

1일차 과제 보고서

- 로봇 19기 예비단원 김근형

1일차 과제

1. turtlesim CLI 제어 패키지 제작(패키지명 turtlesim_cli)

- 기능
 - 터미널모 모드 설정 기능
 - 조종모드
 - 배경색 설정 모드
 - 거북이 모양 설정
 - ⇒ 거북이 모양 바꾸는 기능을 구현하지 못하였다.
 - pen 설정 모드

주요 멤버 변수

각각의 기능을 수행할 때 필요한 ROS2 퍼블리셔, 클라이언트 객체들이 클래스 내에서 지속적으로 접근 가능하도록 하기 위함이다.

1. control_publisher_

- `geometry_msgs::msg::Twist` 메시지 타입의 퍼블리셔로, 터틀의 속도와 방향을 제어하기 위한 명령을 퍼블리싱한다.

2. set_pen_client_

- `turtlesim::srv::SetPen` 서비스 클라이언트로, 펜의 색상이나 두께 등을 설정하기 위해 사용된다.

3. clear_client_

- `std_srvs::srv::Empty` 타입의 서비스 클라이언트로, 배경을 지우는 서비스 호출을 할 때 사용된다.

4. spawn_client_

- `turtlesim::srv::Spawn` 서비스 클라이언트로, 새 터틀을 추가할 때 사용된다.

5. kill_client_

- `turtlesim::srv::Kill` 서비스 클라이언트로, 특정 터틀을 종료하거나 삭제할 때 사용된다.

6. available_shapes_

- 터틀이 사용할 수 있는 다양한 모양을 정의한 `std::vector` 입니다. 각 요소는 이름과 모양에 대한 정보의 쌍으로 구성된다.

```
auto parameter_client = std::make_shared<rclcpp::AsyncParametersClient>(this-
    if (parameter_client->wait_for_service(std::chrono::seconds(5))) {
        parameter_client->set_parameters({
```

```

        rclcpp::Parameter("background_r", r),
        rclcpp::Parameter("background_g", g),
        rclcpp::Parameter("background_b", b)
    });

    auto request = std::make_shared<std_srvs::srv::Empty::Request>();
    auto result = clear_client_->async_send_request(request);

    if (rclcpp::spin_until_future_complete(this->shared_from_this(), resu
        std::cout << "배경 색 변경\n";
    }
}

```

`auto request :`

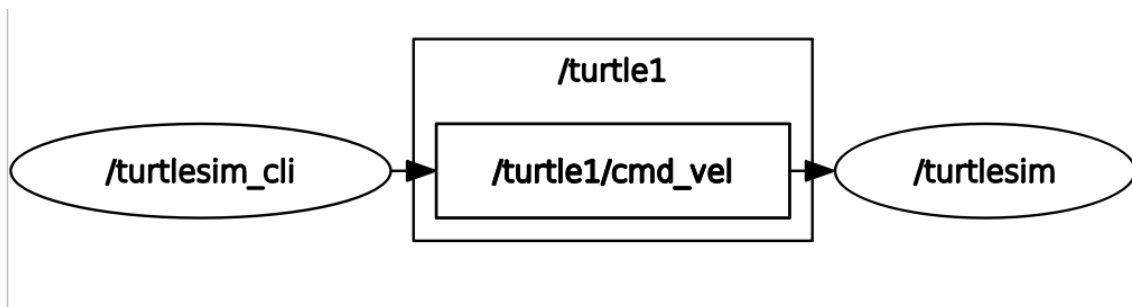
`std_srvs::srv::Empty::Request` 타입의 요청 객체를 생성하는 코드

