```
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
```

```
close all
clear all
clc
```

The goal of this script is to derive the dynamic equations for a 3-link manipulator with only the two first links being actuated, this means the system has 3 links with mass and inertia, but only 2 motors with mass, inertia, and gear reductions

```
% Link masses
m_l1 = 1.0;
m_l2 = 1.0;
m_l3 = 1.0;

% Link length
a_1 = 1;
a_2 = 1;
a_3 = 1;

% Motor masses
m_m1 = 0.25;
m_m2 = 0.25;

% Gear reductions
kr_1 = 1;
kr_2 = 1;

% Distance to center of mass
l_1 = a_1/2;
l_2 = a_2/2;
l_3 = a_3/2;

% Link inertia
I_l1 = (1/12)*m_l1*l_1^2;
I_l2 = (1/12)*m_l2*l_2^2;
I_l3 = (1/12)*m_l3*l_3^2;

% Motor ineria
I_m1 = 0.1;
I_m2 = 0.1;

% Enviornment parameters
%g = 9.81; % m/s^2
```

```
syms g real
syms theta_1 real
syms theta_2 real
syms theta_3 real
syms theta_dot_1 real
syms theta_dot_2 real
syms theta_dot_3 real
syms m_l1 m_l2 m_l3 m_m1 m_m2 real
syms l_1 l_2 l_3 I_l1 I_l2 I_l3 I_m1 I_m2 real
syms a_1 a_2 a_3 real
syms kr_1 kr_2 real
syms g real

%syms m_l1 m_l2 m_l3 m_m1 m_m2 l_1 l_2 l_3 I_l1 I_l2 I_l3 I_m1 I_m2 a_1 a_2 a_3 kr_1 kr

% theta_dot_1 = diff(theta_1,t);
% theta_dot_2 = diff(theta_2,t);
% theta_dot_3 = diff(theta_3,t);

%assume(theta_dot_1,'real');
%assume(theta_dot_2,'real');
%assume(theta_dot_3,'real');

T0_1 = dhTransform(0,0,theta_1,0);
T1_2 = dhTransform(0,a_1,theta_2,0);
T2_3 = dhTransform(0,a_2,theta_3,0);
T3_E = dhTransform(0,a_3,0,0);

T0_E = formula(T0_1*T1_2*T2_3*T3_E);
T0_3 = formula(T0_1*T1_2*T2_3);
T0_2 = formula(T0_1*T1_2);
T0_1
T0_1 = T0_1
R0_1 = T0_1(1:3,1:3);
R0_2 = T0_2(1:3,1:3);
R0_3 = T0_3(1:3,1:3);

% Define Positions along arm
p_l1_s = T0_1*[l_1 0 0 1]'
p_l2_s = T0_2*[l_2 0 0 1]'
p_l3_s = T0_3*[l_3 0 0 1]'

p_m1_s = T0_1*[0 0 0 1]';
p_m2_s = T0_2*[0 0 0 1]';

pl1 = p_l1_s(1:3);
pl2 = p_l2_s(1:3);
pl3 = p_l3_s(1:3);

pm1 = p_m1_s(1:3);
pm2 = p_m2_s(1:3);

% Define Positions to Joints
p0 = [0 0 0]';
p1 = T0_2(1:3,4);
p2 = T0_3(1:3,4);
p3 = T0_E(1:3,4);
```

```
Z0  = [0 0 1]';
Zeros = [0 0 0]';

Jl1_P = [cross(Z0,pl1-p0), Zeros, Zeros];
Jl1_O = [Z0, Zeros, Zeros];
Jl1 = [Jl1_P; Jl1_O]

Jl2_P = [cross(Z0,pl2-p0),cross(Z0,pl2-p1),Zeros];
Jl2_O = [Z0, Z0, Zeros];
Jl2 = [Jl2_P; Jl2_O]

Jl3_P = [cross(Z0,pl3-p0),cross(Z0,pl3-p1),cross(Z0,pl3-p2)];
Jl3_O = [Z0,Z0,Z0];
Jl3 = [Jl3_P; Jl3_O]

Jm1_P = [Zeros, Zeros, Zeros];
Jm1_O = [Z0*kr_1, Zeros, Zeros];
Jm1 = [Jm1_P; Jm1_O]

Jm2_P = [cross(Z0,pm2-p0),Zeros, Zeros];
Jm2_O = [Z0, Z0*kr_2, Zeros];
Jm2 = [Jm2_P; Jm2_O]

q_dot = [theta_dot_1 theta_dot_2 theta_dot_3]';

% Velocities of link center of masses
P_dot_l1 = Jl1_P*q_dot;
P_dot_l2 = Jl2_P*q_dot;
P_dot_l3 = Jl3_P*q_dot;

