Robotics TOOLBOX

(Release 5)

for use with MATLAB

Peter I. Corke
pic@cat.csiro.au
http://www.cat.csiro.au/dmt/programs/autom/pic/matlab.html

April 1999

Peter I. Corke
CSIRO
Manufacturing Science and Technology
Pinjarra Hills, AUSTRALIA.
1999
http://www.cat.csiro.au/dmt/programs/autom/pic/matlab.html

Preface

1 Introduction

This Toolbox provides many functions that are useful in robotics including such things as kinematics, dynamics, and trajectory generation. The Toolbox is useful for simulation as well as analyzing results from experiments with real robots. The Toolbox has been developed and used over the last few years to the point where I now rarely write 'C' code for these kinds of tasks.

The Toolbox is based on a very general method of representing the kinematics and dynamics of serial-link manipulators. These parameters are encapsulated in Matlab objects. Robot objects can be created by the user for any serial-link manipulator and a number of examples are provided for well know robots such as the Puma 560 and the Stanford arm. Such a concise representation of robot kinematics and dynamics facilitates the sharing of robot models across the research community. It is hoped that this would allow simulation results to be compared in a much more meaningful way than is currently done in the literature. The toolbox also provides functions for manipulating datatypes such as vectors, homogeneous transformations and unit-quaternions which are necessary to represent 3-dimensional position and orientation.

The routines are generally written in a straightforward manner which allows for easy understanding at the expense of computational efficiency. If you feel strongly about computational efficiency then you can rewrite the function or create a MEX version.

2 What's new

This version has been greatly reorganized and rewritten to take advantage of Matlab release 5 features. In particular I've moved to using Matlab objects (for links, robots and quaternions) and 3-dimensional matrices. A negative side effect of this is that Octave support is now broken. I've added objects only where they seem most useful. Data structures such as homogeneous transforms, rotation matrices and trajectories still seem, to me anyway, most usefully represented by matrices.

Particular changes;

1. Added a **robot** object that comprises an array of **link** objects. The robot object has many attributes such as: name, manufacturer, base location, gravity vector, graphics handles and so on.

2. The **robot** object contains a base transform for the robot. This is now used by forward and inverse kinematic functions.

3. The **robot** object contains a gravity vector and is used by default in `rne`, `accel` and `gravload`.

4. Added a **link** object that encapsulates all information about a robot joint, its link, the motor and the gearing. A link transformation matrix, A_i, is now calculated by the link object. This means that issues such as standard or modified Denavit-Hartenberg conventions can be hidden with the link object. It has a flag to indicate what convention is in force and returns the correct A matrix.

   Additional data that could reside in a link object include; a graphical 3-D model, detailed dynamic models of the motor system, joint angle limits, torque limits and so on.

5. Added a **quaternion** object which can be constructed from a 3×3 rotation matrix, a vector and angle, or the 4 quaternion elements. Quaternions can be multiplied, divded and raised to integer powers using operator overloading. Many explicit quaternion functions have disappeared from the Toolbox.

6. Cartesian trajectories are now treated as 3 dimensional matrices rather than the previous dodge of flattening the matrices into rows of a 2D matrix. The last subscript in a trajectory matrix is the point index.

7. The vector argument to `ctraj` is now normalized path distance not time. This allows for Cartesian paths with acceleration and deceleration.

8. `manipblty` now has a flag to specify which manipulability measure is required.

9. `plotbot` has been replaced by the `plot` function which is overloaded for **robot** objects.

10. The robot `plot` and `animate` functions allow multiple robots to be displayed an animated within a figure window.
11. `fdyn` no longer uses global variables to pass data to the function being integrated. The user's torque function is now passed `q` and `qd` as distinct vectors, rather than having to unpack them from one vector.
12. Fixed a bug in `jtraj` when initial conditions are specified.
13. The `coriolis` function now ignores joint friction, which could previously corrupt the Coriolis force estimate.
14. Added `nofriction` which returns a **robot** object similar to its argument, but with all friction terms set to zero.
15. New function `ftrans` to transform force/moments between coordinate frames.
16. More functions now operate on trajectories,ie. `gravload`, `inertia`. `inertia` returns a 3D matrix of inertias.
17. New functions `tr2rot` and `rot2tr` convert between 3×3 rotation matrices and 4×4 homogeneous forms.
18. Generally increased checking of input arguments.
19. Increased robustness to row or column orientation of input vectors.
20. The MEX version of `rne` distributed previously is now broken by object support. This will be fixed in due course, but if you have the Matlab C compiler you compile it yourself.

3 Contact

The Toolbox home page is at

http://www.cat.csiro.au/dmt/programs/autom/pic/matlab.html

This page will always list the current released version number as well as bug fixes and new code in between major releases.

A Mailing List is now available. To subscribe you send mail to

```
robot-toolbox-request@dmt.csiro.au
```

with the single word "subscribe" in the body of the message. You can send other administrative messages, such as "help", "list", etc. This list is available to all users to discuss bugs and distribute M-files, and I'll use it to notify users of new releases and so on.

4 How to obtain the toolbox

The Robotics Toolbox is freely available from the MathWorks FTP server `ftp.mathworks.com` in the directory `pub/contrib/misc/robot`. It is best to download all files in that directory since the Toolbox functions are quite interdependent. The file `robot.ps` is a comprehensive manual with a tutorial introduction and details of each Toolbox function. A menu-driven demonstration can be invoked by the function `rtdemo`.

5 MATLAB version issues

The Toolbox works with MATLAB version 5 and greater and has been tested on a Sun with version 5.2.1. The Toolbox does not function under MATLAB v3.x or v4.x due to the significant changes introduced between MATLAB versions.

6 Acknowledgements

I have corresponded with a great many people via email since the first release of this toolbox. Some have identified bugs and shortcomings in the documentation, and even better, some have provided bug fixes and even new modules. I would particularly like to thank Chris Clover of Iowa State University, Anders Robertsson and Jonas Sonnerfeldt of Lund Institute of Technology, Robert Biro and Gary McMurray of Georgia Institute of Technlogy, Jean-Luc Nougaret of IRISA, Leon Zlajpah of Jozef Stefan Institute, University of Ljubljana, for their help.

7 Support, use in teaching, bug fixes, etc.

I'm always happy to correspond with people who have found genuine bugs or deficiencies in the Toolbox, or who have suggestions about ways to improve its functionality. However I do draw the line at providing help for people with their assignments and homework!

Many people are using the Toolbox for teaching and this is something that I would encourage. If you plan to duplicate the documentation for class use then every copy must include the front page.

If you want to cite the Toolbox please use

```
@ARTICLE{Corke96b,
        AUTHOR             = {P.I. Corke},
        JOURNAL            = {IEEE Robotics and Automation Magazine},
        MONTH              = mar,
        NUMBER             = {1},
        PAGES              = {24-32},
        TITLE              = {A Robotics Toolbox for {MATLAB}},
        VOLUME             = {3},
        YEAR               = {1996}
}
```

which is also given in electronic form in the README file.

1 Tutorial

8 Manipulator kinematics

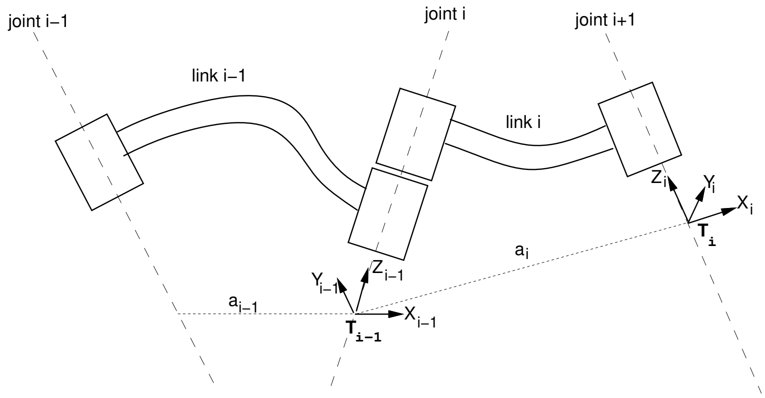
Kinematics is the study of motion without regard to the forces which cause it. Within kinematics one studies the position, velocity and acceleration, and all higher order derivatives of the position variables. The kinematics of manipulators involves the study of the geometric and time based properties of the motion, and in particular how the various links move with respect to one another and with time.

Typical robots are *serial-link manipulators* comprising a set of bodies, called *links*, in a chain, connected by *joints*[1]. Each joint has one degree of freedom, either translational or rotational. For a manipulator with n joints numbered from 1 to n, there are $n+1$ links, numbered from 0 to n. Link 0 is the base of the manipulator, generally fixed, and link n carries the end-effector. Joint i connects links i and $i-1$.

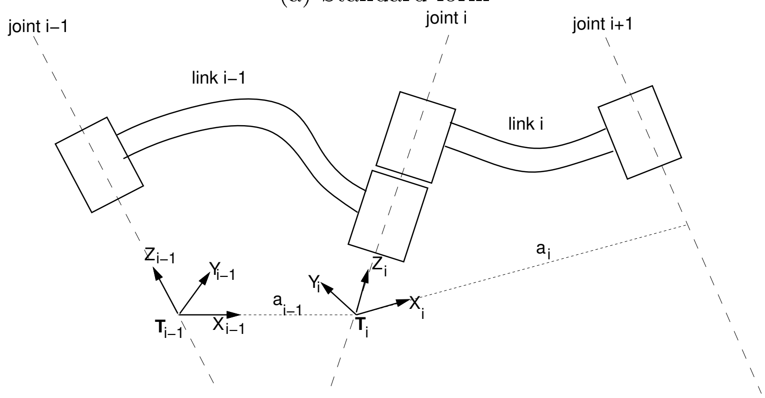
A link may be considered as a rigid body defining the relationship between two neighbouring joint axes. A link can be specified by two numbers, the *link length* and *link twist*, which define the relative location of the two axes in space. The link parameters for the first and last links are meaningless, but are arbitrarily chosen to be 0. Joints may be described by two parameters. The *link offset* is the distance from one link to the next along the axis of the joint. The *joint angle* is the rotation of one link with respect to the next about the joint axis.

To facilitate describing the location of each link we affix a coordinate frame to it — frame i is attached to link i. Denavit and Hartenberg[1] proposed a matrix method of systematically assigning coordinate systems to each link of an articulated chain. The axis of revolute joint i is aligned with z_{i-1}. The x_{i-1} axis is directed along the normal from z_{i-1} to z_i and for intersecting axes is parallel to $z_{i-1} \times z_i$. The link and joint parameters may be summarized as:

[1] Parallel link and serial/parallel hybrid structures are possible, though much less common in industrial manipulators.



(a) Standard form



(b) Modified form

Figure 1: Different forms of Denavit-Hartenberg notation.

link length	a_i	the offset distance between the z_{i-1} and z_i axes along the x_i axis;
link twist	α_i	the angle from the z_{i-1} axis to the z_i axis about the x_i axis;
link offset	d_i	the distance from the origin of frame $i-1$ to the x_i axis along the z_{i-1} axis;
joint angle	θ_i	the angle between the x_{i-1} and x_i axes about the z_{i-1} axis.

For a revolute axis θ_i is the joint variable and d_i is constant, while for a prismatic joint d_i is variable, and θ_i is constant. In many of the formulations that follow we use generalized coordinates, q_i, where

$$q_i = \begin{cases} \theta_i & \text{for a revolute joint} \\ d_i & \text{for a prismatic joint} \end{cases}$$

and generalized forces

$$Q_i = \begin{cases} \tau_i & \text{for a revolute joint} \\ f_i & \text{for a prismatic joint} \end{cases}$$

The Denavit-Hartenberg (DH) representation results in a 4x4 homogeneous transformation matrix

$$^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

representing each link's coordinate frame with respect to the previous link's coordinate system; that is

$$^0\mathbf{T}_i = {}^0\mathbf{T}_{i-1}\ {}^{i-1}\mathbf{A}_i \tag{2}$$

where $^0\mathbf{T}_i$ is the homogeneous transformation describing the pose of coordinate frame i with respect to the world coordinate system 0.

