<RO:BIT ROS2 - 1일차 과제2 보고서>

작성자: 박기홍

직책: 19기 예비 단원

소속: RO:BIT 지능 팀

작성일: 2024년 11월 4일

목차

1. 요약

이 보고서는 turtlesim_draw 패키지 개발 과정과 결과에 대해 설명한다. turtlesim_draw 패키지는 ROS2에 내장된 turtlecim에 대하여 터미널에서 도형(삼각형, 원, 사각형) 그리기, 크기 선택(변으 길이, 원의 지름), pen의 색상, 굵기 선택 등의 모드 설정을 하는 패키지이다.

본 패키지는 RO:BIT의 ROS2 - 1일차 과제2에 대한 패키지로 제시된 요구 사항을 충족하는 데 중점을 두었다. 이 보고서에서는 프로젝트의 개요, 관련 이론, 개발 환경, 개발 과정, 결과 등을 다루고 있다.

2. 프젝젝트 개요

이 프로젝트의 목표는 ROS2 환경에서 터미널 명령을 통해 turtlesim 시뮬레이션에서 거북이가 세 가지의 도형을 그리고, 도형에 대한 크기 선택, pen의 색상과 굵기 등을 선택하는 패키지를 개발하는 것이다. 이를 위해 터미널에서 도형 선택, 크기 선택, pen 설정 기능을 구현하였다.

3. 관련 이론 및 개념

3.1 이론적 배경

이 프로젝트는 ROS2의 통신 구조와 노드, 서비스, 액션 등의 개념을 바탕으로 한다. ROS2(Robot Operating System 2)는 분산 시스템 구조를 가진 로봇 소프트웨어 프레임워크로, 로봇 간의 통신을 효율적으로 처리하기 위해 설계되었다. 특히, ROS2는 노드(Node)라는 기본 단위를 통해 각 기능을 모듈화하고, 이들 간의 메시지 전달을 통해 로봇의 제어와 데이터를 교환한다. 노드 간의 통신은 퍼블리셔-서브스크라이버 모델(Publisher-Subscriber Model)과 서비스(Service), 액션(Action) 등을 사용하여 이루어진다.

퍼블리셔-서브스크라이버 모델은 비동기식 통신을 위해 사용되며, 특정 노드가 데이터를 퍼블리시하면 해당 데이터를 구독하는 노드들이 이를 수신하는 방식으로 동작한다. 반면 서비스는 동기식 통신을 위한 것이며, 요청(Request)과 응답(Response)의 형태로 이루어진다. 액션은 장시간이 소요될 수 있는 작업에 사용되며, 피드백(Feedback)을 통해 작업의 진행 상황을 모니터링할 수 있는 기능을 제공한다.

3.2 관련 개념

- **노드 (Node)**: ROS2의 기본 실행 단위로, 각각의 노드는 독립적인 기능을 수행하며, 서로 통신을 통해 데이터를 주고받는다. 이 프로젝트에서는 turtlesim을 제어하기 위해 여러 노드가 사용되었다.
- **토픽 (Topic)**: 노드 간의 비동기적 데이터 송수신을 위해 사용되는 채널로, 퍼블리셔와 서브스크라이버의 관계를 통해 데이터가 전달된다. 예를 들어, 거북이의 위치 정보나 속도 등의 데이터를 퍼블리시하고 이를 구독하는 방식이다.
- **서비스 (Service)**: 동기식 통신을 위한 메커니즘으로, 클라이언트가 서버에 요청을 보내고 응답을 받는 형태로 동작한다. 본 프로젝트에서는 배경색 설정 등의 작업에 서비스가 사용되었다.

- **ROS2 메시지 타입**: ROS2에서 노드 간 데이터를 주고받기 위해 다양한 메시지 타입을 사용한다. 이 프로젝트에서는 geometry_msgs라는 메시지 타입을 사용하여 거북이의 위치와 펜 설정 등을 처리하였다.

4. 개발 환경

이 프로젝트는 Ubuntu 22.04에서 ROS2 Humble을 기반으로 수행되었다. 하드웨어는 Intel i9 프로세서와 32GB RAM을 갖춘 Lenovo Legion 노트북을 사용하하였다. 또한, ROS2 패키지 개발을 위해 Visual Studio Code와 Git을 사용하여 코드 작성과 버전 관리를 하였다.

