CSCI 545: Homework 2

Deepika Anand

March 30, 2017

Problem 1. (a)

$$p_{cm} = \frac{\sum_{i=0}^{n} m_i p_i^0}{\sum_{i=0}^{n} m_i} \tag{1}$$

where $p_{i=0}^0$ is defined as vector from origin to point i. In other words, representing Frame i in Frame 0.

$$p_1^0 = \begin{bmatrix} l_1 cos\theta_1 \\ l_1 sin\theta_1 \\ 0 \end{bmatrix}$$

$$p_2^0 = \begin{bmatrix} l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2) \\ l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2) \\ 0 \end{bmatrix} \quad p_3^0 = \begin{bmatrix} l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2 + l_3 cos(\theta_1 + \theta_2 + \theta_3) \\ l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2 + l_3 sin(\theta_1 + \theta_2 + \theta_3)) \\ 0 \end{bmatrix}$$

$$p_4^0 = \begin{bmatrix} l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2 + l_3 cos(\theta_1 + \theta_2 + \theta_3) + l_4 cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \\ l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2 + l_3 sin(\theta_1 + \theta_2 + \theta_3) + l_4 sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \\ 0 \end{bmatrix}$$

Since $m_i = 1$ for i = 1 to 4.

Hence,

$$p_{cm} = (1/4) * \begin{bmatrix} 4 * l_1 cos_1 + 3 * l_2 cos_{12} + 2 * l_3 cos_{123} + l_4 cos_{1234} \\ 4 * l_1 sin_1 + 3 * l_2 sin_{12} + 2 * l_3 sin_{123} + l_4 sin_{1234} \\ 0 \end{bmatrix}$$

Problem 1. (b)

Geometric Jacobian for 4 - revolute join is given as J =

$$\begin{bmatrix} J_1 & J_2 & J_3 & J_4 \end{bmatrix}$$

For each join i, J$_i$ = $\begin{bmatrix} z_{i-1} * (p_4^0 - p_{i-1}^0) \\ z_{i-1} \end{bmatrix}$
where z_{i-1} is the axis of rotation and p_i^0 is defined as vector from origin to point i. In other words, representing Frame i in Frame 0. But since we have been asked to consider orientation only. Hence the effective J_i will be

$$\begin{bmatrix} z_{i-1} * (p_4^0 - p_{i-1}^0) \end{bmatrix}$$

In this case, geometric jacbian J is given as
J =

$$\begin{bmatrix} z_0 * (p_4^0) & z_1 * (p_4^0 - p_1^0) & z_2 * (p_4^0 - p_2^0) & z_3 * (p_4^0 - p_3^0) \end{bmatrix}$$

where z$_0$ = $z_1 = z_2 = z_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Problem 1. (c)

Removing all occurrences of p_i^0 with respective values. $\mathrm{p}_1^0 = \begin{bmatrix} l_1c_1 \\ l_1s_1 \end{bmatrix}$

$\mathrm{p}_2^0 = \begin{bmatrix} l_1c_1 + l_2c_{12} \\ l_1s_1 + l_2s_{12} \end{bmatrix}$

$\mathrm{p}_3^0 = \begin{bmatrix} l_1c_1 + l_2c_{12} + l_3c_{123} \\ l_1s_1 + l_2s_{12} + l_3s_{123} \end{bmatrix}$

$\mathrm{p}_4^0 = \begin{bmatrix} l_1c_1 + l_2c_{12} + l_3c_{123} + l_4c_{1234} \\ l_1s_1 + l_2s_{12} + l_3s_{123} + l_4s_{1234} \end{bmatrix}$

Also, $\mathrm{z}_{i-1} * p_{i-1}^0 = \frac{\partial p_4^0}{\partial \theta_i}$

$J = \begin{bmatrix} -l_1s_1 - l_2s_{12} - l_3s_{123} - l_4s_{1234} & -l_{2s}12 - l_3s_{123} - l_4s_{1234} & -l_3s_{123} - l_4s_{1234} & -l_4s_{1234} \\ l_1c_1 + l_2c_{12} + l_3c_{123} + l_4c_{1234} & l_2c_{12} + l_3c_{123} + l_4c_{1234} & l_3c_{123} + l_4c_{1234} & l_4c_{1234} \end{bmatrix}$

$s_{1234} = sin(\theta_1 + \theta_2 + \theta_3 + \theta_4)$

$c_{1234} = cos(\theta_1 + \theta_2 + \theta_3 + \theta_4)$ and so on.

