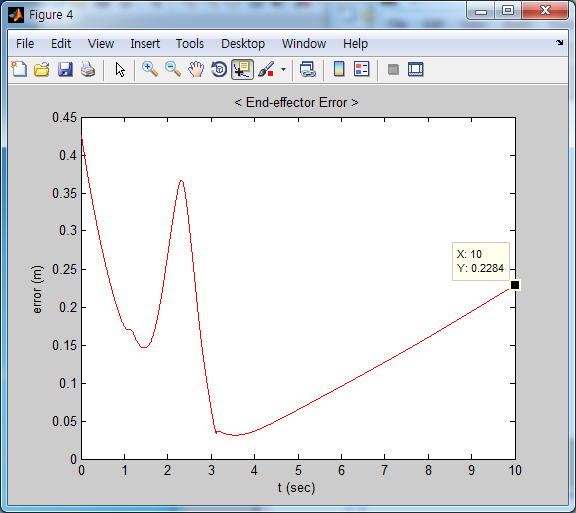
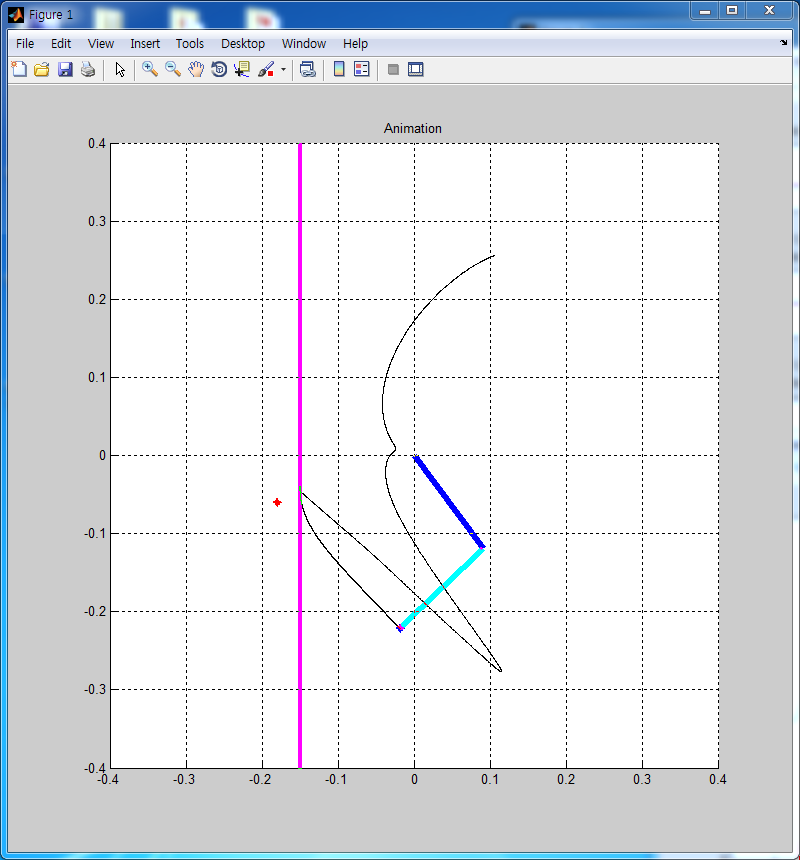
**Problem 1**

