TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

An adaptive linearization and control strategy for mechanical manipulators

Citation for published version (APA):
Mazure, P. F. M. (1987). *An adaptive linearization and control strategy for mechanical manipulators*. (TH Eindhoven. Afd. Werktuigbouwkunde, Vakgroep Produktietechnologie : WPB; Vol. WPA0461). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1987

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 27. Mar. 2025

BB 995247

AN ADAPTIVE LINEARIZATION AND CONTROL STRATEGY FOR MECHANICAL MANIPULATORS

P.J.M. MAZURE Université de
Technologie de Compiègne
France

March - July 1987
WPA-Rapport nr.WPA-0461

- FOREWORD -

Because several persons will work after me on the same subject and will continue my work, I had to develop the theory and put a lot of details in my report, especially all the formulae I have used to develop the software.

In fact, the aim of this report is to render more convenient the task of the people who will follow me. For this reason, the thoroughness of this report should not surprise my readers.

Note also that it was done with the agreement of my coach Dr. L. Loron from the Compiegne University of Technology.

acknowledgements

I want to express my gratitude to my coach, Mr Henk Smit, who with his competence and kindness, contributed to the progress of my training period.

Greatest thanks to P.C. Mulders, who welcomed me into his team, Leon Pijls, Arthur Udo, Willy Smits, Gerard Kreffer, Henk Van Rooij and also to the staff of the International Relations/external training projects for the perfect organization of the traineeship, especially to IR. P.J. Corzilius, MW. T.H. den Haan and MW. P.M.M. van Amelsvoort.

I also want to thank L. Loron from the Compiègne University of Technology who visited me and showed me his interest in the work I was doing.

Because of the work atmosphere and the different way of life in the Netherlands, I will never forget my traineeship in this nice country ...

Dank je wel
en tot ziens

Eindhoven, July 1987

Pascal [illegible]

Contents

§ 1. Summary ..... Page..... 5

§ 2. The work environment..... 7

§ 2.1. The engineering education in Netherlands..... 7

§ 2.2. The Eindhoven University of Technology..... 7

§ 2.3. The Mechanical Engineering Department..... 9

§ 2.4. The F.L.A.I.R. project..... 10

§ 2.5. My training period project..... 10

§ 3. Robot arm dynamics..... 12

§ 3.1. Newton-Euler formulation of equations of motion..... 13

§ 3.1.1. Basic dynamic equations..... 13

§ 3.1.2. Closed-form dynamic equations..... 14

§ 3.2. Lagrangian formulation of manipulator dynamics..... 16

§ 3.2.1. Lagrangian dynamics..... 16

§ 3.2.2. The manipulator inertia tensor..... 17

§ 3.2.3. Deriving Lagrange's equations of motion..... 19

§ 3.3. Inverse dynamics..... 21

§ 3.3.1. Recursive computation..... 22

§ 3.3.2. Moving coordinates..... 25

§ 3.3.3. Luh-Paul-Walker's algorithm..... 27

§ 4. Adaptive control for robot manipulators 32

§ 4.1. The feedforward component 33

§ 4.2. The feedback component 36
§ 4.2.1. Derivation of the state space representation... 36
§ 4.2.1.1. Perturbation equations of motion 36
§ 4.2.1.2. Derivation of the system parameters A(t),B(t) 38
§ 4.2.1.2.1. Derivation of the Jacobian matrix M.... 39
§ 4.2.1.2.2. Derivation of the Jacobian matrix C.... 41
§ 4.2.1.2.3. Derivation of the Jacobian matrix K.... 45

§ 5. Dynamic simulation 51

§ 5.1 Inclusion of nonrigid body effects 51

§ 5.2 The simulation itself 52
§ 5.2.1.Technique for solving for the joint acceleration 54
§ 5.2.2.The different steps to test the simulation..... 56

§ 6. The Software 58

§ 7. About the work atmosphere 59

§ 8. Conclusion 60

§ 9. Appendix 61

Appendix Contents

1. Résumé en français 62

2. About PC-MATLAB 63

3. Listing of the softwares 68

4. Bibliography 105

§.1. Summary

Industrial robots are mechanical manipulators whose dynamics characteristics are highly nonlinear. A purpose of manipulator control is to maintain a prescribed motion for the manipulator along a desired time-based trajectory by applying corrective compensation torques to the actuators to adjust for any deviations of the manipulator from the trajectory.

There is a lot of possibilities to control a robot arm manipulator, but most of the existing control schemes control the manipulator at the joint level or the hand level and emphasize nonlinear compensations of the interaction forces among the various joints. These control are inadequate because they neglect the changes of the load in the task cycle. These changes are significant enough to render conventional feedback control strategies ineffective. Any significant performance gain in robot arm control require the consideration of adaptive control techniques. To control a manipulator which carries a variable or unknown load and moves along a planned path, it is required to compute the forces and the torques needed to drive all its joints accurately and frequently at an adequate sampling frequency (no less than 60 Hz since the resonant frequency of most of the mechanical manipulators are around 10 Hz). That necessity is in fact the bottleneck of a lot of control strategies because of the number of calculation. The aim is to find faster algorithms and also to use all the possibilities of the existing computers (one tendency is the development of concurrent programming for instance).

The chapter § 3.of this report presents a new approach of computation based on the method of Newton-Euler formulation which is independent of the manipulator configuration. This method involves the successive transformation of velocities and accelerations from the base of the manipulator out to the gripper, link by link, using the relationships of moving coordinate systems. Forces are then

transformed back from the gripper to the base to obtain the joint torques. Theoreticaly the mathematical model is "exact"and a further advantage of using this method is that the amount of computation increases linearly with the number of links whereas the conventional method based on the Lagrangian formulation increases as the quartic of the number of links.

The chapter § 4. of this report deals with an adaptive control for a robot manipulator. The proposed adaptive control is based on the linearization of the recursive formulation of Luh's Newton-Euler equations of motion. This idea from Henk Smit allows us to obtain the state matrices A(k) and B(k). When the system parameters are known the regulator matrix is determined with a software like PC-MATLAB.

The total torques applied to the joint actuators then consist of the nominal torques computed from the Newton-Euler equations and the variational torques computed from the feedback component.

§.2. The work environment

The Technische Universiteit Eindhoven (T.U.E, Eindhoven University of Technology) has been founded in 1956. The former name of this school was : Technische Hogeschool Eindhoven (T.H.E)

§.2.1. The engineering education in the Netherlands

Engineering education at University level in the Netherlands is concentrated in separate institutes named 'Universiteit'; at this type of institute only academic engineering degrees are conferred. There are three institutes of that name :

T.H.Delft, founded in 1842,
T.H.Eindhoven, founded in 1956,
T.H.Twente, founded in 1961.

They group about 20.000 students.

The first degree which terminates a course of studies is the Master's degree. You have three types of Master : Ingenieur (Ir.) for technical studies, Meester (Mr.) for law studies and Doctorandus (Drs.) for the rest. In engineering subjects, the Master's degree entitles the graduate to use legally protected abreviation 'Ir.' (Ingenieur in dutch) before his name.

To obtain the Doctor degree (Dr.) no normal course of study is scheduled. The candidate must have an engineer's degree, and to carry out original research.

§.2.2. The Eindhoven University of Technology

The T.U.E offers nine courses of study in which students can qualify as graduate engineers specializing in the following fields :

- Technology in its Social Applications
- Industrial Engineering and Management Science
- Mathematics
- Computer Science

- Technical Physics
- Mechanical Engineering
- Electrical Engineering
- Chemical Engineering
- Architecture, Structural Engineering and Urban Planning

It includes about 6.000 students with only 7% of girls.

Since the University opened, about 8.000 students have graduated from it. The operating costs was in 1985 250 million guilders.

Five years ago, a new system of studies was introduced by the dutch government. It is not very well received by all students.

The former system gave the students almost complete freedom to study or not study. They could choose more or less freely the semester in which they wanted to take a particular examination.
This system allowed students to manage their own time and to spent time in out of school activities like :

jobs with or without any educational interest, practicing sports, managing the students association. Each department has its own students association and their goal is to help students in obtaining extra facilities such as excursions to national companies, annual excursions to foreign countries (Japan, Norway, Germany, Belgium, France...), organizing international student meetings and many others.

The nominal duration of the studies was five years, but averagely the students remained at university about seven years

The new system decreases the freedom of the students and shortens the studies duration to four years, with a maximum of six years. It is said to be difficult and to require generally an extra year. According to that, students under this system do not take so much time either for many external things, nor for extra training periods. Their elders are afraid about how the associative student life will decrease and about the University level. After spending five months at the T.U.E., the students I know are in a very large scale proportion under the former system.

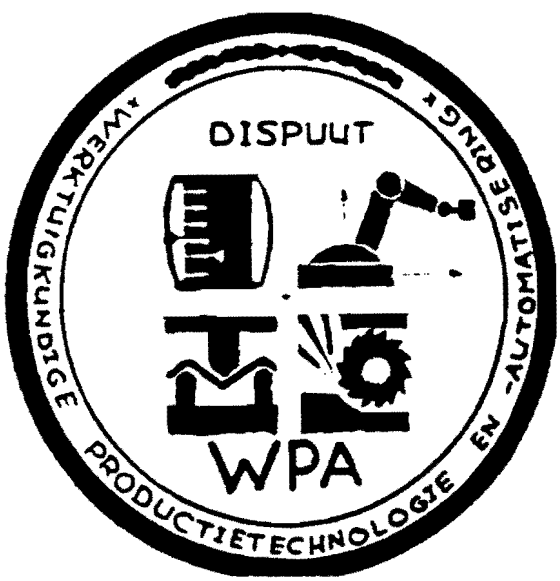
§.2.3. The Mechanical Engineering Department

Design and production are the two main groups into which the highly varied tasks of mechanical engineers are divided. The nature of the tasks carried out by the mechanical engineers varies from scientific research and development to industrial organisation.

Next to their theoretical knowledge, mechanical engineers must have specific practical skills. To this end, the curriculum includes amongst other things, participation in the work done by the department in its four divisions :

- Fundamentals of mechanical engineering
- Product design and development
- Design for industrial processing
- Production engineering and production automation (W.P.A).

The mechanical department has about two hundred employees (teachers and technical staff) and nine hundred students. I have worked and lived for five months among them, in the W.P.A. division.



Vakgroep Produktietechnologie en -Automatisering

tu/e

Technische Universiteit Eindhoven

§.2.4. The F.L.A.I.R. project (Flexible Automation and Industrial Robots)

The research project F.L.A.I.R. is financed and directed by the Dutch government. Its aim is to get some experience in flexible automation and industrial robot systems.

In this project each University has its own task. At the University of Eindhoven, this project is called F.A.I.R.

Both the mechanical and electrical departments are involved in this project as well as several private companies.

The project is divided in five sections :

- The general aspect of automation
- The handling of parts
- Kinematics and dynamics of mechanical structures
- The drive systems, the control systems and applications of those systems
- The arc-welding and the sensory system

§.2.5. My training period project

After the development of a one axis linear robot arm, a new subject of research was defined in the laboratory; the aim of this new subject is to get more experience in robot control and to use parallel computers.

The subject is defined as follows :

Study adaptative control for robot manipulators and the possibility to use state space control techniques.

Design a robot control based on state space techniques for a 2 or 3 dimensional robot arm (and consider possible parallallism in this control).

Simulate a 2 or 3 degrees of freedom robot-arm by implementing its dynamic behavior in a PC-Matlab program, test the robot control with this simulation program with relation to model inaccuracies (e.g friction, changing payloads,dynamic behaviour of motor,...).

Thus, my work was included in this huge subject. During the first six weeks my coach gave me general books about robotics to read in order to obtain the basis which are necessary to perform my task

First I studied :

Spacial descriptions and transformations.
Manipulators Kinematics.
Manipulators Statics.
Manipulators Dynamics.
Trajectory generation and control.
Position control of manipulators.
Force control of manipulators.
Compliant motion control.

After this first step, I studied more specifics articles about :

Adaptative control for robot manipulators
Dynamic computer simulation of robotic mechanisms
On-line computational scheme...

I decided in collaboration with my coach, to implemente a dynamic simulation software of a multi-dimensional robot.

§.3. Robot arm dynamics :

In this chapter, we analyze the dynamic behavior of manipulator arms. The dynamic behavior is described in terms of the time rate of change of the arm configuration in relation to the joint torques exerted by the actuators. This relationship can be expressed by a set of differential equations, called *equations of motion*, that govern the dynamic response of the arm linkage to input joint torques.

The dynamics of an n-joint manipulator can be derived either by *Lagrange-Euler (L.E)*, *Newton-Euler (N.E)*, *Recursive Lagrangian*, or *Generalized d'Alembert formulations*. Two main approaches, the Lagrange-Euler and the Newton-Euler formulations are described here to obtain two different forms of equations of motion suitable for developing the proposed simulation of an adaptive control strategy.

The Newton-Euler formulation is derived by the direct interpretation of Newton's Second Law of Motion, which describes dynamic systems in terms of force and momentum. The equations incorporate all the forces and moments acting on the individual arm links, including the coupling forces and moments between the links. The equations obtained from the Newton-Euler method include the constraint forces acting between adjacent links. Thus, additional arithmetic operations are required to eliminate these terms and to obtain explicit relations between the joint torques and the resultant motion in terms of joint displacements.

In the Lagrangian formulation, on the other hand, the system's dynamic behavior is described in terms of work and energy using generalized coordinates. All the workless forces are automatically eliminated in this method. The resultant equations are generally compact and provide a closed-form expression in terms of joint torques and joint displacements. The derivation is simpler and more systematic than in the Newton-Euler method.

The manipulator's equations of motion are basically a description of the relationship between the input joint torques and the output motion, i.e. the motion of the arm linkage. As in kinematics and in statics, we need to solve the inverse problem of finding the necessary input torques to obtain a desired output motion. Recently, efficient algotithms have been developed that allow the dynamic computations to be carried out on-line in real time. We will see one of them in the chapter § 3.3.3.

§.3.1. Newton-Euler formulation of equations of motion

§.3.1.1. Basic dynamic equations

The dynamic equations of a rigid body can be represented by two equations : one describes the translational motion of the centroid, while the other describes the rotational motion about the centroid.The former is the Newton's equation of motion for a mass particle, and the latter is caled Euler's equation of motion.

For any given link i, and according to the figure 3.1, we derive the two equations :

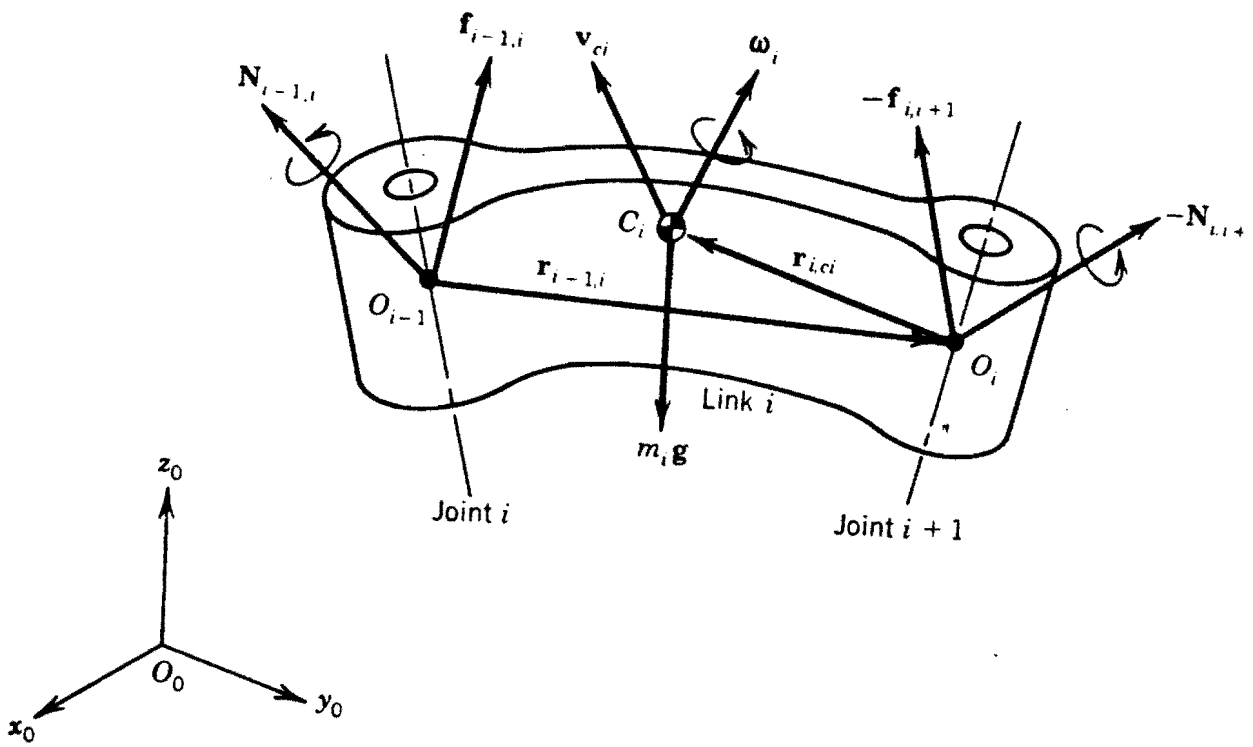


Figure 3.1 *Free body diagram of link i*

$$F_{i-1,i} - F_{i,i+1} + M_i g - M_i \dot{v}_{ci} = 0 \qquad (1)$$

$$N_{i-1,i} - N_{i,i+1} + r_{i,ci} \Lambda F_{i,i+1} - r_{i-1,ci} \Lambda F_{i-1,i} - I_i \dot{\omega}_i - \omega_i \Lambda (I_i \omega_i) = 0 \qquad (2)$$

$F_{i-1,i}$ is the coupling force applied to link i by link i-1,
$N_{i-1,i}$ is the moment applied to link i by link i-1,
$F_{i,i+1}$ is the coupling force applied to link i+1 by link i,
$N_{i,i+1}$ is the moment applied to link i+1 by link i,
M_i is the mass of the link,
I_i is the centroidal inertia tensor of link i
v_{ci} is the linear velocity of the centroid of link i,
ω_i is the angular velocity vector
g is the acceleration of gravity,

Note that all the vectors are defined with the reference to the base coordinate frame O_o-$x_o y_o z_o$.
And as the inertia tensor is :

$$I_i = \begin{bmatrix} \int\{(y-y_c)^2+(z-z_c)^2\}\rho dV & -\int(x-x_c)(y-y_c)\rho dV & -\int(z-z_c)(x-x_c)\rho dV \\ -\int(x-x_c)(y-y_c)\rho dV & \int\{(z-z_c)^2+(x-x_c)^2\}\rho dV & -\int(y-y_c)(z-z_c)\rho dV \\ -\int(z-z_c)(x-x_c)\rho dV & -\int(y-y_c)(z-z_c)\rho dV & \int\{(x-x)^2+(y-y_c)^2\rho dV \end{bmatrix}$$

Where ρ is the mass density, x_c, y_c, z_c are the coordinates of the centroid of the link. It is obvious that the inertia tensor varies with the orientation of the rigid body.

Equations (1)§(2) govern the dynamic behavior of an individual arm link. The complete set of equations for the whole manipulator arm is obtained by evaluating both equations for all the arm links, i=1....n.

§.3.1.2. Closed-form dynamic equations

The main problem is that the Newton-Euler equations are not in an appropriate form for use in dynamic analysis and controller design. They do not explicitly describe the input-output relationship. To derive explicit input-output dynamic relations, we need to separate the input joints torques from the constraint forces and moments.

Individual link motions are not independent, they must satisfy certain kinematic relationships to conform to the geometric constraints. Thus individual centroid position variables are not appropriate for output variables since they are not independent.

The appropriate form of the dynamic equations therefore consists of equations described in terms of all independent position variables and input forces, i.e. joint torques, that are explicitly involved in the dynamic equations. Dynamic equations in such an explicit input-output form are referred to as *closed-form dynamic equations* . In this context, "closed-form" means a solution method based on analytic expressions or on the solution of a polynomial of degree 4 or less, such that noniterative calculations suffice to arrive at a solution. As joint displacements q are a complete and independent set of generalized coordinates that locate the whole arm linkage, and as joint torques τ are a set of independent inputs that are separated from constraint forces and moment, dynamic equations in terms of joint displacements q and joint torques τ are a closed-form dynamic equations.