Problem 1. (d)

J^i = Jacobian for p_i

$$\mathrm{J}^1 = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} - l_3 s_{123} - l_4 s_{1234} \\ l_1 c_1 + l_2 c_{12} + l_3 c_{123} + l_4 c_{1234} \end{bmatrix}$$

$$\mathrm{J}^2 = \begin{bmatrix} -l_2 s_{12} - l_3 s_{123} - l_4 s_{1234} \\ l_2 c_{12} + l_3 c_{123} + l_4 c_{1234} \end{bmatrix}$$

$$\mathrm{J}^3 = \begin{bmatrix} -l_3 s_{123} - l_4 s_{1234} \\ l_3 c_{123} + l_4 c_{1234} \end{bmatrix}$$

$$\mathrm{J}^4 = \begin{bmatrix} -l_4 s_{1234} \\ l_4 c_{1234} \end{bmatrix}$$

Problem 1. (e)

Jacobian for center of mass

Use p_{cm} and differentiate with each θ_i

$$\mathrm{J}_{cm} = \frac{1}{4} * \begin{bmatrix} -4 * l_1 s_1 - 3 * l_2 s_{12} - 2 * l_3 s_{123} - l_4 s_{1234} & 4 * l_1 c_1 + 3 * l_2 c_{12} + 2 * l_3 c_{123} + l_4 c_{1234} \\ -3 * l_2 s_{12} - 2 * l_3 s_{123} - l_4 s_{1234} & 3 * l_2 c_{12} + 2 * l_3 c_{123} + l_4 c_{1234} \\ -2 * l_3 s_{123} - l_4 s_{1234} & 2 * l_3 c_{123} + l_4 c_{1234} \\ -l_4 s_{1234} & l_4 c_{1234} \end{bmatrix}^T$$

T stands for Transpose of this 4X2 matrix.
That is the effective matrix will be of size 2X4

Problem 1. (f)

Center of mass Jacobian

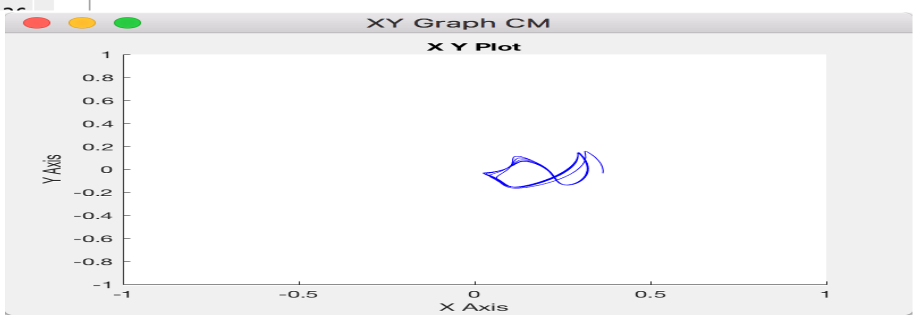
```
% Center of mass Jacobian
J14=-links(4)*sin(theta(1)+theta(2)+theta(3)+theta(4));
J13=-2*links(3)*sin(theta(1)+theta(2)+theta(3))+J14;
J12=-3*links(2)*sin(theta(1)+theta(2))+J13;
J11=-4*links(1)*sin(theta(1))+J12;


J24=links(4)*cos(theta(1)+theta(2)+theta(3)+theta(4));
J23=2*links(3)*cos(theta(1)+theta(2)+theta(3))+J24;
J22=3*links(2)*cos(theta(1)+theta(2))+J23;
J21=4*links(1)*cos(theta(1))+J22;

J = [J11 J12 J13 J14; J21 J22 J23 J24];
J = (1/4).*J
```

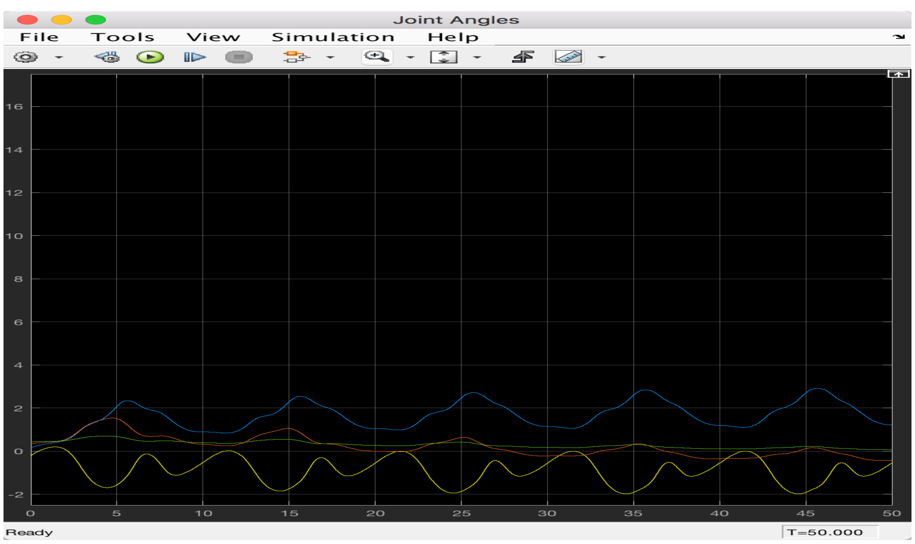
Problem 1. (g)

