Robotics Toolbox for MATLAB
Release 9

Peter Corke
<table>
<thead>
<tr>
<th>Release</th>
<th>9.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release date</td>
<td>April 2012</td>
</tr>
<tr>
<td>Licence</td>
<td>LGPL</td>
</tr>
</tbody>
</table>

Copyright ©2012 Peter Corke
peter.i.corke@gmail.com
[http://www.petercorke.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, and I commend it to you.
## Contents

Introduction ......................................................... 4

1 Introduction 9
   1.1 What’s changed ............................................. 9
       1.1.1 Documentation ...................................... 9
       1.1.2 Changed behaviour .................................. 9
       1.1.3 New functions ...................................... 10
       1.1.4 Improvements ...................................... 12
   1.2 How to obtain the Toolbox ............................... 12
   1.3 MATLAB version issues ................................. 13
   1.4 Use in teaching .......................................... 13
   1.5 Use in research .......................................... 13
   1.6 Support .................................................... 13
   1.7 Related software ........................................ 14
       1.7.1 Octave .............................................. 14
       1.7.2 Python version ..................................... 14
       1.7.3 Machine Vision toolbox ................................ 14
   1.8 Acknowledgements ........................................ 15

2 Functions and classes ........................................ 16
   about .......................................................... 16
   angdiff ......................................................... 16
   angvec2r ....................................................... 17
   angvec2tr ...................................................... 17
   Bug2 ............................................................ 17
   circle .......................................................... 19
   colnorm ........................................................ 19
   ctraj ............................................................ 19
   delta2tr ........................................................ 20
   DHFactor ........................................................ 20
   diff2 ............................................................ 22
   distancexfm .................................................... 22
   Dstar ............................................................ 23
   DXform .......................................................... 27
   e2h ............................................................... 30
   edgelist ......................................................... 30
   EKF ............................................................... 31
   eul2jac .......................................................... 38
eul2r ......................................................... 38
eul2tr ......................................................... 39
gauss2d ....................................................... 40
h2e ........................................................... 40
homline ....................................................... 40
homtrans ..................................................... 41
imeshgrid ..................................................... 41
ishomog ....................................................... 42
isrot ........................................................ 42
isvec ........................................................ 42
jtraj .......................................................... 43
Link .......................................................... 43
lspb .......................................................... 50
Map .......................................................... 51
mdl_Fanuc10L ............................................... 53
mdl_MotomanHP6 ........................................... 54
mdl_puma560 ............................................... 54
mdl_puma560akb .......................................... 55
mdl_quadcopter ............................................ 56
mdl_S4ABB2p8 ............................................. 57
mdl_stanford ............................................... 57
mdl_twolink ............................................... 58
mstraj ....................................................... 59
mtraj ........................................................ 60
Navigation .................................................. 60
numcols ...................................................... 66
numrows ..................................................... 67
oa2r ........................................................ 67
oa2tr ........................................................ 67
ParticleFilter .............................................. 68
PGraph ....................................................... 72
plot2 ........................................................ 85
plot_box ..................................................... 86
plot_circle ................................................... 86
plot_ellipse ................................................ 87
plot_ellipse_inv ........................................... 87
plot_homline ............................................... 87
plot_point .................................................. 88
plot_poly .................................................... 88
plot_sphere ................................................ 89
plot_vehicle ............................................... 89
plotbotopt .................................................. 90
plotp ........................................................ 90
Polygon ...................................................... 90
PRM .......................................................... 90
qplot ......................................................... 95
Quaternion .................................................. 98
r2t ........................................................... 107
RandomPath ............................................... 108
Range Bearing Sensor ..................................... 111
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotx</td>
<td>115</td>
</tr>
<tr>
<td>roty</td>
<td>115</td>
</tr>
<tr>
<td>rotz</td>
<td>115</td>
</tr>
<tr>
<td>rpy2jac</td>
<td>116</td>
</tr>
<tr>
<td>rpy2r</td>
<td>116</td>
</tr>
<tr>
<td>rpy2tr</td>
<td>117</td>
</tr>
<tr>
<td>RRT</td>
<td>118</td>
</tr>
<tr>
<td>rt2tr</td>
<td>120</td>
</tr>
<tr>
<td>rt2demo</td>
<td>121</td>
</tr>
<tr>
<td>se2</td>
<td>122</td>
</tr>
<tr>
<td>Sensor</td>
<td>122</td>
</tr>
<tr>
<td>SerialLink</td>
<td>123</td>
</tr>
<tr>
<td>skew</td>
<td>144</td>
</tr>
<tr>
<td>startup_rtb</td>
<td>144</td>
</tr>
<tr>
<td>t2r</td>
<td>145</td>
</tr>
<tr>
<td>tb_optparse</td>
<td>145</td>
</tr>
<tr>
<td>tpoly</td>
<td>146</td>
</tr>
<tr>
<td>tr2angvec</td>
<td>147</td>
</tr>
<tr>
<td>tr2delta</td>
<td>147</td>
</tr>
<tr>
<td>tr2eul</td>
<td>148</td>
</tr>
<tr>
<td>tr2jac</td>
<td>149</td>
</tr>
<tr>
<td>tr2rpm</td>
<td>149</td>
</tr>
<tr>
<td>tr2rt</td>
<td>150</td>
</tr>
<tr>
<td>tranimate</td>
<td>151</td>
</tr>
<tr>
<td>transl</td>
<td>151</td>
</tr>
<tr>
<td>trinterp</td>
<td>152</td>
</tr>
<tr>
<td>trnorm</td>
<td>152</td>
</tr>
<tr>
<td>trotx</td>
<td>153</td>
</tr>
<tr>
<td>troty</td>
<td>153</td>
</tr>
<tr>
<td>trotz</td>
<td>154</td>
</tr>
<tr>
<td>trplot</td>
<td>154</td>
</tr>
<tr>
<td>trplot2</td>
<td>156</td>
</tr>
<tr>
<td>trprint</td>
<td>157</td>
</tr>
<tr>
<td>unit</td>
<td>157</td>
</tr>
<tr>
<td>Vehicle</td>
<td>158</td>
</tr>
<tr>
<td>vex</td>
<td>165</td>
</tr>
<tr>
<td>wtrans</td>
<td>166</td>
</tr>
<tr>
<td>xaxis</td>
<td>166</td>
</tr>
</tbody>
</table>
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 Changed behaviour

Compared to release 8 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 parameters 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.
1.1. WHAT'S CHANGED

- 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
- ct_traj 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 (e.g. cameras) it useful to consider the rotations about the Z, Y, Z 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 features, 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
  jsgnu
  qplot

  DHFfactor 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 mini-
mum 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
1.2. HOW TO OBTAIN THE TOOLBOX

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

http://www.petercorke.com

The files are available in either gzipped tar format (.gz) or zip format (.zip). The web page requests some information from you such as your country, type of organization and application. This is just a means for me to gauge interest and to help convince my bosses (and myself) that this is a worthwhile activity.

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.

A menu-driven demonstration can be invoked by the function \texttt{rtdemo}.

### 1.3 MATLAB version issues

The Toolbox has been tested under R2011a.

### 1.4 Use in teaching

This is definitely encouraged! You are free to put the PDF manual (\texttt{robot.pdf} or the web-based documentation \texttt{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

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

or

“A robotics toolbox for MATLAB”,
P.Corke,
IEEE Robotics and Automation Magazine,

which is also given in electronic form in the README 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 you 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”


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 (Quat- ernion, 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},

Robotics Toolbox 9.5 for MATLAB®  14  Copyright ©Peter Corke 2012
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

Last, but not least, 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

\[ R = \text{angvec2r}(\theta, v) \]

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

\[ T = \text{angvec2tr}(\theta, v) \]

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

---

**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.
CHAPTER 2. FUNCTIONS AND CLASSES

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Compute a path from start to goal</td>
</tr>
<tr>
<td>visualize</td>
<td>Display the obstacle map (deprecated)</td>
</tr>
<tr>
<td>plot</td>
<td>Display the obstacle map</td>
</tr>
<tr>
<td>display</td>
<td>Display state/parameters in human readable form</td>
</tr>
<tr>
<td>char</td>
<td>Convert to string</td>
</tr>
</tbody>
</table>

Example

```matlab
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


See also

Navigation, DXform, Dstar, PRM

## Bug2.Bug2

**bug2 navigation object constructor**

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

### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'goal', G</td>
<td>Specify the goal point (1 × 2)</td>
</tr>
<tr>
<td>'inflate', K</td>
<td>Inflate all obstacles by K cells.</td>
</tr>
</tbody>
</table>
See also

Navigation.Navigation

**circle**

Compute points on a circle

\texttt{circle(C, R, opt)} plot a circle centred at \( C \) with radius \( R \).

\[ x = \texttt{circle}(C, R, \text{opt}) \]

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

\( C \) is normally \( 2 \times 1 \) but if \( 3 \times 1 \) 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)

**colnorm**

Column-wise norm of a matrix

\[ \texttt{cn} = \texttt{colnorm}(a) \]

returns an \( M \times 1 \) vector of the normals of each column of the matrix \( a \) which is \( N \times M \).

**ctraj**

Cartesian trajectory between two points

\[ \texttt{tc} = \texttt{ctraj}(T0, T1, n) \]

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.
te = ctraj(T0, T1, s) as above but the elements of s (n × 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.

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 infinitesimal 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.
CHAPTER 2. FUNCTIONS AND CLASSES

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>the base transform as a Java string</td>
</tr>
<tr>
<td>tool</td>
<td>the tool transform as a Java string</td>
</tr>
<tr>
<td>command</td>
<td>a command string that will create a SerialLink() object</td>
</tr>
<tr>
<td></td>
<td>representing the specified kinematics</td>
</tr>
<tr>
<td>char</td>
<td>convert to string representation</td>
</tr>
<tr>
<td>display</td>
<td>display in human readable form</td>
</tr>
</tbody>
</table>

Example

```matlab
>> 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

- Robotics, Vision & Control, Sec 7.5.2, 7.7.1, Peter Corke, Springer 2011.

