Zhanpeng He
CSCI545 HW2

Question 1

a)

a) Since the COM of link i is located at P_i, they could be treated as discrete masses in space.

$$\Rightarrow \quad \bar{x} = \sum_{i=1}^{4} m_i x_i \Big/ \sum_{i=1}^{4} m_i = \frac{x_1+x_2+x_3+x_4}{4}$$

$$\bar{y} = \sum_{i=1}^{4} m_i y_i \Big/ \sum_{i=1}^{4} m_i = \frac{y_1+y_2+y_3+y_4}{4}$$

$$\Rightarrow \boxed{P_{cm} = (\bar{x}, \bar{y}) = \frac{1}{4}\sum_{i=1}^{4} P_i}$$

b)

b) $\dfrac{\partial P_4}{\partial \theta} = J(\theta)$

~~$\Rightarrow \dfrac{\partial P_4}{\partial t} \cdot \dfrac{\partial t}{\partial \theta} = J \Rightarrow \dot{P}_4 = J \cdot \dot{\theta}$~~

For revolut joint, $J_{Pi} = Z_{i-1} \times (P - P_{i-1})$

$$\frac{\partial P_4}{\partial \theta} = J(\theta) = (J_{P_1}, J_{P_2}, J_{P_3}, J_{P_4})$$

$$= (Z_0 \times (P_4 - P_0),\ Z_1 \times (P_4 - P_1),\ Z_2 \times (P_4 - P_2),\ Z_3 \times (P_4 - P_3))$$

where Z_i is the joint axis of joint i,
P_i is the vector from the origin of the world coordinate system to the origin of the ith link coordinate system.

c)

c) $P_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$, $P_1 = P_0 + \begin{pmatrix} l_1 \cdot \cos(\theta_1) \\ l_2 \cdot \sin(\theta_1) \\ 0 \end{pmatrix} = \begin{pmatrix} l_1 \cdot \cos(\theta_1) \\ l_2 \cdot \sin(\theta_1) \\ 0 \end{pmatrix}$

$P_2 = P_1 + \begin{pmatrix} l_2 \cdot \cos(\theta_1 + \theta_2) \\ l_2 \cdot \sin(\theta_1 + \theta_2) \\ 0 \end{pmatrix} = \begin{pmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \cdot \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \\ 0 \end{pmatrix}$

$P_3 = P_2 + \begin{pmatrix} l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ l_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ 0 \end{pmatrix} = \begin{pmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ l_1 \cdot \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ 0 \end{pmatrix}$

$P_4 = \begin{pmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) + l_4 \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) + l_4 \sin(\theta_1 + \theta_2 + \theta_3 + \theta_4) \\ 0 \end{pmatrix}$

$Z_i = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ for $i = 0 \cdots 4$

$\Rightarrow \frac{\partial P_4}{\partial \theta} = (J_{P_1}, J_{P_2}, J_{P_3}, J_{P_4})$

$J_{P_1} = \begin{pmatrix} -l_1 S_1 - l_2 S_{12} - l_3 S_{123} - l_4 S_{1234} \\ l_1 C_1 + l_2 C_{12} + l_3 C_{123} + l_4 C_{1234} \end{pmatrix}$

$J_{P_2} = \begin{pmatrix} -l_2 S_{12} - l_3 S_{123} - l_4 S_{1234} \\ l_2 C_{12} + l_3 C_{123} + l_4 C_{1234} \end{pmatrix}$

$J_{P_3} = \begin{pmatrix} -l_3 S_{123} - l_4 S_{1234} \\ l_3 C_{123} + l_4 C_{1234} \end{pmatrix}$

$J_{P4} = \begin{pmatrix} -l_4 S_{1234} \\ l_4 C_{1234} \end{pmatrix}$

Here, S_{ij} means $\sin(\theta_i + \cdots + \theta_j)$, C_{ij} means $\cos(\theta_i + \cdots + \theta_j)$

d)

$$\frac{\partial P_3}{\partial \theta} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} - l_3 s_{123} & -l_2 s_{12} - l_3 s_{123} & -l_3 s_{123} & 0 \\ l_1 c_1 + l_2 c_{12} + l_3 c_{123} & l_2 c_{12} + l_3 c_{123} & l_3 c_{123} & 0 \end{bmatrix}$$

$$\frac{\partial P_2}{\partial \theta} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} & 0 & 0 \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} & 0 & 0 \end{bmatrix}$$

$$\frac{\partial P_1}{\partial \theta} = \begin{bmatrix} -l_1 s_1 & 0 & 0 & 0 \\ l_1 c_1 & 0 & 0 & 0 \end{bmatrix}$$

e)

$$\frac{\partial P_{cm}}{\partial \theta} = \frac{\partial \frac{1}{4} \sum_{i=1}^{4} P_i}{\partial \theta}$$

$$= \frac{1}{4} \sum_{i=1}^{4} \frac{\partial P_i}{\partial \theta}$$

$$\Rightarrow$$

$$\frac{\partial P_{cm}}{\partial \theta} = \frac{1}{4} \begin{bmatrix} -4l_1 s_1 - 3l_2 s_{12} - 2l_3 s_{123} - l_4 s_{1234} & -3l_2 s_2 - 2l_3 s_{123} - l_4 s_{1234} & -2l_3 s_{123} - l_4 s_{1234} & -l_4 s_{1234} \\ 4l_1 c_1 + 3l_2 c_{12} + 2l_3 c_{123} + l_4 c_{1234} & 3l_2 c_{12} + 2l_3 c_{123} + l_4 c_{1234} & 2l_3 c_{123} + l_4 c_{1234} & l_4 c_{1234} \end{bmatrix}$$

Zhanpeng He
CSCI545 HW2

f) Here is a screenshot of my implementation of the calculation of Jacobian:

```
% NOTE: insert your Jacobian calculation here

% Compute Si and Ci in my formula
cosines = zeros(n, 1);
sines = zeros(n, 1);
for i = 2:n
    cosines(i,:) = cos(sum(theta(1:i)));
    sines(i,:) = sin(sum(theta(1:i)));
end

% Compute pi
p = zeros(3, n);
p(:, 1) = [ links(1)*cos(theta(1)), links(1)*sin(theta(1)), 0 ]';
for i = 2:n
    p(:, i) = p(:,i-1)+[links(i)*cosines(i,:), links(i)*sines(i,:), 0]';
end

% Compute Jacobians of the joints
jp = zeros(3, n, n);
z = [0 0 1]';

for i = 1:n
   % z0 x (pn-p0) in my formula
   jp(:,1,i) = cross(z, p(:,i)-[0 0 0]');
   for j = 2:i
      jp(:,j,i) = cross(z, p(:,i)-p(:,j-1));
   end
end

j_mean = sum(jp, 3)/n;
J = j_mean(1:2, :);
```

Zhanpeng He
CSCI545 HW2

g) The formula:

g) Jacobian Transpose derivation:

Minimize cost fuction:

$$F = \frac{1}{2}(x_{target} - x)^T (x_{target} - x)$$
$$= \frac{1}{2}(x_{target} - f(\theta))^T (x_{target} - f(\theta))$$

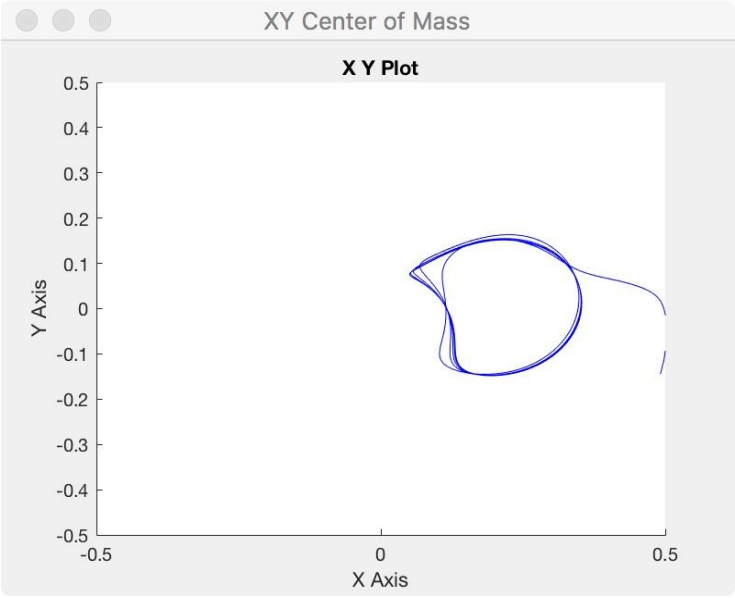
$$\Rightarrow \Delta\theta = -\alpha \left(\frac{\partial F}{\partial \theta}\right)^T$$
$$= \alpha \left((x_{target} - x)^T \frac{\partial f(\theta)}{\partial \theta}\right)^T$$
$$= \alpha J^T(\theta) \Delta x$$

$$\Rightarrow \boxed{\Delta\theta = \alpha J^T(\theta) \Delta x} \Rightarrow \text{Jacobian transpose for inverse kinematics}$$