Jacobian transpose for inverse kinematics

inverse_kinematics.m forward_kinematics.m ArmAnimation.m

```
n = length(theta);
m = length(xd);


% Center of mass Jacobian
J14=-links(4)*sin(theta(1)+theta(2)+theta(3)+theta(4));
J13=-2*links(3)*sin(theta(1)+theta(2)+theta(3))+J14;
J12=-3*links(2)*sin(theta(1)+theta(2))+J13;
J11=-4*links(1)*sin(theta(1))+J12;


J24=links(4)*cos(theta(1)+theta(2)+theta(3)+theta(4));
J23=2*links(3)*cos(theta(1)+theta(2)+theta(3))+J24;
J22=3*links(2)*cos(theta(1)+theta(2))+J23;
J21=4*links(1)*cos(theta(1))+J22;

J = [J11 J12 J13 J14; J21 J22 J23 J24];
J = (1/4).*J

%Part g : Transpose
Jt=transpose(J);
thetad = Jt*xd;
```





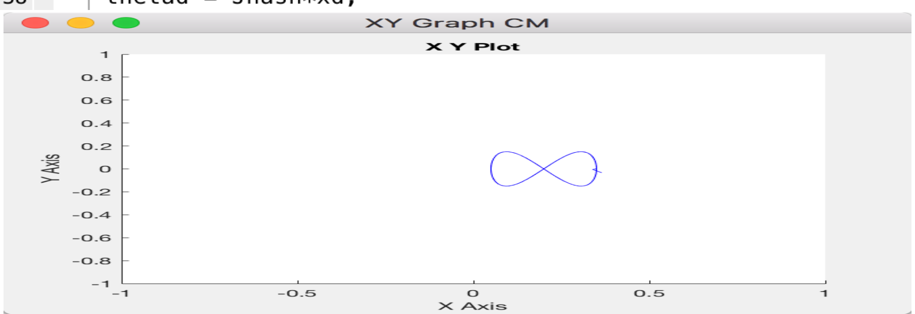
Inverse Transpose method requires tuning of α. In this case $\alpha = 1$ and hence the graph is a little distorted. However on increasing α the graphs becomes smoother.

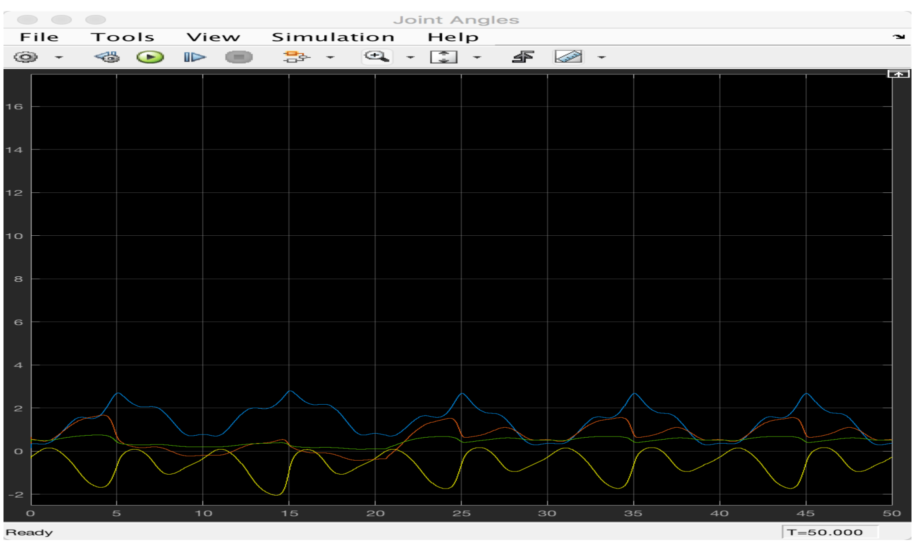
Problem 1. (h)

Pseudo-inverse for inverse kinematics

inverse_kinematics.m | forward_kinematics.m | ArmAnimation.m

```
% Center of mass Jacobian
J14=-links(4)*sin(theta(1)+theta(2)+theta(3)+theta(4));
J13=-2*links(3)*sin(theta(1)+theta(2)+theta(3))+J14;
J12=-3*links(2)*sin(theta(1)+theta(2))+J13;
J11=-4*links(1)*sin(theta(1))+J12;

J24=links(4)*cos(theta(1)+theta(2)+theta(3)+theta(4));
J23=2*links(3)*cos(theta(1)+theta(2)+theta(3))+J24;
J22=3*links(2)*cos(theta(1)+theta(2))+J23;
J21=4*links(1)*cos(theta(1))+J22;

J = [J11 J12 J13 J14; J21 J22 J23 J24];
J = (1/4).*J

%Part g : Transpose
Jt=transpose(J);
%thetad = Jt*xd;

%Part h : Pseudo-Inverse
Jhash = Jt/(J*Jt);
thetad = Jhash*xd;
```





