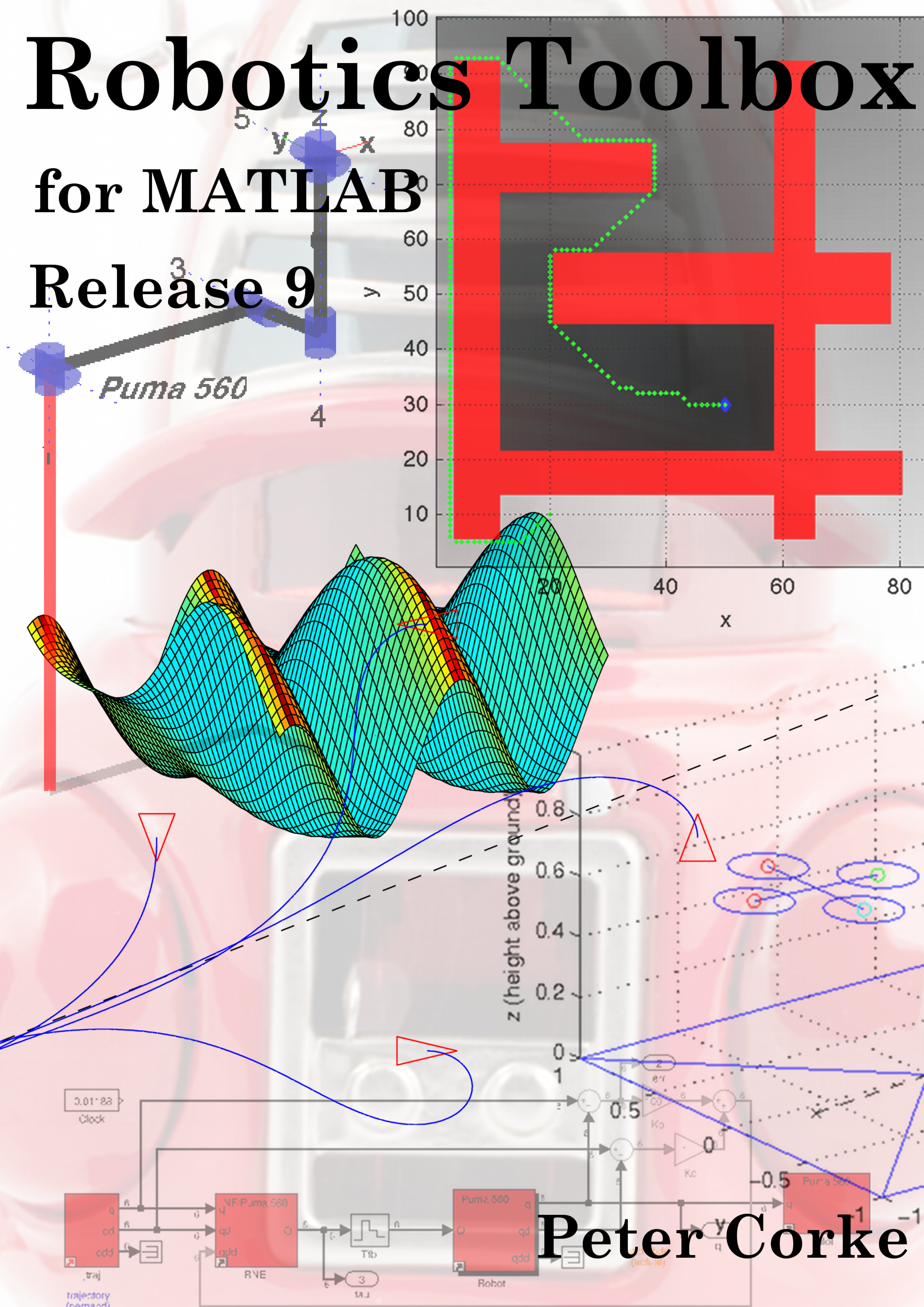
Robotics Toolbox

for MATLAB

Release 9



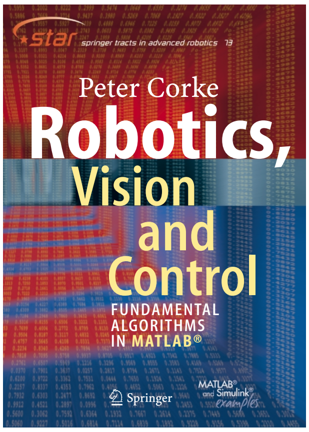
Peter Corke

Release	9.9
Release date	April 2014

Licence	LGPL
Toolbox home page	http://www.petercorke.com/robot
Discussion group	http://groups.google.com.au/group/robotics-tool-box

peter.i.corke@gmail.com
http://www.petercorke.com

Preface



This, the ninth release of the Toolbox, represents over fifteen years of development and a substantial level of maturity. This version captures a large number of changes and extensions generated over the last two years which support my new book "*Robotics, Vision & Control*" shown to the left.

The Toolbox has always provided many functions that are useful for the study and simulation of classical arm-type robotics, for example such things as kinematics, dynamics, and trajectory generation. 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 amongst others. The Toolbox also provides functions for manipulating and converting between datatypes such as vectors, homogeneous transformations and unit-quaternions which are necessary to represent 3-dimensional position and orientation.

This ninth release of the Toolbox has been significantly extended to support mobile robots. For ground robots the Toolbox includes standard path planning algorithms (bug, distance transform, D*, PRM), kinodynamic planning (RRT), localization (EKF, particle filter), map building (EKF) and simultaneous localization and mapping (EKF), and a Simulink model a of non-holonomic vehicle. The Toolbox also including a detailed Simulink model for a quadcopter flying robot.

The routines are generally written in a straightforward manner which allows for easy understanding, perhaps at the expense of computational efficiency. If you feel strongly about computational efficiency then you can always rewrite the function to be more efficient, compile the M-file using the MATLAB® compiler, or create a MEX version.

The manual is now auto-generated from the comments in the MATLAB® code itself which reduces the effort in maintaining code and a separate manual as I used to — the downside is that there are no worked examples and figures in the manual. However the book "*Robotics, Vision & Control*" provides a detailed discussion (600 pages, nearly 400 figures and 1000 code examples) of how to use the Toolbox functions to solve many types of problems in robotics.

Contents

Functions by category

Homogeneous transformations 3D

HT: homogeneous transformation, a 4x4 matrix, in SE(3)

RM: rotation matrix, orthonormal 3x3 matrix, in SO(3)

Functions of the form tr2XX will also accept an HT or RM as the argument

Homogeneous transformations 2D

HT: homogeneous transformation, a 3x3 matrix, in SE(2)

RM: rotation matrix, orthonormal 2x2 matrix, in SO(2)

Homogeneous points and lines

Differential motion

Demonstrations

Interfacing

Arbotix class in the folder robot/interfaces

VREP classes are in the folder robot/interfaces/VREP

Examples

located in the examples folder

Chapter 1

Introduction

1.1 What's changed

1.1.1 Documentation

- The manual (robot.pdf) no longer a separately written document. This was just too hard to keep updated with changes to code. All documentation is now in the m-file, making maintenance easier and consistency more likely. The negative consequence is that the manual is a little "drier" than it used to be.
- The Functions link from the Toolbox help browser lists all functions with hyperlinks to the individual help entries.
- Online HTML-format help is available from `http://www.petercorke.com/RTB/r9/html`

1.1.2 New features and changes

Features of this particular dot point release include:

- A major rewrite of `CodeGenerator`
- A major rewrite of `ikine6s` to handle a number of specific cases: robot with no shoulder offset, robot with shoulder offset (can have lefty/right configuration), Stanford arm (prismatic third joint), Puma 560 arm. The previous code made lots of assumptions applicable to the Puma, which caused errors for other 6-axis robots with spherical wrists.
- Symbolic inverse kinematics (developmental) can be found for robots with 2, 3 or 6 DOF. See `SerialLink.ikine_sym`.
- Aesthetic updates to `plot()` and `teach()` methods of the `SerialLink` object

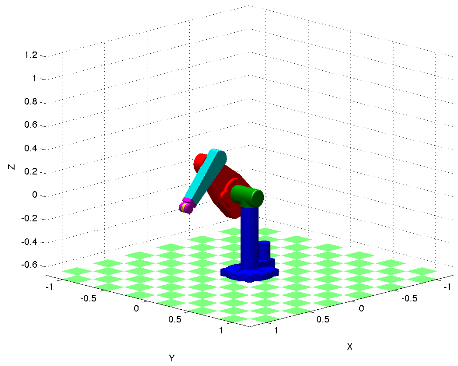


Figure 1.1: New rendered robot model using the `plot3d()` method.

- A new method `plot3d()` which uses STL format solid models to render realistic looking robots as shown in Figure 1.1. This requires STL models, such as those shipped with the package ARTE by Arturo Gil.
- There are subclasses of `Link` called `Revolute`, `Prismatic`, `RevoluteMDH` and `PrismaticMDH` that can be used to make your code clearer and more concise.
- The behaviour of Fast RNE is a bit different. The MATLAB version `@SerialLink/rne.m` is always executed, and it makes the decision whether or not to invoke the MEX file. The MEX file is executed if:
  1. the robot is not symbolic, and
  2. the `SerialLink` property `fast` is true (ie. the 'nofast' option is not given), and
  3. the MEX file exists.
- A significant number of new robot models.
- A major renovation of `DHFactor` to bring it up to spec with the lastest version of Java.
- A 'flip' option added to `tr2eul`.
- A new method `A` for `SerialLink` object that computes a sequence of `Link` A matrices.
- A new method `trchain` for `SerialLink` object that emits the kinematic model as a sequence of elementary transforms.
- A new method `trchain` that expresses kinematics as a chain of elementary transforms.
- A bunch of functions with suffix `2` that deal with SE(2) and SO(2) transforms.
- An improved version of the demo `rtbdemo`, with more functions and an improved interface. It uses the common function `runscript` to step through the

individual demo scripts. It works even better with `cprintf` from MATLAB Central.

- A set of classes (experimental) to interface with the V-REP robotics simulation engine by Coppelia Robotics. See `robot/interfaces/VREP`.
- Since 9.8 the Toolbox now contains the Robotic Symbolic Toolbox by Joërn Malzahn. There are additional functions, as well as symbolic support throughout the SerialLink class.
- Many bug fixes

Compared to release 8.x and earlier:

- The command `startup_rvc` should be executed before using the Toolbox. This sets up the MATLAB search paths correctly.
- The Robot class is now named SerialLink to be more specific.
- Almost all functions that operate on a SerialLink object are now methods rather than functions, for example plot() or fkine(). In practice this makes little difference to the user but operations can now be expressed as robot.plot(q) or plot(robot, q). Toolbox documentation now prefers the former convention which is more aligned with object-oriented practice.
- The parametrers to the Link object constructor are now in the order: theta, d, a, alpha. Why this order? It's the order in which the link transform is created: RZ(theta) TZ(d) TX(a) RX(alpha).
- All robot models now begin with the prefix mdl_, so puma560 is now mdl_puma560.
- The function drivebot is now the SerialLink method teach.
- The function ikine560 is now the SerialLink method ikine6s to indicate that it works for any 6-axis robot with a spherical wrist.
- The link class is now named Link to adhere to the convention that all classes begin with a capital letter.
- The `robot` class is now called `SerialLink`. It is created from a vector of `Link` objects, not a cell array.
- The quaternion class is now named Quaternion to adhere to the convention that all classes begin with a capital letter.
- A number of utility functions have been moved into the a directory common since they are not robot specific.
- skew no longer accepts a skew symmetric matrix as an argument and returns a 3-vector, this functionality is provided by the new function vex.
- tr2diff and diff2tr are now called tr2delta and delta2tr
- ctraj with a scalar argument now spaces the points according to a trapezoidal velocity profile (see lspb). To obtain even spacing provide a uniformly spaced vector as the third argument, eg. linspace(0, 1, N).
- The RPY functions tr2rpy and rpy2tr assume that the roll, pitch, yaw rotations are about the X, Y, Z axes which is consistent with common conventions for

vehicles (planes, ships, ground vehicles). For some applications (eg. cameras) it useful to consider the rotations about the Z, Y, X axes, and this behaviour can be obtained by using the option 'zyx' with these functions (note this is the pre release 8 behaviour).

- Many functions now accept MATLAB style arguments given as trailing strings, or string-value pairs. These are parsed by the internal function `tb_optparse`.

1.1.3 New functions

Release 9 introduces considerable new functionality, in particular for mobile robot control, navigation and localization:

- Mobile robotics:

  Vehicle Model of a mobile robot that has the "bicycle" kinematic model (car-like). For given inputs it updates the robot state and returns noise corrupted odometry measurements. This can be used in conjunction with a "driver" class such as RandomPath which drives the vehicle between random waypoints within a specified rectangular region.

  Sensor

  RangeBearingSensor Model of a laser scanner RangeBearingSensor, subclass of Sensor, that works in conjunction with a Map object to return range and bearing to invariant point features in the environment.

  EKF Extended Kalman filter EKF can be used to perform localization by dead reckoning or map featuers, map buildings and simultaneous localization and mapping.

  DXForm Path planning classes: distance transform DXform, D* lattice planner Dstar, probabilistic roadmap planner PRM, and rapidly exploring random tree RRT.

  Monte Carlo estimator ParticleFilter.

- Arm robotics:

  jsingu

  qplot

  DHFactor a simple means to generate the Denavit-Hartenberg kinematic model of a robot from a sequence of elementary transforms.

- Trajectory related:

  lspb

  tpoly

  mtraj

  mstraj

- General transformation:

wtrans

se2

se3

homtrans

vex performs the inverse function to skew, it converts a skew-symmetric matrix to a 3-vector.

- Data structures:

  Pgraph represents a non-directed embedded graph, supports plotting and minimum cost path finding.

  Polygon a generic 2D polygon class that supports plotting, intersectio/union/difference of polygons, line/polygon intersection, point/polygon containment.

- Graphical functions:

  trprint compact display of a transform in various formats.

  trplot display a coordinate frame in SE(3)

  trplot2 as above but for SE(2)

  tranimate animate the motion of a coordinate frame

  plot_box plot a box given TL/BR corners or center+WH, with options for edge color, fill color and transparency.

  plot_circle plot one or more circles, with options for edge color, fill color and transparency.

  plot_sphere plot a sphere, with options for edge color, fill color and transparency.

  plot_ellipse plot an ellipse, with options for edge color, fill color and transparency.

  plot_ellipsoid plot an ellipsoid, with options for edge color, fill color and transparency.

  plot_poly plot a polygon, with options for edge color, fill color and transparency.

- Utility:

  about display a one line summary of a matrix or class, a compact version of whos

  tb_optparse general argument handler and options parser, used internally in many functions.

- Lots of Simulink models are provided in the subdirectory `simulink`. These models all have the prefix `sl_`.

1.1.4 Improvements

- Many functions now accept MATLAB style arguments given as trailing strings, or string-value pairs. These are parsed by the internal function `tb_optparse`.
- Many functions now handle sequences of rotation matrices or homogeneous transformations.
- Improved error messages in many functions
- Removed trailing commas from if and for statements

1.2 How to obtain the Toolbox

The Robotics Toolbox is freely available from the Toolbox home page at

http://www.petercorke.com

The web page requests some information from you regarding such as your country, type of organization and application. This is just a means for me to gauge interest and to remind myself that this is a worthwhile activity.

The file is available in zip format (.zip). Download it and unzip it. Files all unpack to the correct parts of a hiearchy of directories (folders) headed by `rvctools`.

If you already have the Machine Vision Toolbox installed then download the zip file to the directory above the existing `rvctools` directory, and then unzip it. The files from this zip archive will properly interleave with the Machine Vision Toolbox files.

Ensure that the folder `rvctools` is on your MATLAB® search path. You can do this by issuing the `addpath` command at the MATLAB® prompt. Then issue the command `startup_rvc` and it will add a number of paths to your MATLAB® search path. You need to setup the path every time you start MATLAB® but you can automate this by setting up environment variables, editing your `startup.m` script by pressing the “Update Toolbox Path Cache” button under MATLAB® General preferences.

A menu-driven demonstration can be invoked by the function `rtdemo`.

1.2.1 Documentation

The file `robot.pdf` is a manual that describes all functions in the Toolbox. It is auto-generated from the comments in the MATLAB® code and is fully hyperlinked: to external web sites, the table of content to functions, and the “See also” functions to each other.

The same documentation is available online in alphabetical order at http://www.petercorke.com/RTB/r9/html/index_alpha.html or by category at http://www.petercorke.com/RTB/r9/html/index.html. Documentation is also available via the MATLAB® help browser, “Robotics Toolbox” appears under the Contents.

1.3 MATLAB version issues

The Toolbox has been tested under R2014a.

1.4 Use in teaching

This is definitely encouraged! You are free to put the PDF manual (`robot.pdf` or the web-based documentation `html/*.html` on a server for class use. If you plan to distribute paper copies of the PDF manual then every copy must include the first two pages (cover and licence).

1.5 Use in research

If the Toolbox helps you in your endeavours then I'd appreciate you citing the Toolbox when you publish. The details are

```
@book{Corke11a,
    Author = {Peter I. Corke},
    Date-Added = {2011-01-12 08:19:32 +1000},
    Date-Modified = {2012-07-29 20:07:27 +1000},
    Note = {ISBN 978-3-642-20143-1},
    Publisher = {Springer},
    Title = {Robotics, Vision \& Control: Fundamental Algorithms in {MATLAB}},
    Year = {2011}}
```

or

> P.I. Corke, Robotics, Vision & Control: Fundamental Algorithms in MATLAB. Springer, 2011. ISBN 978-3-642-20143-1.

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

1.6 Support

There is no support! This software is made freely available in the hope that you find it useful in solving whatever problems you have to hand. I am happy to correspond with people who have found genuine bugs or deficiencies but my response time can be long and I can't guarantee that I respond to your email. I am very happy to accept contributions for inclusion in future versions of the toolbox, and you will be suitably acknowledged.

I can guarantee that I will not respond to any requests for help with assignments or homework, no matter how urgent or important they might be to you. That's what your teachers, tutors, lecturers and professors are paid to do.

You might instead like to communicate with other users via the Google Group called "Robotics and Machine Vision Toolbox" http://groups.google.com.au/group/robotics-tool-box

which is a forum for discussion. You need to signup in order to post, and the signup process is moderated by me so allow a few days for this to happen. I need you to write a few words about why you want to join the list so I can distinguish you from a spammer or a web-bot.

1.7 Related software

1.7.1 Octave

Octave is an open-source mathematical environment that is very similar to MATLAB® , but it has some important differences particularly with respect to graphics and classes. Many Toolbox functions work just fine under Octave. Three important classes (Quaternion, Link and SerialLink) will not work so modified versions of these classes is provided in the subdirectory called `Octave`. Copy all the directories from `Octave` to the main Robotics Toolbox directory.

The Octave port is a second priority for support and upgrades and is offered in the hope that you find it useful.

1.7.2 Python version

A python implementation of the Toolbox at http://code.google.com/p/robotics-toolbox-python. All core functionality of the release 8 Toolbox is present including kinematics, dynamics, Jacobians, quaternions etc. It is based on the python numpy class. The main current limitation is the lack of good 3D graphics support but people are working on this. Nevertheless this version of the toolbox is very usable and of course you don't need a MATLAB® licence to use it. Watch this space.

1.7.3 Machine Vision toolbox

Machine Vision toolbox (MVTB) for MATLAB®. This was described in an article

```
@article{Corke05d,
        Author = {P.I. Corke},
        Journal = {IEEE Robotics and Automation Magazine},
        Month = nov,
        Number = {4},
        Pages = {16-25},
        Title = {Machine Vision Toolbox},
        Volume = {12},
        Year = {2005}}
```

and provides a very wide range of useful computer vision functions beyond the Mathwork's Image Processing Toolbox. You can obtain this from http://www.petercorke.com/vision.

1.8 Acknowledgements

Joërn Malzahn has donated a considerable amount of code, his Robot Symbolic Toolbox for MATLAB. 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, thankyou. See the file `CONTRIB` for details. I'd like to especially mention Wynand Smart for some arm robot models, Paul Pounds for the quadcopter model, and Paul Newman (Oxford) for inspiring the mobile robot code.

Chapter 2

Functions and classes

about

Compact display of variable type

about(**x**) displays a compact line that describes the class and dimensions of **x**.

about **x** as above but this is the command rather than functional form

See also

whos

---

angdiff

Difference of two angles

d = **angdiff**(**th1**, **th2**) returns the difference between angles **th1** and **th2** on the circle. The result is in the interval [-pi pi). If **th1** is a column vector, and **th2** a scalar then return a column vector where **th2** is modulo subtracted from the corresponding elements of **th1**.

d = **angdiff**(**th**) returns the equivalent angle to **th** in the interval [-pi pi).

Return the equivalent angle in the interval [-pi pi).

---

angvec2r

Convert angle and vector orientation to a rotation matrix

R = **angvec2r**(**theta**, **v**) is an orthonormal rotation matrix, **R**, equivalent to a rotation of **theta** about the vector **v**.

See also

eul2r, rpy2r

---

angvec2tr

Convert angle and vector orientation to a homogeneous transform

T = **angvec2tr**(**theta**, **v**) is a homogeneous transform matrix equivalent to a rotation of **theta** about the vector **v**.

Note

- The translational part is zero.

See also

eul2tr, rpy2tr, angvec2r

Arbotix

Interface to Arbotix robot controller

Methods

Arbotix	Constructor, establishes serial communications
delete	Destructor, closes serial connection
getpos	Get joint angles
setpos	Set joint angles and optionally speed
relax	Control relax (zero torque) state
setled	Control LEDs on servos
gettemp	Temperature of motors

writedata1	Write byte data to servo control table
writedata2	Write word data to servo control table
readdata	Read servo control table

command	Execute command on servo
flush	Flushes serial data buffer

Example

```
arb=Arbotix('port', '/dev/tty.usbserial-A800JDPN', 'nservos', 5);
```

Notes

- interface is via serial to an Arbotix controller running the pypose sketch

---

Arbotix.Arbotix

Create Arbotix interface object

dm = **Arbotix**(**options**) is an object that represents a connection to a chain of **Arbotix** servos connected via an **Arbotix** controller and serial link to the host computer.

Options

'port', P	Name of the serial port device, eg. /dev/tty.USB0
'baud', B	Set baud rate (default 38400)
'debug', D	Debug level, show communications packets (default 0)
'nservos', N	Number of servos in the chain

Arbotix.a2e

Convert angle to encoder

E = ARB.A2E(**a**) is a vector of encoder values **E** corresponding to the vector of joint angles **a**.

Arbotix.char

Convert Arbotix status to string

C = ARB.char() is a string that succinctly describes the status of the **Arbotix** controller link.

Arbotix.command

Execute command on servo

R = ARB.COMMAND(**id**, **instruc**) executes the instruction **instruc** on servo **id**.

R = ARB.COMMAND(**id**, **instruc**, **data**) as above but the vector **data** forms the payload of the command message, and all numeric values in **data** must be in the range 0 to 255.

The optional output argument **R** is a structure holding the return status.

Notes

- **id** is in the range 0 to N-1, where N is the number of servos in the system.
- Values for **instruc** are defined as class properties INS_*.
- If 'debug' was enabled in the constructor then the hex values are echoed to the screen as well as being sent to the Arbotix.
- If an output argument is requested the serial channel is flushed first.

See also

Arbotix.receive, Arbotix.flush

Arbotix.delete

Close the serial connection

delete(**dm**) closes the serial connection and removes the **dm** object from the workspace.

Arbotix.discover

how many servos in the chain

Arbotix.display

Display parameters

ARB.**display**() displays the servo parameters in compact single line format.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Arbotix object and the command has no trailing semicolon.

See also

Arbotix.char

Arbotix.e2a

Convert encoder to angle

a = ARB.**E2A**(**E**) is a vector of joint angles **a** corresponding to the vector of encoder values **E**.

Arbotix.flush

Flush the receive buffer

ARB.**FLUSH**() flushes the serial input buffer, data is discarded.

s = ARB.**FLUSH**() as above but returns a vector of all bytes flushed from the channel.

Notes

- Every command sent to the Arbotix elicits a reply.
- The method receive() should be called after every command.
- Some Arbotix commands also return diagnostic text information.

See also

Arbotix.receive, Arbotix.parse

Arbotix.getpos

Get position

p = ARB.**GETPOS**(**id**) is the position (0-1023) of servo **id**.

p = ARB.**GETPOS**([]) is a vector ($1 \times N$) of positions of servos 1 to N.

Notes

- N is defined at construction time by the 'nservos' option.

Arbotix.gettemp

Get temperature

T = ARB.**GETTEMP**(**id**) is the tempeature (deg C) of servo **id**.

T = ARB.**GETTEMP**() is a vector ($1 \times N$) of the temperature of servos 1 to N.

Notes

- N is defined at construction time by the 'nservos' option.

Arbotix.parse

Parse Arbotix return strings

ARB.**PARSE**(**s**) parses the string returned from the **Arbotix** controller and prints diagnostic text. The string **s** contains a mixture of Dynamixel style return packets and diagnostic text.

Notes

- Every command sent to the Arbotix elicits a reply.
- The method receive() should be called after every command.
- Some Arbotix commands also return diagnostic text information.

See also

Arbotix.flush, Arbotix.command

Arbotix.readdata

Read byte data from servo control table

R = ARB.**READDATA**(**id**, **addr**) reads the successive elements of the servo control table for servo **id**, starting at address **addr**. The complete return status in the structure **R**, and the byte data is a vector in the field 'params'.

Notes

- **id** is in the range 0 to N-1, where N is the number of servos in the system.
- If 'debug' was enabled in the constructor then the hex values are echoed to the screen as well as being sent to the Arbotix.

See also

Arbotix.receive, Arbotix.command

Arbotix.receive

Decode Arbotix return packet

R = ARB.**RECEIVE**() reads and parses the return packet from the **Arbotix** and returns a structure with the following fields:

id	The id of the servo that sent the message
error	Error code, 0 means OK
params	The returned parameters, can be a vector of byte values

Notes

- Every command sent to the Arbotix elicits a reply.
- The method receive() should be called after every command.
- Some Arbotix commands also return diagnostic text information.
- If 'debug' was enabled in the constructor then the hex values are echoed

Arbotix.relax

Control relax state

ARB.**RELAX**(**id**) causes the servo **id** to enter zero-torque (relax state) ARB.**RELAX**(**id**, FALSE) causes the servo **id** to enter position-control mode ARB.**RELAX**([]) causes servos 1 to N to relax ARB.**RELAX**() as above ARB.**RELAX**([], FALSE) causes servos 1 to N to enter position-control mode.

Notes

- N is defined at construction time by the 'nservos' option.

Arbotix.setled

Set LEDs on servos

ARB.**led**(**id**, **status**) sets the LED on servo **id** to on or off according to the **status** (logical).

ARB.**led**([], **status**) as above but the LEDs on servos 1 to N are set.

Notes

- N is defined at construction time by the 'nservos' option.

Arbotix.setpath

Load a path into Arbotix controller

ARB.**setpath**(**jt**) stores the path **jt** ($P \times N$) in the **Arbotix** controller where P is the number of points on the path and N is the number of robot joints.

Arbotix.setpos

Set position

ARB.**SETPOS**(**id**, **pos**) sets the position (0-1023) of servo **id**. ARB.**SETPOS**(**id**, **pos**, SPEED) as above but also sets the speed.

ARB.**SETPOS**(**pos**) sets the position of servos 1-N to corresponding elements of the vector **pos** ($1 \times N$). ARB.**SETPOS**(**pos**, SPEED) as above but also sets the velocity SPEED ($1 \times N$).

Notes

- **id** is in the range 1 to N
- N is defined at construction time by the 'nservos' option.
- SPEED varies from 0 to 1023, 0 means largest possible speed.

Arbotix.setpos_sync

speed are vectors

Arbotix.syncwrite

column per actuator

Arbotix.writedata1

Write byte data to servo control table

ARB.**WRITEDATA1**(**id**, **addr**, **data**) writes the successive elements of **data** to the servo control table for servo **id**, starting at address **addr**. The values of **data** must be in the range 0 to 255.

Notes

- **id** is in the range 0 to N-1, where N is the number of servos in the system.
- If 'debug' was enabled in the constructor then the hex values are echoed to the screen as well as being sent to the Arbotix.

See also

Arbotix.writedata2, Arbotix.command

Arbotix.writedata2

Write word data to servo control table

ARB.**WRITEDATA2**(**id**, **addr**, **data**) writes the successive elements of **data** to the servo control table for servo **id**, starting at address **addr**. The values of **data** must be in the range 0 to 65535.

Notes

- **id** is in the range 0 to N-1, where N is the number of servos in the system.
- If 'debug' was enabled in the constructor then the hex values are echoed to the screen as well as being sent to the Arbotix.

See also

Arbotix.writedata1, Arbotix.command

bresenham

Generate a line

p = **bresenham**(**x1**, **y1**, **x2**, **y2**) is a list of integer coordinates for points lying on the line segement (**x1**,**y1**) to (**x2**,**y2**). Endpoints must be integer.

p = **bresenham**(**p1**, **p2**) as above but **p1**=[**x1**,**y1**] and **p2**=[**x2**,**y2**].

See also

icanvas

Bug2

Bug navigation class

A concrete subclass of the Navigation class that implements the bug2 navigation algorithm. This is a simple automaton that performs local planning, that is, it can only sense the immediate presence of an obstacle.

Methods

path	Compute a path from start to goal
visualize	Display the obstacle map (deprecated)
plot	Display the obstacle map
display	Display state/parameters in human readable form
char	Convert to string

Example

load map1	% load the map
bug = Bug2(map);	% create navigation object

```
bug.goal = [50, 35];  % set the goal
```

bug.path([20, 10]);   % animate path to (20,10)

Reference

- Dynamic path planning for a mobile automaton with limited information on the environment,, V. Lumelsky and A. Stepanov, IEEE Transactions on Automatic Control, vol. 31, pp. 1058-1063, Nov. 1986.
- Robotics, Vision & Control, Sec 5.1.2, Peter Corke, Springer, 2011.

See also

Navigation, DXform, Dstar, PRM

Bug2.Bug2

bug2 navigation object constructor

b = **Bug2**(**map**) is a bug2 navigation object, and **map** is an occupancy grid, a representation of a planar world as a matrix whose elements are 0 (free space) or 1 (occupied).

Options

'goal', G	Specify the goal point (1×2)
'inflate', K	Inflate all obstacles by K cells.

See also

Navigation.Navigation

---

circle

Compute points on a circle

circle(**C**, **R**, **opt**) plot a **circle** centred at **C** with radius **R**.

x = **circle**(**C**, **R**, **opt**) return an $N \times 2$ matrix whose rows define the coordinates [x,y] of points around the circumferance of a **circle** centred at **C** and of radius **R**.

C is normally 2×1 but if 3×1 then the **circle** is embedded in 3D, and **x** is $N \times 3$, but the **circle** is always in the xy-plane with a z-coordinate of **C**(3).

Options

‘n’, N	Specify the number of points (default 50)

CodeGenerator

Class for code generation

Objects of the CodeGenerator class automatcally generate robot specific code, as either M-functions, C-functions, C-MEX functions, real-time capable Simulink blocks.

The various methods return symbolic expressions for robot kinematic and dynamic functions, and optionally support side effects such as:

- M-functions with symbolic robot specific model code
- real-time capable robot specific Simulink blocks
- mat-files with symbolic robot specific model expressions
- C-functions and -headers with symbolic robot specific model code
- robot specific MEX functions based on the generated C-code (C-compiler must be installed).

Example

```
% load robot model
mdl_twolink

cg = CodeGenerator(twolink);
cg.geneverything();

% a new class has been automatically generated in the robot directory.
addpath robot

tl = @robot();
% this class is a subclass of SerialLink, and thus polymorphic with
% SerialLink but its methods have been overloaded with robot-specific code,
% for example
T = tl.fkine([0.2 0.3]);
% uses concise symbolic expressions rather than the generalized A-matrix
% approach

% The Simulink block library containing robot-specific blocks can be
% opened by
open robot/robotslib.slx
% and the blocks dragged into your own models.
```

Methods

gencoriolis	generate Coriolis/centripetal code
genfdyn	generate forward dynamics code
genfkine	generate forward kinematics code
genfriction	generate joint friction code
gengravload	generate gravity load code
geninertia	generate inertia matrix code
geninvdyn	generate inverse dynamics code
genjacobian	generate Jacobian code
geneverything	generate code for all of the above

Properties (read/write)

basepath	basic working directory of the code generator
robjpath	subdirectory for specialized MATLAB functions
sympath	subdirectory for symbolic expressions
slib	filename of the Simulink library
slibpath	subdirectory for the Simulink library
verbose	print code generation progress on console (logical)
saveresult	save symbolic expressions to .mat-files (logical)
logfile	print modeling progress to specified text file (string)
genmfun	generate executable M-functions (logical)
genslblock	generate Embedded MATLAB Function blocks (logical)
genccode	generate C-functions and -headers (logical)
genmex	generate MEX-functions as replacement for M-functions (logical)
compilemex	automatically compile MEX-functions after generation (logical)

Object properties (read only)

rob	SerialLink object to generate code for (1×1).

Notes

- Requires the MATLAB Symbolic Toolbox
- For robots with > 3 joints the symbolic expressions are massively complex, they are slow and you may run out of memory.
- As much as possible the symbolic calculations are down row-wise to reduce the computation/memory burden.
- Requires a C-compiler if robot specific MEX-functions shall be generated as m-functions replacement (see MATLAB documentation of MEX files).

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

SerialLink, Link

CodeGenerator.CodeGenerator

Construct a code generator object

cGen = CodeGenerator(**rob**) is a code generator object for the SerialLink object **rob**.

cGen = CodeGenerator(**rob**, **options**) as above but with options described below.

Options

The following option sets can be passed as an optional parameter:

'default'	set the options: verbose, saveResult, genMFun, genSLBlock
'debug'	set the options: verbose, saveResult, genMFun, genSLBlock and create a logfile named 'robModel.log' in the working directory
'silent'	set the options: saveResult, genMFun, genSLBlock
'disk'	set the options: verbose, saveResult
'workspace'	set the option: verbose; just outputs symbolic expressions to workspace
'mfun'	set the options: verbose, saveResult, genMFun
'slblock'	set the options: verbose, saveResult, genSLBlock
'ccode'	set the options: verbose, saveResult, genCcode
'mex'	set the options: verbose, saveResult, genMEX

If 'optionSet' is ommitted, then 'default' is used. The options control the code generation and user information:

'verbose'	write code generation progress to command window
'saveResult	save results to hard disk (always enabled, when genMFun and genSLBlock are set)
'logFile', logfile	write code generation progress to specified logfile
'genMFun'	generate robot specific m-functions
'genSLBlock'	generate real-time capable robot specific Simulink blocks
'genccode'	generate robot specific C-functions and -headers
'mex'	generate robot specific MEX-functions as replacement for the m-functions
'compilemex'	select whether generated MEX-function should be compiled directly after generation

Any option may also be modified individually as optional parameter value pairs.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

---

CodeGenerator.addpath

Adds generated code to search path

cGen.**addpath**() adds the generated m-functions and block library to the MATLAB function search path.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

addpath

CodeGenerator.genccodecoriolis

Generate C-function for robot inertia matrix

cGen.**genccodecoriolis**() generates robot-specific C-functions to compute the robot coriolis matrix.

Notes

- Is called by CodeGenerator.gencoriolis if cGen has active flag genccode or genmex.
- The .c and .h files are generated in the directory specified by the ccodepath property of the CodeGenerator object.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.gencoriolis, CodeGenerator.genmexcoriolis

---

CodeGenerator.genccodefdyn

Generate C-code for forward dynamics

cGen.**genccodeinvdyn**() generates a robot-specific C-code to compute the forward dynamics.

