Robot manipulator: Part 4

Instructor: Prof. Patel

Peyman Yadmellat

1. Inverse dynamic

Purpose of this part is to find the torque value of each link according to the given trajectory of manipulator's links.

As given in the book, regarding the mechanical type of joints, two sets of formulas can be used to compute the torque applied by each link on neighbor of that link:

Outward recursions: $i: 0 \rightarrow (N-1)$

$$\begin{cases} {}^{i+1}\omega_{i+1} = {}^{i+1}_{i}R \ {}^{i}\omega_i + \dot{\theta}_{i+1} {}^{i+1}\hat{Z}_{i+1} & (Revolute\ joints) \\ {}^{i+1}\omega_{i+1} = {}^{i+1}_{i}R \ {}^{i}\omega_i & (Prismatic\ joints) \end{cases}$$

$$\begin{cases} {}^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R \ {}^{i}\dot{\omega}_i + {}^{i+1}_{i}R \ {}^{i}\omega_i \times \dot{\theta}_{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1} {}^{i+1}\hat{Z}_{i+1} & (Revolute\ joints) \\ {}^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R \ {}^{i}\dot{\omega}_i & (Prismatic\ joints) \end{cases}$$

$$\begin{cases} {}^{i+1}v_{i+1} = {}^{i+1}_{i}R \left({}^{i}v_i + {}^{i}\omega_i \times {}^{i}P_{i+1} \right) & (Revolute\ joints) \\ {}^{i+1}v_{i+1} = {}^{i+1}_{i}R \left({}^{i}v_i + {}^{i}\omega_i \times {}^{i}P_{i+1} \right) + \dot{d}_{i+1} {}^{i+1}\hat{Z}_{i+1} & (Prismatic\ joints) \end{cases}$$

$$\begin{cases} {}^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R \left[{}^{i}\dot{v}_i + {}^{i}\dot{\omega}_i \times {}^{i}P_{i+1} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{i+1} \right) \right] & (Revolute\ joints) \\ {}^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R \left[{}^{i}\dot{v}_i + {}^{i}\dot{\omega}_i \times {}^{i}P_{i+1} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{i+1} \right) \right] & \\ \quad + 2 {}^{i+1}\omega_{i+1} \times \dot{d}_{i+1} {}^{i+1}\hat{Z}_{i+1} + \ddot{d}_{i+1} {}^{i+1}\hat{Z}_{i+1} & (Prismatic\ joints) \end{cases}$$

$${}^{i+1}\dot{v}_{C_{i+1}} = {}^{i}\dot{v}_i + {}^{i}\dot{\omega}_i \times {}^{i}P_{C_i} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{C_i} \right)$$

$${}^{i+1}F_{i+1} = m_{i+1} {}^{i+1}\dot{v}_{C_{i+1}}$$

$${}^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1} {}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C_{i+1}}I_{i+1} {}^{i+1}\omega_{i+1}$$

Inward recursions: $i: N \rightarrow 1$

$${}^{i}f_i = {}^{i}_{i+1}R \, {}^{i+1}f_{i+1} + {}^{i}F_i$$

$$^{i}n_i = {}^{i}N_i + {}^{i}_{i+1}R\,{}^{i+1}n_{i+1} + {}^{i}P_{C_i} \times {}^{i}F_i + {}^{i}P_{i+1} \times {}^{i}_{i+1}R\,{}^{i+1}f_{i+1}$$

$$\tau_i = {}^{i}n_i^T\ {}^{i}\hat{Z}_i$$

where N is the number of joints. Also, initial and/or final conditions have been considered as ${}^{0}\omega_0 = {}^{0}\dot{\omega}_0 = \mathbf{0}$, ${}^{0}\dot{v}_{0=}G$, ${}^{N+1}f_{N+1} = \mathbf{0}$ and ${}^{N+1}n_{N+1} = \mathbf{0}$. Notice that definition of all undefined terms are as the same as definition given in the book.

Table 1. D-H parameters of the Stanford manipulator

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0°	0	h = 0.4 m	$\theta_1(t)^{\circ}$
2	-90°	0	r = 0.1 m	$\theta_2(t)^{\circ}$
3	90°	0	$f(t)$	0°

Now, by considering time-varying D-H parameter given in Table 1 for the Stanford manipulator, we can obtain following numerical results for the joint positions, velocities, accelerations and torques against time as shown in Figures 1-4.

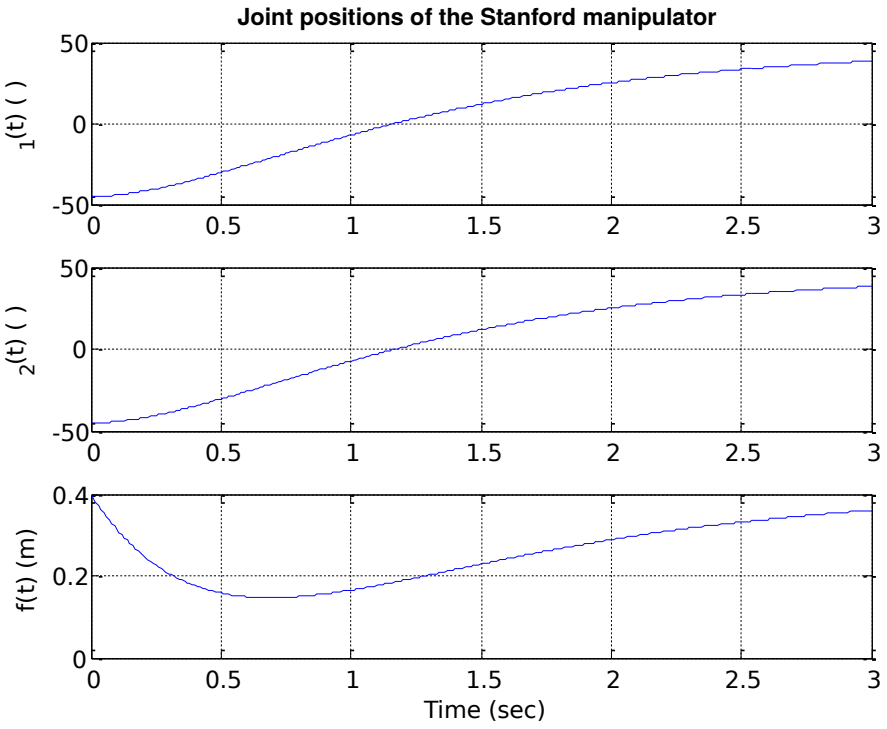

Figure 1- Joint positions of each joint of the Stanford manipulator against time.

