<RO:BIT ROS2 - 2일차 과제1 보고서>

작성자: 박기홍

직책: 19기 예비 단원

소속: RO:BIT 지능 팀

작성일: 2024년 11월 4일

목차

1. 요약
2. 프로젝트 개요
3. 관련 이론 및 개념
    3.1 이론적 배경
    3.2 관련 개념
4. 개발 환경
5. 패키지 분석
6. 개발 과정
    6.1 패키지 설계 및 구조
    6.2 코드 및 주요 기능 설명
    6.3 실패 사례와 시도한 방법
7. 결과
8. 결론 및 향후 발전 방향
9. 참고 문헌
10. 부록

1. 요약

이 보고서는 ROS2 환경에서 Qt GUI를 사용하여 turtlesim 패키지를 제어하는 프로젝트의 개발 과정과 결과를 설명한다. qt_turtlesim_controller 패키지는 1일차 과제에서 수행한 기능들(거북이 조종, 배경색 설정, 거북이 모양 변경, 펜 설정)과 2일차 과제에서 수행한 기능들(도형 그리기, 크기 선택)을 Qt GUI로 통합하여 구현한 패키지이다.

본 패키지는 RO:BIT의 ROS2 - 2일차 과제1에 대한 패키지로 제시된 요구 사항을 충족하는 데 중점을 두었다. 이 보고서에서는 프로젝트의 개요, 관련 이론, 개발 환경, 개발 과정, 결과 등을 다루고 있다.

2. 프젝젝트 개요

이 프로젝트의 목표는 ROS2 환경에서 Qt GUI를 활용하여 turtlesim 패키지를 효과적으로 제어하는 GUI 애플리케이션 패키지인 qt_turtlesim_controller를 개발하는 것이다. 기존의 ROS2 명령어 기반 CLI 제어 방식이 아닌, GUI를 통해 사용자가 직관적으로 turtlesim을 조작할 수 있도록 구현하는 것이 핵심이다.

3. 관련 이론 및 개념

3.1 이론적 배경

이 프로젝트는 ROS2의 퍼블리셔-서브스크라이버 모델을 바탕으로 한다. ROS2 (Robot Operating System 2)는 로봇 소프트웨어 개발을 지원하는 프레임워크로, 노드(Node), 토픽(Topic), 메시지(Message), 퍼블리셔(Publisher), 서브스크라이버(Subscriber) 등의 개념을 활용해 노드 간의 통신을 구현한다. 이러한 구조를 통해 분산 시스템 내에서 모듈 간 데이터 교환을 효율적으로 처리할 수 있다.

퍼블리셔-서브스크라이버 모델: 이 모델은 비동기 방식의 데이터 송수신을 위해 사용된다. 특정 노드가 데이터를 퍼블리시하면(publish), 해당 데이터를 구독(subscribe)하는 다른 노드들이 이를 수신하는 방식이다. 예를 들어, 본 프로젝트의 경우 거북이의 움직임을 제어하기 위해 cmd_vel이라는 토픽을 퍼블리시하고, 이 값을 통해 거북이의 위치와 회전각도를 조절한다.

서비스(Service): 서비스는 요청(Request)과 응답(Response) 형태로 구성되며, 동기식 데이터 송수신을 위해 사용된다. 본 프로젝트에서는 배경색 설정과 펜의 속성 설정을 위해 서비스 구조를 활용하였으며, 이를 통해 즉각적으로 사용자가 요청한 값을 적용할 수 있다.

3.2 관련 개념

- **노드(Node)**: ROS2에서 독립적으로 실행되는 프로세스이다. 본 프로젝트에서는 Talker 노드와 Listener 노드가 각각 퍼블리셔와 서브스크라이버 역할을 수행한다. 두 노드는 독립적으로 실행되며, 서로 다른 터미널에서 구동된다.
- **토픽(Topic)**: 토픽은 노드 간 비동기적으로 데이터를 주고받기 위한 채널이다. 본 프로젝트에서는 cmd_vel 토픽을 사용하여 거북이의 속도와 회전 각도를 제어하였다. cmd_vel은 퍼블리셔가 거북이의 움직임 명령을 송신하고, 이를 수신한 turtlesim 노드가 실제로 거북이를 이동시키는 역할을 한다.

