Sensor.display

Display status of sensor object

S.**display**() displays the state of the sensor object in human-readable form.

Notes

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

See also

Sensor.char

Sensor.plot

Plot sensor reading

S.**plot**(**J**) draws a line from the robot to the $\mathbf{J}^{\text{th}}$ map feature.

Notes

- The line is drawn using the linestyle given by the property ls
- There is a delay given by the property delay

SerialLink

Serial-link robot class

A concrete class that represents a serial-link arm-type robot. Each link and joint in the chain is described by a Link-class object using Denavit-Hartenberg parameters (standard or modified).

Constructor methods

SerialLink	general constructor
L1+L2	construct from Link objects

Display/plot methods

animate	animate robot model
display	print the link parameters in human readable form
dyn	display link dynamic parameters
edit	display and edit kinematic and dynamic parameters
getpos	get position of graphical robot
plot	display graphical representation of robot
plot3d	display 3D graphical model of robot
teach	drive the graphical robot

Testing methods

islimit	test if robot at joint limit
isconfig	test robot joint configuration
issym	test if robot has symbolic parameters
isprismatic	index of prismatic joints
isrevolute	index of revolute joints
isspherical	test if robot has spherical wrist

Conversion methods

char	convert to string
sym	convert to symbolic parameters
todegrees	convert joint angles to degrees
toradians	convert joint angles to radians

SerialLink.SerialLink

Create a SerialLink robot object

R = SerialLink(**links**, **options**) is a robot object defined by a vector of Link class objects which includes the subclasses Revolute, Prismatic, RevoluteMDH or PrismaticMDH.

R = SerialLink(**options**) is a null robot object with no links.

R = SerialLink([R1 R2 ...], **options**) concatenate robots, the base of R2 is attached to the tip of R1. Can also be written as R1*R2 etc.

R = SerialLink(**R1**, **options**) is a deep copy of the robot object **R1**, with all the same properties.

R = SerialLink(**dh**, **options**) is a robot object with kinematics defined by the matrix **dh** which has one row per joint and each row is [theta d a alpha] and joints are

assumed revolute. An optional fifth column sigma indicate revolute (sigma=0) or prismatic (sigma=1). An optional sixth column is the joint offset.

Options

'name', NAME	set robot name property to NAME
'comment', COMMENT	set robot comment property to COMMENT
'manufacturer', MANUF	set robot manufacturer property to MANUF
'base', T	set base transformation matrix property to T
'tool', T	set tool transformation matrix property to T
'gravity', G	set gravity vector property to G
'plotopt', P	set default **options** for .plot() to P
'plotopt3d', P	set default **options** for .plot3d() to P
'nofast'	don't use RNE MEX file

Examples

Create a 2-link robot

```
L(1) = Link([ 0     0   a1  pi/2], 'standard');
L(2) = Link([ 0     0   a2  0], 'standard');
twolink = SerialLink(L, 'name', 'two link');
```

Create a 2-link robot (most descriptive)

```
L(1) = Revolute('d', 0, 'a', a1, 'alpha', pi/2);
L(2) = Revolute('d', 0, 'a', a2, 'alpha', 0);
twolink = SerialLink(L, 'name', 'two link');
```

Create a 2-link robot (least descriptive)

```
twolink = SerialLink([0 0 a1 0; 0 0 a2 0], 'name', 'two link');
```

Robot objects can be concatenated in two ways

```
R = R1 * R2;
R = SerialLink([R1 R2]);
```

Note

- SerialLink is a reference object, a subclass of Handle object.
- SerialLink objects can be used in vectors and arrays
- Link subclass elements passed in must be all standard, or all modified, **dh** parameters.
- When robots are concatenated (either syntax) the intermediate base and tool transforms are removed since general constant transforms cannot be represented in Denavit-Hartenberg notation.