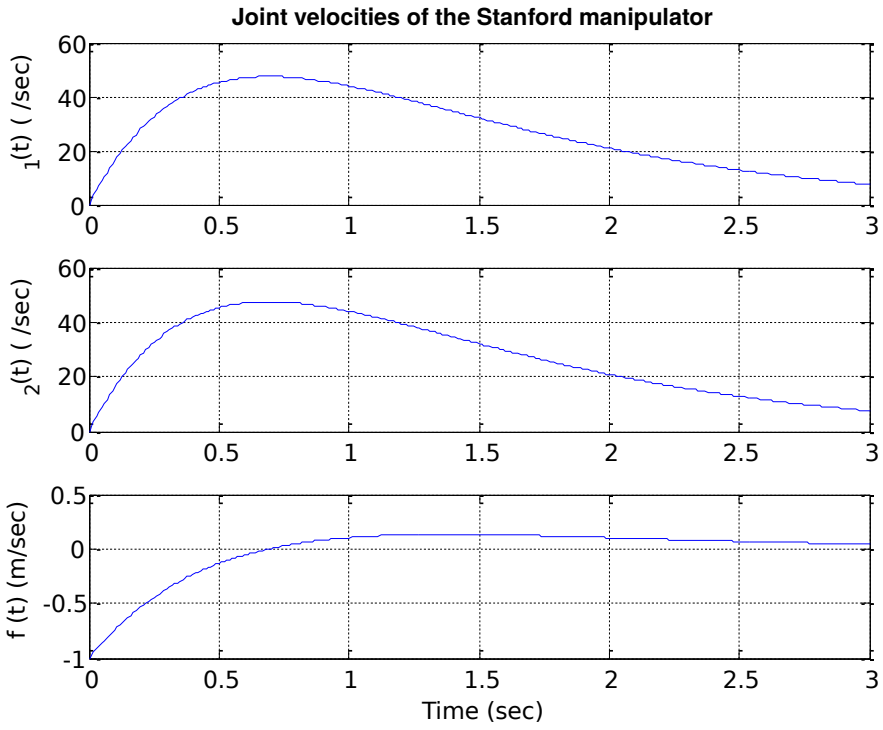

Figure 2- Joint velocities of each joint of the Stanford manipulator against time.

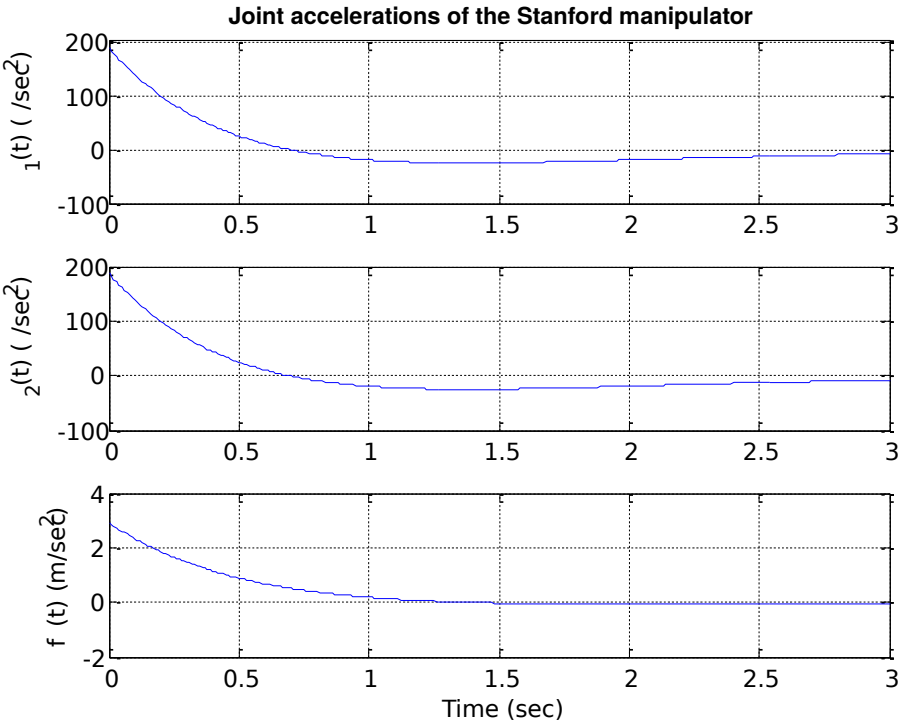

Figure 3- Joint acceleration of each joint of the Stanford manipulator against time.

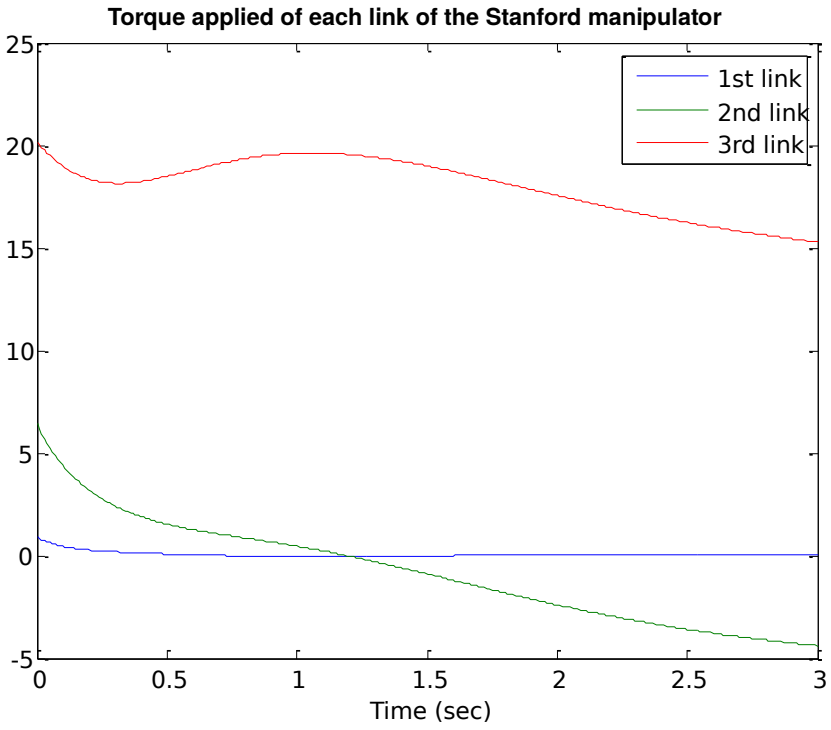

Figure 4- Torque applied by each link of the Stanford manipulator against time.