After arithmetic manipulations, the closed-form dynamic equations of a n-degree-of-freedom manipulator can be given in the form :

$$\tau_i = \sum_{j=1}^{n} H_{ij}\,\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}\,\dot{q}_j\,\dot{q}_k + G_i \qquad i=1,\ldots,n \qquad (3)$$

or $$\tau(t) = H(q)\,\ddot{q}(t) + V(q,\dot{q}) + G(q) \qquad (3')$$

where, τ is an nx1 applied torque vector to joint actuators,

$q(t)$ is the angular positions,

$\dot{q}(t)$ is the angular velocities,

$\ddot{q}(t)$ is an nx1 acceleration vector,

$G(q)$ is an nx1 vector of gravity,

$V(q,\dot{q})$ is an nx1 vector of centrifugal and coriolis terms,

$H(q)$ is the nxn matrix of the manipulator.

§.3.2. Lagrangian formulation of manipulator dynamics

§ 3.2.1. Lagrangian dynamics

As we have seen in the last part of this report, arithmetic operations are needed to derive the closed-form dynamic equations. This represents a complex procedure which requires physical intuition.

An alternative is the Lagrangian formulation, which describes the behavior of a dynamic system in terms of work and energy stored. Thus, the closed-form can be derived systematically in any coordinate system.

We define the Lagrangian $\mathcal{L}$ by

$$\mathcal{L}(q_i, \dot{q}_i) = T - U \tag{4}$$

T is the total kinetic energy stored in the dynamic system.
U is the total potential energy stored in the dynamic system.
$q_1, \ldots, q_n$ is the generalized coordinates.

Using the Lagrangian, equations of motion of the dynamic system are given by

$$\frac{d}{dt}\left\{\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right\} - \frac{\partial \mathcal{L}}{\partial q_i} = Q_i \qquad i=1,\ldots,n \tag{5}$$

where Q_i is the generalized force corresponding to the generalized coordinate q_i.

§.3.2.2 The manipulator inertia tensor

As shown in the figure 3.2, v_{ci} is the velocity vector of the centroid and ω_i the angular velocity vector, with reference to the base coordinate frame, which is an inertial reference frame.

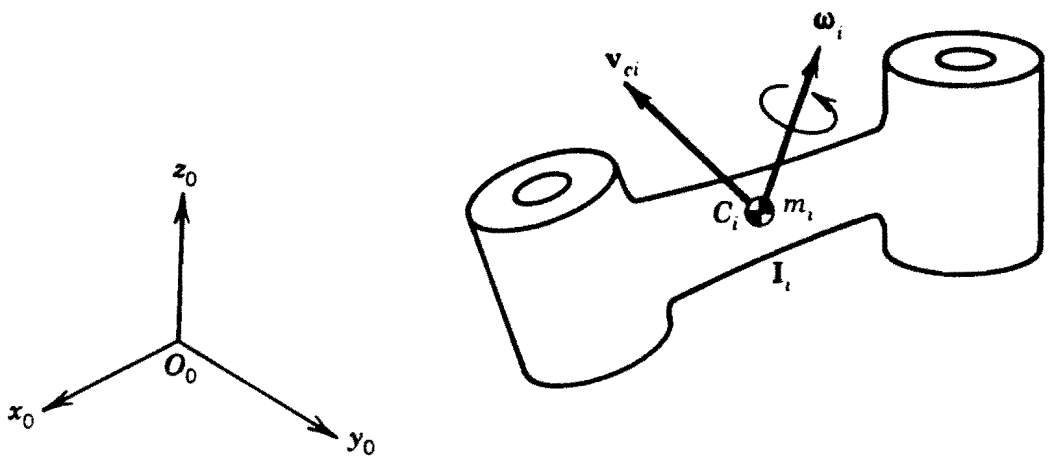


Figure 3.2 *Centroidal velocity and angular velocity of link i*

Thus the kinetic energy of link i is given by

$$T_i = \frac{1}{2} m_i v_{ci}^T v_{ci} + \frac{1}{2} \omega_i^T I_i \omega_i \qquad (6)$$

The total kinetic energy stored in the whole arm linkage is then given by

$$T = \sum_{i=1}^{n} T_i \qquad (7)$$

The expression for the kinematic energy is written in terms of linear velocity and angular velocity of each link member, which are not independent variables. To rewrite the above equations in terms of an independent set of generalized coordinates, we can use the jacobian matrix.

$$v_{ci} = J^i_{L1}\,\dot{q}_1 + \ldots\ldots + J^i_{Li}\,\dot{q}_i = J^i_L\,\dot{q}$$

$$\omega_i = J^i_{A1}\,\dot{q}_1 + \ldots\ldots + J^i_{Ai}\,\dot{q}_i = J^i_A\,\dot{q} \qquad (8)$$

where J^i_{Lj} and J^i_{Aj} are the j-th column vectors of the 3 x n Jacobian matrices J^i_L and J^i_A, for Linear and Angular velocities of link i, respectively.

This yields to :

$$T = \frac{1}{2}\sum_{i=1}^{n}\left(m_i \dot{q}^T J^{i\,T}_L J^i_L\,\dot{q} + \dot{q}^T J^{i\,T}_A\, I_i\, J^i_A\,\dot{q}\right) = \frac{1}{2}\,\dot{q}^T H\,\dot{q} \qquad (9)$$

where H is the n x n matrix given by :

$$H = \sum_{i=1}^{n}\left(m_i J^{iT}_L J^i_L + J^{iT}_A\, I_i\, J^i_A\right) \qquad (10)$$

The matrix H incorporates all the mass properties of the whole arm linkage. This matrix is named the *Manipulator Inertia Tensor*. It is obvious that the Manipulator inertia tensor is *configuration-dependent* and represents the instantaneous composite mass properties of the whole arm linkage at the current arm configuration.

If H_{ij} are the components of these matrix, we can rewrite equation (9) in a scalar form so that,

$$T = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij}\,\dot{q}_i\,\dot{q}_j \qquad (11)$$

§.3.2.3. Deriving Lagrange's equations of motion

In order to derive Lagrange's equations of motion, we need to find the potential energy U. The potential energy stored in the whole arm linkage is given by :

$$U = \sum_{i=1}^{n} m_i \, g^T \, r_{o,ci} \tag{12}$$

We can now derive Lagrange's equations of motion using the total kinetic energy (11) and the total potential energy (12).

$$\frac{d}{dt}\left\{\frac{\partial T}{\partial \dot{q}_i}\right\} = \frac{d}{dt}\left\{\sum_{j=1}^{n} H_{ij}\,\dot{q}_i\right\} = \sum_{j=1}^{n} H_{ij}\,\ddot{q}_j + \sum_{j=1}^{n} \frac{d\,H_{ij}}{dt}\,\dot{q}_j \tag{13}$$

$$\frac{d\,H_{ij}}{dt} = \sum_{k=1}^{n} \frac{\partial\,H_{ij}}{\partial\,q_k}\frac{dq_k}{dt} = \sum_{k=1}^{n} \frac{\partial\,H_{ij}}{\partial\,q_k}\,\dot{q}_k \tag{14}$$

$$\frac{\partial\,T}{\partial\,q_i} = \frac{\partial}{\partial\,q_i}\left\{\frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} H_{jk}\,\dot{q}_j\,\dot{q}_k\right\} = \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} \frac{\partial\,H_{jk}}{\partial\,q_i}\,\dot{q}_j\,\dot{q}_k \tag{15}$$

$$G_i = \frac{\partial\,U}{\partial\,q_i} = \sum_{j=1}^{n} m_j g^T \frac{\partial\,r_{o,cj}}{\partial\,q_i} = \sum_{j=1}^{n} m_j g^T L_{Li}^{j} \tag{16}$$

Equations (13) through (16) into (5) yields :

$$\sum_{j=1}^{n} H_{ij}\,\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}\,\dot{q}_j\,\dot{q}_k + G_i = Q_i \qquad i=1,\ldots,n \tag{17}$$

where

$$h_{ijk} = \frac{\partial H_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial H_{jk}}{\partial q_i} \tag{18}$$

and

$$G_i = \sum_{j=1}^{n} m_j g^T J_{Li}^j \tag{19}$$

The first term represents inertia torques, including interaction torques, while the second term accounts for the Coriolis and centrifugal effects, and the last term is the gravity torque. Equation (3) is the same as equation (17) derived from Newton-Euler equations. Thus the Lagrangian formulation provides the closed-form dynamic equations directly.

or we can also express equation (17) as

$$\tau(t) = H(q)\,\ddot{q}(t) + V(q,\dot{q}) + G(q) \tag{17'}$$

§.3.3. Inverse dynamics

The closed-form dynamic equations derived in the previous sections govern the dynamic responses of a manipulator arm to the input joint torques generated by the actuators. This dynamic process can be illustrated by the block diagram of figure 3.3 where the inputs are joint torques $\tau_1(t),\ldots\ldots,\tau_n(t)$, and the outputs are generalized coordinates, joint displacements $q_1(t),\ldots\ldots,q_n(t)$.
Inverse problems are important to robot control and programming, since they allow one to find the appropriate inputs necessary for producing the desired outputs.

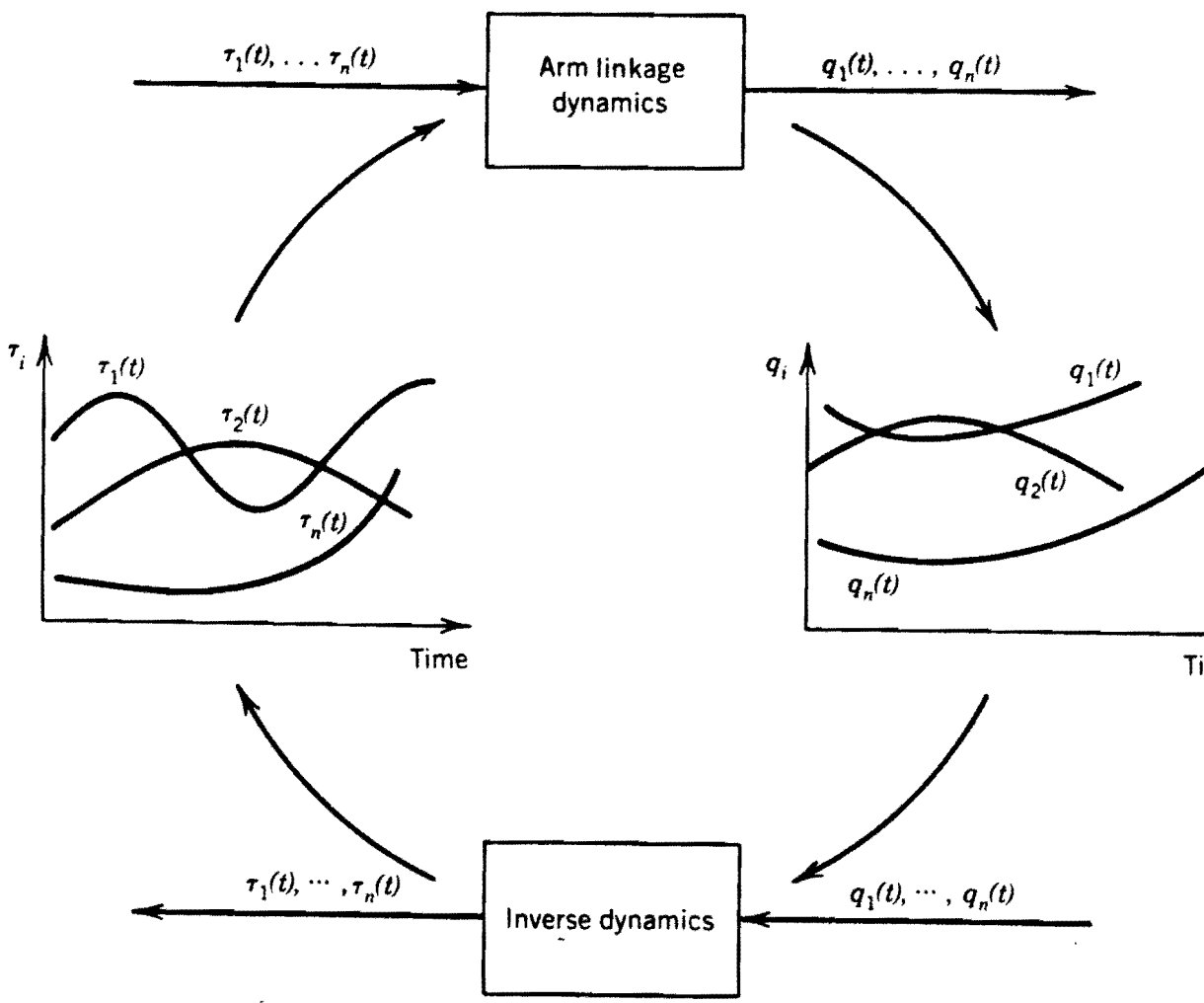


Figure 3.3 *Inverse dynamics*

In this section, we discuss the inverse dynamics process, shown in the block diagramm at the bottom of figure 3.3. The inputs are the desired trajectories, described as time functions $q_1(t)$ through $q_n(t)$. The outputs are the joint torques to be applied at each instant by the actuators in order to follow the specified trajectories, and are obtained by evaluating the right-hand side of the closed-form dynamic equations. Using the specified trajectory data.

$$\tau_i = \sum_{j=1}^{n} H_{ij} \ddot{q}_j + \sum_{j=1}^{n} \sum_{k=1}^{n} h_{ijk} \dot{q}_j \dot{q}_k + G_i \qquad i=1,\ldots,n$$

At each instant, we compute joint velocities $\dot{q}$ and joint accelerations $\ddot{q}$ from the given time functions,and then substitute them to the right-hand side of the above equation.

The total amount of computation becomes extremely large because as, all the coefficients, H_{ij}, h_{ijk}, G_i are configuration-dependent, they need to be computed at each instant. Thus the extremely heavy computation load is a bottleneck for the inverse dynamics.

But, the inverse dynamics approach is particularly important for control, since it allows us to compensate for the highly coupled and nonlinear arm dynamics. Thus we need to find fast computation algorithms.

§.3.3.1 Recursive computation

An efficient algorithm for inverse dynamics computation based on the Newton-Euler formulation have recently been developed. It reduce the computational complexity from $O(n^4)$ to $O(n)$.

The key concept of these method is to formulate dynamic equations in a *recursive form*, so that the computation can be accomplished from one link to another. Figure 3.4 illustrates the outline of the recursive computation algorithm. The algorithm can be applied to any manipulator arm with an open kinematic chain structure.

This formulation yields a set of *forward* and *backward recursive equations* which can be applied to the manipulator links sequentially. The forward recursion propagates kinematics information of each link from the base reference frame to the end-effector. The backward recursion transforms the forces that exert on each link from the end-effector to the base frame and the applied joint torques are computed from these forces.

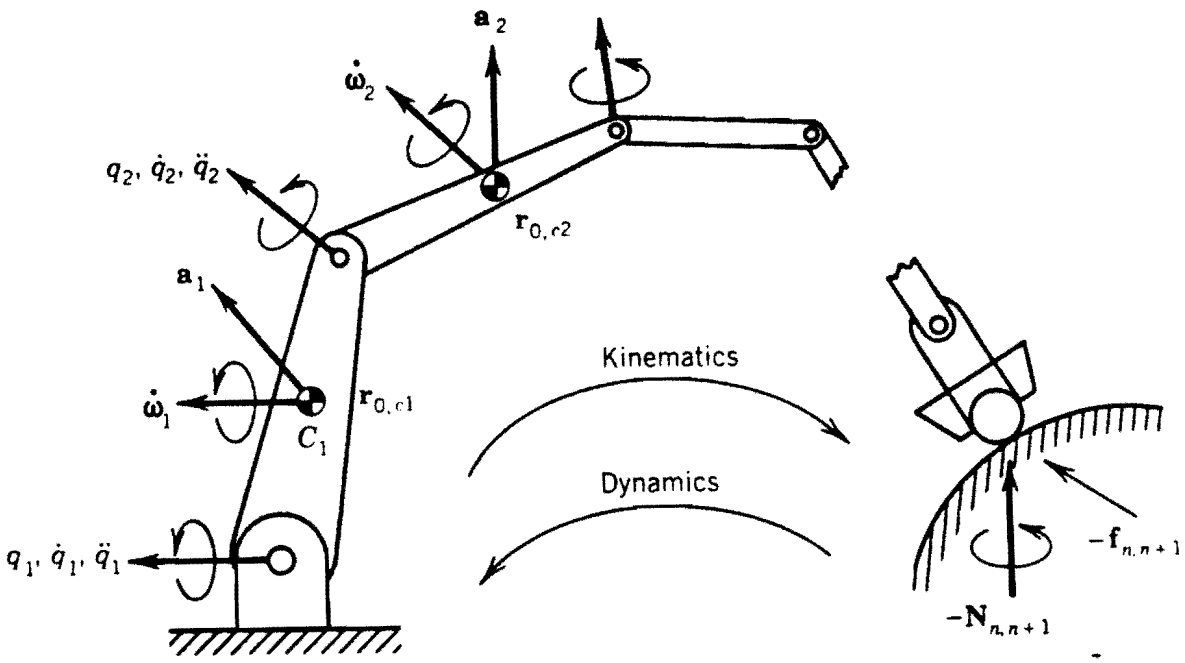


Figure 3.4 *Recursive computation of kinematic and dynamic equations*

The first phase of the recursive Newton-Euler formulation (forward computation) is to determine all the kinematic variables that are needed for evaluating the Newton-Euler equations. These include the linear and angular velocities and accelerations of each link member involved in the serial linkage.
The algorithm starts with the first link.

- Given the joint displacement q_1 and the joint velocity and accelerations $\dot{q}_1$ and $\ddot{q}_1$, the linear and angular velocities and accelerations of the centroid C_1 are determined.

- Then, using the velocities and accelerations of the first link, denoted by v_{c1}, ω_1, $\dot{v}_{c1}$ and $\dot{\omega}_1$, we compute the velocity and acceleration of the second link with the data specified for joint 2, namely q_2, $\dot{q}_2$, $\ddot{q}_2$.

- This procedure is repeated until all the centroidal velocities and accelerations, as well as the angular velocities and accelerations, are determined for all the links involved.

The second phase (backward computation) of the recursive formulation is to evaluate the Newton-Euler equations with the computed kinematic variables to determine the joint torques. We now proceed with the recursive computation starting from the last link back to the proximal links.

To summarize, the backward recursive procedure can be formulated as :

$$^{o}F_{i-1,i} = {}^{o}F_{i,i+1} - M_i g + M_i {}^{o}\dot{v}_{ci} \tag{20}$$

$$^{o}N_{i-1,i} = {}^{o}N_{i,i+1} - {}^{o}r_{i,ci} \wedge {}^{o}F_{i,i+1} + {}^{o}r_{i-1,ci} \wedge {}^{o}F_{i-1,i} + {}^{o}I_i {}^{o}\dot{\omega}_i + {}^{o}\omega_i \wedge ({}^{o}I_i {}^{o}\omega_i) \tag{21}$$

This procedure is repeated until the link number i reaches i = 1. Once the coupling and moment of each joint are determined, the joint torque can be computed with :

- for a prismatic joint :

$$\tau_i = {}^{o}b_{i-1}^{T} \cdot {}^{o}F_{i-1,i} \tag{22}$$

- for a revolute joint :

$$\tau_i = {}^{o}b_{i-1}^{T} \cdot {}^{o}N_{i-1,i} \tag{23}$$

In the first phase of the computation, we need to find the velocity and acceleration of link i, given the motion of the previous link and the specified motion of link i relative to link i-1. To solve this problem, we need to analyze relative motions defined in a moving coordinate frame. In the next section we derive basic results about moving coordinates, and then apply these results to the recursive computation algorithm.

§ 3.3.2. Moving coordinates

According to the Denavit-Hartenberg convention, one coordinate system is attached to each link of the manipulator. Thus the coordinates move together with links. Consider the following three coordinate systems as shown in figure 3.5 : the base coordinates (x_o, y_o, z_o) attached to link 0 with origin 0, the coordinates (x_i, y_i, z_i) attached to link i with origin 0^*, and the coordinates $(x_{i+1}, y_{i+1}, z_{i+1})$ attached to link i+1 with origin 0'.