And my implementation (I did not change the Jacobian calculation, please check my source code in inverse_kinematics.m):

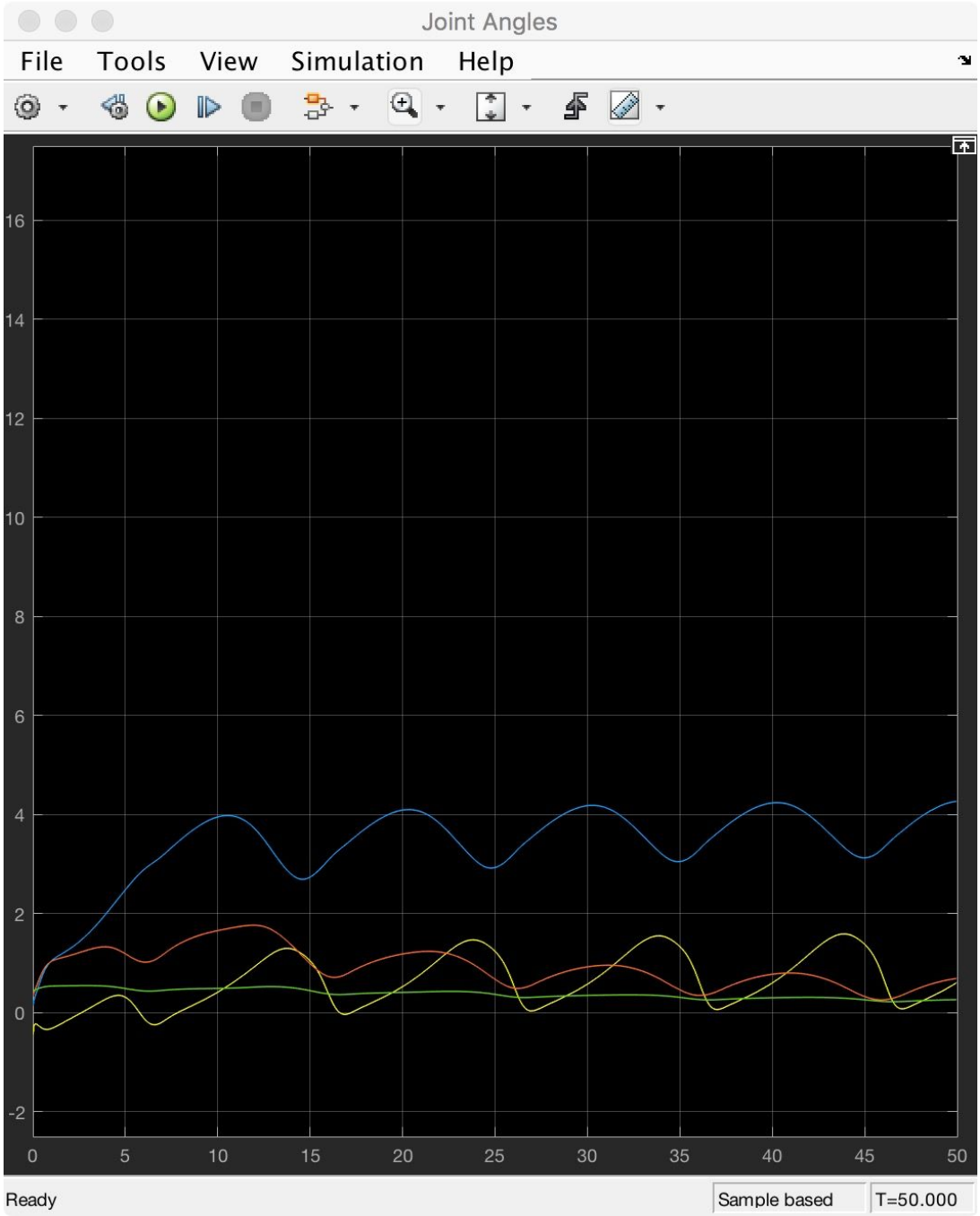
```
% Question 1g
alpha = 1;
thetad = alpha * J' * xd;
```

Print out of XY plane:



Zhanpeng He
CSCI545 HW2

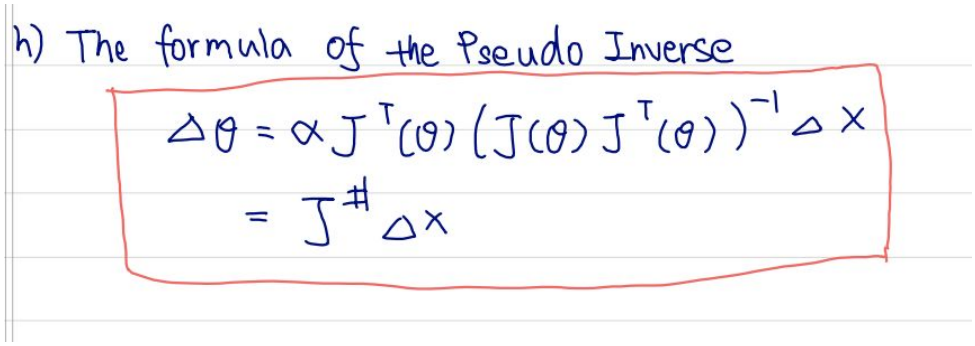
Print out of joints trajectory:



Comment: This method could draw a shape that is very similar to a circle. The performance is not very well. I also notice that the choice of alpha can affect whether the algorithm converges(I tried very small and large alpha, both of them failed to converge).

Zhanpeng He
CSCI545 HW2

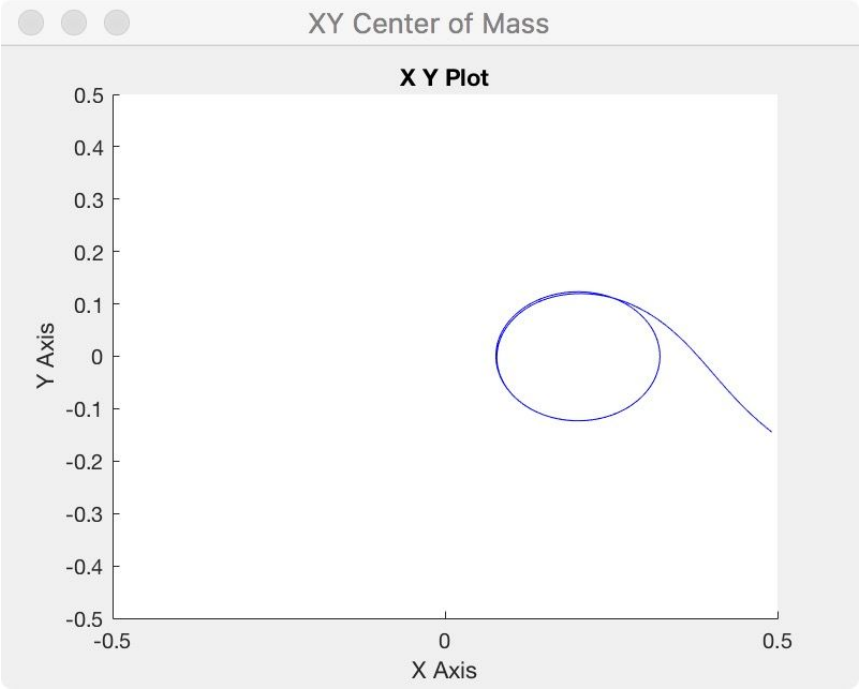
h) The formula:



And my implementation (I did not change the Jacobian calculation, please check my source code in inverse_kinematics.m):

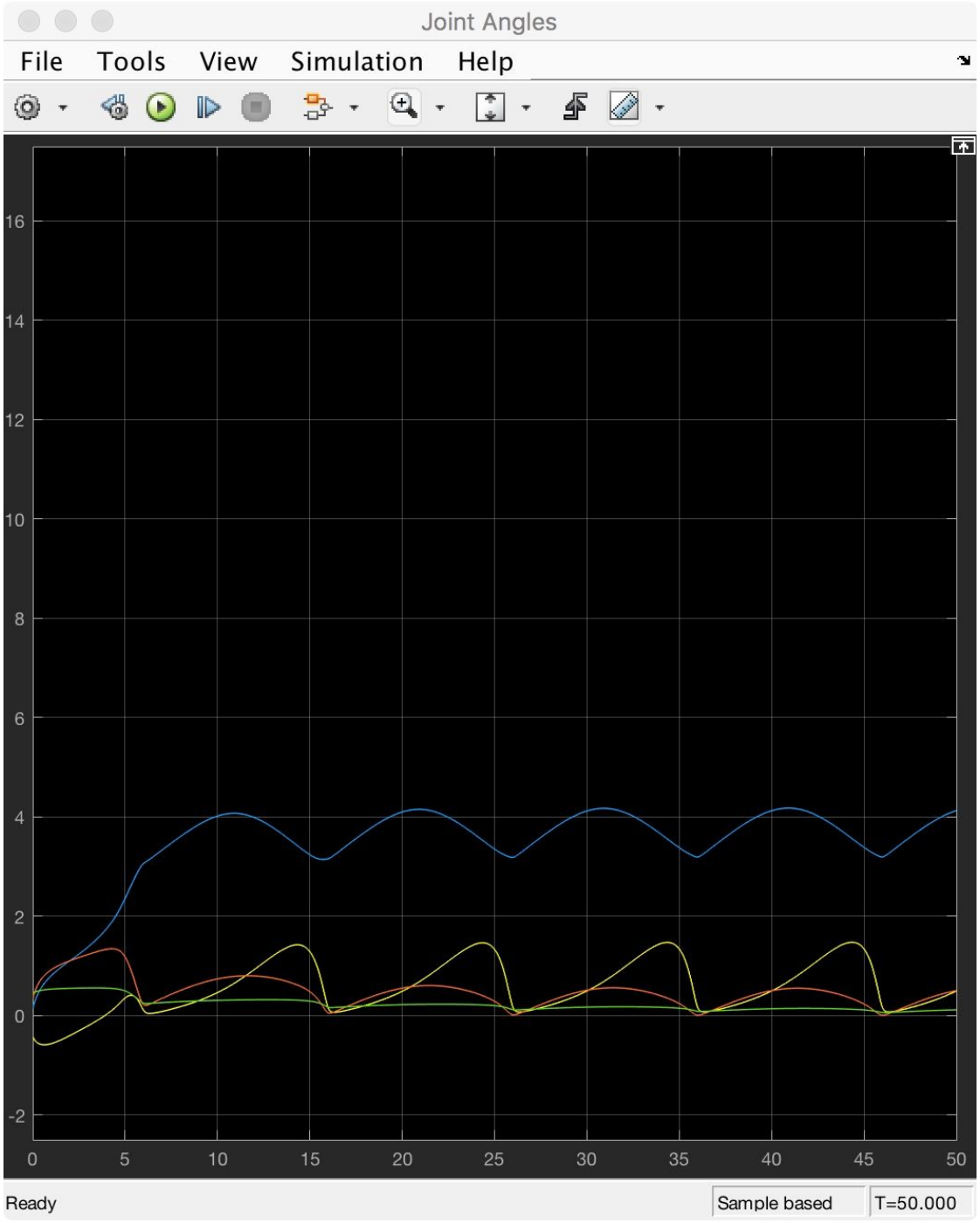
```
% Question 1h
alpha = 0.009;
thetad = alpha * J' * inv(J * J') * xd;
```

Print out of XY plane:



Zhanpeng He
CSCI545 HW2

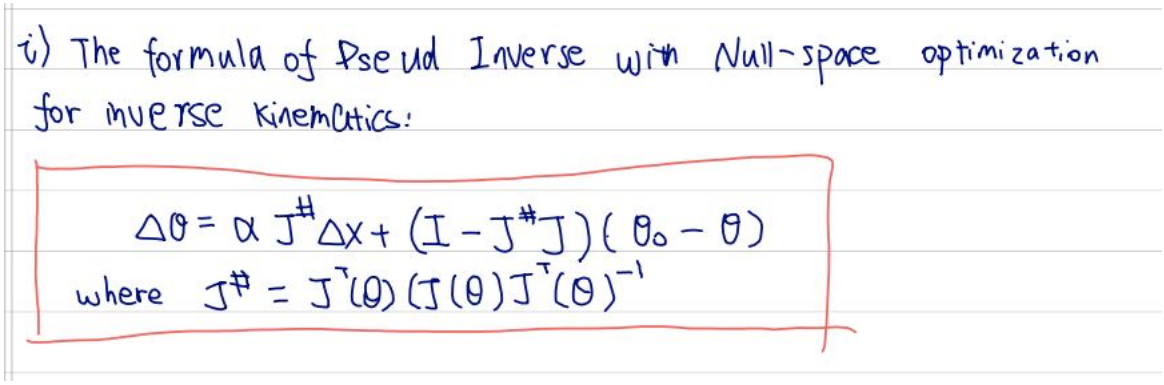
The joints trajectory:



Comments: This method perform much better than that of g). It draws a better circle even though it's not perfect. Comparing to g), it is a second order method. Also, this method converge faster than g). I notice that I have to choose a much smaller alpha to make the

algorithm converge so I guess the result of this method is much larger. However, I can't confirm the reason because I don't know how to investigate the variable values in simulink(this is my second time using matlab). If I use the same alpha as g (alpha = 1), the robot arm will have very bad performance and it is not even drawing anything. Hence, after a few trials, I selected 0.009 as my alpha for this method.

i) Formula:

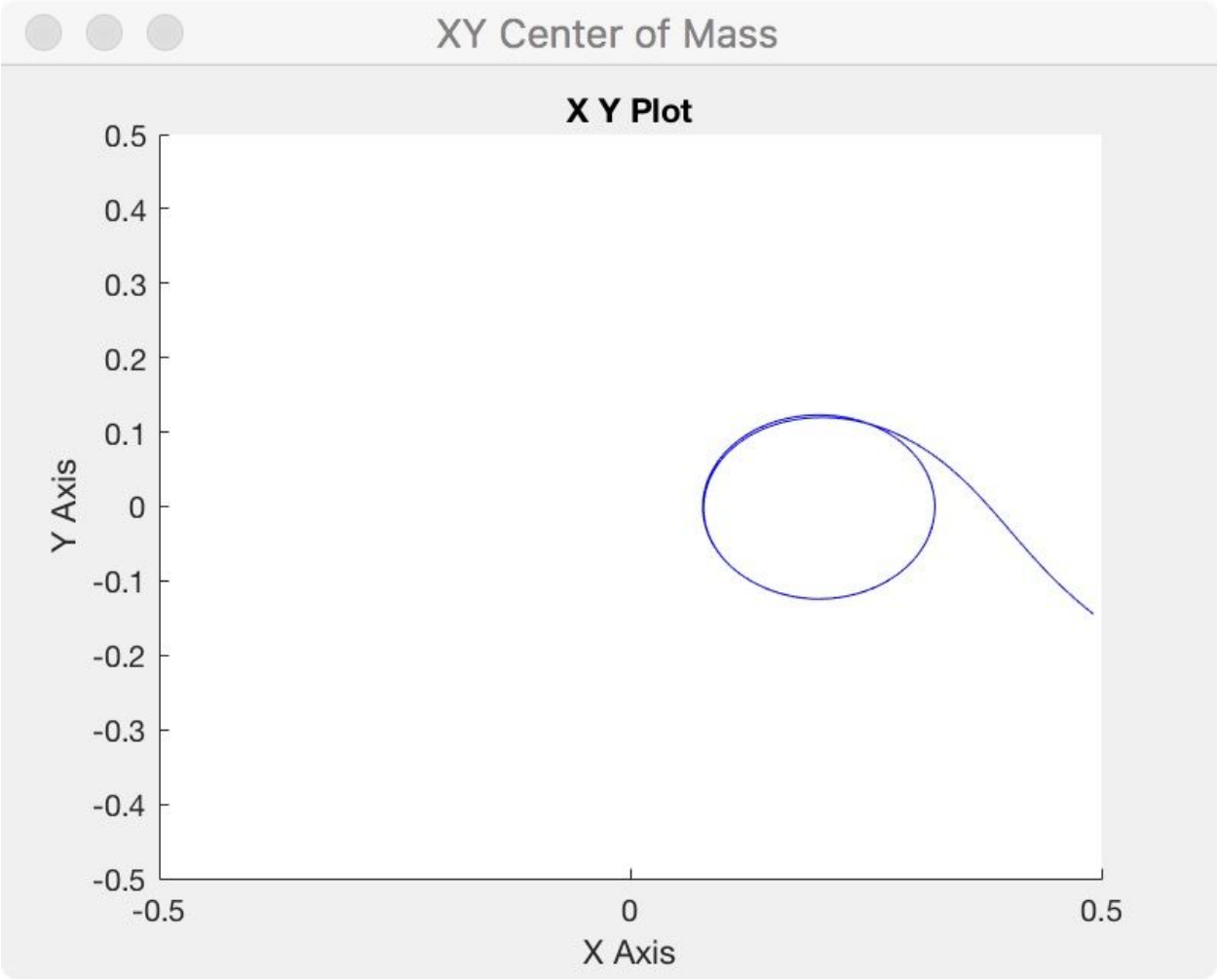


And my implementation (I did not change the Jacobian calculation, please check my source code in inverse_kinematics.m):

```
% Question 1i
alpha = 0.009;
J_sharp = J' * inv( J * J' );
theta_o = 0.1 * ones(n,1);
thetad = alpha * J_sharp * xd + ( eye(n) - J_sharp * J ) * ( theta_o - theta );
```

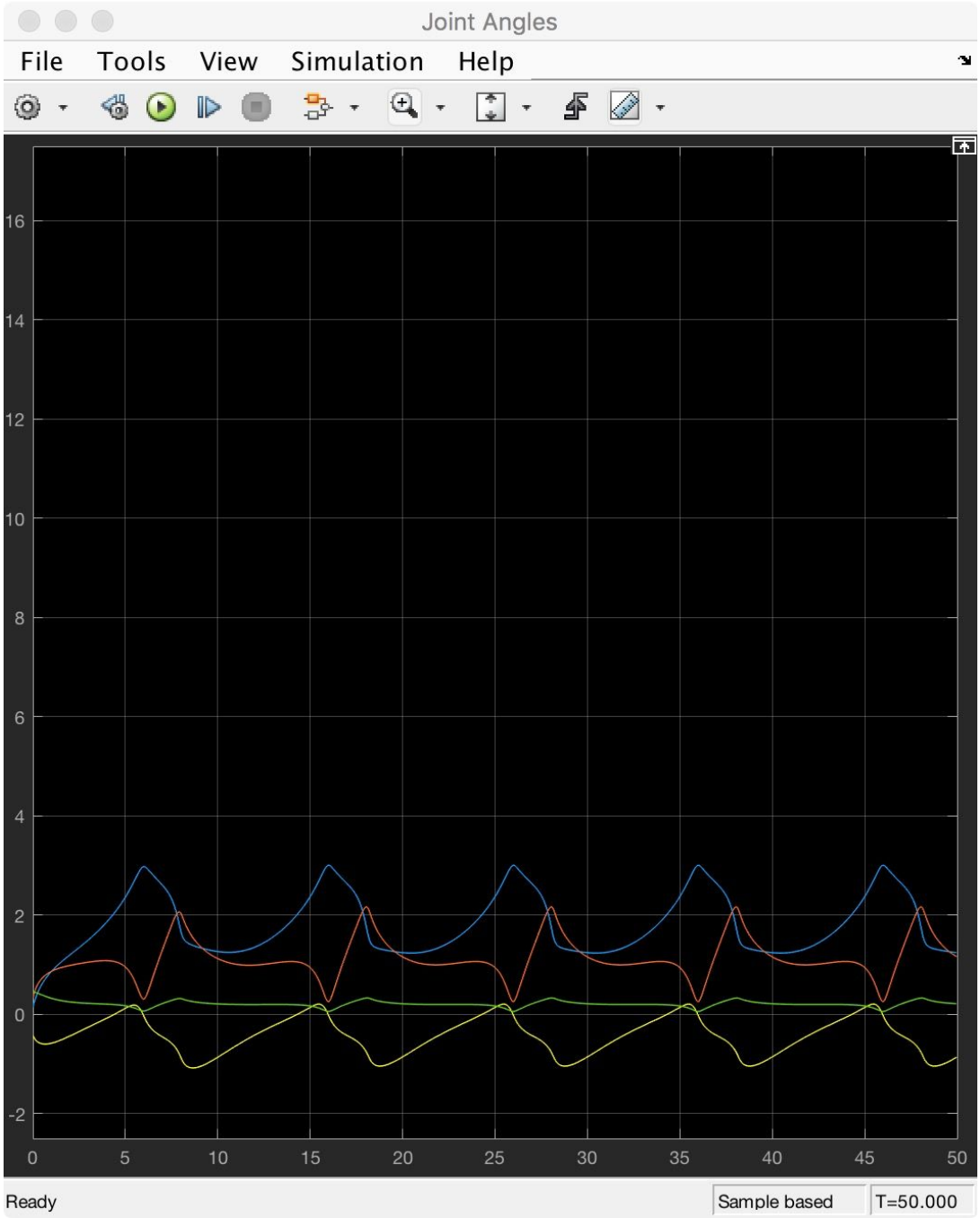
Zhanpeng He
CSCI545 HW2

Print out of XY plane:



Zhanpeng He
CSCI545 HW2

The joints trajectory:



This method perform similarly as h). It draws the circle well and converge fast enough. I kept the same alpha as h) since it performs well. Comparing to h), the explicit optimization criterion provides control over arm configurations.

j)

j) Derivation:

For a small step Δx, minimize the cost function:

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

$\Rightarrow$

(1) $\frac{\partial F}{\partial \lambda} = 0 \Rightarrow \Delta x = J\Delta\theta$

(2) $\frac{\partial F}{\partial \Delta\theta} = 0 \Rightarrow \Delta\theta^T W - \lambda^T J = 0$

