DAY2
과제 1번 보고서

2023741024
로봇학부 박건후

목차

(1) mainwindow.ui

mainwindow.ui를 열었을 때 나타나는 화면입니다.

위의 3개의 것은 Spin Box로써, 값을 입력하거나 위 아래 버튼을 누르면 값이 바뀌고 이에 맞춰 슬롯 함수를 만들면 원하는 동작을 구현할 수 있습니다. 왼쪽부터 bottom, middle, upper 관련 회전부에 대한 각도를 수동으로 설정하는 것을 만들었습니다.

이 부분은 일반 버튼이고 클릭을 했을 시 해당 관련 관절이 시계 혹은 반시계로 돌아가도록 한 것입니다. 예를 들어 bottom_clock이라고 하면 bottom부분의 관절이 시계방향으로 간다는 것입니다. counterclock은 반시계방향을 의미합니다.

그리고 bring은 txt파일에 저장한 로봇의 각도, 방향 정보를 가져와서 저장된 상태를 다시 재현하는 기능을 만들기 위해 만들었습니다. bring을 누르면 해당 슬롯 메서드인 loadFromTxt메서드가 호출되어 정보를 가져와 정보대로 로봇 위치, 각도를 초기화합니다.

(2) arm.cpp파일

```
    explicit ArmView(QWidget *parent = nullptr);
    void setBottomAngle(int deg); //해당 각도로 팔 초기화함 setter
    void setMiddleAngle(int deg); //스텝 박스로 값 바꿀 때 호출되는 메서드
    void setUpperAngle(int deg);  //호출되어 각도 초기화함

    int bottomAngle()const;  //각도값을 얻어오는 getter
    int middleAngle()const;
    int upperAngle()const;

    void stepBottom(int deltaDeg); //자동으로 돌아가는 버튼을 눌렀을 때 그 때 자동으로 각도를 초기화하는 함수
    void stepMiddle(int deltaDeg);
    void stepUpper(int deltaDeg);
```

개략적으로 먼저 설명드리기 위해 arm.h의 선언부분을 짧게 설명드리겠습니다. 팔을 돌리는 것을 계산하며 그 상태를 저장하고 또 로봇팔을 그릴 클래스가 필요했습니다. 그래서 ArmView클래스를 만들었습니다. ArmView클래스는 각도와 팔 길이 값을 저장하고 또 각도 값을 바꾸고, 또 그 상태를 그려야 합니다. 각도 값을 바꿀 때는 자동 모드와 수동 모드가 모두 있었기 때문에 각도값을 바꾸는 방식을 분리하여 메서드를 만들고자 했습니다. 슬롯 메서드가 내부적으로 ArmView의 메서드를 호출합니다. 스핀 박스에서 값이 바뀔 때 호출되는 메서드인

```
void setBottomAngle(int deg); //해당 각도로 팔 초기화함 setter
void setMiddleAngle(int deg); //스텝 박스로 값 바꿀 때 호출되는 메서드
void setUpperAngle(int deg);  //호출되어 각도 초기화함
```

메서드들입니다.

그리고 타이머가 돌아갈 때 타임아웃이 될 때마다 각도를 계산하여 내부의 멤버변수값을 바꾸는 메서드인

```
void stepBottom(int deltaDeg); //자동으로 돌아가는 버튼을 눌렀을 때 그 때 자동으로 각도를 초기화하는 함수
void stepMiddle(int deltaDeg);
void stepUpper(int deltaDeg);
```

메서드들입니다. 이렇게 동작하는 모드에 따라 호출되는 메서드를 분리하여 만들었습니다.

```
private:
    double a0=0, a1=0, a2=0;      // bottom, middle, upper관련 관절 각도
    double L1=120, L2=100, L3=80; // 각 팔의 길이
    int wrap361(int v); //메서드 내부에서 호출하기 때문에 private로 은닉화
    void paintEvent(QPaintEvent *) override;
};
```

이제는 private부분입니다. bottom, middle, upper각각과 연관되는 각도값인 a0, a1, a2를 만들었고, 팔의 길이 값도 상태로써 저장해야 하기에 팔 길이값도 저장하였습니다. 그리고 메서드 내부에서 호출되는 함수 즉, 헬퍼함수를 선언하였습니다. wrap361은 각도를 계산할 때 이를 보정하는 메서드이고, paintEvent는 현재 로봇 멤버변수 상태에 따라 위젯 위에 그림을 그리는 메서드입니다.

