ROS DAY2
과제 4번 보고서

2023741024
로봇학부 박건후

목차

(1) qnod.hpp파일 설명

```
#ifndef Q_MOC_RUN
#include <rclcpp/rclcpp.hpp>
#include "custom_interfaces/msg/robot_arm.hpp" //custom_interfaces 활용 멤버: int32 flag, int32 angle
#include <memory>
#endif
#include<QObject> //subscribe_node에서 QObject 상속받기 위해
#include <QThread>
```

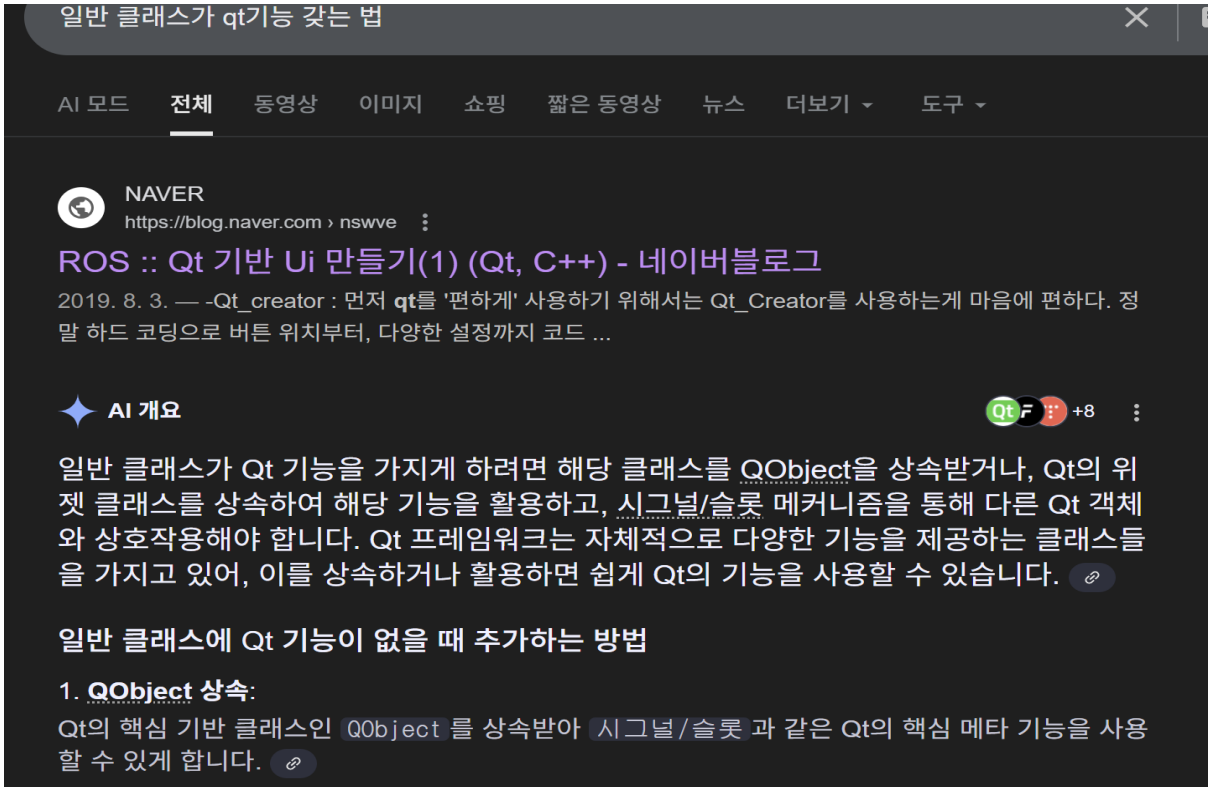
qnode에서 인클루드한 부분입니다. 기본적인 인클루드하는 것 외에 추가된 것은 custom_interfaces 관련 선언부입니다. 이번 과제에서는 publish를 할 때 보내야 할 정보를 생각했었을 때 어떤 축에 대해 어떤 각도로 돌릴 지에 대한 정보를 보내야 한다고 생각했습니다. 그래서 2개의 int 값을 보내야겠다는 생각이 들었고, int값을 2개를 따로 보내는 것보다는 한 번에 보내는 것이 헷갈릴 여지없이 깔끔하게 보낼 수 있다는 생각이 들었습니다. 그래서 새로운 RobotArm.msg라는 파일을 만들어서 int32 flag, int32 angle 이라는 멤버를 기입하였습니다. 그리고 이를 사용하기 위해 인클루드한 것입니다. 그리고 QObject를 인클루드했는데 이는 subscribe_node에서 QObject를 상속시키기 위해 인클루드하였습니다.

```
class publisher: public rclcpp::Node{ //publisher 클래스 선언

    rclcpp::Publisher<custom_interfaces::msg::RobotArm>::SharedPtr pub;


    public:

    publisher();
    void publish_message(); //값을 입력받아 publish하는 메서드
};
class subscribe_node: public QObject, public rclcpp::Node{
  Q_OBJECT //이거 써야 함
  rclcpp::Subscription<custom_interfaces::msg::RobotArm>::SharedPtr sub;
  void topic_callback(const custom_interfaces::msg::RobotArm::SharedPtr msg); //메시지 받았을 때의
  signals:
  void signal_func(int flag, int angle); //signals 설정하기 위해 QObject상속받음


  public:
  subscribe_node();
};
```

