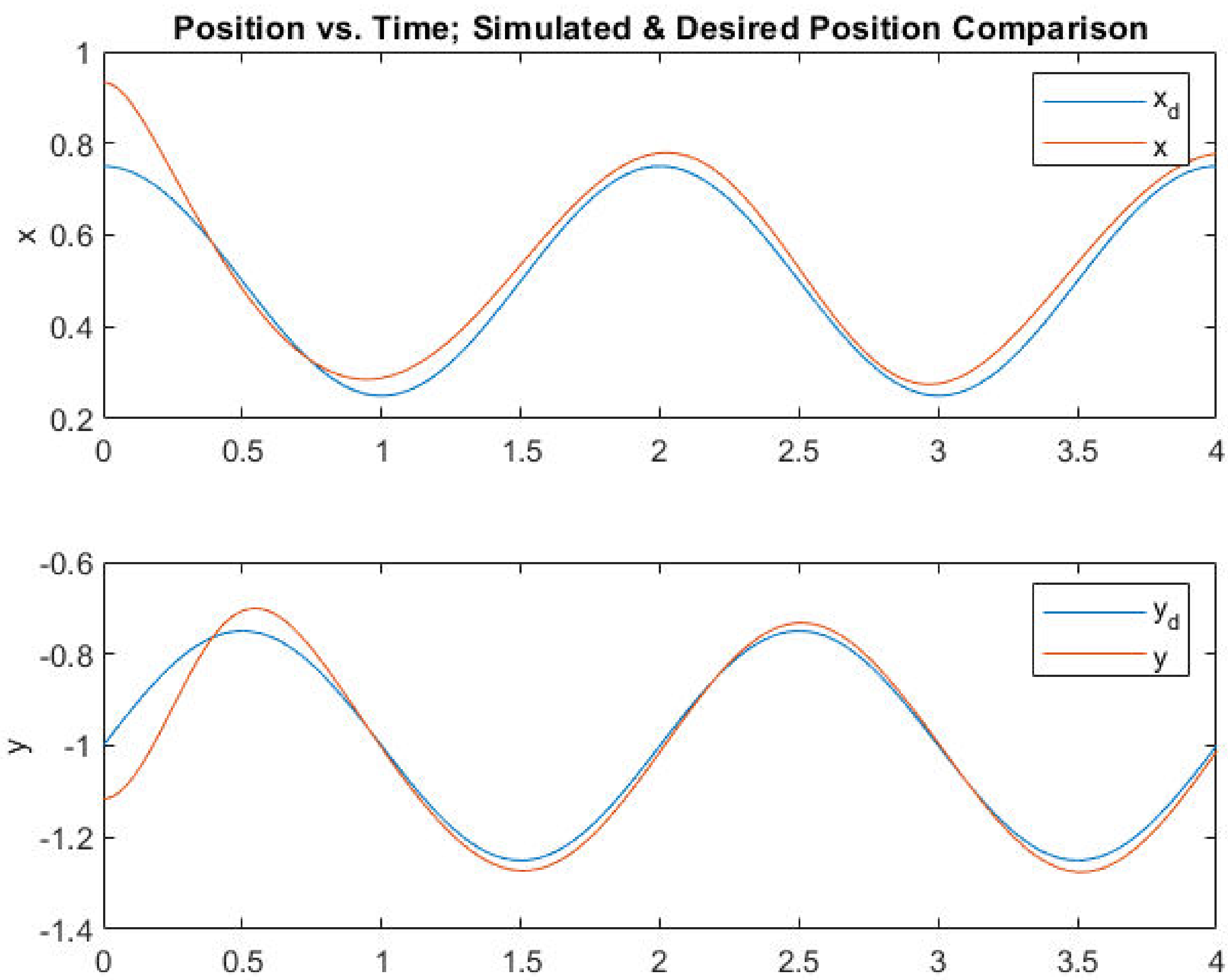
Ryan Dewsnap
32000408
CS403 HW7

1.
$$\tau = J^\top \left[\Lambda\left(\ddot{\mathbf{r}}_C^d + K(\mathbf{r}_C^d - \mathbf{r}_C) + D(\dot{\mathbf{r}}_C^d - \dot{\mathbf{r}}_C)\right) + \mu + \rho\right]$$