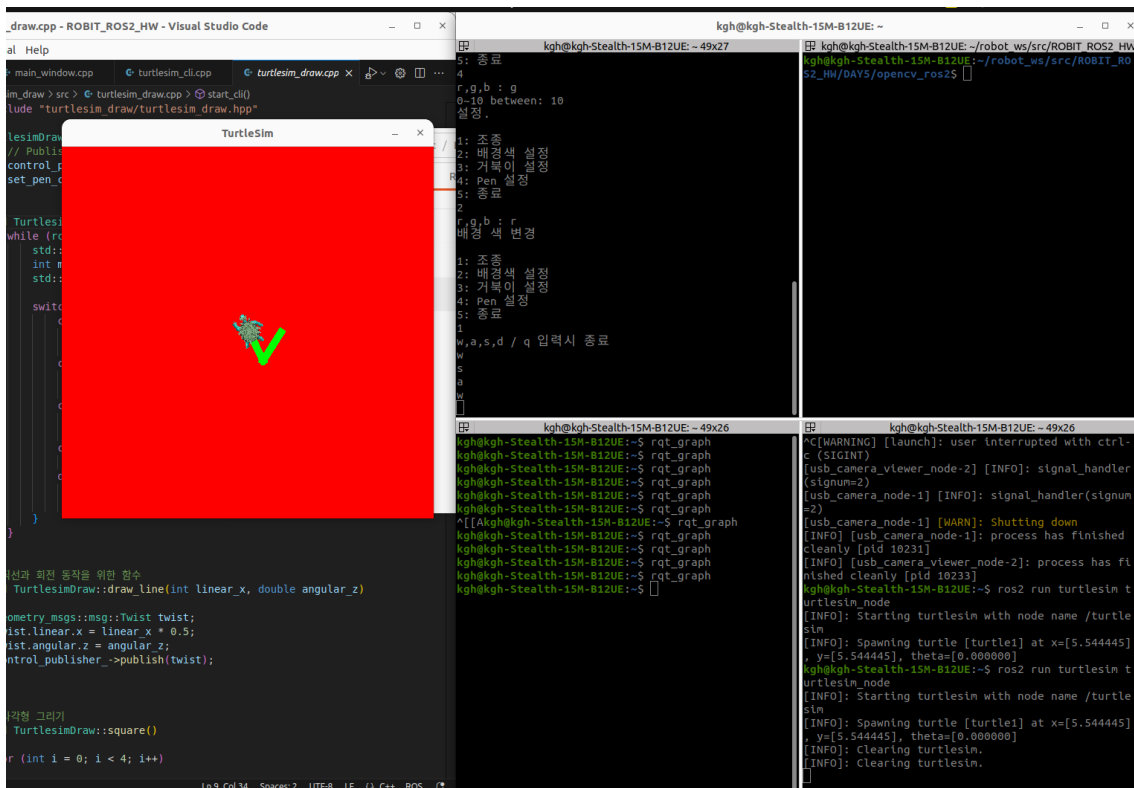
`std_srvs::srv::Empty :`

ROS 2에서 사용하는 기본적인 서비스 메시지 타입으로, 요청 데이터가 필요 없는 경우 사용

`std::make_shared<std_srvs::srv::Empty::Request>()` 는 `std_srvs::srv::Empty::Request` 의 빈 요청 객체를 생성하고, 이를 공유 포인터(`std::shared_ptr`)로 반환

`request` 객체는 이후 `clear_client_->async_send_request(request)` 에 전달되어 `clear` 서비스를 호출할 때 사용됩니다.



