로봇학 실험4

과 목 명	로봇제어
담당교수	조황 교수님
제 출 일	2022.12.21.
학 과	로봇학부
	장홍기2018741053
	강상호2018741047

1.서론

이번 텀 프로젝트는 모터를 제어하여 원하는 위치, 속도, 전류값을 출력하는 것이다.

원하는 출력값들을 얻기 위해서 제어기를 사용하여 출력값을 얻는 것이 이번 프로젝트의 핵심이라고 생각한다. 사용된 제어기는 PD제어기(위치제어기), PI제어기(속도제어기, 전류제어기)이고 matlab simulink에서 설계했던 cascade 구조를 atmel studio에서 구현해 보았다.

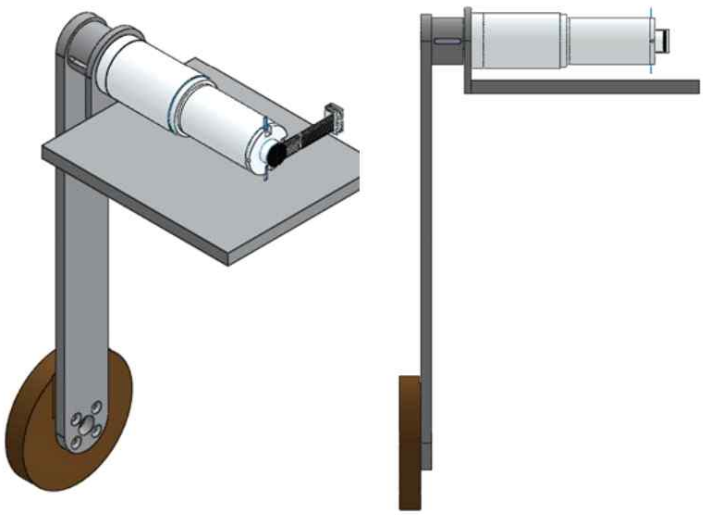
앞서 수업 시간에 배웠던 모터에 관한 기본적인 지식과, 구현한 시뮬레이션 환경을 설명하겠다.

2.시뮬레이션

가. 시스템모델링

(1)Geared motor

-기어는 부하를 효과적으로 구동하기 위해 전동기의 토크나 속도를 변환하는 데 사용한다. 시스템 의 모델 개요는 다음 사진과 같다.



이것은 1link 기어드모터로 구성은 dc모터와 아래의 부하 요소들로 구성되어 있다.

Bar Plate

M: 175g 길이3cm 높이25mm 로 구성되어 있다.

Circular Plate

M: 349g R: 50mm 로 구성되어 있다.

```
bar_m = 0.175;
bar_l = 0.3;
bar_h = 0.025;

circular_m = 0.340;
circular_r = 0.05;
circular_l = 0.3;
```

matlab코드에서 구현한 모습

$$T = (J_M + \frac{1}{\alpha}\left(\frac{N_1}{N_2}\right)^2 J_G)\ddot{\theta}_M + (B_M + \frac{1}{\alpha}\left(\frac{N_1}{N_2}\right)^2 B_G)\dot{\theta}_M$$

수식에서 볼 수 있듯이 기어비 N1:N2 비율이 점점 더 커지게 되면 기어 측 수식이 0에 가까워 지므로 기어가 전체 모터 토크에 영향을 주지 않게 되므로 모터 입장에선 기어비에 대한 영향을 무시할 수 있다.

즉, 우리는 모터에 기어를 달면 토크가 상승하고 속도가 줄어든다는 것을 알 수 있다.

모터는 전기시스템과 기계시스템이 더해져 구성된다는 점과 위에서 본 부하를 이용해, 부하 측의 기계 방정식에 기어 측 방정식으로부터 얻어진 기어 간의 각 속도 관계를 적용함으로써 운동방정식을 모터 측에서 기어를 통해 본 전체적인 기계방정식을 얻을 수 있다.

$$T_M = J_M \frac{dw_M}{dt} + B_M w_M$$

모터 측 기계방정식

$$T_L = J_L \frac{dw_L}{dt} + B_L w_L$$

부하 측 기계방정식

$$v = r_2 w_L = r_1 w_M \rightarrow N_2 w_L = N_1 w_M \quad \cdots\cdots \quad w_L = \frac{N_1}{N_2} w_M$$

기어 간 적용되는 방정식

$$P = T_L w_L = \alpha T_M w_M \quad \cdots\cdots \quad T_L = \alpha T_M \frac{w_M}{w_L}$$

기어 효율에 관한 관계식 적용

$$T_M = \frac{1}{\alpha}(\frac{N_1}{N_2})^2 J_L \frac{dw_M}{dt} + \frac{1}{\alpha} B_L \frac{N_1}{N_2} w_M \text{ - 부하 측 운동 방정식}$$

$$T_M = (J_M + \frac{1}{\alpha}(\frac{N_1}{N_2})^2 J_L)\frac{dw_M}{dt} + (b_M + \frac{1}{\alpha} B_L \frac{N_1}{N_2}) w_M \text{ - 모터 측 기계 방정식}$$

$$T_M = K_t i_a \qquad J_{eq} = J_M + \frac{1}{\alpha}(\frac{N_1}{N_2})^2 J_L \qquad b_{eq} = b_M + \frac{1}{\alpha} B_L \frac{N_1}{N_2}$$

```
j_load = j_bar + j_circular;
b_load = j_load /t_constant;

j_eq = jmg + (1/a)*(N1/N2)^2*j_load; %% 부하가 더해져서 나오는 공식
b_eq = j_eq/t_constant; %% 부하가 더해진 마찰력 계수
```

Matlab 상에서 적용한 모습

위 결과값들을 라플라스 변환을 통해

최종 기계방정식과 전기 방정식을 구할 수 있다.

$$K_T i_a = J_{eq} \frac{dw_M}{dt} + b_{eq} w_M \text{ - 기계 방정식}$$

$$L_a \frac{di_a}{dt} + R_a i_a = V_a - K_e w_M \text{ - 전기 방정식}$$

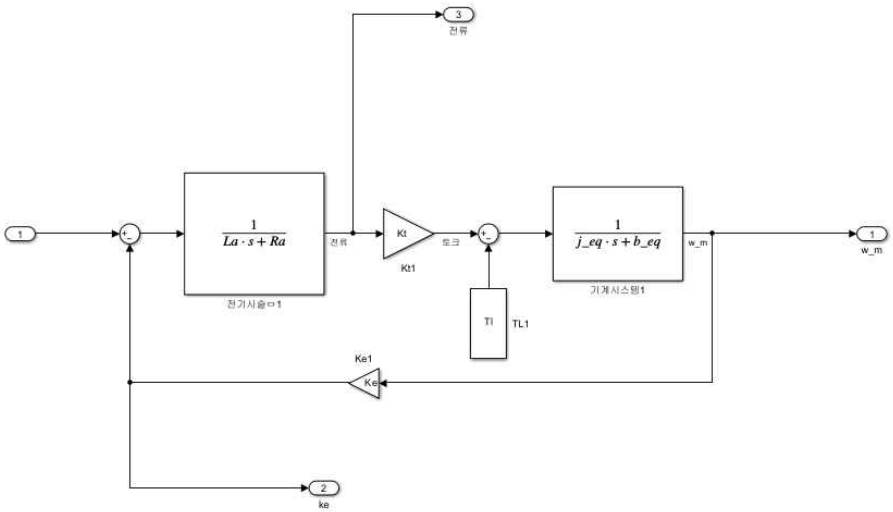
$$K_t I_a(s) = J_{eq} s w_M(s) + b_{eq} w_M(s) \text{ - 라플라스 변환}$$