See also

SerialLink
**diff2**

**point difference**

\( \mathbf{d} = \text{diff3}(\mathbf{v}) \) is the 2-point difference for each point in the vector \( \mathbf{v} \) and the first element is zero. The vector \( \mathbf{d} \) has the same length as \( \mathbf{v} \).

**See also**

diff

---

**distanceexform**

**Distance transform of occupancy grid**

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

\( \mathbf{d} = \text{distanceexform} (\text{world}, \text{goal}, \text{metric}) \) as above but specifies the distance metric as either ‘cityblock’ or ‘Euclidean’

\( \mathbf{d} = \text{distanceexform} (\text{world}, \text{goal}, \text{metric}, \text{show}) \) as above but shows an animation of the distance transform being formed, with a delay of \( \text{show} \) seconds between frames.

**Notes**

- The Machine Vision Toolbox function imorph is required.
- The goal is \([X,Y]\) not MATLAB \([\text{row, col}]\)

**See also**

imorph, DXform
CHAPTER 2. FUNCTIONS AND CLASSES

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
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

See also

Navigation, DXform, PRM

**Dstar.Dstar**

**D* constructor**

\[ ds = \text{Dstar(map, options)} \]

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 converted to a costmap with a unit cost for traversing a cell.

**Options**

- `'goal', G` Specify the goal point \((2 \times 1)\)
- `'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

$C = \text{DS.costmap.get()}$ 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 autogenerate 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 $\text{Inf}$

See also

Dstar.costmap_set, Dstar.costmap_modify

Dstar.costmap_modify

Modify cost map

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

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 traverse. A value of $\text{Inf}$ 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

\( C = \text{DS.distancemap.get()} \) 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

Dstar.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.

Dstar.plot(p) as above but also overlays a path given by the set of points p (M × 2).

See also

Navigation.plot

Dstar.reset

Reset the planner

Dstar.reset() resets the D* planner. The next instantiation of Dstar.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.
CHAPTER 2. FUNCTIONS AND CLASSES

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

```matlab
load map1 % load map
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, distanceform

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**

---

**edgelist**

**Return list of edge pixels for region**

\[ E = \text{edgelist}(\text{im}, \text{seed}) \] return the 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).

\[ E = \text{edgelist}(\text{im}, \text{seed}, \text{direction}) \] returns the list of edge pixels as above, but the direction of edge following is specified. direction \( \neq 0 \) (default) means clockwise, non zero

---

Robotics Toolbox 9.5 for MATLAB® 30 Copyright ©Peter Corke 2012
is counter-clockwise. Note that direction is with respect to y-axis upward, in matrix coordinate frame, not image frame.

**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.

**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.
CHAPTER 2. FUNCTIONS AND CLASSES

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
- 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 $P_0$

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

We run the simulation for 1000 time steps

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

then plot true vehicle path

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

and overlay the estimated path

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

and overlay uncertainty ellipses at every 20 time steps

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

We can plot the covariance against time as

```matlab
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 $V_{\text{est}}$ and $W_{\text{est}}$ and initial vehicle state covariance $P_0$

```matlab
veh = Vehicle(V);
veh.add_driver(RandomPath(20, 2));
map = Map(20);
sensor = RangeBearingSensor(veh, map, W);
ekf = EKF(veh, V_{\text{est}}, P_0, sensor, W_{\text{est}}, map);
```

We run the simulation for 1000 time steps

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

then plot the map and the true vehicle path

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

and overlay the estimated path

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

and overlay uncertainty ellipses at every 20 time steps

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

We can plot the covariance against time as

```matlab
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 $W_{\text{est}}$ and a “perfect” vehicle (no covariance), then run the filter for N time steps.

```matlab
veh = Vehicle(V);
veh.add_driver(RandomPath(20, 2));
sensor = RangeBearingSensor(veh, map, W);
ekf = EKF(veh, [], [], sensor, W_{\text{est}}, []);
```

We run the simulation for 1000 time steps

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

Then plot the true map

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

and overlay the estimated map with 3 sigma ellipses

```matlab
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 \( V_{\text{est}} \) and \( W_{\text{est}} \) and initial state covariance \( P_0 \), then run the filter to estimate the vehicle state at each time step and the map.

\[
\begin{align*}
\text{veh} & = \text{Vehicle}(V); \\
\text{veh}.\text{add}\_\text{driver}(\text{RandomPath}(20, 2)); \\
\text{map} & = \text{Map}(20); \\
\text{sensor} & = \text{RangeBearingSensor}(\text{veh}, \text{map}, W); \\
\text{ekf} & = \text{EKF}(\text{veh}, V_{\text{est}}, P_0, \text{sensor}, W, []);
\end{align*}
\]

We run the simulation for 1000 time steps

\[
\text{ekf}.\text{run}(1000);
\]

then plot the map and the true vehicle path

\[
\begin{align*}
\text{map}.\text{plot}(); \\
\text{veh}.\text{plot}\_\text{xy}('b');
\end{align*}
\]

and overlay the estimated path

\[
\begin{align*}
\text{ekf}.\text{plot}\_\text{xy}('r');
\end{align*}
\]

and overlay uncertainty ellipses at every 20 time steps

\[
\begin{align*}
\text{ekf}.\text{plot}\_\text{ellipse}([], 'g');
\end{align*}
\]

We can plot the covariance against time as

\[
\begin{align*}
\text{clf} \\
\text{ekf}.\text{plot}\_\text{P}();
\end{align*}
\]

Then plot the true map

\[
\begin{align*}
\text{map}.\text{plot}();
\end{align*}
\]

and overlay the estimated map with 3 sigma ellipses

\[
\begin{align*}
\text{ekf}.\text{plot}\_\text{map}(3, 'g');
\end{align*}
\]

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

\[ E = \text{EKF}(\text{vehicle}, \ V\_\text{EST}, \ p0, \ \text{options}) \]

is an EKF that estimates the state of the vehicle with estimated odometry covariance \( V\_\text{EST} \) (2 \( \times \) 2) and initial covariance (3 \( \times \) 3).

\[ E = \text{EKF}(\text{vehicle}, \ V\_\text{EST}, \ p0, \ \text{sensor, W\_EST, map, options}) \]

as above but uses information from a vehicle mounted sensor, estimated sensor covariance \( W\_\text{EST} \) and a map.

Options

- ‘verbose’ Be verbose.
- ‘nohistory’ Don’t keep history.
- ‘joseph’ Use Joseph form for covariance.

Notes

- If \( \text{map} \) is [] then it will be estimated.
- If \( V\_\text{EST} \) and \( p0 \) are [] the vehicle is assumed error free and the filter will only estimate the landmark positions (map).
- If \( V\_\text{EST} \) 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.

See also

Vehicle, Sensor, RangeBearingSensor, Map

EKF.char

Convert to string

\[ E.\text{char}() \]

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
CHAPTER 2. FUNCTIONS AND CLASSES

**EKF.plot_map**

Plot landmarks

E.plot_map(i) overlay the current plot with the estimated landmark position (a +-marker) and a covariance ellipses for i points along the path.

E.plot_map() as above but i=20.

E.plot_map(i, ls) as above but pass line style arguments ls to plot.ellipse.

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 × 3) where each row is x, y, theta.

See also

EKF.plot.ellipse, EKF.plot_P

Robotics Toolbox 9.5 for MATLAB® 37 Copyright ©Peter Corke 2012
CHAPTER 2. FUNCTIONS AND CLASSES

EKF.run

Run the filter

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

Notes

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

---

eul2jac

Euler angle rate Jacobian

\[ J = \text{eul2jac}(\text{eul}) \]

is a Jacobian matrix \((3 \times 3)\) that maps Euler angle rates to angular velocity at the operating point \(\text{eul}=[\text{PHI}, \text{THETA}, \text{PSI}]\).

\[ J = \text{eul2jac}(\phi, \theta, \psi) \]

as above but the Euler angles are passed as separate arguments.

Notes

- Used in the creation of an analytical Jacobian.

See also
rpy2jac, SERIALIINK.JACOBN

---

eul2r

Convert Euler angles to rotation matrix

\[ R = \text{eul2r}(\phi, \theta, \psi, \text{options}) \]

is an orthonormal 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 \(\phi, \theta, \psi\).
\[ R = \text{eul2r}(\text{eul}, \text{options}) \] as above but the Euler angles are taken from consecutive columns of the passed matrix \( \text{eul} = [\phi \theta \psi] \).

**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**

\[ T = \text{eul2tr}(\phi, \theta, \psi, \text{options}) \] 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 \( \phi, \theta, \psi \).

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

**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

kernel

\[ k = \text{gauss2d}(\text{im}, c, \sigma) \]

Returns a unit volume Gaussian smoothing kernel. The Gaussian has a standard deviation of \( \sigma \), and the convolution kernel has a half size of \( w \), that is, \( k \) is \((2W+1) \times (2W+1)\).

If \( w \) is not specified it defaults to \( 2^*\sigma \).

h2e

Homogeneous to Euclidean

homline

Homogeneous line from two points

\[ L = \text{homline}(x_1, y_1, x_2, y_2) \]

returns a \( 3 \times 1 \) vectors which describes a line in homogeneous form that contains the two Euclidean points \((x_1,y_1)\) and \((x_2,y_2)\).

Homogeneous points \( X \) \((3 \times 1)\) on the line must satisfy \( L^*X = 0 \).

