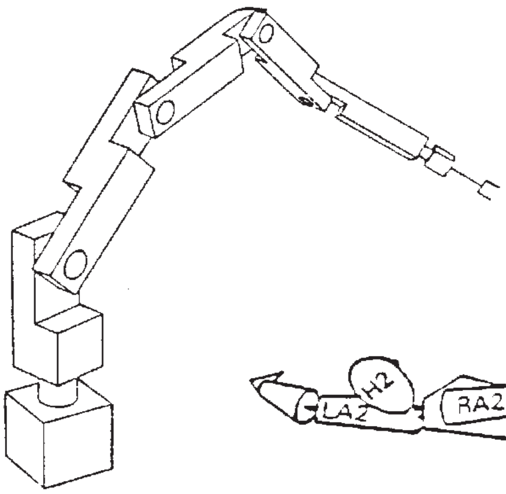
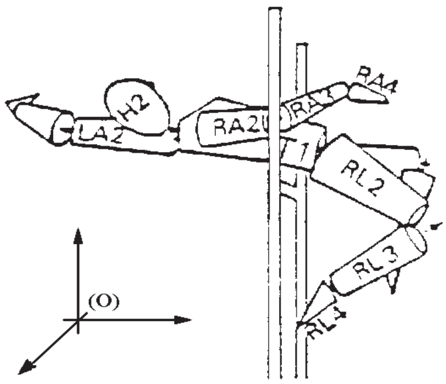
SpaceLib© in MATLAB©

Version 2.2 - November 2005

A software library for
the kinematic and dynamic analysis
of systems of rigid bodies.
Includes general functions for vectors, matrices,
kinematics, dynamics, Euler angles and linear systems

USER'S MANUAL





By G. Legnani, R. Adamini and B. Zappa

Università di Brescia - Dip. Ing. Meccanica
Via Branze 38, 25123 BRESCIA, Italy
Tel. +39/030 3715425 Fax +39/030 3702448

e-mail: giovanni.legnani@ing.unibs.it
http://bsing.ing.unibs.it/~glegnani
http://robotics.ing.unibs.it

With the cooperation of C. Moiola and D. Manara

Typeset in LaTeX

SpaceLib© and its authors.

The first functions contained in SpaceLib© were written in C language by myself in 1988 when a friend of mine working with the *European Space Agency* asked for help. I wrote the program described in § 7.5 of this manual. I realized that a few friends needed software to deal with 3D rototranslations and I started writing a "private" version of SpaceLib©.

After a while other people asked for help and in 1990 I wrote the first "public" version of SpaceLib© with the support of *R. Faglia*. The mathematical bases of SpaceLib© grown and new functions were realized to deal with velocities, accelerations, forces, torques, momentum and angular momentum.

A second public version of SpaceLib© was then realized in 1993 with the help of *R. Adamini* and several dozens of copies have been distributed through the world. People have been using it both for *Robotic* and *Biomechanics* applications. I have used it for my research and lectures and the students have shown a great interest in it.

In 1997, a new version of the library in C language has been realized under my supervision by *D. Amadori*, *P. Ghislotti* and *G. Pugliese*. I made the final refinements with a strong support by *B. Zappa*; *R. Adamini* gave a good theoretical and technical support. This version contains additional functions and an extended documentation.

Many people asked for a new version of SpaceLib© in MATLAB ©[1]. The bases of the MATLAB © version of SpaceLib© have been realized by *C. Moiola* under my supervision. I made a final "strong" correction and I also performed some patches in 2001, 2003 and 2004. In this occasion a new version of the manual have been realized with the help of *D. Tosi* and *M. Camposaragna*.

Time passed and the need for a new version of SpaceLib© which made possible the writing of symbolic equations grown. This stimulated the realization of the SpaceLib© in Maple 9 ©[2]. The basis of this version were put by *F. Bignamini* and *N. Serana* under my supervision. The final version of the library were made by *D. Manara* under my supervision with the cooperation of *A. Rodenghi*.

Between the end of 2004 and the beginning of 2005, *D. Manara* put a great effort in revising all the manuals of three SpaceLib© versions (C, MATLAB ©, Maple 9 ©). This was an occasion to perform small revisions to the three versions. A great effort was put in maintaining aligned the three releases.

This version of SpaceLib© will be probably upgraded in future.

We look forward for comments, suggestions, bugs report and copies of papers related with the use of SpaceLib©. Send them to *G. Legnani* (address on cover page).

Brescia, Italy; November 2005

Giovanni Legnani

[1] MATLAB © is a registered trademark of MathWorks (http://www.mathworks.com) inc.

[2] Maple 9 © is a registered trademark of Maplesoft (http://www.maplesoft.com), a division of Waterloo Maple inc.

Contents

Chapter 1

Introduction

1.1 What is SpaceLib©

`SpaceLib`© is a software library useful for the realization of programs for the kinematic and dynamic analysis of systems of rigid bodies. This library is currently used in *Robotics* and *Biomechanics*. It has been developed at the Mechanical Engineering Department of the University of Brescia.

The library is intended as an aid in writing programs for the analysis of mechanical systems following a particular methodology based on $4{\times}4$ matrices show in [2], [3] and [4][1]. This approach can be considered a powerful generalization of the Transformation Matrix Approach proposed by *Denavit* and *Hartenberg* [1].

The main feature of this methodology is that it allows the development of the analysis of systems of rigid bodies in a systematic way simplifying the symbolic manipulation of equations as well as the realization of efficient numerical programs.

Three versions of the library are presently available, two for numerical simulations in `C` and `MATLAB` © languages, and one version for the symbolic computation in `Maple 9`©. The `MATLAB`© version is useful for a fast development of numeric programs. The `C` version is preferable to obtain fast high-efficient numeric simulations. The `Maple 9`© version is useful when symbolic manipulation is essential, however it also make possible the development of numeric programs.

Particular effort has been posed in order to keep "aligned" the different versions. Functions with the same name in the three versions produces essentially the same results. However intrinsic differences between the languages result in few difformities between the different implementations (see § A, page 119).

All the distributions contain the software source code, and if one likes, he can analyzes it to better understand its use.

1.2 About this manual

This USER'S MANUAL has been written assuming that the reader knows both the `MATLAB`© language and the theory on which `SpaceLib`© is based. The latter subject is widely described in the *references* (page 117). Since `SpaceLib`© has been developed by successive steps, some details contained in the references can differ from pieces of information here contained. In this case, please refer to this manual.

This manual contains:

- introduction;
- general information on the use of `SpaceLib`©;
- a commented directory of the library;
- sample programs;
- a Reference list of papers which describes the mathematical bases of `SpaceLib`©.

[1] copy of [3] and [4] is also included into the distribution file

1.3 Technical information

The first version of SpaceLib was developed using MATLAB 4.0 version, and tested on PC under Windows 9x© and Windows NT© operative systems. The name of the functions are generally no longer than 8 characters.

The library has been patched to work also with MATLAB release 12 (version 6) and tested on Windows 2000©, Windows me©, . . . (see also §2.8).

1.4 Authors' notes and disclaimer warranties

The authors know that the present version of SpaceLib© should be possibly improved in the future. The code or the documentation could contain errors or the documentation could have lacks in some parts. The library and the related information is provided "as is" without warranty of any kind. The authors disclaim all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall the authors or their institution or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if the authors or their suppliers have been advised of the possibility of such damages.
The authors stimulate any suggestion and error reporting by the users.

Chapter 2

Writing programs with SpaceLib

2.1 General information - Read me first

Information contained in this section refers to `MS-Windows` system. The source code is compatible with other operative systems with the possible exception of the directories names. `SpaceLib`©consists of about 100 source `M-files` that should be copied in the directory `x:\...\spacelib`, where `x:\...`is any valid directory (drive and path). For compatibility with windows 3.x© environment, each `M-file` (function or program) consists in one file whose name consist of a maximum of 8 characters. Three subdirectories should be built to keep separated the functions and the demo programs.

The subdirectory `x:\...\spacelib\function` is intended to contains the functions, the directory `x:\...\spacelib\bigexa` to contains the sample programs listed in § 6. Other "short examples" are available in `x:\...\spacelib\shortexa`. The "startup file" `spacelib.m` and the header file `spheader.m` should be placed in `x:\...\spacelib`.

To install `SpaceLib` : create the directory and unzip the distribution file `spclib_m.zip` with the following option to recreate the subdirectories:

```
x:\...\spacelib>pkunzip -d Spclib_m.zip
```

It is also necessary to modify one line of the file `spacelib.m` to update the value of a variable containing the `SpaceLib`©path name. This modification is necessary to update the `MATLAB` ©search path. The line must be modified as follows:

```
spc_lib_dir='x:\ ...\spacelib' % spacelib directory
```

For example, assuming that `SpaceLib`©is installed in the 'standard' directory `c:\programs\matlab`, the line should read:

```
spc_lib_dir='c:\programs\matlab\spacelib' % spacelib directory
```

Users utilizing operative systems different from `MS-Windows`©, should take care of the different name conventions and they should also check the lines immediately following the first one where the three subdirectories are defined. An example follows:

```
%_____________________________________________________________________________
%
%                  GLOBAL DIRECTORIES DECLARATION:
%_____________________________________________________________________________
%
% ***-----> the following line MUST be updated to match your installation!!!}

spc_lib_dir='c:\programmi\matlab\spacelib'            % spacelib directory

spc_lib_dir_f=[spc_lib_dir, '\function']              % functions

spc_lib_dir_s=[spc_lib_dir, '\shortexa']              % short examples

spc_lib_dir_b=[spc_lib_dir, '\bigexa']                % big examples
```

In order to initialize the global constants and variables defined by SpaceLib©, the following lines must be added at the end of the file matlabrc.m which is located in the MATLAB ©home directory

```
% Load Spacelib variables and constants
cd x:\...\spacelib
spacelib
```

As an alternative it is possible to create a file called startup.m to be placed in a directory searched by MATLAB ©(see the MATLAB ©manual). The file must contains the two following lines

```
addpath c:\\users\spacelib_m
spacelib
```

As a further alternative, the user can initialize "by hand" the SpaceLib©environment by typing the two previous lines at the MATLAB ©command window.

All the SpaceLib©functions and sample programs (M-Files) must be placed in the indicated directories, so that MATLAB ©could find them. All the functions or M-files that uses the constants defined in the header file spacelib.m must call in the first line of the program the header file spheader.m.

Example:

```
function FI=actom(fx,fy,fz,cx,cy,cz)

% ACTOM (Spacelib):  Actions to matrix.
%
% Builds the action matrix FI  from the components of the forces fx, fy, fz
% and the torque (or couples) cx, cy, cz.
% Usage:
%
%           FI=actom(fx,fy,fz,cx,cy,cz)
%
% (c) G.Legnani, C. Moiola 1998; adapted from: G.Legnani and R.Faglia 1990
%_______________________________________________________________________

spheader
FI(X,X)=0;  FI(X,Y)= -cz;   FI(X,Z)= cy;    FI(X,U)= fx;
FI(Y,X)= cz;    FI(Y,Y)=0;  FI(Y,Z)= -cx;   FI(Y,U)= fy;
FI(Z,X)= -cy;   FI(Z,Y)= cx;    FI(Z,Z)=0;  FI(Z,U)= fz;
FI(U,X)= -fx;   FI(U,Y)= -fy;   FI(U,Z)= -fz;   FI(U,U)=0;
```

The header file contains only the global variables declarations. This file is listed below:

```
global X Y Z U Xaxis Yaxis Zaxis ORIGIN Rev Pri Tor For SYMM_ SKEW_
OK NOTOK global Xaxis_n Yaxis_n Zaxis_n Row Col NULL3 NULL4 UNIT3
UNIT4 global spc_lib_dir spc_lib_dir_f spc_lib_dir_b spc_lib_dir_s
global PIG PIG2 PIG_2
```

Warning:Although we consider this entities as constants, they are global variables and so all the names defined in the header file must not be used to indicate a new variable and must not be manipulated in any kind of operation (see also § 2.4.3).

Warning: MATLAB are case sensitive, for example U and u are not the same variables. In the early versions of MATLAB ©(e.g. version 4) it was possible **but not recommended** to deactivate this property using the command casesen off; SpaceLib©users should avoid this practice!

Both the M-files and the function files contain useful comments about the routines and the types. The first comment lines are available as *on-line help* and the first one is used also by the lookfor command (see § 2.1.1).
Before using SpaceLib©, users should have a look, at least, at the startup and at the header files.

All the previous subjects are detailed in the following paragraphs.

2.1.1 The MATLAB on line help

Online help, which describes the SpaceLib©functions can be obtained by typing at the MATLAB© prompt the command **help** followed by the function name. For example, the statement

```
help actom
```

has the effect to display:

```
ACTOM (Spacelib): Actions to matrix.

Builds the action matrix FI from the components of the forces fx, fy,  fz
and the torque (or couples) cx,  cy,  cz.}

Usage: FI=actom(fx, fy, fz, cx, cy, cz)

(c) G. Legnani 1998 adapted from G.Legnani and R.Faglia 1990
```

Functions can also be searched using the MATLAB©lookfor command. Note that the functions are searched only in the MATLAB©search path. For example the command:

```
lookfor cardan
```

produces an output like this:

```
CARDATOH (Spacelib): Cardan angles to acceleration matrix.
CARDATOL (Spacelib): Cardan angles to L matrix.
CARDATOM (Spacelib): Cardan angles to position matrix.
CARDATOR (Spacelib): Cardan (or Euler) angles to rotation matrix.
CARDATOW (Spacelib): Cardan angles to velocity matrix.
CARDOMPT (Spacelib): Cardan angles to angular acceleration.
CARDTOME (Spacelib): Cardan angles to velocity matrix.
CARDTOWP (Spacelib): Builds a matrix for cardan acceleration.
INVA (Spacelib): builds the inverse of a matrix A (Euler/Cardan velocity).
MTOCARDA (Spacelib): Position matrix to Cardan angles.
RTOCARDA (Spacelib): Rotation matrix to Cardan or Eulerian angles.
```

The command `lookfor spaceblib`, lists all the SpaceLib©function contained in the MATLAB©search path.

2.2 Notation

In this section are briefly described the notation used in the SpaceLib©. More information can be found in [2], [3] and [4].

2.2.1 Subscript conventions

The relative motions between bodies are represented by matrices which usually appear with some subscripts:

$$M_{i,j} \qquad W_{i,j(k)} \qquad H_{i,j(k)} \qquad L_{i,j(k)}$$

Subscripts i and j specifies the bodies involved, the subscript k, which is in round brackets, denotes the frame onto which the quantities are projected. For instance the velocity of the body (5) with respect to body (3) projected on frame (2) is:

$$W_{3,5(2)}$$

In special cases when subscripts assume "standard" or "obvious values" some of them can be omitted to simplify the notation. This happens for example where the meaning of each matrix is presented. For the same reason the third subscript k is often omitted when $k = i$. The dynamics quantities J, Γ and Φ require just two subscripts:

$$J_{i(k)} \qquad \Gamma_{i(k)} \qquad \Phi_{i(k)}$$

The subscript i denotes the body involved, and (k) has the previous meaning (frame on which the quantities are projected). Frame (0) is the absolute reference frame; in dynamics it is assumed to be also the inertial frame.

Special Matrices Names	***Use***
Φ, `PHI`	Action matrix
`G`	3×3 upper-left submatrix of H matrix
`H`, `Hx`, `Wp`	Acceleration matrix(1)
`Hg`	Gravity acceleration matrix
`J`	Inertia matrix
`L`, `Lx`	L matrix(1)
`m`, `mx`	Position or transformation matrix (1)
`R`, `Rx`	Rotation matrix(1)
`W`, `Wx`	Velocity matrix(1)
Special Points names	
`O`	Frame origin
Axes names	
`X`, `Y`, `Z`, `U`, `a`	Rototranslations axis, Axis of rotation (`a`=`X`, `Y` or `Z`)
Scalar Parameters names	
`i`, `j`, `k`	Parameters related to operation dealing with the `x`, `y`, `z` axes
`q`, `qx`, `qp`, `qpp`	joint variables, first derivative, second derivative(1)
Generic elements names	
`A`, `B`, `C`, `Ax`, `Mx`, `mx`	Matrix name(1)
`Pl`	plane name
`v`, `vx`	Vector name(1)
`L`, `Lx`	line name(1)
`dim`, `n`	Matrix dimension
`P`, `Px`	Point name(1)

Table 2.1: Naming convention for SpaceLib© parameters

2.2.2 Naming convection for parameters

In describing SpaceLib© functions the authors will make use of *matrices*, *vectors*, *axes*, *frames*, *planes*, *line*, *points* and *constants*. *Matrices* and *points* are generally denoted by upper case characters while *vectors*, *axes* and scalar parameters (i.e. "phi", "alpha", "dim") are denoted by lowercase letters. However there are a few exceptions. In order to make more comprehensible the type of the input and output variables, in the calling list, strings are added to indicate the type of them. Note that this is not a declaration but only a help to make more clear the usage of the functions. As an example, function 'extract' is described in the

```
[COL3 u, real fi] = extract(MAT A)
```

and should be used as:

```
[u, fi]=extract(A)
```

The string `COL3`, `real` and `MAT` remember you the type of the parameters. The complete list of types are:

`MAT`	matrix with non predefined dimensions
`MAT3`	3×3 matrix (usually is a rotation matrix)
`MAT4`	4×4 matrix (usually is a position, velocity or acceleration matrix)
`POINT`	4×1 vector (column vector indicating a point in homogeneous coordinates)
`COL3`	3×1 vector (column vector)
`ROW3`	1×3 vector (row vector)
`PLANE`	1×4 vector defining a plane (see § 2.4)
`LINE`	4×2 vector defining a line (see § 2.4)

`real` scalar variable
`int` an integer value

In the user manual the names of the parameters are generally given according to the convention described in table 2.1.

NOTE (1) Refereing to table 2.1, the ‘`x`’ character following a variable name is generally substituted by a digit. It is useful in order to specify two or more variables of the same type in a function prototype (i.e. `m1` and `m2` or `R1` and `R2`).

2.2.3 Units

Although sometimes different set of congruent units can be used, users are suggested to utilize always the *International Units System* (see table 2.2). Angles must be expressed in radians.

`SpaceLib`© **units Table**		
Length	m	meter
Time	s	second
Force	N	newton
Torque	N m	newton · meter
Mass	kg	kilogram
Angle	rad	radian

Table 2.2: International Units System

2.3 Math functions

In `MATLAB`©many mathematical functions are defined. For a complete list, see the `MATLAB`©user’s manual. The more common used together with `SpaceLib`©are listed below.

- `abs(x)` absolute value of x
- `max(a, b)` the maximum between a and b
- `min(a, b)` the minimum between a and b
- `sign(x)` the sign of x which is defined as $\begin{cases} -1 & & x<0 \\ 0 & if & x=0 \\ 1 & & x>0 \end{cases}$
- `det(A)` evaluates the determinant of a square matrix A.
- `inv(M)` evaluates the inverse matrix M^{-1}
- `pinv(M)` evaluates the pseudoinverse matrix.
- `norm(v)` evaluates the norm of a matrix or a vector.
- `rank(A)` evaluates the rank of the matrix A.

2.4 Variables declarations and types

2.4.1 Data types

In `MATLAB`©, each variable is considered as a matrix. Row vectors can be considered as matrix consisting just in one row, and column vectors can be considered like matrices of only one column. Each variable exists only after an initialization. Is not possible to declare variables without initialize them.

2.4.2 Geometrical elements

As better described in the references, in SpaceLib© some geometrical entities are used: *points*, *lines*, *vector*, *planes* and *frames*. A *point* is represented by its homogeneous coordinates x, y, z and u:

$$P = [x,\ y,\ z,\ u]^t$$

In MATLAB ©that is obtained by storing these coordinates into one 4×1 matrix:

```
P=[x; y; z; u]
```

A *line* is represented by one point and by its unit vector:

$$\begin{cases} x = x_p + \alpha \cdot t \\ y = y_p + \beta \cdot t \\ z = z_p + \gamma \cdot t \end{cases} \tag{2.1}$$

where $P = [x_p,\ y_p,\ z_p]^t$ is a point that lies on the line. The vector $[\alpha,\ \beta,\ \gamma]^t$ contains the director cosines which express the direction of the line in a reference frame ($\alpha^2 + \beta^2 + \gamma^2 = 1$); t is the abscissa. In MATLAB ©this is obtained by storing these two vectors into one matrix with four rows and two columns. In the first column is contained the vector that defines the point (in homogeneous coordinates). In the second column are stored the three director cosines:

$$l = \begin{bmatrix} x_p & \alpha \\ y_p & \beta \\ z_p & \gamma \\ 1 & 0 \end{bmatrix}$$

A *plane* is defined by the following equation:

$$a \cdot x + b \cdot y + c \cdot z + d = 0 \tag{2.2}$$

where a, b, c are the components in a reference frame of the unit vector orthogonal to the plane itself ($a^2 + b^2 + c^2 = 1$). The fourth element d expresses the distance with sign of the origin of the reference frame from the plane. A plane is store in a 4-element row vector:

```
pl = [ a b c d ]
```

A *frame* is represented by a 4×4 matrix containing the homogeneous coordinates of the frame axes and of the origin of the frame:

$$\left[\begin{array}{ccc|c} X_x & Y_x & Z_x & x \\ X_y & Y_y & Z_y & y \\ X_z & Y_z & Z_z & z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

2.4.3 Useful constants

Some useful constants have been defined in SpaceLib©. Although they should be considered as constants, for technical reasons in MATLAB © they have been implemented as global variables. Users must not use variables with the same name and should not modify their value. The following constants has been defined (see file spacelib.m):

`OK` = 1, `NOTOK` = 0 — Values returned by some SpaceLib© functions in order to specify the success or the failure of their operations. If a function returns NOTOK, it means that it could not perform the requested operation. In general, it happens if the function was called with non valid values for the input parameters.

`SYMM_ = 1`, `SKEW_ =-1` — Utilized by some functions in order to specify if a matrix is symmetric or skew-symmetric.

`Rev` = 0, `Pri` = 1 — Utilized to denote revolute or prismatic (sliding) pairs.

$\left.\begin{array}{r}\texttt{Tor} = 0\\ \texttt{For} = 1\end{array}\right\}$ Utilized to denote torques or forces.

$\left.\begin{array}{r}\texttt{Row} = 0\\ \texttt{Col} = 0\end{array}\right\}$ Utilized to denote rows and columns.

$\left.\begin{array}{r}\texttt{X=1}\\ \texttt{Y=2}\\ \texttt{Z=3}\\ \texttt{U=4}\end{array}\right\}$ Utilized to denote the four homogeneous coordinates of a point or the three components of a vector or axis. To remember the differences in the constants definition to identify the Cartesian axes between `SpaceLib`© in `C`, `MATLAB`© and `Maple 9`©, refer to the table 2.3.

	`C`	`MATLAB`©	`Maple 9`©
X	0	1	1
Y	1	2	2
Z	2	3	3
U	3	4	4

Table 2.3: Rotation axes naming convention

The following constants have been also defined in order to initialize, when applicable, matrices, points and vectors. They can be generally used only to initialize global or static arrays or matrices.

$\left.\begin{array}{r}\texttt{Xaxis} = [\,1\ 0\ 0\,]'\\ \texttt{Yaxis} = [\,0\ 1\ 0\,]'\\ \texttt{Zaxis} = [\,0\ 0\ 1\,]'\end{array}\right\}$ Applicable to variables of the type `AXIS` in order to set them equal to an axis coordinate.

$\left.\begin{array}{r}\texttt{Xaxis\_n} = [\,\text{-}1\ 0\ 0\,]'\\ \texttt{Yaxis\_n} = [\,0\ \text{-}1\ 0\,]'\\ \texttt{Zaxis\_n} = [\,0\ 0\ \text{-}1\,]'\end{array}\right\}$ Applicable to variables of the type `AXIS` in order to set them equal to a negative axis coordinate.

`ORIGIN = [0 0 0 1]'` } Applicable to a variable of the type `POINT` in order to set it equal to the origin of a frame.

$\left.\begin{array}{r}\texttt{NULL3}\\ \texttt{UNIT3}\\ \texttt{NULL4}\\ \texttt{UNIT4}\end{array}\right\}$ Applicable to 3×3 or 4×4 matrices in order to set them equal to the null or the identity matrix.

$\left.\begin{array}{r}\texttt{PIG\_2 = 1.57...}\\ \texttt{PIG = 3.14...}\\ \texttt{PIG2 = 6.28...}\end{array}\right\}$ Useful trigonometric constants $\pi/2$, π, 2π.

2.4.4 `MATLAB` built-in constants

In `MATLAB`© many constants are defined. Some that may have a significant interest for `SpaceLib`© users are listed in table 2.4. Other constants have been defined in `SpaceLib`© and are described in § 2.4.3. The constant `eps` is also called `zero machine` and it corresponds at the smaller number ε that makes true the relation:

$$1 + \varepsilon > 1$$

Constant	***Value***
`eps`	2.52 e -16
`pi`	3.1415...

Table 2.4: `MATLAB`©built-in constants

2.5 Arrays of matrices.

Sometimes could be useful to realize arrays of matrices (e.g. see the sample programs contained in § 7.1). In recent versions of MATLAB© this is possible using array of cells or multidimensional arrays. However this was not possible with older versions of MATLAB©. SpaceLib©, which was created many years ago, was designed in order to be compatible with new and old versions. For compatibility reasons the examples contained in this manual do not make use of "*cell array*" nor "*multidimensional array*".

In this section we describe how it is possible to simulate an array of matrices using a standard "*bidimensional array*".

We begin with an example. In recent versions of MATLAB© the 4×4 relative position matrices of a serial manipulator can be stored in a cell array named MM as follows

```
jtype=[......];                 % description of robot
theta=[......];
.....
Q=[.....];
for i=1:Nlink                   % put one 4*4 matrix in one element of the cell array
   MM{i}=dhtom(jtype(i),theta(i),d(i),0.,a(i),alpha(i),Q(i));
end
```

while in oldest MATLAB© versions the n 4×4 matrices must be put side by side in a 4×(4*Nlink) bidimensional matrix

```
jtype=[......];                 % description of robot
theta=[......];
.....
Q=[.....];
for i=1:Nlink
   MM(:,4*(i-1)+1:4*i)=dhtom(jtype(i),theta(i),d(i),0.,a(i),alpha(i),Q(i));
end
```

New users may positively profit by the use of cell arrays, while users with compatibility problems must use the other way. More details on this second possibility are reported in the following.

Generally, in our applications matrices should have 4 rows and 4 columns. To simulate matrix arrays (N matrices of 4×4 elements), we can build a matrix with 4 rows and $4 \cdot N$ columns, where N is the number of the array elements. To scan the matrix we use a 4×4 "window". For example, the following code is useful to realize an array with 3 square 4×4 matrices:

```
N=3;                            % Number of array elements
mat=zeros(4, 4*N)               % Initialize Matrix:
```

To select a particular matrix of the array, we could use the MATLAB operator ':' (that means all the rows) and a vector of four elements that defines which columns must be processed.

1 ... 4	5 ... 8			$4 \cdot i - 3$ ... 4i	
M1	M2		...	Mi	...

For example, to select the first matrix of the array use the following the code:

```
mat(:, 1:4)=screwtom(u, fi, P, h) % First matrix of the array
mat(:, 5:8)= ...                  % Second matrix of the array
mat(:, 4*j-3:4*j)= ...            % j-th matrix of the array
% And so on ...
```

In order to make more simple the code, we can do as follows:

```
i=[1:4];
mat(:, i)=screwtom(u, phi, P, h)  % First matrix of the array
```

To select the 'j^{th}' matrix of the array, we must increase the vector 'i' by 4 _not_ by 1:

```
i=[1:4];
mat(:, i+4*(j-1))= ...              % j-th matrix of the array
```

If necessary (see sample program `rob_mat`, § 7.1) we could automatize the vector index creation, useful in `for` loop, in the following mode:

```
for j=1:1:MAXLINK
i=[4*j-3: 4*j];                     % Vector index utilized to select the j-th matrix
mat(:, i)= ...
.
.
end
```

or:

```
i=[-3:0];
for j=1:1:MAXLINK
mat(:, i+4*j)= ...
.
.
end
```

2.6 Functions working on matrices with non predefined dimensions

Some `SpaceLib`© functions have been realized in order to handle matrices of not predefined dimensions. In this sense, `MATLAB`© is much powerful. It could manipulate matrices without predefined dimensions. For this reason, the number of the available functions are reduced with respect to the `C`language version of `SpaceLib`©. For example function `normskew` can be used to normalize a symmetric (or skew-symmetric) matrix of any dimension; e.g.:

```
M=normskew(M, SYMM_)
```

To normalize only the 3×3 upper left part of the matrix, the appropriate code using the same function is:

```
M(1:3, 1:3)=normskew(M(1:3, 1:3), SYMM_)
```

Opposite, in the `C` version of the `SpaceLib`©, the function which performs the same operation need to know the dimension of the matrix and many functions and macros are supplied to deal with all the common cases of 3×3 and 4×4 (sub)matrices: `norm_simm_skew, n_simm3, n_simm34, n_simm4, n_skew3, n_skew34, n_skew4`.

Clearly, even in `MATLAB`©, it is not possibly to performs some operations (sum, product, determinant) with matrices if their dimensions are not congruent.

2.7 Application Examples

Two groups of examples are supplied with `SpaceLib`©:

- Short examples
- Big examples

Short examples are described throughout the § 3 of the manual while big examples are described in depth in § 7.

obsolete name	*new name*
dot	dot3
dist	distp
mod	modulus
solve	solve_l

Table 2.5: function renamed

2.8 Patches

SpaceLib© was initially developed for MATLAB© version 4. Some patches have been performed for compatibility with successive releases of MATLAB©. These operations made necessary to renames the SpaceLib© functions reported in table 2.5. More over, to achieve compatibility with the new versions of MATLAB©, others minor changes had to be performed (changes between rows vectors from/to columns vectors, few operator precedences, ...).

Chapter 3

General function, Kinematics, Dynamics, Euler angles

In this section, the functions of the library are briefly described. The routines are divided in few groups. A mnemonic description of the functions is sometimes added. When in doubt, the type of the parameters of the procedures can be verified looking at the function source code contained in the library file `spacelib.m`. In translating `SpaceLib`©`.C` into `SpaceLib`©`.M` any attempt has been made to maintain a one to one correspondence between the two implementations (parameters, return values, source code), however in some cases it was not possible or convenient.

See the section § 2.2 and the following ones for notation.

3.1 Position, rotation and rototranslation matrices

dhtom

Denavit & Hartenberg parameters to matrix. (extended version)

Calling sequence:	`m = dhtom(jtype, theta, d, b, a, alpha, q)`
Return value:	`MAT4 - m`
Input parameters:	`int - jtype; real - theta, d, b, a, alpha, q`

Builds the position matrix **m** of a link from the extended *Denavit* and *Hartenberg*'s parameters [3], [4] **theta, d, b, a, alpha,** the value of the joint coordinate **q** and the type of the joint **jtype**. **jtype** is an integer whose value must be either `Rev` or `Pri` (§2.4.3). `Rev` and `Pri` are constants defined in the header file `spacelib.m` (§6.1). If the joint type is prismatic, the value of **q** is added to **d**, while for revolute joint **q** is added to **theta**. The matrix computed by the function is equivalent to the following rototranslation combination:

$$ROT(z,\theta)TRAS(z,d)TRAS(x,a)TRAS(y,b)ROT(x,\alpha)$$

If **b** is equal to 0 the extended D&H parameters coincide with the canonical ones [1].

Example: The position matrix `m` of frame (i) referred to frame $(i$-1) (see figure 3.1) is obtained by the following statement:

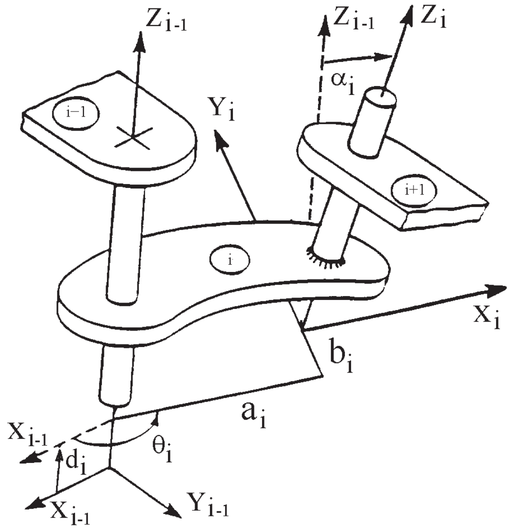
```
m = dhtom (Rev, theta, d, b, a, alpha, q);
```

The resulting matrix `m` is evaluated as:

$$m = \left[\begin{array}{ccc|c} \cos(\theta+q) & -\sin(\theta+q)\cos(\alpha) & \sin(\theta+q)\sin(\alpha) & a\cos(\theta+q)-b\sin(\theta+q) \\ \sin(\theta+q) & \cos(\theta+q)\cos(\alpha) & -\cos(\theta+q)\sin(\alpha) & a\sin(\theta+q)+b\cos(\theta+q) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.1}$$

See also example 3.1.

See also: `dhtomstd`, `rotat`, `screwtom`.



Extended Denavit and Hartenberg parameters	
θ	link rotation
d	link offset
b	shift (b=0 for standard definition)
a	link length
α	link twist

Figure 3.1: Definition of the *Denavit* and *Hartenberg*'s parameters.

dhtomstd

Denavit & Hartenberg parameters to matrix. (standard version)

Calling sequence:	`m = dhtomstd(theta, d, a, alpha)`
Return value:	`MAT4 - m`
Input parameters:	`real - theta, d, a, alpha`

Builds the position matrix **m** of a link from the standard *Denavit* and *Hartenberg*'s parameters [3], [4] **theta, d, a, alpha**. The matrix computed by the function is equivalent to the following rototranslation combination:

$$ROT(z,\theta)TRAS(z,d)TRAS(x,a)ROT(x,\alpha)$$

Example: The position matrix `m` of frame (i) referred to frame (i-1) (see figure 3.1) is obtained by the following statement:

```
m = dhtomstd (theta, d, a, alpha);
```

The resulting matrix `m` is evaluated as:

$$m = \left[\begin{array}{ccc|c} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.2}$$

See also example 3.1.

See also: `dhtom`, `rotat`, `screwtom`.

Example 3.1. *See sample program* `E_DHTOM.M`.