- **서비스(Service)**: 서비스는 동기식 통신 메커니즘으로, 클라이언트가 서버에 요청을 보내고 응답을 받는 형태로 동작한다. 본 프로젝트에서는 배경색을 변경하거나 펜의 색상과 두께를 설정하는 작업에서 서비스를 사용하였다.
- **ROS2 메시지 타입**: ROS2에서는 노드 간 데이터를 주고받기 위해 다양한 메시지 타입을 제공한다. 본 프로젝트에서는 geometry_msgs::msg::Twist 타입을 사용하여 거북이의 위치와 방향을 정의하고, turtlesim::srv::SetPen 타입을 사용하여 펜의 속성(색상, 굵기)을 설정하였다.

4. 개발 환경

이 프로젝트는 Ubuntu 22.04에서 ROS2 Humble을 기반으로 수행되었다. 하드웨어는 Intel i9 프로세서와 32GB RAM을 갖춘 Lenovo Legion 노트북을 사용하하였다. 또한, ROS2 패키지 개발을 위해 Visual Studio Code와 Git을 사용하여 코드 작성과 버전 관리를 하였다.

소프트웨어 환경:

운영체제: Ubuntu 22.04 LTS

ROS2 버전: Humble Hawksbill

IDE 및 VCD: Visual Studio Code, Git

컴파일러: GCC (GNU Compiler Collection) 11.4.0

CMake: 프로젝트 빌드 및 패키지 관리를 위해 CMake를 사용하였으며, ROS2 패키지의 빌드 시스템인 ament_cmake를 사용하여 프로젝트를 구성하였다.

필수 패키지:

ament_cmake: ROS2 빌드 시스템을 지원하는 패키지

rclcpp: ROS2 C++ 클라이언트 라이브러리

std_srvs: 기본 서비스 메시지 패키지로, 배경색 변경 등의 다양한 서비스 구현을 위해 필요

turtlesim: ROS2에서 제공하는 시뮬레이션 패키지

geometry_msgs: 거북이의 위치와 방향을 정의하기 위해 사용되는 메시지 패키지(Twist 메시지 타입을 사용해서 거북이의 속도와 회전 명령을 송수신 할 수 있다)

Qt5: GUI 구축을 위한 라이브러리

5. 패키지 분석

이 프로젝트에서 개발한 qt_turtlesim_controller 패키지는 Qt GUI를 통해 turtlesim의 다양한 기능을 제어하기 위해 설계 되었다. 이 패키지는 ROS2의 퍼블리셔-서브스크라이버 모델과 서비스 구조를 활용했으며, 사용자의 입력을 받아 turtlesim 시뮬레이션을 제어하는 것을 목표로 개발했다.

qt_turtlesim_controller 패키지의 주요 구성 요소는 다음과 같다.

GUI

- **QPushButton**: 각 버튼을 통해 거북이를 조종하고, 도형을 그리고, 배경색과 펜 설정 등을 한다.
- **QLabel**: cmd_vel 토픽의 값을 실시간으로 GUI에 출력한다.

주요 기능

1. **거북이 조종**
2. **배경색 변경**
3. **펜 스타일 변경**
4. **도형 그리기**
5. **크기 변경**
6. **cmd_vel 출력**

이 패키지는 Qt GUI와 ROS2 간의 원활한 상호작용을 위해 로빛 17기 주장이신 이명진 선배님의 "**ros2_create_qt_pkg**" 스크립트를 사용하여 환경을 구성했다.

6. 개발 과정

6.1. 패키지 설계 및 구조

패키지와 노드 구조를 설계 할 때 노드를 독립적으로 구현하는 것에 초점을 맞추어 개발을 진행하는 것이 중요하다. 그러나, 이번 과제는 ros2_create_qt_pkg로 간편하게 Qt 기반의 C++ ROS 패키지를 생성하였으며 이미 구현한 내용을 Qt와 연동하는 것에 초점을 두어 빠르게 개발하기 위해 노드를 독립적으로 구현하지 않았다.

6.2. 코드 및 주요 기능 설명

ROS2와 Qt를 통합하여 패키지를 개발하게 되면 코드의 양이 길어질 수 밖에 없다. 구성에 대한 내용은 주석 처리를 하였으며, 이번 보고서에서는 주요 핵심 기능에 대해서만 다루도록 하겠다.

