

<RO:BIT ROS2 - 1일차 과제1 보고서>

작성자: 박기홍

직책: 19기 예비 단원

소속: RO:BIT 지능 팀

작성일: 2024년 11월 4일

목차

1. 요약
2. 프로젝트 개요
3. 관련 이론 및 개념
 - 3.1 이론적 배경
 - 3.2 관련 개념
4. 개발 환경
5. 패키지 분석
6. 개발 과정
 - 6.1 패키지 설계 및 구조
 - 6.2 코드 및 주요 기능 설명
 - 6.3 실패 사례와 시도한 방법
7. 결과
8. 결론 및 향후 발전 방향
9. 참고 문헌
10. 부록
 - 10.1 레파지토리 링크
 - 10.2 과제 Wiki
 - 10.3 패키지 구조

1. 요약

이 보고서는 turtlesim_cli 패키지 개발 과정과 결과에 대해 설명한다. turtlesim_cli 패키지는 ROS2에 내장된 turtlesim에 대하여 터미널에서 거북이 조종, 배경색 설정, 거북이 모양 설정, pen 설정 등의 모드 설정을 하는 패키지이다.

본 패키지는 RO:BIT의 ROS2 - 1일차 과제1에 대한 패키지로 제시된 요구 사항을 충족하는 데 중점을 두었다. 이 보고서에서는 프로젝트의 개요, 관련 이론, 개발 환경, 개발 과정, 결과 등을 다루고 있다.

2. 프로젝트 개요

이 프로젝트의 목표는 ROS2 환경에서 터미널 명령을 통해 turtlesim 시뮬레이션을 보다 효율적으로 제어할 수 있는 패키지를 개발하는 것이다. 이를 위해 터미널에서 거북이의 이동, 배경색 변경, 거북이 모양 설정, 그리고 펜의 속성 설정 등을 쉽게 조작할 수 있도록 하는 다양한 기능을 구현하였다.

본 프로젝트는 ROS2 학습 과정의 일환으로, ROS2의 메시지 통신 구조와 패키지 개발에 대한 이해를 심화시키기 위한 목적으로 수행되었다. 특히, ROS2에서의 노드 간 통신과 서비스, 액션 클라이언트에 대한 이해를 실제 패키지 구현을 통해 적용하고자 하였다. 이를 통해 ROS2의 핵심 개념들을 실습하며 학습하는 동시에, RO:BIT 예비 단원 과제로 제시된 요구사항을 충족하는 것을 목표로 하였다.

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.

- **액션 (Action):** 비동기적인 긴 작업을 위해 사용되며, 상태 업데이트와 피드백을 제공할 수 있다. 액션은 이 프로젝트에서는 사용되지 않았으나, 개념적으로 이해하고 학습하기 위해 이론적 배경에 포함되었다.
- **ROS2 메시지 타입:** ROS2에서 노드 간 데이터를 주고받기 위해 다양한 메시지 타입을 사용한다. 이 프로젝트에서는 `geometry_msgs`라는 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 사용하였다. 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_cli` 패키지는 ROS2 기본 패키지인 `turtlesim`을 터미널 명령어로 제어하기 위해 설계되었다. 이 패키지는 거북이의 이동, 배경색 변경, 펜 설정 등의 기능을 제공하며, 이를 위해 ROS2의 노드와 서비스 구조를 활용하였다.

`turtlesim_cli` 패키지는 다음과 같은 주요 노드로 구성되어 있다:

- **`turtlesim_control`:** 터미널을 통해 거북이를 이동시키는 명령어를 제공한다. 이 노드는 ROS2의 퍼블리셔-서브스크라이버 구조를 사용하여 거북이의 위치와 방향을 제어한다.
- **`turtlesim_style_bg`:** 배경색을 변경하기 위한 기능을 제공한다. ROS2의 서비스(`/clear`)를 호출하여 배경을 변경하고 화면을 갱신한다.
- **`turtlesim_style_pen`:** 거북이의 펜 색상과 굵기를 설정한다. 이 노드는 사용자 입력을 통해 펜의 속성을 변경하고 이를 반영한다.
- **`turtlesim_style_shape`:** 거북이의 모양을 설정하는 기능을 제공하며, 사용자로부터 입력을 받아 다양한 모양으로 변경한다.

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 네 가지로 나뉜다. 각 기능은 개별 노드로 구현되었으며, 이를 관리하는 메인 노드에서 전체를 통합하여 관리한다.

과제에서 제시된 주요 네 가지 기능은 다음과 같다:

1. 조종 모드: WASD키로 입력을 받아 거북이를 조종한다(turtle_teleop_key 참고.)
2. 배경색 설정 모드: 배경색을 입력 받아 변경한다(RGB, 색상코드, 텍스트 등 자유.)
3. 거북이 모양 설정: 모드 실행 시 선택 가능한 거북이 모양 리스트 출력, 몬 선택시 거북이의 모양을 변경한다.
4. pen 설정 모드: 거북이의 경로 색, 굵기 등을 변경한다.

각 기능은 분산된 구조로 제어하기 위해 독립적인 노드로 설계하였다.

- **turtlesim_control**: 거북이 조종 기능을 수행하도록 설계하였다.
- **turtlesim_style_bg**: 배경색 설정 기능을 수행하도록 설계하였다.
- **turtlesim_style_shape**: 거북이의 모양 설정 기능을 수행하도록 설계하였다.
- **turtlesim_style_pen**: pen 설정 기능을 수행하도록 설계하였다.