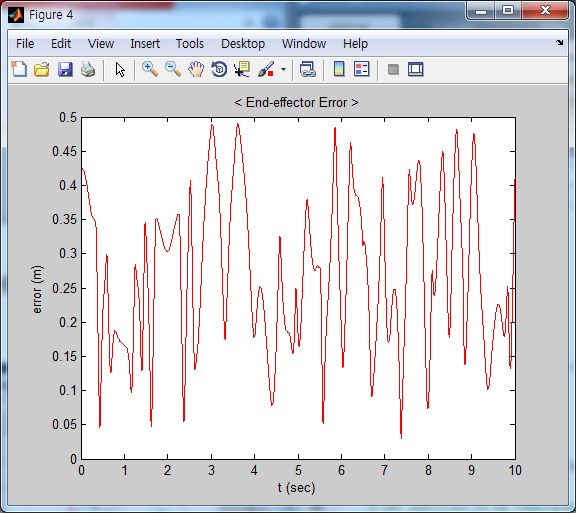
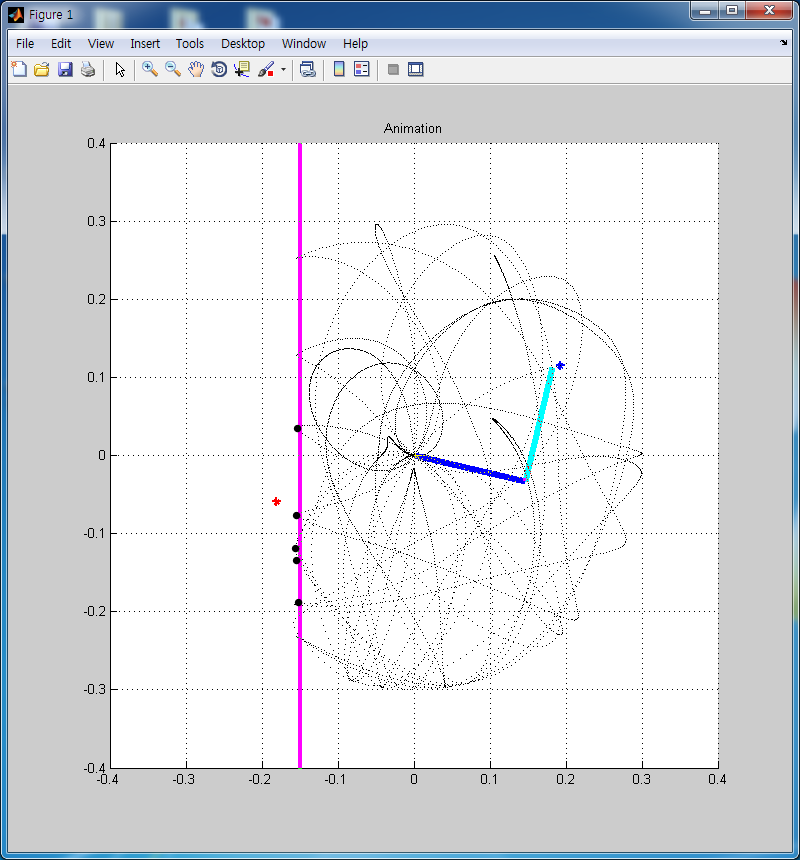
* 1. Impedance Control using PD scheme



Km = [32,32;32,32]; Dm = [35,35;35,35]; 을 넣었을 때의 결과 값이다.

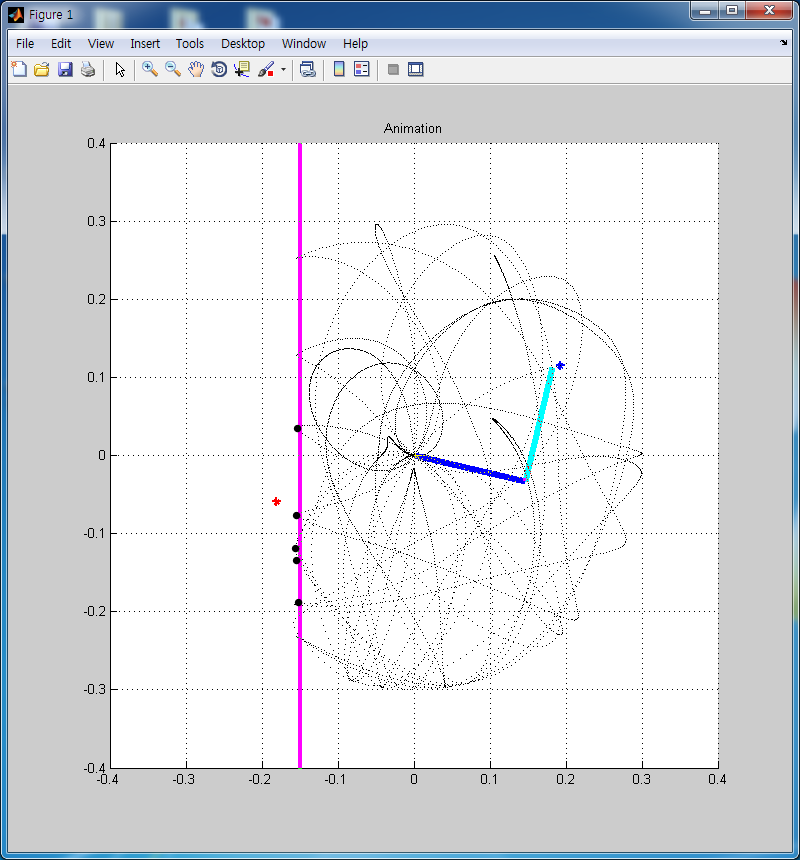
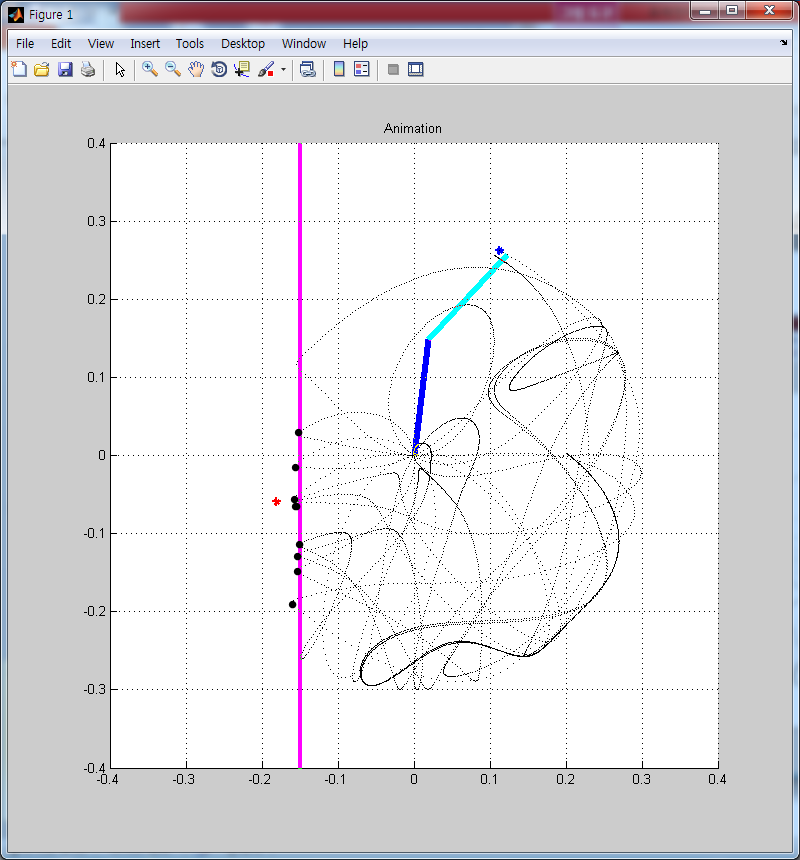
Gain tuning시 제어기의 문제점은 set point에 도달하는 시간이 느리다는 점과 set point에 도달한 후 error 값이 증가한다는 점이었다. 따라서 Gain tuning의 기준을 set point 도달시간 3초, 최종 error 0.23m로 정하고 제어gain을 맞추었다.