위 사진은 publisher랑 subscribe_node의 선언부를 나타낸 것입니다. publisher는 msg파일에서 정의한 타입을 publish 해야 하므로 custom_interfaces::msg::RobotArm으로 인스턴스화 되도록 하였고, 메시지를 보낼 publish_message()메서드를 선언했습니다.

```
class subscribe_node: public QObject, public rclcpp::Node{
  Q_OBJECT //이거 써야 함
  rclcpp::Subscription<custom_interfaces::msg::RobotArm>::SharedPtr sub;
  void topic_callback(const custom_interfaces::msg::RobotArm::SharedPtr msg); //메시지 받았을
  signals:
  void signal_func(int flag, int angle); //signals 설정하기 위해 QObject상속받음


  public:
  subscribe_node();
};
```

그리고 이 subscribe_node를 만들게 될 때 고민했던 것이 있습니다. 이전 과제 2번처럼 ros 부문의 데이터를 어떻게 qt부문의 main_window쪽으로 보낼 것인가에 대한 고민입니다. 우선 과제 4번에서는 특정 값을 퍼블리시하여 로봇팔을 조종해야 하는데 그렇게 되면 퍼블리시 노드가 서브스크라이버 노드에게 정보를 보내고 정보를 받은 서브스크라이버는 그 정보를 통해 ui의 팔 위치를 바꾸고 그림을 그려야 합니다. 지금까지 퍼블리시 노드가 서브스크라이버 노드에게 정보를 보내고 서브스크라이버 노드가 정보를 받는 것까지는 많이 해봤기에 익숙하지만 서브스크라이버 노드가 받은 정보를 어떻게 MainWindow클래스에 넘길지를 많이 고민했습니다. 이전처럼 friend를 사용해서 구현할 수도 있을 것 같지만 캡슐화나 은닉성인 c++구조가 깨질 것 같고, 그렇다고 main_window파일에 subscriber를 만들면 qt-qnode-ros간의 구조가 깨질 것 같았습니다. 그러다가 connect, emit을 통해 시그널이 생기면 어느 파일이든 바로 실행 부분이 옮겨지는 것처럼 goto문과 같은 역할을 해줄 수 있다는 생각이 들었고 추가적으로 connect에서는 매개변수가 signal 메서드와 slot메서드가 일치할 시 자동으로 매개변수를 대응시켜 전달해주므로 서브스크라이버가 메시지를 받아 콜백함수를 호출하면 그 내부에서 정의한 시그널 메서드를 억지로 발생시켜서 connect로 연결된 MainWindow의 slot메서드에게 매개변수를 통한 정보 전달을 하면 되지 않을까라는 생각을 했습니다. 그러려면 subscriber_node에서 시그널을 발생시킬 메서드가 필요했고 시그널을 강제로 발생시킬 emit이라는 키워드도 클래스 내에서 쓸 수 있어야 했습니다. 그래서 이런 기능을 사용자 정의 클래스에 넣기 위해 필요한 작업들을 알아야 했습니다. 추가적으로 필요한 작업이 어떤 것들이 있는지 알기 위해 인터넷 서칭을 했을 때 아래 사진의 구글의 AI개요에서 필요한 작업들을 찾을 수 있었습니다.



