HW1 보고서

로빛 20기 인턴 2025407006 모시온

목차

1. 프로젝트 개요

본 프로젝트는 QT를 활용하여 사용자에게 상호작용 인터페이스를 제공하고 해당 제공되는 서비스를 통해 가상으로 구현한 3축 매니퓰레이터를 제어하는 목적을 지닙니다.

이때, 매니퓰레이터 각각의 관절 및 링크를 C++ 환경에서 사용가능한 Class 개념을 적용하여 사용자가 각 관절을 개별적으로 제어할 수 있어야하며 상태를 문자로 저장하여 불러올 시 저장 상태가 유지될 수 있도록 해야합니다.

또한 시계 또는 반시계 방향 회전을 자동으로 가능하게 제작하여야하는 점을 고려하여 입력 버튼 하나를 통해 시계방향 회전을 구현하였습니다.

2. 알고리즘 설계

2-1. .h/.hpp 헤더 파일 설계

```
#include <QMainWindow>
#include <QGraphicsScene>
#include <QGraphicsRectItem>
#include <QTimer>
```

QT 환경을 통해 가상의 매니퓰레이터를 제어하기 때문에 그와 관련하여 각각의 모듈을 호출하였습니다.

QMainWindow	-	QT의 기본 UI를 구성할 때 요구되는 위젯 모듈
QGraphicsScene	-	UI의 GraphicView에서 도형 및 문자, 선 작성 모듈
QGraphicsRectItem	-	QGraphicsScene 종속관계로 사각형 작성 모듈
QTimer	-	QT 환경에서 시간을 접목하여 제어하는 모듈

```
QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE
```

namespace 설정을 통해 Git을 통한 외부인의 이용을 가능하도록 QT 내부에서 기본으로 정의되어 집니다. 이때, MainWindow Class를 할당합니다.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

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

//UI 슬롯들
private slots:
    void on_save_clicked();
    void on_load_clicked();
    void on_rotate_clicked();

    void on_slider1_actionTriggered(int value);
    void on_slider2_actionTriggered(int value);
    void on_slider3_actionTriggered(int value);
    //타이머 제어 -> 시계 방향!
    void autoRotate();
```

Class로 정의된 MainWindow에서 각각의 public, private, private slots 영역을 설정하였습니다.
Q_OBJECT 는 UI와의 상호작용을 통한 슬롯 또는 신호를 활성화하는 모듈입니다.

우선 public에서는 Mainwindow와 관련하여 앞으로 사용할 위젯을 부모(parent)라는 포인터 변수를 선언하여 상위 및 하위 위젯간의 관계를 종속시킵니다.

이후 UI/Designer를 통해 추가한 슬라이더 3개, 버튼 3개에 대하여 Gotoslot을 통한 각 버튼의 제어를 코드에 상속하여 상호작용이 이루어질 수 있도록 하였습니다.
각각의 버튼들은 다음의 기능을 구현합니다.

save_clicked - 저장 버튼의 눌림 후의 동작 함수
load_clicked - 불러오기 버튼의 눌림 후 동작 함수
rotate_clicked - 공지 사항의 시계 또는 반시계 방향의 회전 동작 함수
(보고서 작성자(필자)의 경우 시계방향으로 정하여 구현하였습니다.)

slider1_actionTriggered - 1번 슬라이더 감지 후 동작 함수
slider2_actionTriggered - 2번 슬라이더 감지 후 동작 함수
slider3_actionTriggered - 3번 슬라이더 감지 후 동작 함수

autorotate - 시계 방향 회전의 경우 시간(Qtimer)를 통해 구현하기 때문에 시간 처리 관련 동작을 포함하는 함수

```
private:
    Ui::MainWindow *ui;

    QGraphicsScene *scene;
    QGraphicsRectItem *link1;
    QGraphicsRectItem *link2;
    QGraphicsRectItem *link3;

    int angle1, angle2, angle3;

    QTimer *timer;
};
```

마지막으로 private 공간에서는 UI의 구성 요소들을 관리하는 변수를 ui 포인터 변수로 선언하며 사용자가 매니퓰레이터를 바라볼 장면을 제공하는 scene, 3축을 기준으로 각각의 링크들을 포인터로 선언하며, 각 관절의 각도는 angle(1~3)을 통해 저장하였습니다.
(Q타이머 기능을 직접적으로 .cpp 소스파일에서 다룰 수 있도록 포인터 변수를 선언)

2-2. .c/.cpp 소스파일 설계

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
//메모장 파일용 모듈 호출
#include <QFile>
#include <QTextStream>
```

이전에 설계한 헤더파일 및 텍스트 파일 저장을 위해 QT 함수 모듈을 호출하였습니다.

