Robotica (2003) volume 21, pp. 261–269. © 2003 Cambridge University Press
DOI: 10.1017/S0263574702004769 Printed in the United Kingdom

Solving the inverse dynamic control for low cost real-time industrial robot control applications

A. Valera†, V. Mata‡, M. Vallés†, F. Valero‡, N. Rosillo‡ and F. Benimeli‡

(Received in Final Form: November 9, 2002)

SUMMARY

This work deals with the real-time robot control implementation. In this paper, an algorithm for solving Inverse Dynamic Problem based on the Gibbs-Appell equations is proposed and verified. It is developed using mainly vectorial variables, and the equations are expressed in a recursive form, it has a computational complexity of $O(n)$. This algorithm will be compared with one based on Newton-Euler equations of motion, formulated in a similar way, and using mainly vectors in their recursive formulation. This algorithm was implemented in an industrial PUMA robot. For the robot control a new and open architecture based on PC had been implemented. The architecture used has two main advantages. First it provides a total open control architecture, and second it is not expensive. Because the controller is based on PC, any control technique can be programmed and implemented, and in this way the PUMA can work on high level tasks, such as automatic trajectory generation, task planning, control by artificial vision, etc.

KEYWORDS: Robotic manipulators; Inverse dynamic problem; Gibbs-Appell formulation; Robot control; Computer control; Digital computer applications.

1. INTRODUCTION

One of the main problems when attempting to establish a control of an industrial robotic system is in its own control unit, since this unit is generally a totally closed subsystem. Due to the fact that it uses its own operating system and, since it is impossible to modify this control system (not even the gain values), users have serious problems to implement conventional as well as advanced control strategies such force control and cooperative control of several robots. It is also important in programming automatic trajectory generation, control based on external sensing (such as vision), control strategies, etc.

In order to solve these kinds of problems we can find some solutions at Moreira et al.,[1] where the control hardware is modified, but generally it is neither trivial nor a cheap task. Valera et al.[2] shows a solution for these drawbacks which presents a very simple, economical and total open architecture for robot control.

This paper shows a real-time robot control using this architecture. The control algorithms are based on the Inverse Dynamic Problem (IDP) implementation using the Gibbs and Appell notation. The literature about the IDP in robots is vast. The interest in its potential applications (verification that the torques needed to execute a proposed trajectory do not exceed the capabilities of the actuators, IDP as part of the Inverse Dynamics Control etc.) has contributed to it.[3,4]

In order to increase their computational efficiency, many algorithms for solving the IDP have been proposed in the last thirty years. These algorithms are based on different Principles of Dynamics (Lagrange, Newton-Euler, Kane), the equations of motion are expressed in a closed-form or recursive formulation, and use different types of variables to express physical quantities. These algorithms can be implemented by means of symbolic programs or strictly numerical ones. Finally, according with the computer architecture with which they will be processed, the algorithms can be sequential or parallel.

Table I shows some of the proposed algorithms for solving the IDP on robots with rigid links and ideal joints.

Custom-made algorithms, which take advantage of the special characteristics of particular industrial robots, must be specially mentioned. Examples of these are proposed by

† Departamento de Ingeniería de Sistemas y Automática, Universidad Politécnica de Valencia, Valencia (SPAIN)
‡ Departamento de Ingeniería Mecánica y de Materiales, Universidad Politécnica de Valencia, Valencia (SPAIN)

Table I. Several algorithms for solving the IDP.

Authors	Dynamic Principle	Formulation	Type of variables	Type of resolution	Type of processing
(Hollerbach, 1980)	Lagrange-Euler	Recursive	Matricial	Numerical	Sequential
(Luh et al., 1980)	Newton-Euler	Recursive	Vectorial	Numerical	Sequential
(Angeles, et al.,[1] 1989)	Kane	Recursive	Tensorial	Numerical	Sequential
(Balafoutis and Patel, 1991)	Newton-Euler	Recursive	Tensorial	Numerical	Sequential
(Khalil and Kleifinger,[7] 1987)	Newton-Euler	Recursive	Vectorial	Symbolic	Sequential
(Lee and Chang,[8] 1986)	Newton-Euler	Recursive	Vectorial	Numerical	Parallel

Murray and Newman,[5] based on the Newton-Euler formulation, expressed in a recursive way and with vectorial variables.

Several authors, including Balafoutis and Patel,[6] have made evident that the computational efficiency of the dynamic algorithms depends fundamentally on the way the calculations are arranged rather than the dynamic principle in which they are based. This idea was already proposed by Hollerbach,[7] in which the dynamic problem was reformulated for robots by using the Principle of Lagrange in a recursive way and using a rotation matrix 3×3 instead of a 4×4 homogeneous transformation matrix. By this method, the computational complexity could be reduced from $O(n^4)$ to $O(n)$. Nevertheless, the computational complexity of the Hollerbach algorithm was as three times larger than the Luh, Walker and Paul algorithm.

On the other hand, it must be pointed out that important differences can be noticed about the computational efficiency assigned to algorithms of the same characteristics applied to robots of the same type. An algorithm based on Newton-Euler formulation, implemented in a recursive way, using vectorial variables and solved in a numerical and sequential way, can be found in Luh et al.[8] which had been assigned by Hollerbach a computational complexity of $150n$–48 multiplications and $131n$–48 additions, where n is the number of degrees of freedom of the robot. Fu et al.[9] provided a version of the same algorithm with a complexity of $117n$–24 multiplications and $103n$–21 additions. Zomaya[10] gave a complexity of $150n$ multiplications and $116n$ additions. Finally, Craig[11] gave a complexity of $126n$–99 multiplications and $106n$–92 additions.

The observed differences come fundamentally from the criteria used for counting operations, for instance, if operations that involve multiplications by variables with 0 or 1 values, are detected. Therefore, it seems necessary to indicate clearly the criteria that are going to be used for counting the operations when comparing the efficiency of algorithms.

The Gibbs-Appell equations were introduced by Gibbs in 1879 and formalised by Appell twenty years later. However, in the robot dynamics field there are few published references to works based on them. Renaud[12] stands out among the first references to the application of the Gibbs-Appell equations to dynamic modelling of robots, for which he produced remarkable commentaries on the previous work of E.P. Popov. Vukobratovic and Kircanski[13] developed a closed-form algorithm with a $O(n^3)$ computational complexity. Desoyer and Lugner[14] developed a recursive algorithm for solving the IDP in robots using the Jacobian matrix in order to avoid algebraic or numerical derivatives. The computational complexity of the proposed algorithm is $O(n^3)$.

