MODELING, IDENTIFICATION AND CONTROL, 1995, VOL. 16, NO. 3, 155–167
doi:10.4173/mic.1995.3.4

Analytical generation of the dynamical equations for mechanical manipulators

GEIR HORN† and SVEIN LINGE†

Keywords: *Manipulator models, symbolic computation*

A package to generate the symbolic dynamic equations describing the relation between forces and movements for serial mechanical linkages with rigid constituents is presented. The relative movement between the rigid parts is assumed to be either a rotation about an axis or a translation along an axis. Two algorithms are implemented, a Lagrange–Euler method and a Newton–Euler method. The former can be used to solve both the inverse and the forward dynamics problems, while the latter requires fewer arithmetical operations but only allows solution of the inverse dynamics problem. Two test examples are presented, the double pendulum and the modified Stanford manipulator.

1. Introduction

ROBMAT (ROBotics with *MAThematica*) is a package for symbolic generation of the dynamical equations for mechanical linkages, such as robot manipulators. The dynamical equations describe how applied forces and movements are related for the mechanism. A manipulator is a mechanical arm made of rigid pieces, usually with a gripper or tool at the moving end (see Fig. 1). The pieces can be moved relative to each other by the use of actuators. Relative movements can be of different kinds, normally either a rotation around an axis or a translation along an axis. We require the relative motion between two successive parts to be either of these. The axis of movement is called the joint axis and the rigid pieces are called links. To limit the structural complexity, only manipulators with an open structural chain of links are handled.

The use of such manipulators started in the 1940s, when systems for handling dangerous radioactive material were developed to free man from this hazardous environment (Goertz 1963). Research, design, and manufacturing of industrial manipulators took off in the 1960s, with repeatable or hazardous tasks as the main application areas. However, experience showed that these devices were more difficult to control than perhaps first imagined.

The dynamical equations for a manipulator can be very complex, containing several force contributors in nontrivial expressions. For very simple manipulators, with only two joints or so, they can be derived by hand. However, the process is tedious and prone to error, and the difficulty increases dramatically with the number of joints. This makes it tempting to utilize a computer for the task. Even when a computer is used, often only the major force contributors are considered, while the minor ones are left out of the mathematical expression. In the package described here, only inertia, Coriolis,

Received 10 November 1994.
† SINTEF-SI, Forskningsveien 1, Postboks 124, Blindern, N-0314 Oslo, Norway. e-mail Geir. Horn@si.sintef. no
Reprinted, with permission, from *The Mathematica Journal*, 1994, Vol. 4, No. 4, 67–73.

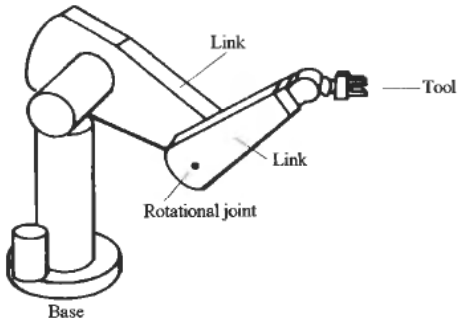

Figure 1. The six joint PUMA 560 manipulator. (From [Craig 1989], reprinted by permission of Addison-Wesley).

centrifugal, and gravity forces are modeled, which is common practice. This means that factors such as friction, flexibility of the 'rigid' links, joint compliance, and so on, are disregarded.

Symbolic generation of the dynamical equations is fundamental for developing good control algorithms, but also for choosing good design solutions (Toumi 1992). The performance of a symbolically modeled manipulator can be analysed before it is actually built, thereby making good design choices possible. The symbolic form is also important from a computational viewpoint, since evaluating it for the numerical values of its components is less computationally expensive than computing the same result from scratch by applying the algorithms to the numerical quantities directly (Ju and Mansour 1989, Koplik and Leu 1986, Leu and Hemati 1986). The autonomous generation of symbolic manipulator dynamics has been addressed by several other researchers, such as (Buffinton 1990; Ju and Mansour 1989; Yin and Yuh 1989; Wloka 1989; Cheng *et al.* 1988; Koplik and Leu 1986; Leu and Hemati 1986). We believe our package is the first to use an algorithm developed by Li (1988) to obtain the dynamical equations in symbolic form.

We address both the forward and the inverse dynamics problem. The forward dynamics problem is the task of computing the motion resulting from given input forces, while the inverse dynamics problem is the task of computing forces and torques required for a desired movement. Our implementation covers one algorithm from each of two approaches, Lagrange–Euler and Newton–Euler, to the derivation of symbolic dynamical equations. The two approaches are covered here briefly; for tutorials, see Driels *et al.* (1988) on the former and White *et al.* (1989) on the latter.