소프트웨어 환경:

운영체제: Ubuntu 22.04 LTS

ROS2 버전: Humble Hawksbill

IDE 및 VCD: Visual Studio Code, Git

컴파일러: GCC (GNU Compiler Collection) 11.4.0

CMake: 프로젝트 빌드 및 패키지 관리를 위해 CMake를 사용하였으며, ROS2 패키지의 빌드 시스템인 ament_cmake를 사용하여 프로젝트를 구성하였다.

필수 패키지:

rclcpp: ROS2 C++ 클라이언트 라이브러리

turtlesim: ROS2 turtlesim 시뮬레이션 패키지

geometry_msgs:거북이의 위치 및 방향을 정의하기 위해 사용되는 메시지 패키지

std_srvs:기본 서비스 메시지 패키지로, 배경색 변경 등의 다양한 서비스 구현을 위해 필요

5. 패키지 분석

이 프로젝트에서 개발한 turtlesim_draw 패키지는 ROS2 기본 패키지인 turtlesim을 터미널 명령어로 제어하기 위해 설계되었다. 이 패키지는 도형 그리기, 크기 선택, 펜 설정 등의 기능을 제공하며, 이를 위해 ROS2의 노드와 서비스 구조를 활용하였다.

turtlesim_draw 패키지는 다음과 같은 주요 노드로 구성되어 있다:

- **turtlesim_draw_shape**: 터미널을 통해 거북이가 도형을 그리도록 하는 명령어를 제공한다.
- **turtlesim_style_length**: 도형의 크기를 변경하기 위한 기능을 제공한다.
- **turtlesim_style_style_pen**: 거북이의 펜 색상과 굵기를 설정한다.
- **turtlesim_draw**: 메인 프로세스 노드로, 모든 노드를 관리하며 사용자로부터 입력을 받아 각 기능에 알맞는 노드를 활성화한다.

패키지의 각 기능은 ROS2의 노드로 구현되어 있으며, 노드 간의 통신을 통해 상호작용하도록 설계되었다. 이러한 모듈화를 통해 기능별로 코드가 분리되어 유지보수가 용이하며, 각 기능의 독립적인 테스트가 가능하다는 것을 알게 되었다.

6. 개발 과정

6.1. 패키지 설계 및 구조

패키지와 노드 구조를 설계 할 때 노드를 독립적으로 구현하는 것에 초점을 맞추어 개발을 진행하였다. 과제에서 제시된 주요 기능은 크게 세 가지로 나뉜다. 각 기능은 개별 노드로 구현되었으며, 이를 관리하는 메인 노드에서 전체를 통합하여 관리한다.

과제에서 제시된 주요 세 가지 기능은 다음과 같다:

1. 도형 그리기: 삼각형, 원, 사각형 중 사용자의 입력에 따라 도형을 그린다.
2. 크기 선택: 도형의 크기를 사용자로부터 입력 받아 설정한다.
3. 펜 설정: 거북이의 경로 색, 굵기 등을 변경한다.

각 기능은 분산된 구조로 제어하기 위해 독립적인 노드로 설계하였다.

- **turtlesim_draw_shape**: 거북이가 도형을 그리는 기능을 수행하도록 설계하였다.
- **turtlesim_style_length**: 도형의 크기 설정 기능을 수행하도록 설계하였다.
- **turtlesim_style_style_pen**: pen 설정 기능을 수행하도록 설계하였다.

그리고 이러한 주요 기능을 통합하고 관리하는 turtlesim_draw 노드를 설계하여, 사용자의 입력에 따라 원하는 모드를 활성화하도록 설계하였다.

6.2. 코드 및 주요 기능 설명

6.2.1. 도형 그리기(turtlesim_draw_shape 노드)

turtlesim_draw_shape 노드는 삼각형, 원, 사각형과 같은 세 가지의 도형을 그리기 위한 기능을 담당한다. ROS2의 퍼블리셔를 사용해서 거북이의 위치와 회전 각도를 제어하고, 삼각형, 원, 사각형을 그릴 수 있다. 이 노드는 geometry_msgs::msg::Twist 메시지를 퍼블리시해서 거북이가 주어진 도형을 따라 이동하도록 설정한다. 또한, 펜의 색상과 두께를 설정할 수 있도록 ROS2 서비스 클라이언트를 사용해서 SetPen 서비스를 호출했다.