In the present work the Gibbs-Appell equations are applied for solving the inverse dynamic problems of robots that have rigid links and ideal pairs. The algorithm proposed has a computational complexity of $O(n)$. This algorithms is formulated in a recursive way, using vectors to express most of the physical magnitudes involved in them (angular velocity, angular acceleration, etc.). In order to achieve a higher computational efficiency, the involved magnitudes in the Gibbs function are expressed with respect to local reference systems in the links. The computational efficiency of this algorithm will be compared with that of the Luh, Walker and Paul algorithm. That can be done since the same type of formulation is used in both (recursive) and the common physical magnitudes are expressed in the same way. It must be stated that the criteria to evaluate the number of operations will be the same in both algorithms.

This paper is organised as follows. In Section 2, the proposed algorithm is developed and an analysis, comparing it with the Luh, Walker and Paul algorithm, of its computational complexity is provided. Section 3 presents a proposed control architecture for industrial robots based on PC. In Section 4, the inverse dynamic control is addressed and is applied to a PUMA-type industrial robot. Finally, Section 5 summarises the development of the paper and suggests directions for future research.

2. THE GIBBS-APPEL FORMULATION APPLIED TO THE INVERSE DYNAMIC PROBLEM IN ROBOTS

In this section, the Gibbs-Appell equations are described, and two different formulations are presented to solve the Inverse Dynamic Problem on robot manipulators. The robots are modelled following the Denavit-Hartenberg modified notation, which considers four parameters θ_i, α_i, a_i, and d_i, as it is shown in Figure 1. In the mentioned notation, the reference system corresponding to link i is located on joint i, and the z-axis is located on the axis in the same node, which connects links i–1 and i.

The reference system i is related to the i–1 reference system by means of the rotation matrix ${}^{i-1}R_i$ and the position vector ${}^{i-1}\vec{r}_{O_{i-1},O_i}$:

$$ {}^{i-1}\boldsymbol{R}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 \\ \cos\alpha_i \cdot \sin\theta_i & \cos\alpha_i \cdot \cos\theta_i & -\sin\alpha_i \\ \sin\alpha_i \cdot \sin\theta_i & \sin\alpha_i \cdot \cos\theta_i & \cos\alpha_i \end{bmatrix} $$

$$ {}^{i-1}\vec{r}_{O_{i-1},O_i} = \begin{bmatrix} a_i \\ d_i \sin\alpha_i \\ -d_i \cos\alpha_i \end{bmatrix} $$

The Gibbs-Appell dynamic equations come from the Gibbs function definition (also known as the energy of the accelerations). When written in its original form for an

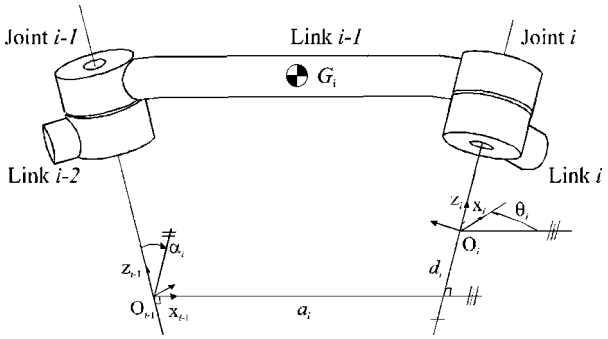

Fig. 1. Modified Denavit-Hartenberg notation.

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

arbitrary solid composed of n-elemental particles with masses m_i an acceleration a_i is (considering an inertial reference system):

$$G=\frac{1}{2}\sum_{i=1}^{n} m_i a_i^2$$

The Gibbs function for the i-th rigid solid is given by

$$G_i=\frac{1}{2}m_i(\ddot{\vec{r}}_{G_i})^{\mathrm{T}}\cdot\ddot{\vec{r}}_{G_i}+\frac{1}{2}(\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot\boldsymbol{I}_{G_i}\cdot\dot{\vec{\omega}}_i+(\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot[\vec{\omega}_i\wedge(\boldsymbol{I}_{G_i}\cdot\vec{\omega}_i)] \tag{1}$$

where m_i is the mass of the i-th link, I_{Gi} is the 3×3 matrix representing the centroidal matrix of inertia of the i-th link, $\vec{\omega}_i$ y $\dot{\vec{\omega}}_i$ are the three-dimensional vectors representing the angular velocity and acceleration of the i-th link, and $\ddot{\vec{r}}_{G_i}$ is the three-dimensional vector representing the acceleration of the mass centre of the i-th link. An inertial reference system is considered to express these magnitudes.

It is possible for these tensorial and vectorial magnitudes to be expressed considering a reference system located in the i-th link, so that the previous expression could be expressed as follows:

$$\begin{aligned}G_i=&\frac{1}{2}m_i({}^0\boldsymbol{R}_i\cdot{}^i\ddot{\vec{r}}_{G_i})^{\mathrm{T}}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\ddot{\vec{r}}_{G_i}\\&+\frac{1}{2}({}^0\boldsymbol{R}_i\cdot{}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\boldsymbol{I}_{G_i}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot({}^0\boldsymbol{R}_i\cdot{}^i\dot{\vec{\omega}}_i)\\&+({}^0\boldsymbol{R}_i\cdot{}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot\{{}^0\boldsymbol{R}_i\cdot{}^i\vec{\omega}_i\wedge[{}^0\boldsymbol{R}_i\cdot{}^i\boldsymbol{I}_{G_i}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\vec{\omega}_i]\}\end{aligned} \tag{2}$$

the expression (2) could rewritten as follows:

$$\begin{aligned}G_i=&\frac{1}{2}m_i({}^i\ddot{\vec{r}}_{G_i})^{\mathrm{T}}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\ddot{\vec{r}}_{G_i}\\&+\frac{1}{2}({}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\boldsymbol{I}_{G_i}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\dot{\vec{\omega}}_i\\&+({}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot\{{}^0\boldsymbol{R}_i\cdot{}^i\vec{\omega}_i\wedge[{}^0\boldsymbol{R}_i\cdot{}^i\boldsymbol{I}_{G_i}\cdot({}^0\boldsymbol{R}_i)^{T}\cdot{}^0\boldsymbol{R}_i\cdot{}^i\vec{\omega}_i]\}\end{aligned} \tag{3}$$

and taking into account the orthogonal nature of the rotation matrix, the scalar G_i would be given by:

$$\begin{aligned}G_i=&\frac{1}{2}m_i({}^i\ddot{\vec{r}}_{G_i})^{\mathrm{T}}\cdot{}^i\ddot{\vec{r}}_{G_i}+\frac{1}{2}({}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot{}^i\boldsymbol{I}_{G_i}\cdot{}^i\dot{\vec{\omega}}_i\\&+({}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot[{}^i\vec{\omega}_i\wedge{}^i\boldsymbol{I}_{G_i}\cdot{}^i\vec{\omega}_i]\end{aligned} \tag{4}$$

where ${}^i\ddot{\vec{r}}_{G_i}$, ${}^i\vec{\omega}_i$, ${}^i\dot{\vec{\omega}}_i$ and ${}^i\boldsymbol{I}_{G_i}$ are expressed in the i-th reference system.