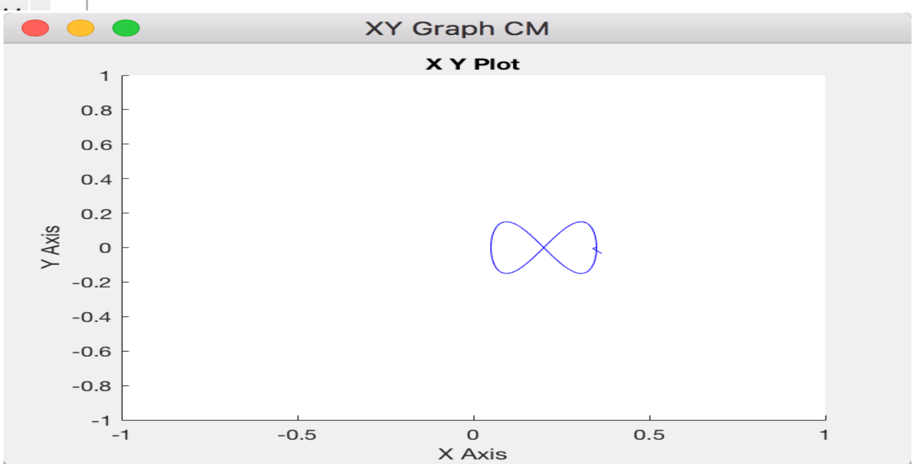
The pseudo-inverse is a least squares best fit approximate solution. and in this case we can see the graph is smooth. Since this graphs doesn't have multiple local minimums therefore since run was enough otherwise multiple random start points are required to avoid stucking in local minima

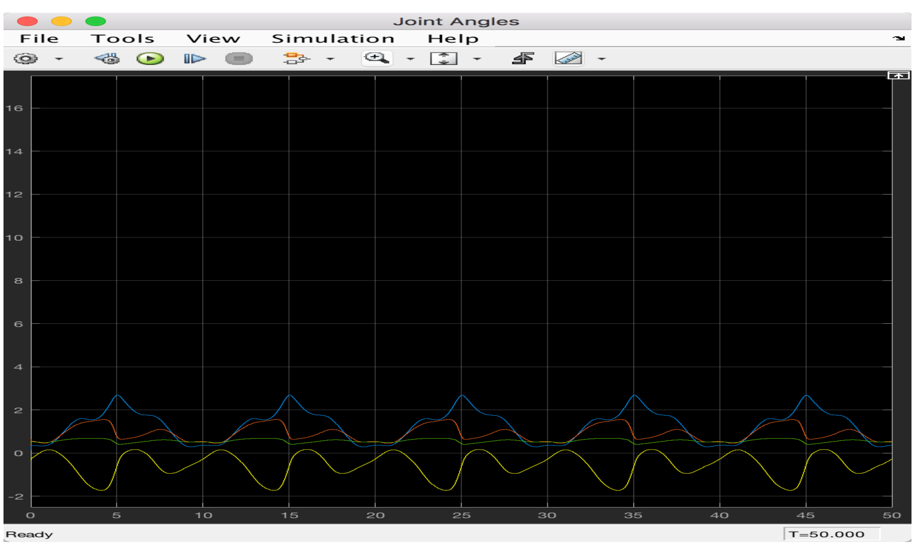
Problem 1. (i)

Pseudo-inverse with Null-space optimization

inverse_kinematics.m | forward_kinematics.m | ArmAnimation.m

```

%Part g : Transpose
Jt=transpose(J);
%thetad = Jt*xd;

%Part h : Pseudo-Inverse
Jhash = Jt/(J*Jt);
thetad = Jhash*xd;

%Part i : pseudo-inverse with Null-space optimization
I = eye(n)
theta0 = [0.5 ;0.5; 0.5; 0.5];
thetad = Jhash*xd + (I-Jhash*J)*(theta0-theta);
```





Null-space optimization allows us to optimize on null-space however the projection on that space never interferes with our optimization equation.This explicit optimization makes the plot smooth.

Problem 1. (j)

Derivation
In this case different DOFs are weighted. So we need to minimize

$$\frac{1}{2}\Delta\theta^T W \Delta\theta \tag{2}$$

subjected to $= J(\theta)\Delta\theta$
Effective λ is

$$F = \frac{1}{2}\Delta\theta^T W \Delta\theta + \lambda^T * (\Delta x - J(\theta)\Delta\theta) \tag{3}$$