Two differing methodologies have been established for assigning coordinate frames, each of which allows some freedom in the actual coordinate frame attachment:

1. Frame i has its origin along the axis of joint $i+1$, as described by Paul[2] and Lee[3, 4].

2. Frame i has its origin along the axis of joint i, and is frequently referred to as 'modified Denavit-Hartenberg' (MDH) form[5]. This form is commonly used in literature dealing with manipulator dynamics. The link transform matrix for this form differs from (1).

Figure 1 shows the notational differences between the two forms. Note that a_i is always the length of link i, but is the displacement between the origins of frame i and frame $i+1$ in one convention, and frame $i-1$ and frame i in the other[2]. The Toolbox provides kinematic functions for both of these conventions — those for modified DH parameters are prefixed by 'm'.

8.1 Forward and inverse kinematics

For an n-axis rigid-link manipulator, the *forward kinematic solution* gives the coordinate frame, or pose, of the last link. It is obtained by repeated application of (2)

$$\begin{aligned} {}^0\mathbf{T}_n &= {}^0\mathbf{A}_1 {}^1\mathbf{A}_2 \cdots {}^{n-1}\mathbf{A}_n && (3) \\ &= \mathcal{K}(\underline{q}) && (4) \end{aligned}$$

which is the product of the coordinate frame transform matrices for each link. The pose of the end-effector has 6 degrees of freedom in Cartesian space, 3 in translation and 3 in rotation, so robot manipulators commonly have 6 joints or degrees of freedom to allow arbitrary end-effector pose. The overall manipulator transform ${}^0\mathbf{T}_n$ is frequently written as $\mathbf{T}_n$, or $\mathbf{T}_6$ for a 6-axis robot. The forward kinematic solution may be computed for any manipulator, irrespective of the number of joints or kinematic structure.

Of more use in manipulator path planning is the *inverse kinematic solution*

$$\underline{q} = \mathcal{K}^{-1}(\mathbf{T}) \tag{5}$$

which gives the joint angles required to reach the specified end-effector position. In general this solution is non-unique, and for some classes of manipulator no closed-form solution exists. If the manipulator has more than 6 joints it is said to be *redundant* and the solution for joint angles is under-determined. If no solution can be determined for a particular manipulator pose that configuration is said to be *singular*. The singularity may be due to an alignment of axes reducing the effective degrees of freedom, or the point $\mathbf{T}$ being out of reach.

The manipulator Jacobian matrix, $\mathbf{J}_\theta$, transforms velocities in joint space to velocities of the end-effector in Cartesian space. For an n-axis manipulator the end-effector Cartesian velocity is

$$\begin{aligned} {}^0\dot{\underline{x}}_n &= {}^0\mathbf{J}_\theta \dot{\underline{q}} && (6) \\ {}^{t_n}\dot{\underline{x}}_n &= {}^{t_n}\mathbf{J}_\theta \dot{\underline{q}} && (7) \end{aligned}$$

[2] Many papers when tabulating the 'modified' kinematic parameters of manipulators list a_{i-1} and α_{i-1} not a_i and α_i.

in base or end-effector coordinates respectively and where $\underline{x}$ is the Cartesian velocity represented by a 6-vector. For a 6-axis manipulator the Jacobian is square and provided it is not singular can be inverted to solve for joint rates in terms of end-effector Cartesian rates. The Jacobian will not be invertible at a kinematic singularity, and in practice will be poorly conditioned in the vicinity of the singularity, resulting in high joint rates. A control scheme based on Cartesian rate control

$$\underline{\dot{q}} = {}^0\mathbf{J}_\theta^{-1}\, {}^0\underline{\dot{x}}_n \tag{8}$$

was proposed by Whitney[6] and is known as *resolved rate motion control.* For two frames A and B related by ${}^A\mathbf{T}_B = [\underline{n}\ \underline{o}\ \underline{a}\ \underline{p}]$ the Cartesian velocity in frame A may be transformed to frame B by

$${}^B\underline{\dot{x}} = {}^B\mathbf{J}_A\, {}^A\underline{\dot{x}} \tag{9}$$

where the Jacobian is given by Paul[7] as

$${}^B\mathbf{J}_A = f({}^A\mathbf{T}_B) = \begin{bmatrix} [\underline{n}\ \underline{o}\ \underline{a}]^T & [\underline{p}\times\underline{n}\ \underline{p}\times\underline{o}\ \underline{p}\times\underline{a}]^T \\ 0 & [\underline{n}\ \underline{o}\ \underline{a}]^T \end{bmatrix} \tag{10}$$

9 Manipulator rigid-body dynamics

Manipulator dynamics is concerned with the equations of motion, the way in which the manipulator moves in response to torques applied by the actuators, or external forces. The history and mathematics of the dynamics of serial-link manipulators is well covered by Paul[2] and Hollerbach[8]. There are two problems related to manipulator dynamics that are important to solve:

- *inverse dynamics* in which the manipulator's equations of motion are solved for given motion to determine the generalized forces, discussed further in Section **??**, and
- *direct dynamics* in which the equations of motion are integrated to determine the generalized coordinate response to applied generalized forces discussed further in Section 9.2.

The equations of motion for an n-axis manipulator are given by

$$\underline{Q} = \mathbf{M}(\underline{q})\underline{\ddot{q}} + \mathbf{C}(\underline{q},\underline{\dot{q}})\underline{\dot{q}} + \mathbf{F}(\underline{\dot{q}}) + \mathbf{G}(\underline{q}) \tag{11}$$

where

- $\underline{q}$ is the vector of generalized joint coordinates describing the pose of the manipulator
- $\underline{\dot{q}}$ is the vector of joint velocities;
- $\underline{\ddot{q}}$ is the vector of joint accelerations
- $\mathbf{M}$ is the symmetric joint-space inertia matrix, or manipulator inertia tensor
- $\mathbf{C}$ describes Coriolis and centripetal effects — Centripetal torques are proportional to $\dot{q}_i^2$, while the Coriolis torques are proportional to $\dot{q}_i \dot{q}_j$
- $\mathbf{F}$ describes viscous and Coulomb friction and is not generally considered part of the rigid-body dynamics
- $\mathbf{G}$ is the gravity loading
- $\underline{Q}$ is the vector of generalized forces associated with the generalized coordinates $\underline{q}$.

The equations may be derived via a number of techniques, including Lagrangian (energy based), Newton-Euler, d'Alembert[3, 9] or Kane's[10] method. The earliest reported work was by Uicker[11] and Kahn[12] using the Lagrangian approach. Due to the enormous computational cost, $O(n^4)$, of this approach it was not possible to compute manipulator torque for real-time control. To achieve real-time performance many approaches were suggested, including table lookup[13] and approximation[14, 15]. The most common approximation was to ignore the velocity-dependent term $\mathbf{C}$, since accurate positioning and high speed motion are exclusive in typical robot applications.

Orin et al.[16] proposed an alternative approach based on the Newton-Euler (NE) equations of rigid-body motion applied to each link. Armstrong[17] then showed how recursion might be applied resulting in $O(n)$ complexity. Luh et al.[18] provided a recursive formulation of the Newton-Euler equations with linear and angular velocities referred to link coordinate frames. They suggested a time improvement from 7.9s for the Lagrangian formulation to 4.5 ms, and thus it became practical to implement 'on-line'. Hollerbach[19] showed how recursion could be applied to the Lagrangian form, and reduced the computation to within a factor of 3 of the recursive NE. Silver[20] showed the equivalence of the recursive Lagrangian and Newton-Euler forms, and that the difference in efficiency is due to the representation of angular velocity.

"Kane's equations" [10] provide another methodology for deriving the equations of motion for a specific manipulator. A number of 'Z' variables are introduced, which while not necessarily of physical significance, lead to a dynamics formulation with low computational burden. Wampler[21] discusses the computational costs of Kane's method in some detail.

Method	Multiplications	Additions	For N=6	
			Multiply	Add
Lagrangian[19]	$32\frac{1}{2}n^4 + 86\frac{5}{12}n^3$ $+171\frac{1}{4}n^2 + 53\frac{1}{3}n$ -128	$25n^4 + 66\frac{1}{3}n^3$ $+129\frac{1}{2}n^2 + 42\frac{1}{3}n$ -96	66,271	51,548
Recursive NE[19]	$150n - 48$	$131n - 48$	852	738
Kane[10]			646	394
Simplified RNE[22]			224	174

Table 1: Comparison of computational costs for inverse dynamics from various sources. The last entry is achieved by symbolic simplification using the software package ARM.

The NE and Lagrange forms can be written generally in terms of the Denavit-Hartenberg parameters — however the specific formulations, such as Kane's, can have lower computational cost for the specific manipulator. Whilst the recursive forms are computationally more efficient, the non-recursive forms compute the individual dynamic terms (**M**, **C** and **G**) directly. A comparison of computation costs is given in Table 1.

9.1 Recursive Newton-Euler formulation

The recursive Newton-Euler (RNE) formulation[18] computes the inverse manipulator dynamics, that is, the joint torques required for a given set of joint angles, velocities and accelerations. The forward recursion propagates kinematic information — such as angular velocities, angular accelerations, linear accelerations — from the base reference frame (inertial frame) to the end-effector. The backward recursion propagates the forces and moments exerted on each link from the end-effector of the manipulator to the base reference frame[3]. Figure 2 shows the variables involved in the computation for one link.

The notation of Hollerbach[19] and Walker and Orin [23] will be used in which the left superscript indicates the reference coordinate frame for the variable. The notation of Luh et al.[18] and later Lee[4, 3] is considerably less clear.

[3]It should be noted that using MDH notation with its different axis assignment conventions the Newton Euler formulation is expressed differently[5].

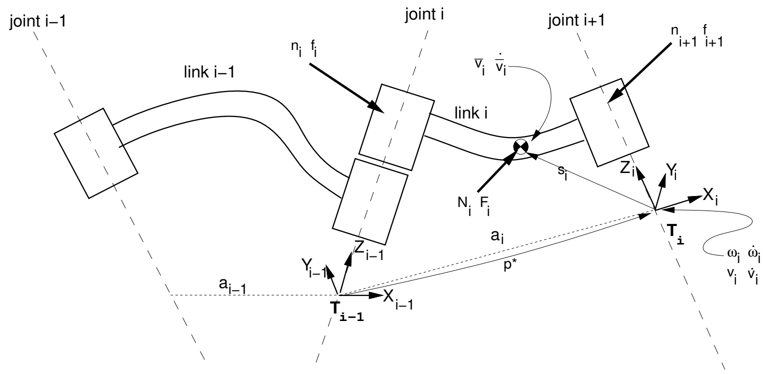


Figure 2: Notation used for inverse dynamics, based on standard Denavit-Hartenberg notation.