For a system consisting of n-bodies, the Gibbs function of the system would be given by

$$G=\sum_{i=1}^{n} G_i \qquad (i=1,2\ldots n) \tag{5}$$

The Gibbs-Appell equations of motion are obtained from deriving the Gibbs function with respect to the generalised accelerations $\ddot{q}_j$, obtaining in this way the generalised inertial forces that are to equate to the generalised external forces, τ_j.

$$\tau_j=\sum_{i=j}^{n}\frac{\partial G_i}{\partial\ddot{q}_j} \qquad (j=1,2\ldots n) \tag{6}$$

that is,

$$\begin{aligned}\tau_j=\sum_{i=j}^{n}\Bigg\{&m_i({}^i\ddot{\vec{r}}_{G_i})^{\mathrm{T}}\cdot\frac{\partial\,{}^i\ddot{\vec{r}}_{G_i}}{\partial\ddot{q}_j}+({}^i\dot{\vec{\omega}}_i)^{\mathrm{T}}\cdot{}^i\mathbf{I}_{G_i}\cdot\frac{\partial\,{}^i\dot{\vec{\omega}}_i}{\partial\ddot{q}_j}\\&+\left(\frac{\partial\,{}^i\dot{\vec{\omega}}_i}{\partial\ddot{q}_j}\right)^{\mathrm{T}}\cdot[{}^i\vec{\omega}_i\wedge({}^i\mathbf{I}_{G_i}\cdot{}^i\vec{\omega}_i)]\Bigg\}\end{aligned} \tag{7}$$

The formulation for the solution of the Inverse Dynamic Problem in robots would be obtained by reorganizing and identifying two different terms in expression (7) as follows:

$$\begin{aligned}\tau_j=&\underbrace{\sum_{i=j}^{n}\left\{\left(\frac{\partial\,{}^i\dot{\vec{\omega}}_i}{\partial\ddot{q}_j}\right)^{\mathrm{T}}\cdot[{}^i\boldsymbol{I}_{G_i}\cdot{}^i\dot{\vec{\omega}}_i+{}^i\vec{\omega}_i\wedge({}^i\boldsymbol{I}_{G_i}\cdot{}^i\vec{\omega}_i)]\right\}}_{A_i}\\&+\underbrace{\sum_{i=j}^{n}\left[\left(\frac{\partial\,{}^i\ddot{\vec{r}}_{G_i}}{\partial\ddot{q}_j}\right)^{T}\cdot m_i\cdot{}^i\ddot{\vec{r}}_{G_i}\right]}_{B_i}\end{aligned} \tag{8}$$

It is remarkable that expression above is similar to the proposed by Angeles et al.[15] for solving the IDP based on the Kane's dynamic formulation.

To obtain the generalised forces, the angular velocities, angular accelerations, the accelerations of the origin of reference system of links and the accelerations of the centre of masses of the links can be obtained using the following known recursive expressions:

$${}^i\vec{\omega}_i={}^i\boldsymbol{R}_{i-1}\cdot{}^{i-1}\vec{\omega}_{i-1}+{}^i\vec{z}_i\cdot\dot{q}_i \tag{9}$$

$${}^i\dot{\vec{\omega}}_i={}^i\boldsymbol{R}_{i-1}\cdot{}^{i-1}\dot{\vec{\omega}}_{i-1}+{}^i\vec{z}_i\cdot\ddot{q}_i+\boldsymbol{R}_{i-1}\cdot{}^{i-1}\vec{\omega}_{i-1}\wedge({}^i\vec{z}_i\cdot\dot{q}_i) \tag{10}$$

$$\begin{aligned}{}^i\ddot{\vec{r}}_{O_i}=&{}^iR_{i-1}[{}^{i-1}\ddot{\vec{r}}_{O_{i-1}}+{}^{i-1}\dot{\vec{\omega}}_{i-1}\wedge({}^{i-1}\vec{\omega}_{i-1}\wedge{}^{i-1}\vec{r}_{O_{i-1},O_i})\\&+{}^{i-1}\dot{\vec{\omega}}_{i-1}\wedge{}^{i-1}\vec{r}_{O_{i-1},O_i}]+(1-\rho_i)[{}^i\vec{z}_i\cdot\ddot{q}_i+2({}^i\vec{\omega}_i\wedge{}^i\vec{z}_i\cdot\dot{q}_i)]\end{aligned} \tag{11}$$

$${}^i\ddot{\vec{r}}_{G_i}={}^i\ddot{\vec{r}}_{O_i}+{}^i\vec{\omega}_i\wedge({}^i\vec{\omega}_i\wedge{}^i\vec{r}_{O_i,G_i})+{}^i\dot{\vec{\omega}}_i\wedge{}^i\vec{r}_{O_i,G_i} \tag{12}$$

where ${}^i\vec{z}_i=[0\ 0\ 1]^T$, and the variable ρ_i allows us to distinguish between the revolute joints ($\rho_i=1$) and the prismatic ones ($\rho_i=0$).

