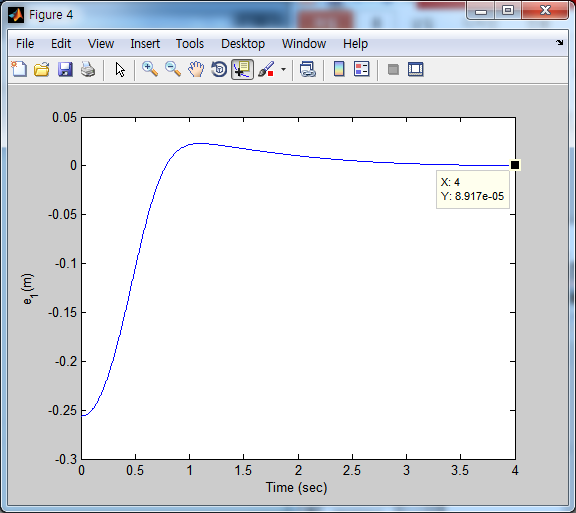
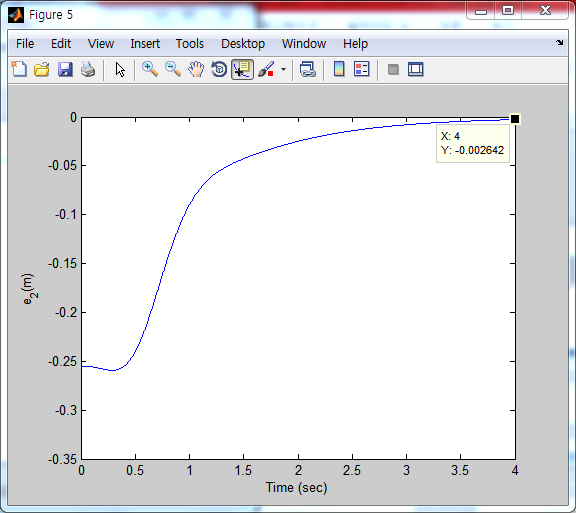
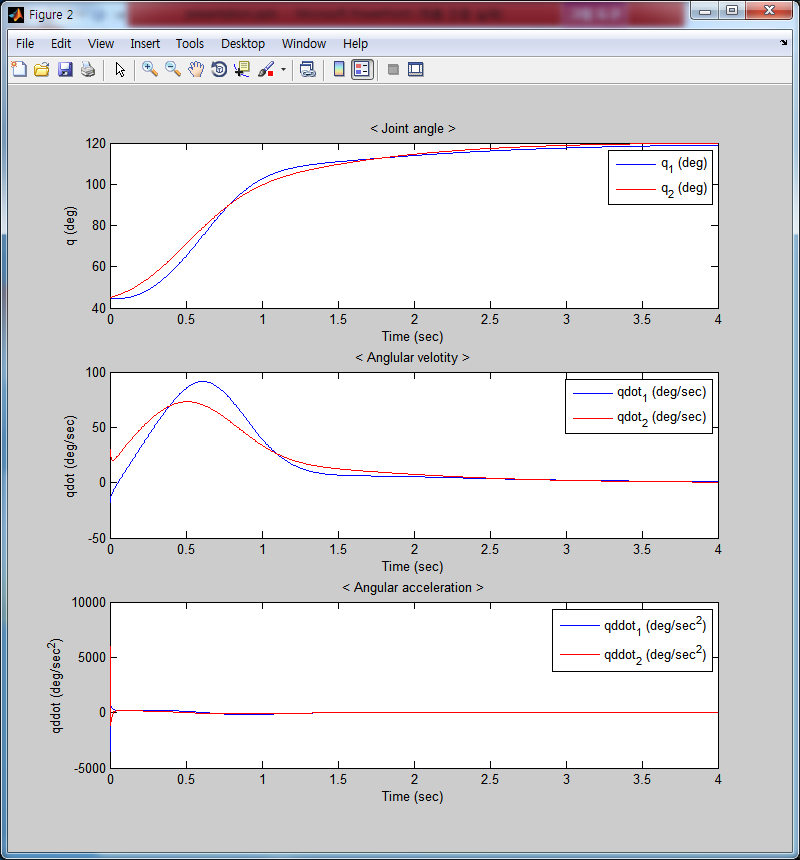
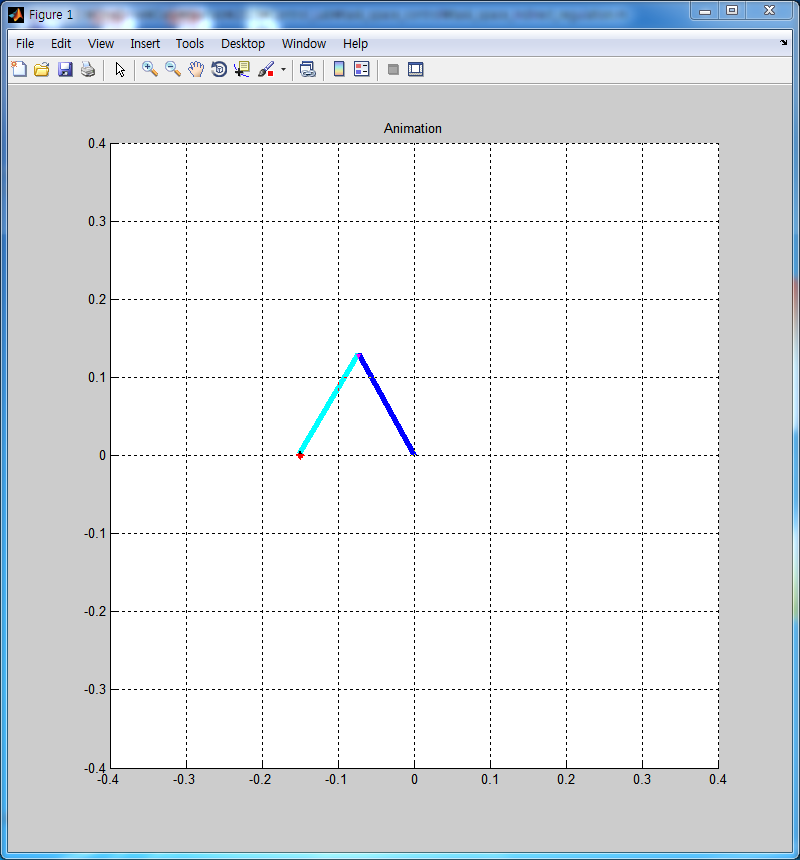
**Problem 1**

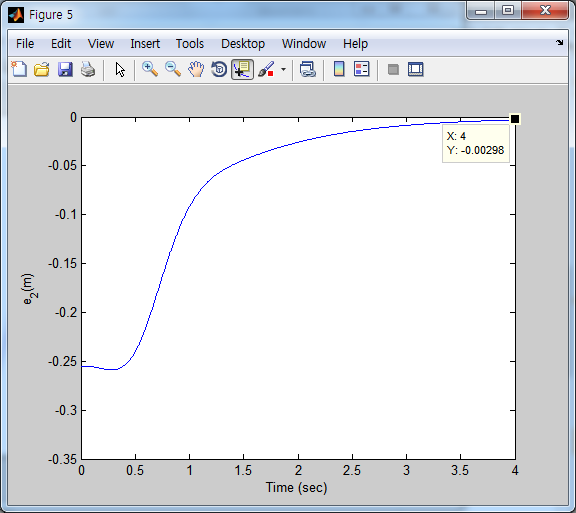
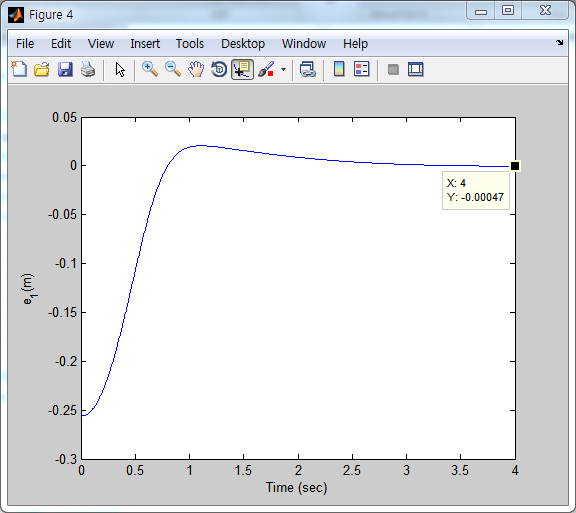
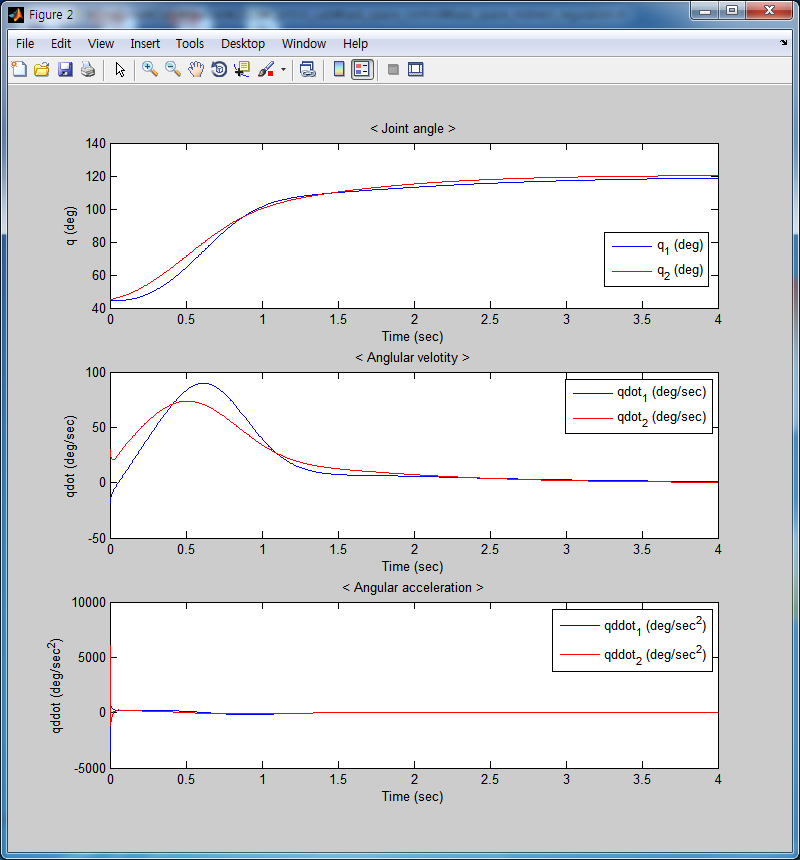
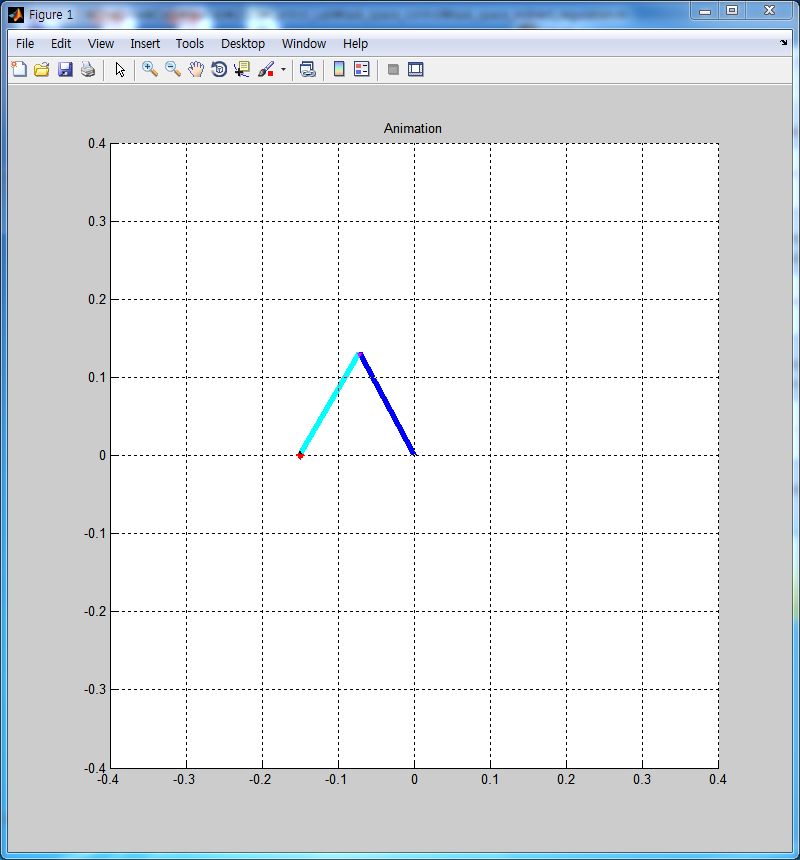
* 1. Indirect Task Space Regulation Control using Jacobian Transpose

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Indirect regulation <two\_link> | 8.92E-05 | 2.64E-03 |
| Indirect regulation <two\_link\_df> | 4.70E-04 | 2.98E-03 |
| difference | -3.81E-04 | -3.38E-04 |