% Angular Velocities of links
O_dot_l1 = Jl1_O*q_dot;
O_dot_l2 = Jl2_O*q_dot;
O_dot_l3 = Jl3_O*q_dot;

% Velocities of the motors
P_dot_m1 = Jm1_P*q_dot;
P_dot_m2 = Jm2_P*q_dot;

% Angular velocities of the motors
O_dot_m1 = kr_1*theta_dot_1*Z0;
O_dot_m2 = kr_2*theta_dot_2*Z0;

% Kinetic Energy of the links
T_li(1) = 1/2*m_l1*(P_dot_l1')*P_dot_l1+1/2*(O_dot_l1')*R0_1*I_l1*R0_1'*O_dot_l1;
T_li(2) = 1/2*m_l2*(P_dot_l2')*P_dot_l2+1/2*(O_dot_l2')*R0_2*I_l2*R0_2'*O_dot_l2;
T_li(3) = 1/2*m_l3*(P_dot_l3')*P_dot_l3+1/2*(O_dot_l3')*R0_3*I_l3*R0_3'*O_dot_l3;

% Kinetic Energy of the motors
T_mi(1) = 1/2*m_m1*(P_dot_m1')*P_dot_m1+1/2*(O_dot_m1')*I_m1*O_dot_m1;
T_mi(2) = 1/2*m_m2*(P_dot_m2')*P_dot_m2+1/2*(O_dot_m2')*I_m2*O_dot_m2;

Ti(1) = T_li(1)+T_mi(1);
Ti(2) = T_li(2)+T_mi(2);
```

```matlab
Ti(3) = T_li(3);
simplify(Ti');

G = [0 -g 0]';
% Potential Energy of links
Ui(1) = -(m_m1*G'*pm1 + m_l1*G'*pl1);
Ui(2) = -(m_m2*G'*pm2 + m_l2*G'*pl2);
Ui(3) = -(m_l3*G'*pl3);
simplify(Ui');
U = Ui(1) + Ui(2) + Ui(3);
T = Ti(1) + Ti(2) + Ti(3);
```

```matlab
X = {theta_1 theta_dot_1 theta_2 theta_dot_2 theta_3 theta_dot_3}
L=T-U;
syms tau1 tau2 tau3 real
Qe = {tau1 tau2 tau3};
Qi = {0 0 0};
par = {m_l1 m_l2 m_l3 m_m1 m_m2 l_1 l_2 l_3 I_l1 I_l2 I_l3 I_m1 I_m2 a_1 a_2 a_3 kr_1 k
Eqn = EulerLagrange(L,X,Qi,Qe,0,par,'m')
```

```matlab
syms theta_ddot_1 theta_ddot_2 theta_ddot_3 real
tau = [tau1; tau2; tau3];
Eq =[Eqn(2) == theta_ddot_1
     Eqn(4) == theta_ddot_2
     Eqn(6) == theta_ddot_3];
```

```matlab
%[,A] = collect(Eq,{theta_ddot_1 theta_ddot_2 theta_ddot_3})
q_ddot = [theta_ddot_1 theta_ddot_2 theta_ddot_3]';
[W,E] = equationsToMatrix(Eq,[tau1 tau2 tau3])
lhs = simplify(W\E);
rhs = [tau1 tau2 tau3]';

[Mass_matrix,Other] = equationsToMatrix(lhs,[theta_ddot_1 theta_ddot_2 theta_ddot_3])
Mass_matrix
%[C_matrix,Else] = equationsToMatrix(Other,[theta_dot_1 theta_dot_2 theta_dot_3])
N_matrix = simplify(-Other)
N_iso = [N_matrix(1) == 0
         N_matrix(2) == 0
         N_matrix(3) == 0];
%out = equationsToMatrix(N_iso,[theta_dot_1 theta_dot_2 theta_dot_3])

EOM_rough = lhs;
EOM_clean = Mass_matrix*q_ddot + N_matrix ;
isequal(EOM_rough(1),EOM_clean(1))
simplify(EOM_rough,"Steps",100)
simplify(EOM_clean,"Steps",100)
save('Mass_matrix.mat','Mass_matrix')
save('N_term.mat','N_matrix')
```

```matlab
 % Assemble B Matrix
% Shortcut derivation of the mass matrix
```

```
B1 = m_l1*(Jl1_P')*Jl1_P+Jl1_O'*R0_1*I_l1*R0_1'*Jl1_O + m_m1*(Jm1_P')*Jm1_P+Jm1_O'*R0_1
B2 = m_l2*(Jl2_P')*Jl2_P+Jl2_O'*R0_2*I_l2*R0_2'*Jl2_O + m_m2*(Jm2_P')*Jm2_P+Jm2_O'*R0_2
B3 = m_l3*(Jl3_P')*Jl3_P+Jl3_O'*R0_3*I_l3*R0_3'*Jl3_O;
B = B1 + B2 + B3;
```

```
close all