The following example shows the use of `dhtom` and `dhtomstd` for the direct kinematics of a serial manipulator. Numerical data refers to the *Stanford Arm* (see fig. 3.2 and table 3.1) which has one prismatic joint and five revolute ones. It is possible to see how the adoption of `dhtom` simplify the writing of the code.

```
clear; spacelib d2=0.2;
d3=0; % 3rd joint coord;

jtype=[Rev Rev Pri Rev Rev Rev]';       % joint type
alpha=[-pi/2 pi/2 0 -pi/2 pi/2 0]';     % --- Denavit and Hartenberg parameters
```

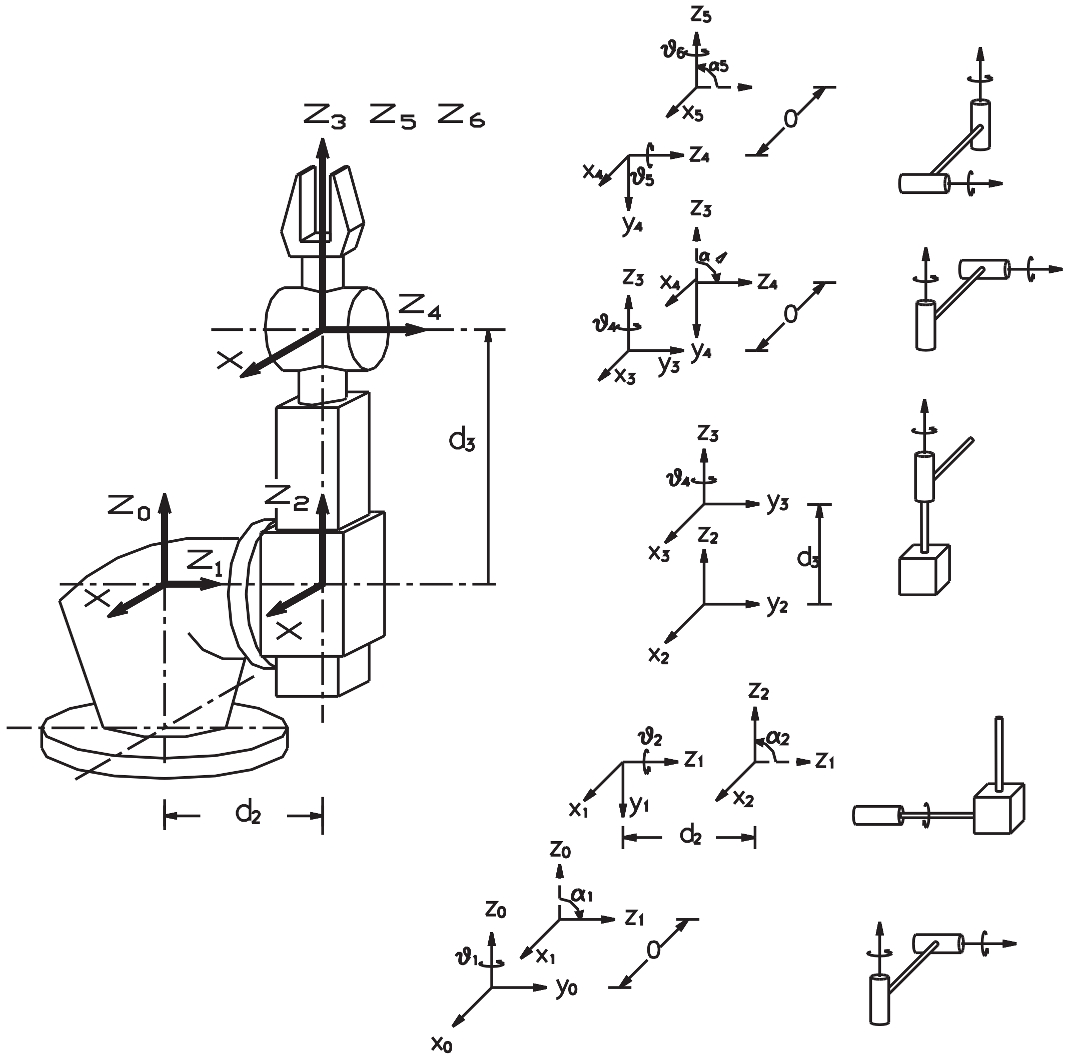


Figure 3.2: The *Stanford arm* with the *Denavit e Hantenberg* frames.

n.link	j.type	θ	d	a	α
1	R	q_1	0	0	$-\pi/2$
2	R	q_2	0.2	0	$\pi/2$
3	P	0	q_3	0	0
4	R	q_4	0	0	$-\pi/2$
5	R	q_5	0	0	$\pi/2$
6	R	q_6	0	0	0

Table 3.1: *Denavit e Hantenberg*'s parameters of *The Stanford arm* used in example 3.1.

```
a=[0 0 0 0 0 0]';                      % --- for the Stanford Arm
d=[0 d2 d3 0 0 0]';
theta=[0 0 0 0 0 0]';

Q=rand(6,1)                            % assign random value to the joint coordinate

                                       % --- direct kinematic using 'dhtomstd'
fprintf(1,'direct kinematic using ''dhtomstd''');
Ma=UNIT4;
for i=1:6
   if jtype(i)==Rev
       m=dhtomstd(Q(i),d(i),a(i),alpha(i));
   else
       m=dhtomstd(theta(i),Q(i),a(i),alpha(i));
   end
   Ma=Ma*m;
end
Ma
                                       % --- direct kinematic using 'dhtom'
fprintf(1,'direct kinematic using ''dhtom''');
Mb=UNIT4;
for i=1:6
    m=dhtom(jtype(i),theta(i),d(i),0.,a(i),alpha(i),Q(i));
    Mb=Mb*m;
end
Mb
                        % --- 'dhtomstd' and 'dhtom' must perform the same result
                        %     so Ma must be equal to Mb and so dM=0
fprintf(1,'''dhtomstd'' and ''dhtom'' must perform the same result');
fprintf(1,'so Ma must be equal to Mb and so dM=0')

dM=Ma-Mb
```

extract

Extracts unit vector of screw axis and rotation angle from rotation matrix.

Calling sequence:	`[u, phi] = extract(A)`
Return value:	`COL3 - u; real - phi.`
Input parameters:	`MAT - A,`

Extracts the unit vector **u** of the screw axis and the rotation angle **phi** from a rotation matrix stored in the upper-left 3×3 submatrix of a matrix **A**. `extract` performs the inverse operation than `rotat`.

Example: (see also example 3.2)

`[u, phi]=extract(R)`	extracts **u** and **phi** from a rotation matrix R.
`[u, phi]=extract(m(1:3, 1:3))`	extracts **u** and **phi** from the rotation sub-matrix of the position matrix m.

See also: `mtoscrew`, `screwtom`, `rotat`.

Example 3.2. *See sample program* `E_EXTRAC.M`.

This example shows the extraction of the screw parameters of the rototranslation, which superimposes frame (1) onto (2) (see figure 3.3). The results are computed in frame (0). The rototranslation is contained in matrix $Q_{1,2(0)}=M_{0,2}\ M_{1,0}$. The position matrix of frame (0) with respect to reference frame (1) and

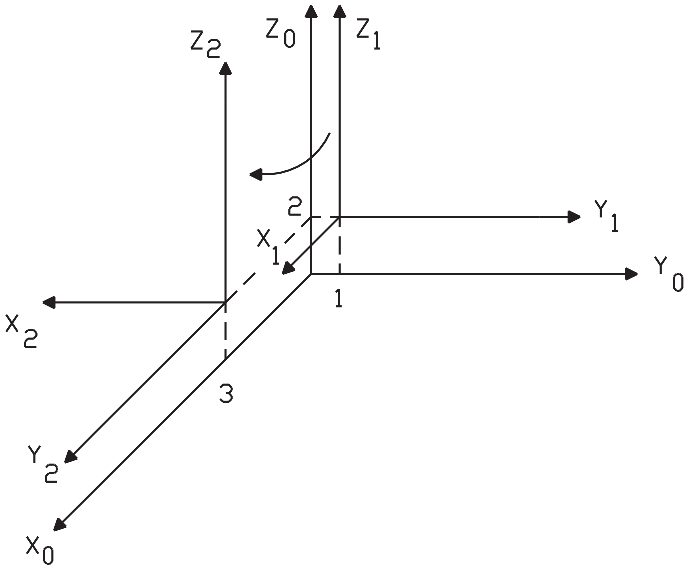


Figure 3.3: Rototranslation frame of example 3.2

the position matrix of frame (2) with respect to reference frame (0) are respectively

$$M_{1,0} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad M_{0,2} = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 3 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

and the rototranslation matrix is

$$Q_{1,2(0)} = M_{0,2}\ M_{1,0} = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 2 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The following statements

```
Q=[0 1 0 2; -1 0 0 0; 0 0 1 0; 0 0 0 1];
[u, phi]=extract(Q(1:3,  1:3));
```

give the following result

$$phi = \pi/2 = 1.57079 \qquad u = [0, 0, -1]^t$$

mtoscrew

Matrix to screw.

Calling sequence:	`[u, phi, P, h] = mtoscrew(Q)`
Return value:	`COL3 - u; real - phi, h; POINT - P.`
Input parameters:	`MAT4 - Q.`

Extracts from a rototranslation matrix **Q** the parameters of the screw displacement (axis **u**, rotation angle **phi**, displacement **h** along **u**, a point of the axis **P**). Point **P** is the point of the screw axis nearest to the origin of the reference frame. `mtoscrew` performs the inverse operation than `screwtom`.

See also example 3.3

See also: `extract`, `rotat`.

Example 3.3. *See sample program* E_MTOSCR.M.

Referring to example 3.2, the following statements:

```
Q=[0 1 0 2; -1 0 0 0; 0 0 1 0; 0 0 0 1];
[u, phi, P, h]=mtoscrew(Q)
```

give the result

$$phi = \pi/2 = 1.57079 \qquad h = 0$$
$$P = [\,1\ -1\ 0\ 1\,]' \qquad u = [\,0\ 0\ -1\,]'$$

screwtom

Screw to Matrix.

Calling sequence:	`Q = screwtom(u, phi, P, h)`
Return value:	`MAT4 - Q.`
Input parameters:	`COL3 - u; real - phi, h; POINT - P.`

Builds the rototranslation matrix **Q** from the axis of the screw displacement **u**, the rotation angle **phi**, the translation **h** along **u** and the coordinates of a point **P** of the axis. `screwtom` performs the inverse operation than `mtoscrew`.

See also example 3.4

See also: `extract`, `rotat`.

Example 3.4. *See sample program* E_SCREWT.M.

Referring to the figure 3.3 with the given values

$$phi = \pi/2 = 1.57079 \qquad h = 0$$
$$P = [1\ -1\ 0\ 1]' \qquad u = [0\ 0\ -1]'$$

the following statements

```
fi=pi/2;
u=Zaxis_n;
P=[1 -1 0 1]';
h=0;
Q=screwtom(u, fi, P, h);
```

give the resulting matrix

$$Q = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 2 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

rotat

Builds the rotation matrix R.

Calling sequence:	`A = rotat(u, phi)`
Return value:	`MAT - A.`
Input parameters:	`COL3 - u; real - phi.`

Builds the rotation matrix **R** from the unit vector **u** and the rotation angle **phi** of the angular displacement; it stores the matrix in the 3×3 matrix. `rotat` performs the inverse operation than `extract`.

Example: (see also example 3.5)

`R=rotat(u, phi)`	builds a 3×3 rotation matrix `R`.
`M(1:3, 1:3)=rotat(u, phi)`	builds a rotation matrix storing it in the 3×3 upper left part of a matrix `M`.

See also: `rotat2`, `rotat24`, `rotat34`.

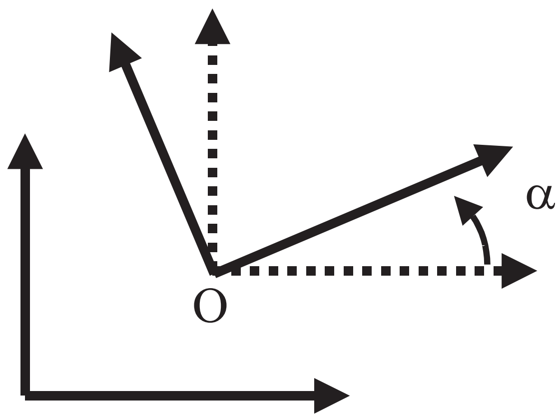


Figure 3.4: `rotat24` example of use: `M01=rotat24(a,alpha,O)` with a=`X`, `Y`, or `Z`

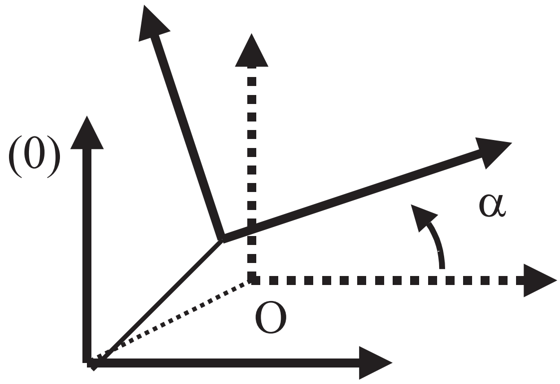


Figure 3.5: `rotat34` example of use: `M01=rotat34(a,alpha,O)` with a=`X`, `Y`, or `Z`

Example 3.5. *See sample program* `E_ROTAT.M`.

Referring to the figure 3.3 with the given values

$$phi = \pi/2 = 1.57079 \qquad u = [0, 0, -1]'$$

the following statements

```
u=Zaxis_n;
phi=pi/2;
A=rotat(u, phi);
```

give the resulting matrix

$$A = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

rotat2

Rotation around a frame axis.

Calling sequence:	`R = rotat2(a, phi)`
Return value:	`MAT3 - R.`
Input parameters:	`int - a; real - phi.`

This function builds a 3×3 rotation matrix **R** describing a rotation of angle **phi** about axis **a**. **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3). The rotation matrix is stored in the 3×3 upper-left part of a matrix **R**. If `a` is equal to `U`, the rotation is assumed null (3×3 identity matrix generated).

See also: `rotat`, `rotat24`, `rotat34`.

rotat24

Rotation matrix around an axis with origin in a given point.

Calling sequence:	`R = rotat24(a, phi, O)`
Return value:	`MAT - R.`
Input parameters:	`int - a; real - phi; POINT - O.`

This function builds a position matrix **m** of a frame whose origin is stored in point **O** and rotated of angle **phi** about axis **a** (see fig. 3.4). **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3). If `a` is equal to `U` the rotation is assumed null (3×3 identity matrix generated).

See also: `rotat`, `rotat2`, `rotat34`.

rotat34

Rotation matrix around an axis with origin in a given point.

Calling sequence:	`M=rotat34(a, phi, O)`
Return value:	`MAT4 - M.`
Input parameters:	`int - a; real - phi; POINT - O.`

Similar to function `rotat24`, but `rotat34` builds a position matrix **m** of a frame whose origin is initially in point **O** before a rotation **a** (see fig. 3.5). **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3). First of all the origin is placed in point **O**, then the frame (origin included) is rotated of angle **phi** about axis **a** of the absolute frame (see § 7.6 for example of use). If `a` is equal to `U` the rotation is assumed null (3×3 identity matrix generated).

See also: `rotat`, `rotat2`, `rotat24`.

traslat

Builds the matrix m of a translation along a vector.

Calling sequence:	`m = traslat (u, h);`
Return value:	`MAT4 m.`
Input parameters:	`VECTOR u; real h.`

Builds the translation matrix **m** from the unit vector **u** and the translation distance **h** of the prismatic displacement.

See also: `traslat2`, `traslat24`, `mtoscrew`.

traslat2

Builds the matrix m of a translation along a frame axis.

Calling sequence:	`m = traslat2 (a, h);`
Return value:	`MAT4 m.`
Input parameters:	`int a; real h.`

Builds the matrix **m** of the translation along axis **a** and with translation distance **q**. **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3).

See also: `traslat`, `traslat24`, `mtoscrew`.

traslat24

Builds the matrix m of a translation along a frame axis with origin in a given point.

Calling sequence:	`m = traslat24 (a, h, p);`
Return value:	`MAT4 m.`
Input parameters:	`int a; real h, POINT p.`

Builds the matrix **m** of the translation along axis **a**, translation distance **q** and origin in point **P**. **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3).

See also: `traslat`, `traslat2`, `mtoscrew`.

3.2 Speed and acceleration matrices

gtom

Gravity acceleration to Matrix.

Calling sequence:	`Hg = gtom(gx, gy, gz)`
Return value:	`MAT4 - Hg.`
Input parameters:	`real - gx, gy, gz.`

Builds the gravity matrix **Hg** starting from the components **gx**, **gy**, **gz** of the gravity acceleration. Usually the z-axis is vertical and points upwards and so the acceleration vector components are gx = 0 m/s^2, gy = 0 m/s^2, gz = -9.81 m/s^2.

See also example 3.6

Example 3.6. *See sample program* E_GTOM.M.

This example shows how to create the acceleration matrix H_g. This is the most common situation where the body falls down along the z direction (see figure 3.6). The following statements

```
gx=0;
gy=0;
gz=-9.81;
Hg=gtom(gx, gy, gz);
```

build the acceleration matrix H_g of the falling body in figure 3.6. H_g is:

$$H_g = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9.81 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

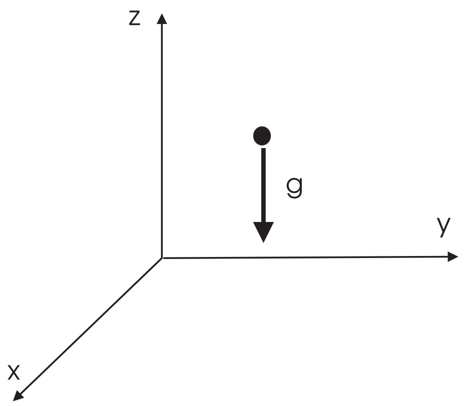


Figure 3.6: Frame definition for example 3.6

gtomgapt

G to omega dot.

Calling sequence:	`omegapto = gtomgapt(G)`
Return value:	`COL3 - omegapto.`
Input parameters:	`MAT3 - G.`

Extracts the angular acceleration vector **omegapto** from the 3×3 upper-left submatrix **G** of the acceleration matrix. It uses the relation

$$\dot{\Omega} = \frac{(G - G^t)}{2} \tag{3.3}$$

See also: `wtovel`.

makel

Builds a L matrix.

Calling sequence:	`L = makel(jtype, u, pitch, P)`
Return value:	`MAT4 - L.`
Input parameters:	`int - jtype; COL3 - u; real - pitch; POINT - P.`

This function builds a *ISA*'s (*Instantaneous Screw Axis*) matrix **L** describing screw motion (including simple rotations or a translation), describing a rotation or a translation about an axis which passes through the point **P** and whose unit vector is **u**. **pitch** is the pitch of the screw. **jtype** specifies the type of the motion. It must be either the constant `Pri` for prismatic joints or `Rev` for revolute or screw joints. `Pri` and `Rev` are constants defined in `spacelib.m`(see also § 2.4.3). If `jtype` is equal to `Pri`, pitch is ignored.

See also example 3.7 and *example* 3.8.

See also: `makel2`

Example 3.7. *See sample program* E_MAKEL.M *and* E_MAKELO.M.

This example shows how to create the *ISA*'s (*Instantaneous Screw Axis*) matrix L of the body n° 2 rotating about an axis coincident with z_0 in two different reference frame (0) and (k). The 2^{nd} element of this revolute joint rotates about axis z of reference frame (0) and has $a = 1.2$ m. The L matrix referred to reference frame (0) is

$$L_{(0)} = \left[\begin{array}{ccc|c} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

This matrix can be built by the following statements:

```
O=ORIGIN;
pitch=0.;
u=Zaxis;
L0=makel(Rev, u, pitch, O);
```

The L matrix referred to frame (k) is

$$L_{(k)} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1.2 \\ 0 & -1 & 0 & 1.2 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

It is built by the following statements

```
P=[0 1.2 1.2 1]';
pitch=0.;
u=Xaxis_n;
Lk=makel(Rev, u, pitch, P);
```

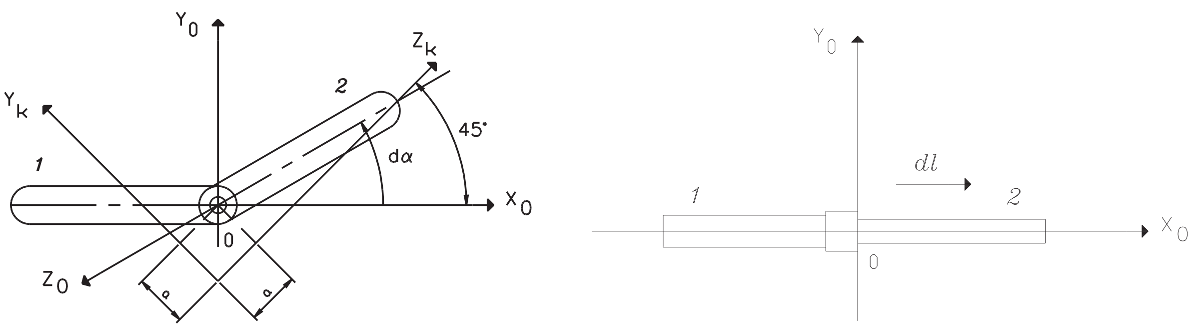


Figure 3.7: Frames definition for examples 3.7 and 3.8

Example 3.8. *See sample program* E_MAKELP.M.

In this example body n ° 2 moves in the direction of x_0. Frame (0) is embedded on body n ° 1. The L matrix of this prismatic joint referred to frame (0) is

$$L_{(0)} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

This matrix is built by the following statements

```
O=ORIGIN;
u=Xaxis;
pitch=0;
L0=makel(Pri, u, pitch, O);
```

makel2

Builds a L matrix - version 2.

Calling sequence:	`L = makel2(jtype, a, pitch, P)`
Return value:	`MAT4 - L.`
Input parameters:	`int - jtype, a; real - pitch; POINT - P.`

This function builds a *ISA*'s (*Instantaneous Screw Axis*) matrix **L** describing a rotation or a translation about an axis parallel to the frame axis **a** and passing through the point **P**. **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3). **pitch** is the pitch of the screw. **jtype** specifies the type of the motion. It must be either the constant `Pri` for prismatic joints or `Rev` for revolute or screw joints. `Pri` and `Rev` are constant defined in `spacelib.m` (see also § 2.4.3). If `jtype` is equal to `Pri`, pitch is ignored.

See also: `makel`

wtol

Extracts L matrix from the corresponding W matrix.

Calling sequence:	`L = wtol(W)`
Return value:	`MAT4 - L.`
Input parameters:	`MAT4 - W.`

Extracts the *ISA*'s (*Instantaneous Screw Axis*) matrix **L** from the corresponding **W** matrix. If **W** is the null matrix, the function returns a **L** matrix filled with zeros (null matrix).

See also example 3.9 and *example* 3.10.

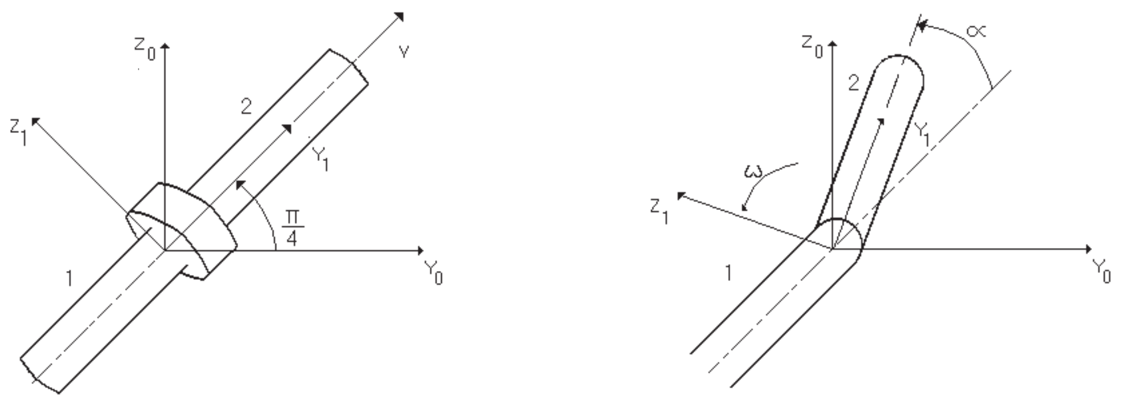


Figure 3.8: Frames definition for examples 3.9 and 3.10

Example 3.9. *See sample program* `E_WTOL_P.M`.

This example shows how to extract the L matrix knowing the velocity one. In this case the considered joint is prismatic and it lies in the Y_0-Z_0 plane forming an angle of 45° with Y_0. The element 2 is connected by a prismatic joint to body 1 which does not move with respect to frame (0). Body 2 has the following velocity matrix W referred to frame (0) embedded on body 1:

$$W_{(0)} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.4142 \\ 0 & 0 & 0 & 1.4142 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

The L matrix is built by the statements:

```
W = [ 0 0 0 0 ;
      0 0 0 1.4142;
      0 0 0 1.4142;
      0 0 0 0];
L = wtol(W);
```

and the result is:

$$L = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7071 \\ 0 & 0 & 0 & 0.7071 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

Example 3.10. *See sample program* E_WTOL_R.M.

This example shows how to extract the *ISA*'s (*Instantaneous Screw Axis*) matrix L knowing the velocity one. In this case the considered joint is revolute and it rotates about axes x_0 orthogonal to plane Y_0-Z_0 passing through the center of the joint. The element 2 is connected by a revolute joint to body 1 which does not move with respect to frame (0). Body 2 has the following velocity matrix referred to frame (0)

$$W_{(0)} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 2 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

The L matrix is built by the statements:

```
W = [0 0 0 0;
     0 0 -2 0;
     0 2 0 0;
     0 0 0 0];
L = wtol(W);
```

and the result is:

$$L = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

wtovel

Velocity matrix to velocity parameters.

Calling sequence:	`[u, omega, vel, P] = wtovel(W)`
Return value:	`COL3 - u; real - omega, vel; POINT - P.`
Input parameters:	`MAT4 - W.`

Extracts the screw parameters from a velocity matrix **W**. The parameters are: **u** axis of rotation (unit vector), **omega** angular speed around **u** (scalar), **vel** linear velocity along **u**, **P** a point of the axis (the closest to the origin). If **omega** is equal to `0` (pure translation) the origin is assumed as **P**. If both **omega** and **vel** are equal to `0`, **u** is undefined.

See also example 3.11.

Example 3.11. *See sample program* E_WTOVEL.M *and* E_WTOV_P.M.

Considering a velocity matrix W

$$W = \left[\begin{array}{ccc|c} 0 & -2 & 2.5 & 2.5 \\ 2 & 0 & -4.5 & 1.7 \\ -2.5 & 4.5 & 0 & 3.2 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

The following statements:

```
W=[0   -2   2.5 2.5;
   2    0  -4.5 1.7;
  -2.5 4.5 0    3.2;
   0    0   0    0 ];
[u, omega, vel, P]= wtovel(W);
```

evaluate the angular velocity *omega*, scalar velo-city *vel* and a point P of the screw axis u. P is the point of the axis nearest to the origin of the reference frame. The result is:

$$u = [\,0.815\ 0.453\ 0.362\,]' \qquad omega = 5.52 \text{ rad/s}$$

$$P = [\,0.151\ \ -0.308\ 0.046\ 1\,]' \qquad vel = 3.965 \text{ m/s}$$

Considering a pure translation movement, the velocity matrix W is:

$$W = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 1.7 \\ 0 & 0 & 0 & 3.2 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

The statements are the same as in the previous case except that the matrix W is filled with different values. The result in this case is:

$$u = [\,0.568\ 0.386\ 0.727\,]' \qquad omega = 00 \text{ rad/s}$$

$$P = [\,0\ 0\ 0\ 1\,]' \qquad vel = 4.40227 \text{ m/s}$$

veactowh

Velocity and Acceleration to W and H matrices.

Calling sequence:	`[W, H] = veactowh(jtype, qp, qpp)`
Return value:	`MAT4 - W, H.`
Input parameters:	`int - jtype, real - qp, qpp.`

Builds both velocity and acceleration matrices in local frame (**W** and **H**) from the values of the joint velocity and acceleration (**qp** and **qpp**) and the type of the joint **jtype**. The axis of the movement is the z axis of the local reference frame. **jtype** is an integer whose value must be either `Rev` or `Pri`. `Rev` and `Pri` are constant defined in the header file `spacelib.m` (see also § 2.4.3). Frames are assumed to be positioned using the Denavit and Hartenberg's convention [3] [4].

See also: `vactowh2`, `vactowh3`.

vactowh2

Velocity and Acceleration to W and H matrices - version 2.

Calling sequence:	`[W, H] = vactowh2(jtype, a, qp, qpp)`
Return value:	`MAT4 - W, H.`
Input parameters:	`int - jtype, a; real - qp, qpp.`

Builds both local speed and acceleration matrices (**W** and **H**) from the values of the velocity and acceleration **qp** and **qpp** of the link around the axis **a** and the type of the joint **jtype**. The motion axis is coincident with x, y, or z of the local reference frame. **jtype** is an integer whose value must be either `Rev` or `Pri`. `Rev` and `Pri` are constant defined in the header file `spacelib.m` (see also § 2.4.3). This function is equivalent to `veactowh` except that the movement axis can be specified. **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3). The following statement:

```
[W, H] = vactowh2(jtype, Z, qp, qpp);
```

is equivalent to

```
[W, H] = veactowh(jtype, qp, qpp);
```

See also example 3.12

See also: `veactowh`.

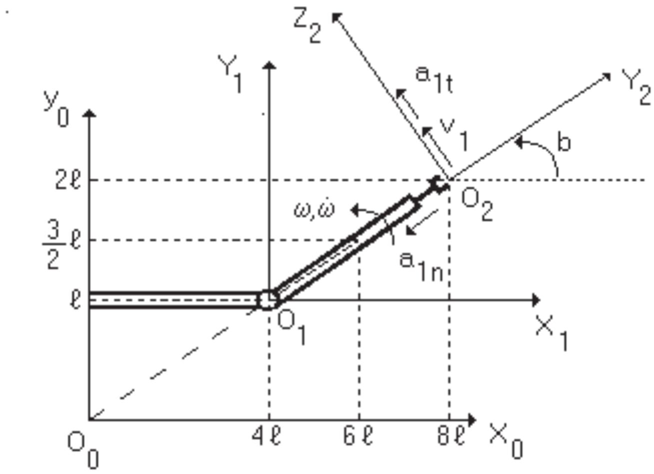


Figure 3.9: Frames definition for example 3.12

Example 3.12. *See sample program* `E_VELWH2.M`.

Two fixed reference frames are defined. The two bodies are connected by a revolute joint (see figure 3.9). One body is fixed while the other rotates about the origin of frame (1). With the given values

$$l = 0.1 \text{ m} \quad \omega = 1.5 \text{ rad/s} \quad \dot\omega = 0.9 \text{ rad/s}^2$$

the velocity and acceleration matrices referred to frame (1) are:

$$W_{12(1)} = \left[\begin{array}{ccc|c} 0 & -1.5 & 0 & 0 \\ 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \qquad H_{12(1)} = \left[\begin{array}{ccc|c} -2.25 & -0.9 & 0 & 0 \\ 0.9 & -2.25 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

These matrices are built by the statements:

```
qp=1.5;
qpp=0.9;
[W1, H1]=vactowh2(Rev, Z, qp, qpp);
```

vactowh3

Velocity and Acceleration to W and H matrices - version 3.

Calling sequence:	`[W, H]=vactowh3(jtype, a, qp, qpp, O)`
Return value:	`MAT4 - W, H.`
Input parameters:	`int - jtype, a; real - qp, qpp; POINT - O.`

Builds both local speed and acceleration matrices (**W** and **H**) from the values of the velocity and acceleration **qp** and **qpp** of the link around the axis **a**. The motion axis is parallel to x, y, or z of the local reference frame. This function is similar to `vactowh2`: the movement axis is parallel to one of the coordinate axes and can pass through a point different from the origin of the reference frame. **a** must be one of the constants `X`, `Y`, `Z`, `U` (see § 2.4.3). **O** is a point of the rotation axis. The following statement:

```
[W, H]=vactowh3(jtype, a, qp, qpp, O);
```

is equivalent to

```
[W, H]=vactowh2(jtype, a, qp, qpp);
```

when the point **O** coincides with the origin of the reference frame (and therefore **a** is one of the axes of the reference frame).

Example: Referring to example 3.12, the matrix $W_{1,2(0)}$ is obtained by the following statement

```
real qp=1.5;
real qpp=0.9;
POINT O1=[0.4 0.1 0 1]';
[W0, H0]=vactowh3(Rev, Z, qp, qpp, O1);
```

See also: `vactowh2`.

coriolis

Coriolis' theorem.

Calling sequence:	`H = coriolis(H0, H1, W0, W1)`
Return value:	`MAT4 - H.`
Input parameters:	`MAT4 - H0, H1, W0, W1.`

Performs the Coriolis' theorem:

$$H = H_0 + H_1 + 2W_0 \cdot W_1 \tag{3.4}$$

3.3 Inertial and Actions Matrices

dyn_eq

Solve Direct Dynamics system.

Calling sequence:	`[Wp, F, test] = dyn_eq(J, Wp, F, var)`
Return value:	`MAT4 - Wp, F; int - test.`
Input parameters:	`MAT4 - J, Wp, F;` 2×6 `matrix - var.`

Evaluates the acceleration term $\dot{W}$ of a rigid body free in space solving the matrix equation

$$\Phi = skew\left(\dot{W} \cdot J\right) \tag{3.5}$$

where `Wp is` $\dot{W}$ `and F is` Φ. It can also extract the velocity matrix W of a body from the angular momentum matrix Γ solving the equation

$$\Gamma = skew\left(W \cdot J\right) \tag{3.6}$$

where `Wp is` W, and `F is` Γ.
var specifies which elements of **Wp** and **F** are unknown (for more details on this function see also § 5).

actom

Actions to Matrix.

Calling sequence:	`FI=actom(fx, fy, fz, cx, cy, cz)`
Return value:	`MAT4 - FI.`
Input parameters:	`real - fx, fy, fz, cx, cy, cz.`

Builds the action matrix **PHI** from the components of the forces **fx**, **fy**, **fz** and the torque (or couples) **cx**, **cy**, **cz**.

Example: With the given values

$$fx = a, \quad fy = b, \quad fz = c, \quad cx = d, \quad cy = e, \quad cz = f$$

the statement

```
FI = actom(fx, fy, fz, cx, cy, cz)
```

fills the matrix FI in the following way

$$FI = \left[\begin{array}{ccc|c} 0 & -f & e & a \\ f & 0 & -d & b \\ -e & d & 0 & c \\ \hline -a & -b & -c & 0 \end{array}\right]$$

jtoj

Inertia moment and mass to inertia matrix.

Calling sequence:	`J=jtoj(m, jxx, jyy, jzz, jxy, jyz, jxz, xg, yg, zg)`
Return value:	`MAT4 - J.`
Input parameters:	`real - m, jxx, jyy, jzz, jxy, jyz, jxz, xg, yg, zg.`

Builds the inertia matrix **J** of a body from the values of its mass **m**, its barycentral moments of inertia **jxx**, **jyy**, **jzz**, **jxy**, **jyz**, **jxz** and the position of its center of mass **xg**, **yg**, **zg**. The barycentral frame must be parallel to the reference frame. The resulting matrix is:

$$J = \left[\begin{array}{ccc|c} Ixx & Iyx & Izx & m\,xg \\ Ixy & Iyy & Izy & m\,yg \\ Ixz & Iyz & Izz & m\,zg \\ \hline m\,xg & m\,yg & m\,zg & m \end{array}\right]$$

The elements I of the **J** matrix are not the usual barycentral moments. These elements are related to the usual barycentral moments as follows:

$$Ixx = \frac{-\,Jxx + Jyy + Jzz}{2} \qquad Iyy = \frac{-\,Jyy + Jxx + Jzz}{2} \qquad Izz = \frac{-\,Jzz + Jxx + Jyy}{2}$$

$$Ixy = -Jxy \qquad Iyz = -Jyz \qquad Izx = -Jzx$$

where the value of `Ixx`, `Iyy`, `Izz` must be positive, therefore `Jxx`, `Jyy`, `Jzz` cannot be assigned random values. The **J** elements are defined as follows

$$Jxx = \int y^2 + z^2 \,\mathrm{dm} \qquad Jyy = \int x^2 + z^2 \,\mathrm{dm} \qquad Jxx = \int x^2 + y^2 \,\mathrm{dm}$$

$$Jxy = \int -xy \,\mathrm{dm} \qquad Jyz = \int -yz \,\mathrm{dm} \qquad Jxz = \int -xz \,\mathrm{dm}$$

The I elements are defined as follows

$$Ixx = \int x^2 \,\mathrm{dm} \qquad Iyy = \int y^2 \,\mathrm{dm} \qquad Ixx = \int z^2 \,\mathrm{dm}$$

$$Ixy = \int xy \,\mathrm{dm} \qquad Iyz = \int yz \,\mathrm{dm} \qquad Ixz = \int xz \,\mathrm{dm}$$

See also example 3.13.

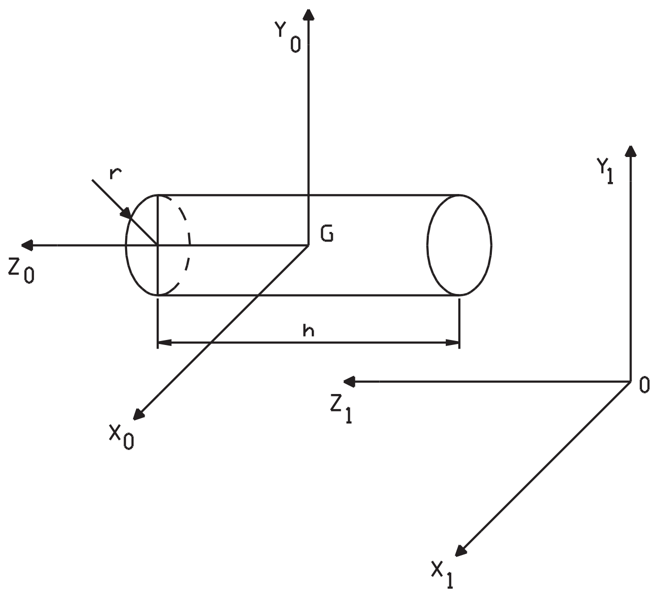


Figure 3.10: Frames definition for example 3.13

Example 3.13. *See sample program* `E_JTOJ.M`.