6.2.1.1. setPenStyle() 메서드

```
// Pen 설정 메서드 (1일차 과제1의 모드와 중복 되는 부분이므로 적용함)
void TurtlesimDrawShape::setPenStyle() {
    auto request = std::make_shared<turtlesim::srv::SetPen::Request>();

    int r, g, b;
    int width;

    std::cout << "Pen Color를 0부터-255 사이로 입력해라 (r g b):";
    std::cin >> r >> g >> b;

    // 입력된 color 값이 정해진 범위 내에 있는지 검사하기
    request->r = std::max(0, std::min(r, 255));
    request->g = std::max(0, std::min(g, 255));
    request->b = std::max(0, std::min(b, 255));

    std::cout << "Pen 두께를 입력해라 (0-10): ";
    std::cin >> width;
    // Pen의 width 제한
    request->width = std::max(0, std::min(width, 10));
    request->off = 0;

    // 비동기 서비스 요청 보내기
    auto future = set_pen_client_->async_send_request(request);
}
```

SetPen 서비스 요청 객체를 생성한다. 이 객체는 펜의 색상과 두께, on/off 상태를 포함하는 데이터 구조를 가지고 있다. 이후, 용자로부터 RGB 값을 입력 받게 했으며 0부터 ~ 255까지의 범위 내로 값에 제한을 걸었다. 입력 받은 RGB 값은 request 객체의 r, g, b 필드에 할당 된다.

사용자로부터 두께를 입력 받고, 0부터 ~ 10까지의 범위 내로 값에 제한을 걸었다. 이후 설정된 요청 객체를 set_pen_client_를 통해 비동기적으로 /turtle1/set_pen 서비스에 전송하였다.

*비동기 전송은 요청 후 응답을 기다리지 않고 다음 작업을 이어서 처리한다. 동기 전송은 요청의 응답을 기다린다.

6.2.1.2. drawTriangle() 메서드

```
// 삼각형 그리는 신호 받았을 때 거북이가 삼각형을 그림
void TurtlesimDrawShape::drawTriangle(double sideLength) {
    auto twist_msg = geometry_msgs::msg::Twist();

    // 삼각형 그리기: 총 세 번 이동 및 회전
    for (int i = 0; i < 3; ++i) {
        // 직선 이동
        twist_msg.linear.x = sideLength;  // 앞으로 이동할 길이 설정
(사용자로부터 입력 받은 값, 없다면 1이 됨)
        twist_msg.angular.z = 0.0;        // 회전 없이 직진(이동중에 고개
틀면 그건 삼각형이 아님 모서리에서 고개를 틀어야 함)
        velocity_publisher->publish(twist_msg);
        rclcpp::sleep_for(std::chrono::milliseconds(1000)); // 딜레이

        // 회전 (120도 회전)
        twist_msg.linear.x = 0.0;          // 이동 멈춤 -> 꼭짓점에서 각도
틀어야 하니까 멈춤 처리
        twist_msg.angular.z = M_PI / 1.5;  // 반시계 방향으로 120도 회전
-> M_PI가 math.h에서 지원하는 3.14 ~~~ 상수임. 이거를 1.5로 나누면 120이
된다고 함
        velocity_publisher->publish(twist_msg);
        rclcpp::sleep_for(std::chrono::milliseconds(1000)); // 딜레이
    }

    // 멈춤
    twist_msg.linear.x = 0.0; // 다 그렸으면 멈추도록 처리함
    twist_msg.angular.z = 0.0;
    velocity_publisher->publish(twist_msg);
}
```

거북이의 속도와 회전 각도를 제어하는 Twist 메시지 객체를 생성했다. 거북이를 제어하기 위해서는 linear.x와 angular.z를 설정하여 각각 직선 이동 속도와 회전 속도를 제어할 수 있다. 삼각형은 세 개의 변과 세 개의 꼭짓점으로 이루어져 있다. 그렇기에 꼭짓점 부분에서 회전 처리를 하기 위해 for() 문을 사용하여 세 번 반복하게 로직을 설계 했다.

