International Journal of Control and Automation
Vol. 10, No. 5 (2017), pp.83-92
http//dx.doi.org/10.14257/ijca.2017.10.5.08

Multibody Dynamics Modeling and Simulating Using Maple and Maplesim

Liu Shaogang[1] and Edris Farah[2],

[1]College of Mechanical & Electrical Engineering, Harbin Engineering University, Harbin, China
[2]Department of Mechanical Engineering, Karary University, Khartoum, Sudan
edrisfarah@yahoo.com

Abstract

This paper presents a general purpose of modeling and simulating multibody dynamics in Maple and Maplesim. The method is exemplified by modeling 2DOF Robot arm in Maple using Newton-Euler formulation algorithm. Drag-and-drop Physical modeling tool in Maplesim is used to simulate the dynamics of the same robotic system. Results from both Maple and Maplesim model are compared. The paper shows in tutorial form how Maple and Maplesim are very excellent environment for modeling and simulating multibody systems.

Keywords*: Robotic; Newton-Euler; Maple; Maplesim; Dynamic Modeling*

1. Introduction

Dynamic modeling means deriving equations that clearly describes the relationship between force and motion. These equations are important to consider in robot simulation, and robot control design algorithms. There are several methods to formulate the robot motion dynamic, such as the Euler-Lagrange formulation [1-4], and Newton-Euler formulation [5-8] which are the most familiar methods. Euler-Lagrange formulation is energy based approach which describe the evolution of a mechanical system subject to holonomic constraints, while The Newton-Euler formulation is quite different because each link of the manipulator is studied in turn. To derive the dynamic model of a manipulator robot with n-link using Newton-Euler formulation; first we need to do a forward recursion which represent the linear and angular motion of the robot, then we do the backward recursion to compute the forces and torques.

In this work we show how to develop and implement the Newton-Euler recursive technique in Maple software and how to generate the differential equations of motion in state-space form and solve them. This paper shows how we can create and simulate the same robotic system by Drag-and-Drop physical modeling tools in Maplesim and get faster results equivalent to Maple results.

2. Newton-Euler Algorithm

The dynamics equations of the robot can be generated iteratively by Newton-Euler formalism. For **m** equal the mass, **I** the inertia tensor, **v** the velocity, Ω the angular velocity, and **F** and **N** are the external forces and moments acting in each link of the robot, the Newton equation is:

$$m\frac{d}{dt}v = \sum_{ext} F \tag{1}$$

And the Euler equation is:

$$I\frac{\partial}{\partial t}\Omega+\Omega\times I\Omega=\sum_{ext}N \tag{2}$$

These two equations can be solved iteratively in two loop circle. First the velocities and accelerations are determined by calculating in an outward direction from the base of the robot to its end-effector. Then, the forces and moments are gained by inward calculations in the opposite direction.

The resulting equations can be combined as matrix-vector differential equations, the so called state-space form:

$$M\left(q\right)\ddot{q}+V\left(q,\dot{q}\right)+G\left(q\right)=\tau \tag{3}$$

Where **M** is (nxn) positive definite mass matrix, **V** is (nx1) vector of centripetal and Coriolis terms, and **G** is (nx1) vector of the gravity components.

In Newton-Euler algorithm we define **R** as the transformation matrix for neighboring link coordinate frame, (theta) and **d** are free Denavit-Hartenberg parameters for rotational and prismatic joints, respectively. **P** is coordinating system origins of neighboring frames, f and n are external forces and moment, and **k** is the unit vector of a link coordinates system pointing along the joint axis.