2. Forward dynamic (Simulation)

In this part, the problem is to calculate the trajectory of links motion according to the given torque. By symbolically computing Newton-Euler equations, following dynamic equation can be achieved:

$$\tau = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta) .$$

Regarding the assumption and configuration of the Stanford manipulator, mass matrix ($M(\theta)$), centrifugal and Coriolis terms ($V(\theta, \dot{\theta})$), and gravity terms ($G(\theta)$) can be obtained as follow

$$M(\boldsymbol{\theta}) = \begin{bmatrix} r^2 m_2 + m_3 f^2 (1 - c_2^2) + r^2 m_3 & -r m_3 c_2 f & -r m_3 s_2 f \\ -r m_3 c_2 f & m_3 f^2 & 0 \\ -r m_3 s_2 f & 0 & m_3 \end{bmatrix}$$

$$V(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \begin{bmatrix} f^2 m_3 \dot{\theta}_1 \dot{\theta}_2 sin(2\theta_2) + r m_3 f \dot{\theta}_2^2 s_2 - 2 r m_3 \dot{\theta}_2 \dot{f} c_2 + f m_3 \dot{\theta}_1 \dot{f} - f m_3 \dot{\theta}_1 \dot{f} cos(2\theta_2) \\ -m_3 f \left(f c_2 s_2 \dot{\theta}_1^2 - 2 \dot{\theta}_2 \dot{f} \right) \\ m_3 f \left(\dot{\theta}_1^2 c_2^2 - \dot{\theta}_1^2 - \dot{\theta}_2^2 \right) \end{bmatrix}$$

$$G(\boldsymbol{\theta}) = [0 \quad -9.8m_3s_2f \quad 9.8m_3c_2]^T$$

Now, we can solve following equation to simulate the motion of a manipulator by computing joint accelerations

$$\ddot{\theta} = M^{-1}(\theta)\left[\tau - +V\left(\theta, \dot{\theta}\right) + G(\theta)\right]$$

Subsequently, we can compute joint velocities and positions by using any numerical integration methods such as Euler integration as given below

$$\dot{\Theta}(t + \Delta t) = \dot{\Theta}(t) + \ddot{\Theta}\Delta t$$

$$\Theta(t + \Delta t) = \Theta(t) + \dot{\Theta}(t)\Delta t + \frac{1}{2}\ddot{\Theta}\Delta t^2$$

where Δt should be sufficiently small to achieve desired accuracy.

Applying obtained torque form inverse dynamic stage, following simulation results have been achieved through forward dynamic and also using Euler integration method with step time=0.001.

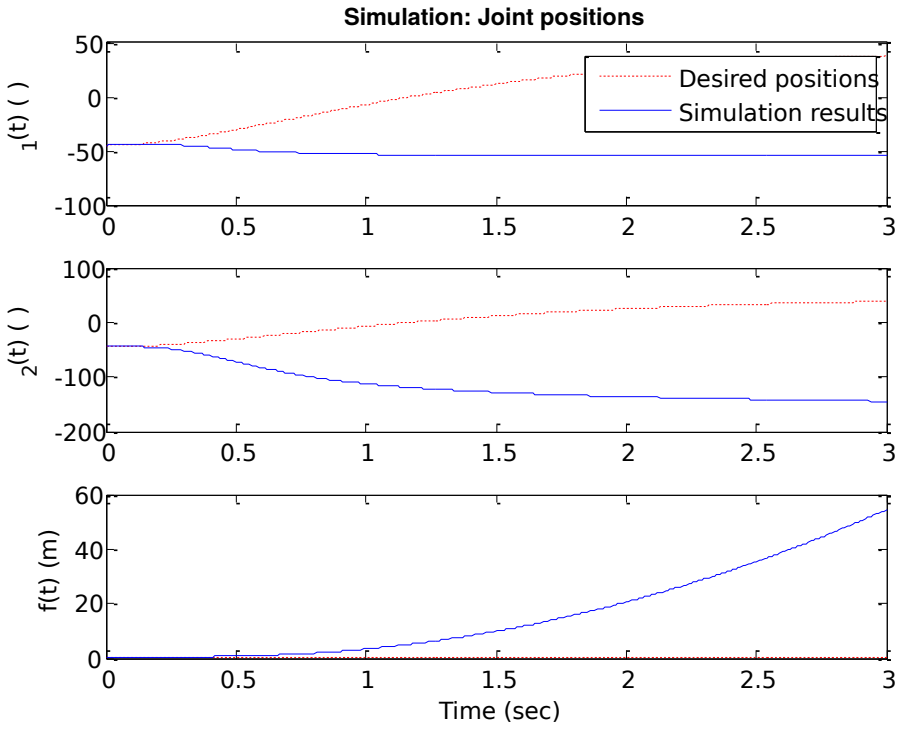

Figure 5- Simulation results for joint positions in comparison with desired joint position.

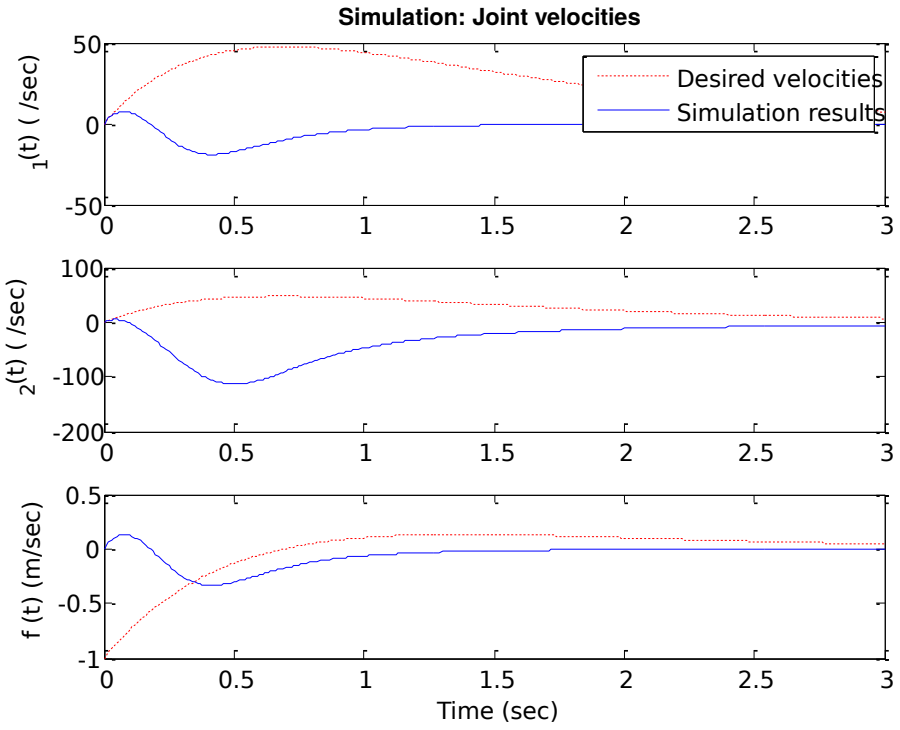

Figure 6- Simulation results for joint velocities in comparison with desired joint velocities.