linear.x에 sideLength를 대입하여 거북이가 앞으로 이동할 거리를 정했다. angular.z를 0.0으로 설정하여 회전 없이 거북이가 바라보는 방향으로 직진하도록 설정하였다. 이후, 설정한 메시지를

퍼블리시하여 거북이에게 이동 명령을 전달하였다. 마지막으로 딜레이를 주어 거북이가 바로 이후에 있는 작업을 처리하는 것이 아닌, 대기하였다가 처리할 수 있도록 설계하였다.
거북이의 직진이 완료 되면, 이동을 멈추고, angular.z를 설정하여 반시계 방향으로 120도 회전 처리를 하였다. 공식은 구글링과 GPT를 사용하여 120도를 처리하기 위해서는 **M_PI / 1.5**의 값을 할당해야 하는 것을 알게 되었다. 이후 마찬가지로 수정된 메시지를 퍼블리시하여 거북이에게 전달했다.

6.2.2. 크기 선택(turtlesim_style_length 노드)

turtlesim_style_length 노드는 도형의 크기를 설정하는 기능을 담당한다. 삼각형과 사각형에 대해서는 변의 길이를, 원에 대해서는 지름을 사용자로부터 입력 받아 저장한다.

6.2.2.1. setPenStyleSideLength() 메서드

```
// 변의 지름
void TurtlesimStyleLength::setPenStyleSideLength() {
    std::cout << "변의 길이를 0부터-10 사이로 입력해라:";
    std::cin >> side_length_;
}
```

사용자로부터 삼각형, 사각형에 대한 변의 길이를 입력 받아 저장하도록 설계하였다.

6.2.2.2. 배경색 설정 메서드

```
// 원의 지름
void TurtlesimStyleLength::setPenStyleDiameterLength() {
    std::cout << "원의 지름을 0부터-10 사이로 입력해라:";
    std::cin >> diameter_length_;
}
```

사용자로부터 원의 지름 값을 입력 받아 저장하도록 설계하였다.

6.2.2.3. getSideLength() 메서드와 getDiameterLength() 메서드

```
// Getter 함수 구현 (변의 길이 반환 메서드)
double TurtlesimStyleLength::getSideLength() const {
    return side_length_;
}

double TurtlesimStyleLength::getDiameterLength() const {
    return diameter_length_;
}
```

사용자의 입력으로 설정된 변의 길이와 지름 값을 반환한다. const 키워드 없이 사용해도 되겠지만, 리팩터링을 통해 getter() 기능을 구현할 때 const를 사용하면 의도치 않은 값 변경을 방지할 수 있다는 것을 알게 되었다.

6.2.3. 펜 설정(turtlesim_style_pen 노드)

6.2.3.1. setPenStyle() 메서드

```
// Pen 설정 메서드 (1일차 과제1의 모드와 중복 되는 부분이므로 적용함)
void TurtlesimStyle::setPenStyle() {
    auto request = std::make_shared<turtlesim::srv::SetPen::Request>();

    int r, g, b;
    int width;

    std::cout << "Pen Color를 0부터-255 사이로 입력해라 (r g b):";
    std::cin >> r >> g >> b;

    // 입력된 color 값이 정해진 범위 내에 있는지 검사하기
    request->r = std::max(0, std::min(r, 255));
    request->g = std::max(0, std::min(g, 255));
    request->b = std::max(0, std::min(b, 255));

    std::cout << "Pen 두께를 입력해라 (0-10): ";
    std::cin >> width;
    // Pen의 width 제한
    request->width = std::max(0, std::min(width, 10));
    request->off = 0;

    // 비동기 서비스 요청 보내기
    auto future = set_pen_client_->async_send_request(request);
}
```

turtlesim_style_pen 노드는 펜의 색상과 두께를 설정하는 기능을 담당한다. 이 기능은 이전에 공부하고 구현한 경험이 있기에 <RO:BIT ROS2 - 1일차 과제1>에서 구현한 기능을 적용하여 구현하였다.