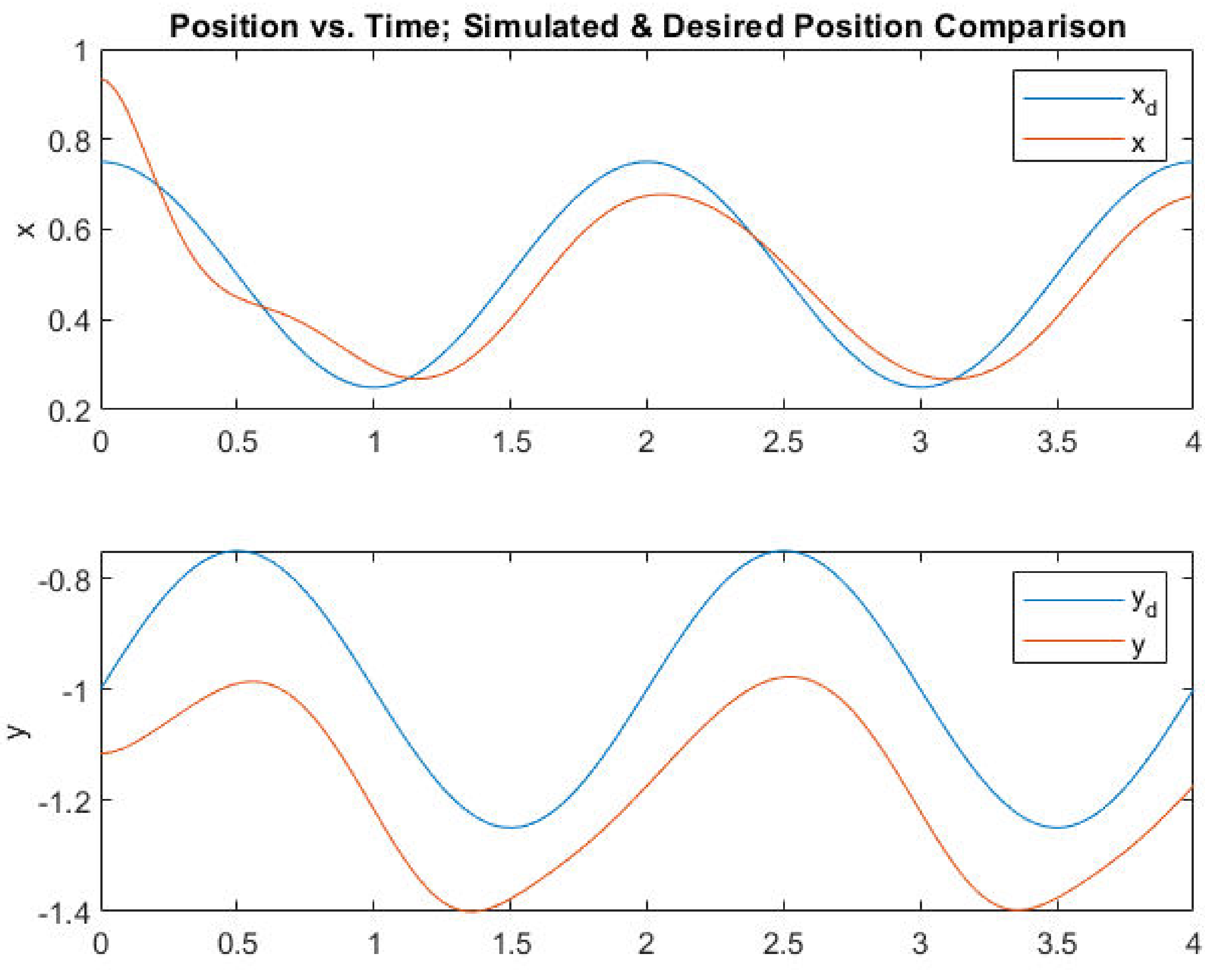
Complete control law: Works as expected, end effector traces circle closely, desired and actual graphs are very close.