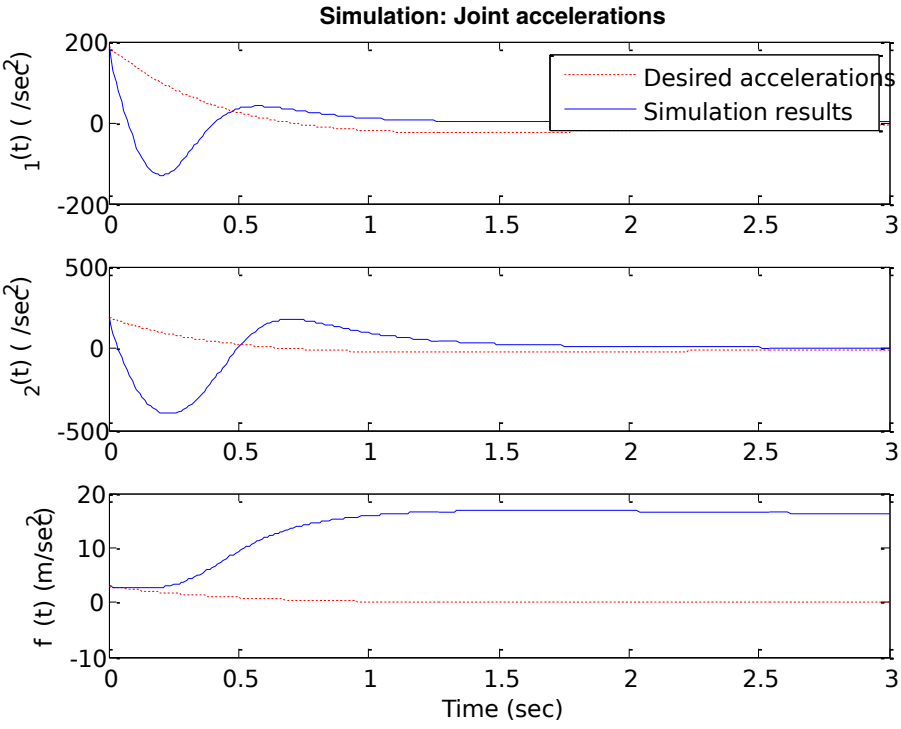

Figure 7- Simulation results for joint acceleration in comparison with desired joint acceleration.

2.1 Discussion

It's obvious that obtaining results from Forward dynamic part diverge from desired joint trajectory motion used in inverse dynamic stage. Two reasons may exist for such a response; (i) the mass matrix approaches its singularity, (ii) lack of integration accuracy. Several approaches were taken in order to avoid the former problem, namely scaling, constant addition, and modifying the position of the center of the mass (subsequently the DH parameter). None of these remedied the problem of singularity. In addition, filtering, use of higher order terms in Euler integration method, decreasing the step time, and "Adam-Bashford" integration technique were used in order to improve the accuracy of integration. Yet, there was no significant difference between these results and that of the Euler integration method. The main issue here might be that there is no comparison (e.g., feedback) between the outcome of the simulation stage and the desired response. As a result, the system cannot recover from the error caused by the numerical computations, round-off error, and integration inaccuracy.

3. Matlab functions

Four Matlab functions together with one Matlab program have been written to compute the torque applied by each link to its neighbor by using inverse dynamic recursions and also to compute joint positions, velocities and accelerations by applying obtained torque profile for the Stanford manipulator.

A main Matlab function is "InvDyn.m" which computes the torque according to joint positions, velocities and accelerations, D-H parameters, and configuration of manipulator. This program has been written for general case, and it can apply to compute torque of other type of manipulators.

Other three Matlab functions entitled "StanfordMass.m", "StanfordVelocity.m", and "StanfordGravity.m" have been specifically written to compute, respectively, mass matrix, velocity vector, and gravity vector of the Stanford manipulator dynamic equation. Each function returns related terms according to input arguments. The input arguments for "StanfordMass" and "StanfordGravity" are joint positions, and for "StanfordVelocity" are joint positions together with joint velocities.

In addition, a Matlab program entitled "Test_Stanford_Dynamic.m" has been written to obtain and to plot the torque profile and also joint motion during the specified time, i.e. $0 \leq t \leq 3\ s$. this program uses Euler integration method in forward dynamic stage. Also another program entitled "Test_Stanford_Dynamic_AdamBashford.m" has been written which uses Adam-Bashford integration technique, but results are similar to Euler method.

Syntax of these functions are considered as follow

InvDyn:

```
Tau=InvDyn(JointVelocities,JointAcceleration,alpha,a,d,theta,...
     TypeOfJoints,m,I,Pc)
```

StanfordMass:

```
M=StanfordMass(NumTheta1,NumTheta2,NumF)
```

StanfordVelocity:

```
V=StanfordVelocity(NumTheta1,NumTheta2,NumF,NumDtheta1,...
     NumDtheta2,NumDf)
```

StanfordGravity:

```
G=StanfordGravity(NumTheta1,NumTheta2,NumF)
```

3.1. Input arguments

`JointVelocities`	is a column or row vector of joint velocities.
`JointAcceleration`	is a column or row vector of joint accelerations.
`alpha`	is a column vector of α_{i-1}, $i = 1,2,\cdots,n$ where n is the number of joints.
`a`	is a column vector of a_{i-1}, $i = 1,2,\cdots,n$ where n is the number of joints.
`d`	is a column vector of d_i, $i = 1,2,\cdots,n$ where n is the number of joints.
`theta`	is a column vector of θ_i, $i = 1,2,\cdots,n$ where n is the number of joints.
`TypeOfJoints`	is a string containing the mechanical type of each joint. Prismatic joint is represented by `'P'`, and `'R'` stands for revolute joint. Mechanical type of joints should be written in order. For example, `'RPRR'` (equivalently `['R','P','R','R']`) means that first joint is revolute, second one is prismatic, and the other two are revolute.
`m`	is a column vector of mass of each link.
`I`	is a matrix containing all inertia tensor matrices which they should place in one matrix in order, for example I=[I1,,I2,...In]
`Pc`	should contain all position of center of mass vectors in order, for example Pc=[Pc1,Pc2,...,Pcn]

NumTheta1	is a numerical value of first joint position
NumTheta2	is a numerical value of second joint position
NumF	is a numerical value of third joint position
NumDtheta1	is a numerical value of first joint acceleration
NumDtheta2	is a numerical value of second joint acceleration
NumDf	is a numerical value of third joint acceleration

3.2. Output arguments

Tau	returns the torque vector in Newton
M	returns the mass matrix of the Stanford manipulator
V	returns the velocity vector of the Stanford manipulator
G	returns the gravity vector of the Stanford manipulator

