ROS2_day2

day2-1

- QThread 상속 노드 (ROS2 & QT GUI)

  목적 : ros2와 gt gui간의 병렬 실행과 상호작용을 원활하게 처리하기 위함

  과제의 핵심 개념은 rod2의 메세지 전달과 gui 이벤트 처리 간에 충돌 없이 작업을 수행

  1. 병렬 실행 처리
     ROS2는 실시간으로 센서 데이터나 사용자 입력 등 다양한 데이터를 처리하는 작업이 많다. 반면, Qt GUI는 사용자와의 인터페이스에서 필요한 응답성을 유지해야 함. ‘QThread’를 사용하면 ROS2 노드와 Qt의 GUI가 각각 별도의 스레드에서 실행되기 때문에, GUI가 ROS의 데이터 처리로 인해 느려지거나 멈추는 현상을 방지할 수 있습니다.
  2. 메시지 전달 및 구독의 비동기 처리
     ROS2에서 발생하는 메시지는 비동기적으로 전달되기 때문에, Qt GUI와의 상호작용 시 콜백 함수로 데이터를 처리. 이때 ROS2의 콜백이 메인 스레드에서 GUI와 경쟁하게 되면, 메시지 수신 시점에 GUI가 멈출 수 있다. ‘QThread’를 통해 ROS2의 콜백을 별도의 스레드에서 처리하면 이러한 문제를 피할 수 있다.
  3. 응답성 유지
     GUI 애플리케이션은 사용자 입력에 대해 빠르게 응답해야 합니다. 예를 들어 버튼을 클릭하거나 슬라이더를 움직이는 등의 동작에 대해 즉각적인 피드백을 제공하는 것이 중요합니다. ROS2와 GUI가 같은 스레드에서 실행될 경우, ROS2의 일부 작업이 길어지면 GUI가 응답하지 않을 수 있습니다. ‘QThread’를 사용하여 ROS2 작업을 분리하면 GUI의 응답성을 유지할 수 있습니다.
  4. ROS2 와 Qt의 서로 다른 실행 주기
     ROS2의 메시지 루프와 Qt의 이벤트 루프는 서로 다르게 동작합니다. 각각의 루프가 독립적으로 동작할 수 있도록 분리하는 것이 좋은 설계이며, 이 역할을 ‘QThread’를 사용해 구현할 수 있습니다. 즉, ROS2의 노드가 백그라운드 스레드로 실행되어 ROS 메시지를 지속적으로 처리하는 동안, 메인 스레드에서 Qt GUI가 사용자와의 상호작용을 독립적으로 처리할 수 있게 됩니다.

- qnode.hpp

```
class QNode : public QThread {
    Q_OBJECT
public:
    QNode();
    ~QNode();

    void sendTwist(double linear, double angular);
    void set_background_color(const std::string &color);
    void setPen(const std::string& color, int width);
    void drawCircle(double radius, double speed);
    void drawSquare(double side_length, double speed);
    void drawTriangle(double side_length, double speed);
    void run() override;

private:
    std::shared_ptr<rclcpp::Node> node;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr tw
    rclcpp::Client<std_srvs::srv::Empty>::SharedPtr clear_cli
    rclcpp::Client<turtlesim::srv::SetPen>::SharedPtr set_pen_
    rclcpp::Subscription<geometry_msgs::msg::Twist>::SharedPt

Q_SIGNALS:
    void rosShutDown();
    void twistReceived(double linear, double angular);
```

과제 day1-1,2에 구현한 로직 기반 함수, 멤버 변수들

- main_window.cpp

```
MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent)
{
    ui->setupUi(this);

    QIcon icon("://ros-icon.png");
    this->setWindowIcon(icon);

    qnode = new QNode();            //ros2 통신을 위한 qnode 객체
```

```
    QObject::connect(qnode, SIGNAL(rosShutDown()), this, SLOT
    connect(qnode, &QNode::twistReceived, this, &MainWindow::

    qnode->start();
}
```

QNode 는 QThread 상속받고 있으며, qnode→start() 를 호출함으로써 별도의 스레드에서 ros2 관련 작업이 실행되게 됨, run()메서드에서 rclcpp::spin(node)를 호출하여 ros2 노드가 지속적으로 토픽을 처리하도록 함

- 도형 그리는 함수에 따로 스레드를 생성

```
void QNode::drawCircle(double r, double speed) {
    std::thread([this, r, speed]() {
        double circumference = 2 * 3.14 * r; // 원의 둘레
        double duration = circumference / speed; // 전체 이동 시
        double travel_time = duration / 20; // 20단계로 나누어 0

        for (int i = 0; i < 20; i++) { // 20단계 반복
            sendTwist(speed, speed / r); // 선속도와 각속도 설정
            std::this_thread::sleep_for(std::chrono::millisec
        }
        sendTwist(0, 0); // 멈추기
    }).detach(); // 스레드 분리
}

void QNode::drawSquare(double side_length, double speed) {
    std::thread([this, side_length, speed]() {
        for (int i = 0; i < 4; i++) {
            sendTwist(speed, 0); // 직진
            std::this_thread::sleep_for(std::chrono::millisec
            sendTwist(0, M_PI / 2); // 90도 회전
            std::this_thread::sleep_for(std::chrono::millisec
        }
        sendTwist(0, 0); // 멈추기
    }).detach(); // 스레드 분리
}
```

```
void QNode::drawTriangle(double side_length, double speed) {
    std::thread([this, side_length, speed]() {
        for (int i = 0; i < 3; i++) {
            sendTwist(speed, 0); // 직진
            std::this_thread::sleep_for(std::chrono::millisec
            sendTwist(0, (2 * 3.14) / 3); // 120도 회전
            std::this_thread::sleep_for(std::chrono::millisec
        }
        sendTwist(0, 0); // 멈추기
    }).detach(); // 스레드 분리
}
```