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

    //GUI제작
    scene = new QGraphicsScene(this);
    ui->graphicsView->setScene(scene);
    ui->graphicsView->setFixedSize(400,400);
```

가장 처음의 MainWindow::MainWindow 문을 통해 생성자로서 부모 위젯으로 설정한 ui를 새로 정의하며 앞으로 내부 멤버 함수 및 변수들을 ui를 통해 객체를 가르키도록 하여 제어합니다.

이후 setupUi(this)를 통해 Class 내부에 준비된 버튼 및 슬라이더를 새로 생성하여

준비될 수 있도록 합니다. 이때, 부모와 자식의 위젯 설정을 연결시켜 종속되도록 합니다.

다음으로 사용자가 매니퓰레이터의 모습을 볼 수 있도록 scene 변수를 QGraphicsScene을 통해 this(메인윈도우)를 부모 위젯으로 설정하여 생성하며
이전 UI 디자이너를 통해 설치한 graphicView를 볼 수 있도록 scene을 해당 설치된 graphicView에 넣어 시각화합니다.
하지만 사용자에게 보여지는 화면의 크기가 중요하므로 이를 setFixedSize를 통해 각각 가로, 세로 길이를 400*400으로 설정합니다.

```
//링크 사용
//Paint Event 참고
//각 링크 순서대로 RGB 색깔 부여
link1 = scene->addRect(0,0,100,10,QPen(Qt::red),QBrush(Qt::red));
link1->setTransformOriginPoint(0,5);

link2 = new QGraphicsRectItem(0,0,100,10,link1);
link2->setBrush(Qt::green);
link2->setTransformOriginPoint(0,5);
link2->setPos(100,0);

link3 = new QGraphicsRectItem(0,0,100,10,link2);
link3->setBrush(Qt::blue);
link3->setTransformOriginPoint(0,5);
link3->setPos(100,0);
```

3축에 따른 각각의 링크 또한 각각의 링크를 통해 제어하므로 다음과 같이 사용하였습니다.
(각각의 링크의 색은 RGB 구성으로 맞추었습니다.)
각각의 링크의 연결 유지가 중요하므로 첫번째로 생성되는 링크1은 Scene위에 addRect를 통해 직사각형의 형태로 직접 생성됩니다. 이때, 과제 공지를 참고하여 PaintEvent의 코드를 분석하여 알아낸 펜과 QBrush를 통해 빨간색으로 채워진 직사각형을 그렸으며 해당 링크의 회전 중심점을 (0,5)로 두어 기준을 정하였습니다.

이후 아래의 링크2와 링크3 또한 새로 생성하나, 생성 위치 중심을 링크1의 끝을 기준으로 생성하도록 하여 연결성을 유지하였습니다. 사이즈는 이전 링크1과 동일하나 색깔은 초록으로 지정하여 구분될 수 있도록 하였습니다.
이때, 링크2의 회전 중심은 링크2의 생성 기준을 링크1로 정하였기에 (0,5)로 설정하여 링크1 끝부분에서 회전하도록 하였습니다. 링크3 또한 링크2를 기준으로 생성하며 (0,5)의 회전 중심 설정을 통해 전체 링크를 모두 연결하였습니다.