Appendix 1. Matlab code of "InvDyn" function:

```
function Tau=InvDyn(JointVelocities,JointAcceleration,alpha,a,d,theta,...
    TypeOfJoints,m,I,Pc)

%This function computes the torque applied by each link to its neighbor.
%-------------------------------------------------------------------------
% Input arguments:
%
%   JointVelocities is a column vector of velocity,
%                   it should be in rad/sec.
%
%   JointAcceleration is a column vector of joint acceleration
%                     it should be in rad/sec^2.
%
%   alpha,a,d,and theta are column vectors corresponding D-H parameters
%
%   TypeOfJoints is a string containing the mechanical type of each joint.
%   Prismatic joint is represented by 'P', and 'R' stands for revolute
%   joint. Mechanical type of joints should be written in order. For
%   example, 'RPRR' (equivalently ['R','P','R','R']) means that first
%   joint is revolute, second one is prismatic, and the other two are
%   revolute.
%
%   m is a column vector of mass of each link
%
%   I is a matrix containing all inertia tensor matrices which they should
%    place in one matrix in order, for example I=[I1,,I2,...In]
%
%   Pc should contain all position of center of mass vectors in order, for
%   example Pc=[Pc1,Pc2,...,Pcn]
%
%-------------------------------------------------------------------------
% Output arguments:
% Tau   returns the torque applied by each link to its neighbor.
```

```
%-------------------------------------------------------------------------
%
% Peyman Yadmellat

% April 04, 2010
% ------------------------------
% ------------------------------


TypeOfJoints=upper(TypeOfJoints); %Set to uppercase

numJoints=length(TypeOfJoints); %number of joints

G=[0;0;9.8];

%Initializing angular and linear velocities
v_new=[0;0;0];
w_new=zeros(3,1);

%Initializing angular and linear accelerations
dv_new=G;
dw_new=zeros(3,1);



% Computing the force and torque acting on each link, i.e. F adn N
% Outward recursions
for i=1:numJoints
    T=Forward_kin(alpha,a,d,theta,i-1,i);

    % Extracting the rotation matrix of joint i respect to its description
    % in frame {i-1}
    R=T(1:3,1:3).';

    % Extracting the translation matrix of joint i-1 respect to its
    % description in frame {i}
    P=T(1:3,4);

    % this set of formulas computes linear and angular velocities of each
    % joint
    v=v_new;
    w=w_new;
    v_new=R*(v+cross(w,P))+...
        isequal(TypeOfJoints(i),'P')*[0;0;JointVelocities(i)];
    w_new=R*w+isequal(TypeOfJoints(i),'R')*[0;0;JointVelocities(i)*pi/180];

    % this set of formulas computes linear and angular accelerations of
    % each joint
    dv=dv_new;
    dw=dw_new;
    dw_new=R*dw+isequal(TypeOfJoints(i),'R')*(cross(R*w,...
        [0;0;JointVelocities(i)*pi/180])+...
        [0;0;JointAcceleration(i)*pi/180]);
    dv_new=R*(dv+cross(dw,P)+cross(w,cross(w,P)))+...
```

```
        isequal(TypeOfJoints(i),'P')*(cross(2*w_new,...
        [0;0;JointVelocities(i)])+[0;0;JointAcceleration(i)]);

    %Computing linear acceleration of the center of mass of each link
    dvc=cross(w_new,Pc(1:numJoints,i))+...
        cross(w_new,w_new+Pc(1:numJoints,i))+dv_new;

    %Computing Force and torque acting on each link
    F(:,i)=m(i)*dvc;
    N(:,i)=I(1:numJoints,(i-1)*numJoints+1:i*numJoints)*dw_new+...
        cross(w_new,I(1:numJoints,(i-1)*numJoints+1:i*numJoints)*w_new);
end

% Inward recursions:

%initializing force and torque exerted on each link
f=zeros(3,1);
n=zeros(3,1);
R=eye(3);
P=zeros(3,1);
n=N(:,numJoints);
f=F(:,numJoints);
Tau(:,numJoints)=isequal(TypeOfJoints(i),'P')*f(3)+...
    isequal(TypeOfJoints(i),'R')*n(3);

% Computing the torque applied  by each link on its neighbor
for i=numJoints-1:-1:1
    T=Forward_kin(alpha,a,d,theta,i,i+1);
    R=T(1:3,1:3);
    P=T(1:3,4);
    n=N(:,i)+R*n+cross(Pc(1:numJoints,i),F(:,i))+cross(P,R*f);
    f=R*f+F(:,i);

Tau(:,i)=isequal(TypeOfJoints(i),'P')*f(3)+isequal(TypeOfJoints(i),'R')*n(3);

end
```

Appendix 2. Matlab code of “StanfordMass” function:

```
function M=StanfordMass(NumTheta1,NumTheta2,NumF)
% This function computes mass matrix of the Stanford manipulator given
% joint positions
%
%
%       Peyman Yadmellat

%       April 4, 2010
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------

syms t theta1 theta2 dtheta1 dtheta2 ddtheta1 ddtheta2 f df ddf
```

```
r=0.1;
m=[4;2;2];
M=subs([f^2*m(3)+m(2)*r^2-cos(theta2)^2*f^2*m(3)+m(3)*r^2,...
    -r*cos(theta2)*m(3)*f,-r*sin(theta2)*m(3)
    -f*m(3)*cos(theta2)*r, f^2*m(3),0
    m(3)*(-sin(theta2)*r),0,m(3)],{theta1 theta2 f},...
    {NumTheta1 NumTheta2 NumF});
```

Appendix 3. Matlab code of "StanfordVelocity" function:

```
function V=StanfordVelocity(NumTheta1,NumTheta2,NumF,NumDtheta1,...
    NumDtheta2,NumDf)
% This function computes velocity vector of the Stanford manipulator given
% joint positions and velocities
%
%
%       Peyman Yadmellat

%       April 4, 2010
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
syms t theta1 theta2 dtheta1 dtheta2 ddtheta1 ddtheta2 f df ddf

r=0.1;
m=[4;2;2];
m1=m(1);
m2=m(2);
m3=m(3);
V=subs([2*sin(theta2)*f^2*m3*cos(theta2)*dtheta1*dtheta2+...
    r*sin(theta2)*m3*dtheta2^2*f-2*r*cos(theta2)*m3*dtheta2*df+...
    2*f*m3*dtheta1*df-2*cos(theta2)^2*f*m3*dtheta1*df
    -1/5*f*m3*(5*cos(theta2)*dtheta1^2*sin(theta2)*f-10*dtheta2*df)
1/5*m3*(-5*dtheta1^2*f-5*dtheta2^2*f+5*dtheta1^2*f*cos(theta2)^2)],...
{theta1 theta2 f dtheta1 dtheta2 df},...
{NumTheta1 NumTheta2 NumF NumDtheta1 NumDtheta2 NumDf});
```

Appendix 4. Matlab code of "StanfordGravity" function:

```
function G=StanfordGravity(NumTheta1,NumTheta2,NumF)
% This function computes gravity vector of the Stanford manipulator given
% joint positions
%
%
%       Peyman Yadmellat

%       April 4, 2010
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------

syms t theta1 theta2 dtheta1 dtheta2 ddtheta1 ddtheta2 f df ddf
r=0.1;
m=[4;2;2];
m1=m(1);
m2=m(2);
m3=m(3);
```

```
G=subs([0
   -1/5*f*m3*(49*sin(theta2))
1/5*m3*(49*cos(theta2))],{theta1 theta2 f},{NumTheta1,NumTheta2,NumF});
```

