MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES

3 4456 0366511 1

ORNL/TM-12056

ornl

OAK RIDGE NATIONAL LABORATORY

MARTIN MARIETTA

Development of the Symbolic Manipulator Laboratory Modeling Package for the Kinematic Design and Optimization of the Future Armor Rearm System Robot

S. March-Leuba
J. F. Jansen
R. L. Kress
S. M. Babcock
R. V. Dubey

rpsd

OAK RIDGE NATIONAL LABORATORY
CENTRAL RESEARCH LIBRARY
CIRCULATION SECTION
4500N ROOM 175
LIBRARY LOAN COPY
DO NOT TRANSFER TO ANOTHER PERSON
If you wish someone else to see this report, send in name with report and the library will arrange a loan.

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-12056

526
86

Ammunition Logistics Program

DEVELOPMENT OF THE SYMBOLIC MANIPULATOR LABORATORY MODELING PACKAGE FOR THE KINEMATIC DESIGN AND OPTIMIZATION OF THE FUTURE ARMOR REARM SYSTEM ROBOT

S. March-Leuba
J. F. Jansen
R. L. Kress
S. M. Babcock
Robotics & Process Systems Division

R. V. Dubey
The University of Tennessee
Mechanical & Aerospace Engineering
Knoxville, Tennessee 37996

Date Published–August 1992

Research sponsored by the
Army's Project Manager—Ammunition Logistics
under Interagency Agreement 1892-A078-A1
between the Department of Energy and the
Armament Research Development Engineering Center
at the Picatinny Arsenal

Prepared by the
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
managed by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-84OR21400

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES

3 4456 0366511 1

CONTENTS

Page

LIST OF FIGURES v

ACKNOWLEDGMENTS vii

ABSTRACT ix

1. INTRODUCTION 1
   1.1 SYMBOLIC MANIPULATOR LABORATORY 1
   1.2 MOTIVATION AND OBJECTIVES 2
      1.2.1 Model Development 2
      1.2.2 Easy-to-read Input-Output 2
      1.2.3 Trigonometric Reductions 3
      1.2.4 Kinematics 3
      1.2.5 Inverse Kinematics 3
      1.2.6 Jacobian 3
      1.2.7 Payload and Gravitational Compensation 3
      1.2.8 On-Line Help 4

2. BACKGROUND AND ORIGINAL CONTRIBUTIONS 5
   2.1 LITERATURE REVIEW 5
      2.1.1 Dynamics Programming Review 5
      2.1.2 Kinematics Programming Review 6
   2.2 ORIGINAL CONTRIBUTIONS 7
      2.2.1 Trigonometric Reductions 7
      2.2.2 Homogeneous Transformations and Inverses 8
      2.2.3 Jacobian Written with Respect to Any Frame 8
      2.2.4 Payload and Gravitational Compensation 8
      2.2.5 Input-Output 9
      2.2.6 Interactive Mode Programming with an On-Line Help 9

3. SYMBOLIC SOLVER INTRODUCTION 11
   3.1 INTRODUCTION TO MATHEMATICA AND TO SML 11
   3.2 USAGE OF THE SOLVER: NUMERIC OR SYMBOLIC INPUT-OUTPUT 12
      3.2.1 Input: Denavit-Hartenberg and Mass Parameter Tables 13
      3.2.2 Output: FORTRAN, C, or Text for Papers or Research 16

4. SYMBOLIC SOLVER TOOLS 21
   4.1 KINEMATICS FUNCTIONS 21
      4.1.1 Homogeneous Transformations, Rotational Matrices, and Position Vectors 21
      4.1.2 Direct Kinematics 25
      4.1.3 Inverse Kinematics: Pieper's Solution 27
      4.1.4 Jacobian Written with Respect to An Arbitrary Frame 30

Page

4.2 STATIC FORCES FUNCTIONS 34
4.2.1 Reaction of Joints to Any Force/Torque Vector Applied at Any Coordinate Frame Attached to the Manipulator 37
4.2.2 Necessary Gravitational Compensation at Each Joint 38
4.3 TRIGONOMETRIC REDUCTIONS 38
4.4 MISCELLANEOUS FUNCTIONS 40
4.4.1 On-Line Help 40

5. CONCLUSIONS AND RECOMMENDATIONS 42
5.1 CONCLUSIONS 42
5.2 RECOMMENDATIONS 43

6. REFERENCES 44

APPENDIX A. USER MANUAL 49
A-1. INPUT TABLES 51
A-2. OUTPUT FORMS 52
A-3. FUNCTION TOOLS 53
A-4. TRIGONOMETRIC, MISCELLANEOUS, AND HELP FUNCTIONS 55
APPENDIX B. HELP CODE LISTING 57
B-1. SML-P 59
B-2. SML-C 67
B-3. REDTRIG 75
APPENDIX C. FUTURE ARMOR REARM SYSTEM MANIPULATOR: KINEMATICS, WORK SPACE, STATIC FORCES MODEL, AND DESIGN 79
APPENDIX D. OTHER SAMPLE APPLICATIONS 93
D-1. CENTER FOR ENGINEERING SYSTEMS ADVANCED RESEARCH MANIPULATOR FORWARD AND INVERSE KINEMATICS 95
D-2. LABORATORY TELEROBOTIC MANIPULATOR FORWARD AND INVERSE KINEMATICS AND GRAVITATIONAL COMPENSATION 106

LIST OF FIGURES

Figure **Page**

3.1. Link frames and parameters in Paul's notation 14

3.2. Link frames and parameters in Craig's notation 15

4.1. Roll, pitch, and yaw (XYZ) angles rotated about fixed axes 25

4.2. ZYX Euler angles 26

4.3. Forces and torques applied to a link of a manipulator and the projection onto the Z_i axes 33

4.4. Forces and torques acting on link i 36

C-1. FARS manipulator coordinate system definition 82

C-2. FARS vehicle reloading an M1A1 tank 83

D-1. CESARm coordinate system definition 96

D-2. Teleoperated system: CESARm and the KRAFT master 97

D-3. LTM coordinate system definition 107

ACKNOWLEDGMENTS

The work presented in this technical memorandum was performed primarily for the Robotics & Process Systems Division of Oak Ridge National Laboratory (ORNL), managed by Martin Marietta Energy Systems, Inc., for the U.S. Department of Energy. This research was sponsored by the Armament Research Development Engineering Center at the Picatinny Arsenal.

This research was supported in part by an appointment to the U.S. Department of Energy Postgraduate Research Program at ORNL administered by Oak Ridge Associated Universities.

ABSTRACT

A new program package, Symbolic Manipulator Laboratory (SML), for the automatic generation of both kinematic and static manipulator models in symbolic form is presented. Critical design parameters may be identified and optimized using symbolic models as shown in the sample application presented for the Future Armor Rearm System (FARS) arm.

The computer-aided development of the symbolic models yields equations with reduced numerical complexity. Important considerations have been placed on the closed form solutions simplification and on the user friendly operation.

The main emphasis of this research is the development of a methodology which is implemented in a computer program capable of generating symbolic kinematic and static forces models of manipulators. The fact that the models are obtained trigonometrically reduced is among the most significant results of this work and the most difficult to implement.

Mathematica (Wolfram 1988), a commercial program that allows symbolic manipulation, is used to implement the program package. SML is written such that the user can change any of the subroutines or create new ones easily. To assist the user, an on-line help has been written to make of SML a user friendly package.

Some sample applications are presented. The design and optimization of the 5-degrees-of-freedom (DOF) FARS manipulator using SML is discussed. Finally, the kinematic and static models of two different 7-DOF manipulators are calculated symbolically.

manipulation of expressions and the trigonometric reductions can take advantage of the geometrical configuration of the manipulator, and considerably reduce the complexity of the output. By the author's experience, the symbolic expressions generated by SML appear to be suitable for real time execution. Numerous examples confirmed the expressions being close to minimum executable time. As an illustration, a comparison between the results for the Jacobian of the Jumbo drilling manipulator obtained with the package of Ho and Sriwattanathamma (1989) and with SML showed a reduction of calculation burden, in some expressions, of one and a half to eight times.

1.2 MOTIVATION AND OBJECTIVES

In currently available computer-aided modeling programs of manipulators, only program tasks for a few well-stipulated outputs can be performed. In fact, most previous programs gave only the Jacobian written with respect to the base coordinate frame and the four-by-four homogeneous transformation between the hand and base frames. Because robotics is a fast-growing field, more flexible modeling software is required. For example, to apply force feedback control in telerobotic operations, the Jacobian of the manipulator, written with respect to the coordinate frame where the force/torque sensor is situated, is necessary.

Each year, a multitude of papers are published with new and better robotics modeling and control algorithms. Furthermore, a program for computer-aided generation of manipulator models should improve as robotics technology develops. The program should offer not only specified and well-defined outputs but also accessibility to its subroutines. Moreover, interested researchers should be able to use or change them for specific purposes.

The motivation of this research is to create a program package for the generation of symbolic models of manipulators. The program package should be easy to use, changeable, and extendable. These capabilities will guarantee the utility of the package for both the expert and the novice. The expert will be able to avoid long, complicated calculations; verify previous results; and create new algorithms. The novice will be able to obtain solutions without being a robotics expert or fluent in Mathematica (Wolfram 1988). The specific objectives of the program package presented in this work are detailed in Sections 1.2.1 through 1.2.8.

1.2.1 Model Development

One objective is to develop a program package that can generate, with minimum input, the kinematic and static symbolic models of a general serial link manipulator and the inverse kinematic model for any 6-DOF manipulator with the last three axes intersecting.

1.2.2 Easy-to-Read Input-Output

Another objective is to create simple and understandable output expressions from standard input. The inputs are the parameters from the Denavit-Hartenberg (D-H) Table (Asada and Slotine 1986; Craig 1986; Paul 1981) and, potentially, the mass parameters table of the manipulator. To take advantage of the geometrical configuration of the manipulator, these parameters can be numeric or symbolic. Finally, the user can choose the output to be written in FORTRAN, C, or Text or can create another output form.

1. INTRODUCTION

The symbolic generation of equations has been used extensively by researchers to evaluate control algorithms for robot manipulators. A symbolic expression has advantages over a numerical algorithm in that it permits qualitative relationship and parameter sensitivity algorithm improvement. From the design view point, symbolic models can be studied to identify critical design parameters. Further, the reduced-order model can be generated and studied. For real-time computing, the symbolic equations have the potential of demanding less computer time. Only symbolic formulations can take full advantage of all possible reductions that arise from the geometrical configuration of the manipulator.

Many algorithms have been presented to generate the kinematic and dynamic equations of motion of a manipulator. Some try to automatically produce the equations in symbolic form. It is well known that the development of the symbolic model of a manipulator is an error-prone process. With the recent introduction of 14-degrees-of-freedom (DOF) manipulators, the computation demands are ever increasing. Moreover, automatic computer-aided programs to produce the models are beginning to be necessary and even indispensable for researchers and people in industry.

Currently, most symbolic program packages generate the dynamic models symbolically. Few provide the kinematic model as output, but none provide the static forces at arbitrary locations on the manipulator. The package presented in this work, Symbolic Manipulator Laboratory (SML), can create the kinematic and static models of any general serial link manipulator in symbolic form.

1.1 SYMBOLIC MANIPULATOR LABORATORY

In robot manipulator design, a symbolic manipulator modeling program must be capable of generating complete manipulator models from minimal manipulator descriptions entered by the user. In contrast with classical numerical programming, symbolic programs can deal with algebraic expressions. An internal algebraic representation enables the symbolic program to encode uniquely algebraic terms facilitating the implementation of mathematical operations. To implement the computer-aided generation of manipulator models presented in this report, Mathematica (Wolfram 1988), a high-level symbolic package, was used.

Mathematica allows not only symbolic but also numeric manipulation of equations and matrices. Numeric-symbolic handling of equations takes advantage of reductions due to common terms and multiplication of algebraic terms by numbers. In classical numerical methodologies, a quantity can be calculated along all the modeling process to be finally multiplied, by a zero; or it can be multiplied by series of sines and cosines that could be trigonometrically reduced if the quantity were taken as a common term. All this contributes to a considerable waste of time in the real-time processing of the model. However, a numeric-symbolic methodology accounts for these reductions before the model is implemented in the numeric coprocessor, requiring less control time and improving the behavior of the manipulator.

Another capability of Mathematica is that the user can create new routines with either numeric or symbolic input-output, providing a useful environment for generating numeric-symbolic models.

The program package presented in this work called SML takes advantage of the potential that Mathematica offers. The trigonometric reduction routines play a central role in this program and represent the main contribution of this report. The numeric-symbolic

1.2.3 Trigonometric Reductions

An efficient output is desired for a numerical program with a near-minimum computational time constraint in the output model. To accomplish this goal, trigonometric reductions subroutines have been implemented. They are based on two different kinds of algorithms: pattern matching and exponential reductions.

1.2.4 Kinematics

Production of a complete kinematic model is needed, including: (1) spatial transformations (i.e., homogeneous transformations, rotational matrices, and positional vectors with respect to any manipulator coordinate frame); (2) direct kinematic equations representing the homogeneous transformation and the positional vector; and (3) the Euler angles between the hand and base frames.

1.2.5 Inverse Kinematics

Another objective is to formulate the inverse kinematics of a general 6-DOF manipulator with the three last axes intersecting in a point. The algorithm is based on Pieper's solution (Pieper 1968) as presented by Craig (1986, p. 112).

1.2.6 Jacobian

An important objective is to form the manipulator Jacobian matrix written with respect to any coordinate frame. Two different algorithms are used for the calculations. The first one, discussed by Asada and Slotine (1986, p. 58), is used in SML to calculate the Jacobian written with respect to the base coordinate frame. The second algorithm is based on the relation between the end-effector force and joint torque (force) which is presented by Craig (1986, p. 152). The last one is implemented in SML to calculate the Jacobian written with respect to the last coordinate frame. Further, a subroutine is written to transform the Jacobian with respect to any frame. First, SML checks which coordinate frame (the base or the last) is closer to the frame asked for, and it automatically calculates the Jacobian by one of the two algorithms (whichever is faster). Then, SML transforms the Jacobian written with respect to the specified coordinate frame and reduces it trigonometrically.

1.2.7 Payload and Gravitational Compensation

It is desired to establish an algorithm to find the effect of payload and gravitational forces on the manipulator. The static forces can be written for application against any coordinate frame of the manipulator. An objective is to make it possible to study the effect of external forces at different locations over all the manipulator joints. The gravitational force can be specified in any direction (not only the classical "-Z" axis) so as to be useful in space or mobile vehicle applications. This force can be applied in all or only some of the links for the case of a simplified model.

SML will provide to the user the reaction forces due to external static and gravitational forces over each joint presenting: (1) the three force components, (2) the three torque components along the three Cartesian vectors that constitute each coordinate frame, and (3) the reaction over the manipulator joints.

Outputs 1 and 2 will allow the researcher to know in advance the internal forces produced inside the manipulator. These reactions provoke deflection and torsion of the links of the manipulator and stress of its joints. Further, they can be used in the joint and link stress design. Knowing in advance the value and direction of maximum deflections and torsions on the manipulator, links can be reduced in weight and size. In this way, not only the joints but also the links can be more accurately designed.

1.2.8 On-Line Help

The final objective is to develop an on-line help package that will aid the researcher in learning how to use all the available subroutines. This help feature will be one of the bases for future improvement of this package because the user will be able to extend the help for specific subroutines.

2. BACKGROUND AND ORIGINAL CONTRIBUTIONS

Many algorithms for computer-aided generation of the motion equations of robot manipulators have been presented. Different techniques to generate these equations have been illustrated by several researchers (Asada and Slotine 1986; Craig 1986; Paul 1981). The kinematic equations in them are based on the notation presented in a report by Denavit and Hartenberg (1955), compactness of which offers the ability to create algorithms to obtain automatically the analytical expressions for the manipulator equations of motion. Furthermore, numerous algorithms for computer-aided generation of equations of motion of manipulators in symbolic form have been studied. The most interesting methods presented previously will be discussed in this chapter.

2.1 LITERATURE REVIEW

Symbolic computer packages for modeling manipulators are relatively easy to implement in languages such as LISP (Malm 1984) or PROLOG (Zewari and Zuguel 1986; Borland 1986). But it is better to use a more complete symbolic package like MACSYMA (Symbolics 1985) or Mathematica (Wolfram 1988). The latter is the package used to implement SML. These software packages are easier to work with, so the researcher can dedicate more time to the development of algorithms and output forms rather than to the execution of the symbolic program. In addition, Mathematica offers 2- and 3-dimensional graphic abilities that can be used to plot the manipulator, its performance or work space.

2.1.1 Dynamics Programming Review

Most of the computer automatic generation algorithms implemented previously were written for only finding the dynamic models of manipulators. One of the first programs was Dynamical Models of Industrial Robots (DYMIR) (Vecchio et al. 1980). For DYMIR, the REDUCE symbolic language was used to implement the Lagrangian dynamic formulation. Later, Cesareo, Nicolo, and Nicosia (1984) used DYMIR; and Matsuoka and Citron (1985) and Tzes, Yurkovich, and Langer (1988) used MACSYMA to apply their programs for modeling light, flexible manipulators.

Murray and Neuman (1984a) unveiled the computer program Algebraic Robot Modeler (ARM). ARM generates symbolically the closed-form dynamic equations by four different methodologies: Newton-Euler, Lagrange, and two different Lagrange-Christoffel formulations. Neuman and Murray (1984; 1985; 1987) and Murray and Neuman (1984b) also presented good efficiency comparisons between different dynamic modeling formulations.

Vukobratovic and Kircanski (1984; 1985; 1986), Kircanski and Vukobratovic (1988), and Kircanski et al. (1988) contributed to the symbolic manipulator dynamics modeling programs. For example, Vukobratovic and Kircanski (1984; 1985) introduced a methodology that yields a numeric-symbolic model. Kircanski et al. (1988) presented a program package for both kinematic and dynamic manipulator models. The package produces the homogeneous transformation matrix between the hand and the base coordinate frames, the Jacobian with respect to the hand and base frame of the manipulator, and its dynamic model. The same authors developed the Symbolic Optimizer-Program (SYO), but the output of their program was not completely reduced, because it used the symbolic

expressions as if they were numerical, not taking full advantage of possible trigonometric simplifications.

Some researchers such as Neuman and Murray (1985) and Aldon and Liegeois (1984) have compared the computational requirements of manipulator dynamics formulation for symbolic processing, and others have created their own symbolic algorithms. For example, Cheng, Weng, and Chen (1988) presented their symbolic derivation of dynamic equations of motion using the PIOGRAM symbolic method (Piogram 1964; 1966). Gupta (1987) contributed with the Symbolic Polynomial Technique. This technique was expanded upon by Townsend and Gupta (1989). This method takes advantage of the latest techniques using parallel computing. Different central processing units (CPUs) can calculate different parameters of the dynamic equations of motion at the same time, reducing the total amount of time necessary to calculate the complete model. It uses a combination of the symbolic and numerical approaches, as SML does, treating the variables of the system as symbols but using the numerical values of the constant parameters of the manipulator.

2.1.2 Kinematics Programming Review

There are not as many references for modeling programs of manipulators that provide kinematic information as for the dynamic one. Only a few computer automatic generation algorithms implemented previously have been written to find the kinematic models of manipulators. Most of the algorithms calculated only the homogeneous transformation between the base and the hand coordinate frames (Malm 1984) and the Jacobian of the manipulator with respect to the base or the hand frames (Vukobratovic and Kircanski 1986; 1987; Kircanski et al. 1988). Even though some researchers tried to solve the problem of the trigonometric reductions, a good solution was not found, because the models presented by the outputs of these programs were not completely trigonometrically reduced.

Vukobratovic and Kircanski (1986; 1987) and Kircanski et al. (1988) contributed to the study of kinematics modeling of manipulators. Vukobratovic and Kircanski (1986) reported an interesting study about the minimum amount of computational time necessary to compute the kinematic model. The Jacobian of the manipulator, written with respect to the hand and base frames, was calculated by using the elements of the homogeneous transformation matrices. Moreover, some typical redundancy was reduced in the calculation of the manipulator motion equations.

The best symbolically automated direct kinematic equation solver offered previously was written by Ho and Sriwattanathamma (1989). In their package, Turbo Prolog (Borland 1986) is used to implement a ruled-based program. The input that is entered into the knowledge base of this rule-based program is composed by the D-H Table (Denavit and Hartenberg 1955; Asada and Slotine 1986; Craig 1986; Paul 1981) of the parameters of the manipulator, but it has to be specified whether a joint is revolute or prismatic. The outputs of the program are (1) the direct kinematic equations, (2) the homogeneous transformation between the hand and the base frame of the manipulator, and (3) the Jacobian written with respect to only the end-effector attached coordinate frame. Trigonometric reductions are achieved by pattern matching. This solution has potential problems with long expressions because of the large computational demand, and it has been proven not to work with all the different possible trigonometric combinations. In fact, the output of the rule-based program package (Ho and Sriwattanathamma 1989) is not completely trigonometrically reduced.

Currently, few program packages are capable of generating the inverse kinematic solution in symbolic form. In general, they can deal with only manipulators which have a spherical wrist or that are reducible to this condition. Not all the cases have been proven to be solved. One such package is SRAST (Herrera-Bendezu, Mu, and Cain 1988). The SRAST program gives the solution for the direct and inverse kinematics, and it is implemented in two levels: the C and the LISP levels. The processor generates the symbolic equations at the C level and executes them at the LISP level. SRAST is composed of two parts: SAST, which solves for the direct kinematic equations (Herrera-Bendezu 1985); and INKAS, which gives the solutions, if they exist, of the inverse kinematics (Mu 1987). Another package is based on research reported by Goldenberg, Benhabib, and Fenton (1985).

In parallel to the development of INKAS, the Symbolic Kinematics Inversion Program (SKIP) was developed at the Institute for Robotics and Computer Control (Rieseler and Wahl 1990). Similar to INKAS, SKIP computes the closed-form solution for a given kinematics by using a set of prototype equations with known a priori solutions.

Some publications exist in the area of symbolic programming of dynamic models and only a few for kinematic models; the author was unable to find any references about studies of external forces and gravitational effect on manipulator programming models.

2.2 ORIGINAL CONTRIBUTIONS

The original contributions of this research to the field of computer-aided symbolic modeling of robot manipulators are described in Sections 2.2.1 through 2.2.6.

2.2.1 Trigonometric Reductions

Trigonometric reductions play an important role in robotics modeling, but they have not been solved completely in an automatic fashion. Ho and Sriwattanathamma (1989) presented a symbolically automated solver that was able to reduce trigonometrically its output. Their program package does not give the output completely trigonometrically reduced. On the examples presented in their paper not only the Jacobian, but also the direct kinematic equations for the Standford (Paul 1981), the Jumbo Drilling (Ho and Sriwattanathamma 1989), and the Puma (Craig 1986) robots can be further reduced trigonometrically.

This report presents an important study and solution for this problem because all the possible trigonometric combinations in robotics are taken into account. Thus, the output is reliable to be completely trigonometrically reduced. In SML, two methods to reduce trigonometric expressions are presented.

1. A classical pattern matching is the first method, where expressions are compared and reduced according to the some patterns. This is one of the fastest and most efficient ways to diminish trigonometrically a short expression. The pattern recognition algorithm is used to check all possible combinations inside the expression. However, if an expression is long, the number of combinations is so large that the reduction of an expression can take so much time that the outcome would be worthless or too expensive.
2. An exponential reduction method, based on changing trigonometric expressions to their corresponding pseudo-exponential expressions, is developed on SML.

The second method has proved to work well with long, complicated expressions that the classical method cannot deal with. Instead of checking for any possible

combination that matches one of the patterns, this method transforms every sine, cosine, and tangent in its pseudo-exponential expression. The operations defined for the pseudo-exponential expressions are faster than pattern matching for producing the desired trigonometric reduction, and they give expressions, based on experience, that are close to minimum time solution.

2.2.2 Homogeneous Transformations and Inverses

Most of the symbolic modeling programs presented until now (Ho and Sriwattanathamma 1989; Kircanski et al. 1988) gave only the homogeneous transformations between the base and the end-effector frames of the manipulator. In contrast, SML can give in symbolic form any transformation between coordinate frames attached to any two links of the manipulator. Currently, this is the only package that also gives the inverses of any of these transformations and is possible because of the trigonometric reduction simplification subroutines. These inverses are useful for constructing the inverse kinematic models and sometimes for control algorithms.

2.2.3 Jacobian Written with Respect to Any Frame

To the knowledge of the author, the package presented in this work is the only one that has the capability to find automatically in symbolic form the Jacobian written with respect to any coordinate frame of the manipulator. The Jacobian can be significantly simplified (Dubey et al. 1988; 1989) and be more powerful when written with respect to a different frame rather than to the base or to the end effector. Further, the Jacobian matrix can be used to find the reaction of the joints due to an arbitrary end-point applied force. This force can be defined with respect to any coordinate frame, where the sensor of the manipulator is located [see Craig (1986, p. 152) and Asada and Slotine (1986, p. 77)].

2.2.4 Payload and Gravitational Compensation

Most of the industrial manipulators in the market have simple Proportional-integral-derivative (PID) controllers of the form

$$f = K_p (X_d - X) + K_i \int (X_d - X)\, dt - K_v \dot{X} , \tag{2.1}$$

where X and X_d are the actual and the desired positions respectively, K_p, K_i and K_v are constants and f is the force exerted on the manipulator joint. Corrections to robot controls due to static forces and gravitational effects are the most computational-efficient technique that can be applied with actual microcomputers (see Critchlow 1985, p. 197).

PID controls can do these corrections automatically without requiring all these calculations; usually, just a couple of lines of code is enough. A PID control works well only as long as the manipulator is moving in free space. When the manipulator touches the environment and is still not in the desired position, the integral part of the control builds up to a large force or torque to be applied to the joint actuator (i.e., integral windup effect). This buildup makes the manipulator unstable and even dangerous because it can break itself or the surface that it is touching. The integration effect is even more pronounced in

telerobotic systems, where the master that directs the slave manipulator may be far from the actual manipulator position. The difference in actual position (slave) and desired position (master) causes the integral term in Equation (2.1) to increase in magnitude with time. The result can be a large and potentially dangerous force/torque. Furthermore, it is important to have the model of the reactions at any joint due to external forces and gravitational effect. SML is the only known package capable of creating this model.

The external forces correction and the gravitational effect on the manipulator are calculated directly from Newton-Euler formulation. In addition, the function gives the internal forces/torques applied to each link. This information is useful for studying compression, torsion, deflection, and stress of manipulator links and joints. SML could help researchers to know in advance the location and the direction of these effects in static conditions.

2.2.5 Input-Output

The input of most computer-aided modeling programs is the D-H Table (1955), and joint description (i.e., revolute or prismatic). In this way, only one variable is possible for each row of the D-H Table. This is not the case of the input for SML, where the symbolic subroutines account for this directly from the D-H Table, making possible to be all the parameters of a row variables.

Furthermore, it is also possible to have 2 revolute and 2 prismatic joints in just one row of the D-H Table. This table has four parameters in a row for each joint: q_i, a_i, a_i, and d_i. If a joint is revolute, then a_i, ai, and d_i are constants and q_i is the variable. But if a joint is prismatic, then q_i, a_i, and a_i are constants and d_i isthe variable. To have a revolute-prismatic (cylindrical) joint, the variables are q_i and d_i. Because the input for SML can be independently numeric or symbolic, the user can choose to give either a number or a symbol to any of the constants or variables. Further, the user can choose the number of DOFs of each joint.

For most of the subroutines of SML, the only necessary input is the D-H Table. But for the functions that give the static and gravitational forces model, SML needs also what is called the Mass Table by SML. This table is composed of four parameters for each link: (1) the first one is the link mass and (2) the next three parameters define the location of the link center of mass with respect to the x-, y-, and z-axes of the coordinate frame attached to that link.

The package offers different ways of displaying the output expressions. The standard ones are (1) FORTRAN form; (2) C form; (3) Text form; and (4) Mathematica internal form, which can be used as an input for future calls to other functions. The user can choose the form or even create a new, preferred output form.