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

6.2. 코드 및 주요 기능 설명

6.2.1. 조종 모드(turtlesim_control 노드)

TurtleSimControl 클래스는 거북이의 이동을 제어하는 노드이다. 이 클래스는 사용자가 입력하는 키에 따라 거북이를 전진, 후진, 좌회전, 우회전하게 하며, ROS2 퍼블리셔를 사용하여 cmd_vel

토픽으로 명령을 발신한다. turtlesim_control 노드에는 생성자, 컨트롤 루프, 그리고 키 입력 처리 메서드로 구성되어 있다.

6.2.1.1. TurtleSimControl 생성자

```
TurtleSimControl::TurtleSimControl() : Node("turtle_sim_control")
{
    // Publisher 생성 - turtle1/cmd_vel Topic으로 Twist 메시지를
    퍼블리싱함
    velocity_publisher_ =
    this->create_publisher<geometry_msgs::msg::Twist>("/turtle1/cmd_vel",
    10);
    std::cout << "TurtleSimControl Node 초기화" << std::endl;
}
```

이 생성자는 노드의 이름을 "turtle_sim_control"로 설정하였으며, 거북이의 속도 명령을 퍼블리시하기 위한 퍼블리셔(velocity_publisher_)를 생성한다.

퍼블리셔는 /turtle1/cmd_vel 토픽으로 geometry_msgs::msg::Twist 메시지를 퍼블리시하며, 퍼블리셔의 큐 사이즈는 10으로 설정되었다.

velocity_publisher_는 거북이의 속도 명령을 전달한다. 생성자가 호출될 때 퍼블리셔를 생성하고 노드가 초기화 되었음을 출력을 통해 알린다.

6.2.1.2. 조종 루프

이 메서드는 사용자로부터 키 입력을 받아 거북이를 제어하는 루프이다. q 또는 Q키를 입력 시 루프를 종료한다. rclcpp::ok()를 통해 노드가 종료될 때까지 반복한다.

```
void TurtleSimControl::controlLoop()
{
    std::cout << "<조종 모드 시작> - WASD 키를 사용하여 거북이를 제어하자
    (종료: Q 입력)" << std::endl;
    char key;
    // rclcpp::ok() -> Node가 정상적으로 실행되고 있으면 true를 반환함
    while (rclcpp::ok())
```

```

{
    std::cin >> key;
    // 입력된 키가 'q' 또는 'Q'라면 조종 모드 종료하기
    if (key == 'q' || key == 'Q')
    {
        RCLCPP_INFO(this->get_logger(), "조종 모드 종료");
        std::cout << "조종 모드 종료" << std::endl;
        break;
    }
    processKeyInput(key);
}
}

```

rcldcpp::ok()는 노드가 정상적으로 실행 중인지 확인한다.

종료키(q 또는 Q키)가 눌리지 않았다면 processKeyInput(key) 메서드를 호출하며 인자로 사용자의 키 값(key)을 넘겨주어 거북이의 이동을 처리한다.

6.2.1.3. 거북이 조종 메서드

switch~case() 문을 사용하여 사용자의 입력한 키에 따라 거북이의 이동을 결정한다. 키 입력에 따라 **geometry_msgs::msg::Twist** 메시지를 생성하고, 이를 퍼블리셔로 퍼블리시한다. WASD 키가 아닌 키는 예외처리를 진행해 주었다.

```

void TurtleSimControl::processKeyInput(char key)
{
    auto twist_msg = geometry_msgs::msg::Twist();

    // 대소문자 관계 없이 이동키 입력 받아 처리함
    switch (key)
    {
    case 'w':
    case 'W':
        twist_msg.linear.x = 1.0;
        twist_msg.angular.z = 0.0;
        break;
    case 's':
    case 'S':
        twist_msg.linear.x = -1.0;
        twist_msg.angular.z = 0.0;

```

```

        break;
    case 'a':
    case 'A':
        twist_msg.linear.x = 0.0;
        twist_msg.angular.z = 1.0;
        break;
    case 'd':
    case 'D':
        twist_msg.linear.x = 0.0;
        twist_msg.angular.z = -1.0;
        break;
    default:
        std::cout << "알 수 없는 키 입력: " << key << std::endl;
        return;
    }

    // 메시지 퍼블리시
    velocity_publisher_->publish(twist_msg);
}

```

twist_msgs는 거북이의 속도를 나타내는 메시지로, **linear.x**와 **angular.z**를 설정하여 이동을 조절한다.

velocity_publisher_를 통해 메시지를 퍼블리시하여 **/turtle1/cmd_vel** 토픽으로 전송한다. 이를 통해 거북이가 지정된 방향으로 움직인다.

6.2.2. 배경 설정 모드(turtlesim_style_bgl 노드)

TurtlesimStyleBg 클래스는 배경색 설정 모드를 위한 노드이다. 이 노드는 사용자가 입력한 RGB 값을 바탕으로 **turtlesim**의 배경색을 설정하는 역할을 한다. 이 노드는 생성자와 배경색 설정 메서드로 구성되어 있다.

6.2.2.1. TurtleSimControl 생성자

노드를 초기화하고, 배경색을 변경할 때 화면을 갱신하기 위한서비스 클라이언트(**clear_client_**)를 생성한다.

```
TurtlesimStyleBg::TurtlesimStyleBg() : Node("turtlesim_style_bg")
{
    // 배경 초기화를 위해 clear_client_를 생성
    clear_client_ =
    this->create_client<std_srvs::srv::Empty>("/clear");
}
```

노드의 이름을 "turtlesim_style_bg"로 설정하였다.

clear_client_는 /clear 서비스를 호출하여 배경색을 변경한 후, 화면을 갱신하기 위해 사용된다. 이 서비스는 변경된 배경색이 즉시 반영되도록 해준다.