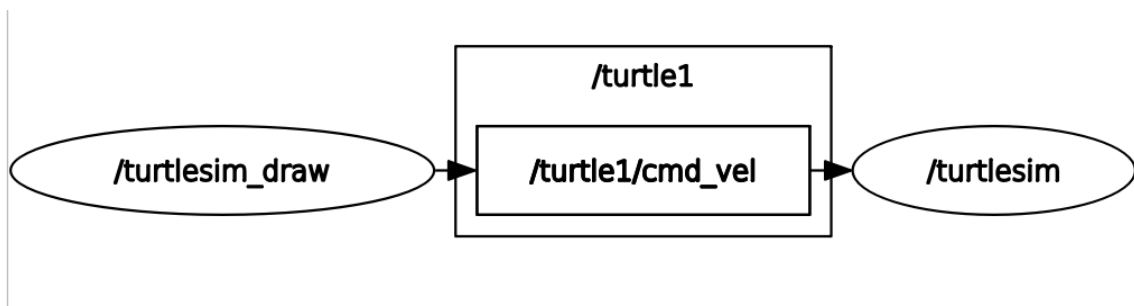


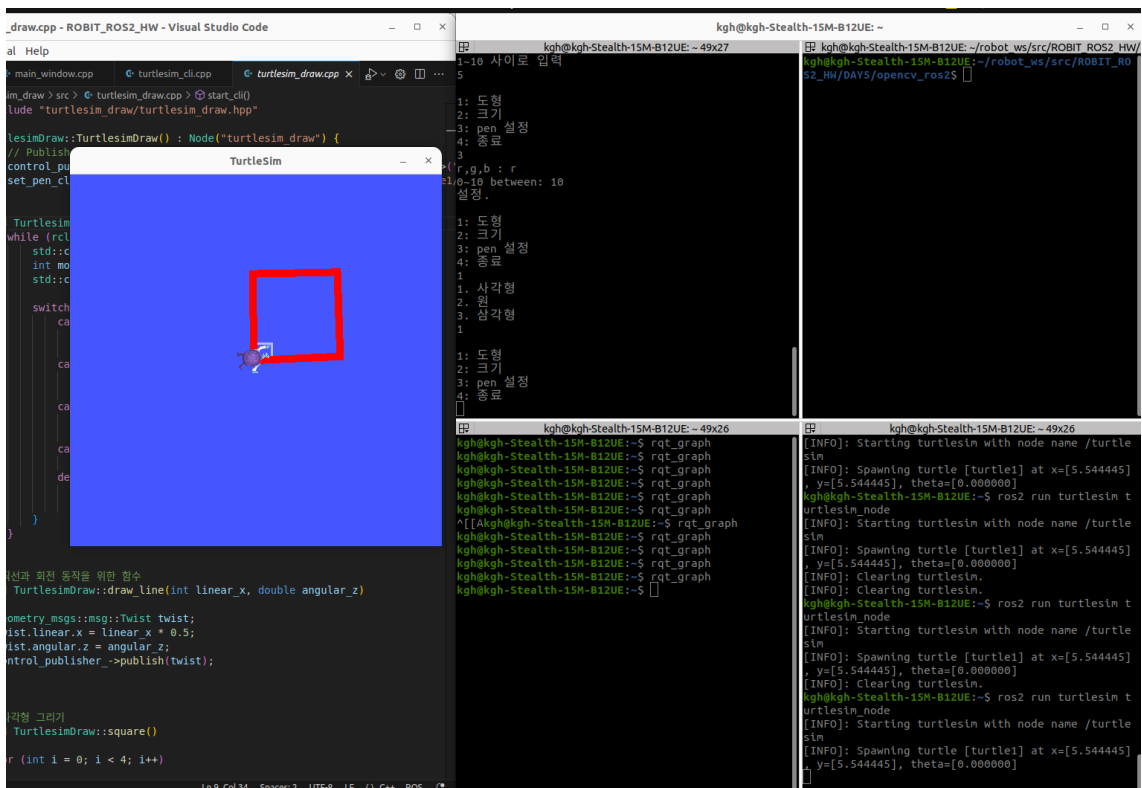
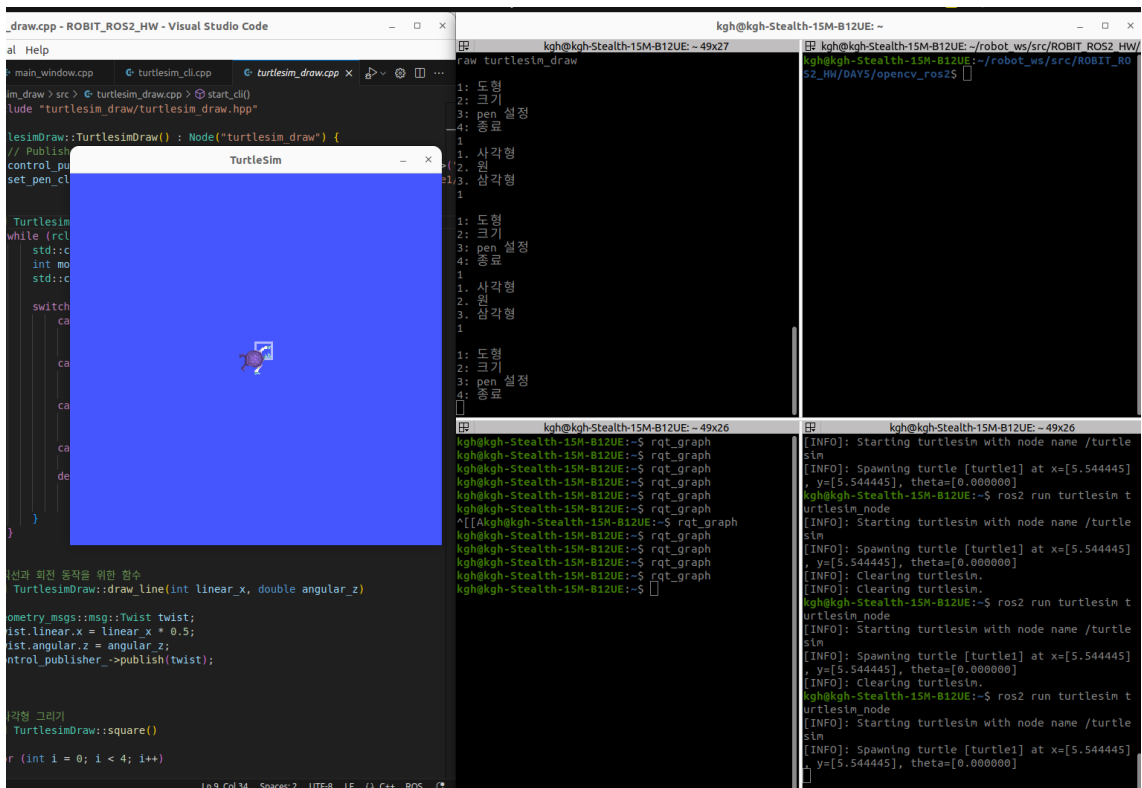
2. turtlesim으로 그림그리기(패키지명 turtlesim_draw)