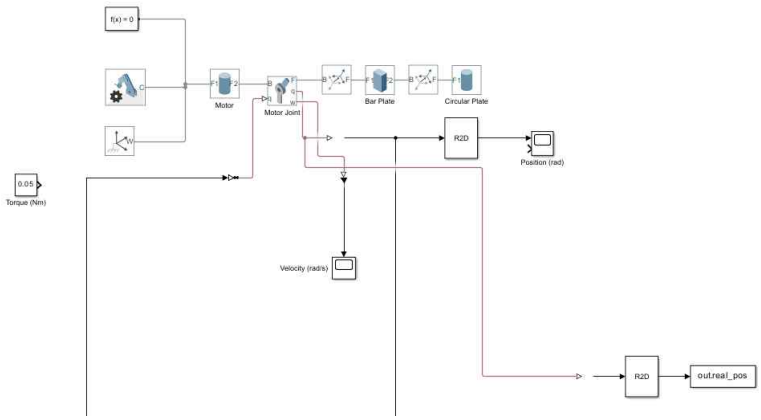
$$L_a s I_a(s) + R_a I_a(s) = V_a(s) - K_e w_M(s) \text{ - 라플라스 변환}$$

$$\rightarrow \frac{w_M(s)}{V_a(s)} = \frac{K_T}{(J_{eq} s + b_{eq})(L_a s + R_a) + K_e K_T}$$

나. 시뮬레이션 환경 구성

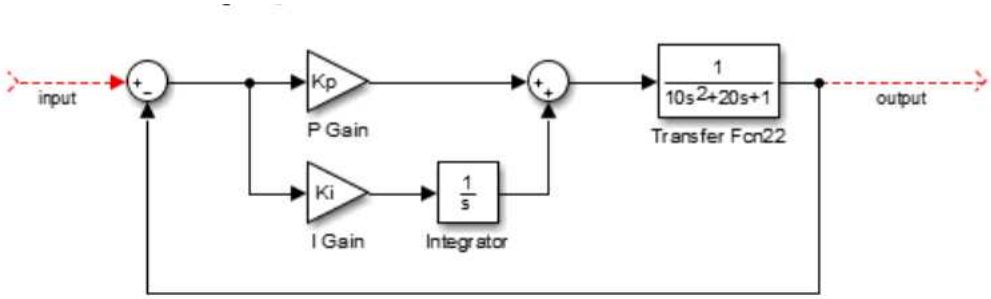
아래의 서브 시스템과 기어드 모터의 simulink model을 다음과 같이 구성하였다.





다. 위치-속도-토크 제어기 구성

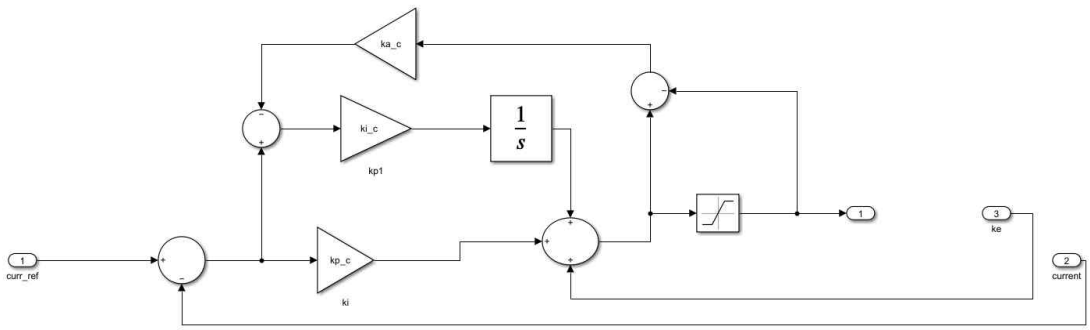
모터의 위치는 속도를 적분한 값이다. 그러므로 위치를 제어하기 위해선, 모터의 속도를 제어해야 한다. 그러므로 위치나 속도 제어가 요구되는 시스템에서 궁극적으로 제어해야 할 대상은 motor의 토크다. 토크는 전기자 전류에 비례하므로 전류를 제어하여 토크를 제어할 수 있다. 그러므로 위치, 속도 제어 시스템에서는 전류제어가 필수적이다.



위 사진은 pi제어기이다. pi제어기는 정상상태 오차를 제거해 오버슈트가 증가한다.

전류제어기, 속도제어기는 정확한 전류, 속도값을 출력하기 위해 pi제어기를 사용한다.

(1)전류제어기(pi)



위 사진은 simulink로 구현한 전류제어기의 cascade 구성이다. 전류제어기는 모터의 전류오차를 입력 받아 전압 지령을 출력한다. 전류제어기를 구성할 때 리미터를 넣어 준다. 그 이유는 모터의 정격 전압을 넘어가는 전압을 인가할 수 없기 때문이다. 그리고 리미터를 걸게되면 오차가 누적되어 지연 처리 되기 때문에, anti-windup을 넣어주어 오차 누적을 방지하는 것이다.

전류제어기 전달함수의 전향보상을 통해 그림2.17과 같은 openloop 시스템으로 볼 수 있다.

여기서 전달함수를 pole-zero cancelatoin(pi제어기의 영점이 시스템의 극점

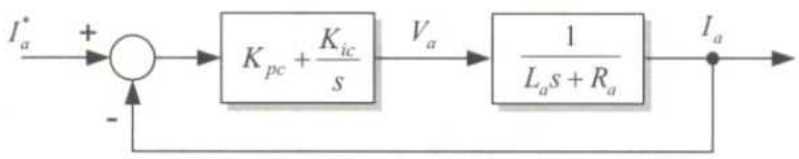


그림 2.17 비례 적분 전류제어기

을 상쇄하도록 설계) 하여 Kp와 Ki를 구할 수 있다.

$$G_c^c(s)=\frac{G_c^o(s)}{1+G_c^o(s)}=\frac{\frac{1}{\frac{L_a}{K_p}s}}{1+\frac{1}{\frac{L_a}{K_p}s}}=\frac{\frac{K_p}{L_a}}{s+\frac{K_p}{L_a}}=\frac{\omega_{cc}}{s+\omega_{cc}}$$

$$K_p=L_a\omega_{cc}$$

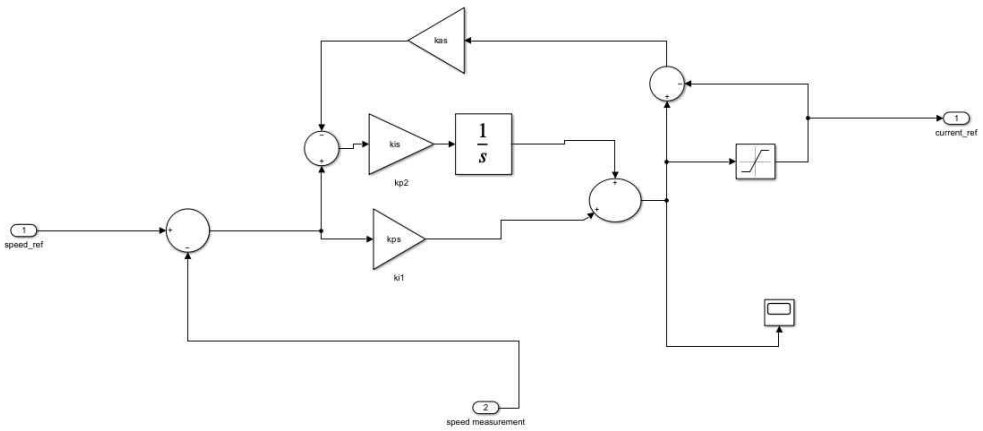
그러므로 $K_i=\frac{R_a}{L}K_p=R_a\omega_{cc}$ 이다.

$$K_a=\frac{1}{K_p}$$ anti-windup은 대게 다음과 같이 사용한다.

```
%% curr control
fcc = 200;
wcc = 2*PI*fcc;
kp_c = La * wcc;
ki_c = Ra * wcc;
ka_c = 1/kp_c;
```

matlab 코드 상에서 gain을 구현하였다.