6.2.4. turtlesim_draw 노드

```
TurtleSimDraw::TurtleSimDraw() : Node("turtlesim_draw")
{
    RCLCPP_INFO(this->get_logger(), "TurtleSimDraw Node 초기화");

    // TurtleSim 실행
    std::system("ros2 run turtlesim turtlesim_node &");

    sideLength = 1.0;
    diameterLength = 1.0;

    run();
}

// 프로세스 시작
void TurtleSimDraw::run()
{
    // 무한으로 입력 받음(escape: Ctrl + C)
    // rclcpp::ok() -> 노드가 실행가능하고 정상적일 때: true, 그렇지 않으면 false를 return함
    while (rclcpp::ok())
    {
        displayMenu(); // 메뉴 띄우기
        int mode = getUserInput(); // 입력 받기
        processInput(mode); // 입력에 따른 프로세스 진행하기
    }
}

void TurtleSimDraw::displayMenu()
{
    std::cout << "===== TurtleSim CLI Menu =====\n";
    std::cout << "1. 도형 선택(삼각형, 원, 사각형)\n";
    std::cout << "2. 크기 설정(변 길이, 지름 등)\n";
    std::cout << "3. Pen 설정(색상, 두께)\n";
    std::cout << "모드 선택하기 (1-3): ";
}
```

```
int TurtleSimDraw::getUserInput()
{
    int input;
    std::cin >> input;
    return input;
}

void TurtleSimDraw::processInput(int mode)
{
    switch (mode)
    {
    case 1:
        {
            std::cout << "도형 선택 모드를 선택했음." << std::endl;
            std::cout << "어떤 도형? <1. 삼각형 | 2. 원 | 3. 사각형>" <<
std::endl;
            int input;
            std::cin >> input;

            auto turtle_draw_shape =
std::make_shared<TurtlesimDrawShape>();

            if(input == 1){
                turtle_draw_shape->drawTriangle(side_length_);
            }else if(input == 2){
                turtle_draw_shape->drawCircle(diameter_length_);
            }else if(input == 3){
                turtle_draw_shape->drawQuadrilateral(side_length_);
            }
        }
        break;
    case 2:
        {
            std::cout << "크기 설정 모드를 선택했음.\n";

            std::cout << "어떤 크기 설정? <1. 변의 길이 | 2. 원의 지름 >"
<< std::endl;
            int inputOption;
            std::cin >> inputOption;

            auto turtle_style_length =
std::make_shared<TurtlesimStyleLength>();

            if(inputOption == 1){
                turtle_style_length->setPenStyleSideLength();
                side_length_ = turtle_style_length->getSideLength();
```

```
            } else if(inputOption == 2) {
                turtle_style_length->setPenStyleDiameterLength();
                diameter_length_ =
turtle_style_length->getDiameterLength();
            }

        }
        break;
    case 3:
        {
            std::cout << "Pen 설정 모드를 선택했음.\n";
            auto turtle_style = std::make_shared<TurtlesimStyle>();
            turtle_style->setPenStyle();  // setPenStyle 호출
        }

        break;
    default:
        std::cout << "없는 모드다.\n";
        break;
    }
}
```

