DAY1
과제 3번 보고서

2023741024
로봇학부 박건후

목차

(1) 서칭을 통해 알게된 정보

이번 과제를 구현하기 위해서는 publish를 토픽 이름인 turtle1/cmd/vel으로 토픽을 발행하는 것이 필요합니다. 그리고 msg_type 설정과 어떤 값을 보낼 것인지에 대한 매개변수 설정이 필요합니다. 현재 보내야 하는 퍼블리시의 토픽 타입은 geometry_msgs/msg/Twist입니다.

이에 추가로 거북이가 그리는 도형의 굵기, 색깔 또한 바꿔야 합니다. turtlesim이 그리는 선의 굵기, 색깔에 대한 설정은 토픽에 없었기에 다른 방법이 있는 건지 찾아보았습니다. https://menggu1234.tistory.com/18 해당 링크에서

/turtleX/set_pen	turtleX의 펜 색상, 두께 등을 설정

위 사진의 내용을 보았고 service 통신 방식에 해당된다는 것을 알았습니다. 그리고 서비스 통신하는 방법이나 라이브러리를 모르기에 인터넷 검색을 하여 해당 아래 링크를 통해 서비스 통신하는 예제 코드를 참고하였습니다.

https://velog.io/@wjdghks987/%EC%84%9C%EB%B9%84%EC%8A%A4-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8DC

```
auto request = std::make_shared<ArithmeticOperator::Request>();
request->arithmetic_operator = distribution(gen);

using ServiceResponseFuture = rclcpp::Client<ArithmeticOperator>::SharedFuture;
auto response_received_callback = [this](ServiceResponseFuture future) {
    auto response = future.get();
    RCLCPP_INFO(this->get_logger(), "Result %.2f", response->arithmetic_result);
    return;
  };

auto future_result =
  arithmetic_service_client_->async_send_request(request, response_received_callback);
```

```
#include "msg_srv_action_interface_example/srv/arithmetic_operator.hpp"


class Operator : public rclcpp::Node
{
public:
  using ArithmeticOperator = msg_srv_action_interface_example::srv::ArithmeticO

  explicit Operator(const rclcpp::NodeOptions & node_options = rclcpp::NodeOpti
  virtual ~Operator();

  void send_request();

private:
  rclcpp::Client<ArithmeticOperator>::SharedPtr arithmetic_service_client_;
};
```

```
arithmetic_service_client_ = this->create_client<ArithmeticOperator>("arithmetic_operato
```

Request의 타입을 통해서 관련 타입의 멤버를 바꾸고 async_send_request로 해당 값을 적어 통신하는 방식, 서비스 객체 포인터를 선언하고 생성하는 방식, 라이브러리 등등을 알 수 있었습니다.

우선 해당 전달하고자 하는 토픽 이름이나 서비스 이름에 대한 타입이 있지 않으므로 해당 타입에 관한 라이브러리를 가져와야 합니다. 해당 타입을 정확히 알기 위해 서칭을 하였을 때 https://www.youtube.com/watch?v=IbD4JymM3_I 라는 링크에서 더 자세히 알 수 있었습니다. 이 유튜브 채널에서는

5. 서비스 요청 (ros2 service call)

서비스 서버에게 서비스 요청(Request)은 다음과 같이 `ros2 service call` 명령어를 이용하며 매개 변수로 서비스명, 서비스 형태, 서비스 요청 내용을 기술하면 된다.

형태 : ros2 service call 〈service_name〉〈service_type〉"〈arguments〉"

https://docs.ros.org/en/foxy/Tutorials/Services/Understanding-ROS2-Services.html#ros2-service-call

서비스 요청을 위해서는 데이터형을 알아야 하는데 데이터형은 interface show 명령어를 통해 알 수 있다.

```
$ ros2 service list
```

```
(base) kime@kime-VirtualBox:~$ ros2 service list
/clear
/kill
/reset
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```

① /clear 서비스

첫 번째로 다룰 /clear 서비스는 turtlesim 노드를 동작할 때 표시되는 이동 궤적을 지우는 서비스이다. 이
스 요청 후에는 모두 지워짐을 확인할 수 있다. 명령어에서 std_srvs/srv/Empty 이라는 서비스 형태가
명령어에서 "⟨arguments⟩" 가 생략될 수 있다.