(2)속도제어기(pi)



위 사진은 simulink로 구현한 속도제어기의 cascade 구성이다. 속도 제어기는 모터 회전속도의 오차를 입력받아 전류 지령을 출력한다. 전류를 제한하고 모터의 전류에도 제한이 있기 때문에 anti-windup을 사용한다. 부하측의 모터 속도를 제어하는 것이 목표이다.

속도제어기 gain을 구하기 위해 전류제어기와 동일한 방식으로 속도제어기의 openloop 전달함수를 구해

$$\frac{K_i}{K_p} = \frac{B}{J}$$

$$\omega_{cs} = \frac{\omega_{cc}}{10}$$

$$\mathrm{K_p} = \frac{J\omega_{cs}}{K_T},\ \mathrm{K_i} = \frac{B\omega_{cs}}{K_T}$$

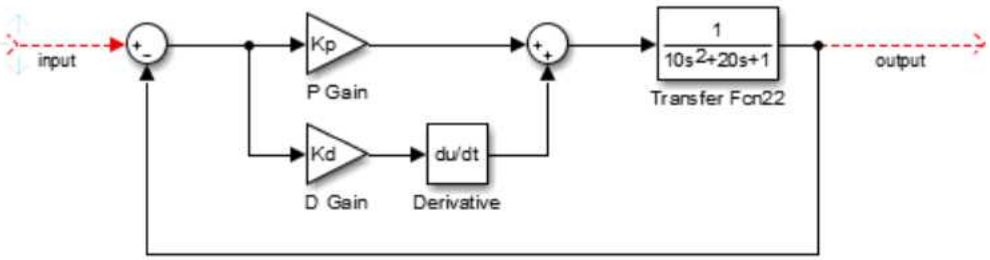
와 를 이용하여 다음과 같은 Kp와 Ki값을 구하였다. 여기서 wcs 속도 제어기의 대역폭은 전류 제어기의 대역폭보다 충분히 작게 잡아야 하기 때문에 1/10 한 것이다.

$$K_a = \frac{1}{K_p} \quad (K_p\text{: 속도제어기의 P gain})$$

속도 제어기의 anti windup은 다음과 같다.

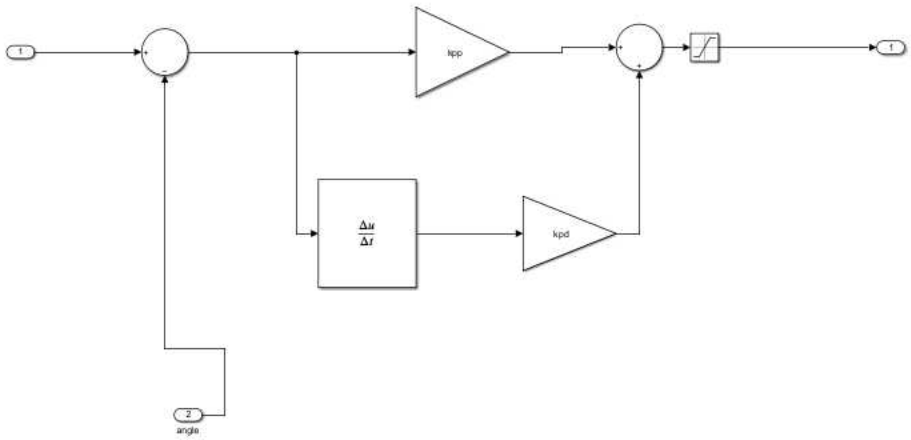
```
%% speed control
wcs = wcc/10;
kps = (1/n) * j_eq * wcs /Kt;
kis = (1/n) * b_eq * wcs /Kt;
kas = 1/kps;
```

matlab 코드 상에서 gain을 구현하였다.



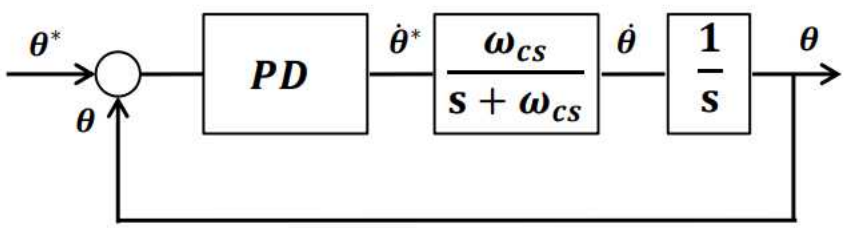
위 사진은 pd제어기이다. pd제어기는 오버슈트를 줄이고 외란에 강하지만 정상상태 오차가 존재한다. 위치제어기는 오차가 있더라도 안정적인 위치제어를 위해 Pd 제어기를 사용한다.

(3)위치 제어기(PD)



위 사진은 simulink로 구현한 위치제어기의 cascade 구성이다. 위치 제어기는 모터 위치의 오차를 입력받아 속도 지령을 출력한다. pd제어기를 사용해 속도를 제한 할 수 있다. 부하 측의 위치(각도)를 제어하는 것이 목표이다.

위치제어기 gain을 구하기 위해 위치 제어기의 open loop 전달함수를 다음과 같다.



여기서 $\frac{\omega_{cs}}{s+\omega_{cs}}$ 속도 closeloop을 openloop형태로 바꾼 것이다.

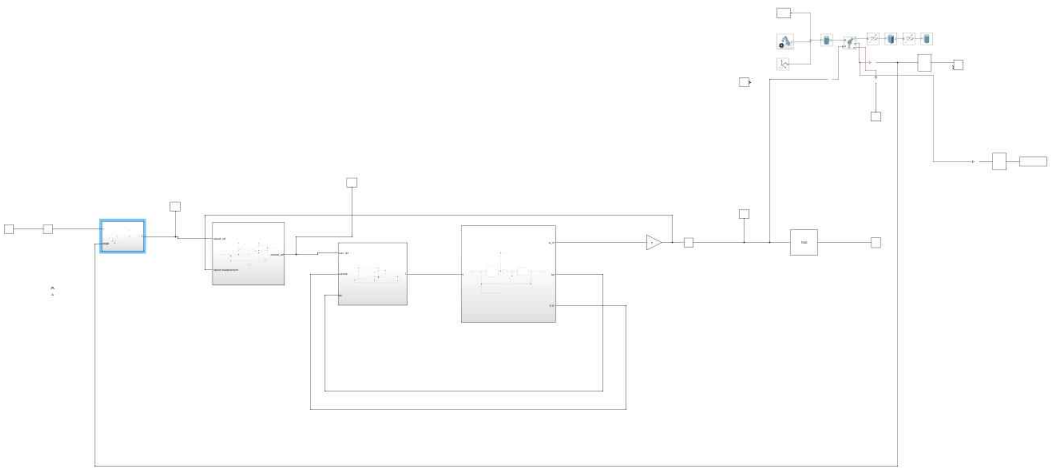
$$G_p^o = (K_p + K_d s)\left(\frac{\omega_{cs}}{s+\omega_{cs}}\right)\frac{1}{s} = \frac{K_d(s+\frac{K_p}{K_d})}{s}\left(\frac{\omega_{cs}}{s+\omega_{cs}}\right)$$
$$= \frac{K_d\omega_{cs}}{s}$$

위치제어기의 openloop전달함수를 통해 $K_d = \frac{\omega_{cp}}{\omega_{cs}}, K_p = \omega_{cp}$ 다음 과 같은 Kd Kp 게인값을 구하였다. 여기서 $\omega_{cp} = \frac{\omega_{cs}}{10}$ 위치 제어기 대역폭은 속도 제어기 대역폭보다 충분히 작게 잡는다.

```
%%  position control

wcp = wcs/10;
kpp = wcp;
kpd = wcp/wcs;
time_constant_c = 3/wcc;
time_constant_v = 3/wcs;
time_constant_p = 3/wcp;
```

matlab 코드 상에서 gain을 구현하였다.

