hw1_보고서

20기 인턴 송수민

목차

1. ROS 2 Lifecycle Node

1.1 배경

- ROS 2에서는 Managed Node 개념을 도입.
- 일반 노드와 달리 상태 기반 동작 -> 외부에서 configure, activate 등으로 제어.
- 주 목적: 하드웨어나 센서처럼 초기화·시작·종료 절차가 중요한 경우를 안정적으로 관리.

1.2 코드 구조 (Lifecycle Talker 예제)

Lifecycle 노드는 일반 노드(rclcpp::Node) 대신
rclcpp_lifecycle::LifecycleNode를 상속받는다.

① 클래스 정의

class LifecycleTalker : public rclcpp_lifecycle::LifecycleNode
-> 일반 노드와 달리 LifecycleNode를 상속해야 상태 관리 기능(콜백, 서비스 인터페이스 등)을 사용할 수 있음.

② 상태 전환 콜백 함수

Lifecycle 노드에는 각 상태 전환 시 자동 호출되는 콜백들이 존재한다.

- on_configure()

unconfigured -> inactive 로 전환될 때 실행
여기서 퍼블리셔/타이머 등 리소스를 생성
이유: 노드가 준비되었지만 아직 활성 상태는 아님

- on_activate()

inactive -> active 로 전환될 때 실행
생성된 퍼블리셔와 타이머를 활성화 -> 메시지 발행 시작
이유: active 상태에서만 외부에 데이터를 내보내도록 보장

- on_deactivate()

active -> inactive 로 전환될 때 실행
퍼블리셔를 비활성화하여 발행 중단
이유: 일시적으로 멈추고 싶을 때 (예: 로봇을 정지시키지만 노드는 유지)

- on_cleanup()
inactive -> unconfigured
퍼블리셔/타이머 파괴(리소스 해제)
이유: 노드를 깨끗한 초기 상태로 돌려 재구성 가능

- on_shutdown()
어떤 상태에서든 종료될 때 실행
자원 정리 및 노드 안전 종료

③ 실행 구조
LifecycleNode는 기본적으로 5가지 서비스 인터페이스를 자동 제공:
get_state, change_state, get_available_states, get_available_transitions, transition_event
외부에서 ros2 lifecycle 명령어나 서비스 호출로 상태를 전환할 수 있음.

2. ROS 2 Quality of Service (QoS)
배경
- ROS 2 : DDS 기반 -> 네트워크 환경/실시간 요구사항에 맞게 세부 QoS 설정 가능
- 무선 네트워크: 빠르지만 패킷 손실 가능 -> Best Effort 적합
- 센서 데이터 스트리밍: 최신 데이터만 필요 -> Depth 작게, Best Effort
- 로봇 제어 신호: 반드시 전달 -> Reliable
>즉, 응용 분야에 따라 신뢰성 vs 성능 최적화 가능.

2.1 QoS Policies
- History
Keep Last: 최근 N개 메시지만 저장
Keep All: 모든 메시지를 저장

- Depth
History = Keep Last일 때, N (큐 크기) 지정

- Reliability
Best Effort: 최대한 전송하지만, 유실 가능
Reliable: 반드시 전달 (재전송 시도)

- Durability
Volatile: 새로운 메시지만 전달
Transient Local: 이전 메시지도 late subscriber에게 전달 (ROS1 Latching 유사)

- Deadline
메시지 사이의 최대 허용 시간

- Lifespan
메시지가 "유효"하다고 보는 최대 시간 (넘으면 폐기)

- Liveliness : Publisher가 살아있음을 알리는 방식
Automatic: 메시지 보내면 자동 확인
Manual: 직접 "살아있음" 신호를 줘야 함

- Lease Duration
Liveliness: 유지되는 시간

2.2 QoS Profiles (미리 정의된 설정)
- Default
History: Keep Last (Depth 10)
Reliability: Reliable
Durability: Volatile

- Sensor Data
Best Effort + 작은 Depth
빠르고 최신 데이터만 필요할 때 사용 (카메라/라이다 등)

- Services
Reliable + Volatile (서비스 요청이 유실되면 안 됨)

- Parameters
서비스 기반 + 큰 Depth (요청이 쌓여도 안전)

2.3 QoS 호환성
Publisher(제공) vs Subscriber(요구) → "Request vs Offer" 모델
ex)
Pub: Reliable ↔ Sub: Best Effort O (호환됨)
Pub: Best Effort ↔ Sub: Reliable X (호환 안 됨 → 메시지 전달 안 됨)

> Durability, Deadline, Liveliness 등 모든 정책이 양쪽 호환되어야 통신 성립.

2.4 QoS 이벤트 (QoS 관련 알림)
- Publisher 이벤트
Deadline Missed (예상 주기 안 맞음)
Liveliness Lost (살아있음 신호 끊김)
Incompatible QoS 발견

- Subscriber 이벤트
Deadline Missed
Liveliness Changed (Pub 죽음/살아남)
Incompatible QoS 발견

3. 구현
- 코드 설명
1) 송신 노드
#include "rclcpp/rclcpp.hpp"
ROS 2 라이프사이클 노드와 일반 ROS 2 기능을 사용하기 위해 필요한 헤더

LifecyclePublisher::LifecyclePublisher()
: rclcpp_lifecycle::LifecycleNode("pub_node")
{}
LifecyclePublisher 클래스 생성자. "pub_node" 이름으로 라이프사이클 노드 초기화.