```
$ ros2 service call /clear std_srvs/srv/Empty
```

```
$ ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{r: 255, g: 255, b: 255, width: 10}"
```

등등의 지식을 얻을 수 있었고 이를 통해 제가 구현하는 것을 위한 타입, 선언해야 할 헤더 파일들을 어떻게 인클루드 해야하는지도 알 수 있었습니다.

(2) 헤더 파일

```
#include <rclcpp/rclcpp.hpp>
#include <geometry_msgs/msg/twist.hpp> //토픽 타입을 사용하기 위해 헤더 선언
#include <turtlesim/srv/set_pen.hpp>  //색, 두께를 바꿀 수 있는 타입 사용을 위해 선언
#include <std_srvs/srv/empty.hpp> //터틀의 화면을 지우기 위해 비워줄 수 있는 타입 선언
class publisher: public rclcpp::Node{
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr pub; //퍼블리셔 객체 선언
    rclcpp::Client<turtlesim::srv::SetPen>::SharedPtr set_pen; //서비스 클라이언트 객체 선언
    public:
    publisher();
    void publish_message(); //퍼블리셔 함수 선언

};
```

토픽명인 turtle1/cmd_vel, 이에 해당되는 타입인 geometry_msgs/msg/Twist가 있습니다. 이 타입을 사용해야 토픽 메시지를 만들고 publish하는 것이 가능하므로 이 타입을 가져오기 위해 geometry_msgs/msg/twist 헤더 파일을 인클루드하였습니다. 이 때 대문자는 사용할 수 없기에 Twist가 아닌 twist로 한 것입니다. 그리고 이렇게 인클루드한 경우에는 CMakeList에 find_package로 find하고 이에 대한 의존성이 생기기에 의존성에 추가해줘야 합니다. 그리고 package.xml 에서도 이에 대한 depend/속성을 추가해줘야 합니다.

그리고 service 통신에 대해서도 헤더파일을 추가해야 했습니다. turtle1/set_pen 서비스명의 타입으로는 turtlesim/srv/set_pen이 대응되므로 해당 타입을 인클루드해야 했습니다. 그리고 유튜브 영상 속에 /clear라는 것 또한 알게 되었는데 이 /clear 명령어의 타입은 std_srvs/srv/empty이기에 이 또한 인클루드해주었습니다. 이렇게 인클루드한 모든 것들 것 CMakeList와 package.xml에 모두 find하고 의존성을 추가해줬습니다.

그 후에는 인클루드한 것을 통해 토픽발행을 위한 Publisher 클래스에 담을 메시지의 타입을 넣어서 pub 포인터를 만들었고, 서비스 통신에서의 클라이언트가 보내는 통신을 하기 위해 보낼 메시지의 타입을 적고 set_pen이라는 포인터를 생성해주었습니다.

(3) 구현 파일

```
#include "hw3.hpp"
#include <iostream>
#include <chrono>
publisher::publisher():Node("publish_node"){ //노드명 설정
    pub = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel",10); //해당 토픽명으로 퍼블리셔 생성
    set_pen = this->create_client<turtlesim::srv::SetPen>("turtle1/set_pen"); //서비스 클라이언트 생성

}
```

헤더파일에 대한 구현 파일로 생성자를 정의한 부분입니다. 이번에는 키 입력을 통해 발행을 하기 때문에 시간 간격을 통해 발행하지 않으므로 타이머에 대한 함수나 포인터가 필요하지 않습니다. 그래서 create 헬퍼함수를 통해 각각 생성을 해주었습니다.