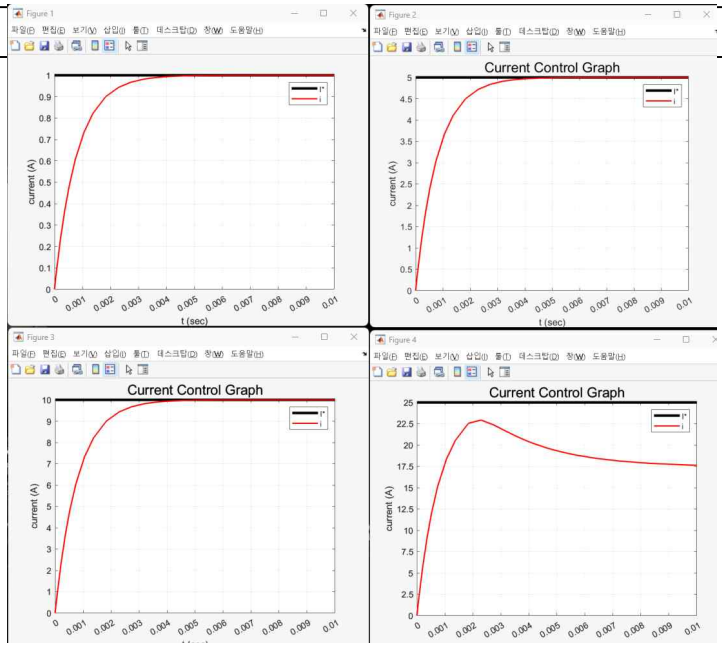
전체적인 전류,속도,위치 cascade 제어기 simulink model 이다.



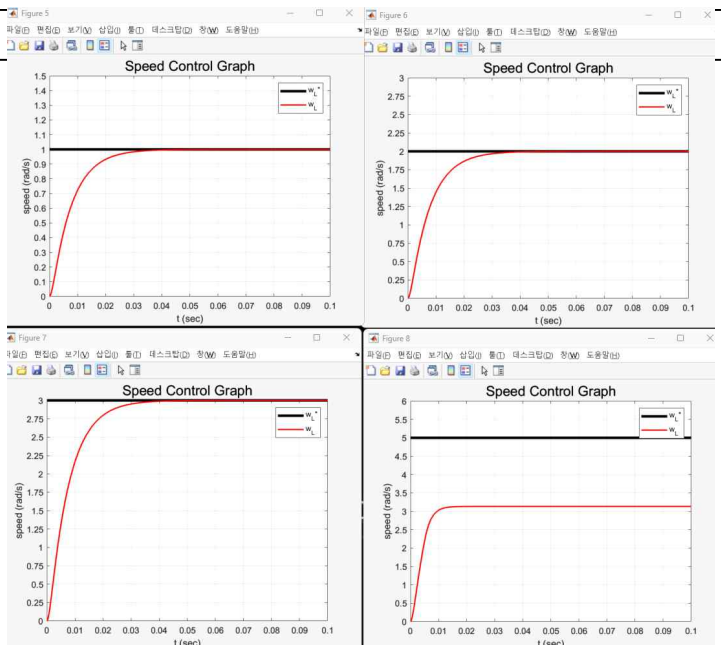
오른쪽부터 순서대로 기어드 dc모터, 서브시스템, 전류제어기, 속도제어기,

위치제어기 순이다.

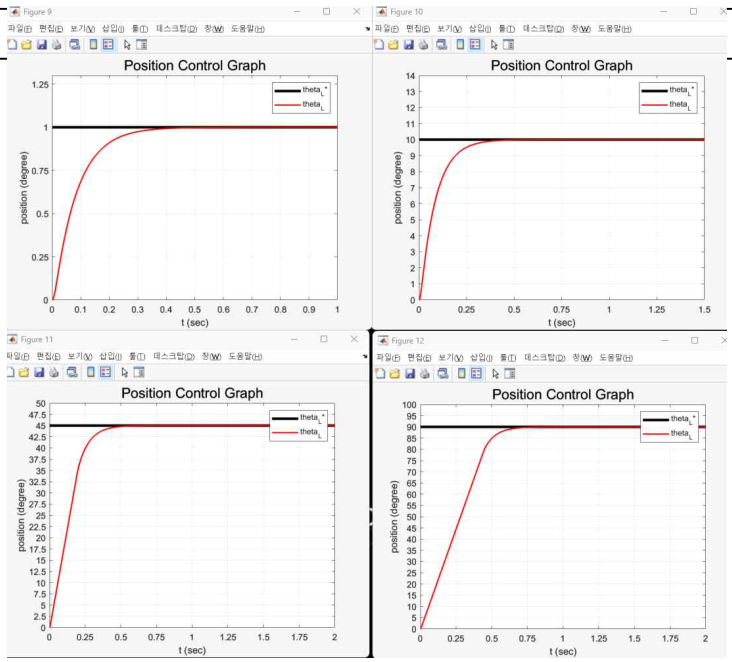
라. 시뮬레이션 결과



전류제어기를 설계하고 전류를 1[A]인가했을 때 응답성을 확인하고 입력전류를 5,10,25[A]로 변화 시키면서 결과값을 확인했다. (1/3wcc) 시간인 0.003 에서25A 일때를 제외하고 95%이상 응답하는 것을 확인했다.

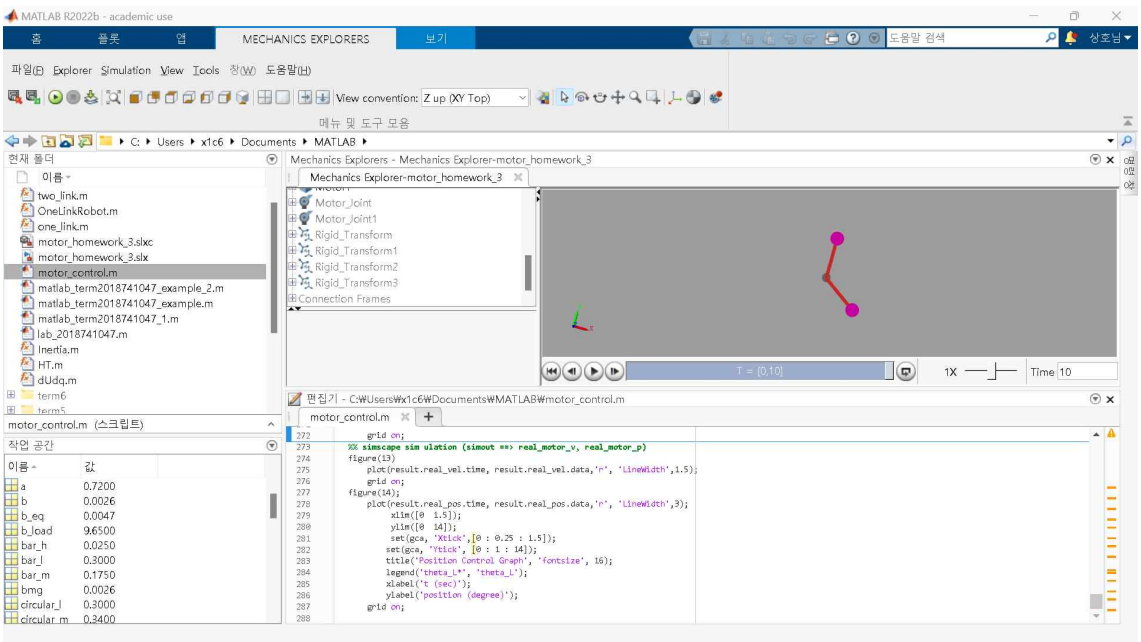


속도제어기를 설계하고 속도를1[rad/s] 인가했을 때 응답성 확인하고 입력속도를 2,3,5[rad/s]로 변화 시키면서 결과값을 확인하였다. (1/3wcs) 시간인 0.003 에서 5v 일때를 제외하고 95%이상 응답하는 것을 확인했다.