2.1. Outward Iterations

Velocities and Accelerations for each joint prismatic or revolute can be calculated from base to end-effector in outward direction by the next equations [9-11]:

1. Angular velocity for revolute joint or prismatic joint:

$$\begin{cases} Revolute: & {}^{i}_{i}\Omega = {}^{i}_{i-1}R\times{}^{i-1}_{i-1}\Omega+\dot{\theta}_i Z_i \\ Prismatic: & {}^{i}_{i}\Omega = {}^{i}_{i-1}R\times{}^{i-1}_{i-1}\Omega \end{cases} \tag{4}$$

2. Angular acceleration for the two kind of joints:

$$\begin{cases} Revolute: & {}^{i}_{i}\dot{\Omega} = {}^{i}_{i-1}R\left({}^{i-1}_{i-1}\dot{\Omega}+{}^{i}_{i}\Omega\times\dot{\theta}_i k_i\right)+\ddot{\theta}_i k_i \\ Prismatic: & {}^{i}_{i}\dot{\Omega} = {}^{i}_{i-1}R\,{}^{i-1}_{i-1}\dot{\Omega} \end{cases} \tag{5}$$

3. Linear acceleration can be calculated by the next equation system:

$$\begin{cases} Rev: & a_i = {}^{i}_{i-1}R\left(a_{i-1}+{}^{i-1}_{i-1}\dot{\Omega}\times{}^{i-1}_{i}p+{}^{i-1}_{i-1}\Omega\times\left({}^{i-1}_{i-1}\Omega\times{}^{i-1}_{i}p\right)\right) \\ Pri: & a_i = {}^{i}_{i-1}R\left(a_{i-1}+{}^{i-1}_{i-1}\dot{\Omega}\times{}^{i-1}_{i}p+{}^{i-1}_{i-1}\Omega\times\left({}^{i-1}_{i-1}\Omega\times{}^{i-1}_{i}p\right)\right)+2\,{}^{i}_{i}\Omega\times\dot{d}_i Z_i+\ddot{d}_i Z_i \end{cases} \tag{6}$$

4. Linear acceleration for center of gravity:

$$a_{Ci}={}^{i}_{i-1}R\left({}^{i}_{i}\dot{v}+{}^{i}_{i}\dot{\Omega}\times{}^{i}_{i}p_C+{}^{i}_{i}\Omega\times\left({}^{i}_{i}\Omega\times{}^{i}_{i}p_C\right)\right) \tag{7}$$

5. Force at the center of gravity:

$$F_i=m_i a_{Ci} \tag{8}$$

6. Moment at the center of gravity:

$$N_i = I_i\,{}^{i}_{i}\dot{\Omega} + {}^{i}_{i}\Omega \times I_i\,{}^{i}_{i}\Omega \qquad (9)$$

2.2. Inward Iterations

Forces and Moments Calculation for each joint prismatic or revolute can be calculated from base to end-effector in inward direction by the next equations:

1. Forces and moments for revolute and prismatic joint:

$$\begin{cases} Force: & {}^{i-1}_{i-1}f = {}^{i-1}_{i}R\,{}^{i}_{i}f + {}^{i-1}_{i-1}F \\ Torque: & {}^{i-1}_{i-1}n = {}^{i-1}_{i}R\,{}^{i}_{i}n + {}^{i-1}_{i}p \times {}^{i-1}_{i}R\,{}^{i}_{i}f + {}^{i-1}_{i-1}p_c \times {}^{i-1}_{i-1}F + {}^{i-1}_{i-1}N \end{cases} \qquad (10)$$

2. Controlling forces and moments for revolute and prismatic joint:

$$\begin{cases} Rev: & \tau_{i-1} = {}^{i-1}_{i-1}n^T Z_{i-1} \\ Pri: & \tau_{i-1} = {}^{i-1}_{i-1}f^T Z_{i-1} \end{cases} \qquad (11)$$

If the base is not rotating the initial value is:

$$\begin{cases} & {}^{0}_{0}\Omega = {}^{0}_{0}\dot{\Omega} = 0, \\ \text{Else} & \\ & {}^{0}_{0}\Omega = \Omega_0 \quad , \quad {}^{0}_{0}\dot{\Omega} = \dot{\Omega}_0 \end{cases} \qquad (12)$$