```
void publisher::publish_message(){
    char ch=0;
    while(1){
     RCLCPP_INFO(this->get_logger(),"W: 정사각형, A: 정삼각형, S: 원, D: 삭제");
     std::cin>>ch; //터미널로부터 값 입력 받기
     //예외처리
     if(!std::cin||(ch!='W'&& ch!= 'w'&&ch!='A'&& ch!='a'&&ch!='S'&& ch!='s'&&ch!='D'&& ch!='d')){
         RCLCPP_INFO(this->get_logger(),"잘못된 입력입니다.");
         std::cin.clear();
         std::cin.ignore(1000,'\n');
         continue;
     }
```

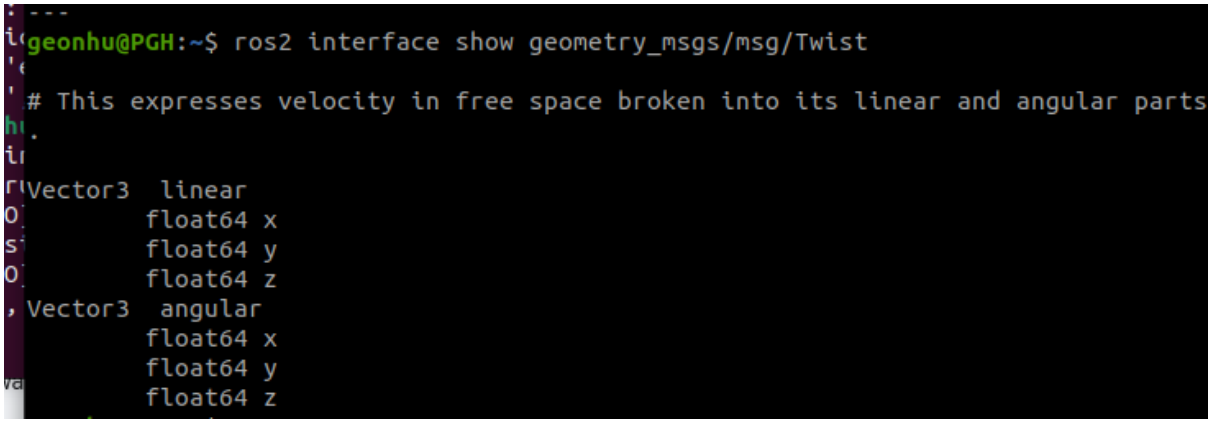
이제 publish를 해줄 메서드인 publish_message에 관한 구현부분입니다. 평범한 입출력처럼 어떤 걸 입력하라는 것을 말해줘야 하므로 어떤 키를 입력해야 된다는 것을 터미널에 출력하도록 했습니다. 그리고 cin을 통해 값 입력을 받는 것으로 했습니다. 기존에 해왔었던 입출력 모두 터미널에 입력하는 형식이었기에 문제가 없을 것이라고 생각했었습니다. 그리고 입력 받은 것이 지정해준 것이 아닐 경우에 대한 예외처리를 해주었습니다. cin의 상태를 정상 상태로 초기화하고, 버퍼를 초기화한 후 다시 입력을 받게 한 것입니다.

```
if(ch == 'W'|| ch=='w'){ //w 입력시 정사각형 그림
   auto req = std::make_shared<turtlesim::srv::SetPen::Request>(); //펜 설정을 바꾸기 위해서는 Request사용해야 함
   req->r = 255; //빨간색으로 설정
   req->g = 0;
   req->b = 0;
   req->width = 3;
   set_pen->async_send_request(req); //서비스 요청
   geometry_msgs::msg::Twist temp; //메시지 타입 객체 생성
   for(int i=0;i<4;i++){
   temp.linear.x=2.0;
   temp.angular.z=0.0;
   pub-> publish(temp);
   std::this_thread::sleep_for(std::chrono::seconds(3));
   temp.linear.x=0.0;
   temp.angular.z=1.57;
   pub-> publish(temp);
   std::this_thread::sleep_for(std::chrono::seconds(3));
}
```

w나 W키보드를 입력받은 경우에 네모를 그리게 만들려고 했습니다. 그림을 그리기 전에 우선 각 그림에 대한 색깔과 두께를 바꾸고 그려야 하므로 우선 서비스 통신을 통해 바꾼 후에 토픽 통신으로 움직이게 하는 것이 좋을 것 같다는 생각이 들었습니다.

```
geonhu@PGH:~$ ros2 interface show turtlesim/srv/SetPen
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
```

처음에는 turtlesim::srv::SetPen 객체를 내부적으로 temp 변수로 만들어서 그 객체에 대한 멤버인 r,g,b, width, off 등을 바꾸려고 했습니다. 하지만 has no member라는 에러가 뜨면서 다른 방법을 찾다가 Request방식을 통해 사용해야 한다는 것을 알게 되었습니다. 그래서 Request 타입의 객체와 스마트 포인터를 생성해서 클라이언트 포인터를 통해 서비스 통신을 할 수 있었습니다. 그 후에는 토픽을 사용하여 이동 정보, 각 정보를 담아서 이를 publish하면 될 것입니다.



위 사진은 geometry_msgs::msg::Twist 타입에 대한 설정해야 할 멤버의 정보를 터미널로부터 가져온 것을 의미하는 사진입니다.