2.2.6 Interactive Mode Programming with an On-Line Help

SML is the only package that can be used in both interactive and batch modes. All the packages presented till now worked only in batch mode. In this mode, the input and options are entered into the program to obtain a specific output. When using a batch mode, the input has to be entered each time the program runs. Because SML presents so many options, it was necessary to include the interactive mode. In this mode, each function of the package can be called separately. Then, each output can be analyzed and used as input for the following calls to functions.

To facilitate the user's work, an on-line help was written based on Mathematica's own help. It allows the user to know, at any moment in a session, how to use and call any subroutine or what is the actual numeric-symbolic value for a variable, a vector, or a matrix.

All these new contributions promise to make this package not only a better solution for the problem of manipulator modeling but also a research tool useful for robotic control algorithms.

3. SYMBOLIC SOLVER INTRODUCTION

A new program package, SML, for the automatic generation of both kinematic and static manipulator models in symbolic form is presented. The computer-aided development of these symbolic models yields equations with reduced numerical complexity. Important considerations have been placed on the closed-form-solutions simplification and on the user-friendly operation. The main emphasis of this research is the development of a methodology, which is implemented in a computer program, capable of generating symbolic kinematic and static forces models of manipulators.

The trigonometric reduction is an important result of this work and the most difficult to implement. Previously, only pattern matching has been used. In addition to pattern matching, another method, based in exponential functions, has been implemented in SML. This method drastically reduces the amount of time necessary to produce the model and to perform its numerical computation.

3.1 INTRODUCTION TO MATHEMATICA AND TO SML

Mathematica (Wolfram 1988), a new program that allows symbolic manipulation, is used to implement SML. Versions of Mathematica are available for the Apple Computer, Inc., Macintosh Plus and larger computers, as well as the Macintosh SE/30 and the Macintosh II, IIx, IIcx, and IIci; 386-based MS-DOS systems; Apollo DN 3000 and 4000 systems; Digital Equipment Corporation VAX VMS and ULTRIX, and DECstation; Hewlett-Packard 9000/300 and 800 systems; International Business Machines AIX/RT systems; MIPS systems; NeXT; Silicon Graphics IRIS systems; Sony NEWS systems; and Sun 3,4 and 386i systems. Furthermore, SML can be used in any of these computers, as long as Mathematica is loaded, with the same format and input-output. This ensures not only compatibility but also the ability to be used in a personal computer (PC). In fact, all the work presented in this report was performed on a Macintosh II PC.

SML, the package presented in this work, is written in a way that allows the user to easily change any of the subroutines or to create new ones. Further, the subroutines can be used independently or jointly, giving the user the ability to create work routines. To assist the user, an on-line help has been written to make this package very user friendly.

SML can be used in an interactive mode or in a batch mode. In the interactive mode, each function of the package can be called separately. Then, each output can be analyzed and used as input for the following calls to functions. In the batch mode, a program to call the functions can be easily written by the user. Furthermore, the user can develop new routines and even create new functions that were not in the original package. In this way, the capability of the program can be extended, thus improving the user's efficiency and accuracy.

The SML program package, presented in this paper, takes advantage of the potential that Mathematica offers. The trigonometric reductions routines play a central role in this program and represent the main contribution of this work. The numeric-symbolic manipulation of expressions and the trigonometric reductions can take advantage of the geometrical configuration of the manipulator, reducing enormously the complexity of the output.

Each SML function is a tool that can be used separately and can call automatically other functions necessary to accomplish its goal. There are three different groups of implemented functions.

1. The first group is constituted by kinematic functions. They calculate everything related to kinematics such as homogeneous transformations, direct kinematic equations, Jacobian, and inverse kinematics for serial 6-DOF manipulators.
2. Static forces and gravitational effects functions constitute the second group. In this group, algorithms are performed to find the reaction of the joints of the manipulator to external static forces like payloads and gravitation.
3. The third group is formed by miscellaneous functions like trigonometric reductions, output forms, and auxiliary functions.

SML consists of three different packages: SML-P.m, SML-C.m, and RedTrig.m. As it was explained, Paul's notation (1981) is used by default, but Craig's notation (1986) also can be used in SML. Because usually only one of them is used at a time, a package has been written for each notation. The three packages can be loaded and used at the same time, even though they are not fully independent, because some functions are repeated to make them able to work separately. .

To load any of the packages, they have to be placed or copied first on the folder (directory) "Robotics," which is to be created by the user inside the folder (directory) "Packages" of Mathematica. The next step is to load Mathematica on the computer and then to load SML on Mathematica, typing any of the following:

```
Needs["Robotics`SML-P`"] ,
Needs["Robotics`SML-C`"] ,                                    (3.1)
Needs["Robotics`RedTrig`"] .
```

The first package (SML-P.m) allows the user to use any of the functions described in this report in Paul's notation (1981), and the second one (SML-C.m) does exactly the same but in Craig's notation (1986). Trigonometric reductions are already included in SML-P.m and SML-C.m; but with the third package (RedTrig.m), only the trigonometric reductions and output forms functions of SML are loaded.

All the functions presented in this work are the ones in Paul's notation (1981). To use Craig's functions (1986), add a C to the name of any kinematic and static functions (e.g. OperTransformC, PosC, DirectKinEqC, etc.). Trigonometric and output forms functions are common, so there is no need to add a C to their names.

3.2 USAGE OF THE SOLVER: NUMERIC OR SYMBOLIC INPUT-OUTPUT

Using the tools given by Mathematica, SML can handle numeric and/or symbolic, input-output. Tables for the mass and geometric properties of the manipulator are entered in the form of matrices. The elements of these matrices can be either a number or a symbol.

Numeric-symbolic handling of equations takes advantage of reductions due to common terms and multiplication of algebraic terms by numbers. In classical numerical methodologies, a quantity can be calculated along all the modeling process to be multiplied, at last, by a zero; or it can be multiplied by series of sines and cosines which could be trigonometrically reduced if the quantity were taken as a common term. All this amounts to a considerable waste of time in the real-time processing of the model. Contrarily, a numeric-symbolic methodology takes these reductions into account before the model is implemented in the numeric coprocessor, making the control work faster and improving the

behavior of the manipulator. Symbolic manipulator models created by SML are diminished close to their minimal expressions, resulting in close-to-minimal computer time demand.

3.2.1 Input: Denavit-Hartenberg and Mass Parameter Tables

The input of most of the computer-aided modeling programs is the D-H Table (1955) and joint description; it is either revolute or prismatic. This is not the case for the SML input, where the symbolic subroutines account for this directly from the D-H Table.

Using Paul's notation (1981), Figure 3.1 shows a pair of adjacent links and their associated joints, coordinate frames, and parameters. The D-H Table in Paul's notation is entered in SML as an n-by-4 matrix, where n is the number of coordinate frames associated to links of the manipulator:

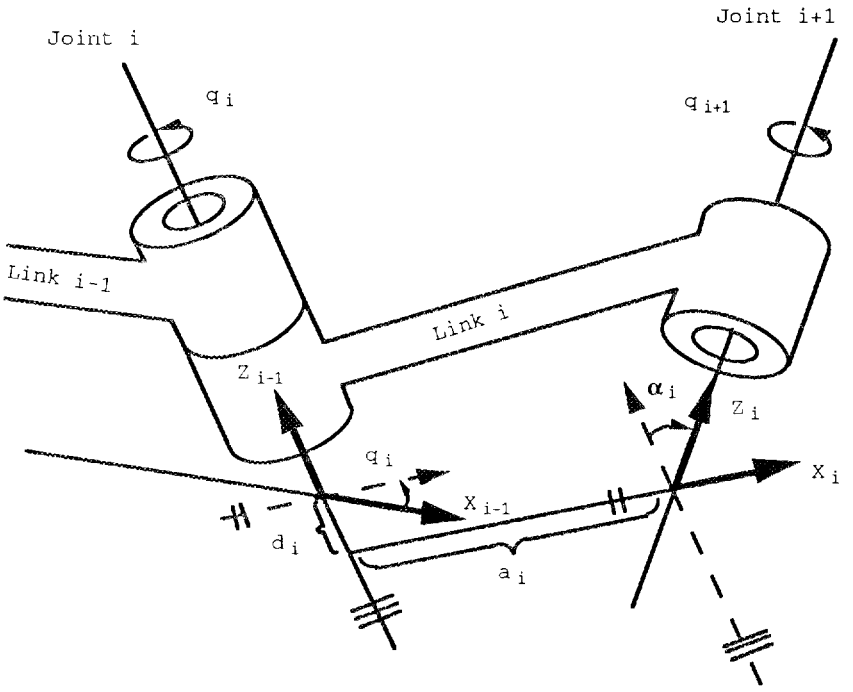
$$\begin{aligned} \text{RobotDHTable} = \{\{&q_1,\ \alpha_1,\ a_1,\ d_1\}, \\ &\{q_2,\ \alpha_2,\ a_2,\ d_2\}, \\ &\quad\vdots \\ &\{q_n,\ \alpha_n,\ a_n,\ d_n\}\}. \end{aligned} \tag{3.2}$$

Figure 3.2 shows a pair of adjacent links and their associated joints, coordinate frames, and parameters using Craig's notation (1986).

The D-H Table in Craig's notation (1986) is also entered in SML as an n-by-4 matrix:

$$\begin{aligned} \text{RobotDHTable} = \{\{&q_1,\ \alpha_0\ ,\ a_0\ ,\ d_1\}, \\ &\{q_2,\ \alpha_1\ ,\ a_1\ ,\ d_2\}, \\ &\quad\vdots \\ &\{q_n,\ \alpha_{n-1},\ a_{n-1},\ d_n\}\}. \end{aligned} \tag{3.3}$$

The principal difference between Paul's notation and Craig's notation is that in the first case frame i is attached to the end of link i, but in the second case it is attached to the beginning of the link. This creates a totally different nomenclature for the kinematics of the manipulator, making a different package necessary for each notation.



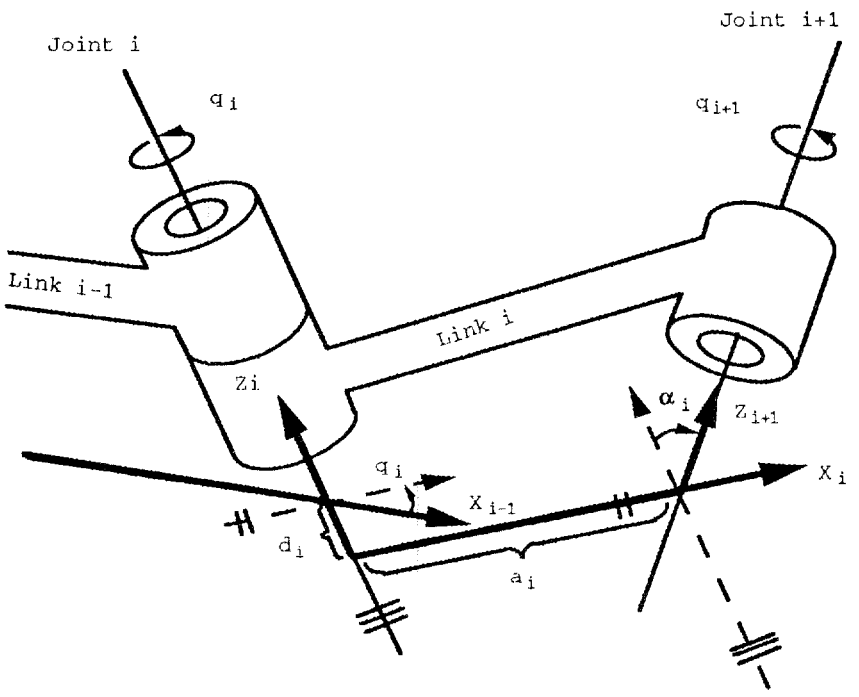
q_i = angle from X_{i-1} to X_i, about Z_{i-1}

α_i = angle from Z_{i-1} to Z_i, about X_i

a_i = length from Z_{i-1} to Z_i, along X_i

d_i = length from X_{i-1} to X_i, along Z_{i-1}

Figure 3.1. Link frames and parameters in Paul's notation.



q_i	=	angle from X_{i-1} to X_i, about Z_i
α_{i-1}	=	angle from Z_{i-1} to Z_i, about X_{i-1}
a_{i-1}	=	length from Z_{i-1} to Z_i, along X_{i-1}
d_i	=	length from X_{i-1} to X_i, along Z_i

Figure 3.2. Link frames and parameters in Craig's notation.

Because any parameter of the D-H Table can be a number or a symbol, it is also possible to have 2- or 3-DOF joints. The D-H Table has four parameters in a row for each joint: q_i, α_i, a_i, and d_i. If a joint is revolute, then α_i, a_i,and d_i are constants and q_i is the variable. But if a joint is prismatic then $q_i = 0$, α_i and a_i are constants, and d_i is the variable. To have a revolute-prismatic (cylindrical) joint, the variables are q_i and d_i. Because the input for SML can be independently numeric or symbolic, the user can choose to give either a number or a symbol to any of the constants or variables. Further, the user can choose the number of DOF of each joint or study the effect of a parameter on the behavior of the manipulator.

For most of the subroutines of SML, the only necessary input is the D-H Table. But for the functions that give the static and gravitational forces model, SML needs also what has been called the Mass Table. This table is composed of four parameters for each link: the first one is the link mass, and the next three parameters define the location of the link center of mass with respect to the X-, Y-, and Z-axes of the coordinate frame attached to that link.

$$\begin{aligned} \text{RobotMassTable:=} \{ & \{m_1, mx_1, my_1, mz_1\}, \\ & \{m_2, mx_2, my_2, mz_2\}, \\ & \vdots \\ & \{m_n, mx_n, my_n, mz_n\}\}. \end{aligned} \tag{3.4}$$

The Mass Table should have the same rows as the D-H Table because each row represents the mass and center of mass of a link where the coordinate frame is attached. Thus, if a link has negligible mass or lengths, a row constituted by zeros should be added at its position in the Mass Table. For example, a joint with two rotational DOFs can be represented by two coordinate frames the origins of which are coincident. In this case, the first link represents a negligible mass link, and a row of zeros should be added to the Mass Table at its position.

3.2.2 Output: FORTRAN, C, or Text for Papers or Research

SML has been implemented from the beginning with the goal of including easy-to-use and multiple options in every function of the package. Different options on the output form give the users more flexibility and accuracy in their work.

Mathematica has specific rules for the form in which the input-output is presented to the computer or user. Like other symbolic languages such as LISP or PROLOG, every call to a function of Mathematica is made by typing its name followed by brackets, inside which are the arguments separated by commas. Even though LISP and PROLOG use parentheses instead brackets, the structure is the same. All the built-in functions in Mathematica use English words with the first letter of each one capitalized. For example, the following function expands products and powers that appear in the numerator of expr:

```
ExpandNumerator[expr_] .                                    (3.5)
```

In modeling manipulators, trigonometric functions are always involved in the equations. For SML to know that cosines, sines, or tangents are functions, the computer needs to represent every trigonometric function in the following mode:

$$\text{Cos[q1], Sin[t1+t2], Tan[q3], Cos[2 t3], etc. ,} \tag{3.6}$$

where q_i or t_i represent an angle in radians (and 2 t3 means 2*t3). This kind of representation is necessary for the computer to be able to operate with these functions. In this form, not only the equations are difficult to read as Text, but they are also incompatible with other numerical languages like C or FORTRAN. Mathematica already provides some basic functions to obtain more compatible kinds of forms, but the output is not the most appropriate for manipulator symbolic modeling. SML presents more attractive outputs through a series of Output Form functions. These new outputs can be just copied into a FORTRAN or a C program to be used for the control of the manipulator.

When a specific symbolic model for a manipulator has been created with SML and after the trigonometric reductions have been applied, a multitude of sines, cosines, and even tangents will be obtained on the equations on the form presented in Equation (3.6). These are not good enough to be implemented in a numerical algorithm in a computer or to be included in a paper or report. The following notation has been used to reduce trigonometric forms in most robotics publications:

$$\begin{aligned} &\text{Cos[q1]} && \text{-> C1} && , \\ &\text{Sin[q1+q2]} && \text{-> S12} && , \\ &\text{Tan[q2]} && \text{-> T2} && . \end{aligned} \tag{3.7}$$

This representation produces a more compact form to be read by the researcher and makes it easier to understand the equations of the model. They are also better suited for implementation in a numerical algorithm because no calculation repetition is made. For example, the expression Cos[q1] is, more than probably, repeated along the manipulator model. Being a trigonometric function, it consumes a lot of CPU time for calculation. Thus, it is interesting to calculate its numerical value only once, call it C1, and use this value throughout the model.

A function called RedAngle has been implemented in SML to reduce the form of trigonometrical functions to the classical nomenclature. RedAngle has three arguments, the second and third ones being optional:

$$\text{RedAngle[expr\_, var\_:q, big\_:0] ,} \tag{3.8}$$

where

- expr is the expression to be reduced, which includes any of the trigonometric functions of the form of Equation (3.6). Expr can be a vector, a matrix, a polynomial, or any kind of expression.
- var defines the angles inside the trigonometric functions used in SML. The default for var is q because most robotic applications publications use it, but any name can be specified by the user as long as it is the same for all the angles inside the expression.
- big is an option to enable RedAngle to deal with subindexes for the angles larger than nine. Big is zero by default, giving any value different from zero enables larger subindexes.

RedAngle can deal with any combination of addition and subtraction of angles inside the trigonometric functions sine, cosine, and tangent. The function sine is reduced to S, cosine to C, and tangent to T. The name q that defines the angles qi in SML disappears. A plus sign is eliminated by default when big is at its default value 0, but it is transformed into P when big is different from 0. This enables users to work with subindexes with more than one digit. A minus sign is always represented by an M. Double, triple, or larger angles are translated using the following notation:

```
2 q1 -> D1;  3 q1 -> T1;  4 q1 -> Q1;  5 q1 -> F1;
6 q1 -> A1;  7 q1 -> B1;  8 q1 -> E1;  9 q1 -> N1; .
```
(3.9)

As an example, Equation (3.10) shows the function RedAngle operating on different expressions:

```
RedAngle[Cos[q1] Sin[q1+q2] + Tan[q2-q3+4 q4]]  ->  C1 S12 + T2M3Q4  ,
RedAngle[a1 Cos[t1] / Sin[t1-2 t2] ,t]          ->  a1 C1 / S1MD2    ,
RedAngle[Sin[q1+q23] + Tan[q2-4 q41], q, 1]     ->  S1P23 + T2MQ41   .
```
(3.10)

RedAngle can be used with any other function that Mathematica offers. An interesting text-form output for robotics application is obtained by using RedAngle and MatrixForm, a built-in function of Mathematica. The following homogeneous transformation A_1^3 between the first and the third coordinate frames of the Puma manipulator was obtained with OperTransform, a function of SML defined in Chapter 4.

$$\text{MatrixForm[RedAngle[A13]]} \rightarrow \begin{bmatrix} C23 & -S23 & 0 & C2\ a2 \\ 0 & 0 & 1 & d3 \\ -S23 & -C23 & 0 & -(S2\ a2) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

Once the trigonometric functions have been reduced to a more compact form, the next important step is to translate the expression to a form compatible with C or FORTRAN. One of the difficulties when translating to any of these languages is that to enter a multidimensional vector, the subindexes have to be specified to assign each value to a different memory allocation. Both languages use different notations: (1) the first index allocation for C is zero but for FORTRAN is one and (2) C uses a pair of brackets for each index "List[0][2]," but FORTRAN needs a list of the subindexes separated by commas inside parentheses "List(1,3)."

A function called ListOutput has been created which prints its first argument regardless of its dimensions. This function checks the kind of input that is entered; it is either a vector, a matrix, or an expression; and it automatically calls the necessary function that will print it out in the specified output form. ListOutput has four parameters, the last three being optional:

```
ListOutput[expr_, name_:"List", form_:Text, var_:q, big_:0] ,
```
(3.12)

where

- expr is the expression to be printed in the specified form. Expr can be either a vector, a matrix, or an expression.
- name is a string that gives the name that will be used for the listing. The default for name is "List."
- form is the desired form in which the output will be printed. Its default is Text, but the following options are available:
  1. form = Text gives expression in Text form. This is probably the easier to read, but it is not good enough to copy and paste to another program such as in a word processor, when powers or divisions are present.
  2. form = C gives the expression in C form.
  3. form = RC prints the expression in C language, reducing the form of sines, cosines, and tangents by using RedAngle.
  4. form = F prints the expression in FORTRAN form.
  5. form = RF gives the expression in FORTRAN, reducing the form of sines, cosines, and tangents.
- var is the same argument as in RedAngle. This argument needs to be specified only when asking for Reduced FORTRAN (RF), Reduced C (RC), or Text forms and a different name that q_i has been used for the angles in the expression.
- big is the option that enables ListOutput to deal with subindexes for the angles larger than nine. Big is zero by default, giving any value different from zero enables larger subindexes.

The forms defined as RC or RF are the most powerful that ListOutput provides. They are not only easy to read, but they can also be copied and pasted to other programs like word processors or program editors in text mode without any problem.

As an example of the output obtained with ListOutput, the following represents the Jacobian of a two-link planar manipulator expressed in four different forms:

1. Mathematica and SML form:

```
Jacob -> {{- L1 Sin[q1]  - L2 Sin[q1+q2], - L2  Sin[q1+q2]},
          { L1 Cos[q1] + L2 Cos[q1+q2],   L2 Cos[q1+q2]}}.
```
(3.13)

2. Text form obtained from ListOutput by default:

```
ListOutput[Jacob] -> List(1,1) = - (L1 S1) - L2 S12
                     List(2,1) =   L1 C1 + L2 C12
                     List(1,2) = - (L2 S12)
                     List(2,2) =   L2 C12 .
```
(3.14)

3. Reduced C form:

```
ListOutput[Jacob,"Jac",RC] -> Jac[0][0] = - (L1*S1) - L2*S12 ;
                              Jac[1][0] =   C1*L1  + C12*L2 ;
                              Jac[0][1] = - (L2*S12);
                              Jac[1][1] =   C12*L2 ; .
```
(3.15)

4. Reduced FORTRAN form:

```
ListOutput[Jacob,"Jac",RF] -> Jac(1,1) = - (L1*S1) - L2*S12
                              Jac(2,1) =   C1*L1  + C12*L2
                              Jac(1,2) = - (L2*S12)
                              Jac(2,2) =   C12*L2 .
```
(3.16)

It is important to note the difference when operating on powers and divisions. The C language has the function "pow(x,y)" to calculate x to the power of y, but in FORTRAN syntax "x**y" has to be specified. The output produced by ListOutput using any of the FORTRAN or C options is perfectly compatible with any editor or word processor. This is not the case when using the option Text, which is compatible with only certain editors like Expressionist.
The following are examples of three different outputs obtained for

ListOutput [Cos[q1]/Cos[q2]^2, "Example", form option]. (3.17)

1. Text option: $\text{Example} = \dfrac{C1}{S2^2}$

2. RC option: Example = C1*pow(S2,-2) ;

3. RF option: Example = C1/S2**2 .

Both functions, RedAngle and ListOutput, have on-line help. The user just needs to type "?RedAngle" or "?ListOutput" to obtain an explanation about the function and its parameters. More kinds of outputs are built into Mathematica. Especially interesting for robotics are the functions MatrixForm and ColumnForm that Mathematica offers.

4. SYMBOLIC SOLVER TOOLS

The goal in this chapter is to develop a series of functions, in a computer package, for use in modeling a general serial link robot manipulator. Each of these functions is a tool that can be used separately and that can call automatically other tools necessary to accomplish its goal. Three different groups of functions exist in SML:

1. The first group is constituted by kinematic functions. They calculate everything related to kinematics such as homogeneous transformations, direct kinematic equations, Jacobian, and inverse kinematics for serial 6-DOF manipulators.
2. Static forces and gravitational effect functions constitute the second group. In this group, algorithms are performed to find the reaction of the joints of the manipulator to external static forces like payloads and gravitation.
3. The third group is formed by miscellaneous functions like trigonometric reductions, output forms, and auxiliary functions.

Even though the trigonometric reduction functions presented in Chapter 5 can be applied to the output of any of the following functions, it has been found to be more effective to include this kind of reductions inside some of the functions of SML. Moreover, taking advantage of the geometrical configuration of the manipulator in each function algorithm improves its speed.

4.1 KINEMATICS FUNCTIONS

Kinematics is the science of motion which treats motion without regard for the forces that cause it. In this section, kinematic functions are developed to symbolically compute the position and orientation of any coordinate frame of the manipulator with respect to any desired coordinate frame. Also included are functions to calculate the Jacobian of the manipulator with respect to an arbitrary coordinate frame, and the inverse kinematics of a 6-DOF manipulator.

The input for each function is the D-H table, as stated in Chapter 3, except for the function that calculates the effect of gravity because the mass parameters table is also needed.

4.1.1 Homogeneous Transformations, Rotational Matrices, and Position Vectors

A homogeneous transformation A_A^B is a four-by-four matrix that describes the position and orientation of a coordinate frame (B) with respect to another frame (A). It is composed of a three-by-three rotational unitary matrix R_A^B and of a three-by-one position vector P_A^B relating both coordinate frames as shown in Equation (4.1).

$$A_A^B = \left[\begin{array}{c|c} R_A^B & P_A^B \\ \hline 0 \quad 0 \quad 0 & 1 \end{array}\right]. \tag{4.1}$$

Homogeneous transformations can be multiplied as general four-by-four matrices. Thus, having A_A^B and A_B^C, then coordinate frame (C) can be written with respect to frame (A) multiplying both matrices:

$$A_A^C = A_A^B \, A_B^C \quad . \tag{4.2}$$

As stated in Chapter 3, Paul's notation (1981) is, by default, the only one described in this report, but Craig's notation (1986) can also be used in SML. Paul's homogeneous transformation matrix between two consecutive coordinate frames is given by

$$A_{i-1}^i = \begin{bmatrix} \mathrm{Cos}[\theta_i] & -\mathrm{Sin}[\theta_i]\,\mathrm{Cos}[\alpha_i] & \mathrm{Sin}[\theta_i]\,\mathrm{Sin}[\alpha_i] & a_i\,\mathrm{Cos}[\theta_i] \\ \mathrm{Sin}[\theta_i] & \mathrm{Cos}[\theta_i]\,\mathrm{Cos}[\alpha_i] & -\mathrm{Cos}[\theta_i]\,\mathrm{Sin}[\alpha_i] & a_i\,\mathrm{Sin}[\theta_i] \\ 0 & \mathrm{Sin}[\alpha_i] & \mathrm{Cos}[\alpha_i] & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.3}$$

where the parameters of the matrix are the same as those presented in Chapter 3, Figure 3.1, for the D-H Table using Paul's notation (1981).

Note that Equation (4.3) represents the homogeneous transformation that gives the position and orientation of a coordinate frame i written with respect to its preceding coordinate frame i-1. If frame i-1 written with respect to frame i is needed, then the inverse of A_{i-1}^i has to be calculated. A general relation between a homogeneous transformation and its inverse is used in SML to save calculation time:

$$A_B^A = \left[A_A^B\right]^{-1} = \left[\begin{array}{c|c} \left[R_A^B\right]^T & -\left[R_A^B\right]^T P_A^B \\ \hline 0 \quad 0 \quad 0 & 1 \end{array}\right]. \tag{4.4}$$

The function OperTransform of SML gives the homogeneous transformation that relates any two coordinate frames of the manipulator. Frames are called from 0 through N, corresponding with their order in the given D-H Table. The 0 coordinate frame is the base frame of the manipulator. If it were necessary to relate the manipulator to a fixed coordinate frame other than the base, it could be done just adding more rows of fixed

parameters at the beginning of the D-H Table. The function OperTransform is called with three arguments:

$$\text{OperTransform[DHTable\_, RefFrame\_, Frame\_]}\,, \tag{4.5}$$

where

- DHTable is the name given to the D-H Table of the manipulator, that should be entered as shown in Equation (3.2); and
- the output of OperTransform is the homogeneous transformation that relates Frame to RefFrame written with respect to RefFrame: $A^{Frame}_{RefFrame}$.