ArmView의 메서드 구현부를 설명드리겠습니다.(arm.cpp파일)

```
//bottom은 a0, middle은 a1, upper는 a2에 대응됨
void ArmView::setBottomAngle(int deg){ a0 = deg; update(); }
void ArmView::setMiddleAngle(int deg){ a1 = deg; update(); }
void ArmView::setUpperAngle(int deg){ a2 = deg; update(); }
```

스핀 박스의 값이 바뀌었을 때는 그 각도 값으로 멤버변수를 초기화하고 update()를 호출하여 paintEvent로 그림을 최신화하면 됩니다.

```cpp
void ArmView::paintEvent(QPaintEvent*) //이걸로 로봇팔 그림
{
    QPainter p(this);
    p.setRenderHint(QPainter::Antialiasing, true);
    p.fillRect(rect(), Qt::white);

    // 베이스 기준점(바닥 가운데쯤)
    QPointF base(width()*0.5, height()*0.5);

    // 좌표계 세팅
    p.translate(base);

    // 1축(바닥 회전)
    p.save();
    p.rotate(-a0); // 시계/반시계는 필요에 맞춰 부호 변경
    p.setPen(QPen(Qt::black, 6));
    p.drawLine(QPointF(0,0), QPointF(L1,0)); // 첫 링크
    p.drawEllipse(QPointF(0,0), 8, 8);
```

로봇팔을 그리는 부분입니다. 좀 더 원이 부드럽게 보정을 해주는 것과 배경을 깔끔하게 하는 작업을 해준 후에 정해둔 중간점으로 bottom관련 관절을 고정해놓고 그 상태부터 로봇의 각 관절의 각도 상태만큼 돌리게 했습니다. 그리고 펜을 초기화하고 선을 그려 팔을 만들고 그 끝에 동그라미를 그려 그 다음 관절이 그려지게 했습니다.

```
    // 2축(중간 관절)
    p.translate(L1, 0);
    p.rotate(-a1);
    p.setPen(QPen(Qt::darkGray, 5));
    p.drawLine(QPointF(0,0), QPointF(L2,0));
    p.drawEllipse(QPointF(0,0), 7, 7);

    // 3축(끝 관절)
    p.translate(L2, 0);
    p.rotate(-a2);
    p.drawLine(QPointF(0,0), QPointF(L3,0));
    p.drawEllipse(QPointF(0,0), 6, 6);

    // 끝부분은 빨간색으로 표시
    p.setBrush(Qt::red);
    p.drawEllipse(QPointF(L3,0), 5, 5);

    p.restore();
```

그 후 2축, 3축에서도 마찬가지 작업을 하였습니다.

```
int ArmView:: wrap361(int v) { v%=361; if(v<0) v+=361; return v; } //각도 범위 조정


void ArmView::stepBottom(int deltaDeg){ a0 = wrap361((int)a0 + deltaDeg); update(); } //각도 변화
void ArmView::stepMiddle(int deltaDeg){ a1 = wrap361((int)a1 + deltaDeg); update(); }
void ArmView::stepUpper(int deltaDeg) { a2 = wrap361((int)a2 + deltaDeg); update(); }

//getter 메서드 만듦
int ArmView::bottomAngle() const { return int(a0); }
int ArmView::middleAngle() const { return int(a1); }
int ArmView::upperAngle()  const { return int(a2); }
```

private멤버변수인 wrap361에서는 정상적인 각도 범위가 되도록 만들었습니다. 각도값이 360을 넘어가지 않도록 만약 0보다 작다면 각도값이 양수가 되도록 하는 등의 작업입니다.

이 wrap361을 사용하는 멤버함수는 step~메서드들입니다. 이 메서드들은 자동으로 시계, 반시계 방향으로 돌아갈 때 각도값을 최신화하며 그 값이 타이머에 의해 계속 최신화되므로 이에 대한 각도 조절을 해줘야만 했습니다. 그래서 wrap361을 만들게 된 것입니다. 대신 +, -의 경계를 없애면 방향의 개념이 없어지기에 그 방향을 저장하는 변수를

mainwindow클래스의 멤버변수에서 저장하도록 하였습니다.

```
//getter 메서드 만듦
int ArmView::bottomAngle() const { return int(a0); }
int ArmView::middleAngle() const { return int(a1); }
int ArmView::upperAngle()  const { return int(a2); }
```

그리고 이 부분은 getter입니다.