```
geometry_msgs::msg::Twist temp; //메시지 타입 객체 생성
for(int i=0;i<4;i++){
temp.linear.x=2.0;
temp.angular.z=0.0;
pub-> publish(temp);
std::this_thread::sleep_for(std::chrono::seconds(3));
temp.linear.x=0.0;
temp.angular.z=1.57;
pub-> publish(temp);
std::this_thread::sleep_for(std::chrono::seconds(3));
```

이 정보를 통해 temp라는 객체를 생성하고 이에 대한 정보를 넣고 publish하여 직진 정보를 넣고, 그 직진까지 가는 시간이 필요하기에 3초 정도를 sleep을 시켜서 딜레이를 넣었습니다. 그리고 3초 후에 직각으로 도는 것으로 설정하고 다시 publish하고 3초 쉬는 것을 4번 하여 정사각형을 만들었습니다. 딜레이를 하기 위해서는 sleep이라는 것이 C언어나 C++에서 사용되는데 이를 구현하기 위해 Sleep 함수 구현을 해주는 것을 인터넷 서칭을 해보니

https://wyatti.tistory.com/entry/C%EC%97%90%EC%84%9C-sleep-%ED%95%A8%EC%88%98-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0-%EA%B8%B0%EB%B3%B8-%EA%B0%80%EC%9D%B4%EB%93%9C#google_vignette 링크의

```cpp
#include <iostream>
#include <thread>
#include <chrono>

int main() {
    std::cout << "Start₩n";

    // 3초 동안 sleep
    std::this_thread::sleep_for(std::chrono::seconds(3));
```

부분을 보고 바로 사용하였습니다.

```cpp
 else if(ch== 'A'||ch=='a'){ // 세모 그림
    auto req = std::make_shared<turtlesim::srv::SetPen::Request>();
    req->r = 0;
    req->g = 255; //초록색으로 설정
    req->b = 0;
    req->width = 7; //두께 설정
    set_pen->async_send_request(req);
    geometry_msgs::msg::Twist temp;
    for(int i=0;i<3;i++){
    temp.linear.x=2.0;
    temp.angular.z=0.0;
    pub-> publish(temp);
    std::this_thread::sleep_for(std::chrono::seconds(3));
    temp.linear.x=0.0;
    temp.angular.z=2.09;
    pub-> publish(temp);
    std::this_thread::sleep_for(std::chrono::seconds(3));
 }
}
```

네모에서 했던 것처럼 이번에는 초록색으로 설정하고 두께를 좀 더 두껍게 설정하여 서비스 통신을 하고, 토픽으로 직진하고 120도 돌게 하는 것을 3번 하게 하여 정삼각형을 그린 것입니다.

```
else if(ch== 'S'||ch=='s'){//원 그림
    auto req = std::make_shared<turtlesim::srv::SetPen::Request>();
    req->r = 0;
    req->g = 0;
    req->b = 255; //파란색으로 설정
    req->width = 10; //두께 설정
    set_pen->async_send_request(req);
    geometry_msgs::msg::Twist temp;
    for(int i=0; i<4; i++){
    temp.linear.x=2.0;
    temp.angular.z=1.8;
    pub-> publish(temp);
    std::this_thread::sleep_for(std::chrono::seconds(1));
    }