OperTransform uses Equations (4.2) and (4.3) to calculate, symbolically, the homogeneous transformation between Frame and RefFrame. If Frame is bigger than RefFrame, then it calculates first $A^{RefFrame+1}_{RefFrame}$, then $A^{RefFrame+2}_{RefFrame+1}$, and after that their multiplication. The function continues n steps until RefFrame + n = Frame:

$$A^{Frame}_{RefFrame} = A^{RefFrame+1}_{RefFrame} \cdot A^{RefFrame+2}_{RefFrame+1} \cdots A^{Frame}_{RefFrame+n-1} \,. \tag{4.6}$$

If Frame is smaller than RefFrame, then the function first calculates $A^{RefFrame}_{RefFrame-1}$ and then its inverse $A^{RefFrame-1}_{RefFrame}$, using Equation (4.4). OperTransform continues n steps until the homogeneous transformation, is found, that relates Frame to RefFrame written with respect to RefFrame:

$$A^{Frame}_{RefFrame} = A^{RefFrame-1}_{RefFrame} \cdot A^{RefFrame-2}_{RefFrame-1} \cdots A^{Frame}_{RefFrame-n+1} \,. \tag{4.7}$$

OperTransform always gives complete trigonometrically reduced output. It has been found that long and complex expressions are tremendously time-consuming when reducing them trigonometrically. It is better to reduce the expressions used along the algorithm of the function rather than reduce the long final expression. Furthermore, trigonometric reductions are performed each time an operation between matrices and vectors is produced.

Each time two matrices are multiplied together, OperTransform reduces trigonometrically their product. Each time an inverse is calculated, this function reduces the product of the transpose of the rotational matrix with the position vector [see Equation (4.4)]. It has been found that only five possible trigonometric combinations can appear when multiplying two homogeneous transformations or calculating its inverse. Those are the sine and cosine of the addition or subtraction of two angles and the addition of the squares of cosine and sine. OperTransform reduces these expressions according to the five following patterns:

```
a_. Sin[x_] Cos[y_] + a_. Cos[x_] Sin[y_]  ->  a Sin[x + y]
a_. Sin[x_] Cos[y_]  - a_. Cos[x_] Sin[y_]  ->  a Sin[x - y]
a_. Cos[x_] Cos[y_] - a_.  Sin[x_] Sin[y_]  ->  a Cos[x + y]        (4.8)
a_. Cos[x_] Cos[y_] + a_. Sin[x_] Sin[y_]  ->  a Cos[x - y]
a_. (Cos[x_])^2    +   a_. (Sin[x_])^2        ->    a
```

The function OperTransform can keep in memory any of the homogeneous transformations that it calculated during a Mathematica session. Any kinematic or dynamic formulation is based on these matrices, so calculating them each time they are needed would be a waste of time.

A function associated with OperTransform is OperTransformAux, which is called with the same arguments as the first one. This function calculates any homogeneous transformation, or its inverse, between two consecutive coordinate frames and keeps them in memory. OperTransform makes calls automatically to this auxiliary function each time it is needed, but it keeps in memory only the homogeneous transformation requested by the user. In this way, all the simple transformations are saved in memory already trigonometrically reduced and ready to be used by OperTransform, reducing enormously the computational burden. As an example, the calculation of the inverse of the homogeneous transformation A_0^6 of a 6-DOF manipulator (the KRAFT master) without using the auxiliary function required 664 seconds on a Macintosh II computer, while only 53 seconds were necessary when using OperTransformAux.

Two other functions of SML are directly related to OperTransform: (1) the function Rot, that gives the three-by-three rotational unitary matrix $R_{RefFrame}^{Frame}$ which describes Frame written with respect to RefFrame; and (2) the function Pos, the output of which expresses the three-by-one position vector $P_{RefFrame}^{Frame}$ from the origin of RefFrame to the origin of Frame, written with respect to RefFrame. Both are called with three arguments, the same as those for OperTransform:

```
Rot[DHTable_, RefFrame_, Frame_]
Pos[DHTable_, RefFrame_, Frame_] .
```
(4.9)

These two functions call first OperTransform[DHTable, RefFrame, Frame], and then they create their own outputs from the four-by-four matrix. Furthermore, the expressions given by Rot and Pos are also completely trigonometrically reduced.

Note that with the Rot and Pos functions, different interesting combinations can be found. As an example, the position vector from coordinate frame 2 to frame 6 of a manipulator, defined by the D-H Table called RobotTable and written with respect to its base frame is found by:

$$^0P_2^6 = \text{Rot[RobotTable, 0, 2] . Pos[RobotTable, 2, 6]} . \tag{4.10}$$

The position vector found after this multiplication is not necessarily completely trigonometrically reduced. Further reduction can be obtained by using the functions RedTrig or RedTrigExp, presented in Chapter 5, by

$$\text{RedTrigExp[} ^0P_2^6 \text{]} \tag{4.11}$$

because based on OperTransform, the functions Rot and Pos can also keep in memory any of the matrices or vectors, respectively, that they calculated.

4.1.2 Direct Kinematics

The kinematic equations of a manipulator arm provide the functional relationship between the end-effector position and orientation and the displacements of all the joints involved in the open kinematic chain.

The kinematic equations are nothing more than the homogeneous transformation relating the coordinate frame attached to the last link of the manipulator with the base coordinate frame. Let us call q_i the displacement of each joint as either an angle or a length. If the manipulator has n joints, then applying Equation (4.2), the kinematic equations of the manipulator become:

$$T = A_0^1(q_1)\ A_1^2(q_2)\ \cdots\ A_{n-1}^n(q_n)\ . \tag{4.12}$$

To find the operator T with SML, the user just needs to use the function OperTransform as shown in Equation (4.13),

$$T = \text{OperTransform[DHTable, 0, n]}\ . \tag{4.13}$$

From the matrix T, can be deduced the position and orientation of frame "n" with respect to the base frame . The position, in Cartesian coordinates, is given by its position vector: $[P_0^n]^T = [P_x, P_y, P_z]$. The orientation is taken from its rotation matrix R_0^n in the form of angles rotated about the coordinated axes. Note that the nine elements of a rotation matrix are not independent, because they are subject to orthogonality conditions and the unitary vector length conditions. Because six conditions exist, only three parameters are independent. These parameters are usually defined as three angles rotated about the Cartesian axes, but several combinations are possible. The most common representations are solved in SML and presented here.

Three different methods of describing the orientation of a coordinate frame that are generally used in robotics are included in SML, so the user may choose one. The three methods will be presented now (see also Asada and Slotine 1986; Craig 1986; Paul 1981).

1. Roll, pitch, and yaw angles about fixed axes
   Start with the frame (B) coincident with a known reference frame (A). Rotate first frame (B) about X_A by an angle yaw(γ), then rotate about Y_A by an angle pitch(β), and then rotate about Z_A by an angle roll(α) (see Figure 4.1).

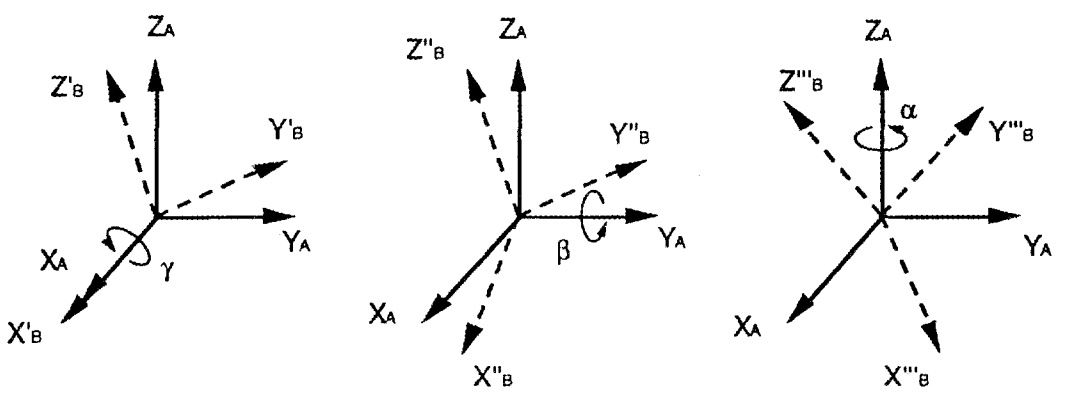


Figure 4.1. Roll, pitch, and yaw (XYZ) angles rotated about fixed axes.

2. Z-Y-X Euler angles
   Start with the frame coincident with a known frame (A). Rotate first frame (B) about Z_B by an angle roll(α), then rotate about the new Y_B by an angle pitch(β), and then rotate about the new X_B by an angle yaw(γ) (see Figure 4.2).

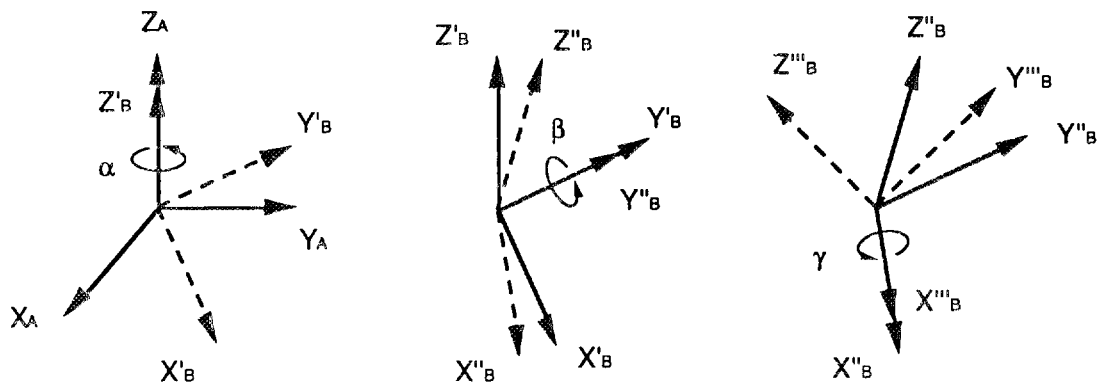


Figure 4.2. ZYX Euler angles.

3. Z-Y-Z Euler angles
   Start with the frame coincident with a known frame (A). Rotate first frame (B) about Z_B by an angle α, then rotate about the new Y_B by an angle β, and then rotate about the new Z_B by an angle γ.

The reason so many different descriptions exist is that no one of them is perfect. All descriptions work perfectly in a range for pitch (the second angle) of slightly less than 180°, but the real problem is that all have singular points. The orientation defined by roll, pitch, and yaw angles about fixed axes and the orientation defined by the Z-Y-X Euler angles have a singularity at pitch $= \pm 90°$. The orientation defined by the Z-Y-Z Euler angles has a singularity at pitch $= \pm 180°$. When the end-effector orientation is closed to one of the singularities, then the solution for the angles degenerates. This effect makes the manipulator uncontrollable because the torques or forces applied can be incremented enormously.

The function DirectKinEq of SML calculates automatically the orientation angles. It gives not only the position but also three different types of orientations. DirectKinEq is called with four arguments, the last three of which are optional:

DirectKinEq[DHTable_, BaseFrame_:0, LastFrame_:n, EulerOrder_:ZYX] , (4.14)

where

- DHTable is the name given to the D-H Table of the manipulator.
- BaseFrame is by default the 0 coordinate frame, but a different one can be specified by the user.

- LastFrame is by default the last coordinate frame of the DHTable.
- EulerOrder is the order of the Euler angles. Two orders can be used: (1) ZYX is the default, where the function gives the ZYX Euler angles or the XYZ angles about fixed axes (both solutions are the same) and (2) ZYZ to obtain the ZYZ Euler angles.

Sometimes the user is interested in knowing the position and orientation of a link other than the last one. For example, a 7-DOF manipulator may have a sensor on the base of its spherical wrist (three rotational DOFs with their axes intersected). A new fifth coordinate frame can be attached to the sensor that will relate its position to the fourth frame, which belongs to the fourth link. The new frame should be added to the D-H Table of the manipulator as a row of constant parameters in the fifth position of this table. To find the position and orientation of the monitored forces and/or torques written with respect to the base frame, the user just needs to call

DirectKinEq[NewDHTable, 0, 5] . (4.15)

If needed, the position vector can be found by calling Pos[NewDHTable, 0, 5] and the rotation matrix, instead the rotated angles, using Rot[NewDHTable, 0, 5].

4.1.3 Inverse Kinematics: Pieper's Solution

Although the inverse kinematics of a completely general 6-DOF robot manipulator does not have a closed-form solution, certain important special cases can be solved. Pieper (1968) studied 6-DOF manipulators with three consecutive rotational axes intersected. Pieper's work applies not only to all rotational axes but also to other configurations that include prismatic joints.

In this report, only the solution for the inverse kinematics of 6-DOF manipulators with the last three rotational axes intersected is presented. More in-depth studies were made on other program packages such as INKAS (Mu 1987) and SKIP (Rieseler and Wahl 1990), and the interested researcher should refer to them. With the solution presented in this work, the basics for the solvability of a more general manipulator are given. Anyway, most available 6-DOF manipulators have a spherical wrist and can be solved with the method presented here.

The function InverseKin of SML calculates automatically, when it exists, the inverse kinematics of a 6-DOF manipulator of which the last three axes intersect. InverseKin is called with just one argument:

InverseKin[DHTable_] , (4.16)

where

- DHTable is the name given to the D-H Table of the manipulator. Because the robot manipulator is 6 DOF with a spherical wrist, its D-H Table will, in general, be presented by Equation (4.17).

$$\text{RobotTable} = \begin{array}{l} \{\{\ q_1,\ \alpha_1,\ a_1,\ d_1\ \}, \\ \{\ q_2,\ \alpha_2,\ a_2,\ d_2\ \}, \\ \{\ q_3,\ \alpha_3,\ a_3,\ d_3\ \}, \\ \{\ q_4,\ \alpha_4,\ a_4,\ d_4\ \}, \\ \{\ q_5,\ \alpha_5,\ 0\ ,\ 0\ \}, \\ \{\ q_6,\ \alpha_6,\ 0\ ,\ 0\ \}\}. \end{array} \tag{4.17}$$

In a manipulator with a spherical wrist, q_4, q_5, and q_6 are the variables corresponding to the last three revolute link joints. The first three joints may be either revolute or prismatic and their variables either q_i or d_i respectively. If joint i is prismatic, then q_i is constant and d_i is the variable; but if joint i is revolute, then q_i is the variable.

The function InverseKin assumes as a known input the homogeneous transformation between the hand and base frames of the manipulator. Lets us call T_0^6 for this transformation written with respect to the base coordinate frame. The output offered by InverseKin is referred to the matrix presented in Equation (4.18).

$$T_0^6 = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.18}$$

When the three last axes intersect, the origin of coordinate frames attached to links 4, 5, and 6 are located at the point of intersection. The position vector of this point written with respect to the first coordinate frame is found by using the functions Pos and Rot of SML:

$$^1P_0^4 = \text{Rot[RobotTable, 1, 0]}\ .\ \text{Pos[RobotTable, 0, 4]}\ . \tag{4.19}$$

For the general case table presented in Equation (4.17), the vector $^1P_0^4$ gives the expression

$$\begin{aligned} P4_x &= a1 + C2\ (a2 + C3\ a3) + S2\ (-(C\alpha 2\ S3\ a3) + S\alpha 2\ d3); \\ P4_y &= S\alpha 1\ (-(S3\ S\alpha 2\ a3) - d2 - C\alpha 2\ d3) + \\ &\quad C\alpha 1\ (S2\ (a2 + C3\ a3) + C2\ (C\alpha 2\ S3\ a3 - S\alpha 2\ d3)); \\ P4_z &= C\alpha 1\ (S3\ S\alpha 2\ a3 + d2 + C\alpha 2\ d3) + \\ &\quad S\alpha 1\ (S2\ (a2 + C3\ a3) + C2\ (C\alpha 2\ S3\ a3 - S\alpha 2\ d3)); \end{aligned} \tag{4.20}$$

where Ci and Si and Cαi and Sαi are the cosine and sine of angles qi and αi respectively. Also, the position vector $^1P_0^4$ shown in the above equation should be equal to

$$^1P_0^4 = R_1^0\ (^0P_0^6 - {}^0P_4^6)\ , \tag{4.21}$$

where ${}^0P_0^6 = [P_x, P_y, P_z]^T$. Thus, the left-hand sides of Equation (4.20) can be substituted with:

$$
\begin{aligned}
P4_x &= C1\ Px^* + Py^*\ S1 \\
P4_y &= C\alpha 1\ (C1\ Py^* - Px^*\ S1) + S\alpha 1\ Pz^* \\
P4_z &= C\alpha 1\ Pz^* + S\alpha 1\ (-(C1\ Py^*) + Px^*\ S1)\ ,
\end{aligned}
\tag{4.22}
$$

where

$$
\begin{bmatrix} Px^* \\ Py^* \\ Pz^* \end{bmatrix} = \begin{bmatrix} Px - d4\ a_x \\ Py - d4\ a_y \\ Pz - d4\ az \end{bmatrix} .
\tag{4.23}
$$

Equations (4.20) and (4.22) create a system of three equations in which the unknowns are the variables corresponding to the first three joints of the manipulator. The inverse kinematics problem has been changed from finding six unknowns to two problems of finding three unknowns.

To complete the solution, InverseKin needs to solve for angles q4, q5, and q6. Computation can be based upon only the rotation matrix RT_0^6 of the specified goal T_0^6 presented in Equation (4.18). Because the first three joint variables have already been solved, the rotation matrix R_0^3 is also known. The following matrix equation gives nine equations that can be solved for the three angles q4, q5, and q6:

$$
R_3^6 = \left[R_0^3\right]^{-1}\ RT_0^6 = R_3^0\ RT_0^6\ .
\tag{4.24}
$$

Substituting the left-hand side of Equation (4.24) with the rotational matrix of the general serial link manipulator with spherical wrist presented in Equation (4.17) gives

$$
\begin{aligned}
R36(1,1) &= C4\ C5\ C6 + S4\ [-(C6\ C\alpha 4\ S5) - S6\ S\alpha 4] \\
R36(2,1) &= C5\ C6\ S4 + C4\ (C6\ C\alpha 4\ S5 + S6\ S\alpha 4) \\
R36(3,1) &= -(C\alpha 4\ S6) + C6\ S5\ S\alpha 4 \\
R36(1,2) &= -(C5\ C\alpha 4\ S4) - C4\ S5 \\
R36(2,2) &= C4\ C5\ C\alpha 4 - S4\ S5 \\
R36(3,2) &= C5\ S\alpha 4 \\
R36(1,3) &= C4\ C5\ S6 + S4\ [-(C\alpha 4\ S5\ S6) + C6\ S\alpha 4] \\
R36(2,3) &= C5\ S4\ S6 + C4\ (C\alpha 4\ S5\ S6 - C6\ S\alpha 4) \\
R36(3,3) &= C6\ C\alpha 4 + S5\ S6\ S\alpha 4 \quad .
\end{aligned}
\tag{4.25}
$$

For many manipulators, Equation (4.24) can be solved for angles q4, q5, and q6 by using exactly the Z-Y-Z Euler angles presented in section 4.1.2 of this work. A more complicated scheme has to be developed to solve for the first three joint variables by using the system of equations created with Equations (4.20) and (4.22). This system of equations is highly non linear, and for some cases, multiple solutions are found. For

several cases, the equations are not solvable, because of linear dependency. This dependency appears when the first three links of the manipulator are constrained to be in a plane.

The third joint variable, either q3 (for revolute) or d3 (for prismatic), is solved directly by the third (if a1 = 0) or second (if a1 ≠ 0) equation of the system by pattern matching. The rules of this pattern matching are shown in Equation (4.26).

```
a_. Cos[x_] + b_. Sin[x_] + d_. == c_  ->   x -> Atan2[a,-b] -
                                          Atan2[c-d,"+-"Sqrt[a^2+b^2-(c-d)^2]],
a_. Cos[x_] == b_                      ->   x -> Atan2["+-"Sqrt[1-(b/a)^2],b/a],
a_. Sin[x_] == b_                      ->   x -> Atan2[b/a,"+-"Sqrt[1-(b/a)^2]],
c_ == a_. Cos[x_] + b_. Sin[x_] + d_.  ->   x -> Atan2[a,-b] -
                                          Atan2[c-d,"+-"Sqrt[a^2+b^2-(c-d)^2]],
b_ == a_. Cos[x_]                      ->   x -> Atan2["+-"Sqrt[1-(b/a)^2],b/a],
b_ == a_. Sin[x_]                      ->   x -> Atan2[b/a,"+-"Sqrt[1-(b/a)^2]],
Atan2[0,x_]                            ->            0                          .
```

(4.26)

The other two equations of the system are then solved together by a similar procedure.

4.1.4 Jacobian Written with Respect to An Arbitrary Frame

The Jacobian of a robot manipulator specifies a mapping from velocities in joint space to velocities in Cartesian space. It also maps payload (external) forces to joint torques (see Asada and Slotine 1986, p. 81). End-point compliance analysis of manipulators also depends on the Jacobian of the manipulator. Furthermore, having the Jacobian of the manipulator in symbolic form and as reduced as possible will affect any control algorithm or research performed on a robot manipulator.

As shown in the examples for the Laboratory Telerobotic Manipulator (Dubey et al. 1988), the Center for Engineering Systems Advanced Research Manipulator (Dubey et al. 1989), and the Robotics Research Manipulators presented in Chapter 6, the Jacobian of the manipulator written with respect to the third frame is used to obtain an efficient algorithm for a 7-DOF redundant manipulator. Those are good examples of how the Jacobian can be significantly simplified and powerfully written with respect to a different coordinate frame rather than to the base or end effector.

Two different algorithms are used on SML to calculate the Jacobian of the manipulator. The first , discussed by Asada and Slotine (1986, p. 58), is used in SML to calculate the Jacobian written with respect to the base coordinate frame. In this algorithm, the effect of each joint on the movement of the end effector is taken into account. The differential movement of the end effector due to joint i produces the ith column of the manipulator Jacobian. The dimension of the Jacobian is six by n; the first three rows are associated with the linear velocity of the end effector, and the three last correspond to its angular velocity. Furthermore, the Jacobian can be partitioned so that

$$J = \begin{bmatrix} J_{L1} & \vdots & J_{L2} & \vdots & \cdots & \vdots & J_{Ln} \\ J_{A1} & \vdots & J_{A2} & \vdots & & \vdots & J_{An} \end{bmatrix}, \tag{4.27}$$

where J_{Li} and J_{Ai} are three-by-one column vectors of the Jacobian matrix associated with the linear and angular velocities, respectively, of the end effector. These vectors are calculated in SML as follows, depending on the type of joint.

1. For a prismatic joint,

$$\begin{bmatrix} J_{Li} \\ J_{Ai} \end{bmatrix} = \begin{bmatrix} b_{i-1} \\ 0 \end{bmatrix}, \tag{4.28}$$

where b_{i-1} is the unit vector pointing along the direction of the joint axes i, which is calculated in SML using the functions OperTransform and Table. Being the last one a built-in function of Mathematica (Wolfram 1988) to build up vectors, matrices, and tensors.

$$b_{i-1} = \text{Table[OperTransform[DHTable, 0, i] [[j, 3]] , \{j, 1, 3\}]} . \tag{4.29}$$

To obtain the ith column of the Jacobian, SML just needs to join the vector J_{Li} to the null vector $J_{Ai} = \{0, 0, 0\}$ by

$$\text{Join}[J_{Li}, \{0, 0, 0\}] . \tag{4.30}$$

2. For a revolute joint,

$$\begin{bmatrix} JLi \\ J_{Ai} \end{bmatrix} = \begin{bmatrix} b_{i-1} \times r_{i-1,e} \\ b_{i-1} \end{bmatrix}, \tag{4.31}$$

where $r_{i-1,e}$ is the position vector from the origin O_{i-1} of the ith coordinate frame to the end effector. This vector is calculated in SML by subtracting the ith frame position vector from the end effector one:

$$r_{i-1,e} = {}^0P_e - {}^0P_i = \text{Pos[DHTable, 0, MTerm] - Pos[DHTable, 0, i]} , \tag{4.32}$$

where MTerm is equal to the number of rows of the DHTable. All the b_{i-1} and $r_{i-1,e}$ vectors are calculated by using OperTransform; thus, all are completely trigonometrically reduced already. The cross product $b_{i-1} \times r_{i-1,e}$ is calculated and reduced trigonometrically by SML. Finally, J_{Li} and J_{Ai} are joined together to obtain the ith column of the Jacobian by

$$\text{Join[} J_{Li}, J_{Ai}] . \tag{4.33}$$

SML recognizes automatically whether a joint is revolute or prismatic by checking the first entry of the joint row on the D-H Table of the manipulator. If q_i has a given numerical value (i.e., 0, Pi/2, Pi/4,...) or any symbolic value (i.e., q, qv, x,...), then the joint is prismatic; the joint is revolute in the opposite case.

The Jacobian is constructed by joining the columns created by either Equation (4.28) or (4.31), depending on the type of joint. The Jacobian matrix obtained with this method is written with respect to the base coordinate frame.

The function JacobianP (JacobianC when using Craig's notation) calculates the Jacobian of the manipulator. It is called with just two arguments:

JacobianP[DHTable_, RefFrame_:0] , (4.34)

where

- DHTable is the name given to the D-H Table of the manipulator, which should be entered as shown in Equation (3.2).
- RefFrame is the coordinate frame with respect to which the Jacobian of the manipulator is required to be written. The default coordinate frame for RefFrame is the base frame.

Both JacobianP and JacobianC have an auxiliary function that transforms the Jacobian of the manipulator. Premultiplying it by the matrix of Equation (4.35), the Jacobian is changed from being written with respect to frame B to be written with respect to frame A.

$$ {}^{A}J = \left[\begin{array}{c|c} R_A^B & 0 \\ \hline 0 & R_A^B \end{array} \right] {}^{B}J \ . \tag{4.35} $$

The auxiliary function JacobTransform can be used independently by the user calling it with four arguments:

JacobTransform[DHTable_, NewFrame_, OldFrame_, OldJac_] , (4.36)

where

- DHTable is the name given to the D-H Table of the manipulator.
- NewFrame is the new frame (A in the case for Equation 4.35) with respect to which the Jacobian is desired to be written.
- OldFrame is the frame (B in the case for Equation 4.35) with respect to which the Jacobian matrix OldJac (${}^{B}J$ on Equation 4.35) is written.

Most often, the user will not even notice that JacobTransform is acting, because JacobianP calls it automatically when it is needed. The biggest obstacle in the use of JacobTransform and the algorithm of Equation (4.35) upon which it is based, is that expressions obtained can be rather complex. The complexity of trigonometric reductions over these expressions depends on the D-H Table of the manipulator. The Jacobian is of six-by-n dimension, means that 6×n elements are to be reduced for a general n-DOF manipulator. The more parameters different from zero on the D-H Table, the more time will be needed to reduce the model trigonometrically. Furthermore, trigonometric reductions may need a great deal of time to be accomplished when using the function JacobTransform. The principal reason because SML has two algorithms to calculate the Jacobian matrix of the manipulator.

A second algorithm based on the Newton-Euler formulation for static forces (see Craig 1986, p. 149; Asada and Slotine 1986, p. 73) is used in SML to calculate the Jacobian written with respect to the end-effector coordinate frame. To the knowledge of the author, this algorithm has never been used before to create the symbolic or numeric Jacobian matrix of a manipulator in any robotic modeling package. In the contrary, most researches use this algorithm to calculate payload effects over the manipulator joints when

the Jacobian is known. On the Newton-Euler algorithm external forces and torques are applied at the end effector, and their effects are studied along all the links of the manipulator. As shown in Figure 4.3, F_i and T_i are the necessary force and torque to keep the link in static equilibrium when it is under the effect of an external force F_e and torque T_e. Vectors F_i and T_i are obtained by simply using the Newton-Euler algorithm, that will be presented in more detail in section 4.2 in this chapter.