Influence of gravitational acceleration g:

$${}^{0}_{0}\dot{v} = (0,0,g)^T \qquad (13)$$

If the robot moving freely in space, then the Forces and moments acting on the end effector are:

$$\begin{cases} & {}^{n}_{n}f = {}^{n}_{n}n = 0, \\ \text{Else} & \\ & {}^{n}_{n}f = f_n \quad , \quad {}^{n}_{n}n = n_n \end{cases} \qquad (14)$$

3. Newton-Euler Algorithm in Maple

Maple is an interactive mathematical problem solving environment with rich sets of data structures and powerful programming language developed by Maplesoft [12]. The highly nonlinear differential equation No.3 can be solved in Maple with (dsolve) command [13] and visualized with (plot) command. In this paper Maple 18 has been used to model an example of a Frictionless double pendulum, which is equivalent to 2dof robot arm to show how we can model and simulate robotic system in Maple.

The coordinate systems of the pendulum are shown in Figure 1. The masses M1 and M2 are concentrated at the center of each link. The two links with length L1 and L2 respectively. Spring damper has been added to joint 1 to simulate torque in joint1.

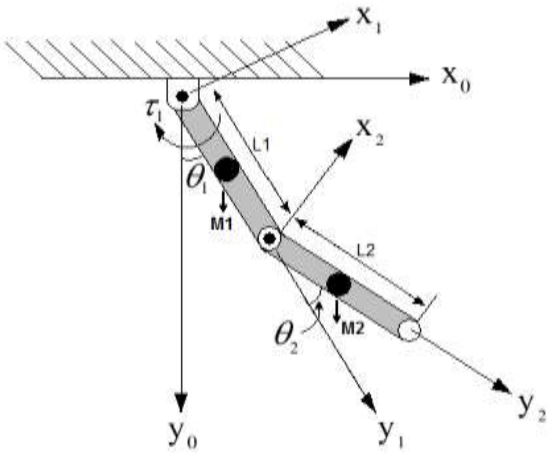

Figure 1. Double Pendulum Coordinates

Outward iterations and inward iterations in the previous section can be implemented in Maple programing as follows:

3.1. Outward Procedure

```
> restart:
> with(LinearAlgebra):
> outward := proc (m::scalar, I_C::matrix, P::Vector, P_C::Vector,
R::matrix, o::Vector, odot::Vector, tdot::scalar, tddot::scalar,
vdot::Vector, o_p::name, odot_p::name, vdot_p::name, F_p::name,
 N_p::name)
local vdot_C, Z ;
Z := Vector([0, 0, 1]) ;
o_p := R.o + tdot*Z ;
odot_p := (R.odot) + (R.o) &x (tdot*Z)+tddot*Z ;
vdot_p := R.(odot &x P) + R.(o &x (o &x P)) + R.vdot ;
vdot_C := (odot_p &x P_C) + o_p &x (o_p &x P_C) + vdot_p;
F_p := m*vdot_C ;
N_p := I_C.odot_p + Z &x (I_C.o_p) ;
end
```

3.2. Inward Procedure

```
> inward:=proc(F::Vector,N::Vector,R::matrix,p::Vector,
p_C::Vector,f_p::Vector,n_p::Vector,f::name,n::name)
f:=F + (R.f_p);
n:=N + (R.n_p)+(p_C &x  F) + p  &x (R.f_p);
n[3]
end
```

3.3. Define Some Matrices and Vectors for the Pendulum

```
> I1 := matrix(3, 3, 0);
> P1 := Vector(3, 0);
> PC1 := Vector([0, L1/2, 0]);
> R1 := matrix([[cos(T1), sin(T1), 0], [-sin(T1), cos(T1), 0], [0, 0, 1]]);
> O0 := Vector(3, 0);
> O0d := Vector(3, 0);
> v0d := Vector([0, -g, 0]);
```

3.4. Perform First Outward Iteration

```
> outward(m1, I1, P1, PC1, R1, O0, O0d,T1d,
   T1dd, v0d, O1, O1d, v1d, F1,N1)
```