  }
```

원을 그릴 때에는 파란색으로 설정했고 두께를 더 두껍게 했습니다. 그리고 기존의 원을 그리던 방식을 그대로 사용하여 반복문을 통해 좀 더 이동하도록 하여 구현했습니다.

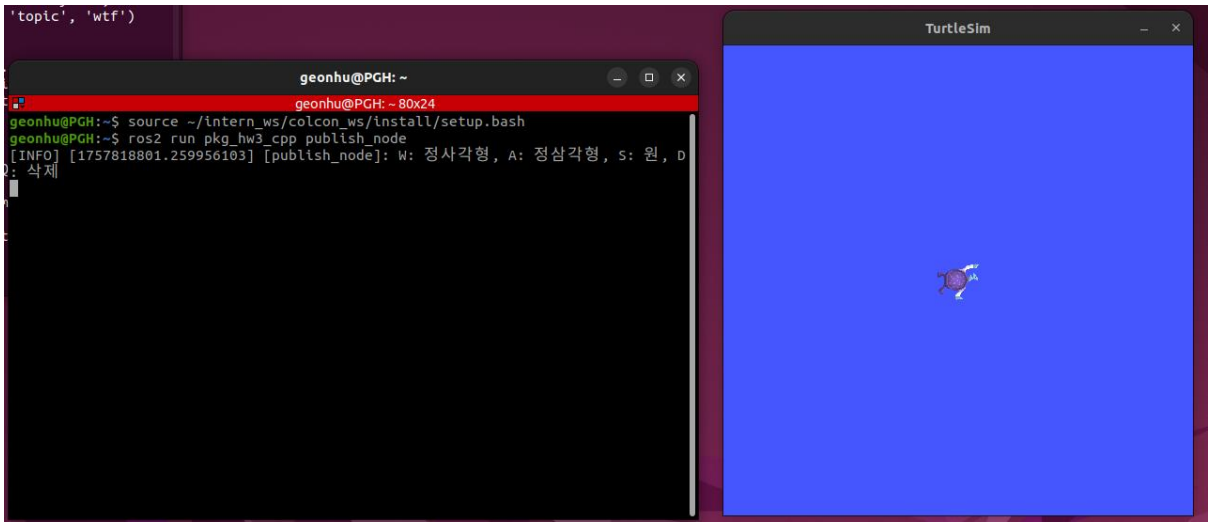
```
else if(ch== 'D'||ch=='d'){ //지금까지 그린 것 삭제
    rclcpp::Client<std_srvs::srv::Empty>::SharedPtr empty_ptr;
    empty_ptr = this->create_client<std_srvs::srv::Empty>("/clear"); //서비스 이름
    auto request = std::make_shared<std_srvs::srv::Empty::Request>(); //서비스 요청하기 위해 Request객체 생성
    empty_ptr->async_send_request(request);
}
```

그리고 D, d를 누르면 지워지는 것이 필요합니다. 하지만 이는 D를 눌렀을 경우에만 실행되는 것이므로 굳이 멤버로 설정하는 것보다는 그때그때 생성해서 사용하는 것이 더 편하고 이롭다고 생각했습니다. 그래서 Client를 else if문 내에서 초기화하고 create로 해당 서비스 이름으로 클라이언트를 생성한 다음에 Request를 생성한 후, Empty타입에는 설정할 멤버가 따로 없기 때문에 바로 request를 넣어서 지금까지 그린 것을 모두 지우는 것을 구현하였습니다.

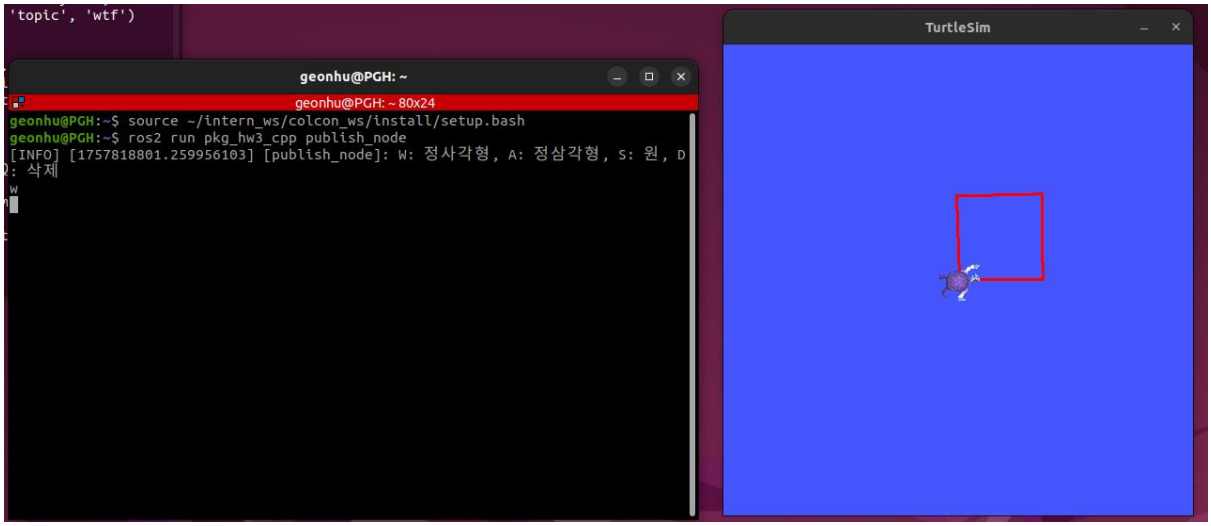
(4) publish main 파일

