C++&Qt day2 hw1 결과 보고서

2025407012/로봇학부/송연우

목차

1. 구현 기능

1) 버튼 클릭 시 자동 회전

: 관절 하나당 버튼 2개를 할당해 버튼을 누르면 각각 시계방향, 반시계방향으로 회전하고, 다시 버튼을 누르면 회전을 멈추는 기능을 구현했습니다. 팔은 QPaint 기능를 통해 그렸고 자동으로 회전하는 기능은 QTimer를 사용해 버튼이 눌려 회전이 활성화되면 100ms마다 관절 각도를 자동 회전시키는 함수를 호출하도록 했습니다.

2) 다이얼 UI로 수동 회전 조작

: UI에 있는 다양한 기능 중 다이얼을 사용해 다이얼을 움직일 때마다 변하는 value값을 이용해 관절 각도를 변하게 하고 팔을 그리는 함수에서 이를 받아 사용하는 형식으로 구현했습니다.

3) 파일 입출력

: fstream 라이브러리에서 제공하는 파일 입출력 함수들을 사용해 mani_save.txt에 각각 관절의 현재 각도를 저장하고, 버튼이 클릭되었을 때 입/출력하도록 했습니다.

2. hpp 파일

다음은 작성한 헤더파일 코드입니다.

```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QPen>
5   #include <QPainter>
6   #include <QPaintEvent>
7   #include <QMainWindow>
8   #include <QTimer>
9
10  using namespace std;
11
12  QT_BEGIN_NAMESPACE
13  namespace Ui { class MainWindow; }
14  QT_END_NAMESPACE
```

먼저 필요한 라이브러리들을 include하고 CUI입출력을 간편히 하기 위해 namespace std를 사용했습니다. 밑의 namespace UI는 파일을 생성했을 때 자동실행된 부분입니다.

```
17  class MainWindow : public QMainWindow
18  {
19      Q_OBJECT
20
21  public:
22      MainWindow(QWidget *parent = nullptr);
23      ~MainWindow();
24      void paintEvent(QPaintEvent *e);
25      void onBtnPress();
26      void rotateJoint_CW(int joint_degree);
27      void rotateJoint_CCW(int joint_degree);
28      void rotate1();
29      void rotate2();
30      void rotate3();
```

MainWindow 클래스입니다. 원래는 따로 Manipulator 클래스를 만들어 MainWindow에서는 UI출력을, Manipulator에서는 관절 각도 조작을 담당하려고 했으나 변수를 쓰는 과정에서 문제가 발생해 MainWIndow 클래스의 public에 기능 구현 함수를 모두 생성하여 사용했습니다.

위의 두 함수는 각각 생성자, 소멸자이고 QPaintEvent를 사용하기 위한 paintEvent함수와 관절 자동 회전에 쓰이는 rotate1, 2, 3함수가 있습니다. 나머지는 선언했지만 사용하지는 않는 함수입니다.

```
32      private slots:
33
34          void on_dial_valueChanged(int value);
35
36          void on_dial_3_valueChanged(int value);
37
38          void on_dial_2_valueChanged(int value);
39
40
41          void on_pushButton_8_clicked();
42
43          void on_pushButton_7_clicked();
44
45          void on_pushButton_clicked();
46
47
48          void on_pushButton_2_clicked();
49
50          void on_pushButton_3_clicked();
51
52          void on_pushButton_4_clicked();
53
54          void on_pushButton_5_clicked();
55
56          void on_pushButton_6_clicked();
57
58      private:
59          Ui::MainWindow *ui;
60          QTimer *Timer1;
61          QTimer *Timer2;
62          QTimer *Timer3;
63          int joint1_angle = 0;
64          int _degree =0;
65          int joint_degree[3]={0};//회전각
66          int rotDirec[3]={0};
67
68      };
```

UI와 상호작용 시 실행되는 다이얼, 버튼 관련 함수들입니다. 다이얼 1, 2, 3은 각각 관절 1, 2, 3의 수동 조작을 담당하고 버튼 1~6은 관절 1, 2, 3의 자동 회전을 제어합니다. 버튼 7, 8은 각각 save, load 함수입니다.

밑부분의 private영역에서는 타이머와 회전각 설정, 그리고 자동 회전에서의 방향과 움직임 정보를 나타내는 rotDirec이 있습니다.