예시 (개념적)

C++

```cpp
#include <QObject> // QObject를 사용하기 위해 헤더 포함

// QObject를 상속받는 MyClass
class MyClass : public QObject
{
    Q_OBJECT // QObject 메타 객체 기능을 사용하기 위한 매크로

public:
    explicit MyClass(QObject *parent = nullptr);
    void doSomethingQtLike();

signals:
    // 여기에 시그널을 정의하여 다른 Qt 객체에 알릴 수 있습니다.
    void someSignal();

public slots:
    // 여기에 슬롯을 정의하여 시그널을 받아 처리할 수 있습니다.
    void someSlot();
};
```

QObject를 인클루드하고, 상속 받고, Q_OBJECT라는 매크로를 클래스 내부에 작성하면 사용자 정의 클래스에서도 slots이나 signals, emit, connect등의 qt의 제일 상위 부모 클래스의 QObject 기능은 모두 사용할 수 있었습니다.

```
class subscribe_node: public QObject, public rclcpp::Node{
  Q_OBJECT //이거 써야 함
  rclcpp::Subscription<custom_interfaces::msg::RobotArm>::SharedPtr sub;
  void topic_callback(const custom_interfaces::msg::RobotArm::SharedPtr msg); //메시지 받았을
  signals:
  void signal_func(int flag, int angle); //signals 설정하기 위해 QObject상속받음


  public:
  subscribe_node();
};
```

이렇게 알게 된 점을 통해 QObject를 인클루드하여 subscribe_node에 상속시켰고, 서브스크라이버 클래스이므로 rclcpp::Node 또한 상속시켰습니다. 결국 하다보니 다중 상속이 되었지만 QObject와 rclcpp::Node는 완전히 다른 클래스이므로 다중 상속했을 때의 문제점인 멤버 충돌 같은 문제가 생길 경우는 적을 것 같아서 이대로 진행했습니다.
Q_OBJECT매크로를 써주고, 안에 signals: 키워드로 메서드를 정해줬습니다. 전달해야 할 정보가 메시지에서 전달받은 int 2개의 정보이므로 int flag, int angle로 매개변수를 설정하였습니다. 그리고 topic_callback이라는 메서드는 메시지를 받았을 때 호출되는 콜백함수입니다.

```
class QNode : public QThread
{
  Q_OBJECT
public:
  QNode();
  ~QNode();
  //std::shared_ptr<publisher> pub_ptr;
  std::shared_ptr<subscribe_node> sub_ptr; // main_
protected:
  void run();

private:


Q_SIGNALS:
  void rosShutDown();
};

#endif /* hw4_pkg_QNODE_HPP_ */
```

subscribe_node 선언부 아래의 QNode선언부입니다. 여기서는 std::shared_ptr로 public에 subscribe_node 스마트 포인터를 선언한 것입니다. 이를 통해 main_window에서 qnode라는 포인터를 통해 subscribe_node에 접근하여 작업을 할 수 있기에 public으로 선언하였고 이는 즉, QNode클래스가 다리 역할을 해주고 있는 것입니다.

(2) qnode.cpp 파일 설명

```cpp
QNode::QNode()
{
  int argc = 0;
  char** argv = NULL;
  rclcpp::init(argc, argv);
  sub_ptr = std::make_shared<subscribe_node>();
  this->start();
}
```

```cpp
void QNode::run()
{
  rclcpp::WallRate loop_rate(20);
  while (rclcpp::ok())
  {
    rclcpp::spin(sub_ptr); //처음에는 sub_ptr을 c
    loop_rate.sleep();
  }
  rclcpp::shutdown();
  Q_EMIT rosShutDown();
}
```