See also

Link, Revolute, Prismatic, RevoluteMDH, PrismaticMDH, SerialLink.plot

SerialLink.A

Link transformation matrices

s = R.A(**J**, **q**) is an SE3 object (4×4) that transforms between link frames for the $\mathbf{J}^{\text{th}}$ joint. **q** is a vector ($1 \times N$) of joint variables. For:

- standard DH parameters, this is from frame {**J**-1} to frame {**J**}.
- modified DH parameters, this is from frame {**J**} to frame {**J**+1}.

s = R.A(**jlist**, **q**) as above but is a composition of link transform matrices given in the list **jlist**, and the joint variables are taken from the corresponding elements of **q**.

Exmaples

For example, the link transform for joint 4 is

```
robot.A(4, q4)
```

The link transform for joints 3 through 6 is

```
robot.A(3:6, q)
```

where q is 1×6 and the elements q(3) .. q(6) are used.

Notes

- Base and tool transforms are not applied.

See also

Link.A

SerialLink.accel

Manipulator forward dynamics

qdd = R.accel(**q**, **qd**, **torque**) is a vector ($N \times 1$) of joint accelerations that result from applying the actuator force/torque ($1 \times N$) to the manipulator robot R in state **q** ($1 \times N$) and **qd** ($1 \times N$), and N is the number of robot joints.

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

qdd = R.**accel**(**x**) as above but **x**=[**q**,**qd**,**torque**] ($1 \times 3N$).

Note

- Useful for simulation of manipulator dynamics, in conjunction with a numerical integration function.
- Uses the method 1 of Walker and Orin to compute the forward dynamics.
- Featherstone's method is more efficient for robots with large numbers of joints.
- Joint friction is considered.

References

- Efficient dynamic computer simulation of robotic mechanisms, M. W. Walker and D. E. Orin, ASME Journa of Dynamic Systems, Measurement and Control, vol. 104, no. 3, pp. 205-211, 1982.

See also

SerialLink.fdyn, SerialLink.rne, SerialLink, ode45

SerialLink.animate

Update a robot animation

R.**animate**(**q**) updates an existing animation for the robot R. This will have been created using R.plot(). Updates graphical instances of this robot in all figures.

Notes

- Called by plot() and plot3d() to actually move the arm models.
- Used for Simulink robot animation.

See also

SerialLink.plot

SerialLink.char

Convert to string

s = R.**char**() is a string representation of the robot's kinematic parameters, showing DH parameters, joint structure, comments, gravity vector, base and tool transform.

SerialLink.cinertia

Cartesian inertia matrix

m = R.**cinertia**(**q**) is the $N \times N$ Cartesian (operational space) inertia matrix which relates Cartesian force/torque to Cartesian acceleration at the joint configuration **q**, and N is the number of robot joints.

See also

SerialLink.inertia, SerialLink.rne

SerialLink.collisions

Perform collision checking

C = R.**collisions**(**q**, **model**) is true if the **SerialLink** object R at pose **q** ($1 \times N$) intersects the solid model **model** which belongs to the class CollisionModel. The model comprises a number of geometric primitives with an associated pose.

C = R.**collisions**(**q**, **model**, **dynmodel**, **tdyn**) as above but also checks dynamic collision model **dynmodel** whose elements are at pose **tdyn**. **tdyn** is an array of transformation matrices ($4 \times 4 \times P$), where P = length(**dynmodel**.primitives). The P^{th} plane of **tdyn** premultiplies the pose of the P^{th} primitive of **dynmodel**.

C = R.**collisions**(**q**, **model**, **dynmodel**) as above but assumes **tdyn** is the robot's tool frame.

Trajectory operation

If **q** is $M \times N$ it is taken as a pose sequence and **C** is $M \times 1$ and the collision value applies to the pose of the corresponding row of **q**. **tdyn** is 4x4xMxP.

Notes

- Requires the pHRIWARE package which defines CollisionModel class. Available from: https://github.com/bryan91/pHRIWARE .
- The robot is defined by a point cloud, given by its points property.
- The function does not currently check the base of the SerialLink object.
- If **model** is [] then no static objects are assumed.

