ROS2_day2 hw2 ROS2 && Qt
과제 보고서

로빛 20기 인턴 현창석

목차

- 코드 설명

```
#ifndef HW2_QNODE_HPP_
#define HW2_QNODE_HPP_

#include <rclcpp/rclcpp.hpp>
#include "std_msgs/msg/string.hpp"
#include "geometry_msgs/msg/twist.hpp"
#include "turtlesim/srv/set_pen.hpp"
#include <QThread>
#include <string>
#include <QObject>
#include <chrono>

class QNode : public QObject, public rclcpp::Node
{
  Q_OBJECT
public:
  explicit QNode(const std::string& name);
  void publishString(const std::string &text);
  void set_pen(int r, int g, int b, int width);
  void drawSquare();
  void drawtriangle();
  void drawcircle();
  void move_forward();
  void move_back();
  void turn_left();
  void turn_right();

private:

  rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr TurtlePublisher;
  rclcpp::Client<turtlesim::srv::SetPen>::SharedPtr client_;
  geometry_msgs::msg::Twist twist;


Q_SIGNALS:
  void rosShutDown();
  void newMessageReceived(QString msg);
  void log(const QString& message);
};

#endif // HW2_QNODE_HPP_
```

먼저 생성자를 정의하여 노드 객체와 위젯을 전달할 수 있게 하였다.
ui 를 초기화한 후 Qtextedit의 포인터를 변수로 선언한다.
UI 안에 로그를 출력하기 위해서 QObject::connect(qnode_.get(), &QNode::log, this, &MainWindow::updateLog); 를 설정하여 QNode의 log 를 update:og와 연결되도록 하였다.
ui->centralwidget->setFocus(); 기보드 입력을 잘 받기 위해서 키보드 포커스를 설정하였다. 그 아래 함수 3개는 ui에서 각 버튼이 눌렸을 때 터틀이 사각혀, 삼각형, 원 그림을 그리는 함수를 실행시켜 도형을 그리도록 하였다. keyPressEvent에서는 switxh 문으로 각 입력 키에 해당하는 움직임 함수를 실행시켜 터틀이 이동하도록 하였다.

```
45
46  void MainWindow::keyPressEvent(QKeyEvent* event)
47  {
48      switch(event->key())
49      {
50      case Qt::Key_W:
51          qnode_->move_forward();
52          break;
53      case Qt::Key_S:
54          qnode_->move_back();
55          break;
56      case Qt::Key_A:
57          qnode_->turn_left();
58          break;
59      case Qt::Key_D:
60          qnode_->turn_right();
61          break;
62      default:
63          break;
64      }
65      QWidget::keyPressEvent(event);
66  }
```

생성자에선 "turtle1/cmd_vel"이라는 이름의 토픽으로 메세지를 퍼블리시할 퍼블리셔를 선언하고 setpen 서비스를 호출하기 위해서 클라이언트를 선언하였다. 서비스의 이름은 /turtle1/set_pen 이다. SetPen.Request()

SetPen 서비스 요청 메시지를 생성한 후 그 아래 set pen 함수로 rgb값으로 펜의 색깔과 width로 두께를 설정할 수 있게 하였다. drawSquare 함수는 터틀이 사각형을 그리도록 명령하는 함수이다 .반복문을 돌며 인자 값에 해당하는 좌표값을 퍼블리시 하는 과정을 통하여 계속 이동하며 터틀이 사각형 모양을 그린다. 각 좌표로 이동할 때 딜레이가 있지 않으면 직선 방향으로만 가기 때문에 딜레이를 넣어 주었다. 500ms간 딜레이 후 로그를 출력한다. citcle 함수, trianlge 함수 또한 마커니즘이 동일하다.