See also
plot_homline
CHAPTER 2. FUNCTIONS AND CLASSES

**homtrans**

**Apply a homogeneous transformation**

\[ p_2 = \text{homtrans}(T, p) \] applies homogeneous transformation \( T \) to the points stored columnwise in \( p \).

- If \( T \) is in SE(2) (3 x 3) and
  - \( p \) is 2 x \( N \) (2D points) they are considered Euclidean (\( \mathbb{R}^2 \))
  - \( p \) is 3 x \( N \) (2D points) they are considered projective (\( p^2 \))

- If \( T \) is in SE(3) (4 x 4) and
  - \( p \) is 3 x \( N \) (3D points) they are considered Euclidean (\( \mathbb{R}^3 \))
  - \( p \) is 4 x \( N \) (3D points) they are considered projective (\( p^3 \))

\[ t_p = \text{homtrans}(T, T_1) \] applies homogeneous transformation \( T \) to the homogeneous transformation \( T_1 \), that is \( t_p=T^*T_1 \). If \( T_1 \) is a 3-dimensional transformation then \( T \) is applied to each plane as defined by the first two dimensions, i.e. if \( T = N \times N \) and \( T_1 = N \times N \times p \) then the result is \( N \times N \times p \).

**See also**

`e2h`, `h2e`

---

**imeshgrid**

**Domain matrices for image**

\[ [u,v] = \text{imeshgrid}(\text{im}) \] return matrices that describe the domain of image \( \text{im} \) and can be used for the evaluation of functions over the image. The element \( u(v,u) = u \) and \( v(v,u) = v \).

\[ [u,v] = \text{imeshgrid}(w, H) \] as above but the domain is \( w \times H \).

\[ [u,v] = \text{imeshgrid}(\text{size}) \] as above but the domain is described \( \text{size} \) which is scalar \( \text{size} \times \text{size} \) or a 2-vector \([w H]\).

**See also**

`meshgrid`
ishomog

Test if argument is a homogeneous transformation

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

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

See also

isrot, isvec

isrot

Test if argument is a rotation matrix

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

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

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

**jtraj**

**Compute a joint space trajectory between two points**

\([q, qd, qdd] = \text{jtraj}(q0, qf, m)\) 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.

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

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

\([q, qd, qdd] = \text{jtraj}(q0, qf, T, qd0, qdf)\) 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.
CHAPTER 2. FUNCTIONS AND CLASSES

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 $3 \times 1$
- I  dynamic: link inertia matrix, symmetric $3 \times 3$, about link COG.
- B  dynamic: link viscous friction (motor referred)
- Tc  dynamic: link Coulomb friction
- G  actuator: gear ratio
- Jm  actuator: motor inertia (motor referred)

Notes

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

References


See also

SerialLink, Link.Link
CHAPTER 2. FUNCTIONS AND CLASSES

**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(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.

**Examples**

A standard Denavit-Hartenberg link

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

the flag 'standard' is not strictly necessary but adds clarity.

For a modified Denavit-Hartenberg link

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.

Link.A

Link transform matrix

\[ T = L.A(q) \] is the link homogeneous transformation matrix \((4 \times 4)\) 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

\[ f = L.\text{friction}(qd) \] 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.
CHAPTER 2. FUNCTIONS AND CLASSES

**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

$L_\text{c} = L_.\text{nofriction}()$ is a link object with the same parameters as $L$ except nonlinear (Coulomb) friction parameter is zero.

$L_\text{c} = L_.\text{nofriction}('all')$ as above except that viscous and Coulomb friction are set to zero.
\( \text{ln} = \text{L.nofriction('coulomb')} \) as above except that Coulomb friction is set to zero.

\( \text{ln} = \text{L.nofriction('viscous')} \) 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**

\( c = \text{L.RP()} \) 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 \( M (3 \times 3) \) which must be symmetric.

---

**Link.set.r**

**Set centre of gravity**

\( L.r = R \) set the link centre of gravity (COG) to \( R (3\text{-vector}) \).
**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 × 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 × 1) and sdd (m × 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 × 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
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

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

which can be displayed by

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

Reference

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

See also

RangeBearingSensor, EKF
CHAPTER 2. FUNCTIONS AND CLASSES

Map.Map

Map of point feature landmarks

\[ m = \text{Map}(n, \text{dim}, \text{options}) \]

is a \text{Map} object that represents \( n \) random point features in a planar region bounded by +/-\text{dim} in the x- and y-directions.

Options

‘verbose’ Be verbose

Map.char

Convert vehicle parameters and state to a string

\[ s = \text{M.char}() \]

is a string showing map parameters in a compact human readable format.

Map.display

Display map parameters

\[ \text{M.display}() \]

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

\[ f = \text{M.feature}(k) \]

is the coordinate \((2 \times 1)\) 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_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:
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_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_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**


**See also**

SerialLink, 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 characteristics 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 characteristics of a quadcopter.

**Properties**

This is a structure with the following elements:

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

**References**


**See also**

sl_quadcopter
**mdl_S4ABB2p8**

Create kinematic model of ABB S4 2.8 robot

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.
CHAPTER 2. FUNCTIONS AND CLASSES

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 6.4, 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:

\[ q_z \]

which 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
mstraj

Multi-segment multi-axis trajectory

\( \text{traj} = \text{mstraj}(p, qdmax, q0, dt, tacc, options) \) 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
- \( tacc \ (1 \times 1) \) 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 \( 1 \).

\( \text{traj} = \text{mstraj}(\text{segments}, qdmax, q0, dt, tacc, qd0, qdf, options) \) 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

\[[q, qd, qdd] = \text{mtraj}(\text{tfunc}, q0, qf, m)\]

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 \(\text{tfunc}\) 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 \(\text{tfunc}\)

\[[S, SD, SDD] = \text{TFUNC}(S0, SF, M)\]

and possible values of \(\text{tfunc}\) include \@lspb\ for a trapezoidal trajectory, or \@tpoly\ for a polynomial trajectory.

\[[q, qd, qdd] = \text{mtraj}(\text{tfunc}, q0, qf, T)\]

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 \(\text{tfunc}\) 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, mtraj, lspb, tpoly

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 overridden 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
CHAPTER 2. FUNCTIONS AND CLASSES

Navigation.Navigation

Create a Navigation object

\[ n = \text{Navigation}(\text{occgrid, options}) \] is a \text{Navigation} object that holds an occupancy grid \text{occgrid}. A number of options can 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

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

Navigation.display

Display status of navigation object

\text{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 overridden 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 overridden 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.

`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 $(m \times m)$ of random numbers.

R = N.rand(L,m) as above but return a matrix $(L \times m)$ 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**

\[ i = \text{N.randi}(rm) \]

return a uniformly distributed random integer in the range 1 to \( rm \) from a private random number stream.

\[ i = \text{N.randi}(rm, m) \]

as above but return a matrix \((m \times m)\) of random integers.

\[ i = \text{N.randn}(rm, L, m) \]

as above but return a matrix \((L \times m)\) 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**

\[ R = \text{N.randn}() \]

return a normally distributed random number from a private random number stream.

\[ R = \text{N.randn}(m) \]

as above but return a matrix \((m \times m)\) of random numbers.

\[ R = \text{N.randn}(L,m) \]

as above but return a matrix \((L \times m)\) of random numbers.
CHAPTER 2. FUNCTIONS AND CLASSES

Notes

- Accepts the same arguments as \texttt{randn()}.
- Seed is provided to \texttt{Navigation} constructor.

See also

\texttt{randn, randstream}

\texttt{Navigation.spinner}

Update progress spinner

\texttt{N.spinner()} displays a simple ASCII progress \texttt{spinner}, a rotating bar.

\texttt{Navigation.verbosity}

Set verbosity

\texttt{N.verbosity(v)} set \texttt{verbosity} to \texttt{v}, where 0 is silent and greater values display more information.

\texttt{numcols}

Return number of columns in matrix

\texttt{nc = numcols(m)} returns the number of columns in the matrix \texttt{m}.

See also

\texttt{numrows}
numrows

Return number of rows in matrix

nr = numrows(m) returns 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.
CHAPTER 2. FUNCTIONS AND CLASSES

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
- `dim` maximum xy dimension
Example

Create a landmark map

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

and a vehicle with odometry covariance and a driver

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

and create a range bearing sensor

```matlab
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 the covariance of the random noise added to the particle states at each iteration to represent uncertainty in configuration.

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

and the covariance of the likelihood function applied to innovation

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

Now construct the particle filter

```matlab
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

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

then plot the map and the true vehicle path

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

and overlay the mean of the particle cloud

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

We can plot the standard deviation against time

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

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

```matlab
pf.plot_pdf()
```

Acknowledgement

Based on code by Paul Newman, Oxford University, http://www.robots.ox.ac.uk/pnewman/
CHAPTER 2. FUNCTIONS AND CLASSES

Reference

Robotics, Vision & Control, Peter Corke, Springer 2011

See also

Vehicle, RandomPath, RangeBearingSensor, Map, EKF

### ParticleFilter

**Particle filter constructor**

\[
\text{pf} = \text{ParticleFilter} (\text{vehicle}, \text{sensor}, q, L, np, \text{options})
\]

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.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) runs the filter for n time steps.