CallbackReturn LifecyclePublisher::on_configure(const rclcpp_lifecycle::State &)
Configure 상태로 전환될 때 호출되는 콜백. 리소스 할당 및 초기 설정 수행.

auto qos = rclcpp::QoS(rclcpp::KeepLast(10)).reliable();
- 큐(버퍼) 크기 10으로 메시지 유지, 최신 10개 메시지만 저장.
- 신뢰성 보장 설정, 메시지가 손실되지 않도록 DDS가 재전송 시도.

pub_ = this->create_publisher<std_msgs::msg::String>("lifecycle_chatter", qos);
"lifecycle_chatter" 토픽용 퍼블리셔 생성.

```
timer_ = this->create_wall_timer(
    std::chrono::seconds(1),
    std::bind(&LifecyclePublisher::timer_callback, this)
);
```
1초 간격 타이머 생성.

timer_->cancel();
처음에는 Deactivate 상태이므로 active가 아니므로 타이머는 중지

CallbackReturn LifecyclePublisher::on_activate(const rclcpp_lifecycle::State &)
Activate 상태로 전환될 때 호출. 퍼블리셔 활성화 후 타이머 시작.

pub_->on_activate();
퍼블리셔 활성화

timer_->reset();
타이머를 실제 데이터 발행 가능 상태로 재시작.

CallbackReturn LifecyclePublisher::on_deactivate(const
rclcpp_lifecycle::State &)
Deactivate 상태로 전환될 때 호출. 퍼블리셔 비활성화 후 타이머 중지.

pub_->on_deactivate();
timer_->cancel();
퍼블리셔 발행 중지, 타이머 취소.

CallbackReturn LifecyclePublisher::on_cleanup(const
rclcpp_lifecycle::State &)
Cleanup 상태로 전환될 때 호출. 할당된 리소스 모두 해제.

pub_.reset();
퍼블리셔 메모리 해제.

timer_.reset();
타이머 메모리 해제.

---

void LifecyclePublisher::timer_callback()
타이머 콜백 함수 정의, 주기적으로 실행됨.

if (pub_->is_activated()) {
퍼블리셔가 활성 상태인지 확인.

auto msg = std_msgs::msg::String();
메시지 객체 생성

msg.data = "Hello from lifecycle publisher";
데이터 문자열 설정.

RCLCPP_INFO(get_logger(), "Publishing: '%s'", msg.data.c_str());
콘솔에 발행 메시지 로그 출력.

pub_->publish(msg);
메시지를 실제로 토픽에 발행.

---

int main(int argc, char **argv)
프로그램 진입

rclcpp::init(argc, argv);
ROS 2 초기화, 노드 사용 준비.

auto node = std::make_shared<LifecyclePublisher>();
라이프사이클 노드 객체 생성.

rclcpp::spin(node->get_node_base_interface());
노드 실행, 콜백(타이머 등) 처리 시작.

rclcpp::shutdown();
ROS 2 정리 및 종료.

2) 수신 노드
#include "rclcpp/rclcpp.hpp“
ROS 2 헤더

#include "lifecycle_msgs/msg/state.hpp"
라이프사이클 상태 메시지 헤더

LifecycleSubscriber::LifecycleSubscriber()
: rclcpp_lifecycle::LifecycleNode("sub_node") {}
"sub_node" 이름으로 라이프사이클 Subscriber 노드 생성.

CallbackReturn LifecycleSubscriber::on_configure(const
rclcpp_lifecycle::State &)
Configure 상태 콜백: 구독자 설정 및 리소스 할당.

auto qos = rclcpp::QoS(rclcpp::KeepLast(10)).reliable();
수신자와 같은 QoS 설정

```
sub_ = this->create_subscription<std_msgs::msg::String>(
    "lifecycle_chatter",
    qos,
    std::bind(&LifecycleSubscriber::callback, this, std::placeholders::_1)
);
```
"lifecycle_chatter" 토픽 구독 생성, 메시지 수신 시 콜백 호출.

CallbackReturn LifecycleSubscriber::on_activate(const
rclcpp_lifecycle::State &)
Activate 상태 콜백: 활성 시 메시지 처리 준비

CallbackReturn LifecycleSubscriber::on_deactivate(const
rclcpp_lifecycle::State &)
Deactivate 상태 콜백: 비활성 시 메시지 수신 중지

CallbackReturn LifecycleSubscriber::on_cleanup(const
rclcpp_lifecycle::State &)
Cleanup 상태 콜백: 구독자 리소스 해제.

---

void LifecycleSubscriber::callback(std_msgs::msg::String::SharedPtr msg)
메시지 수신 콜백 함수.

if (this->get_current_state().id() ==
lifecycle_msgs::msg::State::PRIMARY_STATE_ACTIVE) {
노드가 Active 상태일 때

    RCLCPP_INFO(get_logger(), "Received: '%s'", msg->data.c_str());
}
메시지 출력.

---

int main(int argc, char **argv)
프로그램 진입

rclcpp::init(argc, argv);
ROS 2 초기화

auto node = std::make_shared<LifecycleSubscriber>();
노드 생성

rclcpp::spin(node->get_node_base_interface());
라이프사이클 노드 실행

rclcpp::shutdown();
종료