

## hw3 보고서

20기 인턴 송수민

### 1) 실행 결과



### 2) 코드 분석

#### 1. 헤더파일

```
#include "roscpp/roscpp.hpp"
#include "geometry_msgs/msg/twist.hpp"
#include "turtlesim/srv/set_pen.hpp"
#include <thread>
#include <termios.h>
#include <unistd.h>

class TurtleDrawer : public roscpp::Node
{
public:
    TurtleDrawer();
private:
    void keyboard_loop();
    void draw_shape(const std::string &shape);
    void set_pen(int r, int g, int b, int width);

    roscpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr cmd_pub_;
    roscpp::Client<turtlesim::srv::SetPen>::SharedPtr pen_client_;
    std::thread keyboard_thread_;
};
```

geometry\_msgs/msg/twist.hpp : 로봇의 선형 및 각속도 정보를 담는 Twist 메시지 타입을

사용하기 위한 헤더 파일

작동을 위해 필요한 함수들을 선언하였다.

## 2. main

```
#include "hw3_pkg/my_cpp_node.hpp"
#include <chrono>
#include <cmath>

using namespace std::chrono_literals;

TurtleDrawer::TurtleDrawer() : Node("turtle_drawer")
{
    cmd_pub_ = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel", 10);
    pen_client_ = this->create_client<turtlesim::srv::SetPen>("turtle1/set_pen");

    keyboard_thread_ = std::thread(&TurtleDrawer::keyboard_loop, this);
}

void TurtleDrawer::set_pen(int r, int g, int b, int width)
{
    if (!pen_client_->wait_for_service(1s)) {
        RCLCPP_WARN(this->get_logger(), "SetPen service not available");
        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);
}
```

### TurtleDrawer 생성자

Node("turtle\_drawer")로 부모 클래스를 초기화하며 노드 이름을 지정한다.

create\_publisher로 "turtle1/cmd\_vel" 토픽 퍼블리셔를 만들어서 거북이 속도 명령을 보낼 준비를 한다.

create\_client로 "turtle1/set\_pen" 서비스 클라이언트를 만들어 펜 색상과 굵기를 변경할 준비를 한다.

별도의 스레드(keyboard\_thread\_)를 만들어 keyboard\_loop() 함수를 실행, 키보드 입력을 계속 감시하도록 한다.

### set\_pen 함수

wait\_for\_service로 서비스가 준비될 때까지 최대 1초 대기하고, 준비되지 않으면 경고 로그를 출력 후 종료한다.

요청 객체(Request)를 만들어 r, g, b 색상과 펜 굵기를 설정하고 off = 0으로 펜을 활성화한다.

async\_send\_request로 서비스 요청을 비동기 전송하여 거북이 펜 설정을 적용한다.

```

void TurtleDrawer::keyboard_loop()
{
    struct termios oldt, newt;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_iflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);

    geometry_msgs::msg::Twist msg;

    while (rclcpp::ok()) {
        char c = getchar();
        msg.linear.x = 0.0;
        msg.angular.z = 0.0;

        if (c == 'w') msg.linear.x = 2.0;
        else if (c == 's') msg.linear.x = -2.0;
        else if (c == 'a') msg.angular.z = 1.5;
        else if (c == 'd') msg.angular.z = -1.5;
        else if (c == '1') draw_shape("triangle");
        else if (c == '2') draw_shape("square");
        else if (c == '3') draw_shape("circle");
        else if (c == 'q') break;

        cmd_pub_ -> publish(msg);
        rclcpp::sleep_for(100ms);
    }

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
}

```

터미널 입력을 비동기적으로 처리하기 위해 termios로 터미널을 ICANON과 ECHO 모드를 꺼서 키 입력이 바로 읽히도록 설정한다.

geometry\_msgs::msg::Twist 메시지를 생성하여 거북이 속도를 제어할 준비를 한다.

루프를 돌면서 rclcpp::ok()가 참일 동안 반복하고, getchar()로 키보드 입력을 받는다.

입력된 키에 따라 거북이 속도를 설정한다:

'w' -> 앞으로 이동

's' -> 뒤로 이동

'a' -> 반시계 방향 회전

'd' -> 시계 방향 회전

특정 키에 따라 미리 정의한 도형을 그리도록 호출한다:

'1' -> 삼각형

'2' -> 사각형

'3' -> 원

'q' 입력 시 루프 종료

매 루프마다 cmd\_pub\_를 통해 속도 메시지를 발행하고, 100ms 쉬면서 반복한다.

루프 종료 시 termios 설정을 원래 상태로 되돌려 터미널을 정상 모드로 복원한다.

```

void TurtleDrawer::draw_shape(const std::string &shape)
{
    geometry_msgs::msg::Twist msg;
    if (shape == "triangle") {
        set_pen(255, 0, 0, 3);
        for (int i = 0; i < 3; i++) {
            msg.linear.x = 2.0;
            msg.angular.z = 0.0;
            cmd_pub_->publish(msg);
            rclcpp::sleep_for(1s);

            msg.linear.x = 0.0;
            msg.angular.z = 2.094;
            cmd_pub_->publish(msg);
            rclcpp::sleep_for(1s);
        }
    }
    else if (shape == "square") {
        set_pen(0, 255, 0, 5);
        for (int i = 0; i < 4; i++) {
            msg.linear.x = 2.0;
            msg.angular.z = 0.0;
            cmd_pub_->publish(msg);
            rclcpp::sleep_for(1s);

            msg.linear.x = 0.0;
            msg.angular.z = 1.57;
            cmd_pub_->publish(msg);
            rclcpp::sleep_for(1s);
        }
    }
    else if (shape == "circle") {
        set_pen(0, 0, 255, 2);
        for (int i = 0; i < 3; i++) {
            msg.linear.x = 2.0;
            msg.angular.z = 3;
            cmd_pub_->publish(msg);
            rclcpp::sleep_for(1s);
        }
    }

    msg.linear.x = 0.0;
    cmd_pub_->publish(msg);
}

```

keyboard\_loop 함수로 호출되며 각 도형에 맞게 색상과 두께를 설정하고 사각형과 삼각형 일 시 앞으로 이동하고 회전하고를 반복해서 그리며 원일 시 앞으로 가는 것과 회전하는 것을 같이 해서 원을 그린다.

```

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

```

rclcpp::init으로 ROS 2를 초기화한다.

TurtleDrawer 노드를 생성한다.

rclcpp::spin을 호출해 노드를 계속 돌리면서 내부에서 키보드 입력에 따라 거북이 속도를 제어하고 도형 그리기 콜백을 처리하도록 한다.

사용자가 종료하면 rclcpp::shutdown으로 ROS 2를 정리하고 프로그램을 종료한다.