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

Since W is a diagonal matrix, $(W^{-1})^T = W^{-1}$

$\Rightarrow \Delta\theta = W^{-1} J^T \lambda$

$J\Delta\theta = J W^{-1} J^T \lambda$

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

By (1): $\Delta x = J\Delta\theta$

$\Rightarrow \lambda = (J W^{-1} J^T)^{-1} \Delta x$

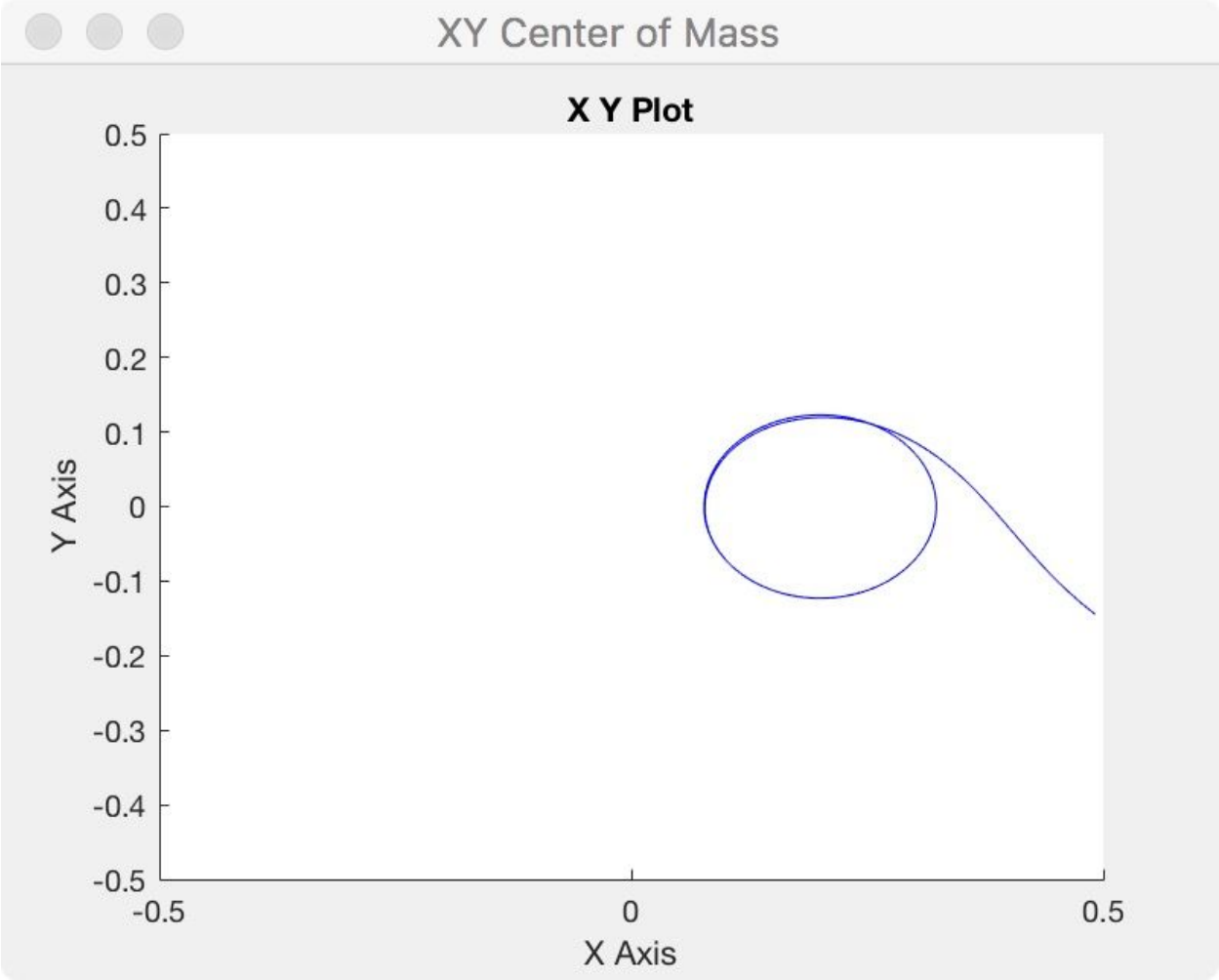
Plug this into $\Delta\theta = W^{-1} J^T \lambda$

$$\boxed{\Delta\theta = W^{-1} J^T (J W^{-1} J^T)^{-1} \Delta x}$$

And my implementation (I did not change the Jacobian calculation, please check my source code in inverse_kinematics.m):

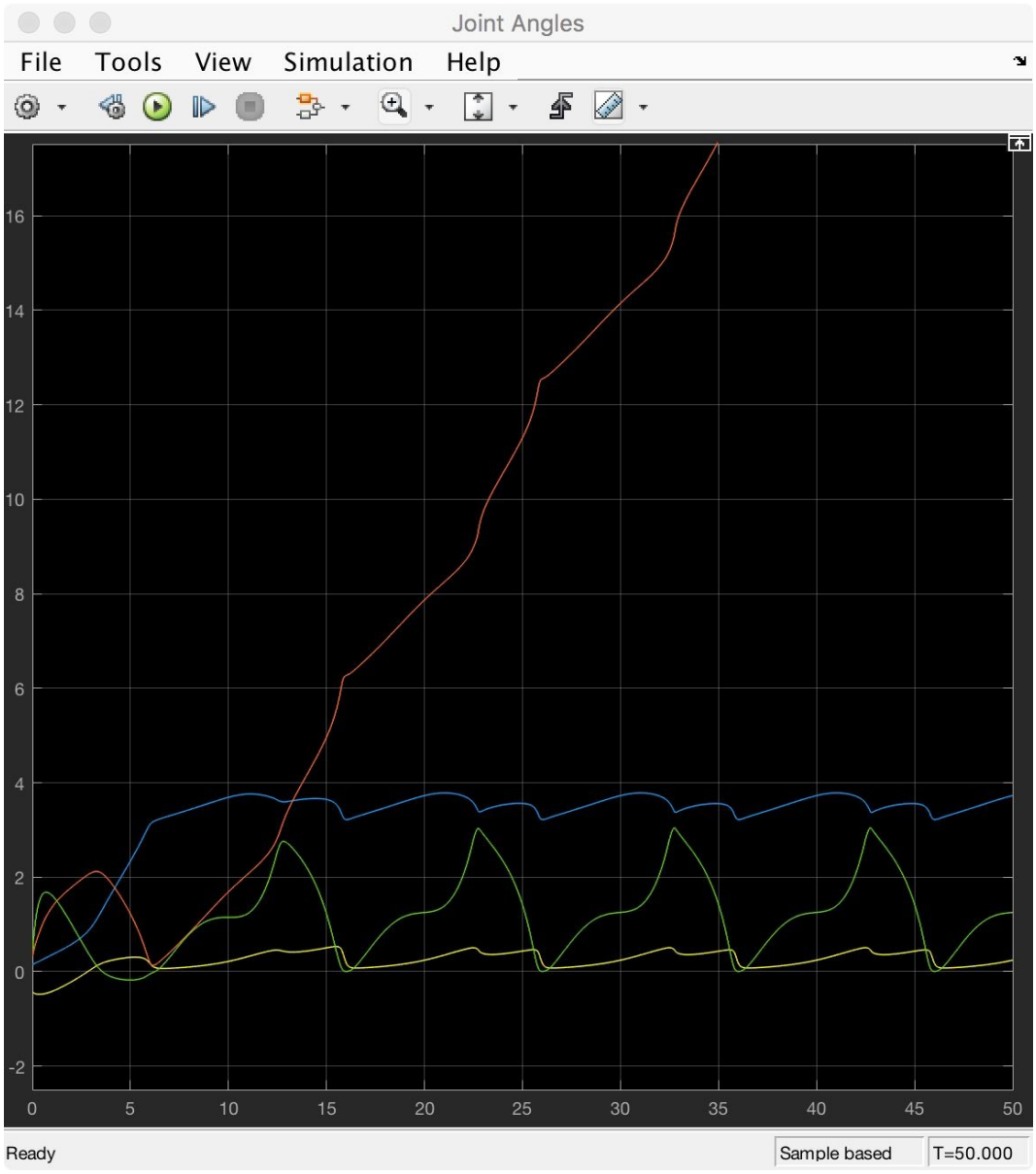
```
% Question 1j
alpha = 0.009;
W = diag([1.0 0.5 0.1 0.01]);
thetad = alpha * inv(W) * J' * inv( J * inv(W) * J' ) * xd;
```

Print out of XY plane:



Zhanpeng He
CSCI545 HW2

The joints trajectory:



Comments:

This method draws the circle well and also converges fast. However, by adding different weights to different joints, the joints behave very differently. The reason is the weights given by the problem has huge bias to some of the joints. The first weight 1 is 100 times the last one 0.01. Hence, one of the joints actually “exploded”.