(3) mainwindow.cpp파일

mainwindow.cpp파일을 보기 전에 개략적으로 설명드리기 위해 mainwindow.h파일을 먼저 설명드리겠습니다.

```
//스핀 박스 슬롯 메서드
void on_spinBox_2_valueChanged(int arg1);

void on_spinBox_valueChanged(int arg1);      Slots named on_foo_ba

void on_spinBox_3_valueChanged(int arg1);

//버튼 눌렸을 때의 슬롯 메서드
void on_bottom_clicked();      Slots named on_foo_bar are error pr

void on_bottom_counter_clicked();

void on_middle_clicked();      Slots named on_foo_bar are error pr

void on_middle_counter_clicked();

void on_upper_clicked();      Slots named on_foo_bar are error pro

void on_upper_counter_clicked();
```

위의 사진은 모두 public slots: 아래에 있는 슬롯 메서드들입니다. 스핀 박스의 값이 바뀌면 그 때 호출되는 슬롯 메서드, 버튼이 눌렀을 때 호출되는 슬롯 메서드들입니다.

```
//타임아웃되었을 때
void onBottomTick();
void onMiddleTick();
void onUpperTick();
//타임아웃되어 저장할 때 호출되는 슬롯메서드
void saveToTxt();                // 저장
//bring버튼이 눌렸을 때 호출되는 슬롯메서드
void loadFromTxt();              // 불러오기 (옵션)
```

이번에도 슬롯 메서드들입니다. on~Tick()메서드들은 time_out이 되면 호출되는 슬롯 메서드들입니다. 타임아웃이 되어 슬롯 메서드들이 호출되면 MainWindow클래스의 ArmView포인터를 통해 각도를 바꾸는 함수인

```
void ArmView::stepBottom(int deltaDeg){ a0 = wrap361((int)a0 + deltaDeg); update(); }
void ArmView::stepMiddle(int deltaDeg){ a1 = wrap361((int)a1 + deltaDeg); update(); }
void ArmView::stepUpper(int deltaDeg) { a2 = wrap361((int)a2 + deltaDeg); update(); }
```

를 내부적으로 호출하여 타임 아웃이 될 때마다 각도가 바뀌어지도록 만들었습니다.

```
//타임아웃되어 저장할 때 호출되는 슬롯메서드
void saveToTxt();                    // 저장
//bring버튼이 눌렸을 때 호출되는 슬롯메서드
void loadFromTxt();                  // 불러오기 (옵션)
```

saveToTxt 슬롯 메서드는 2초라는 시간이 지나면 타임아웃이 되고 그 때 현재 로봇의 각도 정보, 방향 정보를 저장하는 이 메서드가 호출되는 방식으로 작동됩니다. 그래서 txt파일을 만들고 스트림을 통해 파일 출력을 하여 내용을 저장합니다.

그리고 loadFromTxt는 파일로부터 정보를 받고, 그 정보로 로봇의 상태를 초기화합니다.

```
private:
    Ui::MainWindow *ui;
    ArmView* armView;
    QTimer bottomTimer, middleTimer, upperTimer, saveTimer;
    int bottomDir=0, middleDir=0, upperDir=0; // +1 시계, -1 반시계
    QString savePath;                // 저장 파일 경로

    int intervalMs = 20; // 틱 간격(ms)

    void toggleTimer(QTimer& t, int& dirVar, int newDir);
```

말씀드렸듯, ArmView포인터를 멤버변수로 넣어서 타이머를 사용하며 외부 유틸리티를 사용하며 그 내부에서 ArmView의 멤버변수를 조정하고 초기화하는 것을 해야 했기에 ArmView포인터를 MainWindow클래스 내부에서 선언하였습니다. 각각의 움직임에 대한 타이머를 설정하고 방향을 저장하는 ~Dir을 선언하였습니다. 그리고 파일을 생성하고 출력할 때 필요한 경로를 QT에서 쉽게 하기 위해 QString클래스를 사용하였습니다. 자동모드에서는 QTimer로 움직이기 때문에 시간 간격마다 각도를 최신화하기 위해 타임 아웃되는 간격을 20ms로 정하고 이 상태를 저장하였습니다. 그리고 toggleTimer를 통해 버튼이 눌렸을 때와 타이머가 시작되는 것을 연결하는 것을 구현하였습니다.