Outward recursion, $1 \leq i \leq n$**.**

If axis $i+1$ is rotational

$$^{i+1}\underline{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i \left({}^i\underline{\omega}_i + \underline{z}_0 \underline{\dot{q}}_{i+1} \right) \tag{12}$$

$$^{i+1}\underline{\dot{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i \left\{ {}^i\underline{\dot{\omega}}_i + \underline{z}_0 \underline{\ddot{q}}_{i+1} + {}^i\underline{\omega}_i \times \left(\underline{z}_0 \underline{\dot{q}}_{i+1} \right) \right\} \tag{13}$$

$$^{i+1}\underline{v}_{i+1} = {}^{i+1}\underline{\omega}_{i+1} \times {}^{i+1}\underline{p}^*_{i+1} + {}^{i+1}\mathbf{R}_i {}^i\underline{v}_i \tag{14}$$

$$^{i+1}\underline{\dot{v}}_{i+1} = {}^{i+1}\underline{\dot{\omega}}_{i+1} \times {}^{i+1}\underline{p}^*_{i+1} + {}^{i+1}\underline{\omega}_{i+1} \times \left\{ {}^{i+1}\underline{\omega}_{i+1} \times {}^{i+1}\underline{p}^*_{i+1} \right\} + {}^{i+1}\mathbf{R}_i {}^i\underline{\dot{v}}_i \tag{15}$$

If axis $i+1$ is translational

$$^{i+1}\underline{\omega}_{i+1} = {}^{i+1}\mathbf{R}_i {}^i\underline{\omega}_i \tag{16}$$

$$^{i+1}\underline{\dot{\omega}}_{i+1} = {}^{i+1}\mathbf{R}_i {}^i\underline{\dot{\omega}}_i \tag{17}$$

$$^{i+1}\underline{v}_{i+1} = {}^{i+1}\mathbf{R}_i \left(\underline{z}_0 \underline{\dot{q}}_{i+1} + {}^i\underline{v}_i \right) + {}^{i+1}\underline{\omega}_{i+1} \times {}^{i+1}\underline{p}^*_{i+1} \tag{18}$$

$$\begin{aligned} ^{i+1}\underline{\dot{v}}_{i+1} = {} & {}^{i+1}\mathbf{R}_i \left(\underline{z}_0 \underline{\ddot{q}}_{i+1} + {}^i\underline{\dot{v}}_i \right) + {}^{i+1}\underline{\dot{\omega}}_{i+1} \times {}^{i+1}\underline{p}^*_{i+1} + 2\, {}^{i+1}\underline{\omega}_{i+1} \times \left({}^{i+1}\mathbf{R}_i \underline{z}_0 \underline{\dot{q}}_{i+1} \right) \\ & + {}^{i+1}\underline{\omega}_{i+1} \times \left({}^{i+1}\underline{\omega}_{i+1} \times {}^{i+1}\underline{p}^*_{i+1} \right) \end{aligned} \tag{19}$$

$$^i\dot{\underline{\bar{v}}}_i = {}^i\underline{\dot{\omega}}_i \times \underline{s}_i + {}^i\underline{\omega}_i \times \left\{ {}^i\underline{\omega}_i \times \underline{s}_i \right\} + {}^i\underline{\dot{v}}_i \tag{20}$$

$$^i\underline{F}_i = m_i {}^i\dot{\underline{\bar{v}}}_i \tag{21}$$

$$^i\underline{N}_i = \mathbf{J}_i {}^i\underline{\dot{\omega}}_i + {}^i\underline{\omega}_i \times \left(\mathbf{J}_i {}^i\underline{\omega}_i \right) \tag{22}$$

Inward recursion, $n \geq i \geq 1$.

$$
\begin{aligned}
{}^i\underline{f}_i &= {}^i\mathbf{R}_{i+1}\, {}^{i+1}\underline{f}_{i+1} + {}^i\underline{F}_i && (23)\\
{}^i\underline{n}_i &= {}^i\mathbf{R}_{i+1} \left\{ {}^{i+1}\underline{n}_{i+1} + \left({}^{i+1}\mathbf{R}_i\, {}^i\underline{p}_i^* \right) \times {}^{ii+1}\underline{f}_{i+1} \right\} + \left({}^i\underline{p}_i^* + \underline{s}_i \right) \times {}^i\underline{F}_i + {}^i\underline{N}_i && (24)\\
\underline{Q}_i &= \begin{cases} \left({}^i\underline{n}_i\right)^T \left({}^i\mathbf{R}_{i+1}\underline{z}_0\right) & \text{if link } i+1 \text{ is rotational} \\ \left({}^i\underline{f}_i\right)^T \left({}^i\mathbf{R}_{i+1}\underline{z}_0\right) & \text{if link } i+1 \text{ is translational} \end{cases} && (25)
\end{aligned}
$$

where

i	is the link index, in the range 1 to n
$\mathbf{J}_i$	is the moment of inertia of link i about its COM
$\underline{s}_i$	is the position vector of the COM of link i with respect to frame i
$\underline{\omega}_i$	is the angular velocity of link i
$\underline{\dot{\omega}}_i$	is the angular acceleration of link i
$\underline{v}_i$	is the linear velocity of frame i
$\underline{\dot{v}}_i$	is the linear acceleration of frame i
$\underline{\overline{v}}_i$	is the linear velocity of the COM of link i
$\underline{\dot{\overline{v}}}_i$	is the linear acceleration of the COM of link i
$\underline{n}_i$	is the moment exerted on link i by link $i-1$
$\underline{f}_i$	is the force exerted on link i by link $i-1$
$\underline{N}_i$	is the total moment at the COM of link i
$\underline{F}_i$	is the total force at the COM of link i
$\underline{Q}_i$	is the force or torque exerted by the actuator at joint i
${}^{i-1}\mathbf{R}_i$	is the orthonormal rotation matrix defining frame i orientation with respect to frame $i-1$. It is the upper 3×3 portion of the link transform matrix given in (1).

$$
{}^{i-1}\mathbf{R}_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix} \quad (26)
$$

$$
{}^i\mathbf{R}_{i-1} = ({}^{i-1}\mathbf{R}_i)^{-1} = ({}^{i-1}\mathbf{R}_i)^T \quad (27)
$$

${}^i\underline{p}_i^*$ is the displacement from the origin of frame $i-1$ to frame i with respect to frame i.

$$
{}^i\underline{p}_i^* = \begin{bmatrix} a_i \\ d_i \sin\alpha_i \\ d_i \cos\alpha_i \end{bmatrix} \quad (28)
$$

It is the negative translational part of $({}^{i-1}\mathbf{A}_i)^{-1}$.

$\underline{z}_0$ is a unit vector in Z direction, $\underline{z}_0 = [0\ 0\ 1]$

Note that the COM linear velocity given by equation (14) or (18) does not need to be computed since no other expression depends upon it. Boundary conditions are used to introduce the effect of gravity by setting the acceleration of the base link

$$\dot{v}_0 = -\underline{g} \tag{29}$$

where $\underline{g}$ is the gravity vector in the reference coordinate frame, generally acting in the negative Z direction, downward. Base velocity is generally zero

$$v_0 = 0 \tag{30}$$

$$\omega_0 = 0 \tag{31}$$

$$\dot{\omega}_0 = 0 \tag{32}$$

At this stage the Toolbox only provides an implementation of this algorithm using the standard Denavit-Hartenberg conventions.

9.2 Direct dynamics

Equation (11) may be used to compute the so-called inverse dynamics, that is, actuator torque as a function of manipulator state and is useful for on-line control. For simulation the direct, integral or *forward dynamic* formulation is required giving joint motion in terms of input torques.

Walker and Orin[23] describe several methods for computing the forward dynamics, and all make use of an existing inverse dynamics solution. Using the RNE algorithm for inverse dynamics, the computational complexity of the forward dynamics using 'Method 1' is $O(n^3)$ for an n-axis manipulator. Their other methods are increasingly more sophisticated but reduce the computational cost, though still $O(n^3)$. Featherstone[24] has described the "articulated-body method" for $O(n)$ computation of forward dynamics, however for $n < 9$ it is more expensive than the approach of Walker and Orin. Another $O(n)$ approach for forward dynamics has been described by Lathrop[25].

9.3 Rigid-body inertial parameters

Accurate model-based dynamic control of a manipulator requires knowledge of the rigid-body inertial parameters. Each link has ten independent inertial parameters:

- link mass, m_i;
- three first moments, which may be expressed as the COM location, $\underline{s}_i$, with respect to some datum on the link or as a moment $\underline{S}_i = m_i \underline{s}_i$;
- six second moments, which represent the inertia of the link about a given axis, typically through the COM. The second moments may be expressed in matrix or tensor form as

  $$\mathbf{J} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} \tag{33}$$

  where the diagonal elements are the *moments of inertia*, and the off-diagonals are *products of inertia*. Only six of these nine elements are unique: three moments and three products of inertia.

  For any point in a rigid-body there is one set of axes known as the *principal axes of inertia* for which the off-diagonal terms, or products, are zero. These axes are given by the eigenvectors of the inertia matrix (33) and the eigenvalues are the principal moments of inertia. Frequently the products of inertia of the robot links are zero due to symmetry.

A 6-axis manipulator rigid-body dynamic model thus entails 60 inertial parameters. There may be additional parameters per joint due to friction and motor armature inertia. Clearly, establishing numeric values for this number of parameters is a difficult task. Many parameters cannot be measured without dismantling the robot and performing careful experiments, though this approach was used by Armstrong et al.[26]. Most parameters could be derived from CAD models of the robots, but this information is often considered proprietary and not made available to researchers.

References

[1] R. S. Hartenberg and J. Denavit, “A kinematic notation for lower pair mechanisms based on matrices,” *Journal of Applied Mechanics*, vol. 77, pp. 215–221, June 1955.

[2] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge, Massachusetts: MIT Press, 1981.

[3] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics. Control, Sensing, Vision and Intelligence.* McGraw-Hill, 1987.

[4] C. S. G. Lee, "Robot arm kinematics, dynamics and control," *IEEE Computer*, vol. 15, pp. 62–80, Dec. 1982.

[5] J. J. Craig, *Introduction to Robotics.* Addison Wesley, second ed., 1989.

[6] D. Whitney and D. M. Gorinevskii, "The mathematics of coordinated control of prosthetic arms and manipulators," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 20, no. 4, pp. 303–309, 1972.

[7] R. P. Paul, B. Shimano, and G. E. Mayer, "Kinematic control equations for simple manipulators," *IEEE Trans. Syst. Man Cybern.*, vol. 11, pp. 449–455, June 1981.

[8] J. M. Hollerbach, "Dynamics," in *Robot Motion - Planning and Control* (M. Brady, J. M. Hollerbach, T. L. Johnson, T. Lozano-Perez, and M. T. Mason, eds.), pp. 51–71, MIT, 1982.

[9] C. S. G. Lee, B. Lee, and R. Nigham, "Development of the generalized D'Alembert equations of motion for mechanical manipulators," in *Proc. 22nd CDC*, (San Antonio, Texas), pp. 1205–1210, 1983.

[10] T. Kane and D. Levinson, "The use of Kane's dynamical equations in robotics," *Int. J. Robot. Res.*, vol. 2, pp. 3–21, Fall 1983.

[11] J. Uicker, *On the Dynamic Analysis of Spatial Linkages Using 4 by 4 Matrices.* PhD thesis, Dept. Mechanical Engineering and Astronautical Sciences, NorthWestern University, 1965.

[12] M. Kahn, "The near-minimum time control of open-loop articulated kinematic linkages," Tech. Rep. AIM-106, Stanford University, 1969.

[13] M. H. Raibert and B. K. P. Horn, "Manipulator control using the configuration space method," *The Industrial Robot*, pp. 69–73, June 1978.

[14] A. Bejczy, "Robot arm dynamics and control," Tech. Rep. NASA-CR-136935, NASA JPL, Feb. 1974.

[15] R. Paul, "Modelling, trajectory calculation and servoing of a computer controlled arm," Tech. Rep. AIM-177, Stanford University, Artificial Intelligence Laboratory, 1972.

[16] D. Orin, R. McGhee, M. Vukobratovic, and G. Hartoch, "Kinematics and kinetic analysis of open-chain linkages utilizing Newton-Euler methods," *Mathematical Biosciences. An International Journal*, vol. 43, pp. 107–130, Feb. 1979.

[17] W. Armstrong, "Recursive solution to the equations of motion of an n-link manipulator," in *Proc. 5th World Congress on Theory of Machines and Mechanisms*, (Montreal), pp. 1343–1346, July 1979.