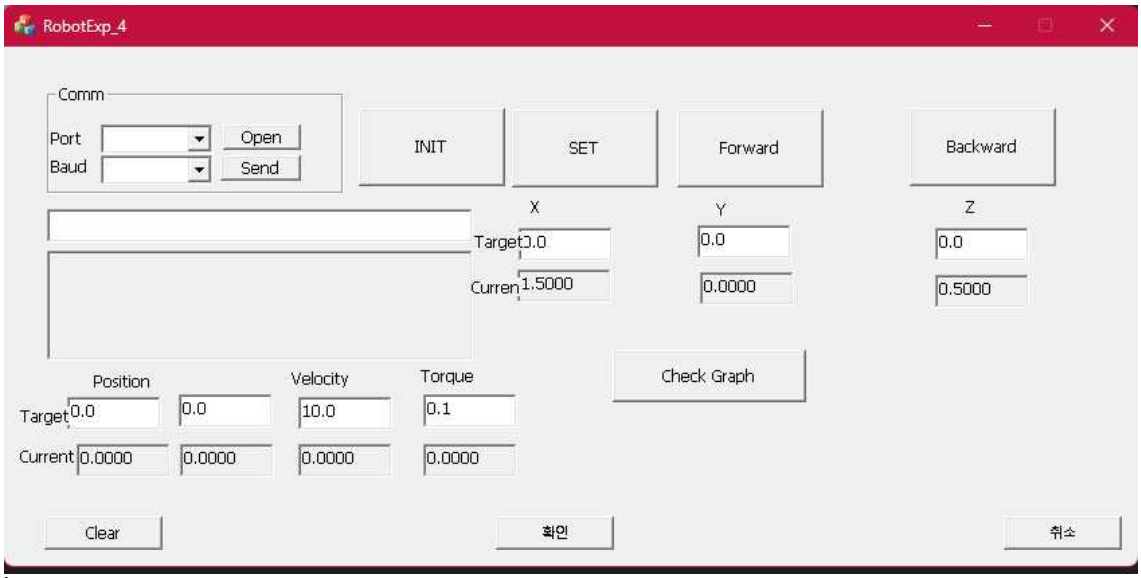
위치제어기를 설계하고 각도를 1[degree]인가했을때 응답성을 확인하고 입력각도를 10,45,90[degree] 로 변화 시키면서 결과값을 확인하였다. (1/3wcp) 시간인 0.003 안에 95%이상 도달하는 것을 90degree 일때 제외하고 모두 잘 응답하는 것을 확인했다.

전체적인 simulation 수행이다. Geared motor가 10초간 제어 되는 것을 확인하였다.(시뮬레이션 영상 첨부)



3. 실험결과

가. 실험 환경구성



Ui

우리는 visual studio 에서 ode 시뮬레이션 환경을 설정한다. vs코드상에서 ui를 구현하고pid제어를 통한 dc모터의 위치, 속도, 전류 제어를 하고자 한다. 그리고 vs mfc에는 간단한 그래프를 그려주는 컨드롤이 없기 때문에, 직접 구현해 주어야한다.

```
// #1 각도2개 입력받아 pdpos[0~2]에 출력
void CRobotExp_4Dlg::SolveForwardKinematics(double dAngle, double dAngle2, double* pdPos)
{

    pdPos[0] = cos(dAngle) * (0.5* cos(dAngle2) + 1) - 0.5 * sin(dAngle) * sin(dAngle2);// end-effector x좌표
    pdPos[1] = 0.5 * cos(dAngle)*sin(dAngle2) + sin(dAngle) * (0.5 * cos(dAngle2)+1);// end-effector y좌표
    pdPos[2] = 0.5;// z좌표
}
```

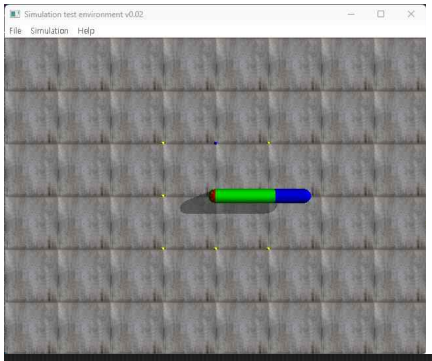
Forward kinematics : 정기구학은 일련의 관절각이 주어졌을 때 엔드이펙터의 직교 좌표상의 위치와 자세(각도)를 구하는 것이다. Homogeneous transform을 이용하여 정기구학의 해를 구한다.

```
// #2 위치 입력받아 pdAngle[0~1]에 출력
void CRobotExp_4Dlg::SolveInverseKinematics(double dX, double dY, double dZ, double* pdAngle)
{
    double L1 = 1.0;
    double L2 = 0.5;

    double x = dX;    //end effecteor x 좌표
    double y = dY;    //end effecteor y 좌표

    double cos2 = (x * x + y * y - (L1 * L1 + L2 * L2)) / (2 * L1 * L2);
    double sin2 = sqrt(1 - cos2*cos2); // theta2가 +부호인 경우

    double cos1 = (L1 + L2 * cos2) * dX + (L2 * sin2) * dY / (L1 + L2 * cos2) * (L1 + L2 * cos2) + (L2 * sin2) * (L2 * sin2);
    double sin1 = (L1 + L2 * cos2) * dY - (L2 * sin2) * dX / (L1 + L2 * cos2) * (L1 + L2 * cos2) + (L2 * sin2) * (L2 * sin2);

    pdAngle[0] = atan2(sin1, cos1); //theta1
    pdAngle[1] = atan2(sin2, cos2); // theta2
}
```

Inverse kinematics : 역기구학은 말단효과장치의 직교 좌표상의 위치 및 자

세(각도)가 주어졌을 때 관절각 들을 구하는 문제이다. 정기구학에 비해 구하는 것이 어렵고 여러가지 해가 존재할 수도 있다. 우리는 manipulator의 기하학적 특성을 이용하여 관절변수 값을 쉽게 구하였다.