도형을 그리는 작업은 시간이 오래 걸림 ⇒ 따로 스레드를 생성하여 비동기적으로 실행함(응답성 유지 목적)
std::thread는 스레드 객체로, 스레드를 생성할 수 있는 기능을 제공

detach()함수
std::thread 뒤에 .detachj() 호출하여 스레드를 분리
도형 그리는 스레드는 독립적으로 실행

day1-2에 도형 그리기 와 로직이 다른데
사각형과 삼각형은 파이=3.14...를 cmath라이브러리 사용해서 M_PI 사용해서 더 정확한 각도 제어
특히 원은 이동 단계 시간 계산을 해줬는데 이유로는 부드러운 움직임과 , ros2 메세지 갱신 주기 조절 때문에 사용

- cmd_vel값 받아오기

```
twist_subscription_ = node->create_subscription<geometry_msgs
    "/turtle1/cmd_vel", 10,
    [this](const geometry_msgs::msg::Twist::SharedPtr msg) {
        emit twistReceived(msg->linear.x, msg->angular.z);
    }
);
```

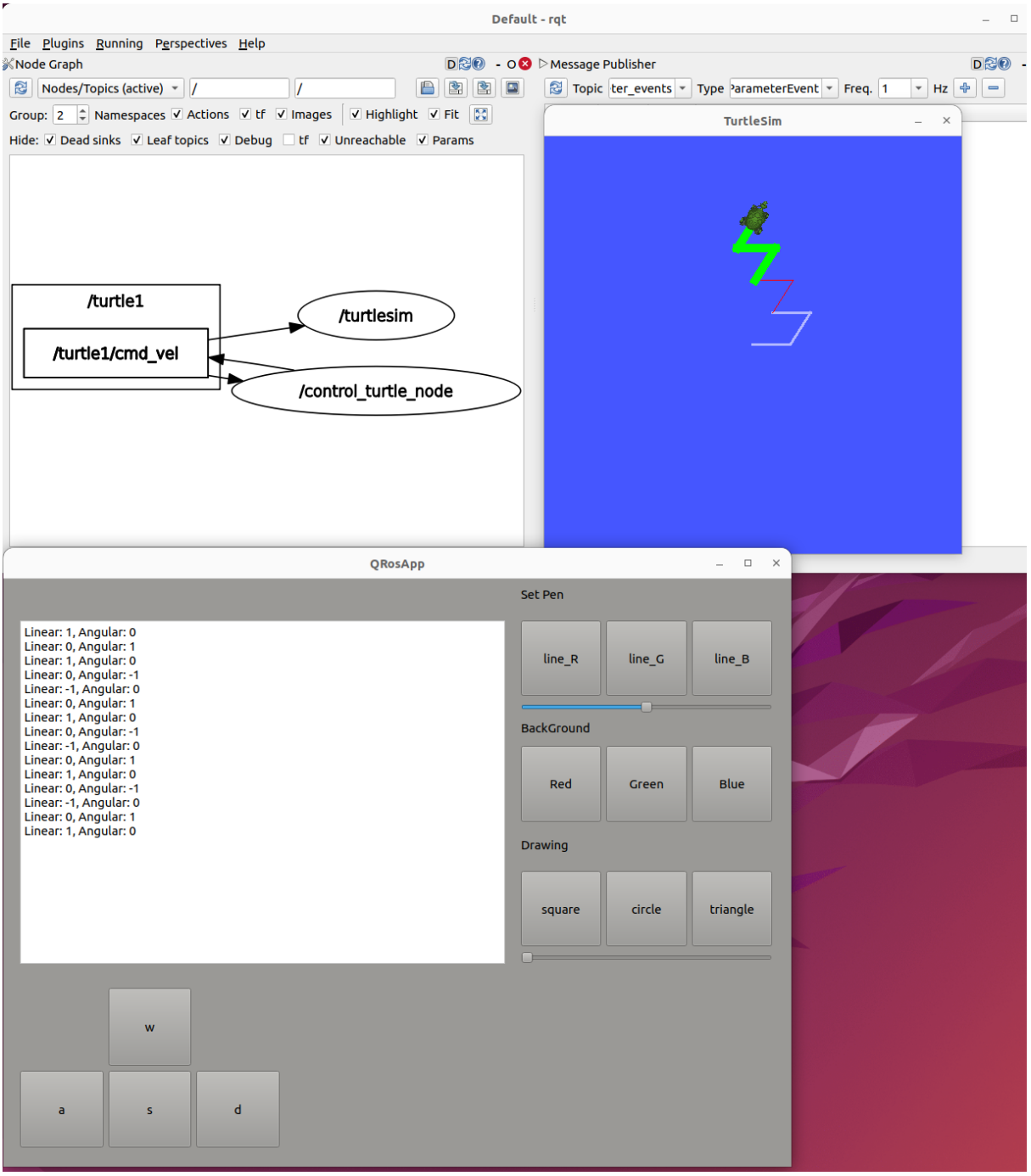
emit twistReceived(msg→linear.x, msg→angular.z);
: 구독 콜백 함수