3.4. Define Further Entries

```
> I2 := matrix(3, 3, 0):
> P2 := Vector([0, L1, 0]):
> PC2 := Vector([0, L2/2, 0]):
> R2 := matrix([[cos(T2), sin(T2), 0], [-sin(T2), cos(T2), 0], [0, 0, 1]]):
```

3.5. Perform a Second Outward Iteration

```
> outward(m2, I2, P2, PC2, R2, O1, O1d, T2d,
   T2dd, v1d, O2, O2d, v2d, F2, N2)
```

3.6. Define Further Entries

```
> f3 := Vector(3,0) :  n3:= Vector(3,0) :
> R3:= matrix(3, 3, 0) :  P3:= Vector(3) :
```

3.7. Perform First Inward Iteration

```
> with(MTM):
> tau[2] := inward(F2, N2, transpose(R3), P3, PC2, f3, n3, f2, n2) :
```

3.8. Perform a Second Inward Iteration

```
> with(MTM):
> tau[1] := inward(F1,N1,transpose(R2),P2,PC1,f2,n2,f1,n1):
```

3.9. Add rotational Spring Damper to joint1

```
> tau[1] := T1*c+T1d*d+tau[1]
```

Where d is the damping constant and c is the spring constant.

3.10. Find M the Mass Matrix

```
> tau[1] := combine(tau[1], trig):
> tau[2] := combine(tau[2], trig):
> tau[1] := collect(tau[1], [T1dd, T2dd, g], distributed):
> tau[2] := collect(tau[2], [T1dd, T2dd, g], distributed):
> M := matrix([[coeff(tau[1], T1dd, 1), coeff(tau[1], T2dd, 1)],
            [coeff(tau[2], T1dd, 1), coeff(tau[2], T2dd, 1)]]) :
```

Result is Mass matrix :

$$M = \begin{bmatrix} L_1L_2m_2\cos(T2) + \frac{1}{4}L_2^2m_2 + \frac{1}{4}L_1^2m_1 + L_1^2m_2 & \frac{1}{4}L_2^2m_2 + \frac{1}{2}L_1L_2m_2\cos(T2) \\ \frac{1}{4}L_2^2m_2 + \frac{1}{2}L_1L_2m_2\cos(T2) & \frac{1}{4}L_2^2m_2 \end{bmatrix}$$

3.11. Invert of the Mass Matrix:

```
> Mv := MatrixInverse(M):
```

3.12. Find V the Centripetal and Coriolis terms:

```
> V := Vector([tcoeff(tau[1], [T1dd, T2dd, g]), tcoeff(tau[2],
         [T1dd, T2dd, g])]):
```

Centripetal and Coriolis Vector:

$$V = \begin{bmatrix} -L_1L_2m_2.T2d.\sin(T2)\left(T1d + \frac{1}{2}T2d\right) + c.T1 + d.T2d \\ \frac{1}{2}L_1L_2m_2.T2d^2.\sin(T2) \end{bmatrix}$$

3.13. Extract G the Gravity Terms

```
> G := Vector([coeff(tau[1], g, 1), coeff(tau[2], g, 1)]):
> G := ScalarMultiply(G, g);
```

GravityVector:

$$G = \begin{bmatrix} \left(\frac{1}{2}\sin(T1+T2)L_2m_2 + \frac{1}{2}L_1m_1\sin(T1) + L_1m_2\sin(T1)\right)g \\ \frac{1}{2}\sin(T1+T2)gL_2m_2 \end{bmatrix}$$

3.14. Set the Differential Equations

```
> RH := Mv.(-Add(V, G)): # right hand side of the state equation
> RH[1] := subs({T1 = theta[1](t), T2 = theta[2](t)}, RH[1]):
> RH[2] := subs({T1 = theta[1](t), T2 = theta[2](t)}, RH[2]):
> RH[1] := subs({T1d = diff(theta[1](t), t),
                    T2d = diff(theta[2](t), t)}, RH[1]):
> RH[2] := subs({T1d = diff(theta[1](t), t),
                    T2d = diff(theta[2](t), t)}, RH[2]):
> ode[1] := diff(theta[1](t), t, t) = RH[1]:
> ode[2] := diff(theta[2](t), t, t) = RH[2]:
```