Developing the A_j term:

$$A_j=\sum_{i=j}^{n}\left\{\left({}^iR_j\cdot\frac{\partial\,{}^j\dot{\vec{\omega}}_j}{\partial\ddot{q}_j}\right)^{T}\cdot[{}^i\boldsymbol{I}_{G_i}\cdot{}^i\dot{\vec{\omega}}_i+{}^i\vec{\omega}_i\wedge({}^i\boldsymbol{I}_{G_i}\cdot{}^i\vec{\omega}_i)]\right\} \tag{13}$$

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

To obtain the $\frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}$ term, we start from expression (10). This derivative could be obtained by a recursive procedure as follows:

$$\text{If } i<j \quad \frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}=[0\ \ 0\ \ 0]^{\mathrm{T}}$$

$$\text{If } i<j \quad \frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}={}^{i}\boldsymbol{R}_{i-1}\cdot {}^{i-1}\boldsymbol{R}_{i-2}\cdot \ldots \cdot {}^{j-1}\boldsymbol{R}_j\cdot \frac{\partial\, {}^{j}\dot{\vec{\omega}}_j}{\partial\, \ddot{q}_j} \tag{14}$$

$$\text{If } i=j \quad \frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}={}^{i}\vec{z}_i$$

Taking into account the expressions in (14), the A_j term could be rewritten as follows:

$$A_j=\left(\frac{\partial\, {}^{j}\dot{\vec{\omega}}_j}{\partial\, \ddot{q}_j}\right)^{T}\cdot \sum_{i=j}^{n}\{{}^{j}\boldsymbol{R}_i\cdot [{}^{i}\boldsymbol{I}_{G_i}\cdot {}^{i}\dot{\vec{\omega}}_i+{}^{i}\vec{\omega}_i\wedge ({}^{i}\boldsymbol{I}_{G_i}\cdot {}^{i}\vec{\omega}_i)]\} \tag{15}$$

In this last expression, it can be seen that there are concurrent terms which could reduce the calculation complexity. Next, an expression that allows the terms to be obtained in a backward recursive way is presented:

$$A_j=\left(\frac{\partial\, {}^{j}\dot{\vec{\omega}}_j}{\partial\, \ddot{q}_j}\right)^{T}\cdot {}^{j}\vec{\alpha}_j \tag{16}$$

where

$${}^{j}\vec{\alpha}_j={}^{j}\boldsymbol{I}_{G_j}\cdot {}^{j}\dot{\vec{\omega}}_j+{}^{j}\vec{\omega}_j\wedge ({}^{j}\boldsymbol{I}_{G_j}\cdot \vec{\omega}_j)+{}^{j}\boldsymbol{R}_{j+1}\cdot {}^{j+1}\vec{\alpha}_{j+1} \tag{17}$$

The B_j term can be obtained using the development of $\frac{\partial\, {}^{i}\ddot{\vec{r}}_{G_i}}{\partial\, \ddot{q}_j}$. This term comes from deriving the expression (12), leading to

$$\frac{\partial\, {}^{i}\ddot{\vec{r}}_{G_i}}{\partial\, \ddot{q}_j}=\frac{\partial\, {}^{i}\ddot{\vec{r}}_{O_i}}{\partial\, \ddot{q}_j}+\frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}\wedge {}^{i}\vec{r}_{O_iG_i} \tag{18}$$

The $\frac{\partial\, {}^{i}\ddot{\vec{r}}_{O_i}}{\partial\, \ddot{q}_j}$ term for revolute joints is obtained using the expression (11):

$$\frac{\partial\, {}^{i}\ddot{\vec{r}}_{O_i}}{\partial\, \ddot{q}_j}={}^{i}\boldsymbol{R}_{i-1}\cdot \left(\frac{\partial\, {}^{i-1}\ddot{\vec{r}}_{O_{i-1}}}{\partial\, \ddot{q}_j}+\frac{\partial\, {}^{i-1}\dot{\vec{\omega}}_{i-1}}{\partial\, \ddot{q}_j}\wedge {}^{i-1}\vec{r}_{O_{i-1},O_i}\right) \tag{19}$$

In a similar way, for prismatic joints we could obtain:

$$\frac{\partial\, {}^{i}\ddot{\vec{r}}_{O_i}}{\partial\, \ddot{q}_j}={}^{i}\boldsymbol{R}_{i-1}\cdot \frac{\partial\, {}^{i-1}\ddot{\vec{r}}_{O_{i-1}}}{\partial\, \ddot{q}_j}+{}^{i}\boldsymbol{R}_{i-1}\cdot \frac{\partial\, {}^{i-1}\dot{\vec{\omega}}_{i-1}}{\partial\, \ddot{q}_j}\wedge {}^{i-1}\vec{r}_{O_{i-1},O_i}+{}^{i}\vec{z}_i\cdot \frac{\partial\, \ddot{q}_i}{\partial\, \ddot{q}_j} \tag{20}$$

Notice that there is an additional term in expression (20) in relation with (19), which must be included if the i-th joint is a prismatic one:

$${}^{i}\vec{z}_i\cdot \frac{\partial\, \ddot{q}_i}{\partial\, \ddot{q}_j} \tag{21}$$

To develop the B_j term, considering expressions (18) and (19), the following expression is used

$$\frac{\partial\, {}^{i}\ddot{\vec{r}}_{G_i}}{\partial\, \ddot{q}_j}=\frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}\wedge {}^{i}\vec{r}_{O_{j-1},G_i}$$

This expression, when substituted in the above description of every B_j term, would give:

$$B_j=\sum_{i=j}^{n}\left\{\left(\frac{\partial\, {}^{i}\dot{\vec{\omega}}_i}{\partial\, \ddot{q}_j}\right)^{T}\cdot ({}^{i}\vec{r}_{O_{j-1},G_i}\wedge m_i\, {}^{i}\ddot{\vec{r}}_{G_i})\right\} \tag{22}$$

Applying vectorial products properties and using again expression (14), would give:

$$B_j=-\left(\frac{\partial\, {}^{j}\dot{\vec{\omega}}_j}{\partial\, \ddot{q}_j}\right)^{T}\cdot {}^{j}\vec{\beta}_j \tag{23}$$

where

$${}^{j}\vec{\beta}_j=\sum_{i=j}^{n}[{}^{j}R_i\cdot (m_i\, {}^{i}\ddot{\vec{r}}_{G_i}\wedge {}^{i}\vec{r}_{O_{j-1},G_i})] \tag{24}$$

This expression could be calculated in a recursive way as follows:

$${}^{j}\vec{\beta}_j=m_j\, {}^{j}\ddot{\vec{r}}_{G_j}\wedge {}^{j}\vec{r}_{O_{j-1},G_j}+{}^{j}\vec{\varphi}_j\wedge {}^{j}\vec{r}_{O_j,O_{j+1}}+{}^{j}R_{j+1}\cdot {}^{j+1}\vec{\beta}_{j+1} \tag{25}$$

where

$${}^{j}\vec{\varphi}_j={}^{j}R_{j+1}\cdot (m_{j+1}\, {}^{j+1}\ddot{\vec{r}}_{G_{j+1}}+{}^{j+1}\vec{\varphi}_{j+1}) \tag{26}$$

This formulation leads to an algorithm to solve the Inverse Dynamic Problem in robots of computational complexity of $O(n)$. The algorithm description and the analysis of its computational complexity will be shown now. It's composed of 6 steps (note that in order to compare the computational complexity with other algorithms, only revolute joints are considered and the robot base is considered fixed).