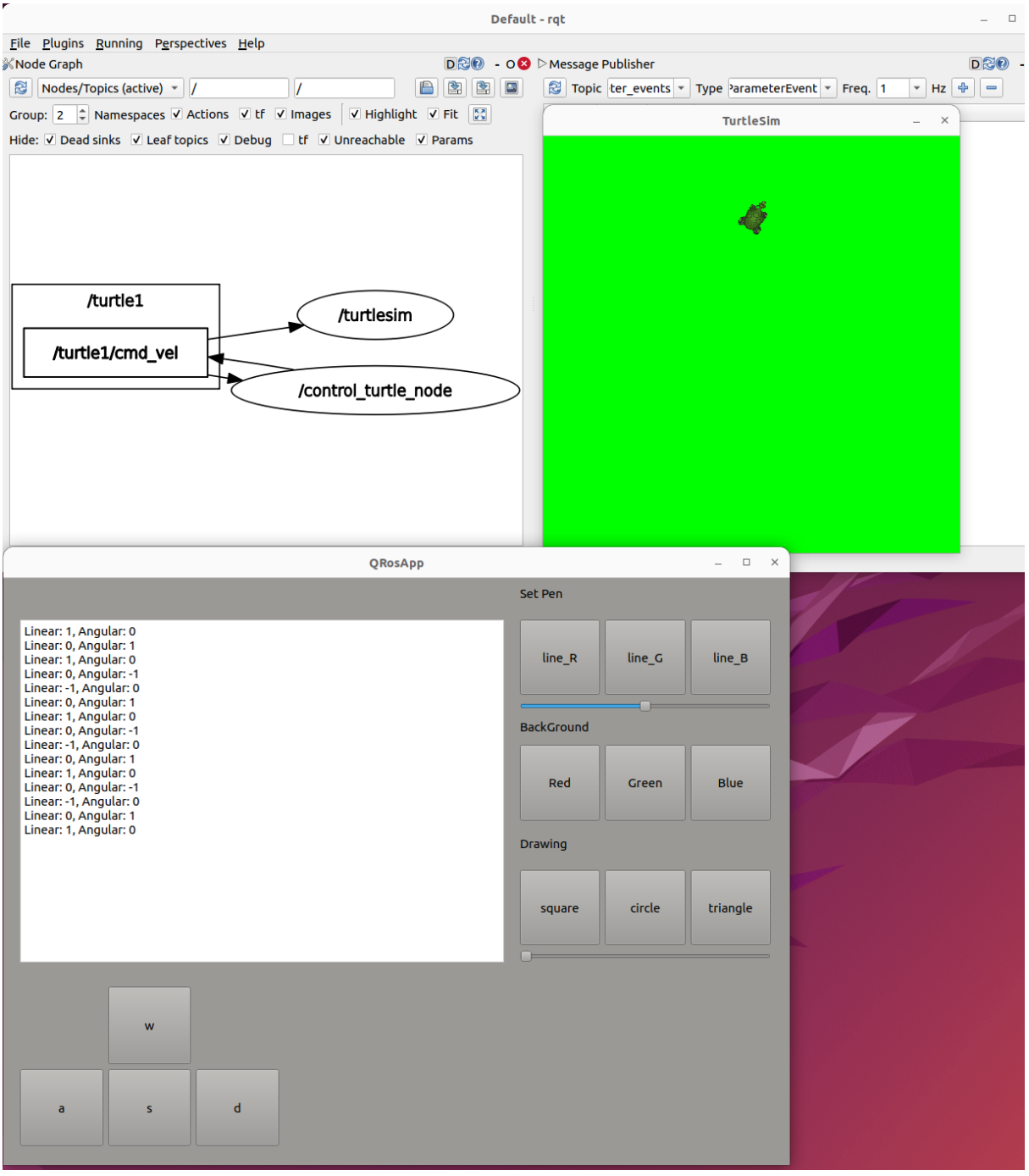
twistReceived라는 Qt신호를 발생

emit 키워드를 사용하여 정의된 신호 발생, 즉 linear, angular 신호를 발생시켜 mainwindow와 연결된있는 슬롯 함수가 이를 처리할 수 있도록 함