6.2.1. MainWindow

6.2.1.1. MainWindow() 생성자

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

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

  qnode = new QNode();

  QObject::connect(qnode, SIGNAL(rosShutDown()), this, SLOT(close()));

  // 버튼 클릭 시 거북이 제어를 위한 연결 (QPushButton 사용)
  connect(ui->btnMoveFront, &QPushButton::clicked, this,
&MainWindow::onMoveFront);
  connect(ui->btnMoveBack, &QPushButton::clicked, this,
&MainWindow::onMoveBack);
  connect(ui->btnTurnLeft, &QPushButton::clicked, this,
&MainWindow::onTurnLeft);
  connect(ui->btnTurnRight, &QPushButton::clicked, this,
&MainWindow::onTurnRight);
```

```
  // 배경색 제어를 위한 연결
  connect(ui->btnDeploy, &QPushButton::clicked, this,
&MainWindow::onDeployBackgroundColor);

  // cmd_vel 업데이트를 위한 연결
  connect(qnode, SIGNAL(rosShutDown()), this, SLOT(close()));
  connect(qnode, SIGNAL(cmdVelUpdated(double, double, double, double,
double, double)), this, SLOT(updateCmdVel(double, double, double,
double, double, double)));

  // Pen 제어를 위한 연결
  connect(ui->btnDeploy_Pen, &QPushButton::clicked, this,
&MainWindow::onDeployPenColor);

  // 삼각형 그리기 버튼 연결
  connect(ui->btnDraw_Triangle, &QPushButton::clicked, this,
&MainWindow::onDrawTriangle);
  // 네모 그리기 버튼
  connect(ui->btnDraw_Quadrilateral, &QPushButton::clicked, this,
&MainWindow::onDrawQuadrilateral);
  // 원형 그리기 버튼
  connect(ui->btnDraw_Circle, &QPushButton::clicked, this,
&MainWindow::onDrawCircle);


}
```

버튼 클릭에 대한 연결과 cmd_vel를 업데이트하기 위한 connect를 진행하였다. connect()는 다음과 같은 구조를 가지고 있다.

connect(발신자, SIGNAL(신호), 수신자, SLOT(슬롯));

발신자: 이벤트를 발생시키는 객체

신호: 발신자가 발생키는 이벤트

수신자: 이벤트 처리 객체

슬롯: 이벤트가 발생되었을 때 처리되는 함수

6.2.1.2. TurtleControl 메서드들

```
void MainWindow::onMoveFront()
{
  qnode->moveTurtle(1.0, 0.0); // 직진 처리
}

void MainWindow::onMoveBack()
{
  qnode->moveTurtle(-1.0, 0.0); // 후진 처리
}

void MainWindow::onTurnLeft()
{
  qnode->moveTurtle(0.0, 1.0); // 좌회전 처리
}

void MainWindow::onTurnRight()
{
  qnode->moveTurtle(0.0, -1.0); // 우회전 처리
}
```

거북이의 이동, 회전을 처리 하기 위해 네 가지의 슬롯 함수로 구성했다. moveTurtle() 함수에 linear_x와 angular_z 값을 전달하여 거북이가 이동되도록 한다.

슬롯 함수: 이벤트 처리를 위해 사용되는 함수(예: 버튼 클릭 이벤트가 발생될 때 호출되는 함수)

6.2.1.3. StyleChange 메서드들

```
// 배경 제어 메서드
void MainWindow::onDeployBackgroundColor()
{
  int r = ui->lineEditBgR->text().toInt();
  int g = ui->lineEditBgG->text().toInt();
  int b = ui->lineEditBgB->text().toInt();
  qnode->setBackgroundColor(r, g, b);
}

// Pen 제어 메서드
void MainWindow::onDeployPenColor()
{
```

```
  int r = ui->lineEditBgR_Pen->text().toInt();
  int g = ui->lineEditBgG_Pen->text().toInt();
  int b = ui->lineEditBgB_Pen->text().toInt();
  int width = ui->lineEditWidth_Pen->text().toInt();
  qnode->setPenStyle(r, g, b, width);
}
```

onDeployBackgroundColor(), onDeployPenColor() 메서드는 각각 UI 상의 lineEdit에 입력된 값을 변수에 할당하여, 각각 백그라운드 컬러와 펜의 컬러를 변경한다.

6.2.1.4. updateCmdVel() 메서드

```
// cmd_vel를 라벨에 업데이트 하는 메서드
void MainWindow::updateCmdVel(double linear_x, double linear_y, double
linear_z, double angular_x, double angular_y, double angular_z)
{
    // "Linear X: 0.0 | Angular X: 0.0" 형식으로 업데이트
    ui->label_Linear_and_Angular_X->setText(QString("Linear X: %1 |
Angular X: %2").arg(linear_x).arg(angular_x));
    ui->label_Linear_and_Angular_Y->setText(QString("Linear Y: %1 |
Angular Y: %2").arg(linear_y).arg(angular_y));
    ui->label_Linear_and_Angular_Z->setText(QString("Linear Z: %1 |
Angular Z: %2").arg(linear_z).arg(angular_z));
}
```

updateCmdVel()는 거북이의 cmd_vel을 라벨에 업데이트 하는 메서드이다. Y 값은 평면에서 없어도 되나, X, Y, Z에 대한 모든 값을 출력 하고 싶어서 추가했다. 그래도 Y의 값은 0.0으로 고정이다.

6.2.1.5. 도형을 그리는 메서드들