$$\frac{\partial F}{\partial \lambda} = 0 \tag{4}$$

$$\Delta x = J(\theta)\Delta\theta \tag{5}$$

Equating

$$\frac{\partial F}{\partial \Delta\theta} = 0 \tag{6}$$

gives,

$$W\Delta\theta = J^T\lambda \tag{7}$$

From 5 and 7

$$\lambda = W(JJ^T)^{-1}J\Delta\theta \tag{8}$$

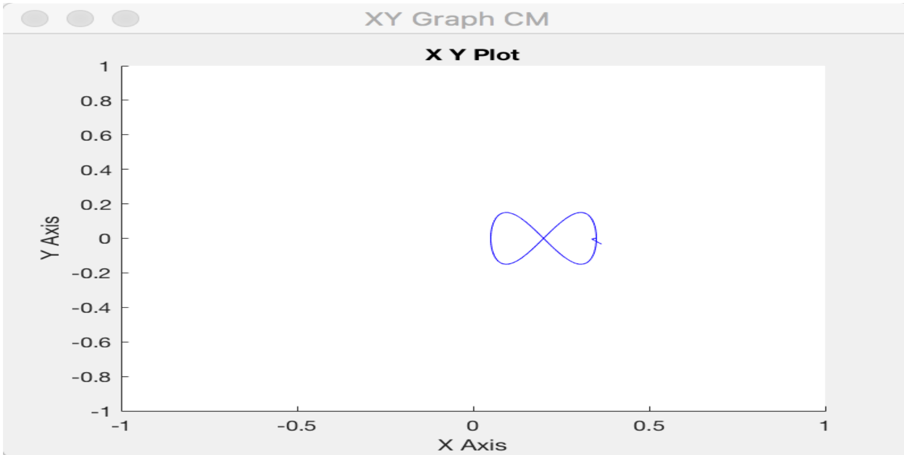
Using this value of λ in 7

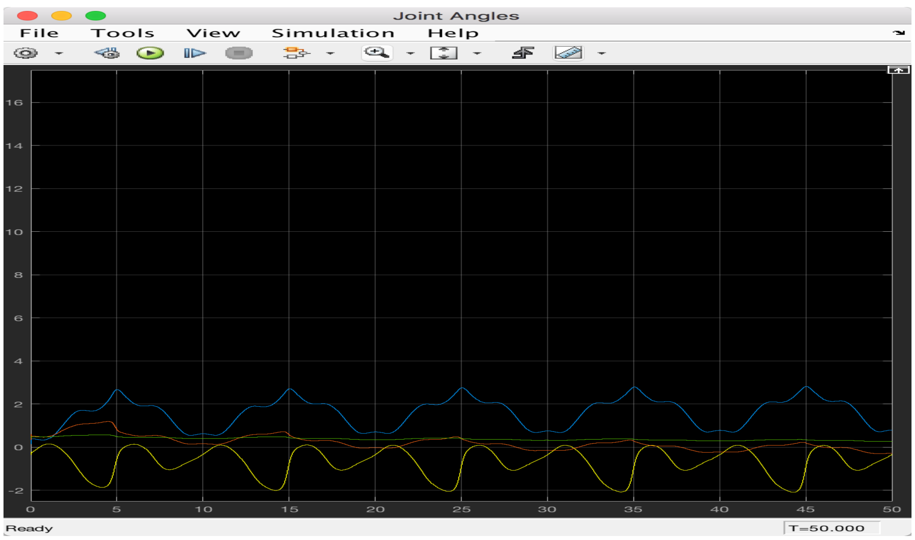
$$\Delta\theta = W^{-1}J^T(W^{-1}JJ^T)^{-1} \tag{9}$$

Now using this equation in code : Weighted pseudo-inverse

inverse_kinematics.m | forward_kinematics.m | ArmAnimation.m

```
%Part g : Transpose
Jt=transpose(J);
%thetad = Jt*xd;

%Part h : Pseudo-Inverse
Jhash = Jt/(J*Jt);
%thetad = Jhash*xd;

%Part i : pseudo-inverse with Null-space optimization
I = eye(n)
theta0 = [0.5 ;0.5; 0.5; 0.5];
%hetad = Jhash*xd + (I-Jhash*J)*(theta0-theta);

%Part j : Weighted pseudo-inverse
w=[0.1,0.2, 0.3, 0.5];
W=eye(n).*[w;w;w;w];
Wi=inv(W);
Jhashw=Wi*J'/(J*Wi*J');
```