[18] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 102, pp. 69–76, 1980.

[19] J. Hollerbach, "A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-10, pp. 730–736, Nov. 1980.

[20] W. M. Silver, "On the equivalance of Lagrangian and Newton-Euler dynamics for manipulators," *Int. J. Robot. Res.*, vol. 1, pp. 60–70, Summer 1982.

[21] C. Wampler, *Computer Methods in Manipulator Kinematics, Dynamics, and Control: a Comparative Study.* PhD thesis, Stanford University, 1985.

[22] J. J. Murray, *Computational Robot Dynamics.* PhD thesis, Carnegie-Mellon University, 1984.

[23] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 104, pp. 205–211, 1982.

[24] R. Featherstone, *Robot Dynamics Algorithms.* Kluwer Academic Publishers, 1987.

[25] R. Lathrop, "Constrained (closed-loop) robot simulation by local constraint propogation.," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 689–694, 1986.

[26] B. Armstrong, O. Khatib, and J. Burdick, "The explicit dynamic model and inertial parameters of the Puma 560 arm," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, (Washington, USA), pp. 510–18, 1986.

2 Reference

See Also dh

For an n-axis manipulator the following matrix naming and dimensional conventions apply.

Symbol	Dimensions	Description
l	**link**	manipulator link object
q	$1 \times n$	joint coordinate vector
q	$m \times n$	m-point joint coordinate trajectory
qd	$1 \times n$	joint velocity vector
qd	$m \times n$	m-point joint velocity trajectory
qdd	$1 \times n$	joint acceleration vector
qdd	$m \times n$	m-point joint acceleration trajectory
robot	**robot**	robot object
T	4×4	homogeneous transform
T	$4 \times 4 \times m$	m-point homogeneous transform trajectory
Q	**quaternion**	unit-quaternion object
M	1×6	vector with elements of 0 or 1 corresponding to Cartesian DOF along X, Y, Z and around X, Y, Z. 1 if that Cartesian DOF belongs to the task space, else 0.
v	3×1	Cartesian vector
t	$m \times 1$	time vector
d	6×1	differential motion vector

Object names are shown in bold typeface.

A trajectory is represented by a matrix in which each row corresponds to one of m time steps. For a joint coordinate, velocity or acceleration trajectory the columns correspond to the robot axes. Things are a little more complicated for homogeneous transform trajectories since MATLAB does not (yet) support 3-dimensional matrices. The approach used in this Toolbox is that each row is a homogeneous transform that has been 'flattened' using the `(:)` operator. Each row can be restored to a 4×4 matrix by using the `reshape` function.

0.1 Units

All angles are in radians. The choice of all other units is up to the user, and this choice will flow on to the units in which homogeneous transforms, Jacobians, inertias and torques are represented.

Homogeneous Transforms	
`eul2tr`	Euler angle to homogeneous transform
`oa2tr`	orientation and approach vector to homogeneous transform
`rot2tr`	extract the 3×3 rotational submatrix from a homogeneous transform
`rotx`	homogeneous transform for rotation about X-axis
`roty`	homogeneous transform for rotation about Y-axis
`rotz`	homogeneous transform for rotation about Z-axis
`rpy2tr`	Roll/pitch/yaw angles to homogeneous transform
`tr2eul`	homogeneous transform to Euler angles
`tr2rot`	homogeneous transform to rotation submatrix
`tr2rpy`	homogeneous transform to roll/pitch/yaw angles
`transl`	set or extract the translational component of a homogeneous transform

Quaternions	
`plot`	display a quaternion as a 3D rotation
`q2tr`	quaternion to homogeneous transform
`qnorm`	normalize a quaternion
`qinterp`	interpolate quaternions

Kinematics	
`diff2tr`	differential motion vector to transform
`fkine`	compute forward kinematics
`ikine`	compute inverse kinematics
`ikine560`	compute inverse kinematics for Puma 560 like arm
`jacob0`	compute Jacobian in base coordinate frame
`jacobn`	compute Jacobian in end-effector coordinate frame
`tr2diff`	homogeneous transform to differential motion vector
`tr2jac`	homogeneous transform to Jacobian

Dynamics	
accel	compute forward dynamics
cinertia	compute Cartesian manipulator inertia matrix
coriolis	compute centripetal/coriolis torque
friction	joint friction
ftrans	transform force/moment
gravload	compute gravity loading
inertia	compute manipulator inertia matrix
itorque	compute inertia torque
nofriction	remove friction from a robot object
rne	inverse dynamics

Manipulator Models	
link	construct a robot link object
puma560	Puma 560 data
puma560akb	Puma 560 data (modified Denavit-Hartenberg)
robot	construct a robot object
stanford	Stanford arm data

Trajectory Generation	
ctraj	Cartesian trajectory
drivepar	Cartesian trajectory parameters
jtraj	joint space trajectory
trinterp	interpolate homogeneous transforms

Graphics	
plot	animate robot

Other	
manipblty	compute manipulability
rtdemo	toolbox demonstration
unit	unitize a vector

accel

Purpose

Compute manipulator forward dynamics

Synopsis

```
qdd = accel(robot, q, qd, torque)
```

Description

Returns a vector of joint accelerations that result from applying the actuator `torque` to the manipulator `robot` with joint coordinates `q` and velocities `qd`.

Algorithm

Uses the method 1 of Walker and Orin to compute the forward dynamics. This form is useful for simulation of manipulator dynamics, in conjunction with a numerical integration function.

See Also

rne, robot, fdyn, ode45

References

M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, 1982.

cinertia

Purpose

Compute the Cartesian (operational space) manipulator inertia matrix

Synopsis

```
M = cinertia(robot, q)
```

Description

`cinertia` computes the Cartesian, or operational space, inertia matrix. `robot` is a robot object that describes the manipulator dynamics and kinematics, and `q` is an n-element vector of joint coordinates.

Algorithm

The Cartesian inertia matrix is calculated from the joint-space inertia matrix by

$$\mathbf{M}(\underline{x}) = \mathbf{J}(\underline{q})^{-T}\mathbf{M}(\underline{q})\mathbf{J}(\underline{q})^{-1}$$

and relates Cartesian force/torque to Cartesian acceleration

$$\underline{F} = \mathbf{M}(\underline{x})\ddot{\underline{x}}$$

See Also

inertia, robot, rne

References

O. Khatib, "A unified approach for motion and force control of robot manipulators: the operational space formulation," *IEEE Trans. Robot. Autom.*, vol. 3, pp. 43–53, Feb. 1987.

coriolis

Purpose

Compute the manipulator Coriolis/centripetal torque components

Synopsis

```
tau_c = coriolis(robot, q, qd)
```

Description

`coriolis` returns the joint torques due to rigid-body Coriolis and centripetal effects for the specified joint state `q` and velocity `qd`. `robot` is a robot object that describes the manipulator dynamics and kinematics.

If `q` and `qd` are row vectors, `tau_c` is a row vector of joint torques. If `q` and `qd` are matrices, each row is interpreted as a joint state vector, and `tau_c` is a matrix each row being the corresponding joint torques.

Algorithm

Evaluated from the equations of motion, using `rne`, with joint acceleration and gravitational acceleration set to zero,

$$\underline{\tau} = \mathbf{C}(\underline{q}, \underline{\dot{q}})\underline{\dot{q}}$$

Joint friction is ignored in this calculation.

See Also

link, rne, itorque, gravload

References

M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, 1982.

cross

Purpose

Vector cross product

Synopsis

```
v = cross(v1, v2)
```

Description

Returns the vector cross product `v1` × `v2` The result has the same orientation (row or column vector) as the two inputs.

Note that this function is no longer provided with the Toolbox since it is now part of the core MATLAB toolbox (`specfun/cross.m`).

ctraj

Purpose

Compute a Cartesian trajectory between two points

Synopsis

```
TC = ctraj(T0, T1, m)
TC = ctraj(T0, T1, r)
```

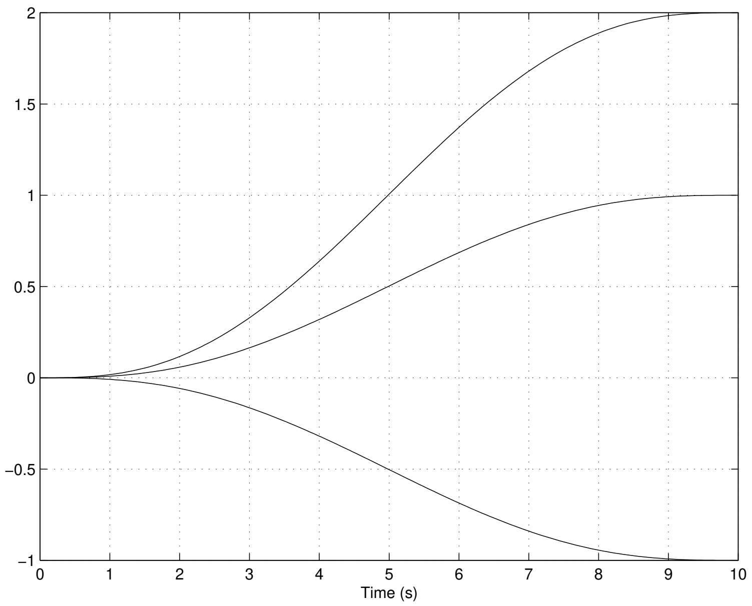
Description

`ctraj` returns a Cartesian trajectory (straight line motion) `TC` from the point represented by homogeneous transform `T0` to `T1`. The number of points along the path is `m` or the length of the given vector `r`. For the second case `r` is a vector of distances along the path (in the range 0 to 1) for each point. The first case has the points equally spaced, but different spacing may be specified to achieve acceptable acceleration profile. `TC` is a $4 \times 4 \times m$ matrix.

Examples

To create a Cartesian path with smooth acceleration we can use the `jtraj` function to create the path vector `r` with continuous derivitives.

```
>> T0 = transl([0 0 0]);  T1 = transl([-1 2 1]);
>> t= [0:0.056:10];
>> r = jtraj(0, 1, t);
>> TC = ctraj(T0, T1, r);
>> plot(t, transl(TC));
```



See Also

trinterp, drivepar, transl

References

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

diff2tr

Purpose Convert a differential motion vector to a homogeneous transform

Synopsis

```
T = diff2tr(d)
```

Description Returns a homogeneous transform representing a differential translation and rotation.

Algorithm For a differential motion $\Delta = [d_x \; d_y \; d_z \; \delta_x \; \delta_y \; \delta_z]$ the corresponding homogeneous transform is

$$\Delta = \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that the rotational submatrix is skew-symmetric.

See Also tr2diff

References

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

drivepar

Purpose

Compute Cartesian path drive parameters

Synopsis

```
dp = drivepar(T0, T1)
```

Description

`drivepar` returns a vector which represents the 'difference' between homogeneous transform `T0` and `T1` as a translation and three angles. This is most frequently used in planning a Cartesian path between homogeneous transform `T0` and `T1` and `dp` would be passed to `trinterp`.

See Also

ctraj, trinterp

References

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

eul2tr

Purpose

Convert Euler angles to a homogeneous transform

Synopsis

```
T = eul2tr([r p y])
T = eul2tr(r,p,y)
```

Description

`eul2tr` returns a homogeneous transformation for the specified Euler angles in radians. These correspond to rotations about the Z, X, and Z axes respectively.

See Also

tr2eul, rpy2tr

References

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

fdyn

Purpose

Integrate forward dynamics

Synopsis

```
[t q qd] = fdyn(robot, t0, t1)
[t q qd] = fdyn(robot, t0, t1, torqfun)
[t q qd] = fdyn(robot, t0, t1, torqfun, q0, qd0)
```

Description

`fdyn` integrates the manipulator equations of motion over the time interval `t0` to `t1` using MATLAB's `ode45` numerical integration function. It returns a time vector `t`, and matrices of manipulator joint state `q` and joint velocities `qd`. Manipulator kinematic and dynamic chacteristics are given by the robot object `robot`. These matrices have one row per time step and one column per joint.