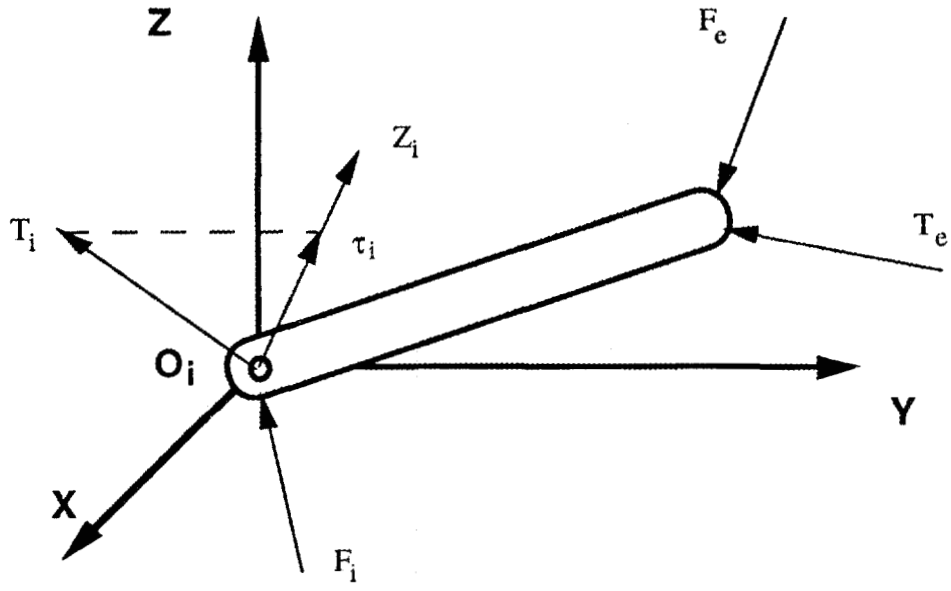


Figure 4.3. Forces and torques applied to a link of a manipulator and the projection onto the Z_i axes.

The obtained torque T_i is projected onto the rotational joint axes, or the force F_i is projected instead if the joint is prismatic. The projected force or torque τ_i is the one that the motor has to supply at the joint to keep the manipulator in static equilibrium. The force F_i and torque T_i will be applied later against the preceding link as if they were external so the procedure continues until the base of the manipulator is reached.

The model of the forces or torques τ_i at every joint of the manipulator is created symbolically in function of the external forces (torques) applied at the end effector. The external force vector $F_e = \{Fx, Fy, Fz\}$ and the external torque vector $T_e = \{Tx, Ty, Tz\}$ are joined together in a 6-dimensional vector $N = \{Fx, Fy, Fz, Tx, Ty, Tz\}$. Then, using the algorithm presented in Craig (1986, p. 152) based on the relation between the end-effector force N and joint torque (force),

$$T = {}^{E}J^{T} \; {}^{E}N \; , \tag{4.37}$$

the transpose of the Jacobian matrix can be found written with respect to the Eth coordinate frame. To find the Jacobian written with respect to the end-effector frame, SML first makes $N = \{1, 0, 0, 0, 0, 0\}$. The forces (torques) found at the joints constitute the first

column of the transpose of the manipulator Jacobian matrix, which is the first row of the Jacobian of the manipulator written with respect to the end-effector coordinate frame,

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_n \end{bmatrix} = \begin{bmatrix} J_{11} & J_{21} & J_{31} & J_{41} & J_{51} & J_{61} \\ J_{12} & J_{22} & J_{32} & J_{42} & J_{52} & J_{62} \\ \vdots & & & & & \\ J_{1n} & J_{2n} & J_{3n} & J_{4n} & J_{5n} & J_{6n} \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ F_z \\ N_x \\ N_y \\ N_z \end{bmatrix} . \tag{4.38}$$

The same procedure is used for N = {0, 1, 0, 0, 0, 0} to find the second row of the manipulator Jacobian, and so on until its six rows are obtained.

The vector of external forces N could be written with respect to any desired coordinate frame of the manipulator to obtain as a result the Jacobian of the manipulator written with respect to that coordinate frame. It was found in SML that it was more complicated to deal with the equations in that way than by using the auxiliary function JacobTransform.

The function JacobianP (or JacobianC) will check first as to whether, the base or the end-effector coordinate frame is closer to the one requested by the user (RefFrame). Then, it will calculate the Jacobian written with respect to the base or end effector, whichever is closer. Finally, it will transform the Jacobian to the required frame by using JacobTransfom.

Usually, the D-H Table of a manipulator has many parameters equal to zero at higher rows on the table because most manipulators have a spherical wrist or at least some of the last joint axes intercepted. This effect causes the rotational matrix of the homogeneous transformations between the last frames to be simpler and less trigonometrically complex than the ones between the initial coordinate frames. Furthermore, the matrix that defines the Jacobian transformation from one coordinate frame to another, shown in Equation (4.35), is usually more complex when transforming between the initial coordinate frames than between the last ones. Furthermore, if the goal is to obtain the Jacobian matrix with respect to a middlemost frame of the robot, it is preferable to derive it with respect to the end-effector coordinate frame and then transform it to a lower frame, rather than to obtain it with respect to the base frame and transform it to higher frames. This effect is taken into account by the function JacobianP, giving priority to calculate the Jacobian matrix written with respect to the end effector rather than to the base coordinate frame.

4.2 STATIC FORCES FUNCTIONS

As shown in Section 2.2.4, correction to the control of robot manipulators due to external static forces and gravitational effects is the most computationally-efficient technique that can be applied with actual microcomputers. Therefore, it is important to have the model of the reactions at any joint due to external forces and gravitational effect. To the knowledge of the author, SML is the only package capable of creating this model.

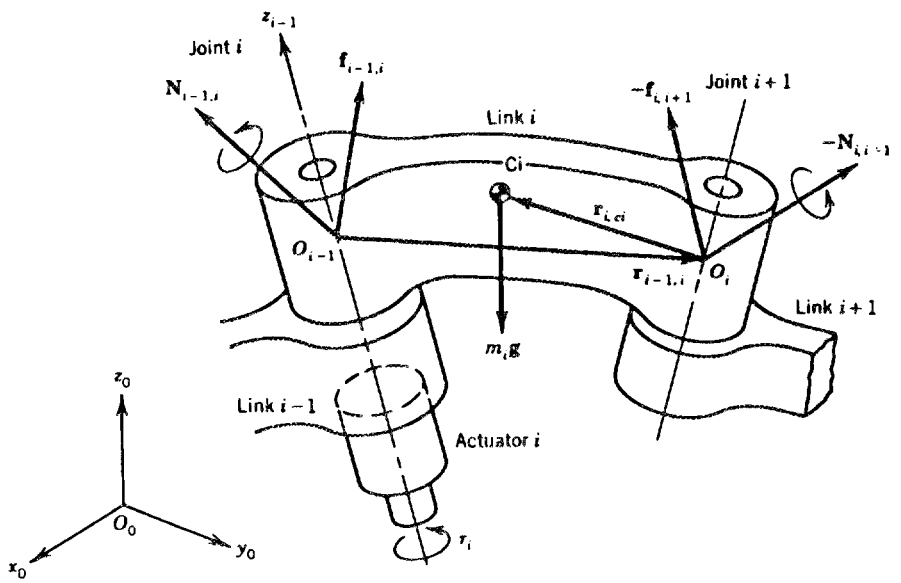
SML provides the user with the reaction forces due to external static and gravitational forces over each joint. That output presents: (1) the three force and the three torque components along the three Cartesian vectors that constitute each coordinate frame and (2) their reaction over the manipulator joints. The first output will allow the researcher to know in advance the internal forces produced inside the manipulator. These reactions

provoke deflection and torsion of the links of the manipulator and stress of its joints. Further, they can be used in the joint and link stress design. Advanced knowledge of the value and direction of maximum deflection and torsion on the manipulator, will allow links to be reduced in weight and size. In this way, not only the joints but also the links can be more accurately designed.

The correction of external forces and the gravitation effect on the manipulator are calculated directly from the Newton-Euler formulation. The forces and moments acting on link i are shown in Figure 4.4. The balance of linear forces and moments acting on the link about the center O_i are given by

$$\begin{aligned} &f_{i-1,i} - f_{i,i+1} + m_i\, g = 0, && i = 1, \ldots, n \\ &N_{i-1,i} - N_{i,i+1} - r_{i-1,i} \times f_{i-1,i} + r_{i,ci} \times m_i\, g = 0, && i = 1, \ldots, n\ . \end{aligned} \tag{4.39}$$

The function StaticForces of SML calculates the effect of external forces and gravitation over the links of the manipulator. Because the Newton-Euler iterative algorithm can be used for both external and gravitational effects, only one function is necessary for their calculation.



$f_{i-1,i}$	= force exerted by link i-1 acting upon link i
$N_{i-1,i}$	= moment appliedto link i by link i-1
m_i	= mass of link link i
g	= the 3×1 gravity aceleration vector
C_i	= centroid of link i
$r_{i-1,i}$	= 3×1 position vector from O_{i-1} to O_i
$r_{i,ci}$	= 3×1 position vector from O_i to C_i

Figure 4.4. Forces and torques acting on link i.

The function StaticForces is called with the following parameters:

```
StaticForces[DHTable_, MassTable_:Zero, VGravity_:{0, 0, -G},
        Fext_:{0, 0, 0, 0, 0, 0}, FrameFext_:MTerm,                    (4.40)
        FrameFextApplied_:MTerm, Force_:False, Torque_:False] ,
```

where

- DHTable is the name given to the D-H Table as in Equation (3.2).
- MassTable is a table composed of four parameters for each link: the first is the link mass, and the next three parameters define the location of the link center of mass with respect to the X-, Y-, and Z-axes of the coordinate frame attached to that link. This table was presented in Equation (3.4), and its default is a four-by-MTerm dimensional matrix composed by zeros.
- VGravity is a three-by-one vector that represents the direction of the gravity acceleration written with respect to the base coordinated frame. Its default is given by {0, 0, -G}, which gives the classical direction along the -Z-axes and the absolute value G in symbolic form.
- Fext is the external force (torque) applied to any coordinate frame (FrameFextApplied) defined by the DHTable. It can be written with respect to any frame (FrameFext). Its default value is {0, 0, 0, 0, 0, 0} being applied at and written with respect to the end-effector coordinate frame (MTerm).
- Force and Torque are options for the output of the function. They are set to False by default, giving only in the output the effect of VGravity and Fext on the joints. If Force and/or Torque is set to any different value (i.e., True), then all the internal forces and/or torques on the links will be included in the output.

On output, the function StaticForces types in text mode [see Equation (3.17)] the list of forces and/or torques requested by the user and an explanation of the used convention. StaticForces writes also in memory the following vectors:

$$\begin{aligned} Fs &= \text{force exerted by link i-1 acting upon link i} \\ Ns &= \text{moment applied to link i by link i-1} \\ M &= \text{force/moment applied to joint i} \quad . \end{aligned} \tag{4.41}$$

These vectors can be viewed or manipulated by the user after a call to the function StaticForces. Any time the function is called, these vectors are overwritten. Therefore, the vectors should be saved with a different name if the plan is to use them later.

4.2.1 Reaction of Joints to Any Force/Torque Vector Applied at Any Coordinate Frame Attached to the Manipulator

A robot manipulator can support forces or torques in different points. The most usual point of contact with the environment is the end effector, but the manipulator can apply a general force at any of its points. The manipulator may have a contact point on one link while supporting a payload with the end effector as, for example, a human arm does when writing on paper to improve its stiffness. Furthermore, it is important to be able to create a model of the effect of different forces at distinct points of the manipulator and written with respect to different coordinate frames of the manipulator. Any combination

can be achieved by the function StaticForces when using it properly. In addition, by the principle of superposition, the effect of different forces can be studied and added for the same manipulator to obtain a model.

To facilitate the use of this package, an auxiliary function based on the StaticForces function has been added. The function Forces is called with the following parameters:

Forces[DHTable_, Fext_:{0, 0, 0, 0, 0, 0}, FrameFext_:MTerm,

FrameFextApplied_:MTerm, Force_:False, Torque_:False] , (4.42)

where the parameters are the same as those for StaticForces in Equation (4.40).

FrameFext and FrameFextApplied should be frames represented by a row of the DHTable. FrameFext is by default the last frame represented by the DHTable. This means that the applied force Fext will be written with respect to the end-effector coordinate frame, but any one in the DHTable can be used. If the manipulator has a sensor attached to a coordinate frame that does not exactly correspond with any link frame, the user will add a row of constant parameters to the DHTable at that position. As an example, to find the effect on the joints of a general force $\{F_x, F_y, F_y, T_x, T_y, T_z\}$ applied at the fifth frame and written with respect to the third frame, the function Forces will be called as follows:

Forces[DHTable, $\{F_x, F_y, F_y, T_x, T_y, T_z\}$, 3, 5] . (4.43)

4.2.2 Necessary Gravitational Compensation at Each Joint

Gravitational effects on the manipulator can be calculated also with the function StaticForces, but an auxiliary function has been added to make SML user friendly. The function Gravitation is called with the following parameters:

Gravitation[DHTable_, MassTable_:Zero, VGravity_:{0, 0, -G},

Force_:False, Torque_:False] , (4.44)

where the parameters are the same as those for StaticForces in Equation (4.40).

4.3 TRIGONOMETRIC REDUCTIONS

An objective of SML is to create simple and understandable output expressions from standard input. To take full advantage of symbolic manipulation of equations, the input parameters can be numeric or symbolic, and so the output can be. Depending on the manipulator D-H Table or in a particular model, a different output structure may be preferred. The objective is to obtain output expressions that are easy to read and are computationally-efficient when implemented in a microprocessor.

An important reduction of the complexity of the equation on robotics comes from the trigonometric reductions. They play an important role in robotics modeling, but they have not been solved completely. This report presents an important study and solution for this problem. In SML, two methods to reduce trigonometric expressions are presented.

(1) A classical pattern matching, where expressions are compared and reduced according to the following patterns.

```
a_. Sin[x_] Cos[y_] + a_. Cos[x_] Sin[y_]  ->  a Sin[x + y]
a_. Sin[x_] Cos[y_] - a_. Cos[x_] Sin[y_]  ->  a Sin[x - y]
a_. Cos[x_] Cos[y_] - a_. Sin[x_] Sin[y_]  ->  a Cos[x + y]          (4.45)
a_. Cos[x_] Cos[y_] + a_. Sin[x_] Sin[y_]  ->  a Cos[x - y]
a_. (Cos[x_])^2     + a_. (Sin[x_])^2      ->  a .
```

This is one of the fastest and most efficient ways to diminish trigonometrically a very short expression. The pattern recognition algorithm is used to check all possible combinations inside the expression. However, if an expression is long, the number of combinations is so large that the reduction of an expression can take so much time that the outcome would be worthless or too expensive.

(2) An exponential reduction method, based on changing trigonometric expressions to their corresponding pseudo-exponential expressions, is given in Equations (4.46) through (4.48).

```
Tan[x_]  ->   Sin[x] / Cos[x]     ,
Cos[x_]  ->   Ex[x] +  Ex[-x]     ,          (4.46)
Sin[x_]  ->  -I Ex[x] + I Ex[-x]  .
```

Some especial properties are defined for this pseudo-exponential function to reduce trigonometrically the expression

```
Ex[x_] Ex[y_]  ->  Ex[x+y]/2          ,
Ex[x_]^n_      ->  Ex[n x]/(2^(n-1))  ,
Ex[0]          =       1/2            ,          (4.47)
  I            =      √-1             .
```

The final step is to transform the expression from the pseudo-exponential to the trigonometric functions by

```
Ex[x_]  ->  Cos[x]/2 + I Sin[x]/2 .          (4.48)
```

This method has proved to work well with long, complicated expressions that the classical method cannot deal with. Instead of checking for any possible combination that matches one of the patterns, this method transforms every sine, cosine, and tangent in its pseudo-exponential expression by using Equation (4.46). The operations defined by Equation (4.47) are faster than pattern matching for producing the desired trigonometric reduction, and they give expressions, based on experience, that are close to minimum time solution.

The classical pattern matching reduction method of Equation (4.45) works well when the expression is short and uncomplicated. It is performed in SML by the function RedTrig, that is called as follows with just one parameter.

RedTrig[expr_] (4.49)

The exponential reduction method is best when used for long and messy expressions, and it is called also with just one parameter.

RedTrigExp[expr_] (4.50)

In both trigonometric reduction functions, expr can be any kind of expression to be reduced. RedTrig and RedTrigExp can be applied to either a vector, a matrix, or any kind of expression.

4.4 MISCELLANEOUS FUNCTIONS

Some miscellaneous functions have been added to SML to make it user friendly or to be used by some of the principal functions of the package. Output form functions were discussed in detail in Section 3.2.2 and will be not presented again here. Some additional functions such CrossProd are added to SML. CrossProd gives the cross product of two vectors (V and U), which constitute the two arguments of the function:

CrossProd[V_, U_] . (4.51)

Another miscellaneous function is PosVector, which gives the position vector of any four-by-four homogenous transformation (Matrix):

PosVector[Matrix_] . (4.52)

ROut is another function which performs first a trigonometric reduction on the expression given by expr and then reduces its output with RedAngle:

ROut[expr_, var_:q , big_:0] , (4.53)

where var and big are the optional arguments presented for the function RedAngle on Equation (3.8). Two other functions are RCForm and RFForm, which reduce first the given expression and present the output in C or FORTRAN, respectively, compatible forms.

RCForm[expr_, var_:q , big_:0]

RFForm[expr_, var_:q , big_:0] . (4.54)

4.4.1 On-Line Help

SML can be used in both interactive and batch modes. When using an interactive mode, each function of the package can be called separately. Then, each output can be analyzed and used as input for the following calls to functions. Because SML presents so many options, it was necessary to include the interactive mode and an on-line help.

To facilitate the user's work, an on-line help based on Mathematica's own help was written. It allows the user to know, at any moment in a session, how to use and call any

subroutine or the actual numeric-symbolic value for a variable, a vector, or a matrix. To obtain information about a function, type "?FunctionName." As an example, to find how to use OperTransform type "?OperTransform" and SML will give as output:

OperTransform[DHTable, RefFrame, Frame] gives the 4x4 Homogeneous

Transformation operator that relates Frame and RefFrame. (4.55)

Enter the Denavit-Hartenberg Table in Paul's Notation.

If a list of the functions included in SML is desired, the user types "?SML" and a list will appear on the screen.

If the program is being used on a Macintosh or a Next machine, an extra help feature is allowed by Mathematica (Wolfram 1988). To obtain a template of one of the functions, the user types the name or a part of it, and highlights it (using the mouse or the keyboard). Then, look in the Action menu for the option Prepare Input to use the feature Make Template. If "Direct" is typed and highlighted, then using the feature Make Template gives the following output, which prepares the function and its parameters:

DirectKinEq[DHTable, EulerOrder, BaseFrame, LastFrame]. (4.56)

5. CONCLUSIONS AND RECOMMENDATIONS

The primary result of this study is the creation of an efficient symbolic modeling package, called SML, for the kinematic analysis and control of robot manipulators. Using the high-level symbolic computer language Mathematica (Wolfram 1988), SML is able to create symbolic manipulator models from minimal manipulator descriptions entered by the user.

In contrast with numerical methodologies, symbolic models can take full advantage of reductions because of particular geometric manipulator configurations. Furthermore, the obtained equations are computationally efficient, making the control that uses them to be real-time efficient.

5.1 CONCLUSIONS

In currently available computer-aided modeling of manipulators, only a few well-stipulated outputs can be obtained. Because robotics is a fast-growing field, more flexible modeling software is required. SML is the only program package for symbolic modeling of manipulators that can be used in an interactive mode and that has an on-line help. This means that the output from a function can be studied and used as input for another function. As an example, the Jacobian of a manipulator can be obtained with respect to different frames to determine which one has a simpler form for its use as an input for a control algorithm (Dubey, Euler, and Babcock 1988; Dubey et al. 1989).

Trigonometric reductions play an important role in robotic modeling because expressions can be greatly diminished in complexity. The functions of SML present their outputs completely trigonometrically reduced. To the knowledge of the author, the only other package for symbolic modeling of manipulators that presented trigonometric reductions was the one from Ho and Sriwattanathamma (1989). As reflected in their paper, the outputs of their program are not completely trigonometrically reduced, but SML gives the same outputs with reduced complexity.

Most of the symbolic modeling programs previously presented gave only some of the homogeneous transformations between some frames of the manipulator. In contrast, SML can give any transformation between any two frames of the robot. Currently, this is the only package that also computes the inverses of these transformations. This is possible because of the trigonometric reduction simplification subroutines. These inverses are useful for constructing the inverse kinematic models and sometimes for control algorithms.

Some of the earlier packages gave the homogeneous transformation between the hand and base frames, but only SML gives specifically the position vector and the orientation angles. These angles are presented in three different options for the user to choose: (1) ZYX Euler angles (2) ZYZ Euler angles and (3) XYZ angles rotated about fixed frames.

The Jacobian matrix is presented in SML written with respect to any frame. This is a big advantage in comparison with other packages, and it is of great use in force and other control algorithms, as shown in the example application appendixes for the FARS manipulator, the Center for Engineering Systems Advanced Research manipulator (CESARm), and the Laboratory Telerobotic Manipulator (LTM).

SML is the only package known to be capable of creating the statics forces model in general form. This model gives the six components of the force (torque) reaction vectors acting at the joints when an external force (torque) is applied at an arbitrary coordinate frame of the manipulator and taking also into account the gravity effect. The example

applications for FARS and for the LTM in appendixes C-1 and C-3 show the gravitational effect model. Using this model, gravitational compensation can be added to the control algorithm improving its accuracy and the behavior of the robotic system.

An on-line help and an easy-to-use and easy-to-understand output is presented by SML. Several output forms are given as a choice for the user, such as those that are compatible with the FORTRAN and C language programs.

Some example applications of SML are presented in Appendix C of this report. Two 7-DOF robots are studied to obtain their forward and inverse kinematics: CESARm, and LTM. Finally, a full kinematic and static study in symbolic form is presented for FARS manipulator. A design optimization for some lengths and angle constraints of the FARS manipulator is performed using the symbolic models obtained from SML.

5.2 RECOMMENDATIONS

One of the fundamental objectives in developing SML was to create an open and interactive package. The package was created such that the user can call any of the functions to create new ones. This means that future research is continuously open. In fact, some new functions currently being implemented in SML by the author are not presented in this report.

The extension of the package to dynamic symbolic robot modeling is obvious, and a function is already working that calculates the diagonal terms of the inertial matrix of a serial manipulator. Symbolic dynamic models have the advantage over numeric models in that no numerical error is introduced. In particular, when higher modes of flexible manipulators models are studied, very ill conditioned matrices are found. Most advanced and sophisticated control algorithms are required to be as close as possible to the exact model. This demands the inclusion of higher modes and a greater numerical error. If we are able to describe the transfer function in symbolic form (Lee 1990), then no numerical error is included, thus improving the behavior of the control algorithm.

Another recommended field of expansion for SML is graphic simulation. The kinematic model can be created with SML and then used to plot the configuration for specific joint values or the work space for specific joint constraints. A good example of plotting the work space as a three-dimensional solid model is presented on the example application for the FARS manipulator. The functions that created the work space were written to be used only with the FARS manipulator, but more sophisticated functions could be created for more general cases.

An immediately achievable important task for future investigations is the creation of a minimization function to reduce the expressions of the models to the minimum amount of computational time. It should take into account all the equations of the model and, penalizing with weights the different algebraic functions, collect common terms to try to minimize the time necessary to compute the model.

6. REFERENCES

Aldon, M. J., and A. Liegeois 1984. "Computational Aspects in Robot Dynamic Modelling," pp. 3-14 in Advanced Software in Robotics, Elsevier Science Publishers, North Holland.

Asada, H., and J.-J. E. Slotine 1986. Robot Analysis and Control, John Wiley and Sons, New York.

Borland 1986. Turbo Prolog: Owner's Handbook, Borland International, Inc., Scotts Valley, Calif.

Cesareo, G., F. Nicolo, and S. Nicosia 1984. "DYMIR: A Code for Generating Dynamical Models of Robots," pp. 115-20 in Proceedings of the First International IEEE Conference on Robotics, ed. R. P. Paul, Atlanta, March 13-15.

Cheng, P.-Y., C.-I Weng, and C.-K. Chen 1988. "Symbolic Derivation of Dynamic Equations of Motion for Robot Manipulators Using Piogram Symbolic Method," IEEE J. Robotics Autom. 4 (6), 599-609.

Craig, J. J., 1986. Introduction to Robotics, Mechanics and Control, Addison-Wesley Publishing Company, Reading, Mass.

Critchlow, J. Arthur 1985. Introduction to Robotics, Macmillan Publishing Company, New York.

Denavit, J., and R. S. Hartenberg 1955. "A Kinematic Notation for Lower-Pair Mechanism Based on Matrices," ASME J. Appl. Mech. 22, 215-21 (June).

Dubey, R. V., J. A. Euler, and S. M. Babcock 1988. "An Efficient Gradient Projection Scheme For A Seven Degree-Of-Freedom Redundant Robot With A Spherical Wrist," pp. 28-36 in Proceedings of the 1988 International IEEE Conference on Robotics and Automatization, ed. IEEE Computer Society Press, Philadelphia, Pennsylvania, April 24-29.

Dubey, R. V., J. A. Euler, R. B. Magness, S. M. Babcock , and J. N. Herndon 1989. "Robotic Control Of The Seven Degree-Of-Freedom Redundant NASA Laboratory Telerobotic Manipulator," NASA Conference on Space Telerobotics, Pasadena, California, January 31- February 2.

Goldenberg, A. A., B. Benhabib, and R. G. Fenton 1985. "A Complete Generalized Solution to the Inverse Kinematics of Robots," IEEE J. Robotics Automa., RA-1 (1), 14-20 (March).

Gupta, S., 1987. "The Symbolic Polynomial Technique for Modeling and Real Time Control of Robot Arms," Master's thesis, University of Virginia, Charlottesville.

Herrera-Bendezu, L. G., 1985. "SAST: A Symbolic Robot Arm Tool," Master's thesis, Department of Electrical Engineering, University of Pittsburgh.

Herrera-Bendezu, L. G., E. Mu, and J. T. Cain 1988. "Symbolic Computation of Robot Manipulator Kinematics," pp. 993-98 in Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 40, Philadelphia, April 24-29, Computer Society Press, Washington, D.C., IEEE J. Robotics Autom.

Ho, C. Y., and J. Sriwattanathamma 1989. "Symbolically Automated Direct Kinematic Equations Solver for Robotics Manipulators," Robotica 7, 243-54 (October).

Kircanski, M., et al. 1988. "A New Program Package for the Generation of Efficient Manipulator Kinematic and Dynamic Equations in Symbolic Form," Robotica 6, 311-18 (October).

Kircanski, M., and M. Vukobratovic 1985. "Computer-Aided Generation of Manipulator Kinematic Models in Symbolic Form," pp. 1043-49 in Proceedings of the 15th International Symposium on Industrial Robots, Vol. 2, Tokyo, September 11-13.

Kress, R.L., H. M. Costello, R. L. Glassell, and S. March-Leuba 1991. "Future Armor Rearm Control System Design," Accepted for publication in Robotics and Autonomous Systems, Special Issue on "Robotics and Intelligent Systems at Oak Ridge National Laboratory," Vol. 10, No. 1-2.

Lee, Jeh Won, 1990. "Dynamic Analisys and Control of Lightweight Manipulators with Flexible Parallel Link Mechanisms," Ph.D. thesis, Georgia Institute of Technology, Georgia.

Malm, J. F., 1984. "Symbolic Matrix Multiplication with LISP Programming," pp. 20/1-20/19 in Robots 8 Conference Proceedings, Vol 2, Detroit, June 4-7, Robotics International of SME, Dearborn, Mich.

March-Leuba, S., 1991. "Symbolic Manipulator Laboratory: An Efficient Symbolic Modeling Package for the Kinematic Analysis and Control of Robot Manipulators," Master's thesis, The University of Tennessee, Knoxville.