Since most readers are probably unfamiliar with fundamental robotics theory, we also explain the use of coordinate systems necessary to produce the dynamical equations via ROBMAT. Following a description of the package, we present two examples: a double pendulum and a modified Stanford manipulator.

2. Mathematical description of a manipulator

To develop a mathematical description of a manipulator, one first has to assign a coordinate system to each link. We present here a common way of doing this, due to Denavit and Hartenberg (1955); a comprehensive treatment can be found in Fu *et al.* (1987).

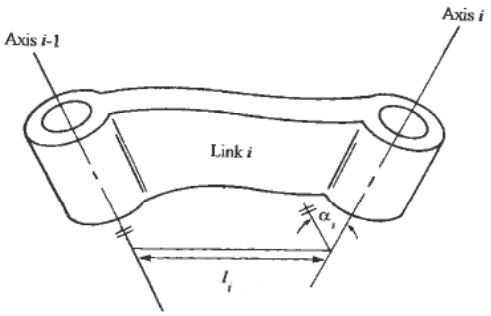

Figure 2. A link connecting two rotational joints. (From [Paul 1981], reprinted by permission of MIT Press).

2.1. *The parameters for one link*

First, one identifies the axes around which we have a rotation, and the axes along which we have a translation. These will be the z-axes of the three-dimensional coordinate systems to be assigned. Consider a link connecting two rotational joints, as shown in Fig. 2.

Any link i can be characterized by two parameters: the length l_i measured along the common normal of both axes, and the spatial twist angle α_i between the two joint axes. These two measures are illustrated in Fig. 2 and are fixed parameters for the link.

We assign the coordinate system for the ith link at its distal end and place the origin of the coordinate frame at the axis of movement where the length normal, along which l_i is measured, meets the axis. From now on, this axis will be referred to as the z_i-axis.

2.3. *Binding links together*

Introducing the next link into the chain, we get a new normal to the z_i-axis measuring the length l_{i+1} of link $i+1$. This line will generally not meet the z_i-axis at the origin. We will denote the offset between the intersection point and the z_i-axis origin by d_{i+1}. This offset is fixed for rotational joints, as illustrated in Fig. 3. For prismatic joints, at which a translational movement takes place along the z_i-axis, this distance will be a function of time.

Between the two normals along which l_i and l_{i+1} are measured, there will be a spatial angle. It is customary to denote this angle θ_{i+1} (Fig. 3). For a rotational joint with axis

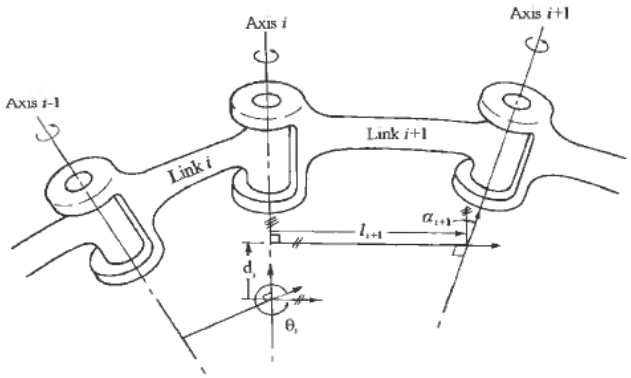

Figure 3. Two links connected at a rotational joint. (From [Paul 1981], reprinted by permission of MIT Press).

z_i, this angle will be a function of time. For a prismatic joint, however, this angle will remain a fixed parameter.

We have three fixed parameters for each link: its defined length l_i, its twist α_i, and either the displacement d_{i+1} (for rotational joints) or the angle θ_{i+1} (for prismatic joints). For practical design reasons, a joint of a real industrial manipulator has only one degree of freedom; it is either rotational or prismatic.

2.4. *The coordinate frames, transformation matrix and position vector*

So far, we have only located the origins and the z axes of the coordinate systems. Let the x_i axis be directed along the normal where l_i is measured and pointing towards the next joint axis, and let the y_i axis be orthogonal to both the x_i and the z_i axes such that the coordinate frame forms a right-hand coordinate system.

The only exception is when the z_i and z_{i-1} axes intersect. Then $l_i = 0$ and we find the direction of the z_i axis as $\mathbf{x}_i = \mathbf{z}_i \times \mathbf{z}_{i-1}$.

At the end of the manipulator chain, the axes of the coordinate system for link n are chosen to be initially aligned with that of the coordinate frame for the link $n-1$, as the final displacement d_n or rotation angle θ_n occurs with respect to z_{n-1}.

Starting with coordinate system $i-1$ and aligning it with the coordinate frame i, we have to

- rotate about z_{i-1} an angle θ_i
- translate along z_{i-1} a distance d_i
- now, x_{i-1} points in direction of x_i, and we translate along this axis a distance l_i
- rotate the twist angle α_i about x_i.