6.2.2.2. 배경색 설정 메서드

사용자로부터 RGB 값을 입력 받아 turtlesim의 배경색을 설정한다. 설정 후, /clear 서비스를 호출하여 변경된 배경색을 즉시 화면에 반영한다.

```
void TurtlesimStyleBg::setBackgroundColor() {
    // 사용자로부터 RGB 값 입력 받기
    int r, g, b;

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

    // 색상 값 유효성 확인 후 rclcpp 파라미터 설정
    auto parameter_client =
    std::make_shared<rclcpp::AsyncParametersClient>(this->shared_from_this(),
    "turtlesim");

    if (parameter_client->wait_for_service(std::chrono::seconds(5))) {
        parameter_client->set_parameters({
            rclcpp::Parameter("background_r", std::max(0, std::min(r,
255))),
            rclcpp::Parameter("background_g", std::max(0, std::min(g,
255))),
            rclcpp::Parameter("background_b", std::max(0, std::min(b,
255)))
        });
    }
}
```

```

        // 배경색 변경 후 화면 클리어를 위해 서비스 호출
        auto request =
std::make_shared<std_srvs::srv::Empty::Request>();
        auto result = clear_client->async_send_request(request);
    }
}

```

사용자가 RGB 값을 직접 입력하도록 한다. 입력 범위는 0부터 255까지이다. 사용자가 입력한 값이 0부터 ~ 255사이의 값인지 max() 함수를 사용하여 잘못된 값이 입력 되었을 때도 정상적으로 처리되도록 예외 처리를 진행하였다.

rclcpp::AsyncParametersClient를 사용하여 turtlesim 노드의 배경색

파라미터(background_r, background_g, background_b)를 설정했다. 서비스가 안전하게 준비될 때까지 최대 5초간 대기하도록 하였다.

배경색을 변경한 후, 화면에 즉시 반영하기 위해 /clear 서비스를 호출했다. std_srvs::srv::Empty 타입의 요청을 생성하여 클라이언트를 통해 서비스를 호출하여 배경색 변경 작업이 진행 된다.

6.2.3. 거북이 모양 설정 모드

이 기능은 생성 되어 있는 거북이를 사용자가 선택한 거북이 모양(leonardo, raffaello, michelangelo, donatello)으로 변경하는 기능이다. 이 기능의 핵심 포인트는 /kill 서비스와 /spawn 서비스이다.

해당 기능은 구현을 완벽하게 하지 못 했다. 본 기능과 관련된 실패 사례와 시도 방법에 대해서는 [6.3실패 사례와 시도한 방법]에서 다루고 있다.

6.2.4. pen 설정 모드

TurtlesimStyle 클래스는 pen의 색상과 두께를 설정하는 노드이다. 이 노드는 사용자가 입력한 RGB 값과 두께 값으로 pen의 색상과 두께를 설정한다. 생성자와 pen의 설정을 담당하는 메서드로 구성되어 있다.

6.2.4.1. TurtlesimStyle 생성자

```
TurtlesimStyle::TurtlesimStyle() : Node("turtlesim_style") {
    // Service Client 생성:
    //set_pen_client_: /turtle1/set_pen 서비스와 통신하기 위한 클라이언트
    객체임 -> turtle 제어 가능
    //turtlesim::srv::SetPen -> Pen Color, width 등을 제어할 수 있는
    서비스 타입임
    set_pen_client_ =
    this->create_client<turtlesim::srv::SetPen>("/turtle1/set_pen");
}
```

노드의 이름을 turtlesim_style로 설정하였다. 뒤늦게 드는 생각은 다른 노드들과의 통일성을 위해 turtlesim_style_pen이라고 네이밍을 하고, 클래스명도 수정하였으면 더 형식적이고 일목요연하게 시스템을 설계할 수 있을 것이라고 생각한다.

6.2.4.2. pen 스타일 설정 메서드

```
// Pen의 Color와 width를 설정하는 메서드
void TurtlesimStyle::setPenStyle() {
    // 요청 객체 생성 -> Pen 설정을 위한 서비스 요청 객체를 생성함
    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;

while (!set_pen_client_->wait_for_service(std::chrono::seconds(5)))
{
    RCLCPP_WARN(this->get_logger(), "Waiting for the
/turtle1/set_pen service to be available...");
}

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

```

배경색 설정 기능과 유사하다. pen 스타일 설정 메서드에서 추가된 내용은 사용자로부터 값을 입력받아 pen의 두께를 설정하는 것이다. 또한, pen의 on/off도 구현하려고 하였으나, 과제에 제시된 기능 구현을 위주로 작업을 진행하느라 해당 기능은 구현하지 않았다. 두께 입력 처리처럼 똑같이 pen의 on/off 여부도 선택하게 하여 처리했으면 더 완벽한 제어 프로그램이 되었을 거라 생각한다.

pen 스타일 설정 메서드 내부에 while() 문 하나가 존재한다. 이 while 문은 /turtle1/set_pen 서비스가 사용 가능한 상태가 될 때까지 계속 대기하는 로직이다. 현재 본 패키지의 규모는 소규모이기에 이 부분은 없어도 크게 문제가 되지 않을거라 생각하지만, 예외 처리를 해보고 싶어서 관련 내용을 찾아보고 적용해 보았다. while()으로 대기를 해도 되고, 아니면 그냥 딜레이로 대기를 한 다음 이후 로직을 처리해도 될 것 같다는 생각이 든다.

6.2.5.turtlesim_cli 노드

TurtleSimCLI클래스는 구현된 모든 노드와 상호작용하며 사용자로부터 입력을 받아 각 노드를 활성화 해주는 역할의 노드이다. 이 노드는 생성자, run() 메서드, displayMenu() 메서드, getUserInput() 메서드, processInput() 메서드로 구성되어 있다.

