 번째는 Deadline이다. 단어 그대로 데이터 전송 또는 수신에 시간 제약을 건다.
마지막 세번 째 로 Liveliness이다. 이는 노드의 활성 상태를 감지하여 통신 두절 시 알림을 출력한다. ,

lifecycle + Qos publisher/subscriber 코드

먼저 hpp 파일의 코드에 대해 설명하겠다
일반노드를 생성할 때 필요했던 헤더 파일들을 선언해주고 cycle 노드를 생성하기 위하여 rclcpp_lifecycle/lifecycle_node.hpp 와 rclcpp_lifecycle/lifecycle_publisher.hpp 을 선언해주었다.

● class MinimalPublisher : public rclcpp_lifecycle::LifecycleNode
라이프사이클 노드를 상속하는 클래스 선언한다. 생성자와 소멸자 선언 후

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn on_configure(const rclcpp_lifecycle::State &state);

● 노드의 상태를 unconfigure 상태에서 inactive 상태로 변환하는 함수 on_configure 을 선언한다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn on_activate(const rclcpp_lifecycle::State &state);

이 함수는 inactive -> active로 lifecycle node의 상태 변환해주는 함수이다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn on_deactivate(const rclcpp_lifecycle::State &state);

이 함수는 active에서 inactive로 node의 상태를 변환해주는 함수이다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn on_cleanup(const rclcpp_lifecycle::State &state);

이 함수는 inactive에서 unconfigured로 변경하는 함수이다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn on_shutdown(const rclcpp_lifecycle::State &state);

이 함수는 노드 소멸 전 잔여 값들을 지우는 함수이다.

private : 란에는
●void timer_callback();
퍼블리시하는콜백함수를 선언하였다.

그 아래에는
●rclcpp::TimerBase::SharedPtr timer_;

메세지를 퍼블리시하는데 필요한 timer 를 선언해주었고

●std::shared_ptr<rclcpp_lifecycle::LifecyclePublisher<std_msgs::msg::String>> publisher_;

메세지를 퍼블리시하는 퍼블리셔, 그리고 이를 활성화/비활성화 할 수 있는 publisher : LifecyclePublisher 타입으로 선언해주었다.

●class MinimalSubscriber : public rclcpp_lifecycle::LifecycleNode

아래 클래스는 서브스크라이버를 구현한 클래스이다. 이 클래스 또한 위에 퍼블리셔를 선언한 클래스 같이 라이프사이클 노드를 상속한다.

public에서는 MinimalSubscriber(); 셍성자와
~MinimalSubscriber(); 소멸자를 선언해준후

퍼블리셔 클래스과 동일하게 lifecycle 함수들을 선언한다 .

●void topic_callback(const std_msgs::msg::String & msg) const;

private 란에서는 토픽으로 부터 값을 받아 명령을 실행하는 토픽 콜백함수를 선언한다.

●rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_; 섭스크라이버의 스마트포인터를 설정해준 후 마지막으로

●bool m_isActivated = false; active 인지 inactive 인지 정의하는 bool함수를 선언하였다.

다음 설명할 것은 lifecycle_pub.cpp 파일의 코드 내용이다.
먼저 생성자 부분이다.

●MinimalPublisher::MinimalPublisher() :
rclcpp_lifecycle::LifecycleNode("minimal_publisher"), count_(0) {

노드의 이름 설정, 카운트를 초기화를 해주고 아래 코드를보면 퍼블리셔와 타이머 생성하는 부분이 주석처리가 되어있는것을 볼 수 있다 . ;
본래 노드를 생성할 때 선언하던 부분을 사용하지 않기 때문이다 .

lifecycle을 구동시키는 함수를 살펴보겠다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
MinimalPublisher::on_configure(const rclcpp_lifecycle::State &state)
먼저 이 함수는 노드의 상태를 unconfigure 상태에서 inactive 상태로 변환해주는 함수로 바로 이곳에서 publisher/subscriber, timer, service 등을 생성한다
.
● publisher_ = this -> create_publisher <std_msgs::msg::String>("topic", rclcpp::SensorDataQoS());
topic 이라는 이름의 토픽에 퍼블리시하는 퍼블리셔를 생성한다. 이때 퍼블리셔를 . QoS 중에서 베스트 에포트를 사용하기 위해 기존에 버퍼를 입력하던 자리에 SensorDataQoS입력하여 빠른 통신을 우선시 하도록 하였다.
timer_ = this -> create_wall_timer(1s,std::bind(&MinimalPublisher::timer_callback,this));
일정 시간마다 퍼블리시를 하기 위해 1초마다 작동해 콜백함수를 호출하는 타이머의 생성부분이다,