Actuator torque may be specified by a user function

```
tau = torqfun(t, q, qd)
```

where `t` is the current time, and `q` and `qd]` are the manipulator joint coordinate and velocity state respectively. Typically this would be used to implement some axis control scheme. If `torqfun` is not specified then zero torque is applied to the manipulator.

Initial joint coordinates and velocities may be specified by the optional arguments `q0` and `qd0` respectively.

Algorithm

The joint acceleration is a function of joint coordinate and velocity given by

$$\underline{\ddot{q}} = \mathbf{M}(\underline{q})^{-1} \left\{ \underline{\tau} - \mathbf{C}(\underline{q}, \underline{\dot{q}})\underline{\dot{q}} - \mathbf{G}(\underline{q}) - \mathbf{F}(\underline{\dot{q}}) \right\}$$

Example

The following example shows how `fdyn()` can be used to simulate a robot and its controller. The manipulator is a Puma 560 with simple proportional and derivative controller. The simulation results are shown in the figure, and further gain tuning is clearly required. Note that high gains are required on joints 2 and 3 in order to counter the significant disturbance torque due to gravity.

```
>> puma560                          % load Puma parameters
>> t = [0:.056:5]';                 % time vector
>> q_dmd = jtraj(qz, qr,t);         % create a path
>> qt = [t q_dmd];
>> Pgain = [20 100 20 5 5 5];   % set gains
>> Dgain = [-5 -10 -2 0 0 0];
>> global qt Pgain Dgain
>> [tsim,q,qd] = fdyn(p560, 0, 5, 'taufunc')
```

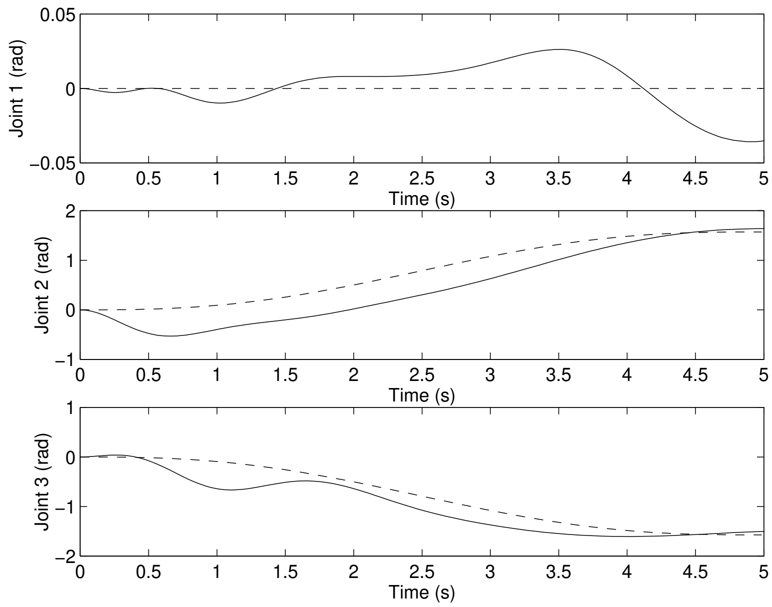
and the invoked function is

```
%
%        taufunc.m
%
% user written function to compute joint torque as a function
% of joint error.  The desired path is passed in via the global
% matrix qt.  The controller implemented is PD with the proportional
% and derivative gains given by the global variables Pgain and Dgain
% respectively.
%
function tau = taufunc(t, q, qd)
    global        Pgain Dgain qt;




    % interpolate demanded angles for this time
    if t > qt(length(qt),1),        % keep time in range
        t = qt(length(qt),1);
    end
    q_dmd = interp1(qt(:,1), qt(:,2:7), t);

    % compute error and joint torque
    e = q_dmd - q;
    tau = e * diag(Pgain) + qd * diag(Dgain)
```



Results of `fdyn()` example. Simulated path shown as solid, and reference path as dashed.

See Also

accel, rne, robot, ode45

References

M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, 1982.

fkine

Purpose

Forward robot kinematics for serial link manipulator

Synopsis

```
T = fkine(robot, q)
```

Description

`fkine` computes forward kinematics for the joint coordinate `q` giving a homogeneous transform for the location of the end-effector. `robot` is a robot object which contains a kinematic model in either standard or modified Denavit-Hartenberg notation. Note that the robot object can specify an arbitrary homogeneous transform for the base of the robot.

If `q` is a vector it is interpreted as the generalized joint coordinates, and `fkine` returns a homogeneous transformation for the final link of the manipulator. If `q` is a matrix each row is interpreted as a joint state vector, and `T` is a $4 \times 4 \times m$ matrix where `m` is the number of rows in `q`.

Cautionary

Note that the dimensional units for the last column of the `T` matrix will be the same as the dimensional units used in the `robot` object. The units can be whatever you choose (metres, inches, cubits or furlongs) but the choice will affect the numerical value of the elements in the last column of `T`. The Toolbox definitions `puma560` and `stanford` all use SI units with dimensions in metres.

See Also

dh,linktran, mfkine

References

R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, Massachusetts, 1981.
J. J. Craig, *Introduction to Robotics.* Addison Wesley, second ed., 1989.

friction

Purpose

Compute joint friction torque

Synopsis

```
tau_f = friction(link, qd)
```

Description

`friction` computes the joint friction torque based on friction parameter data, if any, in the link object `link`. Friction is a function only of joint velocity `qd`

If `qd` is a vector then `tau_f` is a vector in which each element is the friction torque for the the corresponding element in `qd`.

Algorithm

The friction model is a fairly standard one comprising viscous friction and direction dependent Coulomb friction

$$F_i(t) = \begin{cases} B_i\,\dot{q} + \tau_i^- , & \dot{\theta} < 0 \\ B_i\,\dot{q} + \tau_i^+ , & \dot{\theta} > 0 \end{cases}$$

See Also

link,nofriction

References

M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, 1982.

ftrans

Purpose

Force transformation

Synopsis

```
F2 = ftrans(F, T)
```

Description

Transform the force vector `F` in the current coordinate frame to force vector `F2` in the second coordinate frame. The second frame is related to the first by the homogeneous transform `T`. `F2` and `F` are each 6-element vectors comprising force and moment components $[F_x\ F_y\ F_z\ M_x\ M_y\ M_z]$.

See Also

diff2tr

gravload

Purpose

Compute the manipulator gravity torque components

Synopsis

```
tau_g = gravload(robot, q)
tau_g = gravload(robot, q, grav)
```

Description

`gravload` computes the joint torque due to gravity for the manipulator in pose `q`.

If `q` is a row vector, `tau_g` returns a row vector of joint torques. If `q` is a matrix each row is interpreted as as a joint state vector, and `tau_g` is a matrix in which each row is the gravity torque for the the corresponding row in `q`.

The default gravity direction comes from the robot object but may be overridden by the optional `grav` argument.

See Also

robot, link, rne, itorque, coriolis

References

M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, 1982.

ikine

Purpose

Inverse manipulator kinematics

Synopsis

```
q = ikine(robot, T)
q = ikine(robot, T, q0)
q = ikine(robot, T, q0, M)
```

Description

`ikine` returns the joint coordinates for the manipulator described by the object `robot` whose end-effector homogeneous transform is given by `T`. Note that the robot's base can be arbitrarily specified within the robot object.

If `T` is a homogeneous transform then a row vector of joint coordinates is returned. If `T` is a homogeneous transform trajectory of size $4 \times 4 \times m$ then `q` will be an $n \times m$ matrix where each row is a vector of joint coordinates corresponding to the last subscript of `T`.

The initial estimate of `q` for each time step is taken as the solution from the previous time step. The estimate for the first step in `q0` if this is given else 0. Note that the inverse kinematic solution is generally not unique, and depends on the initial value `q0` (which defaults to 0).

For the case of a manipulator with fewer than 6 DOF it is not possible for the end-effector to satisfy the end-effector pose specified by an arbitrary homogeneous transform. This typically leads to non-convergence in `ikine`. A solution is to specify a 6-element weighting vector, `M`, whose elements are 0 for those Cartesian DOF that are unconstrained and 1 otherwise. The elements correspond to translation along the X-, Y- and Z-axes and rotation about the X-, Y- and Z-axes. For example, a 5-axis manipulator may be incapable of independantly controlling rotation about the end-effector's Z-axis. In this case `M = [1 1 1 1 1 0]` would enable a solution in which the end-effector adopted the pose `T` *except* for the end-effector rotation. The number of non-zero elements should equal the number of robot DOF.

Algorithm

The solution is computed iteratively using the pseudo-inverse of the manipulator Jacobian.

$$\dot{\underline{q}} = \mathbf{J}^{+}(\underline{q})\Delta\left(\mathcal{F}(\underline{q}) - T\right)$$

where Δ returns the 'difference' between two transforms as a 6-element vector of displacements and rotations (see `tr2diff`).

Cautionary

Such a solution is completely general, though much less efficient than specific inverse kinematic solutions derived symbolically.

This function will not work for robot objects that use the modified Denavit-Hartenberg conventions.

This approach allows a solution to obtained at a singularity, but the joint coordinates within the null space are arbitrarily assigned.

Note that the dimensional units used for the last column of the `T` matrix must agree with the dimensional units used in the `dh` matrix. These units can be whatever you choose (metres, inches, cubits or furlongs) but they must be consistent. The Toolbox definitions `puma560` and `stanford` all use SI units with dimensions in metres.

See Also

fkine, tr2diff, jacob0, ikine560, robot

References

S. Chieaverini, L. Sciavicco, and B. Siciliano, "Control of robotic systems through singularities," in *Proc. Int. Workshop on Nonlinear and Adaptive Control: Issues in Robotics* (C. C. de Wit, ed.), Springer-Verlag, 1991.

ikine560

Purpose

Inverse manipulator kinematics for Puma 560 like arm

Synopsis

```
q = ikine560(robot, config)
```

Description

`ikine560` returns the joint coordinates corresponding to the end-effector homogeneous transform `T`. It is computed using a symbolic solution appropriate for Puma 560 like robots, that is, all revolute 6DOF arms, with a spherical wrist. The use of a symbolic solution means that it executes over 50 times faster than `ikine` for a Puma 560 solution.

A further advantage is that `ikine560()` allows control over the specific solution returned. `config` is a 3-element vector whose values are:

`config(1)`	-1 or `'l'`	left-handed (lefty) solution
	1 or `'u'`	†right-handed (righty) solution
`config(2)`	-1 or `'u'`	†elbow up solution
	1 or `'d'`	elbow down solution
`config(3)`	-1 or `'f'`	†wrist flipped solution
	1 or `'n'`	wrist not flipped solution

Cautionary

Note that the dimensional units used for the last column of the `T` matrix must agree with the dimensional units used in the `robot` object. These units can be whatever you choose (metres, inches, cubits or furlongs) but they must be consistent. The Toolbox definitions `puma560` and `stanford` all use SI units with dimensions in metres.

See Also

fkine, ikine, robot

References

R. P. Paul and H. Zhang, "Computationally efficient kinematics for manipulators with spherical wrists," *Int. J. Robot. Res.*, vol. 5, no. 2, pp. 32–44, 1986.

Author

Robert Biro and Gary McMurray, Georgia Institute of Technology, gt2231a@acmex.gatech.edu

inertia

Purpose

Compute the manipulator joint-space inertia matrix

Synopsis

```
M = inertia(robot, q)
```

Description

`inertia` computes the joint-space inertia matrix which relates joint torque to joint acceleration

$$\underline{\tau} = \mathbf{M}(\underline{q})\ddot{\underline{q}}$$

`robot` is a robot object that describes the manipulator dynamics and kinematics, and `q` is an n-element vector of joint state. For an n-axis manipulator `M` is an $n \times n$ symmetric matrix.