two\_link.m



two\_link\_df.m



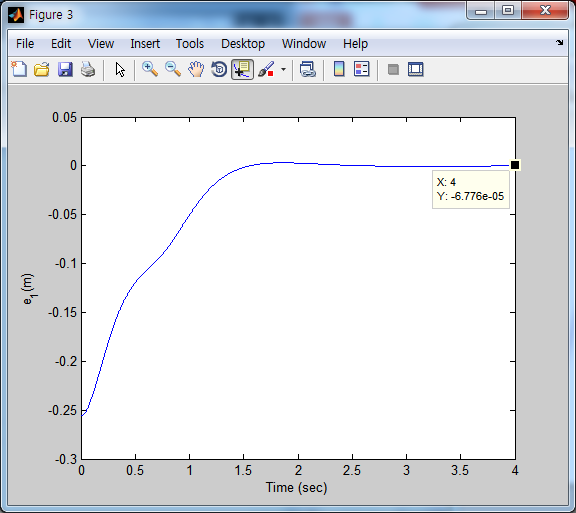
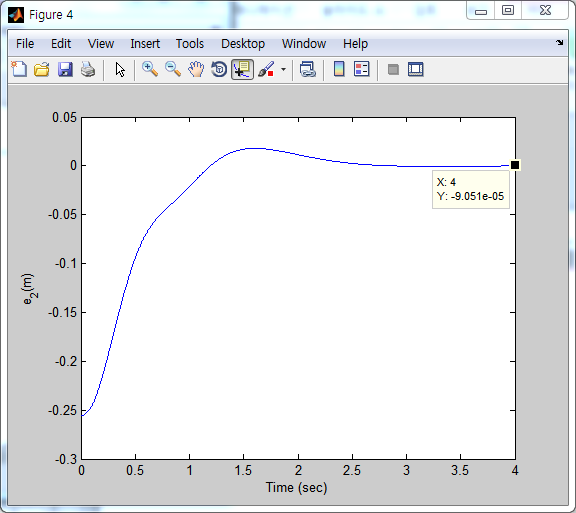
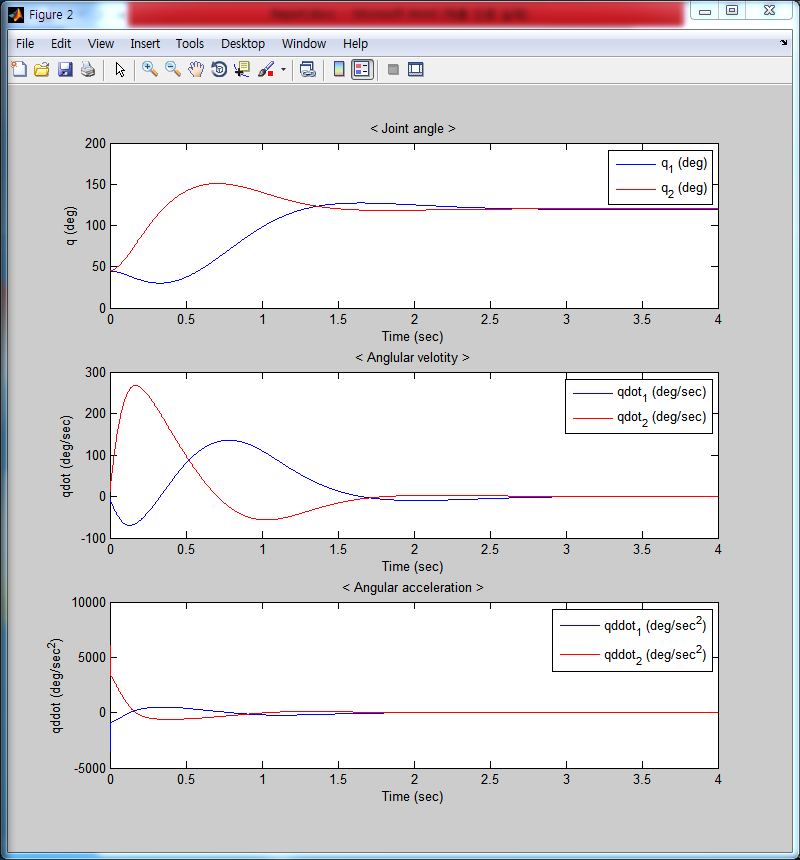
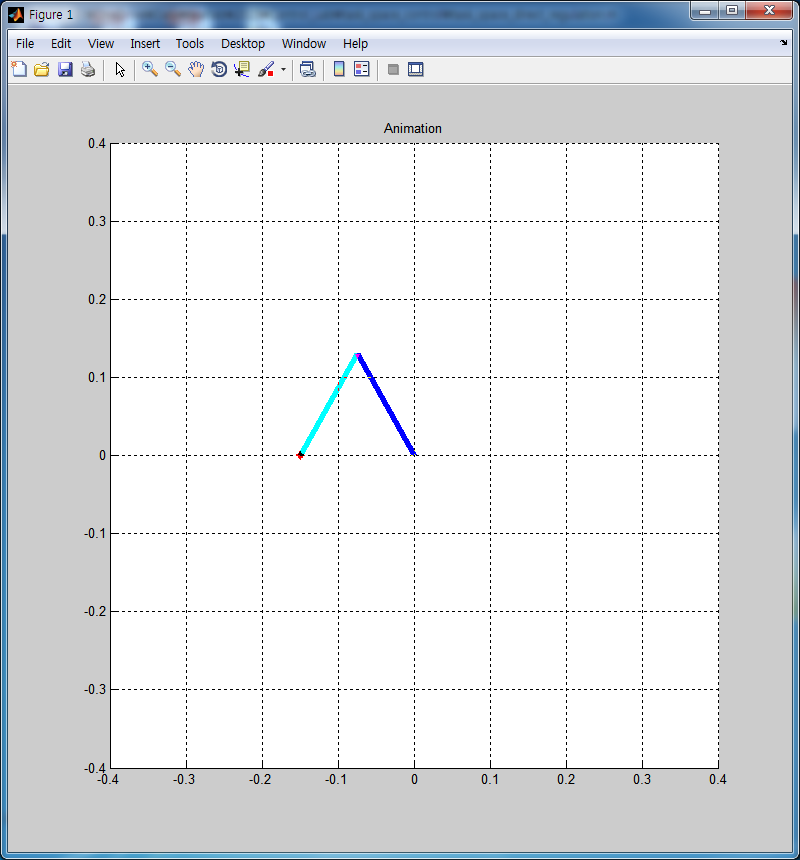
링크의 길이가 0.15m이기 때문에 링크 길이의 0.01배(1.5e-03)를 기준으로 오차의 크고 작고를 따지기로 한다. 또한 모든 control은 disturbance 및 friction이 있을 경우와 없을 경우의 gain값은 동일하다.

disturbance나 friction이 없을 때 Indirect regulation의 경우 end-effector의 최종 오차는 기준 오차보다 작기 때문에 무시할 수 있을 정도이며, disturbance와 friction이 있을 경우도 같은 이유로 무시할 수 있다. 두 경우의 오차 또한 각각 3.81e-04, -3.38e-04이므로 매우 작다. 최종 오차 뿐만이 아니라 전체 오차의 그래프 또한 유사함을 위에서 볼 수 있다. 따라서 Indirect contol은 disturbance의 영향을 덜 받는다는 것을 알 수 있다.

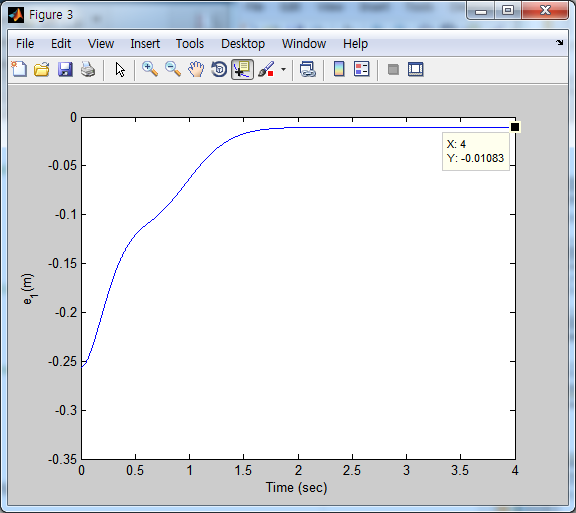
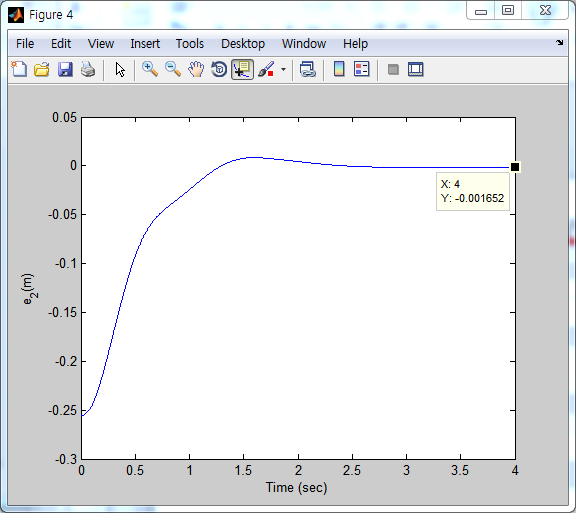
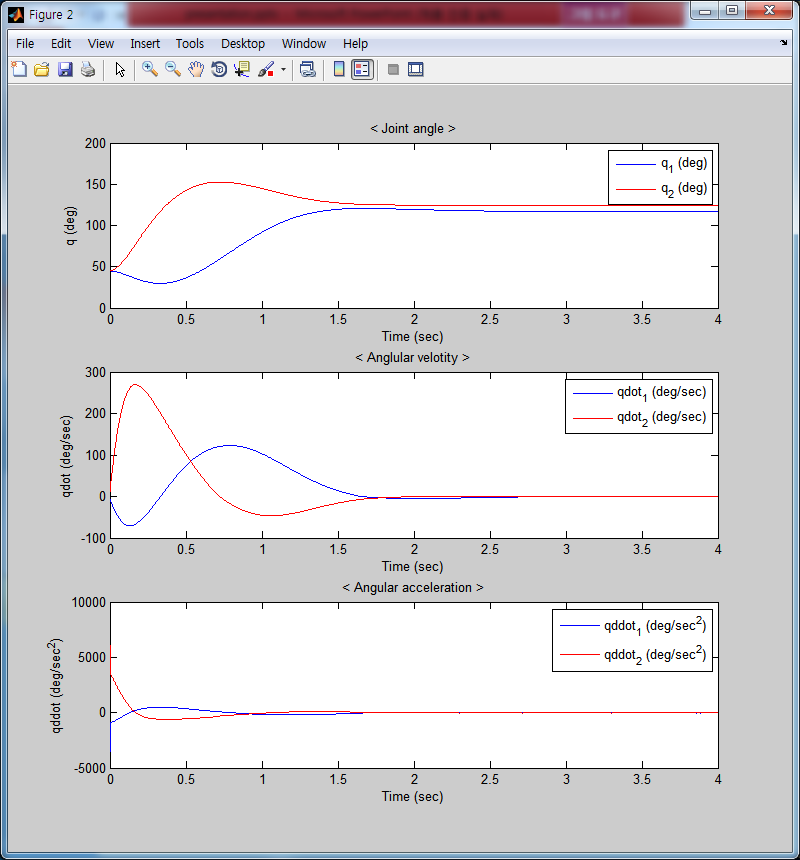
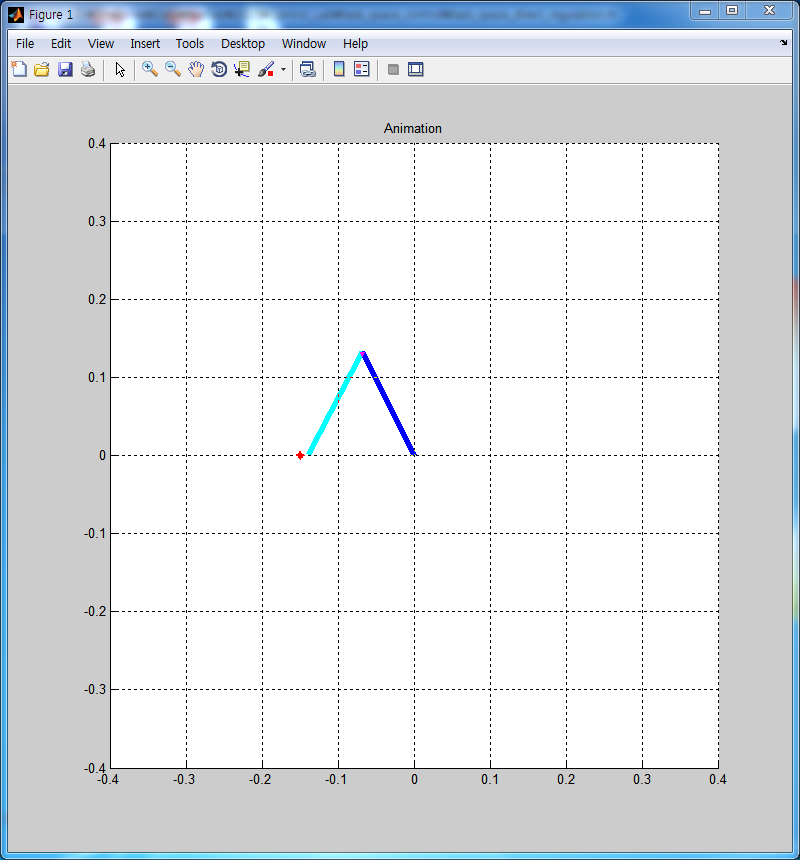
* 1. Direct Task Space Regulation Control

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Direct regulation <two\_link> | 6.78E-05 | 9.05E-05 |
| Direct regulation <two\_link\_df> | **1.08E-02** | 1.65E-03 |
| difference | **-1.08E-02** | -1.56E-03 |

two\_link.m



two\_link\_df.m



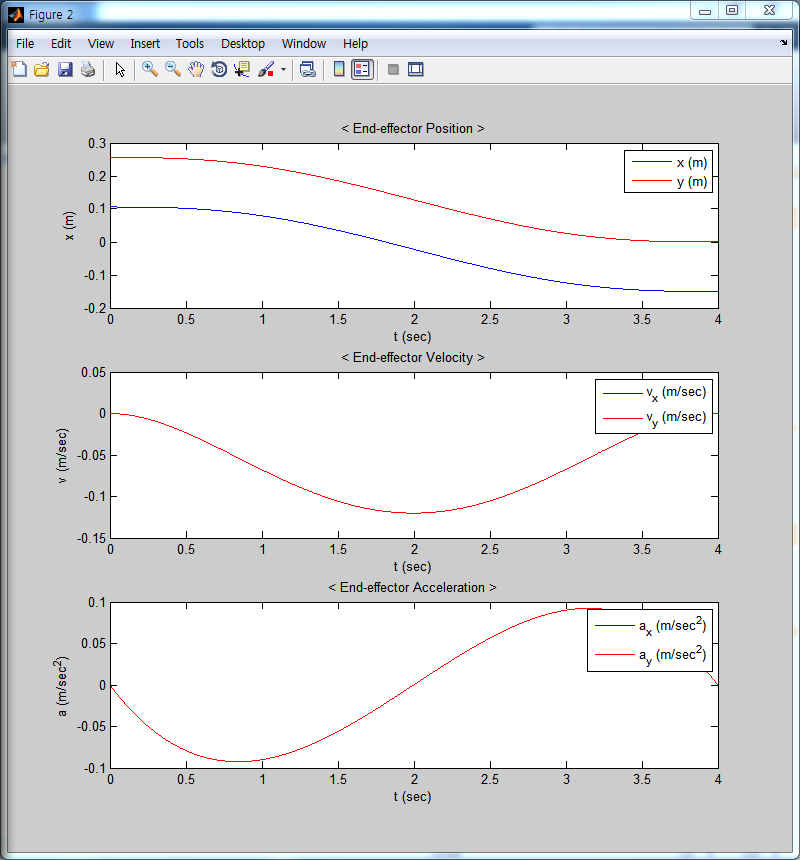
disturbance나 friction이 없을 때 Direct regulation의 경우 end-effector의 최종 오차는 각각 6.78e-05, 9.0e-05이므로, 기준 오차보다 작기 때문에 무시할 수 있다. disturbance와 friction이 있을 경우는 각각 1.08e-2, 1.65e-03으로 disturbance나 friction이 없을 때보다 오차가 많이 커짐을 알 수 있다. 특히 x\_error의 경우 기준오차 이내이다. 이러한 결과로 disturbance와 friction이 있을 때과 없을 때의 차이 또한 크다는 것을 알 수 있다.

* 1. Discuss the performance differences between Indirect and Direct task space control

Indirect regulation과 Direct regulation의 가장 큰 차이점은 Indirect가 Direct보다 gain이나 외부 환경의 변화에 대하여 영향을 덜 받는 다는 점이다. 위의 표 2개를 보면 Indirect regulation의 disturbance 및 friction이 있을 때와 없을 때의 error간 차이의 경우 -3.81e-04, -3.38e-04이고, Direct regulation의 경우는 각각 -1.08e-02, 1.65e-03이다. 이로부터 Indirect가 disturbance에 영향을 덜 받는다는 것을 알 수 있다.