6.2.5.1. TurtleSimCLI 생성자

```

TurtleSimCLI::TurtleSimCLI() : Node("turtlesim_cli")
{

```



```

RCLCPP_INFO(this->get_logger(), "TurtleSimCLI Node 초기화");

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

run();
}

```

turtlesim_cli 노드가 초기화 되면 디버깅을 위해 초기화 되었다는 것을 나타낸다. 이후, 본 패키지가 실행되면 turtlesim도 같이 실행하기 위해 system 명령어를 사용하여 실행 처리를 하였다. 실행 처리 없이 turtlesim은 별도로 실행하고, 이후 본 패키지를 실행해도 될 것 같다는 생각이 든다. 이후, run() 메서드를 통해 프로세스를 시작한다.

6.2.5.2. run() 메서드

```

// 프로세스 시작
void TurtleSimCLI::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
    }
}

```

run() 메서드는 turtlesim_cli 노드가 실행되면 이후 프로세스를 처리하기 위해 실행되는 메서드이다. 기본적으로 본 패키지는 거북이 제어를 위해 사용자로부터 입력을 계속 받고, 그에 따른 화면 처리를 진행해야 한다. 그렇기에 while() 문을 통해 터미널에 텍스트를 출력하고, 사용자로부터 입력 처리를 받으며 입력 처리에 따른 하위 프로세스가 작동 되도록 구현하였다. 이 모든 것을 run() 메서드 내부에 구현해도 되겠지만, RO:BIT에서 교육 받은 내용 중 SOLID패턴의 내용을 적용하기 위해 각 프로세스를 분할하여 구현하게 되었다.

6.2.5.3. displayMenu() 메서드

```

void TurtleSimCLI::displayMenu()
{
    std::cout << "==== TurtleSim CLI Menu =====\n";
    std::cout << "1. 조종 모드\n";
    std::cout << "2. 배경색 설정 모드\n";
    std::cout << "3. 거북이 모양 설정 모드\n";
    std::cout << "4. pen 설정 모드\n";
    std::cout << "모드 선택하기 (1-4): ";
}

```

displayMenu() 메서드는 터미널에 사용자가 사용 가능한 명령어 리스트를 보여주는 역할을 담당한다.

6.2.5.4. getUserInput() 메서드

```

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

```

getUserInput() 메서드는 사용자로부터 입력을 받아 반환하는 메서드이다.

6.2.5.5. processInput() 메서드

```

void TurtleSimCLI::processInput(int mode)
{
    switch (mode)
    {
        case 1:
        {
            std::cout << "조종 모드를 선택했음.\n";
            auto turtle_control = std::make_shared<TurtleSimControl>();
            turtle_control->controlLoop();
        }
        break;
        case 2:
        {
            std::cout << "배경 설정 모드를 선택했음.\n";
            auto turtle_style_bg = std::make_shared<TurtlesimStyleBg>();

```

```

        turtle_style_bg->setBackgroundColor(); // setPenStyle 호출
    }
    break;
case 3:
    {
        std::cout << "거북이 모양 설정 모드를 선택했음.\n";
        auto turtle_style_shape =
std::make_shared<TurtlesimStyleShape>();
        turtle_style_shape->setTurtleShape(); // setTurtleShape 호출
    }

    break;
case 4:
    {
        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;
}
}
}

```

processInput() 메서드는 사용자가 입력한 명령어에 따라 하위 프로세스를 처리하는 역할을 담당한다. 사용자가 선택한 모드를 출력하고, 해당 모드가 탑재되어 있는 노드에 있는 메서드를 호출한다.

6.3. 실패 사례와 시도한 방법

개발 과정에서 발생한 문제는 다양하지만, 가장 큰 문제는 거북이의 모양을 변경하는 기능을 완성하지 못 한 것이다. 로직은 여러 방식으로 구현 시도를 해 보았었다.

1. 현재 존재하는 모든 거북이를 삭제하고 사용자가 입력한 이름의 거북이만을 생성한다.
2. 기본 거북이(turtle1)과 사용자가 생성을 진행할 거북이의 모든 이름을 타겟팅하여 삭제하고, 사용자가 입력한 거북이만을 생성한다.
3. 사용자가 이전에 입력한 거북이의 이름을 타겟팅하여 삭제하고, 사용자가 새로 입력한 이름의 거북이를 생성한다.

마지막으로 구현한 로직은 현재 거북이의 이름을 타겟팅하여 삭제한 다음, 사용자가 입력한 새로운 이름의 거북이를 생성하는 방식이다. 그러나, 이 방식으로, 설계한 로직으로 빌드 후 실행할 경우 최초 1회는 거북이가 삭제 되고 새로운 거북이가 나타나지만, 그 이후부터는 기존 거북이가 삭제 되지 않고 생성 되기만 하는 이슈가 발생 되었다.

디버깅을 통해 현재 거북이의 이름을 확인해 보았으며, 계속 turtle1이라는 것을 확인하게 되었다. 생성자에서 선언한 거북이의 이름의 문제이거나, 로직 자체를 바꾸거나 로직 설계를 재저검할 필요성을 느꼈다. 그래서 보고서를 작성하는 지금도 디버깅을 진행하여 작업을 진행하였으나, 원인을 찾아 해결하지 못했다. 디버깅시 확인되는 에러 코드를 통해 알게 된 사실은 다음과 같다.