* 1. Stiffness Control



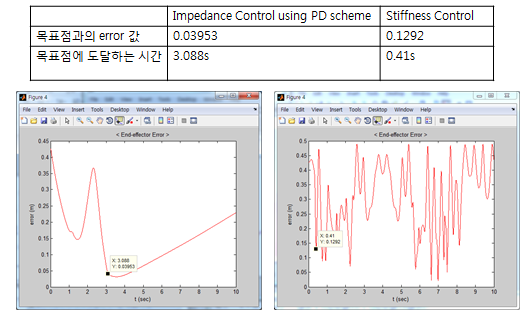
Km = [7.31,7.31;7.31,7.31];를 넣었을 때의 결과 값이다.

Gain tuning 방법은 Set point 근처 지점(오차범위 0.15m) 에 도달하는 수를 세어 가장 많은 제어기를 선정하였다. 하지만 stiffness control은 한계점이 있었다. Gain tuning시 gain의 변화에 따른 경향성을 찾지 못했기 때문에 원하는 만큼의 제어 성능을 얻지 못하였다. 특히 gain값에 따라 매우 민감하게 반응하여(소수점 둘째 자리의 변화에도 민감) gain값을 찾기가 더 어려웠다.



위의 그림은 gain을 맞추기 전(Km = [3,3;3,3];) 과 후의 animation이다. Gain을 맞추기 전은 근처지점에 도달하는 수가 5개, Gain을 맞춘 후는 7개로 gain tuning이 조금은 된 제어기라고 볼 수 있다. 또한 경로를 보면 왼쪽의 그림은 원형 전체적으로 퍼져있는데 오른쪽은 목표점 근처에 조금 더 집중 되어 있다고 볼 수 있다.

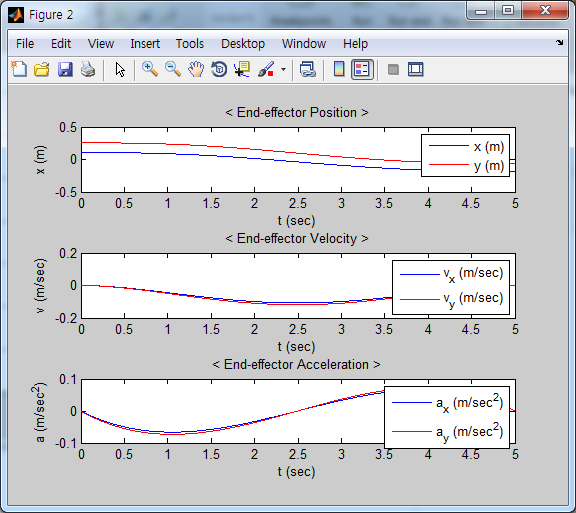
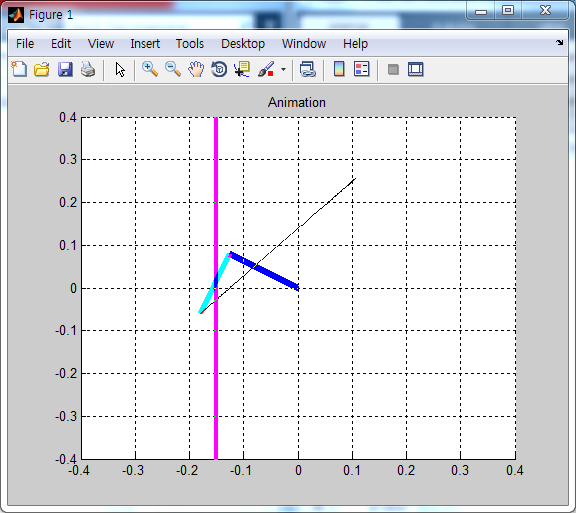
* 1. Discuss the performance differences between above two controllers