This example shows how to create the inertial matrix of a cylinder in two different frames (0) and (1). The first one is centered in the center of mass G. The cylinder in figure 3.10 has $r = 1$ kg/m^3 and $h = 5$ m. G is the center of mass. Its position in frame (0) is [0, 0, 0] and in frame (1) [0, 3, 4]. The following statements:

```
m=15.71;
jxx=36.633; jyy=36.633; jzz=7.85;
jxy=0; jyz=0; jxz=0;
xg=0; yg=0; zg=0;
J=jtoj(m, jxx, jyy, jzz, jxy, jyz, jxz, xg, yg, zg);
```

where `m` is the mass, `jxx`, `jyy`, `jzz`, `jxy`, `jyz`, `jxz` are the usual inertia moments with respect to the center of mass, `xg`, `yg`, `zg` are the coordinates of the center of mass in frame (0), builds the inertia matrix $J_{(0)}$ of the cylinder in the barycentral reference frame:

$$J_{(0)}=\left[\begin{array}{ccc|c} I_{xx} & 0 & 0 & 0 \\ 0 & I_{yy} & 0 & 0 \\ 0 & 0 & I_{zz} & 0 \\ \hline 0 & 0 & 0 & mass \end{array}\right] = \left[\begin{array}{ccc|c} 3.925 & 0 & 0 & 0 \\ 0 & 3.925 & 0 & 0 \\ 0 & 0 & 32.708 & 0 \\ \hline 0 & 0 & 0 & 15.71 \end{array}\right]$$

The following statements:

```
m=15.71;
jxx=36.633; jyy=36.633; jzz=7.85;
jxy=0; jyz=0; jxz=0;
xg=0; yg=3; zg=4;
J=jtoj(m, jxx, jyy, jzz, jxy, jyz, jxz, xg, yg, zg);
```

where `m` is the mass, `jxx`, `jyy`, `jzz`, `jxy`, `jyz`, `jxz` are the usual inertia moments with respect to the center of mass, and `xg`, `yg`, `zg` are the coordinates of the center of mass, builds the inertia matrix $J_{(1)}$ of the cylinder in frame (1) in which all the quantities are expressed:

$$J_{(1)}=\left[\begin{array}{ccc|c} I_{xx} & I_{yx} & I_{zx} & m\cdot xg \\ I_{xy} & I_{yy} & I_{zy} & m\cdot yg \\ I_{xz} & I_{yz} & I_{zz} & m\cdot zg \\ \hline m\cdot xg & m\cdot yg & m\cdot zg & m \end{array}\right] = \left[\begin{array}{ccc|c} 3.925 & 0 & 0 & 0 \\ 0 & 145.315 & 188.52 & 47.13 \\ 0 & 188.52 & 284.068 & 62.84 \\ \hline 0 & 47.13 & 62.84 & 15.71 \end{array}\right]$$

It is easy to verify that inertia matrix $J_{(1)}$ can also be obtained using the function `mamt` (see § 3.4.2):

```
J1=mamt(J0, M10)
```

where `M10` is the position matrix of frame (0) with respect to frame (1). In the example it is assumed that the axes of frames (0) and (1) are parallel to each other while the position of frame (1) with respect to (0) is (x, y ,z)=(0, 3, 4).

PseDot

Pseudo scalar product.

Calling sequence:	`a = PseDot(L, F)`
Return value:	`real - a.`
Input parameters:	`MAT4 - L, F.`

Performs the pseudo-scalar product between matrices **L** and **F**.

Example:

If `W` is the velocity matrix of a body and `F` is the matrix of the actions (forces and torques) applied to it,

$$W = \left[\begin{array}{ccc|c} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \qquad \Phi = \left[\begin{array}{ccc|c} 0 & -c_z & c_y & f_x \\ c_z & 0 & -c_x & f_y \\ -c_y & c_x & 0 & f_z \\ \hline -f_x & -f_y & -f_z & 0 \end{array}\right]$$

where f is the force applied to the body and c is the torque, then the power

$$w = W \odot \Phi = \omega_x c_x + \omega_y c_y + \omega_z c_z + v_x f_x + v_y f_y + v_z f_z$$

is evaluated as

```
PseDot(W,F) % power of forces
```

If `L` represents the screw axis of a joint and `F` is the total action transmitted by it, the actuator force (or torque) is evaluated as

```
PseDot(L,F) % project forces on the direction of the motion
```

3.4 Matrix transformations

3.4.1 Matrix normalization

normal

Normalizes (orthogonalises) a 3×3 rotation matrix or the 3×3 upper-left submatrix of a position matrix.

Calling sequence:	`An = normal(A)`
Return value:	`MAT - An.`
Input parameters:	`MAT - A.`

Normalizes the the 3×3 upper-left submatrix `R` of a position matrix **A**. `R` is iteratively put equal to (see [2])

$$R_{i+1} = \frac{1}{2} \cdot \left(\frac{1}{\sqrt[3]{det(R_i)}} \cdot R_i + \sqrt[3]{det(R_i)} \cdot (R_i^t)^{-1} \right) \tag{3.7}$$

until R_{i+1} does not vary in one iteration. This function is used when the rotation matrix is evaluated by numerical procedure and could contain errors.

See also: `normal_g`, `normal3`.

normal_g

Normalizes (orthogonalises) any square matrix.

Calling sequence:	`An = normal_g(A)`
Return value:	`MAT - An.`
Input parameters:	`MAT - A.`

Transform matrix `A` into the "most similar" orthogonal matrix ($A^{-1} = A^t$) using the equation (3.7).

See also: `normal`, `normal3`.

normal3

Normalizes (orthogonalises) a 3×3 matrix.

Calling sequence:	`Rn = normal3(R)`
Return value:	`MAT3 - Rn.`
Input parameters:	`MAT3 - R.`

Transform the 3×3 matrix `R` into the "most similar" orthogonal matrix ($R^{-1} = R^t$). This is an optimized version of `normal_g` for 3×3 matrices.

See also: `normal3`, `normal_g`.

normskew

Normalizes symmetric or skew-symmetric matrices.

Calling sequence:	`An = normskew(A, sign)`
Return value:	`MAT - An.`
Input parameters:	`MAT - a; int - sign.`

Normalizes a square matrix **A** (extracts the symmetric or skew-symmetric part of **A**). This function can be used when matrix **A** is evaluated by numerical procedure and could contain errors. **sign** is an integer whose value can be either `SYMM_` or `SKEW_`. `SYMM_` and `SKEW_` are constants defined in the header file (see § 2.4.3). If **sign** is equal to `SYMM_` then the function normalizes a symmetric matrices using the equation

$$\mathbf{A} = \frac{A + A^t}{2} \tag{3.8}$$

If **sign** is equal to `SKEW_` then the function normalizes skew-symmetric matrices using the equation

$$\mathbf{A} = \frac{A - A^t}{2} \tag{3.9}$$

3.4.2 Change of reference

mami

Transforms a matrix by the rule of $M \cdot A \cdot M^{-1}$ (mami = $M \cdot A \cdot$ Minverse).

Calling sequence:	`A2 = mami(A1, M)`
Return value:	`MAT - A2.`
Input parameters:	`MAT - A1, M.`

Performs the matrix operation

$$A_{(r)} = M_{r,s} \cdot A_{(s)} \cdot M_{s,r} = M_{r,s} \cdot A_{(s)} \cdot M_{r,s}^{-1} \tag{3.10}$$

useful in the change of reference of Q, L, W and H matrices. **A1** and **A2** are square 4×4 matrices. **m** is a transformation matrix. `mami` performs the inverse operation than `miam`.

See also example 3.14

See also: `miam`, `miamit`, `mamt`.

Example 3.14. *See sample program* E_TRSF_M.M.

Referring to example 3.12, let's consider the velocity matrix $W_{(1)}$ and the acceleration matrix $H_{(1)}$ both referred to reference frame (1) defined as follows

$$W_{(1)}= \left[\begin{array}{ccc|c} 0 & -1.5 & 0 & 0 \\ 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \qquad H_{(1)}= \left[\begin{array}{ccc|c} -2.25 & -0.9 & 0 & 0 \\ 0.9 & -2.25 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

Moreover, the position of frame (1) referred to frame (0) is expressed by the matrix

$$M_{0,1}= \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.4 \\ 0 & 1 & 0 & 0.1 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The velocity and acceleration matrices referred to reference frame (0) are $W_{(0)}$ and $H_{(0)}$ defined as

$$W_{(0)}= \left[\begin{array}{ccc|c} 0 & -1.5 & 0 & 0.150 \\ 1.5 & 0 & 0 & -0.600 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \qquad H_{(0)}= \left[\begin{array}{ccc|c} -2.25 & -0.9 & 0 & 0.900 \\ 0.9 & -2.25 & 0 & -0.135 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

$W_{(0)}$ and $H_{(0)}$ matrices can be built by means of the following statements

```
W1=[0 -1.5 0 0; 1.5 0 0 0;
      0 0 0 0; 0 0 0 0];
H1=[ -2.25 -0.9 0 0; 0.9 -2.25 0 0 ;
      0 0 0 0; 0 0 0 0];
m01=[1 0 0 0.4; 0 1 0 0.1;
     0 0 1 0;   0 0 0 1];
W0=mami(W1, m01);
H0=mami(H1, m01);
```

miam

Transforms by the rule $M^{-1} \cdot A \cdot M$ (miam = Minverse$\cdot A \cdot M$).

Calling sequence:	`MAT A2 = miam(MAT A1, MAT M)`
Return value:	`MAT - A2.`
Input parameters:	`MAT - A1, M.`

Performs the matrix operation

$$A_{(s)} = M_{s,r} \cdot A_{(r)} \cdot M_{r,s} = M_{r,s}^{-1} \cdot A_{(r)} \cdot M_{r,s} \tag{3.11}$$

for 4×4 matrices which are contra-variant with respect to the row index and co-variant with respect to the column index. `miam` performs the inverse operation than `mami`.

See also: `mami`, `miamit`, `mamt`.

miamit

Transforms by the rule $M^{-1} \cdot A \cdot (M^{-1})^t$ (miamit = Minverse $\cdot A \cdot$ Minverse transposed).

Calling sequence:	`A2 = miamit(A1, M)`
Return value:	`MAT - A2.`
Input parameters:	`MAT - A1, M.`

Performs the matrix operation

$$A_{k(r)} = M_{r,s} \cdot A_{k(s)} \cdot M_{r,s}^t \tag{3.12}$$

for 4×4 contra-variant matrices. `miamit` performs the inverse operation than `mamt`.

See also example 3.15

See also: `mami`, `miamit`, `mamt`.

mamt

Transforms by the rule $M \cdot A \cdot M^t$ (mamt = $M \cdot A \cdot$ Mtranspose).

Calling sequence:	`A2 = mamt(A1, M)`
Return value:	`MAT - A2.`
Input parameters:	`MAT - A1, M.`

Performs the matrix operation

$$A_{k(s)} = M_{s,r} \cdot A_{k(r)} \cdot M_{s,r}^t = M_{r,s}^{-1} \cdot A_{k(r)} \cdot M_{r,s}^{-t} \tag{3.13}$$

useful in the change of reference of J, Γ and Φ matrices. **A1** and **A2** are square matrices. **m** is a transformation matrix `mamt` performs the inverse operation than `miamit`.

See also: `mami`, `miamit`, `mamt`.

Example 3.15. *See sample program* `E_TRMAMT.M`.

A system is made up of 3 point bodies whose masses are 5 kg, 1 kg and 2.5 kg. Their position and velocity referred to a reference frame (1) are respectively

$$P_{1(1)} = [2\ 3\ 4\ 1]^t \qquad P_{2(1)} = [0\ 1\ 0\ 1]^t$$

$$P_{3(1)} = [1\ 3\ 0\ 1]^t \qquad \dot{P}_{1(1)} = [1\ 0\ 2\ 0]^t$$

$$\dot{P}_{2(1)} = [4\ 0.5\ 1\ 0]^t \qquad \dot{P}_{3(1)} = [0\ 0\ 1\ 0]^t$$

The angular momentum matrix $\Gamma_{(1)}$ referred to this reference frame can be written as

$$\Gamma_{(1)} = \sum_{i=1}^{3} \left(\dot{P}_i P_i^t - P_i \dot{P}_i^t \right) m_i = \left[\begin{array}{ccc|c} 0 & 19 & -2.5 & 9 \\ -19 & 0 & -38.5 & 0.5 \\ 2.5 & 19 & 0 & 13.5 \\ \hline -9 & -0.5 & -13.5 & 0 \end{array} \right]$$

The position of reference frame (1) respect to another frame, frame (0), is expressed by the following matrix

$$M_{0,1} = \left[\begin{array}{ccc|c} 0 & 1 & 0 & -1.2 \\ -1 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 4 \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

The angular momentum matrix $\Gamma_{(0)}$ in reference frame (0) can be calculated

$$\Gamma_{(0)} = M_{0,1}\, \Gamma_{(1)}\, M_{0,1}^t = \left[\begin{array}{ccc|c} 0 & 29.55 & -52.7 & 9 \\ -29.55 & 0 & -26.75 & 0.5 \\ 52.7 & 26.75 & 0 & 13.5 \\ \hline -9 & -0.5 & -13.5 & 0 \end{array} \right]$$

The previous operations can be executed by means of the following statements

```
GAMMA1=[0 19 -2.5 9; -19 0 -38.5 0.5;
       2.5 38.5 0 13.5; -9 -0.5 -13.5 0]
m=[0 1 0 1.2;-1 0 0 -0.5;
   0 0 1 4; 0 0 0 1];
GAMMA0=mamt(GAMMA1, m);
```

Where `m` is M_{01}, `GAMMA0` is $\Gamma_{(0)}$, `GAMMA1` is $\Gamma_{(1)}$. Opposite, $\Gamma_{(1)}$ can be evaluated from $\Gamma_{(0)}$ by the following statement:

```
GAMMA1=miamit(m, GAMMA0);}
```

3.4.3 General operations

grad

Conversion from radians to degrees

Calling sequence:	`g = grad(r)`
Return value:	`real - g.`
Input parameters:	`real - r.`

Obsolete version. Please use function `deg`.

deg

Conversion from radians to degrees

Calling sequence:	`g = deg(r)`
Return value:	`real - r.`
Input parameters:	`real - g.`

`deg` converts the **r** radians value in **g** degrees value. `deg` performs the inverse operation than `rad`.

See also: `rad`.

rad

Conversion from degrees to radiant.

Calling sequence:	`r = rad(g)`
Return value:	`real - r.`
Input parameters:	`real - g.`

`rad` converts the **g** degrees value in **r** radians value. `rad` performs the inverse operation than `deg`.

See also: `deg`.

jrand

Creates a random matrix with elements in the range min .. max

Calling sequence:	`m = jrand(imax, jmax, min, max)`
Return value:	`MAT - m.`
Input parameters:	`int - imax, jmax, min, max.`

Evaluates a **i**×**j** matrix **A** filled with random elements of **min** .. **max** range.

invers

Inverse of a position matrix.

Calling sequence:	`mi= invers(m)`
Return value:	`MAT4 - mi.`
Input parameters:	`MAT4 - m.`

Evaluates the inverse **mi** of a 4×4 position (transformation) matrix **m** using the equation:

$$M_{0,1} = \left[\begin{array}{ccc|c} & R_{0,1} & & T_{0,1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad M_{0,1}^{-1} = M_{1,0} = \left[\begin{array}{ccc|c} & R_{0,1}^t & & -R_{0,1}^t T_{0,1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.14}$$

This function <u>works only</u> for 4×4 transformation matrices.

mtov

Matrix to vector.

Calling sequence:	`v = mtov(A)`
Return value:	`COL3 - v.`
Input parameters:	`MAT - A.`

Extracts vector $\mathbf{v}$[1] 3×1 from the upper-left 3×3 skew-symmetric submatrix of a matrix $\mathbf{A}$. `mtov` performs the inverse operation than `vtom`.

Example:
The function `mtov(M A, 4, v)` applied to the skew-symmetric matrix A defined as

$$A = \left[\begin{array}{ccc|c} 0 & -c & b & d \\ c & 0 & -a & e \\ -b & a & 0 & f \\ \hline g & h & i & j \end{array}\right]$$

give the resulting vector v defined as $v = [a, b, c,]^t$

vtom

Vector to Matrix.

Calling sequence:	`A = vtom(v)`
Return value:	`MAT - A.`
Input parameters:	`COL3 - v.`

Creates a 3×3 skew-symmetric submatrix from vector $\mathbf{v}$ and stores it in the upper-left part of the matrix $\mathbf{A}$. `vtom` performs the inverse operation than `mtov`.

Example:
The statement

```
mtov(M A, 4, v);
```

applied to the vector v defined as $v = [a, b, c,]^t$ gives the resulting skew-symmetric matrix A defined as follows

$$A = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$$

See also: `vtom`.

skew

Evaluates the skew-symmetric matrix $AB - B^t A^t$.

Calling sequence:	`C = skew(A, B)`
Return value:	`MAT - C.`
Input parameters:	`MAT - A,B.`

Performs the matrix operation $\mathrm{C} = \mathrm{skew}\{\mathrm{A} \cdot \mathrm{B}\} = \mathrm{A} \cdot \mathrm{B} - \mathrm{B}^t \cdot \mathrm{A}^t$ applicable to square matrices with any dimension.

[1] A vector $\overrightarrow{v}$ could be represented in the following forms [3], [4], [2]:

$$v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \qquad \underline{v} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

tracljlt

Trace of $L1 \cdot J \cdot L2^t$

Calling sequence:	`T = tracljlt(L1, J, L2)`
Return value:	`real - T.`
Input parameters:	`MAT4 - L1, J, L2.`

Returns the trace[2] of the matrix product $\mathbf{L1} \cdot \mathbf{J} \cdot \mathbf{L2}^t$. Applicable to matrices with any dimension.

3.5 Conversion between Cardan (or Euler) angles and matrices

There are several types of coordinates which can express the relative angular position of two moving bodies in a 3D space; generally two frames are fixed on the two bodies. In the "*neutral*" position, these frames are parallel to each other. One of the two reference frames is called *absolute*, while the other *moving*. Their relative angular position is expressed by the position of the moving frame with respect to the fixed one. The relative orientation of two frames can be imagined as obtained from the neutral position by three subsequent rotations α, β, γ of the moving frame around three axes: i, j, k. Rotations are generally performed about the axes of the fixed or of the moving frame.

It is possible to show that a group of three rotations α, β, γ around axes i, j, k of the fixed frame are equivalent to a sequence of rotations γ, β, α, around axes k, j, i (reverse order) of the moving frame. As the rotation axes are given, the angular position can be expressed by the three rotation angle values. Rotations about fixed axes multiply to left, while about moving axes to right.

Value of i, j, k for rotations around axes of fixed frame				
Systems	***Cardan (Tait-Brian)*** $i \neq j \neq k \neq i$		***Euler*** $i = k \neq j$	
Cyclic	x, y, z	*Cardan angles*	x, y, x	
	y, z, x		y, z, y	
	z, x, y		z, x, z	*Euler angles*
Anti-cyclic	z, y, x	*Nautic angles*	z, y, z	
	x, z, y		x, z, x	
	y, x, z		y, x, y	

Table 3.2: Cardan angles convention

3.5.1 Position

cardator

Cardan (or Euler) angles to rotation matrix.

Calling sequence:	`A = cardator(q, i, j, k)`
Return value:	`MAT3 - A.`
Input parameters:	`ROW3 - q; int - i, j, k.`

Builds a rotation matrix starting from the cardan or Euler angles. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing the 1^{st}, 2^{nd} and 3^{rd} angle. `cardator` performs the inverse operation than `rtocarda`.

See also: `cardatom`.

[2]The trace of a square matrix is the sum of its diagonal elements. If X is a column matrix it yields $Trace(XX^t) = X^tX$.

rtocarda

Rotation matrix to Cardan (or Euler) angles.

Calling sequence:	`[q1, q2] = rtocarda(R, i, j, k)`
Return value:	`ROW3 - q1, q2.`
Input parameters:	`MAT - R; int - i, j ,k.`

Extracts the Cardan (or the Euler) angles from a rotation matrix **R**. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). The two solutions are stored in the three-element vectors **q1** and **q2**. `rtocarda` performs the inverse operation than `cardator`.

See also example 3.16

See also: `mtocarda`.

Example 3.16. *See sample program* `E_RTOCAR.M`.

The angular position of a generic reference frame (i) referred to frame (i-1) is expressed by the matrix

$$R = \begin{bmatrix} 0.840 & -0.395 & -0.371 \\ -0.415 & -0.029 & -0.909 \\ 0.348 & 0.918 & -0.189 \end{bmatrix}$$

The rotation sequence is made up of a rotation about axis Y, a rotation about axis X and a rotation about axis Z (anti-cyclic cardanic convention). The Cardan angles which perform the rotation from frame (i-1) to frame (i) are calculated by the statements

```
R=[0.840 -0.395 -0.371;
  -0.415 -0.029 -0.909;
   0.348 0.918 -0.189];
[q1, q2]=rtocarda(R, Y, X, Z);
```

The two solutions $q1$ and $q2$ are

$$q1 = [\,-2.042\ 1.141\ \ -1.641\,]$$

$$q2 = [\,1.100\ 2.001\ 1.501\,]$$

cardatom

Cardan angles to position matrix.

Calling sequence:	`m = cardatom(q, i, j, k, O)`
Return value:	`MAT4 - m.`
Input parameters:	`ROW3 - q; int - i, j, k; POINT - O.`

Builds the position matrix **m** of a frame whose origin is in point **O** and whose orientation is specified by an Euler/Cardanic convention. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q**: 3 element vector containing the 1st, 2nd and 3rd rotation angle.

See also example 3.17

See also: `cardator`, `mtocarda`.

Example 3.17. *See sample program* `E_CARDAM.M`.

The position matrix m of a frame whose origin is in the point O=[100 200 300 1]' which has q defined by a rotation of 1 rad about axis x, a second rotation of 2 rad about axis z and a third rotation of 1.5 rad about axis y is built by the following statements:

```
O=[100 200 300 1]';
q=[1 2 1.5];
m=cardatom(q, X, Z, Y, O);
```

The resulting matrix is:

$$m = \left[\begin{array}{ccc|c} -0.029 & -0.909 & -0.415 & 100 \\ 0.874 & -0.225 & 0.431 & 200 \\ -0.485 & -0.350 & 0.801 & 300 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

mtocarda

Position matrix to Cardan angles.

Calling sequence:	`[q1, q2] = mtocarda(m, i, j, k)`
Return value:	`ROW3 - q1, q2.`
Input parameters:	`MAT4 - m; int - i, j, k.`

Builds the Euler/Cardan angles, which specify the position of a frame whose position matrix is **m**. Both solutions are evaluated. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q1** and **q2** are 3 element vectors containing the 1^{st}, 2^{nd} and 3^{rd} rotation angle of the two solutions.

See also: `rtocarda`, `cardatom`.

3.5.2 Velocity and Acceleration

cardatow

Cardan angles to velocity matrix.

Calling sequence:	`W = cardatow(q, qp, i, j, k, O)`
Return value:	`MAT4 - W.`
Input parameters:	`ROW3 - q, qp; int - i, j ,k; POINT - O.`

Builds the velocity matrix **W** of a frame whose origin is **O** and whose orientation is specified by an Euler/Cardan convention. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing the 1^{st}, 2^{nd} and 3^{rd} angle. **qp** is a 3 element vector containing the time derivative of **q**. `cardatow` performs the inverse operation than `wtocarda`.

See also example 3.18

See also: `cardatoh`, `wtocarda`.

Example 3.18. *See sample program* `E_CARDAW.M`.

Let's consider a moving frame whose origin is in the point O=[50 10 100 1]'. The rotation sequence is made up by a rotation of 1 rad around axis x, one of 2.5 rad about axis z and a rotation of 0.9 rad about axis y. The time derivative qp of q is filled with the values 0.2, 4, 1 rad/s. The 4×4 velocity matrix W is built by the following statements

```
O =[50 10 100 1]';
q =[0.1 0.5 0.9];
qp=[0.2 0.4 0.1];
W=cardatow(q, qp, X, Z, Y, O);
```

The resulting matrix W is:

$$W = \left[\begin{array}{ccc|c} 0 & -0.407 & 0.047 & -0.671 \\ 0.407 & 0 & -0.152 & -5.132 \\ -0.047 & 0.152 & 0 & 0.849 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

wtocarda

Velocity matrix to Cardan angles.

Calling sequence:	`[q1, qp1, q2, qp2]= wtocarda(m, W, i, j, k)`
Return value:	`ROW3 - q1, qp1, q2, qp2.`
Input parameters:	`MAT4 - m, W; int - i, j, k.`

Builds the Euler/Cardan angles, first and second time derivative of a frame. It uses the velocity matrix **W** and the position matrix **m**. Both solutions are evaluated. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q1** ans **q2** are a 3 element vectors containing the two angles set. **qp1** and **qp2** are 3 element vectors containing the time derivative of **q1** and **q2** respectively.
NOTE: The first time derivative of Euler/Cardan angles is evaluated using the relation:

$$qpx = omega * Ainverse$$

where `qpx` can be either **qp1** or **qp2**. Internally called by `htocarda`. This function builds the transpose of the matrix find by `cardtowp`.

See also: `htocarda`, `cardatow`.

cardtoom

Cardan angles to angular velocity.

Calling sequence:	`omega = cardtoom(q, qp, i, j, k)`
Return value:	`COL3 - omega.`
Input parameters:	`ROW3 - q, qp; int - i, j, k.`

Evaluates the angular velocity of a moving frame from the three Cardan (or Euler) angles **q** and their time derivative **qp**. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing 1^{st}, 2^{nd} and 3^{rd} angle. **qp** is the time derivative of **q**. **omega** is a 3 element column vector containing the angular velocity.

See also example 3.19

See also: `cardtome`, `cardompt`.

Example 3.19. *See sample program* `E_CRD_OM.M`.

Consider a moving frame which has variable q defined by a rotation of 10 rad around axis z, a rotation of 5 rad around axis y and a rotation of 12 rad around axis x. The first time derivative of q is filled with the values [0 2 1] rad/s. The angular velocity omega is evaluated by the following statements

```
q=[10 5 12];
qp=[0 2 1];
omega=cardtoom(q, qp, Z, Y, X);
```

The resulting vector is:

$$omega = [0.850 - 1.8320.959]'$$

cardtome

Cardan angles to angular velocity matrix.

Calling sequence:	`A=cardtome(q, qp, qpp, i, j, k)`
Return value:	`MAT3 - A.`
Input parameters:	`ROW3 - q, qp, qpp; int - i, j, k.`

Evaluates the angular velocity matrix OMEGA. Equivalent to `cardtoom(q, qp, i, j, k, A);` but stores the angular velocity in a matrix **A**.

See also example 3.20

See also: `cardompt`.

Example 3.20. *See sample program* E_CRD_ME.M.

There are only a few differences between this example and example 3.19. The angular velocity omega is no more stored in a column vector because now it is stored in the 3×3 skew-symmetric upper-left submatrix of a matrix A. So, we have the statements

```
q =[10 5 12];
qp=[0 2 1];
A=cardtome(q, qp, Z, Y, X);
```

The resulting matrix is:

$$A = \begin{bmatrix} 0 & -0.959 & -1.832 \\ 0.959 & 0 & -0.850 \\ 1.832 & 0.850 & 0 \end{bmatrix}$$

cardatoh

Cardan angles to acceleration matrix.

Calling sequence:	`H = cardatoh(q, qp, qpp, i, j, k, O)`
Return value:	`MAT4 - H.`
Input parameters:	`ROW3 - q, qp, qpp; int - i, j, k; POINT - O.`

Builds the acceleration matrix **H** of a frame whose origin is **O** and whose orientation is specified by a Euler/Cardanic convention. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing the 1^{st}, 2^{nd} and 3^{rd} angle. **qp** is a 3 element vector containing the time derivative of **q**. **qpp** is a 3 element vector containing the 2^{nd} time derivative of **q**.

See also example 3.21

See also: `cardatow`, `htocarda`.

Example 3.21. *See sample program* E_CARDAH.M.

Let's consider example 3.18. The second time derivative of q, which is qpp, is filled with the values [0.5 1.2 0.3] rad/s^2. The acceleration matrix H of the frame is built by the following statements

```
O= [50 10 100 1]';
q= [0.1 0.5 0.9];
qp= [0.2 0.4 0.1];
qpp=[0.5 1.2 0.3];
H=cardatoh(q, qp, qpp, X, Z, Y, O);
```

The resulting matrix H is

$$H = \left[\begin{array}{ccc|c} -0.168 & -1.221 & 0.104 & 10.234 \\ 1.235 & -0.189 & -0.302 & -29.688 \\ 0.020 & 0.340 & -0.025 & -1.873 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

When O is put equal to [0 0 0 1]' this example gives the same resulting matrix of function `cardatog` (see example 3.22).

`htocarda`

Acceleration matrix to Cardan angles.

Calling sequence:	`[q1, q2, qp1, qp2, qpp1, qpp2]= htocarda(m, W, H, i, j, k)`
Return value:	`ROW3 - q1, q2, qp1, qp2, qpp1, qpp2.`
Input parameters:	`MAT4 - m, W, H; int - i, j, k.`

Builds the Euler/Cardan angles, first and second time derivative of a frame. It uses the acceleration matrix **H**, the velocity matrix **W** and the position matrix **m**. Both solutions are evaluated. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q1** and **q2** are a 3 element vectors containing the two angles set. **qp1** and **qp2** are a 3 element vectors containing the 1^{st} time derivative of **q1** and **q2** respectively. **qpp1** and **qpp2** are a 3 element vectors containing the 2^{nd} time derivative of **q1** and **q2**.

See also: `mtocarda`, `wtocarda`, `cardatoh`.

`cardatog`

Cardan angles to angular acceleration matrix.

Calling sequence:	`G = cardatog(q, qp, qpp, i, j, k)`
Return value:	`MAT3 - G.`
Input parameters:	`ROW3 - q, qp, qpp; int - i, j, k.`

Evaluates the angular acceleration matrix of a moving frame from the three Cardan (or Euler) angles **q** and their first and second time derivatives **qp** and **qpp**. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing the 1^{st}, 2^{nd} and 3^{rd} angle. **qp** is the first time derivative of **q**. **qpp** is the second time derivative of **q**. **G** is the matrix where the result must be stored.

$$G = \dot{\underline{\omega}} + \underline{\omega}^2. \tag{3.15}$$

See also example 3.22

Example 3.22. *See sample program* `E_CARDTG.M`.

This example is quite similar to example 3.23 (see also example 3.19 and example 3.20). The angular acceleration is now stored in the matrix A, so we have the following statements

```
q =[0.1 0.5 0.9];
qp =[0.2 0.4 0.1];
qpp=[0.5 1.2 0.3];
A=cardatog(q, qp, qpp, Y, X, Z);
```

The resulting matrix is:

$$A = \begin{bmatrix} -0.025 & 0.020 & 0.340 \\ 0.104 & -0.168 & -1.221 \\ -0.302 & 1.235 & -0.189 \end{bmatrix}$$

`cardompt`

Cardan angles to angular acceleration.

Calling sequence:	`omegapto = cardompt(q, qp, qpp, i, j, k)`
Return value:	`COL3 - omegapto.`
Input parameters:	`ROW3 - q, qp, qpp; int - i, j, k.`

Evaluates the angular acceleration of a moving frame from the three Cardan (or Euler) angles **q** and their first and second time derivatives **qp** and **qpp**. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing 1^{st}, 2^{nd} and 3^{rd} angle. **qp** is the first time derivative of **q**. **qpp** is the second time derivative of **q**. **omegapto** is a 3 element vector containing the angular acceleration.

See also example 3.23

Example 3.23. *See sample program* E_CARDPT.M.

Consider a moving frame whose origin is in the point O=[25 23 30 1]. It has variable q defined by a rotation of 1 rad around axis y, a rotation of 1.2 rad around axis x and a rotation of 3 rad around axis z. The first time derivative of q is filled with the values [0, 1, 0] rad/s, while the second time derivative is filled with the values [3 2.5 4.01] rad/s^2. The angular acceleration *omegapto* is evaluated by the following statements

```
q =[1 1.2 3];
qp =[0 1 0];
qpp=[3 2.5 4.01];
omegapto=cardompt(q, qp, qpp, Y, X, Z);
```

The resulting vector is:

$$omegapto = [2.573 - 0.737 - 1.319]'$$

cardatol

Cardan angles to L matrix.

Calling sequence:	`L = cardatol(q, i, j, k)`
Return value:	`MAT4 - L.`
Input parameters:	`ROW3 - q; int - i, j, k.`

Builds the *ISA*'s (*Instantaneous Screw Axis*) matrix **L** of a frame whose orientation is specified by an Euler/Cardan convention. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing the 1st, 2nd and 3rd angle. `cardantol` is internally called by `cardtoom` and `cardompt`.

See also: `cardtoom`, `cardompt`.

cardtowp

Calling sequence:	`R = cardtowp(ROW3 q, int i, int j, int k, int dim)`
Return value:	`MAT - R.`
Input parameters:	`ROW3 - q; int - i, j, k, dim.`

The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). **q** is a 3 element vector containing the 1st, 2nd and 3rd angle. `cardtowp` is internally called by `cardtoom` and `cardompt`.

inva

inverse a matrix A (Euler/Cardan velocities).

Calling sequence:	`[Ai, test] = inva(alpha, beta, sig, i, j, k)`
Return value:	`MAT4 - Ai; int - test.`
Input parameters:	`real - alpha, beta; int - sig, i, j, k.`

Function that builds the inverse of the matrix A. It is useful in order to evaluate the first time derivative of the Euler/Cardan angles. Input parameters: **alpha**, **beta**: the first two Euler/Cardan angles; **sig**: parameters that defines the sign of some elements in the inverse of A. The parameters **i**, **j**, **k** specify the sequence of the rotation axes (their value must be the constant `X`, `Y` or `Z`. See § 2.4.3). **j** must be different from **i** and **k**, **k** could be equal to **i** (see also table 3.2). Output parameters: **Ai**: it's the matrix where the inverse of A is stored. The function also return a value (`test`) that indicates if there are singular positions. Internally called by `wtocarda` and `htocarda`.

3.6 Construction of frames attached to points or vectors

framep

Frames from points.

Calling sequence:	`A = framep(P1, P2, P3, a1, a2)`
Return value:	`MAT3 - A.`
Input parameters:	`POINT - P1, P2, P3; int - a1, a2.`

Builds a rotation matrix describing the angular position of a frame attached to three points. The origin is in **P1**, axis **a1** points from **P1** toward point **P2**, axis **a2** from **P1** toward point **P3** (if possible). Axis **a1** has priority on **a2**. Axis **a1** is directed as (**P2-P1**). Axis **a3** is directed as (**P2-P1**)×(**P3-P1**). Axis **a2** is directed as axis(**a3**)×axis(**a1**). The axes **a1** and **a2** must be either the constant `X`, `Y` or `Z` defined in `spacelib.m` (see also § 2.4.3). **a1** must be different from **a2**. The rotation matrix is stored in the 3×3 upper-left part of the matrix **A**.

See also example 3.24

See also: `frame4p`.

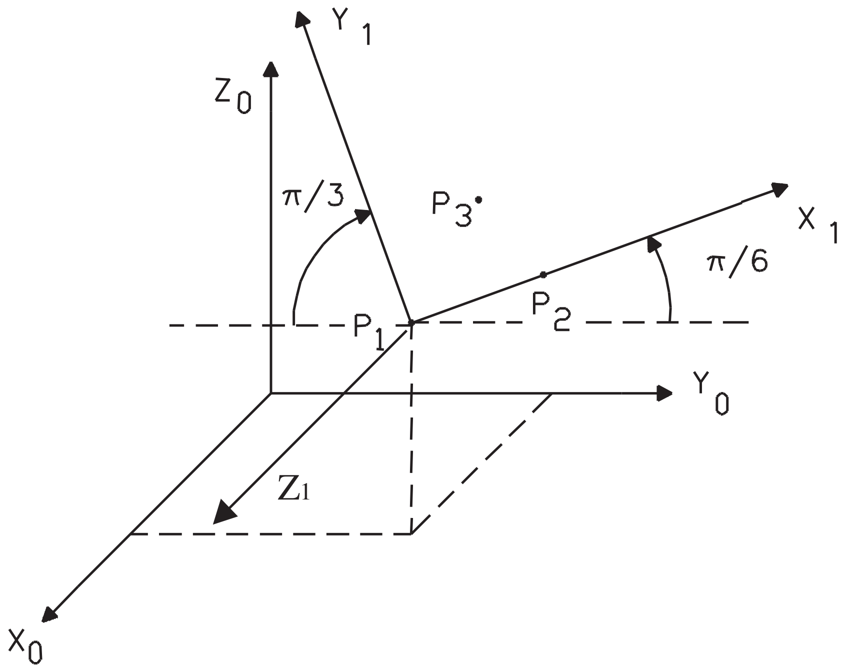


Figure 3.11: The frames used in example 3.24.

Example 3.24. *See sample program* `E_FRAMEP.M`.