```cpp
#include "hw3.hpp"
int main(int argc, char * argv[]){
    rclcpp::init(argc,argv);
    auto node = std::make_shared<publisher>();      //노드 객체 생성
    node->publish_message(); //퍼블리셔 함수 실행 while문으로 무한반복하기 때문에 spin이 필요 없음
    rclcpp::shutdown();
    return 0;
}
```

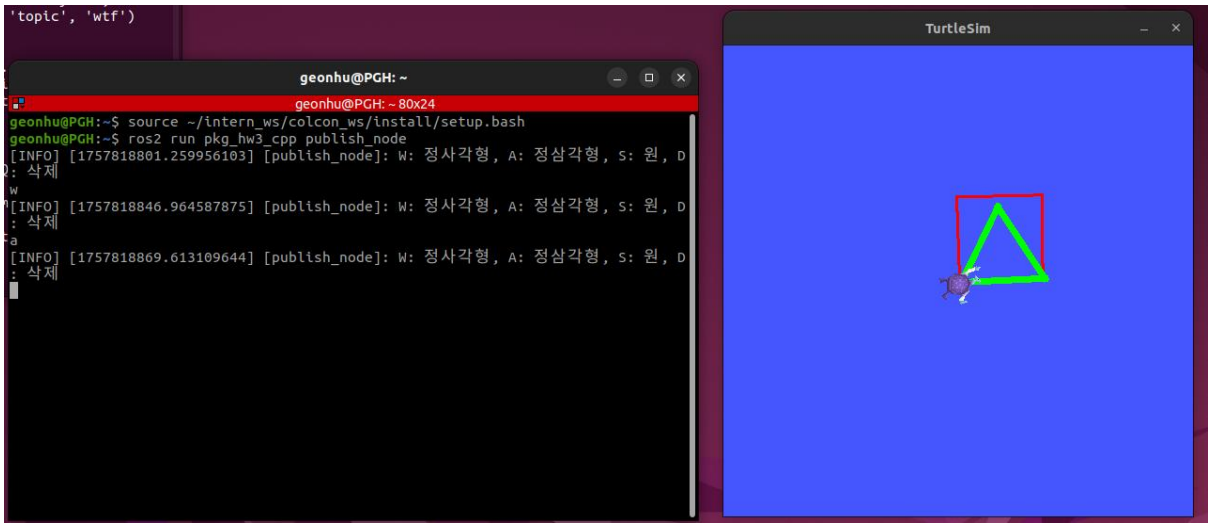
이제 main부입니다. 구현한 클래스를 생성하고 publish_message를 무한 반복하도록 했기 때문에 프로그램이 종료되지 않고 계속 publish할 수 있습니다. 그렇기에 spin()을 없애고 public_message()를 호출을 한 모습입니다.