Zhanpeng He
CSCI545 HW2

k) The formula:

k) Formula:

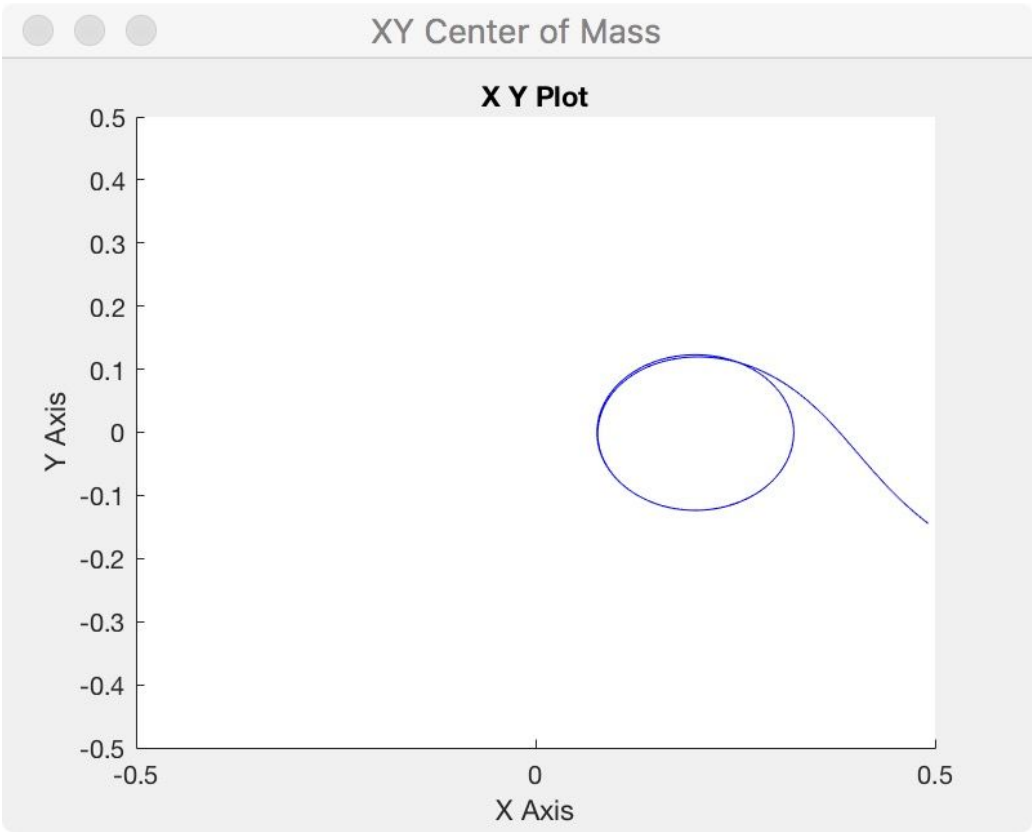
$$\Delta\theta = \alpha J_w^{\#} \delta x + (I - J_w^{\#} J)(\theta_0 - \theta)$$

where $J_w^{\#} = W^{-1} J^T (J W^{-1} J^T)^{-1}$

And my implementation (I did not change the Jacobian calculation, please check my source code in inverse_kinematics.m):

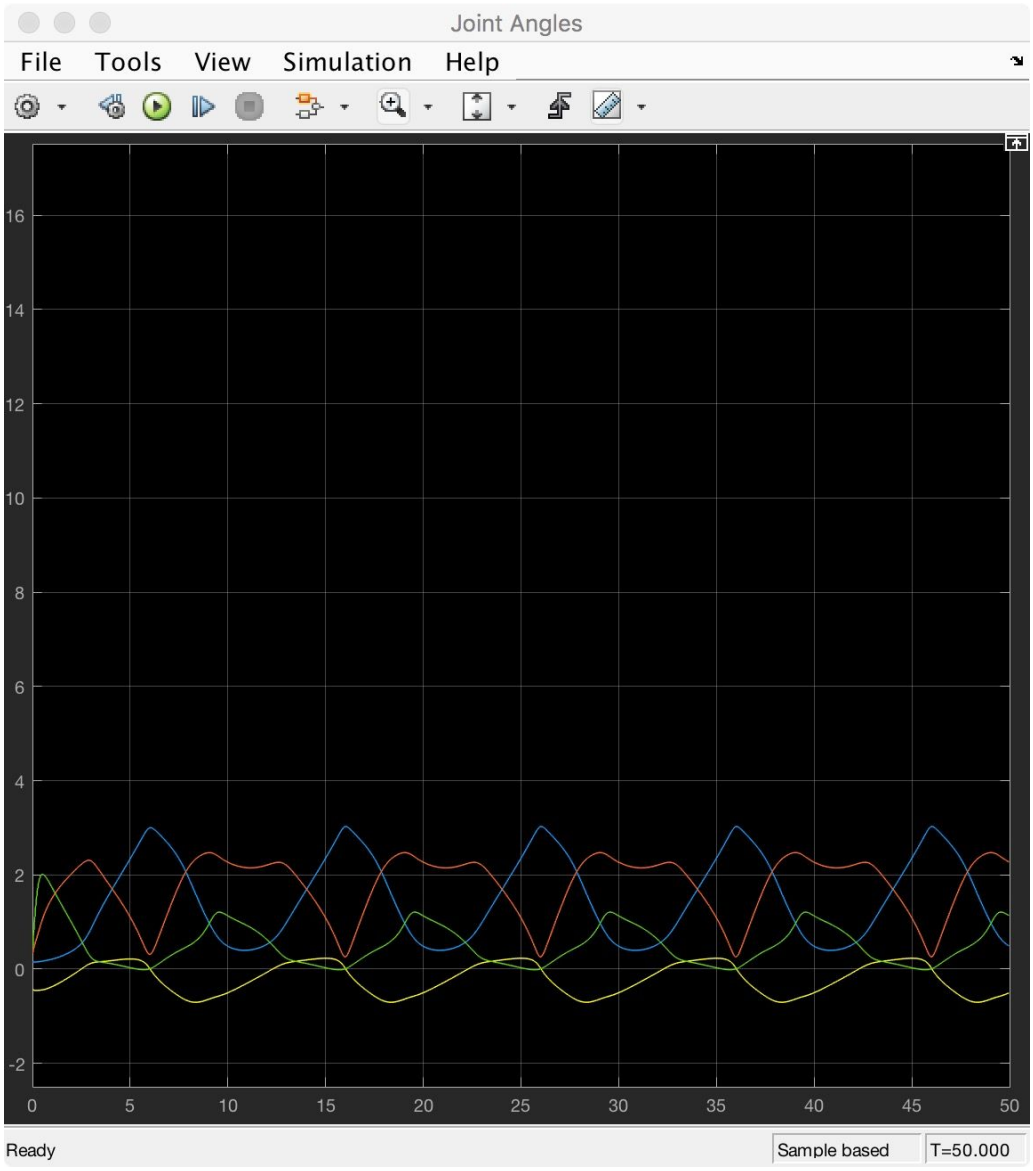
```
% Question 1k
alpha = 0.009;
W = diag([1.0 0.5 0.1 0.01]);
theta_o = 0.1 * ones(n,1);
J_w_sharp = inv(W) * J' * inv( J * inv(W) * J' );
thetad = alpha * J_w_sharp * xd + ( eye(n) - J_w_sharp * J ) * ( theta_o - theta );
```

Print out of XY plane:



Zhanpeng He
CSCI545 HW2

The joints trajectory:



Comments:

Like the previous method, this method draws the circle well and converges fast. By adding explicit optimization criterion, this method control the behavior of the joints and avoid the “explosion” behavior of the previous method. Every joints move within a range of joint angles.

