hw2 보고서

20기 인턴 송수민

1. 메인 윈도우 파일

```cpp
#include "hw2_pkg/main_window.hpp"
#include "ui_MainWindow.h"
#include <QPushButton>
#include <chrono>
#include <thread>

using namespace std::chrono_literals;

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    rclcpp::init(0, nullptr);
    node = std::make_shared<rclcpp::Node>("turtle_qt_controller");
    cmd_pub = node->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel", 10);
    pen_client = node->create_client<turtlesim::srv::SetPen>("turtle1/set_pen");

    connect(ui->ww, &QPushButton::clicked, this, &MainWindow::onW);
    connect(ui->aa, &QPushButton::clicked, this, &MainWindow::onA);
    connect(ui->ss, &QPushButton::clicked, this, &MainWindow::onS);
    connect(ui->dd, &QPushButton::clicked, this, &MainWindow::onD);
    connect(ui->btn1, &QPushButton::clicked, this, &MainWindow::onTriangle);
    connect(ui->btn2, &QPushButton::clicked, this, &MainWindow::onSquare);
    connect(ui->btn3, &QPushButton::clicked, this, &MainWindow::onCircle);

    ros_thread = std::thread([this]() {
        rclcpp::Rate rate(10);
        while (rclcpp::ok()) {
            rclcpp::spin_some(node);
            rate.sleep();
        }
    });
}

MainWindow::~MainWindow()
{
    rclcpp::shutdown();
    if (ros_thread.joinable()) ros_thread.join();
    delete ui;
}
```

먼저 using namespace std::chrono_literals; 구문으로 100ms, 1s 같은 시간 단위를 리터럴로 바로 사용할 수 있게 한다.
그다음 rclcpp::init(0, nullptr);를 통해 ROS2를 초기화한다.

"turtle_qt_controller"라는 이름의 ROS2 노드를 생성한다.

거북이에게 속도를 명령하는 퍼블리셔를 생성한다. 여기서 토픽 이름은 "turtle1/cmd_vel", 큐 사이즈는 10이다.

펜 색상과 두께를 설정하는 서비스 클라이언트를 만든다.

다음으로 connect(...) 구문들을 통해 Qt 버튼과 슬롯 함수를 연결한다.

그 아래에서 ros_thread = std::thread([this]() { ... });를 실행하여 별도의 쓰레드를 만든다. 이 쓰레드 안에서는 rclcpp::Rate rate(10);로 초당 10회 주기로 반복을 돌며, rclcpp::spin_some(node);을 호출하여 ROS2 콜백을 처리한다. 이후 rate.sleep();으로 주기를 맞춘다.

소멸자 MainWindow::~MainWindow()에서는 프로그램 종료 시 실행된다. 먼저 ROS2를 종료시키고, 만약 ROS 쓰레드가 실행 중이라면 join()을 호출해 안전하게 합류한다.
마지막으로 delete ui;로 UI 객체를 해제한다.

```cpp
void MainWindow::move(double linear, double angular)
{
    auto msg = geometry_msgs::msg::Twist();
    msg.linear.x = linear;
    msg.angular.z = angular;
    cmd_pub->publish(msg);
}

void MainWindow::setPen(int r, int g, int b, int width)
{
    if (!pen_client->wait_for_service(1s)) return;
    auto request = std::make_shared<turtlesim::srv::SetPen::Request>();
    request->r = r; request->g = g; request->b = b;
    request->width = width; request->off = 0;
    pen_client->async_send_request(request);
}

void MainWindow::drawShape(const std::string &shape)
{
    if (shape == "triangle") {
        setPen(255,0,0,3);
        for(int i=0;i<3;i++){
            move(2.0,0.0); std::this_thread::sleep_for(1s);
            move(0.0,2.094); std::this_thread::sleep_for(1s);
        }
    } else if (shape == "square") {
        setPen(0,255,0,5);
        for(int i=0;i<4;i++){
            move(2.0,0.0); std::this_thread::sleep_for(1s);
            move(0.0,1.57); std::this_thread::sleep_for(1s);
        }
    } else if (shape == "circle") {
        setPen(0,0,255,2);
        for(int i=0;i<7;i++){
            move(3.0,5); std::this_thread::sleep_for(0.2s);
        }
    }
```

move() 함수

geometry_msgs::msg::Twist 메시지를 생성한다.

msg.linear.x에 linear 값을 넣어 전진·후진 속도를 지정한다.

msg.angular.z에 angular 값을 넣어 회전 속도를 지정한다.

cmd_pub->publish(msg);를 호출하여 turtle1/cmd_vel 토픽으로 메시지를 발행한다.

setPen()함수

pen_client->wait_for_service(1s) : 서비스 서버(turtle1/set_pen)가 준비됐는지 최대 1초 기다린다. 준비 안되면 바로 리턴.

SetPen::Request 객체를 생성하여 RGB 색상(r,g,b), 두께(width), 펜 활성화(off=0) 값을 설정한다.
pen_client->async_send_request(request);로 비동기 요청을 보낸다.

drawShape()함수 에서는 각각의 도형에 대한 팬 색상 설정 후 각각의 도형을 그린다.

```
void MainWindow::onW() { move(2.0,0.0); }
void MainWindow::onS() { move(-2.0,0.0); }
void MainWindow::onA() { move(0.0,1.5); }
void MainWindow::onD() { move(0.0,-1.5); }
void MainWindow::onTriangle() { drawShape("triangle"); }
void MainWindow::onSquare() { drawShape("square"); }
void MainWindow::onCircle() { drawShape("circle"); }
```

버튼이 눌렸을 때 각각의 값들을 저장한다.