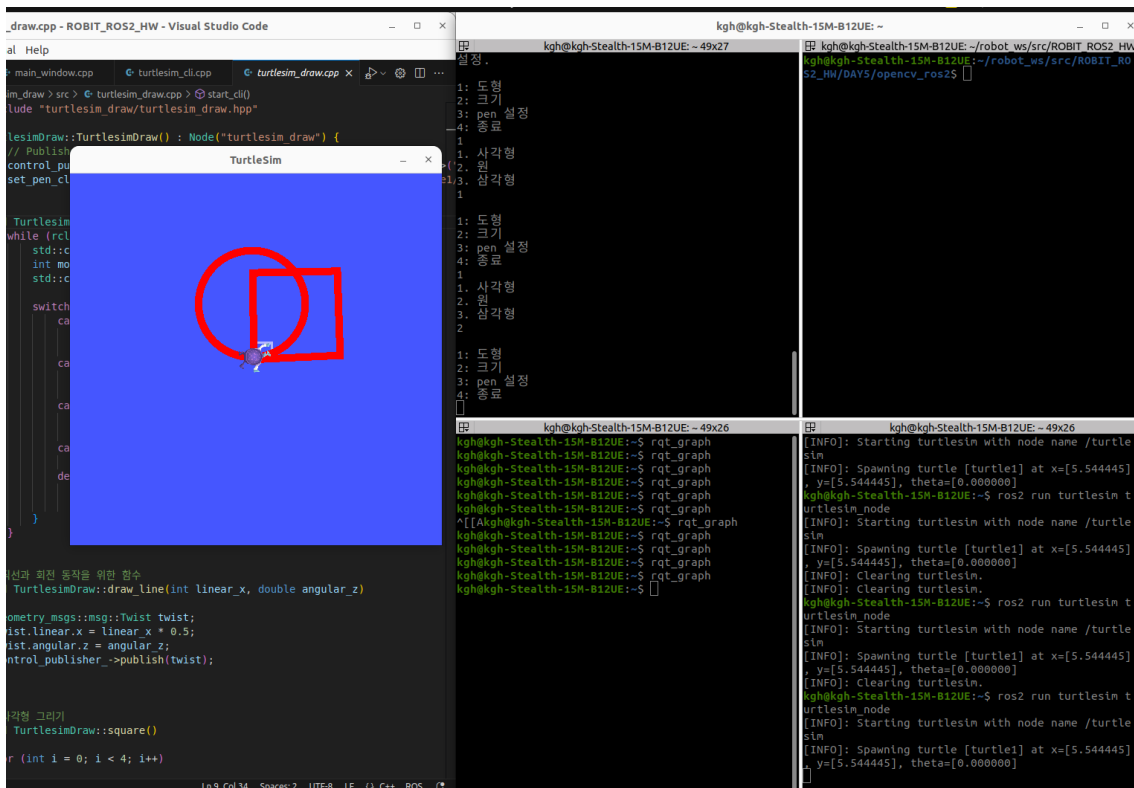
- 기능
 - CLI 입출력을 통해 옵션 설정
 - 삼각형, 원, 사각형 선택
 - 크기 선택
 - pen 색상, 굵기 선택