```
//관점 조정용
ui->graphicsView->centerOn(link1->boundingRect().center());

//각도 초기화
angle1 = angle2 = angle3 = 0;

//타이머 초기화
timer = new QTimer(this);
connect(timer, &QTimer::timeout, this, &MainWindow::autoRotate);

//각 링크 회전각 360도로 지정
ui->slider1->setRange(0, 360);
ui->slider2->setRange(0, 360);
ui->slider3->setRange(0, 360);
```

하지만 매니퓰레이터 대비 넓은 가동범위에 비해 작은 스크린으로 제어 시 시점이 이동하는 문제가 있었으며 이를 해결하기 위하여 다음의
ui->graphicsView->centerOn(link1->boundingRect().center());
를 적용하여 링크1의 둘레를 기준으로 정 가운데를 지칭하는 center()를 계속 바라볼 수 있도록 하였습니다.

처음 코드 실행 시, 각각의 관절 각도를 지정하는 angle(1~3)을 0으로 초기화하며 setRange함수를 통해 각 관절의 최소, 최대 각도를 지정하여 범위를 결정합니다.

timer 또한 QTimer를 통해 메인윈도우를 부모 위젯으로 지정하여 생성하였으며 connect를 통해 timer와 QTimer, 그리고 메인윈도우의 멤버함수 autoRotate를 주소 참조하여 연결지었습니다.

```
void MainWindow::on_slider1_actionTriggered(int value) {
    angle1 = value;
    link1->setRotation(angle1);
}
void MainWindow::on_slider2_actionTriggered(int value) {
    angle2 = value;
    link2->setRotation(angle2);
}
void MainWindow::on_slider3_actionTriggered(int value) {
    angle3 = value;
    link3->setRotation(angle3);
}
```

각 멤버 함수 정의 부분입니다.
angle(1~3)의 값에 현 슬라이더 값을 매개변수 value를 통해 저장하며 이를 링크(1~3)에 회전할 수 있도록 화살표 연산자를 통해 전달합니다.

```
void MainWindow::on_save_clicked() {
    QFile file("angles.txt");
    if(file.open(QIODevice::WriteOnly)){
        QTextStream out(&file);
        out << angle1;
        out << " ";
        out << angle2;
        out << " ";
        out << angle3;
        out << "₩n";
    }
}