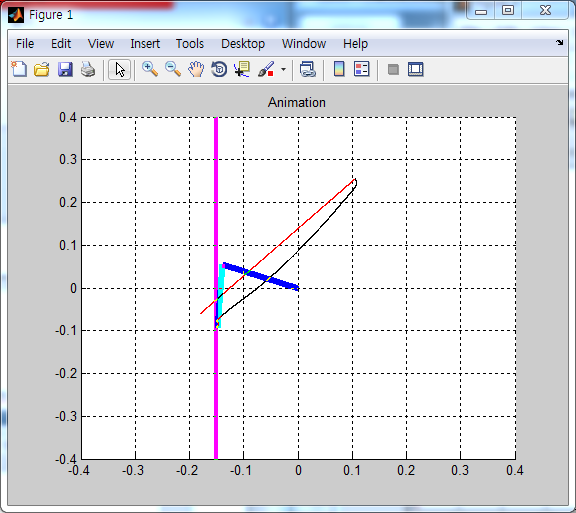
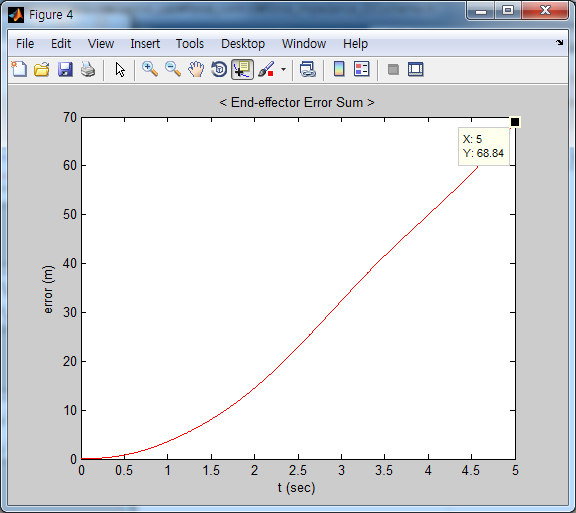
위는 두 제어기 간의 정량적인 비교를 해본 내용이다. 목표점과의 error 값의 기준은 벽에 부딪혔을 때를 기준으로 하였다. 두 번째 제어기는 벽에 여러 번 부딪혔는데 그 중에서 제일 처음 부딪혔을 때 목표점 근방이었기 때문에 그 점을 기준으로 하였다. 목표점과의 error 값은 첫 번째 제어기가 0.03953, 두 번째 제어기가 0.1292로 첫 번째가 목표 값과 더 작은 error 값을 보였으며, 목표점에 도달하는 시간은 첫 번째 제어기가 3.088s, 두 번째 제어기가 0.41s로 두 번째 제어기가 목표점에 더 빨리 도달하였다. Stiffness control의 경우 목표점에 더 빨리 도달한 것으로 보이나 목표점과의 거리가 크므로 PD scheme을 이용한 Impedance Control이 더 좋다고 생각하였다. 이러한 비교는 Stiffness Control의 성능이 의심되므로 이 결과 값만으로 두 제어기를 비교하는 것이 무리가 있다고 생각하였다.