Notes

- All previously estimated states and estimation history is cleared.
**PGraph**

**Graph class**

\[
g = \text{PGraph()} \quad \text{create a 2D, planar, undirected graph}
\]

\[
g = \text{PGraph}(n) \quad \text{create an n-d, undirected graph}
\]

Provides support for graphs that:

- are undirected
- 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**

\[
g.\text{add\_node}(\text{coord}) \quad \text{add vertex, return vid}
\]

\[
g.\text{add\_edge}(v1, v2) \quad \text{add edge from v1 to v2, return eid}
\]

\[
g.\text{setcost}(e, c) \quad \text{set cost for edge e}
\]

\[
g.\text{setdata}(v, u) \quad \text{set user data for vertex v}
\]

\[
g.\text{data}(v) \quad \text{get user data for vertex v}
\]

\[
g.\text{clear}() \quad \text{remove all vertices and edges from the graph}
\]

**Information from graph**

\[
g.\text{edges}(v) \quad \text{list of edges for vertex v}
\]

\[
g.\text{cost}(e) \quad \text{cost of edge e}
\]

\[
g.\text{neighbours}(v) \quad \text{neighbours of vertex v}
\]

\[
g.\text{component}(v) \quad \text{component id for vertex v}
\]

\[
g.\text{connectivity}() \quad \text{number of edges for all vertices}
\]

**Display**

\[
g.\text{plot}() \quad \text{set goal vertex for path planning}
\]

\[
g.\text{highlight\_node}(v) \quad \text{highlight vertex v}
\]

\[
g.\text{highlight\_edge}(e) \quad \text{highlight edge e}
\]

\[
g.\text{highlight\_component}(c) \quad \text{highlight all nodes in component c}
\]

\[
g.\text{highlight\_path}(p) \quad \text{highlight nodes and edge along path p}
\]

\[
g.\text{pick}(\text{coord}) \quad \text{vertex closest to coord}
\]
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
s.distance(v1, v2)  distance between v1 and v2
s.distances(coord)  return sorted distances from coord to all vertices
s.closest(coord)  vertex closest to coord

Object properties (read only)

g.n  number of vertices
s.ne  number of edges
s.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.

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 2π.
• 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 an edge between vertices with id v1 and 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.add_node**

Add a node

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

v = G.add_node(x, v2) as above but connected by an edge 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 \(a^N\) 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

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

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

Notes

- Uses the efficient A* search algorithm.
CHAPTER 2. FUNCTIONS AND CLASSES

References


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

\[ C = G.\text{component}(v) \] is the id of the graph component

**PGraph.connectivity**

Graph connectivity

\[ C = G.\text{connectivity}() \] is a vector \((N \times 1)\) with the number of edges per vertex.

The average vertex connectivity is

\[ \text{mean}(g.\text{connectivity}()) \]

and the minimum vertex connectivity is

\[ \text{min}(g.\text{connectivity}()) \]

**PGraph.coord**

Coordinate of node

\[ x = G.\text{coord}(v) \] is the coordinate vector \((D \times 1)\) of vertex id \(v\).

**PGraph.cost**

Cost of edge

\[ C = G.\text{cost}(E) \] is the cost of edge id \(E\).

**PGraph.data**

Get user data for node

\[ u = G.\text{data}(v) \] 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
CHAPTER 2. FUNCTIONS AND CLASSES

PGraph.distances

Distances from point to vertices

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

\[ \left[ d, w \right] = G.\text{distances}(p) \] as above but also returns \(w\) \((1 \times N)\) with the corresponding vertex id.

See also

PGraph.closest

PGraph.edges

Find edges given vertex

\( E = G.\text{edges}(v) \) return the id of all edges from vertex id \(v\).

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.
CHAPTER 2. FUNCTIONS AND CLASSES

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)
CHAPTER 2. FUNCTIONS AND CLASSES

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 \((N \times \text{\#E})\) 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.
CHAPTER 2. FUNCTIONS AND CLASSES

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.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