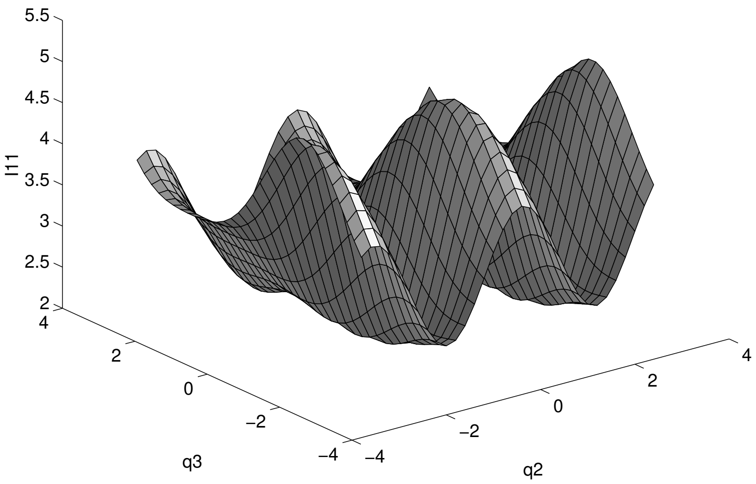
If `q` is a matrix each row is interpreted as a joint state vector, and `I` is an $n \times n \times m$ matrix where `m` is the number of rows in `q`.

Note that if the `robot` contains motor inertia parameters then motor inertia, referred to the link reference frame, will be added to the diagonal of `M`.

Example

To show how the inertia 'seen' by the waist joint varies as a function of joint angles 2 and 3 the following code could be used.

```
>> [q2,q3] = meshgrid(-pi:0.2:pi, -pi:0.2:pi);
>> q = [zeros(length(q2(:)),1) q2(:) q3(:) zeros(length(q2(:)),3)];
>> I = inertia(p560, q);
>> surfl(q2, q3, squeeze(I(1,1,:)));
```



See Also robot, rne, itorque, coriolis, gravload

References M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:205–211, 1982.

ishomog

Purpose Test if argument is a homogeneous transformation

Synopsis `ishomog(x)`

Description Returns true if `x` is a 4×4 matrix.

itorque

Purpose

Compute the manipulator inertia torque component

Synopsis

```
tau_i = itorque(robot, q, qdd)
```

Description

`itorque` returns the joint torque due to inertia at the specified pose `q` and acceleration `qdd` which is given by

$$\underline{\tau}_i = \mathbf{M}(\underline{q})\ddot{\underline{q}}$$

If `q` and `qdd` are row vectors, `itorque` is a row vector of joint torques. If `q` and `qdd` are matrices, each row is interpreted as a joint state vector, and `itorque` is a matrix in which each row is the inertia torque for the corresponding rows of `q` and `qdd`.

`robot` is a robot object that describes the kinematics and dynamics of the manipulator and drive. If `robot` contains motor inertia parameters then motor inertia, referred to the link reference frame, will be added to the diagonal of `M` and influence the inertia torque result.

See Also

robot, rne, coriolis, inertia, gravload

jacob0

Purpose

Compute manipulator Jacobian in base coordinates

Synopsis

```
jacob0(robot, q)
```

Description

`jacob0` returns a Jacobian matrix for the robot object `robot` in the pose `q` and as expressed in the base coordinate frame.

The manipulator Jacobian matrix, ${}^0\mathbf{J}_q$, maps differential velocities in joint space, $\dot{q}$, to Cartesian velocity of the end-effector expressed in the base coordinate frame.

$$ {}^0\underline{\dot{x}} = {}^0\mathbf{J}_q(\underline{q})\underline{\dot{q}} $$

For an n-axis manipulator the Jacobian is a $6 \times n$ matrix.

Cautionary

This function will not work for robot objects that use the modified Denavit-Hartenberg conventions.

See Also

jacobn, diff2tr, tr2diff, diff, robot

References

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

jacobn

Purpose

Compute manipulator Jacobian in end-effector coordinates

Synopsis

```
jacobn(robot, q)
```

Description

`jacobn` returns a Jacobian matrix for the robot object `robot` in the pose `q` and as expressed in the end-effector coordinate frame.

The manipulator Jacobian matrix, ${}^0\mathbf{J}_q$, maps differential velocities in joint space, $\dot{q}$, to Cartesian velocity of the end-effector expressed in the end-effector coordinate frame.

$$ {}^n\underline{\dot{x}} = {}^n\mathbf{J}_q(\underline{q})\underline{\dot{q}} $$

The relationship between tool-tip forces and joint torques is given by

$$ \underline{\tau} = {}^n\mathbf{J}_q(\underline{q})'\,{}^n\underline{F} $$

For an n-axis manipulator the Jacobian is a $6 \times n$ matrix.

Cautionary

This function will not work for robot objects that use the modified Denavit-Hartenberg conventions.

See Also

jacob0, diff2tr, tr2diff, diff, robot

References

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

jtraj

Purpose

Compute a joint space trajectory between two joint coordinate poses

Synopsis

```
[q qd qdd] = jtraj(q0, q1, n)
[q qd qdd] = jtraj(q0, q1, n, qd0, qd1)
[q qd qdd] = jtraj(q0, q1, t)
[q qd qdd] = jtraj(q0, q1, t, qd0, qd1)
```

Description

`jtraj` returns a joint space trajectory `q` from joint coordinates `q0` to `q1`. The number of points is `n` or the length of the given time vector `t`. A 7th order polynomial is used with default zero boundary conditions for velocity and acceleration.

Non-zero boundary velocities can be optionally specified as `qd0` and `qd1`.

The trajectory is a matrix, with one row per time step, and one column per joint. The function can optionally return a velocity and acceleration trajectories as `qd` and `qdd` respectively.

link

Purpose

Link object

Synopsis

```
L = link
L = link([alpha, a, theta, d])
L = link([alpha, a, theta, d, sigma])
L = link(dyn_row)
A = link(q)
show(L)
```

Description

The `link` function constructs a **link** object. The object contains kinematic and dynamic parameters as well as actuator and transmission parameters. The first form returns a default object, while the second and third forms initialize the kinematic model based on Denavit and Hartenberg parameters. By default the standard Denavit and Hartenberg conventions are assumed but a flag (`mdh`) can be set if modified Denavit and Hartenberg conventions are required. The dynamic model can be initialized using the fourth form of the constructor where `dyn_row` is a 1×20 matrix which is one row of the legacy `dyn` matrix.

The second last form given above is not a constructor but a link method that returns the link transformation matrix for the given joint coordinate. The argument is given to the link object using parenthesis. The single argument is taken as the link variable q and substituted for θ or D for a revolute or prismatic link respectively.

The Denavit and Hartenberg parameters describe the spatial relationship between this link and the previous one. The meaning of the fields for each model are summarized in the following table.

`alpha`	α_i	α_{i-1}	link twist angle
`A`	A_i	A_{i-1}	link length
`theta`	θ_i	θ_i	link rotation angle
`D`	D_i	D_i	link offset distance
`sigma`	σ_i	σ_i	joint type; 0 for revolute, non-zero for prismatic

Since Matlab does not support the concept of public class variables methods have been written to allow link object parameters to be referenced (r) or assigned (a) as given by the following table

Method	Operations	Returns
link.alpha	r+a	link twist angle
link.A	r+a	link length
link.theta	r+a	link rotation angle
link.D	r+a	link offset distance
link.sigma	r+a	joint type; 0 for revolute, non-zero for prismatic
link.RP	r	joint type; 'R' or 'P'
link.mdh	r+a	DH convention: 0 if standard, 1 if modified
link.I	r	3×3 symmetric inertia matrix
link.I	a	assigned from a 3×3 matrix or a 6-element vector interpretted as $[I_{xx}\ I_{yy}\ I_{zz}\ I_{xy}\ I_{yz}\ I_{xz}]$
link.m	r+a	link mass
link.r	r+a	3×1 link COG vector
link.G	r+a	gear ratio
link.Jm	r+a	motor inertia
link.B	r+a	viscous friction
link.Tc	r	Coulomb friction, 1×2 vector where $[\tau^+\ \tau^-]$
link.Tc	a	Coulomb friction; for symmetric friction this is a scalar, for asymmetric friction it is a 2-element vector for positive and negative velocity
link.dh	r+a	row of legacy DH matrix
link.dyn	r+a	row of legacy DYN matrix

The default is for standard Denavit-Hartenberg conventions, zero friction, mass and inertias.

The `display` method gives a one-line summary of the link's kinematic parameters. The `show` method displays as many link parameters as have been initialized for that link.

Examples

Robotics Toolbox Release 5

```
>> L = link([-pi/2, 0.02, 0, 0.15])
L =
  -1.570796     0.020000        0.000000        0.150000        R
>> L.RP
ans =
  R
>> L.mdh
ans =
     0
>> L.G = 100;
>> L.Tc = 5;
>> L
L =
  -1.570796     0.020000        0.000000        0.150000        R
>> show(L)
alpha  = -1.5708
A      = 0.02
theta  = 0
D      = 0.15
sigma  = 0
mdh    = 0
G      = 100
Tc     = 5 -5
>>
```

Algorithm

For the standard Denavit-Hartenberg conventions the homogeneous transform

$$^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

represents each link's coordinate frame with respect to the previous link's coordinate system.

For the modified Denavit-Hartenberg conventions it is instead

$$
{}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i \sin\alpha_{i-1} \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i \cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

See Also

robot

References

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

maniplty

Purpose

Manipulability measure

Synopsis

```
m = maniplty(robot, q)
m = maniplty(robot, q, which)
```

Description

`maniplty` computes the scalar manipulability index for the manipulator at the given pose. Manipulability varies from 0 (bad) to 1 (good). `robot` is a robot object that contains kinematic and optionally dynamic parameters for the manipulator. Two measures are supported and are selected by the optional argument `which` can be either `'yoshikawa'` (default) or `'asada'`. Yoshikawa's manipulability measure is based purely on kinematic data, and gives an indication of how 'far' the manipulator is from singularities and thus able to move and exert forces uniformly in all directions.

Asada's manipulability measure utilizes manipulator dynamic data, and indicates how close the inertia ellipsoid is to spherical.

If `q` is a vector `maniplty` returns a scalar manipulability index. If `q` is a matrix `maniplty` returns a column vector and each row is the manipulability index for the pose specified by the corresponding row of `q`.

Algorithm

Yoshikawa's measure is based on the condition number of the manipulator Jacobian

$$\eta_{yoshi} = \sqrt{|\mathbf{J}(\underline{q})\mathbf{J}(\underline{q})'|}$$

Asada's measure is computed from the Cartesian inertia matrix

$$\mathbf{M}(\underline{x}) = \mathbf{J}(\underline{q})^{-T}\mathbf{M}(\underline{q})\mathbf{J}(\underline{q})^{-1}$$

The Cartesian manipulator inertia ellipsoid is

$$\underline{x}'\mathbf{M}(\underline{x})\underline{x} = 1$$

and gives an indication of how well the manipulator can accelerate in each of the Cartesian directions. The scalar measure computed here is the ratio of the smallest/largest ellipsoid axes

$$\eta_{asada} = \frac{\min x}{\max x}$$

Ideally the ellipsoid would be spherical, giving a ratio of 1, but in practice will be less than 1.

See Also

jacob0, inertia,robot

References

T. Yoshikawa, "Analysis and control of robot manipulators with redundancy," in *Proc. 1st Int. Symp. Robotics Research*, (Bretton Woods, NH), pp. 735–747, 1983.

oa2tr

Purpose

Convert OA vectors to homogeneous transform

Synopsis

```
oa2tr(o, a)
```

Description

`oa2tr` returns a rotational homogeneous transformation specified in terms of the Cartesian orientation and approach vectors `o` and `a` respectively.