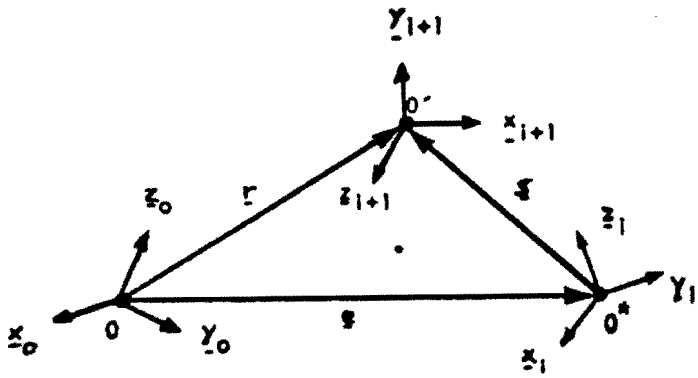


Figure 3.5 *Relationships among three coordinate systems*

Thus 0' is located by the position vector r relative to origin 0, or the vector s relative to origin 0^*. As shown in figure 3.5, these vectors are related by :

$$r = s + e \tag{24}$$

Let ${}^{o}v_i$ and ${}^{o}\omega_i$ be, respectively, the linear and angular velocity of coordinate system (x_i, y_i, z_i) with reference to the base coordinates (x_o, y_o, z_o). Then the velocity of coordinate system $(x_{i+1}, y_{i+1}, z_{i+1})$ with reference to base coordinate is :

$$ {}^{o}v_{i+1} = \frac{d^{*}s}{d\,t} + {}^{o}\omega_{i} \Lambda s + {}^{o}v_{i} \qquad (25) $$

where, d^{*}/dt denotes the time derivative with respect to the moving coordinates (x_i, y_i, z_i). Thus the acceleration is given by

$$ {}^{o}\dot{v}_{i+1} = \frac{d^{*}\,^{2}s}{dt^{2}} + {}^{o}\dot{\omega}_{i} \Lambda s + 2\,{}^{o}\omega_{i} \Lambda \frac{d^{*}s}{d\,t} + {}^{o}\omega_{i} \Lambda ({}^{o}\omega_{i} \Lambda s) + {}^{o}\dot{v}_{i} \qquad (26) $$

in which the third and the fourth terms are respectively, the Coriolis and centrifugal accelerations.

Let ${}^{o}\omega_{i+1}$ and ${}^{i}\omega_{i+1}$ be the angular velocity of coordinate system $(x_{i+1}, y_{i+1}, z_{i+1})$ with reference respectively the systems (x_o, y_o, z_o) and (x_i, y_i, z_i) Then,

$$ {}^{o}\omega_{i+1} = {}^{o}\omega_{i} + {}^{i}\omega_{i+1} \qquad (27) $$

$$ {}^{o}\dot{\omega}_{i+1} = {}^{o}\dot{\omega}_{i} + {}^{i}\dot{\omega}_{i+1} \qquad (28) $$

But,

$$ {}^{i}\dot{\omega}_{i+1} = \frac{d^{*}\,{}^{i}\omega_{i+1}}{d\,t} + {}^{o}\omega_{i} \Lambda {}^{i}\omega_{i+1} \qquad (29) $$

hence, (28) becomes

$$ {}^{o}\dot{\omega}_{i+1} = {}^{o}\dot{\omega}_{i} + \frac{d^{*}\,{}^{i}\omega_{i+1}}{d\,t} + {}^{o}\omega_{i} \Lambda {}^{i}\omega_{i+1} \qquad (30) $$

Equations (25), (26), (27), (30) are basic relations for velocities and accelerations among links and base of the manipulator.

§ 3.3.3. Luh-Walker-Paul's Algorithm

On the basis of the kinematic analysis on the moving coordinate frame, we now formulate the recursive computation algorithm of Newton-Euler dynamic equations. The algorithm was originally developed by Luh, Walker and Paul.

The first phase (Forward computation) consist of kinematic computations. We derive different recursive equations depending on the type of joint (prismatic or revolute).

Forward Equations with respect to the base frame: for i = 1, 2,....n

When the joint i+1 is prismatic :

$$^{o}\omega_{i+1} = {}^{o}\omega_{i} \tag{31}$$

$$^{o}\dot{\omega}_{i+1} = {}^{o}\dot{\omega}_{i} \tag{32}$$

$$^{o}v_{i+1} = {}^{o}v_{i} + \dot{q}_{i+1}.{}^{o}b_{i} + {}^{o}\omega_{i+1} \wedge {}^{o}r_{i,i+1} \tag{33}$$

$$^{o}\dot{v}_{i+1} = {}^{o}\dot{v}_{i} + \ddot{q}_{i+1}.{}^{o}b_{i} + {}^{o}\dot{\omega}_{i+1} \wedge {}^{o}r_{i,i+1} + 2 * {}^{o}\omega_{i+1} \wedge \dot{q}_{i+1}.{}^{o}b_{i} + {}^{o}\omega_{i+1} \wedge \left[{}^{o}\omega_{i+1} \wedge {}^{o}r_{i,i+1}\right] \tag{34}$$

When the joint i+1 is revolute :

$$^{o}\omega_{i+1} = {}^{o}\omega_{i} + \dot{q}_{i+1}.{}^{o}b_{i} \tag{35}$$

$$^{o}\dot{\omega}_{i+1} = {}^{o}\dot{\omega}_{i} + \ddot{q}_{i+1}.{}^{o}b_{i} + {}^{o}\omega_{i} \wedge \dot{q}_{i+1}.{}^{o}b_{i} \tag{36}$$

$$^{o}v_{i+1} = {}^{o}v_{i} + {}^{o}\omega_{i+1} \wedge {}^{o}r_{i,i+1} \tag{37}$$

$$^{o}\dot{v}_{i+1} = {}^{o}\dot{v}_{i} + {}^{o}\dot{\omega}_{i+1} \wedge {}^{o}r_{i,i+1} + {}^{o}\omega_{i+1} \wedge \left[{}^{o}\omega_{i+1} \wedge {}^{o}r_{i,i+1}\right] \tag{38}$$

The Newton-Euler equations are expressed in terms of centroidal accelerations, whereas the recursive formulation is expressed with respect to the origin of the coordinate frame attached to each link.

This is illustrated in figure 3.6, where v_i and ω_i are, respectively, the velocity at the origin of the coordinate frame attached to link i, and the angular velocity of the link.

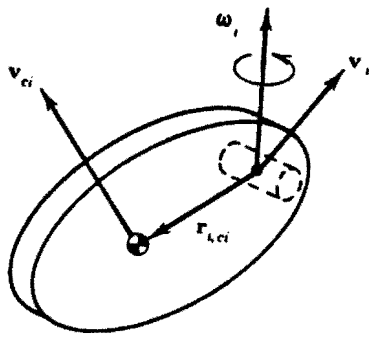


Figure 3.6 *Centroid velocity and joint velocity*

The centroidal velocity is then given by

$$^{o}v_{ci} = {}^{o}v_i + {}^{o}\omega_i \Lambda {}^{o}r_{i,ci} \tag{39}$$

$$^{o}\dot{v}_{ci} = {}^{o}\dot{v}_i + {}^{o}\dot{\omega}_i \Lambda {}^{o}r_{i,ci} + {}^{o}\omega_i \Lambda \left[{}^{o}\omega_i \Lambda {}^{o}r_{i,ci}\right] \tag{40}$$

With all these kinematic parameters, we can now evaluate the backward equations

Backward Equations with respect to the base frame : for i = n,.....,1

$$^{o}F_{i-1,i} = {}^{o}F_{i,i+1} - M_i g + M_i {}^{o}\dot{v}_{ci} \tag{41}$$

$$^{o}N_{i-1,i} = {}^{o}N_{i,i+1} - {}^{o}r_{i,ci} \Lambda {}^{o}F_{i,i+1} + {}^{o}r_{i-1,ci} \Lambda {}^{o}F_{i-1,i} + {}^{o}I_i {}^{o}\dot{\omega}_i + {}^{o}\omega_i \Lambda ({}^{o}I_i {}^{o}\omega_i) \tag{42}$$

$$\tau_i = {}^{o}b_{i-1}{}^{T} \cdot {}^{o}F_{i-1,i} \qquad \text{for a prismatic joint.} \tag{43}$$

$$\tau_i = {}^{o}b_{i-1}{}^{T} \cdot {}^{o}N_{i-1,i} \qquad \text{for a revolute joint} \tag{44}$$

The equations derived in the preceding section are referenced to the base coordinate systems. Unfortunately the inertia tensor I_i is dependent of the orientation of the link i, which is changing. When we evaluate Euler's equation, the inertia tensor must be obtained for each arm configuration. This requires extra computation time. Thus, the computation is more complicated and especially long.

To solve this problem, it is better to rewrite the equations (31) through (44) in link coordinates (in spite of the base coordinate frame). Let ${}^{i}R_{i+1}$ be a 3 x 3 matrix which transforms any vector with reference to $(x_{i+1}, y_{i+1}, z_{i+1})$ coordinate system to a new coordinate system whose coordinates are parallel to (x_i, y_i, z_i), i.e. a pure rotation. Thus the following equations are derived :

Forward Equations with respect to the link coordinates: for i = 1,...n

When the joint i+1 is prismatic :

$$ {}^{i+1}\omega_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}\omega_{i} \tag{45} $$

$$ {}^{i+1}\dot{\omega}_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}\dot{\omega}_{i} \tag{46} $$

$$ {}^{i+1}v_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * \left[{}^{i}v_{i} + z.\dot{q}_{i+1}\right] + {}^{i+1}\omega_{i+1} \wedge \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1} \tag{47} $$

$$ \begin{aligned} {}^{i+1}\dot{v}_{i+1} = {} & \left[{}^{i}R_{i+1}\right]^{-1} * \left[{}^{i}\dot{v}_{i} + z.\ddot{q}_{i+1}\right] + {}^{i+1}\dot{\omega}_{i+1} \wedge \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1} + \\ & 2 * {}^{i+1}\omega_{i+1} \wedge \left[{}^{i}R_{i+1}\right]^{-1} * z.\dot{q}_{i+1} + \\ & {}^{i+1}\omega_{i+1} \wedge \left\{{}^{i+1}\omega_{i+1} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} \end{aligned} \tag{48} $$

When joint i+1 is revolute :

$$ {}^{i+1}\omega_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * \left[{}^{i}\omega_{i} + z.\dot{q}_{i+1}\right] \tag{49} $$

$$ {}^{i+1}\dot{\omega}_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * \left[{}^{i}\dot{\omega}_{i} + z.\ddot{q}_{i+1} + {}^{i}\omega_{i} \wedge z.\dot{q}_{i+1}\right] \tag{50} $$

$$ {}^{i+1}v_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}v_{i} + {}^{i+1}\omega_{i+1} \wedge \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1} \tag{51} $$

$$ {}^{i+1}\dot{v}_{i+1} = \left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}\dot{v}_{i} + {}^{i+1}\dot{\omega}_{i+1} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} + $$

$$ {}^{i+1}\omega_{i+1} \wedge \left\{{}^{i+1}\omega_{i+1} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} \tag{52} $$

The centroidal velocity is then given by

$$ {}^{i}v_{ci} = {}^{i}v_{i} + {}^{i}\omega_{i} \wedge {}^{i}r_{i,ci} \tag{53} $$

$$ {}^{i}\dot{v}_{ci} = {}^{i}\dot{v}_{i} + {}^{i}\dot{\omega}_{i} \wedge {}^{i}r_{i,ci} + {}^{i}\omega_{i} \wedge \left[{}^{i}\omega_{i} \wedge {}^{i}r_{i,ci}\right] \tag{54} $$

Backward Equations with respect to the link coordinates: for i=n,..,1

$$ {}^{i}F_{i-1,i} = \left[{}^{i}R_{i+1}\right] * {}^{i+1}F_{i,i+1} + M_{i} * {}^{i}\dot{v}_{ci} \tag{55} $$

$$ {}^{i}N_{i-1,i} = \left[{}^{i}R_{i+1}\right] * {}^{i+1}N_{i,i+1} - {}^{i}r_{i,ci} \wedge \left[{}^{i}R_{i+1}\right] * {}^{i+1}F_{i,i+1} + $$

$$ {}^{i}r_{i-1,ci} \wedge {}^{i}F_{i-1,i} + {}^{i}I_{i} * {}^{i}\dot{\omega}_{i} + {}^{i}\omega_{i} \wedge \left[{}^{i}I_{i} * {}^{i}\omega_{i}\right] \tag{56} $$

Finally we derive the joint torques as follow :

$$\tau_i = \left\{ \left[{}^{i-1}R_i \right]^{-1} * z \right\}^T * {}^{i}F_{i-1,i} \qquad \text{for a prismatic joint} \qquad (57)$$

$$\tau_i = \left\{ \left[{}^{i-1}R_i \right]^{-1} * z \right\}^T * {}^{i}N_{i-1,i} \qquad \text{for a revolute joint} \qquad (58)$$

§ 4. Adaptive control for robot manipulators :

A purpose of manipulator control is to maintain a prescribed motion for the manipulator along a desired time-based trajectory by applying corrective compensation torques to the actuators to adjust for any deviations of the manipulator from the trajectory.

Most of the existing control schemes control the manipulator at the joint level or the hand level and emphasize nonlinear compensations of the interaction forces among the various joints. These control are inadequate because they neglect the changes of the load in the task cycle. These changes are significant enough to render conventional feedback control strategies ineffective. Any significant performance gain in robot arm control require the consideration of adaptive control techniques.

The proposed adaptive control is based on the linearization of the recursive formulation of Luh's Newton-Euler equations of motion. This idea from Henk Smit allow us to obtain directly the state matrices A(t) and B(t). With the determination of these two matrices, proper control laws can be designed to obtain the required correction torques to reduce the position errors of the manipulator along the nominal trajectory.

The controlled system is then characterized by feedforward and feedback components. Using the recursive Newton-Euler equations of motion seen in the previous chapter as an inverse dynamics of the manipulator, the feedforward component computes the nominal torques which compensate all the interaction forces among the the various joints along the nominal trajectory. Using the linearization of the recursive formulation of Luh's Newton-Euler equations of motion, the feedback component computes the variational torques which reduce the position and the velocity errors to zero along the nominal trajectory.

The total torques applied to the joint actuators then consist of the nominal torques computed from the Newton-Euler equations and the variational torques computed from feedback component.

The figure 4.1 shows the different part of the proposed adaptive control.

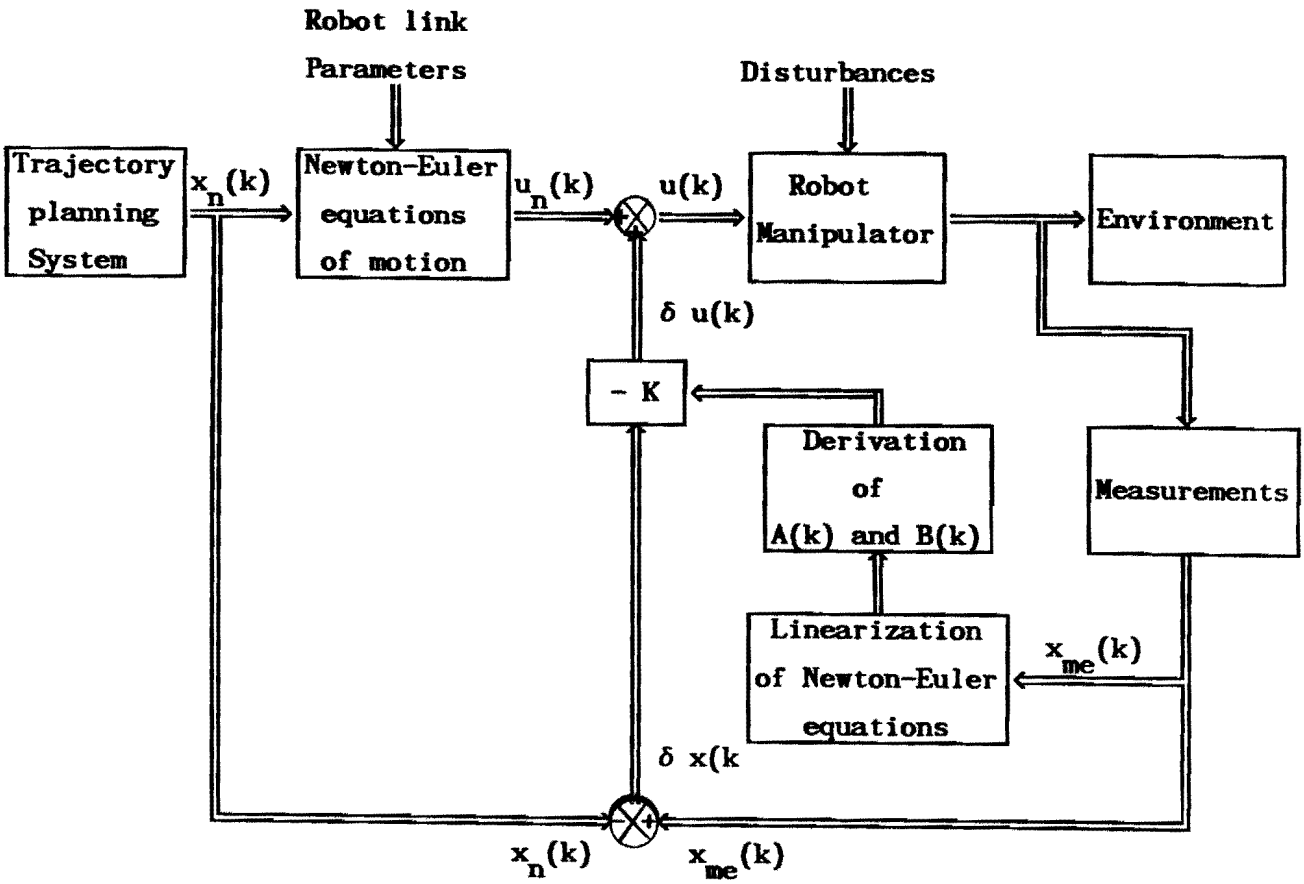


Figure 4.1 *A control Block Diagram of The Adaptive Control System*

§ 4.1. The feedforward component

We derive the nominal joint torques with the Luh-Walker-Paul's algorithm seen in the chapter § 3.3.3 (equation 45 through 58). This algorithm is the fastest of existing algorithms for dynamic computation. It takes 4.5 milliseconds on average to compute the six joint torques on a PDP 11/45 minicomputer using floating point assembly language.

The computation procedure is summarized in Figure 4.2. The left half of the figure shows the kinematics computation, while the right half shows the dynamics computation. The kinematics computation proceeds downwards while the dynamics proceeds upwards.

The input data of joint motions are transmitted horizontally from the left to the right. The data in the last column are the output joint torques computed through the operations shown by the blocks. The equation numbers in each block indicate the computation to be performed at each stage.

Starting from the top left corner, we first specify the velocity and the acceleration of the base link. Note that in this algorithm we can deal with the case when the base frame of the manipulator arm is in motion, if the acceleration of the base frame is known. Note also that the acceleration of gravity is represented as part of acceleration of the base frame, so that the effect of gravity can be included without extra computation.

The first computation block give the velocities and acceleration of the first link according to the link coordinate, which are used in the second step of the computation. Also, the data is transmitted to the right computation block, where the centroidal velocity and acceleration are obtained. The results are further transmitted to the third column, where the Newton-Euler equations are evaluated, and the coupling forces and moments are produced. The result is used to compute the joint torque.