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

then the path, starting from vertex 1 is

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

and the path starting from vertex 2 is

```matlab
  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 size (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

\( \text{plot2}(p) \) 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.

\( \text{plot2}(p, ls) \) as above but the line style arguments \( ls \) are passed to plot.
See also

plot

---

**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 \( Z \).

Options

- ‘edgcolor’ 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.
**plot**

**ellipse**

`plot` ellipse

Draw an ellipse on the current plot

PLOT_ELLIPESE(a, ls) draws an ellipse defined by $X'AX = 0$ on the current plot, centred at the origin, with Matlab line style ls.

PLOT_ELLIPESE(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.

**plot**

**ellipse** _inv_

Plot an ellipse

`plot_ellipse(a, xc, ls)`

ls is the standard line styles.

**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 is $2 \times N$ 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.

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 × 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 transparent and 1 is opaque. The default is 1.

**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

Polygon

Polygon class

A general class for manipulating polygons and vectors of polygons.
CHAPTER 2. FUNCTIONS AND CLASSES

Methods

plot Plot polygon
area Area of polygon
moments Moments of polygon
centroid Centroid of polygon
perimeter Perimeter 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 polygon 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].
CHAPTER 2. FUNCTIONS AND CLASSES

Polygon.area

Area of polygon

\[ a = P.\text{area}() \] is the \textit{area} of the polygon.

Polygon.centroid

Centroid of polygon

\[ x = P.\text{centroid}() \] is the \textit{centroid} of the polygon.

Polygon.char

String representation

\[ s = P.\text{char}() \] is a compact representation of the polygon in human readable form.

Polygon.difference

Difference of polygons

\[ d = P.\text{difference}(q) \] 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

\[ P.\text{display}() \] displays the polygon in a compact human readable form.
CHAPTER 2. FUNCTIONS AND CLASSES

See also

Polygon.char

Polygon.inside

Test if points are inside polygon

\[
in = \text{p.inside}(p)
\]

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

\[
i = \text{P.intersect}(\text{plist})
\]

decides whether or not the \( \text{Polygon} \) \( P \) intersects with \( \text{Polygons} \) \( \text{plist} \). \( i(j) = 1 \) if \( p \) intersects \( \text{polylist}(j) \), else 0.

Polygon.intersect_line

Intersection of polygon and line segment

\[
i = \text{P.intersect_line}(L)
\]

is the intersection points of a polygon \( P \) with the line segment \( L = [x_1 x_2; y_1 y_2] \). \( i \) is an \( N \times 2 \) matrix with one column per intersection, each column is \( [x \ y]' \).

Polygon.intersection

Intersection of polygons

\[
i = \text{P.intersection}(q)
\]

is a \( \text{Polygon} \) 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.moments

Moments of polygon

\[ a = \text{P.moments}(p, q) \] is the pq\(^{th}\) moment of the polygon.

See also

mpq.poly

Polygon.perimeter

Perimeter of polygon

\[ L = \text{P.perimeter}() \] is the perimeter of the polygon.

Polygon.plot

Plot polygon

\[ \text{P.plot}() \] plot the polygon.
\[ \text{P.plot(Is)} \] as above but pass the arguments Is to plot.

Polygon.transform

Transformation of polygon vertices

\[ p2 = \text{P.transform}(T) \] is a new Polygon object whose vertices have been transformed by the \(3 \times 3\) homogeneus transformation \(T\).

Polygon.union

Union of polygons

\[ i = \text{P.union}(q) \] is a Polygon representing the union of polygons P and q.
CHAPTER 2. FUNCTIONS AND CLASSES

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”.

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
prm = PRM(map); % create navigation object
prm.plan() % create roadmaps
prm.path(start, goal) % animate path from this start location

References

- 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
CHAPTER 2. FUNCTIONS AND CLASSES

**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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘goal’</td>
<td>Superimpose the goal position if set</td>
</tr>
<tr>
<td>‘nooverlay’</td>
<td>Don’t overlay the PRM graph</td>
</tr>
</tbody>
</table>

Robotics Toolbox 9.5 for MATLAB® 97 Copyright ©Peter Corke 2012
CHAPTER 2. FUNCTIONS AND CLASSES

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: \(q = s \langle vx, vy, vz>\).

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

\[ q = \cos \left( \frac{\theta}{2} \right) \langle v \sin \left( \frac{\theta}{2} \right) \rangle \]

\(q = \text{quaternion}(x)\) is a unit-\text{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 quaternion
- `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 $3 \times 3$ rotation matrix
- `T` equivalent $4 \times 4$ homogeneous transform matrix

**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` product
- `q*v` rotate vector by `quaternion`, $v$ is $3 \times 1$
- `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**

CHAPTER 2. FUNCTIONS AND CLASSES

See also

trinterp, trplot

Quaternion.Quaternion

Constructor for quaternion objects

Construct a quaternion from various other orientation representations.

\( q = \text{Quaternion}() \) is the identity quaternion \( 1<0,0,0> \) representing a null rotation.

\( q = \text{Quaternion}(q1) \) is a copy of the quaternion \( q1 \)

\( q = \text{Quaternion}([S V1 V2 V3]) \) is a quaternion formed by specifying directly its 4 elements

\( q = \text{Quaternion}(s) \) is a quaternion formed from the scalar \( s \) and zero vector part: \( s<0,0,0> \)

\( q = \text{Quaternion}(v) \) is a pure quaternion with the specified vector part: \( 0<v> \)

\( q = \text{Quaternion}(\text{th}, v) \) is a unit-quaternion corresponding to rotation of \( \text{th} \) about the vector \( v \).

\( q = \text{Quaternion}(R) \) is a unit-quaternion corresponding to the orthonormal rotation matrix \( R \). If \( R \) (3 × 3 × \( N \)) is a sequence then \( q \) (\( N \times 1 \)) is a vector of Quaternions corresponding to the elements of \( R \).

\( q = \text{Quaternion}(T) \) is a unit-quaternion equivalent to the rotational part of the homogeneous transform \( T \). If \( T \) (4 × 4 × \( N \)) is a sequence then \( q \) (\( N \times 1 \)) is a vector of Quaternions corresponding to the elements of \( T \).

Quaternion.char

Convert to string

\( s = Q.\text{char}() \) 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

\( Q.\text{display}() \) 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.double

Convert a quaternion to a 4-element vector

\( v = Q\text{.double}() \) 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.interp**

Interpolate quaternions

\[ q_i = Q1\text{.interp}(q2, s) \]

is a unit quaternion that interpolates a rotation between \( Q1 \) for \( s=0 \) and \( q2 \) for \( s=1 \).

If \( s \) is a vector \( q_i \) is a vector of quaternions, each element corresponding to sequential elements of \( s \).

**Notes**

- This is a spherical linear interpolation (slerp) that can be interpreted as interpolation along a great circle arc on a sphere.
- The value of \( s \) is clipped to the interval 0 to 1.

**See also**

dtraj, Quaternion.scale

---

**Quaternion.inv**

Invert a unit quaternion

\[ q_i = Q\text{.inv}() \]

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 $\text{inv}(Q2)$.

$Q/S$ 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*

- \( Q_1 \times Q_2 \) is a quaternion formed by the Hamilton product of two quaternions.
- \( Q \times V \) is a vector formed by rotating the vector \( V \) by the quaternion \( Q \).
- \( Q \times S \) 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*

\( Q_1 = Q_2 \) is true if the quaternions \( Q_1 \) and \( Q_2 \) are not equal.

**Notes**

- Overloaded operator ‘\(\ne\)’
- Note that for unit Quaternions \( Q \) and \(-Q\) are the equivalent rotation, so non-equality does not mean rotations are not equivalent.
- If \( Q_1 \) is a vector of quaternions, each element is compared to \( Q_2 \) and the result is a logical array of the same length as \( Q_1 \).
- If \( Q_2 \) is a vector of quaternions, each element is compared to \( Q_1 \) and the result is a logical array of the same length as \( Q_2 \).
- If \( Q_1 \) and \( Q_2 \) are vectors of the same length, then the result is a logical array.

**See also**

*Quaternion.eq*
CHAPTER 2. FUNCTIONS AND CLASSES

Quaternion.norm

Quaternion magnitude

\[ q_n = q \cdot \text{norm}(q) \]

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.unit

Quaternion.plot

Plot a quaternion object

\[ Q \cdot \text{plot}(\text{options}) \]

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 \( V = [\text{az el}] \) angles, or ‘auto’ for view toward origin of coordinate frame

See also

trplot

Quaternion.plus

Add quaternions

\[ Q_1 + Q_2 \]

is the element-wise sum of quaternion elements.
CHAPTER 2. FUNCTIONS AND CLASSES

Notes

• Overloaded operator ‘+’
• The result is not guaranteed to be a unit-quaternion.

See also

Quaternion.minus, Quaternion.mtimes

Quaternion.R

Convert to orthonormal rotation matrix

\( R = Q.R() \) is the equivalent \( 3 \times 3 \) orthonormal rotation matrix.

Notes:

• For a quaternion sequence returns a rotation matrix sequence.

Quaternion.scale

Interpolate rotations expressed by quaternion objects

\( q_i = Q.\text{scale}(s) \) 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 interpreted as interpolation along a great circle arc on a sphere.

If \( s \) is a vector \( q_i \) 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 interpreted as interpolation along a great circle arc on a sphere.

See also

ctraj, Quaternion.interp
CHAPTER 2. FUNCTIONS AND CLASSES

Quaternion.T

Convert to homogeneous transformation matrix

\[ T = Q \cdot T() \] is the equivalent \( 4 \times 4 \) homogeneous transformation matrix.

Notes:
- For a quaternion sequence returns a homogeneous transform matrix sequence
- Has a zero translational component.

Quaternion.unit

Unitize a quaternion

\[ qu = Q \cdot \text{unit}() \] 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 \( 2 \times 2 \) then \( T \) is \( 3 \times 3 \), or
  - if \( R \) is \( 3 \times 3 \) then \( T \) is \( 4 \times 4 \).
- Translational component is zero.
- For a rotation matrix sequence returns a homogeneous transform sequence.
CHAPTER 2. FUNCTIONS AND CLASSES

See also
t2r

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 \((2 \times 1)\) [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 1 m/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 a compact human readable format.
RandomPath.demand

Compute speed and heading to waypoint

\[ \text{[speed, steer]} = \text{R.demand()} \]

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 position given vehicle pose and observation
- Gx: Jacobian matrix dg/dxv
- Gz: Jacobian matrix dg/dz

Properties (read/write)

- R: measurement covariance matrix (2 x 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, R, 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 x 2) representing range and bearing covariance.
CHAPTER 2. FUNCTIONS AND CLASSES

Options

- 'range', xmax: maximum range of sensor
- 'range', [xmin xmax]: minimum and maximum range of sensor
- 'angle', TH: detection for angles between -TH to +TH
- 'angle', [THMIN THMAX]: detection for angles between 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

\[ p = S.g(x_v, z) \] is the world coordinate \((1 \times 2)\) of a feature given the sensor observation \(z \ (1 \times 2)\) and vehicle state \(x_v \ (3 \times 1)\).

See also

RangeBearingSensor.Gx, RangeBearingSensor.Gz

RangeBearingSensor.Gx

Jacobian \(dg/dx\)

\[ J = S.Gx(x_v, z) \] is the Jacobian \(dg/dx_v \ (2 \times 3)\) at the vehicle state \(x_v \ (3 \times 1)\) for sensor observation \(z \ (2 \times 1)\).

See also

RangeBearingSensor.g
RangeBearingSensor.Gz

Jacobian \( \frac{dg}{dz} \)

\( J = S.Gz(xv, z) \) is the Jacobian \( \frac{dg}{dz} \) \((2 \times 2)\) at the vehicle state \( xv \) \((3 \times 1)\) for sensor observation \( z \) \((2 \times 1)\).

See also

RangeBearingSensor.g

RangeBearingSensor.h

Landmark range and bearing

\( z = S.h(xv, J) \) is a sensor observation \((1 \times 2)\), range and bearing, from vehicle at pose \( xv \) \((1 \times 3)\) to the map feature \( K \).

\( z = S.h(xv, xf) \) as above but compute range and bearing to a feature at coordinate \( xf \).

Notes

- Supports vectorized operation where \( xv \) \((N \times 3)\) and \( z \) \((N \times 2)\).

See also


RangeBearingSensor.Hw

Jacobian \( \frac{dh}{dv} \)

\( J = S.Hw(xv, k) \) is the Jacobian \( \frac{dh}{dv} \) \((2 \times 2)\) at the vehicle state \( xv \) \((3 \times 1)\) for map feature \( k \).

See also

RangeBearingSensor.h
RangeBearingSensor.Hx

Jacobian dh/dxv

\[ J = S.Hx(x_v, k) \] returns the Jacobian \( dh/dx_v \) (2 \times 3) at the vehicle state \( x_v \) (3 \times 1) for map feature \( k \).

\[ J = S.Hx(x_v, x_f) \] as above but for a feature at coordinate \( x_f \).

See also

RangeBearingSensor.h

RangeBearingSensor.Hxf

Jacobian dh/dxf

\[ J = S.Hxf(x_v, k) \] is the Jacobian \( dh/dx_v \) (2 \times 2) at the vehicle state \( x_v \) (3 \times 1) for map feature \( k \).

\[ J = S.Hxf(x_v, x_f) \] as above but for a feature at coordinate \( x_f \) (1 \times 2).

See also

RangeBearingSensor.h

RangeBearingSensor.reading

Landmark range and bearing

\([z,k] = S.reading()\) is an observation of a random landmark where \( z=[R;\text{THETA}] \) 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 \( k=\text{NaN} \).

See also

RangeBearingSensor.h
CHAPTER 2. FUNCTIONS AND CLASSES

rotx
Rotation about X axis

$$R = \text{rotx}(\theta)$$ is a rotation matrix representing a rotation of \(\theta\) about the x-axis.

See also
roty, rotz, angvec2r

rotz
Rotation about Z axis

$$R = \text{rotz}(\theta)$$ is a rotation matrix representing a rotation of \(\theta\) about the z-axis.

See also
rotx, roty, angvec2r

roty
Rotation about Y axis

$$R = \text{roty}(\theta)$$ is a rotation matrix representing a rotation of \(\theta\) about the y-axis.

See also
rotx, rotz, angvec2r
CHAPTER 2. FUNCTIONS AND CLASSES

**rpy2jac**

Jacobian from RPY angle rates to angular velocity

\[ \mathbf{J} = \text{rpy2jac}(\text{eul}) \]

is a Jacobian matrix \((3 \times 3)\) that maps roll-pitch-yaw angle rates to angular velocity at the operating point \(\text{RPY}=[R,P,Y]\).

\[ \mathbf{J} = \text{rpy2jac}(\mathbf{R}, \mathbf{p}, \mathbf{y}) \]

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

\[ \mathbf{R} = \text{rpy2r}(\text{rpy}, \text{options}) \]

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 \(\text{rpy}\) has multiple rows they are assumed to represent a trajectory and \(\mathbf{R}\) is a three dimensional matrix, where the last index corresponds to the rows of \(\text{rpy}\).

\[ \mathbf{R} = \text{rpy2r}(\text{roll}, \text{pitch}, \text{yaw}, \text{options}) \]

as above but the roll-pitch-yaw angles are passed as separate arguments. If \(\text{roll}, \text{pitch}\) and \(\text{yaw}\) are column vectors they are assumed to represent a trajectory and \(\mathbf{R}\) is a three dimensional matrix, where the last index corresponds to the rows of \(\text{roll}, \text{pitch}, \text{yaw}\).

**Options**

- `deg` Compute angles in degrees (radians default)
- `zyx` Return solution for sequential rotations about \(Z, Y, X\) axes (Paul book)
CHAPTER 2. FUNCTIONS AND CLASSES

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*

\[ T = \text{rpy2tr}(\text{rpy, options}) \]

is a homogeneous transformation equivalent to the specified roll, pitch, yaw angles which correspond to rotations about the X, Y, Z axes respectively. If \( \text{rpy} \) 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 \( \text{rpy} \).

\[ T = \text{rpy2tr}(\text{roll, pitch, yaw, options}) \]

as above but the roll-pitch-yaw angles are passed as separate arguments. If \( \text{roll, pitch} \) and \( \text{yaw} \) 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 \( \text{roll, pitch, yaw} \).

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
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