Author

Bryan Moutrie

See also

CollisionModel, SerialLink

SerialLink.coriolis

Coriolis matrix

C = R.**coriolis**(**q**, **qd**) is the Coriolis/centripetal matrix ($N \times N$) for the robot in configuration **q** and velocity **qd**, where N is the number of joints. The product **C*****qd** is the vector of joint force/torque due to velocity coupling. The diagonal elements are due to centripetal effects and the off-diagonal elements are due to Coriolis effects. This matrix is also known as the velocity coupling matrix, since it describes the disturbance forces on any joint due to velocity of all other joints.

If **q** and **qd** are matrices ($K \times N$), each row is interpretted as a joint state vector, and the result ($N \times N \times K$) is a 3d-matrix where each plane corresponds to a row of **q** and **qd**.

C = R.**coriolis**(**qqd**) as above but the matrix **qqd** ($1 \times 2N$) is [**q qd**].

Notes

- Joint viscous friction is also a joint force proportional to velocity but it is eliminated in the computation of this value.
- Computationally slow, involves $N^2/2$ invocations of RNE.

See also

SerialLink.rne

SerialLink.display

Display parameters

R.**display**() displays the robot parameters in human-readable form.

Notes

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

See also

SerialLink.char, SerialLink.dyn

SerialLink.dyn

Print 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.edit

Edit kinematic and dynamic parameters

R.edit displays the kinematic parameters of the robot as an editable table in a new figure.

R.edit('**dyn**') as above but also includes the dynamic parameters in the table.

Notes

- The 'Save' button copies the values from the table to the SerialLink manipulator object.
- To exit the editor without updating the object just kill the figure window.

SerialLink.fdyn

Integrate forward dynamics

[**T**,**q**,**qd**] = R.**fdyn**(**tmax**, **ftfun**) integrates the dynamics of the robot over the time interval 0 to **tmax** and returns vectors of time **T** ($K \times 1$), joint position **q** ($K \times N$) and joint velocity **qd** ($K \times N$). The initial joint position and velocity are zero. The torque applied to the joints is computed by the user-supplied control function **ftfun**:

```
TAU = FTFUN(T, Q, QD)
```

where **q** ($1 \times N$) and **qd** ($1 \times N$) are the manipulator joint coordinate and velocity state respectively, and **T** is the current time.

[**ti**,**q**,**qd**] = R.**fdyn**(**T**, **ftfun**, **q0**, **qd0**) as above but allows the initial joint position **q0** ($1 \times N$) and velocity **qd0** (1x) to be specified.

[**T**,**q**,**qd**] = R.**fdyn**(T1, **ftfun**, **q0**, **qd0**, ARG1, ARG2, ...) allows optional arguments to be passed through to the user-supplied control function:

```
TAU = FTFUN(T, Q, QD, ARG1, ARG2, ...)
```

For example, if the robot was controlled by a PD controller we can define a function to compute the control

```
function tau = myftfun(t, q, qd, qstar, P, D)

tau = P*(qstar-q) + D*qd;
```

and then integrate the robot dynamics with the control

```
[t,q] = robot.fdyn(10, @myftfun, qstar, P, D);
```

Note

- This function performs poorly with non-linear joint friction, such as Coulomb friction. The R.nofriction() method can be used to set this friction to zero.
- If **ftfun** is not specified, or is given as 0 or [], then zero torque is applied to the manipulator joints.
- The MATLAB builtin integration function ode45() is used.

See also

SerialLink.accel, SerialLink.nofriction, SerialLink.rne, ode45

SerialLink.fellipse

Force ellipsoid for seriallink manipulator

R.**fellipse**(**q**, **options**) displays the force ellipsoid for the robot R at pose **q**. The ellipsoid is centered at the tool tip position.

Options

'2d'	Ellipse for translational xy motion, for planar manipulator
'trans'	Ellipsoid for translational motion (default)
'rot'	Ellipsoid for rotational motion

Display options as per plot_ellipse to control ellipsoid face and edge

color and transparency.