3.15. Assign the Numerical Values

```
> L1 := 1: L2 := 1: m1 := 1: m2 := 1: c := 1: d := 1: g := 9.8
> st := time( )
```

3.16. Solve the Differential Equations

```
> F := dsolve({ode[1], ode[2], theta[1](0) = 0.05, theta[2](0) = 0.05
   ,(D(theta[1]))(0) = 0, (D(theta[2]))(0) = 0},{theta[1](t), theta[2](t)}
   ,type = numeric, method = dverk78, output = listprocedure) :
```

Solution

F:=[t = proc (t)...end proc, θ_1(t) = proc (t)...end proc, $\frac{d}{dt}\theta_1$(t) = proc (t)...end proc,
θ_2(t) = proc (t)...end proc, $\frac{d}{dt}\theta_2$(t) = proc (t)...end proc]

3.17. Plotting Graphs of Trajectories for Joint1

```
> X := eval(theta[1](t),F):
    Z := eval(diff(theta[1](t),t),F):
> plot(X, 0 .. 10,title = "shoulder angle")# fig.2a
> plot(Z, 0 .. 10,title = "shoulder speed")# fig.2b
> plot(D(Z), 0 .. 10,title = "shoulder acceleration")# fig.2c
> plot(-(d.Z+c.X), 0 .. 10,title = "shoulder torque")# fig.2d
```

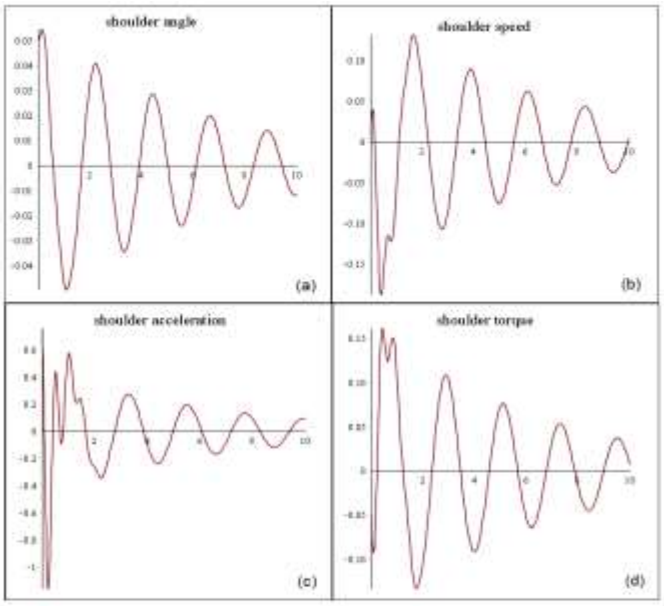

Figure 2. Shoulder Trajectories: a: Angle, b: Speed, C: Acceleration, d: Torque

3.17. Plotting Graphs of Trajectories for Joint 2

```
> Y := eval(theta[2](t),F):
  Q := eval(diff(theta[2](t),t),F):
> plot(Y, 0 .. 10,title ="elbow angle")  # fig.3a
> plot(Q, 0 .. 10,title ="elbow speed")  # fig.3b
> plot(D(Y), 0 .. 10,title ="elbow acceleration")  # fig.3c
> plot(0, 0 .. 10,title ="elbow torque")  # fig.3d
```