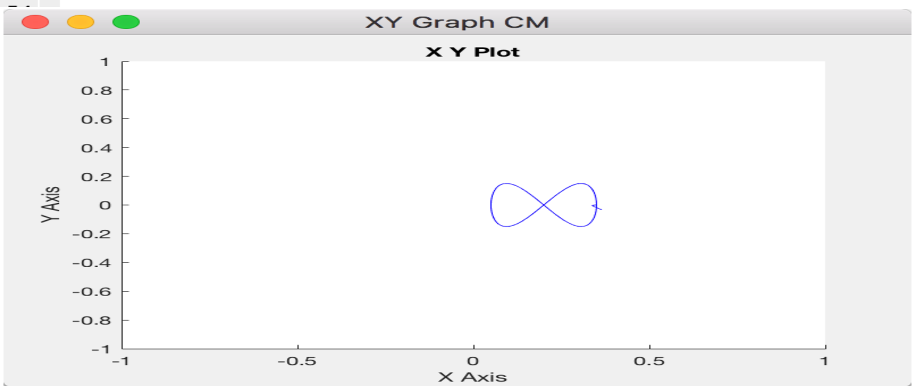


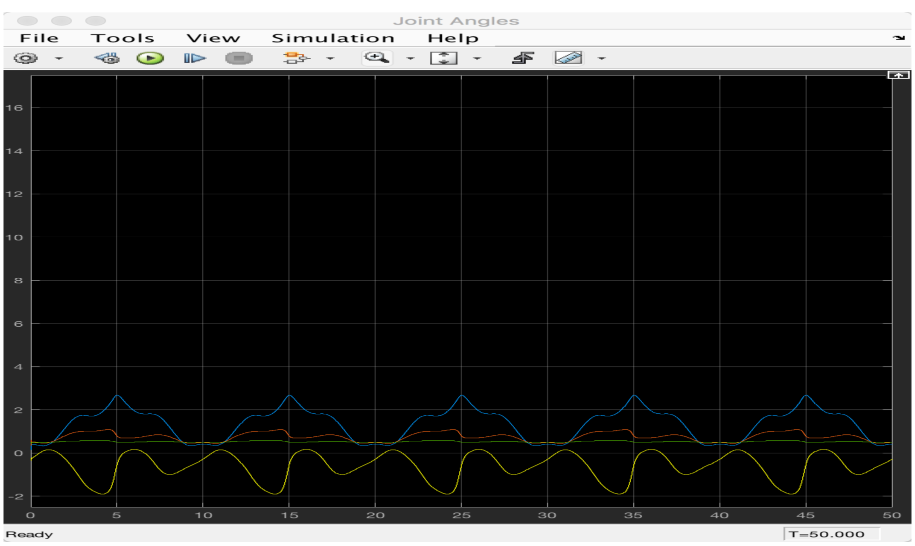


Since weighted pseudo inverse we have used weighted DOF as a result the graph is smooth.

Problem 1. (k)

```
%Part g : Transpose
Jt=transpose(J);
%thetad = Jt*xd;

%Part h : Pseudo-Inverse
Jhash = Jt/(J*Jt);
%thetad = Jhash*xd;

%Part i : pseudo-inverse with Null-space optimization
I = eye(n)
theta0 = [0.5 ;0.5; 0.5; 0.5];
%hetad = Jhash*xd + (I-Jhash*J)*(theta0-theta);

%Part j : Weighted pseudo-inverse
w=[0.1,0.2, 0.3, 0.5];
W=eye(n).*[w;w;w;w];
Wi=inv(W);
Jhashw=Wi*J'/(J*Wi*J');
%thetad = Jhashw * xd

%Part k: Weights in null-space optimization
thetad = Jhashw*xd + (I-Jhashw*J)*(theta0-theta);
```

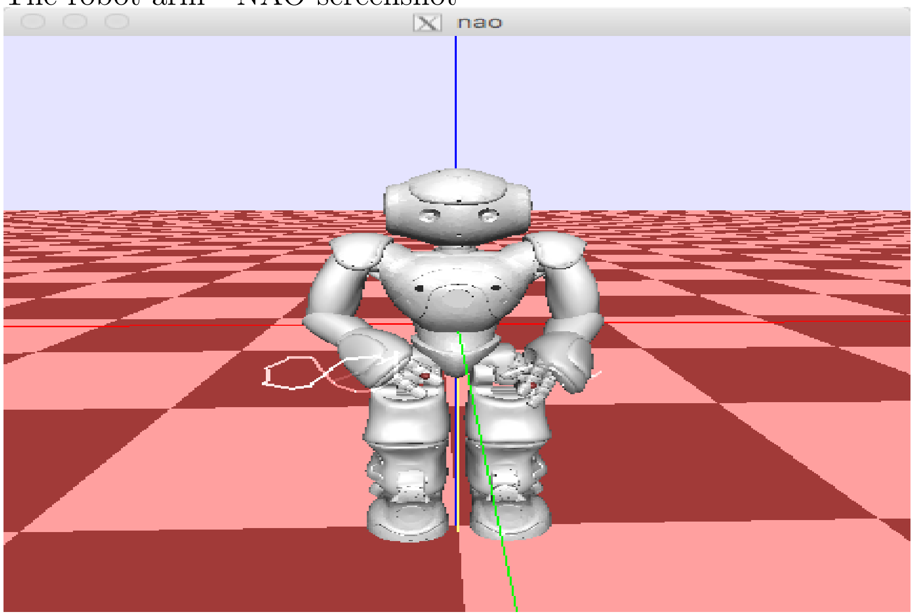




In this case we use the advantage of null space optimization as well as weighted DOF hence the arm movement is smooth.

Problem 2.