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    // 스핀박스 최대값 360
    ui->spinBox->setRange(0, 360);              //최대 두자리 수여서 360까지 값을 받을 수 있도록 설정
    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);
```

mainwindow.ui에서의 스핀박스 객체명을 기입하여 ui포인터가 가리킬 수 있도록 하고 각도 값을 0~360으로 값을 변경할 수 있도록 만들었습니다.

그리고 ArmView객체를 만드는 것을 MainWindow에서 통제하는 방식으로 설계하여 하나의 객체만을 가지고 로봇 팔 정보를 저장하고 그리도록 하였습니다. 로봇 팔을 딱 1개만 만들도록 통제해야 하기 때문입니다.

그림을 그릴 때 위젯 위에 그림을 그리게 되는데 그렇게 만들기 위한 유틸리티를 사용하여 위젯 객체명을 armViewHost로 기입하고 이를 코드에 작성하여 위젯 관련 상위 클래스의 객체를 만들어서 이를 통해 위젯을 초기화하고 ArmView를 레이아웃에 추가하는 것을 할 수 있었습니다. 이 코드에서는 다형성의 개념을 많이 사용한 것 같습니다.

```
//버튼 설정 시그널과 슬롯 연결
connect(ui->bottom_clock,        &QPushButton::clicked, this, &MainWindow::on_bottom_cli
connect(ui->bottom_counterclock, &QPushButton::clicked, this, &MainWindow::on_bottom_cou

connect(ui->middle_clock,        &QPushButton::clicked, this, &MainWindow::on_middle_cli
connect(ui->middle_counterclock, &QPushButton::clicked, this, &MainWindow::on_middle_cou

connect(ui->upper_clock,         &QPushButton::clicked, this, &MainWindow::on_upper_clic
connect(ui->upper_counterclock,  &QPushButton::clicked, this, &MainWindow::on_upper_cour


//타이머 연결
connect(&bottomTimer, &QTimer::timeout, this, &MainWindow::onBottomTick);
connect(&middleTimer, &QTimer::timeout, this, &MainWindow::onMiddleTick);
connect(&upperTimer,  &QTimer::timeout, this, &MainWindow::onUpperTick);
//connect(&saveTimer, &QTimer::timeout, this, &MainWindow::Save);
savePath = QStandardPaths::writableLocation(  //파일을 저장할 경로 생성
               QStandardPaths::DocumentsLocation)
           + "/robot_arm_state.txt";
```

이 부분은 시그널과 슬롯을 연결하는 부분입니다. 버튼 시그널과 해당 슬롯 메서드들을 연결하여 시그널이 발생되면 슬롯 메서드를 알맞게 호출되도록 명시적으로 작성한 것입니다. 그리고 타이머 연결을 하고 경로를 QString헤더파일의 유틸리티 함수를 사용하여 경로를 쉽게 구성하는 부분입니다.

```
// 자동 저장 주기 설정 2초마다 슬롯 함수 호출
connect(&saveTimer, &QTimer::timeout, this, &MainWindow::saveToTxt); //2초마다 자동 txt파일에
connect(ui->loadButton, &QPushButton::clicked, this, &MainWindow::loadFromTxt); //버튼 누르
saveTimer.start(2000);
```

그리고 타임 아웃에 따른 2초마다 자동 저장 connect, 그리고 begin 버튼과 불러오기 슬롯 메서드를 연결한 것입니다.

```
//spin Box에 대한 슬롯 함수 정의 입력 받은 각도 값으로 그만큼 회전
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);
}
```

armView를 통해 스핀박스 값이 바뀌면 각도값을 초기화하는 부분입니다.