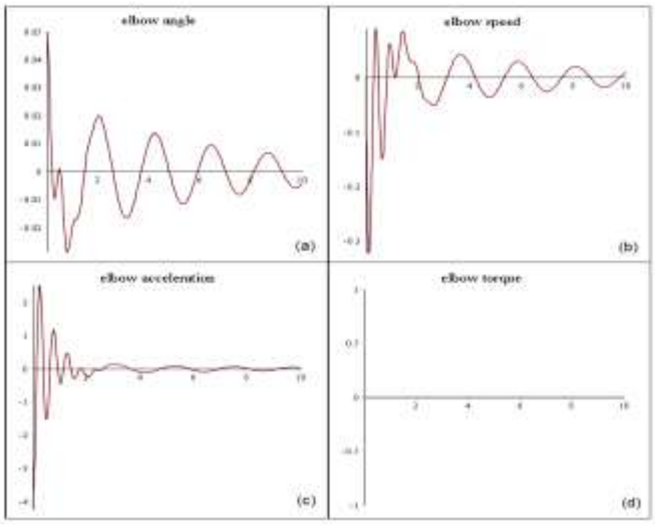

Figure 3. Elbow Trajectories: a: Angle, b: Speed, C: Acceleration, d: Torque

4. System Simulation in Maplesim

Maplesim is a high-performance, multi-domain modeling and simulation environment. It is automatically generated system equations with real-time simulation code for complex systems, developed by Maplesoft. Figure 4 shows the physical modeling of the double pendulum in Maplesim. It was created in Maplesim 6.4 by adding and connecting two revolute joints and two rigid bodies for each link with mass in the center, spring damper is added to simulate the torque in joint 1, and two probes have been added to show the graphs of the simulation result. In each probe angle, angular speed, angular acceleration and torque are selected as required measurements.

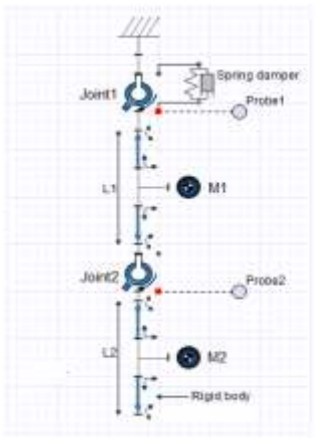

Figure 4. The Double Pendulum Model in Maplesim

The selected measurements in probes will be generated automatically during the simulation as shown in Figure 5, and Figure 6. One can notice that the result graphs from Maplesim are equivalent to Maple results in Figure 2 and Figure 3 which confirm the validity of the method followed for modeling the robot dynamics in Maple.