Notes

- Is called by CodeGenerator.genfdyn if cGen has active flag genccode or genmex.
- The .c and .h files are generated in the directory specified by the ccodepath property of the CodeGenerator object.
- The resulting C-function is composed of previously generated C-functions for the inertia matrix, Coriolis matrix, vector of gravitational load and joint friction vector. This function recombines these components to compute the forward dynamics.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfdyn, CodeGenerator.genccodeinvdyn

---

CodeGenerator.genccodefkine

Generate C-code for the forward kinematics

cGen.**genccodefkine**() generates a robot-specific C-function to compute forward kinematics.

Notes

- Is called by CodeGenerator.genfkine if cGen has active flag genccode or genmex
- The generated .c and .h files are wirtten to the directory specified in the ccodepath property of the CodeGenerator object.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfkine, CodeGenerator.genmexfkine

CodeGenerator.genccodefriction

Generate C-code for the joint friction

cGen.**genccodefriction**() generates a robot-specific C-function to compute vector of friction torques/forces.

Notes

- Is called by CodeGenerator.genfriction if cGen has active flag genccode or genmex
- The generated .c and .h files are wirtten to the directory specified in the ccodepath property of the CodeGenerator object.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfriction, CodeGenerator.genmexfriction

CodeGenerator.genccodegravload

Generate C-code for the vector of

gravitational load torques/forces

cGen.**genccodegravload**() generates a robot-specific C-function to compute vector of gravitational load torques/forces.

Notes

- Is called by CodeGenerator.gengravload if cGen has active flag genccode or genmex
- The generated .c and .h files are wirtten to the directory specified in the ccodepath property of the CodeGenerator object.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.gengravload, CodeGenerator.genmexgravload

---

CodeGenerator.genccodeinertia

Generate C-function for robot inertia matrix

cGen.**genccodeinertia**() generates robot-specific C-functions to compute the robot inertia matrix.

Notes

- Is called by CodeGenerator.geninertia if cGen has active flag genccode or genmex.
- The generated .c and .h files are generated in the directory specified by the ccodepath property of the CodeGenerator object.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninertia, CodeGenerator.genmexinertia

---

CodeGenerator.genccodeinvdyn

Generate C-code for inverse dynamics

cGen.**genccodeinvdyn**() generates a robot-specific C-code to compute the inverse dynamics.

Notes

- Is called by CodeGenerator.geninvdyn if cGen has active flag genccode or genmex.
- The .c and .h files are generated in the directory specified by the ccodepath property of the CodeGenerator object.
- The resulting C-function is composed of previously generated C-functions for the inertia matrix, coriolis matrix, vector of gravitational load and joint friction vector. This function recombines these components to compute the inverse dynamics.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn, CodeGenerator.genccodefdyn

---

CodeGenerator.genccodejacobian

Generate C-functions for robot jacobians

cGen.**genccodejacobian**() generates a robot-specific C-function to compute the jacobians with respect to the robot base as well as the end effector.

Notes

- Is called by CodeGenerator.genjacobian if cGen has active flag genccode or genmex.
- The generated .c and .h files are generated in the directory specified by the ccodepath property of the CodeGenerator object.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genccodefkine, CodeGenerator.genjacobian

---

CodeGenerator.gencoriolis

Generate code for Coriolis force

coriolis = cGen.**gencoriolis**() is a symbolic matrix $(N \times N)$ of centrifugal and Coriolis forces/torques.

Notes

- The Coriolis matrix is stored row by row to avoid memory issues. The generated code recombines these rows to output the full matrix.
- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninertia, CodeGenerator.genfkine

---

CodeGenerator.genfdyn

Generate code for forward dynamics

Iqdd = cGen.**genfdyn**() is a symbolic vector $(1 \times N)$ of joint inertial reaction forces/torques.

Notes

- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninertia, CodeGenerator.genfkine

---

CodeGenerator.genfkine

Generate code for forward kinematics

T = cGen.**genfkine**() generates a symbolic homogeneous transform matrix (4×4) representing the pose of the robot end-effector in terms of the symbolic joint coordinates q1, q2, ...

[**T**, **allt**] = cGen.**genfkine**() as above but also generates symbolic homogeneous transform matrices ($4 \times 4 \times N$) for the poses of the individual robot joints.

Notes

- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn, CodeGenerator.genjacobian

CodeGenerator.genfriction

Generate code for joint friction

f = cGen.**genfriction**() is the symbolic vector ($1 \times N$) of joint friction forces.

Notes

- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn, CodeGenerator.genfdyn

---

CodeGenerator.gengravload

Generate code for gravitational load

g = cGen.**gengravload**() is a symbolic vector ($1 \times N$) of joint load forces/torques due to gravity.

Notes

- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath

- genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
- mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator, CodeGenerator.geninvdyn, CodeGenerator.genfdyn

---

CodeGenerator.geninertia

Generate code for inertia matrix

i = cGen.**geninertia**() is the symbolic robot inertia matrix ($N \times N$).

Notes

- The inertia matrix is stored row by row to avoid memory issues. The generated code recombines these rows to output the full matrix.
- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn, CodeGenerator.genfdyn

---

CodeGenerator.geninvdyn

Generate code for inverse dynamics

tau = cGen.**geninvdyn**() is the symbolic vector ($1 \times N$) of joint forces/torques.

Notes

- The inverse dynamics vector is composed of the previously computed inertia matrix coriolis matrix, vector of gravitational load and joint friction for speedup. The generated code recombines these components to output the final vector.
- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfdyn, CodeGenerator.genfkine

CodeGenerator.genjacobian

Generate code for robot Jacobians

j0 = cGen.**genjacobian**() is the symbolic expression for the Jacobian matrix ($6 \times N$) expressed in the base coordinate frame.

[**j0**, **Jn**] = cGen.**genjacobian**() as above but also returns the symbolic expression for the Jacobian matrix ($6 \times N$) expressed in the end-effector frame.

Notes

- Side effects of execution depends on the cGen flags:
  - saveresult: the symbolic expressions are saved to disk in the directory specified by cGen.sympath
  - genmfun: ready to use m-functions are generated and provided via a subclass of SerialLink stored in cGen.robjpath
  - genslblock: a Simulink block is generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath
  - genccode: generates C-functions and -headers in the directory specified by the ccodepath property of the CodeGenerator object.
  - mex: generates robot specific MEX-functions as replacement for the m-functions mentioned above. Access is provided by the SerialLink subclass. The MEX files rely on the C code generated before.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfkine

CodeGenerator.genmexcoriolis

Generate C-MEX-function for robot coriolis matrix

cGen.**genmexcoriolis**() generates robot-specific MEX-functions to compute robot coriolis matrix.

Notes

- Is called by CodeGenerator.gencoriolis if cGen has active flag genmex
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated functions is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.gencoriolis

CodeGenerator.genmexfdyn

Generate C-MEX-function for forward dynamics

CGEN.**GENMEXFDYN**() generates a robot-specific MEX-function to compute the forward dynamics.

Notes

- Is called by CodeGenerator.genfdyn if cGen has active flag genmex
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.

- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfdyn, CodeGenerator.genmexinvdyn

---

CodeGenerator.genmexfkine

Generate C-MEX-function for forward kinematics

CGEN.**GENMEXFKINE**() generates a robot-specific MEX-function to compute forward kinematics.

Notes

- Is called by CodeGenerator.genfkine if cGen has active flag genmex
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfkine

CodeGenerator.genmexfriction

Generate C-MEX-function for joint friction

CGEN.**GENMEXFRICTION**() generates a robot-specific MEX-function to compute the vector of joint friction.

Notes

- Is called by CodeGenerator.genfriction if cGen has active flag genmex
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.gengravload

CodeGenerator.genmexgravload

Generate C-MEX-function for gravitational load

CGEN.**GENMEXGRAVLOAD**() generates a robot-specific MEX-function to compute gravitation load forces and torques.

Notes

- Is called by CodeGenerator.gengravload if cGen has active flag genmex
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.gengravload

CodeGenerator.genmexinertia

Generate C-MEX-function for robot inertia matrix

cGen.**genmexinertia**() generates robot-specific MEX-functions to compute robot inertia matrix.

Notes

- Is called by CodeGenerator.geninertia if cGen has active flag genmex
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated functions is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninertia

CodeGenerator.genmexinvdyn

Generate C-MEX-function for inverse dynamics

CGEN.**GENMEXINVDYN**() generates a robot-specific MEX-function to compute the inverse dynamics.

Notes

- Is called by CodeGenerator.geninvdyn if cGen has active flag genmex.
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.gengravload

CodeGenerator.genmexjacobian

Generate C-MEX-function for the robot Jacobians

CGEN.**GENMEXJACOBIAN**() generates robot-specific MEX-function to compute the robot Jacobian with respect to the base as well as the end effector frame.

Notes

- Is called by CodeGenerator.genjacobian if cGen has active flag genmex.
- The MEX file uses the .c and .h files generated in the directory specified by the ccodepath property of the CodeGenerator object.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.
- You will need a C compiler to use the generated MEX-functions. See the MATLAB documentation on how to setup the compiler in MATLAB. Nevertheless the basic C-MEX-code as such may be generated without a compiler. In this case switch the cGen flag compilemex to false.

Author

Joern Malzahn, (joern.malzahn@tu-dortmund.de)

See also

CodeGenerator.CodeGenerator, CodeGenerator.genjacobian

CodeGenerator.genmfuncoriolis

Generate M-functions for Coriolis matrix

cGen.**genmfuncoriolis**() generates a robot-specific M-function to compute the Coriolis matrix.

Notes

- Is called by CodeGenerator.gencoriolis if cGen has active flag genmfun
- The Coriolis matrix is stored row by row to avoid memory issues.
- The generated M-function recombines the individual M-functions for each row.

- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.gencoriolis, CodeGenerator.geninertia

---

CodeGenerator.genmfunfdyn

Generate M-function for forward dynamics

cGen.**genmfunfdyn**() generates a robot-specific M-function to compute the forward dynamics.

Notes

- Is called by CodeGenerator.genfdyn if cGen has active flag genmfun
- The generated M-function is composed of previously generated M-functions for the inertia matrix, coriolis matrix, vector of gravitational load and joint friction vector. This function recombines these components to compute the forward dynamics.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn

CodeGenerator.genmfunfkine

Generate M-function for forward kinematics

cGen.**genmfunfkine**() generates a robot-specific M-function to compute forward kinematics.

Notes

- Is called by CodeGenerator.genfkine if cGen has active flag genmfun
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genjacobian

CodeGenerator.genmfunfriction

Generate M-function for joint friction

cGen.**genmfunfriction**() generates a robot-specific M-function to compute joint friction.

Notes

- Is called only if cGen has active flag genmfun
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.gengravload

CodeGenerator.genmfungravload

Generate M-functions for gravitational load

cGen.**genmfungravload**() generates a robot-specific M-function to compute gravitation load forces and torques.

Notes

- Is called by CodeGenerator.gengravload if cGen has active flag genmfun
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninertia

CodeGenerator.genmfuninertia

Generate M-function for robot inertia matrix

cGen.**genmfuninertia**() generates a robot-specific M-function to compute robot inertia matrix.

Notes

- Is called by CodeGenerator.geninertia if cGen has active flag genmfun
- The inertia matrix is stored row by row to avoid memory issues.
- The generated M-function recombines the individual M-functions for each row.

- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.gencoriolis

CodeGenerator.genmfuninvdyn

Generate M-functions for inverse dynamics

cGen.**genmfuninvdyn**() generates a robot-specific M-function to compute inverse dynamics.

Notes

- Is called by CodeGenerator.geninvdyn if cGen has active flag genmfun
- The generated M-function is composed of previously generated M-functions for the inertia matrix, coriolis matrix, vector of gravitational load and joint friction vector. This function recombines these components to compute the inverse dynamics.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn

CodeGenerator.genmfunjacobian

Generate M-functions for robot Jacobian

cGen.**genmfunjacobian**() generates a robot-specific M-function to compute robot Jacobian.

Notes

- Is called by CodeGenerator.genjacobian, if cGen has active flag genmfun
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.gencoriolis

CodeGenerator.genslblockcoriolis

Generate Simulink block for Coriolis matrix

cGen.**genslblockcoriolis**() generates a robot-specific Simulink block to compute Coriolis/centripetal matrix.

Notes

- Is called by CodeGenerator.gencoriolis if cGen has active flag genslblock
- The Coriolis matrix is stored row by row to avoid memory issues.
- The Simulink block recombines the the individual blocks for each row.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.gencoriolis

CodeGenerator.genslblockfdyn

Generate Simulink block for forward dynamics

cGen.**genslblockfdyn**() generates a robot-specific Simulink block to compute forward dynamics.

Notes

- Is called by CodeGenerator.genfdyn if cGen has active flag genslblock
- The generated Simulink block is composed of previously generated blocks for the inertia matrix, coriolis matrix, vector of gravitational load and joint friction vector. The block recombines these components to compute the forward dynamics.
- Access to generated function is provided via subclass of SerialLink whose class definition is stored in cGen.robjpath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfdyn

CodeGenerator.genslblockfkine

Generate Simulink block for forward kinematics

cGen.**genslblockfkine**() generates a robot-specific Simulink block to compute forward kinematics.

Notes

- Is called by CodeGenerator.genfkine if cGen has active flag genslblock.
- The Simulink blocks are generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath.
- Blocks are created for intermediate transforms T0, T1 etc. as well.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfkine

CodeGenerator.genslblockfriction

Generate Simulink block for joint friction

cGen.**genslblockfriction**() generates a robot-specific Simulink block to compute the joint friction model.

Notes

- Is called by CodeGenerator.genfriction if cGen has active flag genslblock
- The Simulink blocks are generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genfriction

CodeGenerator.genslblockgravload

Generate Simulink block for gravitational load

cGen.**genslblockgravload**() generates a robot-specific Simulink block to compute gravitational load.

Notes

- Is called by CodeGenerator.gengravload if cGen has active flag genslblock
- The Simulink blocks are generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also **CodeGenerator**.**CodeGenerator**, **CodeGenerator**.gengravload

---

CodeGenerator.genslblockinertia

Generate Simulink block for inertia matrix

cGen.**genslbgenslblockinertia**() generates a robot-specific Simulink block to compute robot inertia matrix.

Notes

- Is called by CodeGenerator.geninertia if cGen has active flag genslblock
- The Inertia matrix is stored row by row to avoid memory issues.
- The Simulink block recombines the the individual blocks for each row.
- The Simulink blocks are generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninertia

CodeGenerator.genslblockinvdyn

Generate Simulink block for inverse dynamics

cGen.**genslblockinvdyn**() generates a robot-specific Simulink block to compute inverse dynamics.

Notes

- Is called by CodeGenerator.geninvdyn if cGen has active flag genslblock
- The generated Simulink block is composed of previously generated blocks for the inertia matrix, coriolis matrix, vector of gravitational load and joint friction vector.% The block recombines these components to compute the forward dynamics.
- The Simulink blocks are generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.geninvdyn

CodeGenerator.genslblockjacobian

Generate Simulink block for robot Jacobians

cGen.**genslblockjacobian**() generates a robot-specific Simulink block to compute robot Jacobians (world and tool frame).

Notes

- Is called by CodeGenerator.genjacobian if cGen has active flag genslblock
- The Simulink blocks are generated and stored in a robot specific block library cGen.slib in the directory cGen.basepath.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

CodeGenerator.CodeGenerator, CodeGenerator.genjacobian

CodeGenerator.logmsg

Print CodeGenerator logs.

count = CGen.logmsg(FORMAT, A, ...) is the number of characters written to the CGen.logfile. For the additional arguments see fprintf.

Note

Matlab ships with a function for writing formatted strings into a text file or to the console (fprintf). The function works with single target identifiers (file, console, string). This function uses the same syntax as for the fprintf function to output log messages to either the Matlab console, a log file or both.

Authors

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany http://www.rst.e-technik.tu-dortmund.de

See also

multidfprintf, fprintf, sprintf

CodeGenerator.purge

Cleanup generated files

cGen.**purge**() deletes all generated files, first displays a question dialog to make sure the user really wants to delete all generated files.

cGen.**purge**(1) as above but skips the question dialog.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

---

CodeGenerator.rmpath

Removes generated code from search path

cGen.**rmpath**() removes generated m-functions and block library from the MATLAB function search path.

Author

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

rmpath

---

colnorm

Column-wise norm of a matrix

cn = **colnorm**(**a**) is an $M \times 1$ vector of the normals of each column of the matrix **a** which is $N \times M$.

---

colorname

Map between color names and RGB values

rgb = **colorname**(**name**) is the **rgb**-tristimulus value (1×3) corresponding to the color specified by the string **name**. If **rgb** is a cell-array ($1 \times N$) of names then **rgb** is a matrix ($N \times 3$) with each row being the corresponding tristimulus.

XYZ = **colorname**(**name**, 'xyz') as above but the XYZ-tristimulus value corresponding to the color specified by the string **name**.

XY = **colorname**(**name**, 'xy') as above but the xy-chromaticity coordinates corresponding to the color specified by the string **name**.

name = **colorname**(**rgb**) is a string giving the name of the color that is closest (Euclidean) to the given **rgb**-tristimulus value (1×3). If **rgb** is a matrix ($N \times 3$) then return a cell-array ($1 \times N$) of color names.

name = **colorname**(XYZ, 'xyz') as above but the color is the closest (Euclidean) to the given XYZ-tristimulus value.

name = **colorname**(XYZ, 'xy') as above but the color is the closest (Euclidean) to the given xy-chromaticity value with assumed Y=1.

Notes

- Color name may contain a wildcard, eg. "?burnt"
- Based on the standard X11 color database rgb.txt.
- Tristimulus values are in the range 0 to 1

Contents

Toolbox.

Version 9.9.0 2014-04-28

Homogeneous transformations 3D

angvec2r	- angle/vector to RM
angvec2tr	- angle/vector to HT
eul2r	- Euler angles to RM
eul2tr	- Euler angles to HT
oa2r	- orientation and approach vector to RM
oa2tr	- orientation and approach vector to HT
r2t	- RM to HT
rt2tr	- (R,t) to HT
rotx	- RM for rotation about X-axis
roty	- RM for rotation about Y-axis
rotz	- RM for rotation about Z-axis
rpy2r	- roll/pitch/yaw angles to RM
rpy2tr	- roll/pitch/yaw angles to HT
t2r	- HT to RM
tr2angvec	- HT/RM to angle/vector form
tr2eul	- HT/RM to Euler angles
tr2rpy	- HT/RM to roll/pitch/yaw angles
tr2rt	- HT to (R,t)
tranimate	- animate a coordinate frame
trchain	- evaluate a series of transforms
tripleangle	- graphical interactive three angle rotation
transl	- set or extract the translational component of HT
trnorm	- normalize HT
trchain	- chain of SE(3) transforms
trplot	- plot HT as a coordinate frame
trprint	- print an HT
trotx	- HT for rotation about X-axis
troty	- HT for rotation about Y-axis
trotz	- HT for rotation about Z-axis
trscale	- HT for scale change

* HT: homogeneous transformation, a 4×4 matrix, in SE(3) * RM: rotation matrix, orthonormal 3×3 matrix, in SO(3) * Functions of the form tr2XX will also accept an HT or RM as the argument

Homogeneous transformations 2D

rot2	- RM for SE(2) rotation
se2	- HT in SE(2)
transl2	- set or extract the translational component of SE(2) HT
trchain2	- chain of SE(2) transforms
trot2	- SO(2) rotation
trplot2	- plot HT, SE(2), as a coordinate frame

* HT: homogeneous transformation, a 3×3 matrix, in SE(2) * RM: rotation matrix, orthonormal 2×2 matrix, in SO(2)

Homogeneous points and lines

e2h	- Euclidean coordinates to homogeneous
h2e	- homogeneous coordinates to Euclidean
homline	- create line from 2 points
homtrans	- transform points

Differential motion

delta2tr	- differential motion vector to HT
eul2jac	- Euler angles to Jacobian
rpy2jac	- RPY angles to Jacobian
skew	- vector to skew symmetric matrix
tr2delta	- HT to differential motion vector
tr2jac	- HT to Jacobian
vex	- skew symmetric matrix to vector
wtrans	- transform wrench between frames

Trajectory generation

ctraj	- Cartesian trajectory
jtraj	- joint space trajectory
lspb	- 1D trapezoidal trajectory
mtraj	- multi-axis trajectory for arbitrary function
mstraj	- multi-axis multi-segment trajectory
tpoly	- 1D polynomial trajectory
trinterp	- interpolate HT s

Quaternion

Quaternion	- constructor
/	- divide quaternion by quaternion or scalar
	- multiply quaternion by a quaternion or vector
inv	- invert a quaternion
norm	- norm of a quaternion
plot	- display a quaternion as a 3D rotation
unit	- unitize a quaternion
interp	- interpolate a quaternion

Serial-link manipulator

CodeGenerator	- construct a robot specific code generator object
SerialLink	- construct a serial-link robot object
Link	- construct a general robot link object
Prismatic	- construct a prismatic robot link object
Revolute	- construct a revolute robot link object
PrismaticMDH	- construct a prismatic robot link object
RevoluteMDH	- construct a revolute robot link object
	- compound two robots
friction	- return joint friction torques
nofriction	- return a robot object with no friction
perturb	- return a robot object with perturbed parameters
plot	- plot/animate robot
plot3d	- plot/animate robot as solid model
teach	- drive a graphical robot

```
Models

mdl_Fanuc10L          - Fanuc 10L (DH, kine)
mdl_irb140            - ABB IRB140 (DH, kine)
mdl_irb140_mdh        - ABB IRB140 (MDH, kine)
mdl_jaco              - Kinova Jaco arm (DH, kine)
mdl_m16               - Fanuc M16 (DH, kine)
mdl_mico              - Kinova Mico arm (DH, kine)
mdl_MotomanHP6        - Motoman HP6 (DH, kine)
mdl_nao               - Alderabaran NAO arms and legs (DH, kine)
mdl_phantomx          - PhantomX pincher 4DOF hobby arm (DH, kine)
mdl_puma560           - Puma 560 data (DH, kine, dyn)
mdl_puma560akb        - Puma 560 data (MDH, kine, dyn)
mdl_S4ABB2p8          - ABB S4 2.8 (DH, kine)
mdl_stanford          - Stanford arm data (DH, kine, dyn)
mdl_stanford_mdh      - Stanford arm data (MDH, kine, dyn)
mdl_onelink           - simple 1-link example (DH, kine)
mdl_planar1           - simple 1 link planar model (DH, kine)
mdl_planar2           - simple 2 link planar model (DH, kine)
mdl_planar3           - simple 3 link planar model (DH, kine)
mdl_3link3d           - Simple 3DOF arm, no shoulder offset (DH, kine)
mdl_twolink           - simple 2-link example (DH, kine, dyn)
mdl_twolink_mdh       - simple 2-link example (MDH, kine)
mdl_ball              - high DOF chain that forms a ball
mdl_coil              - high DOF chain that forms a coil
mdl_hyper2d           - 2D high DOF chain
mdl_hyper3d           - 3D high DOF chain
mdl_quadcopter        - simple quadcopter model

Kinematic

DHFactor              - transform sequence to DH description
jsingu                - find dependent joints
fkine                 - forward kinematics
ikine                 - inverse kinematics (numeric)
ikine_sym             - inverse kinematics (symbolic)
ikine6s               - inverse kinematics for 6-axis arm with sph.wrist
jacob0                - Jacobian in base coordinate frame
jacobn                - Jacobian in end-effector coordinate frame
maniplty              - compute manipulability

Dynamics

accel                 - forward dynamics
cinertia              - Cartesian manipulator inertia matrix
coriolis              - centripetal/coriolis torque
```

```
fdyn                    - forward dynamics
wtrans                  - transform a force/moment
gravload                - gravity loading
inertia                 - manipulator inertia matrix
itorque                 - inertia torque
rne                     - inverse dynamics
```

Mobile robot

Map	- point feature map object
RandomPath	- driver for Vehicle object
RangeBearingSensor	- "laser scanner" object
Vehicle	- construct a mobile robot object
sl_bicycle	- Simulink "bicycle model" of non-holonomic wheeled vehicle
Navigation	- Navigation superclass
Sensor	- robot sensor superclass
makemap	- build an occupancy grid
plot_vehicle	- plot vehicle

```
Localization

EKF                     - extended Kalman filter object
ParticleFilter          - Monte Carlo estimator

Path planning

Bug2                    - bug navigation
DXform                  - distance transform from map
Dstar                   - D* planner
PRM                     - probabilistic roadmap planner
RRT                     - rapidly exploring random tree
```

Graphics

plot2	- plot trajectory
plotp	- plot points
plot_arrow	- draw an arrow
plot_box	- draw a box
plot_circle	- draw a circle
plot_ellipse	- draw an ellipse
plot_homline	- plot homogeneous line
plot_point	- plot points
plot_poly	- plot polygon
plot_sphere	- draw a sphere
qplot	- plot joint angle trajectories
plot2	- Plot trajectories
plotp	- Plot trajectories
xaxis	- set x-axis scaling
yaxis	- set y-axis scaling
xyzlabel	- label axes x, y and z

Utility

about	- summary of object size and type
angdiff	- subtract 2 angles modulo 2pi
arrow3	- draw a 3D arrow (third party code)
bresenham	- Bresenhan line drawing
circle	- compute/draw points on a circle
colnorm	- columnwise norm of matrix
colorname	- map color name to RGB
diff2	- elementwise diff
edgelist	- trace edge of a shape
gauss2d	- Gaussian distribution in 2D
ishomog	- true if argument is a 4×4 matrix
ismatrix	- true if non scalar
isrot	- true if argument is a 3×3 matrix
isvec	- true if argument is a 3-vector
numcols	- number of columns in matrix
numrows	- number of rows in matrix
peak	- find peak in 1D signal
peak2	- find peak in 2D signal
PGraph	- general purpose graph class
polydiff	- derivative of polynomial
Polygon	- general purpose polygon class
randinit	- initialize random number generator
ramp	- create linear ramp
unit	- unitize a vector
tb_optparse	- toolbox argument parser
distancexform	- compute distance transform
runscript	- interactively step through a script
multidfprintf	- printf extension

Demonstrations

rtbdemo	- Serial-link manipulator demonstration
tripleangle	- demonstrate angle sequences

Interfacing

RobotArm	- Connect SerialLink object to real robot
joystick	- Help for joystick interface mex file
joy2tr	- Update HT based on joystick input
VREP	- VREP interface class
VREP_mirror	- MATLAB mirror for VREP object
VREP_arm	- MATLAB mirror for VREP robot arm
VREP_obj	- MATLAB mirror for VREP object
VREP_camera	- MATLAB mirror for VREP camera object

* Arbotix class in the folder robot/interfaces * VREP classes are in the folder robot/interfaces/VREP

Examples

sl_quadcopter	- Simulink model of a flying quadcopter
sl_braitenberg	- Simulink model a Braitenberg vehicle
movepoint	- non-holonomic vehicle moving to a point
moveline	- non-holonomic vehicle moving to a line
movepose	- non-holonomic vehicle moving to a pose
walking	- example of 4-legged walking robot
eg_inertia	- joint 1 inertia I(q1,q2)
eg_inertia22	- joint 2 inertia I(q3)
eg_grav	- joint 2 gravity load g(q2,q3)

* located in the examples folder

copyright (**C**) 2011 Peter Corke

ctraj

Cartesian trajectory between two points

tc = **ctraj**(**T0**, **T1**, **n**) is a Cartesian trajectory ($4 \times 4 \times$ **n**) from pose **T0** to **T1** with **n** points that follow a trapezoidal velocity profile along the path. The Cartesian trajectory is a homogeneous transform sequence and the last subscript being the point index, that is, T(:,:,i) is the i'th point along the path.

tc = **ctraj**(**T0**, **T1**, **s**) as above but the elements of **s** (**n** $\times$ 1) specify the fractional distance along the path, and these values are in the range [0 1]. The i'th point corresponds to a distance **s**(i) along the path.

Notes

- If **T0** or **T1** is equal to [] it is taken to be the identity matrix.

See also

lspb, mstraj, trinterp, Quaternion.interp, transl

delta2tr

Convert differential motion to a homogeneous transform

T = **delta2tr**(**d**) is a homogeneous transform representing differential translation and rotation. The vector **d**=(dx, dy, dz, dRx, dRy, dRz) represents an infinitessimal motion, and is an approximation to the spatial velocity multiplied by time.

See also

tr2delta

DHFactor

Simplify symbolic link transform expressions

f = **dhfactor**(**s**) is an object that encodes the kinematic model of a robot provided by a string **s** that represents a chain of elementary transforms from the robot's base to its tool tip. The chain of elementary rotations and translations is symbolically factored into a sequence of link transforms described by DH parameters.

For example:

```
s = 'Rz(q1).Rx(q2).Ty(L1).Rx(q3).Tz(L2)';
```

indicates a rotation of q1 about the z-axis, then rotation of q2 about the x-axis, translation of L1 about the y-axis, rotation of q3 about the x-axis and translation of L2 along the z-axis.

Methods

base	the base transform as a Java string
tool	the tool transform as a Java string
command	a command string that will create a SerialLink() object representing the specified kinematics
char	convert to string representation
display	display in human readable form

Example

```
>> s = 'Rz(q1).Rx(q2).Ty(L1).Rx(q3).Tz(L2)';
>> dh = DHFactor(s);
>> dh
DH(q1+90, 0, 0, +90).DH(q2, L1, 0, 0).DH(q3-90, L2, 0, 0).Rz(+90).Rx(-90).Rz(-90)
>> r = eval( dh.command('myrobot') );
```

Notes

- Variables starting with q are assumed to be joint coordinates

- Variables starting with L are length constants.
- Length constants must be defined in the workspace before executing the last line above.
- Implemented in Java
- Not all sequences can be converted to DH format, if conversion cannot be achieved an error is generated.

Reference

- A simple and systematic approach to assigning Denavit-Hartenberg parameters, P.Corke, IEEE Transaction on Robotics, vol. 23, pp. 590-594, June 2007.
- Robotics, Vision & Control, Sec 7.5.2, 7.7.1, Peter Corke, Springer 2011.

See also

SerialLink

---

diff2

Two point difference

d = **diff2**(**v**) is the 2-point difference for each point in the vector v and the first element is zero. The vector **d** has the same length as **v**.

See also

diff

---

distancexform

Distance transform of occupancy grid

d = **distancexform**(**world**, **goal**) is the distance transform of the occupancy grid **world** with respect to the specified goal point **goal** = [X,Y]. The elements of the grid are 0 from free space and 1 for occupied.

d = distancexform(**world**, **goal**, **metric**) as above but specifies the distance metric as either 'cityblock' or 'Euclidean'

d = distancexform(**world**, **goal**, **metric**, **show**) as above but shows an animation of the distance transform being formed, with a delay of **show** seconds between frames.

Notes

- The Machine Vision Toolbox function imorph is required.
- The goal is [X,Y] not MATLAB [row,col]

See also

imorph, DXform

Dstar

D* navigation class

A concrete subclass of the Navigation class that implements the D* navigation algorithm. This provides minimum distance paths and facilitates incremental replanning.

Methods

plan	Compute the cost map given a goal and map
path	Compute a path to the goal
visualize	Display the obstacle map (deprecated)
plot	Display the obstacle map
costmap_modify	Modify the costmap
modify_cost	Modify the costmap (deprecated, use costmap_modify)
costmap_get	Return the current costmap
costmap_set	Set the current costmap
distancemap_get	Set the current distance map
display	Print the parameters in human readable form
char	Convert to string

Properties

costmap	Distance from each point to the goal.

Example

```
load map1            % load map
goal = [50,30];
start=[20,10];
ds = Dstar(map);     % create navigation object
ds.plan(goal)        % create plan for specified goal
ds.path(start)       % animate path from this start location
```

Notes

- Obstacles are represented by Inf in the costmap.
- The value of each element in the costmap is the shortest distance from the corresponding point in the map to the current goal.

References

- The D* algorithm for real-time planning of optimal traverses, A. Stentz, Tech. Rep. CMU-RI-TR-94-37, The Robotics Institute, Carnegie-Mellon University, 1994.
- Robotics, Vision & Control, Sec 5.2.2, Peter Corke, Springer, 2011.

See also

Navigation, DXform, PRM

Dstar.Dstar

D* constructor

ds = Dstar(**map**, **options**) is a D* navigation object, and **map** is an occupancy grid, a representation of a planar world as a matrix whose elements are 0 (free space) or 1 (occupied). The occupancy grid is coverted to a costmap with a unit cost for traversing a cell.

Options

'goal', G	Specify the goal point (2×1)
'metric', M	Specify the distance metric as 'euclidean' (default) or 'cityblock'.
'inflate', K	Inflate all obstacles by K cells.
'quiet'	Don't display the progress spinner

Other options are supported by the Navigation superclass.

See also

Navigation.Navigation

Dstar.char

Convert navigation object to string

DS.**char**() is a string representing the state of the **Dstar** object in human-readable form.

See also

Dstar.display, Navigation.char

Dstar.costmap_get

Get the current costmap

C = DS.**costmap_get**() is the current costmap. The cost map is the same size as the occupancy grid and the value of each element represents the cost of traversing the cell. It is autogenerated by the class constructor from the occupancy grid such that:

- free cell (occupancy 0) has a cost of 1
- occupied cell (occupancy $>$0) has a cost of Inf

See also

Dstar.costmap_set, Dstar.costmap_modify

Dstar.costmap_modify

Modify cost map

DS.**costmap_modify**(**p**, **new**) modifies the cost map at **p**=[X,Y] to have the value **new**. If **p** $(2 \times M)$ and **new** $(1 \times M)$ then the cost of the points defined by the columns of **p** are set to the corresponding elements of **new**.

Notes

- After one or more point costs have been updated the path should be replanned by calling DS.plan().
- Replaces modify_cost, same syntax.

See also

Dstar.costmap_set, Dstar.costmap_get

Dstar.costmap_set

Set the current costmap

DS.**costmap_set**(**C**) sets the current costmap. The cost map is the same size as the occupancy grid and the value of each element represents the cost of traversing the cell. A high value indicates that the cell is more costly (difficult) to traverese. A value of Inf indicates an obstacle.

Notes

- After the cost map is changed the path should be replanned by calling DS.plan().

See also

Dstar.costmap_get, Dstar.costmap_modify

Dstar.distancemap_get

Get the current distance map

C = DS.**distancemap_get**() is the current distance map. This map is the same size as the occupancy grid and the value of each element is the shortest distance from the corresponding point in the map to the current goal. It is computed by **Dstar**.plan.

See also

Dstar.plan

Dstar.modify_cost

Modify cost map

Notes

- Deprecated: use modify_cost instead instead.

See also

Dstar.costmap_set, Dstar.costmap_get

Dstar.plan

Plan path to goal

DS.**plan**() updates DS with a costmap of distance to the goal from every non-obstacle point in the map. The goal is as specified to the constructor.

DS.**plan**(**goal**) as above but uses the specified goal.

Note

- If a path has already been planned, but the costmap was modified, then reinvoking this method will replan, incrementally updating the **plan** at lower cost than a full replan.

Dstar.plot

Visualize navigation environment

DS.**plot**() displays the occupancy grid and the goal distance in a new figure. The goal distance is shown by intensity which increases with distance from the goal. Obstacles are overlaid and shown in red.

DS.**plot**(**p**) as above but also overlays a path given by the set of points **p** ($M \times 2$).

See also

Navigation.plot

Dstar.reset

Reset the planner

DS.**reset**() resets the D* planner. The next instantiation of DS.plan() will perform a global replan.

DXform

Distance transform navigation class

A concrete subclass of the Navigation class that implements the distance transform navigation algorithm which computes minimum distance paths.