Example

To interactively update the force ellipsoid while using sliders to change the robot's pose:

```
robot.teach('callback', @(r,q) r.fellipse(q))
```

Notes

- The ellipsoid is tagged with the name of the robot prepended to ".**fellipse**".
- Calling the function with a different pose will update the ellipsoid.

See also

SerialLink.jacob0, SerialLink.vellipse, plot_ellipse

SerialLink.fkine

Forward kinematics

T = R.**fkine**(**q**, **options**) is the pose of the robot end-effector as an SE3 object for the joint configuration **q** ($1 \times N$).

If **q** is a matrix ($K \times N$) the rows are interpreted as the generalized joint coordinates for a sequence of points along a trajectory. **q**(i,j) is the j^{th} joint parameter for the i^{th} trajectory point. In this case **T** is a an array of SE3 objects (K) where the subscript is the index along the path.

[**T**,**all**] = R.**fkine**(**q**) as above but **all** (N) is a vector of SE3 objects describing the pose of the link frames 1 to N.

Options

'deg'	Assume that revolute joint coordinates are in degrees not radians

Note

- The robot's base or tool transform, if present, are incorporated into the result.
- Joint offsets, if defined, are added to **q** before the forward kinematics are computed.
- If the result is symbolic then each element is simplified.

See also

SerialLink.ikine, SerialLink.ikine6s

SerialLink.friction

Friction force

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

The **friction** model includes:

- Viscous **friction** which is a linear function of velocity.
- Coulomb **friction** which is proportional to sign(**qd**).

Notes

- The **friction** value should be added to the motor output torque, it has a negative value when **qd**>0.
- The returned **friction** value is referred to the output of the gearbox.
- The **friction** parameters in the Link object are referred to the motor.
- Motor viscous **friction** is scaled up by G^2.
- Motor Coulomb **friction** is scaled up by G.
- The appropriate Coulomb **friction** value to use in the non-symmetric case depends on the sign of the joint velocity, not the motor velocity.
- The absolute value of the gear ratio is used. Negative gear ratios are tricky: the Puma560 has negative gear ratio for joints 1 and 3.

See also

Link.friction

SerialLink.gencoords

Vector of symbolic generalized coordinates

q = R.**gencoords**() is a vector ($1 \times N$) of symbols [q1 q2 ... qN].

[**q**,**qd**] = R.**gencoords**() as above but **qd** is a vector ($1 \times N$) of symbols [qd1 qd2 ... qdN].

[**q**,**qd**,**qdd**] = R.**gencoords**() as above but **qdd** is a vector ($1 \times N$) of symbols [qdd1 qdd2 ... qddN].

See also

SerialLink.genforces

SerialLink.genforces

Vector of symbolic generalized forces

q = R.**genforces**() is a vector ($1 \times N$) of symbols [Q1 Q2 ... QN].

See also

SerialLink.gencoords

SerialLink.getpos

Get joint coordinates from graphical display

q = R.**getpos**() returns the joint coordinates set by the last plot or teach operation on the graphical robot.

See also

SerialLink.plot, SerialLink.teach

SerialLink.gravjac

Fast gravity load and Jacobian

[**tau**,**jac0**] = R.**gravjac**(**q**) is the generalised joint force/torques due to gravity **tau** ($1 \times N$) and the manipulator Jacobian in the base frame **jac0** ($6 \times N$) for robot pose **q** ($1 \times N$), where N is the number of robot joints.

[**tau**,**jac0**] = R.**gravjac**(**q**,**grav**) as above but gravitational acceleration is given explicitly by **grav** (3×1).

Trajectory operation

If **q** is $M \times N$ where N is the number of robot joints then a trajectory is assumed where each row of **q** corresponds to a robot configuration. **tau** ($M \times N$) is the generalised joint torque, each row corresponding to an input pose, and **jac0** ($6 \times N \times M$) where each plane is a Jacobian corresponding to an input pose.