2.

$$\tau = J^T \left[\Lambda \left(\ddot{\mathbf{r}}_C^d + K(\mathbf{r}_C^d - \mathbf{r}_C) + D(\dot{\mathbf{r}}_C^d - \dot{\mathbf{r}}_C)\right) + \mu\right]$$

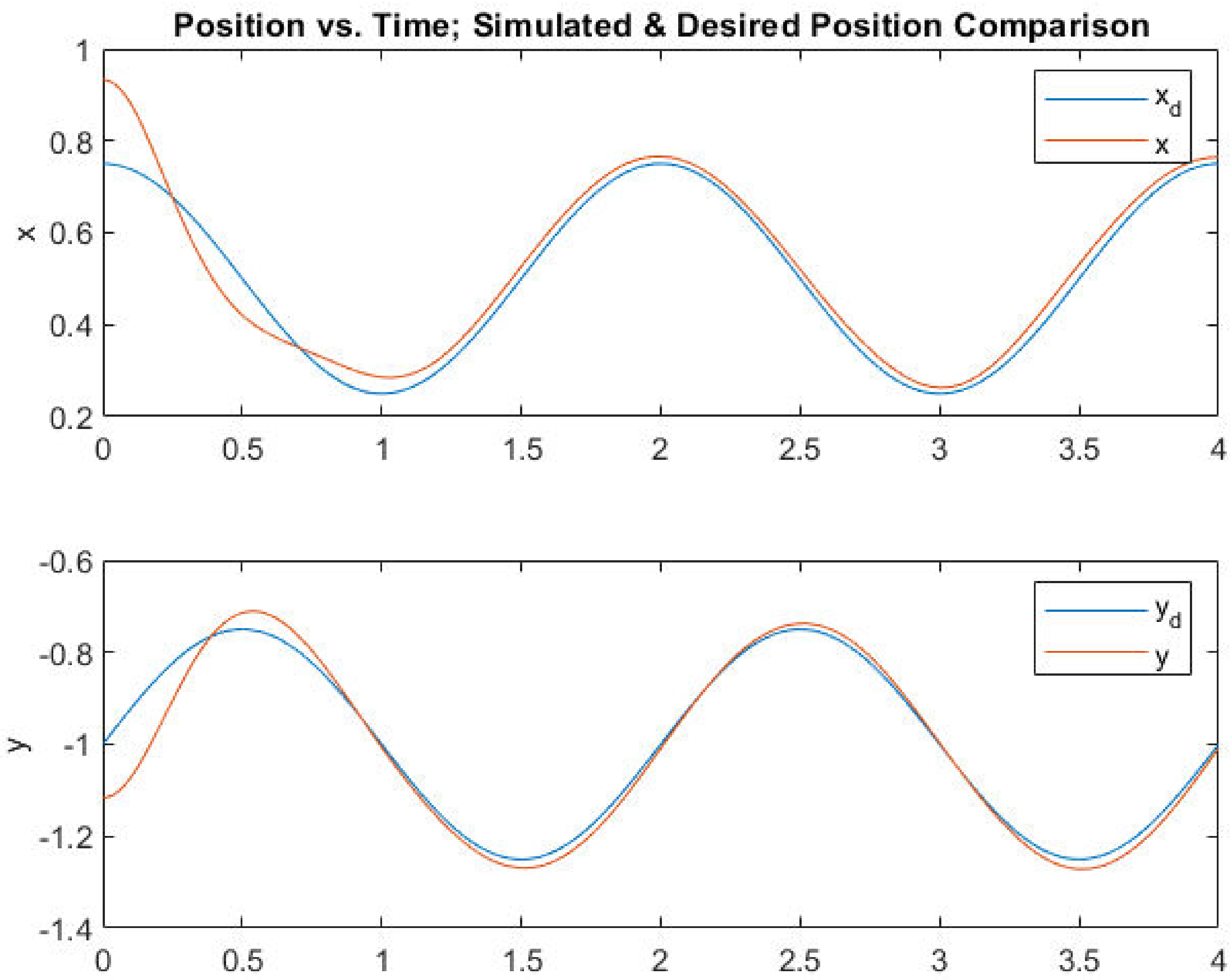
Rho is missing, gravity incorrectly accounted for causing a correct circle to be drawn in the wrong position.