3. cpp 파일

다음은 작성한 소스코드입니다. main에서는 편집한 것이 없으므로 바로 mainwindow.cpp에 대해 설명하겠습니다.

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <iostream>
#include <QTimer>
#include <fstream>

using namespace std;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    Timer1 = new QTimer(this);
    Timer2 = new QTimer(this);
    Timer3 = new QTimer(this);
    Timer1->setInterval(100);
    Timer2->setInterval(100);
    Timer3->setInterval(100);

    connect(Timer1, &QTimer::timeout, this, &MainWindow::rotate1);
    connect(Timer2, &QTimer::timeout, this, &MainWindow::rotate2);
    connect(Timer3, &QTimer::timeout, this, &MainWindow::rotate3);
}

MainWindow::~MainWindow()
{
    delete ui;
}

```

생성자에서의 설정입니다. 타이머에 동적 할당한 뒤에 100ms간격으로 시간을 재게끔 설정합니다. 그리고 자동 회전을 구현하기 위해 connect()로 타이머의 100ms간격마다 rotate함수를 호출하게끔 연결합니다. 소멸자에서는 생성자에서 할당한 메모리를 삭제합니다. Timer을 처음 만들 때 this 코드를 추가했으므로 ui에서 따로 메모리를 삭제하지 않았습니다.

```
32      void MainWindow::paintEvent(QPaintEvent *e)
33      {
34          QPainter painter(this);
35              painter.setRenderHint(QPainter::Antialiasing);
36
37              QPen pen1(Qt::black, 10);
38              painter.setPen(pen1);
39              painter.translate(260, 320);        //화면 중간쯤
40              painter.rotate(joint_degree[0]);       // 관절 1 회전
41              painter.drawLine(0, 0, 100, 0);     //팔 그리기
42
43              // 2번째 관절
44              QPen pen2(Qt::red, 7);
45              painter.setPen(pen2);
46              painter.translate(100, 0);          // 첫 번째 팔 끝
47              painter.rotate(joint_degree[1]);       // 관절 2 회전
48              painter.drawLine(0, 0, 80, 0);
49
50              QPen pen3(Qt::blue, 4);
51              painter.setPen(pen3);
52              painter.translate(80, 0);
53              painter.rotate(joint_degree[2]);
54              painter.drawLine(0, 0, 50, 0);
55
56              QPen pen4(Qt::green, 4); //그리퍼 부분 표시
57              painter.setPen(pen4);
58              painter.translate(50,0);
59
60              painter.drawLine(0, 10, 0, 0);
61              painter.translate(0, 10);
62              painter.drawLine(0, 0, 7, 0);
63              painter.translate(0, -15);
64
65              painter.drawLine(0, 0, 0, 10);
66              painter.translate(0, -5);
67              painter.drawLine(0, 0, 7, 0);
68
```

로봇팔을 그리기 위한 함수입니다. 관절에 붙어 있는 부분과 로봇팔임을 알리기 위한 그리퍼 부분들 각각은 색과 펜 굵기를 다르게 설정해 구별했습니다. 계속해서 갱신되는 회전각을 적용하기 위해 rotate()를 사용해 관절 회전 각도인 joint_degree[]를 받아와 사용했습니다.

```
84  void MainWindow::on_dial_valueChanged(int value)
85  {
86      cout<<"Joint 1 moved: "<<value<<endl;
87      joint_degree[0]=value*4;
88      repaint();
89  }
90
```

수동 조작에서 사용되는 다이얼에 대한 슬롯 함수입니다. value값을 반영하기 위해 joint_degree에 4를 곱한 값(value의 범위가 0~99)을 joint_degree에 저장하고, paintEvent()함수를 즉각 재호출하여 회전된 로봇팔을 그립니다. 나머지 다이얼 2, 3도 같게 설정했습니다.

```
108 void MainWindow::on_pushButton_7_clicked()
109 {
110     //txt load
111     ifstream load;
112
113     load.open("mani_save.txt");
114
115     load>>joint_degree[0];
116     load>>joint_degree[1];
117     load>>joint_degree[2];
118    load.close();
119    repaint();
120 }
121
122 void MainWindow::on_pushButton_8_clicked()
123 {
124     //txt save
125     ofstream save;
126
127     save.open("mani_save.txt");
128
129     save<<joint_degree[0]<<"\n";
130     save<<joint_degree[1]<<"\n";
131     save<<joint_degree[2]<<"\n";
132    save<<endl;
133    save.close();
134 }
```

Save, load 버튼이 클릭되었을 시의 슬롯 함수입니다. 각각 파일 출력/입력을 실행하며, ifstream, ofstream으로 객체를 생성하고 mani_save.txt 파일을 열어 관절 각도값들을 불러오고 저장합니다.