**Problem 2**

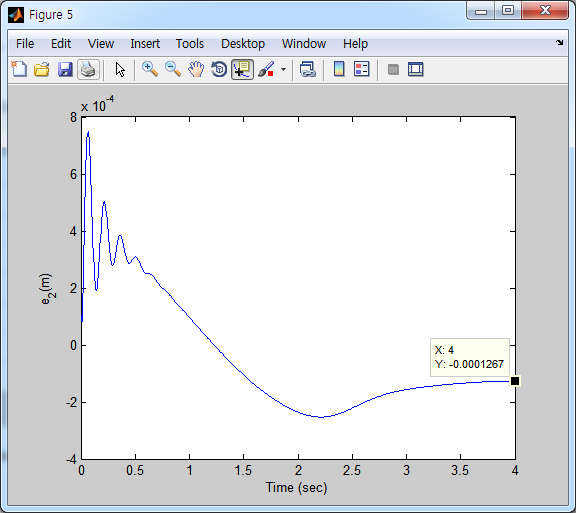
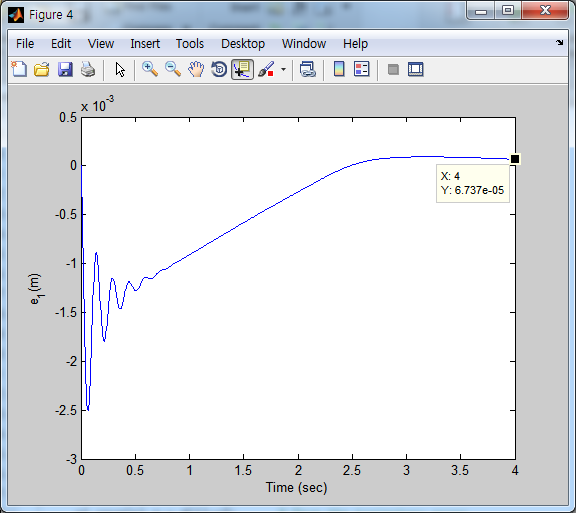
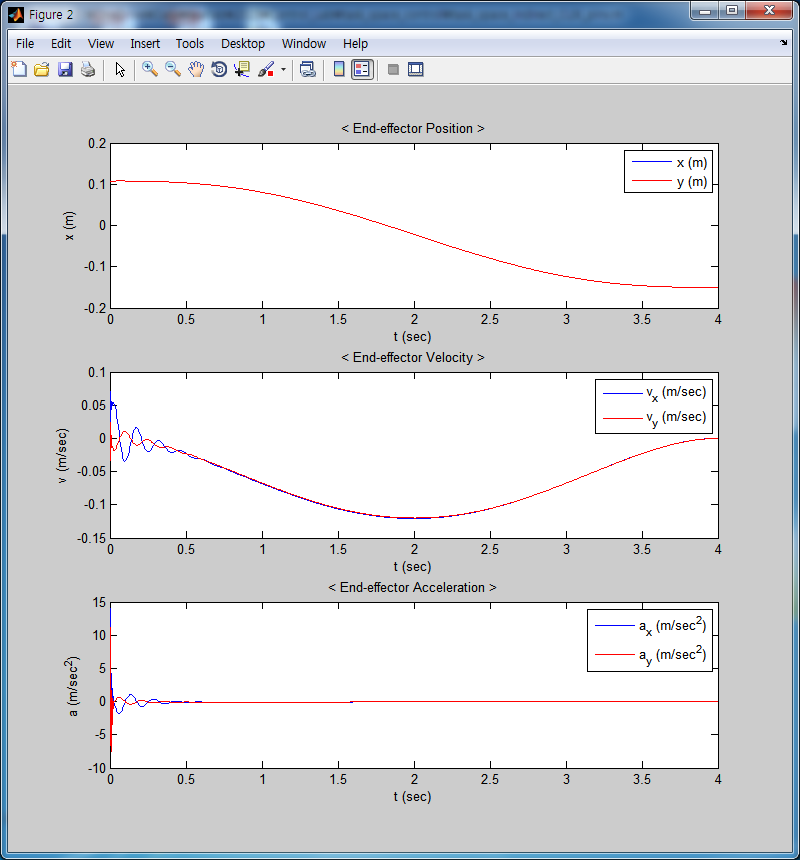
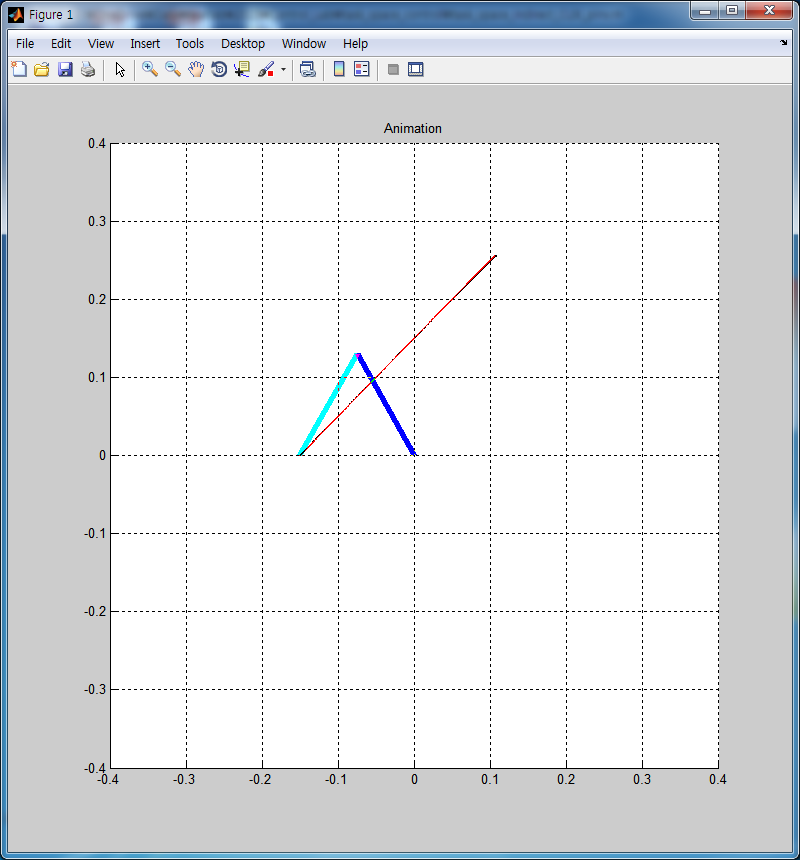
2.1 Generate desired trajectory from xd(0) to xd(4) with elapsed time 4[s] using 5-th order polynomial



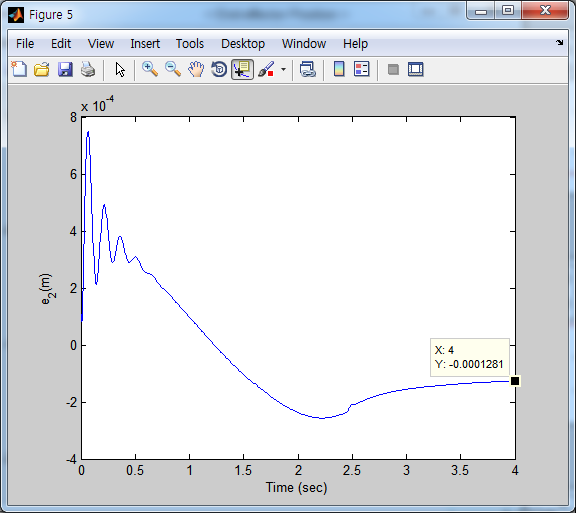
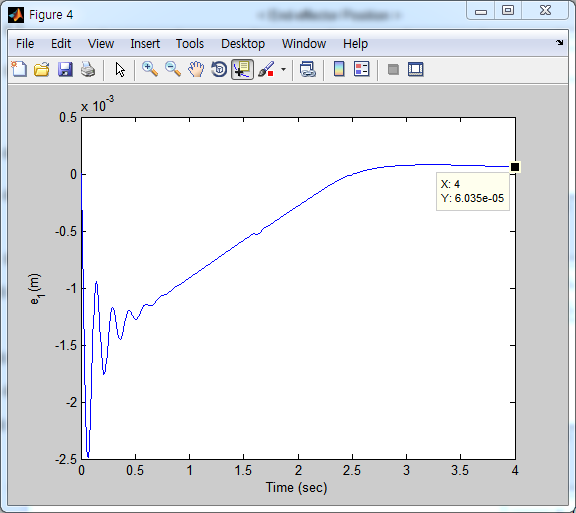
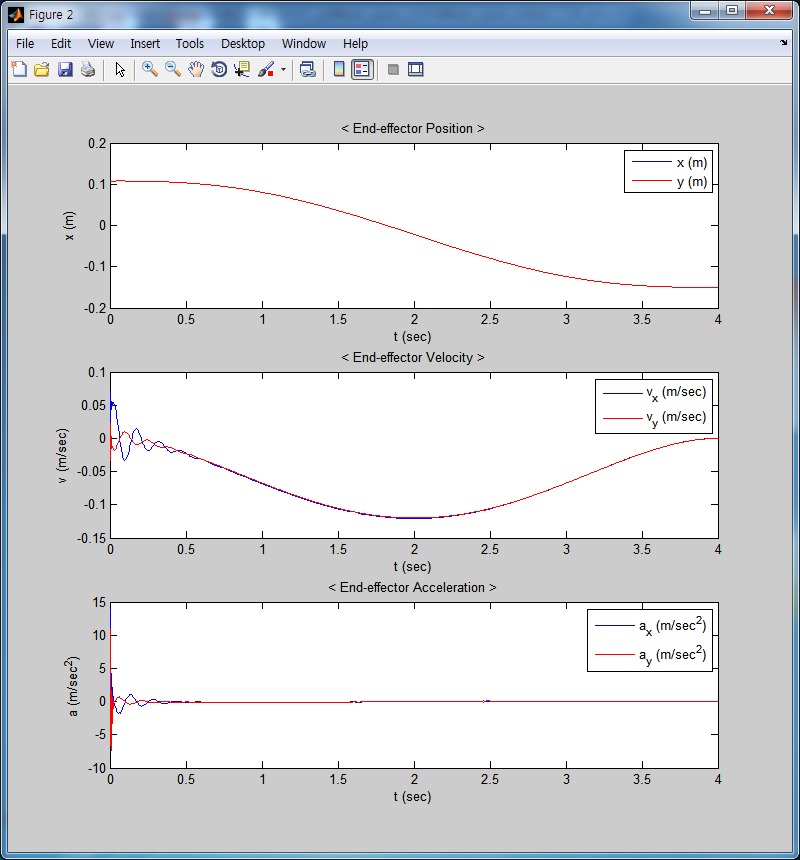
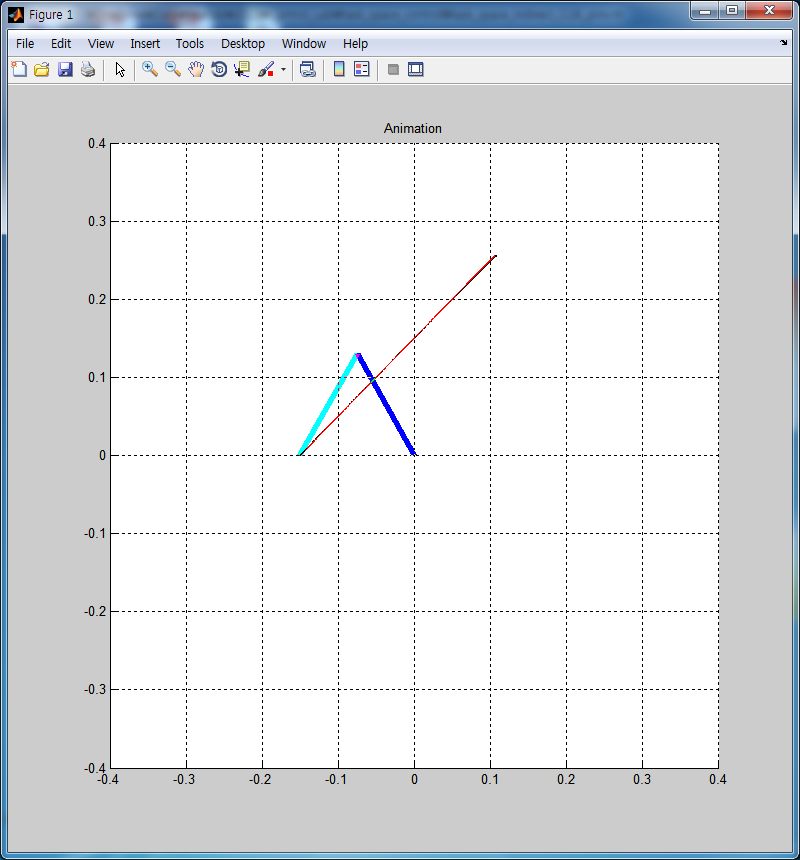
2.2 Indirect Task Space Control using CLIK and Jacobian Pseudoinverse

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Indirect CLIK&Pinv <two\_link> | 6.74E-05 | 1.27E-04 |
| Indirect CLIK&Pinv <two\_link\_df> | 6.04E-05 | 1.28E-04 |
| difference | 7.02E-06 | -1.40E-06 |

two\_link.m



two\_link\_df.m

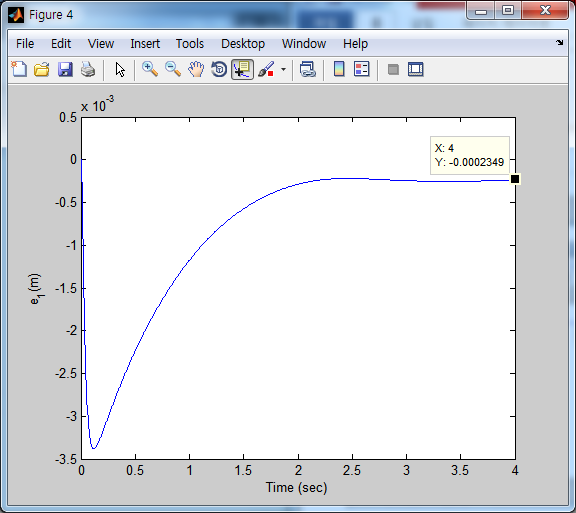
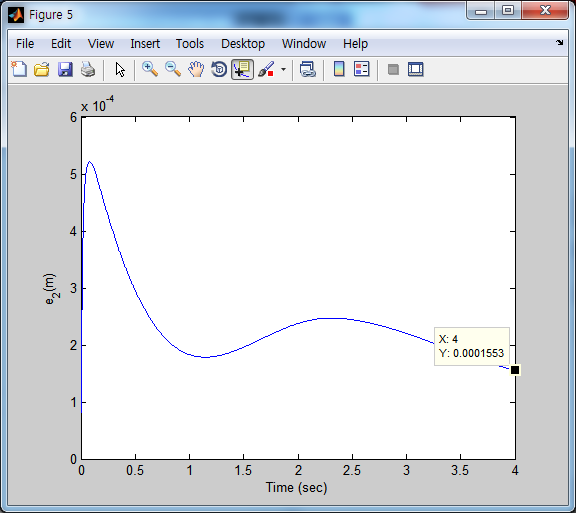
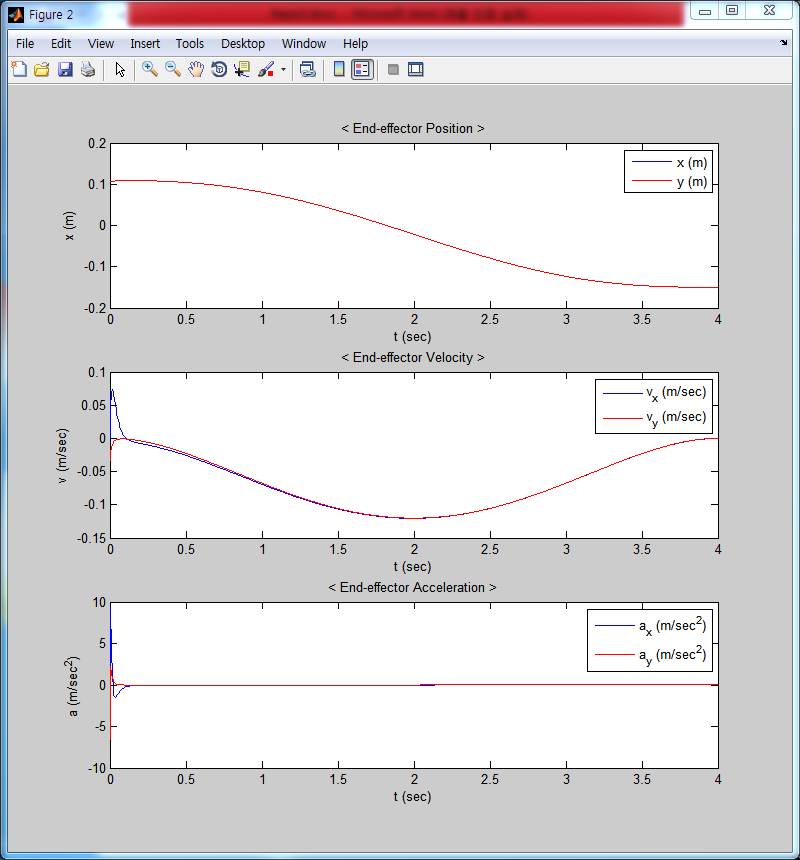
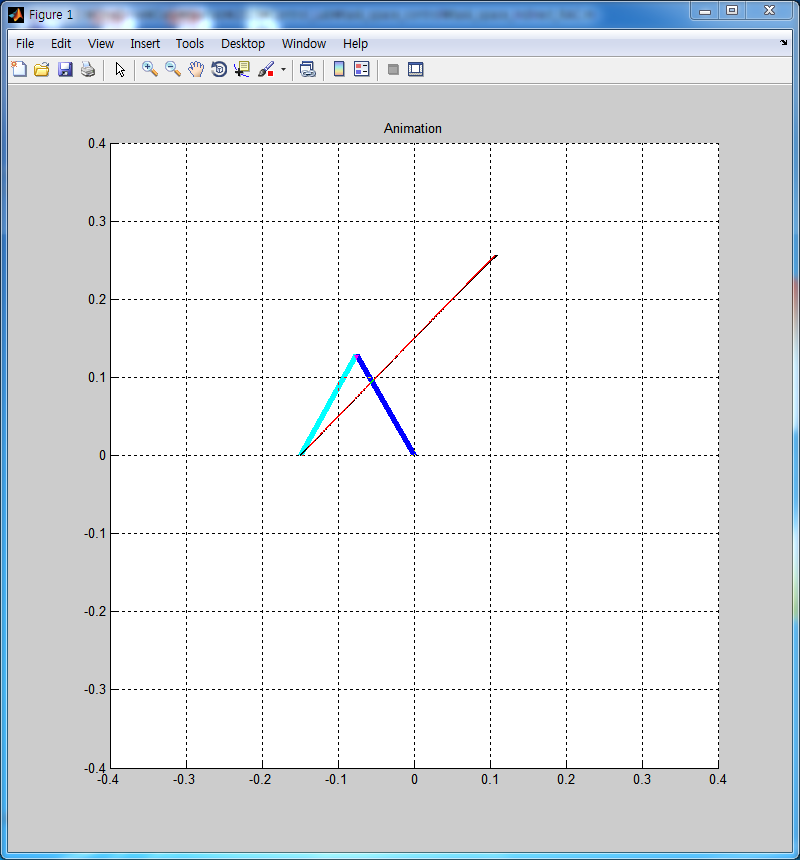