**Problem 2**

2.1 Generate desired trajectory with elapsed time 5s 5-th order polynomial



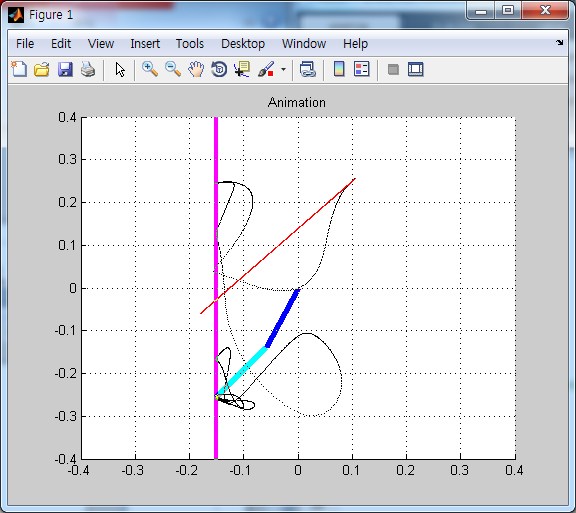
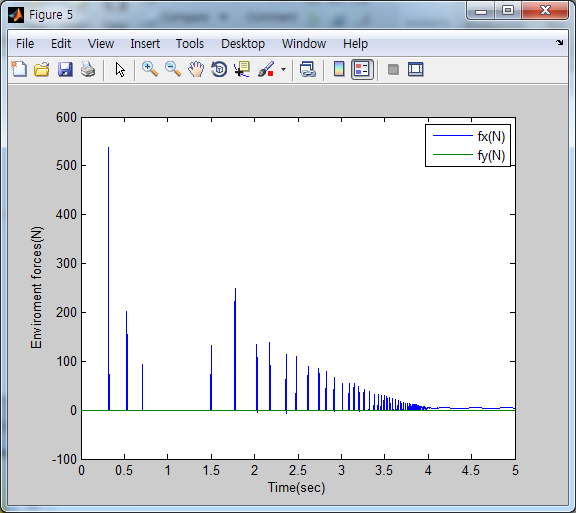
2.2 Impedance Control using IDC scheme



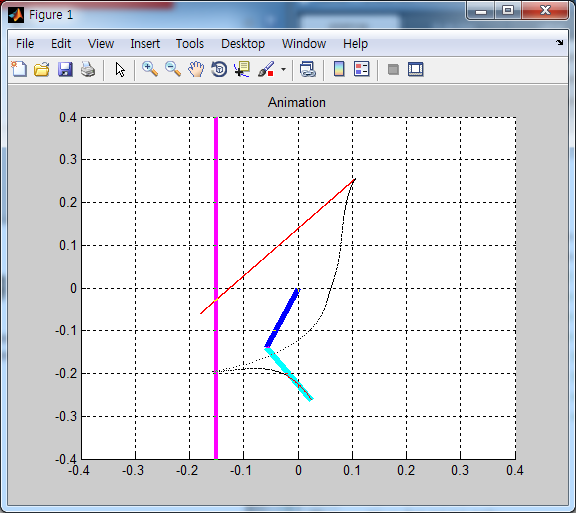
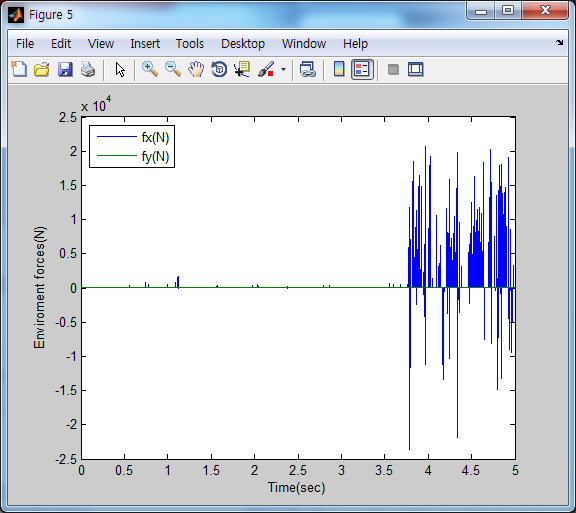
Trajectory tracking이므로 Error의 sum이 작도록 Gain tuning을 하였다. 위는 Km = [10.2,10.2; 10.2, 10.2]; Dm = [0.01, 0.01; 0.01, 0.01]; 의 gain값을 넣은 결과이다.

* 1. Parallel Control using IDC scheme

맞춰야 할 Gain값은 Hm, Kf, Ki, Km, Dm으로 총 5개 였다. Gain tuning을 Desired trajectory를 잘 따라가는지와 Desired contact force를 5N 이내로 유지하는지를 기준으로 하였다. 다음 그림은 gain값을 넣었을 때이다. Hm = [1,0;0,1]; Kf = [20,20;20,20]; Ki = [10,10;10,10]; Km = [40,40;40,40]; Dm = [60,60;60,60];