Methods

plan	Compute the cost map given a goal and map
path	Compute a path to the goal
visualize	Display the obstacle map (deprecated)
plot	Display the distance function and obstacle map
plot3d	Display the distance function as a surface
display	Print the parameters in human readable form
char	Convert to string

Properties

distancemap	The distance transform of the occupancy grid.
metric	The distance metric, can be 'euclidean' (default) or 'cityblock'

Example

```
load map1           % load map
goal = [50,30];     % goal point
start = [20, 10];   % start point
dx = DXform(map);   % create navigation object
dx.plan(goal)       % create plan for specified goal
dx.path(start)      % animate path from this start location
```

Notes

- Obstacles are represented by NaN in the distancemap.
- The value of each element in the distancemap is the shortest distance from the corresponding point in the map to the current goal.

References

- Robotics, Vision & Control, Sec 5.2.1, Peter Corke, Springer, 2011.

See also

Navigation, Dstar, PRM, distancexform

DXform.DXform

Distance transform constructor

dx = **DXform**(**map**, **options**) is a distance transform navigation object, and **map** is an occupancy grid, a representation of a planar world as a matrix whose elements are 0 (free space) or 1 (occupied).

Options

'goal', G	Specify the goal point (2×1)
'metric', M	Specify the distance metric as 'euclidean' (default) or 'cityblock'.
'inflate', K	Inflate all obstacles by K cells.

Other options are supported by the Navigation superclass.

See also

Navigation.Navigation

DXform.char

Convert to string

DX.**char**() is a string representing the state of the object in human-readable form.

See also **DXform**.display, Navigation.**char**

DXform.plan

Plan path to goal

DX.**plan**() updates the internal distancemap where the value of each element is the minimum distance from the corresponding point to the goal. The goal is as specified to the constructor.

DX.**plan**(**goal**) as above but uses the specified goal.

DX.**plan**(**goal**, **s**) as above but displays the evolution of the distancemap, with one iteration displayed every **s** seconds.

Notes

- This may take many seconds.

DXform.plot

Visualize navigation environment

DX.**plot**() displays the occupancy grid and the goal distance in a new figure. The goal distance is shown by intensity which increases with distance from the goal. Obstacles are overlaid and shown in red.

DX.**plot**(**p**) as above but also overlays a path given by the set of points **p** ($M \times 2$).

See also

Navigation.plot

DXform.plot3d

3D costmap view

DX.**plot3d**() displays the distance function as a 3D surface with distance from goal as the vertical axis. Obstacles are "cut out" from the surface.

DX.**plot3d**(**p**) as above but also overlays a path given by the set of points **p** ($M \times 2$).

DX.**plot3d**(**p**, **ls**) as above but plot the line with the linestyle **ls**.

See also

Navigation.plot

e2h

Euclidean to homogeneous

H = **e2h**(**E**) is the homogeneous version (K+1 $\times N$) of the Euclidean points **E** ($K \times N$) where each column represents one point in R^K.

See also

h2e

edgelist

Return list of edge pixels for region

E = **edgelist**(**im**, **seed**) is a list of edge pixels of a region in the image **im** starting at edge coordinate **seed** (i,j). The result **E** is a matrix, each row is one edge point coordinate (x,y).

E = **edgelist**(**im**, **seed**, **direction**) is a list of edge pixels as above, but the direction of edge following is specified. **direction** == 0 (default) means clockwise, non zero is counter-clockwise. Note that direction is with respect to y-axis upward, in matrix coordinate frame, not image frame.

[**E**,**d**] = **edgelist**(**im**, **seed**, **direction**) as above but also returns a vector of edge segment directions which have values 1 to 8 representing W SW S SE **E** NW N NW respectively.

Notes

- **im** is a binary image where 0 is assumed to be background, non-zero is an object.
- **seed** must be a point on the edge of the region.
- The seed point is always the first element of the returned **edgelist**.

Reference

- METHODS TO ESTIMATE AREAS AND PERIMETERS OF BLOB-LIKE OBJECTS: A COMPARISON Luren Yang, Fritz Albregtsen, Tor Lgnnestad and Per Grgttum IAPR Workshop on Machine Vision Applications Dec. 13-15, 1994, Kawasaki

See also

ilabel

EKF

Extended Kalman Filter for navigation

This class can be used for:

- dead reckoning localization
- map-based localization
- map making
- simultaneous localization and mapping (SLAM)

It is used in conjunction with:

- a kinematic vehicle model that provides odometry output, represented by a Vehicle object.
- The vehicle must be driven within the area of the map and this is achieved by connecting the Vehicle object to a Driver object.
- a map containing the position of a number of landmark points and is represented by a Map object.
- a sensor that returns measurements about landmarks relative to the vehicle's location and is represented by a Sensor object subclass.

The EKF object updates its state at each time step, and invokes the state update methods of the Vehicle. The complete history of estimated state and covariance is stored within the EKF object.

Methods

run	run the filter
plot_xy	plot the actual path of the vehicle
plot_P	plot the estimated covariance norm along the path
plot_map	plot estimated feature points and confidence limits
plot_ellipse	plot estimated path with covariance ellipses
plot_error	plot estimation error with standard deviation bounds
display	print the filter state in human readable form
char	convert the filter state to human readable string

Properties

x_est	estimated state
P	estimated covariance
V_est	estimated odometry covariance
W_est	estimated sensor covariance
features	maps sensor feature id to filter state element
robot	reference to the Vehicle object
sensor	reference to the Sensor subclass object
history	vector of structs that hold the detailed filter state from each time step
verbose	show lots of detail (default false)
joseph	use Joseph form to represent covariance (default true)

Vehicle position estimation (localization)

Create a vehicle with odometry covariance V, add a driver to it, create a Kalman filter with estimated covariance V_est and initial state covariance P0

```
veh = Vehicle(V);
veh.add_driver( RandomPath(20, 2) );
ekf = EKF(veh, V_est, P0);
```

We run the simulation for 1000 time steps

```
ekf.run(1000);
```

then plot true vehicle path

```
veh.plot_xy('b');
```

and overlay the estimated path

```
ekf.plot_xy('r');
```

and overlay uncertainty ellipses at every 20 time steps

```
ekf.plot_ellipse(20, 'g');
```

We can plot the covariance against time as

```
clf
ekf.plot_P();
```

Map-based vehicle localization

Create a vehicle with odometry covariance V, add a driver to it, create a map with 20 point features, create a sensor that uses the map and vehicle state to estimate feature range and bearing with covariance W, the Kalman filter with estimated covariances V_est and W_est and initial vehicle state covariance P0

```
veh = Vehicle(V);
veh.add_driver( RandomPath(20, 2) );
map = Map(20);
sensor = RangeBearingSensor(veh, map, W);
ekf = EKF(veh, V_est, P0, sensor, W_est, map);
```

We run the simulation for 1000 time steps

```
ekf.run(1000);
```

then plot the map and the true vehicle path

```
map.plot();
veh.plot_xy('b');
```

and overlay the estimatd path

```
ekf.plot_xy('r');
```

and overlay uncertainty ellipses at every 20 time steps

```
ekf.plot_ellipse([], 'g');
```

We can plot the covariance against time as

```
clf
ekf.plot_P();
```

Vehicle-based map making

Create a vehicle with odometry covariance V, add a driver to it, create a sensor that uses the map and vehicle state to estimate feature range and bearing with covariance W, the Kalman filter with estimated sensor covariance W_est and a “perfect” vehicle (no covariance), then run the filter for N time steps.

```
veh = Vehicle(V);
veh.add_driver( RandomPath(20, 2) );
sensor = RangeBearingSensor(veh, map, W);
ekf = EKF(veh, [], [], sensor, W_est, []);
```

We run the simulation for 1000 time steps

```
ekf.run(1000);
```

Then plot the true map

```
map.plot();
```

and overlay the estimated map with 3 sigma ellipses

```
ekf.plot_map(3, 'g');
```


Simultaneous localization and mapping (SLAM)

Create a vehicle with odometry covariance V, add a driver to it, create a map with 20 point features, create a sensor that uses the map and vehicle state to estimate feature range and bearing with covariance W, the Kalman filter with estimated covariances V_est and W_est and initial state covariance P0, then run the filter to estimate the vehicle state at each time step and the map.

```
veh = Vehicle(V);
veh.add_driver( RandomPath(20, 2) );
map = Map(20);
sensor = RangeBearingSensor(veh, map, W);
ekf = EKF(veh, V_est, P0, sensor, W, []);
```

We run the simulation for 1000 time steps

```
ekf.run(1000);
```

then plot the map and the true vehicle path

```
map.plot();
veh.plot_xy('b');
```

and overlay the estimated path

```
ekf.plot_xy('r');
```

and overlay uncertainty ellipses at every 20 time steps

```
ekf.plot_ellipse([], 'g');
```

We can plot the covariance against time as

```
clf
ekf.plot_P();
```

Then plot the true map

```
map.plot();
```

and overlay the estimated map with 3 sigma ellipses

```
ekf.plot_map(3, 'g');
```

Reference

Robotics, Vision & Control, Chap 6, Peter Corke, Springer 2011

Acknowledgement

Inspired by code of Paul Newman, Oxford University, http://www.robots.ox.ac.uk/ pnewman

See also

Vehicle, RandomPath, RangeBearingSensor, Map, ParticleFilter

EKF.EKF

EKF object constructor

E = EKF(**vehicle**, **v_est**, **p0**, **options**) is an EKF that estimates the state of the **vehicle** with estimated odometry covariance **v_est** (2×2) and initial covariance (3×3).

E = EKF(**vehicle**, **v_est**, **p0**, **sensor**, **w_est**, **map**, **options**) as above but uses information from a **vehicle** mounted sensor, estimated sensor covariance **w_est** and a **map**.

Options

'verbose'	Be verbose.
'nohistory'	Don't keep history.
'joseph'	Use Joseph form for covariance
'dim', D	Dimension of the robot's workspace. Scalar D is $D \times D$, 2-vector D(1)xD(2), 4-vector is D(1)<x<D(2), D(3)<y<D(4).

Notes

- If **map** is [] then it will be estimated.
- If **v_est** and **p0** are [] the vehicle is assumed error free and the filter will only estimate the landmark positions (map).
- If **v_est** and **p0** are finite the filter will estimate the vehicle pose and the landmark positions (map).
- EKF subclasses Handle, so it is a reference object.
- Dimensions of workspace are normally taken from the map if given.

See also

Vehicle, Sensor, RangeBearingSensor, Map

EKF.char

Convert to string

E.char() is a string representing the state of the EKF object in human-readable form.

See also

EKF.display

EKF.display

Display status of EKF object

E.**display**() displays the state of the **EKF** object in human-readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a EKF object and the command has no trailing semicolon.

See also

EKF.char

EKF.init

Reset the filter

E.**init**() resets the filter state and clears the history.

EKF.plot_ellipse

Plot vehicle covariance as an ellipse

E.**plot_ellipse**() overlay the current plot with the estimated vehicle position covariance ellipses for 20 points along the path.

E.**plot_ellipse**(**i**) as above but for **i** points along the path.

E.**plot_ellipse**(**i**, **ls**) as above but pass line style arguments **ls** to **plot_ellipse**. If **i** is [] then assume 20.

See also

plot_ellipse

EKF.plot_error

Plot vehicle position

E.**plot_error**(**options**) plot the error between actual and estimated vehicle path (x, y, theta). Heading error is wrapped into the range [-pi,pi)

out = E.**plot_error**() is the estimation error versus time as a matrix ($N \times 3$) where each row is x, y, theta.

Options

'bound', S	Display the S sigma confidence bounds (default 3). If S =0 do not display bounds.
'boundcolor', C	Display the bounds using color C
LS	Use MATLAB linestyle LS for the plots

Notes

- The bounds show the instantaneous standard deviation associated with the state. Observations tend to decrease the uncertainty while periods of dead-reckoning increase it.
- Ideally the error should lie "mostly" within the +/-3sigma bounds.

See also

EKF.plot_xy, EKF.plot_ellipse, EKF.plot_P

EKF.plot_map

Plot landmarks

E.**plot_map**() overlay the current plot with the estimated landmark position (a +-marker) and a covariance ellipses.

E.**plot_map**(**ls**) as above but pass line style arguments **ls** to plot_ellipse.

p = E.**plot_map**() returns the estimated landmark locations ($2 \times N$) and column I is the I'th map feature. If the landmark was not estimated the corresponding column contains NaNs.

See also

plot_ellipse

EKF.plot_P

Plot covariance magnitude

E.**plot_P**() plots the estimated covariance magnitude against time step.

E.**plot_P**(**ls**) as above but the optional line style arguments **ls** are passed to plot.

m = E.**plot_P**() returns the estimated covariance magnitude at all time steps as a vector.

EKF.plot_xy

Plot vehicle position

E.**plot_xy**() overlay the current plot with the estimated vehicle path in the xy-plane.

E.**plot_xy**(**ls**) as above but the optional line style arguments **ls** are passed to plot.

p = E.**plot_xy**() returns the estimated vehicle pose trajectory as a matrix ($N \times 3$) where each row is x, y, theta.

See also

EKF.plot_error, EKF.plot_ellipse, EKF.plot_P

EKF.run

Run the filter

E.**run**(**n**, **options**) runs the filter for **n** time steps and shows an animation of the vehicle moving.

Options

'plot'	Plot an animation of the vehicle moving

Notes

- All previously estimated states and estimation history are initially cleared.

---

eul2jac

Euler angle rate Jacobian

J = **eul2jac**(**eul**) is a Jacobian matrix (3×3) that maps Euler angle rates to angular velocity at the operating point **eul**=[PHI, THETA, PSI].

J = **eul2jac**(**phi**, **theta**, **psi**) as above but the Euler angles are passed as separate arguments.

Notes

- Used in the creation of an analytical Jacobian.

See also

rpy2jac, SERIALlINK.JACOBN

---

eul2r

Convert Euler angles to rotation matrix

R = **eul2r**(**phi**, **theta**, **psi**, **options**) is an orthonornal rotation matrix equivalent to the specified Euler angles. These correspond to rotations about the Z, Y, Z axes respectively. If **phi**, **theta**, **psi** are column vectors then they are assumed to represent a trajectory and **R** is a three dimensional matrix, where the last index corresponds to rows of **phi**, **theta**, **psi**.

R = **eul2r**(**eul**, **options**) as above but the Euler angles are taken from consecutive columns of the passed matrix **eul** = [**phi theta psi**].

Options

'deg'	Compute angles in degrees (radians default)

Note

- The vectors **phi**, **theta**, **psi** must be of the same length.

See also

eul2tr, rpy2tr, tr2eul

---

eul2tr

Convert Euler angles to homogeneous transform

T = **eul2tr**(**phi**, **theta**, **psi**, **options**) is a homogeneous transformation equivalent to the specified Euler angles. These correspond to rotations about the Z, Y, Z axes respectively. If **phi**, **theta**, **psi** are column vectors then they are assumed to represent a trajectory and R is a three dimensional matrix, where the last index corresponds to rows of **phi**, **theta**, **psi**.

T = **eul2tr**(**eul**, **options**) as above but the Euler angles are taken from consecutive columns of the passed matrix **eul** = [**phi theta psi**].

Options

'deg'	Compute angles in degrees (radians default)

Note

- The vectors **phi**, **theta**, **psi** must be of the same length.
- The translational part is zero.

See also

eul2r, rpy2tr, tr2eul

---

gauss2d

Gaussian kernel

out = **gauss2d**(**im**, **sigma**, **C**) is a unit volume Gaussian kernel rendered into matrix **out** ($W \times H$) the same size as **im** ($W \times H$). The Gaussian has a standard deviation of **sigma**. The Gaussian is centered at **C**=[U,V].

h2e

Homogeneous to Euclidean

E = **h2e**(**H**) is the Euclidean version (K-1 $\times N$) of the homogeneous points **H** ($K \times N$) where each column represents one point in P^K.

See also

e2h

homline

Homogeneous line from two points

L = **homline**(**x1**, **y1**, **x2**, **y2**) is a vector (3×1) which describes a line in homogeneous form that contains the two Euclidean points (**x1**,**y1**) and (**x2**,**y2**).

Homogeneous points X (3×1) on the line must satisfy **L**'*X = 0.

See also

plot_homline

homtrans

Apply a homogeneous transformation

p2 = **homtrans**(**T**, **p**) applies homogeneous transformation **T** to the points stored columnwise in **p**.

- If **T** is in SE(2) (3×3) and
  - **p** is $2 \times N$ (2D points) they are considered Euclidean (R^2)
  - **p** is $3 \times N$ (2D points) they are considered projective ($\mathbf{p}^2$)

- If **T** is in SE(3) (4×4) and
  - **p** is $3 \times N$ (3D points) they are considered Euclidean (R^3)
  - **p** is $4 \times N$ (3D points) they are considered projective ($\mathbf{p}^3$)

tp = **homtrans**(**T**, **T1**) applies homogeneous transformation **T** to the homogeneous transformation **T1**, that is **tp=T*T1**. If **T1** is a 3-dimensional transformation then **T** is applied to each plane as defined by the first two

dimensions, ie. if **T** = $N \times N$ and **T**=$N \times N \times$ **p** then the result is $N \times N \times$ **p**.

See also

e2h, h2e

---

ishomog

Test if argument is a homogeneous transformation

ishomog(**T**) is true (1) if the argument **T** is of dimension 4×4 or $4 \times 4 \times N$, else false (0).

ishomog(**T**, 'valid') as above, but also checks the validity of the rotation matrix.

Notes

- The first form is a fast, but incomplete, test for a transform in SE(3)
- Does not work for the SE(2) case

See also

isrot, isvec

---

isrot

Test if argument is a rotation matrix

isrot(**R**) is true (1) if the argument is of dimension 3×3 or $3 \times 3 \times N$, else false (0).

isrot(**R**, 'valid') as above, but also checks the validity of the rotation matrix.

Notes

- A valid rotation matrix has determinant of 1.

See also

ishomog, isvec

---

isvec

Test if argument is a vector

isvec(**v**) is true (1) if the argument **v** is a 3-vector, else false (0).

isvec(**v**, **L**) is true (1) if the argument **v** is a vector of length **L**, either a row- or column-vector. Otherwise false (0).

Notes

- differs from MATLAB builtin function ISVECTOR, the latter returns true for the case of a scalar, **isvec** does not.

See also

ishomog, isrot

---

joy2tr

Update transform from joystick

T = **joy2tr**(**T**, **options**) updates the homogeneous transform according to a connected joystick device.

Options

'delay', D	Pause for D seconds after reading (default 0.1)
'scale', S	A 2-vector which scales joystick translational and rotational to rates (default [0.5m/s, 0.25rad/s])
'world'	Joystick motion is in the world frame
'tool'	Joystick motion is in the tool frame (default)
'rotate', R	Index of the button used to enable rotation (default 7)

Notes

- Joystick axes 0,1,3 map to X,Y,Z or R,P,Y motion.
- A joystick button enables the mapping to translation OR rotation.
- A 'delay' of zero means no pause
- If 'delay' is non-zero 'scale' maps full scale to m/s or rad/s.
- If 'delay' is zero 'scale' maps full scale to m/sample or rad/sample.

---

joystick

Input from joystick

J = **joystick**() returns a vector of **joystick** values in the range -1 to +1.

[**J**,**b**] = **joystick**() as above but also returns a vector of button values, either 0 (not pressed) or 1 (pressed).

Notes

- The length of the vectors **J** and **b** depend on the capabilities of the **joystick** identified when it is first opened.

---

jsingu

Show the linearly dependent joints in a Jacobian matrix

jsingu(**J**) displays the linear dependency of joints in a Jacobian matrix. This dependency indicated joint axes that are aligned and causes singularity.

See also

SerialLink.jacobn

---

jtraj

Compute a joint space trajectory between two points

[**q**,**qd**,**qdd**] = **jtraj**(**q0**, **qf**, **m**) is a joint space trajectory **q** (**m** $\times N$) where the joint coordinates vary from **q0** ($1 \times N$) to **qf** ($1 \times N$). A quintic (5th order) polynomial is used with default zero boundary conditions for velocity and acceleration. Time is assumed to vary from 0 to 1 in **m** steps. Joint velocity and acceleration can be optionally returned as **qd** (**m** $\times N$) and **qdd** (**m** $\times N$) respectively. The trajectory **q**, **qd** and **qdd** are **m** $\times N$ matrices, with one row per time step, and one column per joint.

[**q**,**qd**,**qdd**] = **jtraj**(**q0**, **qf**, **m**, **qd0**, **qdf**) as above but also specifies initial and final joint velocity for the trajectory.

[**q**,**qd**,**qdd**] = **jtraj**(**q0**, **qf**, **T**) as above but the trajectory length is defined by the length of the time vector **T** (**m** $\times$ 1).

[**q**,**qd**,**qdd**] = **jtraj**(**q0**, **qf**, **T**, **qd0**, **qdf**) as above but specifies initial and final joint velocity for the trajectory and a time vector.

See also

ctraj, SerialLink.jtraj

Link

Robot manipulator Link class

A Link object holds all information related to a robot link such as kinematics parameters, rigid-body inertial parameters, motor and transmission parameters.

Methods

A	link transform matrix
RP	joint type: 'R' or 'P'
friction	friction force
nofriction	Link object with friction parameters set to zero
dyn	display link dynamic parameters
islimit	test if joint exceeds soft limit
isrevolute	test if joint is revolute
isprismatic	test if joint is prismatic
display	print the link parameters in human readable form
char	convert to string

Properties (read/write)

theta	kinematic: joint angle
d	kinematic: link offset
a	kinematic: link length
alpha	kinematic: link twist
sigma	kinematic: 0 if revolute, 1 if prismatic
mdh	kinematic: 0 if standard D&H, else 1
offset	kinematic: joint variable offset
qlim	kinematic: joint variable limits [min max]
m	dynamic: link mass
r	dynamic: link COG wrt link coordinate frame 3×1
I	dynamic: link inertia matrix, symmetric 3×3, about link COG.
B	dynamic: link viscous friction (motor referred)
Tc	dynamic: link Coulomb friction

S

G	actuator: gear ratio
Jm	actuator: motor inertia (motor referred)

Notes

- This is a reference class object
- Link objects can be used in vectors and arrays

References

- Robotics, Vision & Control, Chap 7, P. Corke, SprinSger 2011.

See also

Link, Revolute, Prismatic, SerialLink

Link.Link

Create robot link object

This is class constructor function which has several call signatures.

L = **Link**() is a **Link** object with default parameters.

L = **Link**(**l1**) is a **Link** object that is a deep copy of the link object **l1**.

L = **Link**(**options**) is a link object with the kinematic and dynamic parameters specified by the key/value pairs.

Key/value pairs

'theta', TH	joint angle, if not specified joint is revolute
'd', D	joint extension, if not specified joint is prismatic
'a', A	joint offset (default 0)
'alpha', A	joint twist (default 0)
'standard'	defined using standard D&H parameters (default).
'modified'	defined using modified D&H parameters.
'offset', O	joint variable offset (default 0)
'qlim', **L**	joint limit (default [])
'I', I	link inertia matrix (3×1, 6×1 or 3×3)
'r', R	link centre of gravity (3×1)
'm', M	link mass (1×1)
'G', G	motor gear ratio (default 1)
'B', B	joint friction, motor referenced (default 0)
'Jm', J	motor inertia, motor referenced (default 0)
'Tc', T	Coulomb friction, motor referenced (1×1 or 2×1), (default [0 0])
'revolute'	for a revolute joint (default)

'prismatic' for a prismatic joint 'p'

'standard'	for standard D&H parameters (default).
'modified'	for modified D&H parameters.
'sym'	consider all parameter values as symbolic not numeric

- It is an error to specify 'theta' and 'd'
- The link inertia matrix (3×3) is symmetric and can be specified by giving a 3×3 matrix, the diagonal elements [Ixx Iyy Izz], or the moments and products of inertia [Ixx Iyy Izz Ixy Iyz Ixz].
- All friction quantities are referenced to the motor not the load.
- Gear ratio is used only to convert motor referenced quantities such as friction and interia to the link frame.

Old syntax

L = **Link**(**dh**, **options**) is a link object using the specified kinematic convention and with parameters:

- **dh** = [THETA D A ALPHA SIGMA OFFSET] where OFFSET is a constant displacement between the user joint angle vector and the true kinematic solution.
- **dh** = [THETA D A ALPHA SIGMA] where SIGMA=0 for a revolute and 1 for a prismatic joint, OFFSET is zero.
- **dh** = [THETA D A ALPHA], joint is assumed revolute and OFFSET is zero.

Options

'standard'	for standard D&H parameters (default).
'modified'	for modified D&H parameters.
'revolute'	for a revolute joint, can be abbreviated to 'r' (default)

'prismatic' for a prismatic joint, can be abbreviated to 'p'

Examples

A standard Denavit-Hartenberg link

```
L3 = Link('d', 0.15005, 'a', 0.0203, 'alpha', -pi/2);
```

since 'theta' is not specified the joint is assumed to be revolute, and since the kinematic convention is not specified it is assumed 'standard'.

Using the old syntax

```
L3 = Link([ 0, 0.15005, 0.0203, -pi/2], 'standard');
```

the flag 'standard' is not strictly necessary but adds clarity. Only 4 parameters are specified so sigma is assumed to be zero, ie. the joint is revolute.

```
L3 = Link([ 0, 0.15005, 0.0203, -pi/2, 0], 'standard');
```

the flag 'standard' is not strictly necessary but adds clarity. 5 parameters are specified and sigma is set to zero, ie. the joint is revolute.

```
L3 = Link([ 0, 0.15005, 0.0203, -pi/2, 1], 'standard');
```

the flag 'standard' is not strictly necessary but adds clarity. 5 parameters are specified and sigma is set to one, ie. the joint is prismatic.

For a modified Denavit-Hartenberg revolute joint

```
L3 = Link([ 0, 0.15005, 0.0203, -pi/2, 0], 'modified');
```

Notes

- Link object is a reference object, a subclass of Handle object.
- Link objects can be used in vectors and arrays.

- The parameter D is unused in a revolute joint, it is simply a placeholder in the vector and the value given is ignored.
- The parameter THETA is unused in a prismatic joint, it is simply a placeholder in the vector and the value given is ignored.
- The joint offset is a constant added to the joint angle variable before forward kinematics and subtracted after inverse kinematics. It is useful if you want the robot to adopt a 'sensible' pose for zero joint angle configuration.
- The link dynamic (inertial and motor) parameters are all set to zero. These must be set by explicitly assigning the object properties: m, r, I, Jm, B, Tc, G.

See also

: revolute, Prismatic

Link.A

Link transform matrix

T = L.**A**(**q**) is the link homogeneous transformation matrix (4×4) corresponding to the link variable **q** which is either the Denavit-Hartenberg parameter THETA (revolute) or D (prismatic).

Notes

- For a revolute joint the THETA parameter of the link is ignored, and **q** used instead.
- For a prismatic joint the D parameter of the link is ignored, and **q** used instead.
- The link offset parameter is added to **q** before computation of the transformation matrix.

Link.char

Convert to string

s = L.**char**() is a string showing link parameters in a compact single line format. If L is a vector of **Link** objects return a string with one line per **Link**.

See also

Link.display

Link.display

Display parameters

L.**display**() displays the link parameters in compact single line format. If L is a vector of **Link** objects displays one line per element.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Link object and the command has no trailing semicolon.

See also

Link.char, Link.dyn, SerialLink.showlink

Link.dyn

Show inertial properties of link

L.**dyn**() displays the inertial properties of the link object in a multi-line format. The properties shown are mass, centre of mass, inertia, friction, gear ratio and motor properties.

If L is a vector of **Link** objects show properties for each link.

See also

SerialLink.dyn

Link.friction

Joint friction force

f = L.**friction**(**qd**) is the joint **friction** force/torque for link velocity **qd**.

Notes

- **friction** values are referred to the motor, not the load.
- Viscous **friction** is scaled up by G^2.
- Coulomb **friction** is scaled up by G.
- The sign of the gear ratio is used to determine the appropriate Coulomb **friction** value in the non-symmetric case.

Link.islimit

Test joint limits

L.**islimit**(**q**) is true (1) if **q** is outside the soft limits set for this joint.

Note

- The limits are not currently used by any Toolbox functions.

Link.isprismatic

Test if joint is prismatic

L.**isprismatic**() is true (1) if joint is prismatic.

See also

Link.isrevolute

Link.isrevolute

Test if joint is revolute

L.**isrevolute**() is true (1) if joint is revolute.

See also

Link.isprismatic

Link.nofriction

Remove friction

ln = L.**nofriction**() is a link object with the same parameters as L except nonlinear (Coulomb) friction parameter is zero.

ln = L.**nofriction**('all') as above except that viscous and Coulomb friction are set to zero.

ln = L.**nofriction**('coulomb') as above except that Coulomb friction is set to zero.

ln = L.**nofriction**('viscous') as above except that viscous friction is set to zero.

Notes

- Forward dynamic simulation can be very slow with finite Coulomb friction.

See also

SerialLink.nofriction, SerialLink.fdyn

Link.RP

Joint type

c = L.**RP**() is a character 'R' or 'P' depending on whether joint is revolute or prismatic respectively. If L is a vector of **Link** objects return a string of characters in joint order.

Link.set.I

Set link inertia

L.I = [Ixx Iyy Izz] set link inertia to a diagonal matrix.

L.I = [Ixx Iyy Izz Ixy Iyz Ixz] set link inertia to a symmetric matrix with specified inertia and product of inertia elements.

L.I = M set **Link** inertia matrix to M (3×3) which must be symmetric.

Link.set.r

Set centre of gravity

L.r = R set the link centre of gravity (COG) to R (3-vector).

Link.set.Tc

Set Coulomb friction

L.Tc = F set Coulomb friction parameters to [F -F], for a symmetric Coulomb friction model.

L.Tc = [FP FM] set Coulomb friction to [FP FM], for an asymmetric Coulomb friction model. FP>0 and FM<0.

See also

Link.friction

---

lspb

Linear segment with parabolic blend

[**s**,**sd**,**sdd**] = **lspb**(**s0**, **sf**, **m**) is a scalar trajectory (**m** $\times$ 1) that varies smoothly from **s0** to **sf** in **m** steps using a constant velocity segment and parabolic blends (a trapezoidal path). Velocity and acceleration can be optionally returned as **sd** (**m** $\times$ 1) and **sdd** (**m** $\times$ 1).

[**s**,**sd**,**sdd**] = **lspb**(**s0**, **sf**, **m**, **v**) as above but specifies the velocity of the linear segment which is normally computed automatically.

[**s**,**sd**,**sdd**] = **lspb**(**s0**, **sf**, **T**) as above but specifies the trajectory in terms of the length of the time vector **T** (**m** $\times$ 1).

[**s**,**sd**,**sdd**] = **lspb**(**s0**, **sf**, **T**, **v**) as above but specifies the velocity of the linear segment which is normally computed automatically and a time vector.

Notes

- If no output arguments are specified **s**, **sd**, and **sdd** are plotted.

- For some values of **v** no solution is possible and an error is flagged.

See also

tpoly, jtraj

---

makemap

Make an occupancy map

map = **makemap**(**n**) is an occupancy grid **map** (**n** $\times$ **n**) created by a simple interactive editor. The **map** is initially unoccupied and obstacles can be added using geometric primitives.

map = **makemap**() as above but **n**=128.

map = **makemap**(**map0**) as above but the **map** is initialized from the occupancy grid **map0**, allowing obstacles to be added.

With focus in the displayed figure window the following commands can be entered:

left button	click and drag to create a rectangle
p	draw polygon
c	draw circle
u	undo last action
e	erase **map**
q	leave editing mode and return **map**

See also

: dxform, PRM, RRT

Map

Map of planar point features

A Map object represents a square 2D environment with a number of landmark feature points.

Methods

plot	Plot the feature map
feature	Return a specified map feature
display	Display map parameters in human readable form
char	Convert map parameters to human readable string

Properties

map	Matrix of map feature coordinates $2 \times N$
dim	The dimensions of the map region x,y in [-dim,dim]
nfeatures	The number of map features N

Examples

To create a map for an area where X and Y are in the range -10 to +10 metres and with 50 random feature points

```
map = Map(50, 10);
```

which can be displayed by

```
map.plot();
```

Reference

Robotics, Vision & Control, Chap 6, Peter Corke, Springer 2011

See also

RangeBearingSensor, EKF

Map.Map

Map of point feature landmarks

m = **Map**(**n**, **dim**, **options**) is a **Map** object that represents **n** random point features in a planar region bounded by +/-**dim** in the x- and y-directions.

Options

'verbose'	Be verbose

Map.char

Convert vehicle parameters and state to a string

s = M.**char**() is a string showing map parameters in a compact human readable format.

Map.display

Display map parameters

M.**display**() **display** map parameters in a compact human readable form.

Notes

- this method is invoked implicitly at the command line when the result of an expression is a Map object and the command has no trailing semicolon.

See also

map.char

Map.feature

Return the specified map feature

f = M.**feature**(**k**) is the coordinate (2×1) of the **k**'th **feature**.

Map.plot

Plot the map

M.**plot**() plots the feature map in the current figure, as a square region with dimensions given by the M.dim property. Each feature is marked by a black diamond.

M.**plot**(**ls**) plots the feature map as above, but the arguments **ls** are passed to **plot** and override the default marker style.

Notes

- The **plot** is left with HOLD ON.

Map.show

Show the feature map

Notes

- Deprecated, use **plot** method.

Map.verbosity

Set verbosity

M.**verbosity**(**v**) set **verbosity** to **v**, where 0 is silent and greater values display more information.

---

mdl_ball

Create model of a ball manipulator

MDL_BALL creates the workspace variable ball which describes the kinematic characteristics of a serial link manipulator that folds into a ball shape. By default has 50 joints.

mdl_ball(**n**) as above but creates a manipulator with **n** joints.

Also define the workspace vectors:

q joint angle vector for default ball configuration

Reference

- ”A divide and conquer articulated-body algorithm for parallel O(log(n)) calculation of rigid body dynamics, Part 2”, Int. J. Robotics Research, 18(9), pp 876-892.

Notes

- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, mdl_puma560akb, mdl_stanford, mdl_twolink, mdl_coil

---

mdl_coil

Create model of a coil manipulator

MDL_COIL creates the workspace variable coil which describes the kinematic characteristics of a serial link manipulator that folds into a helix shape. By default has 50 joints.

mdl_ball(**n**) as above but creates a manipulator with **n** joints.

Also define the workspace vectors:

q joint angle vector for default helical configuration

Reference

- ”A divide and conquer articulated-body algorithm for parallel O(log(n)) calculation of rigid body dynamics, Part 2”, Int. J. Robotics Research, 18(9), pp 876-892.

Notes

- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, mdl_puma560akb, mdl_stanford, mdl_twolink, mdl_ball

---

mdl_Fanuc10L

Create kinematic model of Fanuc AM120iB/10L robot

```
mdl_fanuc10L
```

Script creates the workspace variable R which describes the kinematic characteristics of a Fanuc AM120iB/10L robot using standard DH conventions.

Also defines the workspace vector:

q0 mastering position.

Author

Wynand Swart, Mega Robots CC, P/O Box 8412, Pretoria, 0001, South Africa wynand.swart@gmail.com

See also

SerialLink, mdl_puma560akb, mdl_stanford, mdl_twolink

---

mdl_hyper2d

Create model of a hyper redundant planar manipulator

MDL_HYPER2D creates the workspace variable h2d which describes the kinematic characteristics of a serial link manipulator which at zero angles is a straight line in the XY plane. By default has 10 joints.

mdl_hyper2d(**n**) as above but creates a manipulator with **n** joints.

Also define the workspace vectors:

qz joint angle vector for zero angle configuration

R = **mdl_hyper2d**(**n**) functional form of the above, returns the SerialLink object.

[**R**,**qz**] = **mdl_hyper2d**(**n**) as above but also returns a vector of zero joint angles.