disturbance나 friction이 없을 때 문제에 해당하는 제어를 할 경우 end-effector의 최종 오차는 6.74e-05, 1.27e-04로 기준 오차보다 작기 때문에 무시할 수 있을 정도이며, disturbance와 friction이 있을 경우도 각각 6.04e-05, 1.28e-04이므로 같은 이유로 무시할 수 있다. 최종 오차 뿐만이 아니라 전체 오차의 그래프 또한 유사함을 위에서 볼 수 있다. 따라서 Indirect Task Space Control using CLIK and Jacobian Pseudoinverse 은 disturbance의 영향을 덜 받는다는 것을 알 수 있다. CLIK를 이용한 제어에 특이사항이 있다면 초기에 진동이 발생하는데 이는 gain값을 잘 조정하더라도 진폭만 변할 뿐 사라지게 하지는 못하였다. 이를 통하여 토크가 많이 들 것이라고 예측할 수 있다.

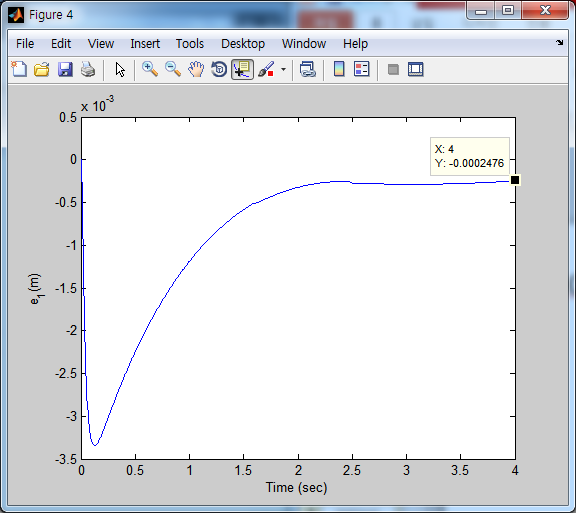
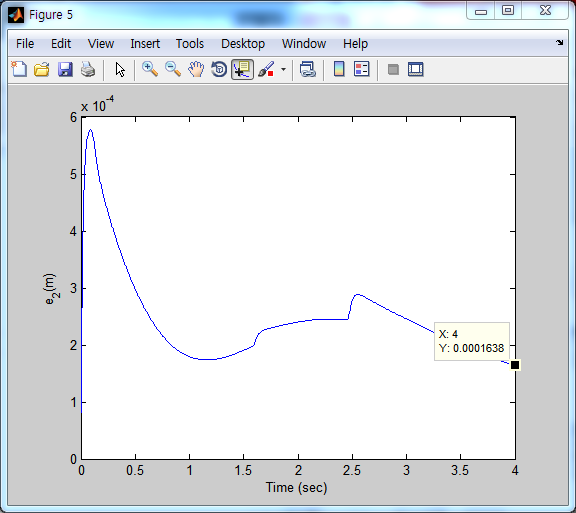
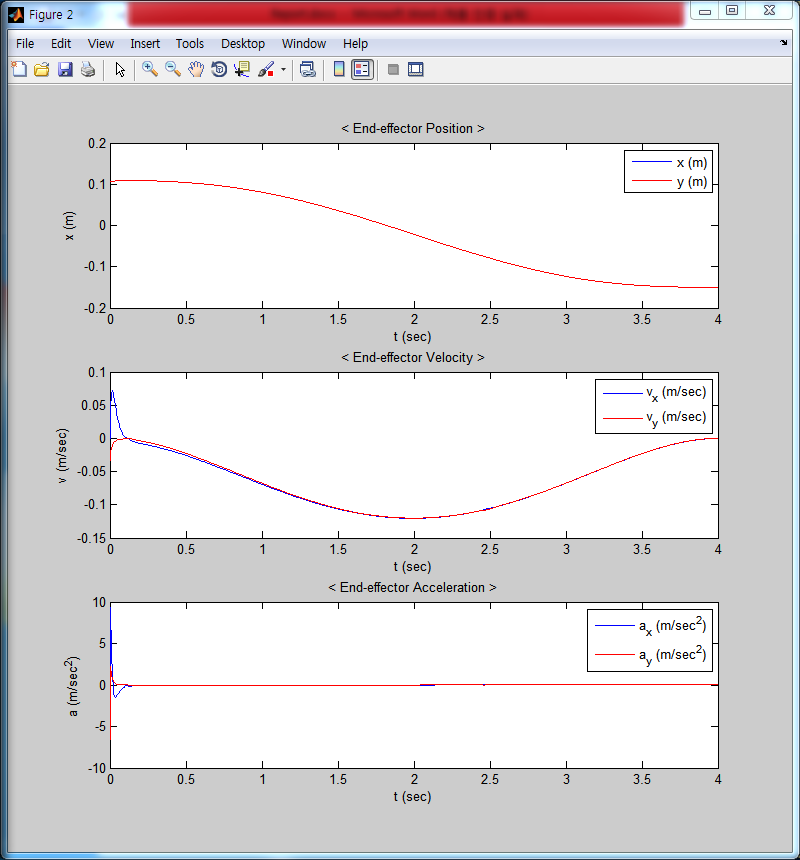
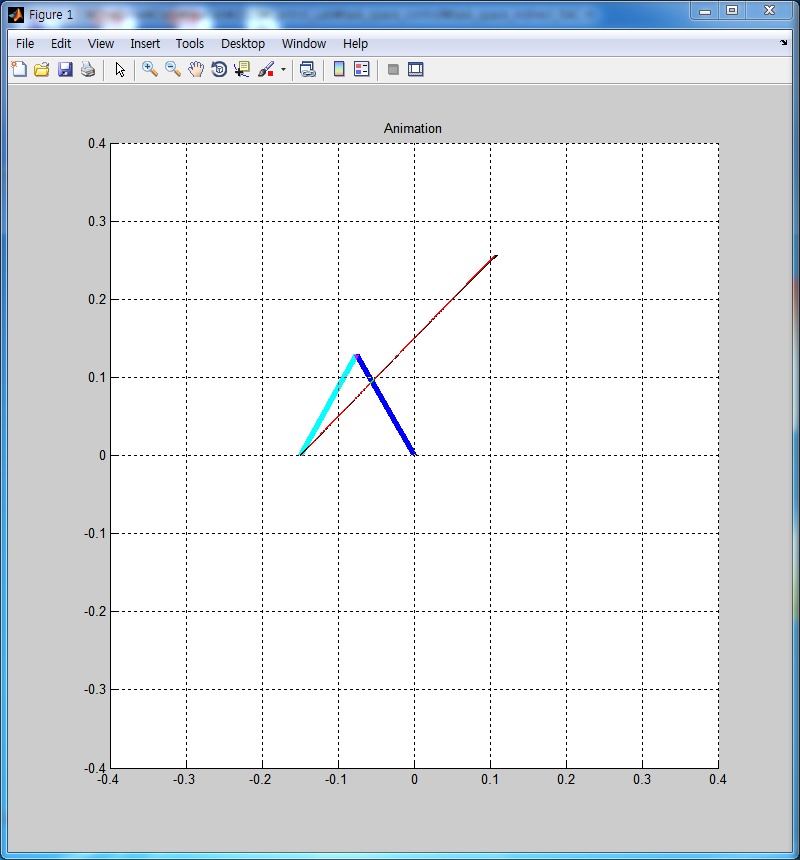
2.3 Indirect Task Space RAC scheme using joint-space IDC scheme

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Indirect RAC <two\_link> | 2.35E-04 | 1.55E-04 |
| Indirect RAC <two\_link\_df> | 2.48E-04 | 1.64E-04 |
| difference | -1.27E-05 | -8.50E-06 |

two\_link.m



two\_link\_df.m

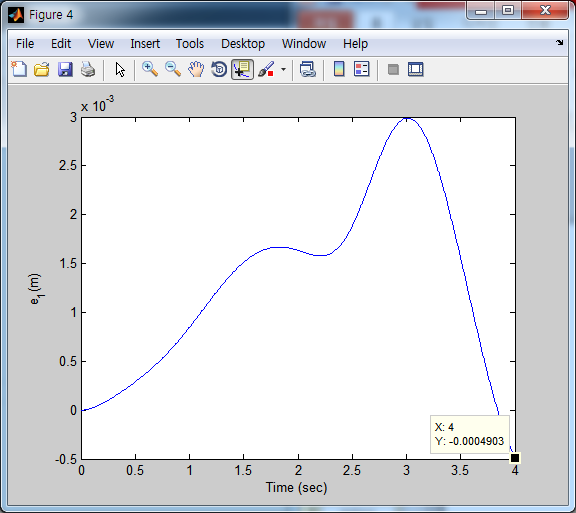
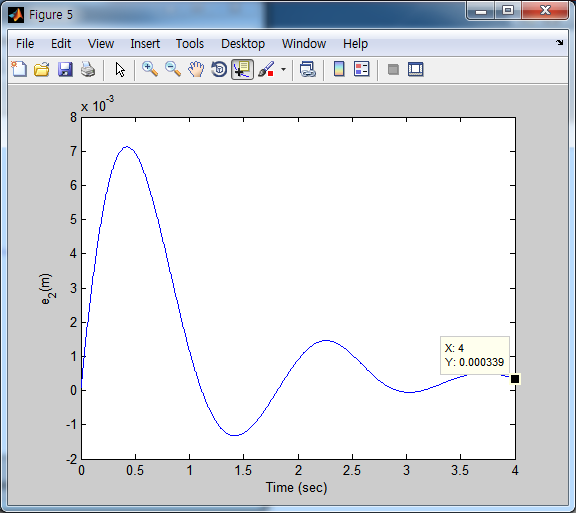
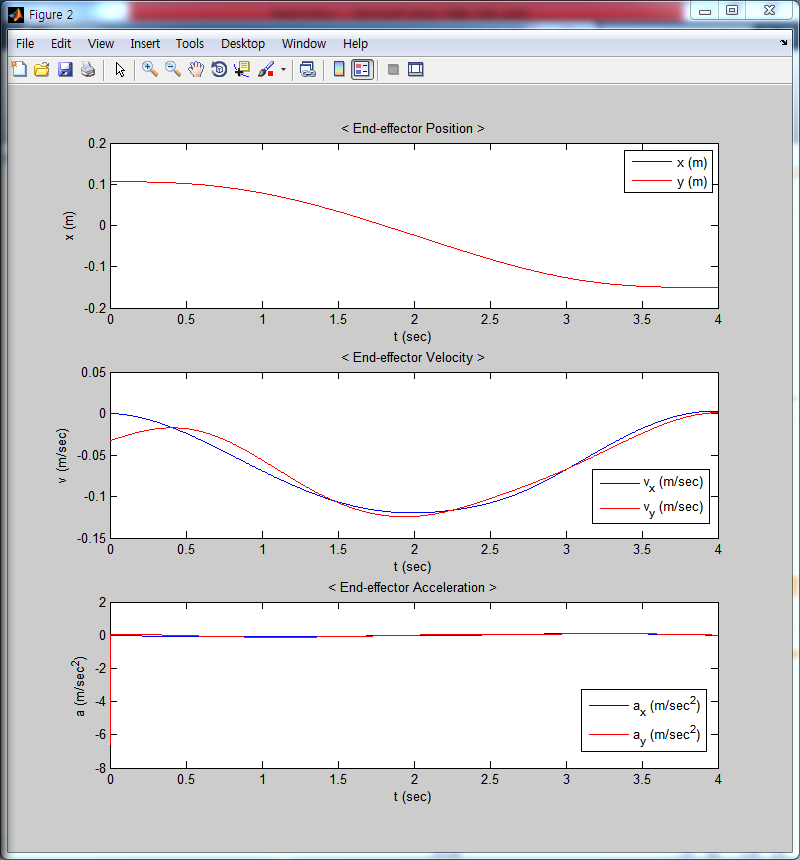
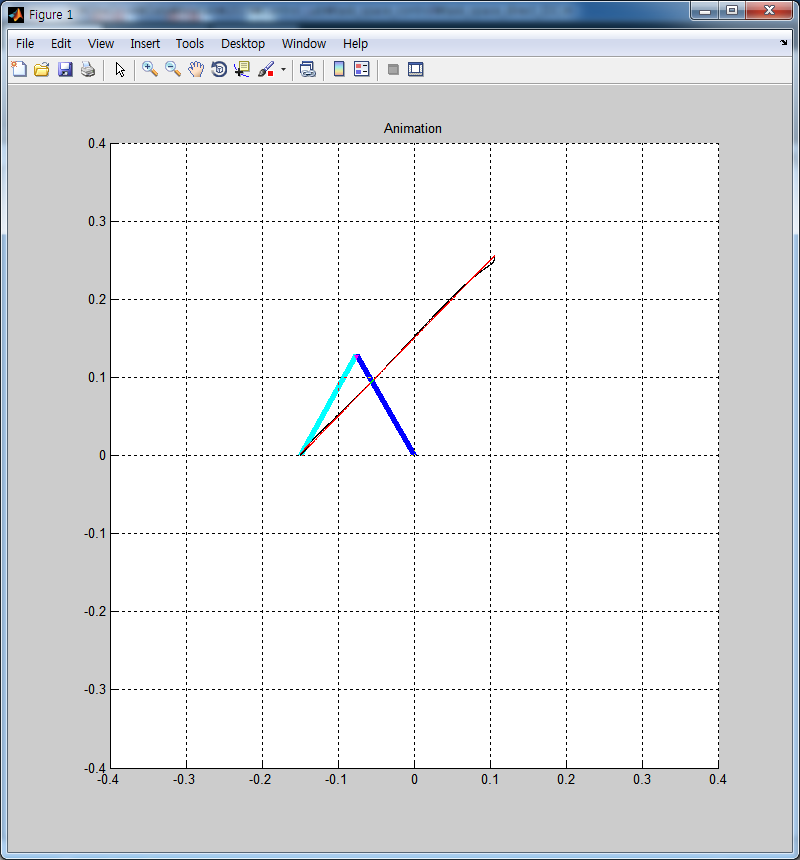


disturbance나 friction이 없을 때 Indirect RAC scheme을 이용할 경우 end-effector의 최종 오차는 2.35e-04, 2.48e-04로 기준 오차보다 작기 때문에 무시할 수 있을 정도이며, disturbance와 friction이 있을 경우의 오차 또한 각각 1.55e-04, 1.64e-04이므로 같은 이유로 무시할 수 있다. 전체 오차의 그래프에서는 약간의 차이를 보인다. y방향의 error의 경우 disturbance와 friction이 있을 경우 중간 중간 끊어지는 형태를 보이는데 이는 friction에 의함이라고 생각할 수 있다. 이는 단지 y방향의 속도가 0근처이기 때문에 생기는 차이이며 결론적으로는 Indirect Task Space RAC scheme using joint-space IDC scheme은 disturbance의 영향을 덜 받는다는 것을 알 수 있다.