```
136  void MainWindow::on_pushButton_clicked()
137  {
138      if(Timer1->isActive() && rotDirec[0] ==1){
139          Timer1->stop();
140          rotDirec[0]=0;
141      }else {
142          rotDirec[0]=1;
143          if(!Timer1->isActive())Timer1->start();
144      }
145  }
146
147
148  void MainWindow::on_pushButton_2_clicked()
149  {
150      if(Timer1->isActive() && rotDirec[0] ==-1){
151          Timer1->stop();
152          rotDirec[0]=0;
153      }else {
154          rotDirec[0]=-1;
155          if(!Timer1->isActive())Timer1->start();
156      }
157  }
```

관절 하나에 할당된 두 개의 CW, CCW 버튼을 클릭했을 때의 슬롯 함수입니다.

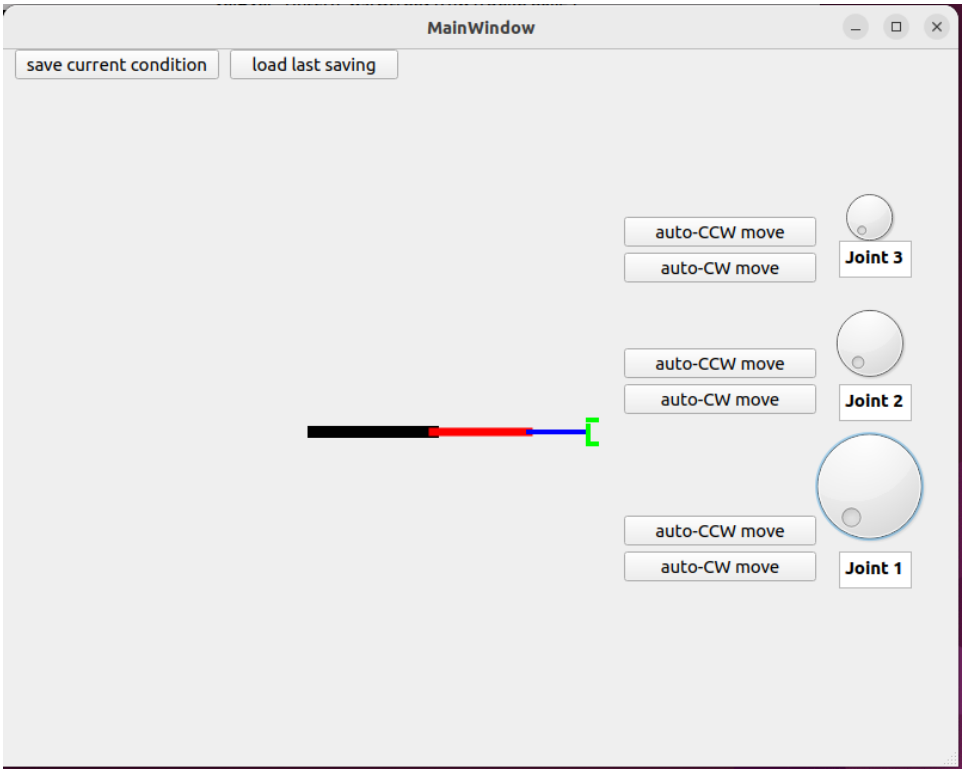
CW버튼이 클릭되었을 때, 타이머가 활성화되어 있고 회전 방향이 0이 아니라 1(시계방향)일 때는 타이머를 멈추고, 회전 변수를 0으로 바꿉니다. 회전하지 않는 상태였다면 회전방향 변수를 1(시계방향)으로 설정하고 타이머가 비활성화 되어있으면 다시 시작하도록 했습니다.

CCW버튼이 클릭되면 위 함수와 동일한 구성에서 회전 방향만 -1(반시계방향)으로 설정합니다. 나머지 버튼 3, 4, 5, 6또한 동일하게 구성했습니다.