The transformation matrix between these frames can be written as a product of 4×4 homogeneous transformation matrices (Denavit and Hartenberg 1955):

$$\mathbf{A}_i = \mathrm{Rot}(z_{i-1}, \theta_i)\,\mathrm{Trans}(0, 0, d_i)\,\mathrm{Trans}(l_i, 0, 0)\,\mathrm{Rot}(x_i, \alpha_i)$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & l_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & l_i\sin\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & l_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given this matrix, we can transform a vector $\mathbf{v}^i$, which is the three-dimensional vector given in the coordinate frame i, augmented with 1 as a fourth element, to the corresponding vector $\mathbf{v}^{i-1}$ represented in the coordinate system $i-1$ by

$$\mathbf{v}^{i-1} = \mathbf{A}_i \mathbf{v}^i$$

In the last column in the transformation matrix, we recognize the position vector from

the origin in coordinate system $i-1$ to the origin of coordinate system i, expressed in coordinate system $i-1$

$$\mathbf{p}_i = [l_i \cos(\theta_i) l_i \sin(\theta_i) d_i]^{\mathrm{T}}$$

2.5. *The inertia tensor*

To describe the mass distribution around the centre of mass of a rigid body, the inertia tensor $\mathbf{I}_c$ is defined for a coordinate system attached to the centre of mass as

$$\mathbf{I}_c = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

The diagonal elements are termed mass moments of inertia, and each is computed as the integral of the mass element times the square of the perpendicular distance to the corresponding axis α:

$$I_{\alpha\alpha} = \iiint (\beta^2 + \gamma^2) dm$$

where β and γ are distances along the other two axes of the coordinate system. Similarly, the inertia products are defined by

$$I_{\alpha\beta} = \iiint \alpha\beta dm$$

To find the kinetic energy of the rigid body, one needs the pseudo-inertia matrix $\mathbf{J}_c$, computed from the inertia tensor as

$$\mathbf{J}_c = \begin{bmatrix} \frac{1}{2}(-I_{xx} + I_{yy} + I_{zz}) & I_{xy} & I_{xz} \\ I_{xy} & \frac{1}{2}(-I_{xx} + I_{yy} + I_{zz}) & I_{yz} \\ I_{xz} & I_{yz} & \frac{1}{2}(-I_{xx} + I_{yy} + I_{zz}) \end{bmatrix}$$

When defining a manipulator in ROBMAT, one must provide an inertia tensor for each link, with respect to a coordinate system at the centre of mass that has identical orientation with the coordinate system attached to the link. Whenever the pseudo inertia matrix is needed, ROBMAT will use the formula above to compute $\mathbf{J}_c$ from the given $\mathbf{I}_c$.

3. Describing a manipulator in ROBMAT

For each joint and link, ROBMAT needs to know its type and fixed parameters. It also needs the inertia tensor for the associated link and a vector giving the position of the centre of mass for the link expressed in the link's coordinate frame. These inputs are arranged as a list

$$L_j = \{P \text{ or } R,\ l_j, \alpha_j, \theta_j \text{ or } d_j, m_j, \mathbf{I}_{cj}, \mathbf{r}_j\}$$

Here, the letter P is used to specify a prismatic joint and the letter R a revolute joint. For the former joint type, the fixed value of θ_j must be given, while the latter requires the fixed value of d_j. m_j denotes the mass of the link, $\mathbf{I}_{cj}$ is the inertia tensor, and $\mathbf{r}_j$ is the $n \times 1$ position vector to the centre of mass, also measured with respect to the link's coordinate frame. Either $\mathbf{I}_{cj}$ *or* $\mathbf{r}_j$, or both, may be omitted, in which case the default

values of zero will be used, corresponding to a point mass at the origin of the link coordinate system. As *Mathematica* is a symbolic tool, we emphasize that these parameters need not be numerical values.

The entire manipulator, consisting of n such link lists, is given to ROBMAT by calling the function DefineManipulator with n arguments, each a list defining one link:

$$\mathsf{M} = \mathsf{DefineManipulator}[\mathrm{L}_1, \ldots, \mathrm{L}_n]$$

This function does a parameter check and returns an object of type manipulator, with any default arguments expanded.

To start using a defined manipulator with ROBMAT, the function

$$\mathsf{ActivateManipulator}[\mathsf{M}]$$

needs to be called with a manipulator object M as argument. This syntax allows one to define several different manipulators, or to maintain different configurations of one manipulator, and then easily change the manipulator for which ROBMAT does its computations. ROBMAT assumes one configuration valid until another is activated with ActivateManipulator. Hence, this syntax also reduces the number of parameters needed to be passed to the functions carrying out the computations.