```
// 삼각형 그리는 메서드
void MainWindow::onDrawTriangle() {
  // lineEdit의 값을 가져오고, 비어 있다면 디폴트 값으로 1 설정함
  // 삼항 연산으로 처리해야 하는 두 가지의 연산을 깔끔하게 구현함
  double sideLength = ui->lineEdit_LengthSide->text().isEmpty() ? 1.0 :
ui->lineEdit_LengthSide->text().toDouble();

  // 삼각형 그리기 메서드 호출
  qnode->drawTriangle(sideLength);
}
```

```
// 네모 그리는 메서드
void MainWindow::onDrawQuadrilateral()
{
    // 네모를 그리기 위한 side 길이 입력받기
    int sideLength = ui->lineEdit_LengthSide->text().isEmpty() ? 1.0 :
ui->lineEdit_LengthSide->text().toDouble();
    qnode->drawQuadrilateral(sideLength);
}

// 원 그리는 메서드
void MainWindow::onDrawCircle()
{
    // 원을 그리기 위한 다이아미터 길이 입력받기
    int diameterLength = ui->lineEdit_LengthDiameter->text().isEmpty() ?
1.0 : ui->lineEdit_LengthDiameter->text().toDouble();
    qnode->drawCircle(diameterLength);
}
```

onDrawTriangle(), onDrawQuadrilateral(), onDrawCircle() 메서드는 각각의 도형을 그리는 메서드이다. 삼항 연산으로 깔끔하게 처리하였다. UI 상의 크기 관련 부분에 값이 비어 있으면 1.0으로 도형이 그려지도록 하였으며, 그렇지 않을 경우 사용자가 입력한 값을 크기로 지정하여 도형이 그려지도록 설계하였다.

6.2.2. qnode

qnode에는 기능들이 구성되어 있다. 기능들은 ROS2 - 1일차 과제1, 2에서 구현한 내용과 동일하기에 대표적으로 하나만 본 보고서에서 설명하겠다.

6.2.2.1. moveTurtle() 메서드