```
207  void MainWindow::rotate1(){
208      joint_degree[0] = (joint_degree[0] + rotDirec[0]*5 + 360) % 360;
209      repaint();
210  }
```

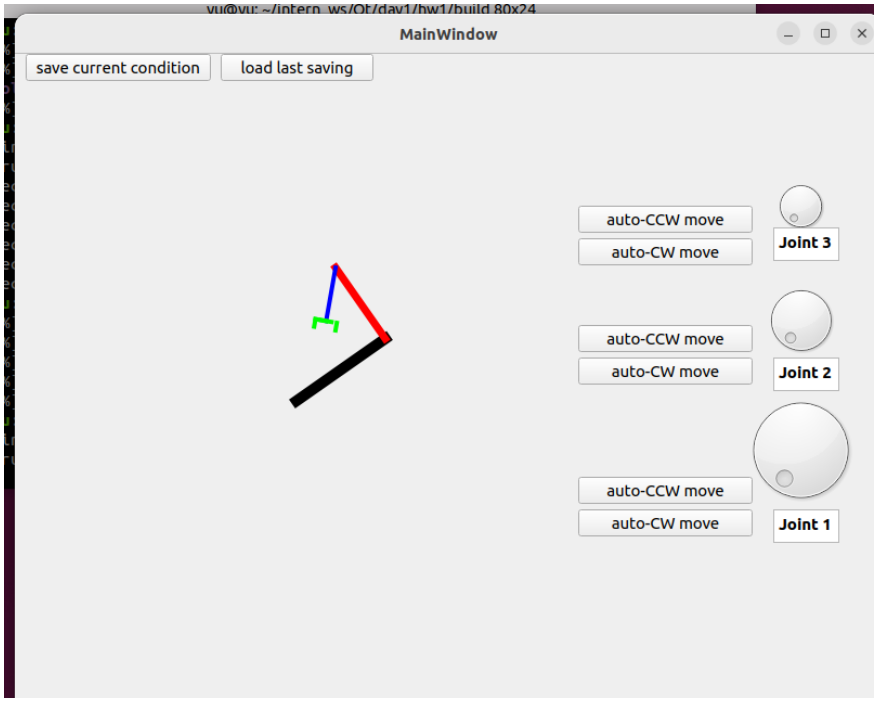
자동 회전에 쓰이는 rotate 함수입니다. 원래 관절 각도에 회전 방향*5로 +-5도를 더해주도, 이를 360으로 나누어 회전각의 값이 360도를 초과하지 않도록 했습니다. 