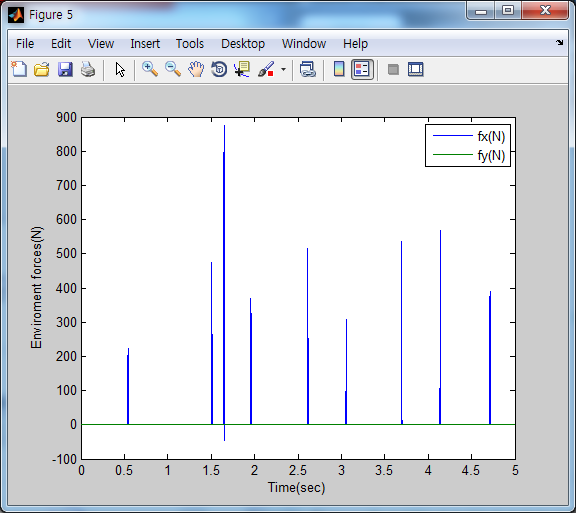
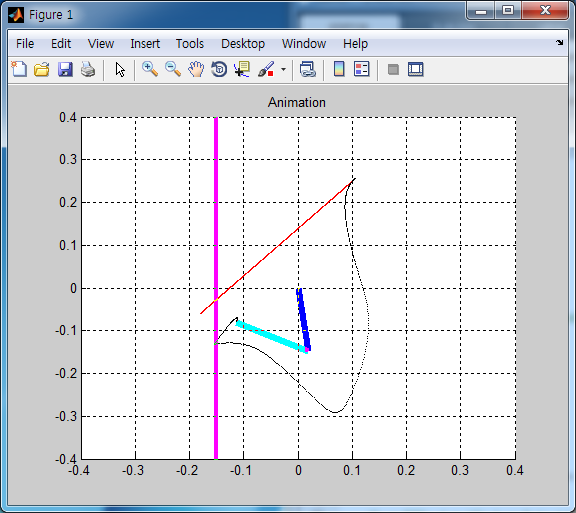


1. 가장 먼저 Hm의 값을 맞췄다. Hm의 경우 너무 작으면 폭주하기 쉽고 너무 크면 경로에서 벗어나기 쉬웠다. 다음 그림은 기준 gain값에서 Hm을 각각 0.1과 2로 바꾸었을 때의 결과이다.



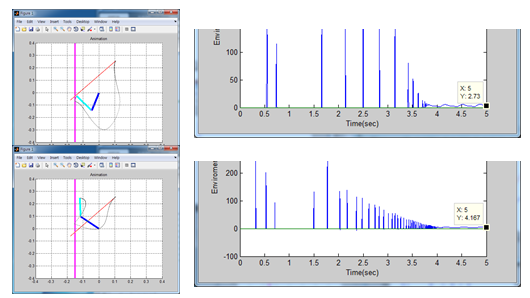
기준 gain값의 결과와 비교하여 보면 gain값이 작을 때는 기준 값보다 끝부분에서 힘을 너무 많이 받고 gain값이 클 때는 경로에서 많이 벗어남을 알 수 있다.

1. 다음으로 Km과 Dm을 바꾸어 보았다. 아래는 Km와 Dm를 각각 2배와 0.5배 했을 때의 결과이다.



Gain 값이 작으면 force control이 잘 안되고, 기준 값 보다 크면 경로에서 벗어나기 쉬웠다

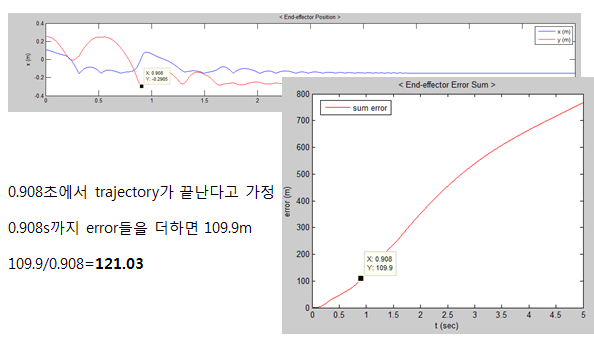
1. 마지막으로 Kf와 Ki를 바꾸어 보았다. 아래는 Kf와 Ki를 각각 2배와 0.5배 했을 때의 결과이다.

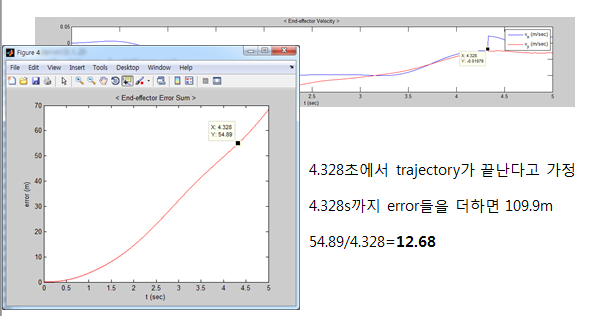


Kf와 Ki가 작을 수록 경로에서 벗어나지만 접촉력은 오히려 더 작았으며, Kf와 Ki가 크면 접촉력이 크며 폭주하기 쉬웠다.

* 1. Discuss the performance differences between above two controllers

두 제어기 간의 성능 비교는 Trajectory tracking만 비교할 수 밖에 없었다. Force control의 비교는 IDC의 경우 force control이 되고 있다는 것을 알 수가 없어 할 수가 없었다. 정량적 비교는 trajectory를 벗어나는 정도를 기준으로 하였다. 아래 그림은 각각 Impedance control과 Parallel control이다.