In step 1, there are the computation of the rotation matrices ${}^{i-1}R_i$ and translation vectors ${}^{i-1}\vec{r}_{O_{i-1},O_i}$. The velocities and accelerations are computed in step 2. Step 3 derives ${}^{i}\dot{\vec{\omega}}_i$ with respect to the generalised accelerations $\ddot{q}^i$, and the A_i and B_i terms are obtained in steps 4 and 5. Finally, step 6 computes the generalised forces τ_i.

The computational complexity of the proposed algorithm is summarised in Table II. Furthermore, the computational complexity of the Luh, Walker and Paul algorithm is reported in Table III.

As can be seen from Table III, the computational complexity of the proposed algorithm is very close to the

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

Table II. Proposed algorithm complexity.

Step	Complexity (×)	(+)
Step 1	4n	0
Step 2	62n − 75	46n − 61
Step 3	0	0
Step 4	24n − 23	18n − 18
Step 5	31n − 38	26n − 37
Step 6	0	n

Newton-Euler based algorithm. It is also remarkable that using the Denavit-Hartenberg notation under Paul's convention, no substantial differences are observed, being in this case the computational complexity $133n - 74$ (×) and $97n - 71$ (+) for a robot with only rotational joints, so that for a six degree of freedom robot, the computational complexity would be 724 (×) and 511 (+).

3. A PROPOSED CONTROL ARCHITECTURE FOR INDUSTRIAL ROBOTS

Increasing demands on the robot systems performance has led to the development of advanced control methods. If the robotic system has an open control stage, the implementation and the use of any of the control methods are very easy since it can use any high level programming language (for instance, the *C* language has been used to program the new control system). In this work a new control architecture was used. This control unit is based on PC, and was implemented to control a 6-joint industrial robot, the PUMA 600.

The original control module consists of one processor LS-11/02, which interprets VAL-II statements and generates trajectories that are sent to the six digital servos (each of them containing a Rockwell 6503 processor). These digital servos generate analog signals through a series of Digital to Analog converters, which are sent to the amplifiers panels connected to the arm. The control loop is closed with the potentiometers and the encoders. Each robot joint has associated a potentiometer that gives the joint absolute position, while the signals of the incremental encoders (200 counts per revolution for the 2 first joints and 250 for the remainder) provide a more precise motion measurement.

In order to avoid the original control unit limitations (high level tasks implementation, modification of the control algorithm, etc.), the control stage has been substituted by a module based on PC. In this way the PC has access to the positions and generates adequate control actions in order to move the different elements of the robot. This proposed architecture is specified in Figure 2.

Table III. Inverse dynamics controllers implemented.

Algorithm		Complexity	***n*=6**
Luh, Walker y Paul	(×)	$121n - 112$	614
	(+)	$90n - 82$	458
Proposed algorithm	(×)	$121n - 136$	590
	(+)	$91n - 116$	430

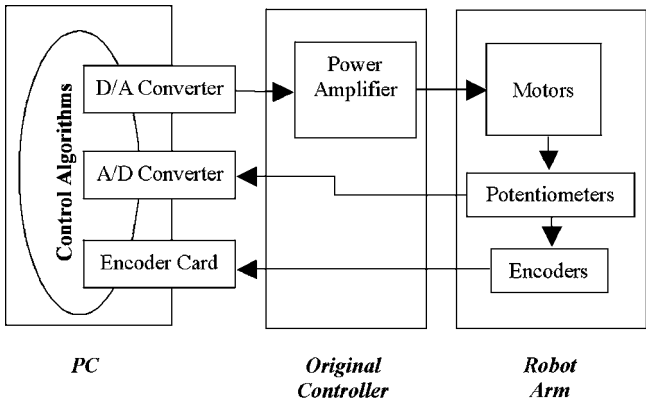

Fig. 2. New control architecture used in this work.

In order to implement this control architecture, four data acquisition cards have been used: a card, *Advantech™ PCL-818*, has been used in order to obtain the analog outputs; another one, *PCL-726*, has been used for supplying the control actions; and the two others, *PCL-833*, have been used for reading encoders. The proposed control architecture has two advantages: First, the simplicity, and second the low cost (the total cost of all these cards doesn't exceed $2.000). In addition, this open architecture gives a powerful platform for programming a higher level tasks.

Technically this control architecture works as follows:

- Analog inputs: These are the potentiometers signals that give the absolute position of each robot joint. Since there are six elements in the robot it is necessary to have 6 analog inputs to obtain the potentiometers voltages (0–5volts). These analog inputs are provided by the PCL-818 card.
- Digital inputs: The robot supplies some *thermo signals* which indicate the joint motors over-temperature. These signals are digital, so it is necessary to have 6 digital inputs to read them. These digital inputs are also provided by the PCL-818 card.
- Encoder signals: The encoders used by the robot are incremental encoders, and each of them provides three signals: channel A, channel B and index pulse. Since each PCL-833 card can control 3 encoders, 2 cards are needed in order to obtain the robot position.
- Analog outputs: Once the 6 control actions have been calculated by the control strategy, it will be necessary to make a digital to analog conversion. Although the power amplifiers of the robot could be fed with 12 volts, we had limited it to 10 volts due to the robot and the PCL-726 card features, therefore 6 channels of the PCL-726 card are used.
- Digital outputs: In order to activate the robot tool it is necessary to have 2 digital outputs. These digital outputs are also provided by the PCL-726.