void MainWindow::on_load_clicked() {
    QFile file("angles.txt");
    if(file.open(QIODevice::ReadOnly)){
        QTextStream in(&file);
        in >> angle1 >> angle2 >> angle3;
        ui->slider1->setValue(angle1);
        ui->slider2->setValue(angle2);
        ui->slider3->setValue(angle3);
    }
}
```

저장 및 불러오기 버튼을 눌렀을 때 동작하는 함수입니다.
미리 호출한 QFile를 통해 angles.txt라는 파일을 생성할 수 있도록 하였으며 파일이 제대로 열렸을 경우 쓰기 모드를 활성화하여 angle(1~3)를 순서대로 저장할 수 있도록 하였습니다.

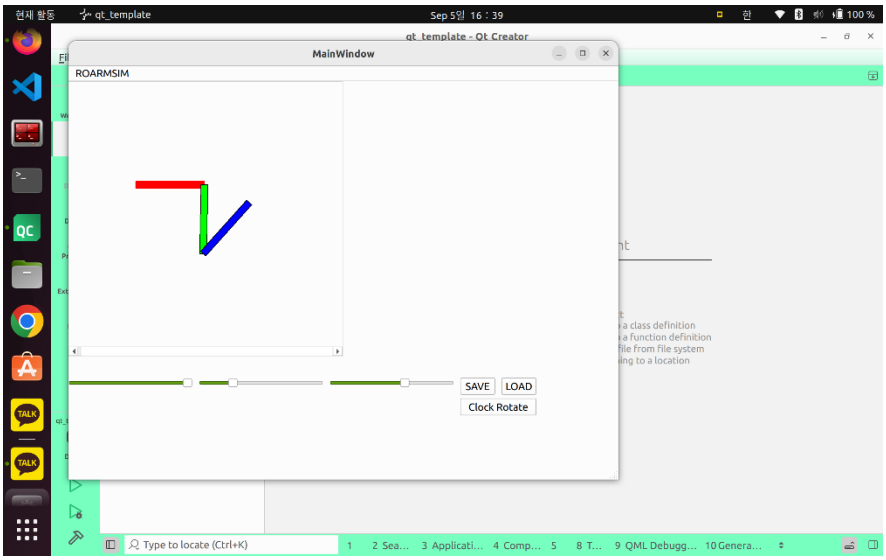
불러오기의 경우 angle1 angle2 angle3 으로 저장된 값을 순서대로 읽을 수 있도록 하기 위하여 띄어쓰기를 분기점으로 하여 각 저장 값이 순서대로 현재의 angle(1~3)에 할당되도록 하였습니다.

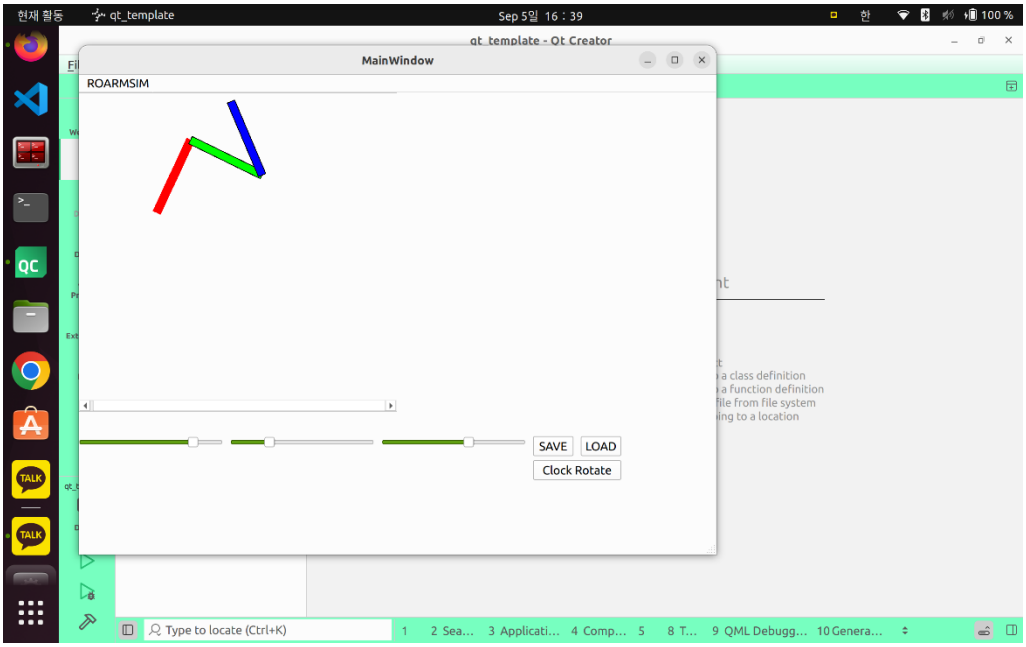
```
void MainWindow::on_rotate_clicked() {
    if(timer->isActive())
        timer->stop();
    else
        timer->start(20);
}

void MainWindow::autoRotate() {
    angle1 = (angle1 + 1) % 360;
    angle2 = (angle2 + 2) % 360;
    angle3 = (angle3 + 3) % 360;

    ui->slider1->setValue(angle1);
    ui->slider2->setValue(angle2);
    ui->slider3->setValue(angle3);

    link1->setRotation(angle1);
    link2->setRotation(angle2);
    link3->setRotation(angle3);

    //관점 이동 방지용
    ui->graphicsView->centerOn(link1->boundingRect().center());
}
```

회전 버튼이 눌린 경우 타이머를 시작하도록 하며 20ms 주기로 신호를 발생시키도록 하였습니다. 만약 이미 가동중인 상태인 경우 정지합니다.

자동 시계 방향 회전을 실행하는 경우 각 angle(1~3)의 범위를 지정하기 위하여 20ms초 주기로 작동할 때 angle(1~3)에 각각 1, 2, 3의 각도 가중치를 다르게 주어 전체 360도를 기준으로 회전할 수 있도록 하였습니다.

3. 실행 결과




현재 활동
qt_template
Sep 5일 16 : 39
100 %
qt_template - Qt Creator
MainWindow
ROARMSIM
SAVE
LOAD
Clock Rotate
Type to locate (Ctrl+K)