```
void QNode::moveTurtle(double linear_x, double angular_z)
{
  auto twist_msg = geometry_msgs::msg::Twist();
  twist_msg.linear.x = linear_x;
  twist_msg.angular.z = angular_z;
```

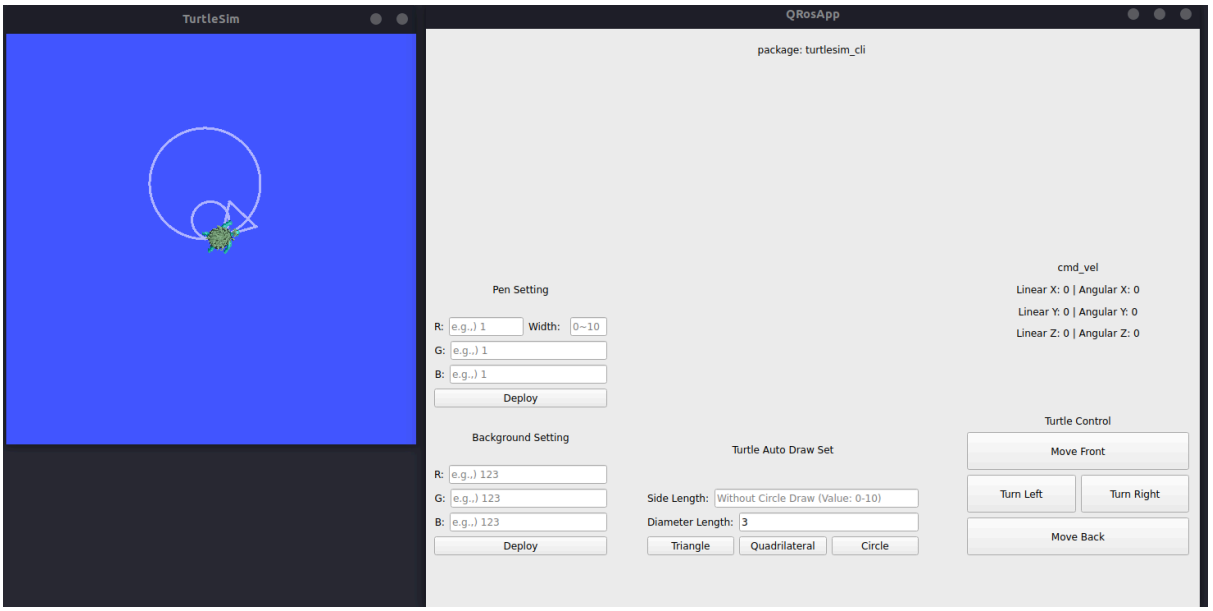
```
  velocity_publisher->publish(twist_msg);
}
```

moveTurtle() 메서드는 거북이를 조종하는 메서드이다. Button을 통해 파라미터로 받은 값에 따라 twist_msg.linear.x와 twist_msg.angular.z에 값을 저장하고, 퍼블리시하여 twist_msg를 /turtle1/cmd_vel 토픽으로 퍼블리시한다.

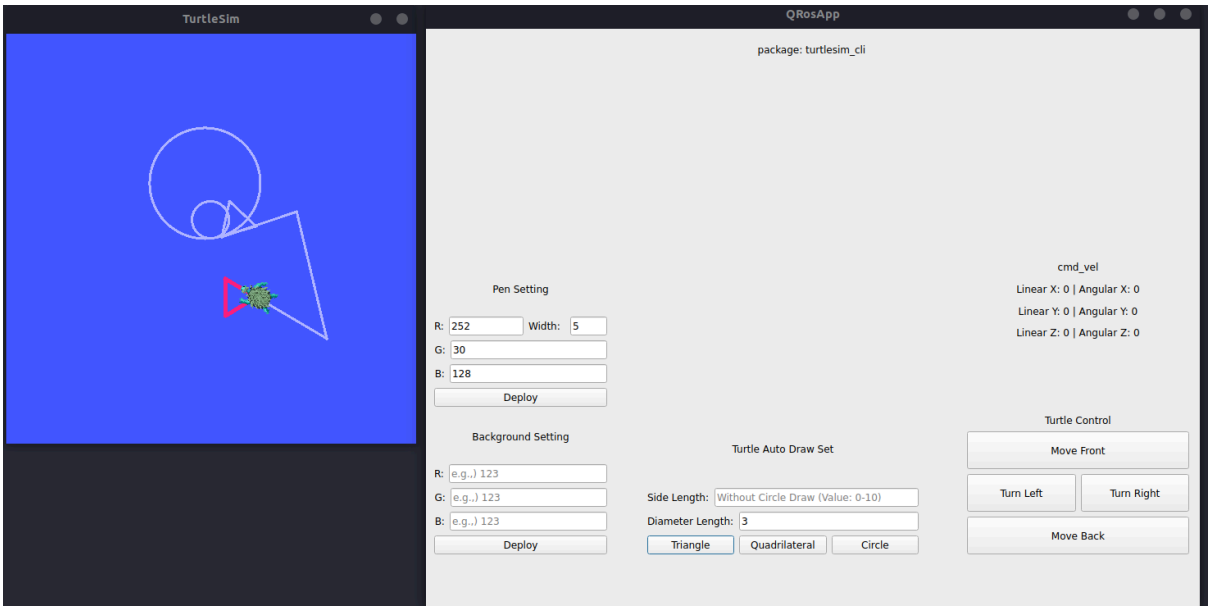
6.3.실패 사례와 시도한 방법

qnode에 기능을 다 구현하는 것이 아니라, 기능별 하나의 노드를 만들어 끌어와 사용하려고 하였으나, 오히려 더 복잡하게 느껴져서 qnode에 구현하게 되었다. 원 그리기 구현이 안 되어 있는 상태에서 이후 과제를 진행하다가, 원 그리기를 구현하여 적용하였다.

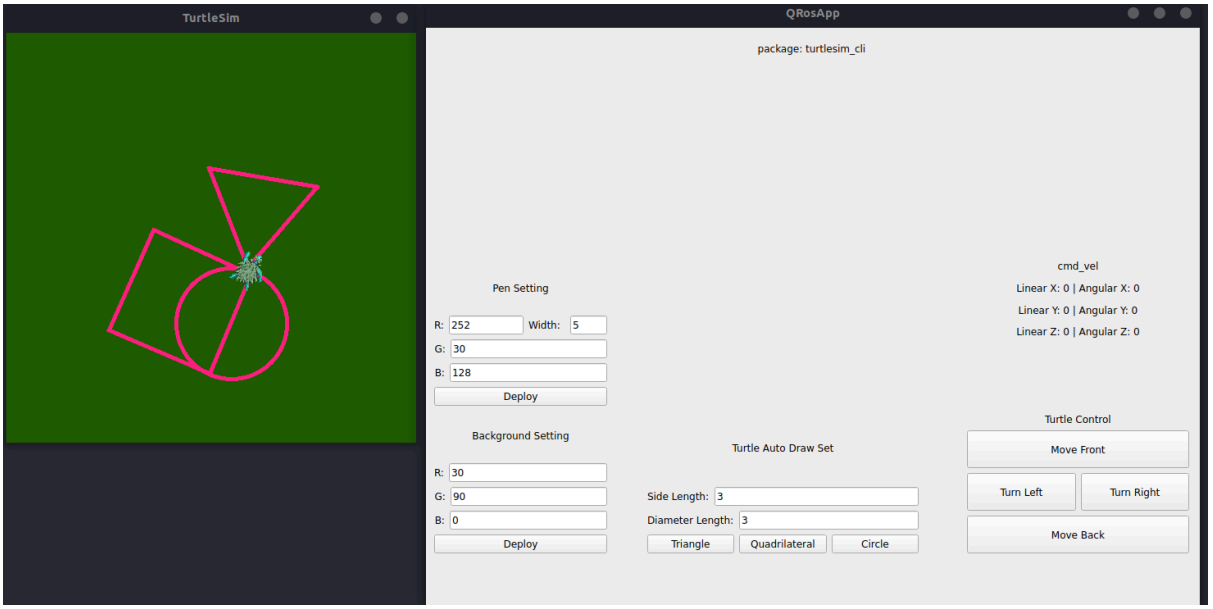
7. 결과



<qt_turtlesim_controller 실행 결과 1>



<qt_turtlesim_controller 실행 결과 2>



<qt_turtlesim_controller 실행 결과 3>

8. 결론 및 향후 발전 방향

이번 프로젝트에서는 ROS2 환경에서 Qt GUI를 활용하여 turtlesim 패키지를 직관적으로 제어할 수 있는 qt_turtlesim_controller 패키지를 개발하였다. 이 패키지는 1일차 과제1과 과제 2의 기능들을 통합하여, 사용자 인터페이스를 통해 거북이의 이동, 배경색 설정, 펜 속성 변경, 다양한 도형 그리기 등을 손쉽게 수행할 수 있도록 구현하였다.