4. INVERSE DYNAMIC CONTROL

With the control architecture depicted previously some controllers, based on inverse dynamic method, have been implemented. This control approach provides a regular static state feedback that transforms the nonlinear system

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

into a linear one (this is known as the inverse dynamics or feedback linealization problem). Potentially this technique is very useful because it reduces the nonlinear control problem to the control problem of a linear system, for which many tools are available. Assuming the dynamic model as:

$$x^{(n)}=f(x)+b(x)u$$

where $f(x)$ is a nonlinear state function and u is the control input. If we use as the control input the expression:

$$u=\frac{1}{b}[v-f] \tag{27}$$

the nonlinearities will be cancelling, and the simple input-output relation will be obtained:

$$x^{(n)}=v$$

where $\boldsymbol{v}$ is a new input vector to be defined below. In the robot case, the dynamic model can be expressed by the equation:

$$\tau=C(q,\dot{q})\dot{q}+G(q)+M(q)\ddot{q} \tag{28}$$

where $C(q,\dot{q})$ is the vector of centrifugal and Coriolis terms, $G(q)$ is the vector of gravity terms, and $M(q)$ is the mass matrix. Working with this equation:

$$\ddot{q}=M^{-1}(q)(\tau-C(q,\dot{q})\dot{q}-G(q))$$

we define the terms:

$$f(x)\equiv M^{-1}(q)(-C(q,\dot{q})\dot{q}-G(q))$$
$$b(x)\equiv M-1(q)$$

using the general expression (27):

$$\tau_c=\frac{1}{M^{-1}(q)}[v-M^{-1}(q)(-C(q,\dot{q})-G(q))]$$

the controllers based on the inverse dynamics could be viewed such as particular cases of the general control law:

$$\tau_c=C(q,\dot{q})\dot{q}+G(q)+M(q)v \tag{29}$$

The inverse dynamics control (29) shows how the non-linearities, such as Coriolis terms as well as gravity terms, can be simply compensated by adding these forces to the control input (Table IV). Depending on the v expression, different controllers can be obtained:[16]

The first controller implemented was the point to point controller. In this case proportional and derivative terms compose the linear auxiliary control input v, and the robot system is exponentially stable by a suitable choice of the matrices K_d and K_p.

The second controller is very similar to the first, but in this case the robot must follow a given time-varying trajectory $q_d(t)$ and its successive derivatives $\dot{q}_d$ and $\ddot{q}_d$ which, respectively, describe the desired velocity and acceleration. This tracking control is very simple but it has several drawbacks: any error in the robot dynamics estimation can cause a variation in the equilibrium point and therefore a position error. The second problem that can occur is related to the dead-zone phenomenon; in this case the static friction at the motor shafts can also provoke a position error. A practical solution to attempt to solve these problems is to insert an integral action in the control law. This is the case of the last controller, where it has been added the integral of the error.

All these controllers need the dynamic elements of the robot. In this work we used the Gibbs-Appell equations presented before to calculate the control action because it is a very efficient way to calculate these elements. In this way, some functions were programmed to calculate with the A_j term the mass matrix $M(q)$ using equation (15), to obtain the vector bias $C(q,\dot{q})\dot{q}+G(q)$ employing the B_j term of equation (23), etc.

On the other hand, because the robot parameters can vary along the time (deformation or wear on the elements, robot payload etc.), in order to implement the robot control, a precise real robot dynamic model is required. A dynamic parameters identification can be obtained by rewritten robot equations:

$$\tau_m=K_m(q,\dot{q},\ddot{q})\cdot\Phi_m$$

where τ_m is the generalized torques, K_m is the observation matrix, and Φ_m is the robot parameters

$$\Phi_m=[m_m\ \ mc_{x_m}\ \ mc_{y_m}\ \ mc_{z_m}\ \ I_{xx_m}\ \ I_{xy_m}\ \ I_{xz_m}\ \ I_{yy_m}\ \ I_{yz_m}\ \ I_{zz_m}]$$

Parameters vector Φ_i can be obtained using less squares or any other matrix technique. In our work, QR decomposition, Singular Values decomposition, pseudo-inverse calculation or Ridge regression have been implemented. In this way, the robot parameters used are shown in Tables V, VI and VII.

Table IV. Computational complexity for robots with $n\geq 3$.

Controller	v
Point to point control	$-K_d\dot{q}-K_pe$
Tracking Control	$\ddot{q}_d-K_d\dot{q}-K_pe$
Tracking Control with integral action	$\ddot{q}_d-K_d\dot{q}-K_pe-K_i\int_0^t e(u)du$

Table V. Denavit-Hartenberg parameters.

Joint	$\alpha(rad)$	a(m)	D(m)
1	0	0	0
2	$-\pi/2$	0	0
3	0	0.432	-0.15
4	$\pi/2$	-0.02	-0.433
5	$-\pi/2$	0	0
6	$\pi/2$	0	0

Table VI. Masses of the link.

Joint	Mass (kg)
1	10.521
2	15.781
3	8.767
4	1.052
5	1.052
6	0.351

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

Table VII. Coordinates in a local reference system.

Link i	Centre of masses (m) ${}^{i}\vec{r}_{O_i G_i}$
1	$[0.0 \; -0.054 \; 0.0]^T$
2	$[0.1398 \; 0.0 \; 0.1491]^T$
3	$[-0.32.10^{-3} \; -0.197 \; 0.0]^T$
4	$[0.0 \; 0.0 \; -0.057]^T$
5	$[0.0 \; -0.007 \; 0.0]^T$
6	$[0.0 \; 0.0 \; 0.0372]^T$

The inertial tensor of links ($kg.m^2$) defined with respect to parallel axes to the local links and passing through their centre of masses is:

$$ {}^{1}I_{G_1} = \begin{bmatrix} 1.612 & 0 & 0 \\ 0 & 0.5091 & 0 \\ 0 & 0 & 1.612 \end{bmatrix} $$

$$ {}^{2}I_{G_2} = \begin{bmatrix} 0.4898 & 0 & 0 \\ 0 & 8.0783 & 0 \\ 0 & 0 & 8.2672 \end{bmatrix} $$

$$ {}^{3}I_{G_3} = \begin{bmatrix} 3.3768 & 0 & 0 \\ 0 & 0.3009 & 0 \\ 0 & 0 & 3.3768 \end{bmatrix} $$

$$ {}^{4}I_{G_4} = \begin{bmatrix} 0.181 & 0 & 0 \\ 0 & 0.181 & 0 \\ 0 & 0 & 0.1273 \end{bmatrix} $$

$$ {}^{5}I_{G_5} = \begin{bmatrix} 0.0735 & 0 & 0 \\ 0 & 0.0735 & 0 \\ 0 & 0 & 0.1273 \end{bmatrix} $$

$$ {}^{6}I_{G_6} = \begin{bmatrix} 0.0071 & 0 & 0 \\ 0 & 0.0071 & 0 \\ 0 & 0 & 0.0141 \end{bmatrix} $$

Before the implementation of the new control unit, some simulations have been carried out with the symbolic algebra software MACSYMA. In the next example, a straight-line trajectory has been considered. The start point has coordinates $[0.6 \; 0.175 \; 0.250]^T m$ and the end point $[0.018 \; 0.757 \; 0]^T m$, the constant orientation given by Euler angles ZYZ is $(45° \; 60° \; 90°)$. The constant linear velocity prescribed for the robot end-effector is $0.1 m/s$, the total time needed for the prescribed robot motion is 8.6 seconds (s). In Figure 3 the initial configuration of the robot, an intermediate and the final one are depicted. The torques required in each joint are depicted in Figure 4 as a function of time.