In this example the three given points $P1$, $P2$ and $P3$ in the absolute frame (0) are used to build the frame (1) With the given values

$$P1 = [5\ 4\ 3\ 1]' \quad P2 = [5\ 6\ 4\ 1]' \quad P3 = [5\ 5\ 5\ 1]'$$

the angular position of reference frame (1) in figure 3.11, referred to frame (0), is expressed by the matrix:

$$R_{01} = \begin{bmatrix} 0 & 0 & 1 \\ 0.894 & -0.447 & 0 \\ 0.447 & 0.894 & 1 \end{bmatrix}$$

These matrices can be built by the statements:

```
P1=[5 4 3 1]';
P2=[5 6 4 1]';
P3=[5 5 5 1]';
R01=framep(P1, P2, P3, X, Y);
```

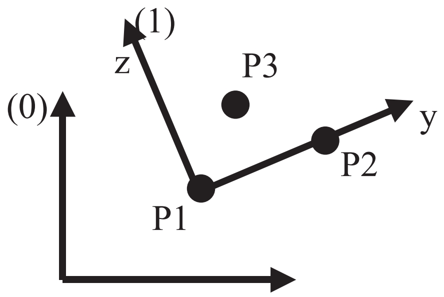


Figure 3.12: `frame4P` example of use: `M01=frame4P(P1,P2,P3,Y,Z)`

frame4p

Frame from three points.

Calling sequence:	`m = frame4p(P1, P2, P3, a1, a2)`
Return value:	`MAT4 - m.`
Input parameters:	`POINT - P1, P2, P3; int - a1, a2.`

Builds a 4×4 position matrix **m** describing the position and orientation of a frame attached to three points. The origin is put in point **P1**, axis **a1** points toward point **P2**, axis **a2** points toward point **P3** (if possible). Axis **a1** has priority on **a2**. The third frame axis is directed as **P2-P1**×**P3-P2**. The three axes form a right frame. More in details: Axis **a1** is directed as **(P2-P1)**. Axis **a3** is directed as **(P2-P1)**×**(P3-P1)** (cross product). Axis **a2** is directed as axis **a3**×axis **a1**. In other words the axis **a2** is chosen in such a way that it lies in the plane defined by the three points (see fig. 3.12). The axes **a1** and **a2** must be either the constant `X`, `Y` or `Z` defined in `spacelib.m` (see also § 2.4.3). **a1** must be different from **a2**.

See also example 3.25

See also: `framep`.

Example 3.25. *See sample program* `E_FRAM4P.M`.

Referring to example 3.24, the origin of frame (1) is in the point $P_1 = [5\ 4\ 3\ 1]'$. The position matrix $m_{0,1}$ of frame (1) is then:

$$m_{0,1} = \left[\begin{array}{ccc|c} 0 & 0 & 1 & 5 \\ 0.894 & -0.447 & 0 & 4 \\ 0.447 & 0.894 & 0 & 3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

This matrix is built by the following statements:

```
P1=[5 4 3 1]';
P2=[5 6 4 1]';
P3=[5 5 5 1]';
m01=frame4p(P1, P2, P3, X, Y);
```

framev

Frame from vectors.

Calling sequence:	`A = framev(v1, v2, a1, a2)`
Return value:	`MAT3 - A.`
Input parameters:	`ROW3 - v1, v2; int - a1, a2.`

Builds a rotation matrix describing the angular position of a frame attached to two vectors. Axis **a1** is directed as **v1**, axis **a2** is directed as **v2** (if possible). Axis **a1** has priority on **a2**. The third frame axis is directed as **v1**×**v2**. The three axes form a right frame. More in details: Axis **a1** is directed as **v1**. Axis **a3** is directed as **v1**×**v2** (cross product). Axis **a2** is directed as axis **a3**×axis **a1**. In other words the axis **a2** is chosen in such a way that it lies in the plane defined by the two vectors. Axes **a1** and **a2** must be either the constant `X`, `Y` or `Z` defined in `spacelib.m` (see also § 2.4.3). **a1** must be different from **a2**.

See also example 3.26

See also: `frame4v`.

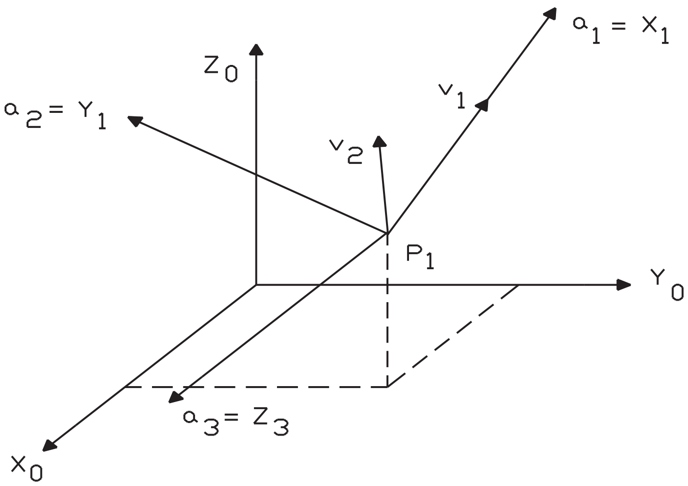


Figure 3.13: Frames used in example 3.26 and 3.27.

Example 3.26. *See sample program* `E_FRAMEV.M`.

In this example a point $P1$ and two given vectors $v1$ and $v2$ in the absolute frame (0) are used to build the frame (1). Referring to example 3.24 these vectors are:

$$v1 = P2 - P1 \qquad v2 = P3 - P1$$

The angular position of reference frame (1) in figure 3.13, referred to frame (0), is expressed by the matrix

$$R_{0,1} = \begin{bmatrix} 0 & 0 & 1 \\ 0.894 & -0.447 & 0 \\ 0.447 & 0.894 & 0 \end{bmatrix}$$

This rotation matrix can be built by the statements:

```
v1=[0 2 1];
v2=[0 1 2];
r01=framev(v1, v2, X, Y);
```

`frame4v`

Frame from a point and two vectors.

Calling sequence:	`m = frame4v(P1, v1, v2, a1, a2)`
Return value:	`MAT4 - m.`
Input parameters:	`POINT - P1; ROW3 - v1, v2; int - a1, a2.`

Builds a 4×4 position matrix **m** describing the position and orientation of a frame attached to two vectors and one point. The origin is put in point **P1**, axis **a1** is directed as vector **v1**, axis **a2** is directed as **v2** (if possible). Axis **a1** has priority on **a2**. The third axis is directed as **v1**×**v2**. The second axis is directed as axis **a3**×axis **a1** to form a right frame. More in details: Axis **a1** is directed as **v1**. Axis **a3** is directed as **v1**×**v2** (cross product). Axis **a2** is directed as axis **a3**×axis **a1**. In other words the axis **a2** is chosen in such a way that it lies in the plane defined by the three points. Axes **a1** and **a2** must be either the constant `X`, `Y` or `Z` defined in `spacelib.m`(see also § 2.4.3). **a1** must be different from **a2**.

See also example 3.27

See also: `framev`.

Example 3.27. *See sample program* `E_FRAM4V.M`.

Referring to Example 3.26, the origin of frame (1) is in the point P_1=[5 4 3]'. The position matrix $m_{0,1}$ of frame (1) is then:

$$m_{0,1}=\left[\begin{array}{ccc|c} 0 & 0 & 1 & 5 \\ 0.894 & -0.447 & 0 & 4 \\ 0.447 & 0.894 & 0 & 3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

This matrix is built by the statements

```
P1=[5 4 3 1]';
v1=[0 2 1];
v2=[0 1 2];
m01=frame4v(P1, v1, v2, X, Y);
```

`aaxis`

Axis of Frame.

Calling sequence:	`A = aaxis (n);`
Return value:	`AXIS A.`
Input parameters:	`int n.`

Returns a 3 elements unit vector **A** parallel to a frame axis **n**. **n** must be either the constant `X`, `Y` or `Z` defined in `spacelib.h` (see also § 2.4.3). For example `a=aaxis(Y)` returns $a = [\ 0.\ 1.\ 0.\ ]$.

3.7 Working with points, lines and planes

3.7.1 Operations on points

`angle`

Angle between points.

Calling sequence:	`alpha = angle(P1, P2, P3)`
Return value:	`real - alpha.`
Input parameters:	`POINT - P1, P2, P3.`

Function returning the angle between three points which is the angle between vectors (**P1-P2**) and (**P3-P2**).

dist

Distance between two points.

Calling sequence:	`d = dist(P1, P2)`
Return value:	`real - d.`
Input parameters:	`POINT - P1, P2.`

Obsolete version. Please use function `distp`.

distp

Distance between two points.

Calling sequence:	`d = distp(P1, P2)`
Return value:	`real - d.`
Input parameters:	`POINT - P1, P2.`

Function returning the distance between two points.

intermed

Intermediate point.

Calling sequence:	`P = intermed(P1, a, P2, b)`
Return value:	`POINT - P`
Input parameters:	`POINT - P1, P2; real - a, b.`

Evaluates point **P** as the middle point between points **P1** and **P2** using two weights **a** and **b**. We have

$$P = \frac{a \cdot P1 + b \cdot P2}{a + b} \tag{3.16}$$

When `a=b=1` the function `intermed` is equivalent to function `middle`. If $a + b$ is equal to zero, it is assumed that $a + b = 1$.

See also: `middle`.

middle

Middle point.

Calling sequence:	`P = middle(P1, P2)`
Return value:	`POINT - P.`
Input parameters:	`POINT - P1, P2.`

Evaluates point **P** as the middle point between points **P1** and **P2**:

$$P = \frac{1}{2}(P1 + P2) \tag{3.17}$$

vect

Vector between points.

Calling sequence:	`v = vect(P1, P2)`
Return value:	`COL3 - v.`
Input parameters:	`POINT - P1, P2.`

Function evaluating the vector **v=P1-P2** (from **P2** toward **P1**).

3.7.2 Operations on lines and planes

line2p

Line from two points.

Calling sequence:	`l = line2p(P1, P2)`
Return value:	`LINE - l.`
Input parameters:	`POINT - P1, P2.`

Builds a line passing from points **P1** and **P2**.

See also: `linpvect`.

linpvect

Line from point and vector.

Calling sequence:	`l = linpvect(P1, v)`
Return value:	`LINE - l.`
Input parameters:	`POINT - P1; COL3 - v.`

Builds a line which passes through point **P1** and having the same direction as vector **v**.

See also: `line2p`.

intersec

Intersection between two lines.

Calling sequence:	`[lmindist, mindist, pl, I, inttype]= intersec(l1, l2)`
Return value:	`LINE - lmindist; real - mindist; PLANE - pl; POINT - I; int - inttype.`
Input parameters:	`LINE - l1, l2.`

Function evaluating the intersection point **I** between lines **l1** and **l2**. This function builds also, when possible, the minimum distance line **lmindist** and a plane **pl** containing **l1** and **l2**. The parameter **inttype** defines whether an intersection point was found or not. **inttype** may have the following values:

1 **l1** and **l2** are oblique lines. **I** is the middle point on the minimum distance line. **pl** does not really contain the two lines.

0 **l1** and **l2** have exactly one intersection point **I**. The minimum distance **mindist** is zero. **pl** contains both **l1** and **l2**.

-1 **l1** and **l2** are the same line. The intersection of the two is the line itself. **pl** can't be built.

2 **l1** is parallel to **l2** (no intersection). **pl** contains both **l1** and **l2**.

See also: `interlpl`, `inter2pl`.

projponl

Projection of point on a line.

Calling sequence:	`[I, dist] = projponl(P1, l)`
Return value:	`POINT - I; real - dist.`
Input parameters:	`POINT - P1; LINE - l.`

Finds the projection **I** of point **P** on line **l**. This function returns also the distance of **P** from **l**.

See also example 3.28

See also: `project`.

Example 3.28. *See sample program* E_PROJPO.M.

The given line l has the origin in point $O = [3\ 7.2\ 2.05\ 1]$' and its direction is $[0.7\ 4\ 9]$. If the line m which is orthogonal to l and passes through the point $P = [5\ 1\ 3\ 1]$' has to be found, this is performed by means of the following statements:

```
l=[3 0.7; 7.2 4; 2.05 9; 1 0];
P=[5 1 3 1]';
[P1, dist]=projponl(l, P);
```

```
l2=line2p(P, P1);
printm('The origin of the new line is:',l2(:, 1))
printm('The direction of the new line is:', l2(:, 2))
```

which produces the following result:

```
The origin of the new line is:  5 1 3 1
The direction of the new line is:  -0.3287 0.8723 -0.3621
```

distpp

Distance of point from a plane.

Calling sequence:	`dist = distpp(pl, P)`
Return value:	`real - dist.`
Input parameters:	`PLANE - pl; POINT - P.`

Evaluates the distance **dist** of point **P** from plane **pl**.

project

Project a point on a plane.

Calling sequence:	`[I, dist] = project(P, pl)`
Return value:	`POINT - I; real - dist.`
Input parameters:	`POINT - P; PLANE - pl.`

Finds the projection **I** of point **P** on plane **pl**. This function returns also the distance **dist** of **P** from **pl**.

See also: `projponl`.

plane

Plane from three points.

Calling sequence:	`Pl = plane(P1, P2, P3)`
Return value:	`PLANE - Pl.`
Input parameters:	`POINT - P1, P2, P3.`

Builds a plane **pl** which contains the three given points **P1**, **P2** and **P3**. **pl** is defined by four elements, the three components in the reference frame of the unit vector orthogonal to the plane itself and the distance of **pl** from the origin of reference frame (considered with the sign).

See also: `plane2`.

plane2

Plane from point and vector.

Calling sequence:	`Pl = plane2(P1, v)`
Return value:	`PLANE - Pl.`
Input parameters:	`POINT - P1; ROW3 - v.`

Builds a plane **pl** which contains point **P1** and is directed as vector **v**.

See also: `plane`.

inter2pl

Intersection of two planes.

Calling sequence:	`l = inter2pl(pl1, pl2)`
Return value:	`LINE - l.`
Input parameters:	`PLANE - pl1, pl2.`

Function evaluating the intersection between two planes **pl1** and **pl2**. The line type is filled with the resulting value.

See also: `intersec`.

interlpl

Intersection of line and plane.

Calling sequence:	`[I, inttype] = interlpl(l, pl)`
Return value:	`POINT - I; int - inttype.`
Input parameters:	`LINE - l; PLANE - pl.`

Function evaluating the intersection point **I** between line **l** and plane **pl.** The parameter **inttype** defines whether an intersection point was found or not. **inttype** has the following values:

1: line **l** lies on plane **pl** and the intersection of the two is the line itself.
-1: line **l** is parallel to plane **pl** (no intersection).
0: line **l** and plane **pl** have exactly one intersection point **I**.

See also: intersec.

Example 3.29. *See sample program* E_INTRLP.M.

Let's consider a plane pl = [0 0 1 -5] and a direction dir = [0 0 1]. If the symmetric point of P = [0 6 10 1]' with respect to plane pl in the direction dir has to be found, this is performed by means of the following statements

```
l=zeros(4, 2);
pl =[0 0 1 -5];
dir=[0 0 1 0]';
P = [0 6 10 1]';
l(:, 1)=P;                      % line l by POINT p directed as dir
l(:, 2)=dir;
[P1, inttype]=interlpl(l, pl); % P1=intersection between l and pl
d=distp(P, P1);                 % d=distance between P and P1
v=vector(dir,d);                % v=vector with direction dir
v(U)=0;                         % and module 'd'
Ps=P1-v;                        % Ps=P1-v
Ps(U)=1;                        % 4th homogeneous coord.of Ps
printm('The point P is:', Ps)  % Ps is the searched point
```

which gives the following result:

$$P = [0\ 6\ 0\ 1]'$$

Most of the functions described in this section are not really necessary in the MATLAB© version of SpaceLib©. They are supplied just for compatibility with the C version.

3.8 Operations on matrices and vectors

Many of the functions described in this section are not really necessary in MATLAB©; but they are supported for compatibility reasons with the C version of SpaceLib .

3.8.1 Matrices and vectors algebra

molt

Matrix multiplication.

Calling sequence:	`C = molt(A, B)`
Return value:	`MAT - C.`
Input parameters:	`MAT - A, B.`

Function not really necessary in MATLAB©: provided just for compatibility with the C *version of* SpaceLib©. Evaluates the matrix product **C=A** $\cdot$ **B** of generic matrices with the appropriate numbers of rows and columns. The product is directly evaluated by MATLAB , simple typing

```
C=A*B
```

however is possible to evaluate matrix product with this function.

rmolt

Multiply a scalar r by a matrix.

Calling sequence:	`B = rmolt(A, r)`
Return value:	`MAT - B.`
Input parameters:	`MAT - A; real - r.`

Function not really necessary in `MATLAB`©*: provided just for compatibility with the* `C` *version of* `SpaceLib`©. Evaluates the matrix **B** elements as a product of a matrix **A** and a scalar **r** (**B=r · A**). **A** and **B** can be the same matrix, so the following call is legal

```
A=rmolt(A, r);
```

and the result is **A=r · A**.

ssum

Sum of matrices

Calling sequence:	`C = ssum(A, B)`
Return value:	`MAT - C.`
Input parameters:	`MAT - A, B.`

Function not really necessary in `MATLAB`©*: provided just for compatibility with the* `C` *version of* `SpaceLib`©. Evaluates the matrix sum of matrices or vectors having every dimensions **C=A+B**. **A** can be equal to **B** and **C**. The sum of two matrices can be directly evaluated by `MATLAB` , simply typing

```
C=A+B
```

However is possible to evaluate matrix product with this function. For instance the following calls are legal:

```
B =sum(A, B);
```

the result is **B=B+A**.

```
B =sum(A, A);
```

the result is **B**=2 · **A**.

sub

Subtraction (for matrices).

Calling sequence:	`C = sub(A, B)`
Return value:	`MAT - C.`
Input parameters:	`MAT - A, B.`

Function not really necessary in `MATLAB`©*: provided just for compatibility with the* `C` *version of* `SpaceLib`©. Evaluates the matrix difference of matrices or vectors having **d1** rows and **d2** columns **C=A-B**. **A** can be equal to **B** and/or **C**. For instance the following call is legal:

```
B=sub(A, B);
```

the result is **B=A-B**.

3.8.2 General operations on matrices

crossmto

Cross product for matrices.

Calling sequence:	`B = crossmto(A1, A2)`
Return value:	`MAT - B.`
Input parameters:	`MAT -A1, A2.`

Obsolete version. Please use function `crosstom`.

Function not really necessary in `MATLAB`©*: provided just for compatibility with the* `C` *version of* `SpaceLib`©. This function performs the operation:

$$B = A2^t \cdot A1 - A1^t \cdot A2$$

If matrices **A1** and **A2** express two vectors $\vec{a1}$ and $\vec{a2}$ in the matricial form, this operation is equivalent to their cross product $\vec{b} = \vec{a1} \times \vec{a2}$ and $\vec{b}$ is stored in matrix **B**.

crosstom

Cross product for matrices.

Calling sequence:	`B = crosstom(A1, A2)`
Return value:	`MAT - B.`
Input parameters:	`MAT or COL3 - A1, A2.`

Function not really necessary in MATLAB©: provided just for compatibility with the `C` *version of* `SpaceLib`©. This function performs the operation:

$$B = A2 \cdot A1^t - A1 \cdot A2^t$$

If matrices **A1** and **A2** express two vectors $\vec{a1}$ and $\vec{a2}$ either in the matricial[3] or vectorial form, this operation is equivalent to their cross product $\vec{b} = \vec{a1} \times \vec{a2}$ and $\vec{b}$ is stored in matrix **B**. **A1** and **A2** must be of the same type: both column vectors or both 3×3 matrices.

$$A1 = \begin{bmatrix} 0 & -a1_z & a1_y \\ a1_z & 0 & -a1_x \\ -a1_y & a1_x & 0 \end{bmatrix} \quad or \quad \begin{bmatrix} a1_x \\ a1_y \\ a1_z \end{bmatrix} \qquad A2 = \begin{bmatrix} 0 & -a2_z & a2_y \\ a2_z & 0 & -a2_x \\ -a2_y & a2_x & 0 \end{bmatrix} \quad or \quad \begin{bmatrix} a2_x \\ a2_y \\ a2_z \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & -b_z & b_y \\ b_z & 0 & -b_x \\ -b_y & b_x & 0 \end{bmatrix} = \begin{bmatrix} 0 & a1_y\, a2_x - a1_x\, a2_y & a1_z\, a2_x - a1_x\, a2_z \\ a1_x\, a2_y - a1_y\, a2_x & 0 & a1_z\, a2_y - a1_y\, a2_z \\ a1_x\, a2_z - a1_z\, a2_x & a1_y\, a2_z - a1_z\, a2_y & 0 \end{bmatrix}$$

clearmat

Clear a matrix (fill it with zeros).

Calling sequence:	`A= clearmat(m, n)`
Return value:	`MAT - A.`
Input parameters:	`int - m, n.`

Function not really necessary in MATLAB©: provided just for compatibility with the `C` *version of* `SpaceLib`©. Fills with zeros a **m**×**n** matrix **A**. Equivalent to the `MATLAB` statement

```
zeros(m,n)
```

idmat

Identity matrix.

Calling sequence:	`A = idmat(id)`
Return value:	`MAT - A.`
Input parameters:	`int - id.`

Function not really necessary in MATLAB©: provided just for compatibility with the `C` *version of* `SpaceLib`©.Makes unitary square matrix **A** having **id**×**id** dimension. Equivalent to the `MATLAB` statement

```
eye(id,id)
```

.

transp

Transpose of a matrix.

Calling sequence:	`At = transp(A)`
Return value:	`MAT - At.`
Input parameters:	`MAT - A.`

Function not really necessary in MATLAB©: provided just for compatibility with the `C` *version of* `SpaceLib`©. Builds the transpose **At** of a matrix **A** . This function is implemented using the `MATLAB` opera-tor ‘.

[3] A vector $\overrightarrow{v}$ could be represented in the following forms [3], [4], [2]:

$$v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \qquad \underline{v} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

pseudinv

Pseudo inverse of a matrix.

Calling sequence:	`Api = pseudinv(A)`
Return value:	`MAT - Api.`
Input parameters:	`MAT - A.`

Function not really necessary in `MATLAB`$^{©}$*: provided just for compatibility with the* `C` *version of* `SpaceLib`$^{©}$. Builds the pseudo-inverse matrix **Api** of a given matrix **A**. This function uses the `MATLAB` function `pinv` that evaluates the pseudoinverse matrix.

```
X = pseudinv(A)
```

produces a matrix `X` of the same dimensions as `A'` so that

$$A \cdot X \cdot A = A, \qquad X \cdot A \cdot X = X$$

and $A \cdot X$ and $X \cdot A$ are *Hermitian*. The computation is based on `SVD(A)` and any singular values less than a tolerance are treated as zero.

3.8.3 General operations on vectors

cross

Cross product. -obsolete-

Calling sequence:	`c = cross(a, b)`
Return value:	`COL3 - c.`
Input parameters:	`COL3 - a, b.`

Evaluates the cross product of two 3×3 vectors (**c**=**a**×**b**). Removed because equivalent to the `MATLAB` function `cross`.

dot

Dot product. -obsolete version of `dot3`

Calling sequence:	`c = dot(a, b)`
Return value:	`ROW3 - c.`
Input parameters:	`ROW3 - a, b.`

The `MATLAB` function `dot` performs the dot product of vectors of any dimension.

dot3

Dot product for 3 element vectors.

Calling sequence:	`c = dot3(a, b)`
Return value:	`real - c.`
Input parameters:	`ROW3 - a, b.`

Returns the value of the dot product of two 3 element vectors **a** and **b** ($c = a^t b$).

dot2

Dot product - version 2.

Calling sequence:	`c = dot2(a, b)`
Return value:	`real - c.`
Input parameters:	`ROW - a, b.`

Returns the value of the dot product of two vectors **a** and **b** ($c = a^t b$).

mod

Module of a vector.

Calling sequence:	`n = mod(a)`
Return value:	`real - n.`
Input parameters:	`VECTOR - a.`

Obsolete version. Please use function `modulus`.

modulus

Module of a vector.

Calling sequence:	`n = modulus(a)`
Return value:	`real - n.`
Input parameters:	`VECTOR - a.`

Returns the module of vector **a** ($|a|$). Not really useful in MATLAB . Supported only for compatibility with the C-language version.

unitv

Unit vector.

Calling sequence:	`[u, t]= unitv(v)`
Return value:	`COL3 - u; real - t.`
Input parameters:	`COL3 - v.`

Extracts the unit vector **u** of a vector **v** (**u**=**v**/|**v**|) and returns the module of the vector.
If v=[0 0 0] is u=[0 0 0].

vector

Evaluate a vector (from module and direction).

Calling sequence:	`v = vector(u, mod)`
Return value:	`COL3 - v.`
Input parameters:	`COL3 - u; real - mod.`

Evaluates a vector **v** which has **mod** as module and **u** as unit vector (**v**=**mod** · **u**).

3.9 Copy functions

mcopy

Matrix copy.

Calling sequence:	`A2 = mcopy(A1)`
Return value:	`MAT - A2.`
Input parameters:	`MAT - A1.`

Copies a matrix **A1** into **A2** ($A2 = A1$). *Function not really necessary in MATLAB©: provided just for compatibility with the C version of SpaceLib©.*

See also: `mmcopy`, `mvcopy`.

mmcopy

Copy a part of a matrix.

Calling sequence:	`B = mmcopy(A, im, jm)`
Return value:	`MAT - B.`
Input parameters:	`MAT - A; int - im, jm.`

Copies the **im**×**jm** upper-left part of matrix **A** into matrix **B**. *Function not really necessary in MATLAB©: provided just for compatibility with the C version of SpaceLib©.*

See also: `mcopy`, `mvcopy`.

3.10 Print functions

fprintm

Print a matrix (with a comment) on a file.

Calling sequence:	`fprintm(out, s, A)`
Input parameters:	`file - out; string - s; MAT - A.`

Prints in a file a matrix **A** preceded by the comment contained in string, **out** is the file descriptor. If the file pointer **out** is put equal to 1, the standard output is the screen. The function can also be used to print a vector, which is handled like a particulary case of a matrix.

See also: `printm`.

printm

Print a matrix (with a comment) on the screen.

Calling sequence:	`printm(s, A)`
Input parameters:	`string - s; MAT - A.`

Prints only on the screen a matrix **A** preceded by the comment contained in **s**.

prmat

Print a position matrix for GRP_man graphics post-processor

Calling sequence:	`prmat (grpout, string, m)`
Input parameters:	`file - grpout; string - s; MAT4 - m.`

Writes to a file a position matrix **m** with the convention of GRP_MAN graphics post processor. Matrix **m** is written into file **grpout** preceded by string **string**.

Chapter 4

Linear System and inverse of matrices

Three functions are supplied for the resolution of linear systems: `solve_l`, `minvers` and `linears`.

- `solve_l` is useful in standard situations: square non singular coefficient matrix and a single right-hand vector.
- `minvers` evaluates the inverse of a matrix solving a particular system.
- `linears` is much more general, it deals also with rectangular coefficient matrices and more than one right-hand vectors to be handled at once.

To evaluate the pseudo-inverse of a matrix please use the `MATLAB`© function `pinv` (see § 3.8.2). In the `MATLAB`© version of `SpaceLib`© the name of the function `linear` has been change to `linears`; the change of the name has been forced to avoid conflicts with the "standard" `MATLAB`© function `linear`. For similar reasons function `solve` has been renamed `solve_l`.

4.1 Function `solve_l`

Function not really necessary in `MATLAB`©: provided just for compatibility with the `C` version of `SpaceLib`©.

4.1.1 General description

Function `solver` is useful to evaluate the solution of a linear system in the form

$$A \cdot x = b \tag{4.1}$$

where A is a square full rank matrix and b is the right-hand side vector. The direct manipulation of vectors and matrices in `MATLAB`© is very useful in this case and gives the solution simply using the equation:

$$x = \frac{b}{A} \qquad x = A^{+}b \tag{4.2}$$

which makes use of the pseudo-inverse. As an alternative it is possible to use the statement

```
x=pinv(A)*b
```

Function `solve_l`also return the rank of the matrix A, that could be rectangular. This means that the user will be able to solve the most common system class elements without using the more complex function `linears`.

4.1.2 Calling list

The calling list for this function is:

solve_l

Solve linear sistem.

Calling sequence:	`[x, irank] = solve_l(A, b)`
Return value:	`ROW - x; int - irank.`
Input parameters:	`MAT - A; COL - b.`

A is the matrix of the coefficients, **b** is the right-hand side vector, **x** is the system solution and **irank** is the rank of the matrix A.

4.2 Function minvers

Function not really necessary in MATLAB©: provided just for compatibility with the C *version of* SpaceLib©.

4.2.1 General description

Function `minvers`, can be considered a particularization of the more general function `linears`. So, `minvers` finds the inverse of a square full rank matrix A solving a particular system. It uses the general property that, whenever the rank of A is equal to its dimensions, the equation

$$A \cdot x = I \tag{4.3}$$

where I is the identity matrix, has one single solution. This solution is the inverse of A. In this case, we utilize the direct manipulation of vectors and matrices in MATLAB©, that gives the solution simply by the statement

```
x=A^ (-1)
```

or

```
x=inv(A)
```

4.2.2 Calling list

The calling list for the function `minvers` is

minvers

Calling sequence:	`MAT - A; int - dim.`
Return value:	`MAT - Ai.`

A is a **dim**×**dim** initial matrix and **Ai** is the inverse matrix.

4.3 Function linears

4.3.1 General description

This section contains information about function `linears`. This function allows the solution of a linear system by using a double pivoting elimination method. The linear system must be in the form

$$A \cdot x = b \tag{4.4}$$

where A is the matrix of coefficients and b is the right-hand side. A is generally a square `n`×`n` matrix, while b is an `n` element column vector. To use function `linear` in order to evaluate vector x, both A and b must be memorized in the same matrix (i.e. matrix H). More than one system with the same matrix A can be solved at the same time; for instance, the following systems can be handled at once:

$$A \cdot x_1 = b_1 \qquad A \cdot x_2 = b_2 \qquad A \cdot x_3 = b_3 \qquad \ldots \qquad A \cdot x_h = b_h \qquad \ldots \qquad A \cdot x_k = b_k \tag{4.5}$$

To solve the systems, matrix A and vectors b_i (at least one must be present) must be stored in the same `r`×`c` matrix H according to the scheme of figure 4.1. During the solution of the system, matrix A is replaced by an identity matrix and vectors b_i are replaced by the solutions of the correspondent system (i.e. x_i replaces b_i). However the order of the elements of each x_i is changed by the double pivoting algorithm and so the solution vectors must be reordered (see § 4.3.2 and § 4.3.4).

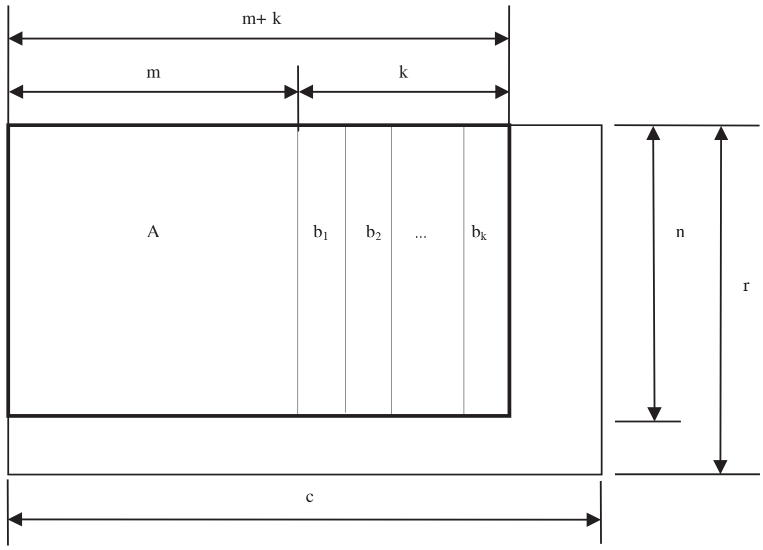


Figure 4.1: Definitions for parameters of function `linears`.
Generally `m`=`n`, `k`=1; always `r`≥`n`, `c`≥`m`+`k`.

4.3.2 Calling list

The calling list for this function is

linears

Solve linear system.

Calling sequence:	`[H, ivet, irank, arm] = linears(H, idim, jdim, imax, jmax, nsol, vpr)`
Return value:	`MAT - H; ROW - ivet; int - irank; real - arm.`
Input parameters:	`MAT - H; int - idim, jdim, imax, jmax, nsol; ROW - vpr.`

H: matrix containing matrix A and vectors b_i.

idim, **jdim**: the physical dimensions of H (`r` and **c** in figure 4.1).

imax, **jmax**: the logical dimensions of A (`n` and **m** in figure 4.1).

nsol: the number of the right-hand vectors (`k` in figure 4.1). In the case of figure 4.1 the matrix has the size `imax`×`jmax` but the data uses just a smaller part of it (`idim`×`jdim`) where of course `idim`≤`imax` and `jdim`≤`jmax`; the other part of the matrix is unused.

ivet: vectors of `m` integers that gives information necessary to reorder the elements of x. `ivet[i]+1==k` means that the value of the `k`th element of x is stored in position `i` of b. A standard way to reorder the solution putting the solution of the `k`th system ($k = 1, 2, \ldots$) in a vector x is as follows:

```
for i=1:1:n x(ivet(i)+1)=H(i, n+k); end
```

irank: an estimation of the rank of matrix A.

arm: the absolute value of the greater element of A during the last pass of elimination.

vpr: vector specifying which variables must be considered as *main* variables. That means that they will be forced in the first positions of vector **ivet** and so of vectors x. The list of the variables must be terminated by a value -1. So, for instance if `m`≥8, then a valid value for **vpr** is `[3 5 0 7 -1]`. In the usual cases, **vpr** is a vector containing only one element whose value is -1:

```
vpr = -1
```

This allows the algorithm to perform full pivoting on all the variables.

NOTE: function `linears` can be also used to detect if a general linear system has or has not solution. In this case matrix A can have a number of columns which is different from the number of the rows (`n`≠`m`). In this case function `linears` transforms the system into an equivalent system (i.e. which has the same solutions). After the execution of `linears` matrix A and vectors b_i will be in the block form of figure 4.2.

ivet	v_1	v_2	v_3	v_4	v_5	v_6	v_7	vector **b**
matrix **A**	1	0	0	0	0	y	y	b_1
	0	1	0	0	0	y	y	b_2
	0	0	1	0	0	y	y	b_3
	0	0	0	1	0	y	y	b_4
	0	0	0	0	1	y	y	b_5
	0	0	0	0	0	0	0	b_6
	0	0	0	0	0	0	0	b_7

Figure 4.2: Final form of a linear system after a call to function `linears` with a 7×7 matrix.

0, 1	are elements whose value is 0 or 1.
y	are elements whose value depends on the system.
$\mathbf{b}_i$	are elements of vector **b**.
$\mathbf{v}_i$	are elements of vector **ivet**.

Table 4.1: Notation used for figure 4.2

All the blocks can have any dimension and may not be present depending on the coefficients stored in matrix A. If in the last rows of the matrix the block of zeros is present, generally the system has not solution (over determined system) unless the last elements of b after the transformation are null (the elements correspondent to the block of zeros, that is b_6 and b_7 in Figure 4.2). If the block of y is present but the block of zeros is not present, the system has an infinite number of solutions that can be found assigning an arbitrary value at the last elements of x (the elements corresponding to the block of y, that is v_6 and v_7 in Figure 4.2).

As an example if the system is as follows:

$$A = \begin{bmatrix} 1 & 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 2 & 1 \\ 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 2 & 2 & 1 \end{bmatrix}; \quad X = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}; \quad b = \begin{bmatrix} 6 \\ 5 \\ 3 \\ 3 \end{bmatrix} \tag{4.6}$$

Function linear detects that matrix A has a rank of 3 and transforms it in the form of figure 4.3. That means that the rank of A is 3, the system has an infinitive number of solutions and that you can assign any value to elements 1 and 4 of x (a_1 and a_4) and then evaluate the corresponding value of the others (a_0, a_2 and a_3). If you put a_1=a_4=0, then the value of the other elements are directly stored in b ($a_3 = 3.5, a_0 = 1, a_2 = -2$). If a_1=7 and a_4=4 then a_3=-5.5, a_0=1 and a_2=5.

	a_3	a_0	a_2	a_1	a_4	**b**
ivet	3	0	2	1	4	
	1.000	0.000	0.000	1.000	0.500	3.500
	0.000	1.000	0.000	0.000	0.000	1.000
	0.000	0.000	1.000	−1.000	0.000	−2.000
	0.000	0.000	0.000	0.000	0.000	0.000

Figure 4.3: Numerical example of the output of function `linears`.

4.3.3 Sample program to solve a linear system `TEST-LIN`

As an example of the usage of function `linears`, let us consider the sample program `TEST-LIN` which reads and solves a linear system whose matrix of coefficient and whose right-hand side vector are memorized in file

INP.DAT (See § 4.3.4). The program reads the dimension of the matrix, the matrix itself and the vector and prints the solution if it exists. If the matrix is not a square full rank matrix, the program prints the resulting matrix and vector after the transformation.