Notes

- The manipulator in default pose is a straight line 1m long.
- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, mdl_hyper3d, mdl_puma560akb, mdl_stanford, mdl_twolink, mdl_coil

mdl_hyper3d

Create model of a hyper redundant 3D manipulator

MDL_HYPER3D creates the workspace variable h3d which describes the kinematic characteristics of a serial link manipulator which at zero angles is a straight line in the XY plane. By default has 10 joints.

mdl_hyper3d(**n**) as above but creates a manipulator with **n** joints.

Also define the workspace vectors:

qz joint angle vector for zero angle configuration

R = **mdl_hyper3d**(**n**) functional form of the above, returns the SerialLink object.

[**R**,**qz**] = **mdl_hyper3d**(**n**) as above but also returns a vector of zero joint angles.

Notes

- The manipulator in default pose is a straight line 1m long.
- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, mdl_hyper2d, mdl_puma560akb, mdl_stanford, mdl_twolink, mdl_coil

mdl_irb140

Create model of ABB IRB 140 Mico manipulator

```
mdl_irb140
```

Script creates the workspace variable mico which describes the kinematic characteristics of an ABB IRB 140 manipulator using standard DH conventions.

Also define the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY' configuration
qd	lower arm horizontal as per data sheet

Reference

- "IRB 140 data sheet", ABB Robotics.
- "Utilizing the Functional Work Space Evaluation Tool for Assessing a

```
System Design and Reconfiguration Alternatives"
A. Djuric and R. J. Urbanic
```

Notes

- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, Revolute, mdl_puma560, mdl_twolink

mdl_irb140_mdh

Create model of the ABB IRB 140 manipulator

```
mdl_irb140_mod
```

Script creates the workspace variable irb which describes the kinematic characteristics of an ABB IRB 140 manipulator using modified DH conventions.

Also define the workspace vectors:

qz	zero joint angle configuration

Reference

- ABB IRB 140 data sheet
- "THE MODELING OF A SIX DEGREE-OF-FREEDOM INDUSTRIAL ROBOT FOR THE PURPOSE OF EFFICIENT PATH PLANNING" Master of Science Thesis, Penn State U, May 2009 Tyler Carter

See also

SerialLink, mdl_irb140, mdl_puma560, mdl_stanford, mdl_twolink

Notes

- SI units of metres are used.
- The tool frame is in the centre of the tool flange.
- Zero angle configuration has the upper arm vertical and lower arm horizontal.

---

mdl_jaco

Create model of Kinova Jaco manipulator

```
mdl_jaco
```

Script creates the workspace variable jaco which describes the kinematic characteristics of a Kinova Jaco manipulator using standard DH conventions.

Also define the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY' configuration

Reference

- "DH Parameters of Jaco" Version 1.0.8, July 25, 2013.

Notes

- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, Revolute, mdl_mico, mdl_puma560, mdl_twolink

---

mdl_m16

Create model of Fanuc M16 Mico manipulator

```
mdl_m16
```

Script creates the workspace variable mico which describes the kinematic characteristics of a Fanuc M16 manipulator using standard DH conventions.

Also define the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY' configuration
qd	lower arm horizontal as per data sheet

Reference

- "Fanuc M-16iB data sheet", http://www.robots.com/fanuc/m-16ib.
- "Utilizing the Functional Work Space Evaluation Tool for Assessing a

```
System Design and Reconfiguration Alternatives"
A. Djuric and R. J. Urbanic
```

Notes

- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, Revolute, mdl_irb140, mdl_puma560, mdl_twolink

mdl_mico

Create model of Kinova Mico manipulator

```
mdl_mico
```

Script creates the workspace variable mico which describes the kinematic characteristics of a Kinova Mico manipulator using standard DH conventions.

Also define the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY' configuration

Reference

- "DH Parameters of Mico" Version 1.0.1, August 05, 2013. Kinova

Notes

- Unlike most other mdl_xxx scripts this one is actually a function that behaves like a script and writes to the global workspace.

See also

SerialLink, Revolute, mdl_jaco, mdl_puma560, mdl_twolink

---

mdl_MotomanHP6

Create kinematic data of a Motoman HP6 manipulator

```
mdl_motomanHP6
```

Script creates the workspace variable R which describes the kinematic characteristics of a Motoman HP6 manipulator using standard DH conventions.

Also defines the workspace vector:

q0	mastering position.

Author:

Wynand Swart, Mega Robots CC, P/O Box 8412, Pretoria, 0001, South Africa wynand.swart@gmail.com

See also

SerialLink, mdl_puma560akb, mdl_stanford, mdl_twolink

---

mdl_nao

Create model of Aldebaran NAO humanoid robot

```
mdl_nao
```

Script creates several workspace variables

leftarm	left-arm kinematics (4DOF)
rightarm	right-arm kinematics (4DOF)
leftleg	left-leg kinematics (6DOF)
rightleg	right-leg kinematics (6DOF)

which describes the kinematic characteristics of the arms and legs of the NAO humanoid.

Reference

- "Forward and Inverse Kinematics for the NAO Humanoid Robot", Nikolaos Kofinas, Thesis, Technical University of Crete July 2012.
- "Mechatronic design of NAO humanoid" David Gouaillier etal. IROS 2009, pp. 769-774.

Notes

- the base transform of arms and legs are constant with respect to the torso frame, which is assumed to be the constant value when the robot is upright. Clearly if the robot is walking these base transforms will be dynamic.
- the first reference uses Modified DH notation, but doesn't explicitly mention this, and the parameter tables have the wrong column headings for Modified DH parameters.
- TODO; add joint limits
- TODO; add dynamic parameters

See also

SerialLink, Revolute

---

mdl_onelink

Create model of a simple 2-link mechanism

```
mdl_twolink
```

Script creates the workspace variable tl which describes the kinematic and dynamic characteristics of a simple planar 2-link mechanism.

Also defines the vector:

qz corresponds to the zero joint angle configuration.

Notes

- It is a planar mechanism operating in the XY (horizontal) plane and is therefore not affected by gravity.
- Assume unit length links with all mass (unity) concentrated at the joints.

References

- Based on Fig 3-6 (p73) of Spong and Vidyasagar (1st edition).

See also

SerialLink, mdl_puma560, mdl_stanford

---

mdl_phantomx

Create model of PhantomX pincher manipulator

```
mdl_phantomx
```

Script creates the workspace variable px which describes the kinematic characteristics of a PhantomX Pincher Robot, a 4 joint hobby class manipulator by Trossen Robotics.

Also define the workspace vectors:

qz zero joint angle configuration

Notes

- uses standard DH conventions.
- Tool centrepoint is middle of the fingertips
- all translational units in mm

Reference

- http://www.trossenrobotics.com/productdocs/assemblyguides/phantomx-basic-robot-arm.html

mdl_planar1

Create model of a simple planar 1-link mechanism

```
mdl_planar1
```

Script creates the workspace variable p1 which describes the kinematic and dynamic characteristics of a simple planar 1-link mechanism.

Also defines the vector:

qz corresponds to the zero joint angle configuration.

Notes

- It is a planar mechanism operating in the XY (horizontal) plane and is therefore not affected by gravity.
- No dynamics in this model

See also

SerialLink, mdl_twolink

mdl_planar2

Create model of a simple planar 2-link mechanism

```
mdl_planar2
```

Script creates the workspace variable p2 which describes the kinematic and dynamic characteristics of a simple planar 2-link mechanism.

Also defines the vector:

qz corresponds to the zero joint angle configuration.

Notes

- No dynamics in this model

See also

SerialLink, mdl_twolink

---

mdl_planar3

Create model of a simple planar 3-link mechanism

```
mdl_planar3
```

Script creates the workspace variable p3 which describes the kinematic and dynamic characteristics of a simple planar 3-link mechanism.

Also defines the vector:

qz	corresponds to the zero joint angle configuration.

Notes

- No dynamics in this model

See also

SerialLink, mdl_twolink

---

mdl_puma560

Create model of Puma 560 manipulator

```
mdl_puma560
```

Script creates the workspace variable p560 which describes the kinematic and dynamic characteristics of a Unimation Puma 560 manipulator using standard DH conventions. The model includes armature inertia and gear ratios.

Also define the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY' configuration
qstretch	arm is stretched out in the X direction
qn	arm is at a nominal non-singular configuration

Reference

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

See also

serialrevolute, mdl_puma560akb, mdl_stanford, mdl_twolink

mdl_puma560akb

Create model of Puma 560 manipulator

```
mdl_puma560akb
```

Script creates the workspace variable p560m which describes the kinematic and dynamic characterstics of a Unimation Puma 560 manipulator modified DH conventions.

Also defines the workspace vectors:

qz	zero joint angle configuration
qr	vertical 'READY' configuration
qstretch	arm is stretched out in the X direction

References

- "The Explicit Dynamic Model and Inertial Parameters of the Puma 560 Arm" Armstrong, Khatib and Burdick 1986

See also

SerialLink, mdl_puma560, mdl_stanford, mdl_twolink

mdl_quadcopter

Dynamic parameters for a quadcopter.

```
mdl_quadcopter
```

Script creates the workspace variable quad which describes the dynamic characterstics of a quadcopter.

Properties

This is a structure with the following elements:

J	Flyer rotational inertia matrix (3×3)
h	Height of rotors above CoG (1×1)
d	Length of flyer arms (1×1)
nb	Number of blades per rotor (1×1)
r	Rotor radius (1×1)
c	Blade chord (1×1)
e	Flapping hinge offset (1×1)
Mb	Rotor blade mass (1×1)
Mc	Estimated hub clamp mass (1×1)
ec	Blade root clamp displacement (1×1)
Ib	Rotor blade rotational inertia (1×1)
Ic	Estimated root clamp inertia (1×1)
mb	Static blade moment (1×1)
Ir	Total rotor inertia (1×1)
Ct	Non-dim. thrust coefficient (1×1)
Cq	Non-dim. torque coefficient (1×1)
sigma	Rotor solidity ratio (1×1)
thetat	Blade tip angle (1×1)
theta0	Blade root angle (1×1)
theta1	Blade twist angle (1×1)
theta75	3/4 blade angle (1×1)
thetai	Blade ideal root approximation (1×1)
a	Lift slope gradient (1×1)
A	Rotor disc area (1×1)
gamma	Lock number (1×1)

References

- Design, Construction and Control of a Large Quadrotor micro air vehicle. P.Pounds, PhD thesis, Australian National University, 2007. http://www.eng.yale.edu/pep5/P_Pounds_Thesis_2008.pdf

See also

sl_quadcopter

mdl_S4ABB2p8

Create kinematic model of ABB S4 2.8robot

```
mdl_s4abb2P8
```

Script creates the workspace variable R which describes the kinematic characteristics of an ABB S4 2.8 robot using standard DH conventions.

Also defines the workspace vector:

q0 mastering position.

Author

Wynand Swart, Mega Robots CC, P/O Box 8412, Pretoria, 0001, South Africa wynand.swart@gmail.com

See also

SerialLink, mdl_puma560akb, mdl_stanford, mdl_twolink

mdl_stanford

Create model of Stanford arm

```
mdl_stanford
```

Script creates the workspace variable stanf which describes the kinematic and dynamic characteristics of the Stanford (Scheinman) arm.

Also defines the vectors:

qz zero joint angle configuration.

Note

- Gear ratios not currently known, though reflected armature inertia is known, so gear ratios are set to 1.

References

- Kinematic data from ”Modelling, Trajectory calculation and Servoing of a computer controlled arm”. Stanford AIM-177. Figure 2.3
- Dynamic data from “Robot manipulators: mathematics, programming and control” Paul 1981, Tables 6.5, 6.6

See also

SerialLink, mdl_puma560, mdl_puma560akb, mdl_twolink

---

mdl_stanford_mdh

Create model of Stanford arm using MDH conventions

```
mdl_stanford_mdh
```

Script creates the workspace variable stanf which describes the kinematic and dynamic characteristics of the Stanford (Scheinman) arm using modified Denavit-Hartenberg parameters.

Also defines the vectors:

qz   zero joint angle configuration.

References

- Kinematic data from ”Modelling, Trajectory calculation and Servoing of a computer controlled arm”. Stanford AIM-177. Figure 2.3
- Dynamic data from “Robot manipulators: mathematics, programming and control” Paul 1981, Tables 6.5, 6.6

See also

SerialLink, mdl_puma560, mdl_puma560akb, mdl_twolink

---

mdl_twolink

Create model of a simple 2-link mechanism

```
mdl_twolink
```

Script creates the workspace variable tl which describes the kinematic and dynamic characteristics of a simple planar 2-link mechanism.

Also defines the vector:

qz corresponds to the zero joint angle configuration.

Notes

- It is a planar mechanism operating in the XY (horizontal) plane and is therefore not affected by gravity.
- Assume unit length links with all mass (unity) concentrated at the joints.

References

- Based on Fig 3-6 (p73) of Spong and Vidyasagar (1st edition).

See also

SerialLink, mdl_puma560, mdl_stanford

---

mdl_twolink_mdh

Create model of a simple 2-link mechanism with modified DH convention

```
mdl_twolink_mdh
```

Script creates the workspace variable tl which describes the kinematic and dynamic characteristics of a simple planar 2-link mechanism using modified Denavit-Hartenberg conventions.

Also defines the vector:

qz corresponds to the zero joint angle configuration.

Notes

- It is a planar mechanism operating in the XY (horizontal) plane

References

- Based on Fig 3.8 (p71) of Craig (3rd edition).

See also

SerialLink, mdl_twolink

---

mstraj

Multi-segment multi-axis trajectory

traj = mstraj(**p**, **qdmax**, **q0**, **dt**, **tacc**, **options**) is a multi-segment trajectory ($K \times N$) based on via points **p** ($M \times N$) and axis velocity limits **qdmax** ($1 \times N$). The path comprises linear segments with polynomial blends. The output trajectory matrix has one row per time step, and one column per axis.

- **p** ($M \times N$) is a matrix of via points, 1 row per via point, one column per axis. The last via point is the destination.
- **qdmax** ($1 \times N$) are axis velocity limits which cannot be exceeded, or
- **qdmax** ($M \times 1$) are the durations for each of the M segments
- **q0** ($1 \times N$) are the initial axis coordinates
- **dt** is the time step
- **tacc** (1×1) this acceleration time is applied to all segment transitions
- **tacc** ($1 \times M$) acceleration time for each segment, **tacc**(i) is the acceleration time for the transition from segment i to segment i+1. **tacc**(1) is also the acceleration time at the start of segment 1.

traj = mstraj(**segments**, **qdmax**, **q0**, **dt**, **tacc**, **qd0**, **qdf**, **options**) as above but additionally specifies the initial and final axis velocities ($1 \times N$).

Options

'verbose'	Show details.

Notes

- If no output arguments are specified the trajectory is plotted.
- The path length K is a function of the number of via points, **q0**, **dt** and **tacc**.
- The final via point **p**(M,:) is the destination.
- The motion has M segments from **q0** to **p**(1,:) to **p**(2,:) to **p**(M,:).
- All axes reach their via points at the same time.
- Can be used to create joint space trajectories where each axis is a joint coordinate.
- Can be used to create Cartesian trajectories with the "axes" assigned to translation and orientation in RPY or Euler angle form.

See also

mstraj, lspb, ctraj

---

mtraj

Multi-axis trajectory between two points

[**q**,**qd**,**qdd**] = **mtraj**(**tfunc**, **q0**, **qf**, **m**) is a multi-axis trajectory (**m** $\times N$) varying from state **q0** ($1 \times N$) to **qf** ($1 \times N$) according to the scalar trajectory function **tfunc** in **m** steps. Joint velocity and acceleration can be optionally returned as **qd** (**m** $\times N$) and **qdd** (**m** $\times N$) respectively. The trajectory outputs have one row per time step, and one column per axis.

The shape of the trajectory is given by the scalar trajectory function **tfunc**

```
[S,SD,SDD] = TFUNC(S0, SF, M);
```

and possible values of **tfunc** include @lspb for a trapezoidal trajectory, or @tpoly for a polynomial trajectory.

[**q**,**qd**,**qdd**] = **mtraj**(**tfunc**, **q0**, **qf**, **T**) as above but specifies the trajectory length in terms of the length of the time vector **T** (**m** $\times 1$).

Notes

- If no output arguments are specified **q**, **qd**, and **qdd** are plotted.
- When **tfunc** is @tpoly the result is functionally equivalent to JTRAJ except that no initial velocities can be specified. JTRAJ is computationally a little more efficient.

See also

jtraj, mstraj, lspb, tpoly

---

multidfprintf

Print formatted text to multiple streams

COUNT = MULTIDFPRINTF(IDVEC, FORMAT, A, ...) performs formatted output to multiple streams such as console and files. FORMAT is the format string as used by sprintf and fprintf. A is the array of elements, to which the format will be applied similar to sprintf and fprint.

IDVEC is a vector ($1 \times N$) of file descriptors and COUNT is a vector ($1 \times N$) of the number of bytes written to each file.

Notes

- To write to the consolde use the file identifier 1.

Example

```
% Create and open a new example file:
fid = fopen('exampleFile.txt','w+');
% Write something to the file and the console simultaneously:
multidfprintf([1 FID],'% s % d % d % d% Close the file:
fclose(FID);
```

Authors

Joern Malzahn 2012 RST, Technische Universitaet Dortmund, Germany. http://www.rst.e-technik.tu-dortmund.de

See also

fprintf, sprintf

Navigation

Navigation superclass

An abstract superclass for implementing navigation classes.

Methods

plot	Display the occupancy grid
visualize	Display the occupancy grid (deprecated)
plan	Plan a path to goal
path	Return/animate a path from start to goal
display	Display the parameters in human readable form
char	Convert to string

rand	Uniformly distributed random number
randn	Normally distributed random number
randi	Uniformly distributed random integer

Properties (read only)

occgrid	Occupancy grid representing the navigation environment
goal	Goal coordinate
seed0	Random number state

Methods that must be provided in subclass

plan	Generate a plan for motion to goal
next	Returns coordinate of next point along path

Methods that may be overriden in a subclass

goal_set	The goal has been changed by nav.goal = (a,b)
navigate_init	Start of path planning.

Notes

- Subclasses the MATLAB handle class which means that pass by reference semantics apply.
- A grid world is assumed and vehicle position is quantized to grid cells.
- Vehicle orientation is not considered.

- The initial random number state is captured as seed0 to allow rerunning an experiment with an interesting outcome.

See also

Dstar, dxform, PRM, RRT

Navigation.Navigation

Create a Navigation object

n = Navigation(**occgrid**, **options**) is a Navigation object that holds an occupancy grid **occgrid**. A number of options can be be passed.

Options

'navhook', F	Specify a function to be called at every step of path
'goal', G	Specify the goal point (2×1)
'verbose'	Display debugging information
'inflate', K	Inflate all obstacles by K cells.
'private'	Use private random number stream.
'reset'	Reset random number stream.
'seed', S	Set the initial state of the random number stream. S must be a proper random number generator state such as saved in the seed0 property of an earlier run.

Notes

- In the occupancy grid a value of zero means free space and non-zero means occupied (not driveable).
- Obstacle inflation is performed with a round structuring element (kcircle).
- The 'private' option creates a private random number stream for the methods rand, randn and randi. If not given the global stream is used.

Navigation.char

Convert to string

N.char() is a string representing the state of the navigation object in human-readable form.

Navigation.display

Display status of navigation object

N.**display**() displays the state of the navigation object in human-readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Navigation object and the command has no trailing semicolon.

See also

Navigation.char

Navigation.goal_change

Notify change of goal

Invoked when the goal property of the object is changed. Typically this is overriden in a subclass to take particular action such as invalidating a costmap.

Navigation.message

display debug message

N.**message**(**s**) displays the string **s** if the verbose property is true.

N.**message**(**fmt**, **args**) as above but accepts printf() like semantics.

Navigation.navigate_init

Notify start of path

Invoked when the path() method is invoked. Typically overriden in a subclass to take particular action such as computing some path parameters. start is the initial position for this path, and nav.goal is the final position.

Navigation.path

Follow path from start to goal

N.**path**(**start**) animates the robot moving from **start** (2×1) to the goal (which is a property of the object).

N.**path**() as above but first displays the occupancy grid, and prompts the user to click a start location. the object).

x = N.**path**(**start**) returns the **path** ($2 \times M$) from **start** to the goal (which is a property of the object).

The method performs the following steps:

- Get start position interactively if not given
- Initialized navigation, invoke method N.navigate_init()
- Visualize the environment, invoke method N.plot()
- Iterate on the next() method of the subclass

See also

Navigation.plot, Navigation.goal

Navigation.plot

Visualize navigation environment

N.**plot**() displays the occupancy grid in a new figure.

N.**plot**(**p**) as above but overlays the points along the path ($M \times 2$) matrix.

Options

'goal'	Superimpose the goal position if set
'distance', D	Display a distance field D behind the obstacle map. D is a matrix of the same size as the occupancy grid.

Navigation.rand

Uniformly distributed random number

R = N.**rand**() return a uniformly distributed random number from a private random number stream.

R = N.**rand**(**m**) as above but return a matrix (**m** × **m**) of random numbers.

R = N.**rand**(**L**,**m**) as above but return a matrix (**L** × **m**) of random numbers.

Notes

- Accepts the same arguments as **rand**().
- Seed is provided to Navigation constructor.

See also

rand, randstream

Navigation.randi

Integer random number

i = N.**randi**(**rm**) return a uniformly distributed random integer in the range 1 to **rm** from a private random number stream.

i = N.**randi**(**rm**, **m**) as above but return a matrix (**m** × **m**) of random integers.

i = N.**randn**(**rm**, **L**,**m**) as above but return a matrix (**L** × **m**) of random integers.

Notes

- Accepts the same arguments as **randn**().
- Seed is provided to Navigation constructor.

See also

randn, randstream

Navigation.randn

Normally distributed random number

R = N.**randn**() return a normally distributed random number from a private random number stream.

R = N.**randn**(**m**) as above but return a matrix (**m** × **m**) of random numbers.

R = N.**randn**(**L**,**m**) as above but return a matrix (**L** × **m**) of random numbers.

Notes

- Accepts the same arguments as **randn**().
- Seed is provided to Navigation constructor.

See also

randn, randstream

Navigation.spinner

Update progress spinner

N.**spinner**() displays a simple ASCII progress **spinner**, a rotating bar.

Navigation.verbosity

Set verbosity

N.**verbosity**(**v**) set **verbosity** to **v**, where 0 is silent and greater values display more information.

---

numcols

Return number of columns in matrix

nc = **numcols**(**m**) is the number of columns in the matrix **m**.

See also

numrows

---

numrows

Return number of rows in matrix

nr = **numrows**(**m**) is the number of rows in the matrix **m**.

See also

numcols

---

oa2r

Convert orientation and approach vectors to rotation matrix

R = **oa2r**(**o**, **a**) is a rotation matrix for the specified orientation and approach vectors (3×1) formed from 3 vectors such that **R** = [N **o** **a**] and N = **o** x **a**.

Notes

- The submatrix is guaranteed to be orthonormal so long as **o** and **a** are not parallel.
- The vectors **o** and **a** are parallel to the Y- and Z-axes of the coordinate frame.

See also

rpy2r, eul2r, oa2tr

---

oa2tr

Convert orientation and approach vectors to homogeneous transformation

T = oa2tr(**o**, **a**) is a homogeneous tranformation for the specified orientation and approach vectors (3×1) formed from 3 vectors such that R = [N **o** **a**] and N = **o** x **a**.

Notes

- The rotation submatrix is guaranteed to be orthonormal so long as **o** and **a** are not parallel.
- The translational part is zero.
- The vectors **o** and **a** are parallel to the Y- and Z-axes of the coordinate frame.

See also

rpy2tr, eul2tr, oa2r

ParticleFilter

Particle filter class

Monte-carlo based localisation for estimating vehicle pose based on odometry and observations of known landmarks.

Methods

run	run the particle filter
plot_xy	display estimated vehicle path
plot_pdf	display particle distribution

Properties

robot	reference to the robot object
sensor	reference to the sensor object
history	vector of structs that hold the detailed information from each time step
nparticles	number of particles used
x	particle states; nparticles x 3
weight	particle weights; nparticles x 1
x_est	mean of the particle population
std	standard deviation of the particle population
Q	covariance of noise added to state at each step
L	covariance of likelihood model
w0	offset in likelihood model
dim	maximum xy dimension

Example

Create a landmark map

```
map = Map(20);
```

and a vehicle with odometry covariance and a driver

```
W = diag([0.1, 1*pi/180].^2);
veh = Vehicle(W);
veh.add_driver( RandomPath(10) );
```

and create a range bearing sensor

```
R = diag([0.005, 0.5*pi/180].^2);
sensor = RangeBearingSensor(veh, map, R);
```

For the particle filter we need to define two covariance matrices. The first is is the covariance of the random noise added to the particle states at each iteration to represent uncertainty in configuration.

```
Q = diag([0.1, 0.1, 1*pi/180]).^2;
```

and the covariance of the likelihood function applied to innovation

```
L = diag([0.1 0.1]);
```

Now construct the particle filter

```
pf = ParticleFilter(veh, sensor, Q, L, 1000);
```

which is configured with 1000 particles. The particles are initially uniformly distributed over the 3-dimensional configuration space.

We run the simulation for 1000 time steps

```
pf.run(1000);
```

then plot the map and the true vehicle path

```
map.plot();
veh.plot_xy('b');
```

and overlay the mean of the particle cloud

```
pf.plot_xy('r');
```

We can plot the standard deviation against time

```
plot(pf.std(1:100,:))
```

The particles are a sampled approximation to the PDF and we can display this as

```
pf.plot_pdf()
```

Acknowledgement

Based on code by Paul Newman, Oxford University, http://www.robots.ox.ac.uk/ pnewman

Reference

Robotics, Vision & Control, Peter Corke, Springer 2011

See also

Vehicle, RandomPath, RangeBearingSensor, Map, EKF

ParticleFilter.ParticleFilter

Particle filter constructor

pf = ParticleFilter(**vehicle**, **sensor**, **q**, **L**, **np**, **options**) is a particle filter that estimates the state of the **vehicle** with a sensor **sensor**. **q** is covariance of the noise added to the particles at each step (diffusion), **L** is the covariance used in the sensor likelihood model, and **np** is the number of particles.

Options

'verbose'	Be verbose.
'private'	Use private random number stream.
'reset'	Reset random number stream.
'seed', S	Set the initial state of the random number stream. S must be a proper random number generator state such as saved in the seed0 property of an earlier run.
'nohistory'	Don't save history.

Notes

- ParticleFilter subclasses Handle, so it is a reference object.
- The initial particle distribution is uniform over the map, essentially the kidnapped robot problem which is quite unrealistic.
- The 'private' option creates a private random number stream for the methods rand, randn and randi. If not given the global stream is used.

See also

Vehicle, Sensor, RangeBearingSensor, Map

ParticleFilter.char

Convert to string

PF.**char**() is a string representing the state of the **ParticleFilter** object in human-readable form.

See also

ParticleFilter.display

ParticleFilter.display

Display status of particle filter object

PF.**display**() displays the state of the **ParticleFilter** object in human-readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a ParticleFilter object and the command has no trailing semicolon.

See also

ParticleFilter.char

ParticleFilter.init

Initialize the particle filter

PF.**init**() initializes the particle distribution and clears the history.

Notes

- Invoked by the run() method.

ParticleFilter.plot_pdf

Plot particles as a PDF

PF.**plot_pdf**() plots a sparse PDF as a series of vertical line segments of height equal to particle weight.

ParticleFilter.plot_xy

Plot vehicle position

PF.**plot_xy**() plots the estimated vehicle path in the xy-plane.

PF.**plot_xy**(**ls**) as above but the optional line style arguments **ls** are passed to plot.

ParticleFilter.run

Run the particle filter

PF.**run**(**n**, **options**) runs the filter for **n** time steps.

Options

'noplot'	Do not show animation.

Notes

- All previously estimated states and estimation history is cleared.

peak

Find peaks in vector

yp = **peak**(**y**, **options**) are the values of the maxima in the vector **y**.

[**yp**,**i**] = **peak**(**y**, **options**) as above but also returns the indices of the maxima in the vector **y**.

[**yp**,**xp**] = **peak**(**y**, **x**, **options**) as above but also returns the corresponding x-coordinates of the maxima in the vector **y**. **x** is the same length of **y** and contains the corresponding x-coordinates.

Options

'npeaks', N	Number of peaks to return (default all)
'scale', S	Only consider as peaks the largest value in the horizontal range +/- S points.
'interp', N	Order of interpolation polynomial (default no interpolation)
'plot'	Display the interpolation polynomial overlaid on the point data

Notes

- To find minima, use **peak**(-V).
- The interp options fits points in the neighbourhood about the **peak** with an N'th order polynomial and its **peak** position is returned. Typically choose N to be odd.

See also

peak2

peak2

Find peaks in a matrix

zp = **peak2**(**z**, **options**) are the peak values in the 2-dimensional signal **z**.

[**zp**,**ij**] = **peak2**(**z**, **options**) as above but also returns the indices of the maxima in the matrix **z**. Use SUB2IND to convert these to row and column coordinates

Options

'npeaks', N	Number of peaks to return (default all)
'scale', S	Only consider as peaks the largest value in the horizontal and vertical range +/- S points.
'interp'	Interpolate peak (default no interpolation)
'plot'	Display the interpolation polynomial overlaid on the point data

Notes

- To find minima, use **peak2**(-V).
- The interp options fits points in the neighbourhood about the peak with a paraboloid and its peak position is returned.

See also

peak, sub2ind

PGraph

Graph class

g = PGraph()	create a 2D, planar, undirected graph
g = PGraph(n)	create an n-d, undirected graph

Provides support for graphs that:

- are directed
- are embedded in coordinate system
- have symmetric cost edges (A to B is same cost as B to A)
- have no loops (edges from A to A)

- have vertices are represented by integers vid
- have edges are represented by integers, eid

Methods

Constructing the graph

g.add_node(coord)	add vertex, return vid
g.add_edge(v1, v2)	add edge from v1 to v2, return eid
g.setcost(e, c)	set cost for edge e
g.setdata(v, u)	set user data for vertex v
g.data(v)	get user data for vertex v
g.clear()	remove all vertices and edges from the graph

Information from graph

g.edges(v)	list of edges for vertex v
g.cost(e)	cost of edge e
g.neighbours(v)	neighbours of vertex v
g.component(v)	component id for vertex v
g.connectivity()	number of edges for all vertices

Display

g.plot()	set goal vertex for path planning
g.highlight_node(v)	highlight vertex v
g.highlight_edge(e)	highlight edge e
g.highlight_component(c)	highlight all nodes in component c
g.highlight_path(p)	highlight nodes and edge along path p

g.pick(coord)	vertex closest to coord

g.char()	convert graph to string
g.display()	display summary of graph

Matrix representations

g.adjacency()	adjacency matrix
g.incidence()	incidence matrix
g.degree()	degree matrix
g.laplacian()	Laplacian matrix

Planning paths through the graph

g.Astar(s, g)	shortest path from s to g
g.goal(v)	set goal vertex, and plan paths
g.path(v)	list of vertices from v to goal

Graph and world points

g.coord(v)	coordinate of vertex v
g.distance(v1, v2)	distance between v1 and v2
g.distances(coord)	return sorted distances from coord to all vertices
g.closest(coord)	vertex closest to coord

Object properties (read only)

g.n	number of vertices
g.ne	number of edges
g.nc	number of components

Notes

- Graph connectivity is maintained by a labeling algorithm and this is updated every time an edge is added.
- Nodes and edges cannot be deleted.
- Support for edge direction is rudimentary.

PGraph.PGraph

Graph class constructor

g=PGraph(**d**, **options**) is a graph object embedded in **d** dimensions.

Options

'distance', M	Use the distance metric M for path planning which is either 'Euclidean' (default) or 'SE2'.
'verbose'	Specify verbose operation

Note

- Number of dimensions is not limited to 2 or 3.
- The distance metric 'SE2' is the sum of the squares of the difference in position and angle modulo 2pi.
- To use a different distance metric create a subclass of PGraph and override the method distance_metric().

PGraph.add_edge

Add an edge

E = G.**add_edge**(**v1**, **v2**) adds a directed edge from vertex id **v1** to vertex id **v2**, and returns the edge id **E**. The edge cost is the distance between the vertices.

E = G.**add_edge**(**v1**, **v2**, **C**) as above but the edge cost is **C**. cost **C**.

Note

- Graph connectivity is maintained by a labeling algorithm and this is updated every time an edge is added.

See also

PGraph.add_node, PGraph.edgedir

PGraph.add_node

Add a node

v = G.**add_node**(**x**) adds a node/vertex with coordinate **x** ($D \times 1$) and returns the integer node id **v**.

v = G.**add_node**(**x**, **v2**) as above but connected by a directed edge from vertex **v** to vertex **v2** with cost equal to the distance between the vertices.

v = G.**add_node**(**x**, **v2**, **C**) as above but the added edge has cost **C**.

See also

PGraph.add_edge, PGraph.data, PGraph.getdata

PGraph.adjacency

Adjacency matrix of graph

a = G.**adjacency**() is a matrix ($N \times N$) where element **a**(i,j) is the cost of moving from vertex i to vertex j.

Notes

- Matrix is symmetric.
- Eigenvalues of **a** are real and are known as the spectrum of the graph.
- The element **a**(I,J) can be considered the number of walks of one edge from vertex I to vertex J (either zero or one). The element (I,J) of $\mathbf{a}^N$ are the number of walks of length N from vertex I to vertex J.

See also

PGraph.degree, PGraph.incidence, PGraph.laplacian

PGraph.Astar

path finding

path = G.**Astar**(**v1**, **v2**) is the lowest cost path from vertex **v1** to vertex **v2**. **path** is a list of vertices starting with **v1** and ending **v2**.

[**path**,C] = G.**Astar**(**v1**, **v2**) as above but also returns the total cost of traversing **path**.

Notes

- Uses the efficient A* search algorithm.

References

- Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". Hart, P. E.; Nilsson, N. J.; Raphael, B. SIGART Newsletter 37: 28-29, 1972.

See also

PGraph.goal, PGraph.path

PGraph.char

Convert graph to string

s = G.**char**() is a compact human readable representation of the state of the graph including the number of vertices, edges and components.

PGraph.clear

Clear the graph

G.**clear**() removes all vertices, edges and components.

PGraph.closest

Find closest vertex

v = G.**closest**(**x**) is the vertex geometrically **closest** to coordinate **x**.

[**v**,**d**] = G.**closest**(**x**) as above but also returns the distance **d**.

See also

PGraph.distances

PGraph.component

Graph component

C = G.**component**(**v**) is the id of the graph **component**

PGraph.connectivity

Graph connectivity

C = G.**connectivity**() is a vector ($N \times 1$) with the number of edges per vertex.

The average vertex **connectivity** is

```
mean(g.connectivity())
```

and the minimum vertex **connectivity** is

```
min(g.connectivity())
```

PGraph.coord

Coordinate of node

x = G.**coord**(**v**) is the coordinate vector ($D \times 1$) of vertex id **v**.

PGraph.cost

Cost of edge

C = G.**cost**(**E**) is the **cost** of edge id **E**.

PGraph.data

Get user data for node

u = G.**data**(**v**) gets the user **data** of vertex **v** which can be of any type such as number, struct, object or cell array.

See also

PGraph.setdata

PGraph.degree

Degree matrix of graph

d = G.**degree**() is a diagonal matrix ($N \times N$) where element **d**(i,i) is the number of edges connected to vertex id i.

See also

PGraph.adjacency, PGraph.incidence, PGraph.laplacian

PGraph.display

Display graph

G.**display**() displays a compact human readable representation of the state of the graph including the number of vertices, edges and components.

See also

PGraph.char

PGraph.distance

Distance between vertices

d = G.**distance**(**v1**, **v2**) is the geometric **distance** between the vertices **v1** and **v2**.