```matlab
rrt = RRT(); % create navigation object
rrt.plan() % create navigation tree
rrt.path(start, goal) % animate path from this start location
```

References

- Robotics, Vision & Control, Section 5.2.5, P. Corke, Springer 2011.

See also

Navigation, PRM, DXform, Dstar, PGraph

Create a RRT navigation object

```matlab
R = RRT.RRT(map, veh, options) is a rapidly exploring tree navigation object for a region with obstacles defined by the map object map.
```
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.

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 × 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.
CHAPTER 2. FUNCTIONS AND CLASSES

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 x 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 x M) formed from an orthonormal rotation matrix R (N x N) and a translation vector t (N x 1) where M=N+1.

For a sequence R (N x N x K) and t (kxN) results in a transform sequence (NxNxk).
CHAPTER 2. FUNCTIONS AND CLASSES

Notes

- Works for \( R \) in SO(2) or SO(3)
  - If \( R \) is \( 2 \times 2 \) and \( t \) is \( 2 \times 1 \), then \( TR \) is \( 3 \times 3 \)
  - If \( R \) is \( 3 \times 3 \) and \( t \) is \( 3 \times 1 \), then \( TR \) is \( 4 \times 4 \)
- The validity of \( R \) is not checked

See also

t2r, r2t, tr2rt

rtdemo

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 \(<\text{Enter}>\) in order to move through the explanation.
- Set PAUSE OFF if you want the scripts to run completely automatically.
CHAPTER 2. FUNCTIONS AND CLASSES

se2

Create planar translation and rotation transformation

\( T = se2(x, y, \theta) \) is a \( 3 \times 3 \) 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

<table>
<thead>
<tr>
<th>display</th>
<th>print the parameters in human readable form</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>convert to string</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>robot</th>
<th>The Vehicle object on which the sensor is mounted</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>The Map object representing the landmarks around the robot</td>
</tr>
</tbody>
</table>

Reference

Robotics, Vision & Control, Peter Corke, Springer 2011

See also

EKF, Vehicle, Map
Sensor.Sensor

Sensor object constructor

$s = \text{Sensor}(\text{vehicle, map})$ is a sensor mounted on the Vehicle object $\text{vehicle}$ and observing the landmark map $\text{map}$. $s = \text{Sensor}(\text{vehicle, map, R})$ is an instance of the Sensor object mounted on a vehicle represented by the object $\text{vehicle}$ and observing features in the world represented by the object $\text{map}$.

Sensor.char

Convert sensor parameters to a string

$s = \text{S.char}()$ 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

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.
CHAPTER 2. FUNCTIONS AND CLASSES

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
    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
    maniplty manipulability

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 x N)
    gravity direction of gravity [gx gy gz]
    base pose of robot's base (4 x 4 homog xform)
    tool robot’s tool transform, T6 to tool tip (4 x 4 homog xform)
    qlim joint limits, [qmin qmax] (N x 2)
    offset kinematic joint coordinate offsets (N x 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)
CHAPTER 2. FUNCTIONS AND CLASSES

Note

- SerialLink is a reference object.
- SerialLink objects can be used in vectors and arrays

Reference


See also

Link, DHFactor

SerialLink.SerialLink

Create a SerialLink robot object

\[
R = \text{SerialLink}(\text{links}, \text{options})\]

is a robot object defined by a vector of Link objects.

\[
R = \text{SerialLink}(\text{dh}, \text{options})\]

is a robot object with kinematics defined by the matrix \(\text{dh}\) 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 \((\sigma=0, \text{ default})\) or prismatic \((\sigma=1)\).

\[
R = \text{SerialLink}(\text{options})\]

is a null robot object with no links.

\[
R = \text{SerialLink}([\text{R1 R2 ...}], \text{options})\]

concatenate robots, the base of R2 is attached to the tip of R1.

\[
R = \text{SerialLink}(\text{R1}, \text{options})\]

is a deep copy of the robot object \(\text{R1}\), with all the same properties.

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
Examples

Create a 2-link robot

\[ L(1) = \text{Link}([0 \ 0 \ a1 \ 0], \text{’standard’}); \]
\[ L(2) = \text{Link}([0 \ 0 \ a2 \ 0], \text{’standard’}); \]
\[ \text{twolink} = \text{SerialLink}(L, \text{’name’, ’two link’}); \]

Robot objects can be concatenated in two ways

\[ R = R1 \ast R2; \]
\[ R = \text{SerialLink}([R1 \ R2]); \]

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, SerialLink.plot

---

**SerialLink.accel**

Manipulator forward dynamics

\[ \text{qdd} = R.\text{accel}(q, qd, \text{torque}) \]

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 \(q, qd, \text{torque}\) are matrices \((K \times N)\) then \(\text{qdd}\) is a matrix \((K \times N)\) where each row is the acceleration corresponding to the equivalent rows of \(q, qd, \text{torque}\).

\[ \text{qdd} = R.\text{accel}(x) \]

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


See also

SerialLink.mex, 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

- Not a general purpose method, 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

\[ C = R \cdot \text{coriolis}(q, qd) \]

\( C \) 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 \( C \cdot qd \) 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 interpreted 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 \( qd \).

\[ C = R \cdot \text{coriolis}(qqd) \]

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

\( R \cdot \text{display}() \) 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 T1, 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

\[ \text{Tau} = \text{torqfun}(T, q, qd, \text{ARG1, ARG2, ...}) \]

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 \texttt{R.nofriction()} method can be used to set this friction to zero.
- If \texttt{torqfun} is not specified, or is given as 0 or [], then zero torque is applied to the manipulator joints.
- The builtin integration function \texttt{ode45()} is used.

\textbf{See also}

\texttt{SerialLink.accel, SerialLink.nofriction, SerialLink.rne, ode45}

\textbf{SerialLink.fkine}

evaluate \texttt{fkine} for each point on a trajectory of theta\(_i\) or q\(_i\) data

\textbf{SerialLink.friction}

\textbf{Friction force}

\texttt{tau} = \texttt{R.friction(qd)} is the vector of joint \texttt{friction} forces/torques for the robot moving with joint velocities \texttt{qd}.

The \texttt{friction} model includes:

- viscous \texttt{friction} which is linear with velocity;
- Coulomb \texttt{friction} which is proportional to \texttt{sign(qd)}.

\textbf{See also}

\texttt{Link.friction}

\textbf{SerialLink.gravload}

\textbf{Gravity loading}

\texttt{taug} = \texttt{R.gravload(q)} is the joint gravity loading for the robot in the joint configuration \texttt{q}. Gravitational acceleration is a property of the robot object.
If \( \mathbf{q} \) is a row vector, the result is a row vector of joint torques. If \( \mathbf{q} \) 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.

\[
\mathbf{tau}_g = \mathbf{R.\text{gravload}}(\mathbf{q}, \mathbf{grav})
\]
is as above but the gravitational acceleration vector \( \mathbf{grav} \) is given explicitly.

See also

SerialLink.mne, SerialLink.itorque, SerialLink.coriolis

**SerialLink.ikine**

*default parameters for solution*

**SerialLink.ikine3**

*Inverse kinematics for 3-axis robot with no wrist*

\( \mathbf{q} = \mathbf{R.\text{ikine3}}(\mathbf{T}) \) is the joint coordinates corresponding to the robot end-effector pose \( \mathbf{T} \) represented by the homogenous transform. This is a analytic solution for a 3-axis robot (such as the first three joints of a robot like the Puma 560).

\( \mathbf{q} = \mathbf{R.\text{ikine3}}(\mathbf{T}, \text{config}) \) 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 \( \mathbf{q} \).

**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

$q = \text{R.ikine6s}(T)$ is the joint coordinates corresponding to the robot end-effector pose $T$ represented by the homogenous transform. This is a analytic solution for a 6-axis robot with a spherical wrist (such as the Puma 560).

$q = \text{R.IKINE6S}(T, \text{config})$ 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

- Only applicable for an all revolute 6-axis robot RRRRRR.
- 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


Author

Robert Biro with Gary Von McMurray, GTRI/ATRP/IIMB, Georgia Institute of Technology 2/13/95
CHAPTER 2. FUNCTIONS AND CLASSES

See also
SerialLink.FKINE, SerialLink.IKINE

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 interpreted 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.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...
CHAPTER 2. FUNCTIONS AND CLASSES

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.itorque

Inertia torque

\[ \tau_{\text{ui}} = R.\text{itorque}(q, \text{qdd}) \]

is the inertia force/torque vector \((1 \times N)\) at the specified joint configuration \(q\) \((1 \times N)\) and acceleration \(\text{qdd}\) \((1 \times N)\), that is, \(\tau_{\text{ui}} = \text{INERTIA}(q) \ast \text{qdd}\).

If \(q\) and \(\text{qdd}\) are matrices \((K \times N)\), each row is interpreted 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.mne, SerialLink.inertia

SerialLink.jacob0

Jacobian in world coordinates

\(j_0 = R.\text{jacob0}(q, \text{options})\)

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 \(V = j_0 \ast QD\) 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, jingu

SerialLink.jacob_dot

Derivative of Jacobian

\[ \mathbf{jdq} = \mathbf{R}.\text{jacob dot}(\mathbf{q}, \mathbf{qd}) \]

is the product \((6 \times 1)\) of the derivative of the Jacobian (in the world frame) and the joint rates.

Notes

- Useful for operational space control \(\mathbf{XDD} = \mathbf{J}(\mathbf{q})\mathbf{QDD} + \mathbf{JDOT}(\mathbf{q})\mathbf{qd}\)
- Written as per the text and not very efficient.

References


See also

: SerialLink.jacob0, diff2tr, tr2diff
CHAPTER 2. FUNCTIONS AND CLASSES

SerialLink.jacobn

Jacobian in end-effector frame

\(\mathbf{j}_n = R.\text{jacobn}(\mathbf{q}, \text{options})\) is the Jacobian matrix \((6 \times N)\) for the robot in pose \(\mathbf{q}\). The manipulator Jacobian matrix maps joint velocity to end-effector spatial velocity \(\mathbf{V} = \mathbf{j}_n^* \mathbf{QD}\) 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


See also

SerialLink.jacob0, jsingu, delta2tr, tr2delta

SerialLink.jtraj

Joint space trajectory

\(\mathbf{q} = R.\text{jtraj}(\mathbf{T}_1, \mathbf{t}_2, \mathbf{k})\) is a joint space trajectory \((\mathbf{k} \times N)\) where the joint coordinates reflect motion from end-effector pose \(\mathbf{T}_1\) to \(\mathbf{t}_2\) in \(\mathbf{k}\) steps with default zero boundary conditions for velocity and acceleration. The trajectory \(\mathbf{q}\) 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. \(R.\text{ikine6s}()\) is used if appropriate, else \(R.\text{ikine}()\). Additional trailing arguments to \(R.\text{jtraj}()\) are passed as trailing arguments to these functions.
See also

jtraj, SerialLink.ikine, SerialLink.ikine6s

SerialLink.maniplty

Manipulability measure

\( \mathbf{m} = \text{R.maniplty}(\mathbf{q}, \text{options}) \) is the manipulability index measure for the robot at the joint configuration \( \mathbf{q} \). 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 \( \mathbf{q} \) is a matrix (\( m \times N \)) then \( \mathbf{m} (m \times 1) \) is a vector of manipulability indices for each pose specified by a row of \( \mathbf{q} \).

\([\mathbf{m}, \mathbf{c}] = \text{R.maniplty}(\mathbf{q}, \text{options})\) as above, but for the case of the Asada measure returns the Cartesian inertia matrix \( \mathbf{c} \).

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 translational motion only
- ‘R’ manipulability for rotational motion only
- ‘yoshikawa’ use Yoshikawa algorithm (default)
- ‘asada’ use Asada algorithm

Notes

- By default the measure 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.
References


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

\( \text{rnf} = \text{R.nofriction()} \) is a robot object with the same parameters as R but with non-linear (Coulomb) friction coefficients set to zero.

\( \text{rnf} = \text{R.nofriction(’all’) as above but all friction coefficients set to zero.} \)

\( \text{rnf} = \text{R.nofriction(’viscous’) as above but only viscous friction coefficients are set to zero.} \)

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’.
CHAPTER 2. FUNCTIONS AND CLASSES

See also

SerialLink.fdyn, Link.nofriction

SerialLink.payload

Add payload mass

\( R.p\text{e}\text{a}\text{l}\text{a}\text{d}(m, p) \) 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

\( rp = R.p\text{e}\text{r}\text{t}\text{u}\text{b}(p) \) 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:

\[
\begin{align*}
  r2 &= p560.p\text{e}\text{r}\text{t}\text{u}\text{b}(0.1); \\
\end{align*}
\]

SerialLink.plot

Graphical display and animation

\( R.p\text{l}\text{o}\text{t}(q, \text{ options}) \) 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, \( W = [x_{mn}, x_{mx} y_{mn} y_{mx} z_{mn} z_{mx}] \)
- `delay`, \( d \) delay between frames for animation (s)
- `fps`, \( fps \) set number of frames per second for display
- `[no]loop` loop over the trajectory forever
- `mag`, scale annotation scale factor
- `cylinder`, \( C \) color for joint cylinders, \( C = [r \ g \ b] \)
- `ortho` orthogonal camera view (default)
- `perspective` perspective camera view
- `xyz` wrist axis label is XYZ
- `noa` wrist axis label is NOA
- `[no]raise` autoraise the figure (very slow).
- `[no]render` controls shaded rendering after drawing
- `[no]base` controls display of base `pedestal`
- `[no]wrist` controls display of wrist
- `[no]shadow` controls display of shadow
- `[no]name` display the robot’s name
- `[no]axes` control display of joint axes
- `[no]joints` controls display of joints

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 teh 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