1번 과제를 바탕으로 2번 과제를 진행하였다.

`draw_shape()` 로 도형 선택 → `square()`, `circle()`, `triangle()` 함수를 통해 도형을 그렸다.







3. demo_node_cpp변경(패키지명 chatter_cli)

- 기능

- 노드 2개(talker, listener), 패키지 한개
- talker 노드에서 터미널로 퍼블리쉬 할 std_msgs/String의 값 입력 받기
- 입력받은 내용 퍼블리쉬
- std_msgs/Int64 사용해서 몇번째 퍼블리쉬 인지 따로 퍼블리쉬
- listener 노드에서 talker 노드에서 퍼블리쉬 한 토픽 2개 서브스크라이브 후 터미널로 출력
- 토픽명 /chatter_cli, /chatter_count 로 고정

- listener

클래스 정의: `Listener` 클래스는 `rclcpp::Node` 를 상속받는다. 이 클래스는 ROS 노드로 동작하며 퍼블리셔로부터 메시지를 수신하는 기능을 한다.

```
rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
```

`Listener` 클래스가 `std_msgs::msg::String` 타입의 메시지를 구독하기 위한 구독자 객체를 가리키는 `subscription_` 이라는 스마트 포인터를 선언

```
subscription_ = this->create_subscription<std_msgs::msg::String>(
    "chatter_cli", 10, std::bind(&Listener::messageCallback, this, std::place
```

"chatter_cli" 라는 토픽을 구독하기 위해 `create_subscription` 메서드를 사용하여 구독자(subscriber)를 생성
`std_msgs::msg::String` 타입의 메시지를 수신하며, 수신된 메시지를 처리하기 위해 `Listener` 클래스의 멤버 함수
`messageCallback` 을 콜백 함수로 설정

```
RCLCPP_INFO(this->get_logger(), "Subscribed: \"%s\", \"%ld\"", latest_message
```

`Listener` 노드가 구독한 메시지와 해당 메시지의 카운트 값을 로그로 출력

- talker

```
Talker::Talker() : Node("talker")
{
    publisher_ = this->create_publisher<std_msgs::msg::String>("chatter_cli", 1
    count_publisher_ = this->create_publisher<std_msgs::msg::Int64>("chatter_co
}
```

`Node("talker")` : `Talker` 라는 이름으로 노드를 초기화

`publisher_` : `std_msgs::msg::String` 메시지를 퍼블리시할 퍼블리셔를 생성하며, 퍼블리시할 토픽은 "chatter_cli" 이다.