Notes

- The gravity vector is defined by the SerialLink property if not explicitly given.
- Does not use inverse dynamics function RNE.
- Faster than computing gravity and Jacobian separately.

Author

Bryan Moutrie

See also

SerialLink.pay, SerialLink, SerialLink.gravload, SerialLink.jacob0

SerialLink.gravload

Gravity load on joints

taug = R.**gravload**(**q**) is the joint gravity loading ($1 \times N$) for the robot R in the joint configuration **q** ($1 \times N$), where N is the number of robot joints. Gravitational acceleration is a property of the robot object.

If **q** is a matrix ($M \times N$) each row is interpreted as a joint configuration vector, and the result is a matrix ($M \times N$) each row being the corresponding joint torques.

taug = R.**gravload**(**q**, **grav**) as above but the gravitational acceleration vector **grav** is given explicitly.

See also

SerialLink.gravjac, SerialLink.rne, SerialLink.itorque, SerialLink.coriolis

---

SerialLink.ikcon

Inverse kinematics by optimization with joint limits

q = R.**ikcon**(**T**) are the joint coordinates ($1 \times N$) corresponding to the robot end-effector pose **T** which is an SE3 object or homogenenous transform matrix (4×4), and N is the number of robot joints.

[**q**,**err**] = robot.**ikcon**(**T**) as above but also returns **err** which is the scalar final value of the objective function.

[**q**,**err**,**exitflag**] = robot.**ikcon**(**T**) as above but also returns the status **exitflag** from fmincon.

[**q**,**err**,**exitflag**] = robot.**ikcon**(**T**, **q0**) as above but specify the initial joint coordinates **q0** used for the minimisation.

[**q**,**err**,**exitflag**] = robot.**ikcon**(**T**, **q0**, **options**) as above but specify the **options** for fmincon to use.

Trajectory operation

In all cases if **T** is a vector of SE3 objects ($1 \times M$) or a homogeneous transform sequence ($4 \times 4 \times M$) then returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is $M \times N$ where N is the number of robot joints. The initial estimate of **q** for each time step is taken as the solution from the previous time step.

err and **exitflag** are also $M \times 1$ and indicate the results of optimisation for the corresponding trajectory step.

Notes

- Requires fmincon from the MATLAB Optimization Toolbox.
- Joint limits are considered in this solution.
- Can be used for robots with arbitrary degrees of freedom.
- In the case of multiple feasible solutions, the solution returned depends on the initial choice of **q0**.
- Works by minimizing the error between the forward kinematics of the joint angle solution and the end-effector frame as an optimisation. The objective function (error) is described as:

```
sumsqr( (inv(T)*robot.fkine(q) - eye(4)) * omega )
```

Where omega is some gain matrix, currently not modifiable.

Author

Bryan Moutrie

See also

SerialLink.ikunc, fmincon, SerialLink.ikine, SerialLink.fkine

---

SerialLink.ikine

Inverse kinematics by optimization without joint limits

q = R.**ikine**(**T**) are the joint coordinates ($1 \times N$) corresponding to the robot end-effector pose **T** which is an SE3 object or homogenenous transform matrix (4×4), and N is the number of robot joints.

This method can be used for robots with any number of degrees of freedom.

Options

'ilimit', L	maximum number of iterations (default 500)
'rlimit', L	maximum number of consecutive step rejections (default 100)
'tol', **T**	final error tolerance (default 1e-10)
'lambda', L	initial value of lambda (default 0.1)
'lambdamin', M	minimum allowable value of lambda (default 0)
'quiet'	be quiet
'verbose'	be verbose
'mask', M	mask vector (6×1) that correspond to translation in X, Y and Z, and rotation about X, Y and Z respectively.
'q0', **q**	initial joint configuration (default all zeros)
'search'	search over all configurations
'slimit', L	maximum number of search attempts (default 100)
'transpose', A	use Jacobian transpose with step size A, rather than Levenberg-Marquadt

Trajectory operation

In all cases if **T** is a vector of SE3 objects ($1 \times M$) or a homogeneous transform sequence ($4 \times 4 \times M$) then returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is $M \times N$ where N is the number of robot joints. The initial estimate of **q** for each time step is taken as the solution from the previous time step.