```matlab
figure(1)
p560.plot(qz);
```

Create a robot in figure 2

```matlab
figure(2)
p560.plot(qz);
```

Now move both robots

```matlab
p560.plot(qn)
```

Multiple robots in the same figure

Multiple robots can be displayed in the same \texttt{plot}, by using “hold on” before calls to \texttt{robot.plot()}.

Create a robot in figure 1

```matlab
figure(1)
p560.plot(qz);
```

Make a clone of the robot named bob

```matlab
bob = SerialLink(p560, ’name’, ’bob’);
```

Draw bob in this figure

```matlab
hold on
bob.plot(qn)
```

To animate both robots so they move together:

```matlab
qtg = jtraj(qr, qz, 100);
for q=qtg’
p560.plot(q’);
bob.plot(q’);
end
```

Notes

• Delay between 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 which can be retrieved using \( q = \text{robot.plot()} \).

See also
plotbotopt, SerialLink.animate, SerialLink.fkine

SerialLink.plot_options

a cell array of options and return a struct

SerialLink.rne

Inverse dynamics

\[
\tau = R.\text{rne}(q, \, qd, \, qdd) \]

is the joint torque required for the robot \( R \) to achieve the specified joint position \( q \), velocity \( qd \) and acceleration \( qdd \).

\[
\tau = R.\text{rne}(q, \, qd, \, qdd, \, \text{grav}) \]

as above but overriding the gravitational acceleration vector in the robot object \( R \).

\[
\tau = R.\text{rne}(q, \, qd, \, qdd, \, \text{grav}, \, \text{fext}) \]

as above but specifying a wrench acting on the end of the manipulator which is a 6-vector \([Fx \, Fy \, Fz \, My \, My \, Mz]\).

\[
\tau = R.\text{rne}(x) \]

as above where \( x = [q, qd, qdd] \).

\[
\tau = R.\text{rne}(x, \, \text{grav}) \]

as above but overriding the gravitational acceleration vector in the robot object \( R \).

\[
\tau = R.\text{rne}(x, \, \text{grav}, \, \text{fext}) \]

as above but specifying a wrench acting on the end of the manipulator which is a 6-vector \([Fx \, Fy \, Fz \, My \, My \, Mz]\).

\[
[\tau, \text{wbase}] = R.\text{rne}(x, \, \text{grav}, \, \text{fext}) \]

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.

Notes

• The robot base transform is ignored.
• The torque computed contains a contribution due to armature inertia and joint friction.
• \texttt{rne} can be either an M-file or a MEX-file.
• 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 \texttt{rne.DH} or \texttt{rne.MDH} depending on the kinematic conventions used by the robot object.
• Currently the MEX-file version does not compute \texttt{wbase}.

See also

\texttt{SerialLink.accel}, \texttt{SerialLink.gravload}, \texttt{SerialLink.inertia}

\section*{SerialLink.teach}

Graphical teach pendant

\texttt{R.teach(options)} 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.

\subsection*{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)
- `degrees` Display angles in degrees (default radians)
- `q0`, `q` Set initial joint coordinates

\subsection*{GUI}

- The record button adds the current joint coordinates as a row to the robot’s qteach property.
- The Quit button destroys the \texttt{teach} window.

\subsection*{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.teach_callback

on changes to a slider or to the edit box showing joint coordinate

`src` the object that caused the event

`name` name of the robot

`j` the joint index concerned (1..N)

`slider` true if the

skew

Create skew-symmetric matrix

`s = skew(v)` is a skew-symmetric matrix formed from $v$ ($3 \times 1$).