1. Tried to kill turtle [], which does not exist: 이 에러코드는 거북이 삭제를 시도했으나, 해당 이름의 거북이가 존재하지 않다는 뜻으로 이해하였다. 그러면 거북이의 이름이 생성 이후 반영되지 않다는 것을 유추해 볼 수 있었다.
2. A turtle named [donatello] already exists: 이 에러코드는 이미 해당 이름을 가진 거북이가 존재할 때, 새로운 거북이를 생성하려고 해서 발생하는 에러코드로 이해하였다. 1번 에러 코드와 모순이지 않나 생각이 든다. 이미 거북이의 이름이 존재하면 삭제가 가능해야 하는데, 왜 1번 에러코드에서는 존재하지 않다고 표기하는지 잘 이해가 안 되었다.

이러한 에러와 이슈를 해결하기 위해 다음과 같은 방법을 찾아보고 시도하였다.

1. 레퍼런스 찾기

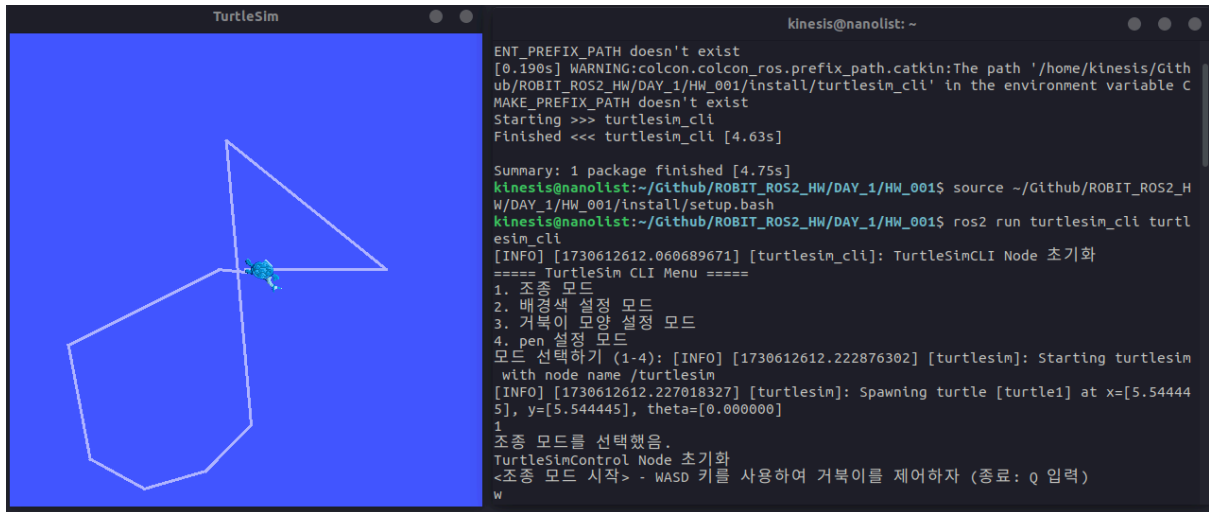
구글링과 오로카, 표윤석 박사님께서 제작하신 로봇운영체제 ROS 2 특강 자료 중 turtlesim 관련 부분을 보았다. 오로카에 업로드된 "010 ROS 2 서비스 (service)"에서는 명령어로 거북이 네 마리를 생성하였다. 나 역시 이를 직접 해보았으며, 서로 다른 거북이 네 마리가 생성되는 것을 확인했다. 이를 통해 거북이의 이름이 이스터에그이고, 사용자로부터 해당 이름을 선택하게 하고 해당 이름의 거북이를 스폰하면 되지 않을까 생각했다.

2. GPT 사용하기

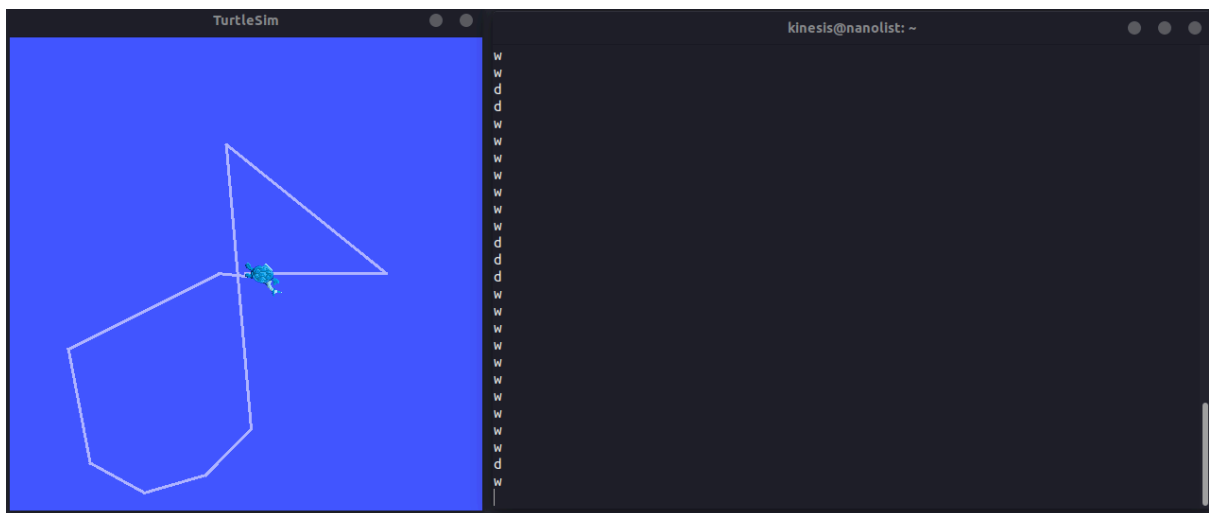
그럼에도 불구하고 제시된 기능을 구현하지 못했다. GPT를 사용하여 내가 처한 상황속에서 왜 안 되고, 구상한 로직에 결함이나 휴먼에러가 있는지 물어보았다. 계속 이 로직이 맞다고 생각하다보니 이 틀 안에 갇혀 생각하게 되는 것만 같았다. GPT를 사용하고 구현을 시도하였음에도 불구하고 끝내 구현하지 못 했다. 그래도 발전 사항이 있다면, 최초 1회는 거북이가 삭제되고 새로운 거북이가 스폰되는 것이다.