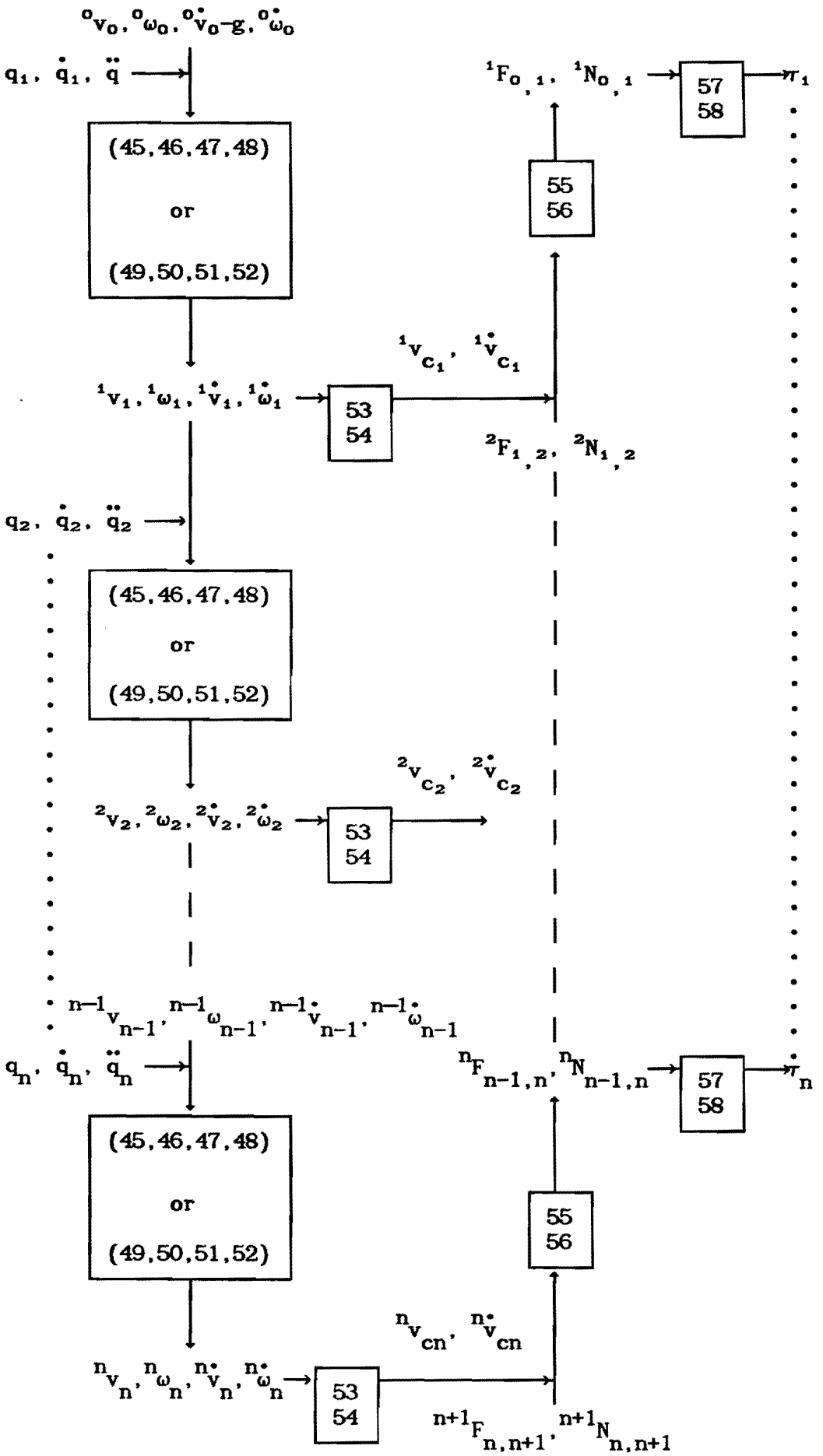


Figure 4.2 *Computational structure of the Luh-Walker-Paul's algorithm*

§ 4.2. The feedback component

§ 4.2.1. Derivation of the state space representation

Defining a 2 x n-dimensional state vector for the system as :

$$x^T(t) = (q^T(t), \dot{q}^T(t)) = (q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n) = (x_1, \ldots, x_{2n}) \qquad (59)$$

and a n-dimensional input vector as

$$u^T(t) = (\tau_1, \ldots, \tau_n) = (u_1, \ldots, u_n) \qquad (60)$$

The closed-form dynamic equations of a n-degree-of-freedom manipulator can be expressed in state space representation as

$$\dot{x}(t) = f(x(t), u(t)) \qquad (61)$$

where f(.) is a 2n x 1 nonlinear vector-valued function and continuously differentiable, and n is the number of D.O.F.

§ 4.2.1.1. Perturbation equations of motion

Suppose that the nominal states $x_d(t)$ of the system (equation 61) are known from the planned trajectory, and the corresponding torques $u_d(t)$ are also known from the computations of the joint torques using the Newton-Euler equations of motion. Then both $x_d(t)$ and $u_d(t)$ satisfy equation (61).

$$\dot{x}_d(t) = f(x_d(t), u_d(t)) \qquad (62)$$

Using the taylor series expansion on equation (61) about the nominal trajectory and substracting equation (62) from it and assuming that the higher order terms are negligible, the associated linearized perturbation model for this control system can be obtained :

$$\delta \dot{x}(t) = \nabla_x f|_d \, \delta x(t) + \nabla_u f|_d \, \delta u(t) \tag{63}$$

$$\delta \dot{x}(t) = A(t) \, \delta x(t) + B(t) \, \delta u(t) \tag{64}$$

where $\nabla_x f|_d$ and $\nabla_u f|_d$ are the Jacobian matrices of $f(x(t),u(t))$ evaluated at $x_d(t)$ and $u_d(t)$, respectively, $\delta x(t) = x(t) - x_d(t)$ and $\delta u(t) = u(t) - u_d(t)$.

The system parameters, A(t) and B(t), of equation (64) depend on the instantaneous manipulator position and velocity along the nominal trajectory and are thus slowly varying in time. Because of the complexity of the manipulator equations of motion, it is extremely difficult to find the elements of A(t) and B(t) explicitly.

However, the design of a feedback control law for the perturbation equations requires that the system parameters of equation (64) be known at all times. Thus a special technique must be used to find the unknown elements in A(t) and B(t).

§ 4.2.1.2 Derivation of the system parameters A(t) and B(t)

As τ is function of q, $\dot{q}$, $\ddot{q}$, we can derive the differential as follow

$$d\tau = \frac{\partial f}{\partial q}.dq + \frac{\partial f}{\partial \dot{q}}.d\dot{q} + \frac{\partial f}{\partial \ddot{q}}d\ddot{q} \qquad (65)$$

this yields the following matrix form

$$\Delta \tau = M \Delta \ddot{q} + C \Delta \dot{q} + K \Delta q \qquad (66)$$

$$\begin{bmatrix} \Delta\tau_1 \\ \vdots \\ \Delta\tau_n \end{bmatrix} = \begin{bmatrix} \frac{\partial \tau_1}{\partial \ddot{q}_1} & \cdots & \frac{\partial \tau_1}{\partial \ddot{q}_n} \\ \vdots & & \vdots \\ \frac{\partial \tau_n}{\partial \ddot{q}_1} & \cdots & \frac{\partial \tau_n}{\partial \ddot{q}_n} \end{bmatrix} \begin{bmatrix} \Delta\ddot{q}_1 \\ \vdots \\ \Delta\ddot{q}_n \end{bmatrix} + \begin{bmatrix} \frac{\partial \tau_1}{\partial \dot{q}_1} & \cdots & \frac{\partial \tau_1}{\partial \dot{q}_n} \\ \vdots & & \vdots \\ \frac{\partial \tau_n}{\partial \dot{q}_1} & \cdots & \frac{\partial \tau_n}{\partial \dot{q}_n} \end{bmatrix} \begin{bmatrix} \Delta\dot{q}_1 \\ \vdots \\ \Delta\dot{q}_n \end{bmatrix} + \begin{bmatrix} \frac{\partial \tau_1}{\partial q_1} & \cdots & \frac{\partial \tau_1}{\partial q_n} \\ \vdots & & \vdots \\ \frac{\partial \tau_n}{\partial q_1} & \cdots & \frac{\partial \tau_n}{\partial q_n} \end{bmatrix} \begin{bmatrix} \Delta q_1 \\ \vdots \\ \Delta q_n \end{bmatrix}$$

then

$$\Delta \ddot{q} = M^{-1} (\Delta \tau - C \Delta \dot{q} - K \Delta q) \qquad (67)$$

When you use the equation (67) issued from equation (65) you can derive an equation of the same form as equation (64) i.e.:

$$\delta \dot{x}(t) = A(t)\, \delta x(t) + B(t)\, \delta u(t)$$

$$\begin{bmatrix} \delta \dot{x}_1 \\ \vdots \\ \delta \dot{x}_n \\ \delta \ddot{x}_1 \\ \vdots \\ \delta \ddot{x}_n \end{bmatrix} = \left[\begin{array}{ccc|ccc} 0 & \cdots & 0 & 1 & & \\ \vdots & & \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 0 & & 1 \\ \hline -M^{-1}K & & & -M^{-1}C & & \end{array}\right] \begin{bmatrix} \delta x_1 \\ \vdots \\ \delta x_n \\ \delta \dot{x}_1 \\ \vdots \\ \delta \dot{x}_n \end{bmatrix} + \left[\begin{array}{ccc} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \\ \hline +M^{-1} & & \end{array}\right] \begin{bmatrix} \delta u_1 \\ \vdots \\ \delta u_n \end{bmatrix} \qquad (68)$$

Formulated in this way, the derivation of the system parameters is obvious, we just need to evaluate the three matrices M, C and K to obtain A(t) and B(t).

When we have A(t) and B(t), the control problem is to find a feedback control law $u(t) = g(x(t))$ such that the closed loop control system $\dot{x}(t) = f(x(t),g(x(t)))$ is asymptotically stable and tracks a desired trajectory as closely as possible over a wide range of payloads for all times.

§ 4.2.1.2.1. Derivation of the Jacobian matrix M

To derive the matrix M, we use the linearization about $\ddot{q}$ of Newton-Euler equations in the vicinity of a nominal trajectory, i.e. equations (45) through (58), this yields

Linearization about $\ddot{q}$ of the forward equations:

When the joint i+1 is prismatic :

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \ddot{q}_m} = 0 \tag{69}$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \ddot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{\omega}_{i}}{\partial\, \ddot{q}_m} \tag{70}$$

$$\frac{\partial\, {}^{i+1}v_{i+1}}{\partial\, \ddot{q}_m} = 0 \tag{71}$$

$$\frac{\partial\, {}^{i+1}\dot{v}_{i+1}}{\partial\, \ddot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \left\{\frac{\partial\, {}^{i}\dot{v}_{i}}{\partial\, \ddot{q}_m} + z.\frac{\partial\, q_{i+1}}{\partial\, \ddot{q}_m}\right\} + \frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \ddot{q}_m} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} \tag{72}$$

When the joint i+1 is revolute :

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \ddot{q}_m} = 0 \tag{73}$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \ddot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \left\{\frac{\partial\, {}^{i}\dot{\omega}_{i}}{\partial\, \ddot{q}_m} + z.\frac{\partial\, \ddot{q}_{i+1}}{\partial\, \ddot{q}_m}\right\} \tag{74}$$

$$\frac{\partial\, {}^{i+1}v_{i+1}}{\partial\, \ddot{q}_m} = 0 \tag{75}$$

$$\frac{\partial\, {}^{i+1}\dot{v}_{i+1}}{\partial\, \ddot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{v}_{i}}{\partial\, \ddot{q}_m} + \frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \ddot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} \tag{76}$$

The linearization about $\ddot{q}$ of the centroidal velocity is given by

$$\frac{\partial\, {}^{i}v_{ci}}{\partial\, \ddot{q}_m} = 0 \tag{77}$$

$$\frac{\partial\, {}^{i}\dot{v}_{ci}}{\partial\, \ddot{q}_m} = \frac{\partial\, {}^{i}\dot{v}_{i}}{\partial\, \ddot{q}_m} + \frac{\partial\, {}^{i}\dot{\omega}_{i}}{\partial\, \ddot{q}_m} \Lambda\, {}^{i}r_{i,ci} \tag{78}$$

Linearization about $\ddot{q}$ of the Backward Equations

$$\frac{\partial\, {}^{i}F_{i-1,i}}{\partial\, \ddot{q}_m} = {}^{i}R_{i+1} * \frac{\partial\, {}^{i+1}F_{i,i+1}}{\partial\, \ddot{q}_m} + M_i * \frac{\partial\, {}^{i}\dot{v}_{ci}}{\partial\, \ddot{q}_m} \tag{79}$$

$$\frac{\partial\, {}^{i}N_{i-1,i}}{\partial\, \ddot{q}_m} = {}^{i}R_{i+1} * \frac{\partial\, {}^{i+1}N_{i,i+1}}{\partial\, \ddot{q}_m} - {}^{i}r_{i,ci} \wedge {}^{i}R_{i+1} * \frac{\partial\, {}^{i+1}F_{i,i+1}}{\partial\, \ddot{q}_m} + {}^{i}r_{i-1,ci} \wedge \frac{\partial\, {}^{i}F_{i-1,i}}{\partial\, \ddot{q}_m} + {}^{i}I_i * \frac{\partial\, {}^{i}\dot{\omega}_i}{\partial\, \ddot{q}_m} \tag{80}$$

Finally, we derive the coefficients of the matrix M by the linearization about $\ddot{q}$ of the joint torques.

$$\frac{\partial\, \tau_i}{\partial\, \ddot{q}_m} = \left\{ \left[{}^{i-1}R_i \right]^{-1} * z \right\}^{T} * \frac{\partial\, {}^{i}F_{i-1,i}}{\partial\, \ddot{q}_m} \quad \text{for a prismatic joint} \tag{81}$$

$$\frac{\partial\, \tau_i}{\partial\, \ddot{q}_m} = \left\{ \left[{}^{i-1}R_i \right]^{-1} * z \right\}^{T} * \frac{\partial\, {}^{i}N_{i-1,i}}{\partial\, \ddot{q}_m} \quad \text{for a revolute joint} \tag{82}$$

Note, that it is necessary to compute all these terms for i = 1,..,n and for m = 1,...,n to obtain the matrix M, which is:

$$M = \begin{bmatrix} \frac{\partial\, \tau_1}{\partial\, \ddot{q}_1} & \cdots & \frac{\partial\, \tau_1}{\partial\, \ddot{q}_n} \\ \vdots & & \vdots \\ \frac{\partial\, \tau_n}{\partial\, \ddot{q}_1} & \cdots & \frac{\partial\, \tau_n}{\partial\, \ddot{q}_n} \end{bmatrix}$$

§ 4.2.1.2.2. Derivation of the Jacobian matrix C :

To derive the matrix C, we use exactly the same way, we linearize about $\dot{q}$ the equations (45) through (58) :

Linearization of the forward equations about $\dot{q}$:

When the joint i+1 is prismatic :

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\omega_{i}}{\partial\, \dot{q}_m} \tag{83}$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{\omega}_{i}}{\partial\, \dot{q}_m} \tag{84}$$

$$\frac{\partial\, {}^{i+1}v_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \left\{\frac{\partial\, {}^{i}v_{i}}{\partial\, \dot{q}_m} + z.\frac{\partial\, \dot{q}_{i+1}}{\partial\, \dot{q}_m}\right\} +$$

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} \tag{85}$$

$$\frac{\partial\, {}^{i+1}\dot{v}_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{v}_{i}}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} +$$

$$2 * \frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * \left[z * \dot{q}_{i+1}\right]\right\} +$$

$$2 * {}^{i+1}\omega_{i+1} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1}.z.\frac{\partial\, \dot{q}_{i+1}}{\partial\, \dot{q}_m}\right\} +$$

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{{}^{i+1}\omega_{i+1} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} +$$

$${}^{i+1}\omega_{i+1} \Lambda \left\{\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} \tag{86}$$

When the joint i+1 is revolute :

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \left\{\frac{\partial\, {}^{i}\omega_{i}}{\partial\, \dot{q}_m} + z.\frac{\partial\, \dot{q}_{i+1}}{\partial\, \dot{q}_m}\right\} \tag{87}$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \left\{\frac{\partial\, {}^{i}\dot{\omega}_{i}}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i}\omega_{i}}{\partial\, \dot{q}_m} \Lambda\, z\dot{q}_{i+1} + {}^{i}\omega_{i} \Lambda\, z.\frac{\partial\, \dot{q}_{i+1}}{\partial\, \dot{q}_m}\right\} \tag{88}$$

$$\frac{\partial\, {}^{i+1}v_{i+1}}{\partial\, \dot{q}_m} = \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}v_{i}}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} \tag{89}$$

$$\begin{aligned}\frac{\partial\, {}^{i+1}\dot{v}_{i+1}}{\partial\, \dot{q}_m} = {} & \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{v}_{i}}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} + \\ & \frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{{}^{i+1}\omega_{i+1} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} + \\ & {}^{i+1}\omega_{i+1} \Lambda \left\{\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, \dot{q}_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\}\end{aligned} \tag{90}$$

The linearization of the centroidal velocity about $\dot{q}$ is given by

$$\frac{\partial\, {}^{i}v_{ci}}{\partial\, \dot{q}_m} = \frac{\partial\, {}^{i}v_{i}}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i}\omega_{i}}{\partial\, \dot{q}_m} \Lambda\, {}^{i}r_{i,ci} \tag{91}$$

$$\begin{aligned}\frac{\partial\, {}^{i}\dot{v}_{ci}}{\partial\, \dot{q}_m} = {} & \frac{\partial\, {}^{i}\dot{v}_{i}}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i}\dot{\omega}_{i}}{\partial\, \dot{q}_m} \Lambda\, {}^{i}r_{i,ci} + \frac{\partial\, {}^{i}\omega_{i}}{\partial\, \dot{q}_m} \Lambda \left[{}^{i}\omega_{i} \Lambda\, {}^{i}r_{i,ci}\right] + \\ & {}^{i}\omega_{i} \Lambda \left\{\frac{\partial\, {}^{i}\omega_{i}}{\partial\, \dot{q}_m} \Lambda\, {}^{i}r_{i,ci}\right\}\end{aligned} \tag{92}$$

Linearization about $\dot{q}$ of the Backward Equations

$$\frac{\partial\, {}^{i}F_{i-1,i}}{\partial\, \dot{q}_m} = {}^{i}R_{i+1} * \frac{\partial\, {}^{i+1}F_{i,i+1}}{\partial\, \dot{q}_m} + M_i * \frac{\partial\, {}^{i}\dot{v}_{ci}}{\partial\, \dot{q}_m} \tag{93}$$

$$\frac{\partial\, {}^{i}N_{i-1,i}}{\partial\, \dot{q}_m} = {}^{i}R_{i+1} * \frac{\partial\, {}^{i+1}N_{i,i+1}}{\partial\, \dot{q}_m} - {}^{i}r_{i,ci} \wedge {}^{i}R_{i+1} * \frac{\partial\, {}^{i+1}F_{i,i+1}}{\partial\, \dot{q}_m} +$$

$$ {}^{i}r_{i-1,ci} \wedge \frac{\partial\, {}^{i}F_{i-1,i}}{\partial\, \dot{q}_m} + {}^{i}I_i * \frac{\partial\, {}^{i}\dot{\omega}_i}{\partial\, \dot{q}_m} + \frac{\partial\, {}^{i}\omega_i}{\partial\, \dot{q}_m} \wedge \left[{}^{i}I_i * {}^{i}\omega_i\right] +$$

$$ {}^{i}\omega_i \wedge \left\{ {}^{i}I_i * \frac{\partial\, {}^{i}\omega_i}{\partial\, \dot{q}_m} \right\} \tag{94}$$

Finally, we derive the coefficients of the matrix C by the linearization about $\dot{q}$ of the joint torques :

$$\frac{\partial\, \tau_i}{\partial\, \dot{q}_m} = \left\{ \left[{}^{i-1}R_i\right]^{-1} * z \right\}^{T} * \frac{\partial\, {}^{i}F_{i-1,i}}{\partial\, \dot{q}_m} \qquad \text{for a prismatic joint} \tag{95}$$

$$\frac{\partial\, \tau_i}{\partial\, \dot{q}_m} = \left\{ \left[{}^{i-1}R_i\right]^{-1} * z \right\}^{T} * \frac{\partial\, {}^{i}N_{i-1,i}}{\partial\, \dot{q}_m} \qquad \text{for a revolute joint} \tag{96}$$

We need to evaluate all these terms for $i = 1,\ldots,n$ and for $m = 1,\ldots,n$ to obtain the matrix C which is :

$$C = \begin{bmatrix} \frac{\partial\, \tau_1}{\partial\, \dot{q}_1} & \cdots & \frac{\partial\, \tau_1}{\partial\, \dot{q}_n} \\ \vdots & & \vdots \\ \frac{\partial\, \tau_n}{\partial\, \dot{q}_1} & \cdots & \frac{\partial\, \tau_n}{\partial\, \dot{q}_n} \end{bmatrix}$$

§ 4.2.1.2.3. Derivation of the Jacobian matrix K :

As in the two last section, to derive the matrix K, we use the linearization about q of equations (45) through (58), this yields

Linearization of the Forward equations about q

When the joint i+1 is prismatic :

$$\frac{\partial\ {}^{i+1}\omega_{i+1}}{\partial\ q_m} = \frac{\partial\ \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\ q_m} * {}^{i}\omega_i + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\ {}^{i}\omega_i}{\partial\ q_m} \tag{97}$$

$$\frac{\partial\ {}^{i+1}\dot{\omega}_{i+1}}{\partial\ q_m} = \frac{\partial\ \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\ q_m} * {}^{i}\dot{\omega}_i + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\ {}^{i}\dot{\omega}_i}{\partial\ q_m} \tag{98}$$

$$\frac{\partial\ {}^{i+1}v_{i+1}}{\partial\ q_m} = \frac{\partial\ \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\ q_m} * \left[{}^{i}v_i + z * \dot{q}_{i+1}\right] + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\ {}^{i}v_i}{\partial\ q_m} +$$

$$\frac{\partial\ {}^{i+1}\omega_{i+1}}{\partial\ q_m} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} +$$

$${}^{i+1}\omega_{i+1} \wedge \left\{\frac{\partial\ \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\ q_m} * {}^{i}r_{i,i+1}\right\} \tag{99}$$

$$\frac{\partial\, {}^{i+1}\dot{v}_{i+1}}{\partial\, q_m} = \frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * \left[{}^{i}\dot{v}_i + z * \ddot{q}_{i+1}\right] + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{v}_i}{\partial\, q_m} +$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, q_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} +$$

$${}^{i+1}\dot{\omega}_{i+1} \Lambda \left\{\frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * {}^{i}r_{i,i+1}\right\} +$$

$$2 * \frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, q_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * \left[z * \dot{q}_{i+1}\right]\right\} +$$

$$2 * {}^{i+1}\omega_{i+1} \Lambda \left\{\frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * z * \dot{q}_{i+1}\right\} +$$