```
| 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 \( 4 \times 4 \), then \( R \) is \( 3 \times 3 \).
  - If \( T \) is \( 3 \times 3 \), then \( R \) is \( 2 \times 2 \).
- 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

\([\text{optout}, \text{args}] = \text{TB_OPTPARSE}(	ext{opt, arglist})\) 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:

```matlab
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’. 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 (extremely 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

\[[s, sd, sdd] = \texttt{tpoly}(s0, sf, m)\] 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 × 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 × 1) about the axis v (1 × 3).

[theta, v] = tr2angvec(T) as above but uses the rotational part of the homogeneous transform T.

If R (3 × 3 × K) or T (4 × 4 × K) represent a sequence then theta (K × 1) is a vector of angles for corresponding elements of the sequence and v (K × 3) are the corresponding axes, one per row.

Notes
- If no output arguments are specified the result is displayed.
- This algorithm is from Paul 1981, other solutions are possible using eigenvectors or Rodriguez formula.

See also
angvec2r, angvec2tr

tr2delta

Convert homogeneous transform to differential motion

d = tr2delta(T0, T1) is the differential motion (6 × 1) corresponding to infinitesimal 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.

\[ d = \text{tr2delta}(T) \] 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 \( T_0 T_1 \) or \( T = \text{eye}(4,4) \).

**See also**

delta2tr, skew

---

**tr2eul**

**Convert homogeneous transform to Euler angles**

\[ \text{eul} = \text{tr2eul}(T, \text{options}) \] are the \( XYZ \) Euler angles expressed as a row vector corresponding to the rotational part of a homogeneous transform \( T \). The 3 angles \( \text{eul} = [\text{PHI}, \text{THETA}, \text{PSI}] \) correspond to sequential rotations about the \( Z \), \( Y \) and \( Z \) axes respectively.

\[ \text{eul} = \text{tr2eul}(R, \text{options}) \] are the \( XYZ \) 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 \( \text{eul} \) corresponds to a step of the trajectory.

**Options**

- **‘deg’** Compute angles in degrees (radians default)

**Notes**

- There is a singularity for the case where \( \text{THETA}=0 \) in which case \( \text{PHI} \) is arbitrarily set to zero and \( \text{PSI} \) is the sum \( \text{PHI}+\text{PSI} \).
See also
eul2tr, tr2rpy

**tr2jac**

Jacobian for differential motion

\[ J = \text{tr2jac}(T) \] is a Jacobian matrix \((6 \times 6)\) 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

\[ \text{rpy} = \text{tr2rpy}(T, \text{options}) \] are the roll-pitch-yaw angles expressed as a row vector corresponding to the rotation part of a homogeneous transform \(T\). The 3 angles \(\text{rpy} = [R, P, Y]\) correspond to sequential rotations about the X, Y and Z axes respectively.

\[ \text{rpy} = \text{tr2rpy}(R, \text{options}) \] 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 \(\text{rpy}\) 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)
CHAPTER 2. FUNCTIONS AND CLASSES

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

$[R, t] = \text{tr2rt}(TR)$ 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 $4 \times 4$, then $R$ is $3 \times 3$ and $T$ is $3 \times 1$.
  - If $TR$ is $3 \times 3$, then $R$ is $2 \times 2$ and $T$ is $2 \times 1$.
- The validity of $R$ is not checked.

See also

rt2tr, r2t, t2r
CHAPTER 2. FUNCTIONS AND CLASSES

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 × 4 × N)
- orthonormal rotation matrix sequence (3 × 3 × N)
- Quaternion vector (N × 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]

See also

trplot

transl

Create translational transform

T = transl(x, y, z) is a homogeneous transform representing a pure translation.

T = transl(p) is a homogeneous transform representing a translation or point p=[x,y,z].

If p (M × 3) it represents a sequence and T (4 × 4 × 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 homogenous transform as a 3-element column vector. If T (4 × 4 × M) is a homogenoeus transform sequence the rows of p
(M × 3) 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
ctraj

---

**trinterp**

Interpolate homogeneous transformations

\[ T = \text{trinterp}(T_0, T_1, s) \] is a homogeneous transform interpolation between \(T_0\) when \(s=0\) to \(T_1\) 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 \(s\).

\[ T = \text{trinterp}(T, s) \] 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

\[ t_n = \text{trnorm}(T) \] 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 \(N = O \times A\), and then we ensure that \(O\) and \(A\) are orthogonal by \(O = A \times N\).
Notes

• Used to prevent finite word length arithmetic causing transforms to become ‘un-normalized’.

See also

oa2tr

\[ \text{trotx}(\theta) \]

Rotation about X axis

\[ T = \text{trotx}(\theta) \]

is a homogeneous transformation \((4 \times 4)\) representing a rotation of \(\theta\) about the x-axis.

Notes

• Translational component is zero.

See also

rotx, troty, trotz

\[ \text{troty}(\theta) \]

Rotation about Y axis

\[ T = \text{troty}(\theta) \]

is a homogeneous transformation \((4 \times 4)\) representing a rotation of \(\theta\) about the y-axis.

Notes

• Translational component is zero.
See also
roty, trotx, trotz

trotz

Rotation about Z axis

$T = \text{trotz}(\theta)$ is a homogeneous transformation $(4 \times 4)$ representing a rotation of $\theta$ about the z-axis.

Notes

• Translational component is zero.

See also
rotz, trotx, troty

trplot

Draw a coordinate frame

$\text{trplot}(T, \text{options})$ draws a 3D coordinate frame represented by the homogeneous transform $T$ $(4 \times 4)$.

$H = \text{trplot}(T, \text{options})$ as above but returns a handle.

$\text{trplot}(H, T)$ moves the coordinate frame described by the handle $H$ to the pose $T$ $(4 \times 4)$.

$\text{trplot}(R, \text{options})$ draws a 3D coordinate frame represented by the orthonormal rotation matrix $R$ $(3 \times 3)$.

$H = \text{trplot}(R, \text{options})$ as above but returns a handle.

$\text{trplot}(H, R)$ moves the coordinate frame described by the handle $H$ to the orientation $R$. 
CHAPTER 2. FUNCTIONS AND CLASSES

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
- **'arrow'**  Use arrows rather than line segments for the axes
- **'width', w**  Width of arrow tips
- **'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)

Examples

- `trplot(T, 'frame', 'A')`
- `trplot(T, 'frame', 'A', 'color', 'b')`
- `trplot(T, 'frame', 'A', 'text_opts', {'FontSize', 10, 'FontWeight', 'bold'})`
- `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
- '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_opt', {'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 homogeneous 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**

```plaintext
>> 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`

---

**unit**

Unitize a vector

`vn = unit(v)` is a unit vector parallel to `v`. 
Note

- Reports error for the case where \( \text{norm}(v) \) 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

**Properties (read/write)**

- `x` true vehicle state \((3 \times 1)\)
- `V` odometry covariance \((2 \times 2)\)
- `odometry` distance moved in the last interval \((2 \times 1)\)
- `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

\[
\mathbf{v} = \text{Vehicle}(\text{diag}([0.1 \ 0.01].^2));
\]

and display its initial state

\[
\mathbf{v}
\]

now apply a speed (0.2m/s) and steer angle (0.1rad) for 1 time step

\[
\mathbf{odo} = \mathbf{v}.\text{update}([0.2, 0.1])
\]

where odo is the noisy odometry estimate, and the new true vehicle state

\[
\mathbf{v}
\]

We can add a driver object

\[
\mathbf{v}.\text{add_driver}(\text{RandomPath}(10))
\]

which will move the vehicle within the region \(-10 < x < 10, -10 < y < 10\) which we can see by

\[
\mathbf{v}.\text{run}(1000)
\]

which shows an animation of the vehicle moving between randomly selected waypoints.

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

\[
\mathbf{v} = \text{Vehicle}(\mathbf{v}_{\text{ACT}}, \text{options})
\]

creates a Vehicle object with actual odometry covariance \(\mathbf{v}_{\text{ACT}} (2 \times 2)\) matrix corresponding to the odometry vector \([dx \ dtheta]\).
CHAPTER 2. FUNCTIONS AND CLASSES

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:

```matlab
[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 a compact human readable format.
See also
Vehicle.display

Vehicle.control

Compute the control input to vehicle

\[ u = V.\text{control}(\text{speed, steer}) \] returns a control input (speed,steer) based on provided controls speed,steer to which speed and steering angle limits have been applied.

\[ u = V.\text{control}() \] 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

\[ V.\text{display}() \] 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

\[ x_n = V.f(x, odo) \] predict next state \( x_n \) \((1 \times 3)\) based on current state \( x \) \((1 \times 3)\) and odometry \( odo \) \((1 \times 2)\) is [distance,change_heading].
\( x_n = V.f(x, odo, w) \) as above but with odometry noise \( w \).

**Notes**
- Supports vectorized operation where \( x \) and \( x_n (N \times 3) \).

**Vehicle.Fv**

**Jacobian \( df/dv \)**

\( J = V.Fv(x, odo) \) returns the Jacobian \( df/dv (3 \times 2) \) 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 \( df/dx (3 \times 3) \) 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 × 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)
CHAPTER 2. FUNCTIONS AND CLASSES

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

\[ p = V.run2(T, x0, \text{speed}, \text{steer}) \] runs the vehicle model for a time \( T \) with speed \text{speed} and steering angle \text{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

\[ odo = V.step(\text{speed}, \text{steer}) \] updates the vehicle state for one timestep of motion at specified \text{speed} and \text{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

\[ \text{odo} = V.\text{update}(u) \] is the true odometry value for motion with \( u = [\text{speed}, \text{steer}] \).

**Notes**

- Appends new state to state history property \( x\_\text{hist} \).
- Odometry is also saved as property odometry.

---

**Vehicle.verbosity**

Set verbosity

\[ \text{V.verbosity}(a) \] set verbosity to \( a \). \( a=0 \) means silent.

---

**vex**

Convert skew-symmetric matrix to vector

\[ v = vex(s) \] is the vector \((3 \times 1)\) which has the skew-symmetric matrix \( s \) \((3 \times 3)\)

\[
\begin{bmatrix}
0 & -v_z & v_y \\
v_z & 0 & -v_x \\
-v_y & v_x & 0
\end{bmatrix}
\]
CHAPTER 2. FUNCTIONS AND CLASSES

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. \( vx = 0.5 \times (s(3,2) - s(2,3)) \)

See also

skew

wtrans

Transform a wrench between coordinate frames

\[
wt = \text{wtrans}(T, w)
\]

is a wrench \((6 \times 1)\) in the frame represented by the homogeneous transform \(T\) \((4 \times 4)\) corresponding to the world frame wrench \(w\) \((6 \times 1)\).

The wrenches \(w\) and \(wt\) are 6-vectors of the form \([Fx Fy Fz Mx My Mz]\).

See also

tr2delta, tr2jac

xaxis

X-axis scaling

\[
xaxis(\text{max}) \quad xaxis([\text{min} \ \text{max}]) \quad xaxis(\text{min}, \text{max})
\]

\(xaxis\) restore automatic scaling for this axis