결국 해당 기능은 완벽하게 구현하지 못했다.

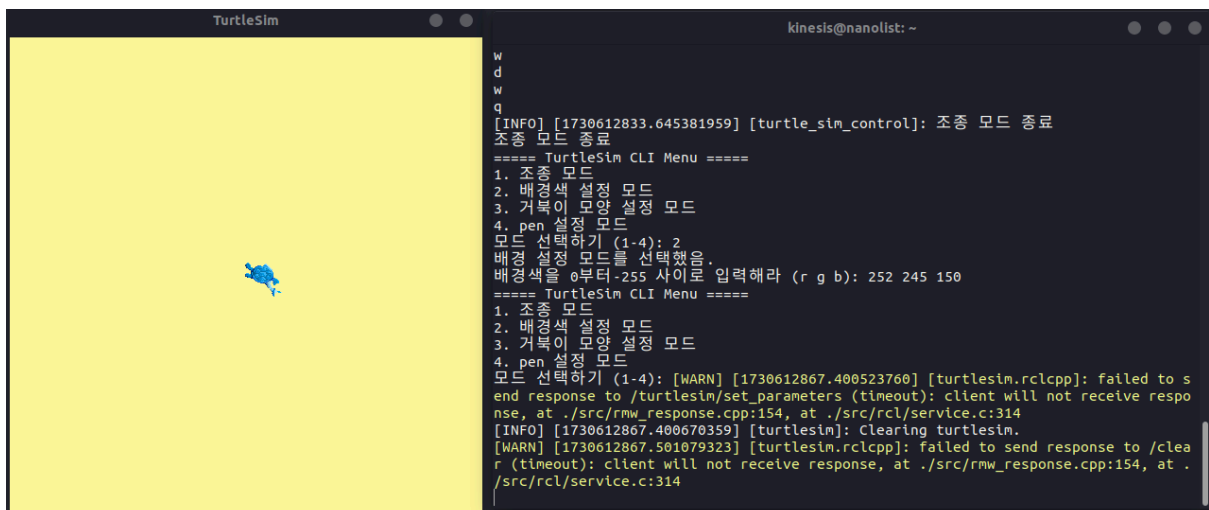
7. 결과



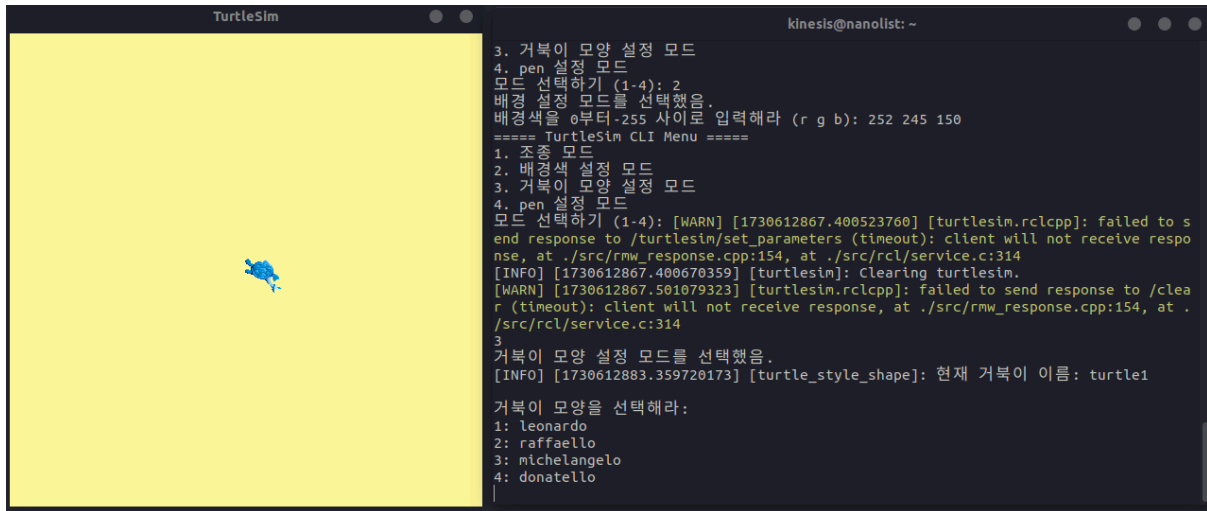
<turtlesim_cli 실행 결과 1>



<turtlesim_cli 실행 결과 2>



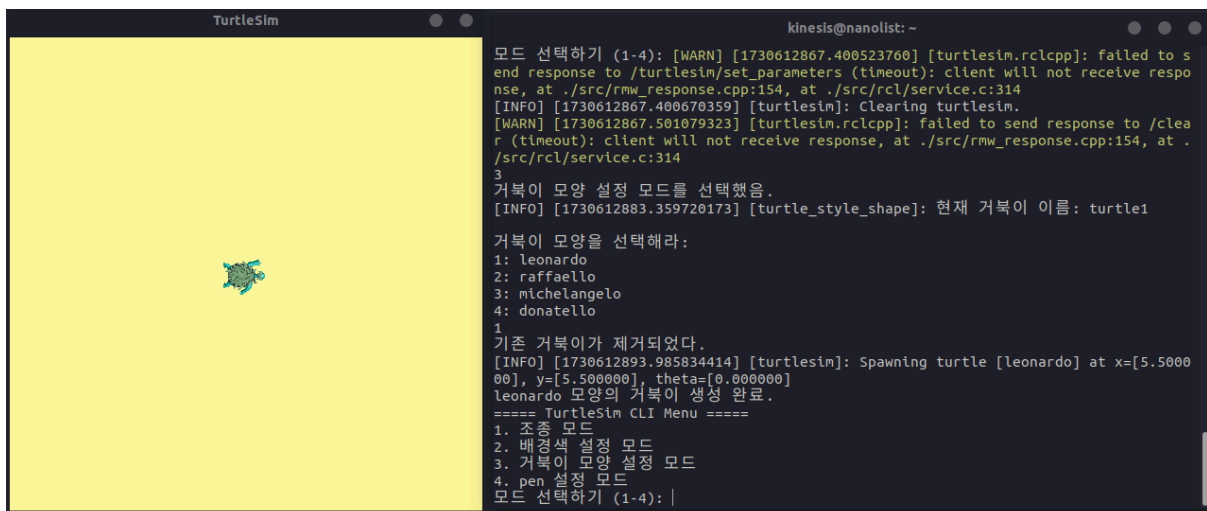
<turtlesim_cli 실행 결과 3>



```
TurtleSim
kinesis@nanolist: ~

3. 거북이 모양 설정 모드
4. pen 설정 모드
모드 선택하기 (1-4): 2
배경 설정 모드를 선택했음.
배경색을 0부터 255 사이로 입력해라 (r g b): 252 245 150
===== TurtleSim CLI Menu =====
1. 조종 모드
2. 배경색 설정 모드
3. 거북이 모양 설정 모드
4. pen 설정 모드
모드 선택하기 (1-4): [WARN] [1730612867.400523760] [turtlesim.rclcpp]: failed to send response to /turtlesim/set_parameters (timeout): client will not receive response, at ./src/rmw_response.cpp:154, at ./src/rcl/service.c:314
[INFO] [1730612867.400670359] [turtlesim]: Clearing turtlesim.
[WARN] [1730612867.501079323] [turtlesim.rclcpp]: failed to send response to /clear (timeout): client will not receive response, at ./src/rmw_response.cpp:154, at ./src/rcl/service.c:314
3
거북이 모양 설정 모드를 선택했음.
[INFO] [1730612883.359720173] [turtle_style_shape]: 현재 거북이 이름: turtle1
거북이 모양을 선택해라:
1: leonardo
2: raffaello
3: michelangelo
4: donatello
|
```