3.

$$\tau = J^T \left[\Lambda \left(\ddot{\mathbf{r}}_C^d + K(\mathbf{r}_C^d - \mathbf{r}_C) + D(\dot{\mathbf{r}}_C^d - \dot{\mathbf{r}}_C) \right) + \rho \right]$$

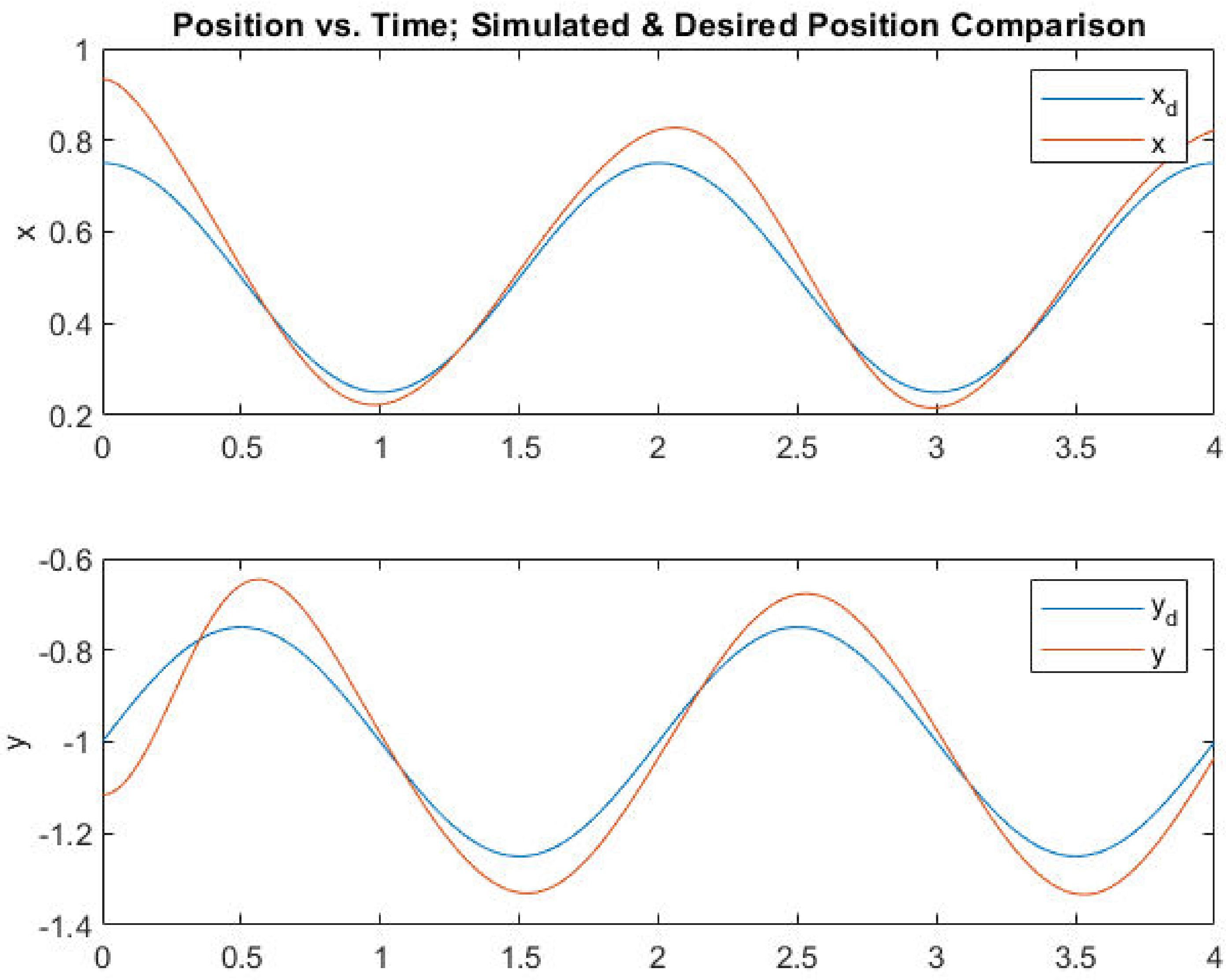
Mu is missing, this seems to affect the beginning adjustment made by the arm. Without mu, the movement is more unstable in the beginning and takes longer to converge with the target path.