ODE(open dynamics engine)




```
void StartDrawStuff() {


    //TO DO

    // 카메라 뷰 설정

    float dPos[3] = { 0.0,0.0,5.0 };     // X, Y, Z 좌표

    float dRot[3] = { 0.0,-90.0,90.0 }; // X축 Y축 Z축 기준 회전

    dsSetViewpoint(dPos, dRot);


}
```

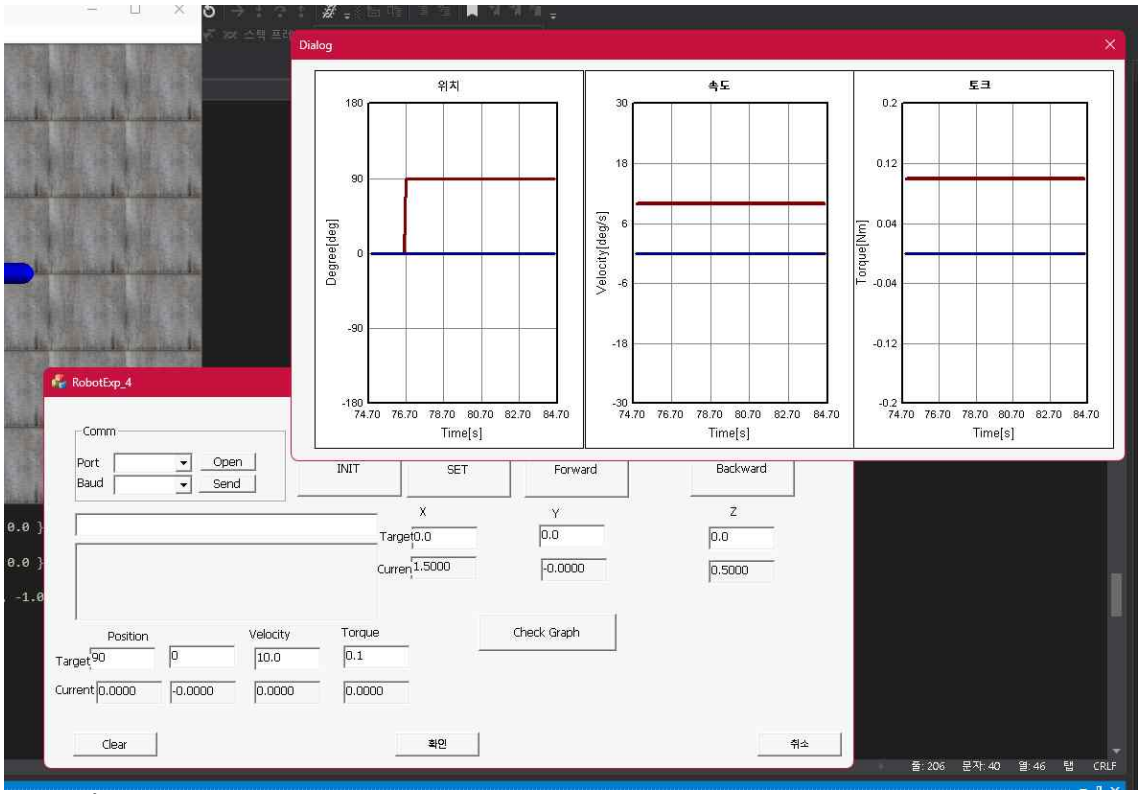
Ode란 무료dynamics engine으로 간편한 3d object 생성을 제공하고, 충돌처리, 객체의 물리정도 등을 제공한다. Ode는 현실적으로 제작하기 힘든 로봇의 제어 검증이나 가상환경의 로봇의 결합등에 활용된다. 링크 길이가 0.5, 1.0, 0.5인 2dof manipulator을 설계하여 90도로 구부러진 것을 위에서 본 모습이다.

```
//질량중심점
dReal x[MAX_JOINT_NUM + 1] = { 0.00, 0.50, 1.25 };
dReal y[MAX_JOINT_NUM + 1] = { 0.0, 0.0, 0.0 };
dReal z[MAX_JOINT_NUM + 1] = { 0.25, 0.5, 0.5 };

//링크 자세
dReal ori_x[MAX_JOINT_NUM + 1] = { 0.0, 0.0, 0.0 };
dReal ori_y[MAX_JOINT_NUM + 1] = { 0.0, 1.0, 1.0 };
dReal ori_z[MAX_JOINT_NUM + 1] = { 1.0, 0.0, 0.0 };
dReal ori_q[MAX_JOINT_NUM + 1] = { 0.0, 90*DEG2RAD, 90*DEG2RAD };

//림 한덩이 링크 길이
dReal length[MAX_JOINT_NUM + 1] = { 0.5, 1.0, 0.5 };
```

질량중심점, 링크 자세, 팀 한덩이 링크 길이 설정코드

```
//각 조인트의 회전축의 위치
dReal c_x[MAX_JOINT_NUM + 1] = { 0.0, 0.0, 1.0 };
dReal c_y[MAX_JOINT_NUM + 1] = { 0.0, 0.0, 0.0 };
dReal c_z[MAX_JOINT_NUM + 1] = { 0.0, 0.5, 0.5 };

//회전축의 로테이션 방향
dReal axis_x[MAX_JOINT_NUM + 1] = { 0.0, 0.0, 0.0 };
dReal axis_y[MAX_JOINT_NUM + 1] = { 0.0, 0.0, 0.0 };
dReal axis_z[MAX_JOINT_NUM + 1] = { 0.0, -1.0, -1.0 };
```

Ode 축, 좌표계 설정 회전방향은 시계방향이므로 y축을 – 방향으로 설정하였다.



NTGragh

Ntgraph는 간단한 ocx로 graph control을 사용할 수 있도록 해주는 activeX control이다. 위치, 속도, 전류를 각각 붉은색은 목표값 파란색은 현재값으로 구현하였다.

```
void CGraphDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: 여기에 메시지 처리기 코드를 추가 및/또는 기본값을 호출합니다.
    m_dCnt += 0.1;
    DataType_t jointData;
    GET_SYSTEM_MEMORY("JointData", jointData);

    ControlData_t motor_data;
    GET_SYSTEM_MEMORY("Comm1Work_Controller_Current", motor_data);

    ControlData_t motor_data_tar;
    GET_SYSTEM_MEMORY("Comm1Work_Controller_Target", motor_data_tar);

    if (m_dCnt >= 10.0)
    {
        m_ntgPos.SetRange(m_dCnt - 10.0, m_dCnt, -180.0, 180.0);
        m_ntgVel.SetRange(m_dCnt - 10.0, m_dCnt, -30.0, 30.0);
        m_ntgTorque.SetRange(m_dCnt - 10.0, m_dCnt, -0.2, 0.2);
    }

    //m_ntgPos.PlotXY(m_dCnt, jointData.Q_tar[0] * RAD2DEG, 1);
    m_ntgPos.PlotXY(m_dCnt, fmod(jointData.Q_tar[0] * RAD2DEG, 360.0), 1);
    m_ntgPos.PlotXY(m_dCnt, motor_data.position * RAD2DEG, 2);

    m_ntgVel.PlotXY(m_dCnt, motor_data_tar.velocity * RAD2DEG, 1);
    m_ntgVel.PlotXY(m_dCnt, motor_data.velocity * RAD2DEG , 2);

    m_ntgTorque.PlotXY(m_dCnt, motor_data_tar.current * 0.0683, 1);
    m_ntgTorque.PlotXY(m_dCnt, motor_data.current * 0.0683 , 2);

    CDialogEx::OnTimer(nIDEvent);
}
```

Get_system_memory ~target 부분에서 motor_data_tar로 목표값을 받아오고
Get_system_memory ~current 부분에서 motor_data로 현재값을 받아와
위치, 속도, 전류 값을 구하여 각각 그래프에2번으로 띄웠다.
nt그래프 구성 코드이다. if문안에서 그래프 y축 범위를 설정해준다.
전류값에 0.0683을 곱해 토크값을 구하였다.
주석 처리된 부분은, 모바일 모드를 위한 부분이다.

```
m_ntgVel.AddElement();
m_ntgVel.SetElementWidth(4);
m_ntgVel.SetElementLineColor(RED);
m_ntgVel.AddElement();
m_ntgVel.SetElementWidth(3);
m_ntgVel.SetElementLineColor(BLUE);

m_ntgTorque.AddElement();
m_ntgTorque.SetElementWidth(4);
m_ntgTorque.SetElementLineColor(RED);
m_ntgTorque.AddElement();
m_ntgTorque.SetElementWidth(3);
m_ntgTorque.SetElementLineColor(BLUE);
```

속도와 전류 그래프를 추가해주었다. 붉은색은 목표값 파란색은 현재값이다.