Appendix 5. Matlab code of "Test_Stanford_Dynamic" program:

```
% This program can be used to get inverse dynamic and forward dynamic
% results for the Stanford manipulator
%
%
%       Peyman Yadmellat

%       April 4, 2010
%-----------------------------------------------------------------------------
%-----------------------------------------------------------------------------



clc
clear all
close all
% Initializing D-H parameters for the Stanford manipulator
m=[4;2;2];
alpha=[0;-90;90];
a=zeros(3,1);
% determining joint type of the Stanford manipulator
TypeOfJoints='rrp';
% setting the Stanford manipulator configuration including mass, inertia
% tensor, and position of center of mass
m=[4;2;2];
I1=zeros(3,3);
I2=zeros(3,3);
I3=zeros(3,3);
I=[I1 I2 I3];

Pc1=[0;0;0];
Pc2=[0;0;0];
Pc3=[0;0;0];
Pc=[Pc1 Pc2 Pc3];

% defining joint positions, velocities and accelerations
syms t theta1 theta2 dtheta1 dtheta2 ddtheta1 ddtheta2 f df ddf
theta1='45*(1+6*exp(-t/0.6)-8*exp(-t/0.8))';
theta2='45*(1+6*exp(-t/0.6)-8*exp(-t/0.8))';
f='0.4*(1+6*exp(-t/0.6)-6*exp(-t/0.8))';

dtheta1='45*(-6*exp(-t/0.6)/0.6+8*exp(-t/0.8)/0.8)';
dtheta2='45*(-6*exp(-t/0.6)/0.6+8*exp(-t/0.8)/0.8)';
df='0.4*(-6*exp(-t/0.6)/0.6+6*exp(-t/0.8)/0.8)';

ddtheta1='45*(6*exp(-t/0.6)/0.36-8*exp(-t/0.8)/0.64)';
ddtheta2='45*(6*exp(-t/0.6)/0.36-8*exp(-t/0.8)/0.64)';
ddf='0.4*(6*exp(-t/0.6)/0.36-6*exp(-t/0.8)/0.64)';
```

```
dt=0.001;    %step time
time=0:dt:3;

%initializing numerical joint positions and velocities for using in Euler
%integration method
NumDtheta(1,:)=zeros(1,3);
Numtheta(1,:)=[deg2rad(subs(theta1,t,time(1))),...
    deg2rad(subs(theta2,t,time(1))),subs(f,t,time(1))];

% "for" loop to compute desired joint positions, joint velocities,
% joint accelerations, and torque along time using inverse dynamic. Also
% this loop is simultaneously used to compute joint position, velocities
% and acceleration resulting from forward dynamic (simulation) by applying
% the torque profiles obtained form inverse dynamic
for i=1:length(time)
%inverse dynamic part:

    %Computing time-varying D-H parameters against time
    theta=[subs(theta1,t,time(i));subs(theta2,t,time(i));0];
    d=[0.4;0.1;subs(f,t,time(i))];

    %Computing desired joint velocities by substituting time
    JointVelocities=[subs(dtheta1,t,time(i));subs(dtheta2,t,time(i))...
        ;subs(df,t,time(i))];

    %Computing desired joint accelerations by substituting time
    JointAcceleration=[subs(ddtheta1,t,time(i));subs(ddtheta2,t,time(i))...
        ;subs(ddf,t,time(i))];

    %Computing torque by using inverse dynamic
    Tau(i,:)=InvDyn(JointVelocities,JointAcceleration,alpha,a,d,theta,...
        TypeOfJoints,m,I,Pc);

%Forward dynamic part:

    %seting variables needed to compute mass matrix, velocity vector
    %and gravity vector for the Stanford manipulator
    NumTheta1=Numtheta(i,1);
    NumTheta2=Numtheta(i,2);
    NumF=Numtheta(i,3);
    NumDtheta1=NumDtheta(i,1);
    NumDtheta2=NumDtheta(i,2);
    NumDf=NumDtheta(i,3);

    %computing mass matrix for the Stanford manipulator
    M=StanfordMass(NumTheta1,NumTheta2,NumF);
    detM(i)=det(M); %computing determinant of mass matrix

    %computing velocity vector for the Stanford manipulator
    V(i,:)=StanfordVelocity(NumTheta1,NumTheta2,NumF,NumDtheta1,...
        NumDtheta2,NumDf);
    %computing gravity vector for the Stanford manipulator
    G(i,:)=StanfordGravity(NumTheta1,NumTheta2,NumF);
```

```
    %using forward dynamic to compute joint acceleration by applying
    % the torque obtained from inverse dynamic stage
    NumDDtheta(i,:)=M\(Tau(i,:)-V(i,:)-G(i,:)).';

    %using Euler integration method to obtain joint velocities
    NumDtheta(i+1,:)=NumDtheta(i,:)+NumDDtheta(i,:)*dt;

    %using Euler integration method to obtain joint positions
    Numtheta(i+1,:)=Numtheta(i,:)+NumDtheta(i,:)*dt+...
        0.5*NumDDtheta(i,:)*dt^2;
end

figure(1)
subplot(311)
plot(time, subs(theta1,t,time))
grid on
% Create ylabel
ylabel('\theta_1(t) (\circ)');

% Create title
title('Joint positions of the Stanford manipulator','FontWeight','bold');

subplot(312)
plot(time, subs(theta2,t,time))
grid on;

% Create ylabel
ylabel('\theta_2(t) (\circ)');

subplot(313)
plot(time, subs(f,t,time))
grid on
% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('f(t) (m)');

figure(2)
subplot(311)
plot(time, subs(dtheta1,t,time))
grid on
% Create ylabel
ylabel('\theta\prime_1(t) (\circ/sec)');

% Create title
title('Joint velocities of the Stanford manipulator','FontWeight','bold');

subplot(312)
plot(time, subs(dtheta2,t,time))
grid on;

% Create ylabel
ylabel('\theta\prime_2(t) (\circ/sec)');
```

```
subplot(313)
plot(time, subs(df,t,time))
grid on
% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('f\prime(t) (m/sec)');

figure(3)
subplot(311)
plot(time, subs(ddtheta1,t,time))
grid on
% Create ylabel
ylabel('\theta\prime\prime_1(t) (\circ/sec^2)');

% Create title
title('Joint accelerations of the Stanford manipulator',...
    'FontWeight','bold');

subplot(312)
plot(time, subs(ddtheta2,t,time))
grid on;

% Create ylabel
ylabel('\theta\prime\prime_2(t) (\circ/sec^2)');

subplot(313)
plot(time, subs(ddf,t,time))
grid on
% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('f\prime\prime(t) (m/sec^2)');
figure(4)
plot(time,Tau(:,1),time,Tau(:,2),time,Tau(:,3))

% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('\tau');

% Create title
title('Torque applied of each link of the Stanford manipulator',...
    'FontWeight','bold');
% Create legend
legend('1st link','2nd link','3rd link');

figure(5)
subplot(311)
plot(time, subs(theta1,time),'--')
```

```
hold on
plot(time,Numtheta(1:length(time),1)*180/pi)
% Create ylabel
ylabel('\theta_1(t) (\circ)');

% Create title
title('Simulation: Joint positions',...
    'FontWeight','bold');
% Create legend
legend('Desired positions','Simulation results');
subplot(312)
plot(time, subs(theta2,time),'--')
hold on
plot(time,Numtheta(1:length(time),2)*180/pi)
% Create ylabel
ylabel('\theta_2(t) (\circ)');
subplot(313)
plot(time, subs(f,time),'--')
hold on
plot(time,Numtheta(1:length(time),3))
% Create ylabel
ylabel('f(t) (m)');
% Create xlabel
xlabel('Time (sec)');

figure(6)
subplot(311)
plot(time, subs(dtheta1,time),'--')
hold on
plot(time,NumDtheta(1:length(time),1)*180/pi)
% Create ylabel
ylabel('\theta\prime_1(t) (\circ/sec)');
% Create title
title('Simulation: Joint velocities',...
    'FontWeight','bold');
% Create legend
legend('Desired velocities','Simulation results');

subplot(312)
plot(time, subs(dtheta2,time),'--')
hold on
plot(time,NumDtheta(1:length(time),2)*180/pi)
% Create ylabel
ylabel('\theta\prime_2(t) (\circ/sec)');
subplot(313)
plot(time, subs(df,time),'--')
hold on
plot(time,NumDtheta(1:length(time),1))
% Create ylabel
ylabel('f\prime(t) (m/sec)');
% Create xlabel
xlabel('Time (sec)');


figure(7)
subplot(311)
```

```
plot(time, subs(ddtheta1,time),'--')
hold on
plot(time,NumDDtheta(1:length(time),1)*180/pi)
% Create ylabel
ylabel('\theta\prime\prime_1(t) (\circ/sec^2)');
% Create title
title('Simulation: Joint accelerations',...
    'FontWeight','bold');
% Create legend
legend('Desired accelerations','Simulation results');
subplot(312)
plot(time, subs(ddtheta2,time),'--')
hold on
plot(time,NumDDtheta(1:length(time),2)*180/pi)
% Create ylabel
ylabel('\theta\prime\prime_2(t) (\circ/sec^2)');

subplot(313)
plot(time, subs(ddf,time),'--')
hold on
plot(time,NumDDtheta(1:length(time),3))
% Create ylabel
ylabel('f\prime\prime(t) (m/sec^2)');
% Create xlabel
xlabel('Time (sec)');

figure(8)
plot(time,detM(1:length(time)),'r')
```