Matsuoka, K., and S. J. Citron 1985. "Symbolic Processing and Dynamic Simulation of Equations of Motion for Flexible Manipulators," Proceedings of the ISA/85 International Conference and Exhibit, Oct. 21-24 , Philadelphia, pp. 1601-10 in Advances in Instrumentation, Research Triangle Park, N.C.

Mu, E., 1987. "INKAS: An Inverse Kinematics Arm Solver," Master's thesis, Department of Electrical Engineering, University of Pittsburgh.

Murray, J. J., and C. P. Neuman 1984a. "ARM: An Algebraic Robot Dynamic Modeling Program," pp. 103-14 in Proceedings of the First International IEEE Conference on Robotics, ed. R. P. Paul, Atlanta, March 13-15.

Murray, J. J., and C. P. Neuman 1984b. Symbolic Linearization of the Newton-Euler Dynamic Robot Model, Technical Report, Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh.

Neuman, C. P., and J. J. Murray 1984. "Linearization and Sensitivity Functions of Dynamical Robot Models," IEEE Trans. Syst. Man Cibern., SMC-14 (6), 805-18.

Neuman, C. P., and J. J. Murray 1985. "Computational Robot Dynamics: Foundations and Applications," J. Robotics Syst. 2 (4), 425-52.

Neuman, C. P., and J. J. Murray 1987. "Symbolically Efficient Formulations for Computational Robot Dynamics," J. Robotics Syst. 4 (6), 743-69.

Paul, R. P., 1981. Robot Manipulators: Mathematics, Programming, and Control, MIT Press, Cambridge, Mass.

Pieper, D., 1968. "The Kinematics of Manipulators Under Computer Control," Ph.D. thesis, Stanford University, Stanford, Calif.

Piogram, R. L., 1964. "Symbolic Representation of Coordinate Transformation," IEEE Trans. Aerosp. Navig. Electron., ANE-11, 128-34 (June).

Piogram, R. L., 1966. "Euler Transformations," IEEE Trans. Autom. Control, AC-11, 707-15 (October).

Rieseler, H., and F. M. Wahl 1990. "Fast Symbolic Computation of the Inverse Kinematics of Robots," pp. 462-67 in Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 1, Philadelphia, April 24-29, Computer Society Press, Washington, D.C., IEEE J. Robotics Autom., (January).

Symbolics 1985. VAX UNIX Macsyma Reference Manual, New York.

Townsend, M. A., and S. Gupta 1989. "Automated Modeling and Rapid Solution of Robot Dynamics Using the Symbolic Polynomial Technique," Trans. ASME J. Mech. Trans. Autom. Des. 111, 537-544 (December).

Tzes, A. P., S. Yurkovich, and F. D. Langer 1988. "A Symbolic Manipulation Package for Modeling of Rigid or Flexible Manipulators," pp. 1526-31 in Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 3, Philadelphia, April 24-29, Computer Society Press, Washington, D.C., IEEE J. Robotics Autom.

Vecchio, L., et al. 1980. "Automatic Generation of Dynamical Models of Manipulators: 5th International Conference on Industrial Robot Technology, March 5-7, 1980, Milan, Italy," pp. 293-99 in Proceedings of the 10th International Symposium on Industrial Robots.

Vukobratovic, M., and N. Kircanski 1984. "A Method for Computer-Aided Construction of Analytical Models of Robotic Manipulators," pp. 519-28 in Proceedings of the First International IEEE Conference on Robotics, ed. R. P. Paul, Atlanta, March 13-15.

Vukobratovic, M., and N. Kircanski 1985. "Real Time Dynamics of Manipulation Robots," Series: Scientific Fundamentals of Robotics 4, Springer-Verlag, Berlin.

Vukobratovic, M., and M. Kircanski 1986. "Kinematics and Trajectory Synthesis of Manipulation Robots," Communications and Control Engineering Series, Springer-Verlag, Berlin.

Wolfram, S., 1988. MATHEMATICA: A System for Doing Mathematics by Computer, Addison-Wesley Publishing Company, Scotts Valley, Calif.

Zewari, S. W., and J. M. Zuguel 1986. "Prolog Implementation in Robot Kinematics," pp. 133-36 in Computers in Engineering.

APPENDIX A

USER MANUAL

APPENDIX A
USER MANUAL

To load any of the packages of SML, they have to be placed or copied first on the folder (directory) "Robotics," which is to be created by the user inside the folder (directory) "Packages" of Mathematica. The next step is to load Mathematica on the computer and then to load SML on Mathematica, typing any of the following:

```
Needs["Robotics`SML-P`"] ,
Needs["Robotics`SML-C`"] ,                                    (A-1)
Needs["Robotics`RedTrig`"] .
```

The first package (SML-P.m) allows the user to use any of the functions described in this report in Paul's notation (1981), and the second one (SML-C.m) does exactly the same but in Craig's notation (1986). Trigonometric reductions are already included in SML-P.m and SML-C.m; but with the third package (RedTrig.m), only the trigonometric reductions and output form functions of SML are loaded.

A-1. INPUT TABLES

The D-H Table in Paul's notation (1981) is entered in SML as an n-by-4 matrix, where n is the number of coordinate frames associated to links of the manipulator:

$$\begin{aligned} \text{RobotDHTable} = \{ & \{q_1,\ \alpha_1,\ a_1,\ d_1\}, \\ & \{q_2,\ \alpha_2,\ a_2,\ d_2\}, \\ & \quad\vdots \\ & \{q_n,\ \alpha_n,\ a_n,\ d_n\}\}, \end{aligned} \tag{A-2}$$

where:

$$q_i = \text{angle from } X_{i-1} \text{ to } X_i, \text{ about } Z_{i-1} ,$$

$$\alpha_i = \text{angle from } Z_{i-1} \text{ to } Z_i, \text{ about } X_i ,$$

$$a_i = \text{length from } Z_{i-1} \text{ to } Z_i, \text{ along } X_i ,$$

$$d_i = \text{length from } X_{i-1} \text{ to } X_i, \text{ along } Z_{i-1} .$$

The mass table is composed of four parameters for each link: the first one is the link mass, and the next three parameters define the location of the link center of mass with respect to the X-, Y-, and Z-axes of the coordinate frame attached to that link.

$$\begin{aligned} \text{RobotMassTable:=} \{ & \{m_1, mx_1, my_1, mz_1\}, \\ & \{m_2, mx_2, my_2, mz_2\}, \\ & \quad\vdots \\ & \{m_n, mx_n, my_n, mz_n\}\}. \end{aligned} \tag{A-3}$$

A-2. OUTPUT FORMS

The function RedAngle reduces the form of trigonometrical functions to the classical nomenclature. RedAngle has three arguments, the second and third ones being optional:

RedAngle[expr_,var_:q, big_:0] , (A-4)

where

- expr is the expression to be reduced, which includes any of the trigonometric functions of the form of Equation (3.6). Expr can be a vector, a matrix, a polynomial, or any kind of expression.
- var defines the angles inside the trigonometric functions used in SML. The default for var is q because most robotic applications publications use it, but any name can be specified by the user as long as it is the same for all the angles inside the expression.
- big is an option to enable RedAngle to deal with subindexes for the angles larger than nine. Big is zero by default, giving any value different from zero enables larger subindexes.

ListOutput prints the expression given as its first argument, regardless of its dimensions, in an easy-to-read and compatible output form. ListOutput has four parameters, the last three being optional:

ListOutput[expr_, name_:"List", form_:Text, var_:q, big_:0] , (A-5)

where

- expr is the expression to be printed in the specified form. Expr can be either a vector, a matrix, or an expression.
- name is a string that gives the name that will be used for the listing. The default for name is "List."
- form is the desired form in which the output will be printed. Its default is Text, but the following options are available:
  1. form = Text gives expression in Text form. This is probably the easier to read, but it is not good enough to copy and paste to another program such as in a word processor, when powers or divisions are present.
  2. form = C gives the expression in C form.
  3. form = RC prints the expression in C language, reducing the form of sines, cosines, and tangents by using RedAngle.
  4. form = F prints the expression in FORTRAN form.
  5. form = RF gives the expression in FORTRAN, reducing the form of sines, cosines, and tangents.
- var is the same argument as in RedAngle. This argument needs to be specified only when asking for Reduced FORTRAN (RF), Reduced C (RC), or Text forms and a different name that q_i has been used for the angles in the expression.
- big is the same option as in RedAngle.

A-3. FUNCTION TOOLS

OperTransform gives the homogeneous transformation that relates any two coordinate frames of the manipulator.

OperTransform[DHTable_, RefFrame_, Frame_] , (A-6)

where

- DHTable is the name given to the D-H Table of the manipulator, that should be entered as shown in Equation (A-2); and
- the output of OperTransform is the homogeneous transformation that relates Frame to RefFrame written with respect to RefFrame: $A^{Frame}_{RefFrame}$.

Two functions are directly associated to OperTransform: (1) Rot, that gives the rotation matrix, and (2) Pos, that presents the position vector that relate any two coordinate frames of the manipulator.

Rot[DHTable_, RefFrame_, Frame_]
Pos[DHTable_, RefFrame_, Frame_] . (A-7)

The function DirectKinEq calculates automatically the position and three different types of orientation angles. DirectKinEq is called with four arguments, the last three of which are optional:

DirectKinEq[DHTable_, BaseFrame_:0, LastFrame_:n, EulerOrder_:ZYX] , (A-8)

where

- DHTable is the name given to the D-H Table of the manipulator.
- BaseFrame is by default the 0 coordinate frame, but a different one can be specified by the user.
- LastFrame is by default the last coordinate frame of the DHTable.
- EulerOrder is the order of the Euler angles. Two orders can be used: (1) ZYX is the default, where the function gives the ZYX Euler angles or the XYZ angles about fixed axes (both solutions are the same) and (2) ZYZ to obtain the ZYZ Euler angles.

InverseKin calculates automatically, when it exists, the inverse kinematics of a 6-DOF manipulator of which the last three rotational joint axes intersect. InverseKin is called with just one argument:

InverseKin[DHTable_] , (A-9)

where

- DHTable is the name given to the D-H Table of the manipulator.

The function JacobianP (JacobianC when using Craig's notation) calculates the Jacobian of the manipulator. It is called with just two arguments:

JacobianP[DHTable_, RefFrame_:0] , (A-10)

where

- DHTable is the name given to the D-H Table of the manipulator.
- RefFrame is the coordinate frame with respect to which the Jacobian of the manipulator is required to be written. The default coordinate frame for RefFrame is the base frame.

The auxiliary function JacobTransform transforms the Jacobian of the manipulator from being written with respect to one coordinate frame to another. It is called with four arguments:

JacobTransform[DHTable_, NewFrame_, OldFrame_, OldJac_] , (A-11)

where

- DHTable is the name given to the D-H Table of the manipulator.
- NewFrame is the new frame with respect to which the Jacobian is desired to be written.
- OldFrame is the frame with respect to which the Jacobian matrix (OldJac) is written.

The function StaticForces calculates the effect of external forces and gravitation over the links of the manipulator. It is called with the following parameters:

StaticForces[DHTable_, MassTable_:Zero, VGravity_:{0, 0, -G},
Fext_:{0, 0, 0, 0, 0, 0}, FrameFext_:MTerm, (A-12)
FrameFextApplied_:MTerm, Force_:False, Torque_:False] ,

where

- DHTable is the name given to the D-H Table as in Equation (A-2).
- MassTable should be entered as in Equation (A-3), and its default is a four-by-MTerm dimensional matrix composed by zeros.
- VGravity is a three-by-one vector that represents the direction of the gravity acceleration written with respect to the base coordinated frame. Its default is given by {0, 0, -G}, which gives the classical direction along the -Z-axes and the absolute value G in symbolic form.
- Fext is the external force (torque) applied to any coordinate frame (FrameFextApplied) defined by the DHTable. It can be written with respect to any frame (FrameFext). Its default value is {0, 0, 0, 0, 0, 0} being applied at the end-effector coordinate frame (MTerm) and written with respect to the same frame.
- Force and Torque are options for the output of the function. They are set to False by default, giving only on output the effect of VGravity and Fext over the joints. If Force and/or Torque is set to any different value (i.e., True), then all the internal forces and/or torques on the links will be included on the output.

Two auxiliary functions facilitate the use of StaticForces: (1) Forces, that calculates the effect of external forces, and (2) Gravitation, that calculates the effect of gravitation, over the links of the manipulator.

```
Forces[DHTable_, Fext_:{0, 0, 0, 0, 0, 0}, FrameFext_:MTerm,
        FrameFextApplied_:MTerm, Force_:False, Torque_:False] ,
                                                                (A-13)
Gravitation[DHTable_, MassTable_:Zero, VGravity_:{0, 0, -G},
                                Force_:False, Torque_:False] .
```

A-4. TRIGONOMETRIC, MISCELLANEOUS, AND HELP FUNCTIONS

Trigonometric reductions on an expression (expr) are obtained with the functions RedTrig and RedTrigExp. Use the first one for very simple expressions, and the second function for more complicated and messy ones. Both are called with just one argument, which can be any kind of expression (ie., vector, matrix, or list).

```
RedTrig[ expr_],
                                                                (A-14)
RedTrigExp[ expr_] .
```

CrossProd gives the cross product of two vectors (V and U), which constitute the two arguments of the function:

```
CrossProd[V_, U_] .                                             (A-15)
```

PosVector, which gives the position vector of any four-by-four homogeneous transformation (Matrix):

```
PosVector[Matrix_] .                                            (A-16)
```

ROut is another function that performs first a trigonometric reduction on the expression given by expr and then reduces its output with RedAngle:

```
ROut[expr_, var_:q, big_:0] ,                                   (A-17)
```

where var and big are the optional arguments presented for the function RedAngle. Two other functions are RCForm and RFForm, which reduce first the given expression and present the output in C or FORTRAN, respectively, compatible forms.

```
RCForm[expr_, var_:q , big_:0]
                                                                (A-18)
RFForm[expr_, var_:q , big_:0] .
```

An on-line help has been included in SML. To obtain information about a function, type "?FunctionName." If a list of the functions included in SML is desired, the user types "?SML" and a list will appear on the screen.

APPENDIX B

HELP CODE LISTING

APPENDIX B-1

SML-P.m, SYMBOLIC MANIPULATOR LABORATORY

By Santiago March-Leuba, November 1991

>> This file contains routines for robot manipulator modeling. Several functions are included to calculate the kinematic and static models of a manipulator. In addition, some trigonometric reductions and output form functions developed for use in robotics are incorporated. This file is called SML-P, in that it performs models based on Paul's notation. To use Craig's notation, look for the file called SML-C. A list of the functions in SML is presented below with help on how to use them. <<

```
BeginPackage["Robotics`SML-P`"];
```

GENERAL HELP

```
SML::usage = "Symbolic Manipulator Laboratory (SML)     \n
was writen by Santiago March-Leuba at the Oak Ridge      \n
National Laboratory to be used in symbolic modeling of   \n
robot manipulators. Functions ending in C are to be used \n
with Craig's notation; otherwhise use Paul's notation.   \n
List of Functions on SML:   \n
1.- Trigonometric Reductions and Output Forms \n
      RedTrig, RedTrigExp, RedAngle,        \n
      ListOutput, ROut, RFForm, RCForm.     \n
2.- Kinematics Functions:                   \n
      OperTransform,  Rot,  Pos,            \n
      DirectKinEq,  InverseKin,  JacobianP, \n
      OperTransformC, RotC, PosC,           \n
      DirectKinEqC, InverseKinC, JacobianC. \n
3.- Static Forces Functions:                \n
      StaticForces,  Forces,  Gravitation,  \n
      StaticForcesC, ForcesC, GravitationC. \n
4.- Miscellaneous Functions:          \n
      CrossProd, PosVector.           \n
Input Tables: \n
1.- Denavit-Hartenberg table (DHTable).     \n
2.- Mass parameters table (MassTable).";
```

```
DHTable::usage = "DHTable is the input used in kinematic \n
and static forces functions.  It is entered on the form:    \n
a)when using Paul's notation: \n
  DHTable = {{q1, alf1, a1, d1}, \n
             {q2, alf2, a2, d2}, \n
                    . . .\n
             {qn, alfn, an, dn}}, \n
  where: qi   = angle from Xi-1 to Xi, about Zi-1, \n
         alfi = angle from Zi-1 to Zi, about Xi,   \n
         ai   = length from Zi-1 to Zi, along Xi,  \n
         di   = length from Xi-1 to Xi, about Zi-1.\n \n
a)when using Craig's notation: \n
  DHTable = {{q1,  alf0,  a0,   d1}, \n
             {q2,  alf1,  a1,   d2}, \n
                    . . .\n
             {qn, alfn-1, an-1, dn}}, \n
  where: qi   = angle from Xi-1 to Xi, about Zi,   \n
         alfi = angle from Zi-1 to Zi, about Xi-1, \n
         ai   = length from Zi-1 to Zi, along Xi-1,\n
         di   = length from Xi-1 to Xi, about Zi.";

MassTable::usage = "MassTable is the input necessary for \n
the functions Gravitation and StaticForces.  It is entered  \n
on the form:    \n
  MassTable = {{m1, mx1, my1, my1}, \n
               {m2, mx2, mz2, mz2}, \n
                      . . .\n
               {mn, mxn, myn, mzn}}, \n
 where: (1) mi is the mass of link i; and (2) mxi, myi, and\n
 mzi are the locations of the centroid of link i along the \n
 X-, Y-, and Z-axes of the coordinate frame attached to   \n
 that link.";
```

TRIGONOMETRIC REDUCTIONS

```
RedTrig::usage = "RedTrig[expr]
gives expr Trigonometricaly reduced using    \n
classical pattern matching.";

RedTrigExp::usage = "RedTrigExp[expr]
gives expr Trigonometricaly reduced using  \n
pseudo-exponential functions. After using RedTrigExp, any     \n
of the four following functions can help to obtain a          \n
simpler output:                                               \n
  ToMin[expr], ToMinC[expr], ToPaper[expr], ToMinCS[expr].    \n
They have been listed in order of time consumption and        \n
sophistication. If the expression to deal with is long        \n
and complicated, it is to the user advantage to use one after \n
the other, checking at any step to determine whether the      \n
output is good enough.";
```

OUTPUT FORMS

```
RedAngle::usage = "RedAngle[expr, var, big]
gives the expression (expr),  \n
regardless of its dimensions, reducing the form of tangents, \n
sines, and cosines: Tan[q1] -> T1, Sin[q2+q3] -> S23,        \n
Cos[q1-q4] -> C1M4. By default, the angles are defined by qi;\n
to use a different one, specify it on var;                   \n
          example: using ti makes var=t.                     \n
When big > 0, subindexes bigger than 9 are allowed.";

ListOutput::usage = "ListOutput[List, name, form, var, big]
prints the given List,  \n
regardless of its dimensions, as a multidimensional vector \n
with its subindexes. The printed name of the list is       \n
given by name, ''List'' being its default.                 \n
Optional parameter form:                                   \n
  form = Text(by default), gives the list on Text Form.    \n
  form = C, gives the list on C Form.                      \n
  form = RC, gives the list on C Form, reducing the form of \n
    tangents, sines, and cosines: Tan[q1] -> T1,           \n
    Sin[q2+q3] -> S23,  Cos[q1-q4] -> C1M4. By default,    \n
    the angles are defined by qi; to use a     \n
    different one, specify it on var;                    \n
              example: using ti make var=t.              \n
  form = F, gives the list on FORTRAN Form.              \n
  form = RF, gives the list on FORTRAN Form reducing the form \n
    of sines and cosines. \n
When big > 0, subindexes bigger than 9 are allowed.";
```

FUNCTION -- Auxiliar --

```
CrossProd::usage = "CrossProd[V_,U_]
gives the cross product of the two vectors V and U.";

PosVector::usage = "PosVector[Matrix_]
gives the posititon vector of the \n
4 x 4 homogeneous transformation Matrix.";
```

■ FUNCTION -- " OPERATORS " -- (Paul's Notation) --

```
OperTransform::usage = "OperTransform[DHTable, RefFrame, Frame]
gives the 4 x 4 homogeneous \n
transformation operator that relates Frame and RefFrame. \n
Enter the Denavit-Hartenberg table in Paul's notation." ;
```

■ FUNCTION -- " ROTATIONAL MATRIX " --

```
Rot::usage = " Rot[DHTable, RefFrame, Frame]
gives the rotational matrix that \n
relates Frame and RefFrame. Enter the Denavit-Hartenberg table\n
in Pauls's notation." ;
```

■ FUNCTION -- " POSITION VECTOR " --

```
Pos::usage = " Pos[DHTable, RefFrame, Frame]
gives the position vector that \n
relates Frame and RefFrame. Enter the Denavit-Hartenberg \n
table in Paul's notation." ;
```

■ FUNCTION -- " PAUL'S TO CRAIG'S NOTATION " --

```
PaulToCraig::usage = " PaulToCraig[PDHTable]
transforms the Denavit-Hartenberg table \n
from Paul's To Craig's notation."
```

■ FUNCTION -- " KINEMATIC EQUATIONS " --

```
DirectKinEq::usage = "\
DirectKinEq[DHTable, EulerOrder, BaseFrame, LastFrame]
gives \n
the kinematic equations :Px, Py, Pz, Roll, Pitch, Yaw. Enter  \n
the D-H Table in Paul's notation.                             \n
  The default for BaseFrame is '0' and for LastFrame is the    \n
number of rows of the given D-H Table'.                        \n
EulerOrder is the order of the Euler angles. Its default is    \n
ZYX where the function gives the ZYX Euler angles, or the XYZ  \n
angles about fixed axes (both solutions are the same).         \n
  Use EulerOrder = ZYZ to get the ZYZ Euler angles.            \n
  On output, the angles are Roll:with respect to X; Pitch:with \n
respect to Y; and Yaw:with respect to Z.                       \n
To use the defaults, call the function with only the first     \n
parameter: DirectKinEq[DHTable]." ;
```

■ FUNCTION -- " INVERSE KINEMATICS " --

```
InverseKin::usage = "InverseKin[DHTable]
gives the inverse kinematics solution, when \n
it exist, of a 6-DOF manipulator with the last three rotational \n
axes intersected. Enter the D-H Table in Paul's notation." ;
```

■ FUNCTION -- " JACOBIAN - PAUL'S " --

```
JacobianP::usage = " JacobianP[DHTable, RefFrame]
gives the Jacobian of the robot \n
written with respect to any specified reference frame:          \n
    {Cartesian Coordinates speeds} = Jacobian . {Joint speeds},\n
where:{Cart. Coord. speeds} = {dx/dt, dy/dt, dz/dt, wx, wy, wz}\n
written with respect to RefFrame. The default for RefFrame is  \n
the baseframe.  Enter the DHTable in Paul's notation.";

JacobTransform::usage = "\
 JacobTransform[DHTable,newFrame,oldFrame,oldJac]
transforms \n
the Jacobian of the robot (oldJac), written with respect to any\n
specified reference frame (oldFrame), to a different frame     \n
(newFrame). Enter the DHTable of the robot in Paul's notation.";
```

■ FUNCTION -- " STATIC AND GRAVITATIONAL FORCES " --

```
StaticForces::usage = "StaticForces[DHTable, MassTable, \
Vgravity, Fext, FrameFext, FrameFextApplied, Force, Torque]
gives the force(torque) that\n
is necessary to apply at each joint to keep the robot in static\n
conditions when under the effect of an external force Fext\n
and/or Gravity.\n
1)MassTable is composed of four parameters for each link. The\n
  first one defines its mass and the next three define\n
  the location of the center of mass.\n
2)Vgravity: 3 x 1 vector that represents the direction and value\n
  of the gravity acceleration. By default, Vgravity={0,0,-G}. \n
3)Fext: 6 x 1 vector that represents the external force(torque)\n
  {Fx, Fy, Fz, Tx, Ty, Tz} along the three axes applied to \n
  any coordinate frame (FrameFextApplied) and written with respect`
  to any frame (FrameFext). By default, Fext = {0,0,0,0,0,0},\n
  FrameFextApplied = FrameFext = LastFrame of the DHTable.\n
4)Force and Torque are by default false. Setting them to true\n
  includes in the output all the internal reactions on the links.";

Forces::usage = "Forces[DHTable, Fext, \
FrameFext, FrameFextApplied, Force, Torque]\n
gives the force(torque) that is necesary to apply at each     \n
joint to keep the robot, given by DHTable, in static          \n
conditions when under the effect of an external force Fext.   \n
1)Fext: 6 x 1 vector that represents the external force(torque) \n
  {Fx, Fy, Fz, Tx, Ty, Tz} along the three axes.  Applied to  \n
  any coordinate frame (FrameFextApplied) and written with respect
  to any frame (FrameFext). By default, Fext = {0,0,0,0,0,0},  \n
  FrameFextApplied = FrameFext = LastFrame of the DHTable.    \n
2)Force and Torque are by default false.  If they are set to  \n
  a different value, then all the internal forces and torques \n
  on the links are included in the output" ;

Gravitation::usage = "Gravitation[DHTable, MassTable, \
Vgravity, Force, Torque]
gives the\n
force(torque) that is necesary to apply at each joint to keep\n
the robot, given by DHTable, in static conditions when under\n
the effect of Gravity.\n
1)MassTable is composed of four parameters for each link. The\n
  first one defines its mass and the next three define\n
```

```
  the location of the center of mass.\n
2)Vgravity: 3 x 1 vector that represents the direction and value\n
  of the gravity acceleration. By default, Vgravity={0,0,-G}. \n
3)Force and Torque are by default false.  If they are set to \n
  a different value, then all the internal forces and torques\n
  on the links are included in the output" ;

EndPackage[]

Null
```

APPENDIX B-2

SML-C.m, SYMBOLIC MANIPULATOR LABORATORY

By Santiago March-Leuba, November 1991

>> This file contains routines for robot manipulator modeling. Several functions are included to calculate the kinematic and static models of a manipulator. In addition, some trigonometric reductions and output form functions developed for use in robotics are incorporated. This file is called SML-C, in that it performs models based on Craig's notation. To use Paul's notation, look for the file called SML-P. A list of the functions in SML is presented below with help on how to use them. <<

```
BeginPackage["Robotics`SML-C`"];
```

GENERAL HELP

```
SML::usage = "Symbolic Manipulator Laboratory (SML)     \n
was writen by Santiago March-Leuba at the Oak Ridge      \n
National Laboratory to be used in symbolic modeling of   \n
robot manipulators. Functions ending in C are to be used \n
with Craig's notation; otherwhise use Paul's notation.   \n
List of Functions on SML:   \n
1.- Trigonometric Reductions and Output Forms \n
      RedTrig, RedTrigExp, RedAngle,        \n
      ListOutput, ROut, RFForm, RCForm.     \n
2.- Kinematics Functions:                   \n
      OperTransform,  Rot,  Pos,            \n
      DirectKinEq,  InverseKin,  JacobianP, \n
      OperTransformC, RotC, PosC,           \n
      DirectKinEqC, InverseKinC, JacobianC. \n
3.- Static Forces Functions:                \n
      StaticForces,  Forces,  Gravitation,  \n
      StaticForcesC, ForcesC, GravitationC. \n
4.- Miscellaneous Functions:         \n
      CrossProd, PosVector.          \n
Input Tables: \n
1.- Denavit-Hartenberg table (DHTable).     \n
2.- Mass parameters table (MassTable).";
```

```
DHTable::usage = "DHTable is the input used in kinematic \n
and static forces functions.  It is entered on the form:    \n
a)when using Paul's notation: \n
  DHTable = {{q1, alf1, a1, d1}, \n
             {q2, alf2, a2, d2}, \n
                    . . .\n
             {qn, alfn, an, dn}}, \n
  where: qi   = angle from Xi-1 to Xi, about Zi-1, \n
         alfi = angle from Zi-1 to Zi, about Xi,   \n
         ai   = length from Zi-1 to Zi, along Xi,  \n
         di   = length from Xi-1 to Xi, about Zi-1.\n \n
a)when using Craig's notation: \n
  DHTable = {{q1,  alf0,  a0,   d1}, \n
             {q2,  alf1,  a1,   d2}, \n
                    . . .\n
             {qn, alfn-1, an-1, dn}}, \n
  where: qi   = angle from Xi-1 to Xi, about Zi,   \n
         alfi = angle from Zi-1 to Zi, about Xi-1, \n
         ai   = length from Zi-1 to Zi, along Xi-1,\n
         di   = length from Xi-1 to Xi, about Zi.";

MassTable::usage = "MassTable is the input necessary for \n
the functions Gravitation and StaticForces.  It is entered  \n
on the form:    \n
  MassTable = {{m1, mx1, my1, my1}, \n
               {m2, mx2, mz2, mz2}, \n
                      . . .\n
               {mn, mxn, myn, mzn}}, \n
 where: (1) mi is the mass of link i; and (2) mxi, myi, and\n
 mzi are the locations of the centroid of link i along the \n
 X-, Y-, and Z-axes of the coordinate frame attached to   \n
 that link.";
```