The input matrix A and vector b for the example just presented are

$$A = \begin{bmatrix} 1 & 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 2 & 1 \\ 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 2 & 2 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 6 \\ 5 \\ 3 \\ 3 \end{bmatrix} \tag{4.7}$$

while file INP.DAT is filled with the values listed in table 4.2.

FILE INP.DAT (*DATA_FILE*)						*MEANING*
4	5					Matrix dimensions
1	1	1	2	1	6	value of A and b
0	1	1	2	1	5	
0	0	2	2	1	3	
0	0	2	2	1	3	

Table 4.2: Content of the file INP.DAT

4.3.4 The program (TEST-LIN.M)

```
%_____________________________________________________________________________________________
%                                        TEST-LIN.M
% Sample program which reads and solves a linear system whose matrix of coefficient and whose
% right-hand side vector are memorized in the file INP.DAT
%_____________________________________________________________________________________________

spheader
Nmax=6;
Mmax=7;
H=zeros(Nmax,Mmax+1);
A=zeros(Nmax,Mmax);
b=zeros(Nmax,1);
x=zeros(Mmax,1);
ivet=zeros(Mmax,1);
vpr=zeros(1);
vpr(1)=-1;
clc                          % Read matrix of coefficients and right-hand side vector from file
fid=fopen('inp.dat','r');
if (fid==-1)
        error('Error: unable to open input file in TEST_LIN.M ')
end
n=fscanf(fid,'%d',1);                                             % Read dimension of the matrix
m=fscanf(fid,'%d',1);
if (n>Nmax)|(m>Mmax)
        error(' Error in TEST_LIN.M : The matrix is too big ')
end
c=1;                                                              % Read matrix and vector
for i=1:1:n
        for j=1:1:m
                t(c)=fscanf(fid,'%f',1);
                A(i,j)=t(c);
        end
t(c)=fscanf(fid,'%f',1);
b(i)=t(c);
c=c+1;
end
```

```
for i=1:1:n                                                     % Copy matrix of coefficients
        for j=1:1:m
                H(i,j)=A(i,j);
        end
end
for j=1:1:n                                                     % Copy right-hand side vector
        H(j,m+1)=b(j);
end
fprintf(1,'\n   Matrix and right-and side vector\n\n');
for i=1:1:n
        for j=1:1:m+1;
                fprintf(1,'%7.3f',H(i,j));
        end
fprintf(1,'\n');
end
[H,ivet,irank,arm]=linears(H,Nmax,Mmax+1,n,m,1,vpr);            % Solve the system
if (n==m) & (irank==n)
        fprintf(1,'The solution is: \n');
        for i=1:1:n
                x(ivet(i)+1)=H(i,n);
        end
        fprintm(1,'x=', x)
else
        if (n~=m)
                fprintf('\n\nThe number of rows and columns are different.\n');
        end
        printm('The rank of the matrix is: ', irank);
        printm(' ivet:',ivet);
        fprintf('\nThe matrix is\n ');
        for i=1:1:n
                for j=1:1:m+1
                        fprintf('%3.3f        ',H(i,j))
                end
        fprintf('\n');
        end
end
```

Chapter 5

Direct Dynamics: function dyn_eq

5.1 General discussion

This paragraph discusses function dyn_eq (see also § 3.3).
This function solves the equations

$$\Phi = skew\{\dot{W} \cdot J\} \tag{5.1}$$
$$\Gamma = skew\{W \cdot J\} \tag{5.2}$$

since they have the same form, we will discuss only the first one.

Generally in the first equation the unknown is $\dot{W}$, while in the second one is W. Both matrix Φ and $\dot{W}$ have six independent values; so in total they have 12 elements. If J is known and a total of 6 elements out of the 12 of Φ and $\dot{W}$ are known, it is possible to evaluate the others. To perform this operation function dyn_eq needs informations about which values are known and which values must be evaluated. This is performed by the 2×6 integer matrix `var`.
`var` must be filled by six "1" to indicate the elements to be evaluated and by six "0" to indicate the elements which are known. The first line is relative to the acceleration (angular and linear) and the second one to the actions (torques and forces). The first three columns are relative to angular terms (angular acceleration and torques) and the second to linear terms (acceleration and forces) according to this scheme:

$$var = \left[\begin{array}{ccc|ccc} \dot{w}_x & \dot{w}_y & \dot{w}_z & a_x & a_y & a_z \\ \hline t_x & t_y & t_z & f_x & f_y & f_z \end{array}\right] \tag{5.3}$$

so in the usual cases in which Φ is known and it's necessary to evaluate $\dot{W}$, matrix `var` must be filled in the following way:

$$var = \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{5.4}$$

Opposite, if $\dot{W}$ is known and you want to evaluate Φ, the correct values are:

$$var = \left[\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \tag{5.5}$$

in this last special case, dyn_eq gives the same results as function `skew(Wp, J)`. In every case, you must have exactly one "1" and one "0" in every column of `var`.

5.2 The Calling List.

The calling list (see § 3.3) for function dyn_eq to solve equation (5.1) is:

```
[Wp, F, test] = dyn_eq(J, Wp, F, var);
```

where:

J 4×4 inertia matrix of the body
Wp 4×4 acceleration matrix containing $\dot{W}$
F 4×4 action matrix containing Φ
var 2×6 matrix which specifies which elements of $\dot{W}$ and Φ are unknown

The calling list for function dyn_eq to solve equation (5.2) is:

```
[W, G, test] = dyn_eq(J, W, G, var);
```

where:

J 4×4 inertia matrix of the body
W 4×4 velocity matrix containing W
G 4×4 momentum matrix containing Γ
var 2×6 matrix which specifies which elements of W and Γ are unknown

dyn_eq returns value 'test' that is OK if the operation could be performed properly or NOTOK if an error had been detected. OK and NOTOK are constants defined in spacelib.m (see also § 2.4.3). An error occurs if dyn_eq is called with parameters that have not physical meanings (e.g. J is not positive defined or both $\dot{w}_i$ and t_i or a_i and f_i are unknown for any i).

Chapter 6

Header files

6.1 The header file spacelib.m

This is the header file which must be called to initialize SpaceLib©.

```
%_______________________________________________________________________________
%
%                       HEADER FILE     SpaceLib.M (November 2005)
%
% In  this file  are defined all the constants that are used by the
% SpaceLib functions. These constants are memorized in global variables.
%
% HEADER FILE USAGE:
%
% 1) If this M-file is invoked by the matlab command window or by the
% matlabrc.m file, all the global  variables are automatically loaded
% in memory (see user's manual).
%
% Typing the istruction "who global", MATLAB displays the list of the global
% variables loaded in memory.
%
% 2) Every function that uses the global variables, must  include the header
% file 'spheader' in the first  line  of  the  program (see chapter 2.1 of the
% user manual).
%
% 3) The directories containig SpaceLib are assigned to global variables
%    and the default directory is set accordingly.
%
% WARNING 1: The global variables defined in this of the constants defined in
% the header file have special meaning for many SpaceLib functions.
% Their value MUST NOT changed at any time.
%
% WARNING 2: there is a line similar to this
%
%    spc_lib_dir='c:\users\spacelib_m'  % SpaceLib directory
%
% that MUST be updated to match your installation!!!
%_______________________________________________________________________________


clc

%_______________________________________________________________________________
%
%               GLOBAL VARIABLES DECLARATION:
```

```
%_______________________________________________________________________________

spheader % declare global variables

%_______________________________________________________________________________
%
%       GLOBAL VARIABLES INITIALIZATION:
%_______________________________________________________________________________

X=1;    Y=2;    Z=3;    U=4;

Xaxis = [1 0 0]';
Yaxis = [0 1 0]';
Zaxis = [0 0 1]';

Xaxis_n = [-1 0 0]';
Yaxis_n = [0 -1 0]';
Zaxis_n = [0 0 -1]';

ORIGIN=[0 0 0 1]';

Rev   =0;  Pri   = 1;
Tor   =0;  For   = 1;
SYMM_ =1;  SKEW_ =-1;
Row   =0;  Col   = 1;

OK=1;   NOTOK=0;

PIG=pi;
PIG_2=pi/2;
PIG2=2*pi;

NULL3=zeros(3);
NULL4=zeros(4);
UNIT3=eye(3);
UNIT4=eye(4);

%_______________________________________________________________________________
%
%                  GLOBAL DIRECTORIES DECLARATION:
%_______________________________________________________________________________

% ***-----> the following line MUST be updated to match your installation!!!
spc_lib_dir='c:\users\spacelib_m'  % spacelib directory

spc_lib_dir_f=[spc_lib_dir,'\function']     % functions
spc_lib_dir_s=[spc_lib_dir,'\shortexa']     % short examples
spc_lib_dir_b=[spc_lib_dir,'\bigexa']       % big examples

matlabpath([matlabpath,';', spc_lib_dir,';', spc_lib_dir_f, ';',spc_lib_dir_s,';',
            spc_lib_dir_b]);

tmp= ['cd ',spc_lib_dir];
eval(tmp);
clear tmp;

%_______________________________________________________________________________
%
%                  PRINT "HEADER":
%_______________________________________________________________________________

fprintf('\n___________________________      SpaceLib      ___________________________\n')
```

```
fprintf('                              VERSION 2.2\n')
fprintf('                              A software library for\n')
fprintf('                          the kinematic and dynamic analysis\n')
fprintf('                              of systems of rigid bodies.\n\n')
fprintf('                 Includes general functions for vectors, matrices,\n')
fprintf('              kinematics, dynamics, Euler angles and linear systems\n\n')
fprintf('                      (c) G.LEGNANI  B.ZAPPA   R.ADAMINI 1990 - 2005\n\n')
fprintf('                MATLAB version with the cooperation of C.MOIOLA\n')
fprintf('           University of Brescia - Mechanical Engineering Department\n')
fprintf('                      Via Branze 38, 25123 BRESCIA, Italy\n')
fprintf('                     e-mail: giovanni. legnani @ ing.unibs.it\n')
fprintf('                     www:http://bsing.ing.unibs.it/~glegnani\n\n')
fprintf('                          SpaceLib (c) loaded in workspace\n')
fprintf('\n');
fprintf(' bug fixed January 2004 and November 2005');
fprintf('    (tested wih matlab 6.0.0.88 release 12)\n');
fprintf(' see readme.txt and user''s manual for release notes\n');
fprintf('___________________________________________________________________________\n')

cd
who global
```

6.2 The header file spheader.m

The file **spheader.m***, that must be 'included' in every program or function that uses* ***SpaceLib***$^{©}$ *constants, contains only the global variable* **declaration** *described in § 2.1.*

```
global X Y Z U Xaxis Yaxis Zaxis ORIGIN Rev Pri Tor For SYMM_ SKEW_ OK NOTOK
global Xaxis_n Yaxis_n Zaxis_n Row Col NULL3 NULL4 UNIT3 UNIT4
global spc_lib_dir spc_lib_dir_f spc_lib_dir_b spc_lib_dir_s
global PIG PIG2 PIG_2
```

Chapter 7

Sample programs

7.1 Program Rob_Mat

7.1.1 General information

This section presents the bases of a computer program for the automatic solution of the *direct kinematic problem* and the *inverse dynamic problem* for an industrial robot. To solve the direct kinematic problem means to find the motion of the end-effector of a *given* robot when the motions of its joint actuators are known. To solve the inverse dynamic problem means to find the actuators and the constraint actions (torques and forces) between the contiguous links of a given robot when the *external actions* and the *motion* of the manipulator are known.

In this example the robot has been regarded as an open chain of rigid bodies (links) which are jointed to each other by revolute or prismatic pairs (see figure 7.1). The program reads from a *"description file"* (`*.DAT`) the structure of the robot (link, lengths, masses, joint types etc.) and from a *"motion file"* (`*.MOT`) the motors movement and, as output, prints the motion (position, velocity and acceleration) of each link, as well as the internal actions between each couple of contiguous links.

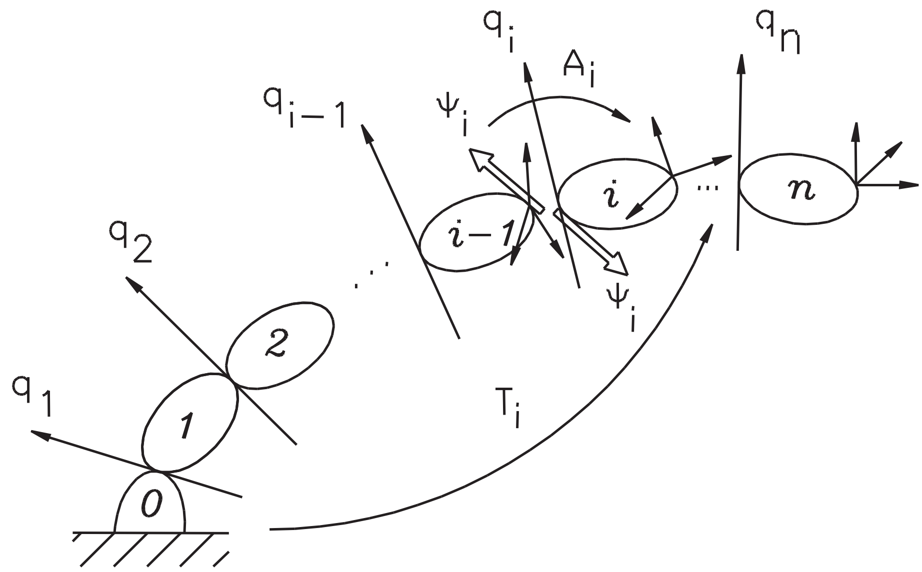


Figure 7.1: The scheme of a general serial manipulator.

7.1.2 The solution algorithm

Initially, the sample program `ROB-MAT` calculates the absolute position, speed and acceleration of all the links of the robot. This task is iteratively executed to evaluate the kinematic quantities of the links, starting from the base of the robot and proceeding to the end-effector. Conversely, the dynamic analysis is iteratively executed from the end to the base. The table 7.1 explains the meaning of the symbols used in `ROB-MAT`.

The program is structured in three main parts[1]: `DATA INPUT`, `CALCULATIONS` and `DATA OUTPUT`.

`DATA INPUT` can be divided into four steps:

1) Input data describing the geometrical structure of the manipulator:
   - *the number of links* constituting the robot;
   - for each link "i":
     * *the joint type*;
     * *five parameters* to describe the position of the frame (i), fixed on link "i", with respect to the frame (i-1), fixed on link "i-1", according to an extension to Denavit and Hartenberg approach (see [3], [4]);
2) Input data describing the dynamic parameters of the manipulator:
   - for each link "i":
     * *the six barycentral inertial moments*;
     * *the mass*;
     * *the coordinates of the center of mass* referred to the local frame (i);
3) Input data describing the external actions on manipulator:
   - *the three components of gravity acceleration* referred to the base frame;
   - *the actions* (the components of force and the components of torque) *applied on the end-effector* of the robot referred to the local frame of the gripper;
4) Input data describing the motions of the actuators:
   - for each link "i" and for each instant:
     * *the relative position*, *speed* and *acceleration* of frame (i) with respect to frame (i-1);

The `CALCULATION` part, deeply using the subroutines of the library, can be briefly described by means of the following statements (see § 7.1.4):

- Step (1) relates with sorts of *initialization procedures* to change from the scalar to the matrix environment.
- Steps (2) to (9) are included in a `for` cycle to repeat the *kinematic calculations* (absolute position, speed and acceleration) for all the links forming the robot.
- Step (10) initializes the dynamic calculations reading the external actions on the end-effector from file, building the external actions matrix and transforming it from local to base frame.
- Steps (11) to (13) performs the *dynamic calculations* (i.e. evaluates the internal actions), and are included in a `for` cycle where the counter i decreases from the total number of the robot's links to 1.
- While the kinematic calculations iteratively develop from the base of the robot to the end-effector, the dynamic calculations begin from the hand of the manipulator ending at the base.
- Note that all the matrices have been brought back to the base frame (steps (6), (7), (10), (11)) before executing the main operations (steps (8), (9), (12), (13)): this is not a set choice (one can assume as reference any frame), but it seems the easiest approach.

The program has a very simple `DATA OUTPUT` just intended for its debug, therefore only the most significant matrices are printed.

[1]It is important to remark that the only purpose of the above program is to give a simple example of the library use, so that any programmer can find better programming solutions.

7.1.3 Using ROB-MAT

As wider described above, `ROB-MAT` program requires as input:

- `DATA_FILE`: file describing the geometry of the robot, its inertial parameters and the external actions;
- `MOTION_FILE`: file containing, for every link, joint position, velocity and acceleration.

and as output:

- `OUT_FILE`: file where `ROB-MAT` will print all the matrices describing the movements of the links and the joint internal actions.

The program reads from the `DATA_FILE` the description of the robot and from the `MOTION_FILE` the motion of its motors and it prints in `OUT_FILE` file all the matrices describing the movements of the links and the joint internal actions.
The three dimensional array which contain position, velocity and acceleration matrices, is realized with the notation `A(:, k)`, where the parameter `(:, k)` defines a "window" of four columns and all the rows in the matrix (in fact `k=4*i-3:4*i`). So, `A(:, k)` corresponds to `A(i)`, and `A(:, k+4)` corresponds to `A(i+1)`, because 4 is added to the subscript of each component of the vector `k` (see § 2.5).

Program Symbols	*Meaning*
`A[:,k]`	Relative location of the frame (i) with respect to frame $(i-1)$
`T[:,k]`	Absolute location of the frame (i) with respect to frame (0)
`IT[:,k]`	Inverse of T[:,k]
`W[:,k]`	Relative velocity matrix of the frame (i) with respect to frame $(i-1)$ seen in frame $(i-1)$
`WO[:,k]`	Relative velocity matrix of the frame (i) with respect to frame $(i-1)$ seen in frame (0)
`WA[:,k]`	Absolute velocity matrix of the frame (i) with respect to frame (0) seen in frame (0)
`H[:,k]`	Relative acceleration of the frame (i) with respect to frame $(i-1)$ seen in frame $(i-1)$
`HO[:,k]`	Relative acceleration of the frame (i) with respect to frame $(i-1)$ seen in frame (0)
`HA[:,k]`	Absolute acceleration of the frame (i) with respect to frame (0) seen in frame (0)
`Hg`	Matrix including the three components of gravity acceleration seen in the absolute frame (0)
`Ht`	Sum of `HA[:,k]` and `Hg`
`J[:,k]`	Mass distribution of the link (i) with respect to the origin of the frame (i) seen in frame (i)
`JO[:,k]`	Mass distribution of the link (i) with respect to the origin of the frame (0) seen in frame (0)
`FI[:,k]`	Actions (forces and torques) due to inertia and weight applied on link (i) seen in (0)
`ACTO[:,k]`	Matrix embodies the constraint actions on joint (i) seen in absolute frame (0)

Table 7.1: Meaning of the symbols used in the program `ROB_MAT`

7.1.4 Listing of the program ROB_MAT.M

```
%_________________________________________________________________________________________
%
%                                         ROB_MAT
%
% Program  for  the  automatic  solution of the direct kinematic problem and the inverse
```

```
% dynamic problem for any serial manipulator. The  program  reads from a "description file
% (*.DAT)" the structure of the robot (number of links, lenghts, masses, joint type,ecc.),
% and from a " motion file (*.MOT)" the motors movement, and, as output, prints the motion
% (position, velocity, acceleration) of each link.
% (c) G.Legnani 1998 adapted from G.Legnani and R.Faglia 1990
%_____________________________________________________________________________________________

clc
string1=input('Digit the name of the input DATA FILE: ','s');
data=fopen(string1,'r');
if (data==-1)
        error('Error in ROB_MAT, unable to open DATA FILE ')
end
string2=input('Digit the name of the input MOTION FILE: ','s');
motion=fopen(string2,'r');
if (motion==-1)
        error('Error in ROB_MAT, unable to open the MOTION FILE ')
end
string3=input('Digit the name of the OUTPUT FILE (S=Screen): ','s');
string3=upper(string3); % uppercase;
if (string3=='S')
        out=1;
else
        out=fopen(string3,'wt');
end
if (out==-1)
        error('Error in ROB_MAT, unable to open OUTPUT FILE ')
end
nlink=fscanf(data,'%d',1);
                        %_____INIZIALIZATIONS
T =eye(4,4*(nlink+1));                                     % MATRICES:
WA=zeros(4,4*(nlink+1));
HA=zeros(4,4*(nlink+1));
J =zeros(4,4*nlink);
W =zeros(4,4*nlink);
WO=zeros(4,4*nlink);
HO=zeros(4,4*nlink);
A =zeros(4,4*nlink);
theta=zeros([1,nlink]);                                    % VECTORS:
jtype=zeros([1,nlink]);
a=zeros([1,nlink]);
b=zeros([1,nlink]);
alfa=zeros([1,nlink]);
for i=1:1:nlink                                         % for each link (STEP 1)
        kk=4*i-3;
        k=[kk:kk+3];
        jtype(i)=fscanf(data,'%d',1);                   % Read Denavit & Hartenberg parameters
        theta(i)=fscanf(data,'%f',1);
        s(i)=    fscanf(data,'%f',1);
        b(i)=    fscanf(data,'%f',1);
        a(i)=    fscanf(data,'%f',1);
        alfa(i)= fscanf(data,'%f',1);
        m=  fscanf(data,'%f',1);                        % Read Dinamic Data
        jxx=fscanf(data,'%f',1);
        jxy=fscanf(data,'%f',1);
        jxz=fscanf(data,'%f',1);
        jyy=fscanf(data,'%f',1);
        jyz=fscanf(data,'%f',1);
        jzz=fscanf(data,'%f',1);
        xg= fscanf(data,'%f',1);
        yg= fscanf(data,'%f',1);
        zg= fscanf(data,'%f',1);
```

```
        J(:,k)=jtoj(m,jxx,jyy,jzz,jxy,jyz,jxz,xg,yg,zg);% Builds Inertia Matrix
end                                                        % end 1ST step
gx=fscanf(data,'%f',1);
gy=fscanf(data,'%f',1);
gz=fscanf(data,'%f',1);
fx=fscanf(data,'%f',1);
fy=fscanf(data,'%f',1);
fz=fscanf(data,'%f',1);
cx=fscanf(data,'%f',1);
cy=fscanf(data,'%f',1);
cz=fscanf(data,'%f',1);
Hg=gtom(gx,gy,gz);                                         % Builds gravity matrix
dt=fscanf(motion,'%f',1);
%________________________FOR EACH INSTANT OF TIME:___________________________
%____________________________DIRECT KINEMATICS_____________________________
for time=0:dt:~feof(motion)
        for i=1:1:nlink
                kk=4*i-3;
                k=[kk:kk+3];
                q   = fscanf(motion,'%f',1);
                qp  = fscanf(motion,'%f',1);
                [qpp,count] = fscanf(motion,'%f',1);
                if count~=1
                        fclose('all');
                        return
                end
                                                          % Builds relative position matrix (3)
                A(:,k)=dhtom(jtype(i),theta(i),s(i),b(i),a(i),alfa(i),q);
                           % Builds relative velocity and acceleration matrix in local frame(4)
                [ W(:,k),H(:,k) ]=veactowh(jtype(i),qp,qpp);
                                                         % Evaluates absolute position matrix (5)
                T(:,k+4)=T(:,k)*A(:,k);
                         % Transform relative velocity matrix from local frame to base frame(6)
                WO(:,k)=mami( W(:,k),T(:,k) );
                    % Transform relative acceleration matrix from local frame to base frame (7)
                HO(:,k)=mami( H(:,k),T(:,k) );
                WA(:,k+4)=WA(:,k)+WO(:,k);               % Evaluates absolute velocity matrix (8)
                                                     % Evaluates absolute acceleration matrix (9)
                HA(:,k+4)=coriolis(HA(:,k),HO(:,k),WA(:,k),WO(:,k));
        end                                                          % end of cycle (kinematics)
%____________________SOLUTION OF THE INVERSE DINAMYC PROBLEM_______________
        EXT=actom(fx,fy,fz,cx,cy,cz);
        ACTO(:,4*nlink+1:4*nlink+4)=mamt(EXT,T(:,4*nlink+1:4*nlink+4));
        for kk=4*nlink:-4:4
                k=[kk-3:kk];
                JO(:,k)=mamt(J(:,k),T(:,k+4));
                Ht=Hg-HA(:,k+4);
                FI(:,k)=skew(Ht,JO(:,k));
                ACTO(:,k)=FI(:,k)+ACTO(:,k+4);
        end
%_________________________OUTPUT RESULTS _______________________
        if (string3=='S') string3=' SCREEN '; end
        fprintf(1,'\n\n-------- Print Output results on FILE: %s ----------\n',string3 )
        for i=1:1:nlink
                kk=4*i-3;
                k=[kk:kk+3];
                fprintf(out,'\n\n Link %d \n\n',i);
                fprintm(out,'Relative Position Matrix                   A',   A(:,k));
                fprintm(out,'Absolute Position Matrix                   T',   T(:,k+4));
                fprintm(out,'Relative Velocity Matrix in frame (i)      W',   W(:,k));
                fprintm(out,'Relative Velocity Matrix in frame (0)     WO',  WO(:,k));
                fprintm(out,'Absolute Velocity Matrix in frame (0)     WA',  WA(:,k+4));
```

```
                fprintm(out,'Relative Acceleration Matrix in frame (i)  H',   H(:,k));
                fprintm(out,'Relative Acceleration Matrix in frame  (0)H0',  H0(:,k));
                fprintm(out,'Absolute Acceleration Matrix in frame  (0) A',  HA(:,k+4));
                fprintm(out,'Inertia Matrix in frame (i)                J',   J(:,k));
                fprintm(out,'Inertia Matrix in frame (0)                0',  J0(:,k));
                fprintm(out,'Total actions                             FI',  FI(:,k));
                fprintm(out,'Actions on Joint (i)                    ACT0',ACT0(:,k));
        end                                                                 % end output results
end                                                                         % end main loop
fclose('all');                                                              % close all files
```

7.1.5 Use of Rob-Mat

Example n.1: SCARA ICOMATIC03© ROBOT

Here is an example of the simulation of a 3 d.o.f. SCARA ICOMATIC03© ROBOT (see figure 7.2), described in file SCARA.DAT, acting a trajectory of two points included in file SCARA. MOT. File SCARA.DAT

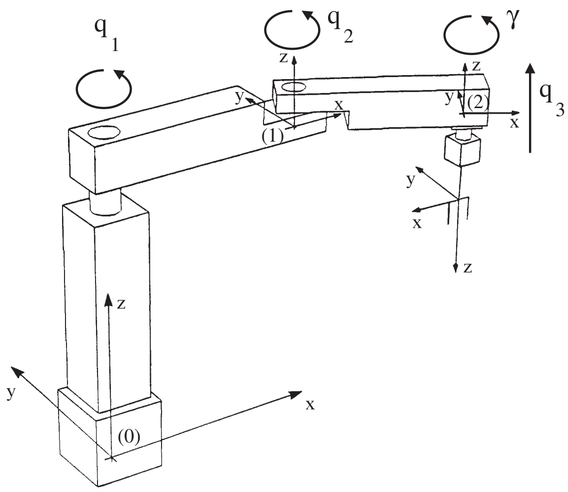


Figure 7.2: Kinematic structure of the SCARA robot.

contains the geometry of the robot SCARA ICOMATIC03©, its inertial parameters and the external actions on the gripper (see table 7.2). In the example the angle γ is considered constant ($\gamma = 0$) and so the x axes of the two last frames are parallel to each other.
File SCARA.MOT includes the motion of the actuators of the robot, given in terms of displacement, speed and acceleration (see table 7.3). In this example the law of motion is formed by two points only; obviously the program is able to elaborate laws of motions composed by a larger number of points!
File SCARA.OUT includes the output matrices. They have not been printed here for the file is very long; however they can be found in the BIGEXA directory of SpaceLib©.

Example n.2: SMART© ROBOT

Here is an example of the simulation of a SMART© ROBOT (6 degrees of freedom) described in file SMART.DAT acting a trajectory of only one point included in file SMART.MOT (see table 7.4 and figure 7.3).

File SMART.OUT includes the output matrices. It have not been printed here for the file is very long; however they can be found in the BIGEXA directory of SpaceLib©.

NOTE: The geometrical data of the robot links correspond to the actual robot, while the values of the dynamical parameters have been estimated very approximatively.

DATA_FILE `SCARA.DAT`	MEANING
3	number of link
	FIRST LINK
0	Joint type
0 0 0 0.33 0	Denavit and Hartenberg parameters
10	Mass of the first link
0.03 0 0	Inertia moments jxx, jxy, jxz
0.03 0	Inertia moments jyy, jyz
0.05	Inertia moments jzz
-0.05 0.0 0.0	Center of Mass coordinates Xg, Yg, Zg
	SECOND LINK
0	
0 0 0 0.33 0	
10	
0.03 0 0	
0.03 0	
0.05	
-0.05 0 0	
	THIRD LINK (END-EFFECTOR)
1	
0 -0.1 0 0 3.1415	
3	
0.0008 0 0	
0.0008 0	
0.0015	
0 0 -0.10	
	EXTERNAL ACTIONS
0 0 -9.8	Gravity components in base frame (0)
0 0 0 0 0 0	External forces and torques applied on the end effector

Table 7.2: Content of the file `SCARA.DAT`

MOTION_FILE `SCARA.MOT`	MEANING
0.05	dt
	FIRST POINT
0 1 10	displacement, speed and acceleration of the first motor
0 2 20	displacement, speed and acceleration of the second motor
0 0 0	displacement, speed and acceleration of the third motor
	SECOND POINT
0.1 1.1 11	displacement, speed and acceleration of the first motor
0.1 2.1 21	displacement, speed and acceleration of the second motor
0 0 0	displacement, speed and acceleration of the third motor
	OTHERS POINT
...	

Table 7.3: Content of the file `SCARA.MOT`

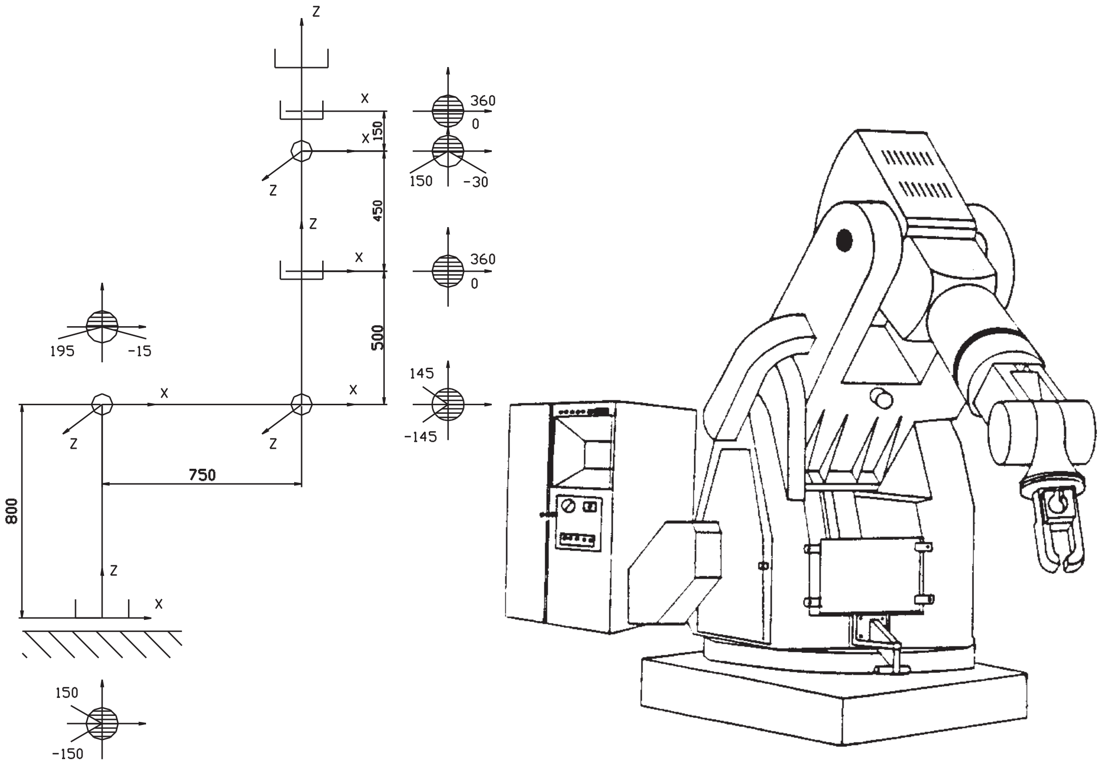


Figure 7.3: Frames definition for `SMART ROBOT`.

DATA_FILE `SMART.DAT`			MOTION_FILE `SMART.MOT`
6			0.05
0	0	0	0 0.5 2
0 0.8 0 0 1.57079	0 0 0.5 0 -1.57079	0 0 0.15 0 -1.57079	1.57 0.5 2
600	200	100	-1.57 0.5 2
100 0 0	10 0 0	3 0 0	
120 0	10 0	3 0	0 0.5 2
100	18	5	0.26 0.5 2
0 -0.40 0	0 0 -0.2	0 0 -0.07	0 0.5 2
0	0	0	
0 0 0 0.75 0	0 0.45 0 0 1.57079	0 0.0 0 0 0	
300	100	200	
18 0 0	3 0 0	4 0 0	
10 0	5 0	4 0	
10	3	6	
-0.30 0 0	0 -0.20 0	0 0 -0.5	
		0 0 -9.8	
		0 0 0 0 0 0	

Table 7.4: Content of the files `SMART.DAT` and `SMART.MOT`

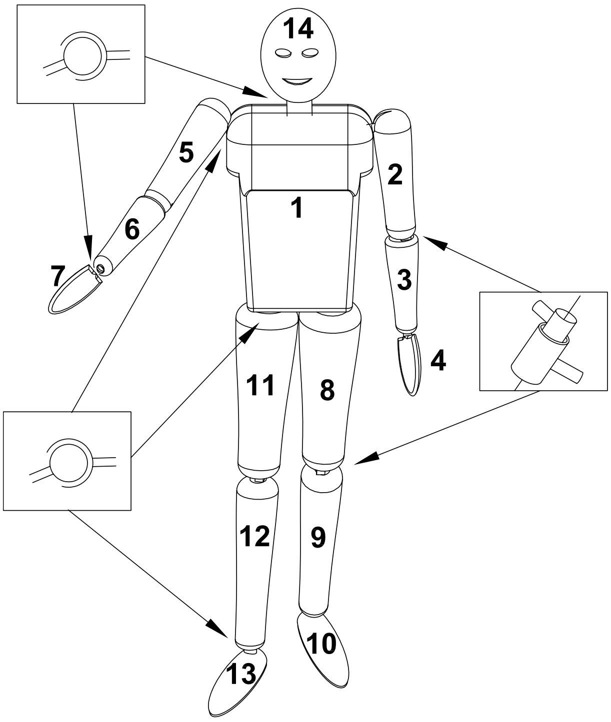


Figure 7.4: The model of the human body considered in the references [8], [9], and [14]:: the human joints are approximated by spherical or revolute hinges.

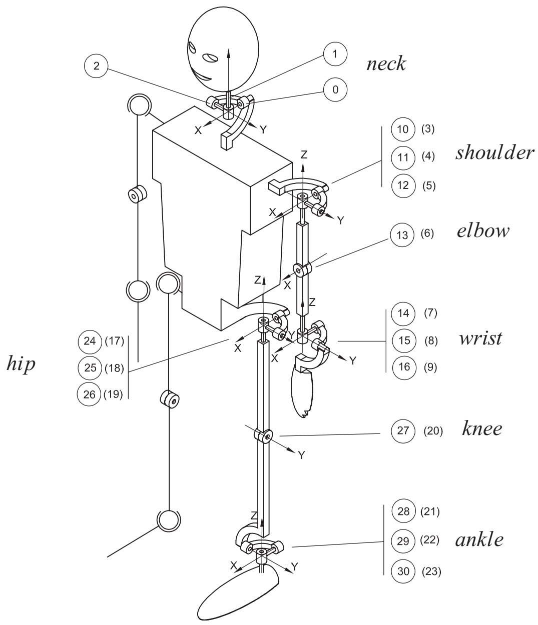


Figure 7.5: Schematization of the spherical joints by revolute hinges and enumeration of the degrees of freedom. Number in parentheses refer to right side. See section 7.2 for a simplified version of the model.

7.2 Program Test

7.2.1 General information

This sample program demonstrates the use of `dyn_eq` function for the solution of the direct dynamic problem of a two-link system floating in the 3D space (see figure 7.6). In practice, the program predicts the trajectory of the system. This is an educational simplification of the problem of finding the trajectory of a man during a jump (or dive) widely described in [8], [9] and [14] to which one can refer for more details. The program reads from a file (`TEST.DAT`) the links description (masses, inertias, ...) and from another file (`TEST.MOT`) the motion of the motor and prints on the screen the trajectory of the two links (position, velocity and acceleration matrices).