●RCUTILS_LOG_INFO_NAMED(this->get_name(), "on_configured () is called from state %s",state.label().c_str());

이 코드에서는 함수가 실행되었을 때 현재 노드의 상태가 on_configured 라는 것을 로그에 출력한다 .

함수가 성공적으로 실행되면
● return ●●●
rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
성공적으로 동작했다는 것을 반환한다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
MinimalPublisher::on_activate(const rclcpp_lifecycle::State &state)

함수는 ;

●publisher_ -> on_activate()

inactive -> active로 lifecycle node의 상태 변환해준다. 실제로 publisher/subscriber 등이 데이터를 생성 및 처리한다.
그 이하는 위의 함수와 동일하다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
MinimalPublisher::on_deactivate(const rclcpp_lifecycle::State &state)

이 함수는 active 상태에서 inactive 상태로 노드의 상태를 박꾸는 함수이다.
●publisher_ -> on_deactivate();
따라서 timer callback에서 publisher_를 통해 퍼블리시 해도 기능을 하지 못한다.
그 이하는 위의 함수들과 동일하다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
MinimalPublisher::on_cleanup(const rclcpp_lifecycle::State &state)

이 함수는 노드의상태를 inactive에서 unconfigured로 변경하는 함수이며

●timer_.reset();
 이전에 생성했던 timer와
● publisher_.reset();
publisher의 메모리를 해제한다.

그 이하는 위의 함수들과 동일하다.

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn
MinimalPublisher::on_shutdown(const rclcpp_lifecycle::State &state)

이 함수는 node가 기능을 잃기전 안전하게 필요없는 메모리를 해제하는 역할하여
다시 한번 unconfigured 함수와 같이 이전에 생성했던 타이머와 퍼블리시의 메모리를 해제

한다..

●void MinimalPublisher::timer_callback () {
이 함수 에서는

●auto message = std_msgs::msg::String();
이렇게 선언한 메세지에

●message.data = std::string("message_tranported") + std::to_string(count_++);

스트링 형태의 메세지 데이터를 담아서

●if(!publisher_-> is_activated())

publisher_가 active상태인지 확인한후

상태에 따라서 로그를 출력한다.

●publisher_->publish(message);
액티브 상태라면 메시지 데이터를 퍼블리시한다 .

●int main(int argc, char *argv[])
메인함수에서는
● rclcpp::init(argc,argv);
초기화 해준 후
std::shared_ptr<MinimalPublisher>
● minimal_publisher = std::make_shared<MinimalPublisher>();
퍼블리셔객체를 선언한다
● rclcpp::spin(minimal_publisher->get_node_base_interface());

spin 사용시 기존 노드를 사용할 때와 다르게 class 자체를 넣는 것이 아닌
    //node의 base interface를 넣는다.