TRIGONOMETRIC REDUCTIONS

```
RedTrig::usage = "RedTrig[expr]
gives expr Trigonometricaly reduced using    \n
classical pattern matching.";

RedTrigExp::usage = "RedTrigExp[expr]
gives expr Trigonometricaly reduced using  \n
pseudo-exponential functions. After using RedTrigExp, any     \n
of the four following functions can help to obtain a          \n
simpler output:                                               \n
  ToMin[expr], ToMinC[expr], ToPaper[expr], ToMinCS[expr].    \n
They have been listed in order of time consumption and        \n
sophistication. If the expression to deal with is long        \n
and complicated, it is to the user advantage to use one after \n
the other, checking at any step to determine whether the      \n
output is good enough.";
```

OUTPUT FORMS

```
RedAngle::usage = "RedAngle[expr, var, big]
gives the expression (expr),  \n
regardless of its dimensions, reducing the form of tangents, \n
sines, and cosines: Tan[q1] -> T1, Sin[q2+q3] -> S23,        \n
Cos[q1-q4] -> C1M4. By default, the angles are defined by qi;\n
to use a different one, specify it on var;                   \n
          example: using ti makes var=t.                     \n
When big > 0, subindexes bigger than 9 are allowed.";

ListOutput::usage = "ListOutput[List, name, form, var, big]
prints the given List,  \n
regardless of its dimensions, as a multidimensional vector \n
with its subindexes. The printed name of the list is       \n
given by name, ''List'' being its default.                 \n
Optional parameter form:                                   \n
  form = Text(by default), gives the list on Text Form.    \n
  form = C, gives the list on C Form.                      \n
  form = RC, gives the list on C Form, reducing the form of \n
    tangents, sines, and cosines: Tan[q1] -> T1,           \n
    Sin[q2+q3] -> S23,  Cos[q1-q4] -> C1M4. By default,    \n
    the angles are defined by qi; to use a     \n
    different one, specify it on var;                    \n
              example: using ti make var=t.              \n
  form = F, gives the list on FORTRAN Form.              \n
  form = RF, gives the list on FORTRAN Form reducing the form \n
    of sines and cosines. \n
When big > 0, subindexes bigger than 9 are allowed.";
```

FUNCTION -- Auxiliar --

```
CrossProd::usage = "CrossProd[V_,U_]
gives the cross product of the two vectors V and U.";

PosVector::usage = "PosVector[Matrix_]
gives the posititon vector of the \n
4 x 4 homogeneous transformation Matrix.";
```

FUNCTION -- " OPERATORS " -- (Craig's notation) --

```
OperTransformC::usage = "OperTransformC[DHTable, RefFrame, Frame]
gives the 4 x 4 homogeneous transformation operator that \n
relates Frame and RefFrame. \n
Enter the Denavit-Hartenberg table in Craig's notation." ;
```

■ FUNCTION -- " ROTATIONAL MATRIX " --

```
RotC::usage = " RotC[DHTable, RefFrame, Frame]
gives the rotation matrix that \n
relates Frame and RefFrame. Enter the Denavit-Hartenberg table\n
in Craig's notation." ;
```

■ FUNCTION -- " POSITION VECTOR " --

```
PosC::usage = " PosC[DHTable, RefFrame, Frame]
gives the Position Vector that \n
relates Frame and RefFrame. Enter the Denavit-Hartenberg \n
table in Craig's notation." ;
```

■ FUNCTION -- " PAUL'S TO CRAIG'S NOTATION " --

```
PaulToCraig::usage = " PaulToCraig[PDHTable]
transforms the Denavit-Hartenberg Table \n
from Paul's To Craig's notation." ;
```

■ FUNCTION -- " KINEMATIC EQUATIONS " --

```
DirectKinEqC::usage = "\
DirectKinEqC[DHTable, EulerOrder, BaseFrame, LastFrame]
gives \n
the kinematic equations :Px, Py, Pz, Roll, Pitch, Yaw. Enter  \n
the D-H Table in Craig's notation.                            \n
  The default for BaseFrame is '0' and for LastFrame is the   \n
number of rows of the given D-H Table'.                       \n
EulerOrder is the order of the Euler angles. Its default is   \n
ZYX where the function gives the ZYX Euler angles, or the XYZ \n
angles about fixed axes (both solutions are the same).        \n
  Use EulerOrder = ZYZ to get the ZYZ Euler angles.           \n
  On output, the angles are Roll:with respect to X; Pitch:with \n
respect to Y; and Yaw:with respect to Z.                      \n
To use the defaults, call the function with only the first    \n
parameter: DirectKinEqC[DHTable]." ;
```

■ FUNCTION -- " INVERSE KINEMATICS " --

```
InverseKinC::usage = "InverseKinC[DHTable]
gives the inverse kinematics solution, when \n
it exist, of a 6-DOF manipulator with the last three rotational \n
axes intersected. Enter the D-H Table in Craig's notation." ;
```

■ FUNCTION -- " JACOBIAN " --

```
JacobianC::usage = " JacobianC[DHTable, RefFrame]
gives the Jacobian of the robot \n
written with respect to any specified reference frame:          \n
    {Cartesian Coordinates speeds} = Jacobian . {Joint speeds},\n
where:{Cart. Coord. speeds} = {dx/dt, dy/dt, dz/dt, wx, wy, wz}\n
written with respect to RefFrame. The default for RefFrame is  \n
the number of rows of the given DHTable, \n
given in Craig's notation.";

JacobTransformC::usage = "\
 JacobTransformC[DHTable,newFrame,oldFrame,oldJac]
transforms \n
the Jacobian of the robot (oldJac), written with respect to any\n
specified reference frame (oldFrame), to a different frame     \n
(newFrame). Enter the DHTable of the robot in Craig's notation.";
```

■ FUNCTION -- " STATIC AND GRAVITATIONAL FORCES " --

```
StaticForcesC::usage = "StaticForcesC[DHTable, MassTable, \
Vgravity, Fext, FrameFext, FrameFextApplied, Force, Torque]
gives the force(torque) that\n
is necessary to apply at each joint to keep the robot in static\n
conditions when under the effect of an external force Fext\n
and/or Gravity.\n
1)MassTable is composed of four parameters for each link. The\n
  first one defines its mass and the next three define\n
  the location of the center of mass.\n
2)Vgravity: 3 x 1 vector that represents the direction and value\n
  of the gravity acceleration. By default, Vgravity={0,0,-G}. \n
3)Fext: 6 x 1 vector that represents the external force(torque)\n
  {Fx, Fy, Fz, Tx, Ty, Tz} along the three axes applied to \n
  any coordinate frame (FrameFextApplied) and written with respect`
  to any frame (FrameFext). By default, Fext = {0,0,0,0,0,0},\n
  FrameFextApplied = FrameFext = LastFrame of the DHTable.\n
4)Force and Torque are by default false. Setting them to true\n
  includes in the output all the internal reactions on the links.";

ForcesC::usage = "ForcesC[DHTable, Fext, \
FrameFext, FrameFextApplied, Force, Torque]\n
gives the force(torque) that is necesary to apply at each     \n
joint to keep the robot, given by DHTable, in static          \n
conditions when under the effect of an external force Fext.   \n
1)Fext: 6 x 1 vector that represents the external force(torque) \n
  {Fx, Fy, Fz, Tx, Ty, Tz} along the three axes.  Applied to  \n
  any coordinate frame (FrameFextApplied) and written with respect
  to any frame (FrameFext). By default, Fext = {0,0,0,0,0,0},  \n
  FrameFextApplied = FrameFext = LastFrame of the DHTable.    \n
2)Force and Torque are by default false.  If they are set to  \n
  a different value, then all the internal forces and torques \n
  on the links are included in the output" ;

GravitationC::usage = "GravitationC[DHTable, MassTable, \
Vgravity, Force, Torque]
gives the\n
force(torque) that is necesary to apply at each joint to keep\n
the robot, given by DHTable, in static conditions when under\n
the effect of Gravity.\n
1)MassTable is composed of four parameters for each link. The\n
  first one defines its mass and the next three define\n
  the location of the center of mass.\n
2)Vgravity: 3 x 1 vector that represents the direction and value\n
  of the gravity acceleration. By default, Vgravity={0,0,-G}. \n
```

```
3)Force and Torque are by default false.  If they are set to \n
  a different value, then all the internal forces and torques\n
  on the links are included in the output" ;

EndPackage[];

Null
```

APPENDIX B-3

RedTrig.m, SYMBOLIC MANIPULATOR LABORATORY

By Santiago March-Leuba, November 1991

>> This file contains routines for trigonometric reductions and output forms developed to be useful in robotics.

If SML-C or SML-P are to be loaded, this file is not needed, because all its functions are included in the other two packages. <<

```
BeginPackage["Robotics`RedTrig`"]
```

GENERAL HELP

```
SML::usage = "Symbolic Manipulator Laboratory (SML)     \n
was writen by Santiago March-Leuba at the Oak Ridge      \n
National Laboratory to be used in symbolic modeling of   \n
robot manipulators. Functions ending in C are to be used \n
with Craig's notation; otherwhise use Paul's notation.   \n
List of Functions on SML:   \n
1.- Trigonometric Reductions and Output Forms \n
      RedTrig, RedTrigExp, RedAngle,          \n
      ListOutput, ROut, RFForm, RCForm.       \n
2.- Kinematics Functions:                     \n
      OperTransform,  Rot,  Pos,              \n
      DirectKinEq,  InverseKin,  JacobianP,   \n
      OperTransformC, RotC, PosC,             \n
      DirectKinEqC, InverseKinC, JacobianC.   \n
3.- Static Forces Functions:                  \n
      StaticForces,  Forces,  Gravitation,    \n
      StaticForcesC, ForcesC, GravitationC.   \n
4.- Miscellaneous Functions:          \n
      CrossProd, PosVector.           \n
Input Tables: \n
1.- Denavit-Hartenberg table (DHTable).       \n
2.- Mass parameters table (MassTable).";
```

TRIGONOMETRIC REDUCTIONS

```
RedTrig::usage = "RedTrig[expr]
gives expr Trigonometricaly reduced using    \n
classical pattern matching.";

RedTrigExp::usage = "RedTrigExp[expr]
gives expr Trigonometricaly reduced using  \n
pseudo-exponential functions. After using RedTrigExp, any     \n
of the four following functions can help to obtain a          \n
simpler output:                                               \n
  ToMin[expr], ToMinC[expr], ToPaper[expr], ToMinCS[expr].    \n
They have been listed in order of time consumption and        \n
sophistication. If the expression to deal with is long        \n
and complicated, it is to the user advantage to use one after \n
the other, checking at any step to determine whether the      \n
output is good enough.";
```

OUTPUT FORMS

```
RedAngle::usage = "RedAngle[expr, var, big]
gives the expression (expr),  \n
regardless of its dimensions, reducing the form of tangents, \n
sines, and cosines: Tan[q1] -> T1, Sin[q2+q3] -> S23,        \n
Cos[q1-q4] -> C1M4. By default, the angles are defined by qi;\n
to use a different one, specify it on var;                   \n
          example: using ti makes var=t.                     \n
When big > 0, subindexes bigger than 9 are allowed.";

ListOutput::usage = "ListOutput[List, name, form, var, big]
prints the given List,  \n
regardless of its dimensions, as a multidimensional vector \n
with its subindexes. The printed name of the list is       \n
given by name, ''List'' being its default.                 \n
Optional parameter form:                                   \n
  form = Text(by default), gives the list on Text Form.    \n
  form = C, gives the list on C Form.                      \n
  form = RC, gives the list on C Form, reducing the form of \n
    tangents, sines, and cosines: Tan[q1] -> T1,           \n
    Sin[q2+q3] -> S23,  Cos[q1-q4] -> C1M4. By default,    \n
    the angles are defined by qi; to use a     \n
    different one, specify it on var;                    \n
              example: using ti make var=t.              \n
  form = F, gives the list on FORTRAN Form.              \n
  form = RF, gives the list on FORTRAN Form reducing the form \n
    of sines and cosines. \n
When big > 0, subindexes bigger than 9 are allowed.";
```

■ FUNCTION -- Auxiliar --

```
CrossProd::usage = "CrossProd[V_,U_]
gives the cross product of the two vectors V and U.";

PosVector::usage = "PosVector[Matrix_]
gives the posititon vector of the \n
4 x 4 homogeneous transformation Matrix.";
```

APPENDIX C

FUTURE ARMOR REARM SYSTEM (FARS) MANIPULATOR: KINEMATICS, WORK SPACE, STATIC FORCES MODEL, AND DESIGN

APPENDIX C

FUTURE ARMOR REARM SYSTEM MANIPULATOR: KINEMATICS, WORK SPACE, STATIC FORCES MODEL, AND DESIGN

This example application presents a complete study of the kinematics of the Future Armor Rearm System (FARS) Manipulator (Kress et al. 1991). The design of some of FARS lengths and angular constraints with the goal of optimization of the work space is also presented here. The manipulator coordinate system is shown in Figure C-1, from which the D-H Table in Craig's notation (Craig 1986) is obtained. It is a 5 degree-of-freedom (DOF) manipulator, of which the first four joints are rotational, and the fifth is prismatic.

The goal of the FARS vehicle, is to automatically reload the Army's new M1A1 Block III tank as shown in Figure C-2. The automated shell-handling hardware is composed of four major systems: the articulated boom and docking port used for connecting with the tank, the carousel used for storage and selection of shells, the lift table, and the boom conveyor used for transfer of shells along the boom into the M1A1 tank. The articulated boom plus the extra rotational DOF of the carousel constitute the FARS manipulator, the kinematics and work space of which are presented in this example application for the Symbolic Manipulator Laboratory (SML).

The shells are to be transfered through the interior of the manipulator. Angular joint constraints cannot be large, because the shells are long and cannot turn in a small angle. This size limitation creates a very constrained robot with a reduced work space. An optimization design of lengths and angular constraints is presented here to maximize the work space of the FARS manipulator.

The following pages with the example are printed directly from SML, thus presenting the same format as on the computer monitor. Note that bold characters here are either input for SML and Mathematica (Wolfram 1988) or text comments, and the plain nonbold represents output obtained from SML or Mathematica.

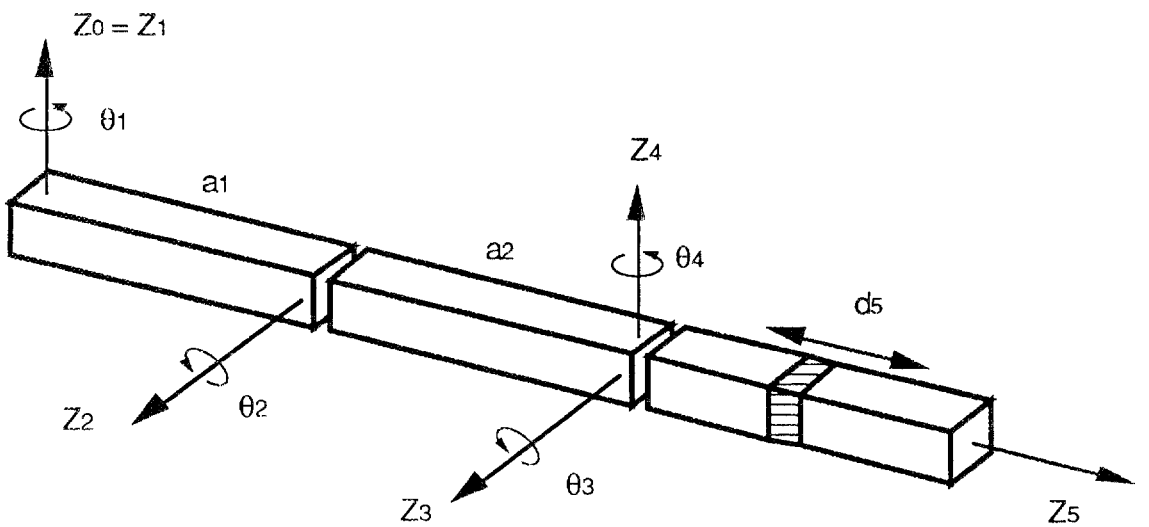


Figure C-1. FARS manipulator coordinate system definition.

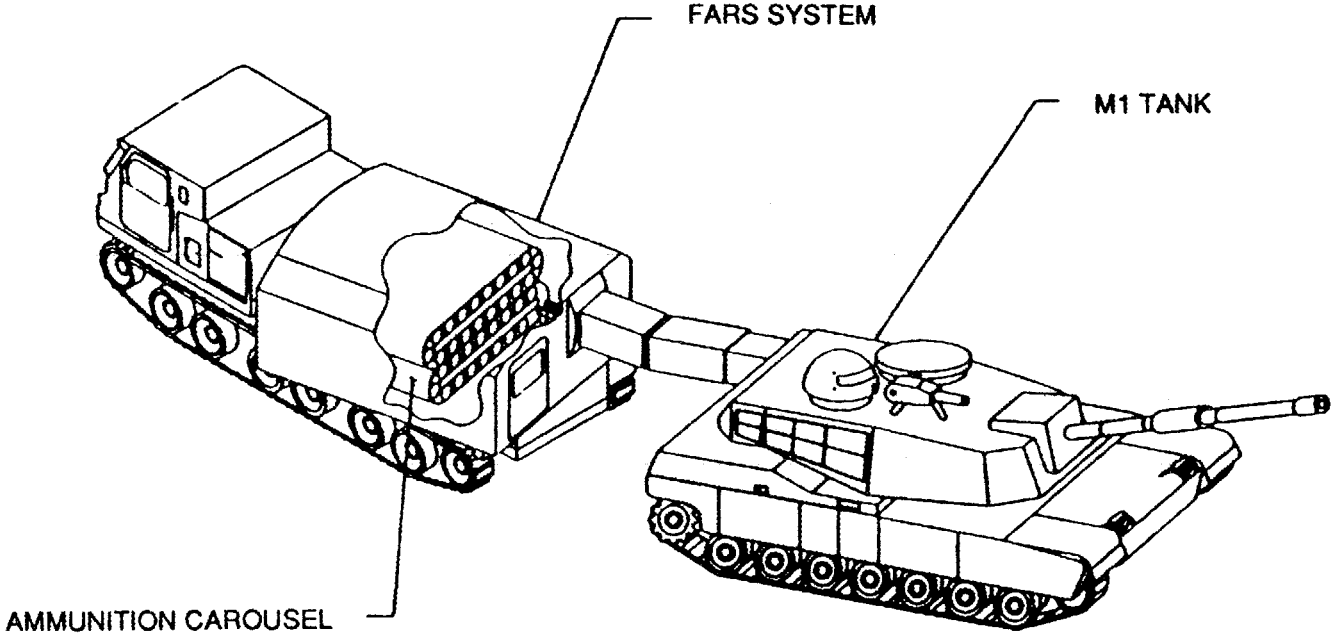


Figure C-2. FARS vehicle reloading an M1A1 tank.

FARS Manipulator D-H Table in Craig's notation and Mass Table.

```
FarsTable := {{q1,    0 , 0,   0 },
              {q2, Pi/2, a1,  0 },
              {q3,    0 , a2,  0 },
              {q4,-Pi/2, 0,   0 },
              { 0, Pi/2, 0,   d5}};

FarsMass := {{m1, xm1, ym1, zm1},
             {m2, xm2, ym2, zm2},
             { 0,  0 ,  0 ,  0 },
             {m4, xm4, ym4, zm4},
             {m5, xm5, ym5, zm5}}
```

Homogeneous transformation between second and fifth frames in two different output forms.

```
OperTransformC[FarsTable,2,5]

{{Cos[q3] Cos[q4], -Sin[q3], Cos[q3] Sin[q4],

   a2 + d5 Cos[q3] Sin[q4]},

  {Cos[q4] Sin[q3], Cos[q3], Sin[q3] Sin[q4],

   d5 Sin[q3] Sin[q4]}, {-Sin[q4], 0, Cos[q4], d5 Cos[q4]},

  {0, 0, 0, 1}}

MatrixForm[ RedAngle[%]]
```

C3 C4	-S3	C3 S4	a2 + C3 S4 d5
C4 S3	C3	S3 S4	S3 S4 d5
-S4	0	C4	C4 d5
0	0	0	1

■ Direct Kinematic equations.

```
DirectKinEqC[FarsTable]

/** KINEMATIC EQUATIONS **/
/*Position: Px, Py, Pz */
/*Orientation: */
   Roll: respect to X;
   Pitch: respect to Y;
   Yaw: respect to Z;  */
/* - Roll, Pitch, Yaw about the fixed axes X Y Z  .Or
/* - ZYX Euler angles
Px = C1 a1 + C1 C2 a2 + C4 S1 d5 + C1 C23 S4 d5;
Py = S1 a1 + C2 S1 a2 - C1 C4 d5 + C23 S1 S4 d5;
Pz = S2 a2 + S23 S4 d5;
Roll  = Atan2[C23, S23 S4];
Pitch = Atan2[-(C4 S23),
                        2   2                        2
                Sqrt[S23  S4  + (C1 C23 C4 - S1 S4) ]];
Yaw   = Atan2[C23 C4 S1 + C1 S4, C1 C23 C4 - S1 S4];
```

■ Obtain the Jacobian written with respect to the fifth frame.

```
FJ5 = RedAngle[ JacobianC[FarsTable,5] ]

{{S4 a1 + C2 S4 a2 + C23 d5, C4 S3 a2, 0, d5, 0},

  {-(C4 S23 d5), C3 a2 + S4 d5, S4 d5, 0, 0},

  {-(C4 a1) - C2 C4 a2, S3 S4 a2, 0, 0, 1},

  {C4 S23, -S4, -S4, 0, 0}, {C23, 0, 0, 1, 0},

  {S23 S4, C4, C4, 0, 0}}
```

■ **Obtain the Jacobian written with respect to the base frame.**

```
FJ0 = JacobianC[FarsTable,0];
ListOutput[ Collection[FJ0,{q1,q2}], "FJ0"]
```

```
FJ0(1,1) = C1 C4 d5 + S1 (-a1 - C2 a2 - C23 S4 d5)
FJ0(2,1) = C4 S1 d5 + C1 (a1 + C2 a2 + C23 S4 d5)
FJ0(3,1) = 0
FJ0(4,1) = 0
FJ0(5,1) = 0
FJ0(6,1) = 1

FJ0(1,2) = C1 (-(S2 a2) - S23 S4 d5)
FJ0(2,2) = S1 (-(S2 a2) - S23 S4 d5)
FJ0(3,2) = C2 a2 + C23 S4 d5
FJ0(4,2) = S1
FJ0(5,2) = -C1
FJ0(6,2) = 0

FJ0(1,3) = -(C1 S23 S4 d5)
FJ0(2,3) = -(S1 S23 S4 d5)
FJ0(3,3) = C23 S4 d5
FJ0(4,3) = S1
FJ0(5,3) = -C1
FJ0(6,3) = 0

FJ0(1,4) = C1 C23 C4 d5 - S1 S4 d5
FJ0(2,4) = C23 C4 S1 d5 + C1 S4 d5
FJ0(3,4) = C4 S23 d5
FJ0(4,4) = -(C1 S23)
FJ0(5,4) = -(S1 S23)
FJ0(6,4) = C23

FJ0(1,5) = C4 S1 + C1 C23 S4
FJ0(2,5) = -(C1 C4) + C23 S1 S4
FJ0(3,5) = S23 S4
FJ0(4,5) = 0
FJ0(5,5) = 0
FJ0(6,5) = 0
```

■ **Static forces model: force/torque that each joint has to support to keep the manipulator in static equilibrium under the effect of an external general force {Fx,Fy,Fz,Mx,My,Mz} applied at the end-effector.**

```
FFars = ForcesC[FarsTable, {Fx,Fy,Fz,Mx,My,Mz}]

"/** STATIC AND GRAVITATIONAL FORCES COMPENSATION **/"
"/* Fs[i] = Force exerted on link i by link i-1  */"
"/* Ns[i] = Torque exerted on link i by link i-1 */"
"/* M[i]  = Necesary Force/Torque in Motor i     */"
M[1] = -(-((C4*Fz - Fx*S4)*a1) +
        S2*(-(S3*(My + Fx*d5)) +
        C3*(Mz*S4 + C4*(Mx - Fy*d5))) +
        C2*(-((C4*Fz - Fx*S4)*a2) + C3*(My + Fx*d5) +
        S3*(Mz*S4 + C4*(Mx - Fy*d5))));
M[2] = -(C4*Mz + (C3*Fy + S3*(C4*Fx + Fz*S4))*a2 -
                                  S4*(Mx - Fy*d5));
M[3] = -(C4*Mz - S4*(Mx - Fy*d5));
M[4] = -(My + Fx*d5);
M[5] = -Fz;
```

■ Optimization Design.

□ An optimization design of lengths and angular constraints is presented in the next example for which the goal is to maximize the work space of the FARS manipulator.

■ First step was to obtain the inverse kinematics.

□ The known input is the position vector and the orientation of the last link (see Figure C-1) of the manipulator. Let us define T05 as the homogeneous transformation between the first and the fifth frame. The unknown parameters are called U in the following:

```
T05 = {{ U, U, ax, Px},
       { U, U, ay, Py},
       { U, U, az, Pz},
       { 0, 0,  0,  1}};
```

□ The equations to solve for the inverse kinematics were obtained from SML as follows:

```
A10 = OperTransformC[FarsTable, 1, 0];
A53 = OperTransformC[FarsTable, 5, 3];
A13 = OperTransformC[FarsTable, 1, 3];
K13 = PosVector[ A10 . T05 . A53 ] ;
P13 = PosVector[ A13 ] ;
Do[ Print[ RedAngle[K13[[i]]], " = ", RedAngle[P13[[i]]]
                                                  ] , {i,1,3}]
```

```
C1 Px + Py S1 - (C1 ax + S1 ay) d5 = a1 + C2 a2
C1 Py - Px S1 - (-(S1 ax) + C1 ay) d5 = 0
Pz - az d5 = S2 a2
```

□ There are only three unknowns in the above equations: q1, q2, and d5. Adding the square of the three equations and reducing them trigonometrically, a fourth order polynomial in d5 is obtained that can be solved with the help of *Mathematica*. Then, q1 and q2 can be solved from the same equations.