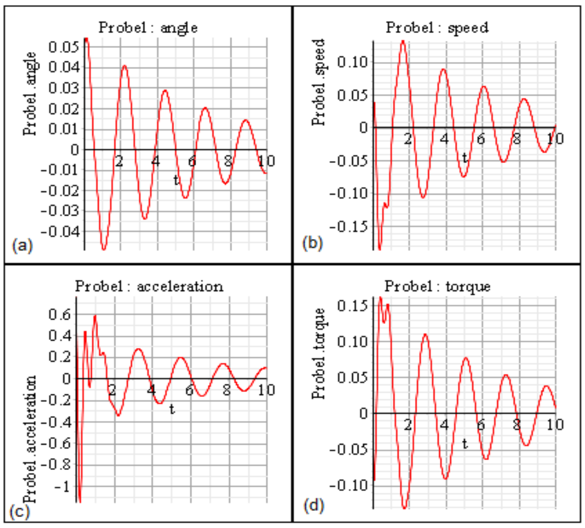

Figure 5. Probe1 Results: a: Angle, b: Speed,C: Acceleration, d: Torque

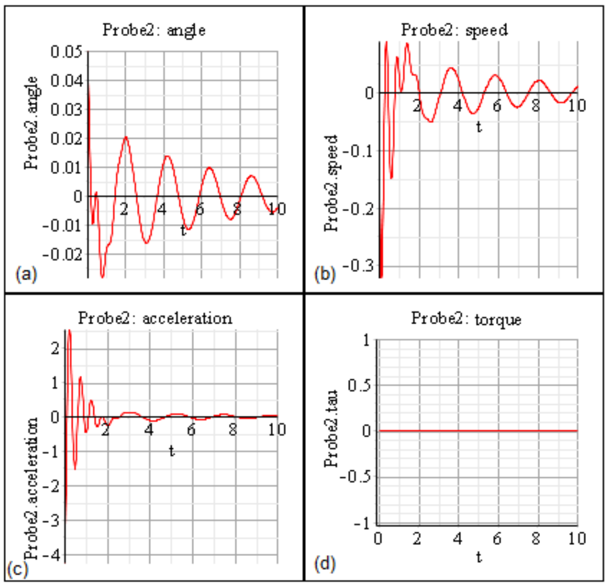

Figure 6. Probe2 Results: a: Angle, b: Speed, C: Acceleration, d: Torque

5. Conclusion

It has been shown that Maple and Maplesim are an excellent environment for modeling and simulating multibody systems. These kind of symbolic programming gives a much deeper understanding of multibody dynamics than pure numeric equations, also the use of drag-and-drop tool in maplesim is bringing your model to live and make it easy for you to change the parameters and update your model.

Acknowledgment

The authors gratefully acknowledge the financial support of the International Exchange Program of Harbin Engineering University for innovation and talents cultivation.

References

[1] V. Falkenhahn, "Dynamic modeling of constant curvature continuum robots using the Euler-Lagrange formalism", in Intelligent Robots and Systems, 2014 IEEE/RSJ International Conference on, **(2014)**.
[2] H. Li, "Dynamics and optimization of a 2-DOF parallel robot with flexible links", In Intelligent Control and Automation, 7th World Congress on, IEEE, **(2008)**.
[3] D. Li, "Dynamic Model of a 3 DOF Direct Drive Robot and Its Control Mode", In Control and Automation, IEEE International Conference on, IEEE, **(2007)**.
[4] B. Dasgupta and T. S. Mruthyunjaya, "A Newton-Euler formulation for the inverse dynamics of the Stewart platform manipulator", Mechanism and Machine Theory, vol. 33.8, **(1998)**, pp. 1135-1152.
[5] W. Khalil, "Dynamic Modeling of Robots Using Newton-Euler Formulation", Informatics in Control, Automation and Robotics. Springer Berlin Heidelberg, **(2011)**, pp. 3-20.
[6] E. Stoneking, "Newton-euler dynamic equations of motion for a multi-body spacecraft", AIAA Guidance, Navigation, and Control Conference, **(2007)**.
[7] A. De Luca and L. Ferrajoli, "A modified newton-euler method for dynamic computations in robot fault detection and control", Robotics and Automation, ICRA'09, IEEE International Conference on, **(2009)**.
[8] V. Aslanov, G. Kruglov and V. Yudintsev, "Newton–Euler equations of multibody systems with changing structures for space applications", Acta Astronautica, vol. 68.11, **(2011)**, pp. 2080-2087.
[9] F. L. Lewis, D. M. Dawson and C. T. Abdallah, "Robot manipulator control: theory and practice", CRC Press, **(2003)**.
[10] M. W. Spong, S. Hutchinson and M. Vidyasagar, "Robot modeling and control", New York: Wiley, vol. 3, **(2006)**.
[11] J. J. Craig, "Introduction to robotics: mechanics and control", Upper Saddle River, NJ, USA: Pearson/Prentice Hall, **(2005)**,.
[12] Available from: http://www.maplesoft.com/. 15, Sep, **(2015)**.
[13] A. Heck and W. Koepf, "Introduction to MAPLE", New York: Springer-Verlag, **(1993)**.

Authors

Liu Shaogang, he is Professor, doctoral tutor. Harbin Engineering University, mechanical design and theory research on design and theory of mechanical discipline leaders, director of the Institute and the government of Heilongjiang province science and Technology Economic Advisory Committee of experts. Currently engaged in mechanical and electronic engineering, mechanical design and theory, computer control system, pneumatic emission (ejection) research technology, fire protection technology, ecological protection technology and equipment. He has many publication in a scholar journal in the above research areas.

Edris Farah, his nationality is Sudanese. He received his B.S. degree in mechanical engineering from KARARY University Sudan in 2001, his M.S. degree from KARARY University in mechanical design in 2010, and his Ph.D. degree in mechatronic engineering is from Harbin engineering University China in 2015. His research area interest is in surgical robotic systems.