See also

PGraph.distances

PGraph.distances

Distances from point to vertices

d = G.**distances**(**x**) is a vector ($1 \times N$) of geometric distance from the point **x** (**d** $\times$ 1) to every other vertex sorted into increasing order.

[**d**,**w**] = G.**distances**(**p**) as above but also returns **w** ($1 \times N$) with the corresponding vertex id.

See also

PGraph.closest

PGraph.edgedir

Find edge direction

d = G.**edgedir**(**v1**, **v2**) is the direction of the edge from vertex id **v1** to vertex id **v2**.

If we add an edge from vertex 3 to vertex 4

```
g.add_edge(3, 4)
```

then

```
g.edgedir(3, 4)
```

is positive, and

```
g.edgedir(4, 3)
```

is negative.

See also

PGraph.add_node, PGraph.add_edge

PGraph.edges

Find edges given vertex

E = G.**edges**(**v**) is a vector containing the id of all **edges** from vertex id **v**.

See also

PGraph.edgedir

PGraph.get.n

Number of vertices

G.n is the number of vertices in the graph.

See also

PGraph.ne

PGraph.get.nc

Number of components

G.nc is the number of components in the graph.

See also

PGraph.component

PGraph.get.ne

Number of edges

G.ne is the number of **edges** in the graph.

See also

PGraph.n

PGraph.goal

Set goal node

G.**goal**(**vg**) computes the cost of reaching every vertex in the graph connected to the **goal** vertex **vg**.

Notes

- Combined with G.path performs a breadth-first search for paths to the **goal**.

See also

PGraph.path, PGraph.Astar

PGraph.highlight_component

Highlight a graph component

G.**highlight_component**(**C**, **options**) highlights the vertices that belong to graph component **C**.

Options

'NodeSize', S	Size of vertex circle (default 12)
'NodeFaceColor', **C**	Node circle color (default yellow)
'NodeEdgeColor', **C**	Node circle edge color (default blue)

See also

PGraph.highlight_node, PGraph.highlight_edge, PGraph.highlight_component

---

PGraph.highlight_edge

Highlight a node

G.**highlight_edge**(**v1**, **v2**) highlights the edge between vertices **v1** and **v2**.

G.**highlight_edge**(**E**) highlights the edge with id **E**.

Options

'EdgeColor', C	Edge edge color (default black)
'EdgeThickness', T	Edge thickness (default 1.5)

See also

PGraph.highlight_node, PGraph.highlight_path, PGraph.highlight_component

---

PGraph.highlight_node

Highlight a node

G.**highlight_node**(**v**, **options**) highlights the vertex **v** with a yellow marker. If **v** is a list of vertices then all are highlighted.

Options

'NodeSize', S	Size of vertex circle (default 12)
'NodeFaceColor', C	Node circle color (default yellow)
'NodeEdgeColor', C	Node circle edge color (default blue)

See also

PGraph.highlight_edge, PGraph.highlight_path, PGraph.highlight_component

---

PGraph.highlight_path

Highlight path

G.**highlight_path**(**p**, **options**) highlights the path defined by vector **p** which is a list of vertices comprising the path.

Options

'NodeSize', S	Size of vertex circle (default 12)
'NodeFaceColor', C	Node circle color (default yellow)
'NodeEdgeColor', C	Node circle edge color (default blue)
'EdgeColor', C	Node circle edge color (default black)

See also

PGraph.highlight_node, PGraph.highlight_edge, PGraph.highlight_component

---

PGraph.incidence

Incidence matrix of graph

in = G.**incidence**() is a matrix $(N \times NE)$ where element **in**(i,j) is non-zero if vertex id i is connected to edge id j.

See also

PGraph.adjacency, PGraph.degree, PGraph.laplacian

PGraph.laplacian

Laplacian matrix of graph

L = G.**laplacian**() is the Laplacian matrix $(N \times N)$ of the graph.

Notes

- **L** is always positive-semidefinite.
- **L** has at least one zero eigenvalue.
- The number of zero eigenvalues is the number of connected components in the graph.

See also

PGraph.adjacency, PGraph.incidence, PGraph.degree

PGraph.merge

the dominant and submissive labels

PGraph.neighbours

Neighbours of a vertex

n = G.**neighbours**(**v**) is a vector of ids for all vertices which are directly connected **neighbours** of vertex **v**.

[**n**,**C**] = G.**neighbours**(**v**) as above but also returns a vector **C** whose elements are the edge costs of the paths corresponding to the vertex ids in **n**.

PGraph.neighbours_d

Directed neighbours of a vertex

n = G.**neighbours_d**(**v**) is a vector of ids for all vertices which are directly connected neighbours of vertex **v**. Elements are positive if there is a link from **v** to the node, and negative if the link is from the node to **v**.

[**n**,**C**] = G.**neighbours_d**(**v**) as above but also returns a vector **C** whose elements are the edge costs of the paths corresponding to the vertex ids in **n**.

PGraph.path

Find path to goal node

p = G.**path**(**vs**) is a vector of vertex ids that form a **path** from the starting vertex **vs** to the previously specified goal. The **path** includes the start and goal vertex id.

To compute **path** to goal vertex 5

```
g.goal(5);
```

then the **path**, starting from vertex 1 is

```
p1 = g.path(1);
```

and the **path** starting from vertex 2 is

```
p2 = g.path(2);
```

Notes

- Pgraph.goal must have been invoked first.
- Can be used repeatedly to find paths from different starting points to the goal specified to Pgraph.goal().

See also

PGraph.goal, PGraph.Astar

PGraph.pick

Graphically select a vertex

v = G.**pick**() is the id of the vertex closest to the point clicked by the user on a plot of the graph.

See also

PGraph.plot

PGraph.plot

Plot the graph

G.**plot**(**opt**) plots the graph in the current figure. Nodes are shown as colored circles.

Options

'labels'	Display vertex id (default false)
'edges'	Display edges (default true)
'edgelabels'	Display edge id (default false)
'NodeSize', S	Size of vertex circle (default 8)
'NodeFaceColor', C	Node circle color (default blue)
'NodeEdgeColor', C	Node circle edge color (default blue)
'NodeLabelSize', S	Node label text sizer (default 16)
'NodeLabelColor', C	Node label text color (default blue)
'EdgeColor', C	Edge color (default black)
'EdgeLabelSize', S	Edge label text size (default black)
'EdgeLabelColor', C	Edge label text color (default black)
'componentcolor'	Node color is a function of graph component

PGraph.setcost

Set cost of edge

G.**setcost**(**E**, **C**) set cost of edge id **E** to **C**.

PGraph.setdata

Set user data for node

G.**setdata**(**v**, **u**) sets the user data of vertex **v** to **u** which can be of any type such as number, struct, object or cell array.

See also

PGraph.data

PGraph.vertices

Find vertices given edge

v = G.**vertices**(**E**) return the id of the **vertices** that define edge **E**.

---

plot2

Plot trajectories

plot2(**p**) plots a line with coordinates taken from successive rows of **p**. **p** can be $N \times 2$ or $N \times 3$.

If **p** has three dimensions, ie. $N \times 2 \times M$ or $N \times 3 \times M$ then the M trajectories are overlaid in the one plot.

plot2(**p**, **ls**) as above but the line style arguments **ls** are passed to plot.

See also

plot

---

plot_arrow

Plot arrow

plot_arrow(**p**, **options**) draws an arrow from P1 to P2 where **p**=[P1; P2].

See also

arrow3

---

plot_box

a box on the current plot

plot_box(**b**, **ls**) draws a box defined by **b**=[XL XR; YL YR] with optional Matlab linestyle options **ls**.

plot_box(**x1**,**y1**, **x2**,**y2**, **ls**) draws a box with corners at (**x1**,**y1**) and (**x2**,**y2**), and optional Matlab linestyle options **ls**.

plot_box('centre', P, 'size', W, **ls**) draws a box with center at P=[X,Y] and with dimensions W=[WIDTH HEIGHT].

plot_box('topleft', P, 'size', W, **ls**) draws a box with top-left at P=[X,Y] and with dimensions W=[WIDTH HEIGHT].

---

plot_circle

Draw a circle on the current plot

plot_circle(**C**, **R**, **options**) draws a circle on the current plot with centre **C**=[X,Y] and radius **R**. If **C**=[X,Y,Z] the circle is drawn in the XY-plane at height Z.

H = **plot_circle**(**C**, **R**, **options**) as above but return handles. For multiple circles **H** is a vector of handles, one per circle.

Options

'edgecolor'	the color of the circle's edge, Matlab color spec
'fillcolor'	the color of the circle's interior, Matlab color spec
'alpha'	transparency of the filled circle: 0=transparent, 1=solid
'alter', **H**	alter existing circles with handle **H**

For an unfilled ellipse any MATLAB LineProperty **options** can be given, for a filled ellipse any MATLAB PatchProperty **options** can be given.

See also

plot_ellipse

---

plot_ellipse

Draw an ellipse on the current plot

plot_ellipse(**a**, **ls**) draws an ellipse defined by X'AX = 0 on the current plot, centred at the origin, with Matlab line style **ls**.

plot_ellipse(**a**, **C**, **ls**) as above but centred at **C**=[X,Y]. current plot. If **C**=[X,Y,Z] the ellipse is parallel to the XY plane but at height Z.

H = **plot_circle**(**C**, **R**, **options**) as above but return handles. For multiple circles **H** is a vector of handles, one per circle.

Options

'edgecolor'	the color of the circle's edge, Matlab color spec
'fillcolor'	the color of the circle's interior, Matlab color spec
'alpha'	transparency of the filled circle: 0=transparent, 1=solid
'alter', **H**	alter existing circles with handle **H**

See also

plot_circle

---

plot_homline

Draw a line in homogeneous form

H = **plot_homline**(**L**, **ls**) draws a line in the current figure **L**.X = 0. The current axis limits are used to determine the endpoints of the line. Matlab line specification **ls** can be set.

The return argument is a vector of graphics handles for the lines.

See also

homline

---

plot_point

point features

plot_point(**p**, **options**) adds point markers to a plot, where **p** $(2 \times N)$ and each column is the point coordinate.

Options

'textcolor', colspec	Specify color of text
'textsize', size	Specify size of text
'bold'	Text in bold font.
'printf', fmt, data	Label points according to printf format string and corresponding element of data
'sequence'	Label points sequentially

Additional options are passed through to PLOT for creating the marker.

Examples

Simple point plot

```
P = rand(2,4);
plot_point(P);
```

Plot points with markers

```
plot_point(P, '*');
```

Plot points with square markers and labels

```
plot_point(P, 'sequence', 's');
```

Plot points with circles and annotations

```
data = [1 2 4 8];
plot_point(P, 'printf', {' P%d', data}, 'o');
```

See also

plot, text

---

plot_poly

Plot a polygon

plotpoly(**p**, **options**) plot a polygon defined by columns of **p** which can be $2 \times N$ or $3 \times N$.

options

'fill'	the color of the circle's interior, Matlab color spec
'alpha'	transparency of the filled circle: 0=transparent, 1=solid.

See also

plot, patch, Polygon

---

plot_sphere

Plot spheres

plot_sphere(**C**, **R**, **color**) add spheres to the current figure. **C** is the centre of the sphere and if its a $3 \times N$ matrix then N spheres are drawn with centres as per the columns. **R** is the radius and **color** is a Matlab color spec, either a letter or 3-vector.

H = **plot_sphere**(**C**, **R**, **color**) as above but returns the handle(s) for the spheres.

H = **plot_sphere**(**C**, **R**, **color**, **alpha**) as above but **alpha** specifies the opacity of the sphere were 0 is transparant and 1 is opaque. The default is 1.

Example

Create four spheres

```
plot_sphere( mkgrid(2, 1), .2, 'b')
```

and now turn on a full lighting model

```
lighting gouraud
light
```

NOTES

- The sphere is always added, irrespective of figure hold state.
- The number of vertices to draw the sphere is hardwired.

---

plot_vehicle

Plot ground vehicle pose

plot_vehicle(**x**,**options**) draw representation of ground robot as an oriented triangle with pose **x** (1×3) [x,y,theta] or **x** (3×3) as homogeneous transform in SE(2).

Options

'scale', S	Draw vehicle with length S x maximum axis dimension
'size', S	Draw vehicle with length S

See also

Vehicle.plot

---

plotbotopt

Define default options for robot plotting

A user provided function that returns a cell array of default plot options for the SerialLink.plot method.

See also

SerialLink.plot

---

plotp

Plot trajectories

plotp(**p**) plots a set of points **p**, which by Toolbox convention are stored one per column. **p** can be $N \times 2$ or $N \times 3$. By default a linestyle of 'bx' is used.

plotp(**p**, **ls**) as above but the line style arguments **ls** are passed to plot.

See also

plot, plot2

---

polydiff

pd = polydiff(p)

Return the coefficients of the derivative of polynomial p

Polygon

Polygon class

A general class for manipulating polygons and vectors of polygons.

Methods

plot	Plot polygon
area	Area of polygon
moments	Moments of polygon
centroid	Centroid of polygon
perimeter	Perimter of polygon
transform	Transform polygon
inside	Test if points are inside polygon
intersection	Intersection of two polygons
difference	Difference of two polygons
union	Union of two polygons
xor	Exclusive or of two polygons
display	print the polygon in human readable form
char	convert the polgyon to human readable string

Properties

vertices	List of polygon vertices, one per column
extent	Bounding box [minx maxx; miny maxy]
n	Number of vertices

Notes

- This is reference class object
- Polygon objects can be used in vectors and arrays

Acknowledgement

The methods inside, intersection, difference, union, and xor are based on code written by:

Kirill K. Pankratov, kirill@plume.mit.edu, http://puddle.mit.edu/ glenn/kirill/saga.html

and require a licence. However the author does not respond to email regarding the licence, so use with care, and modify with acknowledgement.

Polygon.Polygon

Polygon class constructor

p = **Polygon**(**v**) is a polygon with vertices given by **v**, one column per vertex.

p = **Polygon**(**C**, **wh**) is a rectangle centred at **C** with dimensions **wh**=[WIDTH, HEIGHT].

Polygon.area

Area of polygon

a = P.**area**() is the **area** of the polygon.

Polygon.centroid

Centroid of polygon

x = P.**centroid**() is the **centroid** of the polygon.

Polygon.char

String representation

s = P.**char**() is a compact representation of the polgyon in human readable form.

Polygon.difference

Difference of polygons

d = P.**difference**(**q**) is polygon P minus polygon **q**.

Notes

- If polygons P and **q** are not intersecting, returns coordinates of P.
- If the result **d** is not simply connected or consists of several polygons, resulting vertex list will contain NaNs.

Polygon.display

Display polygon

P.**display**() displays the polygon in a compact human readable form.

See also

Polygon.char

Polygon.inside

Test if points are inside polygon

in = **p**.**inside**(**p**) tests if points given by columns of **p** are **inside** the polygon. The corresponding elements of **in** are either true or false.

Polygon.intersect

Intersection of polygon with list of polygons

i = P.**intersect**(**plist**) indicates whether or not the **Polygon** P intersects with

i(**j**) = 1 if p intersects polylist(**j**), else 0.

Polygon.intersect_line

Intersection of polygon and line segment

i = P.**intersect_line**(**L**) is the intersection points of a polygon P with the line segment **L**=[x1 x2; y1 y2]. **i** is an $N \times 2$ matrix with one column per intersection, each column is [x y]'.

Polygon.intersection

Intersection of polygons

i = P.**intersection**(**q**) is a **Polygon** representing the **intersection** of polygons P and **q**.

Notes

- If these polygons are not intersecting, returns empty polygon.
- If **intersection** consist of several disjoint polygons (for non-convex P or **q**) then vertices of **i** is the concatenation of the vertices of these polygons.

Polygon.linechk

Input checking for line segments.

Polygon.moments

Moments of polygon

a = P.**moments**(**p**, **q**) is the pq'th moment of the polygon.

See also

mpq_poly

Polygon.perimeter

Perimeter of polygon

L = P.**perimeter**() is the **perimeter** of the polygon.

Polygon.plot

Plot polygon

P.**plot**() **plot** the polygon.

P.**plot**(**ls**) as above but pass the arguments **ls** to **plot**.

Polygon.transform

Transformation of polygon vertices

p2 = P.**transform**(**T**) is a new **Polygon** object whose vertices have been transfored by the 3×3 homgoeneous transformation **T**.

Polygon.union

Union of polygons

i = P.**union**(**q**) is a **Polygon** representing the **union** of polygons P and **q**.

Notes

- If these polygons are not intersecting, returns a polygon with vertices of both polygons separated by NaNs.
- If the result P is not simply connected (such as a polygon with a “hole”) the resulting contour consist of counter- clockwise “outer boundary” and one or more clock-wise “inner boundaries” around “holes”.

Polygon.xor

Exclusive or of polygons

i = P.**union**(**q**) is a **Polygon** representing the **union** of polygons P and **q**.

Notes

- If these polygons are not intersecting, returns a polygon with vertices of both polygons separated by NaNs.
- If the result P is not simply connected (such as a polygon with a “hole”) the resulting contour consist of counter- clockwise “outer boundary” and one or more clock-wise “inner boundaries” around “holes”.

Prismatic

Robot manipulator Prismatic link class

A subclass of the Link class: holds all information related to a robot link such as kinematics parameters, rigid-body inertial parameters, motor and transmission parameters.

Notes

- This is reference class object
- Link class objects can be used in vectors and arrays

References

- Robotics, Vision & Control, Chap 7 P. Corke, Springer 2011.

See also

Link, Revolute, SerialLink

PrismaticMDH

Robot manipulator Prismatic link class for MDH convention

A subclass of the Link class: holds all information related to a robot link such as kinematics parameters, rigid-body inertial parameters, motor and transmission parameters.

Notes

- This is reference class object
- Link class objects can be used in vectors and arrays
- Modified Denavit-Hartenberg parameters are used

References

- Robotics, Vision & Control, Chap 7 P. Corke, Springer 2011.

See also

Link, Prismatic, RevoluteMDH, SerialLink

PRM

Probabilistic RoadMap navigation class

A concrete subclass of the Navigation class that implements the probabilistic roadmap navigation algorithm. This performs goal independent planning of roadmaps, and at the query stage finds paths between specific start and goal points.

Methods

plan	Compute the roadmap
path	Compute a path to the goal
visualize	Display the obstacle map (deprecated)
plot	Display the obstacle map
display	Display the parameters in human readable form
char	Convert to string

Example

```
load map1              % load map
goal = [50,30];        % goal point
start = [20, 10];      % start point
prm = PRM(map);        % create navigation object
prm.plan()             % create roadmaps
prm.path(start, goal)  % animate path from this start location
```

References

- Probabilistic roadmaps for path planning in high dimensional configuration spaces, L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, IEEE Transactions on Robotics and Automation, vol. 12, pp. 566-580, Aug 1996.
- Robotics, Vision & Control, Section 5.2.4, P. Corke, Springer 2011.

See also

Navigation, DXform, Dstar, PGraph

PRM.PRM

Create a PRM navigation object

p = PRM(**map**, **options**) is a probabilistic roadmap navigation object, and **map** is an occupancy grid, a representation of a planar world as a matrix whose elements are 0 (free space) or 1 (occupied).

Options

'npoints', N	Number of sample points (default 100)
'distthresh', D	Distance threshold, edges only connect vertices closer than D (default 0.3 max(size(occgrid)))

Other **options** are supported by the Navigation superclass.

See also

Navigation.Navigation

PRM.char

Convert to string

P.**char**() is a string representing the state of the **PRM** object in human-readable form.

See also

PRM.display

PRM.path

Find a path between two points

P.**path**(**start**, **goal**) finds and displays a **path** from **start** to **goal** which is overlaid on the occupancy grid.

x = P.**path**(**start**) returns the **path** ($2 \times M$) from **start** to **goal**.

PRM.plan

Create a probabilistic roadmap

P.**plan**() creates the probabilistic roadmap by randomly sampling the free space in the map and building a graph with edges connecting close points. The resulting graph is kept within the object.

PRM.plot

Visualize navigation environment

P.**plot**() displays the occupancy grid with an optional distance field.

Options

'goal'	Superimpose the goal position if set
'nooverlay'	Don't overlay the PRM graph

---

qplot

plot joint angles

qplot(**q**) is a convenience function to plot joint angle trajectories ($M \times 6$) for a 6-axis robot, where each row represents one time step.

The first three joints are shown as solid lines, the last three joints (wrist) are shown as dashed lines. A legend is also displayed.

qplot(**T**, **q**) as above but displays the joint angle trajectory versus time **T** ($M \times 1$).

See also

jtraj, plot

Quaternion

Quaternion class

A quaternion is a compact method of representing a 3D rotation that has computational advantages including speed and numerical robustness. A quaternion has 2 parts, a scalar s, and a vector v and is typically written: q = s <vx, vy, vz>.

A unit-quaternion is one for which $s^2+vx^2+vy^2+vz^2 = 1$. It can be considered as a rotation by an angle theta about a unit-vector V in space where

```
q = cos (theta/2) < v sin(theta/2)>
```

q = **quaternion**(**x**) is a unit-**quaternion** equivalent to **x** which can be any of:

- orthonormal rotation matrix.
- homogeneous transformation matrix (rotation part only).
- rotation angle and vector

Methods

inv	inverse of quaterion
norm	norm of **quaternion**
unit	unitized **quaternion**
plot	same options as trplot()
interp	interpolation (slerp) between q and q2, 0<=s<=1
scale	interpolation (slerp) between identity and q, 0<=s<=1
dot	derivative of **quaternion** with angular velocity w
R	equivalent 3×3 rotation matrix
T	equivalent 4×4 homogeneous transform matrix
double	**quaternion** elements as 4-vector
inner	inner product of two quaternions

Arithmetic operators are overloaded

q1==q2	test for **quaternion** equality
q1 =q2	test for **quaternion** inequality
q+q2	elementwise sum of quaternions
q-q2	elementwise difference of quaternions
q*q2	**quaternion** product
q*v	rotate vector by **quaternion**, v is 3×1
s*q	elementwise multiplication of **quaternion** by scalar
q/q2	q*q2.inv
q^n	q to power n (integer only)

Properties (read only)

s	real part
v	vector part

Notes

- **quaternion** objects can be used in vectors and arrays

References

- Animating rotation with **quaternion** curves, K. Shoemake, in Proceedings of ACM SIGGRAPH, (San Fran cisco), pp. 245-254, 1985.

- On homogeneous transforms, quaternions, and computational efficiency, J. Funda, R. Taylor, and R. Paul, IEEE Transactions on Robotics and Automation, vol. 6, pp. 382-388, June 1990.
- Robotics, Vision & Control, P. Corke, Springer 2011.

See also

trinterp, trplot

Quaternion.Quaternion

Constructor for quaternion objects

Construct a **quaternion** from various other orientation representations.

q = **Quaternion**() is the identity quaternion 1<0,0,0> representing a null rotation.

q = **Quaternion**(**q1**) is a copy of the quaternion **q1**

q = **Quaternion**([S V1 V2 V3]) is a quaternion formed by specifying directly its 4 elements

q = **Quaternion**(**s**) is a quaternion formed from the scalar **s** and zero vector part: **s**<0,0,0>

q = **Quaternion**(**v**) is a pure quaternion with the specified vector part: 0<**v**>

q = **Quaternion**(**th**, **v**) is a unit-quaternion corresponding to rotation of **th** about the vector **v**.

q = **Quaternion**(**R**) is a unit-quaternion corresponding to the orthonormal rotation matrix **R**. If **R** $(3 \times 3 \times N)$ is a sequence then **q** $(N \times 1)$ is a vector of Quaternions corresponding to the elements of **R**.

q = **Quaternion**(**T**) is a unit-quaternion equivalent to the rotational part of the homogeneous transform **T**. If **T** $(4 \times 4 \times N)$ is a sequence then **q** $(N \times 1)$ is a vector of Quaternions corresponding to the elements of **T**.

Quaternion.char

Convert to string

s = Q.**char**() is a compact string representation of the quaternion's value as a 4-tuple. If Q is a vector then **s** has one line per element.

Quaternion.display

Display the value of a quaternion object

Q.**display**() displays a compact string representation of the quaternion's value as a 4-tuple. If Q is a vector then S has one line per element.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Quaternion object and the command has no trailing semicolon.

See also

Quaternion.char

Quaternion.dot

Quaternion derivative

qd = Q.**dot**(**omega**) is the rate of change of a frame with attitude Q and angular velocity OMEGA (1×3) expressed as a quaternion.

Quaternion.double

Convert a quaternion to a 4-element vector

v = Q.**double**() is a 4-vector comprising the quaternion elements [s vx vy vz].

Quaternion.eq

Test quaternion equality

Q1==Q2 is true if the quaternions Q1 and Q2 are equal.

Notes

- Overloaded operator '=='.
- Note that for unit Quaternions Q and -Q are the equivalent rotation, so non-equality does not mean rotations are not equivalent.
- If Q1 is a vector of quaternions, each element is compared to Q2 and the result is a logical array of the same length as Q1.
- If Q2 is a vector of quaternions, each element is compared to Q1 and the result is a logical array of the same length as Q2.
- If Q1 and Q2 are vectors of the same length, then the result is a logical array

See also

Quaternion.ne

Quaternion.inner

Quaternion inner product

v = Q1.**inner**(**q2**) is the **inner** (dot) product of two vectors (1×4), comprising the elements of Q1 and **q2** respectively.

Notes

- Q1.**inner**(Q1) is the same as Q1.norm().

See also

Quaternion.norm

Quaternion.interp

Interpolate quaternions

qi = Q1.**interp**(**q2**, **s**) is a unit-quaternion that interpolates a rotation between Q1 for **s**=0 and **q2** for **s**=1.

If **s** is a vector **qi** is a vector of quaternions, each element corresponding to sequential elements of **s**.

Notes

- This is a spherical linear interpolation (slerp) that can be interpretted as interpolation along a great circle arc on a sphere.
- The value of **s** is clipped to the interval 0 to 1.

See also

ctraj, Quaternion.scale

Quaternion.inv

Invert a unit-quaternion

qi = Q.**inv**() is a quaternion object representing the inverse of Q.

Quaternion.minus

Subtract quaternions

Q1-Q2 is the element-wise difference of quaternion elements.

Notes

- Overloaded operator '-'
- The result is not guaranteed to be a unit-quaternion.

See also

Quaternion.plus, Quaternion.mtimes

Quaternion.mpower

Raise quaternion to integer power

Q^N is the quaternion Q raised to the integer power N.

Notes

- Overloaded operator '^'
- Computed by repeated multiplication.

See also

Quaternion.mrdivide, Quaternion.mpower, Quaternion.plus, Quaternion.minus

---

Quaternion.mrdivide

Quaternion quotient.

Q1/Q2	is a quaternion formed by Hamilton product of Q1 and **inv**(Q2).
Q/S	is the element-wise division of quaternion elements by the scalar S.

Notes

- Overloaded operator '/'

See also

Quaternion.mtimes, Quaternion.mpower, Quaternion.plus, Quaternion.minus

---

Quaternion.mtimes

Multiply a quaternion object

Q1*Q2	is a quaternion formed by the Hamilton product of two quaternions.
Q*V	is a vector formed by rotating the vector V by the quaternion Q.
Q*S	is the element-wise multiplication of quaternion elements by the scalar S.

Notes

- Overloaded operator '*'

See also

Quaternion.mrdivide, Quaternion.mpower, Quaternion.plus, Quaternion.minus

---

Quaternion.ne

Test quaternion inequality

Q1 =Q2 is true if the quaternions Q1 and Q2 are not equal.

Notes

- Overloaded operator ' ='
- Note that for unit Quaternions Q and -Q are the equivalent rotation, so non-equality does not mean rotations are not equivalent.
- If Q1 is a vector of quaternions, each element is compared to Q2 and the result is a logical array of the same length as Q1.
- If Q2 is a vector of quaternions, each element is compared to Q1 and the result is a logical array of the same length as Q2.
- If Q1 and Q2 are vectors of the same length, then the result is a logical array.

See also

Quaternion.eq

Quaternion.norm

Quaternion magnitude

qn = **q**.**norm**(**q**) is the scalar **norm** or magnitude of the quaternion **q**.

Notes

- This is the Euclidean **norm** of the quaternion written as a 4-vector.
- A unit-quaternion has a **norm** of one.

See also

Quaternion.inner, Quaternion.unit

Quaternion.plot

Plot a quaternion object

Q.**plot**(**options**) plots the quaternion as a rotated coordinate frame.

Options

Options are passed to trplot and include:

'color', C	The color to draw the axes, MATLAB colorspec C
'frame', F	The frame is named F and the subscript on the axis labels is F.
'view', V	Set **plot** view parameters V=[az el] angles, or 'auto' for view toward origin of coordinate frame

See also

trplot

Quaternion.plus

Add quaternions

Q1+Q2 is the element-wise sum of quaternion elements.

Notes

- Overloaded operator '+'
- The result is not guaranteed to be a unit-quaternion.

See also

Quaternion.minus, Quaternion.mtimes

Quaternion.R

Convert toorthonormal rotation matrix

R = Q.**R**() is the equivalent 3×3 orthonormal rotation matrix.

Notes:

- For a quaternion sequence returns a rotation matrix sequence.

Quaternion.scale

Interpolate rotations expressed by quaternion objects

qi = Q.**scale**(**s**) is a unit-quaternion that interpolates between identity for **s**=0 to Q for **s**=1. This is a spherical linear interpolation (slerp) that can be interpretted as interpolation along a great circle arc on a sphere.

If **s** is a vector **qi** is a cell array of quaternions, each element corresponding to sequential elements of **s**.

Notes

- This is a spherical linear interpolation (slerp) that can be interpretted as interpolation along a great circle arc on a sphere.

See also

ctraj, Quaternion.interp

Quaternion.T

Convert to homogeneous transformation matrix

T = Q.**T**() is the equivalent 4×4 homogeneous transformation matrix.

Notes:

- For a quaternion sequence returns a homogeneous transform matrix sequence
- Has a zero translational component.

Quaternion.unit

Unitize a quaternion

qu = Q.**unit**() is a **unit**-quaternion representing the same orientation as Q.

See also

Quaternion.norm

---

r2t

Convert rotation matrix to a homogeneous transform

T = **r2t**(**R**) is a homogeneous transform equivalent to an orthonormal rotation matrix **R** with a zero translational component.

Notes

- Works for **T** in either SE(2) or SE(3)
  - if **R** is 2×2 then **T** is 3×3, or
  - if **R** is 3×3 then **T** is 4×4.
- Translational component is zero.
- For a rotation matrix sequence returns a homogeneous transform sequence.

See also

t2r

---

randinit

Reset random number generator

RANDINIT reset the defaul random number stream.

See also

randstream

RandomPath

Vehicle driver class

Create a "driver" object capable of driving a Vehicle object through random waypoints within a rectangular region and at constant speed.

The driver object is attached to a Vehicle object by the latter's add_driver() method.

Methods

init	reset the random number generator
demand	return speed and steer angle to next waypoint
display	display the state and parameters in human readable form
char	convert to string

Properties

goal	current goal coordinate
veh	the Vehicle object being controlled
dim	dimensions of the work space (2×1) [m]
speed	speed of travel [m/s]
closeenough	proximity to waypoint at which next is chosen [m]

Example

```
veh = Vehicle(V);
veh.add_driver( RandomPath(20, 2) );
```

Notes

- It is possible in some cases for the vehicle to move outside the desired region, for instance if moving to a waypoint near the edge, the limited turning circle may cause the vehicle to temporarily move outside.
- The vehicle chooses a new waypoint when it is closer than property closeenough to the current waypoint.

- Uses its own random number stream so as to not influence the performance of other randomized algorithms such as path planning.

Reference

Robotics, Vision & Control, Chap 6, Peter Corke, Springer 2011

See also

Vehicle

RandomPath.RandomPath

Create a driver object

d = **RandomPath**(**dim**, **options**) returns a “driver” object capable of driving a Vehicle object through random waypoints. The waypoints are positioned inside a rectangular region bounded by +/- **dim** in the x- and y-directions.

Options

‘speed’, S	Speed along path (default 1m/s).
‘dthresh’, **d**	Distance from goal at which next goal is chosen.

See also

Vehicle

RandomPath.char

Convert to string

s = R.**char**() is a string showing driver parameters and state in in a compact human readable format.

RandomPath.demand

Compute speed and heading to waypoint

[**speed**,**steer**] = R.**demand**() returns the speed and steer angle to drive the vehicle toward the next waypoint. When the vehicle is within R.closeenough a new waypoint is chosen.

See also

Vehicle

RandomPath.display

Display driver parameters and state

R.**display**() displays driver parameters and state in compact human readable form.

See also

RandomPath.char

RandomPath.init

Reset random number generator

R.**init**() resets the random number generator used to create the waypoints. This enables the sequence of random waypoints to be repeated.

See also

randstream

RangeBearingSensor

Range and bearing sensor class

A concrete subclass of the Sensor class that implements a range and bearing angle sensor that provides robot-centric measurements of point features in the world. To enable this it has references to a map of the world (Map object) and a robot moving through the world (Vehicle object).

Methods

reading	range/bearing observation of random feature
h	range/bearing observation of specific feature
Hx	Jacobian matrix dh/dxv
Hxf	Jacobian matrix dh/dxf
Hw	Jacobian matrix dh/dw

g	feature positin given vehicle pose and observation
Gx	Jacobian matrix dg/dxv
Gz	Jacobian matrix dg/dz

Properties (read/write)

W	measurement covariance matrix (2×2)
interval	valid measurements returned every interval'th call to reading()

Reference

Robotics, Vision & Control, Chap 6, Peter Corke, Springer 2011

See also

Sensor, Vehicle, Map, EKF

RangeBearingSensor.RangeBearingSensor

Range and bearing sensor constructor

s = **RangeBearingSensor**(**vehicle**, **map**, **w**, **options**) is an object representing a range and bearing angle sensor mounted on the Vehicle object **vehicle** and observing an environment of known landmarks represented by the map object **map**. The sensor covariance is R (2×2) representing range and bearing covariance.

Options

'range', xmax	maximum range of sensor
'range', [xmin xmax]	minimum and maximum range of sensor
'angle', TH	detection for angles betwen -TH to +TH
'angle', [THMIN THMAX]	detection for angles betwen THMIN and THMAX
'skip', I	return a valid reading on every I'th call
'fail', [TMIN TMAX]	sensor simulates failure between timesteps TMIN and TMAX

See also

Sensor, Vehicle, Map, EKF

RangeBearingSensor.g

Compute landmark location

p = S.**g**(**xv**, **z**) is the world coordinate (1×2) of a feature given the sensor observation **z** (1×2) and vehicle state **xv** (3×1).

See also

RangeBearingSensor.Gx, RangeBearingSensor.Gz

RangeBearingSensor.Gx

Jacobian dg/dx

J = S.**Gx**(**xv**, **z**) is the Jacobian dg/dxv (2×3) at the vehicle state **xv** (3×1) for sensor observation **z** (2×1).

See also

RangeBearingSensor.g

RangeBearingSensor.Gz

Jacobian dg/dz

J = S.**Gz**(**xv**, **z**) is the Jacobian dg/dz (2×2) at the vehicle state **xv** (3×1) for sensor observation **z** (2×1).

See also

RangeBearingSensor.g

RangeBearingSensor.h

Landmark range and bearing

z = S.**h**(**xv**, **J**) is a sensor observation (1×2), range and bearing, from vehicle at pose **xv** (1×3) to the map feature K.

z = S.**h**(**xv**, **xf**) as above but compute range and bearing to a feature at coordinate **xf**.

z = s.**h**(**xv**) as above but computer range and bearing to all map features. **z** has one row per feature.

Notes

- Noise with covariance W is added to each row of **z**.
- Supports vectorized operation where **xv** ($N \times 3$) and **z** ($N \times 2$).