4. The dynamical equations

Several methods exist for the description of manipulator dynamics. Apart from the Lagrange–Euler and Newton–Euler methods, there are methods utilizing other principles, such as Kane's equations (Kane and Levinson 1983), the D'Alembert principle (Lee *et al.* 1983), and the Gibbs function (Rudas and Toth 1993). However, the first two methods have gained the most widespread use, Lagrange–Euler because its form is appropriate for control purposes, and Newton–Euler because of its low demand on arithmetical computations. ROBMAT implements both methods, which are briefly outlined below.

4.1. *Generalized coordinates*

A manipulator is not free to move everywhere; its movement is constrained to its reachable subspace, determined by all possible configurations of the joint variables d and θ. Because of the constraints, there are relations among the coordinates and the mechanical equations of motion are not all independent. To overcome this difficulty, generalized coordinates are introduced. A generalized coordinate can be any useful quantity; it need not be a conventional orthogonal position coordinate.

For a manipulator, the generalized coordinate q_{i+1} is the joint variable, that is, d_{i+1} for a prismatic joint or θ_{i+1} for a revolute joint. The vector of all such coordinates,

$$\mathbf{q} = [q_1, \ldots, q_n]^{\mathrm{T}}$$

is a vector in the *joint space* of the manipulator.

4.2. *The Lagrange–Euler approach*

To obtain the equations of motion by the Lagrange–Euler approach, first form the Lagrangian energy function,

$$\mathrm{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathrm{T}(\dot{\mathbf{q}}, t) - \mathrm{V}(\mathbf{q}, t)$$

where $T(\dot{\mathbf{q}}, t)$ is the kinetic energy and $V(\mathbf{q}, t)$ is the potential energy, such that the generalized force acting at joint i is determined by

$$\mathscr{F}_i = -\frac{\partial}{\partial q_i} V(\mathbf{q}, t) = \frac{\partial}{\partial q_i} L(\mathbf{q}, \dot{\mathbf{q}}, t)$$

Now we are able to state Hamilton's principle of least action: *The motion of the system from time t_1 to t_2 is such that the action (line) integral*

$$W = \int_{t_1}^{t_2} L(\mathbf{q}, \dot{\mathbf{q}}, t) dt$$

attains a minimum.

A sufficient condition for the integral to have a minimum value is

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = F_i$$

for all i, which is the famous Lagrange equation of motion (Goldstein 1981). Here, F_i is an externally applied generalized force, which is either a force or a torque, depending upon whether q_i is a linear or an angular coordinate. This force comes in addition to $\mathscr{F}_i$ and does not originate from the potential.

Lagrange's equations were used by Uicker (1966), Kahn and Roth (1971), Paul (1981), Bejczy (1974) to derive the dynamic equation of robot manipulators with n degrees of freedom (see Paul (1981) for a comprehensive treatment). The resulting equations can be written as

$$\mathbf{F} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + 2\mathbf{B}(\mathbf{q})\dot{\mathbf{q}}\dot{\mathbf{q}} + \mathbf{C}(\mathbf{q})\dot{\mathbf{q}}^2 + \mathbf{G}(\mathbf{q})$$

which is called the configuration space equation, since the coefficient matrices only depend on the current arm configuration $\mathbf{q}$ (Raibert 1977). Here, $\mathbf{F}$ is an $n \times 1$ vector of externally applied forces needed to generate the movement described by $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$. $\mathbf{M}(\mathbf{q})$ is an $n \times n$ symmetric positive definite inertia matrix and $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}$ represents the inertial forces vector. The $n \times [n(n-1)/2]$ Coriolis coefficient matrix $\mathbf{B}(\mathbf{q})$ is composed of submatrices

$$\mathbf{B}(\mathbf{q}) = [\mathbf{B}_1(\mathbf{q}), \mathbf{B}_2(\mathbf{q}) \ldots, \mathbf{B}_{n-1}(\mathbf{q})]$$

where $\mathbf{B}_i(\mathbf{q})$ is an $n \times (n-i)$ matrix whose submatrix from the $i+1$th row to the nth row is skew-symmetric. Further, the $[n(n-1)/2] \times 1$ vector of joint velocity products is given by

$$\dot{\mathbf{q}}\dot{\mathbf{q}} = [(\dot{\mathbf{q}}\dot{\mathbf{q}}_1)^{\mathrm{T}}, (\dot{\mathbf{q}}\dot{\mathbf{q}}_2)^{\mathrm{T}}, \ldots, (\dot{\mathbf{q}}\dot{\mathbf{q}}_{n-1})^{\mathrm{T}}]^{\mathrm{T}}$$

$$\dot{\mathbf{q}}\dot{\mathbf{q}}_i = [\dot{q}_i\dot{q}_{i+1}, \dot{q}_i\dot{q}_{i+2}, \ldots, \dot{q}_i\dot{q}_n]^{\mathrm{T}}$$