Underactuated robots

For the case where the manipulator has fewer than 6 DOF the solution space has more dimensions than can be spanned by the manipulator joint coordinates.

In this case we specify the 'mask' option where the mask vector (1×6) specifies the Cartesian DOF (in the wrist coordinate frame) that will be ignored in reaching a solution. The mask vector has six elements that correspond to translation in X, Y and Z, and rotation about X, Y and Z respectively. The value should be 0 (for ignore) or 1. The number of non-zero elements should equal the number of manipulator DOF.

For example when using a 3 DOF manipulator rotation orientation might be unimportant in which case use the option: 'mask', [1 1 1 0 0 0].

For robots with 4 or 5 DOF this method is very difficult to use since orientation is specified by **T** in world coordinates and the achievable orientations are a function of the tool position.

References

- Robotics, Vision & Control, P. Corke, Springer 2011, Section 8.4.

Notes

- This has been completely reimplemented in RTB 9.11
- Does NOT require MATLAB Optimization Toolbox.
- Solution is computed iteratively.
- Implements a Levenberg-Marquadt variable step size solver.
- The tolerance is computed on the norm of the error between current and desired tool pose. This norm is computed from distances and angles without any kind of weighting.
- The inverse kinematic solution is generally not unique, and depends on the initial guess Q0 (defaults to 0).
- The default value of Q0 is zero which is a poor choice for most manipulators (eg. puma560, twolink) since it corresponds to a kinematic singularity.
- Such a solution is completely general, though much less efficient than specific inverse kinematic solutions derived symbolically, like ikine6s or ikine3.
- This approach allows a solution to be obtained at a singularity, but the joint angles within the null space are arbitrarily assigned.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.
- Joint limits are not considered in this solution.
- The 'search' option peforms a brute-force search with initial conditions chosen from the entire configuration space.
- If the 'search' option is used any prismatic joint must have joint limits defined.

See also

SerialLink.ikcon, SerialLink.ikunc, SerialLink.fkine, SerialLink.ikine6s

---

SerialLink.ikine3

Inverse kinematics for 3-axis robot with no wrist

q = R.**ikine3**(**T**) is the joint coordinates (1×3) corresponding to the robot end-effector pose **T** represented by the homogenenous transform. This is a analytic solution for a 3-axis robot (such as the first three joints of a robot like the Puma 560).

q = R.**ikine3**(**T**, **config**) as above but specifies the configuration of the arm in the form of a string containing one or more of the configuration codes:

'l'	arm to the left (default)
'r'	arm to the right
'u'	elbow up (default)

'd' elbow down

Notes

- The same as IKINE6S without the wrist.
- The inverse kinematic solution is generally not unique, and depends on the configuration string.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.

Trajectory operation

In all cases if **T** is a vector of SE3 objects ($1 \times M$) or a homogeneous transform sequence ($4 \times 4 \times M$) then returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is $M \times 3$.

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

Analytical inverse kinematics

q = R.**ikine**(**T**) are the joint coordinates ($1 \times N$) corresponding to the robot end-effector pose **T** which is an SE3 object or homogenenous transform matrix (4×4), and N is the number of robot joints. This is a analytic solution for a 6-axis robot with a spherical wrist (the most common form for industrial robot arms).

If **T** represents a trajectory ($4 \times 4 \times M$) then the inverse kinematics is computed for all M poses resulting in **q** ($M \times N$) with each row representing the joint angles at the corresponding pose.

q = R.IKINE6S(**T**, **config**) as above but specifies the configuration of the arm in the form of a string containing one or more of the configuration codes:

'l'	arm to the left (default)
'r'	arm to the right
'u'	elbow up (default)
'd'	elbow down
'n'	wrist not flipped (default)
'f'	wrist flipped (rotated by 180 deg)

Trajectory operation

In all cases if **T** is a vector of SE3 objects ($1 \times M$) or a homogeneous transform sequence ($4 \times 4 \times M$) then R.ikcon() returns the joint coordinates corresponding to each of the transforms in the sequence.