- Once q1, q2, and d5 are known, it can be solved for q4, q5, and q6 by:

```
A10 = OperTransformC[FarsTable, 1, 0];
A15 = OperTransformC[FarsTable, 1, 5];
K15 = Table[ (A10 . T05)[[i,3]], {i,1,3} ] ;
P15 = Table[ A15 [[i,3]], {i,1,3}] ;
Do[ Print[ RedAngle[K15[[i]]], " = ", RedAngle[P15[[i]]]
                                                ] , {i,1,3}]
```

```
C1 ax + S1 ay = C23 S4
-(S1 ax) + C1 ay = -C4
az = S23 S4
```

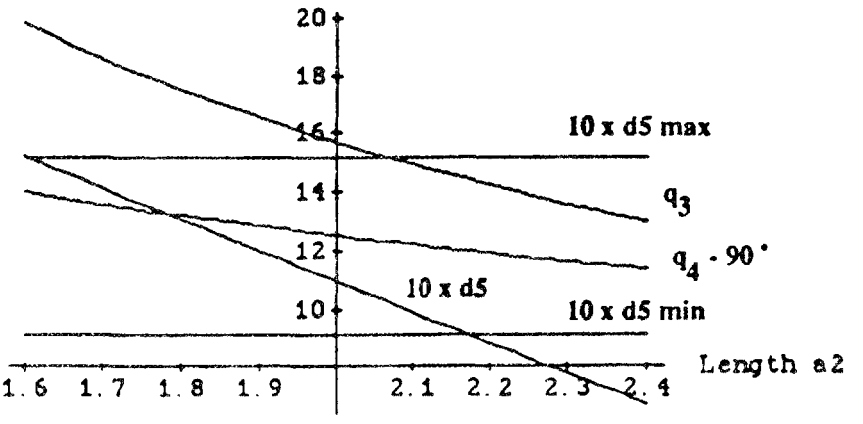
- A function was created that solves for the inverse kinematics of FARS manipulator.

```
FarsInvKin::usage = " FarsInvKin gives the Inverse
 Kinematic Solution for FARS Manipulator.  This
 Function has seven parameters, being the last three
 of them optional.
 FarsInvKin[Px_, Py_, Pz_, AngY_, AngZ_,
                    a1_:1.98120, a2_:2.02564, Prt_:0]";
```

- FarsInvKin calculates the joint angles (q1, q2, q3, and q4), and the joint distance d5 necessary to reach, with the last link of FARS, a position and orientation given by Px, Py, Pz, AngY, and AngZ. The solution is a function of the lengths a1 and a2 of the manipulator, which allow us to plot the joint solutions as functions of a2, when a1 is fixed at 6.5 feet (1.98120 meters). The more conflicting joint constraints are for q3, q4, and d5. Thus, only these are ploted here.

```
InvKin[a2_] := InvKin[a2] =
                FarsInvKin[5.03, 0, 0, 10, 10, 1.9812, a2]
Plot3[a2_] := InvKin[a2][[3]];
Plot4[a2_] := InvKin[a2][[4]] - 90;
Plot5[a2_] := InvKin[a2][[5]];

Plot[ {Plot3[a2], Plot4[a2], Plot5[a2] 10, 9.1, 15.2},
      {a2, 1.6, 2.4},
       Ticks->{Range[1.6,2.4,.1],Range[8,24,2]},
       AxesLabel -> {"  Length a2"," "}]
```



-Graphics-

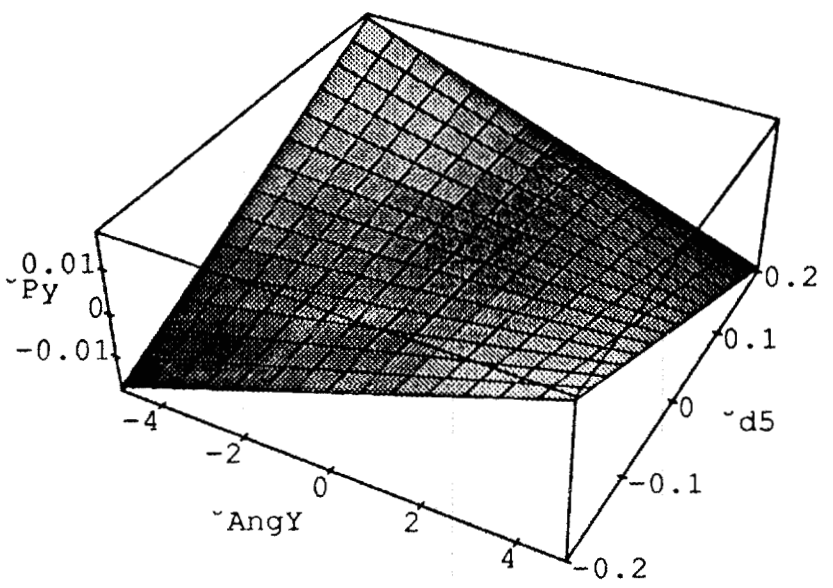
☐ The figure above shows angles q3 and q4 (q4-90) in degrees, and length d5 (10 x d5) in meters as functions of the second link length a2, in meters. Only because SML allows symbolic modeling we could obtain this plot. These plots were obtained for different configurations of the robot. The following conditions for a good design of FARS manipulator were obtained by analyzing a series of plots like the one above:

- Use the largest posible value of a2. It shoul be at least 1.98 meters.
- Joint 3 is to have the largest possible range of motion to allow a large value of q3.
- Choose d5 to be as small as possible.
- Extend the reach along the x axis to the largest possible value making Px at least 5.3 meters.

■ Work Space study.

□ **Different soubroutines were created to study the effect of variation of lengths on the work space of the manipulator. The next figure shows the relation between the position (Py) and orientation (AngY) and the last link length (d5).**

FARS at: Px=5.5, Py=0, Pz=0, AngY=AngZ=0



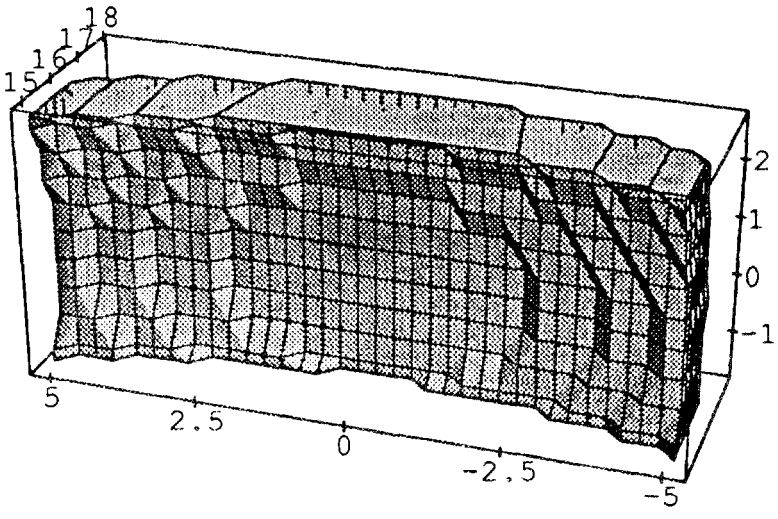
-SurfaceGraphics-

- Finally, a three dimensional solid plot of the work space is obtained as shown below. This was obtained using the symbolic inverse kinematics solution.

FARS 3D WORK SPACE FOR AngY and AngZ EQUAL TO 0 DEGREES.

Constraints:

* a1 = 6.5 feet
* a2 = 6.6458 feet
* -90 < A1 < 90 degrees
* -15 < A2 < 32 degrees
* -24 < A3 < 24 degrees
* (90 - 24) < A4 < (90 + 24) degrees
* 3 < d5 < 5 feet



APPENDIX D

OTHER SAMPLE APPLICATIONS

APPENDIX D-1

CENTER FOR ENGINEERING SYSTEMS ADVANCED RESEARCH MANIPULATOR FORWARD AND INVERSE KINEMATICS

This example application presents the implementation of the forward kinematics and an algorithm for the inverse kinematics of the Center for Engineering Systems Advanced Research manipulator (CESARm) of Oak Ridge National Laboratory based on a paper presented by Dubey, Euler, and Babcock (1988).

The CESARm manipulator coordinate system is shown in Figure D-1, from which the D-H Table in Paul's notation (Paul 1981) is obtained. Figure D-2 shows the CESARm (slave) and the KRAFT (master) used together in a teleoperated robotic system at Oak Ridge National Laboratory.

CESARm is a 7-DOF manipulator. Because of its redundant configuration, special algorithms such as the one developed by Dubey, Euler, and Babcock (1988) are necessary to control it. Following the algorithm described in their paper and using SML, computational efficient closed-form solutions are obtained for the joint rates as a function of the Cartesian velocities of the end effector. The following pages with the example are printed directly from SML, thus presenting the same format as on the computer monitor. Note that bold characters here are either input for SML or text comments, and the nonbold text represents output obtained from SML.

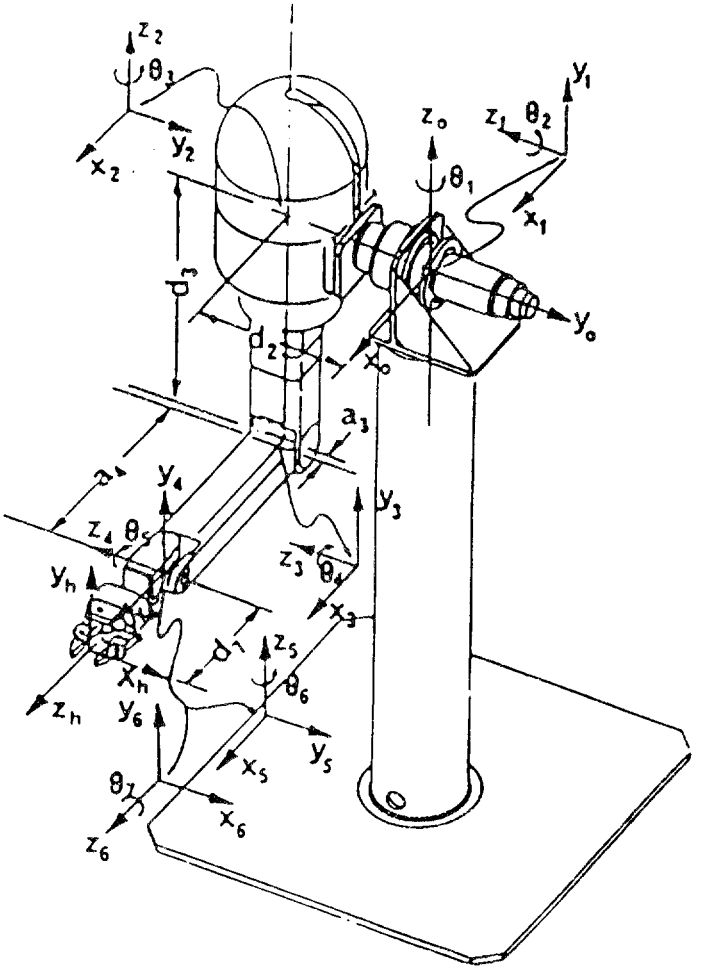


Figure D-1. CESARm coordinate system definition.

Source: Dubey, Euler, and Babcock 1988.

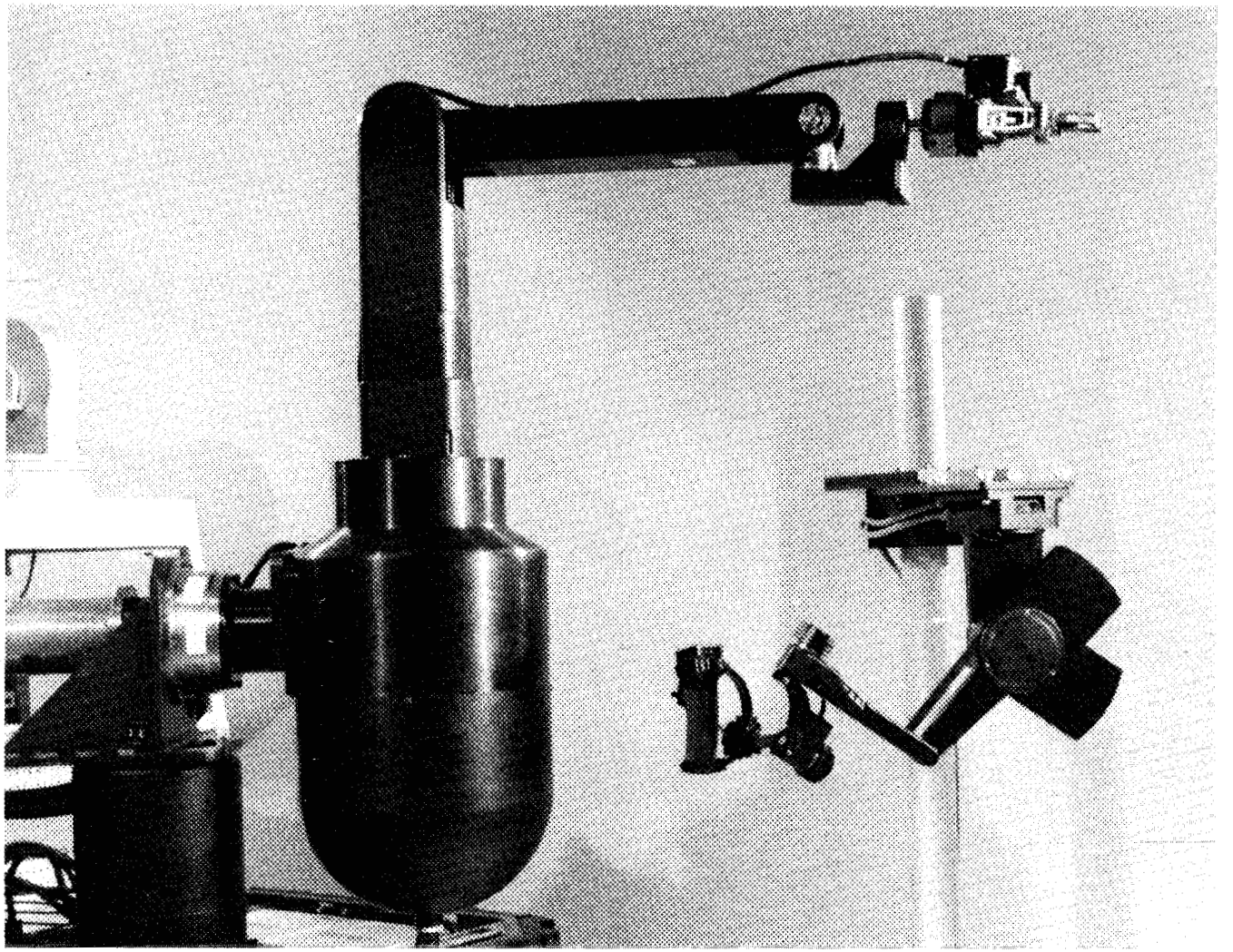


Figure D-2. Teleoperated system: CESARm and the KRAFT master (ORNL-Photo 5224-89).

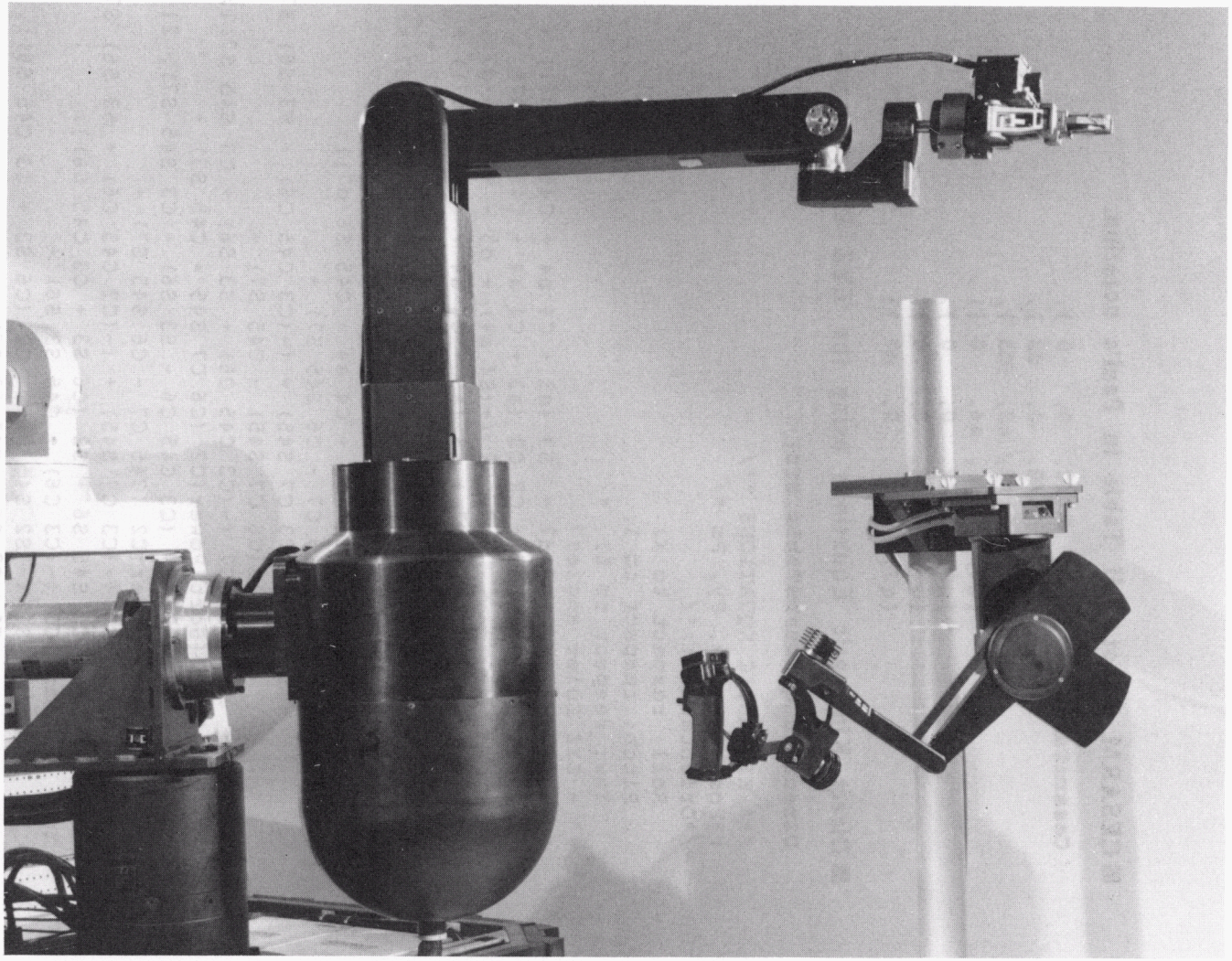


Figure D-2. Teleoperated system: CESARm and the KRAFT master (ORNL-Photo 5224-89).

■ CESARM Robot D-H Table in Paul's notation.

```
CesarmTable := {{q1,  Pi/2,  0,   0 },
                {q2, -Pi/2,  0,  d2 },
                {q3,  Pi/2, a3, -d3 },
                {q4,   0  , a4,   0 },
                {q5, -Pi/2,  0,   0 },
                {q6,  Pi/2,  0,   0 },
                {q7,   0  ,  0,  d7 }}
```

■ Direct Kinematic Equations using the ZYZ Euler angles

```
DirectKinEq[CesarmTable, ZYZ]

/** KINEMATIC EQUATIONS **/
/*Position: Px, Py, Pz */
/*Orientation: */
   Roll: respect to X;
   Pitch: respect to Y;
   Yaw: respect to Z;  */
/* - ZYZ Euler angles
Px = S1 (d2 + C3 C6 d7 -  S3 (a3 + C4 a4 + C45 S6 d7)) +
     C1 (C2 C6 S3 d7 + C2 C3 (a3 + C4 a4 + C45 S6 d7) +
                        S2 (-(S4 a4) + d3 - S45 S6 d7));
Py = C1 (-d2 - C3 C6 d7 + S3 (a3 + C4 a4 + C45 S6 d7)) +
     S1 (C2 C6 S3 d7 + C2 C3 (a3 + C4 a4 + C45 S6 d7) +
                        S2 (-(S4 a4) + d3 - S45 S6 d7));
Pz = C2 (S4 a4 - d3 + S45 S6 d7) +
     S2 (C6 S3 d7 + C3 (a3 + C4 a4 + C45 S6 d7));
Roll  = Atan2[C2 (C45 C7 - C6 S45 S7) +
              S2 (-(C3 C7 S45) + (-(C3 C45 C6) + S3 S6) S7),
              C2 (-(C6 C7 S45) - C45 S7) +
              S2 (C7 (-(C3 C45 C6) + S3 S6) + C3 S45 S7)];
Pitch = Atan2[Sqrt[Power[C2 (C6 C7 S45 + C45 S7) +
              S2 (C7 (C3 C45 C6 - S3 S6) - C3 S45 S7), 2] +
              Power[C2 (C45 C7 - C6 S45 S7) +
              S2 (-(C3 C7 S45) + (-(C3 C45 C6) + S3 S6) S7), 2]],
              C2 S45 S6 + S2 (C6 S3 + C3 C45 S6)];
Yaw   = Atan2[C1 (-(C3 C6) + C45 S3 S6) +
              S1 (-(S2 S45 S6) + C2 (C6 S3 + C3 C45 S6)),
              S1 (C3 C6 - C45 S3 S6) +
              C1 (-(S2 S45 S6) + C2 (C6 S3 + C3 C45 S6))];
```

■ **Obtain the Jacobian from the wrist written with respect to the third frame. Making d7 = 0, the Jacobian from the wrist instead from the end-effector will be obtained.**

```
CesarmTable := CesarmTable /. d7 -> 0;

J3 = RedTrig[JacobianP[CesarmTable,3]];
ListOutput[Collect[J3,{Cos[q3],Sin[q3]}],"Jac3"]

Jac3(1,1) = C2 C3 d2 + S3 (-(S2 S4 a4) + S2 d3)
Jac3(2,1) = S3 (S2 a3 + C4 S2 a4) - S2 d2
Jac3(3,1) = -(C2 a3) - C2 C4 a4 + C2 S3 d2 + C3 (S2 S4 a4 - S2 d3)
Jac3(4,1) = C3 S2
Jac3(5,1) = C2
Jac3(6,1) = S2 S3

Jac3(1,2) = C3 (-(S4 a4) + d3)
Jac3(2,2) = C3 (a3 + C4 a4)
Jac3(3,2) = S3 (-(S4 a4) + d3)
Jac3(4,2) = -S3
Jac3(5,2) = 0
Jac3(6,2) = C3

Jac3(1,3) = 0
Jac3(2,3) = 0
Jac3(3,3) = -a3 - C4 a4
Jac3(4,3) = 0
Jac3(5,3) = 1
Jac3(6,3) = 0

Jac3(1,4) = -(S4 a4)
Jac3(2,4) = C4 a4
Jac3(3,4) = 0
Jac3(4,4) = 0
Jac3(5,4) = 0
Jac3(6,4) = 1

Jac3(1,5) = 0
Jac3(2,5) = 0
Jac3(3,5) = 0
Jac3(4,5) = 0
Jac3(5,5) = 0
Jac3(6,5) = 1

Jac3(1,6) = 0
Jac3(2,6) = 0
```

```
Jac3(3,6) = 0
Jac3(4,6) = -S45
Jac3(5,6) = C45
Jac3(6,6) = 0

Jac3(1,7) = 0
Jac3(2,7) = 0
Jac3(3,7) = 0
Jac3(4,7) = C45 S6
Jac3(5,7) = S45 S6
Jac3(6,7) = C6
```

■ Contruct a not singular jacobian J* from any six independent columns of the Jacobian. Dropping the second column the following Jacobian is obtained.

```
Jstar = Table[ {J3[[j,1]],J3[[j,3]],J3[[j,4]],
                J3[[j,5]],J3[[j,6]],J3[[j,7]]}, {j,1,6}];
```

```
RedAngle[Jstar]
```

```
{{-(S2 S3 S4 a4) + C2 C3 d2 + S2 S3 d3, 0, -(S4 a4), 0, 0, 0},

  {S2 S3 a3 + C4 S2 S3 a4 - S2 d2, 0, C4 a4, 0, 0, 0},

  {-(C2 a3) - C2 C4 a4 + C3 S2 S4 a4 + C2 S3 d2 - C3 S2 d3,

   -a3 - C4 a4, 0, 0, 0, 0}, {C3 S2, 0, 0, 0, -S45, C45 S6},

  {C2, 1, 0, 0, C45, S45 S6}, {S2 S3, 0, 1, 1, 0, C6}}
```

■ Following the algorithm, the Jacobian can be decomposed in two matrices using the first three and last three rows.

```
J1star = Table[ Jstar[[i,j]] ,{i,1,3} ,{j,1,3}];
J2star = Table[ Jstar[[i,j]] ,{i,4,6} ,{j,1,6}];
```

■ The solution for the first three joint rates can be obtained from: J1star {θ1, θ3, θ4} = {x1, x2, x3}.

Furthermore, the solution using Mathematica is found by: {θ1, θ3, θ4} = Inverse[J1star] {x1, x2, x3}.

```
ListOutput[J1star]
```

```
List(1,1) = -(S2 S3 S4 a4) + C2 C3 d2 + S2 S3 d3
List(2,1) = S2 S3 a3 + C4 S2 S3 a4 - S2 d2
List(3,1) = -(C2 a3) - C2 C4 a4 + C3 S2 S4 a4 +
              C2 S3 d2 - C3 S2 d3

List(1,2) = 0
List(2,2) = 0
List(3,2) = -a3 - C4 a4

List(1,3) = -(S4 a4)
List(2,3) = C4 a4
List(3,3) = 0
```

```
J1starT = J1star . {t1,t3,t4};
ListOutput[J1starT, "x"]

x(1) = (-(S2 S3 S4 a4) + C2 C3 d2 + S2 S3 d3) t1 - S4 a4 t4
x(2) = (S2 S3 a3 + C4 S2 S3 a4 - S2 d2) t1 + C4 a4 t4
x(3) = (-(C2 a3) - C2 C4 a4 + C3 S2 S4 a4 + C2 S3 d2 -
         C3 S2 d3) t1 +   (-a3 - C4 a4) t3

Sol1 = Solve[{J1starT[[1]] == x1, J1starT[[2]] == x2}, {t1,t4}];
Sol2 = Solve[ J1starT[[3]] == x3,       {t3}];

tt1 = RedAngle[Together[t1 //. Sol1 [[1]] ]]

                    C4 x1 + S4 x2
-----------------------------------------------
S2 S3 S4 a3 + C2 C3 C4 d2 - S2 S4 d2 + C4 S2 S3 d3

tt4  = RedAngle[Together[t4 //. Sol1 [[1]] ]]

(-(S2 S3 a3 x1) - C4 S2 S3 a4 x1 + S2 d2 x1 - S2 S3 S4 a4 x2 +

    C2 C3 d2 x2 + S2 S3 d3 x2) /

  (S2 S3 S4 a3 a4 + C2 C3 C4 a4 d2 - S2 S4 a4 d2 + C4 S2 S3 a4 d3)

tt3  = RedAngle[ t3 //. Sol2 [[1]] ]

(-(C2 a3 t1) - C2 C4 a4 t1 + C3 S2 S4 a4 t1 + C2 S3 d2 t1 -

    C3 S2 d3 t1 - x3) / (a3 + C4 a4)
```

☐ Reducing terms

```
Den = Denominator[tt1]; "Den" == Den

Den == S2 S3 S4 a3 + C2 C3 C4 d2 - S2 S4 d2 + C4 S2 S3 d3

t1 == Numerator[tt1] Simplify[Den/Denominator[tt1] ] /"Den"

      C4 x1 + S4 x2
t1 == -------------
            Den

tt4 = Numerator[tt4] Simplify[Den/Denominator[tt4] ] /"Den" ;
t4 == Collect[Numerator[tt4],{x1,x2}] / Denominator[tt4]

t4 == ((-(S2 S3 a3) - C4 S2 S3 a4 + S2 d2) x1 +

     (-(S2 S3 S4 a4) + C2 C3 d2 + S2 S3 d3) x2) / (Den a4)
```

```
t3 == Collect[Numerator[tt3],{t1,x3}] / Denominator[tt3]

t3 == ((-(C2 a3) - C2 C4 a4 + C3 S2 S4 a4 + C2 S3 d2 -

        C3 S2 d3) t1 - x3) / (a3 + C4 a4)
```

■ The solution for the last three joint rates can be obtained from the following.