```
//버튼 누르면 타이머가 시작되어 움직이게 되는데 타이머를 시작해주는 메서드
void MainWindow::toggleTimer(QTimer& t, int& dirVar, int newDir) {
    if (t.isActive() && dirVar == newDir) { t.stop(); return; } // 같은
    dirVar = newDir;
    t.start(intervalMs);
}
```

toggleTimer함수로 타이머를 작동시키며 같은 버튼을 눌렀을 때 타이머를 멈춤으로써 팔 움직이는 것을 멈추는 것을 구현한 부분입니다. 그래서 dirVar이 newDir값과 같은 즉 방향이 같은 값이 들어왔을 때 멈춘다는 것입니다.

```
// 버튼 눌렸을 때 호출되는 슬롯 메서드. ArmView를 직접 굴릴 방향 지정
void MainWindow::on_bottom_clicked()         { toggleTimer(bottomTimer, bottomDir, +1); }
void MainWindow::on_bottom_counter_clicked() { toggleTimer(bottomTimer, bottomDir, -1); }

void MainWindow::on_middle_clicked()         { toggleTimer(middleTimer, middleDir, +1); }
void MainWindow::on_middle_counter_clicked() { toggleTimer(middleTimer, middleDir, -1); }

void MainWindow::on_upper_clicked()          { toggleTimer(upperTimer,  upperDir,  +1); }
void MainWindow::on_upper_counter_clicked()  { toggleTimer(upperTimer,  upperDir,  -1); }

// ArmView 메서드 직접 호출. 타이머가 시작되어서 timeout이 되었을 때 호출되는 슬롯 메서드
void MainWindow::onBottomTick() { armView->stepBottom(bottomDir * 2); } //20ms당 2도 회전
void MainWindow::onMiddleTick() { armView->stepMiddle(middleDir * 2); }
void MainWindow::onUpperTick()  { armView->stepUpper(upperDir  * 2); }
```

그리고 버튼이 눌렀을 때 각각의 슬롯 메서드들이 호출되고 이것이 타이머 스타트하는 toggleTimer함수를 호출하여 각도까지 타이머에 따라 자동으로 제어되는 연결 부분 중 하나입니다. 시계방향일 때는 값을 1로 하고 반시계일 때는 -1로 방향을 잡아줬습니다.