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, q_m} \Lambda \left\{{}^{i+1}\omega_{i+1} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} +$$

$${}^{i+1}\omega_{i+1} \Lambda \left\{\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, q_m} \Lambda \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\}\right\} +$$

$${}^{i+1}\omega_{i+1} \Lambda \left\{{}^{i+1}\omega_{i+1} \Lambda \left\{\frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * {}^{i}r_{i,i+1}\right\}\right\} \qquad (100)$$

When the joint i+1 is revolute :

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, q_m} = \frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * \left[{}^{i}\omega_i + z.\dot{q}_{i+1}\right] + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\omega_i}{\partial\, q_m} \qquad (101)$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, q_m} = \frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * \left[{}^{i}\dot{\omega}_i + z.\ddot{q}_{i+1} + {}^{i}\omega_i \wedge z.\dot{q}_{i+1}\right] +$$

$$\left[{}^{i}R_{i+1}\right]^{-1} * \left\{\frac{\partial\, {}^{i}\dot{\omega}_i}{\partial\, q_m} + \frac{\partial\, {}^{i}\omega_i}{\partial\, q_m} \wedge z.\dot{q}_{i+1}\right\} \qquad (102)$$

$$\frac{\partial\, {}^{i+1}v_{i+1}}{\partial\, q_m} = \frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * {}^{i}v_i + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}v_i}{\partial\, q_m} +$$

$$\frac{\partial\, {}^{i+1}\omega_{i+1}}{\partial\, q_m} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} +$$

$${}^{i+1}\omega_{i+1} \wedge \left\{\frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * {}^{i}r_{i,i+1}\right\} \qquad (103)$$

$$\frac{\partial\, {}^{i+1}\dot{v}_{i+1}}{\partial\, q_m} = \frac{\partial \left[{}^{i}R_{i+1}\right]^{-1}}{\partial\, q_m} * {}^{i}\dot{v}_i + \left[{}^{i}R_{i+1}\right]^{-1} * \frac{\partial\, {}^{i}\dot{v}_i}{\partial\, q_m} +$$

$$\frac{\partial\, {}^{i+1}\dot{\omega}_{i+1}}{\partial\, q_m} \wedge \left\{\left[{}^{i}R_{i+1}\right]^{-1} * {}^{i}r_{i,i+1}\right\} +$$

$$
{}^{i+1}\dot{\omega}_{i+1} \wedge \left\{ \frac{\partial \left[{}^{i}R_{i+1} \right]^{-1}}{\partial q_m} * {}^{i}r_{i,i+1} \right\} +
$$

$$
\frac{\partial \, {}^{i+1}\omega_{i+1}}{\partial q_m} \wedge \left\{ {}^{i+1}\omega_{i+1} \wedge \left\{ \left[{}^{i}R_{i+1} \right]^{-1} * {}^{i}r_{i,i+1} \right\} \right\} +
$$

$$
{}^{i+1}\omega_{i+1} \wedge \left\{ \frac{\partial \, {}^{i+1}\omega_{i+1}}{\partial q_m} \wedge \left\{ \left[{}^{i}R_{i+1} \right]^{-1} * {}^{i}r_{i,i+1} \right\} \right\} +
$$

$$
{}^{i+1}\omega_{i+1} \wedge \left\{ {}^{i+1}\omega_{i+1} \wedge \left\{ \frac{\partial \left[{}^{i}R_{i+1} \right]^{-1}}{\partial q_m} * {}^{i}r_{i,i+1} \right\} \right\} \tag{104}
$$

The linearization about q of the centroidal velocity is given by

$$
\frac{\partial \, {}^{i}v_{ci}}{\partial q_m} = \frac{\partial \, {}^{i}v_{i}}{\partial q_m} + \frac{\partial \, {}^{i}\omega_{i}}{\partial q_m} \wedge {}^{i}r_{i,ci} + {}^{i}\omega_{i} \wedge \frac{\partial \, {}^{i}r_{i,ci}}{\partial q_m} \tag{105}
$$

$$
\frac{\partial \, {}^{i}\dot{v}_{ci}}{\partial q_m} = \frac{\partial \, {}^{i}\dot{v}_{i}}{\partial q_m} + \frac{\partial \, {}^{i}\dot{\omega}_{i}}{\partial q_m} \wedge {}^{i}r_{i,ci} + {}^{i}\dot{\omega}_{i} \wedge \frac{\partial \, {}^{i}r_{i,ci}}{\partial q_m} +
$$

$$
\frac{\partial \, {}^{i}\omega_{i}}{\partial q_m} \wedge \left[{}^{i}\omega_{i} \wedge {}^{i}r_{i,ci} \right] + {}^{i}\omega_{i} \wedge \left\{ \frac{\partial \, {}^{i}\omega_{i}}{\partial q_m} \wedge {}^{i}r_{i,ci} \right\} +
$$

$$
{}^{i}\omega_{i} \wedge \left\{ {}^{i}\omega_{i} \wedge \frac{\partial \, {}^{i}r_{i,ci}}{\partial q_m} \right\} \tag{106}
$$

Linearization about q of the Backward Equations

$$\frac{\partial\ {}^{i}F_{i-1,i}}{\partial\ q_m} = \frac{\partial\ \left[{}^{i}R_{i+1}\right]}{\partial\ q_m} * {}^{i+1}F_{i,i+1} + {}^{i}R_{i+1} * \frac{\partial\ {}^{i+1}F_{i,i+1}}{\partial\ q_m} +$$

$$M_i * \frac{\partial\ {}^{i}\dot{v}_{ci}}{\partial\ q_m} \qquad (107)$$

$$\frac{\partial\ {}^{i}N_{i-1,i}}{\partial\ q_m} = \frac{\partial\ \left[{}^{i}R_{i+1}\right]}{\partial\ q_m} * {}^{i+1}N_{i,i+1} + {}^{i}R_{i+1} * \frac{\partial\ {}^{i+1}N_{i,i+1}}{\partial\ q_m} -$$

$${}^{i}r_{i,ci}\ \Lambda \left\{\frac{\partial\ \left[{}^{i}R_{i+1}\right]}{\partial\ q_m} * {}^{i+1}F_{i,i+1} + {}^{i}R_{i+1} * \frac{\partial\ {}^{i+1}F_{i,i+1}}{\partial\ q_m}\right\} +$$

$${}^{i}r_{i-1,ci}\ \Lambda\ \frac{\partial\ {}^{i}F_{i-1,i}}{\partial\ q_m} + {}^{i}I_i * \frac{\partial\ {}^{i}\dot{\omega}_i}{\partial\ q_m} + \frac{\partial\ {}^{i}\omega_i}{\partial\ q_m}\ \Lambda\ \left[{}^{i}I_i * {}^{i}\omega_i\right] +$$

$${}^{i}\omega_i\ \Lambda \left\{{}^{i}I_i * \frac{\partial\ {}^{i}\omega_i}{\partial\ q_m}\right\} - \frac{\partial\ {}^{i}r_{i,ci}}{\partial\ q_m}\ \Lambda \left\{\left[{}^{i}R_{i+1}\right] * {}^{i+1}F_{i,i+1}\right\} +$$

$$\frac{\partial\ {}^{i}r_{i-1,ci}}{\partial\ q_m}\ \Lambda\ {}^{i}F_{i-1,i} + \frac{\partial\ {}^{i}I_i}{\partial\ q_m} * {}^{i}\dot{\omega}_i + {}^{i}\omega_i\ \Lambda \left\{\frac{\partial\ {}^{i}I_i}{\partial\ q_m} * {}^{i}\omega_i\right\} \qquad (108)$$

Finally, we derive all the coefficients of the third matrix K by the linearization about q of the joint torques as follow

$$\frac{\partial \tau_i}{\partial q_m} = \left\{ \frac{\partial \left[{}^{i-1}R_i \right]^{-1}}{\partial q_m} * z \right\}^T * {}^{i}F_{i-1,i} + \left\{ \left[{}^{i-1}R_i \right]^{-1} * z \right\}^T * \frac{\partial {}^{i}F_{i-1,i}}{\partial q_m} \quad (109)$$

$$\frac{\partial \tau_i}{\partial q_m} = \left\{ \frac{\partial \left[{}^{i-1}R_i \right]^{-1}}{\partial q_m} * z \right\}^T * {}^{i}N_{i-1,i} + \left\{ \left[{}^{i-1}R_i \right]^{-1} * z \right\}^T * \frac{\partial {}^{i}N_{i-1,i}}{\partial q_m} \quad (110)$$

We need here also to evaluate all these terms for i = 1,...,n and for M = 1,.....,n to obtain K which is

$$K = \begin{bmatrix} \frac{\partial \tau_1}{\partial q_1} & \cdots & \frac{\partial \tau_1}{\partial q_n} \\ \vdots & & \vdots \\ \frac{\partial \tau_n}{\partial q_1} & \cdots & \frac{\partial \tau_n}{\partial q_n} \end{bmatrix}$$

Now that we have the M, C, K matrices the derivation of A(t) and B(t) is easy. (see equation 68)

Note also that is it not necessary to compute all the terms which are known to be equal to zero. For instance ω_i is only a function of q_m when m is less or equal to i.

§ 5. Dynamic Simulation

§ 5.1. Inclusion of nonrigid body effects

It is important to realize that the dynamic equations we have derived do not encompass all the effects acting on a manipulator. They include just those forces which arise from rigid body mechanics. The most important source of forces that are not included is friction. All mechanisms are, of course affected by frictional forces. In present day manipulators in which significant gearing is typical, the forces due to friction can actually be quite large (perhaps, 25% of the torque required to move the manipulator in typical situations.

In order to make dynamics equations reflect the reality of the physical device, it is important to model (at least approximately) these forces of friction. A very simple model for friction is viscious friction in which the torque due to the friction is proportional to the velocity of joint motion. Thus we have:

$$\tau_{friction} = v\dot{q}, \tag{111}$$

where v is a viscious friction constant. Another possible simple model for friction, Coulomb friction, is sometimes used. Coulomb friction is constant except for a sign dependence on the joint velocity:

$$\tau_{friction} = c\ sgn(\dot{q}), \tag{112}$$

where c is a Coulomb friction constant. The value of c is often taken at one value when $\dot{q} = 0$, the static coefficient, and at a lower value, the dynamic coefficient when $\dot{q} \neq 0$. Whether a joint of a particular manipulator exhibits viscious or Coulomb friction is a complicated issue of lubrication and other effects. A reasonable model is to include both,

$$\tau_{friction} = c\ sgn(\dot{q}) + v\dot{q} \tag{113}$$

It turns out that in many manipulator joints, friction also displays a dependence on the joint position. A major cause of this effect might be gears which are not perfectly round (their eccentricity would cause friction to change according to joint position. So fairly complex friction model would have the form

$$\tau_{friction} = f(q,\dot{q}) \tag{114}$$

These friction models are then added to the other dynamic terms derived from the rigid body model, yielding the more complete model

$$\tau(t) = H(q)\,\ddot{q}(t) + V(q,\dot{q}) + G(q) + F(q,\dot{q}). \tag{115}$$

There are also other effects which are also neglected in this model. For example, the assumption of rigid body links means that we have failed to include bending effects (which give rise to resonances) in our equations of motion.

§ 5.2. The simulation itself

Because of the combinatorial complexity of the equations, researchers in the past have applied simplifying assumption to the model for the mechanism under consideration so that a solution could be obtained. However, the results thus obtained may only be valid in a limited range of operation, (for instance coriolis forces are non-negligible in high-speed motion).

Recently, more general solutions have been obtained for the dynamic equations of motion in which most of the simplifying assumptions have been removed (Newton-Euler equations).

A block diagram showing the component part of a typical dynamic computer simulation for control law development is given in Figure 5.1.

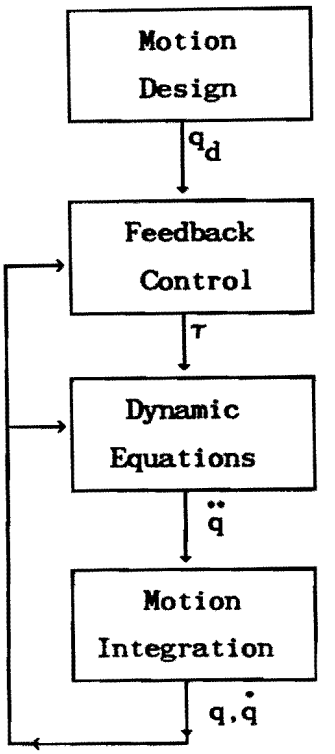


Figure 5.1 *Block diagram of a typical computer simulation*

As trained or commanded by human operator or as automatically generated through programmed algorithms, the complete trajectory is furnished by the Motion Design section. Through a control law, torques are applied by the actuators to the mechanism. Through solution of the dynamic equations, the joint accelerations may be obtained, and through integration, the actual trajectory is determined.

The basic problem as shown in Figure 5.1 is to solve for the relative joint accelerations given the input torques at a particular trajectory point (specified joint positions and velocities).

As we use the Newton-Euler formulation to obtain the dynamic equations of the manipulator, these equations contain the forces and the moments due to physical constraints at each joint. But, to simulate the manipulator, these forces and moments are eliminated from the equations. Hooker and Margulies have shown how this can be done for the Newton-Euler formulation. What is required for the simulation is not an explicit analytic expression for each of the terms in this equation but merely the general form of this equation.

It is :

$$H(q)\ddot{q} + V(q,\dot{q})\dot{q} + G(q) + K(q)^{T}k + F(q,\dot{q}) = \tau \qquad (116)$$

where

$H(q)$ n x n symmetric, nonsingular moment of inertia matrix.

$V(q,\dot{q})$ n x n matrix specifying centrifugal and coriolis effects.

$G(q)$ n x 1 vector specifying the effect due to the gravity.

$K(q)$ 6 xn Jacobian matrix specifying the torques (forces) created at each joint due to external forces and moments exerted on link. the last link.

k 6 x 1 vector of external moments and forces exerted on the last link

τ n x 1 vector of torques (forces) at each joint actuator.

q n x 1 vector of joint variables.

$F(q,\dot{q})$ n x 1 vector which represents the friction models

§ 5.2.1. Technique for solving for the Joint Accelerations

From equation (116), note that the joint torques (forces) are linear functions of the joint accelerations. Therefore if **b** is defined to be a bias vector which is equal to the torques (forces) due to the gravity, centrifugal and coriolis accelerations, and external forces and moments on the last link. We can also include the friction model in the bias. Then

$$b = V(q,\dot{q})\dot{q} + G(q) + K(q)^{T}k + F(q,\dot{q}) \qquad (117)$$

Thus the accelerations of the joint variables can be obtained by solving the linear equation

$$H(q)\ddot{q} = (\tau - b) \qquad (118)$$

This bias vector b can easily be computed by a function which compute the Newton-Euler equations with $q, \dot{q}$ and k to their current state, but letting $\ddot{q} = 0$. In fact, b is the torque computed considering that $\ddot{q} = 0$. To take in account the friction we must first compute $F(q, \dot{q})$ and after add that vector into the bias.

The difficult part in solving equation (116) is in evaluating the elements of the matrix H. There is different way to obtain it, the easiest is accomplished by setting q to its current state, but letting $\ddot{q} = e_j$, and use a function which calculate the Luh-Walker and Paul's algorithm where the velocity terms, the gravitational effects and the efffects due to external forces and moments are eliminated. Where e_j is a n x 1 vector with the jth element equal to 1 and 0 everywhere else. The result is h_j which is the jth column of H. That is, h_j is the torque (force) on the joint actuators when the joint velocities are zero; there are no external forces; there are no gravitational effects; and the joint accelerations, $\ddot{q}$ are equal to e_j.

Once the elements of H are determined, then the joint acceleration is obtained by solving equation 118. The exact procedure is:

STEP 1: Compute the bias vector b, using function 1.

STEP 2: Compute the matrix H one column at a time, using function 2.

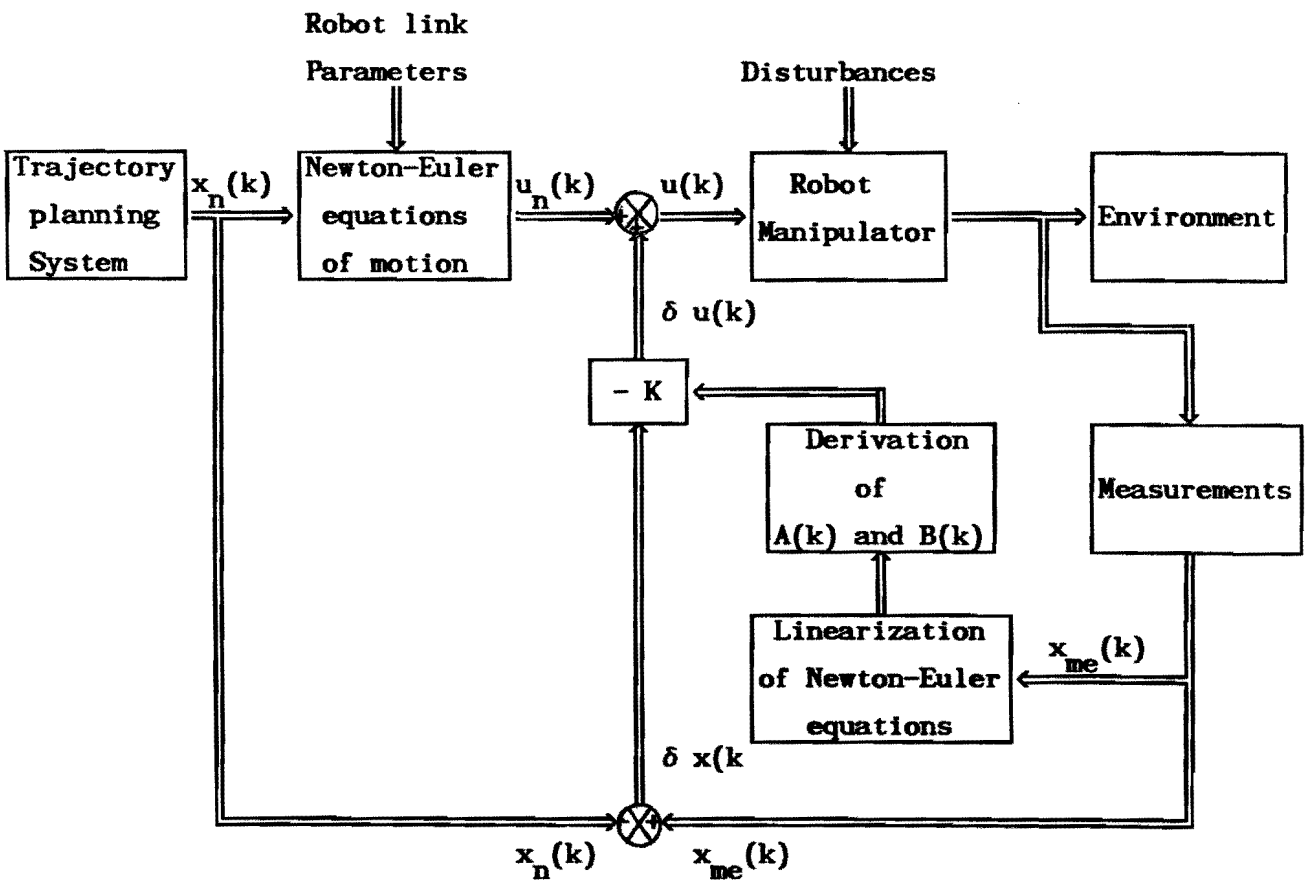
STEP 3: Solve the linear equation 118 for the joint accelerations

This method to compute the joint acceleration is the easiest, but, more efficient methods exist. One of these takes into account the fact that the moment of inertia matrix H is symmetric, but also utilizes a recursive procedure for computing the mass, and the moment of inertia matrix of the composite system of links j through n. Because I did not have enough time I could not implemente that method, but you can find more information about it in the article "Efficient Dynamic Computer Simulation of Robotic Mechanisms" from Walker, M.W. and Orin, D.E.