J2star {θ1, θ3, θ4} + J3star {θ5, θ6, θ7}= {x4, x5, x6},

{θ1, θ3, θ4} = Inverse[J1star] {x1, x2, x3}.

```
ListOutput[J2star]

List(1,1) = C3 S2
List(2,1) = C2
List(3,1) = S2 S3

List(1,2) = 0
List(2,2) = 1
List(3,2) = 0

List(1,3) = 0
List(2,3) = 0
List(3,3) = 1

List(1,4) = 0
List(2,4) = 0
List(3,4) = 1

List(1,5) = -S45
List(2,5) = C45
List(3,5) = 0

List(1,6) = C45 S6
List(2,6) = S45 S6
List(3,6) = C6

J2starT = J2star . {t1,t3,t4,t5,t6,t7};
ListOutput[J2starT, "x"]

x(1) = C3 S2 t1 - S45 t6 + C45 S6 t7
x(2) = C2 t1 + t3 + C45 t6 + S45 S6 t7
x(3) = S2 S3 t1 + t4 + t5 + C6 t7

Sol3 = Solve[{J2starT[[1]] == x4, J2starT[[2]] == x5}, {t6,t7}];
Sol4 = Solve[ J2starT[[3]] == x6,        {t5}];

tt5 = RedAngle[Together[t5 //. Sol4 [[1]] ]]
```

```
-(S2 S3 t1) - t4 - C6 t7 + x6
tt6  = RedAngle[Together[RedTrig[t6 //. Sol3 [[1]] ]]]
-(C2 C45 t1) + C3 S2 S45 t1 - C45 t3 - S45 x4 + C45 x5
tt7  = RedAngle[Together[RedTrig[t7 //. Sol3 [[1]] ]]]
-(C3 C45 S2 t1) - C2 S45 t1 - S45 t3 + C45 x4 + S45 x5
-----------------------------------------------------
                          S6
```

□ **Reducing terms using Mathematica**

```
t5 == tt5
t5 == -(S2 S3 t1) - t4 - C6 t7 + x6
t6 == Collect[tt6,{C45,S45}]
t6 == S45 (C3 S2 t1 - x4) + C45 (-(C2 t1) - t3 + x5)
t7 == Collect[Numerator[tt7],{C45,S45}] / Denominator[tt7]
      C45 (-(C3 S2 t1) + x4) + S45 (-(C2 t1) - t3 + x5)
t7 == ------------------------------------------------
                          S6
```

FINAL SOLUTION

```
Den == S2 S3 S4 a3 + C2 C3 C4 d2 - S2 S4 d2 + C4 S2 S3 d3

       C4 x1 + S4 x2
t1 == -------------
            Den

t3 == ((-(C2 a3) - C2 C4 a4 + C3 S2 S4 a4 + C2 S3 d2 -

         C3 S2 d3) t1 - x3) / (a3 + C4 a4)

t4 == ((-(S2 S3 a3) - C4 S2 S3 a4 + S2 d2) x1 +

      (-(S2 S3 S4 a4) + C2 C3 d2 + S2 S3 d3) x2) / (Den a4)

t5 == -(S2 S3 t1) - t4 - C6 t7 + x6

t6 == S45 (C3 S2 t1 - x4) + C45 (-(C2 t1) - t3 + x5)

       C45 (-(C3 S2 t1) + x4) + S45 (-(C2 t1) - t3 + x5)
t7 == ------------------------------------------------
                                S6
```

APPENDIX D-2

LABORATORY TELEROBOTIC MANIPULATOR (LTM) FORWARD AND INVERSE KINEMATICS AND GRAVITATIONAL COMPENSATION

This example application presents the implementation of an algorithm for the inverse kinematics of the Laboratory Telerobotic Manipulator (LTM) of Oak Ridge National Laboratory based on the paper presented by Dubey et al. (1989). The direct kinematic equations and the gravitation compensation model are also obtained by using SML.

The LTM coordinate system is shown in Figure D-3, from which the D-H Table in Paul's notation (Paul 1981) is obtained.

LTM is a 7-DOF manipulator; thus, it needs special algorithms such as the one developed by Dubey et al. (1989) to be controlled. Following the algorithm described in their paper and using SML, computational-efficient closed-form solutions are obtained for the joint rates as a function of the Cartesian velocities of the end effector. The following pages with the example are printed directly from SML, thus presenting the same format as on the computer monitor. Note that bold characters here are either input for SML or text comments, and the nonbold text represents output obtained from SML.

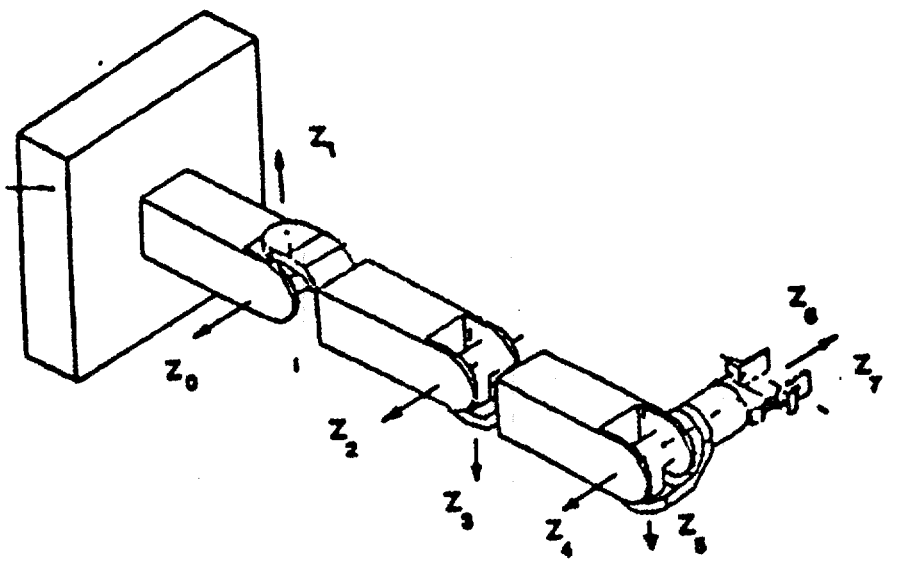


Figure D-3. LTM coordinate system definition.

Source: Dubey et al. 1989.

```
                        C2 (C3 C4 C5 C6 - C6 S3 S5 - C3 S4 S6))) +
                   S7 (S1 (C3 C5 - C4 S3 S5) +
                       C1 (S2 S4 S5 + C2 (C5 S3 + C3 C4 S5))), 2]]];
Yaw    = Atan2[C7 (C1 (C4 C5 C6 S3 + C3 C6 S5 - S3 S4 S6) +
                   S1 (S2 (C5 C6 S4 + C4 S6) +
                   C2 (C3 C4 C5 C6 - C6 S3 S5 - C3 S4 S6))) +
               S7 (C1 (-(C3 C5) + C4 S3 S5) +
                   S1 (S2 S4 S5 + C2 (C5 S3 + C3 C4 S5))),
               C7 (S1 (-(C4 C5 C6 S3) - C3 C6 S5 + S3 S4 S6) +
                   C1 (S2 (C5 C6 S4 + C4 S6) +
                   C2 (C3 C4 C5 C6 - C6 S3 S5 - C3 S4 S6))) +
               S7 (S1 (C3 C5 - C4 S3 S5) +
                   C1 (S2 S4 S5 + C2 (C5 S3 + C3 C4 S5)))];
```

- **Obtain the Jacobian from the wrist written with respect to the third frame. Making d7 = 0, we will obtain the Jacobian from the wrist instead from the end-effector.**

```
LTMTable := LTMTable /. d7 -> 0;

J3 = RedTrig[JacobianP[LTMTable,3]];
ListOutput[Collect[J3,{Cos[q3],Sin[q3]}],"Jac3"]

Jac3(1,1) = S3 (C2 a2 + S2 S4 a4)
Jac3(2,1) = -(C4 S2 S3 a4)
Jac3(3,1) = -(C2 C4 a4) + C3 (-(C2 a2) - S2 S4 a4)
Jac3(4,1) = -(C3 S2)
Jac3(5,1) = C2
Jac3(6,1) = -(S2 S3)

Jac3(1,2) = C3 S4 a4
Jac3(2,2) = -a2 - C3 C4 a4
Jac3(3,2) = S3 S4 a4
Jac3(4,2) = S3
Jac3(5,2) = 0
Jac3(6,2) = -C3

Jac3(1,3) = 0
Jac3(2,3) = 0
Jac3(3,3) = -(C4 a4)
Jac3(4,3) = 0
Jac3(5,3) = 1
Jac3(6,3) = 0

Jac3(1,4) = -(S4 a4)
Jac3(2,4) = C4 a4
Jac3(3,4) = 0
```

■ LTM Robot D-H Table in Paul's notation.

```
LTMTable := {{q1, -Pi/2,  0,   0 },
              {q2,  Pi/2, a2,   0 },
              {q3,  Pi/2,  0,   0 },
              {q4, -Pi/2, a4,   0 },
              {q5,  Pi/2,  0,   0 },
              {q6,  Pi/2,  0,   0 },
              {q7,   0  ,  0,  d7 }}
```

■ Direct Kinematic Equations.

```
DirectKinEq[LTMTable]

/** KINEMATIC EQUATIONS **/
/*Position: Px, Py, Pz */
/*Orientation: */
   Roll: respect to X;
   Pitch: respect to Y;
   Yaw: respect to Z;  */
/* - Roll, Pitch, Yaw about the fixed axes X Y Z  .Or
/* - ZYX Euler angles

Px = -(C4 S1 S3 a4) + C1 (C2 a2 + C2 C3 C4 a4 + S2 S4 a4) +
      (S1 (-(C3 S5 S6) + S3 (-(C6 S4) - C4 C5 S6)) +
       C1 (-(C2 S3 S5 S6) + C2 C3 (C6 S4 + C4 C5 S6) +
             S2 (-(C4 C6) + C5 S4 S6))) d7;
Py =  C1 C4 S3 a4 + S1 (S2 S4 a4 + C2 (a2 + C3 C4 a4)) +
     (C1 (C3 S5 S6 + S3 (C6 S4 + C4 C5 S6)) +
      S1 (-(C2 S3 S5 S6) + C2 C3 (C6 S4 + C4 C5 S6) +
           S2 (-(C4 C6) + C5 S4 S6))) d7;
Pz =  C2 S4 a4 + S2 (-a2 - C3 C4 a4) +
     (C2 (-(C4 C6) + C5 S4 S6) +
      S2 (S3 S5 S6 + C3 (-(C6 S4) - C4 C5 S6))) d7;

Roll  =Atan2[C7 (C2 S4 S5 + S2 (-(C5 S3) - C3 C4 S5)) +
             S7 (C2 (-(C5 C6 S4) - C4 S6) +
                 S2 (-(C6 S3 S5) + C3 (C4 C5 C6 - S4 S6))),
             C2 (-(C4 C6) + C5 S4 S6) +
             S2 (S3 S5 S6 + C3 (-(C6 S4) - C4 C5 S6))];
Pitch = Atan2[C7 (C2 (-(C5 C6 S4) - C4 S6) +
                  S2 (C6 (C3 C4 C5 - S3 S5) - C3 S4 S6)) +
               S7 (-(C2 S4 S5) + S2 (C5 S3 + C3 C4 S5)),
         Sqrt[Power[(C2 (-(C4 C6) + C5 S4 S6) +
                     S2 (S3 S5 S6 + C3 (-(C6 S4) - C4 C5 S6))),2] +
              Power[C7 (S1 (-(C4 C5 C6 S3) - C3 C6 S5 + S3 S4 S6) +
                        C1 (S2 (C5 C6 S4 + C4 S6) +
```

```
Jac3(4,4) = 0
Jac3(5,4) = 0
Jac3(6,4) = 1

Jac3(1,5) = 0
Jac3(2,5) = 0
Jac3(3,5) = 0
Jac3(4,5) = -S4
Jac3(5,5) = C4
Jac3(6,5) = 0

Jac3(1,6) = 0
Jac3(2,6) = 0
Jac3(3,6) = 0
Jac3(4,6) = C4 S5
Jac3(5,6) = S4 S5
Jac3(6,6) = C5

Jac3(1,7) = 0
Jac3(2,7) = 0
Jac3(3,7) = 0
Jac3(4,7) = C6 S4 + C4 C5 S6
Jac3(5,7) = -(C4 C6) + C5 S4 S6
Jac3(6,7) = -(S5 S6)
```

- **The end-effector velocity is transformed to being in base coordinates to be written with respect to the wrist coordinate frame by premultiplying the velocity vector with the following rotation matrix,**

```
MatrixForm[RedAngle[Rot[LTMTable,3,0]]]

C1 C2 C3 - S1 S3      C2 C3 S1 + C1 S3      -(C3 S2)

C1 S2                 S1 S2                 C2

C3 S1 + C1 C2 S3      -(C1 C3) + C2 S1 S3   -(S2 S3)
```

- Contruct a not singular jacobian J* from any six independent columns of the Jacobian. Dropping the second column the following Jacobian is obtained

```
Jstar = Table[ {J3[[j,2]],J3[[j,3]],J3[[j,4]],
                J3[[j,5]],J3[[j,6]],J3[[j,7]]}, {j,1,6}];
```

```
RedAngle[Jstar]

{{C3 S4 a4, 0, -(S4 a4), 0, 0, 0},

  {-a2 - C3 C4 a4, 0, C4 a4, 0, 0, 0},

  {S3 S4 a4, -(C4 a4), 0, 0, 0, 0},

  {S3, 0, 0, -S4, C4 S5, C6 S4 + C4 C5 S6},

  {0, 1, 0, C4, S4 S5, -(C4 C6) + C5 S4 S6},

  {-C3, 0, 1, 0, C5, -(S5 S6)}}
```

- Following the algorithm, the Jacobian can be decomposed in two matrices using the first three and last three rows.

```
J1star = Table[ Jstar[[i,j]] ,{i,1,3} ,{j,1,3}];
J2star = Table[ Jstar[[i,j]] ,{i,4,6} ,{j,1,3}];
J3star = Table[ Jstar[[i,j]] ,{i,4,6} ,{j,4,6}];
```

■ **The solution for the first three joint rates can be obtained from: J1star {θ2, θ3, θ4} = {x1, x2, x3}.**

Furthermore, the solution using Mathematica is obtained by: {θ2, θ3, θ4} = Inverse[J1star] {x1, x2, x3}.

```
ListOutput[J1star]

List(1,1) = C3 S4 a4
List(2,1) = -a2 - C3 C4 a4
List(3,1) = S3 S4 a4

List(1,2) = 0
List(2,2) = 0
List(3,2) = -(C4 a4)

List(1,3) = -(S4 a4)
List(2,3) = C4 a4
List(3,3) = 0


J1starT = J1star . {t2,t3,t4};
ListOutput[J1starT, "x"]

x(1) = C3 S4 a4 t2 - S4 a4 t4
x(2) = (-a2 - C3 C4 a4) t2 + C4 a4 t4
x(3) = S3 S4 a4 t2 - C4 a4 t3

Sol1 = Solve[{J1starT[[1]] == x1, J1starT[[2]] == x2}, {t2,t4}];
Sol2 = Solve[ J1starT[[3]] == x3,        {t3}];

tt2 = RedAngle[Together[t2 //. Sol1 [[1]] ]]

-(C4 x1) - S4 x2
----------------
     S4 a2

tt4  = RedAngle[Together[t4 //. Sol1 [[1]] ]]

-(a2 x1) - C3 C4 a4 x1 - C3 S4 a4 x2
------------------------------------
               S4 a2 a4

tt3  = RedAngle[ t3 //. Sol2 [[1]] ]

S3 S4 a4 t2 - x3
----------------
     C4 a4
```

□ Reducing terms

```
t2 == tt2

      -(C4 x1) - S4 x2
t2 == ----------------
            S4 a2

t4 == Collect[Numerator[tt4],{x1,x2}] / Denominator[tt4]

      (-a2 - C3 C4 a4) x1 - C3 S4 a4 x2
t4 == ---------------------------------
                   S4 a2 a4

t3 == tt3

      S3 S4 a4 t2 - x3
t3 == ----------------
            C4 a4
```

■ The solution for the last three joint rates can be obtained from:

J2star {θ1, θ3, θ4} + J3star {θ5, θ6, θ7}= {x4, x5, x6},

Furthermore, the solution using Mathematica is obtained by:

{θ1, θ3, θ4} = Inverse[J1star] {x1, x2, x3},

{θ5, θ6, θ7} = Inverse[J3star] ({x4, x5, x6} - J2star {x1, x2, x3}).

```
Deter = RedAngle[RedTrig[Det[J3star]]]

S6

inv = RedTrig[Inverse[J3star] Det[J3star]];
MatrixForm[RedAngle[inv]]

C4 C5 C6 - S4 S6  C5 C6 S4 + C4 S6  -(C6 S5)

C4 S5 S6          S4 S5 S6          C5 S6

C4 C5             C5 S4             -S5


Jsol = {x4,x5,x6} - J2star . {t2,t3,t4};
RedAngle[Jsol]

{-(S3 t2) + x4, -t3 + x5, C3 t2 - t4 + x6}
```

SOLUTION

```
       -(C4 x1) - S4 x2
t2 == ----------------
             S4 a2

       S3 S4 a4 t2 - x3
t3 == ----------------
             C4 a4

       (-a2 - C3 C4 a4) x1 - C3 S4 a4 x2
t4 == ---------------------------------
                     S4 a2 a4

(t5, t6, t7) == 1/S6 inv Jsol

 inv(1,1) = C4 C5 C6 - S4 S6
 inv(2,1) = C4 S5 S6
 inv(3,1) = C4 C5

 inv(1,2) = C5 C6 S4 + C4 S6
 inv(2,2) = S4 S5 S6
 inv(3,2) = C5 S4

 inv(1,3) = -(C6 S5)
 inv(2,3) = C5 S6
 inv(3,3) = -S5


 Jsol(1) = -(S3 t2) + x4
 Jsol(2) = -t3 + x5
 Jsol(3) = C3 t2 - t4 + x6
```

■ LTM gravitational compensation model

□ The mass table is needed to obtain the effect of the gravitation over the joints of the LTM. Note that even though links 1, 3, and 7 have no mass, rows of zeros are added at their positions.

```
LTMMass := {{ 0,   0 ,   0 ,   0 },
            {m2, xm2, ym2, zm2},
            { 0,   0 ,   0 ,   0 },
            {m4, xm4, ym4, zm4},
            { 0,   0 ,   0 ,   0 },
            {m6, xm6, ym6, zm6},
            { 0,   0 ,   0 ,   0 }}
```

□ A simple command is enough to find the gravitational compensation model:

```
LTMGrav = Gravitation[LTMTable, LTMMass];
```

□ Using the Function ListOutput and collecting terms, the model is obtained in FORTRAN compatible form:

```
M(1) = -(-(S2*(C3*(C4*(-(C5*G*S6*m6*xm6*
                         (C5*(C1*C3 - C2*S1*S3) -
                         S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
                         G*S5*m6*xm6*
                         (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                         S6*(S5*(C1*C3 - C2*S1*S3) +
                         C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) -
                         S4*(G*m4*xm4*(C1*C3 - C2*S1*S3) +
                         C6*G*m6*xm6*
                         (C5*(C1*C3 - C2*S1*S3) -
                         S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) -
                         a4*(-(C5*G*m6*
                         (C5*(C1*C3 - C2*S1*S3) -
                         S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) -
                         S5*(-(G*S6*m6*
                         (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                         S6*(S5*(C1*C3 - C2*S1*S3) +
                         C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
                         C6*G*m6*
                         (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                         C6*(S5*(C1*C3 - C2*S1*S3) +
                         C5*(S1*S2*S4 +
                             C4*(C2*C3*S1 + C1*S3)))))))) +
                         S3*(G*m4*xm4*
                         (C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
```

```
        G*S5*S6*m6*xm6*
        (C5*(C1*C3 - C2*S1*S3) -
        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) +
        C5*G*m6*xm6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
        a4*(C6*G*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
        G*S6*m6*
        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 +
            C4*(C2*C3*S1 + C1*S3))))))) -
C2*(-(C1*G*m2*xm2) -
        S4*(-(C5*G*S6*m6*xm6*
        (C5*(C1*C3 - C2*S1*S3) -
        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
        G*S5*m6*xm6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) -
        a2*(S3*
        (-(S4*
        (G*m4*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*G*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
        G*S6*m6*
        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 +
            C4*(C2*C3*S1 + C1*S3))))) +
        C4*(G*m4*(S1*S2*S4 +
                  C4*(C2*C3*S1 + C1*S3)) -
        G*S5*m6*
        (C5*(C1*C3 - C2*S1*S3) -
        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) +
        C5*(-(G*S6*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
        C6*G*m6*
```

```
                                (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                                C6*(S5*(C1*C3 - C2*S1*S3) +
                                C5*(S1*S2*S4 +
                                    C4*(C2*C3*S1 + C1*S3))))))) -
                                C3*(-(G*m4*(C1*C3 - C2*S1*S3)) -
                                C5*G*m6*
                                (C5*(C1*C3 - C2*S1*S3) -
                                S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) -
                                S5*(-(G*S6*m6*
                                (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                                S6*(S5*(C1*C3 - C2*S1*S3) +
                                C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
                                C6*G*m6*
                                (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                                C6*(S5*(C1*C3 - C2*S1*S3) +
                                C5*(S1*S2*S4 +
                                    C4*(C2*C3*S1 + C1*S3))))))) -
                                C4*(G*m4*xm4*(C1*C3 - C2*S1*S3) +
                                C6*G*m6*xm6*
                                (C5*(C1*C3 - C2*S1*S3) -
                                S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) -
                                a4*(-(C5*G*m6*
                                (C5*(C1*C3 - C2*S1*S3) -
                                S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) -
                                S5*(-(G*S6*m6*
                                (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                                S6*(S5*(C1*C3 - C2*S1*S3) +
                                C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
                                C6*G*m6*
                                (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                                C6*(S5*(C1*C3 - C2*S1*S3) +
                                C5*(S1*S2*S4 +
                                    C4*(C2*C3*S1 + C1*S3)))))))));
M(2) = -(-(G*S1*S2*m2*xm2) +
                        S3*(C4*(-(C5*G*S6*m6*xm6*
                                (C5*(C1*C3 - C2*S1*S3) -
                                S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
                                G*S5*m6*xm6*
                                (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                                S6*(S5*(C1*C3 - C2*S1*S3) +
                                C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) -
                                S4*(G*m4*xm4*(C1*C3 - C2*S1*S3) +
                                C6*G*m6*xm6*
                                (C5*(C1*C3 - C2*S1*S3) -
                                S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) -
                                a4*(-(C5*G*m6*
```

```
        (C5*(C1*C3 - C2*S1*S3) -
        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) -
        S5*(-(G*S6*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
        C6*G*m6*
        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 +
        C4*(C2*C3*S1 + C1*S3)))))))) -
a2*(C4*(G*m4*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*G*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
        G*S6*m6*
        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
        S4*(G*m4*(S1*S2*S4 +
                  C4*(C2*C3*S1 + C1*S3)) -
        G*S5*m6*
        (C5*(C1*C3 - C2*S1*S3) -
        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) +
        C5*(-(G*S6*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
        C6*G*m6*
        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        C6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 +
        C4*(C2*C3*S1 + C1*S3)))))))) -
C3*(G*m4*xm4*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
        G*S5*S6*m6*xm6*
        (C5*(C1*C3 - C2*S1*S3) -
        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) +
        C5*G*m6*xm6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
        a4*(C6*G*m6*
        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
        S6*(S5*(C1*C3 - C2*S1*S3) +
        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
```

```
                        G*S6*m6*
                        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                        C6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 +
                        C4*(C2*C3*S1 + C1*S3)))))));
M(3) = -(S4*(-(C5*G*S6*m6*xm6*
                        (C5*(C1*C3 - C2*S1*S3) -
                        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
                        G*S5*m6*xm6*
                        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                        S6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
                C4*(G*m4*xm4*(C1*C3 - C2*S1*S3) +
                        C6*G*m6*xm6*
                        (C5*(C1*C3 - C2*S1*S3) -
                        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) -
                        a4*(-(C5*G*m6*
                        (C5*(C1*C3 - C2*S1*S3) -
                        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) -
                        S5*(-(G*S6*m6*
                        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                        S6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))) +
                        C6*G*m6*
                        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                        C6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 +
                        C4*(C2*C3*S1 + C1*S3))))))));
M(4) = -(G*m4*xm4*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                G*S5*S6*m6*xm6*
                    (C5*(C1*C3 - C2*S1*S3) -
                        S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))) +
                C5*G*m6*xm6*
                    (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                        S6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
                a4*(C6*G*m6*
                        (C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                        S6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))) +
                        G*S6*m6*
                        (S6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) +
                        C6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3))))));
M(5) = -(C6*G*m6*xm6*(C5*(C1*C3 - C2*S1*S3) -
                    S5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))),

M(6) = -(G*m6*xm6*(C6*(C4*S1*S2 - S4*(C2*C3*S1 + C1*S3)) -
                    S6*(S5*(C1*C3 - C2*S1*S3) +
                        C5*(S1*S2*S4 + C4*(C2*C3*S1 + C1*S3)))));
M(7) = 0;
```

INTERNAL DISTRIBUTION

1. G. A. Armstrong
2-3. S. M. Babcock
4. B. L. Burks
5. P. L. Butler
6. H. M. Costello
7-8. R. M. Davis
9. R. V. Dubey
10. R. L. Glassell
11. W. H. Glover
12. M. J. Haire
13. D. C. Haley
14. W. R. Hamel
15. J. H. Hannah
16. J. N. Herndon
17-18. J. F. Jansen
19-20. R. L. Kress
21. C. T. Kring
22. D. S. Kwon
23. P. D. Lloyd
24. R. C. Mann
25. C. March-Leuba
26. J. March-Leuba
27-36. S. March-Leuba
37-39. S. A. Meacham
40. R. P. Oliver
41. F. G. Pin
42. K. E. Plummer
43. T. L. Ray
44. B. S. Richardson
45. J. C. Rowe
46. S. L. Schrock
47. K. U. Vandergriff
48. ORNL Central Research Library
49. ORNL Document Reference Section
50. ORNL Laboratory Records
51. ORNL Laboratory Records RC
52. ORNL Patent Section
53-54. RPSD Publications Office

EXTERNAL DISTRIBUTION

55. Clinton Bastin, Manager, LMR Reprocessing Projects, Division of Fuels and Reprocessing, Office of Facilities, Fuel Cycle, and Test Programs, NE-471, Department of Energy, Washington, DC 20545.

56. L.F. Blanker, Fusion and Nuclear Technology Branch, Energy Programs Division, Department of Energy, X-10 Site, P.O. Box 2008, Oak Ridge, Tennessee 37831-6269.

57. Fred Eldredge, Ammunition Equipment Director, Tooele Army Depot, Building 1005, Tooele, Utah 84074.

58. Office of Assistant Manager for Energy Research and Development, DOE Oak Ridge Field Office, P.O. Box 2008, Oak Ridge, Tennessee 37831-6269.

59-68. Office of Scientific and Technical Information, DOE Oak Ridge Field Office, P.O. Box 62, Oak Ridge, Tennessee 37831.

69-70. Nelson Gravenstede, FARS Project Director (PM-AMMOLOG), HQ ARDC, AMLPM-AL Picatinny Arsenal, Dover, New Jersey 07801-5001.

71-72. Gary Kent, Deputy Director (PM-AMMOLOG), HQ ARDC, AMLPM-AL Picatinny Arsenal, Dover, New Jersey 07801-5001.

73. José March España, Albacete 3 - 7, Valencia 46007, Spain.

74. Vicente Mata Amela, Universidad Politécnica de Valencia, Departamento de Ingeniería Mecánica y de Materiales, Camino Vera sn, Valencia 46022, Spain.

75. Angel Perez Navarro, CIEMAT, Avenida Complutense 22, Madrid 28040, Spain.

76. Rafael Perez, The University of Tennessee, Department of Nuclear Engineering, 315 Pasqua Building, Knoxville, Tennessee 37996.

77. Col. Thomas Tobin, Project Manager - Ammunition Logistics, HQ ARDC, AMLPM-AL Picatinny Arsenal, Dover, New Jersey 07801-5001.

78. Mohan Trivedi, The University of Tennessee, Ferris Hall, Knoxville, Tennessee 37996-2100.