```
// ArmView 메서드 직접 호출. 타이머가 시작되어서 timeout이 되었을 때 호출되는 슬롯 메서드
void MainWindow::onBottomTick() { armView->stepBottom(bottomDir * 2); } //20ms당 2도 회전
void MainWindow::onMiddleTick() { armView->stepMiddle(middleDir * 2); }
void MainWindow::onUpperTick()  { armView->stepUpper(upperDir  * 2); }
```

그리고 타임아웃이 되었을 때 이 슬롯 메서드들이 호출되는데 내부적으로 step~메서드들을 호출하면서 최종적으로 타이머에 따라 각도값을 초기화하는 부분입니다.

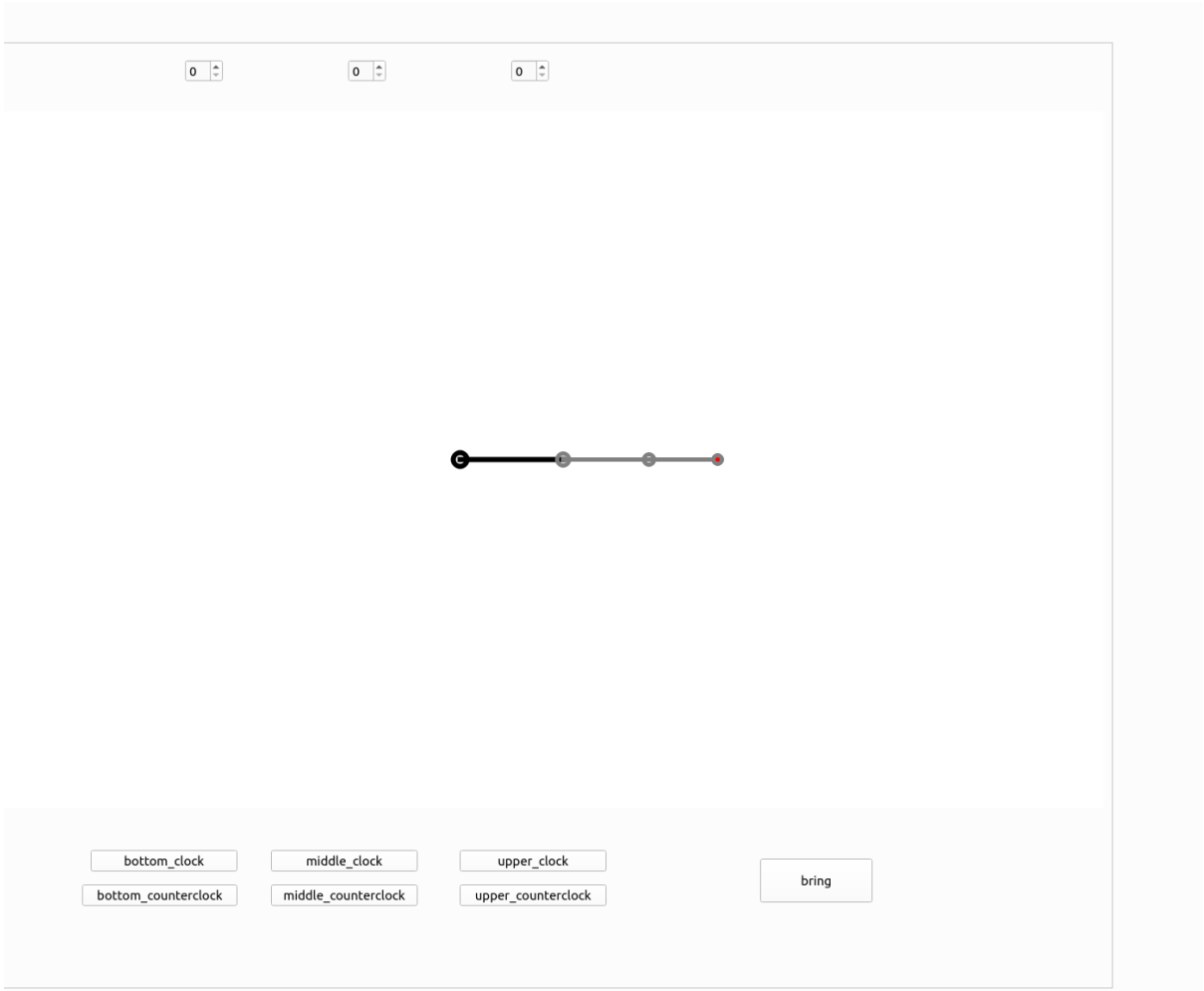
```
void MainWindow::saveToTxt() //txt파일에 각도, 방향 저장함
{
    QFile f(savePath);
    if(!f.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Save fail", "파일 열기 실패: " + savePath);
        return;
    }
    QTextStream out(&f);
    // 형식: a0 a1 a2 bottomDir middleDir upperDir
    out << armView->bottomAngle() << ' '