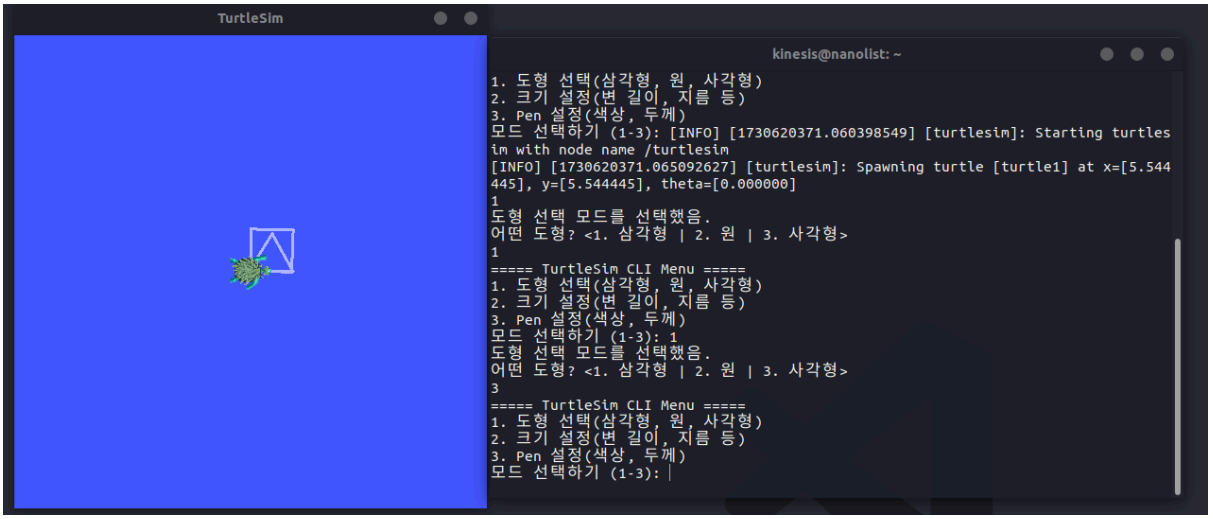
turtlesim_draw 노드는 메인 노드로, 프로그램의 시작점이자 사용자의 입력을 받아 각 기능을 호출한다.

displayMenu()와 getUserInput()을 통해 메뉴를 출력하고 입력 받은 후, processInput() 메서드를 통해 각 기능(도형 그리기, 크기 선택, 펜 설정)을 실행했다.

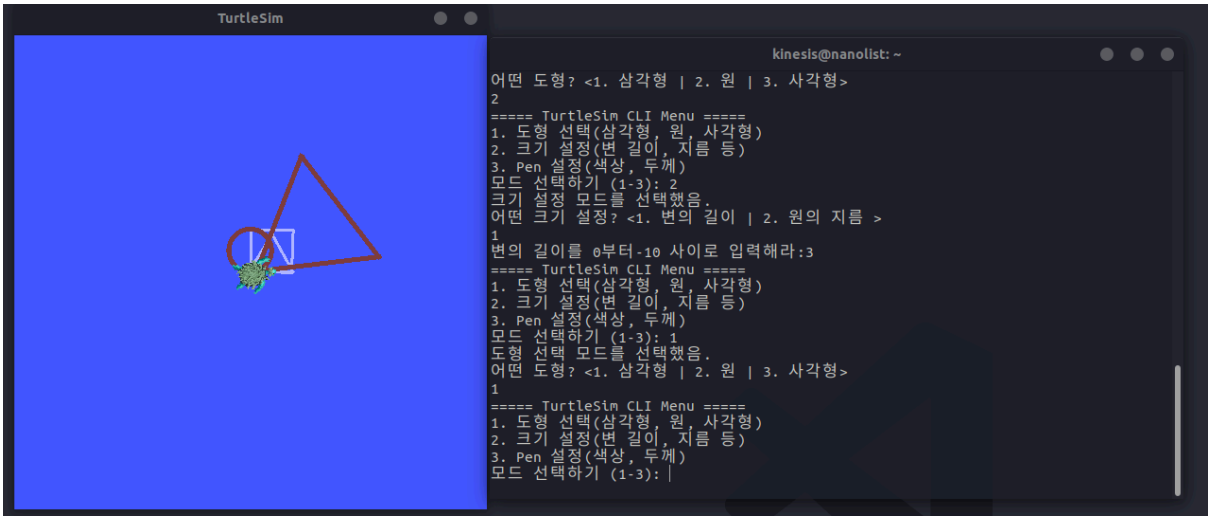
6.3실패 사례와 시도한 방법

원을 그리는 부분에서 거북이가 예상한 것 보다 더 회전하거나 도중에 멈추는 현상이 발생 되었다. 원을 그리는 공식은 찾아가고, GPT를 사용하면서 알게 되었지만, 작은 선을 나누어서 각도를 미세하게 조절하면서 그리면 되는 것을 알게 되었다. 원이 그려지다가 도중에 멈추는 현상은 로직 내의 segments 변수의 value를 수정하고 디버깅해가며 의도한 대까지 원이 그려지도록 수정하였다.