개발하면서 UI 배치에도 시간을 사용했으나, 나는 프론트엔드 개발자가 아니며, 백엔드에도 신경을 써서 기능 구현을 중심으로 작업해야 한다는 말씀이 떠올랐다. 그래서 UI는 볼 수 있을 정도로만 배치하고, 기능 구현에 시간을 몰두했다.

9. 참고 문헌

- 표윤석. (2020.08.18.). "000 로봇 운영체제 ROS 강좌 목차". https://cafe.naver.com/openrt/24070
- 표윤석. (2024.09.05.). "로봇 운영체제 ROS 2 특강 발표 자료", https://docs.google.com/presentation/d/1QvUHj8iZCWIGiWRCgHFG-DLAmoUK6khO9xBXCzjPTTs/edit
- i_robo_u. (n.d.). "인공지능 로봇 개발". Velog. https://velog.io/@i_robo_u/series/%EC%9D%B8%EA%B3%B5%EC%A7%80%EB%8A%A5%EB%A1%9C%EB%B4%87%EA%B0%9C%EB%B0%9C

10. 부록

10.1 레파지토리 링크

ROBIT_ROS2_HW: https://github.com/kinesis19/ROBIT_ROS2_HW

10.2 과제 Wiki

ROS2_HW_DAY_2: https://github.com/kinesis19/ROBIT_ROS2_HW/wiki/ROS2_HW_DAY2

10.3 패키지 구조

패키지의 구조는 다음과 같다:

```
qt_turtlesim_controller
├── CMakeLists.txt
├── include
│   └── qt_turtlesim_controller
│       ├── main_window.hpp
│       └── qnode.hpp
├── package.xml
├── resources
│   ├── images
│   │   └── icon.png
│   └── images.qrc
├── src
│   ├── main.cpp
│   ├── main_window.cpp
│   └── qnode.cpp
└── ui
    └── mainwindow.ui
```