See also

RangeBearingSensor.Hx, RangeBearingSensor.Hw, RangeBearingSensor.Hxf

---

RangeBearingSensor.Hw

Jacobian dh/dv

J = S.**Hw**(**xv**, **k**) is the Jacobian dh/dv (2×2) at the vehicle state **xv** (3×1) for map feature **k**.

See also

RangeBearingSensor.h

RangeBearingSensor.Hx

Jacobian dh/dxv

J = S.**Hx**(**xv**, **k**) returns the Jacobian dh/dxv (2×3) at the vehicle state **xv** (3×1) for map feature **k**.

J = S.**Hx**(**xv**, **xf**) as above but for a feature at coordinate **xf**.

See also

RangeBearingSensor.h

RangeBearingSensor.Hxf

Jacobian dh/dxf

J = S.**Hxf**(**xv**, **k**) is the Jacobian dh/dxv (2×2) at the vehicle state **xv** (3×1) for map feature **k**.

J = S.**Hxf**(**xv**, **xf**) as above but for a feature at coordinate **xf** (1×2).

See also

RangeBearingSensor.h

RangeBearingSensor.reading

Landmark range and bearing

[**z**,**k**] = S.**reading**() is an observation of a random landmark where **z**=[R,THETA] is the range and bearing with additive Gaussian noise of covariance R (specified to the constructor). **k** is the index of the map feature that was observed. If no valid measurement, ie. no features within range, interval subsampling enabled or simulated failure the return is **z**=[] and **k**=NaN.

See also

RangeBearingSensor.h

Revolute

Robot manipulator Revolute link class

A subclass of the Link class: holds all information related to a robot link such as kinematics parameters, rigid-body inertial parameters, motor and transmission parameters.

Notes

- This is reference class object
- Link class objects can be used in vectors and arrays

References

- Robotics, Vision & Control, Chap 7 P. Corke, Springer 2011.

See also

Link, Prismatic, SerialLink

RevoluteMDH

Robot manipulator Revolute link class for MDH convention

A subclass of the Link class: holds all information related to a robot link such as kinematics parameters, rigid-body inertial parameters, motor and transmission parameters.

Notes

- This is reference class object
- Link class objects can be used in vectors and arrays

- Modified Denavit-Hartenberg parameters are used

References

- Robotics, Vision & Control, Chap 7 P. Corke, Springer 2011.

See also

Link, Prismatic, SerialLink

See also

Link, PrismaticMDH, Revolute, SerialLink

RobotArm

Serial-link robot arm class

A subclass of SerialLink than includes an interface to a physical robot.

Methods

plot	display graphical representation of robot
teach	drive the physical and graphical robots
mirror	use the robot as a slave to drive graphics
jmove	joint space motion of the physical robot
cmove	Cartesian space motion of the physical robot
isspherical	test if robot has spherical wrist
islimit	test if robot at joint limit
fkine	forward kinematics
ikine6s	inverse kinematics for 6-axis spherical wrist revolute robot
ikine3	inverse kinematics for 3-axis revolute robot
ikine	inverse kinematics using iterative method
jacob0	Jacobian matrix in world frame
jacobn	Jacobian matrix in tool frame

Properties (read/write)

links	vector of Link objects ($1 \times N$)
gravity	direction of gravity [gx gy gz]
base	pose of robot's base (4×4 homog xform)
tool	robot's tool transform, T6 to tool tip (4×4 homog xform)
qlim	joint limits, [qmin qmax] ($N \times 2$)
offset	kinematic joint coordinate offsets ($N \times 1$)
name	name of robot, used for graphical display
manuf	annotation, manufacturer's name
comment	annotation, general comment
plotopt	options for plot() method (cell array)

Object properties (read only)

n	number of joints
config	joint configuration string, eg. 'RRRRRR'
mdh	kinematic convention boolean (0=DH, 1=MDH)

Note

- RobotArm is a subclass of SerialLink.
- RobotArm is a handle subclass object.
- RobotArm objects can be used in vectors and arrays

Reference

- Robotics, Vision & Control, Chaps 7-9, P. Corke, Springer 2011.
- Robot, Modeling & Control, M.Spong, S. Hutchinson & M. Vidyasagar, Wiley 2006.

See also

SerialLink, Link, DHFactor

RobotArm.RobotArm

Construct a RobotArm object

ra = **RobotArm**(**L**, **m**, **options**) is a robot object defined by a vector of Link objects **L** with a physical robot (machine) interface **m**.

Options

'name', name	set robot name property
'comment', comment	set robot comment property
'manufacturer', manuf	set robot manufacturer property
'base', base	set base transformation matrix property
'tool', tool	set tool transformation matrix property
'gravity', g	set gravity vector property
'plotopt', po	set plotting options property

See also

SerialLink.SerialLink, Arbotix.Arbotix

RobotArm.cmove

Cartesian space move

RA.**cmove**(**T**) moves the robot arm to the pose specified by the homogeneous transformation (4×4).

%

Notes

- A trajectory is computed from the current configuration to QD.

See also

RobotArm.jmove, Arbotix.setpath

RobotArm.delete

Destroy the RobotArm object

RA.**delete**() destroys the machine interface and the **RobotArm** object.

RobotArm.getq

Get the robot joint angles

q = RA.**getq**() are a vector of robot joint angles.

Notes

- If the robot has a gripper, its value is not included in this vector.

RobotArm.gripper

Control the robot gripper

RA.**gripper**(**C**) sets the robot **gripper** according to **C** which is 0 for closed and 1 for open.

Notes

- Not all robots have a **gripper**.
- The **gripper** is assumed to be the last servo motor in the chain.

RobotArm.jmove

Joint space move

RA.**jmove**(**qd**) moves the robot arm to the configuration specified by the joint angle vector **qd** $(1 \times N)$.

RA.**jmove**(**qd**, **T**) as above but the total move takes **T** seconds.

Notes

- A trajectory is computed from the current configuration to **qd**.

See also

RobotArm.cmove, Arbotix.setpath

RobotArm.mirror

Mirror the robot pose to graphics

RA.**mirror**() places the robot arm in relaxed mode, and as it is moved by hand the graphical animation follows.

See also

SerialLink.teach, SerialLink.plot

RobotArm.teach

Teach the robot

RA.**teach**() invokes a simple GUI to allow joint space motion, as well as showing an animation of the robot on screen.

See also

SerialLink.teach, SerialLink.plot

---

rot2

SO2 Rotation matrix

R = **rot2**(**theta**) is an SO(2) rotation matrix representing a rotation of **theta** radians.

R = **rot2**(**theta**, 'deg') as above but **theta** is in degrees.

See also

trot2, rotx, roty, rotz

---

rotx

Rotation about X axis

R = rotx(**theta**) is a rotation matrix representing a rotation of **theta** radians about the x-axis.

R = rotx(**theta**, 'deg') as above but **theta** is in degrees.

See also

roty, rotz, angvec2r, rot2

roty

Rotation about Y axis

R = roty(**theta**) is a rotation matrix representing a rotation of **theta** radians about the y-axis.

R = roty(**theta**, 'deg') as above but **theta** is in degrees.

See also

rotx, rotz, angvec2r, rot2

rotz

Rotation about Z axis

R = rotz(**theta**) is a rotation matrix representing a rotation of **theta** radians about the z-axis.

R = rotz(**theta**, 'deg') as above but **theta** is in degrees.

See also

rotx, roty, angvec2r, rot2

---

rpy2jac

Jacobian from RPY angle rates to angular velocity

J = **rpy2jac**(**eul**) is a Jacobian matrix (3 × 3) that maps roll-pitch-yaw angle rates to angular velocity at the operating point RPY=[R,P,Y].

J = **rpy2jac**(**R**, **p**, **y**) as above but the roll-pitch-yaw angles are passed as separate arguments.

Notes

- Used in the creation of an analytical Jacobian.

See also

eul2jac, SerialLink.JACOBN

---

rpy2r

Roll-pitch-yaw angles to rotation matrix

R = **rpy2r**(**rpy**, **options**) is an orthonormal rotation matrix equivalent to the specified roll, pitch, yaw angles which correspond to rotations about the X, Y, Z axes respectively. If **rpy** has multiple rows they are assumed to represent a trajectory and **R** is a three dimensional matrix, where the last index corresponds to the rows of **rpy**.

R = **rpy2r**(**roll**, **pitch**, **yaw**, **options**) as above but the roll-pitch-yaw angles are passed as separate arguments. If **roll**, **pitch** and **yaw** are column vectors they are assumed to represent a trajectory and **R** is a three dimensional matrix, where the last index corresponds to the rows of **roll**, **pitch**, **yaw**.

Options

'deg'	Compute angles in degrees (radians default)
'zyx'	Return solution for sequential rotations about Z, Y, X axes (Paul book)

Note

- In previous releases ($<$8) the angles corresponded to rotations about ZYX. Many texts (Paul, Spong) use the rotation order ZYX. This old behaviour can be enabled by passing the option 'zyx'

See also

tr2rpy, eul2tr

---

rpy2tr

Roll-pitch-yaw angles to homogeneous transform

T = **rpy2tr**(**rpy**, **options**) is a homogeneous transformation equivalent to the specified roll, pitch, yaw angles which correspond to rotations about the X, Y, Z axes respectively. If **rpy** has multiple rows they are assumed to represent a trajectory and **T** is a three dimensional matrix, where the last index corresponds to the rows of **rpy**.

T = **rpy2tr**(**roll**, **pitch**, **yaw**, **options**) as above but the roll-pitch-yaw angles are passed as separate arguments. If **roll**, **pitch** and **yaw** are column vectors they are assumed to represent a trajectory and **T** is a three dimensional matrix, where the last index corresponds to the rows of **roll**, **pitch**, **yaw**.

Options

'deg'	Compute angles in degrees (radians default)
'zyx'	Return solution for sequential rotations about Z, Y, X axes (Paul book)

Note

- In previous releases ($<$8) the angles corresponded to rotations about ZYX. Many texts (Paul, Spong) use the rotation order ZYX. This old behaviour can be enabled by passing the option 'zyx'

See also

tr2rpy, rpy2r, eul2tr

RRT

Class for rapidly-exploring random tree navigation

A concrete subclass of the Navigation class that implements the rapidly exploring random tree (RRT) algorithm. This is a kinodynamic planner that takes into account the motion constraints of the vehicle.

Methods

plan	Compute the tree
path	Compute a path
plot	Display the tree
display	Display the parameters in human readable form
char	Convert to string

Example

```
goal = [0,0,0];
start = [0,2,0];
veh = Vehicle([], 'stlim', 1.2);
rrt = RRT([], veh, 'goal', goal, 'range', 5);
rrt.plan()              % create navigation tree
rrt.path(start, goal)   % animate path from this start location
```

Robotics, Vision & Control compatability mode:

```
goal = [0,0,0];
start = [0,2,0];
rrt = RRT();            % create navigation object
rrt.plan()              % create navigation tree
rrt.path(start, goal)   % animate path from this start location
```

References

- Randomized kinodynamic planning, S. LaValle and J. Kuffner, International Journal of Robotics Research vol. 20, pp. 378-400, May 2001.
- Probabilistic roadmaps for path planning in high dimensional configuration spaces, L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, IEEE Transactions on Robotics and Automation, vol. 12, pp. 566-580, Aug 1996.

- Robotics, Vision & Control, Section 5.2.5, P. Corke, Springer 2011.

See also

Navigation, PRM, DXform, Dstar, PGraph

RRT.RRT

Create a RRT navigation object

R = RRT.RRT(**map**, **veh**, **options**) is a rapidly exploring tree navigation object for a region with obstacles defined by the map object **map**.

R = RRT.RRT() as above but internally creates a Vehicle class object and does not support any **map** or **options**. For compatibility with RVC book.

Options

'npoints', N	Number of nodes in the tree
'time', T	Period to simulate dynamic model toward random point
'range', **R**	Specify rectangular bounds

- **R** scalar; X: -**R** to +**R**, Y: -**R** to +**R**
- **R** (1×2); X: -**R**(1) to +**R**(1), Y: -**R**(2) to +**R**(2)
- **R** (1×4); X: **R**(1) to **R**(2), Y: **R**(3) to **R**(4)

'goal', P	Goal position (1×2) or pose (1×3) in workspace
'speed', S	Speed of vehicle [m/s] (default 1)
'steermax', S	Maximum steer angle of vehicle [rad] (default 1.2)

Notes

- Does not (yet) support obstacles, ie. **map** is ignored but must be given.
- 'steermax' selects the range of steering angles that the vehicle will be asked to track. If not given the steering angle range of the vehicle will be used.
- There is no check that the steering range or speed is within the limits of the vehicle object.

Reference

- Robotics, Vision & Control Peter Corke, Springer 2011. p102.

See also

Vehicle

RRT.char

Convert to string

R.**char**() is a string representing the state of the **RRT** object in human-readable form.

invoke the superclass **char**() method

RRT.path

Find a path between two points

x = R.**path**(**start**, **goal**) finds a **path** ($N \times 3$) from state **start** (1×3) to the **goal** (1×3).

P.**path**(**start**, **goal**) as above but plots the **path** in 3D. The nodes are shown as circles and the line segments are blue for forward motion and red for backward motion.

Notes

- The **path** starts at the vertex closest to the **start** state, and ends at the vertex closest to the **goal** state. If the tree is sparse this might be a poor approximation to the desired start and end.

RRT.plan

Create a rapidly exploring tree

R.**plan**(**options**) creates the tree roadmap by driving the vehicle model toward random goal points. The resulting graph is kept within the object.

Options

'goal', P	Goal pose (1×3)
'noprogress'	Don't show the progress bar
'samples'	Show samples

- '.' for each random point x_rand

- 'o' for the nearest point which is added to the tree
- red line for the best path

RRT.plot

Visualize navigation environment

R.**plot**() displays the navigation tree in 3D.

---

rt2tr

Convert rotation and translation to homogeneous transform

TR = **rt2tr**(**R**, **t**) is a homogeneous transformation matrix $(M \times M)$ formed from an orthonormal rotation matrix **R** $(N \times N)$ and a translation vector **t** $(N \times 1)$ where M=N+1.

For a sequence **R** $(N \times N \times K)$ and **t** (kxN) results in a transform sequence (NxNxk).

Notes

- Works for **R** in SO(2) or SO(3)
  - If **R** is 2×2 and **t** is 2×1, then **TR** is 3×3
  - If **R** is 3×3 and **t** is 3×1, then **TR** is 4×4
- The validity of **R** is not checked

See also

t2r, r2t, tr2rt

---

rtbdemo

Robot toolbox demonstrations

Displays popup menu of toolbox demonstration scripts that illustrate:

- homogeneous transformations
- trajectories
- forward kinematics
- inverse kinematics
- robot animation
- inverse dynamics
- forward dynamics

Notes

- The scripts require the user to periodically hit <Enter> in order to move through the explanation.
- Set PAUSE OFF if you want the scripts to run completely automatically.

---

runscript

Run an M-file in interactive fashion

runscript(**fname**, **options**) runs the M-file **fname** and pauses after every executable line in the file until a key is pressed. Comment lines are shown without any delay between lines.

Options

'delay', D	Don't wait for keypress, just delay of D seconds (default 0)
'cdelay', D	Pause of D seconds after each comment line (default 0)
'begin'	Start executing the file after the commnent line %%begin (default true)
'dock'	Cause the figures to be docked when created
'path', P	Look for the file **fname** in the folder P (default .)
'dock'	Dock figures within GUI

Notes

- If not file extension is given in **fname**, .m is assumed.
- If the executable statement has comments immediately afterward (no blank lines) then the pause occurs after those comments are displayed.
- A simple '-' prompt indicates when the script is paused, hit enter.

- If the function cprintf() is in your path, the display is more colorful, you can get this file from MATLAB Central.

See also

eval

---

se2

Create planar translation and rotation transformation

T = se2(**x**, **y**, **theta**) is a 3×3 homogeneous transformation SE(2) representing translation **x** and **y**, and rotation **theta** in the plane.

T = se2(**xy**) as above where **xy**=[**x**,**y**] and rotation is zero

T = se2(**xy**, **theta**) as above where **xy**=[**x**,**y**]

T = se2(**xyt**) as above where **xyt**=[**x**,**y**,**theta**]

See also

trplot2

Sensor

Sensor superclass

An abstact superclass to represent robot navigation sensors.

Methods

plot	plot a line from robot to map feature
display	print the parameters in human readable form
char	convert to string

Properties

robot	The Vehicle object on which the sensor is mounted
map	The Map object representing the landmarks around the robot

Reference

Robotics, Vision & Control, Peter Corke, Springer 2011

See also

EKF, Vehicle, Map

Sensor.Sensor

Sensor object constructor

s = Sensor(**vehicle**, **map**, **options**) is a sensor mounted on the Vehicle object **vehicle** and observing the landmark map **map**.

s = Sensor(**vehicle**, **map**, **R**, **options**) is an instance of the Sensor object mounted on a vehicle represented by the object **vehicle** and observing features in the world represented by the object **map**.

Options

'animate'	animate the action of the laser scanner
'ls', LS	laser scan lines drawn with style ls (default 'r-')
'skip', I	return a valid reading on every I'th call
'fail', [TMIN TMAX	sensor simulates failure between timesteps TMIN and TMAX

Sensor.char

Convert sensor parameters to a string

s = S.char() is a string showing sensor parameters in a compact human readable format.

Sensor.display

Display status of sensor object

S.**display**() displays the state of the sensor object in human-readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Sensor object and the command has no trailing semicolon.

See also

Sensor.char

Sensor.plot

Plot sensor reading

S.**plot**(**J**) draws a line from the robot to the map feature **J**.

Notes

- The line is drawn using the linestyle given by the property ls
- There is a delay given by the property delay

SerialLink

Serial-link robot class

A concrete class that represents a serial-link arm-type robot. The mechanism is described using Denavit-Hartenberg parameters, one set per joint.

Methods

plot	display graphical representation of robot
teach	drive the graphical robot
isspherical	test if robot has spherical wrist
islimit	test if robot at joint limit
isconfig	test robot joint configuration

fkine	forward kinematics
ikine6s	inverse kinematics for 6-axis spherical wrist revolute robot
ikine	inverse kinematics using iterative numerical method
ikine_sym	analytic inverse kinematics obtained symbolically
jacob0	Jacobian matrix in world frame
jacobn	Jacobian matrix in tool frame
maniplty	manipulability
A	link transforms

jtraj	a joint space trajectory

accel	joint acceleration
coriolis	Coriolis joint force
dyn	show dynamic properties of links
fdyn	joint motion
friction	friction force
gravload	gravity joint force
inertia	joint inertia matrix
nofriction	set friction parameters to zero
rne	joint torque/force
payload	add a payload in end-effector frame
perturb	randomly perturb link dynamic parameters

Properties (read/write)

links	vector of Link objects ($1 \times N$)
gravity	direction of gravity [gx gy gz]
base	pose of robot's base (4×4 homog xform)
tool	robot's tool transform, T6 to tool tip (4×4 homog xform)
qlim	joint limits, [qmin qmax] ($N \times 2$)
offset	kinematic joint coordinate offsets ($N \times 1$)
name	name of robot, used for graphical display
manuf	annotation, manufacturer's name
comment	annotation, general comment
plotopt	options for **plot**() method (cell array)
fast	use mex version of RNE. Can only be set true if the mex file exists. Default is true.

Object properties (read only)

n	number of joints
config	joint configuration string, eg. 'RRRRRR'
mdh	kinematic convention boolean (0=DH, 1=MDH)
theta	kinematic: joint angles ($1 \times N$)
d	kinematic: link offsets ($1 \times N$)
a	kinematic: link lengths ($1 \times N$)
alpha	kinematic: link twists ($1 \times N$)

Note

- SerialLink is a reference object.
- SerialLink objects can be used in vectors and arrays

Reference

- Robotics, Vision & Control, Chaps 7-9, P. Corke, Springer 2011.
- Robot, Modeling & Control, M.Spong, S. Hutchinson & M. Vidyasagar, Wiley 2006.

See also

Link, DHFactor

SerialLink.SerialLink

Create a SerialLink robot object

R = SerialLink(**links**, **options**) is a robot object defined by a vector of Link objects.

R = SerialLink(**dh**, **options**) is a robot object with kinematics defined by the matrix **dh** which has one row per joint and each row is [theta d a alpha] and joints are assumed revolute. An optional fifth column sigma indicate revolute (sigma=0, default) or prismatic (sigma=1).

R = SerialLink(**options**) is a null robot object with no links.

R = SerialLink([R1 R2 ...], **options**) concatenate robots, the base of R2 is attached to the tip of R1.

R = SerialLink(**R1**, **options**) is a deep copy of the robot object **R1**, with all the same properties.

Options

'name', NAME	set robot name property to NAME
'comment', COMMENT	set robot comment property to COMMENT
'manufacturer', MANUF	set robot manufacturer property to MANUF
'base', T	set base transformation matrix property to T
'tool', T	set tool transformation matrix property to T
'gravity', G	set gravity vector property to G
'plotopt', P	set default **options** for .plot() to P
'plotopt3d', P	set default **options** for .plot3d() to P
'nofast'	don't use RNE MEX file

Examples

Create a 2-link robot

```
L(1) = Link([ 0     0   a1  0], 'standard');
L(2) = Link([ 0     0   a2  0], 'standard');
twolink = SerialLink(L, 'name', 'two link');
```

Robot objects can be concatenated in two ways

```
R = R1 * R2;
R = SerialLink([R1 R2]);
```

Note

- SerialLink is a reference object, a subclass of Handle object.
- SerialLink objects can be used in vectors and arrays
- When robots are concatenated (either syntax) the intermediate base and tool transforms are removed since general constant transforms cannot be represented in Denavit-Hartenberg notation.

See also

Link, Revolute, Prismatic, SerialLink.plot

SerialLink.A

Evaluate link transform matrices

s = R.**A**(**jlist**, **q**) is a homogeneous transform (4×4) that results from chaining the link transform matrices given in the list JLIST, and the joint variables are taken from the corresponding elements of Q.

For example, the link transform for joint 4 is

```
robot.A(4, q)
```

and for joints 3 through 6 is

```
robot.A([3 4 5 6], q)
```

Notes

- base and tool transforms are not applied.
- JLIST and Q must be the same length.

SerialLink.accel

Manipulator forward dynamics

qdd = R.**accel**(**q**, **qd**, **torque**) is a vector ($N \times 1$) of joint accelerations that result from applying the actuator force/torque to the manipulator robot in state **q** and **qd**. If **q**, **qd**, **torque** are matrices ($K \times N$) then **qdd** is a matrix ($K \times N$) where each row is the acceleration corresponding to the equivalent rows of **q**, **qd**, **torque**.

qdd = R.**accel**(**x**) as above but **x**=[**q**,**qd**,**torque**].

Note

- 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.

References

- Efficient dynamic computer simulation of robotic mechanisms, M. W. Walker and D. E. Orin, ASME Journa of Dynamic Systems, Measurement and Control, vol. 104, no. 3, pp. 205-211, 1982.

See also

SerialLink.rne, SerialLink, ode45

SerialLink.animate

Update a robot animation

R.**animate**(**q**) updates an existing animation for the robot R. This will have been created using R.plot().

Updates graphical instances of this robot in all figures.

Notes

- Called by plot() and plot3d() to actually move the arm models
- Used for Simulink robot animation.

See also

SerialLink.plot

SerialLink.char

Convert to string

s = R.**char**() is a string representation of the robot's kinematic parameters, showing DH parameters, joint structure, comments, gravity vector, base and tool transform.

SerialLink.cinertia

Cartesian inertia matrix

m = R.**cinertia**(**q**) is the $N \times N$ Cartesian (operational space) inertia matrix which relates Cartesian force/torque to Cartesian acceleration at the joint configuration **q**, and N is the number of robot joints.

See also

SerialLink.inertia, SerialLink.rne

SerialLink.coriolis

Coriolis matrix

C = R.**coriolis**(**q**, **qd**) is the Coriolis/centripetal matrix ($N \times N$) for the robot in configuration **q** and velocity **qd**, where N is the number of joints. The product **C*****qd** is the vector of joint force/torque due to velocity coupling. The diagonal elements are due to centripetal effects and the off-diagonal elements are due to Coriolis effects. This matrix is also known as the velocity coupling matrix, since gives the disturbance forces on all joints due to velocity of any joint.

If **q** and **qd** are matrices ($K \times N$), each row is interpretted as a joint state vector, and the result ($N \times N \times K$) is a 3d-matrix where each plane corresponds to a row of **q** and **qd**.

C = R.**coriolis**(**qqd**) as above but the matrix **qqd** ($1 \times 2N$) is [**q** **qd**].

Notes

- Joint friction is also a joint force proportional to velocity but it is eliminated in the computation of this value.
- Computationally slow, involves $N^2/2$ invocations of RNE.

See also

SerialLink.rne

SerialLink.display

Display parameters

R.**display**() displays the robot parameters in human-readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a SerialLink object and the command has no trailing semicolon.

See also

SerialLink.char, SerialLink.dyn

SerialLink.dyn

display inertial properties

R.**dyn**() displays the inertial properties of the **SerialLink** object in a multi-line format. The properties shown are mass, centre of mass, inertia, gear ratio, motor inertia and motor friction.

R.**dyn**(**J**) as above but display parameters for joint **J** only.

See also

Link.dyn

SerialLink.fdyn

Integrate forward dynamics

[**T**,**q**,**qd**] = R.**fdyn**(**T1**, **torqfun**) integrates the dynamics of the robot over the time interval 0 to **T** and returns vectors of time TI, joint position **q** and joint velocity **qd**. The initial joint position and velocity are zero. The torque applied to the joints is computed by the user function **torqfun**:

[**ti**,**q**,**qd**] = R.**fdyn**(**T**, **torqfun**, **q0**, **qd0**) as above but allows the initial joint position and velocity to be specified.

The control torque is computed by a user defined function

```
TAU = TORQFUN(T, Q, QD, ARG1, ARG2, ...)
```

where **q** and **qd** are the manipulator joint coordinate and velocity state respectively, and **T** is the current time.

[**T**,**q**,**qd**] = R.**fdyn**(**T1**, **torqfun**, **q0**, **qd0**, ARG1, ARG2, ...) allows optional arguments to be passed through to the user function.

Note

- This function performs poorly with non-linear joint friction, such as Coulomb friction. The R.nofriction() method can be used to set this friction to zero.
- If **torqfun** is not specified, or is given as 0 or [], then zero torque is applied to the manipulator joints.
- The builtin integration function ode45() is used.

See also

SerialLink.accel, SerialLink.nofriction, SerialLink.rne, ode45

SerialLink.fkine

Forward kinematics

T = R.**fkine**(**q**) is the pose (4×4) of the robot end-effector as a homogeneous transformation for the joint configuration **q** $(1 \times N)$.

If **q** is a matrix $(K \times N)$ the rows are interpretted as the generalized joint coordinates for a sequence of points along a trajectory. **q**(i,j) is the j'th joint parameter for the i'th trajectory point. In this case **T** is a 3d matrix $(4 \times 4 \times K)$ where the last subscript is the index along the path.

[**T**,**all**] = R.**fkine**(**q**) as above but **all** $(4 \times 4 \times N)$ is the pose of the link frames 1 to N, such that **all**(:,:,k) is the pose of link frame k.

Note

- The robot's base or tool transform, if present, are incorporated into the result.
- Joint offsets, if defined, are added to **q** before the forward kinematics are computed.

See also

SerialLink.ikine, SerialLink.ikine6s

SerialLink.friction

Friction force

tau = R.**friction**(**qd**) is the vector of joint **friction** forces/torques for the robot moving with joint velocities **qd**.

The **friction** model includes:

- viscous **friction** which is linear with velocity;
- Coulomb **friction** which is proportional to sign(**qd**).

See also

Link.friction

SerialLink.gencoords

Vector of symbolic generalized coordinates

q = R.**gencoords**() is a vector ($1 \times N$) of symbols [q1 q2 ... qN].

[**q**,**qd**] = R.**gencoords**() as above but **qd** is a vector ($1 \times N$) of symbols [qd1 qd2 ... qdN].

[**q**,**qd**,**qdd**] = R.**gencoords**() as above but **qdd** is a vector ($1 \times N$) of symbols [qdd1 qdd2 ... qddN].

SerialLink.genforces

Vector of symbolic generalized forces

q = R.**genforces**() is a vector ($1 \times N$) of symbols [Q1 Q2 ... QN].

SerialLink.gravload

Gravity loading

taug = R.**gravload**(**q**) is the joint gravity loading for the robot in the joint configuration **q**. Gravitational acceleration is a property of the robot object.

If **q** is a row vector, the result is a row vector of joint torques. If **q** is a matrix, each row is interpreted as a joint configuration vector, and the result is a matrix each row being the corresponding joint torques.

taug = R.**gravload**(**q**, **grav**) is as above but the gravitational acceleration vector **grav** is given explicitly.

See also

SerialLink.rne, SerialLink.itorque, SerialLink.coriolis

SerialLink.ikine

Inverse manipulator kinematics

q = R.**ikine**(**T**) are the joint coordinates corresponding to the robot end-effector pose **T** which is a homogenenous transform.

q = R.**ikine**(**T**, **q0**, **options**) specifies the initial estimate of the joint coordinates.

This method can be used for robots with 6 or more degrees of freedom.

Underactuated robots

For the case where the manipulator has fewer than 6 DOF the solution space has more dimensions than can be spanned by the manipulator joint coordinates.

q = R.**ikine**(**T**, **q0**, **m**, **options**) similar to above but where **m** is a mask vector (1×6) which specifies the Cartesian DOF (in the wrist coordinate frame) that will be ignored in reaching a solution. The mask vector has six elements that correspond to translation in X, Y and Z, and rotation about X, Y and Z respectively. The value should be 0 (for ignore) or 1. The number of non-zero elements should equal the number of manipulator DOF.

For example when using a 3 DOF manipulator rotation orientation might be unimportant in which case **m** = [1 1 1 0 0 0].

For robots with 4 or 5 DOF this method is very difficult to use since orientation is specified by **T** in world coordinates and the achievable orientations are a function of the tool position.

Trajectory operation

In all cases if **T** is $4 \times 4 \times$ **m** it is taken as a homogeneous transform sequence and R.**ikine**() returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is **m** $\times N$ where N is the number of robot joints. The initial estimate of **q** for each time step is taken as the solution from the previous time step.

Options

'pinv'	use pseudo-inverse instead of Jacobian transpose (default)
'ilimit', L	set the maximum iteration count (default 1000)
'tol', **T**	set the tolerance on error norm (default 1e-6)
'alpha', A	set step size gain (default 1)
'varstep'	enable variable step size if pinv is false
'verbose'	show number of iterations for each point
'verbose=2'	show state at each iteration
'plot'	plot iteration state versus time

Notes

- Solution is computed iteratively.
- Solution is sensitive to choice of initial gain. The variable step size logic (enabled by default) does its best to find a balance between speed of convergence and divergence.
- Some experimentation might be required to find the right values of tol, ilimit and alpha.
- The pinv option leads to much faster convergence (default)
- The tolerance is computed on the norm of the error between current and desired tool pose. This norm is computed from distances and angles without any kind of weighting.
- The inverse kinematic solution is generally not unique, and depends on the initial guess **q0** (defaults to 0).
- The default value of **q0** is zero which is a poor choice for most manipulators (eg. puma560, twolink) since it corresponds to a kinematic singularity.
- Such a solution is completely general, though much less efficient than specific inverse kinematic solutions derived symbolically, like ikine6s or ikine3.
- This approach allows a solution to be obtained at a singularity, but the joint angles within the null space are arbitrarily assigned.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.
- Joint limits are not considered in this solution.

See also

SerialLink.fkine, SerialLink.ikinem, tr22angvec, SerialLink.jacob0, SerialLink.ikine6s

---

SerialLink.ikine3

Inverse kinematics for 3-axis robot with no wrist

q = R.**ikine3**(**T**) is the joint coordinates corresponding to the robot end-effector pose **T** represented by the homogenenous transform. This is a analytic solution for a 3-axis robot (such as the first three joints of a robot like the Puma 560).

q = R.**ikine3**(**T**, **config**) as above but specifies the configuration of the arm in the form of a string containing one or more of the configuration codes:

'l'	arm to the left (default)
'r'	arm to the right
'u'	elbow up (default)
'd'	elbow down

Notes

- The same as IKINE6S without the wrist.
- The inverse kinematic solution is generally not unique, and depends on the configuration string.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.

Reference

Inverse kinematics for a PUMA 560 based on the equations by Paul and Zhang From The International Journal of Robotics Research Vol. 5, No. 2, Summer 1986, p. 32-44

Author

Robert Biro with Gary Von McMurray, GTRI/ATRP/IIMB, Georgia Institute of Technology 2/13/95

See also

SerialLink.FKINE, SerialLink.IKINE

SerialLink.ikine6s

Inverse kinematics for 6-axis robot with spherical wrist

q = R.**ikine6s**(**T**) is the joint coordinates corresponding to the robot end-effector pose **T** represented by the homogenenous transform. This is a analytic solution for a 6-axis robot with a spherical wrist (the most common form for industrial robot arms).

q = R.**IKINE6S**(**T**, **config**) as above but specifies the configuration of the arm in the form of a string containing one or more of the configuration codes:

'l'	arm to the left (default)
'r'	arm to the right
'u'	elbow up (default)
'd'	elbow down
'n'	wrist not flipped (default)
'f'	wrist flipped (rotated by 180 deg)

Notes

- Treats a number of specific cases:
  - Robot with no shoulder offset

- Robot with a shoulder offset (has lefty/righty configuration)
- Robot with a shoulder offset and a prismatic third joint (like Stanford arm)
- The Puma 560 arms with should and elbow offsets

- The inverse kinematic solution is generally not unique, and depends on the configuration string.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.
- Only applicable for standard Denavit-Hartenberg parameters

Reference

- Inverse kinematics for a PUMA 560, Paul and Zhang, The International Journal of Robotics Research, Vol. 5, No. 2, Summer 1986, p. 32-44

Author

The Puma560 specific case by Robert Biro with Gary Von McMurray, GTRI/ATRP/IIMB, Georgia Institute of Technology 2/13/95

See also

SerialLink.FKINE, SerialLink.IKINE

SerialLink.ikine_sym

Symbolic inverse kinematics

q = R.**IKINE_SYM**(**n**, **options**) is a vector (**n** $\times$ 1) of symbolic expressions for the inverse kinematic solution of the **SerialLink** object ROBOT. The solution is in terms of the desired end-point pose of the robot which is represented by the symbolic matrix and elements

```
nx ox ax px
ny oy ay py
nz oz az pz
```

Elements of **q** may be cell arrays that contain multiple expressions, representing different possible joint configurations.

n can have only specific values:

- 2 solve for translation px and py
- 3 solve for translation px, py and pz
- 6 solve for translation and orientation

Options

'file', F	Write the solution to an m-file named F

Notes

- This code is experimental and has a lot of diagnostic prints
- Based on the classical approach using Pieper's method

SerialLink.ikinem

Inverse manipulator kinematics by minimization

q = R.**ikinem**(**T**) is the joint coordinates corresponding to the robot end-effector pose **T** which is a homogenenous transform.

q = R.**ikinem**(**T**, **q0**, **options**) specifies the initial estimate of the joint coordinates.

In all cases if **T** is $4 \times 4 \times M$ it is taken as a homogeneous transform sequence and R.**ikinem**() returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is $M \times N$ where N is the number of robot joints. The initial estimate of **q** for each time step is taken as the solution from the previous time step.

Options

'pweight', P	weighting on position error norm compared to rotation error (default 1)
'stiffness', S	Stiffness used to impose a smoothness contraint on joint angles, useful when N is large (default 0)
'qlimits'	Enforce joint limits
'ilimit', L	Iteration limit (default 1000)
'nolm'	Disable Levenberg-Marquadt

Notes

- PROTOTYPE CODE UNDER DEVELOPMENT, intended to do numerical inverse kinematics with joint limits
- The inverse kinematic solution is generally not unique, and depends on the initial guess **q0** (defaults to 0).
- The function to be minimized is highly nonlinear and the solution is often trapped in a local minimum, adjust **q0** if this happens.
- The default value of **q0** is zero which is a poor choice for most manipulators (eg. puma560, twolink) since it corresponds to a kinematic singularity.