7.2.2 Theory in brief

The system (see figure 7.6) is free in the space and the relative position of the two links is forced by a motor which imposes a motion law $q(t)$, $\dot{q}(t)$, $\ddot{q}(t)$. If the following data are known:

- inertia of the two links (J_1 and J_2)
- the initial position and velocity of body 1 ($M_{0,1}$ and $W_{0,1}$)
- the relative motion between the links (q, $\dot{q}$ and $\ddot{q}$, and so $M_{1,2}$, $W_{1,2}$, $H_{1,2}$)

then it is possible to evaluate the acceleration of link 1. This is what is necessary in order to obtain the trajectory of the system by a numerical integration. More in detail the acceleration of body 1 is the sum of two terms one of which is known while the other is the unknown.
The dynamic equation[2] of the system is

$$\Phi_g = skew\left\{H_{0,1} \cdot J_{1(0)}\right\} + skew\left\{H_{0,2} \cdot J_{2(0)}\right\} = [0] \tag{7.1}$$

[2]For the subscript convection see § 2.2.1.

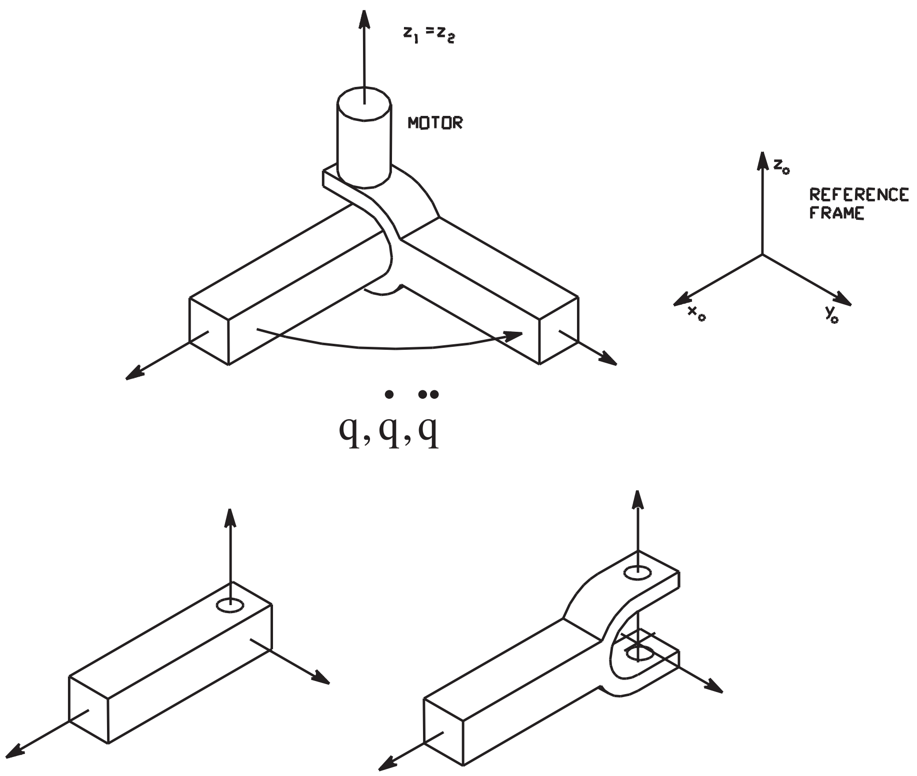


Figure 7.6: The system of the example `test`.

since the system is free in the space and it is not subjected to the gravity force, then Φ_g is the null matrix [0]. The matrices describing the motion of the two bodies are related by the following relations:

- the acceleration of body 1 is the sum of two terms. The first is known, while the second is the unknown

$$H_{0,1} = W_{0,1}^2 + \dot{W}_{0,1} \tag{7.2}$$

- the position of body 2 is

$$M_{0,2} = M_{0,1} \cdot M_{1,2} \qquad M_{1,2} = M\,(q) \tag{7.3}$$

- the velocity of body 2 is

$$W_{0,2} = W_{0,1} + W_{1,2(0)} \qquad W_{1,2(0)} = M_{0,1} \cdot W_{1,2} \cdot M_{0,1}^{-1} \qquad W_{1,2} = W(\dot{q}) \tag{7.4}$$

- the acceleration of body 2 is

$$H_{0,2} = H_{0,1} + H_{1,2(0)} + 2 \cdot W_{0,1} \cdot W_{1,2(0)} \qquad H_{1,2(0)} = M_{0,1} \cdot H_{1,2} \cdot M_{0,1}^{-1} \tag{7.5}$$

$$H_{1,2} = W_{1,2}^2 + \dot{W}_{1,2} \qquad \dot{W}_{1,2} = \dot{W}(\ddot{q}) \tag{7.6}$$

- the inertia of the two links can be expressed in base frame by the following relations

$$J_{1(0)} = M_{0,1} \cdot J_1 \cdot M_{0,1}^t \qquad J_{2(0)} = M_{0,2} \cdot J_2 \cdot M_{0,2}^t \tag{7.7}$$

J_1 and J_2 are constant and their value is known, while $M_{1,2}$, $W_{1,2}$ and $\dot{W}_{1,2}$ can be easily evaluated by knowing $q(t)$, $\dot{q}(t)$ and $\ddot{q}(t)$. At last $M_{0,1}$ and $W_{0,1}$ are known at the initial time t=0 and $\dot{W}_{0,1}$ will be the result of the following calculation.

The dynamic equation (7.1) can be "exploded" by means of three successive steps:

- union of the two terms

$$[0] = skew\left\{H_{0,1} \cdot J_{1(0)} + H_{0,2} \cdot J_{2(0)}\right\} \tag{7.8}$$

- explosion of H terms

$$[0] = skew\left\{\left(W_{0,1}^2 + \dot{W}_{0,1}\right) \cdot J_{1(0)} + \left(H_{0,1} + H_{1,2(0)} + 2 \cdot W_{0,1} \cdot W_{1,2(0)}\right) \cdot J_{2(0)}\right\} \tag{7.9}$$

- new explosion of H terms

$$[0] \ = skew\left\{\left(W_{0,1}^2 + \dot{W}_{0,1}\right) \cdot J_{1(0)} + \left(\left(W_{0,1}^2 + \dot{W}_{0,1}\right) + H_{1,2(0)} + 2 \cdot W_{0,1} \cdot W_{1,2(0)}\right) \cdot J_{2(0)}\right\} \tag{7.10}$$

and then the terms contained in the skew operator are divided in order to separate the terms containing the unknown $\dot{W}_{0,1}$ from the others.

$$skew\left\{W_{0,1}^2 \cdot J_{1(0)} + \left(W_{0,1}^2 + H_{1,2(0)} + 2 \cdot W_{0,1} \cdot W_{1,2(0)}\right) \cdot J_{2(0)}\right\} = skew\left\{-\dot{W}_{0,1} \cdot \left(J_{1(0)} + J_{2(0)}\right)\right\} \tag{7.11}$$

or shortly

$$= skew\left\{-\dot{W}_{0,1} \cdot J_{tot}\right\} \tag{7.12}$$

with the positions

$$\begin{gathered} J_{tot} = J_{1(0)} + J_{2(0)} \\ \Phi = skew\left\{H_{0,1}^* \cdot J_{1(0)} + H_{0,2}^* \cdot J_{2(0)}\right\} \end{gathered} \tag{7.13}$$

where $H_{0,1}^*$ and $H_{0,2}^*$ are the "partial" acceleration of body 1 and 2 (i.e. their absolute acceleration evaluated considering $\dot{W}_{0,1}$=[0])

$$H_{0,1}^* = W_{0,1}^2 \qquad\qquad H_{0,2}^* = H_{0,1}^* + H_{1,2(0)} + 2 \cdot W_{0,1} \cdot W_{1,2(0)} \tag{7.14}$$

equation (7.12) can be solved by using the `dyn_eq` function of `SpaceLib`©.

Then the total absolute acceleration of bodies 1 and 2 can be evaluated. It yields:

$$H_{0,1} = H_{0,1}^* + \dot{W}_{0,1} \qquad\qquad H_{0,2(0)} = H_{0,2}^* + \dot{W}_{0,1} \tag{7.15}$$

Although more raffinate integration methods can be set up, the new position and speed of link 1 at the time (t+Δt) can be approximatively evaluated, for instance, by the simple following integration method

$$\begin{gathered}\left\{\begin{array}{l} M_{0,1<t+\Delta t>} \cong M_{0,1} + \dot{M}_{0,1}\Delta t + \frac{1}{2}\ddot{M}_{0,1}\Delta t^2 = \left([1] + W_{0,1}\Delta t + \frac{1}{2}H_{0,1}\Delta t^2\right) M_{0,1} = \Delta M \cdot M_{0,1} \\ W_{0,1<t+\Delta t>} \cong W_{0,1} + \dot{W}_{0,1}\Delta t \end{array}\right. \\ \textit{with} \\ \left\{\begin{array}{l} \dot{M}_{0,1} = W_{0,1} \cdot M_{0,1} \\ \ddot{M}_{0,1} = H_{0,1} \cdot M_{0,1} \\ \Delta M = \left([1] + W_{0,1} \cdot \Delta t + \frac{1}{2} \cdot H_{0,1} \cdot \Delta t^2\right) \end{array}\right. \qquad [1] = \text{identity matrix} \end{gathered} \tag{7.16}$$

All of these operations must be repeated iteratively in order to evaluate the trajectory of the system.

7.2.3 The program (cross reference)

The variables of the program have the meaning listed in table 7.5. The initial position and speed of body 1 are assigned by initializing the matrices `m1`, `W1`.
The inertia matrices of the bodies are assigned by initializing the matrices `J1` and `J2`.
The relative motion between the links is described by three variables `q`, `qp`, `qpp` (position, speed and acceleration).

Program Symbols	*Equation Symbols*	*Meaning*
q, qp and qpp	q, $\dot{q}$ and $\ddot{q}$	position, speed and acceleration of the motor
m1	$M_{0,1}$	abs. position of body 1
W1	$W_{0,1}$	abs. velocity of body 1 (in frame 0)
H1	$\begin{cases} H^*_{0,1} \\ H_{0,1} \end{cases}$	partial acceleration of body 1 (in frame 0) absolute acceleration of body 1 (in frame 0)
Wp	$\dot{W}_{0,1}$	Unknown part of acceleration of body 1
m2	$M_{0,2}$	abs. position of body 2
W2	$W_{0,2(0)}$	abs. velocity of body 2 (in frame 0)
H2	$\begin{cases} H^*_{0,2} \\ H_{0,2} \end{cases}$	partial acceleration of body (in frame 0) absolute acceleration of body 2 (in frame 0)
m12	$M_{1,2}$	relative position of body 1 and 2
W12	$W_{1,2}$	rel. velocity between body 1 and 2 (in frame 1)
H12	$H_{1,2}$	rel. acceleration between body 1 and 2 (in frame 1)
W120	$W_{1,2(0)}$	rel. velocity between body 1 and 2 (in frame 0)
H120	$H_{1,2(0)}$	rel. acceleration between body 1 and 2 (in frame 0)
J1, J2, J10, J20, Jtot	J_1, J_2, $J_{1(0)}$, $J_{2(0)}$, J_{tot}	Inerzia
F1		$skew\left\{H^*_{0,1} \cdot J_{1(0)}\right\} + skew\left\{H^*_{0,2} \cdot J_{2(0)}\right\}$
F2		$skew\left\{H^*_{0,2} \cdot J_{2(0)}\right\}$

Table 7.5: Cross reference for the program TEST

7.2.4 Scheme of the program

The program consists of the following steps (letters and digits refer to the program source code listed in the following pages).

1. Reads the description of the links and the initial condition (position and velocity) of link 1 from file `TEST.DAT`.

2. For each instant

   a) reads from file `TEST.MOT` the motion (q, $\dot{q}$, $\ddot{q}$) of the motor.

   b) evaluates `m12`, the relative position matrix of body one and two.

   c) evaluates `m2`, the absolute position of body 2.

   d) evaluates partial acceleration of link1: `H1` = `W1` · `W1`.

   e) evaluates `W12` and `H12`, the relative velocity and acceleration between the bodies.

   f) evaluates `W120` and `H120` referring `W12` and `H12` to the reference frame.

   g) evaluates absolute velocity `W02` and partial acceleration of link 2 `H02`.

   h) evaluates `J10` and `J20` referring `J1` and `J2` to the base frame.

   i) evaluates `Jtot` = `J10` + `J20`.

   j) evaluates `F2` = *skew*(`H2`, `J20`) and `F1` = `F2` + skew(`H1`, `J10`).

   k) finds the unknown `Wp` by using the `dyn_eq` function.

   l) evaluates the total acceleration of links 1 & 2 `H1` = `H1` +`Wp` and `H2` = `H2` +`Wp`.

   m) evaluate matrix `dm`: `dm` = [1] + `W01 dt` + 0.5 `H01 dt`^{2}.

   n) evaluates the new absolute position of link 1 (`t = t+dt`).

   o) evaluates the new absolute velocity of link 1 (`t = t+dt`).

DATA_FILE TEST.DAT	MEANING
	LINK 1
10 1 1 1	mass, Jx, Jy, Jz inertia moments
0 0 0	Jxy, Jyz, Jxz
1 0 0	Xg, Yg, Zg centre of mass position
	LINK 2
10 1 1 1	mass, Jx, Jy, Jz inertia moments
0 0 0	Jxy, Jyz, Jxz
1 0 0	Xg, Yg, Zg centre of mass position
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	velocity matrix of link 1
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1	position matrix of link 1

Table 7.6: Content of the file TEST.DAT

3. Repeats steps a÷o until the motion file is completely scanned.

Note: An improved version of the program (TEST_NEW) is also contained; it is based on the following considerations.

- *The angular moment of the system should be constant, but inaccuracy in the integration method corrupts it. In this new version of the program, some statements have been added to preserve the total angular momentum obtaining an improved final accuracy.*
- *At each integration step, the linear and angular momentum are evaluated and a velocity* `dW` *added to each link of the system in order to set the value of the linear and angular momentum equal to their initial value (*`G=G0` *for* `t`*=0.)*

7.2.5 The format of the inputfiles

The BIGEXA directory of SpaceLib© contains an example of input files (TEST.DAT and TEST.MOT). They are here listed in order to show their format. The input file TEST.DAT has the format listed in table 7.6 while the law of motion file TEST.MOT has the format show in table 7.7. The first line of the file TEST.MOT contains the time step dt, while the other lines contain the value of the motor position, speed and acceleration ad each time.

7.2.6 Source code of TEST.M

```
%_______________________________________________________________________________________________
%
%                                        PROGRAM TEST.M
%
% " Program for the trajectory prediction of a two-link system floating in the space".
%
% This program demonstrates the use of dyn_eq function for the solution of the direct dynamic
% problem of a two-link system floating in the 3-D space.
% The program predicts the trajectory of the system. The program reads from a file (TEST.DAT)
% the links description (mass, inertias, the coordinates of the centre of mass) and from
% another file (TEST.MOT) the motion of the motors, and prints on the screen the trajectory of
```

MOTION_FILE TEST.MOT			MEANING
0.002			dt (step of time)
0.000000E+00	0.000000E+00	0.000000E+00	$q\ \dot{q}\ \ddot{q}$ (t=0)
5.556963E-07	8.334976E-04	8.333988E-01	.. .. .. (t=dt)
4.444593E-06	3.334976E-03	1.665613	.. .. .. (t=2 · dt)
1.499523E-05	7.494372E-03	2.495461	.. .. ..
3.552638E-05	1.331228E-02	3.321762	
6.934328E-05	2.077827E-02	4.143342	
...	...	...	

Table 7.7: Content of the file TEST.MOT

```
% the two links.
%                    (c) G.Legnani and  C.Moiola 1998 adapted from G.Legnani and R.Faglia 1990
%_______________________________________________________________________________________________

spheader
clc
                                              % Initializations
Zax=Zaxis;
O=ORIGIN;
var=[1 1 1 1 1 1 ; 0 0 0 0 0 0];
Wp=NULL4;
                                              % Open description file
fil=fopen([spc_lib_dir_b,'\test.dat'],'r');
if (fil==-1)
        error('Error on input file TEST.DAT ')
end
out=input('output to screen? (1=yes)');
if (out~=1)
   outfile=['testoutold.out'];
   out=fopen(outfile,'wt');
end;
if (out==-1)
        error('Error in TEST.M: Unable to open output file ')
end
                                              % Read description of the links step(1)
m=  fscanf(fil,'%f',1);
jxx=fscanf(fil,'%f',1); jyy=fscanf(fil,'%f',1); jzz=fscanf(fil,'%f',1);
jxy=fscanf(fil,'%f',1); jyz=fscanf(fil,'%f',1); jxz=fscanf(fil,'%f',1);
xg= fscanf(fil,'%f',1);       yg= fscanf(fil,'%f',1);       zg= fscanf(fil,'%f',1);
                                              % Builds Inertia Matrix of link 1
J1=jtoj(m,jxx,jyy,jzz,jxy,jyz,jxz,xg,yg,zg);
m=  fscanf(fil,'%f',1);
jxx=fscanf(fil,'%f',1); jyy=fscanf(fil,'%f',1); jzz=fscanf(fil,'%f',1);
jxy=fscanf(fil,'%f',1); jyz=fscanf(fil,'%f',1); jxz=fscanf(fil,'%f',1);
xg= fscanf(fil,'%f',1); yg= fscanf(fil,'%f',1); zg= fscanf(fil,'%f',1);
                                              % Builds Inertia Matrix of link 2
J2=jtoj(m,jxx,jyy,jzz,jxy,jyz,jxz,xg,yg,zg);
                                              % Read initial condition of the system
W1= fscanf(fil,'%f',[4 4]);                   % Read velocity matrix of link 1
W1=W1';
m1= fscanf(fil,'%f',[4 4]);                   % Read position matrix of link 1
m1=m1';
                                              % Open motion file
fil=fopen([spc_lib_dir_b,'\test.mot'],'r');
if (fil==-1)
```

```
        error('Error on motion file TEST.MOT ')
end
dt=fscanf(fil,'%f',1);                        % Read integration step "dt"
for t=0:dt:(~feof(fil))                       % Loop for each istant of time step(2)
        q  =fscanf(fil,'%f',1);               % Read motion of motor (a)
        qp =fscanf(fil,'%f',1);
        [qpp,count]=fscanf(fil,'%f',1);
        if (count~=1)       % Check end of motion file. If motion file is empty -> end of loop
                return
        end
        m12=screwtom(Zax,q,0,0);              % Relative position of link 1 & 2 (b)
        m2=m1*m12;                            % Absolute position of link 2     (c)
        H1=W1^2;                              % Partial acceleration of link 1 (d)
        [W12,H12]= vactowh2(Rev,Z,qp,qpp);    % Rel. vel & acc. of link 1&2 (e)
        W120=mami(W12,m1);                    % (f)
        H120=mami(H12,m1);
        W120(1:3,1:3)=normskew(W120(1:3,1:3),SKEW_);   % normalization reducing num. error
        W2=W1+W120;                           % Abs.vel.and partial acceleration of link 2(g)
        H2=coriolis(H1,H120,W1,W120);
        J10=mamt(J1,m1);                      % Referinertia moments to absolute frame
        J20=mamt(J2,m2);
        J10=normskew(J10,SYMM_);              % normalization reducing num. error
        J20=normskew(J20,SYMM_);              % normalization reducing num. error
        Jtot=J10+J20;                         % Total inertia (i)
        F1=skew(H1,J10);                      % Evaluate inertia actions (j)
        F2=skew(H2,J20);
        F=F1+F2;
        [Wp,ff,exitcode]=dyn_eq(Jtot,Wp,F,var);% Evaluate Wp (k)
        if (exitcode==NOTOK)
                fprintf('\n\n exitcode= %d \n\n',exitcode);
                return;
        end
        H1=H1-Wp;
        H2=H2-Wp;                             % Absolute acceleration of link 1 & 2 (l)
% ----------- Print  output  results ----------
        fprintf(out,'\n\n---  time:%4.3f   q: %9.6E   qp: %6.5E   qpp: %6.4f\n\n',t,q,qp,qpp);
        fprintm(out,' Position matrix of link 1',          m1);
        fprintm(out,' Absolute position matrix of link 2',  m2);
        fprintm(out,' Velocity matrix of link 1',          W1);
        fprintm(out,' Absolute velocity matrix of link 2',  W2);
        fprintm(out,' Acceleration matrix of link 1',      H1);
        fprintm(out,' Absolute acceleration matrix of link 2',H2);
        if (out==1)
          pause;
        else
          fprintf(1,'\n\n---  time:%4.3f   q: %9.6E   qp: %6.5E   qpp: %6.4f\n\n',t,q,qp,qpp);
        end
        dm=UNIT4 + W1*dt + 0.5 * H1*dt^2;     % Builds matrix dm = [1] + Wdt + 1/2 H dt^2  (m)
        dm=normal(dm);
        m1=dm*m1;                             % New position of link 1 (n)
        W1 =W1+Wp*dt;                         % New velocity of link 1 (o)
end                                           % end of main loop
fclose('all')
```

7.2.7 Source code of TEST_NEW.M

```
%_______________________________________________________________________________________________
%
%                                       PROGRAM TEST_NEW.M
%
% " Program for the trajectory prediction of a two-link system floating in the space".
%
```

```
% This program demonstrates the use of dyn_eq function for the solution of the direct dynamic
% problem of a two-link system floating in the 3-D space. The program predicts the trajectory
% of the system. This is an improved version of Test.m. The angular moment of the system should
% be constant, but inaccuracy in the integration method corrupt it. In this version, some
% statments have been added to preserve the total angular momentum obtaining an improved final
% accuracy. At each integration step the angular momentum is evaluated and a velocity dW is
% added to the system in order to set the value of the angular momentum equal to its initial
% value (G=Go for t==0.) The program reads from a file (TEST.DAT) the links description (mass,
% inertias, the coordinates of the centre of mass) and from another file (TEST.MOT) the motion
% of the motors, and prints on the screen the trajectory of the two links.
%
% (c) G.Legnani and C.Moiola 1998 adapted from G.Legnani and R.Faglia 1990
%___________________________________________________________________________________________

spheader
clc
                                                    % Initializations
Zax=Zaxis;
O=ORIGIN;
var=[1 1 1 1 1 1 ;
     0 0 0 0 0 0];
Wp=NULL4;
W0=NULL4;
                                                    % Open description file
fil=fopen([spc_lib_dir_b,'\test.dat'],'r');
if (fil==-1)
        error('Error on input file TEST.DAT ')
end
out=input('output to screen? (1=yes)');
if (out~=1)
   outfile=[spc_lib_dir_b,'\testout.out'];
   out=fopen(outfile,'wt');
end;
if (out==-1)
        error('Error in TEST.M: Unable to open otput file ')
end
                                                    % Read description of the links step(1)
m=  fscanf(fil,'%f',1);
jxx=fscanf(fil,'%f',1); jyy=fscanf(fil,'%f',1); jzz=fscanf(fil,'%f',1);
jxy=fscanf(fil,'%f',1); jyz=fscanf(fil,'%f',1); jxz=fscanf(fil,'%f',1);
xg= fscanf(fil,'%f',1); yg= fscanf(fil,'%f',1); zg= fscanf(fil,'%f',1);
                                                    % Builds Inertia Matrix of link 1
J1=jtoj(m,jxx,jyy,jzz,jxy,jyz,jxz,xg,yg,zg);
m=  fscanf(fil,'%f',1);
jxx=fscanf(fil,'%f',1); jyy=fscanf(fil,'%f',1); jzz=fscanf(fil,'%f',1);
jxy=fscanf(fil,'%f',1); jyz=fscanf(fil,'%f',1); jxz=fscanf(fil,'%f',1);
xg= fscanf(fil,'%f',1); yg= fscanf(fil,'%f',1); zg= fscanf(fil,'%f',1);
                                                    % Builds Inertia Matrix of link 2
J2=jtoj(m,jxx,jyy,jzz,jxy,jyz,jxz,xg,yg,zg);
                                                    % Read initial condition of the system
W1= fscanf(fil,'%f',[4 4]);                         % Read velocity matrix of link 1
W1=W1';
m1= fscanf(fil,'%f',[4 4]);                         % Read position matrix of link 1
m1=m1';
                                                    % Open motion file
fil=fopen([spc_lib_dir_b,'\test.mot'],'r');
if (fil==-1)
        error('Error on motion file TEST.MOT ')
end
dt=fscanf(fil,'%f',1);                              % Read integration step "dt"
for t=0:dt:(~feof(fil))                             % Loop for each istant of time step(2)
        q  =fscanf(fil,'%f',1);                     % Read motion of motor (a)
```

```
        qp =fscanf(fil,'%f',1);
        [qpp,count]=fscanf(fil,'%f',1);
                            % Check end of motion file. If motion file is empty -> end of loop
        if (count~=1)
                return
        end
        m12=screwtom(Zax,q,0,0);                    % Relative position of link 1 & 2 (b)
        m2=m1*m12;                                  % Absolute position of link 2     (c)
                                                    % step (d) moved forward
        [W12,H12]= vactowh2(Rev,Z,qp,qpp);          % Rel. vel & acc. of link 1&2 (e)
        W120=mami(W12,m1);                          % (f)
        H120=mami(H12,m1);
        W120(1:3,1:3)=normskew(W120(1:3,1:3),SKEW_);% normalization reducing num. error
        W2=W1+W120;                                 % Absolute velocity  of link 2 (g1)
                                            % evaluation of partial acceleration moved forward
        J10=mamt(J1,m1);                            % Refer inertia moments to absolute frame
        J20=mamt(J2,m2);
        J10=normskew(J10,SYMM_);                    % normalization reducing num. error
        J20=normskew(J20,SYMM_);                    % normalization reducing num. error
        Jtot=J10+J20;                               % Total inertia (i)
%---- preserve total angular momentum
                G1=skew(W1,J1);
                G2=skew(W2,J2);
                G=G1+G2;
                if (t==0) Go=G; end;
                [dW,G,exitcode]=dyn_eq(Jtot,W0,G-Go,var);
                W1=W1-dW;                           % correct velocity
                W2=W2-dW;
%----
                H1=W1^2;                            % Partial acceleration of link 1 (d)
        H2=coriolis(H1,H120,W1,W120);               % Partial acc. of link 2 (g2)
        F1=skew(H1,J10);                            % Evaluate inertia actions (j)
        F2=skew(H2,J20);
        F=F1+F2;
        [Wp,ff,exitcode]=dyn_eq(Jtot,Wp,F,var);     % Evaluate Wp (k)
        if (exitcode==NOTOK)
                fprintf('\n\n exitcode= %d \n\n',exitcode);
                return;
        end
        H1=H1-Wp;
        H2=H2-Wp;                                   % Absolute acceleration of link 1 & 2 (l)
% -----------  Print  output  results ----------
        fprintf(out,'\n\n---  time:%4.3f    q: %9.6E   qp: %6.5E  qpp: %6.4f\n\n',t,q,qp,qpp);
        fprintm(out,' Position matrix of link 1',           m1);
        fprintm(out,' Absolute position matrix of link 2',  m2);
        fprintm(out,' Velocity matrix of link 1',           W1);
        fprintm(out,' Absolute velocity matrix of link 2',  W2);
        fprintm(out,' Acceleration matrix of link 1',       H1);
        fprintm(out,' Absolute acceleration matrix of link 2',H2);
        if (out==1)
          pause;
        else
          fprintf(1,'\n---  time:%4.3f   q: %9.6E  qp: %6.5E  qpp: %6.4f\n',t,q,qp,qpp);
        end
                                             % Builds matrix dm = [1] + Wdt + 1/2 H dt^2  (m)
        dm=UNIT4 + W1*dt + 0.5 * H1*dt^2;
        dm=normal(dm);
        m1=dm*m1;                                  % New position of link 1 (n)
        W1 =W1+Wp*dt;                              % New velocity of link 1 (o)
end                                                % end of main loop
fclose('all')
```

7.3 Rototranslation

In this paragraph it is shown how to calculate the rototranslation (i.e. the axis of rototranslation) of the triangle which passes from the position (1) to the position (2) as in figure 7.7: Point P_1 moves to

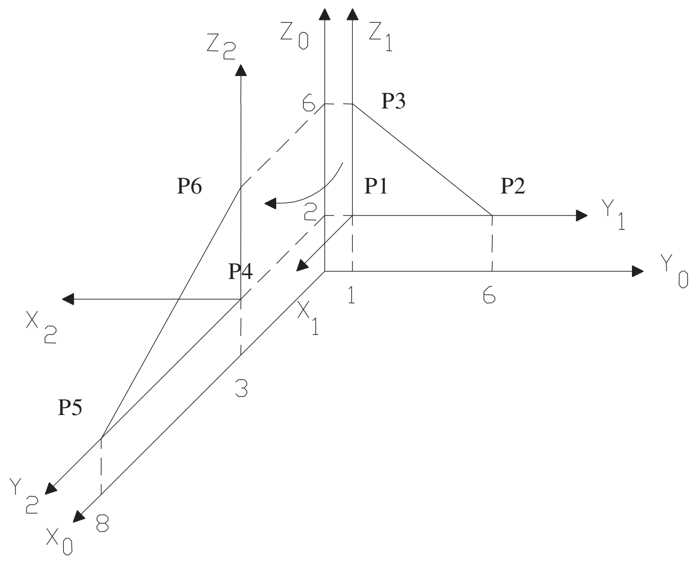


Figure 7.7: The frames definition for the example of rototranslation.

P_4, P_2 moves to P_5, P_3 moves to P_6. The frame attached to the triangle moves from $X_1Y_1Z_1$ to $X_2Y_2Z_2$. The position matrix of frame (2) with respect to reference frame (0) and of frame (0) with respect to (1) are

$$M_{1,0}=\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad M_{0,2}=\left[\begin{array}{ccc|c} 0 & 1 & 0 & 3 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.17}$$

and the desired rototranslation matrix is

$$Q_0 = M_{0,2} \cdot M_{1,0}=\left[\begin{array}{ccc|c} 0 & 1 & 0 & 2 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.18}$$

The rotation is clearly a rotation of $\pi/2$ about an axis anti-parallel to Z_0. Using SpaceLib© this result can be obtained with the following statements

```
spheader
clc
P1=[0 1 2 1 ]';
P2=[0 6 2 1 ]';
P3=[0 1 6 1 ]';
P4=[3 0 2 1 ]';
P5=[8 0 2 1 ]';
P6=[3 0 6 1 ]';
m01=frame4p(P1,P2,P3,Y,Z);
m02=frame4p(P4,P5,P6,Y,Z);
m10=invers(m01);
Q=m02*m10;
[u,fi,P,h]=mtoscrew(Q);
```

It gives the result:

Rotation angle	phi = $\pi/2$ = 1.57079
Axis direction	u = $[0\ 0\ \text{-}1]^t$
Point	P = $[1\ \text{-}1\ 0\ 1]^t$
Translation	h = 0

7.4 Scara robot

7.4.1 Theory in brief

This example shows how to solve the direct kinematic problem for the position and velocity of the Scara robot.

There are five reference frames. Frame (0) is the fixed frame, frame (1) is attached to the base while frames (2), (3) and (4) are embedded in link 1, 2 and 3 respectively. The auxiliary frame (a) is a moving frame whose origin is in the center of the gripper and whose axes are parallel to the reference frame (0).
The program described in § 7.4.2 is based on the conventions of the figure 7.4.1. The 1^{st} link of this Scara robot is 1.5 m long, while the 2^{nd} and the 3^{rd} link are 0.33 m long. Its joint variables Q and the first time derivative $\dot{Q}$ of Q is

$$Q = [\alpha, \beta, h*] = \left[\frac{\pi}{4}, \frac{\pi}{6}, \frac{1}{2}\right] \qquad \dot{Q} = \left[\frac{5\pi}{4}, \frac{5\pi}{4}, \frac{-1}{2}\right]$$

The position of frame (a) referred to frame (0) is expressed by the matrix

$$M_{0,a} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.319 \\ 0 & 1 & 0 & 0.552 \\ 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.19}$$

The velocity matrix of the center of the gripper in reference (0) is

$$W_{0,4(0)} = \left[\begin{array}{ccc|c} 0 & -7.854 & 0 & 0.916 \\ 7.854 & 0 & 0 & -0.916 \\ 0 & 0 & 0 & -0.5 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \tag{7.20}$$

The velocity matrix of the gripper in reference frame (a) is

$$W_{0,4(a)} = M_{a,0} \cdot W_{0,4(0)} \cdot M_{0,a} = \tag{7.21}$$

$$= \left[\begin{array}{ccc|c} 0 & -7.854 & 0 & -3.420 \\ 7.854 & 0 & 0 & 1.587 \\ 0 & 0 & 0 & -0.5 \\ \hline 0 & 0 & 0 & 0 \end{array}\right]$$

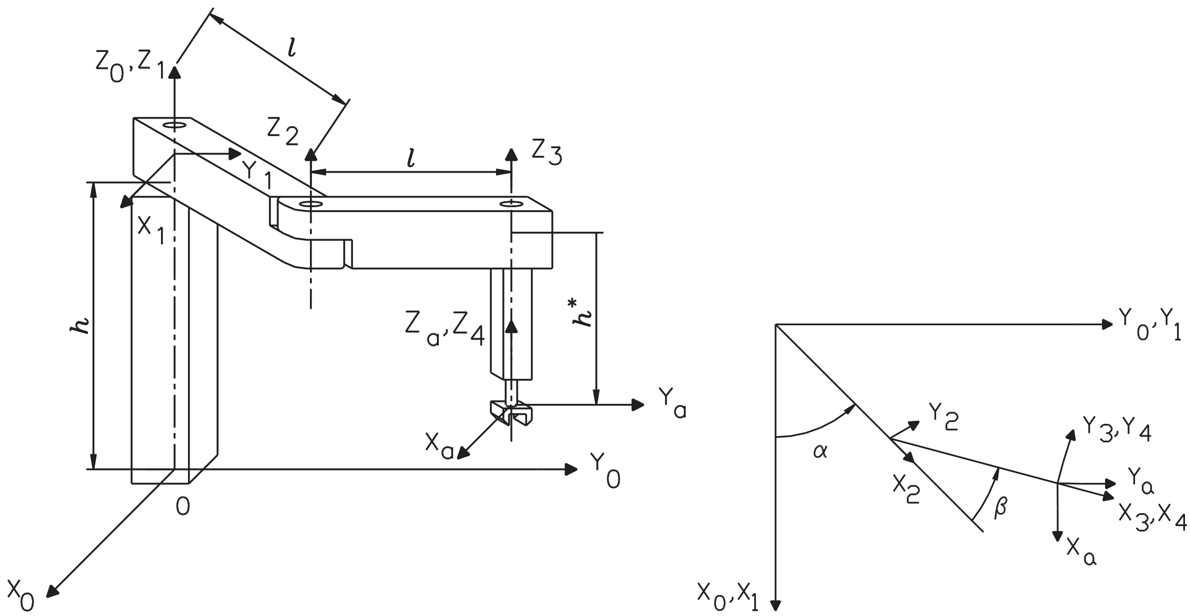


Figure 7.8: The frames definition for the example of robot Scara.

7.4.2 Listing of the program ROBSCARA.M

```
%--------------------------------------------------------------------------------------------
% ROBSCARA.M:  Sample program for direct kinematics of Scara robot
%             (See User's Manual)
%
%              University of Brescia
%              Mechanical Eng. Department
%              Via Branze 38
%              25123 BRESCIA - ITALY
%
%              giovanni.legnani@ing.unibs.it
%--------------------------------------------------------------------------------------------

spheader
clc

q = [pi/4  pi/6  0.5];              % joint variables array
qp= [pi*5/4 pi*5/4 -0.5];           % joint var. first time derivative
O=ORIGIN;
O1=[0   0   1.5   1]';              % origin of frame 1 in frame 0
O2=[0.33 0.  0  1]';                % origin of frame 2 in frame 1
O3=[0.33 0.  0  1]';                % origin of frame 3 in frame 2
O4=[0    0 -0.5 1]';                % origin of frame 4 in frame 3
Oa=[0. 0  1.5 1]';                  % origin of frame a in frame 0

m01=rotat34(Z,0,O1);                % builds relative position matrices
m12=rotat34(Z,q(1),O2);
m23=rotat34(Z,q(2),O3);
m34=rotat34(Z,0,O4);

m02=m01*m12;                        % builds absolute position matrices
m03=m02*m23;
m04=m03*m34;

m0a=idmat(4);                       % builds position matrix of frame
Oa=m04(:,4);                        % (a) in frame (0)
m0a(:,4)=Oa;

L12r=makel2(Rev,Z,0,O);             % builds relative L matrices
L23r=makel2(Rev,Z,0,O);
L34r=makel2(Pri,Z,0,O);

L12f= mami(L12r,m01);               % evaluate L matrices in frame (0)
L23f= mami(L23r,m02);
L34f= mami(L34r,m03);

W01=zeros(4);
                                    % builds relative velocity matrices
W12=L12f*qp(1);
W23=L23f*qp(2);
W34=L34f*qp(3);

W04=W01+W12+W23+W34;                % builds abs. W matrix of frame 4 in frame 0

W04a=miam(W04,m0a);                 % Evaluates W matrix of frame 4 in frame (a)

printm(' The absolute position matrix  of  the  gripper  is:  M04', m04);
printm(' The position matrix  of frame "a" referred to frame "O" is:  M0a', m0a);
printm(' The velocity matrix of the gripper in frame (0) is:  W04', W04);
printm(' The velocity matrix of the gripper in frame (a) is:  W04a', W04a);
```

7.5 Satellite

7.5.1 General information

This sample program demonstrates the use of several `SpaceLib`© functions described in this manual for the solution of a real problem. In practice, the program calculates how to obtain the best movement to spread out the antennas from a satellite.