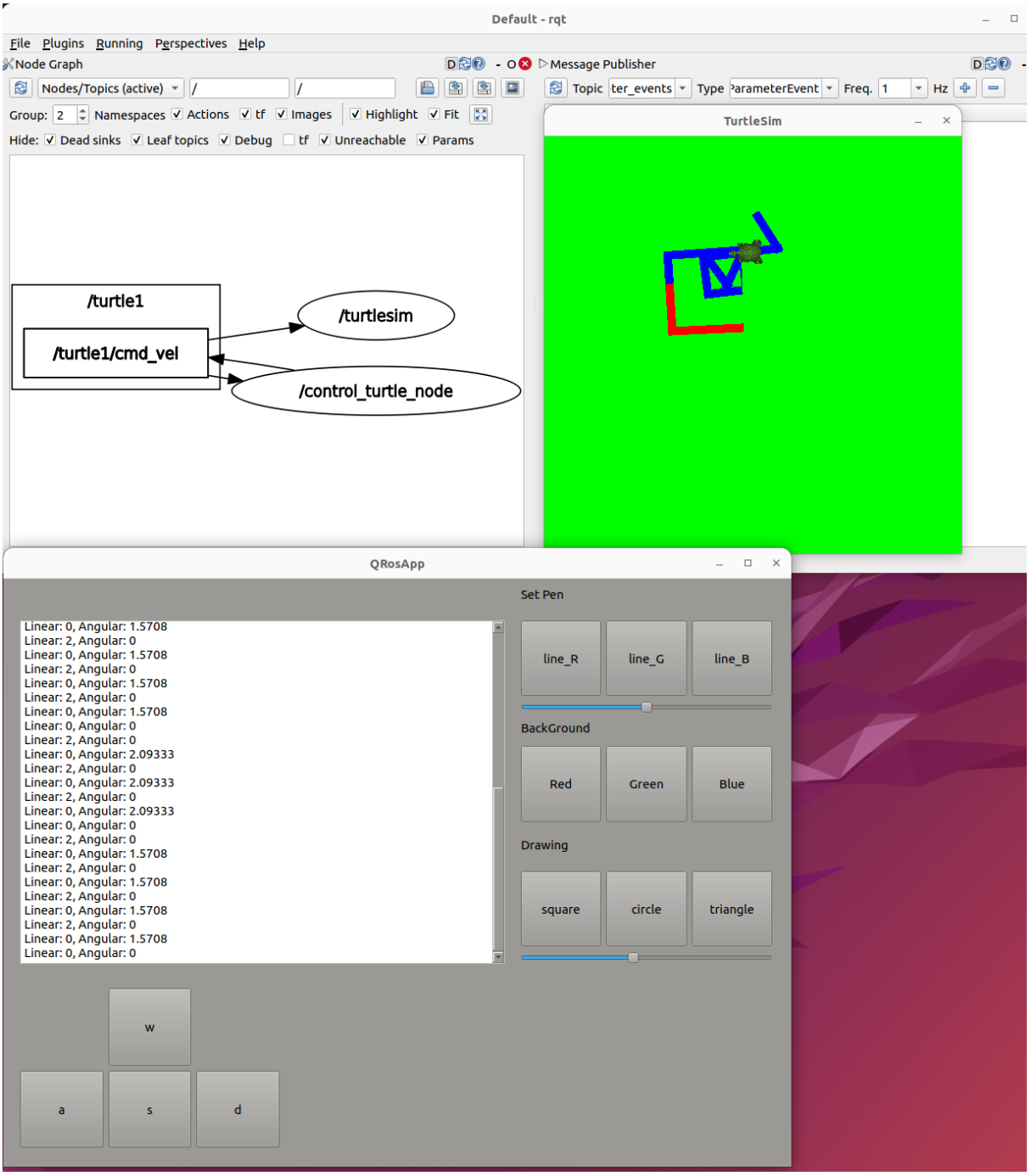
```
connect(qnode, &QNode::twistReceived, this, &MainWindow::upda
```

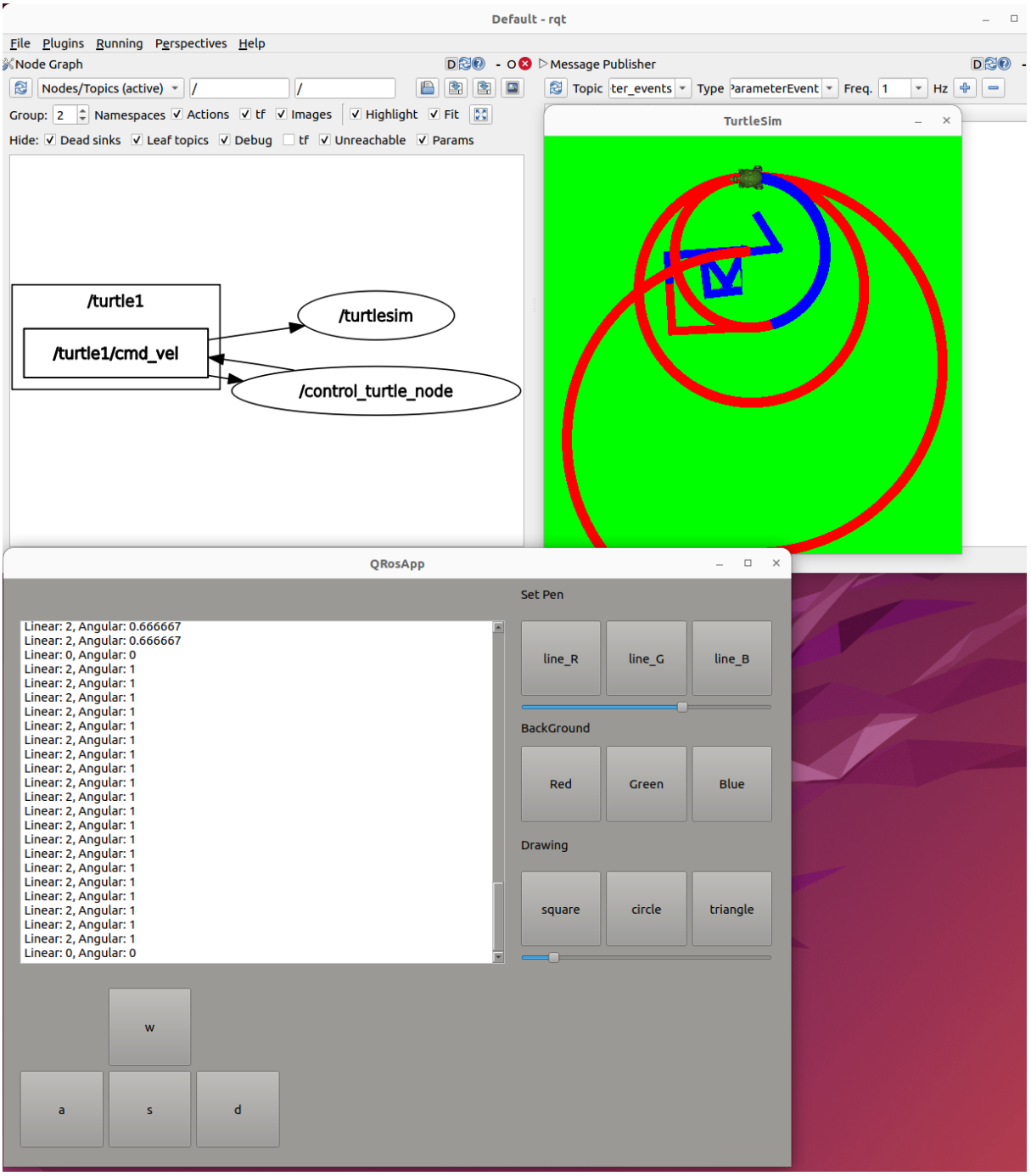
연결된 슬롯 ⇒ ubdateTextBrowser가 cmd_vel 값 출력