Algorithm

$$\mathbf{T} = \begin{bmatrix} \underline{\hat{o}} \times \underline{\hat{a}} & \underline{\hat{o}} & \underline{\hat{a}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\underline{\hat{o}}$ and $\underline{\hat{a}}$ are unit vectors corresponding to `o` and `a` respectively.

See Also

rpy2tr, eul2tr

quaternion/plot

Purpose

Plot quaternion rotation

Synopsis

`plot(Q)`

Description

`plot` is overloaded for **quaternion** objects and displays a 3D plot which shows how the standard axes are transformed under that rotation.

Examples

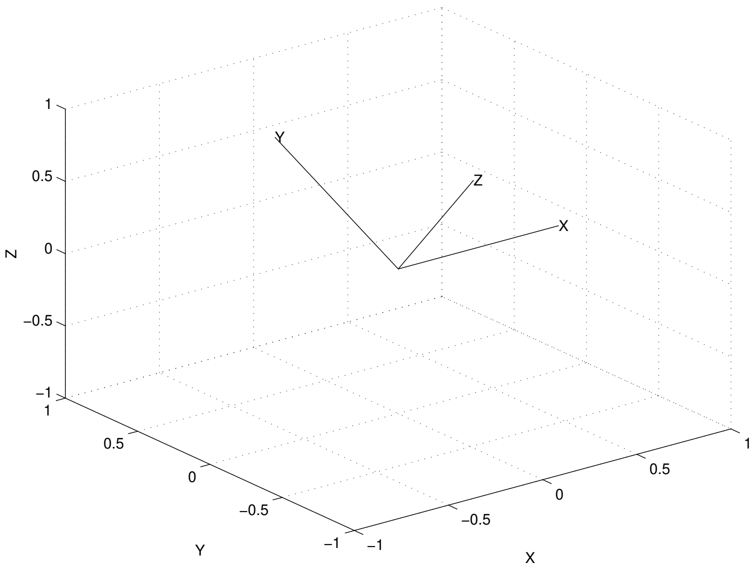
A rotation of 0.3rad about the X axis. Clearly the X axis is invariant under this rotation.

```
>> q=quaternion(rotx(0.3))

q =

  0.85303<0.52185, 0, 0>

>> plot(q)
```



See Also quaternion

robot/plot

Purpose

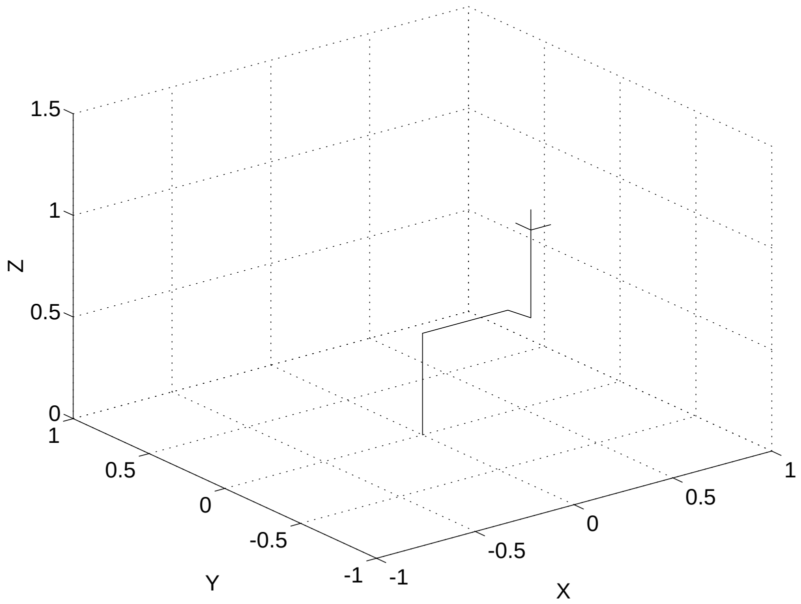
Graphical robot animation

Synopsis

```
plot(robot, q)
plot(robot, q, opt)
animate(robot, q)
```

Description

`plot` ss overloaded for **robot** objects and displays a graphical representation of the robot given the kinematics information in `robot`. The robot is represented by a simple stick figure where line segments join the origins of the link coordinate frames. If `q` is a matrix representing a joint-space trajectory then an animation of the robot motion is shown.



The wrist coordinate frame is shown by 3 short orthogonal line segments which are colored:

red (X or normal), green (Y or orientation) and blue (Z or approach).

Options are specified by `opt` which is a string which can contain one or more of the following keys:

l	leave trail, that is, don't erase the robot from the previous time step
w	don't draw the wrist axis coordinate frame
r	repeat mode, run the animation 50 times

The last function allows a graphical robot created by `plot` to be animated. The graphical handles are stored within the **robot** object by `plot`.

Examples

To animate two Pumas moving in the same figure window.

```
>> clf
>> p560b = p560;  % duplicate the robot
>> plot(p560, qr);   % display it at ready position
>> hold on
>> plot(p560b, qr);  % display it at ready position
>> t = [0:0.056:10];
>> jt = jtraj(qr, qstretch, t);  % trajectory to stretch pose
>> for q = jt', % for all points on the path
>>    animate(p560, q);
>>    animate(p560b, q);
>> end
```

See Also

fkine, robot

puma560

Purpose

Create a Puma 560 robot object

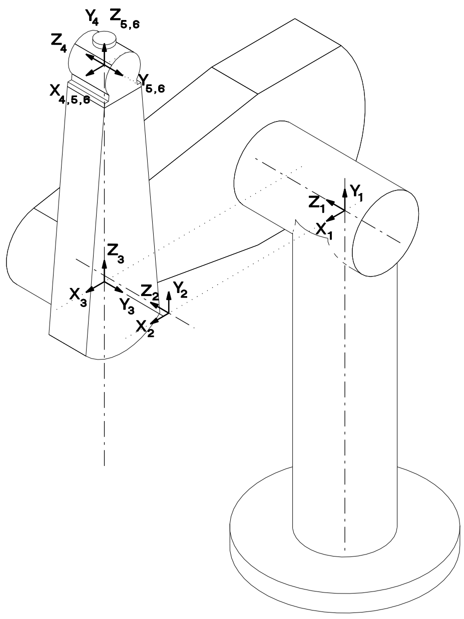
Synopsis

`puma560`

Description

Creates the **robot** object `p560` which describes the kinematic and dynamic characteristics of a Unimation Puma 560 manipulator. The kinematic conventions used are as per Paul and Zhang, and all quantities are in standard SI units.

Also defines the joint coordinate vectors `qz`, `qr` and `qstretch` corresponding to the zero-angle, ready and fully extended poses respectively.



Details of coordinate frames used for the Puma 560 shown here in its zero angle pose.

See Also

robot, puma560akb, stanford

References

R. P. Paul and H. Zhang, "Computationally efficient kinematics for manipulators with spherical wrists," *Int. J. Robot. Res.*, vol. 5, no. 2, pp. 32–44, 1986.

P. Corke and B. Armstrong-Hélouvry, "A search for consensus among model parameters reported for the PUMA 560 robot," in *Proc. IEEE Int. Conf. Robotics and Automation*, (San Diego), pp. 1608–1613, May 1994.

P. Corke and B. Armstrong-Hélouvry, "A meta-study of PUMA 560 dynamics: A critical appraisal of literature data," *Robotica*, vol. 13, no. 3, pp. 253–258, 1995.

puma560akb

Purpose

Create a Puma 560 robot object

Synopsis

`puma560akb`

Description

Creates the **robot** object which describes the kinematic and dynamic characteristics of a Unimation Puma 560 manipulator. Craig's modified Denavit-Hartenberg notation is used, with the particular kinematic conventions from Armstrong, Khatib and Burdick. All quantities are in standard SI units.

Also defines the joint coordinate vectors `qz`, `qr` and `qstretch` corresponding to the zero-angle, ready and fully extended poses respectively.

See Also

robot stanford

References

B. Armstrong, O. Khatib, and J. Burdick, "The explicit dynamic model and inertial parameters of the Puma 560 arm," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, (Washington, USA), pp. 510–18, 1986.

quaternion

Purpose

Quaternion object

Synopsis

```
q = quaternion
q = quaternion(qq)
q = quaternion(theta, v)
q = quaternion([s vx vy vz])
q = quaternion(R)
```

Description

`quaternion` is the constructor for a **quaternion** object. The first form creates a default quaternion, and the second form returns a new object with the same value as its argument. The third form initializes the quaternion to a rotation of `theta` about the vector `v`.

The fourth form sets the four quaternion elements directly where `s` is the scalar component and `[vx vy vz]` the vector. The final method sets the quaternion to a rotation equivalent to the given 3×3 rotation matrix, or the rotation submatrix of a 4×4 homogeneous transform.

Some operators are overloaded for the quaternion class

`q1 * q2`	returns quaternion product or compounding
`q * v`	returns a quaternion vector product, that is the vector v is rotated by the quaternion. v is a 3×3 vector
`q1 / q2`	returns $q_1 * q_2^{-1}$
`q1 ∧ j`	returns q^j where j is an integer exponent. For $j > 0$ the result is obtained by repeated multiplication. For $j < 0$ the final result is inverted.
`double(q)`	returns the quaterion coeffients as a 4-element row vector
`plot(q)`	displays a 3D plot showing the standard coordinate frame after rotation by `q`.

Examples

```
>> q1 = quaternion( rotx(0.2) )
q1 =

  0.86025 <0.50987, 0, 0>

>> q2 = quaternion( roty(0.3) )
q2 =
   0.85303 <0, 0.52185, 0>


>> q1 * q2
ans =
  0.73382 <0.43494, 0.44893, 0.26608>

>> q1*q1
ans =

  0.48007 <0.87723, 0, 0>

>> q1^2
ans =
   0.48007 <0.87723, 0, 0>

>> q1/q1
ans =
  1 <0, 0, 0>

>> q1/q2
ans =
  0.73382 <0.43494, -0.44893, -0.26608>

>> q1*q2^-1
ans =
  0.73382 <0.43494, -0.44893, -0.26608>
```

See Also

quaternion/plot

References

K. Shoemake, "Animating rotation with quaternion curves.," in *Proceedings of ACM SIGGRAPH*, (San Francisco), pp. 245–254, The Singer Company, Link Flight Simulator Division, 1985.

qinterp

Purpose Interpolate unit-quaternions

Synopsis `QI = qinterp(Q1, Q2, r)`

Description Return a unit-quaternion that interpolates between Q1 and Q2 as `r` varies between 0 and 1 inclusively. This is a spherical linear interpolation (slerp) that can be interpreted as interpolation along a great circle arc on a sphere.

References K. Shoemake, “Animating rotation with quaternion curves.,” in *Proceedings of ACM SIGGRAPH*, (San Francisco), pp. 245–254, The Singer Company, Link Flight Simulator Division, 1985.

qnorm

Purpose Normalize a quaternion

Synopsis `QN = qnorm(Q)`

Description Return a unit-quaternion object than represents the same rotation as the quaternion `Q`.

robot

Purpose

Robot object

Synopsis

```
r = robot
r = robot(rr)
r = robot(link ...)
r = robot(DH ...)
r = robot(DYN ...)
```

Description

`robot` is the constructor for a robot object. The first form creates a default robot, and the second form returns a new robot object with the same value as its argument. The third form creates a robot from a cell array of link objects which define the robot's kinematics and optionally dynamics. The fourth and fifth forms create a robot object from legacy DH and DYN format matrices.

The last three forms all accept optional trailing string arguments which are taken in order as being robot name, manufacturer and comment.