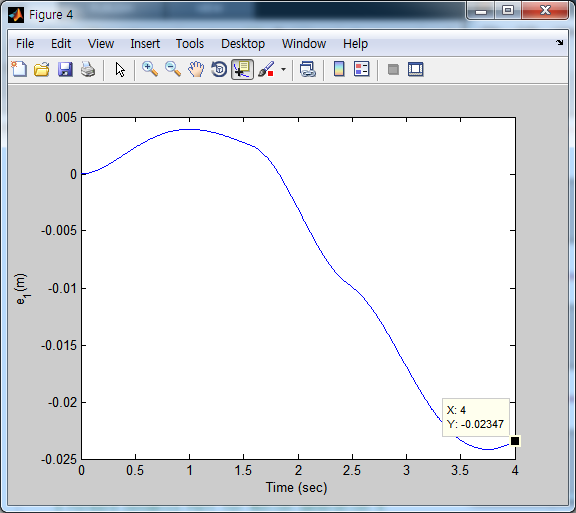
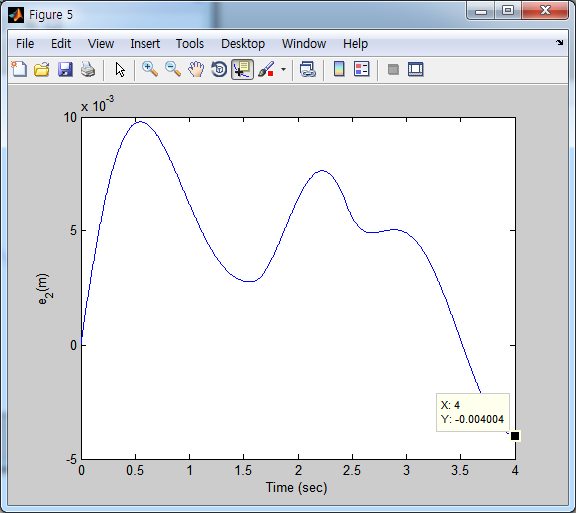
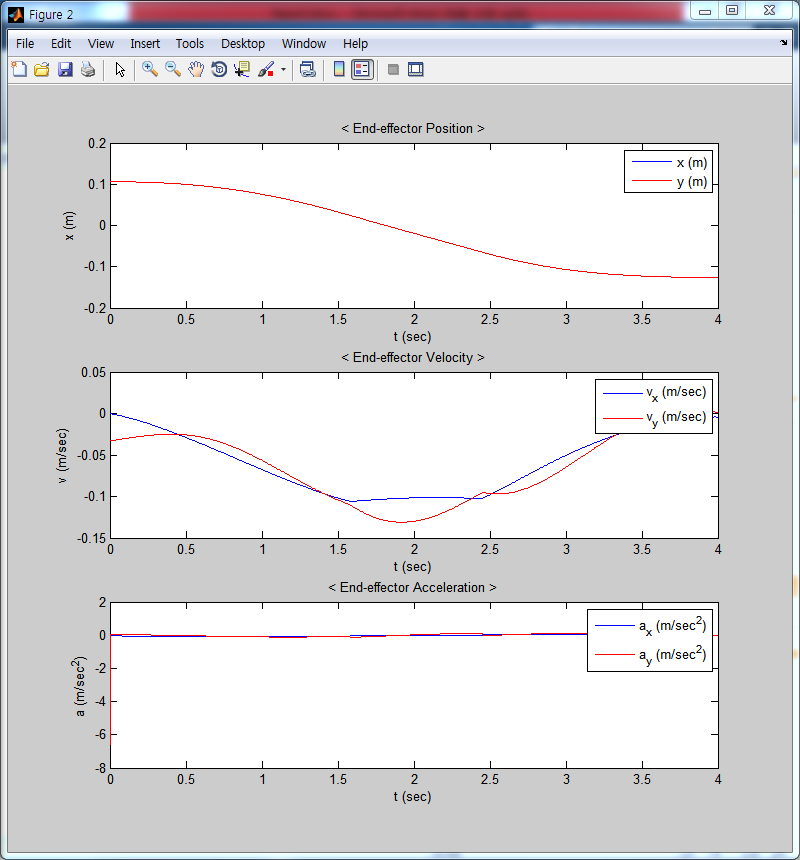
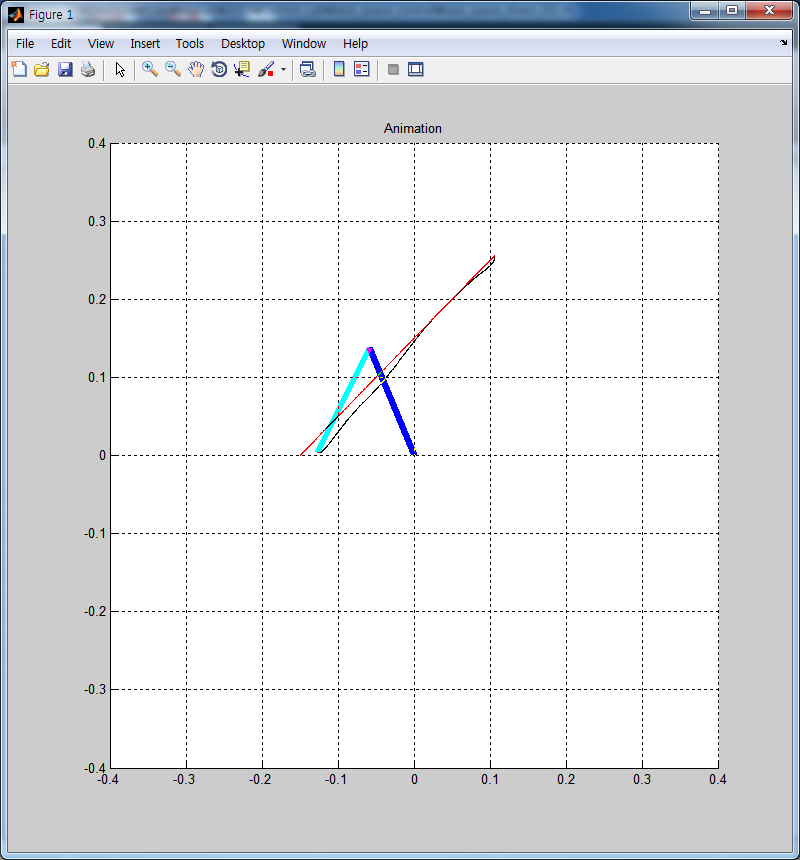
2.4 Direct Task Space IDC

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Direct IDC <two\_link> | 4.90E-04 | 3.39E-04 |
| Direct IDC <two\_link\_df> | **2.35E-02** | 4.00E-03 |
| difference | **-2.30E-02** | -3.67E-03 |

two\_link.m



two\_link\_df.m

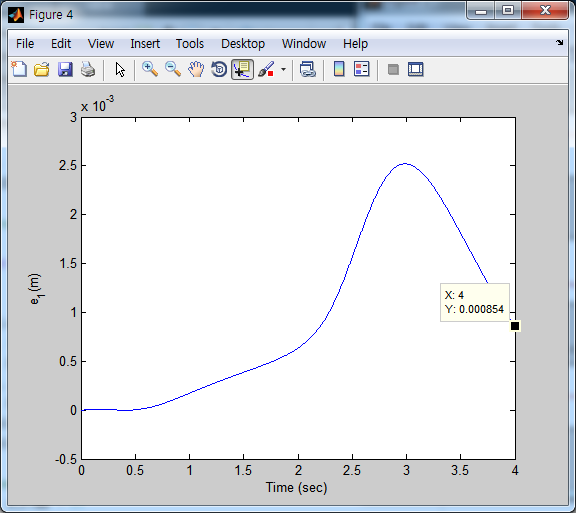
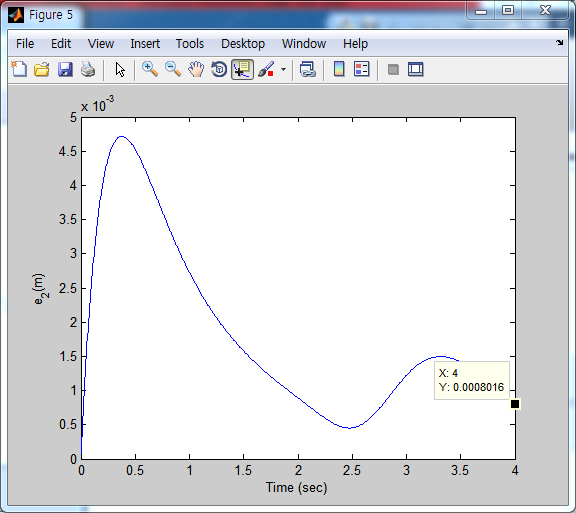
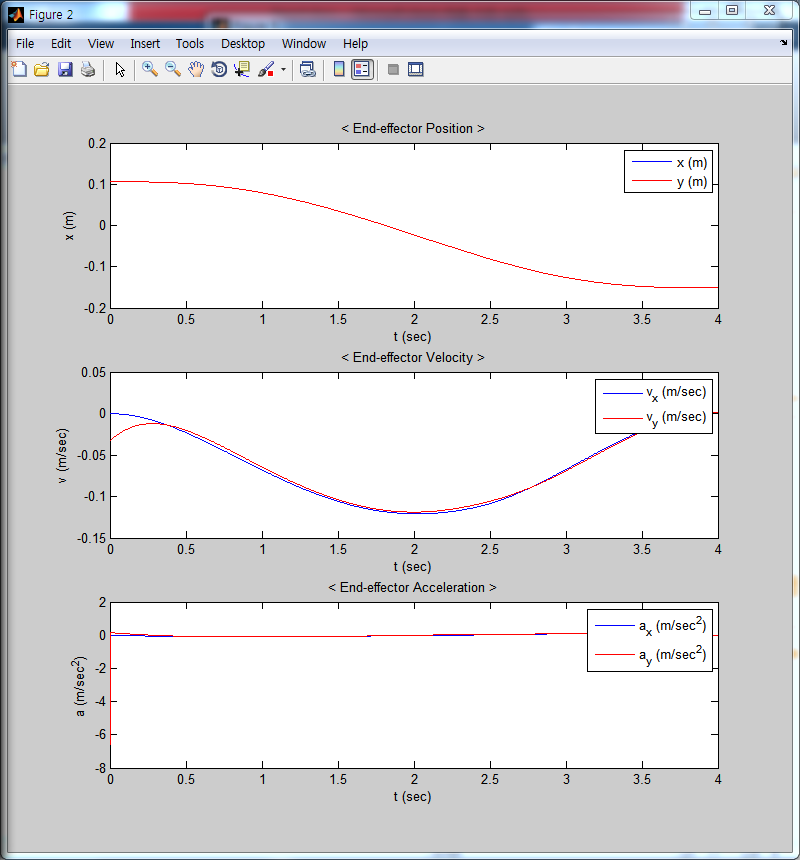
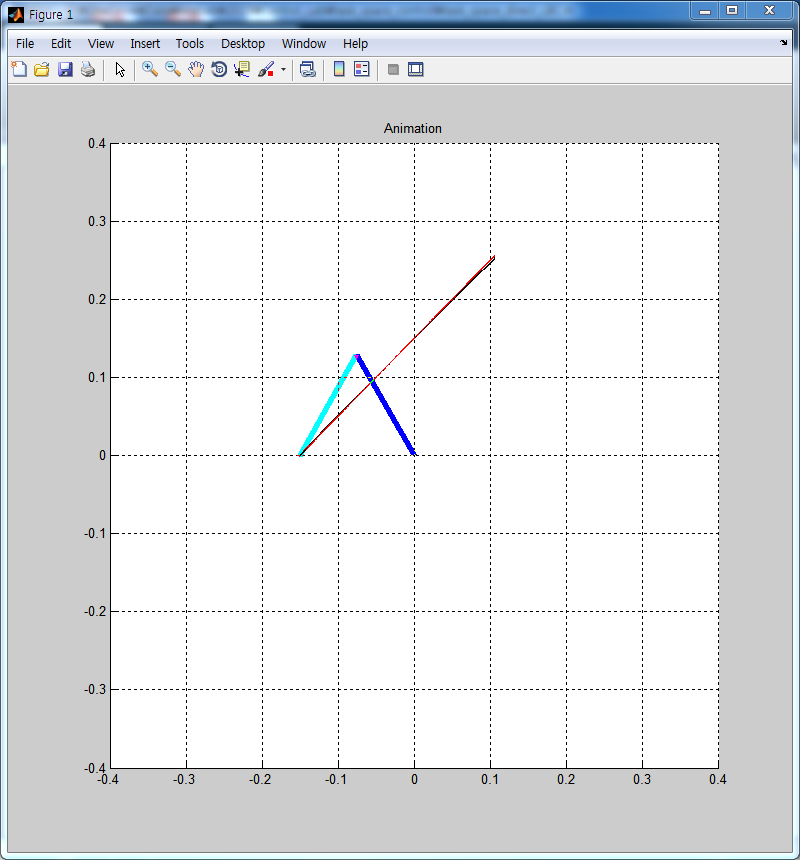


disturbance나 friction이 없을 때 Direct Task Space IDC의 경우 end-effector의 최종 오차는 각각 4.90e-4, 3.39e-4이므로, 기준 오차보다 작기 때문에 무시할 수 있다. disturbance와 friction이 있을 경우는 각각 2.35e-2, 4.00e-03으로 disturbance나 friction이 없을 때보다 오차가 많이 커짐을 알 수 있다. 특히 x\_error의 경우 기준오차 이내이다. 이는 two-link의 경로에서도 차이를 볼 수 있을 정도 이다. 또한 x\_error의 그래프에서도 큰 차이를 보이는데 이는 constant disturbance 때문이다.