Figure 5 shows the positions of the real robot controlled by the inverse dynamic strategies with a ramp and splines references. In both cases the joint positions follow without problems the input references. Several types of computers have been used to obtain the execution times of these control algorithms. If a Pentium 150Mhz. is employed, the execution time was 0.225 ms, and with a Pentium II 350Mhz. the time required was 0.061 ms.

5. CONCLUSIONS

This paper has shown the implementation of the inverse dynamics controllers using an open control system for the PUMA industrial robot arm. Due to the components used (PC and conventional data acquisition cards), the control stage is very economic and flexible. This flexibility allows, for instance, programming and comparing advanced control strategies, control algorithms based on artificial vision, as well as integrating the PUMA in a flexible manufacturing system, etc.

In this open control unit, several algorithms based on the Gibbs-Appell equations have been proposed and verified for solving the Inverse Dynamic Problem and the control problem. These algorithms have computational complexities slightly lower than the algorithm based on the Newton-Euler equations of motion, formulated in a similar way, and using mainly vectors in its recursive formulation. This fact confirms the conclusion of other authors who claim that the efficiency of the dynamic algorithms arises from the type of formulation used, rather than the Principle of Dynamics considered. In this way, it can be expected that further reductions in computational complexity may be

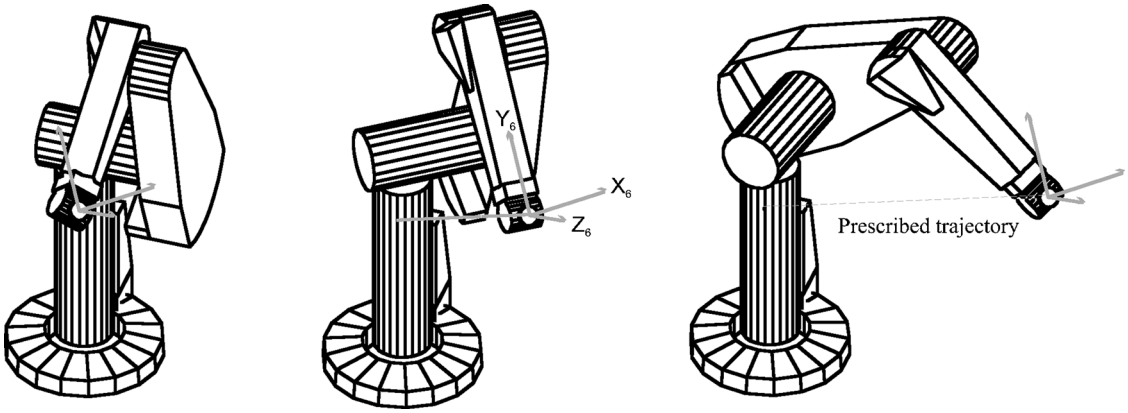

Fig. 3. Prescribed trajectory for the Puma robot.

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

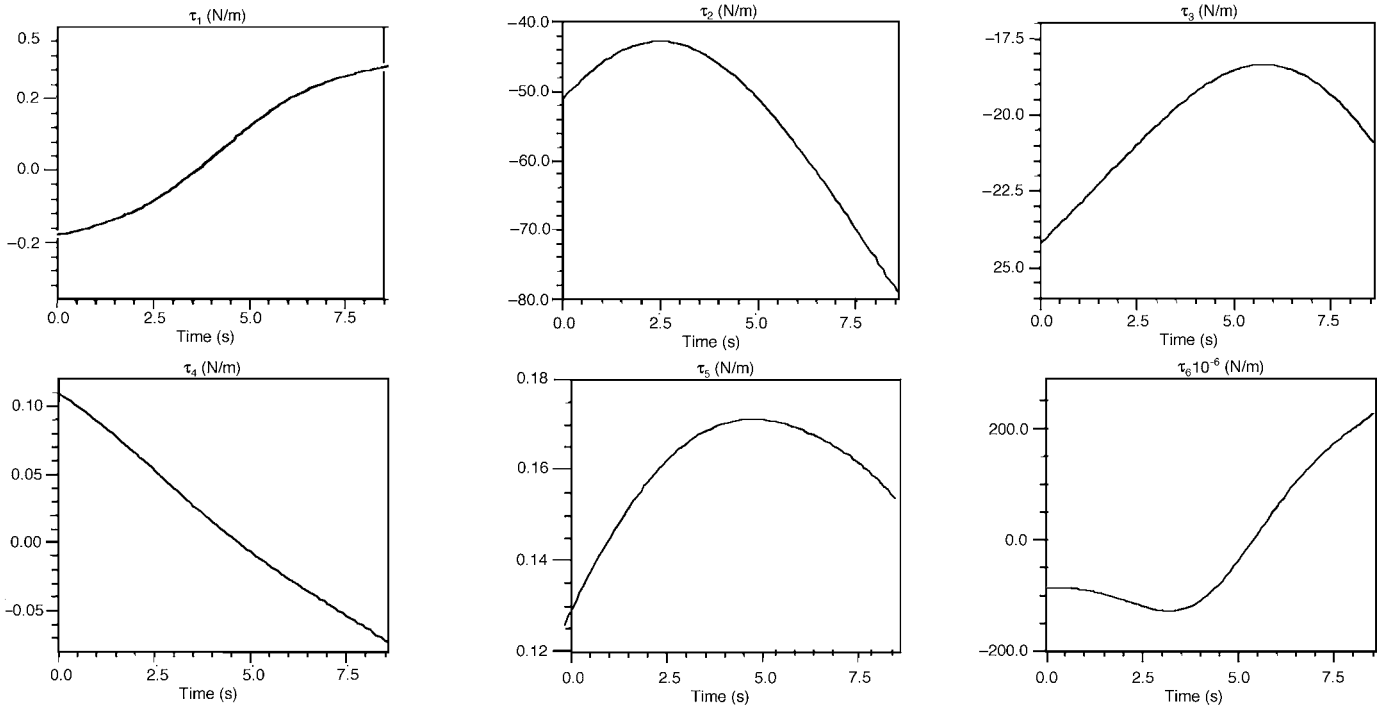

Fig. 4. Torques required per joint.