QNode 클래스의 구현부입니다. sub_ptr을 make_shared로 초기화하고, subscribe_node 객체를 생성한 부분입니다, 그리고 run()에서는 spin()에 sub_ptr을 넣어서 계속 메시지를 받도록 대기하게 하였습니다.

```
publisher::publisher():Node("publish_node"){ // publisher 클래스 생성자, 노드 이름 설정
    pub = this->create_publisher<custom_interfaces::msg::RobotArm>("hw4_pkg",10); //토
}

void publisher::publish_message(){
    int part=0;
    int angle=0;
    while(1){
     RCLCPP_INFO(this->get_logger(),"움직일 부분(하단: 1, 중간부: 2, 상단: 3), 움직일 각도");
     std::cin>>part; //터미널로부터 값 입력 받기
     //예외처리
     if(!std::cin|| part<0 || part>4){ //1,2,3 정수만 받음
         RCLCPP_INFO(this->get_logger(),"잘못된 입력입니다. 처음부터 다시 입력하세요");
         std::cin.clear();
         std::cin.ignore(1000,'\n');
         continue;
     }
     std::cin>>angle;
     if(!std::cin|| angle<0 || angle>360){ //각도는 0~360
         RCLCPP_INFO(this->get_logger(),"잘못된 입력입니다. 처음부터 다시 입력하세요");
         std::cin.clear();
         std::cin.ignore(1000,'\n');
         continue;
     }
      custom_interfaces::msg::RobotArm temp; //temp객체 생성해서 아래에서 멤버 모두 값 초기화하
      temp.flag=part;
      temp.angle = angle;
      pub->publish(temp);
      // std::this_thread::sleep_for(std::chrono::seconds(3));
   }
}
```

publisher 클래스 구현부입니다. 생성자에서 지금까지 했던 방식대로 생성을 해주었고, publish_message에서는 터미널에 어떤 값을 입력해야 하는지 알려주고 int 변수들로 각각 입력받는 모습입니다. 그리고 입력받을 때마다 바로 아래에 if문으로 예외처리를 해주었습니다. 그 후에는 입력받은 값을 msg파일로 만든 타입의 temp객체에 넣어서 publish 해주었습니다.

subscribe_node의 구현부입니다. 생성자에서는 노드명을 넣어주고, 토픽명을 일치시켜주는 작업을 했습니다. 그리고 메시지를 받았을 때의 호출되는 콜백함수에서는 매개변수로 전달받은 객체를 가리키고 있는 상태인데 그 객체의 flag, angle 멤버에 접근하여 정보를 알아내고, signal_func() 시그널 함수를 강제로 emit키워드로 호출시키는 부분입니다. 그렇게 되면 main_window에서 connect되어 있던 slots함수에게 매개변수 값과 함께 실행되는 것이 넘어가게 되어 MainWindow클래스 쪽으로 정보를 넘길 수 있었습니다.

(3) main_window.hpp/cpp파일 설명

```
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  MainWindow(QWidget* parent = nullptr);
  ~MainWindow();
  QNode* qnode;

private:
  Ui::MainWindow* ui;
  ArmView* armView;  //armview 포인터 생성
  void closeEvent(QCloseEvent* event);
private slots:

  void control_arm(int flag, int angle); //flag, angle
  void on_spinBox_valueChanged(int v);
  void on_spinBox_2_valueChanged(int v);
  void on_spinBox_3_valueChanged(int v);
};
```

MainWindow 클래스 선언부입니다. 여기서는 팔을 돌리는 arm.h파일을 인클루드하여 arm.h파일에 정의한 ArmView 클래스의 포인터를 선언하였습니다. 이는 스핀박스의 값이 바뀔 때만 사용하였습니다. 그리고 그 아래에 슬롯함수를 선언하였습니다. control_arm이 signal_func가 emit되면 호출되는 slots함수입니다. 그래서 매개변수도 int flag, int angle 로 작성한 것입니다. 그 아래에는 이전 qt과제에서 했던 spinbox값이 바뀌었을 경우에 대한 슬롯메서드들을 가져온 것입니다.

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


  ui->spinBox->setRange(0, 360);              //최대 두자리 수여서 360까지 값을 받을 수 있도록 설
  ui->spinBox_2->setRange(0, 360);
  ui->spinBox_3->setRange(0, 360);
  armView = new ArmView(this);                //동적할당하여 객체 생성하여 이 객체로 팔 조절
    auto *layout = new QVBoxLayout(ui->armViewHost); //로봇팔을 위젯 위에 그리기 위한 설정
    layout->setContentsMargins(0,0,0,0);
    layout->addWidget(armView);
  qnode = new QNode();
  connect(qnode->sub_ptr.get(), &subscribe_node::signal_func, this, &MainWindow::contro
  QObject::connect(qnode, SIGNAL(rosShutDown()), this, SLOT(close()));
}
```

이제 MainWindow의 구현부입니다. 위 사진은 생성자 부분으로써, setupUi를 해서 기본적인 ui 초기화와 자동으로 connect설정되어 있는 것들이 내부적으로 connect가 활성화되는 것입니다. 그리고 ui->spinBox의 메서드로 0~360으로 스핀박스의 값의 범위를 제한하였습니다. 그 후에 ArmView클래스의 객체를 생성하고 이를 위젯에 올리는 등의 qt에서 했던 작업을 그대로 하였습니다.