clear all
clc

tic
params;

% Initial conditions
disturbance = -0.0;
qi = ik(0,0.75)
theta_1_0 = qi(1);
theta_2_0 = qi(2);
theta_3_0 = wrapToPi(-(qi(1)+qi(2)+pi/2+disturbance));
theta_dot_1_0 = 0;
theta_dot_2_0 = 0;
theta_dot_3_0 = 0;

tspan = [0 15];

% ODE45 settings
options = odeset('RelTol',1e-3,'AbsTol',1e-5);
%options = odeset('reltol', 1E-7);
z_0 = [theta_1_0; theta_2_0; theta_3_0; theta_dot_1_0; theta_dot_2_0; theta_dot_3_0];

B_mat = load('Mass_matrix.mat');
B_f = B_mat.Mass_matrix;
N_mat = load('N_term.mat');
N_f = N_mat.N_matrix;

[time, z] = ode23tb(@(t,y) manipulator(t,y,B_f,N_f), tspan, z_0, options);

toc
```

Get varibles from simulation

```
theta_1 = z(:,1);
theta_2 = z(:,2);
theta_3 = z(:,3);

q = [theta_1 theta_2 theta_3]';
phi = wrapToPi(-q(1,:)-q(2,:)-q(3,:)+pi/2);
for i = 1:length(time)
    Xe(:,i) = fk(q(1:2,i));
end

hold on
plot(time,theta_1)
plot(time,theta_2)
plot(time,theta_3)
hold off

figure
```

```
subplot(3,1,1)
plot(time,Xe(1,:))
title("Endeffector position step response (Balance Controller)")
ylabel("Position X (m)")
xlabel("Time (sec)")

subplot(3,1,2)
plot(time,Xe(2,:))
ylabel("Position Y (m)")
xlabel("Time (sec)")
%axis([0 tspan(1,2) 0.71 0.79])

subplot(3,1,3)
plot(time,phi)
ylabel("Angle Phi (rad)")
xlabel("Time (sec)")

% Calculating link endpoints
L1_x = a_1*cos(theta_1);
L1_y = a_1*sin(theta_1);
L2_x = L1_x + a_2*cos(theta_1+theta_2);
L2_y = L1_y + a_2*sin(theta_1+theta_2);
L3_x = L2_x + a_3*cos(theta_1+theta_2+theta_3);
L3_y = L2_y + a_3*sin(theta_1+theta_2+theta_3);

[L1_x ts] = resample(L1_x,time);
[L1_y ts] = resample(L1_y,time);
[L2_x ts] = resample(L2_x,time);
[L2_y ts] = resample(L2_y,time);
[L3_x ts] = resample(L3_x,time);
[L3_y ts] = resample(L3_y,time);

figure()
axis([-1.75 1.75 -1 2.5]);
xlabel('x');
ylabel('y');

L1_line = line('xdata', [0, L1_x(1)], 'ydata', [0, L1_y(1)],...
    'linewidth', 3, 'color', 'blue');
L2_line = line('xdata', [L1_x(1), L2_x(1)], 'ydata', [L1_y(1), L2_y(1)],...
    'linewidth', 3, 'color', 'red');
L3_line = line('xdata', [L2_x(1), L3_x(1)], 'ydata', [L2_y(1), L3_y(1)],...
    'linewidth', 3, 'color', 'green');
```

```
delay = tspan(1,2)/length(ts);
hold on
%yline(0.75);
%xline(0);
grid on
v = VideoWriter('Output.avi');
open(v);
% Draw and redraw the objects with new xdata, ydata
for i=1:length(ts)
```

```
    pause(delay)
    set(L1_line, 'xdata', [      0, L1_x(i)], 'ydata', [      0, L1_y(i)]);
    set(L2_line, 'xdata', [L1_x(i), L2_x(i)], 'ydata', [L1_y(i), L2_y(i)]);
    set(L3_line, 'xdata', [L2_x(i), L3_x(i)], 'ydata', [L2_y(i), L3_y(i)]);
    frame = getframe(gcf);
    writeVideo(v,frame);
    drawnow;
end

hold off
close(v)
```