7.5.2 Theory in brief

7.5.2.1 The problem For the installation of a satellite, launched with the rocket *Ariane*, it's necessary to spread out two antennas. During the launching phase, the antennas are bent and they are positioned as in fig.1 inside figure 7.13. When the service orbit has been reached, the antennas are spread out and they are oriented as in fig.2 inside figure 7.13. Each antenna reaches the service position through three subsequent rotations; considering the left antenna, these rototranslations are:

1) a rotation θ_1 around an axis passing through the point P_2 and orthogonal to the plane P_1-P_2-P_3: with this rotation the point P_1 of the antenna is aligned with the diagonal P_3-P_2 (fig.3 inside fig.7.13);

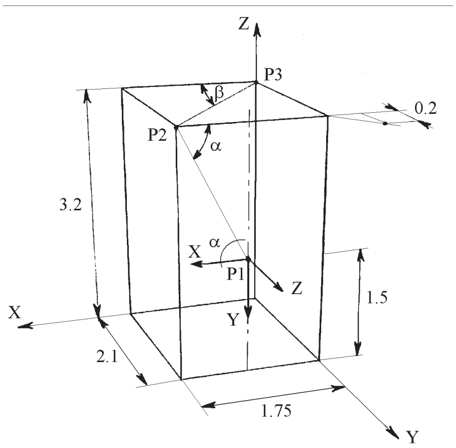


Figure 7.9: Dimensions of the rocket.

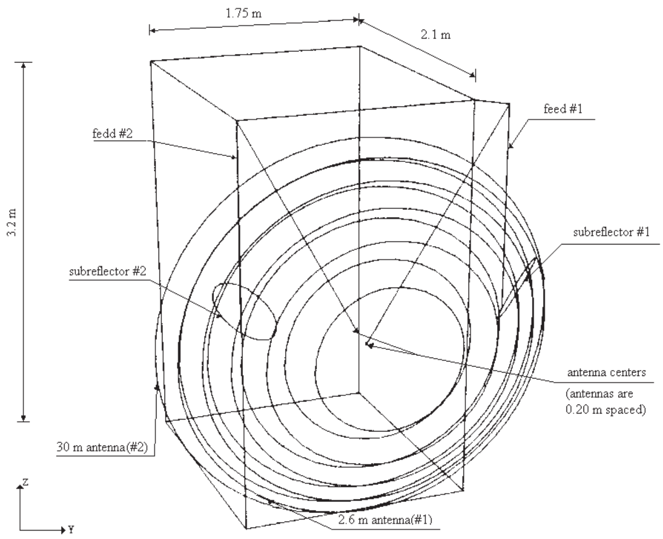


Figure 7.10: Antennas in the initial position.

2) a rotation θ_2 around an axis which coincides with the diagonal P_2-P_3 and which puts the concavity upwards (fig.4 and 5 inside figure 7.13);
3) a rotation θ_3 of 26 ° around an axis orthogonal to the diagonal; this rotation moves the antenna to its final configuration (fig.5 and 6 inside figure 7.13).

The problem was to work out if the antenna could be put in the correct position by means of just one single rototranslation.

7.5.2.2 The solution For geometrical properties is known that any combination of two or more rotations about the same point is equivalent to a "global" rotation. The following statements show how to evaluate the global rotation which allows to put in position each antenna with one single rotation movement. That's why just below are calculated the rototranslation axes of each antenna (direction cosines and a point of the axis), the rotation angles and the translations about these axes. The real dimensions of this satellite are presented in the figure 7.9, which also shows the reference frame of the rocket and the frame embedded on the antenna (initial position) (figures 7.9 and 7.10). Point P_1 is the center of left antenna, while point P_2 approximates the location of the hinge. Point P_2 can be ≅ 0.2 m higher or 0.3 m lower than the edge of the box. P_2 can also be ≅ 0.1 m outside the box. Angles α and β do not change during the whole movement, therefore they can be calculated by means of the following statements

```
α = atan2((3.2-1.5), (1.75/2)) = atan2 (1.7, 0.875) = 62.76° = 1.0953 rad
β = atan2 (2.1, 1.75) = 50.19° = 0.8759 rad
```

The sequence from the initial configuration to the final one is made up by the following steps:

- *Initial configuration* The initial position matrix of the frame of the left antenna is expressed by the matrix

$$M_i = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.875 \\ 0 & 0 & 1 & 2.1 \\ 0 & -1 & 0 & 1.5 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad (7.22)$$

- *Step 1*

  The left antenna turns about axis `Z2`, which is orthogonal to the plane containing points `P1`, `P2`, `P3`; point `P1` (the center of the antenna) get aligned with `P2` and `P3`. The right antenna turns about `Z1`, which is orthogonal to the `PA`, `PB`, `PC` plane.

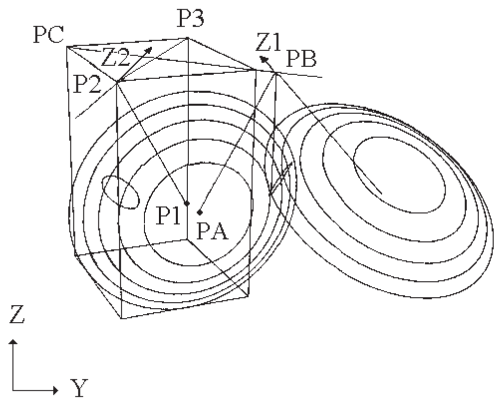


Figure 7.11: Step 1 of the antennas deployment.

- *Step 2*

  Both antennas have reached the right position relative to their feeds, which are locked onto antennas. The subreflectors still have to deploy and the antennas still have to complete their rotations.

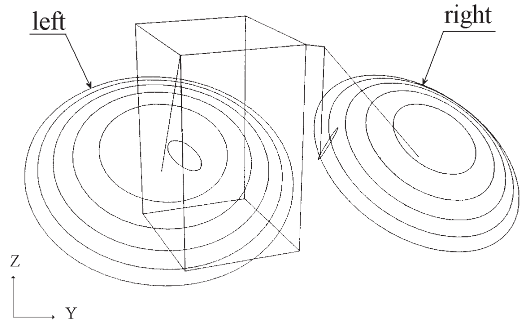


Figure 7.12: Step 2 of the antennas deployment.

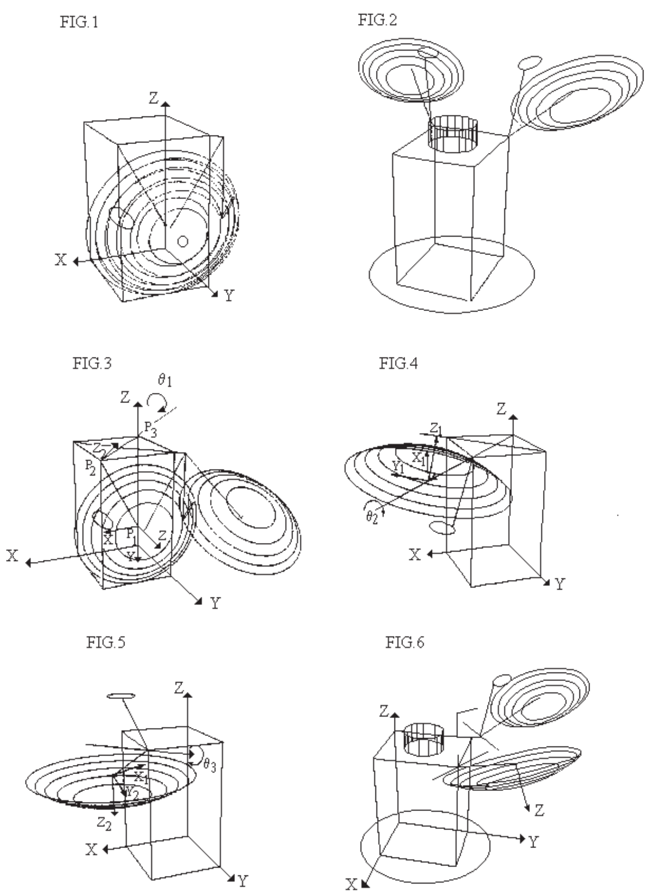


Figure 7.13: The phases of the antennas deployment.

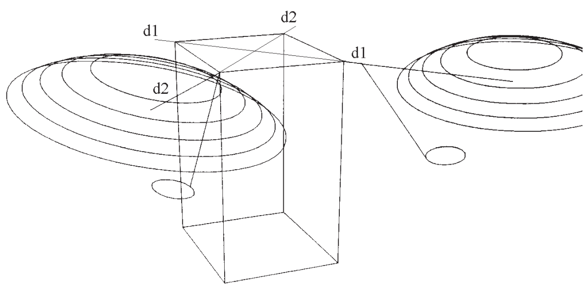
- *Step 3*

  The antennas have completed their rotations (reference lines are on spacecraft top diagonal `d1` and `d2`). The subreflectors have deployed and reached right positions relative to antennas and feeds. They are locked onto feeds. The new position of point `P1` is now:

$$\begin{aligned} P1\,(x) &= P2\,(x) + d \cdot \cos\beta \\ P1\,(y) &= P2\,(y) + d \cdot \sin\beta \\ P1\,(z) &= P2\,(z) \end{aligned} \tag{7.23}$$

while `d` is evaluated from figure 7.9 as the distance between `P2` and `P1`. The z axis of the moving frame is now pointing upwards (parallel to the rocket frame) while the direction of the others is presented in figure 6 inside figure 7.13. The position of the center of the left antenna at the end of step 3 is expressed by the following matrix

$$M_3 = \left[\begin{array}{ccc|c} -\cos(\alpha+\beta) & \sin(\alpha+\beta) & 0 & 2.97 \\ -\sin(\alpha+\beta) & -\cos(\alpha+\beta) & 0 & 3.57 \\ 0 & 0 & 1 & 3.20 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} 0.3899 & 0.9208 & 0 & 2.97 \\ -0.9208 & 0.3899 & 0 & 3.57 \\ 0 & 0 & 1 & 3.20 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.24}$$



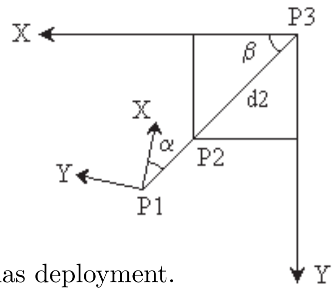


Figure 7.14: Step 3 of the antennas deployment.

- *Step 4*

  The antennas are turned face up through rotations of 180 ° about diagonals (`d1` and `d2`).

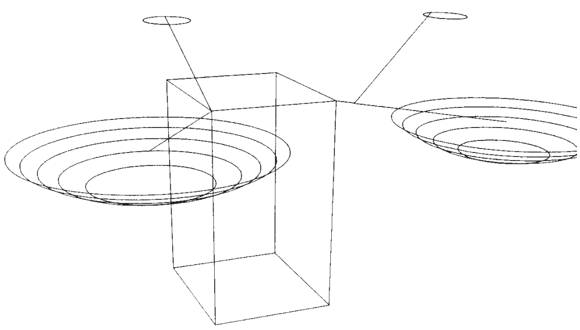


Figure 7.15: Step 4 of the antennas deployment.

The position of the center of the left antenna at the end of step 4 is expressed by the following matrix

$$M_4 = \left[\begin{array}{ccc|c} -\cos(\beta-\alpha) & -\sin(\beta-\alpha) & 0 & 2.97 \\ -\sin(\beta-\alpha) & \cos(\beta-\alpha) & 0 & 3.57 \\ 0 & 0 & -1 & 3.20 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} -0.976 & 0.218 & 0 & 2.97 \\ 0.218 & 0.976 & 0 & 3.57 \\ 0 & 0 & -1 & 3.20 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.25}$$

- *Step 5*

The antennas are filled 26° up to reach the working configurations, through a rotation about axes `n1-n1` (axes `n1-n1` and `n2-n2` are normal to axes `d1-d1` and `d2-d2`). The unit vector of the rotation axis (`d1-d1`) has the following cosines

$$u_x = \sin(\beta) = 0.768$$
$$u_y = -\cos(\beta) = -0.640$$
$$u_z = 0$$

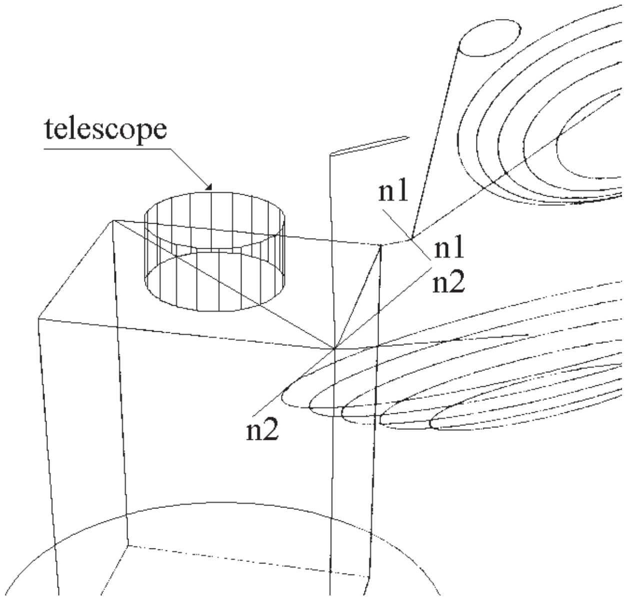


Figure 7.16: Step 5 of the antennas deployment.

The rotation axis passes through point `P2` and there is no translation along the axis.
So the Rototranslation matrix is

$$Q_5 = \left[\begin{array}{ccc|c} 0.959 & -0.050 & -0.281 & 1.075 \\ -0.050 & 0.940 & -0.337 & 1.290 \\ 0.281 & 0.337 & 0.899 & -0.874 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.26}$$

- *Final configuration*

The final position of the left antenna is expressed by the following matrix

$$M_f = Q_5 \cdot M_4 = \left[\begin{array}{ccc|c} -0.946 & 0.160 & 0.281 & 2.850 \\ 0.253 & 0.907 & 0.337 & 3.420 \\ -0.201 & 0.390 & -0.899 & 4.038 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.27}$$

The aim has now been reached:
the Rototranslation matrix which expresses the whole movement can be written as

$$Q_{tot} = M_f \cdot M_i^{-1} = \left[\begin{array}{ccc|c} -0.946 & 0.281 & -0.160 & 3.329 \\ 0.253 & 0.337 & -0.907 & 3.852 \\ -0.201 & -0.899 & -0.390 & 6.686 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{7.28}$$

From this matrix we can extract the axis unit vector whose cosines are

$$u_x = 0.1633 \qquad u_y = 0.8175 \qquad u_z = -0.5523$$

A point of the Rototranslation axis is `P2` $= [1.712, 1.908, 3.330, 1]^t$.

The rotation angle is 178.58° and there is no translation along the axis.

A program which performs and prints on the screen the presented calculations is listed in § 7.5.3.

7.5.3 Source code of SAT

(c) G.Legnani and D.Manara 2004 adapted from (c) G.Legnani 1998 and (c) G.Legnani and R.Faglia 1990

```
%---------------------------------------------------------------------------
%
% SAT.M
%
% Solution of  the  application example SAT described in the SPACELIB user's
% manual (page 113). This  program   evaluate   the   parameters   of   the
% rototraslation  necessary  to  orientate  an  antenna  mounted  on a space
% satellite.
%
% (c) G.Legnani 1998 adapted from (c) G.Legnani and R.Faglia 1990
%---------------------------------------------------------------------------


spheader

% Values of initial configuration
P1=[ 0.875 2.1 1.5 1]';
P2=[ 1.75  2.1 3.2 1]';
mi=[ 1  0  0  0.875; 0  0  1  2.1; 0 -1 0 1.5; 0 0 0 1];

alpha=atan2( P2(Z)-P1(Z) , P2(X)/2 );
beta= atan2( P2(Y),P2(X) );

sb=sin(beta); cb=cos(beta);
sb_a=sin(beta-alpha);   cb_a=cos(beta-alpha);

d=distp(P1,P2);

m4(X,X)=-cb_a; m4(X,Y)=-sb_a; m4(X,Z)=0; m4(X,U)=P2(X)+d*cb;
m4(Y,X)=-sb_a; m4(Y,Y)= cb_a; m4(Y,Z)=0; m4(Y,U)=P2(Y)+d*sb;
m4(Z,X)=0; m4(Z,Y)=0; m4(Z,Z)=-1; m4(Z,U)=P2(Z);
m4(U,X)=0; m4(U,Y)=0; m4(U,Z)=0; m4(U,U)=1;

% STEP 5

u5=[sb -cb 0]';

Q5=screwtom(u5,rad(26),P2,0);

% Final Configuration
mf=Q5*m4;

% Rototraslation
miinv=invers(mi);
Qtot=mf*miinv;
[utot,fi,P,h]=mtoscrew(Qtot);

clc

% ----- PRINT OUTPUT RESULTS
fprintf(1,'\n\n-------------------------      Results      -----------------------------\n');
printm('The rototraslation axis u is :       ',utot);
fprintf(1,'\n The rotation angle along axis u is:    %3.3f [deg]    %2.3f [rad]',deg(fi),fi)
fprintf(1,'\n\n The traslation along the axis u is :   %3.3f\n', h)
printm('The point P of the axis is:       ',P)
```

7.6 Elbow robot

7.6.1 General information

Three different sample programs are described. They deal with the kinematics of a serial manipulator and present the use of many `SpaceLib`© functions. The robot under study is a 6 d.o.f. Elbow robot (see

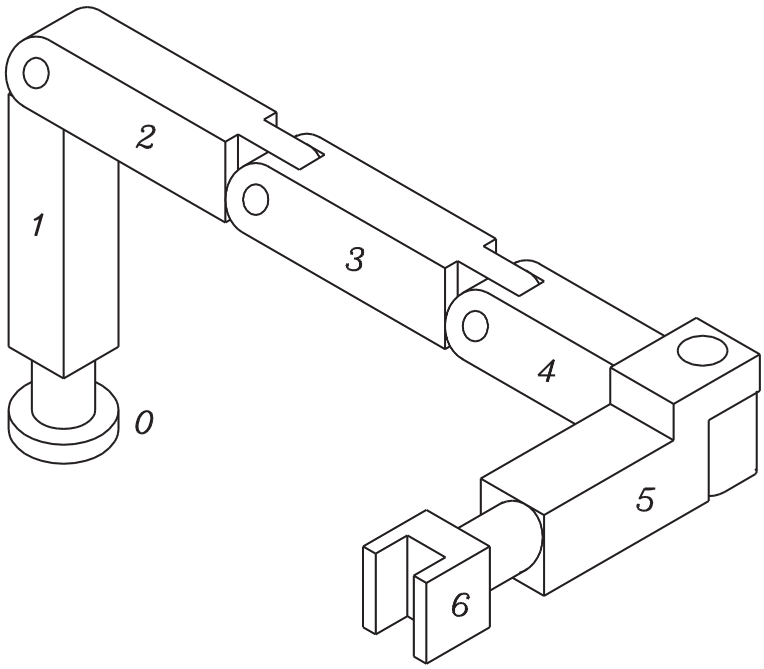


Figure 7.17: Kinematic structure of the `Elbow` robot.

[16], [3]). The robot has six revolute joints. This section contains two programs obtained with different approaches for the study of the direct kinematics of the robot (`ELB_D_DH` and `ELB_D_PA`) and one based on a numerical approach for the inverse kinematics (`ELB_I_DH`). The two programs for the direct kinematics accept the same input and produce identical output. The programs for the direct kinematics evaluates the gripper motion starting from the joint motions, while the program for the inverse kinematics is able to evaluate the joint motions necessary to produce an assigned gripper motion. The output file of the program for the inverse kinematics can be used as input for the programs for the direct kinematics and vice versa. These facts are summarized in the scheme of figure 7.18. In the previous scheme `gripper1.mot`

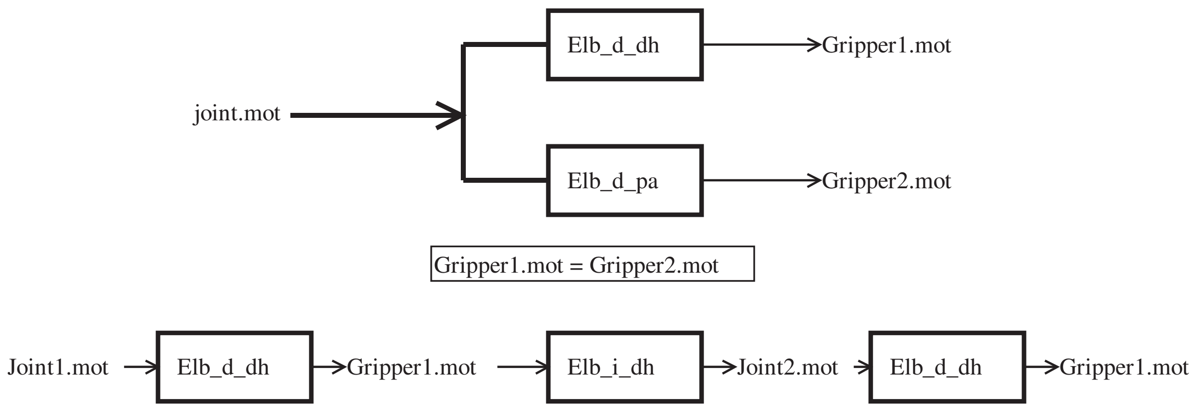


Figure 7.18: Input output files for Elbow robot

is identical to `gripper2.mot`. Since a robot can have many inverse solutions the program evaluates just one of them and in the scheme `joint1.mot` could be different from `joint2.mot`.

7.6.2 Format of the data and motion files

The SpaceLib© contains an example of input/output files (ELBOW.DAT, JOINT.MOT, GRIPPER.MOT and GUESS.1ST) for the described programs. Their format is presented in table 7.8. The input file ELBOW.DAT

	ELB_D_DH ELB_D_PA	ELB_I_DH
ELBOW.DAT	Input	Input
JOINT.MOT	Input	Output
GRIPPER.MOT	Output	Input
GUESS.1^{ST}	*Not used*	Input

Table 7.8: Input/Output files for the ELBOW ROBOT

which describes the link lengths has the format show in table 7.10 The file JOINT.MOT which describes the joint motions has the format described in table 7.9. The file GRIPPER.MOT which describes the gripper

JOINT.MOT			*Meaning*
0.01			dt
0.00114	0.25450	28.27440	Rotation, speed, acceleration of 1^{st} motor
0.00106	0.23560	26.17990	Rotation, speed, acceleration of 2^{nd} motor
0.00153	0.33930	37.69910	Rotation, speed, acceleration of 3^{rd} motor
-0.00153	-0.33930	-37.69910	Rotation, speed, acceleration of 4^{th} motor
0.00358	0.79520	88.35730	Rotation, speed, acceleration of 5^{th} motor
0.02863	6.36170	706.85828	Rotation, speed, acceleration of 6^{th} motor
0.00510	0.53720	28.27440	Rotation, speed, acceleration of 1^{st} motor
0.00473	0.49740	26.17990	Rotation, speed, acceleration of 2^{nd} motor
...	...	...	....

Table 7.9: Content of the file JOINT.MOT

ELBOW.DAT	*Meaning*
1.5	length of link 1
0.8	length of link 2
0.8	length of link 3
0.2	Length of link 4
0.0	Length of link 5
0.2	Length of link 6

Table 7.10: Content of the file ELBOW.DAT

motion has the format specified by the table 7.11 where α, β, γ denote the gripper orientation using an appropriate Cardan/Euler convention; X, Y, Z are the gripper position Single quote and double quote mark the time derivative. The file GUESS.1ST contains the value of the joint rotations for the first guess when solving the inverse kinematics problem. It has the simple format of table 7.12.

7.6.3 The file Joint.mot

This section describes the criteria under which the sample file JOINT.MOT has been created[3]. To study the robot movement, for each link a symmetrical law of motion at constant acceleration of the kind

[3]The first point of the joint motions hasn't been stored in JOINT.MOT because the robot is in a singular configuration at time t=0.

GRIPPER.MOT			*Meaning*
0.01			dt
0	1	2	Cardan/Euler convention used X=0, Y=1, Z=2 for compatibility with the C-language version
0.029690	0.000136	0.004718	α, β, γ, t=0
6.597086	0.060601	1.048386	α', β', γ'
732.919800	20.197580	115.902000	α", β", γ"
0.197946	1.800940	1.503133	X, Y, Z
-0.459088	0.207974	0.695856	X', Y', Z'
-51.222855	22.670288	77.394531	X", Y", Z"
0.132292	0.002702	0.020876	α, β, γ, t=dt
13.918917	0.567294	2.161054	α', β', γ'
.....	.....	.....	.....

Table 7.11: Content of the file GRIPPER.MOT

GUESS.1ST	*Meaning*
.0 .1 .2 .0 .1 .0	q_1 q_2 q_3 q_4 q_5 q_6

Table 7.12: Content of the file GUESS.1ST

1/3 -1/3 -1/3 has been considered (see. figure 7.19). The notation 1/3 -1/3 -1/3 indicates that the movements consist of three parts (acceleration, constant speed, deceleration) of identical duration. The joint motions are stored with a time step ΔT=0.01 seconds.

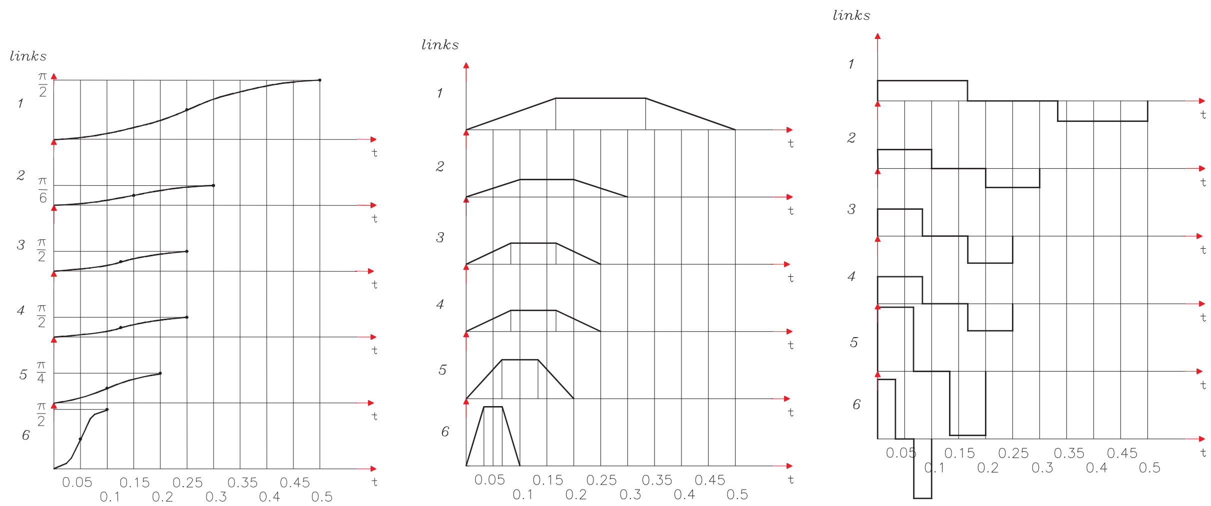


Figure 7.19: The law of motion contained in file JOINT.MOT.

7.6.4 Direct kinematics

The two sample programs here presented are: ELB_D_DH and ELB_D_PA; the first one is based on the *Denavit & Hartemberg* notation [1] and the relative frames are positioned according to figure 7.20 while the second one is very similar but the relative frames are positioned as described in figure 7.21. The two programs accept the same input files and produce identical output.

Both programs display the gripper motion to the screen using the position, the velocity and the acceleration matrices. Velocity and acceleration are expressed in a *auxiliary frame* (parallel to the base frame) whose origin is in the TCP (*gripper center*). The auxiliary frame is not shown in the figures. The base frame Xa, Ya, Za and the gripper frame Xb, Yb, Zb are identical for the two programs, while the intermediate frames have been positioned using different approaches.

The gripper motion is also stored in a output file. The gripper position is represented by the TCP position, velocity and acceleration, while the orientation is stored as Cardan angles and their time derivatives; the chosen sequence of rotation is *rot* X, *rot* Y, *rot* Z (see also § 3.2).

7.6.5 The sample program ELB_D_DH

Frame positions

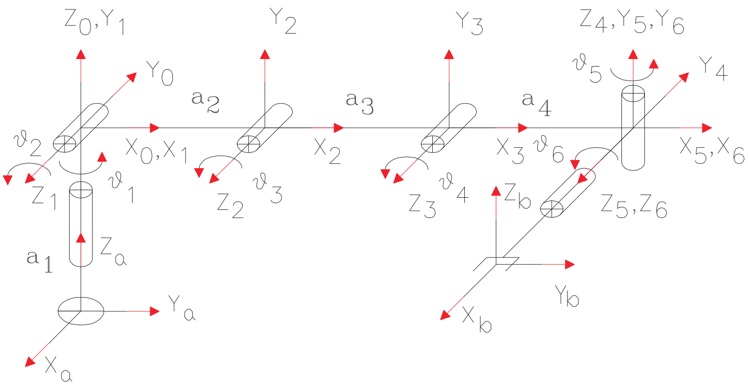


Figure 7.20: Frames definition for the example of *elbow* robot with the program ELB_D_DH.

The base frame is Xa, Ya, Za which does not move with respect to X0, Y0, Z0. The gripper frame Xb, Yb, Zb does not move with respect to X6, Y6, Z6. The reference frames from X1, Y1, Z1 to X6, Y6, Z6 attached to the links are positioned following the *Denavit* and *Hartenberg* convention (see also [1]) and so at the end of the links.

The program ELB_D_DH

```
%_______________________________________________________________________________________________
%
%                                                   ELB_D_DH.M
% Program for the DIRECT kinematics of ELBOW robot. Frames assigned according to Denavit and
% Hartenberg conventions. The output of this program is compatible with the input of elb_i_dh.c
% The input  of this program is compatible with the output of elb_i_dh.c
% Input file:  ELBOW.DAT and  JOINT.MOT.
% Output file: GRIPPER.MOT
%                                                                   (c) G. Legnani, C. Moiola 1998
%_______________________________________________________________________________________________

spheader
MAXLINK=6;
clc

ii=X;                  % Euler/Cardan convention
jj=Y;                  % for gripper
kk=Z;                  % angular position
theta=zeros([1,MAXLINK+1]);                       % Denavit & Hartemberg's parameters (D&H)
d    =zeros([1,MAXLINK+1]);
a    =zeros([1,MAXLINK+1]);
b    =zeros([1,MAXLINK+1]);
fi   =[0 PIG_2 0 0 3*PIG_2 PIG_2 0];
                                                    % Matrices initializations and declarations
mabs   =zeros(4,4*(MAXLINK+1)); % array containing abs. pos.mat.of frame (i) in frame (0)
Wabs   =zeros(4,4*(MAXLINK+1)); % array containing abs. vel.mat.of frame (i) in frame (0)
Habs   =zeros(4,4*(MAXLINK+1)); % array containing abs. vel.mat.of frame (i) in frame (0)
```

```
mreli_1=zeros(4,4*(MAXLINK+1)); % array containing pos. mat. of frame (i) seen in frame (i-1)
Wreli_1=zeros(4,4*(MAXLINK+1)); % array containing rel.vel.mat.of frame (i) seen in frame (i-1)
Hreli_1=zeros(4,4*(MAXLINK+1)); % array containing rel.acc.mat.of frame (i) seen in frame (i-1)
Wrel0  =zeros(4,4*(MAXLINK+1)); % array containing rel.vel.mat.of frame (i) seen in frame (0)
Hrel0  =zeros(4,4*(MAXLINK+1)); % array containing rel.acc.mat.of frame (i) seen in frame (0)
Last =[0 1 0 0;       % transformation matrix from frame (6) to gripper  element Z-U is in a[7]
       0 0 1 0;
       1 0 0 0;
       0 0 0 1 ];
first=ORIGIN;                    % origin of frame 0 with respect to base,Z value is in a[1]
string1=input('Digit the name of the input DATA FILE: ','s');
data=fopen(string1,'r');
if (data==-1)
      error('Error in ELB_D_DH.M, unable to open DATA FILE ')
end
string2=input('Digit the name of the input MOTION FILE: ','s');
motion=fopen(string2,'r');
if (motion==-1)
      error('Error in ELB_D_DH.M, unable to open the MOTION FILE ')
end
string3=input('Digit the name of the OUTPUT FILE (S=Screen): ','s');
string3=upper(string3);
if (string3=='S')
      out=1;
else
      out=fopen(string3,'wt');
end
if (out==-1)
      error('Error in ELB_D_DH.M, unable to open OUTPUT FILE ')
end
a(1)=0;
for i=2:1:MAXLINK+1                           %MAXLINK+1    % read link lenghts from data file
      a(i)=fscanf(data,'%f',1);
end
first(Z)=a(2);
mabs(:,1:4)=rotat24(Z,PIG_2,first);           % pos. mat. of frame 0 from base frame */
Last(Z,U)=a(7);                               % gripper position in frame 6
Aus=UNIT4;
a(2)=0;                                       % D&H parameter 'a' of link 1 and link 6 are zero
a(7)=0;
dt=fscanf(motion,'%f',1);                     % read time step from motion file
fprintf(out,'\n%f',dt);                       % write dt to out file
fprintf(out,'\n%d %d %d\n\n',ii-1,jj-1,kk-1); % write Cardan convention to out file
time=0;
while ~feof(motion)
      for i=1:1:MAXLINK
            p=4*i-3;
            pp=[p:p+3];
             q=         fscanf(motion,'%f',1);                              % read joint motion
             qp=        fscanf(motion,'%f',1);
            [qpp,count]=fscanf(motion,'%f',1);
            if count~=1 break, end
                                                                % Builds relative position matrix
            mreli_1(:,pp)=dhtom(Rev,theta(i),d(i),b(i),a(i+1),fi(i+1),q);
                           % Builds relative velocity and acceleration matrix in local frame(4)
            [ Wreli_1(:,pp),Hreli_1(:,pp) ]=veactowh(Rev,qp,qpp);
            mabs(:,pp+4)=mabs(:,pp)*mreli_1(:,pp);      % Absolute position matrix of frame (i)
            Wrel0(:,pp)=mami(Wreli_1(:,pp),mabs(:,pp));         % W and H matrices in frame (i)
            Hrel0(:,pp)=mami(Hreli_1(:,pp),mabs(:,pp));
            Wabs(:,pp+4)=Wabs(:,pp)+Wrel0(:,pp);           % Evaluates absolute velocity matrix
                                                        % Evaluates absolute acceleration matrix
            Habs(:,pp+4)=coriolis(Habs(:,pp),Hrel0(:,pp),Wabs(:,pp),Wrel0(:,pp));
```

```
      end                                                                       % end on MAXLINK loop
    if count~=1 break, end
      gripper=mabs(:,pp+4)*Last;                                   % gripper position mabs(:,pp+4)
      Aus(X,U)=gripper(X,U);
      Aus(Y,U)=gripper(Y,U);
      Aus(Z,U)=gripper(Z,U);
      Waus=NULL4; Haus=NULL4;
      Waus=miam(Wabs(:,pp+4),Aus);                                              % transform velocity
      Haus=miam(Habs(:,pp+4),Aus);                          % and acceleration in ausiliar frame
                              % extracts Cardan angles (and their time derivatives) of gripper
      [q1,q2,qp1,qp2,qpp1,qpp2]=htocarda(gripper,Waus,Haus,ii,jj,kk);
      fprintf(1,'\nTime=%f\n',time);                       % Print Output Results only on the screen
      printm('The position matrix of the gripper is:',gripper);
      printm('The velocity matrix of the gripper is:',Waus);
      printm('The acceleration matrix of the gripper is:',Haus);
      fprintf('\n\nPress any key to continue\n\n');
      pause;
      fprintf(out,'\n');                             % Print Output Results on the screen or in a FILE
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',  q1(X),  q1(Y),  q1(Z));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f', qp1(X), qp1(Y), qp1(Z));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',qpp1(X),qpp1(Y),qpp1(Z));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',gripper(X,U),gripper(Y,U),gripper(Z,U));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',Waus(X,U),Waus(Y,U),Waus(Z,U));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',Haus(X,U),Haus(Y,U),Haus(Z,U));
    time=time+dt;
end                                                                                % end main loop
fclose('all');
```

7.6.6 The sample program ELB_D_PA

Frame positions

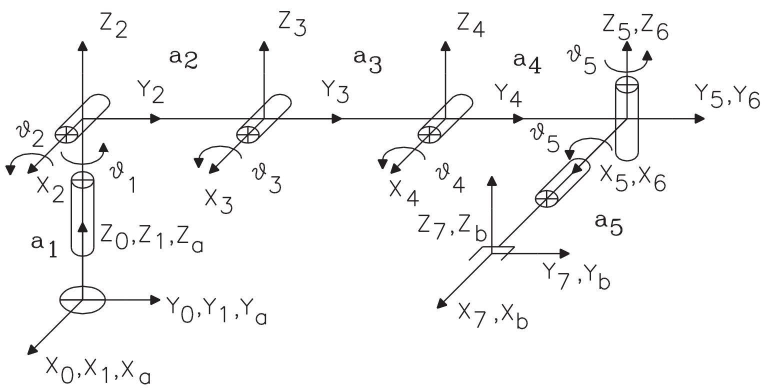


Figure 7.21: Frames definition for the example of *elbow* robot with the program `ELB_D_DH.C`.