```
connect(qnode->sub_ptr.get(), &subscribe_node::signal_func, this, &MainWindow::control_arm);
```

위 사진이 signal_func와 control_arm을 연결한 부분입니다. subscribe_node객체의 포인터가 필요하므로 main_window에서 이를 접근하기 위해 qnode->sub_ptr.get()으로 스마트 포인터의 get메서드를 써서 객체의 실제 포인터를 반환받아 작성할 수 있었습니다.

```
void MainWindow::on_spinBox_2_valueChanged(int arg1)
{
    armView->setMiddleAngle(arg1);
}


void MainWindow::on_spinBox_valueChanged(int arg1)
{
    armView->setBottomAngle(arg1);
}


void MainWindow::on_spinBox_3_valueChanged(int arg1)
{
    armView->setUpperAngle(arg1);
}
```

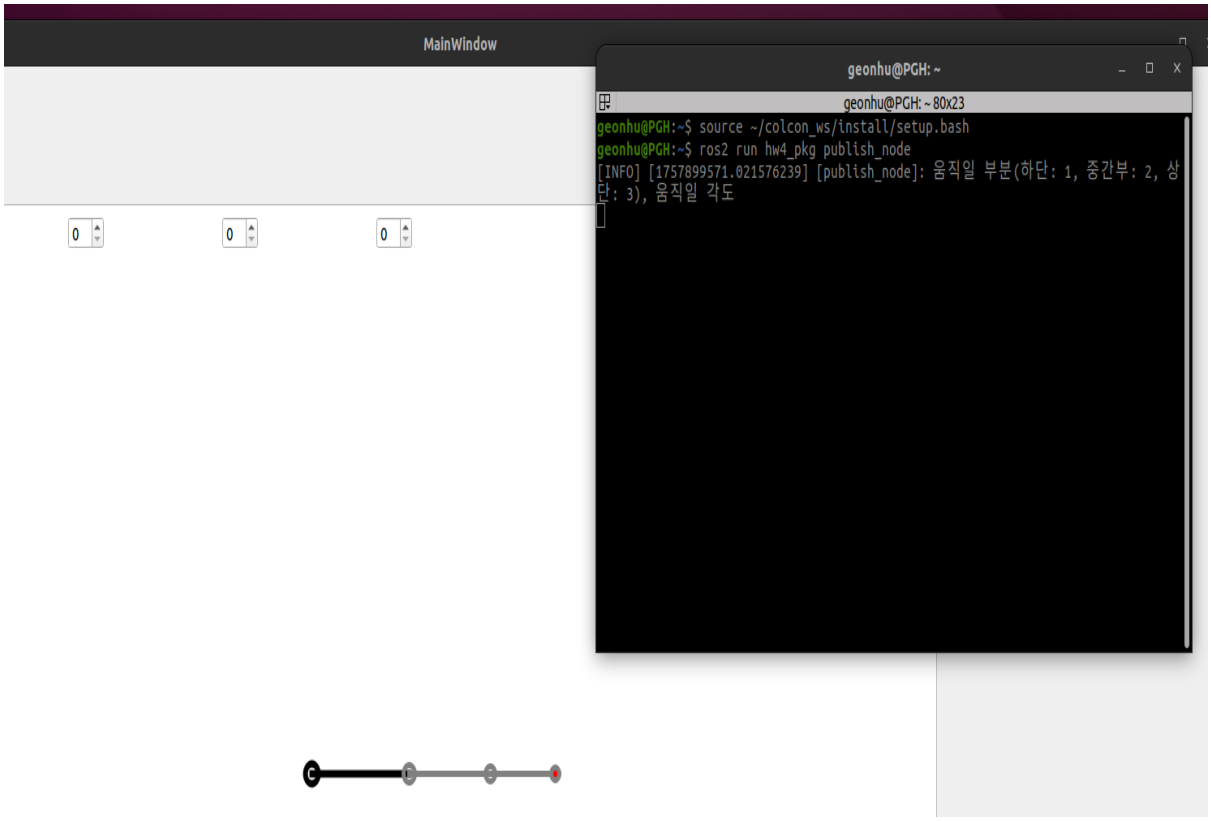
그 이후에는 기존의 스핀박스에 대한 슬롯 메서드를 작성하였습니다. 스핀박스의 값이 바뀌게 되면 바뀐 해당 스핀박스의 슬롯 메서드가 호출되어 arm.h의 메서드가 호출되고 update()하여 정의한 paintEvent메서드가 호출되어 그림을 그리게 되는 것입니다. 만약 1번 스핀박스 슬롯 메서드가 호출되면 3축 중에 제일 아랫부분 관절이 돌아가게 되고 2번 스핀박스 슬롯 메서드가 호출되면 중간 부분의 관절이 돌아가게 되고 3번 스핀박스 슬롯메서드가 호출되면 상단 관절이 돌아가게 되는 것입니다.