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Freq.
Hz
Group:
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtle1
/turtle1/cmd_vel
/turtlesim
/control_turtle_node
TurtleSim
QRosApp
Linear: 1, Angular: 0
Linear: 0, Angular: 1
Linear: 1, Angular: 0
Linear: 0, Angular: -1
Linear: -1, Angular: 0
Linear: 0, Angular: 1
Linear: 1, Angular: 0
Linear: 0, Angular: -1
Linear: -1, Angular: 0
Linear: 0, Angular: 1
Linear: 1, Angular: 0
Linear: 0, Angular: -1
Linear: -1, Angular: 0
Linear: 0, Angular: 1
Linear: 1, Angular: 0
Set Pen
line_R
line_G
line_B
BackGround
Red
Green
Blue
Drawing
square
circle
triangle
w
a
s
d



- 네모를 그리는 와중에 pen 색을 바꾸면 바뀌는 거 확인


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Freq.
Hz
Group:
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtle1
/turtle1/cmd_vel
/turtlesim
/control_turtle_node
TurtleSim
QRosApp
Linear: 0, Angular: 1.5708
Linear: 2, Angular: 0
Linear: 0, Angular: 2.09333
Set Pen
line_R
line_G
line_B
BackGround
Red
Green
Blue
Drawing
square
circle
triangle
w
a
s
d


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Freq.
Hz
Group:
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtle1
/turtle1/cmd_vel
/turtlesim
/control_turtle_node
TurtleSim
QRosApp
Linear: 2, Angular: 0.666667
Linear: 2, Angular: 0.666667
Linear: 0, Angular: 0
Linear: 2, Angular: 1
Linear: 0, Angular: 0
Set Pen
line_R
line_G
line_B
BackGround
Red
Green
Blue
Drawing
square
circle
triangle
w
a
s
d

day2-2

- talker.cpp

```
TalkerNode::TalkerNode(QObject* parent) : QObject(parent) {
    node_ = rclcpp::Node::make_shared("talker_node");
    publisher_ = node_->create_publisher<std_msgs::msg::Strin
}

TalkerNode::~TalkerNode() {
}

void TalkerNode::publishMessage(const QString& message) {
    auto msg = std::make_unique<std_msgs::msg::String>();
    msg->data = message.toStdString();
    publisher_->publish(std::move(msg));
    emit messagePublished(message);
}
```

노드 초기화, 퍼블리셔 설정,

publishMessage()함수 : QString을 입력으로 받아 이를 std::string으로 변환 후 ros2메세지로 발행, 새로운 메시지에 대해 gui에 알리기 위해 messagePublished가 발생

- listener.cpp

```
ListenerNode::ListenerNode(QObject* parent) : QObject(parent)
    node_ = rclcpp::Node::make_shared("listener_node");
    subscriber_ = node_->create_subscription<std_msgs::msg::S
        [this](const std_msgs::msg::String::SharedPtr msg) {
            onMessageReceived(msg);
        });
        spin_thread_ = std::thread([this]() {
        rclcpp::spin(node_);
    });
}
```

```
ListenerNode::~ListenerNode() {

}

void ListenerNode::onMessageReceived(const std_msgs::msg::Str
    RCLCPP_INFO(node_->get_logger(), "Received message: %s", 
    emit messageReceived(QString::fromStdString(msg->data));
}
```

노드 초기화, sub 설정

별도의 스레드에서 spin실행

: ROS2 노드가 메시지를 계속 수신하면서 Qt의 이벤트 루프가 중단 없이 작동하도록 하기 위해 별도의 스레드(spin_thread_)에서 rclcpp::spin이 실행. 이를 통해 노드는 Qt UI 스레드를 차단하지 않고 활성 상태를 유지하여 원활한 GUI 상호작용을 가능

메시지 처리 - onMessageReceived
: onMessageReceived() 함수는 수신된 메시지를 처리하고, Qt 신호(messageReceived)를 발생시킵니다. 이 신호는 GUI가 수신된 메시지를 실시간으로 표시할 수 있게 합니다.