Set 버튼을 누룰시 현재 관절모드로 설정해둔상태이다 if문이 없다면 모바일 모드를 의미한다.

```
// RobotExp_4Dlg.cpp : 구현 파일
//

#include "stdafx.h"
#include "RobotExp_4.h"
#include "RobotExp_4Dlg.h"
#include "afxdialogex.h"
#include "DataType.h"
#include <math.h>
#include"SystemMemory.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

 bool flag_x = false;

 // 응용 프로그램 정보에 사용되는 CAboutDlg 대화 상자입니다.
```

Bool형 전역변수를 통해 ode와 motor통신에 따른 ode 연동을 달리 설정하였다.

```
void CRobotExp_4Dlg::OnBnClickedButtonInit()
{
    flag_x = true;
```

```
void CRobotExp_4Dlg::OnBnClickedButtonSet()
{
    flag_x = true;
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
    ControlData_t motor_data;
    DataType_t ode_data;

    SET_SYSTEM_MEMORY("JointData", ode_data);

    CString str;
    m_editTarPos1.GetWindowText(str);
    ode_data.Q_tar[0] = atof(str.GetBuffer()) * DEG2RAD;

    m_editTarPos2.GetWindowText(str);
    ode_data.Q_tar[1] = atof(str.GetBuffer()) * DEG2RAD;

    m_editTarPos1.GetWindowText(str);
    double k = atof(str.GetBuffer());
    if (k >= 360.0)
    {
        k = k - 360.0;
    }
    motor_data.position = k * DEG2RAD;

    m_editTarVel.GetWindowText(str);
    motor_data.velocity = atof(str.GetBuffer()) * DEG2RAD;

    m_editTarTorq.GetWindowText(str);
    motor_data.current = atof(str.GetBuffer()) / 0.0683;

    SET_SYSTEM_MEMORY("JointData", ode_data);
    SET_SYSTEM_MEMORY("Comm1Work_Controller_Target", motor_data);
}

void CRobotExp_4Dlg::OnBnClickedButtonForward()
{
    flag_x = false;
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
}

void CRobotExp_4Dlg::OnBnClickedButtonInverse()
{
    flag_x = false;
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
}

void CRobotExp_4Dlg::OnBnClickedButtonSet()
{
    flag_x = true;
    // TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.
}
```

True일 경우 motor 에 움직임에 따라 ode가 바뀌고

Flase일 경우 joint 값에 따라 바뀐다.

```
void SimLoopDrawStuff(int pause)
{

    extern bool flag_x;
    //TO DO

    DataType_t jointData;

    GET_SYSTEM_MEMORY("JointData", jointData);

    ControlData_t motor_data_cur;
    GET_SYSTEM_MEMORY("Comm1Work_Controller_Current", motor_data_cur);



    if (flag_x)
    {
        g_tar_q[0] = motor_data_cur.position;
    }
    else
    {
        g_tar_q[0] = jointData.Q_tar[0];
        g_tar_q[1] = jointData.Q_tar[1];
    }
```

mfc&avr 패킷통신 코드 부분

```
packet.data.pos = g_Pcur * 1000;
packet.data.velo = g_Vcur * 1000;
packet.data.cur = g_Ccur * 1000;
```

패킷을 보내는 avr 코드 부분이다. 기존의 limit값을 현재값인 current 값으로 바꿔주었다.

1000을 곱하는 이유는 값이 너무 작아 보내는 패킷통신값을 증폭시킨다. 이 값은

```
g_Pdes = g_PacketBuffer.data.pos / 1000.;
g_Vlimit = g_PacketBuffer.data.velo / 1000.;
if(g_Vlimit < 0)   g_Vlimit = -g_Vlimit;
if(g_Vlimit > 2.2689)   g_Vlimit = 2.2689;
g_Climit = g_PacketBuffer.data.cur / 1000.;
if(g_Climit < 0)   g_Climit = -g_Climit;
if(g_Climit > 2.5)   g_Climit = 2.5;
```

원형큐와 수신된 데이터를 처리하는 코드로 받는 곳에서 1000으로 나눠주었다.

나. 제어기 구성

앞선 시간에 우리가 과제나, 수업시간에 배웠던 내용을 토대로 제어기를 설계하였다. 아무래도 코드로만 이루어져 있어서 어려움이 있었으나. cascade 형태를 취하기 위해 각각의 if문에서 나오는 결과값을 사용하였다.

```
//위치 제어기
if((g_TimerCnt % 100) == 0){
   g_TimerCnt = 0;
   //위치 제어기 변수 선언
   double kp_p = 12.56;
   double kd_p = 0.1;

   g_Ppre = g_Perr;
   g_Perr = g_Pdes - g_Pcur;
   //제어기 설계
   g_pos_control = kp_p * g_Perr + kd_p * (g_Perr-g_Ppre)/0.05;

   //I-term anti
   if(g_pos_control > 2.2689){
      g_pos_control = 2.2689;
   }
   else if(g_pos_control < -2.2689){
      g_pos_control = -2.2689;
   }
}
```

g_pos_control (위치 제어기 결과값)을 속도 제어기의 지령값으로 넣었다.

속도 제어기 설계 부분에서 상당히 많은 어려움이 있었는데 matlab 환경에

```
//속도 제어기
g_Vdes = g_pos_control;
if((g_TimerCnt % 10) == 0){
    //속도 제어기 변수 선언
    double kp_v = 2.6262;
    double ki_v = 47.5; // 원래값 689.4846 --> 4번 2로 나눠서 됨
    double ka_v = 1/2.6262;
    //saturation 설정
    if(g_Vdes >= g_Vlimit){
        g_Vdes = g_Vlimit;
    }
    else if(g_Vdes <= -g_Vlimit){
        g_Vdes = -g_Vlimit;
    }
```

서 본 P_i (pid제어 I gain)의 값이 실제 제어를 하기에 적절치 않았다.

```
g_Vcur = (g_Pcur - g_Pvcur) / 0.005;
g_Pvcur = g_Pcur;
g_Verr = g_Vdes - g_Vcur;
//제어기 설계
g_vel_control = kp_v*g_Verr + ki_v*g_Verr_sum*0.005;
g_Verr_sum += g_Verr;
//I-term anti
if(g_vel_control >= 2.08){
    g_Verr_sum -=(g_vel_control - 2.08) * ka_v;
    g_vel_control = 2.08;
}
else if(g_vel_control <= -2.08){
    g_Verr_sum -= (g_vel_control + 2.08) * ka_v;
    g_vel_control = -2.08;
}
```

현재 위치와 이전 위치의 차이로 속도를 구하고, 이를 통해 error를 구해서 제어기를 구성한 모습이다 (//제어기설계 부분) 이부분에서 속도 제어를 위한 PI 제어기를 구현 하였고, 누적오차를 제거했다. 다음 제어기인 전류 제어기를 설계하기 위해서 g_vel_control 값을 넘겨 주었다.

```
g_TimerCnt++;
//전류 제어기 변수 선언
double kp_c = 0.8264;
double ki_c = 2210;
double ka_c = 1/0.8264;
//전류 제어기
g_Cdes = g_vel_control;
//saturation 설정
if(g_Cdes >= g_Climit){
    g_Cdes = g_Climit;
}
else if(g_Cdes <= -g_Climit){
     g_Cdes = -g_Climit;
}
//g_ADC를 통해 현재 전류값 받아오기
g_Ccur = -( ((g_ADC / 1024. * 5.) - 2.5) * 10.);
```

마지막 제어기인 전류제어기의 초반 부분이다. (사진이 커서 두부분으로 나눔) PI제어에 쓰일 값은 matlab에서 그대로 가져왔다.