The product $2\mathbf{B}(\mathbf{q})\dot{\mathbf{q}}\dot{\mathbf{q}}$ represents the Coriolis forces vector. $\mathbf{C}(\mathbf{q})\dot{\mathbf{q}}^2$ is the centrifugal force vector, where $\mathbf{C}(\mathbf{q})$ is an $n \times n$ matrix of centrifugal coefficients and $\dot{\mathbf{q}}^2$ is the $n \times 1$ vector

$$\dot{\mathbf{q}}^2 = [\dot{q}_1^2, \dot{q}_2^2, \ldots, \dot{q}_n^2]^{\mathrm{T}}$$

The configuration space equation express clearly the different contributors to the dynamics (excluding the dynamics of gear friction, backlash, and the electronic control device). It therefore forms the best available basis for the design of control systems and the dynamic simulation of manipulators.

ROBMAT computes the coefficient matrices of the configuration space equation for the activated manipulator by means of the function ConfigurationSpace[q, g], where **q** is the vector of generalized joint variables and **g** is the gravitation field vector with respect to the base coordinate frame. This function returns the list of matrices {**M**, **B**, **C**, **G**}. The matrices will be simplified unless ROBMAT is told not to call Simplify by setting the option Simplification to False.

Until recently, computing the configuration space coefficient matrices was rather laborious. ROBMAT uses the recursive method of Li (1988) for this computation, reducing the computational burden in terms of arithmetical operations to be only slightly less efficient than the Newton–Euler approach, at least for manipulators of up to about six degrees of freedom.

To encourage the study of the effect of the different physical terms in the configuration space equation, ROBMAT contains two functions for computing the vectors used in the Coriolis term and in the centrifugal term: qq[$\dot{\mathbf{q}}$] computes the vector $\dot{\mathbf{q}}\dot{\mathbf{q}}$ and $\mathbf{q}^2[\dot{\mathbf{q}}]$ computes the vector $\dot{\mathbf{q}}^2$.

4.3. *The Newton–Euler approach*

The force acting on the centre of mass of a rigid body accelerating with acceleration $\dot{\mathbf{v}}_c$ is given by Newton's second law

$$\mathbf{F}_c = m\dot{\mathbf{v}}_c$$

where m is the mass of the body. For a body rotating with angular velocity ω_c around its centre of mass with an angular acceleration $\dot{\omega}_c$, Euler's equation

$$\mathbf{N}_c = \mathbf{I}_c\dot{\omega}_c + \omega_c \times (\mathbf{I}_c\omega)$$

gives the torque $\mathbf{N}_c$ that acts on the body to cause this rotation. Here, $\mathbf{I}_c$ is the inertia tensor of the rigid body about its centre of mass.

By successively applying these two equations to the links of the manipulator, Luh *et al.* (1980) derived a recursive algorithm for computing the dynamic equations of motion. Their algorithm consists of the following steps.

(1) Compute the angular accelerations and the linear accelerations of the mass centre of each link. As the base of the manipulator is not moving, the computation starts at the base and propagates outwards.
(2) Then apply the Newton–Euler equations to compute the inertial force and moment acting at the centre of mass of each link.
(3) Each link experiences forces and torques exerted on it by its neighbours. Hence, this time we start at the tip of the manipulator, with the link having no outer neighbours, and work back towards the base, computing the net force and moment exerted on link i from the net force and moments on link $i + 1$ and the quantities computed in step 2 for link i.

The resulting expression for the externally applied forces in terms of **q**, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ can be computed by the function NewtonEuler[**q**, **g**], provided that ROBMAT is able to obtain the derivatives from the symbolic vector **q**. If this is not the case, the full form NewtonEuler[**q**, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, **g**] should be used. In both cases, the vector **g** is the gravitation vector expressed in the base coordinate frame. Again, the returned expression for **F** is simplified by ROBMAT unless the user sets the Simplification option to False.

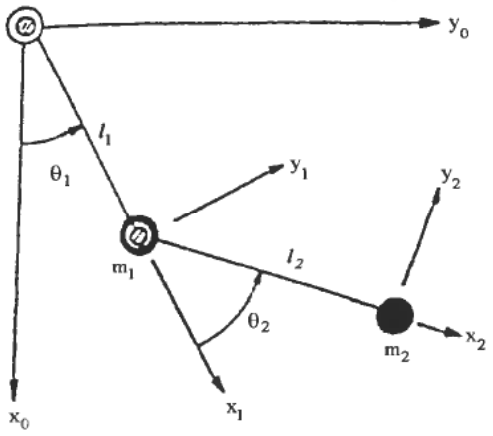

Figure 4. The double pendulum. l_1 and l_2 are the link lengths, m_1 and m_2 are the masses of the two links and θ_1 and θ_2 are the joint angles.