Appendix 6. Matlab code of "Test_Stanford_Dynamic_AdamBashford" program:

```
% This program can be used to get inverse dynamic and forward dynamic
% results for the Stanford manipulator using Adam-Bashford integration
% technique
%
%       Peyman Yadmellat

%       April 4, 2010
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------


clc
clear all
close all
% Initializing D-H parameters for the Stanford manipulator
m=[4;2;2];
alpha=[0;-90;90];
a=zeros(3,1);
% determining joint type of the Stanford manipulator
TypeOfJoints='rrp';
% setting the Stanford manipulator configuration including mass, inertia
% tensor, and position of center of mass
```

```
m=[4;2;2];
I1=zeros(3,3);
I2=zeros(3,3);
I3=zeros(3,3);
I=[I1 I2 I3];

Pc1=[0;0;0];
Pc2=[0;0;0];
Pc3=[0;0;0];
Pc=[Pc1 Pc2 Pc3];

% defining joint positions, velocities and accelerations
syms t theta1 theta2 dtheta1 dtheta2 ddtheta1 ddtheta2 f df ddf
theta1='45*(1+6*exp(-t/0.6)-8*exp(-t/0.8))';
theta2='45*(1+6*exp(-t/0.6)-8*exp(-t/0.8))';
f='0.4*(1+6*exp(-t/0.6)-6*exp(-t/0.8))';

dtheta1='45*(-6*exp(-t/0.6)/0.6+8*exp(-t/0.8)/0.8)';
dtheta2='45*(-6*exp(-t/0.6)/0.6+8*exp(-t/0.8)/0.8)';
df='0.4*(-6*exp(-t/0.6)/0.6+6*exp(-t/0.8)/0.8)';

ddtheta1='45*(6*exp(-t/0.6)/0.36-8*exp(-t/0.8)/0.64)';
ddtheta2='45*(6*exp(-t/0.6)/0.36-8*exp(-t/0.8)/0.64)';
ddf='0.4*(6*exp(-t/0.6)/0.36-6*exp(-t/0.8)/0.64)';


dt=0.001;    %step time
time=0:dt:3;

%initializing numerical joint positions and velocities for using in Euler
%integration method
NumDtheta(1,:)=zeros(1,3);
Numtheta(1,:)=[deg2rad(subs(theta1,t,time(1))),...
    deg2rad(subs(theta2,t,time(1))),subs(f,t,time(1))];

% "for" loop to compute desired joint positions, joint velocities,
% joint accelerations, and torque along time using inverse dynamic. Also
% this loop is simultaneously used to compute joint position, velocities
% and accelerations resulting from forward dynamic (simulation) by applying
% the torque profiles obtained form inverse dynamic
for i=1:length(time)
%inverse dynamic part:

    %Computing time-varying D-H parameters against time
    theta=[subs(theta1,t,time(i));subs(theta2,t,time(i));0];
    d=[0.4;0.1;subs(f,t,time(i))];

    %Computing desired joint velocities by substituting time
    JointVelocities=[subs(dtheta1,t,time(i));subs(dtheta2,t,time(i))...
        ;subs(df,t,time(i))];

    %Computing desired joint accelerations by substituting time
    JointAcceleration=[subs(ddtheta1,t,time(i));subs(ddtheta2,t,time(i))...
        ;subs(ddf,t,time(i))];
```

```
    %Computing torque by using inverse dynamic
    Tau(i,:)=InvDyn(JointVelocities,JointAcceleration,alpha,a,d,theta,...
        TypeOfJoints,m,I,Pc);

%Forward dynamic part:

    %seting variables needed to compute mass matrix, velocity vector
    %and gravity vector for the Stanford manipulator
    NumTheta1=Numtheta(i,1);
    NumTheta2=Numtheta(i,2);
    NumF=Numtheta(i,3);
    NumDtheta1=NumDtheta(i,1);
    NumDtheta2=NumDtheta(i,2);
    NumDf=NumDtheta(i,3);

    %computing mass matrix for the Stanford manipulator
    M=StanfordMass(NumTheta1,NumTheta2,NumF);
    detM(i)=det(M); %computing determinant of mass matrix

    %computing velocity vector for the Stanford manipulator
    V(i,:)=StanfordVelocity(NumTheta1,NumTheta2,NumF,NumDtheta1,...
        NumDtheta2,NumDf);
    %computing gravity vector for the Stanford manipulator
    G(i,:)=StanfordGravity(NumTheta1,NumTheta2,NumF);

    %using forward dynamic to compute joint acceleration by applying
    % the torque obtained from inverse dynamic stage
    NumDDtheta(i,:)=M\(Tau(i,:)-V(i,:)-G(i,:)).';

    %using Euler integration method to obtain joint velocities
    NumDtheta(i+1,:)=NumDtheta(i,:)+NumDDtheta(i,:)*dt;

    %using Euler integration method to obtain joint positions
    Numtheta(i+1,:)=Numtheta(i,:)+NumDtheta(i,:)*dt+...
        0.5*NumDDtheta(i,:)*dt^2;


    %using Adam Bashford integration method to obtain joint velocities
    if i>3
    NumDtheta(i+1,:)=NumDtheta(i,:)+(55/24)*NumDDtheta(i,:)*dt-...
                    (59/24)*NumDDtheta(i-1,:)*dt+(37/24)*NumDDtheta(i-2,:)*dt-
...
                     (9/24)*NumDDtheta(i-3,:)*dt;