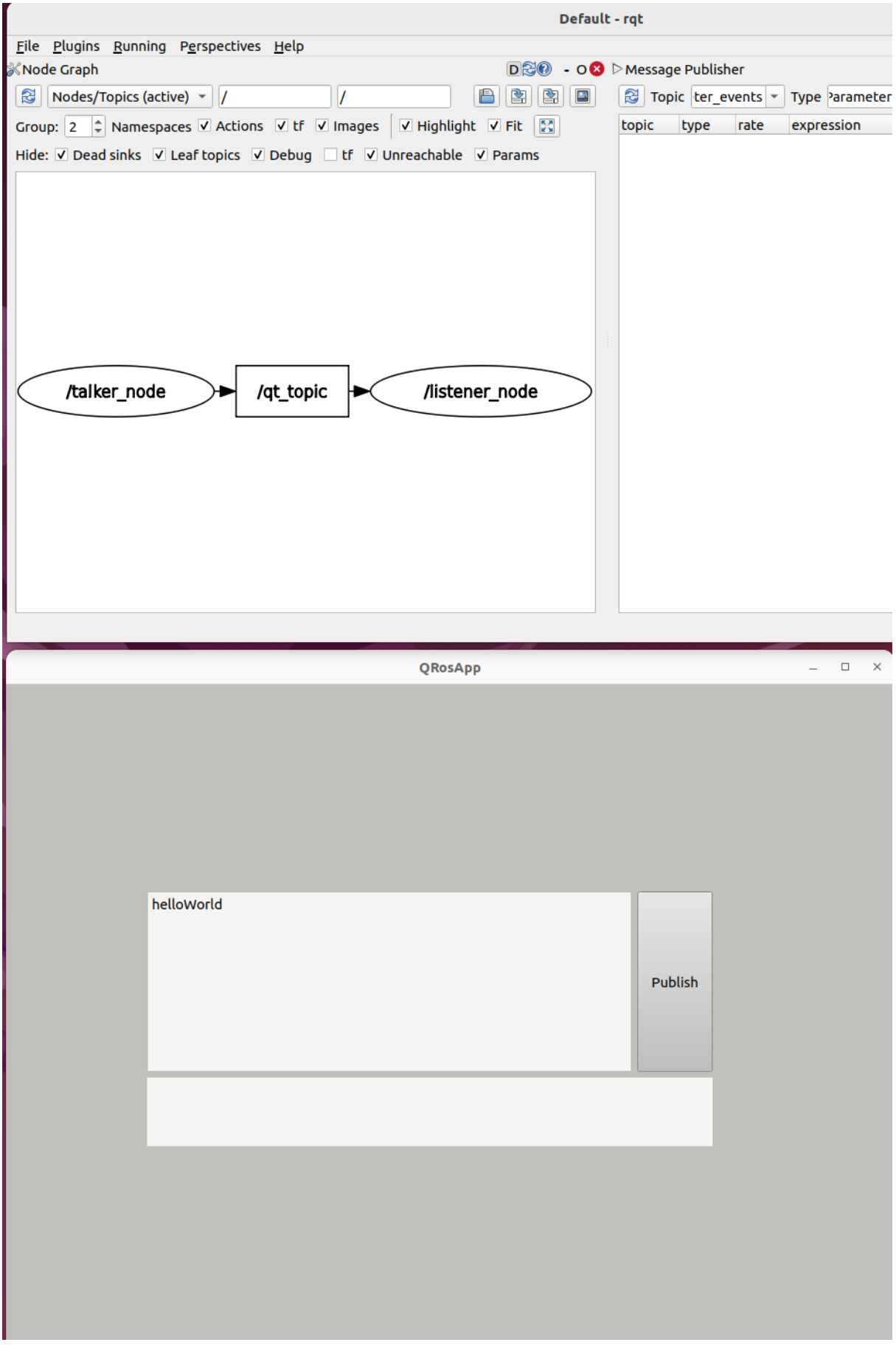
main_window

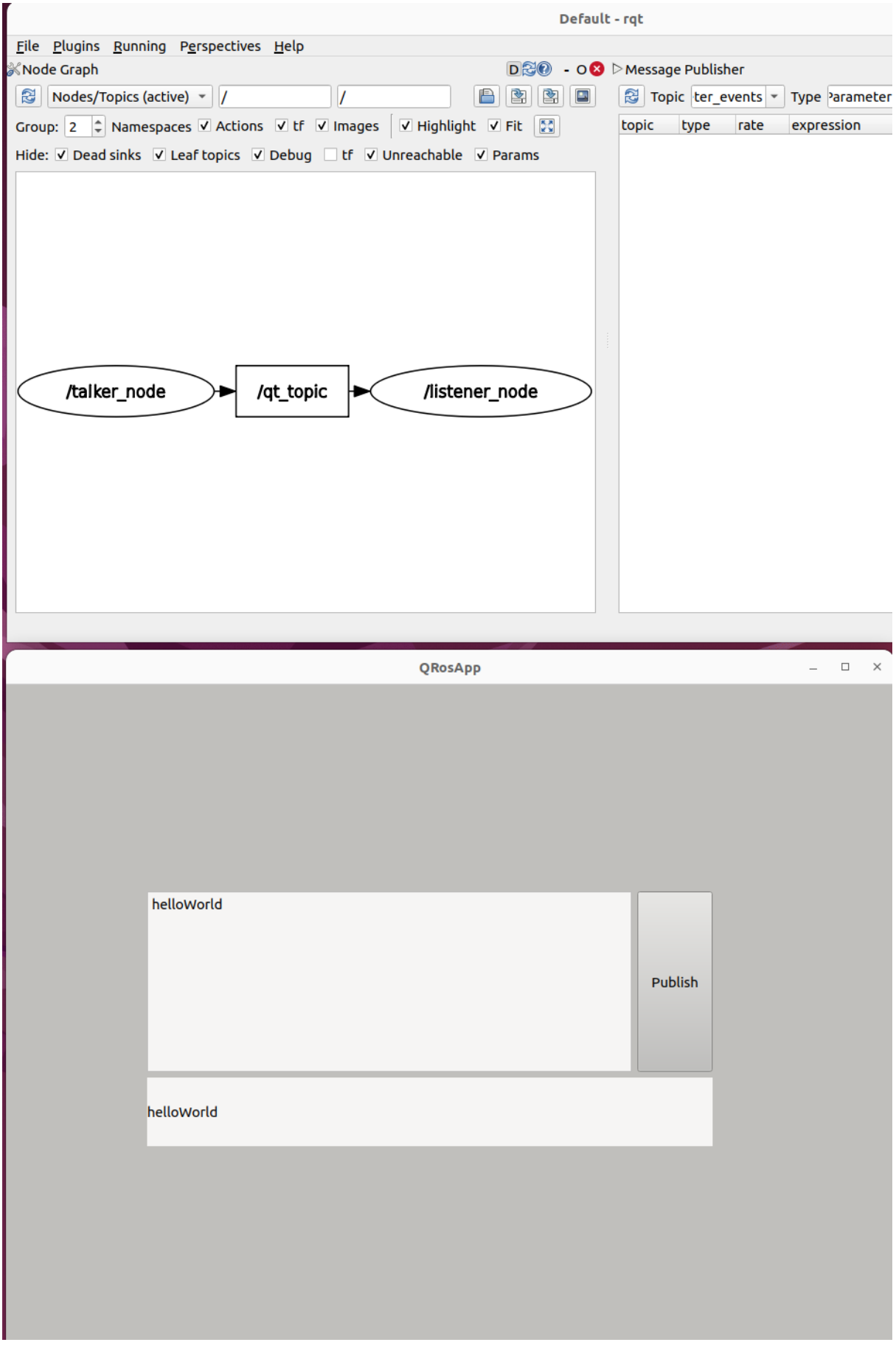
```
   TalkerNode* talker_node_;
   ListenerNode* listener_node_;

   connect(ui->pub_btn, &QPushButton::clicked, this, &MainWind
   connect(listener_node_, &ListenerNode::messageReceived, ui-
```