One can constat that the previous method has a lot of similarities with the Jacobian matrix M derived in the chapter § 4.2.1.2.1. That assumption has to be verify, but in that case it could be a great improvement in the simulation algorithm and save a lot of time. Normally, the matrix M is equal to the matrix H. So the computation of H is already done to obtain the A(t) and B(t) matrices.

§ 5.2.2. The different step to test the simulation

For convenience we put again the figure 4.1.



A control Block Diagram of The Adaptive Control System

The first step is to choose a sampletime and to make the trajectory discrete to obtain the set $q(k)$, $\dot{q}(k)$, $\ddot{q}(k)$ for each sampletime.

The second step is the calculation of the nominal torques $u_n(k)$. For that, we use the Luh-Walker-Paul's algorithm. The input is the set $q(k)$, $\dot{q}(k)$, $\ddot{q}(k)$ and the output is the vector $u_n(k)$, which is in fact the torque desired in each actuator to reach the state determined by $q(k)$, $\dot{q}(k)$, $\ddot{q}(k)$.

The third step is to calculate the joint accelerations with the dynamic model of the robot. The input of the dynamic model is the torque determined in the previous step added with the corrective torque obtained from the feedback loop. The derivation of the corrective torque $\delta u(k)$ is explained in chapter § 4.2.2. It is in fact the multiplication of the regulator matrix obtained with PC-MATLAB with the position error $\delta x(k)$

The fourth step is to derive the actual trajectory. For that, we can use integration to determine q and $\dot{q}$ with $\ddot{q}$.

We are now able to analyse and to interpret the results of that simulation.

CONCLUSION

I have implemented several programs to apply this algorithm.
The first one allow us to obtain the input torques with the Luh-Walker-Paul's algorithm.
The second one is the linearization of the previous algorithm to derive the system parameters A(k) and B(k).
The third one is the simulation itself.

§ 6. The software

We decided to use PC-MATLAB, a program available at the TUE. MATLAB is an interactive program to help you with your scientific and engineering numeric calculations. This allows you to solve many numerical problems in a fraction of the time it would take to write a program in a language like Fortran, Basic, or C. Furthermore, problem solution are expressed in MATLAB almost exactly as they are written mathematically. (you can find more information about PC-MATLAB in Appendix 2.).

In the Appendix you can find the different parts of the software. But because of the delay needed to print the report this is not the last version. In fact I used that delay to do the last modifications and especially to put comments in my different softwares. Because I wanted to know the result of that controller, I tested its validity for few desired trajectories.

It is obvious that I will give to my coach all my results and the last version of the software with comments in it.

§ 7. About the work atmosphere

I want to mention in this report how friendly the work atmosphere was during my training period.

After only a few days, I was very well integrated not only into the laboratory but also with the people I lived with. In this short period of time, we did a lot of things together not only in our work but also after work (we had dinner together, we played bowling, we did sports, we went out...). I want also to mention that the french are really appreciated in the Netherlands by the Dutch. As a consequence, our integration is really easy because a lot of people here like to talk with the french. At the beginning people were so friendly that I was really surprised by their attitude but now, I am used to it and I appreciate very much that attitude.

After these five months, I consider that I have very good friends in Holland (all the persons I was working with and so many others I can not mention here). From that point of view, I think that it could not be better...

As a conclusion, I just want to say that I enjoyed very much these five months in the Netherlands and that I will have a lot of difficulties to leave all my friends...

§ 8. Conclusion

During my training period I learnt a lot of things:

- From the knowledge point of view (about robotics, about control theory, about programming and how to obtain and to exploit scientific documents...). I learnt also how to develop a project at research level.
- From the personal point of view (working within a team, learning another way of life, speaking different langages -english, dutch or german are usual in Netherlands-, and also meeting a lot of people from everywhere in the world. It is a fantastic cultural enrichment...)

Unfortunately, five months spent at the TUE were not enough to finish a such project. It would have been very interesting for me to continue on this subject, because at the end I was very motivated by what I was doing. And especially to analyse myself the result of my work.

The following step is to test and to analyse the results of this adaptive controller. I think it could be a very nice subject for other training periods. The software can be improved by taking into account the different parallelisms of this algorithm and use the concurrent programming. For instance the derivation of the Jacobian matrices M, C, and K can be done simultaneously.

If the results are good, it would be very interesting to develop the programmation of this algorithm with a transputer.

APPENDIX

1. Résumé en français

Mon stage de longue durée s'est déroulé à l'Université de Technologie de Eindhoven au sein du groupe Production et Automatisme rattaché au Génie Mécanique.

Ce laboratoire travail dans le cadre du projet gouvernemental F.L.A.I.R. (FLexible Automation and Industrial Robots). Ce projet vise à organiser dans plusieurs Universités et entreprises privées la recherche concernant l'atelier flexible et la robotique.

Mon travail consistait à commencer une nouvelle étude dans le domaine de la dynamique d'un robot à plusieurs degrés de liberté et de ce qu'on appelle le control auto-adaptatif. C'est à dire élaborer un control qui tienne automatiquement compte des variations d'inertie ou de la charge du robot lors d'un déplacement.

Le stage s'est décomposé en deux principales parties:
- Une première partie de recherche bibliographique et d'étude d'articles afin d'acquérir les bases nécessaires pour mener à bien ma tache et surtout pour savoir se qu'il se faisait actuellement dans le domaine de la recherche en dynamique. Ceci dans le but de concevoir un algorithm de simulation.
- La deuxième partie fut d'implémenter cet algorithm à l'aide de PC.MATLAB pour simuler la dynamique d'une chaine cinématique ouverte à plusieurs degrés de liberté.

Malheureusement, ces cinq mois auront été trop courts pour executer la partie la plus intéressante du travail à savoir, analyser et interpreter complètement les résultats afin d'effectuer des modifications dans la stratégie de control et de tester son éfficacité. Néanmoins, le programme implémenté a été testé et les résultats sont fiables. Ce vaste sujet donnera lieu, je pense à beaucoup d'autre stages très intéressants.

En conclusion, je dirais qu'il n'y a pas de mots assez fort pour qualifier ces cinq mois passés en Hollande tant le travail et la vie m'ont plu. Cela m'a vraiment donné envie, premièrement de continuer dans le domaine de la recherche et deuxièmement d'étudier dans d'autres pays...

PC-MATLAB

for MS-DOS Personal Computers

Version 3.1-PC
February 27, 1987

by Cleve Moler, John Little and Steve Bangert

The MathWorks, Inc.
20 North Main St., Suite 250
Sherborn, MA 01770
(617) 653-1415

Preface

What is MATLAB?

MATLAB is an interactive program to help you with your scientific and engineering numeric calculations. The name MATLAB stands for *matrix laboratory*. Originally written in Fortran for mainframe computers, it provides easy access to matrix software developed by the LINPACK and EISPACK projects. Together, LINPACK and EISPACK represent the state of the art in software for matrix computation.

MATLAB is an interactive system whose basic data element is a matrix that does not require dimensioning. This allows you to solve many numerical problems in a fraction of the time it would take to write a program in a language like Fortran, Basic, or C. Furthermore, problem solutions are expressed in MATLAB almost exactly as they are written mathematically.

MATLAB has evolved over more than half a decade with input from many users. In university environments it has become the standard instructional tool used in introductory courses in applied linear algebra, as well as advanced courses in other areas. In industrial settings, MATLAB is used for research, and to solve practical engineering and mathematical problems. Typical uses include general purpose numeric computation, algorithm prototyping, and solving the special purpose problems with matrix formulations that arise in disciplines like automatic control theory, statistics, and digital signal processing (time-series analysis).

The highly optimized, second generation MATLAB that runs on IBM and other MS-DOS compatible personal computers is called PC-MATLAB. On larger computers, like Sun Workstations and VAX computers, the modern version of MATLAB is called PRO-MATLAB. On the Macintosh, it's MacMATLAB. Entirely written in the C language, MATLAB is a complete "integrated" system, including graphics, programmable macros, IEEE arithmetic, a fast interpreter, and many analytical commands. Experienced users of earlier versions of MATLAB will find the modern version of MATLAB to be a significantly more powerful version of an old friend. New users of MATLAB will benefit from the years of experience gained in developing the earlier versions. Both will find that MATLAB has evolved beyond the "matrix laboratory" to become a versatile scientific "spreadsheet" for numeric calculations.

About This Guide:

The MATLAB *User's Guide* is divided into three main parts:

COMPUTER The first section contains instructions that are specific to the particular computer that you are using. This includes *installation* instructions, explaining the mechanics of making backup copies, configuring the system, and invoking the program. Also explained in this section are *interactions* with MATLAB that are system dependent, like last-line editing and hardcopy operations. For users of Sun Workstation or Macintosh versions, the special added features that allow mouse and window system interaction are discussed.

TUTORIAL The *tutorial* is an introduction to the MATLAB system. The basic features including matrix manipulation, graphics, language features, and M-files are described. Many examples are given.

REFERENCE The *reference* section contains extensive details on all MATLAB functions, including information on how the algorithms work.

How To Use This Guide:

Once MATLAB is installed and running, this guide will not be needed very often. The on-line help and demo facilities provide most of what's needed in order to use the program.

New users should start by reading the *tutorial*. The most important things to learn are how to enter matrices, how to use the colon ":" operator, and how to invoke functions. After the basics are mastered, the on-line help facility is usually sufficient to learn the other commands.

Users of earlier versions of MATLAB should skim the *tutorial* to find out about graphics and M-files, which are the major differences between the old and the new MATLAB.

Users familiar with MATLAB on other machines should look at the computer-specific first section to find out about system dependent features.

Experienced users can rely on just the on-line help facility. Occasionally it may be useful to look up more detail on a particular function in the *reference section* of the manual, or to skim the *reference section* to discover new features.

About The Cover:

The covers of this guide and other *MathWorks, Inc.* publications depict various solutions to a problem which has played a small, but interesting, role in the history of numerical methods during the last 30 years. The problem involves finding the modes of vibration of a membrane supported by an L-shaped domain consisting of three unit squares. The nonconvex corner in the domain generates singularities in the solutions, thereby providing challenges for both the underlying mathematical theory and the computational algorithms. There are important applications, including wave guides, structures and semiconductors.

Two of the founders of modern numerical analysis, George Forsythe and J. H. Wilkinson, worked on the problem in the 1950's. (See G. E. Forsythe and W. R. Wasow, *Finite-Difference Methods for Partial Differential Equations*, Wiley, 1960.) One of the authors of this guide (Moler) used finite difference techniques to compute solutions in 1965. Typical computer runs took up to half an hour of dedicated computer time on what were then Stanford University's primary computers, an IBM 7090 and a Burroughs B5000.

The first version of the approach we now use was published in 1967 by L. Fox, P. Henrici and C. Moler (SIAM J. Numer. Anal. 4, 1967, pp.89-102.) It replaced finite differences by combinations of distinguished fundamental solutions to the underlying differential equation formed from Bessel and trigonometric functions. The idea is a generalization of the fact that the real and imaginary parts of complex analytic functions are solutions to Laplace's equation. In the early 1970's new matrix algorithms, particularly Gene Golub's orthogonalization techniques for least squares problems, provided further algorithmic improvements.

Today, MATLAB allows us to express the entire algorithm in a few dozen lines, to compute the solution with great accuracy in a few minutes on a computer at home, and to readily manipulate three dimensional displays of the results. Our MATLAB work was one step in the development of a demonstration program for the Intel iPSC hypercube multiprocessor which computes and displays moving pictures of the solution to the time-dependent wave equation extension of the membrane problem. We have included our MATLAB program, membrane.m, with the M-files in the *Utility Library*.

Who Wrote PC-MATLAB?

The original MATLAB was written in Fortran by Cleve Moler, in an evolutionary process over several years. The underlying matrix algorithms are from the many people who worked on the LINPACK and EISPACK projects.

The new MATLAB program was written in C by *The MathWorks, Inc.* Steve Bangert wrote the parser/user interface, Steve Kleiman and Marc Ullman implemented the graphics, and John Little and Cleve Moler wrote the analytical/numerical routines and the M-files. John Little and Cleve Moler are the authors of this user's guide.

Probably the most important feature of MATLAB, and one that we took care to perfect, is the easy extensibility. This allows *you* to become a contributing author too, creating your own applications. We look forward to the promise of many scientists, mathematicians and engineers quickly developing new and interesting commands, all without writing a single line of Fortran or other "low-level" code.

January 14, 1985
Portola Valley, California

NE.M is the subroutine which calculate the Newton Euler equations of motion, the input is the set $(q, \dot{q}, \ddot{q})$ and the output is the joint torques needed to follow the trajectories.

```
%*************************************************************************
%                                                                        *
%                              NE.M                                      *
%                                                                        *
%                                                                        *
% Luh-Walker-Paul's algorithm (relative to links coordinates)            *
% for a 2 D. O. F.                                                       *
%*************************************************************************
format short e
n=2;
Q=[1 0 0 0
   1 0 0 0];
l1=0.500;
l2=0.250;
M=[10
    5];
I=[0     0                  0
   0     M(1,1)*l1^2/12     0
   0     0                  M(1,1)*l1^2/12
   0     0                  0
   0     M(2,1)*l2^2/12     0
   0     0                  M(2,1)*l2^2/12];
VW=zeros(3*(n+1),6);
VW(1:3,4)=[0 -9.80621 0]';
changeq=input('do you want to change Q, yes=1,no=0 ?     ');
if changeq==1,
[Q]=MAKEQ(n,changeq);
end
%*************************************************************************
%                                                                        *
```

```
%       DENAVIT HARTENBERG PARAMETERS  twist,length,offset,jointangle
%
%*****************************************************************************
pp=[0 11 0 0
    0 12 0 0];
%*****************************************************************************
%*****************************************************************************
i=1;
  if i<=n,
    if Q(i,1)==1,
     pp(1,4)=(pi/180*input('TETA 1, first joint angle  :     '));
     elseif Q(i,1)==0,
     pp(1,3)=input('D1, first joint parameter   :     ');
    end
  end
i=2;
  if i<=n,
    if Q(i,1)==1,
     pp(2,4)=(pi/180*input('TETA 2, second joint angle :     '));
     elseif Q(i,1)==0,
     pp(2,3)=input('D2, second joint parameter   :     ');
    end
  end
i=3;
  if i<=n,
    if Q(i,1)==1,
     pp(3,4)=(pi/180*input('TETA 3, third joint angle  :     '));
     elseif Q(i,1)==0,
     pp(3,3)=input('D3, third joint parameter   :     ');
    end
  end
i=4;
  if i<=n,
    if Q(i,1)==1,
```

```
     pp(4,4)=(pi/180*input('TETA 4, fourth joint angle :     '));
     elseif Q(i,1)==0,
     pp(4,3)=input('D4, fourth joint parameter   :     ');
    end
  end
i=5;
  if i<=n,
    if Q(i,1)==1,
     pp(5,4)=(pi/180*input('TETA 5, fifth joint angle  :     '));
     elseif Q(i,1)==0,
     pp(5,3)=input('D5, fifth joint parameter   :     ');
    end
  end
i=6;
  if i<=n,
    if Q(i,1)==1,
     pp(6,4)=(pi/180*input('TETA 6, sixth joint angle  :     '));
     elseif Q(i,1)==0,
     pp(6,3)=input('D6, sixth joint parameter   :     ');
    end
  end
% r is the distance between O(i) and O(i+1) in frame i
r=[l1*cos(pp(1,4)) l2*cos(pp(2,4))
   l1*sin(pp(1,4)) l2*sin(pp(2,4))
    0                0          ];
%rc is the distance between O(i) and the centroid mass of the link i in frame i
rc=[-l1/2  -l2/2
       0      0
       0      0];
%rcc is the distance between O(i-1) and the centroid mass of the link i in frame
i
rcc=[l1/2 l2/2
      0     0
      0     0];
```

```
%*************************************************************************
%                                                                        *
%            CALCULATION OF THE HOMOGENEOUS MATRIX                       *
%                                                                        *
%*************************************************************************
tt=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
  for i=0:n-1,
   [t(4*i+1:4*i+4,1:4)]=trans(i,Q,pp(i+1,1),pp(i+1,2),pp(i+1,3),pp(i+1,4));
  end
i=n;
   [t(4*i+1:4*i+4,1:4)]=tt;
t
%*************************************************************************
%                                                                        *
%        CALCULATION OF THE ANGULAR AND LINEAR VELOCITIES, ACCELERATION  *
%                                                                        *
%*************************************************************************
 for i=0:n-1,
    if Q(i+1,1)==1,
      [VW(3*i+4:3*i+6,1:4)]=R(Q,r,VW,t,i);
    elseif Q(i+1,1)==0,
      [VW(3*i+4:3*i+6,1:4)]=T(Q,r,VW,t,i);
    end
end
%*************************************************************************
%                                                                        *
%calculation of the linear and angular velocities of each centroid mass. *
%                                                                        *
%*************************************************************************
for i=1:n,
VW(3*i+1:3*i+3,5)=VW(3*i+1:3*i+3,3)+vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i));
VW(3*i+1:3*i+3,6)=VW(3*i+1:3*i+3,4)+vectorpr(VW(3*i+1:3*i+3,2),rc(1:3,i))+...
             vectorpr(VW(3*i+1:3*i+3,1),vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i)));
end
```

```
VW
%*************************************************************************
%                                                                        *
%calculation of forces and torques                                       *
%                                                                        *
%*************************************************************************
F=zeros(3,n+1);
N=zeros(3,n+1);
F(1,1)=input('x force  :    ');
F(2,1)=input('y force  :    ');
F(3,1)=input('z force  :    ');
N(1,1)=input('x torque :    ');
N(2,1)=input('y torque :    ');
N(3,1)=input('z torque :    ');
 for i=n:-1:1,
  F(1:3,n+2-i)=t(4*i+1:4*i+3,1:3)*F(1:3,n+1-i)+M(i,1)*VW(3*i+1:3*i+3,6);
  N(1:3,n+2-i)=t(4*i+1:4*i+3,1:3)*N(1:3,n+1-i)-vectorpr(rc(1:3,i),...
   t(4*i+1:4*i+3,1:3)*F(1:3,n+1-i))+vectorpr(rcc(1:3,i),F(1:3,n+2-i))+...
    I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,2)+vectorpr(VW(3*i+1:3*i+3,1),...
     (I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,1)));
 end
F
N
%*************************************************************************
%                                                                        *
% calculation of torques.                                                *
%                                                                        *
%*************************************************************************
z=[0 0 1]';
tau=zeros(1,n);
for i=n:-1:1,
  if Q(i,1)==1,
       tau(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*N(1:3,n+2-i);
  elseif Q(i,1)==0,
```

```
        tau(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*F(1:3,n+2-i);
    end
end
tau
```

TRANS.M is the subroutine which derive the different homogeneous matrices which transform a vector from one coordinate system to the other.

```
%*************************************************************************
%                                                                        *
%                          TRANS.M                                       *
%                                                                        *
%*************************************************************************
function [tt]=trans(i,Q,twist,length,offset,jointangle);
if Q(i+1,1)==1,
  tt(1,1:4)=[cos(jointangle)   -cos(twist)*sin(jointangle)
sin(twist)*sin(jointangle)    length*cos(jointangle)];
  tt(2,1:4)=[sin(jointangle)    cos(twist)*cos(jointangle)
-sin(twist)*cos(jointangle)    length*sin(jointangle)];
  tt(3,1:4)=[      0                 sin(twist)                      cos(twist)
offset         ];
  tt(4,1:4)=[      0                   0                               0
1           ];
elseif Q(i+1,1)==0,
  tt(1,1:4)=[cos(jointangle)   -cos(twist)*sin(jointangle)
sin(twist)*sin(jointangle)              0          ];
  tt(2,1:4)=[sin(jointangle)    cos(twist)*cos(jointangle)
-sin(twist)*cos(jointangle)              0          ];
  tt(3,1:4)=[      0                 sin(twist)                      cos(twist)
offset         ];
  tt(4,1:4)=[      0                   0                               0
1           ];
end
```

R.M is a subroutine which calculate the angular and linear velocities and accelerations for a revolute joint.

```
%*************************************************************************
%                                                                        *
%                              R.M                                       *
%                                                                        *
%*************************************************************************
%calculation of omega(i), omega(i)point, v(i), v(i)point for a revolute joint.
function [V]=R(Q,r,VW,t,i);
A=[t(4*i+1:4*i+3,1:3)]'
z=[0 0 1]';
V(1:3,1)=A*(VW(3*i+1:3*i+3,1)+z*Q(i+1,3));
V(1:3,2)=A*(VW(3*i+1:3*i+3,2)+z*Q(i+1,4))+vectorpr(VW(3*i+1:3*i+3,1),(z*Q(i+1,3)
));
V(1:3,3)=A*VW(3*i+1:3*i+3,3)+vectorpr(V(1:3,1),(A*r(1:3,i+1)));
V(1:3,4)=A*VW(3*i+1:3*i+3,4)+vectorpr(V(1:3,2),(A*r(1:3,i+1)))...
         +vectorpr(V(1:3,1),(vectorpr(V(1:3,1),(A*r(1:3,i+1)))));
→→→→→→→→→→→→→→→
```