`count_publisher_` : `std_msgs::msg::Int64` 메시지를 퍼블리시할 퍼블리셔를 생성하며, 퍼블리시 횟수를 퍼블리시할 토픽은 "chatter_count" 이다.

```
void Talker::publishMessage(const std::string &message)
{
    auto msg = std_msgs::msg::String();
    msg.data = message;
    publisher_->publish(msg); // "chatter_cli" 토픽으로 메시지 퍼블리시

    // 퍼블리시 횟수 증가
    publish_count_++;

    // 퍼블리시 횟수를 Int64 형식으로 생성하여 퍼블리시
    auto count_msg = std_msgs::msg::Int64();
    count_msg.data = publish_count_;
    RCLCPP_INFO(this->get_logger(), "Publishing : \"%s\", \"%ld\"", message.c_s
    count_publisher_->publish(count_msg); // "chatter_count" 토픽으로 퍼블리시
}
```

`msg` 객체에 전달받은 `message` 를 담아 "chatter_cli" 토픽으로 퍼블리시한다.

`count_msg` 에 `publish_count_` 값을 담아 "chatter_count" 토픽으로 퍼블리시한다.

- main

```
auto talker = std::make_shared<Talker>();
auto listener = std::make_shared<Listener>();
```

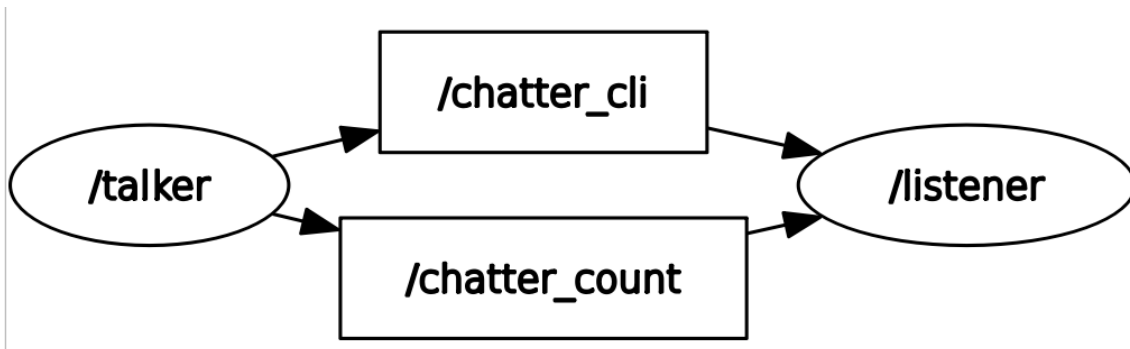
`talker` 와 `listener` 두 개의 노드 생성

```
std::thread listener_thread([&]() {
    rclcpp::spin(listener); file:///home/kggh/Pictures/Screenshots/Screenshot%2
});
```

`Listener` 노드를 별도의 스레드에서 `spin` 시켜 메시지를 계속 수신하고 처리하도록 한다.

스레드를 사용해 `Listener` 를 독립적으로 실행함으로써 `main` 함수에서 `Talker` 를 통한 메시지 퍼블리싱 작업과 병행해서 작동하게 한다.

- spin 함수란
 - 노드가 실행되는 동안 메시지나 서비스 요청, 타이머 이벤트가 발생할 때까지 기다린다.
 - 이벤트가 발생하면, 해당 메시지나 요청을 처리하기 위해 등록된 콜백 함수를 호출한다.




```
kgh@kgh-Stealth-15M-B12UE: ~  
kgh@kgh-Stealth-15M-B12UE: ~ 49x27 kgh@kgh-Stealth-15M-B12UE: ~/robot_ws/src/ROBIT ROS2 HW/  
kgh@kgh-Stealth-15M-B12UE:~$ ros2 run chatter_cli chatter_cli  
input: sdaf  
[INFO]: Publishing : "sdaf", "1"  
[INFO]: Subscribed: "sdaf", "1"  
input: sadf  
[INFO]: Publishing : "sadf", "2"  
[INFO]: Subscribed: "sadf", "2"  
input: wer  
[INFO]: Publishing : "wer", "3"  
[INFO]: Subscribed: "wer", "3"  
input: safd  
[INFO]: Publishing : "safd", "4"  
[INFO]: Subscribed: "safd", "4"  
input:   
kgh@kgh-Stealth-15M-B12UE:~$  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
^[[A kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$ rqt_graph  
kgh@kgh-Stealth-15M-B12UE:~$  
kgh@kgh-Stealth-15M-B12UE:~$
```