- Such a solution is completely general, though much less efficient than specific inverse kinematic solutions derived symbolically, like ikine6s or ikine3.% - Uses Levenberg-Marquadt minimizer LMFsolve if it can be found, if 'nolm' is not given, and 'qlimits' false
- The error function to be minimized is computed on the norm of the error between current and desired tool pose. This norm is computed from distances and angles and 'pweight' can be used to scale the position error norm to be congruent with rotation error norm.
- This approach allows a solution to obtained at a singularity, but the joint angles within the null space are arbitrarily assigned.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.
- Joint limits become explicit contraints if 'qlimits' is set.

See also

fminsearch, fmincon, SerialLink.fkine, SerialLink.ikine, tr2angvec

---

SerialLink.inertia

Manipulator inertia matrix

i = R.**inertia**(**q**) is the symmetric joint **inertia** matrix $(N \times N)$ which relates joint torque to joint acceleration for the robot at joint configuration **q**.

If **q** is a matrix $(K \times N)$, each row is interpretted as a joint state vector, and the result is a 3d-matrix $(N \times N \times K)$ where each plane corresponds to the **inertia** for the corresponding row of **q**.

Notes

- The diagonal elements **i**(J,J) are the **inertia** seen by joint actuator J.
- The off-diagonal elements **i**(J,K) are coupling inertias that relate acceleration on joint J to force/torque on joint K.
- The diagonal terms include the motor **inertia** reflected through the gear ratio.

See also

SerialLink.RNE, SerialLink.CINERTIA, SerialLink.ITORQUE

SerialLink.isconfig

Test for particular joint configuration

R.**isconfig**(**s**) is true if the robot has the joint configuration string given by the string **s**.

Example:

```
robot.isconfig('RRRRRR');
```

See also

SerialLink.config

SerialLink.islimit

Joint limit test

v = R.**islimit**(**q**) is a vector of boolean values, one per joint, false (0) if **q**(i) is within the joint limits, else true (1).

Notes

- Joint limits are purely advisory and are not used in any other function. Just seemed like a useful thing to include...

See also

Link.islimit

SerialLink.isspherical

Test for spherical wrist

R.**isspherical**() is true if the robot has a spherical wrist, that is, the last 3 axes are revolute and their axes intersect at a point.

See also

SerialLink.ikine6s

SerialLink.issym

Check if Link or SerialLink object is a symbolic model

res = L.**issym**() is true if the Link L has symbolic parameters.

res = R.**issym**() is true if the **SerialLink** manipulator R has symbolic parameters

Authors

Jrn Malzahn 2012 RST, Technische Universitt Dortmund, Germany http://www.rst.e-technik.tu-dortmund.de

---

SerialLink.itorque

Inertia torque

taui = R.**itorque**(**q**, **qdd**) is the inertia force/torque vector $(1 \times N)$ at the specified joint configuration **q** $(1 \times N)$ and acceleration **qdd** $(1 \times N)$, that is, **taui** = INERTIA(**q**)***qdd**.

If **q** and **qdd** are matrices $(K \times N)$, each row is interpretted as a joint state vector, and the result is a matrix $(K \times N)$ where each row is the corresponding joint torques.

Note

- If the robot model contains non-zero motor inertia then this will included in the result.

See also

SerialLink.rne, SerialLink.inertia

SerialLink.jacob0

Jacobian in world coordinates

j0 = R.**jacob0**(**q**, **options**) is the Jacobian matrix $(6 \times N)$ for the robot in pose **q** $(1 \times N)$. The manipulator Jacobian matrix maps joint velocity to end-effector spatial velocity V = **j0***QD expressed in the world-coordinate frame.

Options

'rpy'	Compute analytical Jacobian with rotation rate in terms of roll-pitch-yaw angles
'eul'	Compute analytical Jacobian with rotation rates in terms of Euler angles
'trans'	Return translational submatrix of Jacobian
'rot'	Return rotational submatrix of Jacobian

Note

- The Jacobian is computed in the world frame and transformed to the end-effector frame.
- The default Jacobian returned is often referred to as the geometric Jacobian, as opposed to the analytical Jacobian.

See also

SerialLink.jacobn, jsingu, deltatr, tr2delta, jsingu

SerialLink.jacob_dot

Derivative of Jacobian

jdq = R.**jacob_dot**(**q**, **qd**) is the product (6×1) of the derivative of the Jacobian (in the world frame) and the joint rates.

Notes

- Useful for operational space control XDD = J(**q**)QDD + JDOT(**q**)**qd**
- Written as per the text and not very efficient.

References

- Fundamentals of Robotics Mechanical Systems (2nd ed) J. Angleles, Springer 2003.

See also

SerialLink.jacob0, diff2tr, tr2diff

SerialLink.jacobn

Jacobian in end-effector frame

jn = R.**jacobn**(**q**, **options**) is the Jacobian matrix ($6 \times N$) for the robot in pose **q**. The manipulator Jacobian matrix maps joint velocity to end-effector spatial velocity V = **jn***QD in the end-effector frame.

Options

'trans'	Return translational submatrix of Jacobian
'rot'	Return rotational submatrix of Jacobian

Notes

- This Jacobian is often referred to as the geometric Jacobian.

Reference

Differential Kinematic Control Equations for Simple Manipulators, Paul, Shimano, Mayer, IEEE SMC 11(6) 1981, pp. 456-460

See also

SerialLink.jacob0, jsingu, delta2tr, tr2delta

SerialLink.jtraj

Joint space trajectory

q = R.**jtraj**(**T1**, **t2**, **k**) is a joint space trajectory ($\mathbf{k} \times N$) where the joint coordinates reflect motion from end-effector pose **T1** to **t2** in **k** steps with default zero boundary conditions for velocity and acceleration. The trajectory **q** has one row per time step, and one column per joint, where N is the number of robot joints.

Note

- Requires solution of inverse kinematics. R.ikine6s() is used if appropriate, else R.ikine(). Additional trailing arguments to R.**jtraj**() are passed as trailing arguments to these functions.

See also

jtraj, SerialLink.ikine, SerialLink.ikine6s

SerialLink.maniplty

Manipulability measure

m = R.**maniplty**(**q**, **options**) is the manipulability index measure for the robot at the joint configuration **q**. It indicates dexterity, that is, how isotropic the robot's motion is with respect to the 6 degrees of Cartesian motion. The measure is high when the manipulator is capable of equal motion in all directions and low when the manipulator is close to a singularity.

If **q** is a matrix (**m** $\times N$) then **m** (**m** $\times$ 1) is a vector of manipulability indices for each pose specified by a row of **q**.

[**m**,**ci**] = R.**maniplty**(**q**, **options**) as above, but for the case of the Asada measure returns the Cartesian inertia matrix **ci**.

Two measures can be selected:

- Yoshikawa's manipulability measure is based on the shape of the velocity ellipsoid and depends only on kinematic parameters.
- Asada's manipulability measure is based on the shape of the acceleration ellipsoid which in turn is a function of the Cartesian inertia matrix and the dynamic parameters. The scalar measure computed here is the ratio of the smallest/largest ellipsoid axis. Ideally the ellipsoid would be spherical, giving a ratio of 1, but in practice will be less than 1.

Options

'T'	manipulability for transational motion only (default)
'R'	manipulability for rotational motion only
'all'	manipulability for all motions
'dof', D	D is a vector (1×6) with non-zero elements if the corresponding DOF is to be included for manipulability
'yoshikawa'	use Yoshikawa algorithm (default)
'asada'	use Asada algorithm

Notes

- The 'all' option includes rotational and translational dexterity, but this involves adding different units. It can be more useful to look at the translational and rotational manipulability separately.
- Examples in the RVC book can be replicated by using the 'all' option

References

- Analysis and control of robot manipulators with redundancy, T. Yoshikawa, Robotics Research: The First International Symposium (**m**. Brady and R. Paul, eds.), pp. 735-747, The MIT press, 1984.
- A geometrical representation of manipulator dynamics and its application to arm design, H. Asada, Journal of Dynamic Systems, Measurement, and Control, vol. 105, p. 131, 1983.

See also

SerialLink.inertia, SerialLink.jacob0

SerialLink.mtimes

Concatenate robots

R = R1 * R2 is a robot object that is equivalent to mechanically attaching robot R2 to the end of robot R1.

Notes

- If R1 has a tool transform or R2 has a base transform these are discarded since DH convention does not allow for arbitrary intermediate transformations.

SerialLink.nofriction

Remove friction

rnf = R.**nofriction**() is a robot object with the same parameters as R but with non-linear (Coulomb) friction coefficients set to zero.

rnf = R.**nofriction**('all') as above but all friction coefficients set to zero.

rnf = R.**nofriction**('viscous') as above but only viscous friction coefficients are set to zero.

Notes

- Non-linear (Coulomb) friction can cause numerical problems when integrating the equations of motion (R.fdyn).
- The resulting robot object has its name string prefixed with 'NF/'.

See also

SerialLink.fdyn, Link.nofriction

SerialLink.payload

Add payload mass

R.**payload**(**m**, **p**) adds a **payload** with point mass **m** at position **p** in the end-effector coordinate frame.

See also

SerialLink.rne, SerialLink.gravload

SerialLink.perturb

Perturb robot parameters

rp = R.**perturb**(**p**) is a new robot object in which the dynamic parameters (link mass and inertia) have been perturbed. The perturbation is multiplicative so that values are multiplied by random numbers in the interval (1-**p**) to (1+**p**). The name string of the perturbed robot is prefixed by '**p/**'.

Useful for investigating the robustness of various model-based control schemes. For example to vary parameters in the range +/- 10 percent is:

```
r2 = p560.perturb(0.1);
```

SerialLink.plot

Graphical display and animation

R.**plot**(**q**, **options**) displays a graphical animation of a robot based on the kinematic model. A stick figure polyline joins the origins of the link coordinate frames. The robot is displayed at the joint angle **q** $(1 \times N)$, or if a matrix $(M \times N)$ it is animated as the robot moves along the M-point trajectory.

Options

'workspace', W	Size of robot 3D workspace, W = [xmn, xmx ymn ymx zmn zmx]
'floorlevel', L	Z-coordinate of floor (default -1)
'delay', D	Delay betwen frames for animation (s)
'fps', fps	Number of frames per second for display, inverse of 'delay' option
'[no]loop'	Loop over the trajectory forever
'[no]raise'	Autoraise the figure
'movie', M	Save frames as files in the folder M
'trail', L	Draw a line recording the tip path, with line style L
'scale', S	Annotation scale factor
'zoom', Z	Reduce size of auto-computed workspace by Z, makes robot look bigger
'ortho'	Orthographic view
'perspective'	Perspective view (default)
'view', V	Specify view V='x', 'y', 'top' or [az el] for side elevations, plan view, or general view by azimuth and elevation angle.
'[no]shading'	Enable Gouraud shading (default true)
'lightpos', L	Position of the light source (default [0 0 20])
'[no]name'	Display the robot's name
'[no]wrist'	Enable display of wrist coordinate frame
'xyz'	Wrist axis label is XYZ
'noa'	Wrist axis label is NOA
'[no]arrow'	Display wrist frame with 3D arrows
'[no]tiles'	Enable tiled floor (default true)
'tilesize', S	Side length of square tiles on the floor (default 0.2)
'tile1color', C	Color of even tiles [r g b] (default [0.5 1 0.5] light green)
'tile2color', C	Color of odd tiles [r g b] (default [1 1 1] white)
'[no]shadow'	Enable display of shadow (default true)
'shadowcolor', C	Colorspec of shadow, [r g b]
'shadowwidth', W	Width of shadow line (default 6)
'[no]jaxes'	Enable display of joint axes (default true)
'[no]joints'	Enable display of joints
'jointcolor', C	Colorspec for joint cylinders (default [0.7 0 0])
'jointdiam', D	Diameter of joint cylinder in scale units (default 5)
'linkcolor', C	Colorspec of links (default 'b')
'[no]base'	Enable display of base 'pedestal'
'basecolor', C	Color of base (default 'k')
'basewidth', W	Width of base (default 3)

The **options** come from 3 sources and are processed in order:

- Cell array of **options** returned by the function PLOTBOTOPT (if it exists)
- Cell array of **options** given by the 'plotopt' option when creating the SerialLink object.
- List of arguments in the command line.

Many boolean **options** can be enabled or disabled with the 'no' prefix. The various option sources can toggle an option, the last value is taken.

Graphical annotations and options

The robot is displayed as a basic stick figure robot with annotations such as:

- shadow on the floor
- XYZ wrist axes and labels
- joint cylinders and axes

which are controlled by **options**.

The size of the annotations is determined using a simple heuristic from the workspace dimensions. This dimension can be changed by setting the multiplicative scale factor using the 'mag' option.

Figure behaviour

- If no figure exists one will be created and the robot drawn in it.
- If no robot of this name is currently displayed then a robot will be drawn in the current figure. If hold is enabled (hold on) then the robot will be added to the current figure.
- If the robot already exists then that graphical model will be found and moved.

Multiple views of the same robot

If one or more plots of this robot already exist then these will all be moved according to the argument **q**. All robots in all windows with the same name will be moved.

Create a robot in figure 1

```
figure(1)
p560.plot(qz);
```

Create a robot in figure 2

```
figure(2)
p560.plot(qz);
```

Now move both robots

```
p560.plot(qn)
```

Multiple robots in the same figure

Multiple robots can be displayed in the same **plot**, by using "hold on" before calls to robot.**plot**().

Create a robot in figure 1

```
figure(1)
p560.plot(qz);
```

Make a clone of the robot named bob

```
bob = SerialLink(p560, 'name', 'bob');
```

Draw bob in this figure

```
hold on
bob.plot(qn)
```

To animate both robots so they move together:

```
qtg = jtraj(qr, qz, 100);
for q=qtg'

p560.plot(q');
bob.plot(q');

end
```

Making an animation movie

- The 'movie' **options** saves frames as files NNNN.png into the specified folder
- The specified folder will be created
- To convert frames to a movie use a command like:

```
ffmpeg -r 10 -i %04d.png out.avi
```

Notes

- The **options** are processed when the figure is first drawn, to make different **options** come into effect it is neccessary to clear the figure.
- The link segments do not neccessarily represent the links of the robot, they are a pipe network that joins the origins of successive link coordinate frames.
- Delay betwen frames can be eliminated by setting option 'delay', 0 or 'fps', Inf.
- By default a quite detailed **plot** is generated, but turning off labels, axes, shadows etc. will speed things up.
- Each graphical robot object is tagged by the robot's name and has UserData that holds graphical handles and the handle of the robot object.
- The graphical state holds the last joint configuration
- The size of the **plot** volume is determined by a heuristic for an all-revolute robot. If a prismatic joint is present the 'workspace' option is required. The 'zoom' option can reduce the size of this workspace.

See also

SerialLink.plot3d, plotbotopt, SerialLink.animate, SerialLink.teach, SerialLink.fkine

SerialLink.plot3d

Graphical display and animation of solid model robot

R.**plot3d**(**q**, **options**) displays and animates a solid model of the robot. The robot is displayed at the joint angle **q** ($1 \times N$), or if a matrix ($M \times N$) it is animated as the robot moves along the M-point trajectory.

Options

'color', C	A cell array of color names, one per link. These are mapped to RGB using colorname(). If not given, colors come from the axis ColorOrder property.
'alpha', A	Set alpha for all links, 0 is transparant, 1 is opaque (default 1)
'path', P	Overide path to folder containing STL model files
'workspace', W	Size of robot 3D workspace, W = [xmn, xmx ymn ymx zmn zmx]
'floorlevel', L	Z-coordinate of floor (default -1)
'delay', D	Delay betwen frames for animation (s)
'fps', fps	Number of frames per second for display, inverse of 'delay' option
'[no]loop'	Loop over the trajectory forever
'[no]raise'	Autoraise the figure
'movie', M	Save frames as files in the folder M
'scale', S	Annotation scale factor
'ortho'	Orthographic view (default)
'perspective'	Perspective view
'[no]wrist'	Enable display of wrist coordinate frame
'xyz'	Wrist axis label is XYZ
'noa'	Wrist axis label is NOA
'[no]arrow'	Display wrist frame with 3D arrows
'[no]tiles'	Enable tiled floor (default true)
'tilesize', S	Side length of square tiles on the floor (default 0.2)
'tile1color', C	Color of even tiles [r g b] (default [0.5 1 0.5] light green)
'tile2color', C	Color of odd tiles [r g b] (default [1 1 1] white)
'[no]jaxes'	Enable display of joint axes (default true)
'[no]joints'	Enable display of joints
'[no]base'	Enable display of base shape

Notes

- Solid models of the robot links are required as STL ascii format files, with extensions .stl
- Suitable STL files can be found in the package ARTE: A ROBOTICS TOOLBOX FOR EDUCATION by Arturo Gil
- The root of the solid models is an installation of ARTE with an empty file called arte.m at the top level
- Each STL model is called 'linkN'.stl where N is the link number 0 to N

- The specific folder to use comes from the SerialLink.model3d property
- The path of the folder containing the STL files can be specified using the 'path' option
- The height of the floor is set in decreasing priority order by:
  - 'workspace' option, the fifth element of the passed vector
  - 'floorlevel' option
  - the lowest z-coordinate in the link1.stl object

Authors

- Peter Corke, based on existing code for plot()
- Bryan Moutrie, demo code on the Google Group for connecting ARTE and RTB
- Don Riley, function rndread() extracted from cad2matdemo (MATLAB File Exchange)

See also

SerialLink.plot, plotbotopt3d, SerialLink.animate, SerialLink.teach, SerialLink.fkine

---

SerialLink.rndread

CAD STL ASCII files, which most CAD programs can export.

Used to create Matlab patches of CAD 3D data. Returns a vertex list and face list, for Matlab patch command.

filename = 'hook.stl'; % Example file.

SerialLink.rne

Inverse dynamics

tau = R.**rne**(**q**, **qd**, **qdd**) is the joint torque required for the robot R to achieve the specified joint position **q**, velocity **qd** and acceleration **qdd**.

tau = R.**rne**(**q**, **qd**, **qdd**, **grav**) as above but overriding the gravitational acceleration vector in the robot object R.

tau = R.**rne**(**q**, **qd**, **qdd**, **grav**, **fext**) as above but specifying a wrench acting on the end of the manipulator which is a 6-vector [Fx Fy Fz Mx My Mz].

tau = R.rne(**x**) as above where **x=[q,qd,qdd]**.

tau = R.rne(**x**, **grav**) as above but overriding the gravitational acceleration vector in the robot object R.

tau = R.rne(**x**, **grav**, **fext**) as above but specifying a wrench acting on the end of the manipulator which is a 6-vector [Fx Fy Fz Mx My Mz].

[**tau**,**wbase**] = R.rne(**x**, **grav**, **fext**) as above but the extra output is the wrench on the base.

If **q**,**qd** and **qdd** $(M \times N)$, or **x** $(M \times 3N)$ are matrices with M rows representing a trajectory then **tau** $(M \times N)$ is a matrix with rows corresponding to each trajectory step.

Fast rne

This algorithm is relatively slow, and a MEX file can provide better performance. The MEX file is executed if:

- the robot is not symbolic, and
- the SerialLink property fast is true, and
- the MEX file exists.

Notes

- The robot base transform is ignored.
- Currently the MEX-file version does not compute **wbase**.
- The torque computed contains a contribution due to armature inertia and joint friction.
- See the README file in the mex folder for details on how to configure MEX-file operation.
- The M-file is a wrapper which calls either RNE_DH or RNE_MDH depending on the kinematic conventions used by the robot object.

See also

SerialLink.accel, SerialLink.gravload, SerialLink.inertia

SerialLink.teach

Graphical teach pendant

R.**teach**(**q**) drive a graphical robot by means of a graphical slider panel. If no graphical robot exists one is created in a new window. Otherwise all current instances of the graphical robot are driven. The robots are set to the initial joint angles **q**.

R.**teach**(**q**, **options**) as above but with options.

R.**teach**(**options**) as above but with options and the initial joint angles are taken from the pose of an existing graphical robot, or if that doesn't exist then zero.

Options

'eul'	Display tool orientation in Euler angles
'rpy'	Display tool orientation in roll/pitch/yaw angles
'approach'	Display tool orientation as approach vector (z-axis)
'radians'	Display angles in radians (default degrees)
'callback', C	Set a callback function

GUI

- The record button invokes the user specified callback function, and is passed the joint coordinate vector.
- The Quit button destroys the **teach** window.

Notes

- The slider limits are derived from the joint limit properties. If not set then for
  - a revolute joint they are assumed to be [-pi, +pi]
  - a prismatic joint they are assumed unknown and an error occurs.

See also

SerialLink.plot

SerialLink.trchain

Display kinematic parameters as a chain of 3D transforms

s = R.**TRCHAIN**(**options**) is a string of elementary transforms that describe the kinematics of the seriallink robot arm. The string **s** comprises a number of tokens of the form X(ARG) where X is one of Tx, Ty, Tz, Rx, Ry, or Rz. ARG is a joint variable, or a constant angle or length dimension.

For example:

```
>> mdl_puma560
>> p560.trchain
ans =
Rz(q1)Rx(90)Rz(q2)Tx(0.431800)Rz(q3)Tz(0.150050)Tx(0.020300)Rx(-90)
Rz(q4)Tz(0.431800)Rx(90)Rz(q5)Rx(-90)Rz(q6)
```

Options

'[no]deg'	Express angles in degrees rather than radians (default deg)
'sym'	Replace length parameters by symbolic values L1, L2 etc.

See also

trchain, trotx, troty, trotz, transl

skew

Create skew-symmetric matrix

s = **skew**(**v**) is a **skew**-symmetric matrix formed from **v** (3×1).

```
| 0   -vz  vy|
| vz   0  -vx|
|-vy   vx  0 |
```

See also

vex

startup_rtb

Initialize MATLAB paths for Robotics Toolbox

Adds demos, examples to the MATLAB path, and adds also to Java class path.

---

t2r

Return rotational submatrix of a homogeneous transformation

R = **t2r**(**T**) is the orthonormal rotation matrix component of homogeneous transformation matrix **T**:

Notes

- Works for **T** in SE(2) or SE(3)
  - If **T** is 4×4, then **R** is 3×3.
  - If **T** is 3×3, then **R** is 2×2.
- The validity of rotational part is not checked
- For a homogeneous transform sequence returns a rotation matrix sequence

See also

r2t, tr2rt, rt2tr

---

tb_optparse

Standard option parser for Toolbox functions

[**optout**,**args**] = **tb_optparse**(**opt**, **arglist**) is a generalized option parser for Toolbox functions. It supports options that have an assigned value, boolean or enumeration types (string or int).

The software pattern is:

```
function(a, b, c, varargin)
opt.foo = true;
opt.bar = false;
opt.blah = [];
opt.choose = {'this', 'that', 'other'};
opt.select = {'#no', '#yes'};
opt = tb_optparse(opt, varargin);
```

Optional arguments to the function behave as follows:

'foo'	sets opt.foo <- true
'nobar'	sets opt.foo <- false
'blah', 3	sets opt.blah <- 3
'blah', x,y	sets opt.blah <- x,y
'that'	sets opt.choose <- 'that'
'yes'	sets opt.select <- 2 (the second element)

and can be given in any combination.

If neither of 'this', 'that' or 'other' are specified then opt.choose <- 'this'. Alternatively if:

```
opt.choose = {[], 'this', 'that', 'other'};
```

then if neither of 'this', 'that' or 'other' are specified then opt.choose <- []

If neither of 'no' or 'yes' are specified then opt.select <- 1.

Note:

- That the enumerator names must be distinct from the field names.
- That only one value can be assigned to a field, if multiple values

```
are required they must be converted to a cell array.
```

- To match an option that starts with a digit, prefix it with 'd_', so the field 'd_3d' matches the option '3d'.

The allowable options are specified by the names of the fields in the structure opt. By default if an option is given that is not a field of opt an error is declared.

Sometimes it is useful to collect the unassigned options and this can be achieved using a second output argument

```
[opt,arglist] = tb_optparse(opt, varargin);
```

which is a cell array of all unassigned arguments in the order given in varargin.

The return structure is automatically populated with fields: verbose and debug. The following options are automatically parsed:

'verbose'	sets opt.verbose <- true
'verbose=2'	sets opt.verbose <- 2 (very verbose)
'verbose=3'	sets opt.verbose <- 3 (extremeley verbose)
'verbose=4'	sets opt.verbose <- 4 (ridiculously verbose)
'debug', N	sets opt.debug <- N
'setopt', S	sets opt <- S
'showopt'	displays opt and arglist

tpoly

Generate scalar polynomial trajectory

[s,sd,sdd] = **tpoly**(**s0**, **sf**, **m**) is a scalar trajectory (**m** $\times$ 1) that varies smoothly from **s0** to **sf** in **m** steps using a quintic (5th order) polynomial. Velocity and acceleration can be optionally returned as **sd** (**m** $\times$ 1) and **sdd** (**m** $\times$ 1).

[s,sd,sdd] = **tpoly**(**s0**, **sf**, **T**) as above but specifies the trajectory in terms of the length of the time vector **T** (**m** $\times$ 1).

Notes

- If no output arguments are specified **s**, **sd**, and **sdd** are plotted.

---

tr2angvec

Convert rotation matrix to angle-vector form

[**theta**,**v**] = **tr2angvec**(**R**) converts an orthonormal rotation matrix **R** into a rotation of **theta** (1 $\times$ 1) about the axis **v** (1 $\times$ 3).

[**theta**,**v**] = **tr2angvec**(**T**) as above but uses the rotational part of the homogeneous transform **T**.

If **R** ($3 \times 3 \times K$) or **T** ($4 \times 4 \times K$) represent a sequence then **theta** ($K \times 1$)is a vector of angles for corresponding elements of the sequence and **v** ($K \times 3$) are the corresponding axes, one per row.

Notes

- If no output arguments are specified the result is displayed.

See also

angvec2r, angvec2tr

---

tr2delta

Convert homogeneous transform to differential motion

d = **tr2delta**(**T0**, **T1**) is the differential motion (6×1) corresponding to infinitessimal motion from pose **T0** to **T1** which are homogeneous transformations. **d**=(dx, dy, dz, dRx, dRy, dRz) and is an approximation to the average spatial velocity multiplied by time.

d = **tr2delta**(**T**) is the differential motion corresponding to the infinitessimal relative pose **T** expressed as a homogeneous transformation.

Notes

- **d** is only an approximation to the motion **T**, and assumes that **T0**   **T1** or **T**   eye(4,4).

See also

delta2tr, skew

---

tr2eul

Convert homogeneous transform to Euler angles

eul = **tr2eul**(**T**, **options**) are the ZYZ Euler angles expressed as a row vector corresponding to the rotational part of a homogeneous transform **T**. The 3 angles **eul**=[PHI,THETA,PSI] correspond to sequential rotations about the Z, Y and Z axes respectively.

eul = **tr2eul**(**R**, **options**) are the ZYZ Euler angles expressed as a row vector corresponding to the orthonormal rotation matrix **R**.

If **R** or **T** represents a trajectory (has 3 dimensions), then each row of **eul** corresponds to a step of the trajectory.

Options

'deg'	Compute angles in degrees (radians default)
'flip'	Choose first Euler angle to be in quadrant 2 or 3.

Notes

- There is a singularity for the case where THETA=0 in which case PHI is arbitrarily set to zero and PSI is the sum (PHI+PSI).

See also

eul2tr, tr2rpy

---

tr2jac

Jacobian for differential motion

J = **tr2jac**(**T**) is a Jacobian matrix (6×6) that maps spatial velocity or differential motion from the world frame to the frame represented by the homogeneous transform **T**.

See also

wtrans, tr2delta, delta2tr

---

tr2rpy

Convert a homogeneous transform to roll-pitch-yaw angles

rpy = **tr2rpy**(**T**, **options**) are the roll-pitch-yaw angles expressed as a row vector corresponding to the rotation part of a homogeneous transform **T**. The 3 angles **rpy**=[R,P,Y] correspond to sequential rotations about the X, Y and Z axes respectively.

rpy = **tr2rpy**(**R**, **options**) are the roll-pitch-yaw angles expressed as a row vector corresponding to the orthonormal rotation matrix **R**.

If **R** or **T** represents a trajectory (has 3 dimensions), then each row of **rpy** corresponds to a step of the trajectory.

Options

'deg'	Compute angles in degrees (radians default)
'zyx'	Return solution for sequential rotations about Z, Y, X axes (Paul book)

Notes

- There is a singularity for the case where P=pi/2 in which case **R** is arbitrarily set to zero and Y is the sum (**R**+Y).
- Note that textbooks (Paul, Spong) use the rotation order ZYX.

See also

rpy2tr, tr2eul

---

tr2rt

Convert homogeneous transform to rotation and translation

[**R**,**t**] = **tr2rt**(**TR**) split a homogeneous transformation matrix ($N \times N$) into an orthonormal rotation matrix **R** ($M \times M$) and a translation vector **t** ($M \times 1$), where N=M+1.

A homogeneous transform sequence **TR** ($N \times N \times K$) is split into rotation matrix sequence **R** ($M \times M \times K$) and a translation sequence **t** ($K \times M$).

Notes

- Works for **TR** in SE(2) or SE(3)
  - If **TR** is 4×4, then **R** is 3×3 and T is 3×1.
  - If **TR** is 3×3, then **R** is 2×2 and T is 2×1.
- The validity of **R** is not checked.

See also

rt2tr, r2t, t2r

---

tranimate

Animate a coordinate frame

tranimate(**p1**, **p2**, **options**) animates a 3D coordinate frame moving from pose **p1** to pose **p2**. Poses **p1** and **p2** can be represented by:

- homogeneous transformation matrices (4×4)
- orthonormal rotation matrices (3×3)
- Quaternion

tranimate(**p**, **options**) animates a coordinate frame moving from the identity pose to the pose **p** represented by any of the types listed above.

tranimate(**pseq**, **options**) animates a trajectory, where **pseq** is any of

- homogeneous transformation matrix sequence ($4 \times 4 \times N$)
- orthonormal rotation matrix sequence ($3 \times 3 \times N$)
- Quaternion vector ($N \times 1$)

Options

'fps', fps	Number of frames per second to display (default 10)
'nsteps', n	The number of steps along the path (default 50)
'axis', A	Axis bounds [xmin, xmax, ymin, ymax, zmin, zmax]
'movie', M	Save frames as files in the folder M

Notes

- The 'movie' options saves frames as files NNNN.png.
- When using 'movie' option ensure that the window is fully visible.
- To convert frames to a movie use a command like:

```
ffmpeg -r 10 -i %04d.png out.avi
```

See also

trplot

transl

Create translational transform

T = **transl**(**x**, **y**, **z**) is an SE(3) homogeneous transform (4×4)representing a pure translation.

T = **transl**(**p**) is an SE(3) homogeneous transform representing a translation or point **p**=[**x**,**y**,**z**]. If **p** $(M \times 3)$ it represents a sequence and **T** $(4 \times 4 \times M)$ is a sequence of homogenous transforms such that **T**(:,:,i) corresponds to the i'th row of **p**.

p = **transl**(**T**) is the translational part of a homogeneous transform **T** as a 3-element column vector. If **T** $(4 \times 4 \times M)$ is a homogeneous transform sequence the rows of **p** $(M \times 3)$ are the translational component of the corresponding transform in the sequence.

[**x**,**y**,**z**] = **transl**(**T**) is the translational part of a homogeneous transform **T** as three components. If **T** $(4 \times 4 \times M)$ is a homogeneous transform sequence then **x**,**y**,**z** $(1 \times M)$ are the translational components of the corresponding transform in the sequence.

Notes

- Somewhat unusually this function performs a function and its inverse. An historical anomaly.

See also

ctraj

---

transl2

Create an SE2 translational transform

T = **transl2**(**x**, **y**) is an SE2 homogeneous transform (3×3)representing a pure translation.

T = **transl2**(**p**) is a homogeneous transform representing a translation or point **p**=[**x**,**y**]. If **p** $(M \times 2)$ it represents a sequence and **T** $(3 \times 3 \times M)$ is a sequence of homogenous transforms such that **T**(:,:,i) corresponds to the i'th row of **p**.

p = **transl2**(**T**) is the translational part of a homogeneous transform as a 2-element column vector. If **T** $(3 \times 3 \times M)$ is a homogeneous transform sequence the rows of **p** $(M \times 2)$ are the translational component of the corresponding transform in the sequence.

Notes

- Somewhat unusually this function performs a function and its inverse. An historical anomaly.

See also

transl

---

trchain

Chain 3D transforms from string

T = **trchain**(**s**, **q**) is a homogeneous transform (4×4) that results from compounding a number of elementary transformations defined by the string **s**. The string **s** comprises a number of tokens of the form X(ARG) where X is one of Tx, Ty, Tz, Rx, Ry, or Rz. ARG is the name of a variable in main workspace or qJ where J is an integer in the range 1 to N that selects the variable from the Jth column of the vector **q** ($1 \times N$).

For example:

```
trchain('Rx(q1)Tx(a1)Ry(q2)Ty(a3)Rz(q3) Rx(pi/2)', [1 2 3])
```

is equivalent to computing:

```
trotx(1) * transl(a1,0,0) * troty(2) * transl(0,a3,0) * trotz(3)
```

Notes

- The string can contain spaces between elements or on either side of ARG.
- Works for symbolic variables in the workspace and/or passed in via the vector **q**.
- For symbolic operations that involve use of the value pi, make sure you define it first in the workspace: pi = sym('pi');

See also

trchain2, trotx, troty, trotz, transl

---

trchain2

Chain 2D transforms from string

T = **trchain2**(**s**, **q**) is a homogeneous transform (3×3) that results from compounding a number of elementary transformations defined by the string **s**. The string **s** comprises a number of tokens of the form X(ARG) where X is one of Tx, Ty or R. ARG is the name of a variable in main workspace or qJ where J is an integer in the range 1 to N that selects the variable from the Jth column of the vector **q** ($1 \times N$).

For example:

```
trchain('R(q1)Tx(a1)R(q2)Ty(a3)R(q3)', [1 2 3])
```

is equivalent to computing:

```
trot2(1) * transl2(a1,0) * trot2(2) * transl2(0,a3) * trot2(3)
```

Notes

- The string can contain spaces between elements or on either side of ARG.
- Works for symbolic variables in the workspace and/or passed in via the vector **q**.
- For symbolic operations that involve use of the value pi, make sure you define it first in the workspace: pi = sym('pi');

See also

trchain, trot2, transl2

---

trinterp

Interpolate homogeneous transformations

T = **trinterp**(**T0**, **T1**, **s**) is a homogeneous transform interpolation between **T0** when **s**=0 to **T1** when **s**=1. Rotation is interpolated using quaternion spherical linear interpolation. If **s** ($N \times 1$) then **T** ($4 \times 4 \times N$) is a sequence of homogeneous transforms corresponding to the interpolation values in **s**.

T = **trinterp**(**T**, **s**) is a transform that varies from the identity matrix when **s**=0 to **T** when R=1. If **s** ($N \times 1$) then **T** ($4 \times 4 \times N$) is a sequence of homogeneous transforms corresponding to the interpolation values in **s**.

See also

ctraj, quaternion

trnorm

Normalize a homogeneous transform

tn = **trnorm**(**T**) is a normalized homogeneous transformation matrix in which the rotation submatrix R = [N,O,A] is guaranteed to be a proper orthogonal matrix. The O and A vectors are normalized and the normal vector is formed from N = O x A, and then we ensure that O and A are orthogonal by O = A x N.

Notes

- Used to prevent finite word length arithmetic causing transforms to become 'unnormalized'.