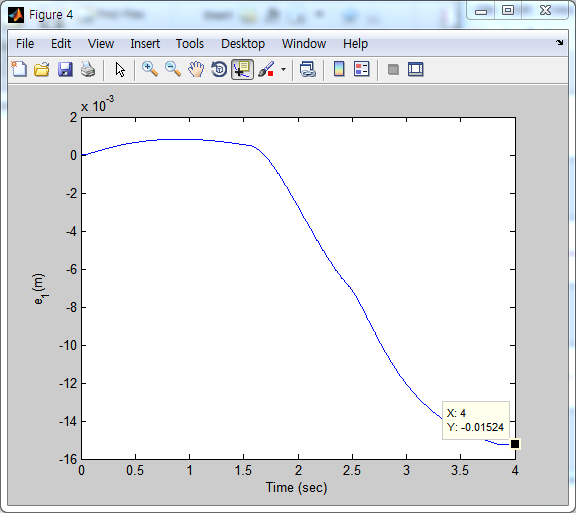
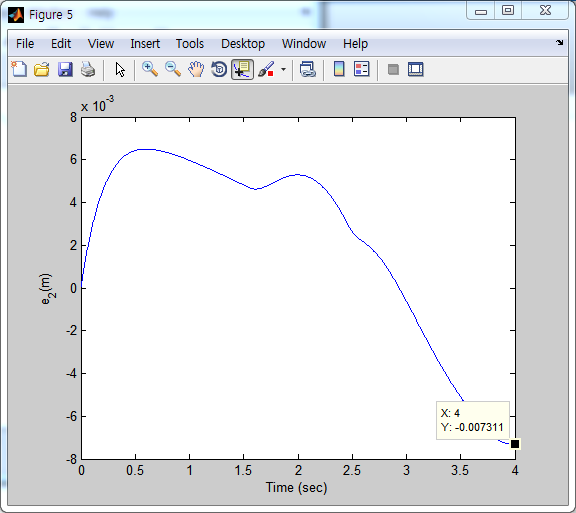
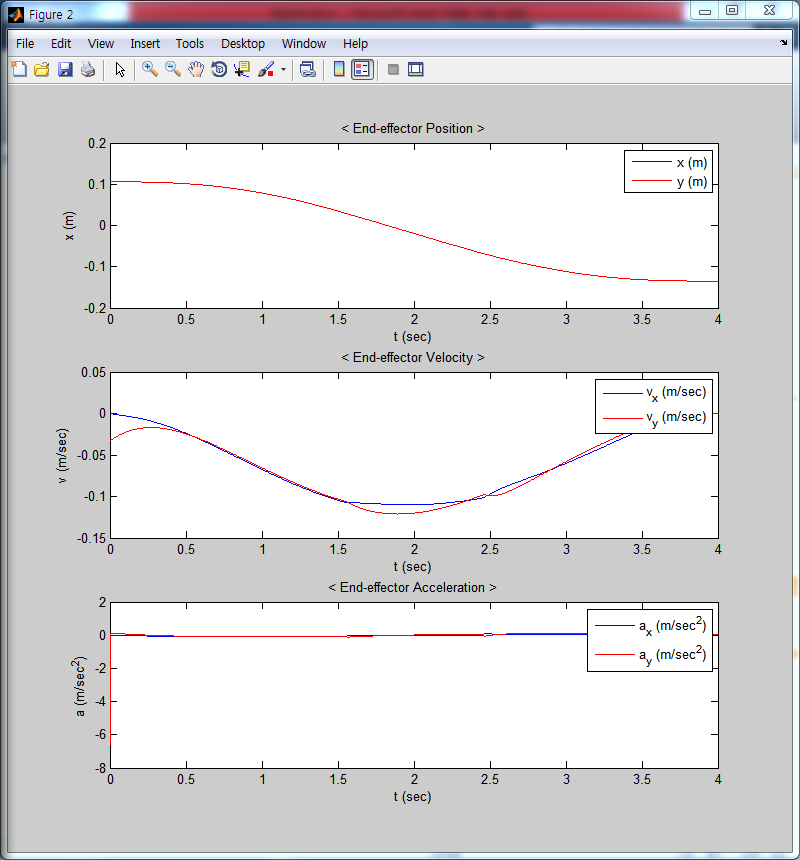
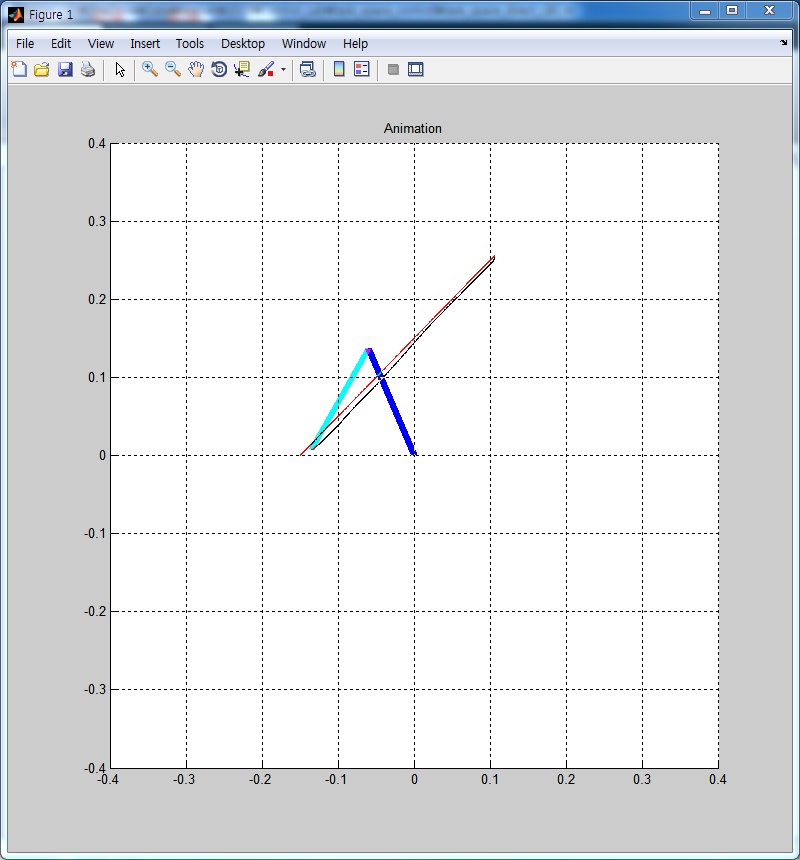
2.5 Direct Task Space LBC

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Direct LBC <two\_link> | 8.54E-04 | 8.02E-04 |
| Direct LBC <two\_link\_df> | **1.52E-02** | 7.31E-03 |
| difference | **-1.44E-02** | -6.51E-03 |

two\_link.m



two\_link\_df.m



disturbance나 friction이 없을 때 Direct Task Space LBC의 경우 end-effector의 최종 오차는 각각 8.54e-4, 8.02e-4이므로, 기준 오차보다 작기 때문에 무시할 수 있다. disturbance와 friction이 있을 경우는 각각 1.52e-2, 7.31e-03으로 disturbance나 friction이 없을 때보다 오차가 많이 커짐을 알 수 있다. 특히 x\_error의 경우 기준오차 이내이다. 이는 two-link의 경로에서도 차이를 볼 수 있을 정도 이다. 또한 x\_error의 그래프에서도 큰 차이를 보이는데 이는 constant disturbance 때문이다. 이런 그래프의 경향은 Direct Task Space IDC와 같다.

2.6 Discuss the performance differences between Indirect and Direct task space control

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Indirect CLIK&Pinv <two\_link> | 6.74E-05 | 1.27E-04 |
| Indirect CLIK&Pinv <two\_link\_df> | 6.04E-05 | 1.28E-04 |
| difference | 7.02E-06 | -1.40E-06 |
| Indirect RAC <two\_link> | 2.35E-04 | 1.55E-04 |
| Indirect RAC <two\_link\_df> | 2.48E-04 | 1.64E-04 |
| difference | -1.27E-05 | -8.50E-06 |

|  |  |  |
| --- | --- | --- |
|  | x\_error | y\_error |
| Direct IDC <two\_link> | 4.90E-04 | 3.39E-04 |
| Direct IDC <two\_link\_df> | **2.35E-02** | 4.00E-03 |
| difference | **-2.30E-02** | -3.67E-03 |
| Direct LBC <two\_link> | 8.54E-04 | 8.02E-04 |
| Direct LBC <two\_link\_df> | **1.52E-02** | 7.31E-03 |
| difference | **-1.44E-02** | -6.51E-03 |

위의 표는 2.2~2.5까지 4개의 task space controller의 최종위치 에러 값을 나타낸 것이다. 이를 보면 Indirect에 비해 Direct의 경우가 disturbance와 friction이 있을 때와 없을 때의 에러 간의 차이가 더 큰 것을 알 수 있다. 이는 1번의 Regulation의 결과와 같다.

* 1. Select the best controller and suggest the logical basis on your choice

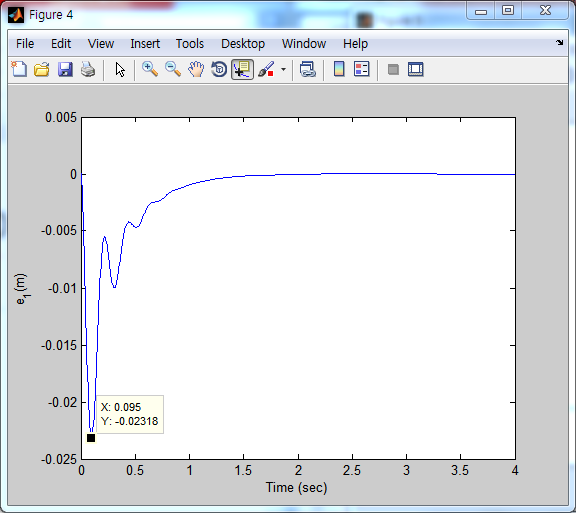
2.6의 결과에 따라 Direct보다 Indirect가 disturbance나 friction에 영향을 덜 받는 것으로 보아 Indirect가 더 좋은 제어기라고 가정하였다. 따라서 Indirect인 tacking controller는 CLIK&Pseudoinv 와 Indirect RAC 둘을 비교하게 되었다. 또한 좋은 제어기란 gain 값의 변화에 영향을 덜 받는 것이라고 가정하였기 때문에 gain 값 중 하나만 변화시키면서 기준(진폭의 크기:0.02)을 통해 gain의 max와 min값을 결정하였다.

1) Indirect Task Space Control using CLIK and Jacobian Pseudoinverse

* 결과 : Kp, Kd에 민감하게 반응

|  |  |  |
| --- | --- | --- |
|  | min | max |
| **kp** |  | **3** |
| **kd** | **0.2** | **1.5** |
| k | 5 | 400 |

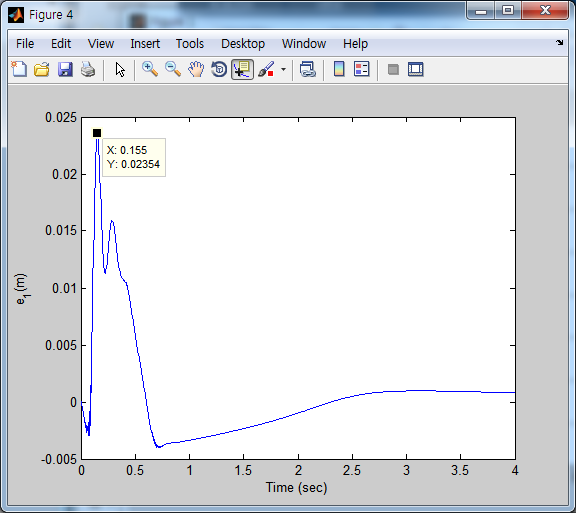
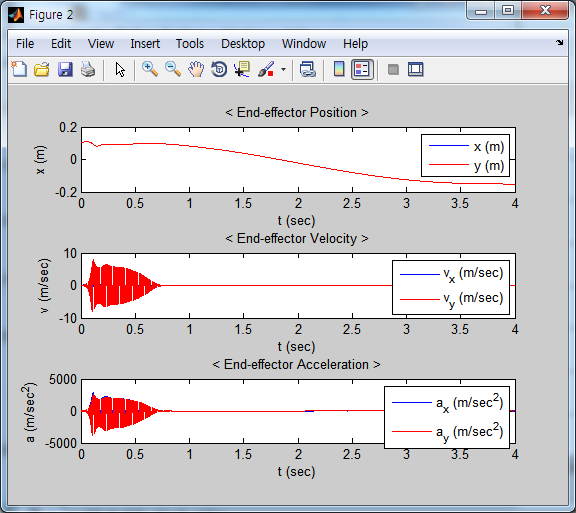
1. kp = 3



1. kd = 0.2



1. kd = 1.5



진폭이 0.02이하를 만족시키는 kp의 최댓값은 3이고, 최솟값은 0이다.

진폭이 0.02이하를 만족시키는 kd의 최솟값은 약 0.2이고, kd는 약 1.5이다. Kd가 1.5일경우 진폭의 크기 뿐만 아니라 end-effector의 속도 및 가속도도 문제가 된다. 따라서 실제 사용할 수 있는 kd의 값은 1.5보다 작을 것으로 예상된다.

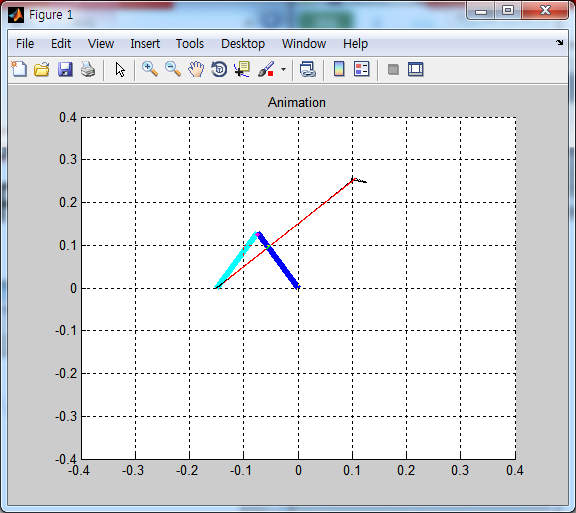
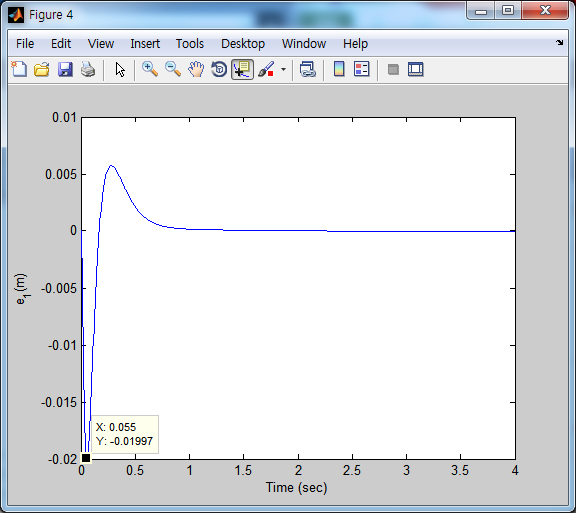
진폭이 0.02이하를 만족시키는 k의 값은 표와 같이 측정되었으나 kp나 kd보다 범위가 100배 이상 크므로 생각하지 않기로 한다.

2) Indirect Task Space RAC scheme using joint-space IDC scheme

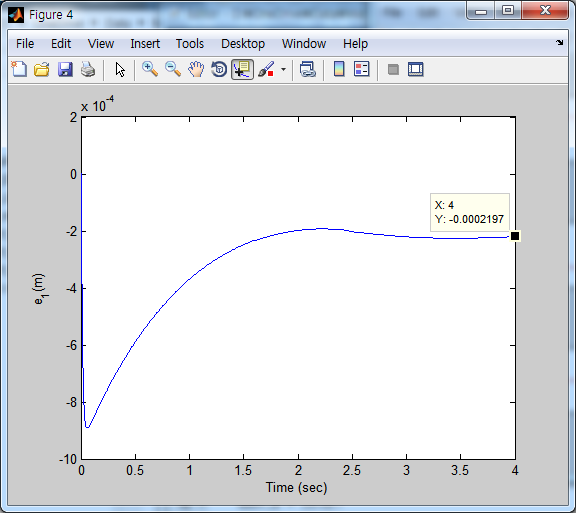
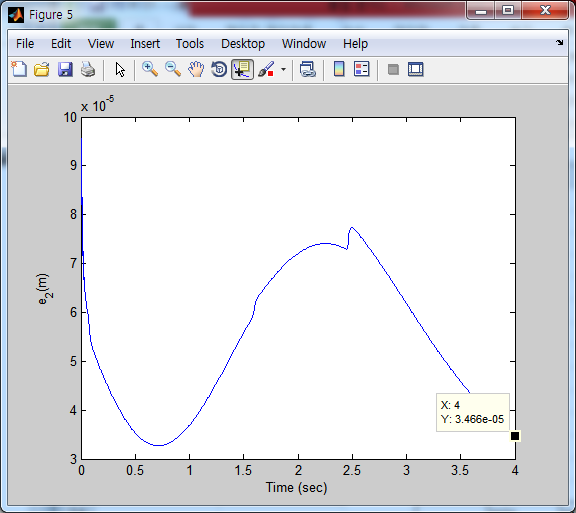
* 결과 : CLIK보다 kp,kp의 범위가 넓다

|  |  |  |
| --- | --- | --- |
|  | min | max |
| kp |  | 400 |
| kd | 10 | (150) |

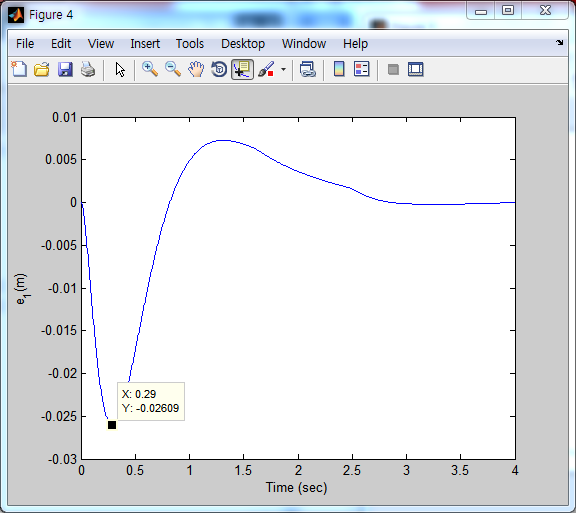
1. kp = 400



1. kd = 150



1. kd = 10



진폭이 0.02이하를 만족시키는 kp의 최솟값은 0이고 최댓값은 약 400이다.