```
//setValue로 위 slot 메서드들을 호출하는 시그널 발생시킴
void MainWindow::control_arm(int flag, int angle){
  if(flag==1){
    ui->spinBox->setValue(angle);
  }
  else if(flag==2){
    ui->spinBox_2->setValue(angle);
  }
  else if(flag==3){
    ui->spinBox_3->setValue(angle);
  }
}
```

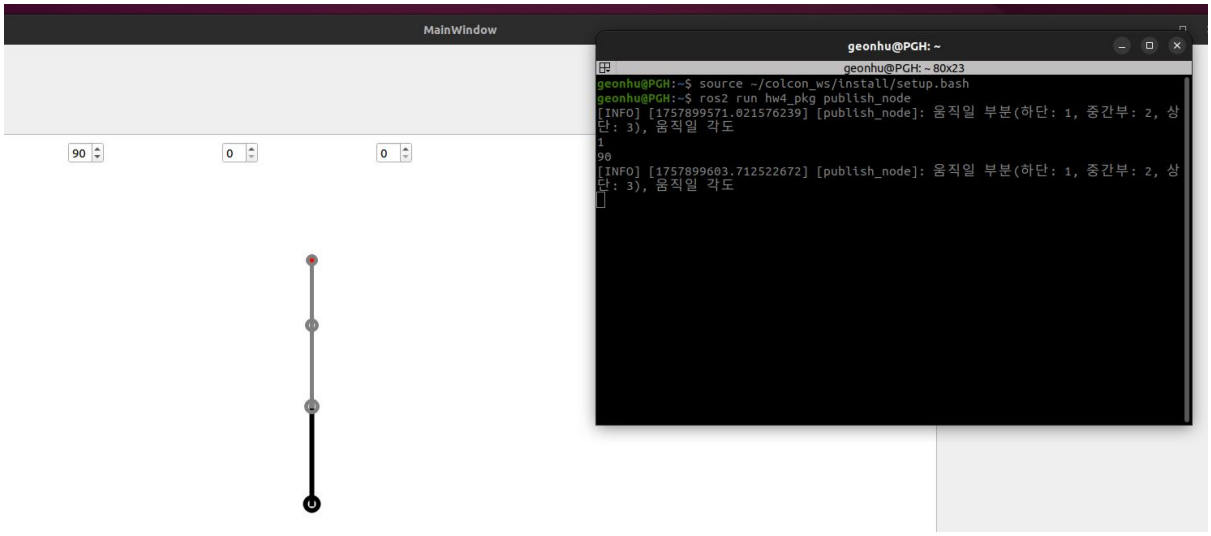
이제 control_arm 부분입니다. control_arm 슬롯 메서드가 호출되었다면 터미널에서 1번은 하단, 2번은 중간부, 3번은 상단이라고 했었으므로 그에 맞게 전달받은 매개변수값을 통해 flag값에 따라 setValue를 통해 스핀박스값이 바뀌는 시그널을 발생시켜서 해당 스핀박스의 슬롯함수가 호출되어 해당 관절을 돌아가게 한 것으로 굉장히 간단하게 구성하였습니다. 기존에 있던 것을 활용하며 추가적인 인터페이스 구성을 하니 재미를 느낄 수 있었습니다.