4.

$$\tau = J^T \left[\Lambda \left(K(\mathbf{r}_C^d - \mathbf{r}_C) + D(\dot{\mathbf{r}}_C^d - \dot{\mathbf{r}}_C) \right) + \mu + \rho \right]$$

This control law excludes acceleration in lambda. The position is mostly off and draws a bigger circle than the target as it neglects those outward forces acting on it as it draws a circle.



```
clear all

%% Paramter preparation
syms th1 dth1 ddth1 th2 dth2 ddth2 real
syms c1 l1 c2 l2 m1 I1 m2 I2 real
syms g tau1 tau2 real

q = [th1; th2];
dq = [dth1; dth2];
ddq = [ddth1; ddth2];

u = [tau1; tau2];
p = [c1; l1; c2; l2; m1; I1; m2; I2; g];

%% Unit Vectors
ihat = [1; 0; 0];
jhat = [0; 1; 0];
khat = [0; 0; 1];

ahat = sin(th1)*ihat - cos(th1)*jhat;
bhat = sin(th1+th2)*ihat - cos(th1+th2)*jhat;

%% Kinematics
Rc1 = [c1*sin(th1); -c1*cos(th1); 0];                                          % COM1 Position
Rc2 = [l1*sin(th1) + c2*sin(th1+th2); -l1*cos(th1)-c2*cos(th1+th2); 0];     % COM2 Position
R1 = l1*ahat;        % endpoint1 position
R2 = R1 + l2*bhat;   % endpoint2 position

ddt = @(r) jacobian(r, [q; dq])*[dq; ddq];

v1 = ddt(Rc1);
v2 = ddt(Rc2);  % COM velocities

%% Kinetic and Potential engergy
T1 = 1/2*m1*dot(v1, v1) + 1/2*I1*(dth1)^2;
T2 = 1/2*m2*dot(v2, v2) + 1/2*I2*(dth1+dth2)^2;
T = T1 + T2;                         % Total kinetic

V1 = m1*g*dot(Rc1, -(-jhat));
V2 = m2*g*dot(Rc2, -(-jhat));
V = V1 + V2;                         % Total potential

%% Generalized Force
Q = [tau1; tau2];

%% Lagrange equation
L = T - V;
```

```matlab
g = ddt(jacobian(L,dq).') - jacobian(L,q).' - Q;

A = simplify(jacobian(g,ddq));
b = simplify(A*ddq - g);

gravity = simplify(jacobian(V,q)).';
coriolis = simplify(-b - gravity + Q);

z = [q; dq];
dz = [dq; ddq];

rA = [l1*sin(th1); -l1*cos(th1)];
rB = [l1*sin(th1) + l2*sin(th1+th2); -l1*cos(th1) - l2*cos(th1+th2)];
keypoints = [rA rB];

J_B = jacobian(rB, q);
J_B_dot = reshape(ddt(J_B(:)), size(J_B));
vB = J_B*dq;

%% Save files
matlabFunction(A, 'file', 'A_pend', 'var', {z p});
matlabFunction(b, 'file', 'b_pend', 'var', {z u p});
matlabFunction(keypoints, 'file','keypoints_pend', 'var',{z p});
matlabFunction(J_B, 'file', 'Jacobian_rB','var',{z p});
matlabFunction(vB, 'file', 'velocity_rB','var',{z p});
matlabFunction(gravity, 'file','grav_pend', 'var', {z p});
matlabFunction(coriolis, 'file','coriolis_pend', 'var', {z p});
matlabFunction(J_B_dot, 'file', 'Jdot_rB', 'var',{z p});
```

```
function u = controller(z, param, x_des, dx_des, ddx_des)
   % ******** Implement your controller ********
   keypoints = keypoints_pend(z, param);
   rB = keypoints(:,2);
   err_position = x_des - rB;

   vB = velocity_rB(z, param);
   err_velocity = dx_des - vB;

   J_B = Jacobian_rB(z, param);

   Kp = 50;
   Kd = 5;
%     command = (ddx_des + Kp*err_position + Kd*err_velocity); % command for Lambda
   command = (Kp*err_position + Kd*err_velocity); % for last ctrl law

   % Oscillation
   dim = length(z);
   M = A_pend(z, param);
   z_vel_zero = z;
   z_vel_zero(dim/2+1:end) = zeros(dim/2, 1);
   u_zero = zeros(size(command));
   grav = -b_pend(z_vel_zero, u_zero, param);
   coriolis = -b_pend(z, u_zero, param) - grav;
   Jdot = Jdot_rB(z, param);

   Lambda_inv = J_B*inv(M)*J_B.';
   Lambda = inv(Lambda_inv);
   mu = Lambda*J_B*inv(M)*coriolis - Lambda*Jdot*z(dim/2+1:end);
   rho = Lambda*J_B*inv(M)*grav;

   %% Force commands
   F = Lambda*command + mu + rho;
%     F = Lambda*command + mu;
%     F = Lambda*command + rho;

   u = J_B.'*F;
end
```