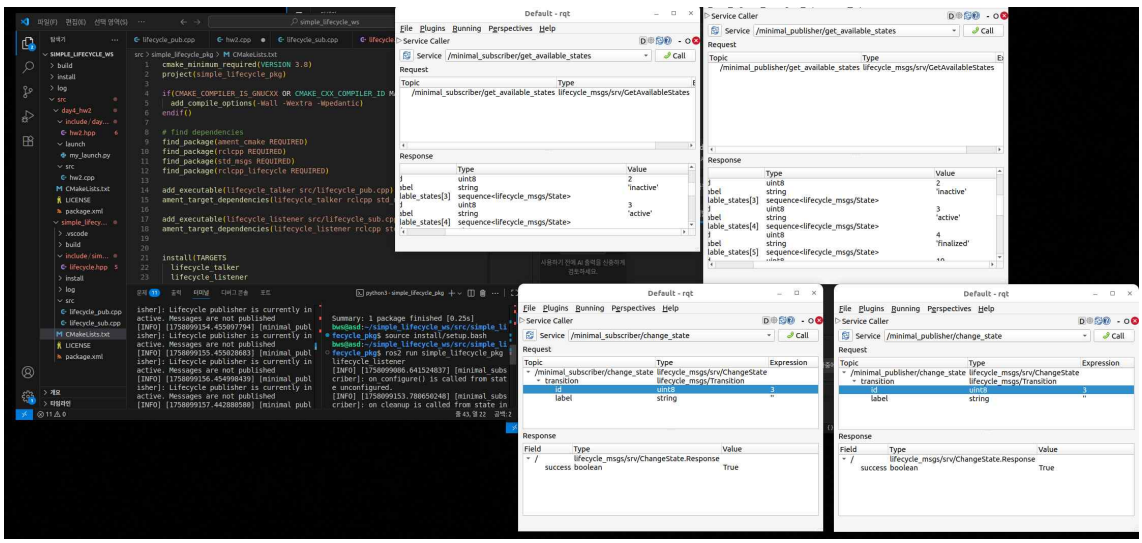
다음은 sub.cpp 파일에 대해서 설명하겠다 . pub 파일과 겹치는 부분이 많아서 차이점에 대해서 간략히 설명하도록 하겠다.

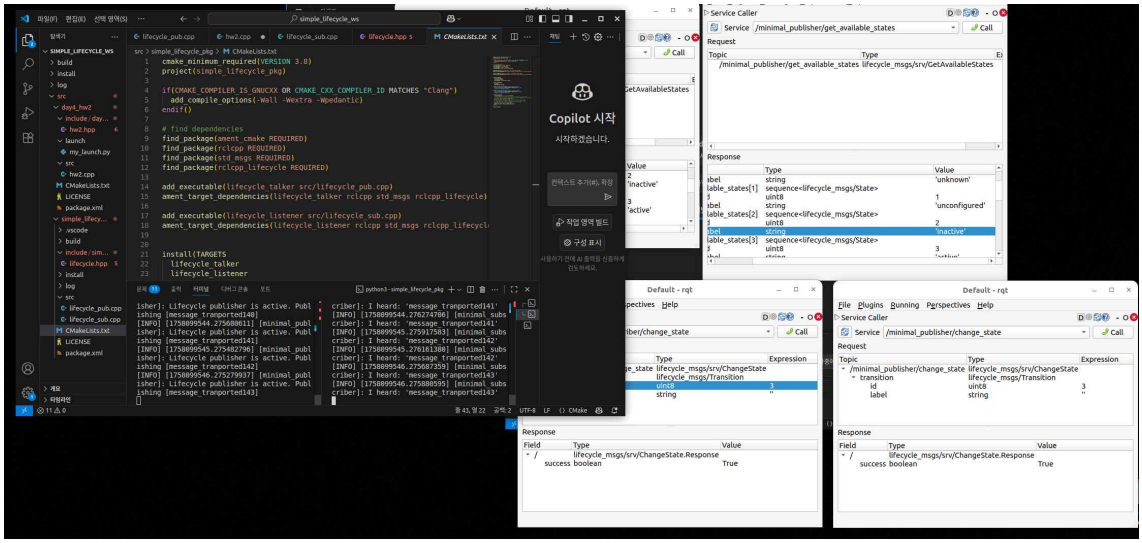
●using std::placeholders::_1;
이 코드를 선언하여 바인드 함수 사용시 첫 번째 인자를 항상 사용하겠다는 선언을 해주었다 .

●rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn MinimalSubscriber::on_configure(const rclcpp_lifecycle::State &state) 함수에서

●subscription_ = this->create_subscription<std_msgs::msg::String>(// lifecycle 정의상 subscription이 생성되는 것은 on_configure가 호출될 때 행해지는 것이 적절하기 때문이다

- 실행 결과




simple_lifecycle_ws
CMakeLists.txt
Copilot 시작
시작하겠습니다.
Service Caller
/minimal_publisher/get_available_states
/minimal_publisher/change_state
Default - rqt
Request
Response
Call