4.4. *The forward dynamics problem*

To simulate the movement under a given applied force $\hat{\mathbf{F}}$, we need a differential equation for $\mathbf{q}$, which is easily found from the configuration space equation as

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})[\hat{\mathbf{F}} - 2\mathbf{B}(\mathbf{q})\dot{\mathbf{q}}\dot{\mathbf{q}} - \mathbf{C}(\mathbf{q})\dot{\mathbf{q}}^2 - \mathbf{G}(\mathbf{q})]$$

The mass matrix is invertible because it is positive definite. This equation is also needed for studying the effects of imperfect modeling (see Craig (1989)).

5. The double pendulum

The double pendulum is an arrangement of two links that rotate relative to each other in a plane under the action of gravity. The two axes of rotation are parallel and oriented at a right angle to the plane of movement (see Fig. 4). This linkage is a common test arrangement in robotics. The dynamical equations can be found elsewhere (for example, in Craig (1989)), providing a check of the implemented algorithms.

For simplicity, the link masses are regarded as point masses. The linkage is specified with the DefineManipulator function:

```
In[1]: =    << RobMat.m

In[2]: =    DoublePendulum = DefineManipulator[
              {"R", L[1], 0, 0, m[1]}, {"R", L[2], 0, 0, m[2]}]

Out[2] =    —manipulator—
```

This manipulator will not be the subject for dynamical calculations until the ROBMAT activation command is issued:

```
In[3]: =    ActivateManipulator[DoublePendulum]
```

The components of the configuration space equation for the currently active manipulator are calculated by the function ConfigurationSpace. This function takes as input the symbolic vector of generalized coordinates and the gravity vector $\mathbf{g}$ expressed in the base coordinate system and returns the list of matrices $\{\mathbf{M}, \mathbf{B}, \mathbf{C}, \mathbf{G}\}$:

```
In[4]: = {M, B, CC, G} = ConfigurationSpace[
           {q[1][t], q[2][t]}, {g, 0, 0}, Simplification —> True];

In[5]: = M

Out[5] = {{L[1]^2 m[1] + L[1]^2 m[2] +
           2 Cos[q[2][t]] L[1] L[2] m[2] + L[2]^2 m[2],
           L[2] (Cos[q[2][t]] L[1] + L[2]) m[2]},
           {L[2] (Cos[q[2][t]] L[1] + L[2]) m[2], L[2]^2 m[2]}}

In[6]: = B

Out[6] = {{ - (L[1] L[2] m[2] Sin[q[2][t]])}, {0}}

In[7]: CC

Out[7] = {{0, - (L[1] L[2] m[2] Sin[q[2][t]])},
           {L[1] L[2] m[2] Sin[q[2][t]], 0}}

In[8]: = G

Out[8] = {g (L[1] m[1] Sin[q[1][t]] + L[1] m[2] Sin[q[1][t]] +
           L[2] m[2] Sin[q[1][t] + q[2][t]]),
           g L[2] m[2] Sin[q[1][t] + q[2][t]]}
```

The dynamics can also be computed using the Newton–Euler method by the function NewtonEuler:

```
In[9] = F = NewtonEuler[{q[1][t], q[2][t]}, {g, 0, 0},
          Simplification —> True]

Out[9] = {g L[1] m[1] Sin[q[1][t]] + g L[1] m[2] Sin[q[1][t]] +
           g L[2] m[2] Sin[q[1][t] + q[2][t]] -
           2 L[1] L[2] m[2] Sin[q[2][t]] (q[1])'[t] (q[2])'[t] -
           L[1] L[2] m[2] Sin[q[2][t]] (q[2])'[t]^2 +
           L[1]^2 m[1] (q[1])''[t] + L[1]^2 m[2] (q[1])''[t] +
           2 Cos[q[2][t]] L[1] L[2] m[2] (q[1])''[t] +
           L[2]^2 m[2] (q[1])''[t] +
           Cos[q[2][t]] L[1] L[2] m[2] (q[2])''[t] +
           L[2]^2 m[2] (q[2])''[t],
           L[2] m[2] (g Sin[q[1][t] + q[2][t]] +
             L[1] Sin[q[2][t]] (q[1])'[t]^2 +
             Cos[q[2][t]] L[1] (q[1])''[t] + L[2] (q[1])''[t] +
             L[2] (q[2])''[t])}
```