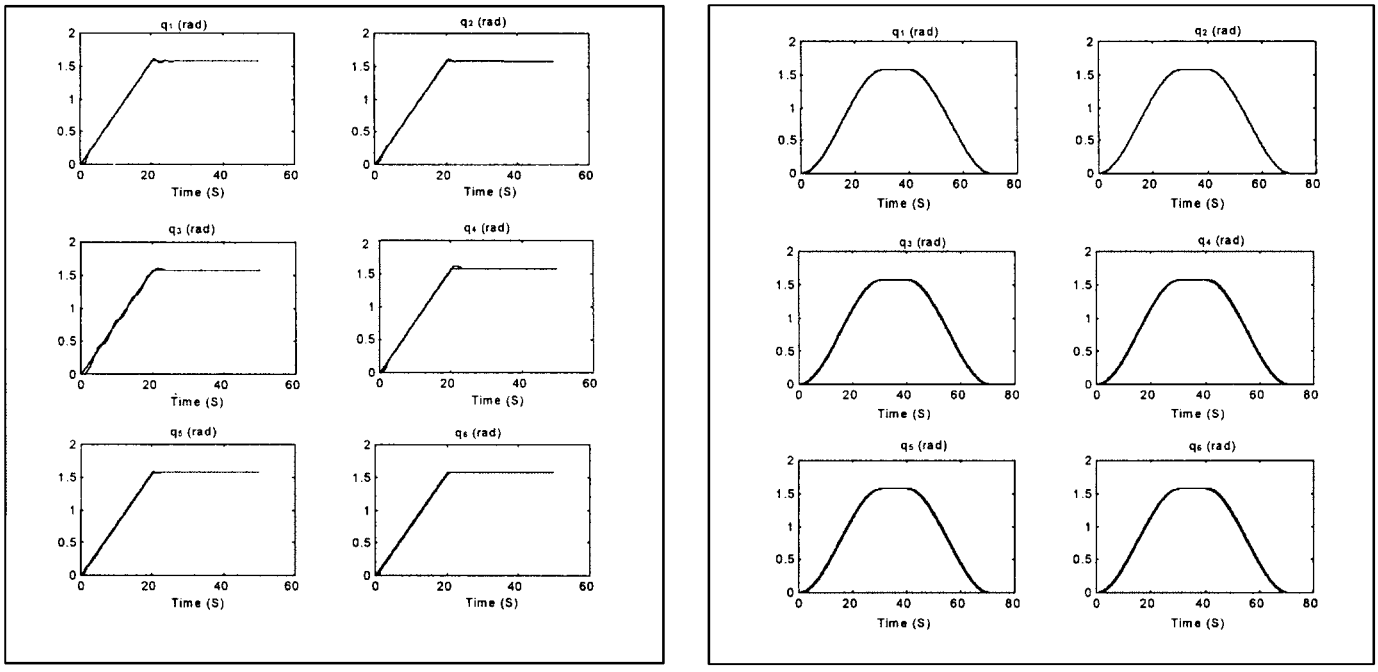

Fig. 5. Real robot positions for ramp and splines references.

achieved by using tensorial notation rather than vectorial notation. For instance, important savings could be obtained developing the term corresponding to the Moment of Inertia (Euler equation) in the *A* terms in a tensorial form.

The paper showed the robot response in the simulation and in the real execution, working with the point to point and the tracking problem. Several aspects of the overall robot response can be analysed easily. In addition, the PC environment has allow us to connect the data structure with commercial CADCS packages, such as MATLAB, MATHEMATICA etc.

References

1. N. Moreira, P. Alvito and P. Lima, "First steps towards an open control architecture for a PUMA 560", *Proc. 2nd Portuguese Conf. on Automatic Control* Porto, Portugal (1996) pp. 625–630.
2. A. Valera, J.V. Vergara, J. Tornero and E. García, "Control a PUMA 500 Using a New Open Architecture", *Proc. 3rd*

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press

Portuguese Conf. on Automatic Control Coimbra, Portugal (1998) pp. 795–798.
3. W. Khalil and J-F. Kleinfinger, "Minimum Operations and Minimum parameters of the Dynamic Models of Tree Structure Robots", *IEEE J. of Robotics and Automation* **3**(6), 517–526 (1987).
4. C.S.G. Lee and P.R. Chang, "Efficient Parallel Algorithm For Robot Inverse Dynamics Computation", *IEEE Trans. on Systems, Man, and Cybernetics* **16**(4), 532–542 (1986).
5. J.J. Murray and C.P. Newman "Organizing Customized Robot Dynamics Algorithms for Efficient Numerical Evaluation", *IEEE Trans. on Systems, Man, and Cybernetics* **18**(1), 115–125 (1988).
6. C.A. Balafoutis and R.A. Patel, *Dynamic Analysis of Robot Manipulators: A Cartesian Tensor Approach* (Kluwer Academic Press, 1991).
7. J.M. Hollerbach, "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity", *IEEE Trans. on Systems, Man, and Cybernetics* **SMC-10**(11), 730–736 (1980).
8. J.Y.S. Luh, M.W. Walker and R.P.C. Paul, "On-line Computational Scheme for Mechanical Manipulators", *J. Dynamic Systems, Measurement, and Control* **102**, 69–76 (1980).
9. K.S. Fu, R.C. Gonzalez and C.S.G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence* (McGraw-Hill, 1987).
10. A.Y. Zomaya, *Modelling and Simulation of Robot Manipulators: A Parallel Processing Approach* (World Scientific Publishing Co., Singapore, 1992).
11. J.J. Craig, *Introduction to Robotics: Mechanics and Control* (Addison-Wesley, Reading, 1986).
12. M. Renaud, "Contribution a l'etude de la modélisation et de la commande des systèmes mécaniques articulés" *Ph.D. Thesis* (Université Paul Sabatier, Toulouse, 1975).
13. M. Vukobratovic and N. Kircanski, *Real-Time Dynamics of Manipulation Robots* (Springer-Verlag, Berlin, 1985).
14. K. Desoyer and P. Lugner, "Recursive formulation for the analytical or numerical application of the Gibbs-Appell method to the dynamics of robots", *Robotica* **7** 343–347 (1989).
15. J. Angeles, O. Ma and A. Rojas, "An Algorithm for the Inverse Dynamics of n-Axis General Manipulators Using Kane's Equations", *Computers Math. Applic*. **17**(12), 1545–1561 (1989).
16. A. Valera, *Análisis Comparativo de Técnicas de Control de Robots Rígidos y Flexibles* (Editorial de la Universidad Politécnica de Valencia, Valencia, 2000).

https://doi.org/10.1017/S0263574702004769 Published online by Cambridge University Press