```
#include "../include/hw4_pkg/qnode.hpp"
int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<publisher>(); //퍼블리셔 노드 생성
    node->publish_message(); //spin만 하면 publish_message가 호출될
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

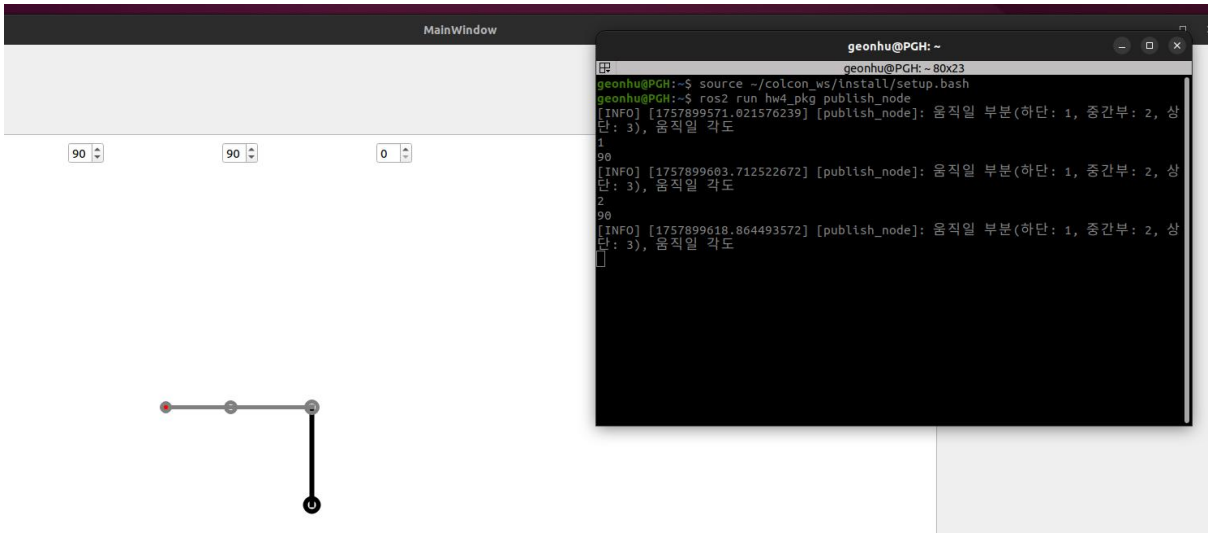
마지막으로 publish하는 노드에서 작동하는 main부를 보겠습니다. 이전처럼 spin만을 호출하게 되면 publish_message가 호출될 수가 없기 때문에 spin함수 위에 publish_message()를 호출하였습니다. 감사합니다.



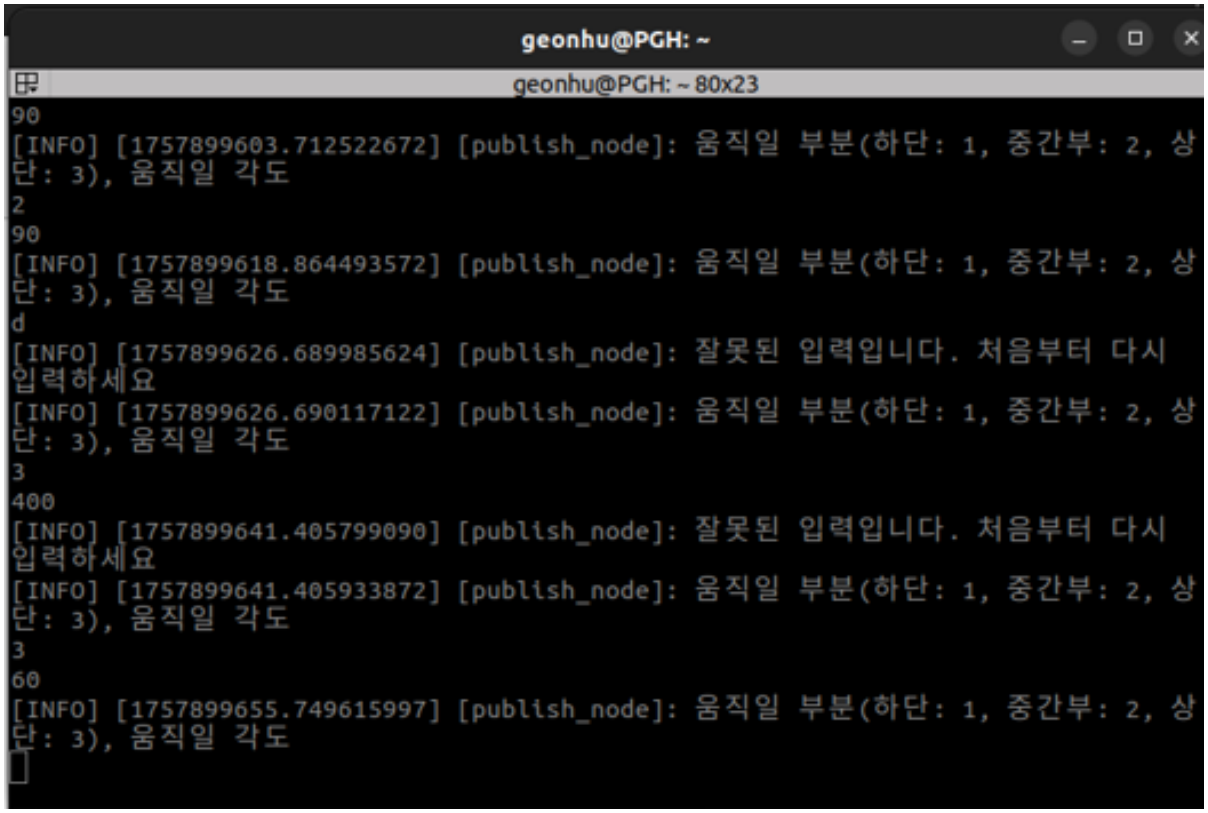
현재 2개의 노드를 실행하여 하나는 publisher, 하나는 gui가 열린 것입니다. 위의 사진은 열고 나서의 초기상황입니다.