위의 결과에 따르면 전체 경로의 error의 평균값이 Impedance Control의 경우가 12.68로 더 작다는 것을 알 수 있다.

**Impedance Control using PD scheme**

% Constant

k = 32; d = 35;

Km = [k,k;k,k];

Dm = [d,d;d,d];

% Robot Implementation

for i = 0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

[t,y] = ode45('two\_link\_env',[0, s\_time] , [q(1); q(2); qdot(1); qdot(2)] );

index = size(y);

q(1) = y(index(1), 1);

q(2) = y(index(1), 2);

qdot(1) = y(index(1), 3);

qdot(2) = y(index(1), 4);

qddot(1) = temp2(1);

qddot(2) = temp2(2);

% Control part to be designed %

u = Ja'\*(Km\*(xd-x) - Dm\*xdot) + G;

% Save environment force

f\_a(:,n)=fa;

err(1) = abs(xd(1)-x2);

err(2) = abs(xd(2)-y2);

e\_old = e;

e = sqrt(err(1)^2 + err(2)^2);

if(i==0) de = 0;

else de = e - e\_old;

end

end

**Stiffness Control**

% Constant

k = 7.31;

Km = [k,k;k,k];

% Robot Implementation

for i = 0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

[t,y] = ode45('two\_link\_env',[0, s\_time] , [q(1); q(2); qdot(1); qdot(2)] );

index = size(y);

q(1) = y(index(1), 1);

q(2) = y(index(1), 2);

qdot(1) = y(index(1), 3);

qdot(2) = y(index(1), 4);

qddot(1) = temp2(1);

qddot(2) = temp2(2);

% Control part to be designed %

u = Ja'\*Km\*Ja\*(qd-q);

% Save environment force

f\_a(:,n)=fa;

err(1) = abs(xd(1)-x2);

err(2) = abs(xd(2)-y2);

e = sqrt(err(1)^2 + err(2)^2);

end

**Quintic trajectory**

% Terminal time %

t0 = 0;

tf = 5;

% Initial Configurations defined in RADIAN %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.18;-0.06];

q = [pi/4;pi/4];

qdot = [0;0]; qddot = [0;0];

xdot = [0;0]; xddot = [0;0];

% Robot trajectories %

x\_0 = x; x\_f = x\_d;

xdot\_0 = xdot; xdot\_f = [0;0];

xddot\_0 = xddot; xddot\_f = [0;0];

[a10,a11,a12,a13,a14,a15]=QuinticPolynomialPath(x\_0(1), xdot\_0(1), xddot\_0(1), x\_f(1), xdot\_f(1), xddot\_f(1));

[a20,a21,a22,a23,a24,a25]=QuinticPolynomialPath(x\_0(2), xdot\_0(2), xddot\_0(2), x\_f(2), xdot\_f(2), xddot\_f(2));

% Initial Control Input %

u=[0;0];

% Robot Implementation

for i = 0 : s\_time : tf

% Desired Trajectory

t=i;

x\_d(1)=a10+a11\*t+a12\*t^2+a13\*t^3+a14\*t^4+a15\*t^5;

x\_d(2)=a20+a21\*t+a22\*t^2+a23\*t^3+a24\*t^4+a25\*t^5;

xdot\_d(1)=a11+2\*a12\*t+3\*a13\*t^2+4\*a14\*t^3+5\*a15\*t^4;

xdot\_d(2)=a21+2\*a22\*t+3\*a23\*t^2+4\*a24\*t^3+5\*a25\*t^4;

xddot\_d(1)=2\*a12+6\*a13\*t+12\*a14\*t^2+20\*a15\*t^3;

xddot\_d(2)=2\*a22+6\*a23\*t+12\*a24\*t^2+20\*a25\*t^3;

x\_d = [x\_d(1);x\_d(2)];

xdot\_d = [xdot\_d(1);xdot\_d(2)];

xddot\_d = [xddot\_d(1);xddot\_d(2)];

x2 = x\_d(1); y2 = x\_d(2);

[q1, q2] = IK(x2,y2);

x1 = l1\*cos(q1); y1 = l1\*sin(q1);

% For Animation %

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

end

**Force control using IDC scheme**

% Constant %

k = 10.2; d = 0.001;

Km = [k,k;k,k];

Dm = [d,d;d,d];

% Initial Configurations defined in RADIAN %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.18;-0.06];

q = [pi/4;pi/4];

qdot = [0;0]; qddot = [0;0];

xdot = [0;0]; xddot = [0;0];

% Robot trajectories %

x\_0 = x; x\_f = x\_d;

xdot\_0 = xdot; xdot\_f = [0;0];

xddot\_0 = xddot; xddot\_f = [0;0];

[a10,a11,a12,a13,a14,a15]=QuinticPolynomialPath(x\_0(1), xdot\_0(1), xddot\_0(1), x\_f(1), xdot\_f(1), xddot\_f(1));