    %using Adam Bashford integration method to obtain joint positions
    Numtheta(i+1,:)=Numtheta(i,:)+(55/24)*NumDtheta(i,:)*dt-...
                    (59/24)*NumDtheta(i-1,:)*dt+(37/24)*NumDtheta(i-2,:)*dt-
...
                     (9/24)*NumDtheta(i-3,:)*dt;
     else
        NumDtheta(i+1,:)=NumDtheta(i,:)+NumDDtheta(i,:)*dt;
        Numtheta(i+1,:)=Numtheta(i,:)+NumDtheta(i,:)*dt+...
                        0.5*NumDDtheta(i,:)*dt^2;
```

```
    end
end

figure(1)
subplot(311)
plot(time, subs(theta1,t,time))
grid on
% Create ylabel
ylabel('\theta_1(t) (\circ)');

% Create title
title('Joint positions of the Stanford manipulator','FontWeight','bold');

subplot(312)
plot(time, subs(theta2,t,time))
grid on;

% Create ylabel
ylabel('\theta_2(t) (\circ)');

subplot(313)
plot(time, subs(f,t,time))
grid on
% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('f(t) (m)');

figure(2)
subplot(311)
plot(time, subs(dtheta1,t,time))
grid on
% Create ylabel
ylabel('\theta\prime_1(t) (\circ/sec)');

% Create title
title('Joint velocities of the Stanford manipulator','FontWeight','bold');

subplot(312)
plot(time, subs(dtheta2,t,time))
grid on;

% Create ylabel
ylabel('\theta\prime_2(t) (\circ/sec)');

subplot(313)
plot(time, subs(df,t,time))
grid on
% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('f\prime(t) (m/sec)');
```

```
figure(3)
subplot(311)
plot(time, subs(ddtheta1,t,time))
grid on
% Create ylabel
ylabel('\theta\prime\prime_1(t) (\circ/sec^2)');

% Create title
title('Joint accelerations of the Stanford manipulator',...
    'FontWeight','bold');

subplot(312)
plot(time, subs(ddtheta2,t,time))
grid on;

% Create ylabel
ylabel('\theta\prime\prime_2(t) (\circ/sec^2)');

subplot(313)
plot(time, subs(ddf,t,time))
grid on
% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('f\prime\prime(t) (m/sec^2)');
figure(4)
plot(time,Tau(:,1),time,Tau(:,2),time,Tau(:,3))

% Create xlabel
xlabel('Time (sec)');

% Create ylabel
ylabel('\tau');

% Create title
title('Torque applied of each link of the Stanford manipulator',...
    'FontWeight','bold');
% Create legend
legend('1st link','2nd link','3rd link');

figure(5)
subplot(311)
plot(time, subs(theta1,time),'--')
hold on
plot(time,Numtheta(1:length(time),1)*180/pi)
% Create ylabel
ylabel('\theta_1(t) (\circ)');

% Create title
title('Simulation: Joint positions',...
    'FontWeight','bold');
% Create legend
```

```
legend('Desired positions','Simulation results');
subplot(312)
plot(time, subs(theta2,time),'--')
hold on
plot(time,Numtheta(1:length(time),2)*180/pi)
% Create ylabel
ylabel('\theta_2(t) (\circ)');
subplot(313)
plot(time, subs(f,time),'--')
hold on
plot(time,Numtheta(1:length(time),3))
% Create ylabel
ylabel('f(t) (m)');
% Create xlabel
xlabel('Time (sec)');

figure(6)
subplot(311)
plot(time, subs(dtheta1,time),'--')
hold on
plot(time,NumDtheta(1:length(time),1)*180/pi)
% Create ylabel
ylabel('\theta\prime_1(t) (\circ/sec)');
% Create title
title('Simulation: Joint velocities',...
    'FontWeight','bold');
% Create legend
legend('Desired velocities','Simulation results');

subplot(312)
plot(time, subs(dtheta2,time),'--')
hold on
plot(time,NumDtheta(1:length(time),2)*180/pi)
% Create ylabel
ylabel('\theta\prime_2(t) (\circ/sec)');
subplot(313)
plot(time, subs(df,time),'--')
hold on
plot(time,NumDtheta(1:length(time),1))
% Create ylabel
ylabel('f\prime(t) (m/sec)');
% Create xlabel
xlabel('Time (sec)');


figure(7)
subplot(311)
plot(time, subs(ddtheta1,time),'--')
hold on
plot(time,NumDDtheta(1:length(time),1)*180/pi)
% Create ylabel
ylabel('\theta\prime\prime_1(t) (\circ/sec^2)');
% Create title
title('Simulation: Joint accelerations',...
    'FontWeight','bold');
% Create legend
```

```
legend('Desired accelerations','Simulation results');
subplot(312)
plot(time, subs(ddtheta2,time),'--')
hold on
plot(time,NumDDtheta(1:length(time),2)*180/pi)
% Create ylabel
ylabel('\theta\prime\prime_2(t) (\circ/sec^2)');

subplot(313)
plot(time, subs(ddf,time),'--')
hold on
plot(time,NumDDtheta(1:length(time),3))
% Create ylabel
ylabel('f\prime\prime(t) (m/sec^2)');
% Create xlabel
xlabel('Time (sec)');

figure(8)
plot(time,detM(1:length(time)),'r')
```