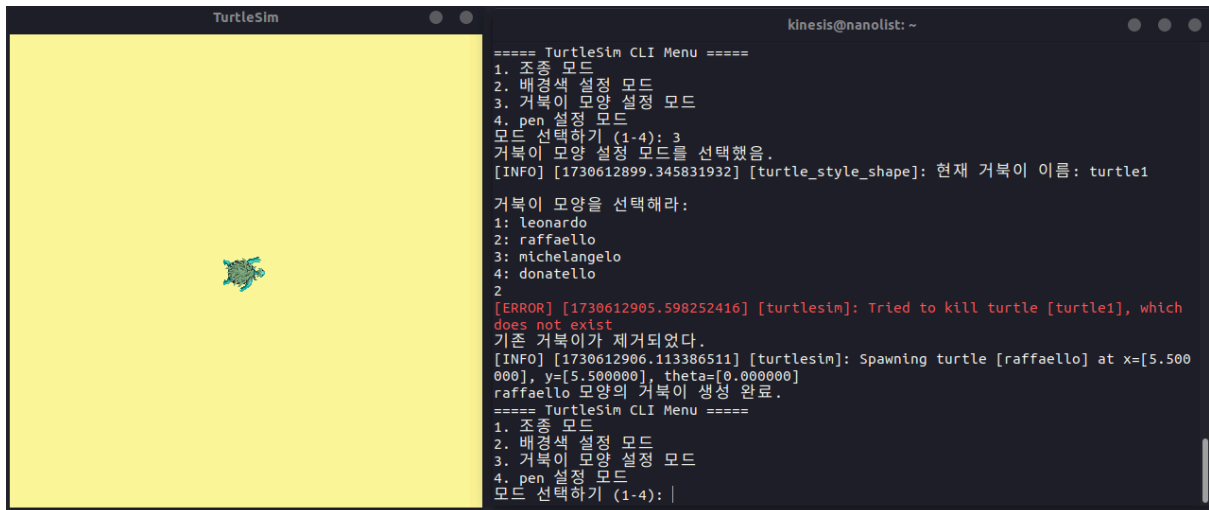
<turtlesim_cli 실행 결과 4>



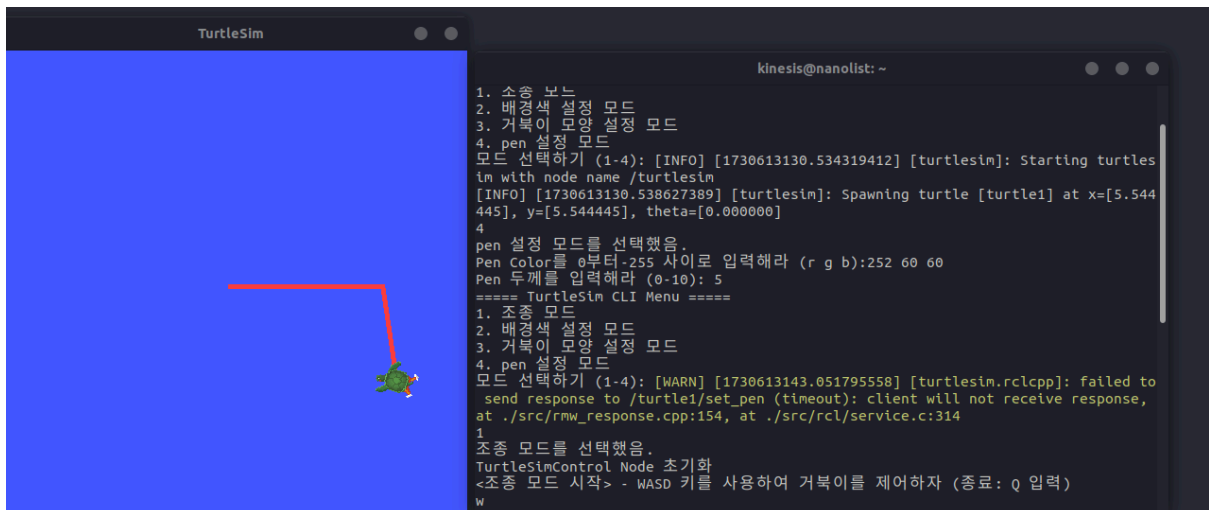
```
TurtleSim
kinesis@nanolist: ~

모드 선택하기 (1-4): [WARN] [1730612867.400523760] [turtlesim.rclcpp]: failed to send response to /turtlesim/set_parameters (timeout): client will not receive response, at ./src/rmw_response.cpp:154, at ./src/rcl/service.c:314
[INFO] [1730612867.400670359] [turtlesim]: Clearing turtlesim.
[WARN] [1730612867.501079323] [turtlesim.rclcpp]: failed to send response to /clear (timeout): client will not receive response, at ./src/rmw_response.cpp:154, at ./src/rcl/service.c:314
3
거북이 모양 설정 모드를 선택했음.
[INFO] [1730612883.359720173] [turtle_style_shape]: 현재 거북이 이름: turtle1
거북이 모양을 선택해라:
1: leonardo
2: raffaello
3: michelangelo
4: donatello
1
기존 거북이가 제거되었다.
[INFO] [1730612893.985834414] [turtlesim]: Spawning turtle [leonardo] at x=[5.000000], y=[5.000000], theta=[0.000000]
leonardo 모양의 거북이 생성 완료.
===== TurtleSim CLI Menu =====
1. 조종 모드
2. 배경색 설정 모드
3. 거북이 모양 설정 모드
4. pen 설정 모드
모드 선택하기 (1-4): |
```

<turtlesim_cli 실행 결과 5>



<turtlesim_cli 실행 결과 6>



<turtlesim_cli 실행 결과 7>

8. 결론 및 향후 발전 방향

이번 프로젝트는 ROS2 환경에서 터미널 명령을 통해 turtlesim 시뮬레이션을 직관적으로 제어할 수 있는 패키지이다. 주요 기능은 개발한 패키지 내에서 사용자로부터 입력을 받아 거북이를 조종, 배경색 변경, 거북이 모양 변경, pen 설정이다. 이 중에서 거북이의 모양을 변경하는 기능에서 이슈가 발생하였으며, 해결하기 위해 시도해 보았으나 끝내 완벽하게 구현해 내지는 못 했다. 하나의 모드에서 다 포함하여 구현해도 되었지만, 최대한 교육 받은 내용을 바탕으로 구현하기 위해 고민을 많이하고 시간을 많이 들인 패키지이다.