        << armView->middleAngle() << ' '
        << armView->upperAngle()  << ' '
        << bottomDir << ' ' << middleDir << ' ' << upperDir << '\n';
    f.close();
    // qDebug() << "Saved to" << savePath;
}
```

QFile헤더 파일을 사용하여 C언어에서의 파일 입출력과 유사하게 쉽게 파일 입출력을 하였습니다. if문으로 예외처리를 하고 예외가 아니라면 출력 스트림을 통해 파일에 로봇의 정보 값을 출력하고 있는 것입니다.

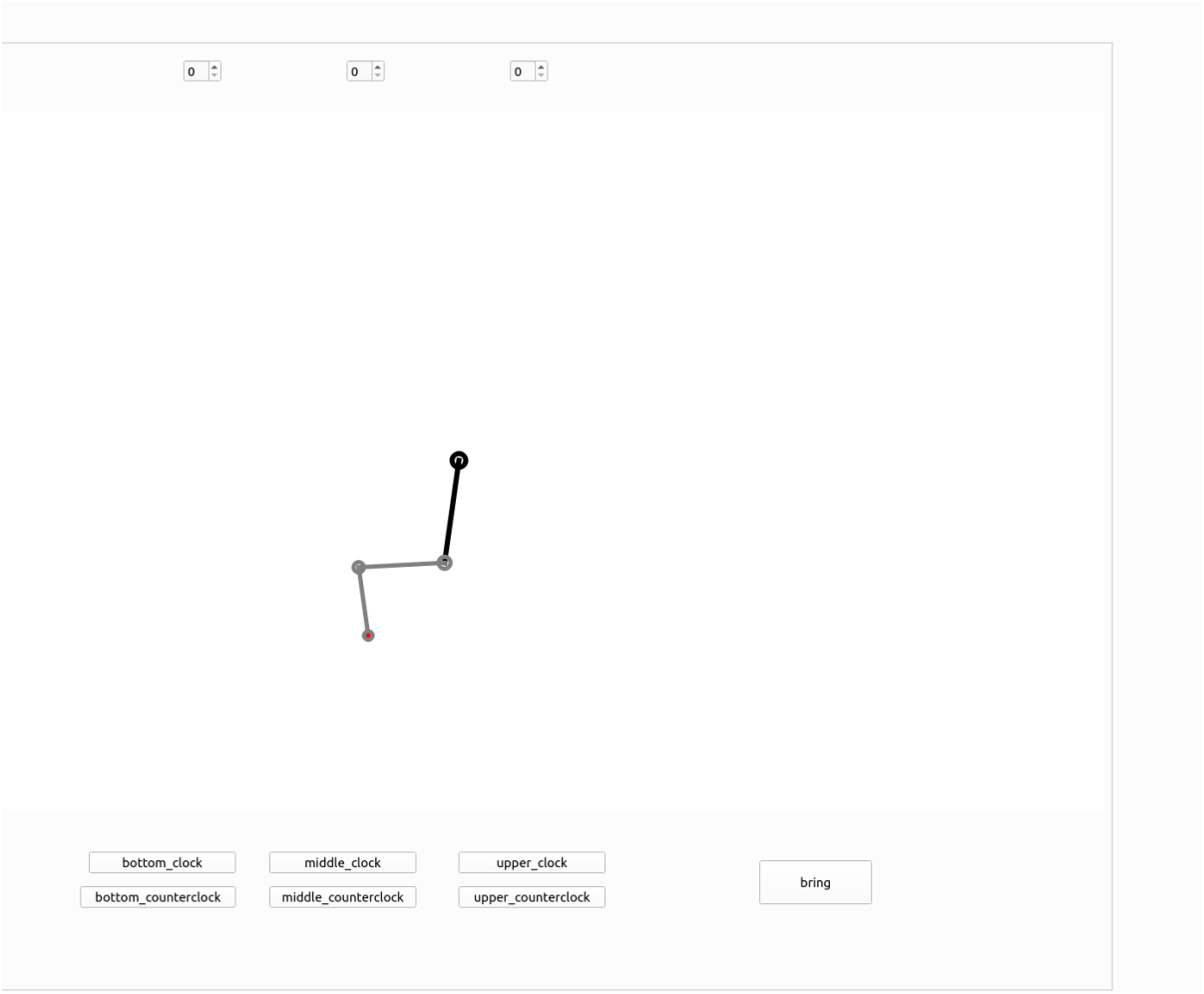
```
void MainWindow::loadFromTxt() //정보를 가져오는 함수. bring버튼 누르면 정보 가져와서
{
    QFile f(savePath);
    if(!f.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Load fail", "파일 열기 실패: " + savePath);
        return;
    }
    QTextStream in(&f);
    int a0, a1, a2, bdir, mdir, udir;
    in >> a0 >> a1 >> a2 >> bdir >> mdir >> udir;
    f.close();