전류 목표값을 (전류지령) 속도 제어기의 출력값으로 설정 하였다. (cascade), saturation을 하기위해 g_limit을 사용하였다.

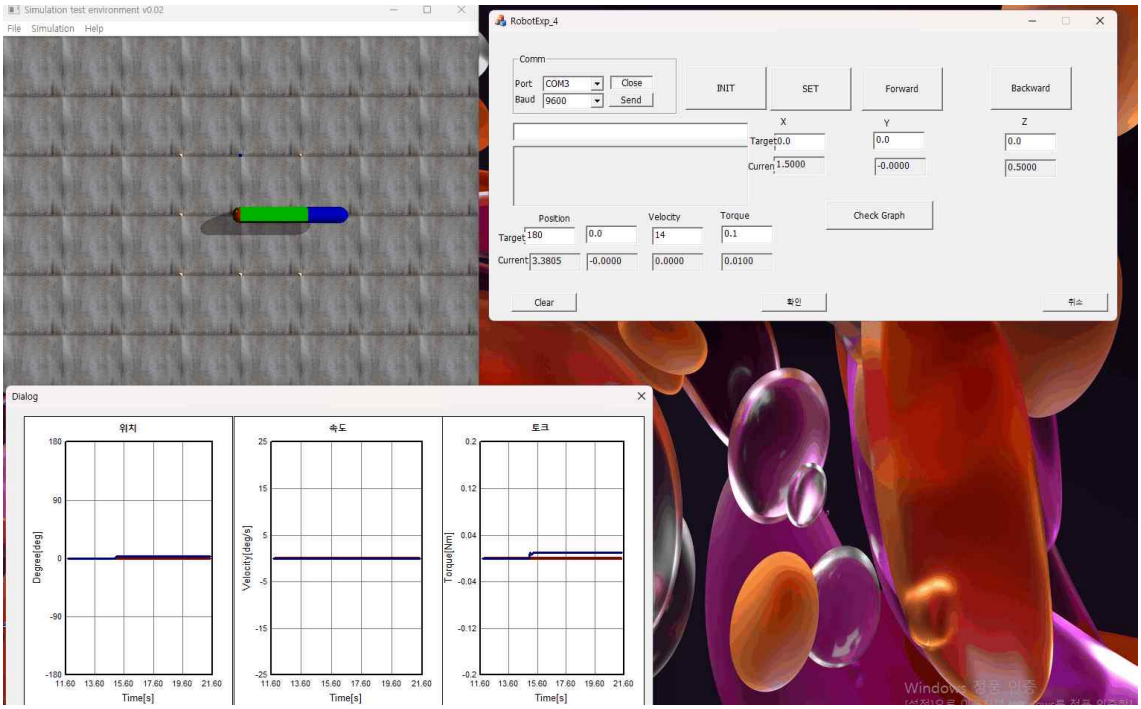
```
g_Cerr = g_Cdes - g_Ccur;
//제어기 설계
cur_control = g_Cerr *kp_c + g_Cerr_sum * ki_c * 0.0005;
cur_control += g_Vcur * 0.0683;

g_Cerr_sum += g_Cerr;
//I-term anti
if(cur_control > 24){
    g_Cerr_sum -= (cur_control - 24) * ka_c;
    cur_control = 24;
}
else if(cur_control < -24){
    g_Cerr_sum -= (cur_control + 24) * ka_c;
    cur_control = -24;
}
```

전류제어기도 위의 속도제어기와 마찬가지로 PI제어 이기 때문에 둘의 형식은 거의 비슷하다. error = target – current 의 개념을 이용해서 g_Cerr은 목표지령 – 현재값 으로 만들었다. cur_control 변수에 PI 제어를 넣었고, cur_control에 += g_Vcur * 0.0683 는 역기전력 전향 보상으로 넣어 주었다.

밑의 if 문은 anti-windup을 구현한 것이다.

다. 실험결과

결과 GIF이다. 위치 지령을 180, 속도 지령을 14로 해두고 테스트를 하였다.



전류제어가 잘 되나 확인하기 위해서 막대를 잡았다. GIF를 보듯이 전류를 제대로 추정하는 모습을 볼 수 있었다. 중간 중간 속도 값이 튀는데 이건 이제 모터 토크가 내가 잡는힘보다 세서 놓쳤을 때 값이 튀는걸 확인 할 수 있었고, 잡을 땐 위치 그래프도 잠시 주춤하는 모습을 보였지만 외력(내가 잡는힘)이 사라지자 마자 180로 아주 잘 추종하는 모습을 볼 수 있었다.

4. 고찰

matlab에서 구한 제어기 이득값, 특히 P gain 이 아닌 I gain이 시뮬레이션 상에서 구한 값과 현실에 적용해서 직접돌리는 값의 차이가 났다. 이런 차이가 왜 생기나 생각을 해보았는데 여러 가지 이유가 있을 것 같다.

첫 번째로 모터와 부하에 연결부분이 생각 한 것 만큼 부드럽지 않았는데 부드럽지 않아도 미끄러짐이 발생했다.

두 번째로 부하에 대한 속성이다. gain 값 튜닝을 하고 모터를 돌릴 때 부하가 분리되는 일이 있었다. 시뮬레이션을 할 때는 부하가 한 몸인 것처럼 생각하고 시뮬레이션을 진행하였는데 실제 모터를 돌려보니 모터의 진동이나 각속도에 의해 부하가 분리될 수 도 있음을 알게 되었다.

세 번째로 중력에 대한 성분을 고려해주지 않았다. 하드웨어 쪽 문제를 해결하기 위해 모터를 돌렸을 때 부하가 중력방향과 같은 방향으로 떨어지자 역기전력이 크게 걸리면서 모터가 멈추었다. 그리고 중력방향과 같을 때의 모터속도와 중력방향과 다를때의 모터속도에 약간의 차이가 있었는데 이는 속도제어를 할 때 고려해 주지 않았다.

네 번째로 시뮬레이션과 다른 환경이다. 이게 무슨말인가 하면 우리팀은 이번 텀프로젝트 때 정말 많은 오류가 있었다. 엔코더 관련 칩도 깨지고, 오실로스코프로 엔코더가 제대로 작동되는지 보는데 연결 헤더가 잘못되어서 다시 납땜도 하고, 기판 전원부가 제대로 부착되지 않아 모터가 돌아가는 도중에 선이 끊기면서 시스템이 쇼트가 나기도 하였다. isp 선이 무슨 이유에서인지 몰라도 컴퓨터와 보드의 통신이 제대로 되지 않을 때도 있었다.

이런 위에 나열한 여러 가지 이유로 컴퓨터 데이터 상에서 보이지 않는 error 들이 쌓이고 쌓였기 때문에 I gain 값이 matlab 상에서 계산한 값보다 작게 나와야 하는 이유이지 않을까 그렇게 생각한다.

이번 텀프로젝트를 하면서 하드웨어적으로나 소프트웨어적으로 문제가 발생했을 때 논리적으로 순서를 가지고 어디가 틀렸는지 확인을 해서 고치는 경험을 하게 되었고.
평상시에는 제어기를 대할 때 라플라스 변환을 거친 수식화한 제어기만 눈으로 보다가 이렇게 학기 실습과목에서 직접 설계를 할 수 있게 되어서 제이기 구조나 TIMER에서 제어기를 어떤 방식으로 설계를 해야 하는지, gain 값들이 어떤 작용을 하는지 몸으로 직접 알게 되었다.