진폭이 0.02이하를 만족시키는 kd의 최솟값은 약 10이고, 최댓값은 측정하지 못하였다. Kd가 400일 경우 end-effector의 position이 -2.2e-3, 3.5e-5이다. 이는 kd가 50일때(원래 kd값)보다 더 작은 값으로 kd가 커질수록 error가 더 작아짐을 알 수 있다. 따라서 kd의 최댓값은 150보다는 클 것이라는 것을 예측할 수 있다. 하지만 Kd가 150을 넘어갈 경우 MATLAB에서 연산을 하지 못하기 때문에 측정하지 못하였다.

3) 결론

CLIK보다 RAC가 더 성능이 좋다는 결론이 내려졌다. CLIK는 kd와 kp에 너무 민감한 반응을 보였고 gain의 범위가 kp는 약 3, kd는 약 1.5정도 밖에 되지 않았다. 반면에 RAC의 경우, kp의 범위는 약 400, kd의 범위는 100이상이므로 CLIK보다 약 100배 정도의 범위를 갖는 것을 알 수 있고 이러한 정보들로부터 RAC가 가장 좋은 성능을 가진 Task Space Controller라는 결론을 내렸다.

**task\_space\_indirect\_regulation.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.005;

% Terminal time %

t0=0; tf=4;

% Variable for converting RADIAN -> DEGREE or DEGREE -> RADIAN

R2D = 180/pi;

D2R = pi/180;

% Initial Configurations defined in RADIAN %

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

qdot=[0; 0];

qddot=[0; 0];

% start position

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)];

% set position

x\_d = [-0.15; 0];

% Initial Input %

u = [0;0];

q\_d = [0; 0];

% PD gain

kp = 0.5; kd = 0.3; k = 50; % two\_link & two\_link\_df

K = [k, 0; 0, k];

Kp = [kp, 0; 0, kp];

Kd = [kd, 0; 0, kd];

% Iteration number %

n=1;

% Plot Setting %

figure(1)

title('Animation')

hold on

axis([-0.4 0.4 -0.4 0.4]);

grid

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

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

%[t,y] = ode45('two\_link',[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);

% Forward Kinematics %

x1 = l1\*cos(q(1));

y1 = l1\*sin(q(1));

x2 = x1 + l2\*cos(q(1)+q(2));

y2 = y1 + l2\*sin(q(1)+q(2));

x = [x2;y2]; % end-effector position

% Solve the IK using Jacobian transpose

Ja = [-l1\*sin(q(1))-l2\*sin(q(1)+q(2)) -l2\*sin(q(1)+q(2));

l1\*cos(q(1))+l2\*cos(q(1)+q(2)) l2\*cos(q(1)+q(2))];

qdot\_d = Ja' \* K \* (x\_d-x);

q\_d = q\_d + qdot\_d \* s\_time;

% Control part to be designed %

u = Kp\*(q\_d-q) - Kd\*(qdot-qdot\_d) + G;

% Save the results of joint angles, angular velocities, angular acceleration

q1\_save(n) = q(1)\*R2D; % Save the joint angle in degree

q2\_save(n) = q(2)\*R2D;

qdot1\_save(n) = qdot(1)\*R2D; % Save the angular velocity in degree

qdot2\_save(n) = qdot(2)\*R2D;

qddot1\_save(n) = qddot(1)\*R2D; % Save the angular acceleration in degree

qddot2\_save(n) = qddot(2)\*R2D;

e1\_save(n) = x\_d(1)-x2; % Save the trajectory error in degree

e2\_save(n) = x\_d(2)-y2;

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Increase the Iteration Number

n=n+1;

% Draw Animation

if rem(n,10) == 0

set(p1,'X', Ax1, 'Y',Ay1)

set(p2,'X', Ax2, 'Y',Ay2)

drawnow

end

end

plot(x\_d(1), x\_d(2),'\*','LineWidth',2,'Color','r');

T=t0:s\_time:tf;

% Plot the graph of joint angles

figure(2)

subplot(3,1,1)

plot(T, q1\_save, T, q2\_save, 'r')

title('< Joint angle >')

legend('q\_1 (deg)', 'q\_2 (deg)')

xlabel('Time (sec)')

ylabel('q (deg)')

subplot(3,1,2)

plot(T, qdot1\_save, T, qdot2\_save, 'r')

title('< Anglular velotity >')

legend('qdot\_1 (deg/sec)', 'qdot\_2 (deg/sec)')

xlabel('Time (sec)')

ylabel('qdot (deg/sec)')

subplot(3,1,3)

plot(T, qddot1\_save, T, qddot2\_save, 'r')

title('< Angular acceleration >')

legend('qddot\_1 (deg/sec^2)', 'qddot\_2 (deg/sec^2)')

xlabel('Time (sec)')

ylabel('qddot (deg/sec^2)')

% Plot the graph of Tracking error

figure(4)

plot(T, e1\_save)

xlabel('Time (sec)')

ylabel('e\_1(m)')

figure(5)

plot(T, e2\_save)

xlabel('Time (sec)')

ylabel('e\_2(m)')

**task\_space\_direct\_regulation.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.002;

% Terminal time %

t0=0; tf=4;

% Variable for converting RADIAN -> DEGREE or DEGREE -> RADIAN

R2D = 180/pi;

D2R = pi/180;

% Initial Configurations defined in RADIAN %

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

qdot=[0; 0];

qddot=[0; 0];

% start position

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)];

% set position

x\_d = [-0.15; 0];

% PD gain

kp = 10; kd = 5; % two\_link

% kp = 12; kd = 5; % two\_link\_df

Kp = [kp, 0; 0, kp];

Kd = [kd, 0; 0, kd];

% Initial Input %

u = [0;0];

q\_d = [0; 0];

% Iteration number %

n=1;

% Plot Setting %

figure(1)

title('Animation')

hold on

axis([-0.4 0.4 -0.4 0.4]);

grid

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

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

[t,y] = ode45('two\_link',[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);

% Solve the IK using Jacobian transpose

Ja = [-l1\*sin(q(1))-l2\*sin(q(1)+q(2)) -l2\*sin(q(1)+q(2));

l1\*cos(q(1))+l2\*cos(q(1)+q(2)) l2\*cos(q(1)+q(2))];

% Control part to be designed %

u = Ja'\*Kp\*(x\_d - x) - Ja'\*Kd\*Ja\*qdot + [(m1+m2)\*g\*l1\*cos(q(1)) + m2\*g\*l2\*cos(q(1)+q(2)); m2\*g\*l2\*cos(q(1)+q(2))];

% Forward Kinematics %

x1 = l1\*cos(q(1));

y1 = l1\*sin(q(1));

x2 = x1 + l2\*cos(q(1)+q(2));

y2 = y1 + l2\*sin(q(1)+q(2));

x = [x2;y2];

% Save the results of joint angles, angular velocities, angular acceleration

q1\_save(n) = q(1)\*R2D; % Save the joint angle in degree

q2\_save(n) = q(2)\*R2D;

qdot1\_save(n) = qdot(1)\*R2D; % Save the angular velocity in degree

qdot2\_save(n) = qdot(2)\*R2D;

qddot1\_save(n) = qddot(1)\*R2D; % Save the angular acceleration in degree

qddot2\_save(n) = qddot(2)\*R2D;

e1\_save(n) = x\_d(1)-x2; % Save the trajectory error in degree

e2\_save(n) = x\_d(2)-y2;

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Increase the Iteration Number

n=n+1;

% Draw Animation

if rem(n,10) == 0

set(p1,'X', Ax1, 'Y',Ay1)

set(p2,'X', Ax2, 'Y',Ay2)

drawnow

end

end

plot(x\_d(1), x\_d(2),'\*','LineWidth',2,'Color','r');

T=t0:s\_time:tf;

% Plot the graph of joint angles

figure(2)

subplot(3,1,1)

plot(T, q1\_save, T, q2\_save, 'r')

title('< Joint angle >')

legend('q\_1 (deg)', 'q\_2 (deg)')

xlabel('Time (sec)')

ylabel('q (deg)')

subplot(3,1,2)

plot(T, qdot1\_save, T, qdot2\_save, 'r')

title('< Anglular velotity >')

legend('qdot\_1 (deg/sec)', 'qdot\_2 (deg/sec)')

xlabel('Time (sec)')

ylabel('qdot (deg/sec)')

subplot(3,1,3)

plot(T, qddot1\_save, T, qddot2\_save, 'r')

title('< Angular acceleration >')

legend('qddot\_1 (deg/sec^2)', 'qddot\_2 (deg/sec^2)')

xlabel('Time (sec)')

ylabel('qddot (deg/sec^2)')

% Plot the graph of Tracking error

figure(3)

plot(T, e1\_save)

xlabel('Time (sec)')

ylabel('e\_1(m)')

figure(4)

plot(T, e2\_save)

xlabel('Time (sec)')

ylabel('e\_2(m)')

**task\_space\_quintic.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.0001;

% Terminal time %

t0=0; tf=4;

% Initial Configurations defined in RADIAN %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)];

% Robot trajectories %

x\_0 = x; x\_f = [-0.15;0];

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

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

% Iteration numbers

n=1; %Iterator for main loop

n\_trj=1; %Iterator for trajectories of joint points

% Plot Setting %

figure(1)

title('Animation')

grid

hold on

axis([-0.4 0.4 -0.4 0.4]);

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : 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;

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

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

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

% Save the results of end-effector position, velocity, acceleration

x1\_save(n) = x\_d(1); % Save the end-effector position

x2\_save(n) = x\_d(2);

xdot1\_save(n) = xdot\_d(1); % Save the end-effector velocity

xdot2\_save(n) = xdot\_d(2);

xddot1\_save(n) = xddot\_d(1); % Save the end-effector acceleration

xddot2\_save(n) = xddot\_d(2);

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Update the animation

if rem(n,100) == 0

set(p1,'X', Ax1, 'Y',Ay1);

set(p2,'X', Ax2, 'Y',Ay2);

drawnow

end

% Save 1st and 2nd joint's location, (x1, y1) and (x2, y2)

if rem(n,10) == 0

x\_save(n\_trj) = x\_d(1);

y\_save(n\_trj) = x\_d(2);

n\_trj = n\_trj + 1;

end

% Increase the iteration number

n=n+1;

end

% Draw trajectory

plot(x\_save, y\_save, 'k.', 'MarkerSize', 2)

T=t0:s\_time:tf;

% Plot the graph of end-effector

figure(2)

subplot(3,1,1)

plot(T, x1\_save, T, x2\_save, 'r')

title('< End-effector Position >')

legend('x (m)', 'y (m)')

xlabel('t (sec)')

ylabel('x (m)')

subplot(3,1,2)

plot(T, xdot1\_save, T, xdot2\_save, 'r')

title('< End-effector Velocity >')

legend('v\_x (m/sec)', 'v\_y (m/sec)')

xlabel('t (sec)')

ylabel('v (m/sec)')

subplot(3,1,3)

plot(T, xddot1\_save, T, xddot2\_save, 'r')

title('< End-effector Acceleration >')

legend('a\_x (m/sec^2)', 'a\_y (m/sec^2)')

xlabel('t (sec)')

ylabel('a (m/sec^2)')

**task\_space\_indrect\_CLIK\_pinv.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.005;

% Variable for converting RADIAN -> DEGREE or DEGREE -> RADIAN

R2D = 180/pi;

D2R = pi/180;

% Terminal time %

t0=0; tf=4;

% Initial Configurations %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.15;0];

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));

% PD gain

kp = 0.5; kd = 1; k = 100; % two\_link

%kp = 0.5; kd = 1; k = 100; % two\_link\_df

K = [k, 0; 0, k];

Kp = [kp, 0; 0, kp];

Kd = [kd, 0; 0, kd];

% Initial Input %

u = [0;0];

q\_d = [0;0];

qdot\_d = [0;0];

% Iteration numbers

n=1; %Iterator for main loop

n\_trj=1; %Iterator for trajectories of joint points

% Plot Setting %

figure(1)

title('Animation')

grid

hold on

axis([-0.4 0.4 -0.4 0.4]);

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

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

[t,y] = ode45('two\_link\_df',[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);

q = [q(1);q(2)];

qdot = [qdot(1);qdot(2)];

qddot = [qddot(1);qddot(2)];

% Jacobian

Ja = [-l1\*sin(q(1))-l2\*sin(q(1)+q(2)) -l2\*sin(q(1)+q(2));

l1\*cos(q(1))+l2\*cos(q(1)+q(2)) l2\*cos(q(1)+q(2))];

Jadot = [-l1\*cos(q(1))\*qdot(1)-l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2));

-l1\*sin(q(1))\*qdot(1)-l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2))];

%Jadot = (Ja - Ja\_old)/s\_time;

% Forward Kinematics

x1 = l1\*cos(q(1));

y1 = l1\*sin(q(1));

x2 = x1 + l2\*cos(q(1)+q(2));

y2 = y1 + l2\*sin(q(1)+q(2));

x = [x2;y2]; % end-effector position

xdot = Ja\*qdot; % end-effector velocity

xddot = Ja\*qddot + Jadot\*qdot; % end-effector acceleration

% 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)];

% Solve the IK using the CLIK

qdot\_d = pinv(Ja)\*(xdot\_d + K\*(x\_d-x));

q\_d = q\_d + qdot\_d \* s\_time;

% Control part to be designed : PDG %

u = Kp\*(q\_d-q) - Kd\*(qdot-qdot\_d) + G;

% Save the results of end-effector position, velocity, acceleration

x1\_save(n) = x(1); % Save the end-effector position

x2\_save(n) = x(2);

xdot1\_save(n) = xdot(1); % Save the end-effector velocity

xdot2\_save(n) = xdot(2);

xddot1\_save(n) = xddot(1); % Save the end-effector acceleration

xddot2\_save(n) = xddot(2);

e1\_save(n) = x\_d(1)-x2; % Save the trajectory error

e2\_save(n) = x\_d(2)-y2;

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Update the animation

if rem(n,10) == 0

set(p1,'X', Ax1, 'Y',Ay1);

set(p2,'X', Ax2, 'Y',Ay2);

drawnow

end

% Save 1st and 2nd joint's location, (x1, y1) and (x2, y2)

x2\_save(n\_trj) = x2;

y2\_save(n\_trj) = y2;

x2d\_save(n\_trj) = x\_d(1);

y2d\_save(n\_trj) = x\_d(2);

% Increase the iteration number

n=n+1;

n\_trj = n\_trj + 1;

end

% Draw trajectory of (x1, y1) and (x2, y2)

plot(x2\_save, y2\_save, 'k.', 'MarkerSize', 2)

plot(x2d\_save, y2d\_save, 'k.', 'MarkerSize', 2, 'Color', 'r')

T=t0:s\_time:tf;