Zhanpeng He
CSCI545 HW2

Question 2:

1) **I have chosen these points:**
   a) **(X, Y, Z) = (-0.16554, 0.10235, -0.08703)**
   b) **(X, Y, Z) = (-0.12554, 0.10235, -0.08703)**
   c) **(X, Y, Z) = (-0.12554, 0.10235, -0.04703)**
   d) **(X, Y, Z) = (-0.16554, 0.10235, -0.04704)**

Here is a snippet of my codes that set the four target:

```
double y_target = 0.10235;

switch(axis_id){
  case 0:
    ctarget[LEFT_HAND].x[_X_] = -0.16554;
    ctarget[LEFT_HAND].x[_Z_] = -0.08703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  case 1:
    ctarget[LEFT_HAND].x[_X_] = -0.12554;
    ctarget[LEFT_HAND].x[_Z_] = -0.08703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  case 2:
    ctarget[LEFT_HAND].x[_X_] = -0.12554;
    ctarget[LEFT_HAND].x[_Z_] = -0.04703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  case 3:
    ctarget[LEFT_HAND].x[_X_] = -0.16554;
    ctarget[LEFT_HAND].x[_Z_] = -0.04703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  default:
    break;
}
```

Zhanpeng He
CSCI545 HW2

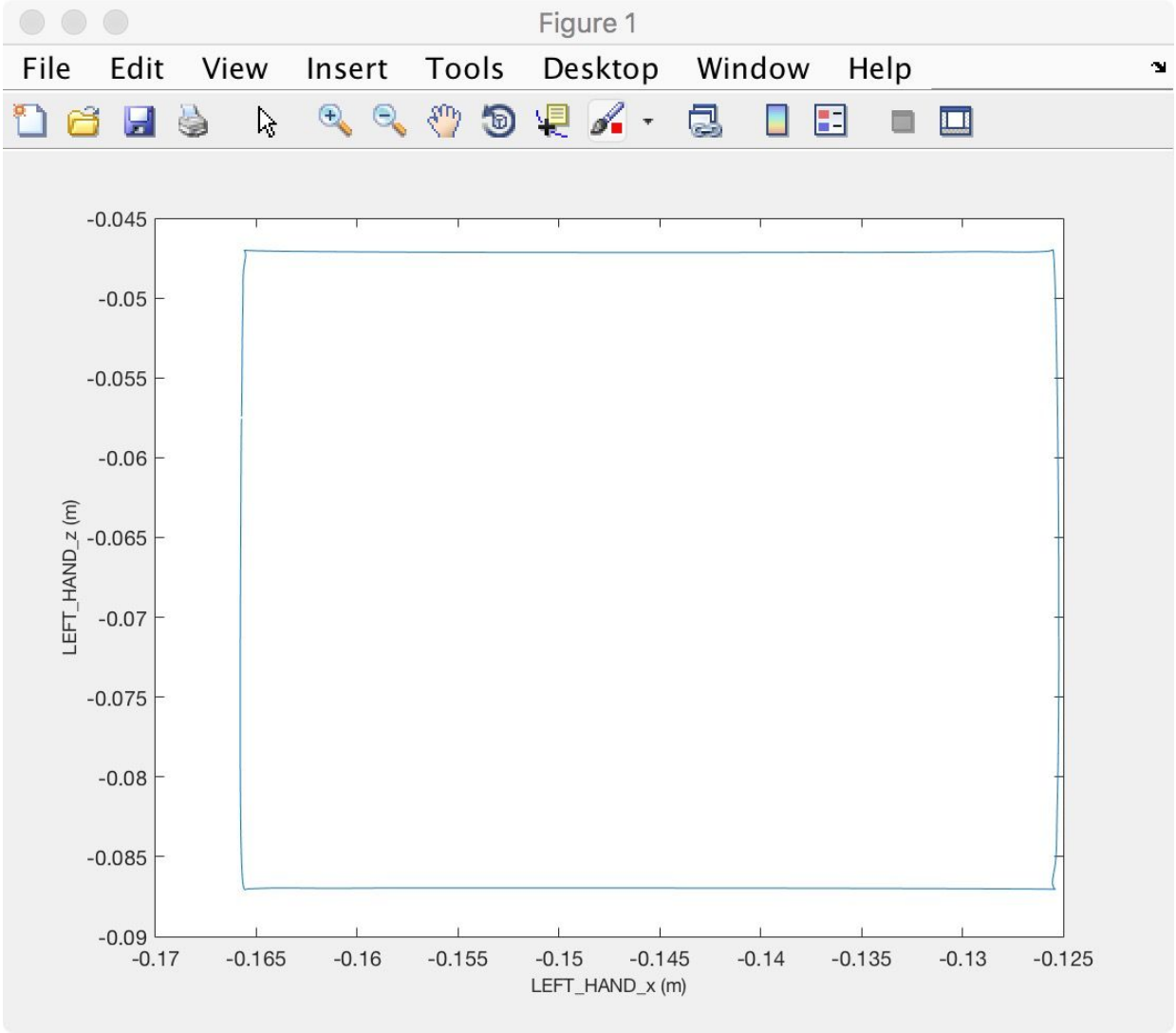
b) Here are the codes that I mainly worked on. Please refer to cubic_spline_task.cpp for details.

```
// compute the update for the desired states
double t = movement_time - tau;

// to determine current target state and the time left to finish the current axis
double axis_time_left = (double)axis_time - (t - axis_time*((int)(t/axis_time)));

double y_target = 0.10235;

switch(axis_id){
  case 0:
    ctarget[LEFT_HAND].x[_X_] = -0.16554;
    ctarget[LEFT_HAND].x[_Z_] = -0.08703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  case 1:
    ctarget[LEFT_HAND].x[_X_] = -0.12554;
    ctarget[LEFT_HAND].x[_Z_] = -0.08703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  case 2:
    ctarget[LEFT_HAND].x[_X_] = -0.12554;
    ctarget[LEFT_HAND].x[_Z_] = -0.04703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  case 3:
    ctarget[LEFT_HAND].x[_X_] = -0.16554;
    ctarget[LEFT_HAND].x[_Z_] = -0.04703;
    ctarget[LEFT_HAND].x[_Y_] = y_target;
    break;
  default:
    break;
}

if(axis_time_left <= time_step){
  // printf("[INFO] Catch time of 0! time left: %.2f. Current time: %.2f\n", axis_time_left, t);
  axis_time_left = axis_time;
  axis_id = (axis_id+1)%4;
  //printf("[INFO] Switch to task: %d, time left updated: %f\n", axis_id, axis_time_left);
  //printf("[INFO] Current task: (X_TARGET, Z_TARGET)=(%.2f, %.2f)\n", x_target, z_target);
}
```

```
double x_next=0.0, x_next_d=0.0, x_next_dd=0.0;
double z_next=0.0, z_next_d=0.0, z_next_dd=0.0;
double y_next=0.0, y_next_d=0.0, y_next_dd=0.0;

cubic_spline_next_step (cnext[LEFT_HAND].x[_X_], cnext[LEFT_HAND].xd[_X_], cnext[LEFT_HAND].xdd[_X_], ctarget[LEFT_HAND].x[_X_], 0.0, 0.0,
    axis_time_left, time_step, &x_next, &x_next_d, &x_next_dd);
cubic_spline_next_step (cnext[LEFT_HAND].x[_Z_], cnext[LEFT_HAND].xd[_Z_], cnext[LEFT_HAND].xdd[_Z_], ctarget[LEFT_HAND].x[_Z_], 0.0, 0.0,
    axis_time_left, time_step, &z_next, &z_next_d, &z_next_dd);

cubic_spline_next_step (cnext[LEFT_HAND].x[_Y_], cnext[LEFT_HAND].xd[_Y_], cnext[LEFT_HAND].xdd[_Y_], ctarget[LEFT_HAND].x[_Y_], 0.0, 0.0,
    axis_time_left, time_step, &y_next, &y_next_d, &y_next_dd);

cnext[LEFT_HAND].x[_X_] = x_next;
cnext[LEFT_HAND].x[_Z_] = z_next;
cnext[LEFT_HAND].x[_Y_] = y_next;
cnext[LEFT_HAND].xd[_X_] = x_next_d;
cnext[LEFT_HAND].xd[_Z_] = z_next_d;
cnext[LEFT_HAND].xd[_Y_] = y_next_d;
cnext[LEFT_HAND].xdd[_X_] = x_next_dd;
cnext[LEFT_HAND].xdd[_Z_] = z_next_dd;
cnext[LEFT_HAND].xdd[_Y_] = y_next_dd;

tau -= time_step;
```

