```
function zdot = manipulator(t, z_0,B_f,N_f)
theta_1 = z_0(1);
theta_2 = z_0(2);
theta_3 = z_0(3);
theta_dot_1 = z_0(4);
theta_dot_2 = z_0(5);
theta_dot_3 = z_0(6);

params;

% Inertia matrix
%mat = load('Mass_matrix.mat');
var = [I_l1, I_l2, I_l3, I_m1, I_m2, a_1, a_2, kr_1, kr_2, l_1, l_2, l_3, m_l1, m_l2, m
B = subs(B_f,symvar(B_f),var);
%B = diag(diag(B));
% move sym functions

% Nonlinear term
%mat = load('N_term.mat');
var = [a_1, a_2, g, l_1, l_2, l_3, m_l1, m_l2, m_l3, m_m2, theta_1, theta_2, theta_3, t
N = subs(N_f,symvar(N_f),var);

B_f = double(vpa(B,3));
N_f = double(vpa(N,3));

% Inertia matrix
b11 = I_l1 + m_l1*l_1^2+kr_1^2*I_m1+I_l2+m_l2*(a_1^2+l_2^2+2*a_1*l_2*cos(theta_2)) + I_
b12 = I_l2+m_l2*(l_2^2+a_1*l_2*cos(theta_2))+kr_2*I_m2;
b21 = b12;
b22 = I_l2+ m_l2*l_2^2+kr_2^2*I_m2;

% Christoffel symbols
c111 = 0;
c112 = -m_l2*a_1*l_2*sin(theta_2);
c122 = c112;
c211 = -c112;
c212 = 0;
c222 = 0;
h = c112;

% Gravity terms
g11 = (m_l1*l_1+m_m2*a_1+m_l2*a_1)*g*cos(theta_1) + m_l2*l_2*g*cos(theta_1+theta_2);
g22 = m_l2*l_2*g*cos(theta_1+theta_2);

% Assemble manipulator dynamics
B_a = [b11 b12; b21 b22];
C_a = [h*theta_dot_2 h*(theta_dot_1+theta_dot_2); -h*theta_dot_1 0];
%N_a = [2*h*theta_dot_2 h*theta_dot_2; h*theta_dot_2 0]-2*[h*theta_dot_2 h*(theta_dot_1
G_a = [g11; g22];

% Jacobian Analytical
j11 = -a_2*sin(theta_1+theta_2)-a_1*sin(theta_1);
j12 = -a_2*sin(theta_1+theta_2);
```

```
j21 = a_2*cos(theta_1+theta_2)+a_1*cos(theta_1);
j22 = a_2*cos(theta_1+theta_2);
Ja = [j11 j12; j21 j22];

% Derivative of Jacobian Analytical
s1 = a_2*sin(theta_1+theta_2)*(theta_dot_1+theta_dot_2);
s2 = a_2*cos(theta_1+theta_2)*(theta_dot_1+theta_dot_2);
jd11 = -s2-a_1*cos(theta_1)*theta_dot_1;
jd12 = -s2;
jd21 = -s1-a_1*sin(theta_1)*theta_dot_1;
jd22 = -s1;
Ja_dot = [jd11 jd12; jd21 jd22];

% Define joint variable
q = [theta_1 theta_2 theta_3]';
q_arm = [theta_1 theta_2]';
q_dot = [theta_dot_1 theta_dot_2 theta_dot_3]';
q_dot_arm = [theta_dot_1 theta_dot_2]';

phi = wrapToPi((theta_1 + theta_2 + theta_3) - pi/2); % Pendulum measured from pos y
phi_dot = theta_dot_3;

% Get desired variables
Xd = [0; 0.75];
Xd_dot = [0;0];
Xd_ddot = [0;0];

% Trajectory from 4 to 7
if t > 4.0
    Xi = [0.0; 0.75];
    Xf = Xi + [0.250; 0.250];
    Vi = [0;0];
    Vf = [0;0];
    [x v a] = ContinousCPTraj(Xi,Xf,Vi,Vf,4,3,t);

    Xd = x;
    Xd_dot = v;
    Xd_ddot = a;
end



%if t > 3
%    Xd =
%    Xd_dot =
%    Xd_ddot =
%end


% Define operational space variables
Xe = fk(q_arm);
Xe_dot = Ja*q_dot_arm;

if abs(phi) < 0.3 | t > 0.85
```

```
    SWING_UP = false;
else
    SWING_UP = true;
end

%SWING_UP = true;

if SWING_UP == true
    % Get trajectory values
%     Xi = [-1.5; 0.75];
%     Xf = Xi + [3.0; 0.0];
%     Vi = [0;0];
%     Vf = [6.0;0];
%     [x,v,a] = ContinousCPTraj(Xi,Xf,Vi,Vf,1.5,t);
    [x,v,a] = SinusoidalSwingUp(0.485,0.5,t);
    Xd(1) = x(1);
    Xd_dot(1) = v(1);
    Xd_ddot(1) = a(1);
end

if SWING_UP == false
    % Pendulum control Full State Feedback
    K_pend = [-100 -200 500 100];
    R_pend = [Xd(1) Xd_dot(1) 0 0]';
    X_pend = [Xe(1) Xe_dot(1) phi phi_dot]';
    U_pend = Xd(1) - K_pend*(X_pend);
    %Xd_dot(1) = U_pend;
    Xd_ddot(1) = U_pend;
    Kd(1,1) = Kd(1,1)*0; % kill manipulator velocity control
    Kp(1,1) = Kp(1,1)*0; % kill manipulator position control
end

% Calculate manipulator error
X_tilda = Xd-Xe;
X_tilda_dot = Xd_dot-Xe_dot;

% Inverse dynamics operational space controller
y_arm = Ja\(Xd_ddot+Kd*X_tilda_dot+Kp*X_tilda-Ja_dot*q_dot_arm);
y = [y_arm(1) y_arm(2) 0]';
% Inverse dynamics linearizing control
% Controller compensating only arm
%u = B_a*y_arm + C_a*q_dot_arm + G_a;
% Controller compensating full
u = B_f*y + N_f;

%u = 0;
u = [u(1) u(2) 0]';
if t > 10
    u = [0 0 0]';
end

% Full Model
RHS =-N_f -Fv*q_dot + u;
LHS = B_f;
```

```
q_ddot = double(LHS\RHS);
theta_ddot_1 = q_ddot(1);
theta_ddot_2 = q_ddot(2);
theta_ddot_3 = q_ddot(3);

Xd
Xd_dot
Xd_ddot
phi
theta_dot_3
t
SWING_UP

zdot = [theta_dot_1; theta_dot_2; theta_dot_3; theta_ddot_1; theta_ddot_2; theta_ddot_3
end
```