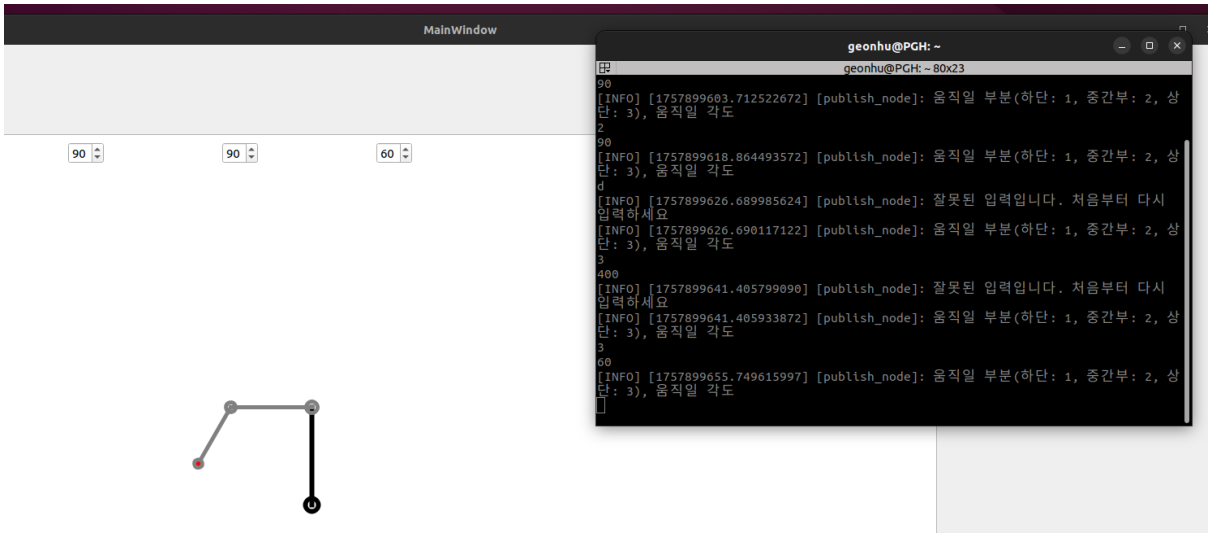
위 사진은 1을 입력하고 90을 입력하여 스핀박스에 90이 찍히고 팔이 돌아간 모습입니다.



그리고 2를 입력하고 90을 입력하여 중간부분 관절이 90도 돌아간 것입니다.



그리고 c같은 문자를 입력했을 때 터미널에서 예외처리가 된 것을 볼 수 있고 400도를 입력할 때도 그에 대한 예외처리를 한 것을 보실 수 있습니다.



3을 입력하고 60을 입력하여 상단 관절이 60도 돌아가게 하였습니다.