T.M is a subroutine which calculate the angular and linear velocities and accelerations for a prismatic joint.

```
%*************************************************************************
%                                                                        *
%                              T.M                                       *
%                                                                        *
%*************************************************************************
%calculation of omega(i),omega(i)point,v(i),v(i)point for a prismatic joint.
function [V]=T(Q,r,VW,t,i);
z=[0 0 1]';
A=[t(4*i+1:4*i+3,1:3)]';
V(1:3,1)=A*VW(3*i+1:3*i+3,1);
V(1:3,2)=A*VW(3*i+1:3*i+3,2);
V(1:3,3)=A*(VW(3*i+1:3*i+3,3)+z*Q(i+1,3))+vectorpr(V(1:3,1),(A*r(1:3,i+1)));
V(1:3,4)=A*(VW(3*i+1:3*i+3,4)+z*Q(i+1,4))+vectorpr(V(1:3,2),(A*r(1:3,i+1)))...

+2*vectorpr(V(1:3,1),(A*z*Q(i+1,3)))+vectorpr(V(1:3,1),vectorpr(V(1:3,1),(A*r(1:
3,i+1))));
```

VECTORPR.M is a subroutine which calculate the vector product.

```
%**************************************************************************
%                                                                         *
%                          VECTORPR.M                                     *
%                                                                         *
%calculate v1^v2=result                                                   *
%                                                                         *
%**************************************************************************
function result = vectorproduct(v1,v2);
   result=zeros(3,1);
     result(1)=v1(2)*v2(3)-v1(3)*v2(2);
     result(2)=v1(3)*v2(1)-v1(1)*v2(3);
     result(3)=v1(1)*v2(2)-v1(2)*v2(1);
```

MAKEQ.M is in fact the input matrix to enter the robot link parameters.

```
%**************************************************************************
%                                                                         *
%                              MAKEQ.M                                    *
%                                                                         *
%**************************************************************************
function [Q]=MAKEQ(n,changeq);
if changeq==1,
       %now you can change the parameters :
       %the first column represents the type of the joint,
       %                    1=revolute
       %                    0=prismatic
       %the second column represents q, the position
       %the third column represents q point, the velocity
       %the fourth column represents q double point, the acceleration.
for i=1:n,
    Q(i,1)=input('revolute=1/prismatic=0 :');
end
for i=1:n,
    Q(i,2)=input('position  q :');
end
for i=1:n,
    Q(i,3)=input('velocity  q point :');
end
for i=1:n,
    Q(i,4)=input('acceleration  q double point :');
end
end
for i=1:n,
    Q(i,1:4)=[Q(i,1) Q(i,2) Q(i,3) Q(i,4)];
end
```

This software allow us to obtain the matrices A(t) and B(t).

The first part is the calculation of the Luh-Walker-Paul's algorithm;

The second part is the linearization about q;

The third part is the linearization about $\dot{q}$;

The fourth part is the linearization about $\ddot{q}$;

The last part is the derivation of the two matrices A(t) and B(t).

```
%*************************************************************************
%                                                                        *
%                              LLLIN.M                                   *
%                                                                        *
%                                                                        *
% Linearization of the                                                   *
% Luh-Walker-Paul's algorithm (relative to links coordinates)            *
% for a 2 D. O. F.                                                       *
%*************************************************************************
format short e
n=2;
Q=[1 0 0 0
   1 0 0 0];
l1=0.500;
l2=0.250;
M=[10
    5];
I=[0     0                  0
   0     M(1,1)*l1^2/12     0
   0     0                  M(1,1)*l1^2/12
   0     0                  0
   0     M(2,1)*l2^2/12     0
   0     0                  M(2,1)*l2^2/12];
VW=zeros(3*(n+1),6);
VW(1:3,4)=[0 -9.80621 0]';
changeq=input('do you want to change Q, yes=1,no=0 ?     ');
```

```
if changeq==1,
[Q]=MAKEQ(n,changeq);
end
%*****************************************************************************
%                                                                            *
%       DENAVIT HARTENBERG PARAMETERS  twist,length,offset,jointangle        *
%                                                                            *
%*****************************************************************************
pp=[0 l1 0 0
    0 l2 0 0];
%
%
i=1;
  if i<=n,
    if Q(i,1)==1,
     pp(1,4)=(pi/180*input('TETA 1, first joint angle  :    '));
     elseif Q(i,1)==0,
     pp(1,3)=input('D1, first joint parameter   :    ');
    end
  end
i=2;
  if i<=n,
    if Q(i,1)==1,
     pp(2,4)=(pi/180*input('TETA 2, second joint angle :    '));
     elseif Q(i,1)==0,
     pp(2,3)=input('D2, second joint parameter   :    ');
    end
  end
i=3;
  if i<=n,
    if Q(i,1)==1,
     pp(3,4)=(pi/180*input('TETA 3, third joint angle  :    '));
     elseif Q(i,1)==0,
     pp(3,3)=input('D3, third joint parameter   :    ');
```

```
    end
  end
i=4;
  if i<=n,
    if Q(i,1)==1,
     pp(4,4)=(pi/180*input('TETA 4, fourth joint angle :     '));
     elseif Q(i,1)==0,
     pp(4,3)=input('D4, fourth joint parameter   :     ');
    end
  end
i=5;
  if i<=n,
    if Q(i,1)==1,
     pp(5,4)=(pi/180*input('TETA 5, fifth joint angle  :     '));
     elseif Q(i,1)==0,
     pp(5,3)=input('D5, fifth joint parameter   :     ');
    end
  end
i=6;
  if i<=n,
    if Q(i,1)==1,
     pp(6,4)=(pi/180*input('TETA 6, sixth joint angle  :     '));
     elseif Q(i,1)==0,
     pp(6,3)=input('D6, sixth joint parameter   :     ');
    end
  end
% r is the distance between O(i) and O(i+1) in frame i
r=[l1*cos(pp(1,4)) l2*cos(pp(2,4))
   l1*sin(pp(1,4)) l2*sin(pp(2,4))
    0                  0          ];
%rc is the distance between O(i) and the centroid mass of the link i in frame i
rc=[-l1/2  -l2/2
       0      0
       0      0];
```

```
%rcc is the distance between O(i-1) and the centroid mass of the link i in frame
i
rcc=[l1/2 l2/2
      0     0
      0     0];
%*************************************************************************
%                                                                        *
%           CALCULATION OF THE HOMOGENEOUS MATRIX                        *
%                                                                        *
%*************************************************************************
tt=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
  for i=0:n-1,
   [t(4*i+1:4*i+4,1:4)]=trans(i,Q,pp(i+1,1),pp(i+1,2),pp(i+1,3),pp(i+1,4));
  end
i=n;
   [t(4*i+1:4*i+4,1:4)]=tt;
t
%*************************************************************************
%                                                                        *
%        CALCULATION OF THE ANGULAR AND LINEAR VELOCITIES, ACCELERATION  *
%                                                                        *
%*************************************************************************
 for i=0:n-1,
    if Q(i+1,1)==1,
      [VW(3*i+4:3*i+6,1:4)]=R(Q,r,VW,t,i);
    elseif Q(i+1,1)==0,
      [VW(3*i+4:3*i+6,1:4)]=T(Q,r,VW,t,i);
    end
end
%*************************************************************************
%                                                                        *
%calculation of the linear and angular velocities of each centroid mass.  *
%                                                                        *
%*************************************************************************
```

```
for i=1:n,
VW(3*i+1:3*i+3,5)=VW(3*i+1:3*i+3,3)+vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i));
VW(3*i+1:3*i+3,6)=VW(3*i+1:3*i+3,4)+vectorpr(VW(3*i+1:3*i+3,2),rc(1:3,i))+...
            vectorpr(VW(3*i+1:3*i+3,1),vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i)));
end
VW
%*************************************************************************************
%                                                                                    *
%calculation of forces and torques                                                   *
%                                                                                    *
%*************************************************************************************
F=zeros(3,n+1);
N=zeros(3,n+1);
F(1,1)=input('x force  :    ');
F(2,1)=input('y force  :    ');
F(3,1)=input('z force  :    ');
N(1,1)=input('x torque :    ');
N(2,1)=input('y torque :    ');
N(3,1)=input('z torque :    ');
 for i=n:-1:1,
  F(1:3,n+2-i)=t(4*i+1:4*i+3,1:3)*F(1:3,n+1-i)+M(i,1)*VW(3*i+1:3*i+3,6);
  N(1:3,n+2-i)=t(4*i+1:4*i+3,1:3)*N(1:3,n+1-i)-vectorpr(rc(1:3,i),...
   t(4*i+1:4*i+3,1:3)*F(1:3,n+1-i))+vectorpr(rcc(1:3,i),F(1:3,n+2-i))+...
    I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,2)+vectorpr(VW(3*i+1:3*i+3,1),...
     (I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,1)));
 end
F
N
%*************************************************************************************
%                                                                                    *
% calculation of torques.                                                           *
%                                                                                    *
%*************************************************************************************
z=[0 0 1]';
```

```
tau=zeros(1,n);
for i=n:-1:1,
  if Q(i,1)==1,
       tau(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*N(1:3,n+2-i);
  elseif Q(i,1)==0,
       tau(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*F(1:3,n+2-i);
  end
end
tau
% ***************************************************************************************
%                                                                                       *
%                              LINEARIZATION   /qi                                      *
%                                                                                       *
%***************************************************************************************
LVW1=zeros(3*(n+1),6);
LVW2=zeros(3*(n+1),6);
D=[0 -1  0
   1  0  0
   0  0  0];
j=1;
for i=0:n-1,
   if Q(i+1,1)==1,
   [LVW1(3*i+4:3*i+6,1:4)]=LR(D,Q,r,VW,LVW1,t,i,j);
%   elseif Q(i+1,1)==0,
%   [LVW1(3*i+4:3*i+6,1:4)]=LT(D,Q,r,VW,LVW1,t,i,j);
end
end
for i=1:n,
a5=LVW1(3*i+1:3*i+3,3);
b5=vectorpr(LVW1(3*i+1:3*i+3,1),rc(1:3,i));
c5=0;
  if j<=i,
    LVW1(3*i+1:3*i+3,5)=a5+b5;
  elseif j>i,
```

```
    LVW1(3*i+1:3*i+3,5)=[0 0 0]';
  end
a6=LVW1(3*i+1:3*i+3,4);
b6=vectorpr(LVW1(3*i+1:3*i+3,2),rc(1:3,i));
c6=vectorpr(LVW1(3*i+1:3*i+3,1),vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i)));
d6=vectorpr(VW(3*i+1:3*i+3,1),vectorpr(LVW1(3*i+1:3*i+3,1),rc(1:3,i)));
  if j<=i,
    LVW1(3*i+1:3*i+3,6)=a6+b6+c6+d6;
  elseif j>i,
    LVW1(3*i+1:3*i+3,6)=[0 0 0]';
  end
end
LVW1
j=2;
for i=0:n-1,
   if Q(i+1,1)==1,
   [LVW2(3*i+4:3*i+6,1:4)]=LR(D,Q,r,VW,LVW2,t,i,j);
%   elseif Q(i+1,1)==0,
%   [LVW2(3*i+4:3*i+6,1:4)]=LT(D,Q,r,VW,LVW2,t,i,j);
end
end
for i=1:n,
a5=LVW2(3*i+1:3*i+3,3);
b5=vectorpr(LVW2(3*i+1:3*i+3,1),rc(1:3,i));
c5=0;
  if j<=i,
    LVW2(3*i+1:3*i+3,5)=a5+b5;
  elseif j>i,
    LVW2(3*i+1:3*i+3,5)=[0 0 0]';
  end
a6=LVW2(3*i+1:3*i+3,4);
b6=vectorpr(LVW2(3*i+1:3*i+3,2),rc(1:3,i));
c6=vectorpr(LVW2(3*i+1:3*i+3,1),vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i)));
d6=vectorpr(VW(3*i+1:3*i+3,1),vectorpr(LVW2(3*i+1:3*i+3,1),rc(1:3,i)));
```

```
  if j<=i,
    LVW2(3*i+1:3*i+3,6)=a6+b6+c6+d6;
  elseif j>i,
    LVW2(3*i+1:3*i+3,6)=[0 0 0]';
  end
end
LVW2
%*****************************************************************************
%                                                                            *
%linearization of forces and torques      /qi                               *
%                                                                            *
%*****************************************************************************
j=1;
   LF1=zeros(3,n+1);
   LN1=zeros(3,n+1);
       for i=n:-1:1,
         A=t(4*i+1:4*i+3,1:3);
         af=D*A*F(1:3,n+1-i);
         bf=A*LF1(1:3,n+1-i);
         cf=M(i,1)*LVW1(3*i+1:3*i+3,6);
                  if j<i+1,
                    LF1(1:3,n+2-i)=bf+cf;
                  elseif j==i+1,
                    LF1(1:3,n+2-i)=af+bf;
                  elseif j>i+1,
                    LF1(1:3,n+2-i)=bf;
                  end
         an=D*A*N(1:3,n+1-i);
         bn=A*LN1(1:3,n+1-i);
         cn=[0 0 0]';
         dn=-vectorpr(rc(1:3,i),D*A*F(1:3,n+1-i));
         en=-vectorpr(rc(1:3,i),A*LF1(1:3,n+1-i));
         fn=[0 0 0]';
         gn=vectorpr(rcc(1:3,i),LF1(1:3,n+2-i));
```

```
        hn=[0 0 0]';
        in=I(3*i-2:3*i,1:3)*LVW1(3*i+1:3*i+3,2);
        jn=vectorpr(LVW1(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,1)));
        kn=[0 0 0]';
        ln=vectorpr(VW(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*LVW1(3*i+1:3*i+3,1)));
                  if j<i+1,
                    LN1(1:3,n+2-i)=bn+en+gn+in+jn+ln;
                  elseif j==i+1,
                    LN1(1:3,n+2-i)=an+bn+dn+en+gn;
                  elseif j>i+1,
                    LN1(1:3,n+2-i)=bn+en+gn;
                  end
      end
LF1
LN1
%
%
j=2;
   LF2=zeros(3,n+1);
   LN2=zeros(3,n+1);
       for i=n:-1:1,
         A=t(4*i+1:4*i+3,1:3);
         af=D*A*F(1:3,n+1-i);
         bf=A*LF2(1:3,n+1-i);
         cf=M(i,1)*LVW2(3*i+1:3*i+3,6);
                  if j<i+1,
                    LF2(1:3,n+2-i)=bf+cf;
                  elseif j==i+1,
                    LF2(1:3,n+2-i)=af+bf;
                  elseif j>i+1,
                    LF2(1:3,n+2-i)=bf;
                  end
         an=D*A*N(1:3,n+1-i);
         bn=A*LN2(1:3,n+1-i);
```

```
        cn=[0 0 0]';
        dn=-vectorpr(rc(1:3,i),D*A*F(1:3,n+1-i));
        en=-vectorpr(rc(1:3,i),A*LF2(1:3,n+1-i));
        fn=[0 0 0]';
        gn=vectorpr(rcc(1:3,i),LF2(1:3,n+2-i));
        hn=[0 0 0]';
        in=I(3*i-2:3*i,1:3)*LVW2(3*i+1:3*i+3,2);
        jn=vectorpr(LVW2(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,1)));
        kn=[0 0 0]';
        ln=vectorpr(VW(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*LVW2(3*i+1:3*i+3,1)));
                if j<i+1,
                  LN2(1:3,n+2-i)=bn+en+gn+in+jn+ln;
                elseif j==i+1,
                  LN2(1:3,n+2-i)=an+bn+dn+en+gn;
                elseif j>i+1,
                  LN2(1:3,n+2-i)=bn+en+gn;
                end
      end
LF2
LN2
%*************************************************************************************
%                                                                                    *
% linearization of torques.       /qi                                              *
%                                                                                    *
%*************************************************************************************
Ltau1=zeros(1,n);
j=1;
for i=n:-1:1,
  if Q(i,1)==1,
      atau=[t(4*i-3:4*i-1,1:3)'*D'*z]'*N(1:3,n+2-i);
      btau=[t(4*i-3:4*i-1,1:3)'*z]'*LN1(1:3,n+2-i);
          if j==i,
                  Ltau1(1,i)=atau+btau;
          elseif j~=i,
```

```
                    Ltau1(1,i)=btau;
            end
  elseif Q(i,1)==0,
       Ltau1(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LF1(1:3,n+2-i);
  end
end
Ltau1
Ltau2=zeros(1,n);
j=2;
for i=n:-1:1,
  if Q(i,1)==1,
       atau=[t(4*i-3:4*i-1,1:3)'*D'*z]'*N(1:3,n+2-i);
       btau=[t(4*i-3:4*i-1,1:3)'*z]'*LN2(1:3,n+2-i);
            if j==i,
                    Ltau2(1,i)=atau+btau;
            elseif j~=i,
                    Ltau2(1,i)=btau;
            end
  elseif Q(i,1)==0,
       Ltau2(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LF2(1:3,n+2-i);
  end
end
Ltau2
%*************************************************************************************
%                                                                                    *
%                          LINEARIZATION   /qi point                                 *
%                                                                                    *
%*************************************************************************************
LLVW1=zeros(3*(n+1),6);
LLVW2=zeros(3*(n+1),6);
j=1;
for i=0:n-1,
   if Q(i+1,1)==1,
   [LLVW1(3*i+4:3*i+6,1:4)]=LLR(Q,r,VW,LLVW1,t,i,j);
```

```
%   elseif Q(i+1,1)==0,
%   [LLVW1(3*i+4:3*i+6,1:4)]=LLT(Q,r,VW,LLVW1,t,i,j);
end
end
for i=1:n,
a5=LLVW1(3*i+1:3*i+3,3);
b5=vectorpr(LLVW1(3*i+1:3*i+3,1),rc(1:3,i));
  if j<=i,
    LLVW1(3*i+1:3*i+3,5)=a5+b5;
  elseif j>i,
    LLVW1(3*i+1:3*i+3,5)=[0 0 0]';
  end
a6=LLVW1(3*i+1:3*i+3,4);
b6=vectorpr(LLVW1(3*i+1:3*i+3,2),rc(1:3,i));
c6=vectorpr(LLVW1(3*i+1:3*i+3,1),vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i)));
d6=vectorpr(VW(3*i+1:3*i+3,1),vectorpr(LLVW1(3*i+1:3*i+3,1),rc(1:3,i)));
  if j<=i,
    LLVW1(3*i+1:3*i+3,6)=a6+b6+c6+d6;
  elseif j>i,
    LLVW1(3*i+1:3*i+3,6)=[0 0 0]';
  end
end
LLVW1
j=2;
for i=0:n-1,
   if Q(i+1,1)==1,
   [LLVW2(3*i+4:3*i+6,1:4)]=LLR(Q,r,VW,LLVW2,t,i,j);
%   elseif Q(i+1,1)==0,
%   [LLVW2(3*i+4:3*i+6,1:4)]=LLT(Q,r,VW,LLVW2,t,i,j);
end
end
for i=1:n,
a5=LLVW2(3*i+1:3*i+3,3);
b5=vectorpr(LLVW2(3*i+1:3*i+3,1),rc(1:3,i));
```

```
  if j<=i,
    LLVW2(3*i+1:3*i+3,5)=a5+b5;
  elseif j>i,
    LLVW2(3*i+1:3*i+3,5)=[0 0 0]';
  end
a6=LLVW2(3*i+1:3*i+3,4);
b6=vectorpr(LLVW2(3*i+1:3*i+3,2),rc(1:3,i));
c6=vectorpr(LLVW2(3*i+1:3*i+3,1),vectorpr(VW(3*i+1:3*i+3,1),rc(1:3,i)));
d6=vectorpr(VW(3*i+1:3*i+3,1),vectorpr(LLVW2(3*i+1:3*i+3,1),rc(1:3,i)));
  if j<=i,
    LLVW2(3*i+1:3*i+3,6)=a6+b6+c6+d6;
  elseif j>i,
    LLVW2(3*i+1:3*i+3,6)=[0 0 0]';
  end
end
LLVW2
%**************************************************************************
%                                                                         *
%linearization of forces and torques   /qi point                         *
%                                                                         *
%**************************************************************************
j=1;
   LLF1=zeros(3,n+1);
   LLN1=zeros(3,n+1);
       for i=n:-1:1,
         A=t(4*i+1:4*i+3,1:3);
             if j<=i,
                LLF1(1:3,n+2-i)=A*LLF1(1:3,n+1-i)+M(i,1)*LLVW1(3*i+1:3*i+3,6);
             elseif j>i,
                LLF1(1:3,n+2-i)=A*LLF1(1:3,n+1-i);
                   end
         an=A*LLN1(1:3,n+1-i);
         bn=-vectorpr(rc(1:3,i),A*LLF1(1:3,n+1-i));
         cn=vectorpr(rcc(1:3,i),LLF1(1:3,n+2-i));
```

```
        dn=I(3*i-2:3*i,1:3)*LLVW1(3*i+1:3*i+3,2);
        en=vectorpr(LLVW1(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,1)));
        fn=vectorpr(VW(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*LLVW1(3*i+1:3*i+3,1)));
                  if j<=i,
                    LLN1(1:3,n+2-i)=an+bn+cn+dn+en+fn;
                  elseif j>i,
                    LLN1(1:3,n+2-i)=an+bn+cn;
                  end
      end
LLF1
LLN1
%
%
j=2;
   LLF2=zeros(3,n+1);
   LLN2=zeros(3,n+1);
       for i=n:-1:1,
         A=t(4*i+1:4*i+3,1:3);
             if j<=i,
                LLF2(1:3,n+2-i)=A*LLF2(1:3,n+1-i)+M(i,1)*LLVW2(3*i+1:3*i+3,6);
             elseif j>i,
                LLF2(1:3,n+2-i)=A*LLF2(1:3,n+1-i);
                   end
         an=A*LLN2(1:3,n+1-i);
         bn=-vectorpr(rc(1:3,i),A*LLF2(1:3,n+1-i));
         cn=vectorpr(rcc(1:3,i),LLF2(1:3,n+2-i));
         dn=I(3*i-2:3*i,1:3)*LLVW2(3*i+1:3*i+3,2);
         en=vectorpr(LLVW2(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*VW(3*i+1:3*i+3,1)));
         fn=vectorpr(VW(3*i+1:3*i+3,1),(I(3*i-2:3*i,1:3)*LLVW2(3*i+1:3*i+3,1)));
                   if j<=i,
                     LLN2(1:3,n+2-i)=an+bn+cn+dn+en+fn;
                   elseif j>i,
                     LLN2(1:3,n+2-i)=an+bn+cn;
                   end
```

```
      end
LLF2
LLN2
%**************************************************************************
%                                                                         *
% linearization of torques.     /qi point                               *
%                                                                         *
%**************************************************************************
LLtau1=zeros(1,n);
j=1;
for i=n:-1:1,
  if Q(i,1)==1,
       LLtau1(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLN1(1:3,n+2-i);
  elseif Q(i,1)==0,
       LLtau1(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLF1(1:3,n+2-i);
  end
end
LLtau1
LLtau2=zeros(1,n);
j=2;
for i=n:-1:1,
  if Q(i,1)==1,
      LLtau2(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLN2(1:3,n+2-i);
  elseif Q(i,1)==0,
      LLtau2(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLF2(1:3,n+2-i);
  end
end
LLtau2
%**************************************************************************
%                                                                         *
%                  LINEARIZATION   /qi double point                       *
%                                                                         *
%**************************************************************************
LLLVW1=zeros(3*(n+1),6);
```

```
LLLVW2=zeros(3*(n+1),6);
j=1;
for i=0:n-1,
   if Q(i+1,1)==1,
   [LLLVW1(3*i+4:3*i+6,1:4)]=LLLR(r,LLLVW1,t,i,j);
%   elseif Q(i+1,1)==0,
%   [LLLVW1(3*i+4:3*i+6,1:4)]=LLLT(r,LLLVW1,t,i,j);
end
end
for i=1:n,
a6=LLLVW1(3*i+1:3*i+3,4);
b6=vectorpr(LLLVW1(3*i+1:3*i+3,2),rc(1:3,i));
  if j<=i,
    LLLVW1(3*i+1:3*i+3,6)=a6+b6;
  elseif j>i,
    LLLVW1(3*i+1:3*i+3,6)=[0 0 0]';
  end
end
LLLVW1
j=2;
for i=0:n-1,
   if Q(i+1,1)==1,
   [LLLVW2(3*i+4:3*i+6,1:4)]=LLLR(r,LLLVW2,t,i,j);
%   elseif Q(i+1,1)==0,
%   [LLLVW2(3*i+4:3*i+6,1:4)]=LLLT(r,LLLVW2,t,i,j);
end
end
for i=1:n,
a6=LLLVW2(3*i+1:3*i+3,4);
b6=vectorpr(LLLVW2(3*i+1:3*i+3,2),rc(1:3,i));
  if j<=i,
    LLLVW2(3*i+1:3*i+3,6)=a6+b6;
  elseif j>i,
    LLLVW2(3*i+1:3*i+3,6)=[0 0 0]';
```

```
  end
end
LLLVW2
%**************************************************************************************
%                                                                                     *
%linearization of forces and torques   /qi double point                              *
%                                                                                     *
%**************************************************************************************
j=1;
   LLLF1=zeros(3,n+1);
   LLLN1=zeros(3,n+1);
       for i=n:-1:1,
         A=t(4*i+1:4*i+3,1:3);
             if j<=i,

LLLF1(1:3,n+2-i)=A*LLLF1(1:3,n+1-i)+M(i,1)*LLLVW1(3*i+1:3*i+3,6);
             elseif j>i,
                LLLF1(1:3,n+2-i)=A*LLLF1(1:3,n+1-i);
                   end
         an=A*LLLN1(1:3,n+1-i);
         bn=-vectorpr(rc(1:3,i),A*LLLF1(1:3,n+1-i));
         cn=vectorpr(rcc(1:3,i),LLLF1(1:3,n+2-i));
         dn=I(3*i-2:3*i,1:3)*LLLVW1(3*i+1:3*i+3,2);
                   if j<=i,
                     LLLN1(1:3,n+2-i)=an+bn+cn+dn;
                   elseif j>i,
                     LLLN1(1:3,n+2-i)=an+bn+cn;
                   end
       end
LLLF1
LLLN1
%
%
j=2;
```

```
  LLLF2=zeros(3,n+1);
  LLLN2=zeros(3,n+1);
      for i=n:-1:1,
        A=t(4*i+1:4*i+3,1:3);
            if j<=i,
             LLLF2(1:3,n+2-i)=A*LLLF2(1:3,n+1-i)+M(i,1)*LLLVW2(3*i+1:3*i+3,6);
            elseif j>i,
             LLLF2(1:3,n+2-i)=A*LLLF2(1:3,n+1-i);
                 end
        an=A*LLLN2(1:3,n+1-i);
        bn=-vectorpr(rc(1:3,i),A*LLLF2(1:3,n+1-i));
        cn=vectorpr(rcc(1:3,i),LLLF2(1:3,n+2-i));
        dn=I(3*i-2:3*i,1:3)*LLLVW2(3*i+1:3*i+3,2);
                  if j<=i,
                    LLLN2(1:3,n+2-i)=an+bn+cn+dn;
                  elseif j>i,
                    LLLN2(1:3,n+2-i)=an+bn+cn;
                  end
      end
LLLF2
LLLN2
%**************************************************************************************
%                                                                                     *
% linearization of torques.      /qi double point                                     *
%                                                                                     *
%**************************************************************************************
LLLtau1=zeros(1,n);
j=1;
for i=n:-1:1,
  if Q(i,1)==1,
       LLLtau1(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLLN1(1:3,n+2-i);
  elseif Q(i,1)==0,
       LLLtau1(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLLF1(1:3,n+2-i);
  end
```

```
end
LLLtau1
LLLtau2=zeros(1,n);
j=2;
for i=n:-1:1,
  if Q(i,1)==1,
      LLLtau2(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLLN2(1:3,n+2-i);
  elseif Q(i,1)==0,
      LLLtau2(1,i)=[t(4*i-3:4*i-1,1:3)'*z]'*LLLF2(1:3,n+2-i);
  end
end
LLLtau2
%*****************************************************************************
%                                                                            *
%                        DETERMINATION OF THE 'A' MATRIX                     *
%                                                                            *
%*****************************************************************************
[A]=MAKEA(n,Ltau1,Ltau2,LLtau1,LLtau2,LLLtau1,LLLtau2);
A
%*****************************************************************************
%                                                                            *
%                        DETERMINATION OF THE 'B' MATRIX                     *
%                                                                            *
%*****************************************************************************
[B]=MAKEB(n,Ltau1,Ltau2,LLtau1,LLtau2,LLLtau1,LLLtau2);
B
```