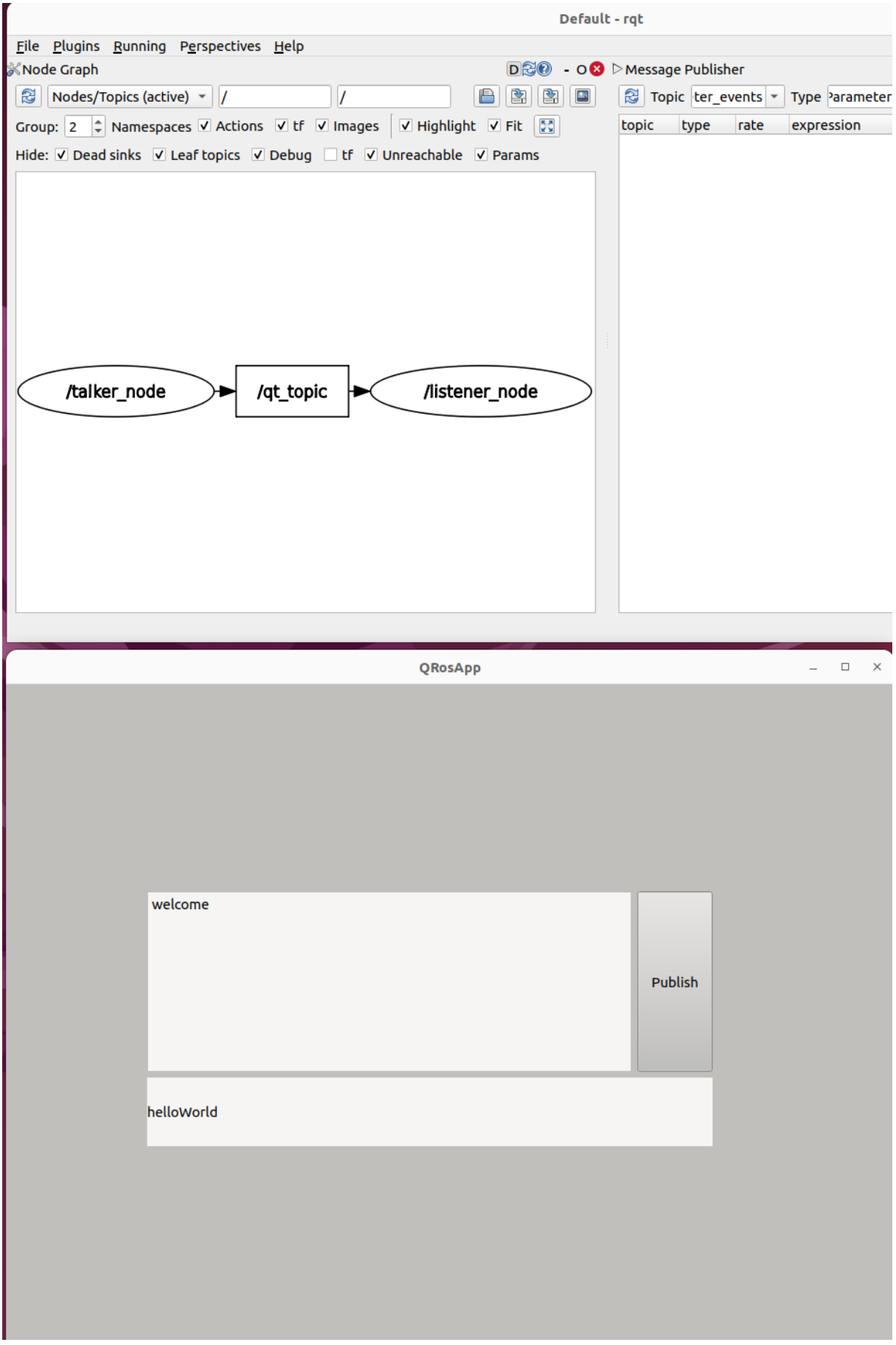
각각의 두 포인터는 노드를 각각 관리하게 함, 이를 통해 gui메시지 pub,sub