See also

oa2tr

trot2

SE2 rotation matrix

T = **trot2**(**theta**) is a homogeneous transformation (3×3) representing a rotation of **theta** radians.

T = **trot2**(**theta**, 'deg') as above but **theta** is in degrees.

Notes

- Translational component is zero.

See also

rot2, transl2, trotx, troty, trotz

trotx

Rotation about X axis

T = **trotx**(**theta**) is a homogeneous transformation (4×4) representing a rotation of **theta** radians about the x-axis.

T = **trotx**(**theta**, 'deg') as above but **theta** is in degrees.

Notes

- Translational component is zero.

See also

rotx, troty, trotz, trot2

troty

Rotation about Y axis

T = **troty**(**theta**) is a homogeneous transformation (4×4) representing a rotation of **theta** radians about the y-axis.

T = **troty**(**theta**, 'deg') as above but **theta** is in degrees.

Notes

- Translational component is zero.

See also

roty, trotx, trotz, trot2

---

trotz

Rotation about Z axis

T = **trotz**(**theta**) is a homogeneous transformation (4×4) representing a rotation of **theta** radians about the z-axis.

T = **trotz**(**theta**, 'deg') as above but **theta** is in degrees.

Notes

- Translational component is zero.

See also

rotz, trotx, troty, trot2

---

trplot

Draw a coordinate frame

trplot(**T**, **options**) draws a 3D coordinate frame represented by the homogeneous transform **T** (4×4).

H = **trplot**(**T**, **options**) as above but returns a handle.

trplot(**H**, **T**) moves the coordinate frame described by the handle **H** to the pose **T** (4×4).

trplot(**R**, **options**) draws a 3D coordinate frame represented by the orthonormal rotation matrix **R** (3×3).

H = **trplot**(**R**, **options**) as above but returns a handle.

trplot(**H**, **R**) moves the coordinate frame described by the handle **H** to the orientation **R**.

Options

'color', C	The color to draw the axes, MATLAB colorspec C
'noaxes'	Don't display axes on the plot
'axis', A	Set dimensions of the MATLAB axes to A=[xmin xmax ymin ymax zmin zmax]
'frame', F	The frame is named F and the subscript on the axis labels is F.
'text_opts', opt	A cell array of MATLAB text properties
'handle', **H**	Draw in the MATLAB axes specified by the axis handle **H**
'view', V	Set plot view parameters V=[az el] angles, or 'auto' for view toward origin of coordinate frame
'length', s	Length of the coordinate frame arms (default 1)
'arrow'	Use arrows rather than line segments for the axes
'width', w	Width of arrow tips (default 1)
'thick', t	Thickness of lines (default 0.5)
'3d'	Plot in 3D using anaglyph graphics
'anaglyph', A	Specify anaglyph colors for '3d' as 2 characters for left and right (default colors 'rc'):

```
'r'   red
'g'   green
'b'   green
'c'   cyan
'm'   magenta
```

'dispar', D	Disparity for 3d display (default 0.1)
'text'	Enable display of X,Y,Z labels on the frame
'labels', L	Label the X,Y,Z axes with the 1st, 2nd, 3rd character of the string L
'rgb'	Display X,Y,Z axes in colors red, green, blue respectively

Examples

```
trplot(T, 'frame', 'A')
trplot(T, 'frame', 'A', 'color', 'b')
trplot(T1, 'frame', 'A', 'text_opts', {'FontSize', 10, 'FontWeight', 'bold'})
trplot(T1, 'labels', 'NOA');
```

```
h = trplot(T, 'frame', 'A', 'color', 'b');
trplot(h, T2);
```

3D anaglyph plot

```
trplot(T, '3d');
```

Notes

- The arrow option requires the third party package arrow3.
- The handle **H** is an hgtransform object.
- When using the form **trplot**(**H**, ...) the axes are not rescaled.
- The '3d' option requires that the plot is viewed with anaglyph glasses.
- You cannot specify 'color'

See also

trplot2, tranimate

---

trplot2

Plot a planar transformation

trplot2(**T**, **options**) draws a 2D coordinate frame represented by the SE(2) homogeneous transform **T** (3×3).

H = **trplot2**(**T**, **options**) as above but returns a handle.

trplot2(**H**, **T**) moves the coordinate frame described by the handle **H** to the SE(2) pose **T** (3×3).

Options

'axis', A	Set dimensions of the MATLAB axes to A=[xmin xmax ymin ymax]
'color', c	The color to draw the axes, MATLAB colorspec
'noaxes'	Don't display axes on the plot
'frame', F	The frame is named F and the subscript on the axis labels is F.
'text_opts', opt	A cell array of Matlab text properties
'handle', h	Draw in the MATLAB axes specified by h
'view', V	Set plot view parameters V=[az el] angles, or 'auto' for view toward origin of coordinate frame
'length', s	Length of the coordinate frame arms (default 1)
'arrow'	Use arrows rather than line segments for the axes
'width', w	Width of arrow tips

Examples

```
trplot(T, 'frame', 'A')
trplot(T, 'frame', 'A', 'color', 'b')
trplot(T1, 'frame', 'A', 'text_opts', {'FontSize', 10, 'FontWeight', 'bold'})
```

Notes

- The arrow option requires the third party package arrow3.
- Generally it is best to set the axis bounds

See also

trplot

---

trprint

Compact display of homogeneous transformation

trprint(**T**, **options**) displays the homogoneous transform in a compact single-line format. If **T** is a homogeneous transform sequence then each element is printed on a separate line.

s = **trprint**(**T**, **options**) as above but returns the string.

trprint T is the command line form of above, and displays in RPY format.

Options

'rpy'	display with rotation in roll/pitch/yaw angles (default)
'euler'	display with rotation in ZYX Euler angles
'angvec'	display with rotation in angle/vector format
'radian'	display angle in radians (default is degrees)
'fmt', f	use format string f for all numbers, (default %g)
'label', l	display the text before the transform

Examples

```
>> trprint(T2)
t = (0,0,0), RPY = (-122.704,65.4084,-8.11266) deg

>> trprint(T1, 'label', 'A')

A:t = (0,0,0), RPY = (-0,0,-0) deg
```

See also

tr2eul, tr2rpy, tr2angvec

---

trscale

Create a homogeneous matrix corresponding to pure scale

T = **trscale**(**s**) is a 4×4 homogeneous transform corresponding to a pure scale change. If **s** is a scalar the same scale factor is used for x,y,z, else it can be a 3-vector.

---

unit

Unitize a vector

vn = **unit**(**v**) is a **unit** vector parallel to **v**.

Note

- Reports error for the case where norm(**v**) is zero.

Vehicle

Car-like vehicle class

This class models the kinematics of a car-like vehicle (bicycle model). For given steering and velocity inputs it updates the true vehicle state and returns noise-corrupted odometry readings.

Methods

init	initialize vehicle state
f	predict next state based on odometry
step	move one time step and return noisy odometry
control	generate the control inputs for the vehicle
update	update the vehicle state
run	run for multiple time steps
Fx	Jacobian of f wrt x
Fv	Jacobian of f wrt odometry noise
gstep	like step() but displays vehicle
plot	plot/animate vehicle on current figure
plot_xy	plot the true path of the vehicle
add_driver	attach a driver object to this vehicle
display	display state/parameters in human readable form
char	convert to string

Static methods

plotv	plot/animate a pose on current figure

Properties (read/write)

x	true vehicle state (3×1)
V	odometry covariance (2×2)
odometry	distance moved in the last interval (2×1)
rdim	dimension of the robot (for drawing)
L	length of the vehicle (wheelbase)
alphalim	steering wheel limit
maxspeed	maximum vehicle speed
T	sample interval
verbose	verbosity
x_hist	history of true vehicle state $(N \times 3)$
driver	reference to the driver object
x0	initial state, restored on init()

Examples

Create a vehicle with odometry covariance

```
v = Vehicle( diag([0.1 0.01].^2 );
```

and display its initial state

```
v
```

now apply a speed (0.2m/s) and steer angle (0.1rad) for 1 time step

```
odo = v.update([0.2, 0.1])
```


where odo is the noisy odometry estimate, and the new true vehicle state

```
v
```

We can add a driver object

```
v.add_driver( RandomPath(10) )
```

which will move the vehicle within the region -10<x<10, -10<y<10 which we can see by

```
v.run(1000)
```

which shows an animation of the vehicle moving between randomly selected wayoints.

Notes

- Subclasses the MATLAB handle class which means that pass by reference semantics apply.

Reference

Robotics, Vision & Control, Peter Corke, Springer 2011

See also

RandomPath, EKF

Vehicle.Vehicle

Vehicle object constructor

v = **Vehicle**(**v_act**, **options**) creates a **Vehicle** object with actual odometry covariance **v_act** (2×2) matrix corresponding to the odometry vector [dx dtheta].

Options

'stlim', A	Steering angle limit (default 0.5 rad)
'vmax', S	Maximum speed (default 5m/s)
'L', L	Wheel base (default 1m)
'x0', x0	Initial state (default (0,0,0))
'dt', T	Time interval
'rdim', R	Robot size as fraction of plot window (default 0.2)
'verbose'	Be verbose

Notes

- Subclasses the MATLAB handle class which means that pass by reference semantics apply.

Vehicle.add_driver

Add a driver for the vehicle

V.**add_driver**(**d**) connects a driver object **d** to the vehicle. The driver object has one public method:

```
[speed, steer] = D.demand();
```

that returns a speed and steer angle.

Notes

- The Vehicle.step() method invokes the driver if one is attached.

See also

Vehicle.step, RandomPath

Vehicle.char

Convert to a string

s = V.**char**() is a string showing vehicle parameters and state in in a compact human readable format.

See also

Vehicle.display

Vehicle.control

Compute the control input to vehicle

u = V.**control**(**speed**, **steer**) returns a **control** input (speed,steer) based on provided controls **speed**,**steer** to which speed and steering angle limits have been applied.

u = V.**control**() returns a **control** input (speed,steer) from a "driver" if one is attached, the driver's DEMAND() method is invoked. If no driver is attached then speed and steer angle are assumed to be zero.

See also

Vehicle.step, RandomPath

Vehicle.display

Display vehicle parameters and state

V.**display**() displays vehicle parameters and state in compact human readable form.

Notes

- This method is invoked implicitly at the command line when the result of an expression is a Vehicle object and the command has no trailing semicolon.

See also

Vehicle.char

Vehicle.f

Predict next state based on odometry

xn = V.**f**(**x**, **odo**) predict next state **xn** (1×3) based on current state **x** (1×3) and odometry **odo** (1×2) is [distance,change_heading].

xn = V.**f**(**x**, **odo**, **w**) as above but with odometry noise **w**.

Notes

- Supports vectorized operation where **x** and **xn** ($N \times 3$).

Vehicle.Fv

Jacobian df/dv

J = V.**Fv**(**x**, **odo**) returns the Jacobian df/dv (3×2) at the state **x**, for odometry input **odo**.

See also

Vehicle.F, Vehicle.Fx

Vehicle.Fx

Jacobian df/dx

J = V.**Fx**(**x**, **odo**) is the Jacobian df/dx (3×3) at the state **x**, for odometry input **odo**.

See also

Vehicle.f, Vehicle.Fv

Vehicle.init

Reset state of vehicle object

V.**init**() sets the state V.x := V.x0, initializes the driver object (if attached) and clears the history.

V.**init**(**x0**) as above but the state is initialized to **x0**.

Vehicle.plot

Plot vehicle

V.**plot**(**options**) plots the vehicle on the current axes at a pose given by the current state. If the vehicle has been previously plotted its pose is updated. The vehicle is depicted as a narrow triangle that travels "point first" and has a length V.rdim.

V.**plot**(**x**, **options**) plots the vehicle on the current axes at the pose **x**.

Vehicle.plot_xy

Plots true path followed by vehicle

V.**plot_xy**() plots the true xy-plane path followed by the vehicle.

V.**plot_xy**(**ls**) as above but the line style arguments **ls** are passed to plot.

Notes

- The path is extracted from the x_hist property.

Vehicle.plotv

Plot ground vehicle pose

H = **Vehicle.plotv**(**x**, **options**) draws a representation of a ground robot as an oriented triangle with pose **x** (1×3) [x,y,theta]. **H** is a graphics handle. If **x** ($N \times 3$) is a matrix it is considered to represent a trajectory in which case the vehicle graphic is animated.

Vehicle.plotv(**H**, **x**) as above but updates the pose of the graphic represented by the handle **H** to pose **x**.

Options

'scale', S	Draw vehicle with length S x maximum axis dimension
'size', S	Draw vehicle with length S
'color', C	Color of vehicle.
'fill'	Filled with solid color as per 'color' option
'fps', F	Frames per second in animation mode (default 10)

Example

Generate some path $3 \times N$

```
p = PRM.plan(start, goal);
```

Set the axis dimensions to stop them rescaling for every point on the path

```
axis([-5 5 -5 5]);
```

Now invoke the static method

```
Vehicle.plotv(p);
```


Notes

- This is a static method.

Vehicle.run

Run the vehicle simulation

V.**run**(**n**) runs the vehicle model for **n** timesteps and plots the vehicle pose at each step.

p = V.**run**(**n**) runs the vehicle simulation for **n** timesteps and return the state history (**n** $\times$ 3) without plotting. Each row is (x,y,theta).

See also

Vehicle.step

Vehicle.run2

run the vehicle simulation

p = V.**run2**(**T**, **x0**, **speed**, **steer**) runs the vehicle model for a time **T** with speed **speed** and steering angle **steer**. **p** ($N \times 3$) is the path followed and each row is (x,y,theta).

Notes

- Faster and more specific version of run() method.

See also

Vehicle.run, Vehicle.step

Vehicle.step

Advance one timestep

odo = V.**step**(**speed**, **steer**) updates the vehicle state for one timestep of motion at specified **speed** and **steer** angle, and returns noisy odometry.

odo = V.**step**() updates the vehicle state for one timestep of motion and returns noisy odometry. If a "driver" is attached then its DEMAND() method is invoked to compute speed and steer angle. If no driver is attached then speed and steer angle are assumed to be zero.

Notes

- Noise covariance is the property V.

See also

Vehicle.control, Vehicle.update, Vehicle.add_driver

Vehicle.update

Update the vehicle state

odo = V.**update**(**u**) is the true odometry value for motion with **u**=[speed,steer].

Notes

- Appends new state to state history property x_hist.
- Odometry is also saved as property odometry.

Vehicle.verbosity

Set verbosity

V.**verbosity**(**a**) set **verbosity** to **a**. **a**=0 means silent.

---

vex

Convert skew-symmetric matrix to vector

v = **vex**(**s**) is the vector (3×1) which has the skew-symmetric matrix **s** (3×3)

```
| 0   -vz  vy|
| vz   0  -vx|
|-vy   vx  0 |
```

Notes

- This is the inverse of the function SKEW().
- No checking is done to ensure that the matrix is actually skew-symmetric.
- The function takes the mean of the two elements that correspond to each unique element of the matrix, ie. vx = 0.5*(**s**(3,2)-**s**(2,3))

See also

skew

VREP

V-REP simulator communications object

A VREP object holds all information related to the state of a connection. References are passed to other objects which mirror the V-REP environment in MATLAB.

This class handles the interface to the simulator and low-level object handle operations.

Methods throw exception if an error occurs.

Methods

gethandle	get handle to named object
getchildren	get children belonging to handle
object	return a VREP_obj object for named object
arm	return a VREP_arm object for named robot
camera	return a VREP_camera object for named vosion sensor
hokuyo	return a VREP_hokuyo object for named Hokuyo scanner
getpos	return position of object given handle
setpos	set position of object given handle
getorient	return orientation of object given handle
setorient	set orientation of object given handle
getpose	return pose of object given handle
setpose	set pose of object given handle
setobjparam_bool	set object boolean parameter
setobjparam_int	set object integer parameter
setobjparam_float	set object float parameter
getobjparam_bool	get object boolean parameter
getobjparam_int	get object integer parameter
getobjparam_float	get object float parameter
signal_int	send named integer signal
signal_float	send named float signal
signal_str	send named string signal
setparam_bool	set object boolean parameter
setparam_int	set object integer parameter
setparam_float	set object float parameter
delete	shutdown the connection and cleanup
startsim	start the simulator running
stopsim	stop the simulator running
pausesim	pause the simulator
getversion	get V-REP version number
checkcomms	return status of connection
pausecomms	pause the comms
display	print the link parameters in human readable form
char	convert to string

See also

VREP_obj, VREP_arm, VREP_camera, VREP_hokuyo

VREP.VREP

VREP object constructor

v = **VREP**(**options**) create a connection to the V-REP simulator.

v = **VREP**(**path**, **options**) as above but specify the root directory of V-REP.

Options

'version', V	Version of V-REP, V=304, 311 etc
'timeout', T	Timeout T in ms (default 2000)
'cycle', C	Cycle time C in ms (default 5)
'port', P	Override communications port
'reconnect'	Reconnect on error (default noreconnect)

VREP.arm

Return VREP_arm object

V.**arm**(**name**) is a factory method that returns a VREP_arm object for the V-REP robot object named NAME.

See also

VREP_arm

VREP.camera

Return VREP_camera object

V.**camera**(**name**) is a factory method that returns a VREP_camera object for the V-REP vision sensor object named NAME.

See also

VREP_camera

VREP.checkcomms

Check communications to V-REP simulator

V.**checkcomms**() is true if a valid connection to the V-REP simulator exists.

VREP.delete

VREP object destructor

delete(**v**) closes the connection to the V-REP simulator

VREP.getchildren

Return children of object

C = V.**getchildren**(**H**) is a vector of integer handles for the V-REP object denoted by the integer handle **H**.

VREP.gethandle

Return handle to VREP object

H = V.**gethandle**(**name**) is an integer handle for named V-REP object.

H = V.**gethandle**(**fmt**, **arglist**) as above but the name is formed from sprintf(**fmt**, **arglist**).

VREP.getjoint

Get value of V-REP joint object

V.**getjoint**(**H**, **q**) is the position of joint object with integer handle **H**.

VREP.getobjparam_bool

get boolean parameter of a V-REP object

V.**getobjparam_bool**(**H**, **param**) gets the boolean parameter with identifier **param** of object with integer handle **H**.

VREP.getobjparam_float

get float parameter of a V-REP object

V.**getobjparam_bool**(**H**, **param**) gets the float parameter with identifier **param** of object with integer handle **H**.

VREP.getobjparam_int

get Integer parameter of a V-REP object

V.**getobjparam_int**(**H**, **param**) gets the integer parameter with identifier **param** of object with integer handle **H**.

VREP.getorient

Get orientation of V-REP object

V.**getorient**(**H**) is the orientation as a rotation matrix (3×3) of the V-REP object with integer handle **H**.

V.**getorient**(**H**, 'euler', OPTIONS) as above but returns ZYZ Euler angles.

V.**getorient**(**H**, **hrr**) as above but orientation is relative to the position of object with integer handle HR.

V.**getorient**(**H**, **hrr**, 'euler', OPTIONS) as above but returns ZYZ Euler angles.

Options

See tr2eul.

VREP.getpos

Get position of V-REP object

V.**getpos**(**H**) is the position (1×3) of the V-REP object with integer handle **H**.

V.**getpos**(**H**, **hr**) as above but position is relative to the position of object with integer handle **hr**.

VREP.getpose

Get pose of V-REP object

V.**getpose**(**H**) is the pose (4×4) of the V-REP object with integer handle **H**.

V.**getpose**(**H**, **hr**) as above but pose is relative to the pose of object with integer handle R.

VREP.getversion

Get version of the V-REP simulator

V.**getversion**() is the version of the V-REP simulator server as an integer MNNNN where M is the major version number and NNNN is the minor version number.

VREP.hokuyo

Return VREP_hokuyo object

V.**hokuyo**(**name**) is a factory method that returns a VREP_hokuyo object for the V-REP Hokuyo laser scanner object named NAME.

See also

VREP_hokuyo

VREP.mobile

Return VREP_mobile object

V.**mobile**(**name**) is a factory method that returns a VREP_mobile object for the V-REP **mobile** base object named NAME.

See also

VREP_mobile

VREP.object

Return VREP_obj object

V.**objet**(**name**) is a factory method that returns a VREP_obj object for the V-REP object named NAME.

See also

VREP_obj

VREP.pausecomms

Pause communcations to the V-REP simulator

V.**pausecomms**(**p**) pauses communications to the V-REP simulation engine if **p** is true else resumes it. Useful to ensure an atomic update of simulator state.

VREP.setjoint

Set value of V-REP joint object

V.**setjoint**(**H**, **q**) sets the position of joint object with integer handle **H** to the value **q**.

VREP.setjointtarget

Set target value of V-REP joint object

V.**setjointtarget**(**H**, **q**) sets the target position of joint object with integer handle **H** to the value **q**.

VREP.setjointvel

Set velocity of V-REP joint object

V.**setjointvel**(**H**, **qd**) sets the target velocity of joint object with integer handle **H** to the value **qd**.

VREP.setobjparam_bool

Set boolean parameter of a V-REP object

V.**setobjparam_bool**(**H**, **param**, **val**) sets the boolean parameter with identifier **param** of object **H** to value **val**.

VREP.setobjparam_float

Set float parameter of a V-REP object

V.**setobjparam_bool**(**H**, **param**, **val**) sets the float parameter with identifier **param** of object **H** to value **val**.

VREP.setobjparam_int

Set Integer parameter of a V-REP object

V.**setobjparam_int**(**H**, **param**, **val**) sets the integer parameter with identifier **param** of object **H** to value **val**.

VREP.setorient

Set orientation of V-REP object

V.**setorient**(**H**, **R**) sets the orientation of V-REP object with integer handle **H** to that given by rotation matrix **R** (3×3).

V.**setorient**(**H**, **T**) sets the orientation of V-REP object with integer handle **H** to rotational component of homogeneous transformation matrix **T** (4×4).

V.**setorient**(**H**, **E**) sets the orientation of V-REP object with integer handle **H** to ZYZ Euler angles (1×3).

V.**setorient**(**H**, **x**, **hr**) as above but orientation is set relative to the orientation of object with integer handle **hr**.

VREP.setparam_bool

Set boolean parameter of the V-REP simulator

V.**setparam_bool**(**name**, **val**) sets the boolean parameter with name **name** to value **val** within the V-REP simulation engine.

VREP.setparam_float

Set float parameter of the V-REP simulator

V.**setparam_float**(**name**, **val**) sets the float parameter with name **name** to value **val** within the V-REP simulation engine.

VREP.setparam_int

Set intger parameter of the V-REP simulator

V.**setparam_int**(**name**, **val**) sets the integer parameter with name **name** to value **val** within the V-REP simulation engine.

VREP.setpos

Set position of V-REP object

V.**setpos**(**H**, **T**) sets the position of V-REP object with integer handle **H** to **T** (1×3).

V.**setpos**(**H**, **T**, **hr**) as above but position is set relative to the position of object with integer handle **hr**.

VREP.setpose

Set pose of V-REP object

V.**setpos**(**H**, **T**) sets the pose of V-REP object with integer handle **H** according to homogeneous transform **T** (4×4).

V.**setpos**(**H**, **T**, **hr**) as above but pose is set relative to the pose of object with integer handle **hr**.

VREP.signal_float

Send a float signal to the V-REP simulator

V.**signal_float**(**name**, **val**) send a float signal with name **name** and value **val** to the V-REP simulation engine.

VREP.signal_int

Send an integer signal to the V-REP simulator

V.**signal_int**(**name**, **val**) send an integer signal with name **name** and value **val** to the V-REP simulation engine.

VREP.signal_str

Send a string signal to the V-REP simulator

V.**signal_str**(**name**, **val**) send a string signal with name **name** and value **val** to the V-REP simulation engine.

VREP.simpause

Pause V-REP simulation

V.**simpause**() pauses the V-REP simulation engine. Use V.simstart() to resume the simulation.

See also

VREP.simstart

VREP.simstart

Start V-REP simulation

V.**simstart**() starts the V-REP simulation engine.

See also

VREP.simstop, VREP.simpause

VREP.simstop

Stop V-REP simulation

V.**simstop**() stops the V-REP simulation engine.

See also

VREP.simstart

VREP.youbot

Return VREP_youbot object

V.**youbot**(**name**) is a factory method that returns a VREP_youbot object for the V-REP YouBot object named NAME.

See also

vrep_youbot

VREP_arm

V-REP mirror of robot arm object

Mirror objects are MATLAB objects that reflect objects in the V-REP environment. Methods allow the V-REP state to be examined or changed.

This is a concrete class, derived from VREP_mirror, for all V-REP robot arm objects and allows access to joint variables.

Methods throw exception if an error occurs.

Example

```
vrep = VREP();
arm = vrep.arm('IRB140');
q = arm.getq();
arm.setq(zeros(1,6));
arm.setpose(T);  % set pose of base
```

Methods

getq	return joint coordinates
setq	set joint coordinates
animate	animate a joint coordinate trajectory

Superclass methods (VREP_obj)

getpos	return position of object given handle
setpos	set position of object given handle
getorient	return orientation of object given handle
setorient	set orientation of object given handle
getpose	return pose of object given handle
setpose	set pose of object given handle

can be used to set/get the pose of the robot base.

Superclass methods (VREP_base)

setobjparam_bool	set object boolean parameter
setobjparam_int	set object integer parameter
setobjparam_float	set object float parameter

Properties

n	Number of joints

See also

VREP_mirror, VREP_obj, VREP_arm, VREP_camera, VREP_hokuyo

VREP_arm.VREP_arm

Create a robot arm mirror object

R = **VREP_arm**(**name**, **options**) is a mirror object that corresponds to the robot arm named **name** in the V-REP environment.

Options

'fmt', F	Specify format for joint object names (default '%s_joint%d')

Notes

- The number of joints is found by searching for objects with names systematically derived from the root object name, by default named NAME_N where N is the joint number starting at 0.

See also

VREP.arm

VREP_arm.animate

Animate V-REP robot

R.**animate**(**qt**, **options**) **animate** the corresponding V-REP robot with configurations taken consecutive rows of **qt** ($M \times N$) which represents an M-point trajectory.

Options

'delay', D	Delay (s) betwen frames for animation (default 0.1)
'fps', fps	Number of frames per second for display, inverse of 'delay' option
'[no]loop'	Loop over the trajectory forever

See also

SerialLink.plot

VREP_arm.getq

Get joint angles of V-REP robot

R.**getq**() is the vector of joint angles ($1 \times N$) from the corresponding robot arm in the V-REP simulation.

VREP_arm.setq

Set joint angles of V-REP robot

R.**setq**(**q**) sets the joint angles of the corresponding robot arm in the V-REP simulation to **q** ($1 \times N$).

VREP_arm.teach

Graphical teach pendant

R.**teach**(**options**) drive a V-REP robot by means of a graphical slider panel.

Options

'degrees'	Display angles in degrees (default radians)
'q0', q	Set initial joint coordinates

Notes

- The slider limits are all assumed to be [-pi, +pi]

See also

SerialLink.plot

VREP_camera

V-REP mirror of vision sensor object

Mirror objects are MATLAB objects that reflect objects in the V-REP environment. Methods allow the V-REP state to be examined or changed.

This is a concrete class, derived from VREP_mirror, for all V-REP vision sensor objects and allows access to images and image parameters.

Methods throw exception if an error occurs.

Example

```
vrep = VREP();
camera = vrep.camera('Vision_sensor');
im = camera.grab();
camera.setpose(T);
R = camera.getorient();
```

Methods

grab	return an image from simulated camera
setangle	set field of view
setresolution	set image resolution
setclipping	set clipping boundaries

Superclass methods (VREP_obj)

getpos	return position of object given handle
setpos	set position of object given handle
getorient	return orientation of object given handle
setorient	set orientation of object given handle
getpose	return pose of object given handle
setpose	set pose of object given handle

can be used to set/get the pose of the robot base.

Superclass methods (VREP_base)

setobjparam_bool	set object boolean parameter
setobjparam_int	set object integer parameter
setobjparam_float	set object float parameter

Properties

n	Number of joints

See also

VREP_mirror, VREP_obj, VREP_arm, VREP_camera, VREP_hokuyo

VREP_camera.VREP_camera

Create a camera mirror object

C = **VREP_camera**(**name**, **options**) is a mirror object that corresponds to the a vision senor named **name** in the V-REP environment.

Options

'fov', A	Specify field of view in degreees (default 60)
'resolution', N	Specify resolution. If scalar $N \times N$ else N(1)xN(2)
'clipping', Z	Specify near Z(1) and far Z(2) clipping boundaries

Notes

- Default parameters are set in the V-REP environment

See also

VREP_obj

VREP_camera.getangle

Fet field of view for V-REP vision sensor

fov = C.**getangle**(**fov**) is the field-of-view angle to **fov** in radians.

VREP_camera.getclipping

Get clipping boundaries for V-REP vision sensor

C.**getclipping**() is the near and far clipping boundaries (1×2) in the Z-direction as a 2-vector [NEAR,FAR].

VREP_camera.getresolution

Get resolution for V-REP vision sensor

R = C.**getresolution**() is the image resolution (1×2) of the vision sensor **R**(1)xR(2).

VREP_camera.grab

Get image from V-REP vision sensor

im = C.**grab**(**options**) is an image ($W \times H$) returned from the V-REP vision sensor.

C.**grab**(**options**) as above but the image is displayed using idisp.

Options

'grey'	Return a greyscale image (default color).

Notes

- V-REP simulator must be running
- Very slow, ie. seconds to **grab** a 256×256 image
- Color images can be quite dark, ensure good light sources
- Uses the signal 'handle_rgb_sensor' to trigger a single image generation.

See also

idisp

VREP_camera.setangle

Set field of view for V-REP vision sensor

C.**setangle**(**fov**) set the field-of-view angle to **fov** in radians.

VREP_camera.setclipping

Set clipping boundaries for V-REP vision sensor

C.**setclipping**(**near**, **far**) set clipping boundaries to the range of Z from **near** to **far**. Objects outside this range will not be rendered.

VREP_camera.setresolution

Set resolution for V-REP vision sensor

C.**setresolution**(**R**) set image resolution to **R** $\times$ **R** if **R** is a scalar or **R**(1)xR(2) if it is a 2-vector.

VREP_mirror

V-REP mirror object class

Mirror objects are MATLAB objects that reflect objects in the V-REP environment. Methods allow the V-REP state to be examined or changed.

This abstract class is the root class for all V-REP mirror objects.

Methods throw exception if an error occurs.

Methods

setobjparam_bool	set object boolean parameter
setobjparam_int	set object integer parameter
setobjparam_float	set object float parameter

See also

VREP_obj, VREP_arm, VREP_camera, VREP_hokuyo

VREP_mirror.VREP_mirror

VREP_mirror object constructor

v = **VREP_mirror**(**name**) creates a V-REP mirror object.

VREP_mirror.getobjparam_bool

Get boolean parameter of V-REP object

V.**getparam_bool**(**name**, **val**) is the boolean parameter with name **name** of the corresponding V-REP object.

VREP_mirror.getobjparam_float

Get float parameter of V-REP object

V.**getparam_float**(**name**, **val**) is the float parameter with name **name** of the corresponding V-REP object.

VREP_mirror.getobjparam_int

Get integer parameter of V-REP object

V.**getparam_int**(**name**, **val**) is the integer parameter with name **name** of the corresponding V-REP object.

VREP_mirror.setobjparam_bool

Set boolean parameter of V-REP object

V.**setparam_bool**(**name**, **val**) sets the boolean parameter with name **name** to value **val** within the V-REP simulation engine.

VREP_mirror.setobjparam_float

Set float parameter of V-REP object

V.**setparam_float**(**name**, **val**) sets the float parameter with name **name** to value **val** within the V-REP simulation engine.

VREP_mirror.setobjparam_int

Set integer parameter of V-REP object

V.**setparam_int**(**name**, **val**) sets the integer parameter with name **name** to value **val** within the V-REP simulation engine.

VREP_obj

V-REP mirror of simple object

Mirror objects are MATLAB objects that reflect objects in the V-REP environment. Methods allow the V-REP state to be examined or changed.

This is a concrete class, derived from VREP_mirror, for all V-REP objects and allows access to pose and object parameters.

Example

```
vrep = VREP();
bill = vrep.object('Bill');  % get the human figure Bill
bill.setpos([1,2,0]);
bill.setorient([0 pi/2 0]);
```

Methods throw exception if an error occurs.

Methods

getpos	return position of object given handle
setpos	set position of object given handle
getorient	return orientation of object given handle
setorient	set orientation of object given handle
getpose	return pose of object given handle
setpose	set pose of object given handle

Superclass methods (VREP_base)

setobjparam_bool	set object boolean parameter
setobjparam_int	set object integer parameter
setobjparam_float	set object float parameter

display	print the link parameters in human readable form
char	convert to string

Properties (read/write)

See also

VREP_mirror, VREP_obj, VREP_arm, VREP_camera, VREP_hokuyo

VREP_obj.VREP_obj

VREP_obj mirror object constructor

v = **VREP_base**(**name**) creates a V-REP mirror object for a simple V-REP object type.

VREP_obj.getorient

Get orientation of V-REP object

V.**getorient**() is the orientation of the corresponding V-REP object as a rotation matrix (3×3).

V.**getorient**('euler', OPTIONS) as above but returns ZYZ Euler angles.

V.**getorient**(**base**) is the orientation of the corresponding V-REP object relative to the **VREP_obj** object **base**.

V.**getorient**(**base**, 'euler', OPTIONS) as above but returns ZYZ Euler angles.

Options

See tr2eul.

VREP_obj.getpos

Get position of V-REP object

V.**getpos**() is the position (1×3) of the corresponding V-REP object.

V.**getpos**(**base**) as above but position is relative to the **VREP_obj** object **base**.

VREP_obj.getpose

Get pose of V-REP object

V.**getpose**() is the pose (4×4) of the the corresponding V-REP object.

V.**getpose**(**base**) as above but pose is relative to the pose the **VREP_obj** object **base**.

VREP_obj.setorient

Set orientation of V-REP object

V.**setorient**(**R**) sets the orientation of the corresponding V-REP to rotation matrix **R** (3×3).

V.**setorient**(**T**) sets the orientation of the corresponding V-REP object to rotational component of homogeneous transformation matrix **T** (4×4).

V.**setorient**(**E**) sets the orientation of the corresponding V-REP object to ZYZ Euler angles (1×3).

V.**setorient**(**x**, **base**) as above but orientation is set relative to the orientation of **VREP_obj** object **base**.

VREP_obj.setpos

Set position of V-REP object

V.**setpos**(**T**) sets the position of the corresponding V-REP object to **T** (1×3).

V.**setpos**(**T**, **base**) as above but position is set relative to the position of the **VREP_obj** object **base**.

VREP_obj.setpose

Set pose of V-REP object

V.**setpose**(**T**) sets the pose of the corresponding V-REP object to **T** (4×4).

V.**setpose**(**T**, **base**) as above but pose is set relative to the pose of the **VREP_obj** object **base**.

wtrans

Transform a wrench between coordinate frames

wt = **wtrans**(**T**, **w**) is a wrench (6×1) in the frame represented by the homogeneous transform **T** (4×4) corresponding to the world frame wrench **w** (6×1).

The wrenches **w** and **wt** are 6-vectors of the form [Fx Fy Fz Mx My Mz].

See also

tr2delta, tr2jac

xaxis

Set X-axis scaling

xaxis(**max**) set x-axis scaling from 0 to **max**.

xaxis(**min**, **max**) set x-axis scaling from **min** to **max**.

xaxis([**min max**]) as above.

xaxis restore automatic scaling for x-axis.

xyzlabel

Label X, Y and Z axes

XYZLABEL label the x-, y- and z-axes with 'X', 'Y', and 'Z' respectiveley

---

yaxis

Y-axis scaling

yayis(**max**) yayis(min, **max**)

YAXIS restore automatic scaling for this axis