The robot arm - NAO screenshot



Code

```
static int myTarget = 0;
static int
run_draw_task(void)
{
  int j, i;
  double sum=0;
  double aux;

  if (tau <= -0.5*0 ) {
  if (myTarget == 0) {
  ctarget[RIGHT_HAND].x[_X_] += 0.05/2;
  ctarget[RIGHT_HAND].x[_Z_] -= 0.017;
  tau = 1;
  myTarget = 1;
  run_draw_task();
  } else if (myTarget == 1) {
  ctarget[RIGHT_HAND].x[_X_] += 0.05/2;
  tau = 1;
  myTarget = 3;
  run_draw_task();
  } else if (myTarget == 3) {
  ctarget[RIGHT_HAND].x[_X_] += 0.03;
  ctarget[RIGHT_HAND].x[_Z_] += 0.017;
  tau = 1;
```

```
myTarget = 4;
run_draw_task();
} else if (myTarget == 4) {
ctarget[RIGHT_HAND].x[_X_] += 0.01;
ctarget[RIGHT_HAND].x[_Z_] += 0.017;
tau = 1;
myTarget = 5;
run_draw_task();
} else if (myTarget == 5) {
ctarget[RIGHT_HAND].x[_X_] += 0.009;
ctarget[RIGHT_HAND].x[_Z_] += 0.017;
tau = 1;
myTarget = 6;
run_draw_task();
} else if (myTarget == 6) {
ctarget[RIGHT_HAND].x[_X_] += 0.01/2;
ctarget[RIGHT_HAND].x[_Z_] += 0.017;
tau = 1;
myTarget = 7;
run_draw_task();
} else if (myTarget == 7) {
ctarget[RIGHT_HAND].x[_X_] += 0.025;
ctarget[RIGHT_HAND].x[_Z_] += 0.01;
tau = 1;
myTarget = 8;
run_draw_task();
} else if (myTarget == 8) {
ctarget[RIGHT_HAND].x[_X_] += 0.025;
ctarget[RIGHT_HAND].x[_Z_] -= 0.007;
tau = 1;
myTarget = 9;
run_draw_task();
} else if (myTarget == 9) {
ctarget[RIGHT_HAND].x[_X_] += 0.005;
ctarget[RIGHT_HAND].x[_Z_] -= 0.03;
tau = 1;
myTarget = 10;
run_draw_task();
} else if (myTarget == 10) {
ctarget[RIGHT_HAND].x[_X_] -= 0.03;
ctarget[RIGHT_HAND].x[_Z_] -= 0.02;
tau = 1;
myTarget = 11;
run_draw_task();
} else if (myTarget == 11) {
```

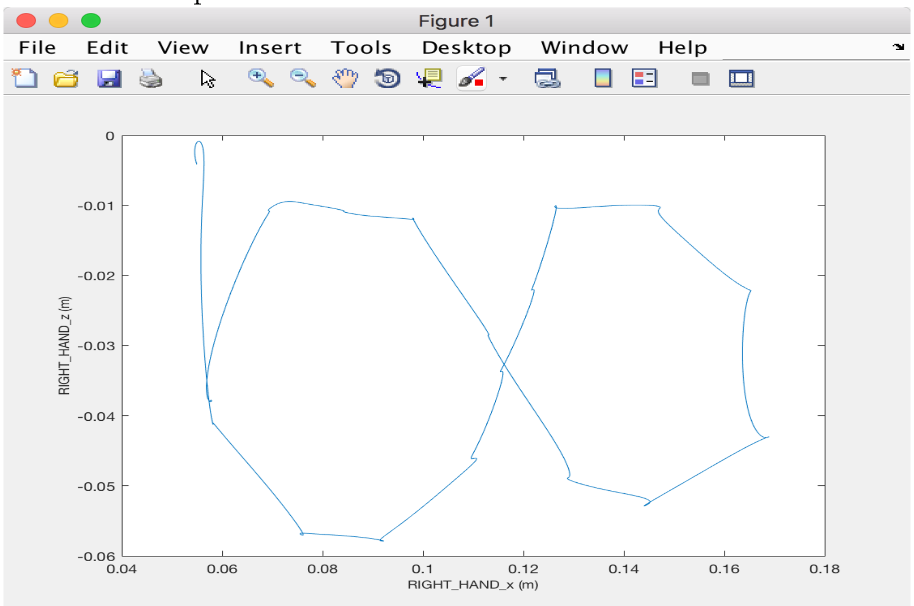
```
ctarget[RIGHT_HAND].x[_X_] -= 0.02;
tau = 1;
myTarget = 12;
run_draw_task();
} else if (myTarget == 12) {
ctarget[RIGHT_HAND].x[_X_] -= 0.02;
ctarget[RIGHT_HAND].x[_Z_] += 0.02;
tau = 1;
myTarget = 13;
run_draw_task();
} else if (myTarget == 13) {
ctarget[RIGHT_HAND].x[_X_] -= 0.02;
ctarget[RIGHT_HAND].x[_Z_] += 0.015;
tau = 1;
myTarget = 14;
run_draw_task();
}  else if (myTarget == 14) {
ctarget[RIGHT_HAND].x[_X_] -= 0.02;
tau = 1;
myTarget = 15;
run_draw_task();
} else if (myTarget == 15) {
ctarget[RIGHT_HAND].x[_X_] -= 0.018;
ctarget[RIGHT_HAND].x[_Z_] -= 0.005;
tau = 1;
myTarget = 16;
run_draw_task();
} else if (myTarget == 16) {
ctarget[RIGHT_HAND].x[_X_] -= 0.02;
ctarget[RIGHT_HAND].x[_Z_] -= 0.03;
tau = 1;
myTarget = 17;
run_draw_task();
}
else {
freeze();
}
  return TRUE;
}
```