LLL.M is a subroutine which calculate the linearization of the angular and linear velocities and accelerations about $\ddot{q}$ for a rotational joint.

```
%*************************************************************************
%                                                                        *
%                              LLLR.M                                    *
%                                                                        *
%*************************************************************************
function [LLLV]=LLLR(r,LLLVW,t,i,j);
z=[0 0 1]';
A=[t(4*i+1:4*i+3,1:3)]';
 if j<i+1,
   LLLV(1:3,2)=A*(LLLVW(3*i+1:3*i+3,2));
 elseif j==i+1,
   LLLV(1:3,2)=A*z;
 elseif j>i+1,
   LLLV(1:3,2)=[0 0 0]';
 end
 if j<i+1,
   LLLV(1:3,4)=A*LLLVW(3*i+1:3*i+3,4)+vectorpr(LLLV(1:3,2),(A*r(1:3,i+1)));
 elseif j==i+1,
   LLLV(1:3,4)=vectorpr(LLLV(1:3,2),(A*r(1:3,i+1)));
 elseif j>i+1,
   LLLV(1:3,4)=[0 0 0]';
 end
```

LLR.M is a subroutine which calculate the derivation of the linear and revolute velocities and accelerations about $\dot{q}$ for a revolute joint.

```
%*************************************************************************
%                                                                        *
%                              LLR.M                                     *
%                                                                        *
%*************************************************************************
function [LLV]=LLR(Q,r,VW,LLVW,t,i,j);
z=[0 0 1]';
A=[t(4*i+1:4*i+3,1:3)]';
 if j<i+1,
   LLV(1:3,1)=A*LLVW(3*i+1:3*i+3,1);
 elseif j==i+1,
   LLV(1:3,1)=A*z;
 elseif j>i+1,
   LLV(1:3,1)=[0 0 0]';
 end
 if j<i+1,
   LLV(1:3,2)=A*(LLVW(3*i+1:3*i+3,2)+vectorpr(LLVW(3*i+1:3*i+3,1),z*Q(i+1,3)));;
 elseif j==i+1,
   LLV(1:3,2)=A*vectorpr(VW(3*i+1:3*i+3,1),z);
 elseif j>i+1,
   LLV(1:3,2)=[0 0 0]';
 end
 if j<i+1,
   LLV(1:3,3)=A*LLVW(3*i+1:3*i+3,3)+vectorpr(LLV(1:3,1),(A*r(1:3,i+1)));
 elseif j==i+1,
   LLV(1:3,3)=vectorpr(LLV(1:3,1),(A*r(1:3,i+1)));
 elseif j>i+1,
   LLV(1:3,3)=[0 0 0]';
 end
a4=A*LLVW(3*i+1:3*i+3,4);
b4=vectorpr(LLV(1:3,2),(A*r(1:3,i+1)));
```

```
c4=vectorpr(LLV(1:3,1),vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),A*r(1:3,i+1)));
d4=vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),vectorpr(LLV(1:3,1),A*r(1:3,i+1)));
 if j<i+1,
   LLV(1:3,4)=a4+b4+c4+d4;
 elseif j==i+1,
   LLV(1:3,4)=b4+c4+d4;
 elseif j>i+1,
   LLV(1:3,4)=[0 0 0]';
 end
```

LR.M is a subroutine which calculate linearization ofthe angular and linear velocities and accelerations about q for a revolute joint.

```
%**************************************************************************
%                                                                         *
%                              LR.M                                       *
%                                                                         *
%**************************************************************************
function [LV]=LR(D,Q,r,VW,LVW,t,i,j);
z=[0 0 1]';
A=[t(4*i+1:4*i+3,1:3)]';
a1=A*D'*(VW(3*i+1:3*i+3,1)+z*Q(i+1,3));
b1=A*LVW(3*i+1:3*i+3,1);
 if j<i+1,
   LV(1:3,1)=b1;
 elseif j==i+1,
   LV(1:3,1)=a1;
 elseif j>i+1,
   LV(1:3,1)=[0 0 0]';
 end
a2=A*D'*(VW(3*i+1:3*i+3,2)+z*Q(i+1,4)+vectorpr(VW(3*i+1:3*i+3,1),z*Q(i+1,3)));
b2=A*(LVW(3*i+1:3*i+3,2)+vectorpr(LVW(3*i+1:3*i+3,1),z*Q(i+1,3)));
 if j<i+1,
   LV(1:3,2)=b2;
 elseif j==i+1,
   LV(1:3,2)=a2;
 elseif j>i+1,
   LV(1:3,2)=[0 0 0]';
 end
a3=A*D'*VW(3*i+1:3*i+3,3);
b3=A*LVW(3*i+1:3*i+3,3);
c3=vectorpr(LV(1:3,1),(A*r(1:3,i+1)));
d3=vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),(A*D'*r(1:3,i+1)));
 if j<i+1,
```

```
   LV(1:3,3)=b3+c3;
 elseif j==i+1,
   LV(1:3,3)=a3+c3;
 elseif j>i+1,
   LV(1:3,3)=[0 0 0]';
 end
a4=A*D'*VW(3*i+1:3*i+3,4);
b4=A*LVW(3*i+1:3*i+3,4);
c4=vectorpr(LV(1:3,2),(A*r(1:3,i+1)));
d4=vectorpr(VW(3*(i+1)+1:3*(i+1)+3,2),(A*D'*r(1:3,i+1)));
e4=vectorpr(LV(1:3,1),vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),A*r(1:3,i+1)));
f4=vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),vectorpr(LV(1:3,1),A*r(1:3,i+1)));
g4=vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),vectorpr(VW(3*(i+1)+1:3*(i+1)+3,1),A*D'*r(
1:3,i+1)));
 if j<i+1,
   LV(1:3,4)=b4+c4+e4+f4;
 elseif j==i+1,
   LV(1:3,4)=a4+c4+e4+f4;
 elseif j>i+1,
   LV(1:3,4)=[0 0 0]';
 end
```

MAKEA.M is the subroutine which derive the matrix parameters A(t) at the end of the linearization. see equation 68.

```
%**************************************************************************
%                                                                         *
%                              MAKEA.M                                    *
%                                                                         *
%**************************************************************************
function [A]=MAKEA(n,Ltau1,Ltau2,LLtau1,LLtau2,LLLtau1,LLLtau2);
%
%              determination of the M matrix
%
MM=zeros(n,n);
   MM(1:n,1)=[LLLtau1]';
   MM(1:n,2)=[LLLtau2]';
%
%              determination of the C matrix
%
C=zeros(n,n);
   C(1:n,1)=[LLtau1]';
   C(1:n,2)=[LLtau2]';
%
%             determination of the K matrix
%
K=zeros(n,n);
   K(1:n,1)=[Ltau1]';
   K(1:n,2)=[Ltau2]';
%
%            determination of the A matrix
%
  A(1:n,1:n)=zeros(n);
  A(1:n,n+1:2*n)=eye(n);
  A(n+1:2*n,1:n)=-inv(MM)*K;
  A(n+1:2*n,n+1:2*n)=-inv(MM)*C;
```

MAKEB.M is the subroutine which at the end derive one of the two matrices parameters see equation 68.

```
%**************************************************************************
%                                                                         *
%                    MAKEB.M, determination of the B matrix               *
%                                                                         *
%**************************************************************************
function [B]=MAKEB(n,Ltau1,Ltau2,LLtau1,LLtau2,LLLtau1,LLLtau2);
%
%                    determination of the M matrix
%
MM=zeros(n,n);
   MM(1:n,1)=[LLLtau1]';
   MM(1:n,2)=[LLLtau2]';
%
%                    determination of the B matrix
%
  B(2*n,n)=zeros(2*n,n);
  B(n+1:2*n,1:n)=inv(MM);
```

4. Bibliography

Anex, R. P., and Hubbard, M., "Modeling and adaptive control of a mechanical manipulator", A.S.M.E. *Journal of Dynamic Systems, Measurement, and Control*, 106, 3 (1984).

Asada, H., and Slotine, J. J. E., Robot analysis and control, Massachusetts Institute of Technology, John Wiley and Sons (1986).

Astrom, K.J., "Theory and Applications of Adaptive Control: A Survey", *Automatica*, 19 (5), 1983.

Bejczy, A.K., "Robot Arm Dynamics and Control", *Jet Propulsion Laboratory, California Institute of Technology*, TM 33-69, 1974.

Craig, J. , Introduction to Robotics, Addison-Wesley Publishing Company (1986).

Denavit, J., and Hartenberg, R.S., "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices", *Transaction of the* A.S.M.E., *journal of applied mechanics*, June 1955, pp. 215-221

Dubowsky, S and Desforges, D.T., "The application of model referenced adaptive control to robotic manipulator", *Transaction of the* A.S.M.E., *journal of dynamic systems, measurements and control*, vol 101, sep 1979, pp 193-200.

Hollerbach, J.M., "A Recursive Formulation of Lagrangian Manipulator Dynamics", I.E.E.E. *Trans. Systems,Man,Cybernetics*, Vol 10, No 11, 1980.

Hooker, W.W., and Margulies, G., "The dynamical Attitude equations for an n-body Satellite", *Journal Astronaut Science*, XII (4), 123, Winter, 1965.

Hooker, W.W., "A Set of r Dynamical Attitude Equations for an Arbitrary n-body satellite Having r Rotational Degrees of Freedom", *American Institute of Aeronautics and Astronautics Journal*, Vol. 8, No. 7, 1970, pp. 1205-1207.

Koivo, A.J., and Guo, T.H., "Adaptive linear controller for robotic manipulators", *I.E.E.E. Transaction on automatic control*, vol AC-28, No. 1, february 1883, pp 162-171.

Lee, C. S. G., "Robot Arm Kinematics, Dynamics, and Control", Computer, vol 15, no. 12, pp 62-80 (1982).

Lee, C. S. G., Chung, M.J., Lee,B.H., "Adaptive Control for Robot Manipulators in joint and Cartesian coordinates", I.E.E.E., CH2008, 1984,pp.530-539.

Lee, C. S. G., Gonzalez, R.C. and Fu, K.S., Tutorial on robotics, *I.E.E.E. computer society press*, pp 47-65 (1983).

Luh, J. Y. S., Walker, M. W. and Paul, R. P. C., "On-line Computational Scheme for Mechanical Manipulators", *Transaction of the A.S.M.E., Journal of Dynamic Systems, Measurement, and Control*, vol 102 , June 1980, pp.69-76.

Orin, D.E., McGhee.R.B., Vukobratovic, M., and Hartoch, G., "Kinematic and Kinetic Analysis of Open Chain Linkages Utilizing Newton-Euler Methods", *Math. Biosc.*, Vol. 43, 1979, pp. 107-130.

Orin, D.E., Schrader, W.W., "Efficient Computation of the Jacobian for Robot Manipulators", *Int. J. Robotics Research*, 1984.

Paul, R.P., Robot Manipulators ,Mathematics Programming and Control, *M.I.T. Press*, Cambridge, MA, 1981.

Paul, R.P., Luh, J.Y.S., "Advanced Industrial Robot Control Systems", *Purdue University*, RR78-25, 1978.

Raibert, M.H. and Horn, B.K.P., "Manipulator Control Using the Configuration Space Method", *Industrial Robot*, Vol. 5, 1978.

Renaud, M., "An Efficient Iterative Analytical Procedure for Obtaining a Robot Manipulator Dynamic Model", *Int. Symp. Robot. Res.*, Bretton Woods, 1983.

Shiller, Z. and Dubowsky, S., "On the Optimal Control of Robotic Manipulators with Actuator and End-effector Constraints", I.E.E.E. *Int. Conf. On Robotics and Automation*, 1985

Silver, D.B., " On the Equivalence of the Lagrangian and Newton-Euler Dynamics for Manipulators", *Int. J. Robotics Research*, Vol 1, 1982

Walker, M.W., Orin, D.E., "Efficient Dynamic Computer Simulation of Robotic Mechanisms", *Transaction of the A.S.M.E., journal of dynamic systems, measurements and control*, vol 104, sep 1982, pp 205-211