    // 각도/방향 복원
    armView->setBottomAngle(a0);
    armView->setMiddleAngle(a1);
    armView->setUpperAngle(a2);
    bottomDir = bdir; middleDir = mdir; upperDir = udir;


    //되돌린 상태에서 정지
    bottomTimer.stop();
    middleTimer.stop();
    upperTimer.stop();

}
```

파일로부터 정보를 받아와서 이를 ArmView객체의 각도 정보와 방향 정보를 초기화하는 부분입니다. 그래서 저장된 파일의 상태대로 다시 로봇의 상태를 초기화하는 것입니다. 이번에는 입력 스트림을 통해 값을 받아오고 그 값으로 로봇의 상태를 메서드를 통해 초기화하였습니다.

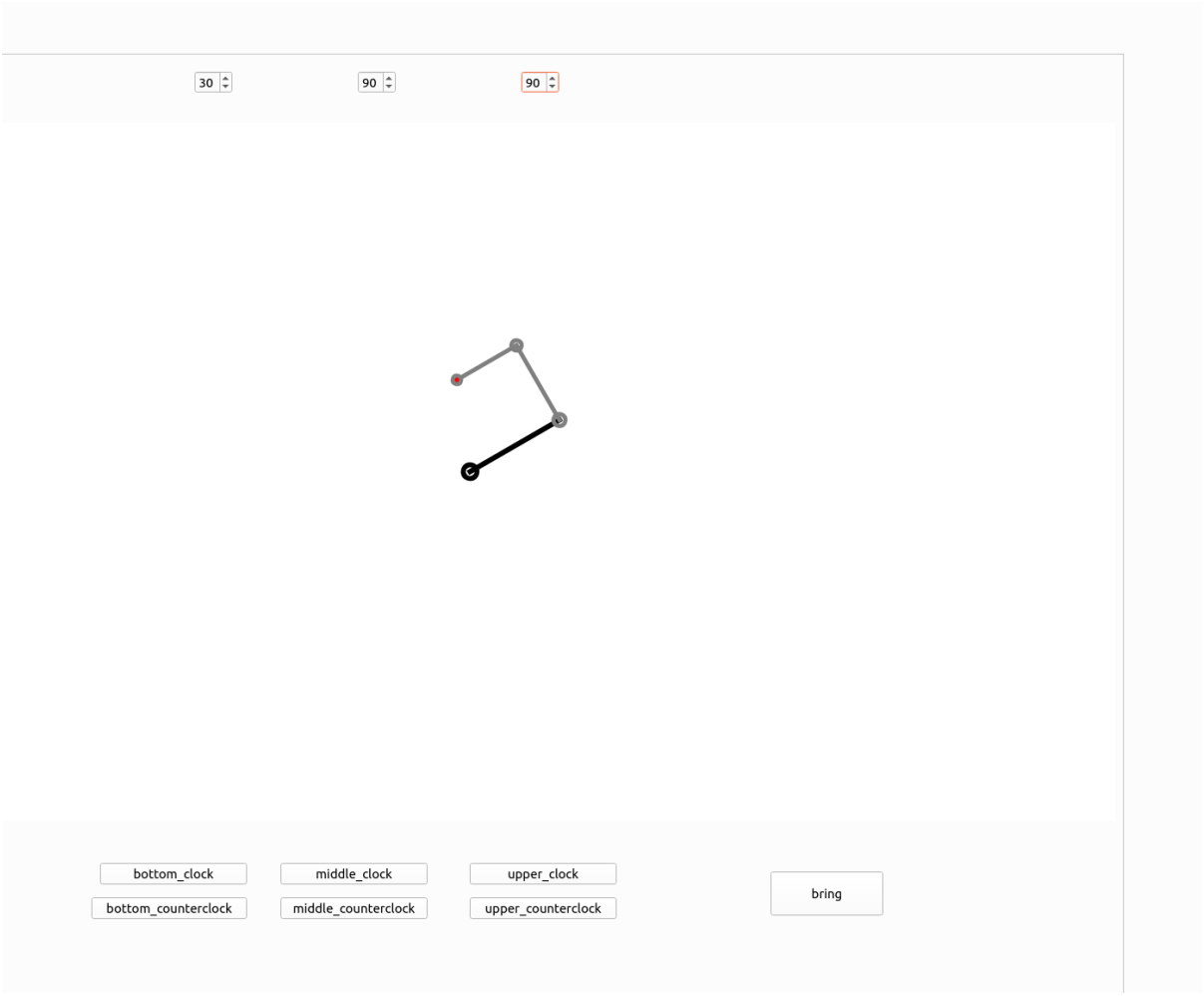
(4) 시연 사진



실행을 했을 때 이처럼 정지해있습니다.



현재 상황은 bottom_clock, middle_clock, upper_clock을 모두 누르고 임의의 시간 때 캡처한 것입니다. 이 와중에도 스핀 박스 값을 바꾸면 해당 관절의 각도가 그 각도로 다시 로봇팔이 초기화됩니다.



이번에는 버튼을 누르지 않고, 스핀 박스만으로 30, 90, 90각도로 값을 설정한 것입니다. 이 때 bottom 관련 관절이 30도 이고, 그 상태에서의 middle관련 관절이 90만큼 돌아가고, 또 upper관련 관절이 90도 돌아간 것을 볼 수 있습니다.