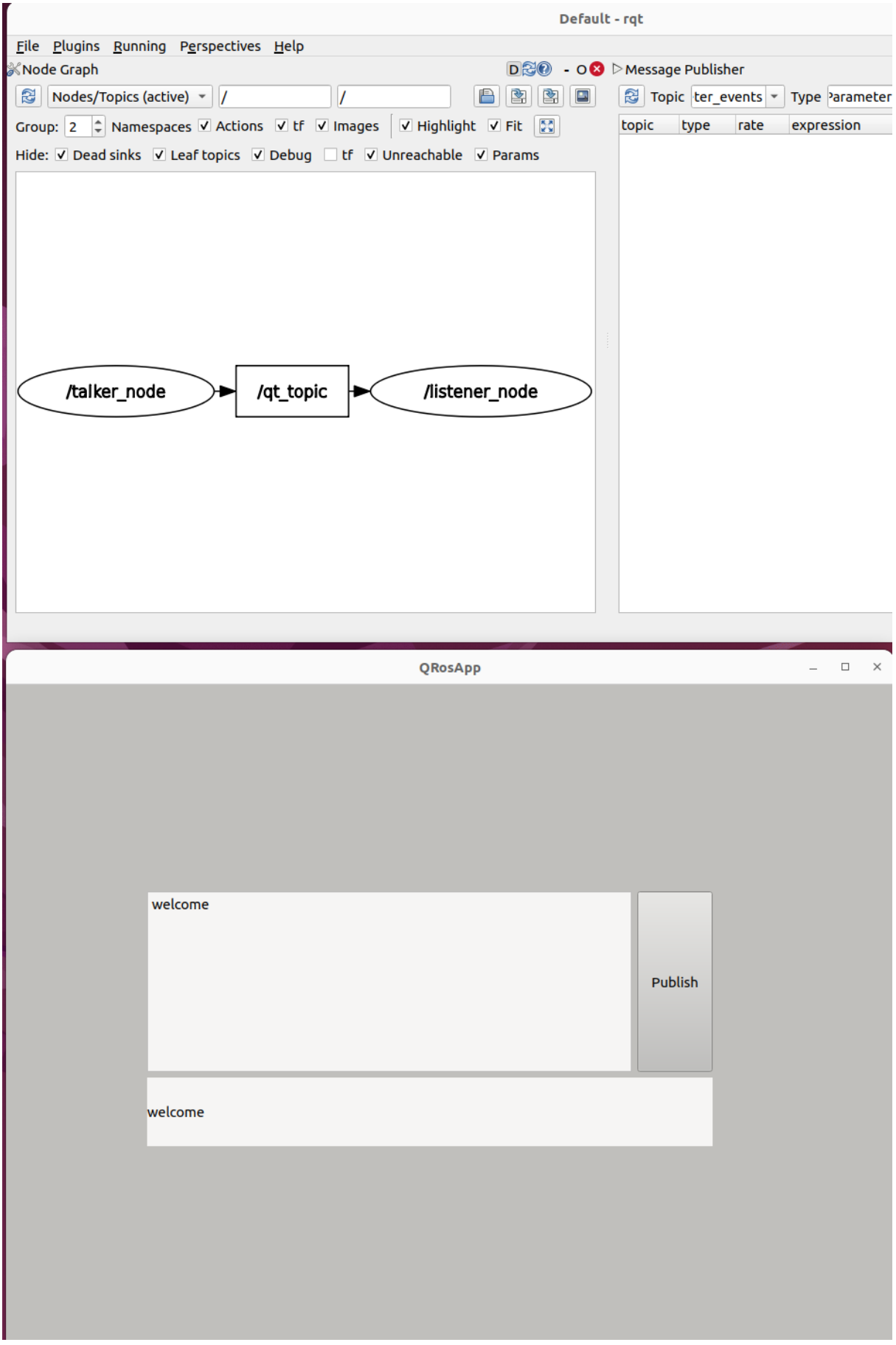
connect 함수를 통해 버튼 클릭시 publish

connect 함수를 통해 메시지 수신시 messageReceived 신호 발생, label에 텍스트 없데이트


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Group: 2 Namespaces Actions tf Images Highlight Fit
topic type rate expression
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
/talker_node
/qt_topic
/listener_node
QRosApp
helloWorld
Publish


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Group: 2 Namespaces Actions tf Images Highlight Fit
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
topic type rate expression
/talker_node
/qt_topic
/listener_node
QRosApp
helloWorld
Publish
helloWorld


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Group: 2 Namespaces Actions tf Images Highlight Fit
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
topic type rate expression
/talker_node
/qt_topic
/listener_node
QRosApp
welcome
Publish
helloWorld


Default - rqt
File Plugins Running Perspectives Help
Node Graph
Message Publisher
Nodes/Topics (active)
Topic
Type
Group: 2 Namespaces Actions tf Images Highlight Fit
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
topic type rate expression
/talker_node
/qt_topic
/listener_node
QRosApp
welcome
Publish
welcome