The reference frame are attached to the links in such a way that in the "home" position ($q_1 = q_2 = \ldots = q_6 = 0$) the frames are all parallel to each other. The frames are positioned at the beginning of the links. Different from the first case, the frames result to be seven. The frames #6 and #7 move together but they have different positions. The base frame `Xa`, `Ya`, `Za` coincides with `X0`, `Y0`, `Z0`. The gripper frame `Xb`, `Yb`, `Zb` coincides with `X7`, `Y7`, `Z7`.

The program ELB_D_PA

```
%_____________________________________________________________________________________________
%                                           ELB_D_PA.M
% Program for the DIRECT kinematics of ELBOW robot. Frames assigned according to Denavit and
% Hartenberg conventions. The output of this program is compatible with the input of elb_i_dh.m
% The input  of this program is compatible with the output of elb_i_dh.m
% Input file:  ELBOW.DAT and  JOINT.MOT.                                Output file: GRIPPER.MOT
% (c) G. Legnani, C. Moiola 1998
%_____________________________________________________________________________________________

spheader
MAXLINK=6;
clc
axis=[U Z X X X Z X U];
ii=X;                % Euler/Cardan convention
jj=Y;                % for gripper
kk=Z;                % angular position
a  =zeros([1,MAXLINK+2]);                          % Denavit & Hartemberg's parameters (D&H)
O  =zeros([MAXLINK+2,U]);
                                                 % Matrices initializations and declarations
mabs   =zeros(4,4*(MAXLINK+2));     % array containing abs. pos. mat. of frame (i) in frame (0)
Wabs   =zeros(4,4*(MAXLINK+2));     % array containing abs. vel. mat. of frame (i) in frame (0)
Habs   =zeros(4,4*(MAXLINK+2));     % array containing abs. vel. mat. of frame (i) in frame (0)
mreli_1=zeros(4,4*(MAXLINK+2));   % array containing pos. mat. of frame (i) seen in frame (i-1)
Wreli_1=zeros(4,4*(MAXLINK+2)); % array containing rel.vel.mat.of frame (i) seen in frame (i-1)
Hreli_1=zeros(4,4*(MAXLINK+2)); % array containing rel.acc.mat.of frame (i) seen in frame (i-1)
Wrel0  =zeros(4,4*(MAXLINK+2));   % array containing rel.vel.mat.of frame (i) seen in frame (0)
Hrel0  =zeros(4,4*(MAXLINK+2));   % array containing rel.acc.mat.of frame (i) seen in frame (0)

string1=input('Digit the name of the input DATA FILE: ','s');
data=fopen(string1,'r');
if (data==-1)
      error('Error in ELB_D_DH.M, unable to open DATA FILE ')
end
string2=input('Digit the name of the input MOTION FILE: ','s');
motion=fopen(string2,'r');
if (motion==-1)
      error('Error in ELB_D_DH.M, unable to open the MOTION FILE ')
end
string3=input('Digit the name of the OUTPUT FILE (S=Screen): ','s');
string3=upper(string3);
if (string3=='S')
      out=1;
else
      out=fopen(string3,'wt');
end
if (out==-1)
      error('Error in ELB_D_DH.M, unable to open OUTPUT FILE ')
end

for i=2:1:MAXLINK+1                                          % read link lenghts from data file
      a(i)=fscanf(data,'%f',1);
end
O(1,X)=0.;   O(1,Y)=0.;   O(1,Z)=0.;   O(1,U)=1.;          % rel. origin of frame (i) in (i-1)
O(2,X)=0.;   O(2,Y)=0.;   O(2,Z)=a(2); O(2,U)=1.;
O(3,X)=0.;   O(3,Y)=a(3); O(3,Z)=0.;   O(3,U)=1.;
O(4,X)=0.;   O(4,Y)=a(4); O(4,Z)=0.;   O(4,U)=1.;
O(5,X)=0.;   O(5,Y)=a(5); O(5,Z)=0.;   O(5,U)=1.;
O(6,X)=0.;   O(6,Y)=0.;   O(6,Z)=0.;   O(6,U)=1.;
O(7,X)=a(7); O(7,Y)=0.;   O(7,Z)=0.;   O(7,U)=1.;
Aus=UNIT4;
```

```
mabs(:,1:4)=UNIT4;
dt=fscanf(motion,'%f',1);                                      % read time step from motion file
fprintf(out,'\n%f',dt);                                                 % write dt to out file
fprintf(out,'\n%d %d %d\n\n',ii-1,jj-1,kk-1);             % write Cardan convention to out file
time=0;
while ~feof(motion)
      for i=1:1:MAXLINK
            p=4*i-3;
            pp=[p:p+3];
            q=fscanf(motion,'%f',1);                                          % read joint motion
            qp=fscanf(motion,'%f',1);
            [qpp,count]=fscanf(motion,'%f',1);
            if count~=1 break, end
            mreli_1(:,pp)=rotat24(axis(i+1),q,O(i,X:U));      % Builds relative position matrix
                           % Builds relative velocity and acceleration matrix in local frame(4)
            [ Wreli_1(:,pp),Hreli_1(:,pp) ]=vactowh3(Rev,axis(i+1),qp,qpp,O(i,X:U));
            mabs(:,pp+4)=mabs(:,pp)*mreli_1(:,pp);      % Absolute position matrix of frame (i)
            Wrel0(:,pp)=mami(Wreli_1(:,pp),mabs(:,pp));         % W and H matrices in frame (i)
            Hrel0(:,pp)=mami(Hreli_1(:,pp),mabs(:,pp));
            Wabs(:,pp+4)=Wabs(:,pp)+Wrel0(:,pp);          % Evaluates absolute velocity matrix
                                                       % Evaluates absolute acceleration matrix
            Habs(:,pp+4)=coriolis(Habs(:,pp),Hrel0(:,pp),Wabs(:,pp),Wrel0(:,pp));
      end                                                                 % end on MAXLINK loop
    if count~=1 break, end
      mreli_1(:,pp+4)=UNIT4;
      mreli_1(X,p+7)=O(7,X);
      Wreli_1(:,pp+4)=NULL4;
      Hreli_1(:,pp+4)=NULL4;
      mabs(:,pp+8)=mabs(:,pp+4)*mreli_1(:,pp+4);
      Wrel0(:,pp+4)=mami(Wreli_1(:,pp+4),mabs(:,pp+4));         % W and H matrices in frame (i)
      Hrel0(:,pp+4)=mami(Hreli_1(:,pp+4),mabs(:,pp+4));
      Wabs(:,pp+8)=Wabs(:,pp+4)+Wrel0(:,pp+4);            %  Evaluates absolute velocity matrix
                                                       % Evaluates absolute acceleration matrix
      Habs(:,pp+8)=coriolis(Habs(:,pp+4),Hrel0(:,pp+4),Wabs(:,pp+4),Wrel0(:,pp+4));
                              % extracts Cardan angles (and their time derivatives) of gripper
      [q1,q2,qp1,qp2,qpp1,qpp2]=htocarda(mabs(:,pp+8),Wabs(:,pp+8),Habs(:,pp+8),ii,jj,kk);
      Aus(X,U)=mabs(X,p+11);
      Aus(Y,U)=mabs(Y,p+11);
      Aus(Z,U)=mabs(Z,p+11);
      Waus=miam(Wabs(:,pp+4),Aus);                                          % transform velocity
      Haus=miam(Habs(:,pp+4),Aus);                            % and acceleration in ausiliar frame
                              % extracts Cardan angles (and their time derivatives) of gripper
      [q1,q2,qp1,qp2,qpp1,qpp2]=htocarda(mabs(:,pp+8),Waus,Haus,ii,jj,kk);
      fprintf(1,'\nTime=%f\n',time);                    % Print Output Results only on the screen
      printm('The position matrix of the gripper is:',mabs(:,pp+8));
      printm('The velocity matrix of the gripper is:',Waus);
      printm('The acceleration matrix of the gripper is:',Haus);
      fprintf('\nPress any key to continue\n\n');
      pause;
    fprintf(out,'\n');                          % Print Output Results on the screen or in a FILE
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',  q1(X),  q1(Y),  q1(Z));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f', qp1(X), qp1(Y), qp1(Z));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',qpp1(X),qpp1(Y),qpp1(Z));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',mabs(X,p+11),mabs(Y,p+11),mabs(Z,p+11));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',  Waus(X,U),Waus(Y,U),Waus(Z,U));
      fprintf(out,'\n%7.6f      %7.6f      %7.6f',Haus(X,U),Haus(Y,U),Haus(Z,U));
    time=time+dt;
end                                                                             % end main loop
fclose('all');
```

7.6.7 Inverse kinematics

The program for the inverse kinematics reads the robot description and the requested motion for the gripper producing the correspondent joints motion.The files used are:

`GRIPPER.MOT` is a file with the same format as the output file obtained from the sample programs for the direct kinematics.

`JOINT.MOT` is a file with the same format as the input file for the sample programs for the direct kinematics. This file will contain the joint motion. This output file has a format compatible with the input files for the direct kinematics.

`GUESS.1ST` is a file containing 6 value to be used as first guess for the iterative process which evaluates the joint angles.

To run the program, just type in `MATLAB` Command Window, `ELB_I_DH`, so the program asks for the other input and output files, that must be typed in at the prompt. For example:

```
'' ELB_I_DH
'' Type in the name of the input DATA file:
ELBOW.DAT
'' Type in the name of the input MOTION file:
GRIPPER.MOT
'' Type in the name of the input GUESS file:
GUESS.1ST
'' Type in the name of the OUTPUT file (S=screen):
JOINT.MOT
```

7.6.8 The sample program ELB_I_DH.

```
%_______________________________________________________________________________________________
%               ELB_I_DH.m program for the INVERSE kinematics of ELBOW robot.
%
% Frames assigned according to Denavit and Hartenberg conventions. The output of this program
% is compatible with the input  of elb_d_dh.m The input  of this program is compatible with the
% output of elb_d_dh.m
%   Input file:   ELBOW.DAT GRIPPER.MOT GUESS.1ST                       Output file:  JOINT.MOT
% (c) G. Legnani, C. Moiola 1998
%_______________________________________________________________________________________________

clear
spheader
MAXLINK=6;
theta= zeros(1,MAXLINK+1);                                 % Denavit & Hartemberg's parameters
d    = zeros(1,MAXLINK+1);
b    = zeros(1,MAXLINK+1);
fi   = [0 PIG_2 0 0 3*PIG_2 PIG_2 0];
a    = zeros(1,MAXLINK+1);
q    = zeros(1,MAXLINK);                                   % joint angles
qp   = zeros(1,MAXLINK);                                   % array of joint vel. variables
qpp  = zeros(1,MAXLINK);                                   % array of jint acc. variables
%ds   = zeros(1,MAXLINK);                                  % sol. of the eq. J*dq=ds
dq   = zeros(1,MAXLINK);                                   % sol. of Newton/Raphson alg. step
buf  = zeros(1,MAXLINK);
toll=0.0005;                                               % precision of the solution
maxiter=15;                                                % max. num. of iter. in N-R alg.
orig=ORIGIN;
mrelp_1=zeros(4,4*(MAXLINK+1));    % array containing pos. mat. of frame(p) seen in frame (p-1)
Wrelp_1=zeros(4,4*(MAXLINK+1));  %array containing rel.vel.mat.of frame (p) seen in frame (p-1)
Hrelp_1=zeros(4,4*(MAXLINK+1));  %array containing rel.acc.mat.of frame (p) seen in frame (p-1)
```

```
mabs=zeros(4,4*(MAXLINK+1));       % array containing abs. pos. mat. of frame (p) in base frame
Wabs=zeros(4,4*(MAXLINK+1));       % array containing abs. vel. mat. of frame (i) in base frame
Habs=zeros(4,4*(MAXLINK+1));       % array containing abs. acc. mat. of frame (i) in base frame
mabsinv=NULL4;    % invers position matrix of the frame positioned in the center of the gripper
Lrelp=NULL4;                       % L relative matrix of p-th joint seen in frame (p-1)
Lrel0=NULL4;                       % L relative matrix of p-th joint seen in base frame
Wrel0=zeros(4,4*(MAXLINK+1));    % array containing rel.vel.mat.of frame (p) seen in base frame
Hrel0=zeros(4,4*(MAXLINK+1));    % array containing rel.acc.mat.of frame (p) seen in base frame
Wtar=NULL4;                        % target velocity matrix
Htar=NULL4;                        % target acceleration matrix
dH=NULL4;                          % Htar - H~    H~ is the acceleration evaluated with qpp=0
Aus =UNIT4;
Waus=NULL4;
Haus=NULL4;
Wabs(:,1:4)=NULL4;
Habs(:,1:4)=NULL4;
gripper=zeros(4,4);
Last =[ 0 1 0 0 ;                  % transformation matrix from
        0 0 1 0 ;                  % frame (6) to gripper
        1 0 0 0 ;                  % element Z-U is in a(6)
        0 0 0 1 ];
first=ORIGIN;                      % origin of frame 0 with respect to base, Z value is in a(1)

string1=input('Digit the name of the input DATA FILE: ','s');
data=fopen(string1,'r');
if (data==-1)
      error('Error in ELB_I_DH.M, unable to open DATA FILE ')
end
string2=input('Digit the name of the input MOTION FILE: ','s');
motion=fopen(string2,'r');
if (motion==-1)
      error('Error in ELB_I_DH.M, unable to open the MOTION FILE ')
end
string3=input('Digit the name of the GUESS  FILE ','s');
guess=fopen(string3,'r');
if (guess==-1)
      error('Error in ELB_I_DH.M, unable to open 1ST_GUESS FILE ')
end
string4=input('Digit the name of the OUTPUT  FILE (S=Screen): ','s');
string4=upper(string4);
if (string4=='S')
      out=1;
else
      out=fopen(string4,'wt');
end
if (out==-1)
      error('Error in ELB_I_DH.M, unable to open OUTPUT FILE ')
end

for p=2:1:MAXLINK+1
            a(p)=fscanf(data,'%f',1);       % read robot description
end
for p=1:1:MAXLINK
            q(p)=fscanf(guess,'%f',1);      % 1st guess for q
end

Jac=zeros(MAXLINK,MAXLINK);                 % Matrices initialization
first(Z)=a(2);
mtar=NULL4;
dH=NULL4;
mabs(:,1:4)=rotat24(Z,PIG_2,first);         % pos. mat. of frame 0 from base frame
Last(Z,U)=a(7);                             % gripper position in frame 6
```

```
a(2)=0;                                  % D&H parameter 'a' of link 1 and link 6 are zero
a(7)=0;
dt=fscanf(motion,'%f',1);                % read time step
ii=fscanf(motion,'%d',1)+1;              % read Cardan convention
jj=fscanf(motion,'%d',1)+1;
kk=fscanf(motion,'%d',1)+1;
fprintf(out,'\n %f \n',dt);
t=0;
while       ~feof(motion)                % main loop
      [q1(1),count]=fscanf(motion,'%f',1);  % read joint motion
      if count~=1 break, end
      q1(2)  = fscanf(motion,'%f',1);
      q1(3)  = fscanf(motion,'%f',1);
      qp1(1) = fscanf(motion,'%f',1);
      qp1(2) = fscanf(motion,'%f',1);
      qp1(3) = fscanf(motion,'%f',1);
      qpp1(1)= fscanf(motion,'%f',1);
      qpp1(2)= fscanf(motion,'%f',1);
      qpp1(3)= fscanf(motion,'%f',1);
      O(X)   = fscanf(motion,'%f',1);
      O(Y)   = fscanf(motion,'%f',1);
      O(Z)   = fscanf(motion,'%f',1);
      O(U)=1;
      vel(1) = fscanf(motion,'%f',1);
      vel(2) = fscanf(motion,'%f',1);
      vel(3) = fscanf(motion,'%f',1);
      acc(1) = fscanf(motion,'%f',1);
      acc(2) = fscanf(motion,'%f',1);
     [acc(3),count]=fscanf(motion,'%f',1);
      if (count~=1)
            break;
      end
      mtar=cardatom(q1,ii,jj,kk,O);                        % builds target position matrix
      mtar= vmcopy(O,3,4,Col,mtar,4,4);
      for k=1:1:maxiter
            for i=1:1:MAXLINK
                  p=4*i-3;
                  pp=[p:p+3];
                                                                 % builds rel. pos. matrix
                  mrelp_1(:,pp)=dhtom(Rev,theta(i),d(i),b(i),a(i+1),fi(i+1),q(i));
                  mabs(:,pp+4)=mabs(:,pp)*mrelp_1(:,pp);         % builds abs. pos. matrix
                  Lrelp=makel2(Rev,Z,0.,orig);           % builds rel. L matrix in base frame
                  Lrel0=mami(Lrelp,mabs(:,pp));            % builds rel L matrix in frame (p)
                  buf(1)=Lrel0(X,U);
                  buf(2)=Lrel0(Y,U);
                  buf(3)=Lrel0(Z,U);
                  buf(4)=Lrel0(Z,Y);
                  buf(5)=Lrel0(X,Z);
                  buf(6)=Lrel0(Y,X);
                  Jac=vmcopy(buf,6,i,Col,Jac,MAXLINK,MAXLINK);
            end                                                        % fine ciclo MAXLINK
            gripper=molt(mabs(:,pp+4),Last);
            dm=(mtar-gripper);
            n=norm(dm);
            if (n>toll)                          % tests if solution has been reached
                  mabsinv=invers(gripper);
                  dS=dm*mabsinv;
                  ds(1)=dS(X,U);
                  ds(2)=dS(Y,U);
                  ds(3)=dS(Z,U);
                  ds(4)=dS(Z,Y);
                  ds(5)=dS(X,Z);
```

```
            ds(6)=dS(Y,X);
            [dq,rankm]=solve_l(Jac,ds');
            if(rankm~=MAXLINK)                      % builds the joint var. at next step
                  fprintf(1,'\n*** rank is %d: singular position!',rankm);
            end
            q=q+dq';
      else
            break;
      end                                           % end if
end                                                 % fine del ciclo maxiter
if (k<maxiter)
      Aus(X,U)=gripper(X,U);
      Aus(Y,U)=gripper(Y,U);
      Aus(Z,U)=gripper(Z,U);
      Waus=cardatow(q1,qp1,ii,jj,kk,0);             % builds target velocity matrix
      Waus=vmcopy(vel,3,4,Col,Waus,4,4);
      Wtar=mami(Waus,Aus);         % transform velocity from ausiliar frame to base frame
      Haus=cardatoh(q1,qp1,qpp1,ii,jj,kk,0);        % builds target acceleration matrix
      Haus=vmcopy(acc,3,4,Col,Haus,4,4);
      Htar=mami(Haus,Aus);           % transform accel. from ausiliar frame to base frame
      buf(1)=Wtar(X,U);                             % builds joint velocity array
      buf(2)=Wtar(Y,U);
      buf(3)=Wtar(Z,U);
      buf(4)=Wtar(Z,Y);
      buf(5)=Wtar(X,Z);
      buf(6)=Wtar(Y,X);
      [qp,rankm]=solve_l(Jac,buf');
      if(rankm~=MAXLINK)
            fprintf(1,'\n*** rank is %d: singular position!\n',rankm);
      end
      for i=1:1:MAXLINK                             % acceleration
         p=4*i-3;
         pp=[p:p+3];
         [Wrelp_1(:,pp),Hrelp_1(:,pp)]=veactowh(Rev,qp(i),0.);
         Wrel0(:,pp)=mami(Wrelp_1(:,pp),mabs(:,pp)); % W and H matrices in frame 0
         Hrel0(:,pp)=mami(Hrelp_1(:,pp),mabs(:,pp));
         Wabs(:,pp+4)=Wabs(:,pp)+Wrel0(:,pp);       % abs. vel. and acc. matrices
         Habs(:,pp+4)=coriolis(Habs(:,pp),Hrel0(:,pp),Wabs(:,pp),Wrel0(:,pp));
 end                                                % end ciclo for sui MAXLINK
 dH=Htar-Habs(:,pp+4);
 buf(1)=dH(X,U);                                    % builds joint acceleration array
 buf(2)=dH(Y,U);
 buf(3)=dH(Z,U);
 buf(4)=dH(Z,Y);
 buf(5)=dH(X,Z);
 buf(6)=dH(Y,X);
 [qpp,rankm]=solve_l(Jac,buf');
 if(rankm~=MAXLINK) fprintf('\n*** rank is %d: singular position!\n',rankm), end

 else
      fprintf('\nNewton-Raphson method does not converge\n');
      return;
 end                                                % close  k<maxiter loop
 fprintf('\n Time=%f\n',t);                         % Print output results
   printm('The joint angles q are',      q);
   printm('The joint velocity qp are',      qp');
   printm('The joint acceleration qpp are',qpp');
 fprintf('\n\nPress any key to continue\n\n');
 pause;
 for p=1:1:MAXLINK
      fprintf(out,'\n%15.5f      %15.5f      %15.5f',q(p),qp(p),qpp(p));
 end
```

```
       fprintf(out,'\n');
    t=t+dt;
end                                                        % end main loop
fclose('all');
```

Bibliography

[1] Denavit J., Hartenberg R. S., “A Kinematics Notation for Lower-Pair Mechanisms based on Matrices”, *Trans. ASME J. Appl. Mech.* 22, 215-221, June 1955.

[2] G. Legnani “Robotica Industriale”, CEA Casa Editrice Ambrosiana, 2003, isbn-88-408-1262-8.

[3] G. Legnani, F. Casolo, P. Righettini, B. Zappa “A homogeneus matrix approach to 3D kinematics and dynamics - I. Theory” Mech. Mach. Theory Vol. 31, No. 5, pp. 573-583, 1996.

[4] G. Legnani, F. Casolo, P. Righettini, B. Zappa “A homogeneus matrix approach to 3D kinematics and dynamics - II. Applications to Chains of Rigid Bodies and Serial Manipulators” Mech. Mach. Theory Vol. 31, No 5, pp. 589-605, 1996.

[5] G. Legnani, R. Riva “Un Modello per lo Studio di Robot Industriali” 7th AIMETA National Congress 1984. Trieste, Italy.

[6] G. Legnani, “Matrix Methods in Robot Kinematics” (in Italian), Doctoral Dissertation, Politecnico di Milano, Italy 1986.

[7] R. Faglia, G. Legnani “S.A.M. Robot: Simulazione ed Analisi Meccanica di Robot Industriali” Pixel n.11 1987 Ed. Il Rostro, Milano, Italy.

[8] F. Casolo, G. Legnani “A New Approach to Identify Kinematic Peculiarities in Human Motion” XII Int. Cong. of Biomechanics - UCLA 1989 - Los Angeles USA.

[9] F. Casolo, G. Legnani “A New Approach to the Dynamics of Human Motion” 2nd Int. Symposium on Computer Simulation in Biomechanics, 1989 Davis, California USA.

[10] F. Casolo, G. Legnani “A Consistent Matrix Approach for the Kinematics and Dynamics of Systems of Rigid Bodies” AIMETA nat.cong. 1988 Bari-Italy.

[11] G. Legnani, R. Riva “A New Matrix Approach to the Dynamic of Spatial Mechanisms” The 5th Int. Symposium on Linkages and Computer Aided Design Methods. Bucharest 1989.

[12] F. Casolo, R. Faglia, G. Legnani “Three dimensional analysis of systems of rigid bodies” X National Symposium DECUS, Baveno Italy 13-14 Aprile 1989.

[13] F. Casolo, R. Faglia, G. Legnani “Industrial Robots: Application of a Rational Solution for the Direct and Inverse Dynamic Problem.” 2nd Int. Workshop on Advances in Robot Kinematics. Linz - Austria 10/12 Sept. 1990.

[14] F. Casolo, G. Legnani “Dynamic Simulation of Whole Body Motion: an Application to Horse Vaulting” 10th AIMETA national congress. Pisa Italy, 2-5 October 1990.

[15] R. P. Paul “Robot Manipulators: Mathematics, Programming and Controlling” The MT Press Cambridge, England, 1981.

[16] N. Doriot, L. Cheze, “A three-dimensional kinematic and dynamic study of the lower limb during the stance phase of gait using an homogeneous matrix approach.” IEEE Trans Biomed Eng. 2004 Jan; 51(1): 21-7. Related Articles, Links.

Appendix A

Comparison between the versions of SpaceLib©.

A.1 Table of comparison

The following table lists all the SpaceLib functions comparing the three releases (C, MATLAB and Maple 9).

Table A.1: Table of comparison

C	MATLAB ©	Maple 9 ©	description
actom	actom	actoM	*Actions to Matrix.*
angle	angle	Angle	*Angle between points.*
axis	aaxis	Axis axis	*Axis of Frame.*
cardanto_G cardanto_G3 cardanto_G4	cardatog	cardantoG	*Cardan angles to angular acceleration matrix.*
cardanto_omega cardanto_omega3 cardanto_omega4	cardtome	cardanto_OMEGA	*Cardan angles to angular velocity matrix.*
cardanto_OMEGA	cardtoom	cardanto_omega	*Cardan angles to angular velocity.*
cardanto_OME-GAPTO	cardompt	cardanto_ome-gapto	*Cardan angles to angular acceleration.*
		cardanto_OME-GAPTO	*Cardan angles to angular acceleration matrix.*
cardantoH	cardatoh	cardantoH	*Cardan angles to acceleration matrix.*
cardantol	cardatol	cardantoL	*Cardan angles to L matrix.*
cardantoM	cardatom	cardantoM	*Cardan angles to position matrix.*
cardantor cardantor3 cardantor4	cardator	cardantoR	*Cardan (or Euler) angles to rotation matrix.*
cardantoW	cardatow	cardantoW	*Cardan angles to velocity matrix.*
cardantoWPROD WPRODtocardan	cardtowp	cardantoWPROD	
cardtoH		cardtoH	*Cardan angles to acceleration matrix.*
cardtoM		cardtoM	*Cardan angles to position matrix.*

C	MATLAB©	Maple 9©	description
cardtoW		cardtoW	*Cardan angles to velocity matrix.*
clear clear3 clear4 clearv3	clearmat [1]	clear [2] nullM [2]	*Clear a matrix (fill it with zeros).*
coriolis	coriolis	coriolis	*Coriolis' theorem.*
cross	cross [3]	cross	*Vector cross product.*
crossMtoM	crossmto [3] crosstom [1]	crossMtoM [2]	*Cross product for matrices (Matricial form).*
crossvtom crossmtom [3]			*Cross product for matrices (Vector form).*
deg	deg	deg	*Conversion from radians to degrees.*
dhtom	dhtom	dhtom	*Denavit & Hartenberg's parameters to matrix (Extended version).*
DHtoMstd	dhtomstd	DHtoMstd	*Denavit & Hartenberg's parameters to matrix (Standard Version).*
dist	distp dist [3]	distp	*Distance between two points.*
distpp	distpp	distpp	*Distance of point from a plane.*
dot	dot [3] dot3	vdot3	*3 elements vector dot (scalar) product.*
dot2	dot2	vdot	*any elements vector dot (scalar) product.*
dyn_eq	dyn eq	dyn_eq	*Solve Direct Dynamics system.*
dzerom			*Double Machine's zero.*
eultoH		eultoH	*Cardan angles to acceleration matrix.*
eultoM		eultoM	*Cardan angles to position matrix.*
eultoW		eultoW	*Cardan angles to velocity matrix.*
extract	extract	extract	*Extracts unit vector of screw axis and rotation angle from rotation matrix.*
fprintm3 fprintm4 fprintv	fprintm	fprintm	*Print a matrix (with a comment) on a file.*
		fmod	*fractional part of x/y.*
frame4P	frame4p	frame4P	*Frame from three points.*
frame4V	frame4v	frame4V	*Frame from a point and two vectors.*
frameP frameP3 frameP4	framep	frameP	*Frames from points.*
frameV frameV3 frameV4	framev	frameV	*Frame from vectors.*
fzerom			*Float Machine's zero.*
	grad [3]		*Conversion from radians to degrees.*

[1] *Function not really necessary in MATLAB©: provided just for compatibility with the* C *version of* SpaceLib©.
[2] *Function not really necessary in Maple 9©: provided just for compatibility with the* C *version of* SpaceLib©.
[3] obsolete version.

C	MATLAB ©	Maple 9 ©	description
gtom	gtom	gtoM	*Gravity acceleration to Matrix.*
Gtomegapto	gtomgapt	Gtomegapto	*G to omega dot.*
Htocard		Htocard	*Acceleration matrix to Cardan angles.*
Htocardan	htocarda	Htocardan	*Acceleration matrix to Cardan angles.*
Htoeul		Htoeul	*Acceleration matrix to Cardan angles.*
Htonaut		Htonaut	*Acceleration matrix to Cardan angles.*
idmat idmat3 idmat4	idmat [1]	idmat [2] eye [2]	*Identity matrix.*
intermediate	intermed	intermediate	*Middle weight point.*
inters2pl	inter2pl	inters2pl	*Intersecton of two planes.*
intersection	intersect	intersection	*Intersection between two lines.*
interslpl	interlpl	interslpl	*Intersecton of line and plane.*
invA	inva	invA	
invers	invers	invers	*Inverse of a position matrix.*
	jrand	jrand	*Creates a random matrix with elements in the range min..max.*
jtoJ	jtoj	jtoJ	*Inertia moment and mass to inertia matrix.*
line2p	line2p	line2p	*Line from two points.*
linear	linears	linear	*Linear System.*
linepvect	linpvect	linepvect	*Line from point and vector.*
makeL	makel	makeL	*Builds a L matrix.*
makeL2	makel2	makeL2	*Builds a L matrix - version 2.*
		Mcheck	*Checks for Position/Rotation Matrix.*
		Mcheck2	*Checks for Position/Rotation Matrix version 2.*
mcopy mcopy3 mcopy4	mcopy [1]		*Matrix copy.*
middle	middle	middle	*Middle point.*
minvers	minvers [1]	minvers [2]	*Matrix Inverse System.*
		Miszero	*Test Zero Matrix.*
mmcopy mcopy34 mcopy43	mmcopy [1]		*Copy a part of a matrix.*
mod	mod [3] modulus	modulus	*Module of a vector.*
molt molt3 molt4	molt [1]		*Matrix multiplication.*
moltmv3			*Multiply a matrix by a vector.*
moltp			*Multiply a matrix by a point.*
Mtocard		Mtocard	*Position matrix to Cardan angles.*
Mtocardan	mtocarda	Mtocardan	*Position matrix to Cardan angles.*

C	MATLAB©	Maple 9©	description
Mtoeul		Mtoeul	*Position matrix to Cardan angles.*
Mtonaut		Mtonaut	*Position matrix to Cardan angles.*
mtoscrew	mtoscrew	Mtoscrew	*Matrix to screw.*
mtov mtov3 mtov4	mtov	Mtov	*Matrix to vector.*
mvcopy			*Copy a row or a column from a matrix.*
nauttoH		nauttoH	*Cardan angles to acceleration matrix.*
nauttoM		nauttoM	*Cardan angles to position matrix.*
nauttoW		nauttoW	*Cardan angles to velocity matrix.*
norm norm3 norm4			*Norm of a matrix.*
norm_simm_skew n_simm3 n_simm34 n_simm4 n_skew3 n_skew34 n_skew4	normskew	norm_simm_skew	*Normalizes symmetric or skew-symmetric matrices.*
normal normal3 normal4	normal normal3	normalR	*Normalizes (orthogonalises) a 3×3 rotation matrix or the 3×3 upper-left submatrix of a position matrix.*
	normal_g	normal_g	*Normalizes (orthogonalises) any square matrix.*
pcopy			*Point copy.*
plane	plane	plane3p	*Plane from three points.*
plane2	plane2	planepv	*Plane from point and vector.*
printm printm4 printv	printm	printm	*Print a matrix (with a comment) on the screen.*
printmat iprintmat			*Prints a real elements matrix.*
prmat	prmat	prmat	*Print a position matrix for GRP man graphics post-processor.*
project	project	project	*Project a point on a plane.*
projponl	projponl	projponl	*Projection of point on line.*
psedot	psedot	psedot	*Pseudo scalar product.*
pseudo_inv	pseudinv [1]	pseudo_inv [2]	*Pseudo inverse of a matrix.*
		Origin	*Origin Point.*
rmolt rmolt3 rmolt4	rmolt [2]		*Multiply a scalar r by a matrix.*
rmoltv			*Multiply a scalar r by a vector.*
rad	rad	rad	*Conversion from degrees to radians.*
rotat	rotat	rotat	*Builds the rotation matrix R.*

C	MATLAB ©	Maple 9 ©	**description**
rotat2 rotat23	rotat2	rotat2	*Rotation around a frame axis.*
rotat24	rotat24	rotat24	*Rotation matrix around an axis with origin in a given point.*
rotat34	rotat34	rotat34	*Rotation matrix around an axis with origin in a given point.*
rtocardan rtocardan3 rtocardan4	rtocarda	Rtocardan	*Rotation matrix to Cardan (or Euler) angles.*
screwtom	screwtom	screwtoM	*Screw to Matrix.*
skew skew4	skew	skew	*Skew operator.*
solve	solve_l [1]	solver [2]	*Solve linear system.*
sub sub3 sub4 subv	sub [1]		*Subtraction for matrices or vectors.*
sum sum3 sum4 sumv	ssum [1]		*Sum of matrices or vectors.*
trac_ljlt4	tracljlt	trac_ljlt	*Trace of* L_1 J ${L_2}^t$.
traslat	traslat	traslat	*Builds the matrix m of a traslation along a vector.*
traslat2	traslat2	traslat2	*Builds the matrix m of a traslation along a frame axis.*
traslat24	traslat24	traslat24	*Builds the matrix m of a traslation along a frame axis with origin in a given point.*
transp transp3 transp4	transp [1]		*Transpose of a matrix.*
trasf_mami	mami	trasf_mami	*Transforms a matrix by the rule of* M A M^{-1} *(mami = mAminverse).*
trasf_mamt trasf_mamt4	mamt	trasf_mamt	*Transforms a matrix by the rule of* M A M^t *(mami = mAmtranspose).*
trasf_miam	miam	trasf_miam	*Transforms a matrix by the rule of* M^{-1} A M *(miam = minverseAm).*
trasf_miamit	miamit	trasf_miamit	*Transforms a matrix by the rule of* M^{-1} A M^{-t} *(miamit = minverseAminverse transposed).*
unitv	unitv	unitv	*Unit vector.*
vcopy			*Vector copy.*
vect	vect	vect	*Vector between points.*
vector	vector		*Evaluate a vector (from module and direction).*
		vect3	*Vector(3) from vector.*

C	MATLAB©	Maple 9©	**description**
velacctoWH	veactowh	velactoWH	*Velocity and Acceleration to W and H matrices.*
velacctoWH2	vactowh2	velactoWH2	*Velocity and Acceleration to W and H matrices - version 2.*
velacctoWH3	vactowh3	velactoWH3	*Velocity and Acceleration to W and H matrices - version 3.*
		viszero	*Test Zero vector.*
vmcopy			*Copy a vector into a row or a column of a matrix.*
vtom vtom3 vtom4	vtom	vtoM	*Vector to matrix.*
Wtocard		Wtocard	*Velocity matrix to Cardan angles.*
Wtocardan	wtocarda	Wtocardan	*Velocity matrix to Cardan angles.*
Wtoeul		Wtoeul	*Velocity matrix to Cardan angles.*
WtoL	wtol	WtoL	*Extracts L matrix from the corresponding W matrix.*
Wtonaut		Wtonaut	*Velocity matrix to Cardan angles.*
Wtovel	wtovel	Wtovel	*Velocity matrix to velocity parameters.*
zerom			*Machine's zero.*

Index