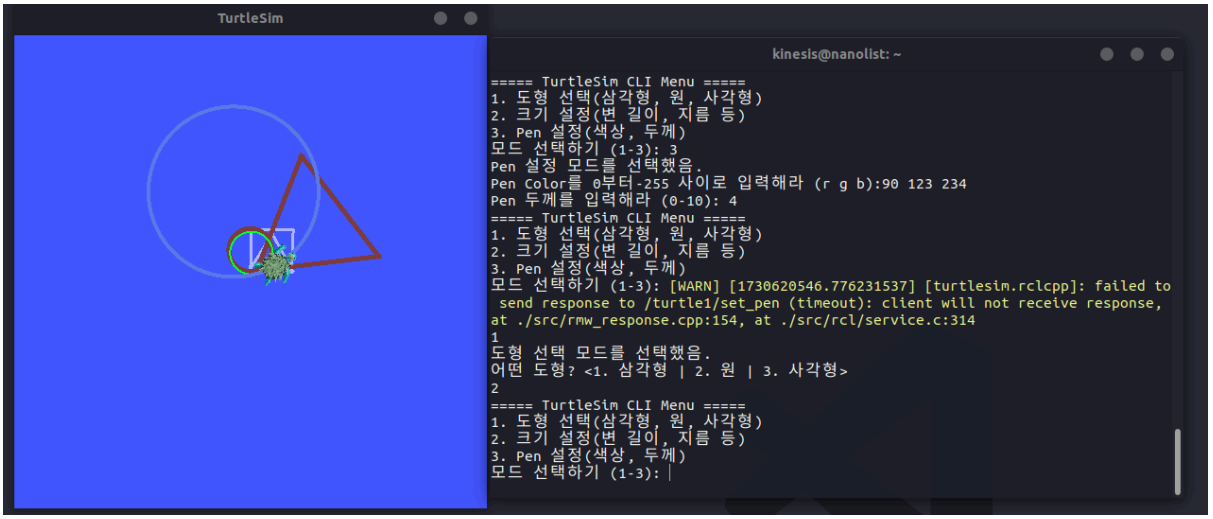
7. 결과



<turtlesim_draw 실행 결과 1>



<turtlesim_draw 실행 결과 2>



<turtlesim_draw 실행 결과 3>

8. 결론 및 향후 발전 방향

이번 프로젝트는 ROS2 환경에서 터미널 명령을 통해 turtlesim 시뮬레이션을 직관적으로 제어할 수 있는 패키지 개발이 과제로 진행 되었다. 주요 기능은 개발한 패키지 내에서 사용자로부터 입력을 받아 거북이가 도형을 그리도록 명령, 도형의 크기 설정, 펜 스타일 설정이다. 펜 스타일 설정 같은 경우에는 이전 과제에서 공부하고 구현한 로직이 있기에 이를 적용하여 구현하였다.

9. 참고 문헌

- 표윤석. (2020.08.18.). "000 로봇 운영체제 ROS 강좌 목차". https://cafe.naver.com/openrt/24070
- 표윤석. (2024.09.05.). "로봇 운영체제 ROS 2 특강 발표 자료", https://docs.google.com/presentation/d/1QvUHj8iZCWIGiWRCgHFG-DLAmoUK6khO9xBXCzjPTTs/edit

10. 부록

10.1 레파지토리 링크

ROBIT_ROS2_HW: https://github.com/kinesis19/ROBIT_ROS2_HW

10.2 과제 Wiki

ROS2_HW_DAY_1: https://github.com/kinesis19/ROBIT_ROS2_HW/wiki/ROS2_HW_DAY1

10.3 패키지 구조

패키지의 구조는 다음과 같다:

```
turtlesim_draw
├── CMakeLists.txt
├── include
│   ├── turtlesim_draw
│   ├── turtlesim_draw.hpp
│   ├── turtlesim_draw_shape.hpp
│   ├── turtlesim_style_length.hpp
│   └── turtlesim_style_pen.hpp
├── package.xml
└── src
    ├── main.cpp
    ├── turtlesim_draw.cpp
    ├── turtlesim_draw_shape.cpp
    ├── turtlesim_style_length.cpp
    └── turtlesim_style_pen.cpp
```