This result is identical to that obtained from the configuration space equation:

```
In[10]: = f = M.D[{q[1][t], q[2][t]}, {t, 2}] +
            2B.qq[D[{q[1][t], q[2][t]}, t]] +
            CC.q2[D[{q[1][t], q[2][t]}, t]] + G // Simplify;
```

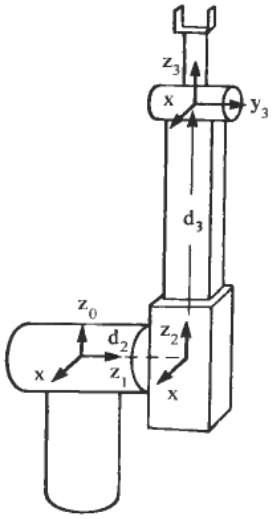

Figure 5. The Stanford manipulator. (From [Paul 1981], reprinted by permission of MIT Press.)

```
In[11]:= F - f // Simplify
Out[11]= {0, 0}
```

6. The modified Stanford manipulator

The Stanford manipulator (Fig. 5) is another common test manipulator (Scheinman 1969). It is often used in a modified form by excluding the three outermost links. As with the double pendulum, the dynamic equations in symbolic form can be found elsewhere (for example, Cheng *et al.* (1988)), providing another check on the correctness of the package. The manipulator is specified with the following inputs to ROBMAT:

```
In[12]:= Inertia[i_Integer] :=
           DiagonalMatrix[{Ixx[i], Iyy[i], Izz[i]}]
In[13]:= Stanford = DefineManipulator[
           {"R", 0, -Pi/2, 0, m[1], Inertia[1], {0, y[1], z[1]}},
           {"R", 0, Pi/2, d[2], m[2], Inertia[2], {0, y[2], 0}},
           {"P", 0, 0, 0, m[3], Inertia[3], {0, 0, z[3]}} ]
Out[13]= —manipulator—
```

Due to *Mathematicas's* difficulties simplifying the expressions, the division by 2 along the diagonal of the pseudo inertia matrix may need to be replaced with a multiplication by 0.5. The user can switch from the default division to multiplication by setting an option when the manipulator is activated.

```
In[14]:= ActivateManipulator[Stanford,
           PseudoInertiaCoefficient —> 0.5];
```

The dynamics are calculated by the following inputs. Since the results are rather lengthy, they are not shown here. The reader is referred to the notebook RobMat.ma, included with the electronic supplement or found at http://www.oslo.sintef.no/avd/31/3170/index.html

```
In[15]:= {M, B, CC, G} = ConfigurationSpace[
           {q[1][t], q[2][t], d[3][t]}, {0, 0, -g},
           Simplification --> True];

In[16]:= F = NewtonEuler[ {q[1][t], q[2][t], d[3][t]},
           {0, 0, -g}, Simplification --> True];
```

7. Conclusions

The ROBMAT package generates the dynamical equations for a rigid-link, open-chain manipulator with translational and rotational joints. Two different algorithms are implemented, one Lagrange–Euler and one Newton–Euler. Li's Lagrange–Euler algorithm (Li 1988) is used to generate the dynamical equations symbolically. ROBMAT provides the possibility of solving both the inverse and the forward dynamics problem at acceptable speeds, using the method most appropriate for the problem at hand.

The simplification process is by far the most time consuming step in generating the dynamical equations. When no simplification is demanded, we found that the Lagrange–Euler algorithm is faster than the Newton–Euler algorithm for both manipulators. For the simplified and final expressions, the Lagrange–Euler implementation is substantially faster than Newton–Euler for the modified Stanford manipulator, even when tested with zero inertia to avoid the pseudo-inertia problem, while the opposite was observed for the double pendulum. This is only an example of what may be expected since the two methods result in different unsimplified expressions for the same manipulator, which in turn gives Simplify a more or less easy job. We tried different combinations of Simplify for the two steps of the Lagrange–Euler method and observed that using Simplify only at the last step gave the fastest solution. This means that an early simplification in a succession of computations might not be the wisest thing to do.

Several possible extensions to ROBMAT would be useful. One is the ability to generate efficient C code. *Mathematica*'s CForm can currently be used, but the resulting code is far from optimal as common subexpressions are computed repeatedly, instead of being stored in memory. Other useful extensions would be the ability to model closed chain structures, to produce the linearized dynamical equations, and to generate the discrete-time dynamical model.

Acknowledgment

The authors would like to thank Professor Marc Raibert of MIT for kindly providing the reference Raibert (1977) immediately after our desperate inquiry informing him that our excellent librarians were forced to resign.

References

Bejczy, A. K. (1974). *Robot Arm Dynamics and Control.* NASA-JPL Technical Memorandum, 33–669 (Feb.).