ROS2에 대한 기본적인 개념을 한 번 보고 패키지를 개발하는 것은 어렵게 느껴졌다. 과제 수행을 위해 공부하면서 개발하는 것이 맞으나, ROS2의 주요 개념과 로직을 이해하는 데 있어 시간이 오래 걸렸다.

정해진 데드라인이 존재하기에, 최대한 과제 수행을 위해 필요한 필요 개념만을 급하게 공부하면서 패키지 개발을 진행한 것이 기억에 남는다. 물론 개발 직군에서는 무인도에 남겨져서 생존하는 것처럼 스스로 공부하며 성장하는 것이 맞고, 극히 동의한다. 그러나, 공부를 하며 이해하는 데 시간이 오래 걸렸기에 최초 제출일에는 미완성 된 기능이 많이 존재했다. 이해력을 키우고, ROS2 관련 개념에 대하여 충분한 시간을 가지고 천천히 제대로 알고 공부하고 싶다는 생각이 계속 든다.

이후 과제를 수행하면서 ROS2에 대한 개념과 로직 구성 방식에 대해서는 어느정도 이해하게 된 것 같아 다행이라는 생각이 든다. 시간이 된다면 4일차까지의 모든 과제를 완수한 이후에 머릿속에 엉켜 있는 개념들을 풀어가며 재정리하려고 한다. 이를 통해 ROS2에 대한 개념을 완벽히 이해하여 이후 과제를 수행하려고 한다.

9. 참고 문헌

- 표윤석. (2020.08.27.). "010 ROS 2 서비스 (service)".
<https://cafe.naver.com/openrt/24128>
- 표윤석. (2024.09.05.). "로봇 운영체제 ROS 2 특강 발표 자료",
<https://docs.google.com/presentation/d/1QvUHj8iZCWIGiWRCgHFG-DLAmoUK6kh09xBXCzjPTTs/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_cli

- |—— CMakeLists.txt
- |—— include
 - | |—— turtlesim_cli
 - | |—— turtlesim_cli.hpp
 - | |—— turtlesim_control.hpp
 - | |—— turtlesim_style_bg.hpp
 - | |—— turtlesim_style_pen.hpp
 - | |—— turtlesim_style_shape.hpp
- |—— package.xml
- |—— src
 - |—— main.cpp
 - |—— turtlesim_cli.cpp
 - |—— turtlesim_control.cpp
 - |—— turtlesim_style_bg.cpp
 - |—— turtlesim_style_pen.cpp
 - |—— turtlesim_style_shape.cpp