생성자에선 "turtle1/cmd_vel"이라는 이름의 토픽으로 메세지를 퍼블리시할 퍼블리셔를 선언하고 setpen 서비스를 호출하기 위해서 클라이언트를 선언하였다. 서비스의 이름은 /turtle1/set_pen 이다. SetPen.Request()
SetPen 서비스 요청 메시지를 생성한 후 그 아래 set pen 함수로 rgb값으로 펜의 색깔과 width로 두께를 설정할 수 있게 하였다. drawSquare 함수는 터틀이 사각형을 그리도록 명령하는 함수이다 .반복문을 돌며 인자 값에 해당하는 좌표값을 퍼를리시 하는 과정을 통하여 계속 이동하며 터틀이 사각형 모량을 그린다. 각 좌표로 이동할 때 딜레이가 있지 않으면 직선 방향으로만 가기 때문에 딜레이를 넣어 주었다. 500ms간 딜레이 후 로그를 출력한다. citcle 함수, trianlge 함수 또한 먀커니즘이 동일하다.

```cpp
#include "hw2/qnode.hpp"
#include <iostream>
#include <thread>

QNode::QNode(const std::string& name) : Node(name)
{
  TurtlePublisher = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel", 10);
  client_ = this->create_client<turtlesim::srv::SetPen>("/turtle1/set_pen");
}


void QNode::set_pen(int r, int g, int b, int width)
{
    auto request = std::make_shared<turtlesim::srv::SetPen::Request>();
    request->r = r;
    request->g = g;
    request->b = b;
    request->width = width;
    request->off = 0;

    client_->async_send_request(request);
}

void QNode::drawSquare()
{
    using namespace std::chrono_literals;
    auto twist = geometry_msgs::msg::Twist();

    set_pen(255, 0, 0, 7);
    for (int i = 0; i < 4; i++)
    {
        if (i == 0)
            twist.linear.x = 2.0;
        else if (i == 1)
        {
            twist.linear.x = 0.0;
            twist.linear.y = 2.0;
        }
        else if (i == 2)
        {
            twist.linear.x = -2.0;
            twist.linear.y = 0.0;
```

```
44            else if (i == 3)
45            {
46                twist.linear.x = 0.0;
47                twist.linear.y = -2.0;
48            }
49            TurtlePublisher->publish(twist);
50            std::this_thread::sleep_for(500ms);
51            QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
52                                  .arg(twist.linear.x)
53                                  .arg(twist.linear.y)
54                                  .arg(twist.angular.z);
55            emit log(log_msg);
56        }
57    }
58
59    void QNode::drawtriangle()
60    {
61        using namespace std::chrono_literals;
62        auto twist = geometry_msgs::msg::Twist();
63
64        set_pen(0, 255, 0, 10);
65        for (int i = 0; i < 5; i++)
66        {
67            if (i == 0)
68                twist.linear.x = 2.0;
69            else if (i == 1)
70            {
71                twist.linear.x = 2.0;
72                twist.linear.y = 2.0;
73            }
74            else if (i == 2)
75            {
76                twist.linear.x = -2.0;
77                twist.linear.y = 0.0;
78            }
79            else if (i == 3)
80                twist.linear.x = -2.0;
81            else if (i == 4)
82            {
```

```
83                 twist.linear.x = 2.0;
84                 twist.linear.y = -2.0;
85             }
86             TurtlePublisher->publish(twist);
87             std::this_thread::sleep_for(500ms);
88             QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
89                                     .arg(twist.linear.x)
90                                     .arg(twist.linear.y)
91                                     .arg(twist.angular.z);
92             emit log(log_msg);
93         }
94     }
95
96     void QNode::drawcircle()
97     {
98         using namespace std::chrono_literals;
99         auto twist = geometry_msgs::msg::Twist();
100         set_pen(0, 0, 255, 15);
101
102
103         twist.linear.x = 2.0;
104         twist.angular.z = 2.0;
105         TurtlePublisher->publish(twist);
106         std::this_thread::sleep_for(500ms);
107         QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
108                                 .arg(twist.linear.x)
109                                 .arg(twist.linear.y)
110                                 .arg(twist.angular.z);
111         emit log(log_msg);
112     }
113
114     void QNode::move_forward() {
115
116         twist.linear.x = 0.0;
117         twist.linear.y = 0.0;
118         twist.angular.z = 0.0; // reset
119         twist.linear.x = 2.0;
120         TurtlePublisher->publish(twist);
121         QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
```

```
117         twist.linear.y = 0.0;
118         twist.angular.z = 0.0; // reset
119         twist.linear.x = 2.0;
120         TurtlePublisher->publish(twist);
121         QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
122                               .arg(twist.linear.x)
123                               .arg(twist.linear.y)
124                               .arg(twist.angular.z);
125 
126         emit log(log_msg);
127     }
128 
129 
130     void QNode::move_back() {
131         twist.linear.x = 0.0;
132         twist.linear.y = 0.0;
133         twist.angular.z = 0.0; // reset
134         twist.linear.x = -2.0;
135         TurtlePublisher->publish(twist);
136         QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
137                               .arg(twist.linear.x)
138                               .arg(twist.linear.y)
139                               .arg(twist.angular.z);
140         emit log(log_msg);
141     }
142 
143     void QNode::turn_left() {
144         twist.linear.x = 0.0; // reset
145         twist.linear.y = 0.0;
146         twist.angular.z = 0.0;
147          twist.angular.z = 2.0;
148         TurtlePublisher->publish(twist);
149         QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
150                               .arg(twist.linear.x)
151                               .arg(twist.linear.y)
152                               .arg(twist.angular.z);
153         emit log(log_msg);
154     }
```

```
154     }
155
156     void QNode::turn_right() {
157         twist.linear.x = 0.0;
158         twist.linear.y = 0.0;
159         twist.angular.z = 0.0; // reset
160         twist.angular.z = -2.0;
161         RCLCPP_INFO(this->get_logger(), "Publishing Twist: linear.x=%.2f, angular.z=%.2f", twist.linear
162         TurtlePublisher->publish(twist);
163         QString log_msg = QString("linear.x=%1, linear.y=%2, angular.z=%3")
164                               .arg(twist.linear.x)
165                               .arg(twist.linear.y)
166                               .arg(twist.angular.z);
167         emit log(log_msg);
168     }
```