그리고 바뀐 관절 각도를 반영하기 위해 repaint()로 paintEvent함수를 즉각 재호출해 팔을 다시 그리게 했습니다.

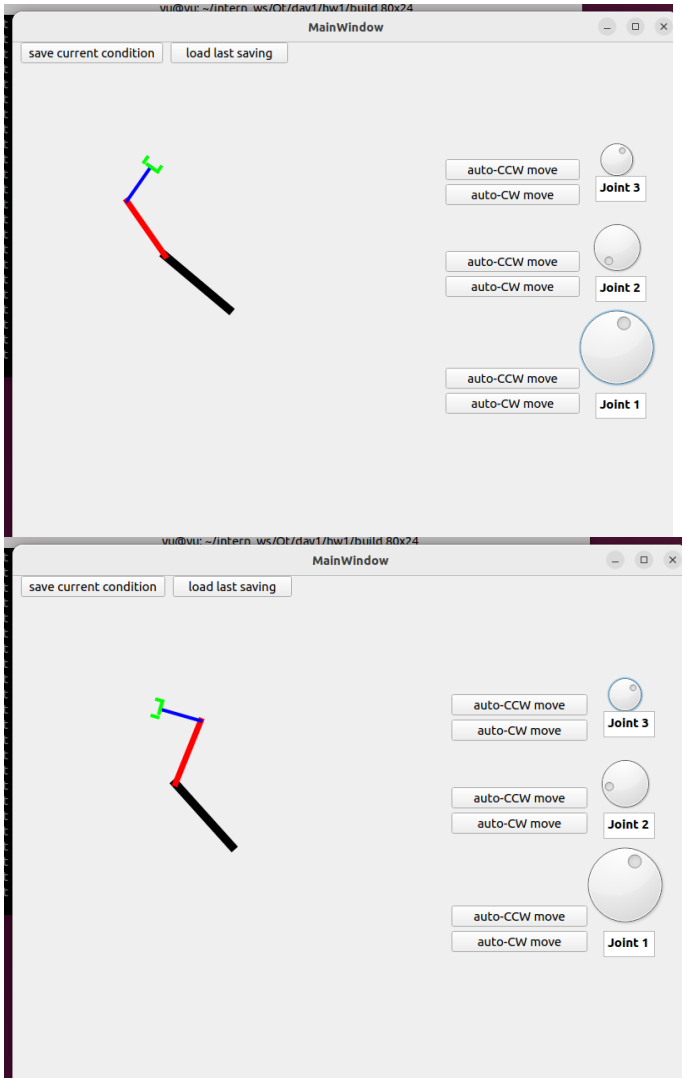
4. 실행 결과



프로그램을 빌드해 실행한 화면입니다. 맨 위 상단에 save, load버튼이 있고 다이얼로 관절 각도를 조절합니다. CCW, CW버튼으로 각 관절을 자동으로 회전하게 할 수 있습니다.

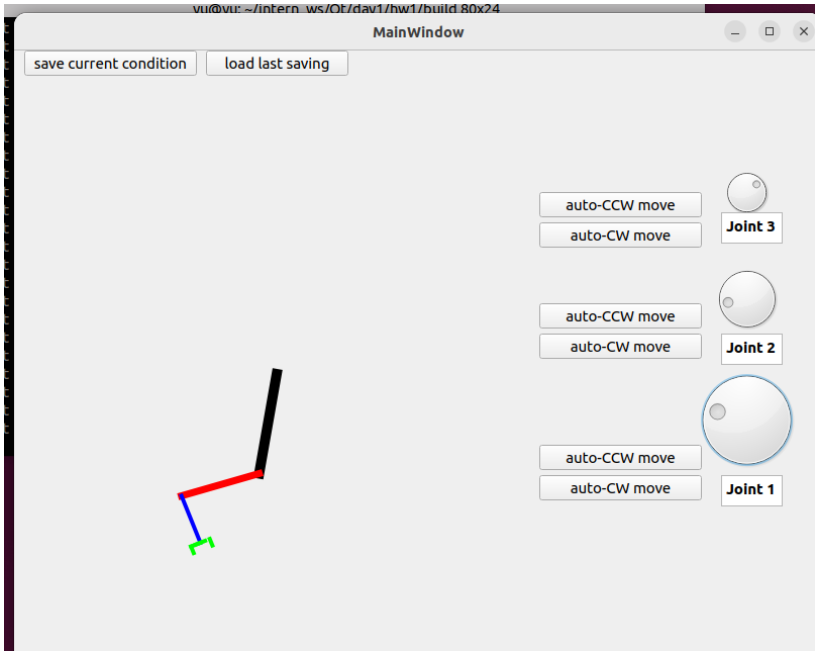
먼저 자동 회전입니다. 모두 CW버튼을 누른 상태입니다. 다이얼을 건드리지 않아도 자동으로 움직입니다.