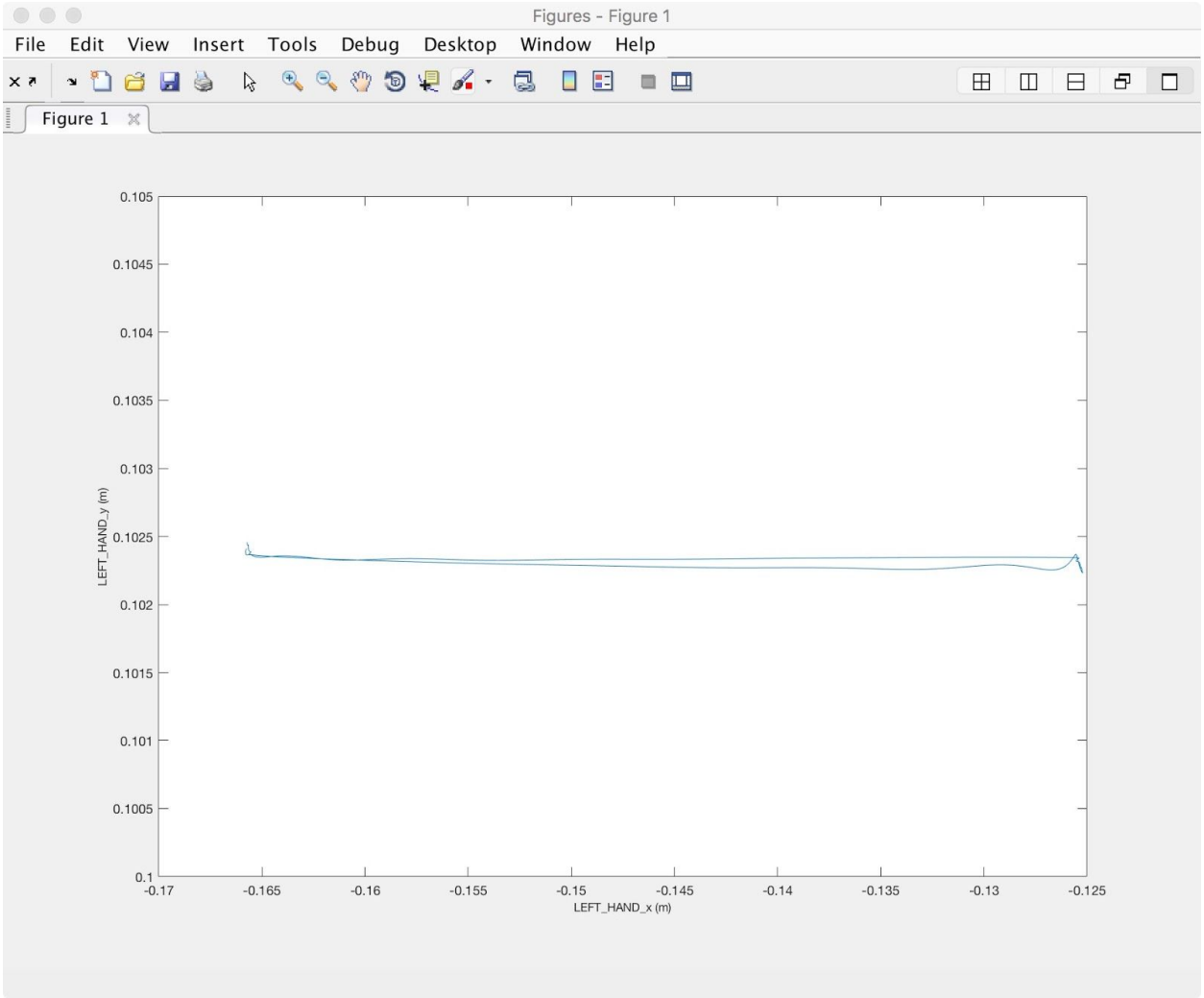
c) Here are the x-z, z-y, x-y graphs for one iteration of drawing:

Zhanpeng He
CSCI545 HW2



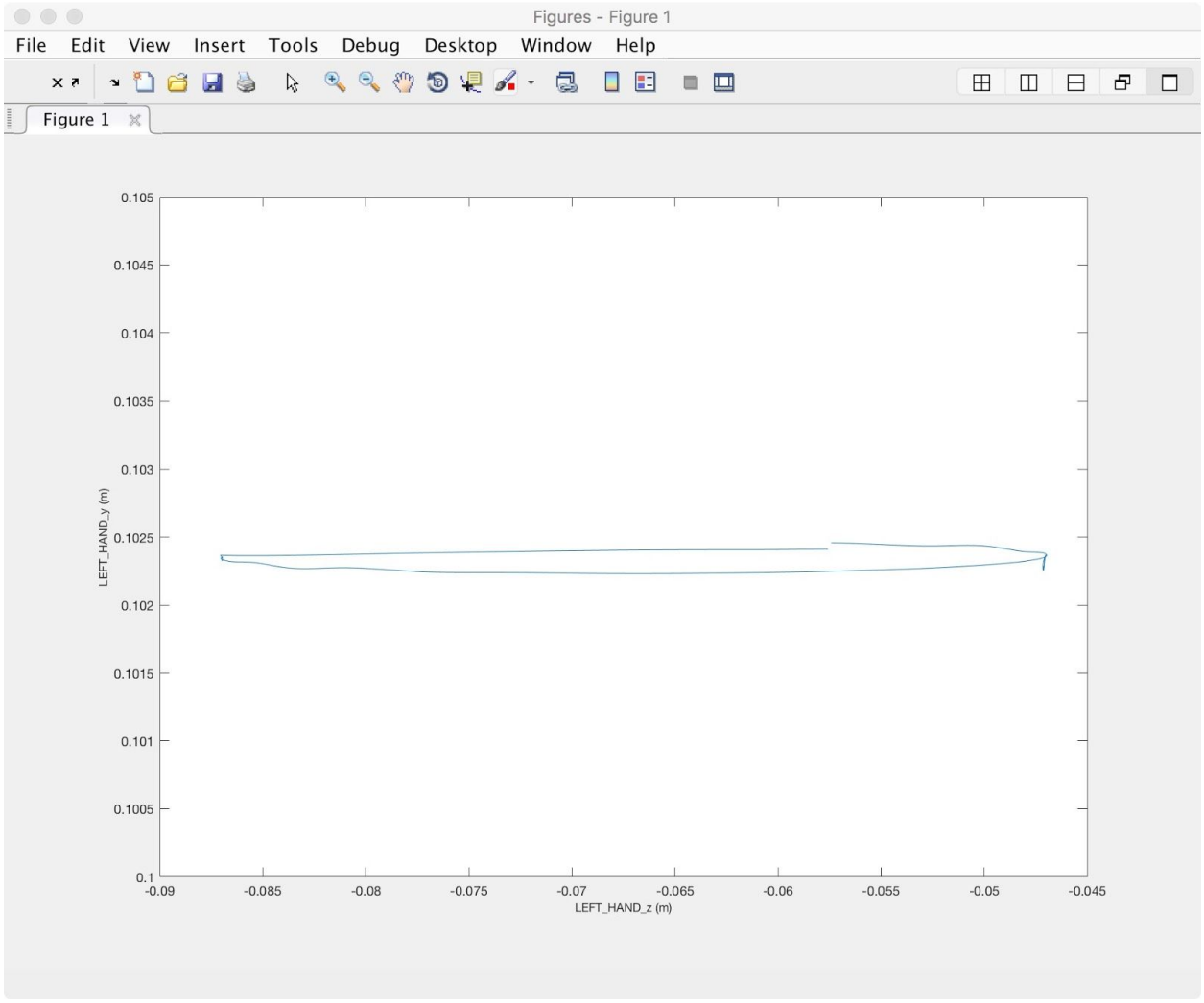
For the xz graph, we can see that it draws a square well. Actually, it can finish all target very well except the one on the bottom right corner. There are a couple possible reasons. One of them can be that the y axis is not nicely stable at the coordinate that I set. If you take a look at the x-y, y-z graph, it failed to stay at constant.

Zhanpeng He
CSCI545 HW2



Like I said, y did not stay at constant nicely. The reason might be I used cnext as current state for the argument of the cubic spline and it might be an inaccurate estimation of current state.

Zhanpeng He
CSCI545 HW2



Like x-y, y did not stay at constant.