Screenshot - Code

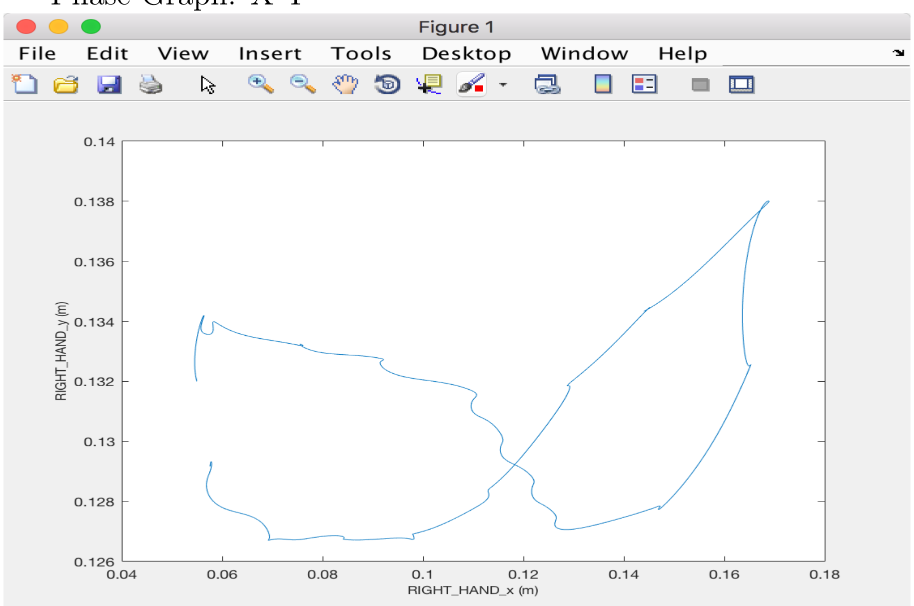
draw_task.cpp | CMakeLists.txt | initUserTasks.c

```
/* has the movement time expired? I intentially r
if (tau <= -0.5*0 ) {
    if (myTarget == 0) {
        ctarget[RIGHT_HAND].x[_X_] += 0.05/2;
        ctarget[RIGHT_HAND].x[_Z_] -= 0.017;
        tau = 1;
        myTarget = 1;
        run_draw_task();
    } else if (myTarget == 1) {
        ctarget[RIGHT_HAND].x[_X_] += 0.05/2;
        tau = 1;
        myTarget = 3;
        run_draw_task();
    } else if (myTarget == 3) {
        ctarget[RIGHT_HAND].x[_X_] += 0.03;
        ctarget[RIGHT_HAND].x[_Z_] += 0.017;
        tau = 1;
        myTarget = 4;
        run_draw_task();
    } else if (myTarget == 4) {
        ctarget[RIGHT_HAND].x[_X_] += 0.01;
        ctarget[RIGHT_HAND].x[_Z_] += 0.017;
        tau = 1;
        myTarget = 5;
        run_draw_task();
    } else if (myTarget == 5) {
        ctarget[RIGHT_HAND].x[_X_] += 0.009;
        ctarget[RIGHT_HAND].x[_Z_] += 0.017;
        tau = 1;
        myTarget = 6;
        run_draw_task();
    } else if (myTarget == 6) {
        ctarget[RIGHT_HAND].x[_X_] += 0.01/2;
        ctarget[RIGHT_HAND].x[_Z_] += 0.017;
        tau = 1;
        myTarget = 7;
        run_draw_task();
    } else if (myTarget == 7) {
        ctarget[RIGHT_HAND].x[_X_] += 0.025;
        ctarget[RIGHT_HAND].x[_Z_] += 0.01;
        tau = 1;
        myTarget = 8;
        run_draw_task();
    } else if (myTarget == 8) {
        ctarget[RIGHT_HAND].x[_X_] += 0.025;
        ctarget[RIGHT_HAND].x[_Z_] -= 0.007;
        tau = 1;
        myTarget = 9;
        run_draw_task();
```

Phase Graph: X-Z



Phase Graph: X-Y



CLMCPLOT-Data