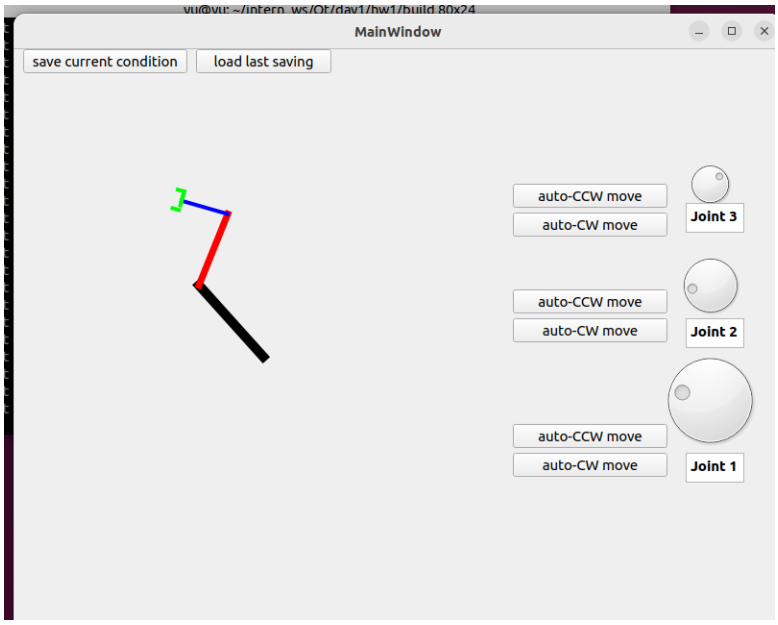


다이얼을 조정하면 수동으로도 움직입니다.

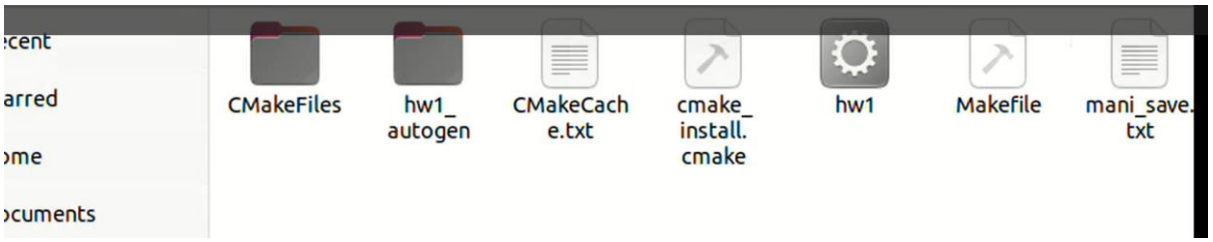
그리고 save버튼을 클릭하면 txt파일에 현재 각도가 저장되고,



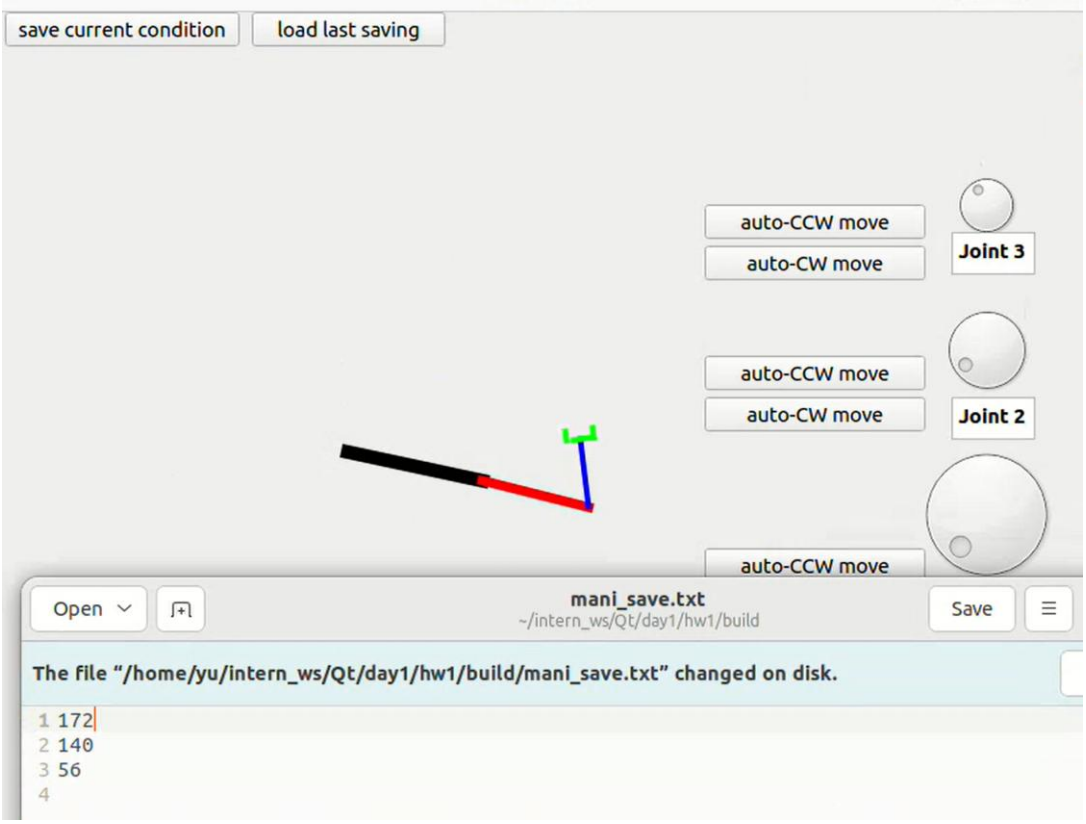
다른 상태에서 load 버튼을 눌렀을 때



아까 전의 상태로 돌아옵니다.



가장 오른쪽에 있는 것이 mani_save.txt 파일입니다.



값이 저장됨을 알 수 있습니다.