현재 이렇게 실행하자마자 어떤 걸 입력하면 해당 그림을 그려주는 것을 출력되게 됩니다. 그리고 turtlesim_node를 다른 터미널을 통해 열었습니다.

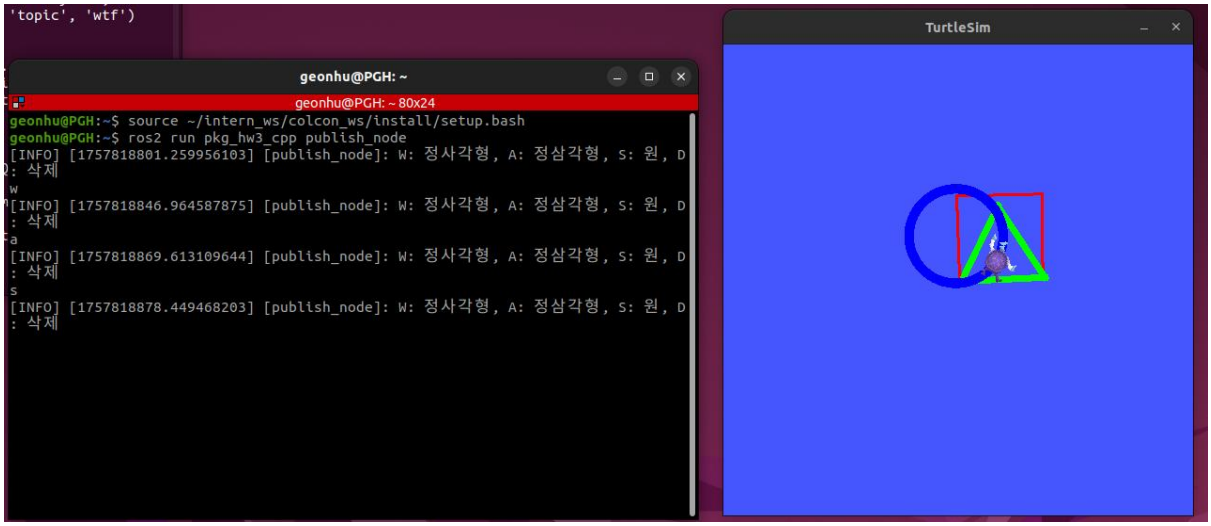


그리고 w를 입력하고 엔터를 쳐서 네모를 그렸습니다.

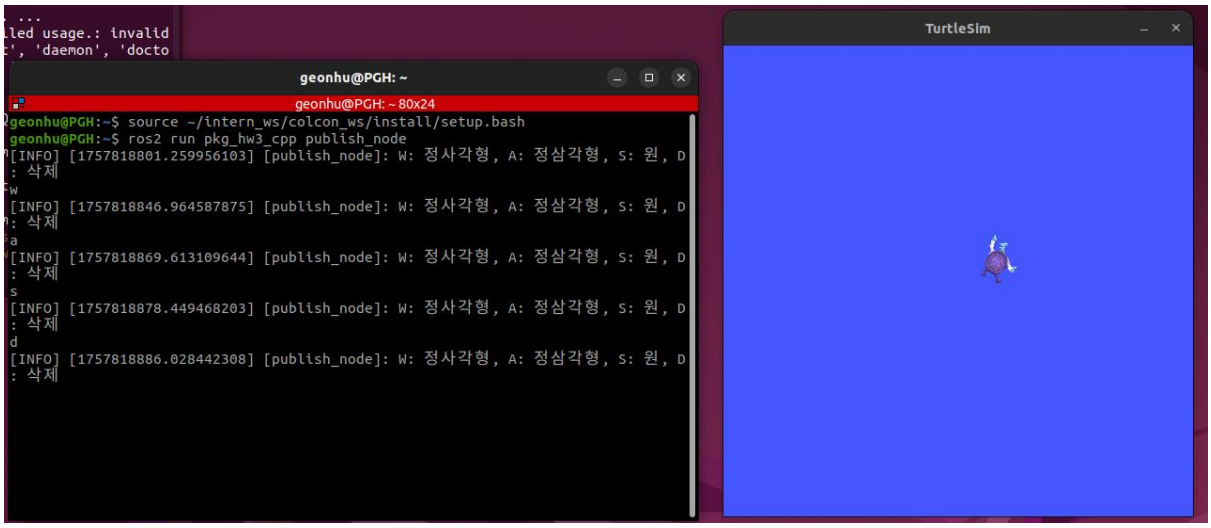


그리고 그 자리에서 a를 입력하고 엔터를 치니 초록색으로 좀 더 두껍게 그린 모습입니다

.



그리고 s를 누르고 엔터를 치니 더 두껍게 파란색으로 원을 그린 것을 볼 수 있습니다.



그리고 d를 누르고 엔터를 치니 그린 모든 것들이 사라진 것을 보실 수 있습니다.