[a20,a21,a22,a23,a24,a25]=QuinticPolynomialPath(x\_0(2), xdot\_0(2), xddot\_0(2), x\_f(2), xdot\_f(2), xddot\_f(2));

% Robot Implementation

for i = 0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

[t,y] = ode45('two\_link\_env',[0, s\_time] , [q(1); q(2); qdot(1); qdot(2)] );

index = size(y);

q(1) = y(index(1), 1);

q(2) = y(index(1), 2);

qdot(1) = y(index(1), 3);

qdot(2) = y(index(1), 4);

qddot(1) = temp2(1);

qddot(2) = temp2(2);

% Desired Trajectory

t=i;

x\_d(1)=a10+a11\*t+a12\*t^2+a13\*t^3+a14\*t^4+a15\*t^5;

x\_d(2)=a20+a21\*t+a22\*t^2+a23\*t^3+a24\*t^4+a25\*t^5;

xdot\_d(1)=a11+2\*a12\*t+3\*a13\*t^2+4\*a14\*t^3+5\*a15\*t^4;

xdot\_d(2)=a21+2\*a22\*t+3\*a23\*t^2+4\*a24\*t^3+5\*a25\*t^4;

xddot\_d(1)=2\*a12+6\*a13\*t+12\*a14\*t^2+20\*a15\*t^3;

xddot\_d(2)=2\*a22+6\*a23\*t+12\*a24\*t^2+20\*a25\*t^3;

x\_d = [x\_d(1);x\_d(2)];

xdot\_d = [xdot\_d(1);xdot\_d(2)];

xddot\_d = [xddot\_d(1);xddot\_d(2)];

% Control part to be designed %

u = H\*inv(Ja)\*(xddot\_d-Jadot\*qdot) + C\*qdot + G + Ja'\*(Dm\*(xdot\_d-xdot) + Km\*(x\_d-x));

**Parallel control using IDC scheme**

% Constant

h = 1; kf = 20; ki = 10; k = 40; d = 60;

Km = [k,k;k,k];

Dm = [d,d;d,d];

Hm = [h,0;0,h];

Kf = [kf,kf;kf,kf];

Ki = [ki,ki;ki,ki];

% Initial Configurations defined in RADIAN %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.18;-0.06];

q = [pi/4;pi/4];

qdot = [0;0]; qddot = [0;0];

xdot = [0;0]; xddot = [0;0];

% Robot trajectories %

x\_0 = x; x\_f = x\_d;

xdot\_0 = xdot; xdot\_f = [0;0];

xddot\_0 = xddot; xddot\_f = [0;0];

[a10,a11,a12,a13,a14,a15]=QuinticPolynomialPath(x\_0(1), xdot\_0(1), xddot\_0(1), x\_f(1), xdot\_f(1), xddot\_f(1));

[a20,a21,a22,a23,a24,a25]=QuinticPolynomialPath(x\_0(2), xdot\_0(2), xddot\_0(2), x\_f(2), xdot\_f(2), xddot\_f(2));

% Initial Control Input %

u = [0;0];

fad = [5;0];

error\_sum\_f = 0;

sum\_err =[0;0];

sum\_e=0;

% Robot Implementation

for i = 0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

[t,y] = ode45('two\_link\_env',[0, s\_time] , [q(1); q(2); qdot(1); qdot(2)] );

index = size(y);

q(1) = y(index(1), 1);

q(2) = y(index(1), 2);

qdot(1) = y(index(1), 3);

qdot(2) = y(index(1), 4);

qddot(1) = temp2(1);

qddot(2) = temp2(2);

% Desired Trajectory

t=i;

x\_d(1)=a10+a11\*t+a12\*t^2+a13\*t^3+a14\*t^4+a15\*t^5;

x\_d(2)=a20+a21\*t+a22\*t^2+a23\*t^3+a24\*t^4+a25\*t^5;

xdot\_d(1)=a11+2\*a12\*t+3\*a13\*t^2+4\*a14\*t^3+5\*a15\*t^4;

xdot\_d(2)=a21+2\*a22\*t+3\*a23\*t^2+4\*a24\*t^3+5\*a25\*t^4;

xddot\_d(1)=2\*a12+6\*a13\*t+12\*a14\*t^2+20\*a15\*t^3;

xddot\_d(2)=2\*a22+6\*a23\*t+12\*a24\*t^2+20\*a25\*t^3;

% Control part to be designed %

error\_f = fad-fa;

error\_sum\_f = error\_f\*s\_time + error\_sum\_f;

u0x = xddot\_d + (Hm)\(Dm\*(xdot\_d-xdot) + Km\*(x\_d-x));

u0f = -(Hm)\(Kf\*error\_f+Ki\*error\_sum\_f);

u = Ja'\*(H\*(u0x+u0f) + C\*xdot + G -fa);

end

로봇제어실험

<force control>

2013046422

김효정