Since Matlab does not support the concept of public class variables methods have been written to allow robot object parameters to be referenced (r) or assigned (a) as given by the following table

method	Operation	Returns
robot.n	r	number of joints
robot.link	r+a	cell array of link objects
robot.name	r+a	robot name string
robot.manuf	r+a	robot manufacturer string
robot.comment	r+a	general comment string
robot.gravity	r	3-element vector defining gravity direction
robot.mdh	r	DH convention: 0 if standard, 1 if modified. Determined from the link objects.
robot.base	r+a	homogeneous transform defining base of robot
robot.dh	r	legacy DH matrix
robot.dyn	r	legacy DYN matrix

Default values for robot parameters are:

robot.name	'noname'
robot.manuf	''
robot.comment	''
robot.gravity	[0 0 9.81] $\mathrm{m/s^2}$
robot.base	`eye(4,4)`

The multiplication operator, `*`, is overloaded and the product of two robot objects is a robot which is the series connection of the multiplicands.

The `plot()` function is also overloaded and is used to provide a robot animation.

Examples

```
>> L{1} = link([ pi/2  0  0  0])
L =
    [1x1 link]

>> L{2} = link([ 0  0 0.5 0])
L =
    [1x1 link]    [1x1 link]
```

```
>> r = robot(L)

r =
 (2 axis, RR)
                grav = [0.00 0.00 9.81]
                standard D&H parameters

  alpha            A               theta            D              R/P
1.570796        0.000000        0.000000        0.000000        R
0.000000        0.000000        0.500000        0.000000        R

>>
```

See Also link,plot

rne

Purpose

Compute inverse dynamics via recursive Newton-Euler formulation

Synopsis

```
tau = rne(robot, q, qd, qdd)
tau = rne(robot, [q qd qdd])

tau = rne(robot, q, qd, qdd, grav)
tau = rne(robot, [q qd qdd], grav)

tau = rne(robot, q, qd, qdd, grav, fext)
tau = rne(robot, [q qd qdd], grav, fext)
```

Description

`rne` computes the equations of motion in an efficient manner, giving joint torque as a function of joint position, velocity and acceleration.

If `q`, `qd` and `qdd` are row vectors then `tau` is a row vector of joint torques. If `q`, `qd` and `qdd` are matrices then `tau` is a matrix in which each row is the joint torque for the corresponding rows of `q`, `qd` and `qdd`.

Gravity direction is defined by the robot object but may be overridden by providing a gravity acceleration vector `grav = [gx gy gz]`.

An external force/moment acting on the end of the manipulator may also be specified by a 6-element vector `fext = [Fx Fy Fz Mx My Mz]` in the end-effector coordinate frame.

The torque computed may contain contributions due to armature inertia and joint friction if these are specified in the parameter matrix `dyn`.

The MEX file version of this function is around 300 times faster than the M file.

Algorithm

Coumputes the joint torque

$$\underline{\tau} = \mathbf{M}(\underline{q})\ddot{\underline{q}} + \mathbf{C}(\underline{q}, \dot{\underline{q}})\dot{\underline{q}} + \mathbf{F}(\dot{\underline{q}}) + \mathbf{G}(\underline{q})$$

where $\mathbf{M}$ is the manipulator inertia matrix, $\mathbf{C}$ is the Coriolis and centripetal torque, $\mathbf{F}$ the viscous and Coulomb friction, and $\mathbf{G}$ the gravity load.

Cautionary This function will not work for robot objects that use the modified Denavit-Hartenberg conventions.

See Also robot, fdyn, accel, gravload, inertia, friction

Limitations A MEX file is currently only available for Sparc architecture.

References

J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. On-line computational scheme for mechanical manipulators. *ASME Journal of Dynamic Systems, Measurement and Control*, 102:69–76, 1980.

rotvec

Purpose

Rotation about a vector

Synopsis

```
T = rotvec(v, theta)
```

Description

`rotvec` returns a homogeneous transformation representing a rotation of `theta` radians about the vector `v`.

See Also

rotx, roty, rotz

rotx,roty,rotz

Purpose Rotation about X, Y or Z axis

Synopsis

```
T = rotx(theta)
T = roty(theta)
T = rotz(theta)
```

Description Return a homogeneous transformation representing a rotation of `theta` radians about the X, Y or Z axes.

See Also rotvec

rpy2tr

Purpose Roll/pitch/yaw angles to homogeneous transform

Synopsis

```
T = rpy2tr([r p y])
T = rpy2tr(r,p,y)
```

Description `rpy2tr` returns a homogeneous transformation for the specified roll/pitch/yaw angles in radians. These correspond to rotations about the Z, X, Y axes respectively.

See Also tr2rpy, eul2tr

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

rtdemo

Purpose Robot Toolbox demonstration

Synopsis `rtdemo`

Description This script provides demonstrations for most functions within the Robotics Toolbox.

stanford

Purpose

Create a Stanford manipulator robot object

Synopsis

```
stanford
```

Description

Creates the **robot** object `stan` which describes the kinematic and dynamic characteristics of a Stanford manipulator. Specifies armature inertia and gear ratios. All quantities are in standard SI units.

See Also

robot, puma560

References

R. Paul, "Modeling, trajectory calculation and servoing of a computer controlled arm," Tech. Rep. AIM-177, Stanford University, Artificial Intelligence Laboratory, 1972.

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

tr2diff

Purpose

Convert a homogeneous transform to a differential motion vector

Synopsis

```
d = tr2diff(T)
d = tr2diff(T1, T2)
```

Description

The first form of `tr2diff` returns a 6-element differential motion vector representing the incremental translation and rotation described by the homogeneous transform `T`. It is assumed that `T` is of the form

$$\begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The translational elements of `d` are assigned directly. The rotational elements are computed from the mean of the two values that appear in the skew-symmetric matrix.

The second form of `tr2diff` returns a 6-element differential motion vector representing the displacement from `T1` to `T2`, that is, `T2 - T1`.

$$d = \begin{bmatrix} \underline{p}_2 - \underline{p}_1 \\ 1/2\,(\underline{n}_1 \times \underline{n}_2 + \underline{o}_1 \times \underline{o}_2 + \underline{a}_1 \times \underline{a}_2) \end{bmatrix}$$

See Also

diff2tr, diff

References

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

tr2eul

Purpose

Convert a homogeneous transform to Euler angles

Synopsis

```
[a b c] = tr2eul(T)
```

Description

`tr2eul` returns a vector of Euler angles, in radians, corresponding to the rotational part of the homogeneous transform `T`.

See Also

eul2tr, tr2rpy

References

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

tr2jac

Purpose

Compute a Jacobian to map differential motion between frames

Synopsis

```
jac = tr2jac(T)
```

Description

`tr2jac` returns a 6×6 Jacobian matrix to map differential motions or velocities between frames related by the homogeneous transform `T`.

If `T` represents a homogeneous transformation from frame A to frame B, ${}^{A}\mathbf{T}_{B}$, then

$$^{B}\underline{\dot{x}} = {}^{B}\mathbf{J}_{A}\,{}^{A}\underline{\dot{x}}$$

where ${}^{B}\mathbf{J}_{A}$ is given by `tr2jac(T)`.

See Also

tr2diff, diff2tr, diff

References

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

tr2rpy

Purpose

Convert a homogeneous transform to roll/pitch/yaw angles

Synopsis

```
[a b c] = tr2rpy(T)
```

Description

`tr2rpy` returns a vector of roll/pitch/yaw angles, in radians, corresponding to the rotational part of the homogeneous transform `T`.

See Also

rpy2tr, tr2eul

References

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

transl

Purpose

Translational transformation

Synopsis

```
T = transl(x, y, z)
T = transl(v)
v = transl(T)
xyz = transl(TC)
```

Description

The first two forms return a homogeneous transformation representing a translation expressed as three scalar `x`, `y` and `z`, or a Cartesian vector `v`.

The third form returns the translational part of a homogeneous transform as a 3-element column vector.

The fourth form returns a matrix whose columns are the X, Y and Z columns of the $4 \times 4 \times m$ Cartesian trajectory matrix `TC`.

See Also

ctraj, rotx, roty, rotz, rotvec

trinterp

Purpose

Interpolate homogeneous transforms

Synopsis

```
T = trinterp(T0, T1, r)
T = trinterp(T0, dp, r)
```

Description

`trinterp` interpolates between the two homogeneous transforms `T0` and `T1` as `r` varies between 0 and 1 inclusively. This is generally used for computing straight line or 'Cartesian' motion.

The second form uses a drive parameter matrix computed by `drivepar` which represents the 'difference' between `T0` and `T1` and may be more efficient when computing many interpolated points between the same two endpoints.

See Also

ctraj, drivepar

References

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

trnorm

Purpose

Normalize a homogeneous transformation

Synopsis

```
TN = trnorm(T)
```

Description

Returns a normalized copy of the homogeneous transformation `T`. Finite word length arithmetic can lead to homogeneous transformations in which the rotational submatrix is not orthogonal, that is, $det(\mathbf{R}) \neq -1$.

Algorithm

Normalization is performed by orthogonalizing the rotation submatrix $\underline{n} = \underline{o} \times \underline{a}$.

See Also

diff2tr, diff

References

J. Funda, "Quaternions and homogeneous transforms in robotics," Master's thesis, University of Pennsylvania, Apr. 1988.

unit

Purpose

Unitize a vector

Synopsis

```
vn = unit(v)
```

Description

`unit` returns a unit vector aligned with `v`.

Algorithm

$$\underline{v}_n = \frac{\underline{v}}{||\underline{v}||}$$

dh (legacy)

Purpose

Matrix representation of manipulator kinematics

Description

A `dh` matrix describes the kinematics of a manipulator in a general way using the standard Denavit-Hartenberg conventions. Each row represents one link of the manipulator and the columns are assigned according to the following table.

Column	Symbol	Description
1	α_i	link twist angle
2	A_i	link length
3	θ_i	link rotation angle
4	D_i	link offset distance
5	σ_i	joint type; 0 for revolute, non-zero for prismatic

If the last column is not given, toolbox functions assume that the manipulator is all-revolute. For an n-axis manipulator `dh` is an $n \times 4$ or $n \times 5$ matrix.

The first 5 columns of a `dyn` matrix contain the kinematic parameters and maybe used anywhere that a `dh` kinematic matrix is required — the dynamic data is ignored.

Lengths A_i and D_i may be expressed in any unit, and this choice will flow on to the units in which homogeneous transforms and Jacobians are represented. All angles are in radians.

See Also

dyn,puma560,stanford,mdh

References

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

dyn (legacy)

Purpose

Matrix representation of manipulator kinematics and dynamics

Description

A `dyn` matrix describes the kinematics and dynamics of a manipulator in a general way using the standard Denavit-Hartenberg conventions. Each row represents one link of the manipulator and the columns are assigned according to the following table.

Column	Symbol	Description
1	α	link twist angle
2	A	link length
3	θ	link rotation angle
4	D	link offset distance
5	σ	joint type; 0 for revolute, non-zero for prismatic
6	mass	mass of the link
7	rx	link COG with respect to the link coordinate frame
8	ry	
9	rz	
10	Ixx	elements of link inertia tensor about the link COG
11	Iyy	
12	Izz	
13	Ixy	
14	Iyz	
15	Ixz	
16	Jm	armature inertia
17	G	reduction gear ratio; joint speed/link speed
18	B	viscous friction, motor referred
19	Tc+	coulomb friction (positive rotation), motor referred
20	Tc-	coulomb friction (negative rotation), motor referred

For an n-axis manipulator, `dyn` is an $n \times 20$ matrix. The first 5 columns of a `dyn` matrix contain the kinematic parameters and maybe used anywhere that a `dh` kinematic matrix is required — the dynamic data is ignored.

All angles are in radians. The choice of all other units is up to the user, and this choice will flow on to the units in which homogeneous transforms, Jacobians, inertias and torques are represented.

See Also dh