Buffinton, K. W. (1990). Applications of PC-based multibody dynamics software in robotics. *Proc. of the ISMM Int. Symp. Computer Applications in Design, Simulation and Analysis*, March 5–7, New Orleans, USA, pp. 33–40.

Cheng, P. Y., Weng, C. O. and Chen, C. K. (1988). Symbolic derivation of dynamic equations of motion for robot manipulators using piogram symbolic method. *IEEE Journal of Robotics and Automation*, **4** (6), 599–609.

CRAIG, J. J. (1989). *Introduction to Robotics—Mechanics and Control.* 2nd ed. (Addison-Wesley, Massachusetts).

DENAVIT, J. and HARTENBERG, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 215–221 (June).

DRIELS, M. R., FAN, U. J. and PATHRE, U. S. (1988). The application of Newton–Euler recursive methods to the derivation of closed form dynamic equations. *Journal of Robotic Systems* **5** (3), 229–248.

FU, K. S., LEE, C. S. G. and GONZALES, R. C. (1987). *Robotics: Control, Sensing, Vision, and Intelligence*, Ch. 2 (McGraw-Hill, New York).

GOERTZ, R. C. (1963). Manipulators used for handling radioactive materials. *Human Factors in Technology,* chapter 27, ed. by E. M. Bennett (McGraw-Hill).

GOLDSTEIN, H. (1981). *Classical Mechanics* 2nd ed. (Addison-Wesley).

JU, M. S. and MANSOUR, J. M. (1989). Comparison of methods for developing the dynamics of rigid-body systems. *International Journal of Robotics Research*, **8** (6), 19–27.

KAHN, M. and ROTH, B. (1971). The near-minimum-time control of open loop kinematic chains. *Trans. ASME, Series G*, **93,** 164–172.

KANE, T. R. and LEVINSON, D. A. (1983). The use of Kane's dynamical equations in robotics. *The International Journal of Robotics Research*, **2** (3), 3–21.

KOPLIK, J. and LEU, M. C. (1986). Computer generation of robot dynamics equations and the related issues. *Journal of Robotic Systems*, **3** (3), 301–319.

LEE, C. S. G., LEE, B. H. and NIGAM, R. (1983). Development of the generalized D'Alembert equations of motion for robot manipulators. *Proc. 22nd conf. Dec. and Contr.*, San Antonio, TX, pp. 1205–1210.

LEU, M. C., and HEMATI, N. (1986). Automated symbolic derivation of dynamic equations of motion for robotic manipulators. *Trans. ASME J. Dyn. Syst., Meas. and Contr.*, **108,** 172–179.

LI, C. J. (1988) A new method of dynamics for robot manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, **18**.

LUH, J. Y. S., WALKER, M. W. and PAUL, R. P. (1980). On-line computational scheme for mechanical manipulators. *Transactions of the ASME*, pp. 69–76.

MURRAY, J. J.and NEUMAN, C. P. (1984) ARM: An algebraic robot dynamic modeling program. *Proc. of 1st. Int. IEEE Conf. on Robotics* ed. by R. P. Paul. Atlanta, GA, March, pp. 103–114.

PAUL, R. P. (1981). *Robot Manipulators: Mathematics Programming and Control* (M.I.T. Press, Cambridge, MA).

RAIBERT, M. H. (1977). *Mechanical Arm Control Using a State Space Memory.* Technical Paper MS77-750. Society of Manufacturing Engineers (SME), Dearborn, Michigan.

RUDAS, J. I., and TOTH, A. (1993). Efficient recursive algorithm for inverse dynamics. *Mechatronics*, **3**.

SCHEINMAN, V. D. (1969). *Design of a Computer Manipulator. AIM 92.* Stanford Artificial Intelligence Laboratory, Stanford University.

TOUMI, K. Y. (1992). Analysis and design of manipulators with decoupled and configuration-invariant inertia tensors using remote actuation. *Journal of Dynamic Systems, Measurement and Control* 114 (June), 204–212.

UICKER, J. J. JR. (1966). Dynamic force analysis of spatial linkages. *Mechanisms Conference.*

WHITE, W. N. JR., NIEMANN, D. D. and LYNCH, P. M. (1989). The presentation of Lagrange's equations in introductory robotics courses. *IEEE Transactions on Education,* **32** (Feb.), 39–46.

WLOKA, D. W. (1989). Efficient calculation of generalized forces in a robot simulation environment. *Proc. of the 3rd. European Simulation Congress.* Sept. 5–8, Belgium, pp. 595–601.

YIN, S., and YUH, J. (1989). An efficient algorithm for automatic generation of manipulator dynamic equations. *IEEE International Conference on Robotics and Automation.* May 14–19, Scottsdale, AZ.4.4, pp. 1812–1817.