% Plot the graph of end-effector

figure(2)

subplot(3,1,1)

plot(T, x1\_save, T, x2\_save, 'r')

title('< End-effector Position >')

legend('x (m)', 'y (m)')

xlabel('t (sec)')

ylabel('x (m)')

subplot(3,1,2)

plot(T, xdot1\_save, T, xdot2\_save, 'r')

title('< End-effector Velocity >')

legend('v\_x (m/sec)', 'v\_y (m/sec)')

xlabel('t (sec)')

ylabel('v (m/sec)')

subplot(3,1,3)

plot(T, xddot1\_save, T, xddot2\_save, 'r')

title('< End-effector Acceleration >')

legend('a\_x (m/sec^2)', 'a\_y (m/sec^2)')

xlabel('t (sec)')

ylabel('a (m/sec^2)')

% Plot the graph of Tracking error

figure(4)

plot(T, e1\_save)

xlabel('Time (sec)')

ylabel('e\_1(m)')

figure(5)

plot(T, e2\_save)

xlabel('Time (sec)')

ylabel('e\_2(m)')

**task\_space\_indrect\_RAC.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.005;

% Variable for converting RADIAN -> DEGREE or DEGREE -> RADIAN

R2D = 180/pi;

D2R = pi/180;

% Terminal time %

t0=0; tf=4;

% Initial Configurations defined in RADIAN %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.15;0];

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));

% PD gain

kp = 17; kd = 50; % two\_link

%kp = 17; kd = 50; % two\_link\_df

Kp = [kp,0;0,kd];

Kd = [kd,0 ;0,kd];

% Initial Input %

u = [0;0];

q\_d = [0;0];

qdot\_d = [0;0];

% Iteration numbers

n=1; %Iterator for main loop

n\_trj=1; %Iterator for trajectories of joint points

% Plot Setting %

figure(1)

title('Animation')

grid

hold on

axis([-0.4 0.4 -0.4 0.4]);

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

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

%[t,y] = ode45('two\_link\_df',[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);

q = [q(1);q(2)];

qdot = [qdot(1);qdot(2)];

qddot = [qddot(1);qddot(2)];

% Jacobian

Ja = [-l1\*sin(q(1))-l2\*sin(q(1)+q(2)) -l2\*sin(q(1)+q(2));

l1\*cos(q(1))+l2\*cos(q(1)+q(2)) l2\*cos(q(1)+q(2))];

Jadot = [-l1\*cos(q(1))\*qdot(1)-l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2));

-l1\*sin(q(1))\*qdot(1)-l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2))];

%Jadot = (Ja - Ja\_old)/s\_time;

% Forward Kinematics %

x1 = l1\*cos(q(1));

y1 = l1\*sin(q(1));

x2 = x1 + l2\*cos(q(1)+q(2));

y2 = y1 + l2\*sin(q(1)+q(2));

x = [x2;y2]; % end-effector position

xdot = Ja\*qdot; % end-effector velocity

xddot = Ja\*qddot + Jadot\*qdot; % end-effector acceleration

% 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)];

% Solve the IK using the CLIK

qddot\_d =pinv(Ja)\*(xddot\_d -Jadot\*qdot + Kd\*(xdot\_d-xdot) + Kp\*(x\_d-x));

qdot\_d = qdot\_d + qddot\_d\*s\_time;

q\_d = q\_d + qdot\_d\*s\_time;

% Control part to be designed : PDG %

u0 = qddot\_d + Kd\*(qdot\_d - qdot) + Kp\*(q\_d - q);

u = H\*u0 + C\*qdot + G;

% Save the results of end-effector position, velocity, acceleration

x1\_save(n) = x(1); % Save the end-effector position

x2\_save(n) = x(2);

xdot1\_save(n) = xdot(1); % Save the end-effector velocity

xdot2\_save(n) = xdot(2);

xddot1\_save(n) = xddot(1); % Save the end-effector acceleration

xddot2\_save(n) = xddot(2);

e1\_save(n) = x\_d(1)-x2; % Save the trajectory error

e2\_save(n) = x\_d(2)-y2;

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Update the animation

if rem(n,10) == 0

set(p1,'X', Ax1, 'Y',Ay1);

set(p2,'X', Ax2, 'Y',Ay2);

drawnow

end

% Save 1st and 2nd joint's location, (x1, y1) and (x2, y2)

x2\_save(n\_trj) = x2;

y2\_save(n\_trj) = y2;

x2d\_save(n\_trj) = x\_d(1);

y2d\_save(n\_trj) = x\_d(2);

% Increase the iteration number

n=n+1;

n\_trj = n\_trj + 1;

end

% Draw trajectory of (x1, y1) and (x2, y2)

plot(x2\_save, y2\_save, 'k.', 'MarkerSize', 2)

plot(x2d\_save, y2d\_save, 'k.', 'MarkerSize', 2, 'Color', 'r')

T=t0:s\_time:tf;

% Plot the graph of end-effector

figure(2)

subplot(3,1,1)

plot(T, x1\_save, T, x2\_save, 'r')

title('< End-effector Position >')

legend('x (m)', 'y (m)')

xlabel('t (sec)')

ylabel('x (m)')

subplot(3,1,2)

plot(T, xdot1\_save, T, xdot2\_save, 'r')

title('< End-effector Velocity >')

legend('v\_x (m/sec)', 'v\_y (m/sec)')

xlabel('t (sec)')

ylabel('v (m/sec)')

subplot(3,1,3)

plot(T, xddot1\_save, T, xddot2\_save, 'r')

title('< End-effector Acceleration >')

legend('a\_x (m/sec^2)', 'a\_y (m/sec^2)')

xlabel('t (sec)')

ylabel('a (m/sec^2)')

% Plot the graph of Tracking error

figure(4)

plot(T, e1\_save)

xlabel('Time (sec)')

ylabel('e\_1(m)')

figure(5)

plot(T, e2\_save)

xlabel('Time (sec)')

ylabel('e\_2(m)')

**task\_space\_direct\_IDC.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.005;

% Terminal time %

t0=0; tf=4;

% Variable for converting RADIAN -> DEGREE or DEGREE -> RADIAN

R2D = 180/pi;

D2R = pi/180;

% PD gain

kp = 10; kd = 2; % two\_link

%kp = 10; kd = 2; % two\_link\_df

Kp = [kp,0 ;0,kp];

Kd = [kd,0 ;0,kd];

% Initial Configurations defined in RADIAN %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.15;0];

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));

% Iteration numbers

n=1; %Iterator for main loop

n\_trj=1; %Iterator for trajectories of joint points

% Initial Input %

u = [0;0];

q\_d = [0;0];

qdot\_d = [0;0];

% Plot Setting %

figure(1)

title('Animation')

grid

hold on

axis([-0.4 0.4 -0.4 0.4]);

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

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

%[t,y] = ode45('two\_link\_df',[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);

q = [q(1);q(2)];

qdot = [qdot(1);qdot(2)];

qddot = [qddot(1);qddot(2)];

% Jacobian

Ja = [-l1\*sin(q(1))-l2\*sin(q(1)+q(2)) -l2\*sin(q(1)+q(2));

l1\*cos(q(1))+l2\*cos(q(1)+q(2)) l2\*cos(q(1)+q(2))];

Jadot = [-l1\*cos(q(1))\*qdot(1)-l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2));

-l1\*sin(q(1))\*qdot(1)-l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2))];

%Jadot = (Ja - Ja\_old)/s\_time;

% Forward Kinematics

x1 = l1\*cos(q(1));

y1 = l1\*sin(q(1));

x2 = x1 + l2\*cos(q(1)+q(2));

y2 = y1 + l2\*sin(q(1)+q(2));

x = [x2;y2]; % end-effector position

xdot = Ja\*qdot; % end-effector velocity

xddot = Ja\*qddot + Jadot\*qdot; % end-effector acceleration

% 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)];

% Design Task Space Controller

u0 = pinv(Ja)\*(xddot\_d - Jadot\*qdot + Kd\*(xdot\_d-xdot) + Kp\*(x\_d-x));

u = H\*u0 + C\*qdot + G;

% Save the results of end-effector position, velocity, acceleration

x1\_save(n) = x(1); % Save the end-effector position

x2\_save(n) = x(2);

xdot1\_save(n) = xdot(1); % Save the end-effector velocity

xdot2\_save(n) = xdot(2);

xddot1\_save(n) = xddot(1); % Save the end-effector acceleration

xddot2\_save(n) = xddot(2);

e1\_save(n) = x\_d(1)-x2; % Save the trajectory error

e2\_save(n) = x\_d(2)-y2;

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Update the animation

if rem(n,10) == 0

set(p1,'X', Ax1, 'Y',Ay1);

set(p2,'X', Ax2, 'Y',Ay2);

drawnow

end

% Save 1st and 2nd joint's location, (x1, y1) and (x2, y2)

x2\_save(n\_trj) = x2;

y2\_save(n\_trj) = y2;

x2d\_save(n\_trj) = x\_d(1);

y2d\_save(n\_trj) = x\_d(2);

% Increase the iteration number

n=n+1;

n\_trj = n\_trj + 1;

end

% Draw trajectory of (x1, y1) and (x2, y2)

plot(x2\_save, y2\_save, 'k.', 'MarkerSize', 2)

plot(x2d\_save, y2d\_save, 'k.', 'MarkerSize', 2, 'Color', 'r')

T=t0:s\_time:tf;

% Plot the graph of end-effector

figure(2)

subplot(3,1,1)

plot(T, x1\_save, T, x2\_save, 'r')

title('< End-effector Position >')

legend('x (m)', 'y (m)')

xlabel('t (sec)')

ylabel('x (m)')

subplot(3,1,2)

plot(T, xdot1\_save, T, xdot2\_save, 'r')

title('< End-effector Velocity >')

legend('v\_x (m/sec)', 'v\_y (m/sec)')

xlabel('t (sec)')

ylabel('v (m/sec)')

subplot(3,1,3)

plot(T, xddot1\_save, T, xddot2\_save, 'r')

title('< End-effector Acceleration >')

legend('a\_x (m/sec^2)', 'a\_y (m/sec^2)')

xlabel('t (sec)')

ylabel('a (m/sec^2)')

% Plot the graph of Tracking error

figure(4)

plot(T, e1\_save)

xlabel('Time (sec)')

ylabel('e\_1(m)')

figure(5)

plot(T, e2\_save)

xlabel('Time (sec)')

ylabel('e\_2(m)')

**task\_space\_drect\_LBC.m**

clear

clear all

close all

home

global H;

global C;

global G;

global u;

global temp1;

global temp2;

global m1;

global m2;

global l1;

global l2;

global g;

global tf;

global t0;

% Robot Parameters

m1 = 0.5;

m2 = 0.5;

l1 = 0.15;

l2 = 0.15;

g = 9.806;

% Control frequency is defined as follows %

s\_time = 0.005;

% Terminal time %

t0=0; tf=4;

% Variable for converting RADIAN -> DEGREE or DEGREE -> RADIAN

R2D = 180/pi;

D2R = pi/180;

% PD gain

kd = 5; lamb = 2; % two\_link

%kd = 5; lamb = 2; % two\_link\_df

lambda = [lamb,0;0,lamb];

Kd = [kd, 0;0,kd];

% Initial Configurations %

x = [0.15/sqrt(2); 0.15 + 0.15/sqrt(2)]; x\_d = [-0.15;0];

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));

% Iteration numbers

n=1; %Iterator for main loop

n\_trj=1; %Iterator for trajectories of joint points

% Initial Input %

u = [0;0];

q\_d = [0;0];

qdot\_d = [0;0];

% Plot Setting %

figure(1)

title('Animation')

grid

hold on

axis([-0.4 0.4 -0.4 0.4]);

Ax1 = [0, l1];

Ay1 = [0, 0];

Ax2 = [l1, l1+l2];

Ay2 = [0, 0];

p1 = line(Ax1,Ay1,'EraseMode','xor','LineWidth',[5],'Color','b');

p2 = line(Ax2,Ay2,'EraseMode','xor','LineWidth',[5],'Color','c');

% Robot Implementation

for i = t0 : s\_time : tf

% Forward Dynamics Part for Motion Generation %

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

%[t,y] = ode45('two\_link\_df',[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);

q = [q(1);q(2)];

qdot = [qdot(1);qdot(2)];

qddot = [qddot(1);qddot(2)];

% Jacobian

Ja = [-l1\*sin(q(1))-l2\*sin(q(1)+q(2)) -l2\*sin(q(1)+q(2));

l1\*cos(q(1))+l2\*cos(q(1)+q(2)) l2\*cos(q(1)+q(2))];

Jadot = [-l1\*cos(q(1))\*qdot(1)-l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*cos(q(1)+q(2))\*(qdot(1)+qdot(2));

-l1\*sin(q(1))\*qdot(1)-l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2)), -l2\*sin(q(1)+q(2))\*(qdot(1)+qdot(2))];

%Jadot = (Ja - Ja\_old)/s\_time;

% Forward Kinematics

x1 = l1\*cos(q(1));

y1 = l1\*sin(q(1));

x2 = x1 + l2\*cos(q(1)+q(2));

y2 = y1 + l2\*sin(q(1)+q(2));

x = [x2;y2]; % end-effector position

xdot = Ja\*qdot; % end-effector velocity

xddot = Ja\*qddot + Jadot\*qdot; % end-effector acceleration

% 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)];

% Design Task Space Controller

zdot = pinv(Ja)\*(xdot\_d + lambda\*(x\_d-x));

zddot = pinv(Ja)\*(xddot\_d + lambda\*(xdot\_d-xdot) - Jadot\*zdot);

u = H\*zddot + C\*zdot + G + Ja'\*Kd\*Ja\*(zdot-qdot);

% Save the results of end-effector position, velocity, acceleration

x1\_save(n) = x(1); % Save the end-effector position

x2\_save(n) = x(2);

xdot1\_save(n) = xdot(1); % Save the end-effector velocity

xdot2\_save(n) = xdot(2);

xddot1\_save(n) = xddot(1); % Save the end-effector acceleration

xddot2\_save(n) = xddot(2);

e1\_save(n) = x\_d(1)-x2; % Save the trajectory error

e2\_save(n) = x\_d(2)-y2;

% Calculate the coordinates of robot geometry for animation

Ax1 = [0, x1];

Ay1 = [0, y1];

Ax2 = [x1, x2];

Ay2 = [y1, y2];

% Update the animation

if rem(n,10) == 0

set(p1,'X', Ax1, 'Y',Ay1);

set(p2,'X', Ax2, 'Y',Ay2);

drawnow

end

% Save 1st and 2nd joint's location, (x1, y1) and (x2, y2)

x2\_save(n\_trj) = x2;

y2\_save(n\_trj) = y2;

x2d\_save(n\_trj) = x\_d(1);

y2d\_save(n\_trj) = x\_d(2);

% Increase the iteration number

n=n+1;

n\_trj = n\_trj + 1;

end

% Draw trajectory of (x1, y1) and (x2, y2)

plot(x2\_save, y2\_save, 'k.', 'MarkerSize', 2)

plot(x2d\_save, y2d\_save, 'k.', 'MarkerSize', 2, 'Color', 'r')

T=t0:s\_time:tf;

% Plot the graph of end-effector

figure(2)

subplot(3,1,1)

plot(T, x1\_save, T, x2\_save, 'r')

title('< End-effector Position >')

legend('x (m)', 'y (m)')

xlabel('t (sec)')

ylabel('x (m)')

subplot(3,1,2)

plot(T, xdot1\_save, T, xdot2\_save, 'r')

title('< End-effector Velocity >')

legend('v\_x (m/sec)', 'v\_y (m/sec)')

xlabel('t (sec)')

ylabel('v (m/sec)')

subplot(3,1,3)

plot(T, xddot1\_save, T, xddot2\_save, 'r')

title('< End-effector Acceleration >')

legend('a\_x (m/sec^2)', 'a\_y (m/sec^2)')

xlabel('t (sec)')

ylabel('a (m/sec^2)')

% Plot the graph of Tracking error

figure(4)

plot(T, e1\_save)

xlabel('Time (sec)')

ylabel('e\_1(m)')

figure(5)

plot(T, e2\_save)

xlabel('Time (sec)')

ylabel('e\_2(m)')