Notes

- Treats a number of specific cases:
  - Robot with no shoulder offset
  - Robot with a shoulder offset (has lefty/righty configuration)
  - Robot with a shoulder offset and a prismatic third joint (like Stanford arm)
  - The Puma 560 arms with shoulder and elbow offsets (4 lengths parameters)
  - The Kuka KR5 with many offsets (7 length parameters)
- The inverse kinematics for the various cases determined using ikine_sym.
- The inverse kinematic solution is generally not unique, and depends on the configuration string.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.
- Only applicable for standard Denavit-Hartenberg parameters

Reference

- Inverse kinematics for a PUMA 560, Paul and Zhang, The International Journal of Robotics Research, Vol. 5, No. 2, Summer 1986, p. 32-44

Author

- The Puma560 case: Robert Biro with Gary Von McMurray, GTRI/ATRP/IIMB, Georgia Institute of Technology, 2/13/95

- Kuka KR5 case: Gautam Sinha, Autobirdz Systems Pvt. Ltd., SIDBI Office, Indian Institute of Technology Kanpur, Kanpur, Uttar Pradesh.

See also

SerialLink.fkine, SerialLink.ikine, SerialLink.ikine_sym

SerialLink.ikine_sym

Symbolic inverse kinematics

q = R.**IKINE_SYM**(**k**, **options**) is a cell array ($C \times 1$) of inverse kinematic solutions of the **SerialLink** object ROBOT. The cells of **q** represent the different possible configurations. Each cell of **q** is a vector ($N \times 1$), and the J^{th} element is the symbolic expression for the J^{th} joint angle. The solution is in terms of the desired end-point pose of the robot which is represented by the symbolic matrix (3×4) with elements

```
nx ox ax tx
ny oy ay ty
nz oz az tz
```

where the first three columns specify orientation and the last column specifies translation.

k <= N can have only specific values:

- 2 solve for translation tx and ty
- 3 solve for translation tx, ty and tz
- 6 solve for translation and orientation

Options

'file', F Write the solution to an m-file named F

Example

```
mdl_planar2
sol = p2.ikine_sym(2);
length(sol)
ans =

2       % there are 2 solutions

s1 = sol{1}  % is one solution
q1 = s1(1);      % the expression for q1
q2 = s1(2);      % the expression for q2
```


References

- Robot manipulators: mathematics, programming and control Richard Paul, MIT Press, 1981.
- The kinematics of manipulators under computer control, D.L. Pieper, Stanford report AI 72, October 1968.

Notes

- Requires the MATLAB Symbolic Math Toolbox.
- This code is experimental and has a lot of diagnostic prints.
- Based on the classical approach using Pieper's method.

SerialLink.ikinem

Numerical inverse kinematics by minimization

q = R.**ikinem**(**T**) is the joint coordinates corresponding to the robot end-effector pose **T** which is a homogenenous transform.

q = R.**ikinem**(**T**, **q0**, **options**) specifies the initial estimate of the joint coordinates.

In all cases if **T** is $4 \times 4 \times M$ it is taken as a homogeneous transform sequence and R.**ikinem**() returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is $M \times N$ where N is the number of robot joints. The initial estimate of **q** for each time step is taken as the solution from the previous time step.

Options

'pweight', P	weighting on position error norm compared to rotation error (default 1)
'stiffness', S	Stiffness used to impose a smoothness contraint on joint angles, useful when N is large (default 0)
'qlimits'	Enforce joint limits
'ilimit', L	Iteration limit (default 1000)
'nolm'	Disable Levenberg-Marquadt

Notes

- PROTOTYPE CODE UNDER DEVELOPMENT, intended to do numerical inverse kinematics with joint limits
- The inverse kinematic solution is generally not unique, and depends on the initial guess **q0** (defaults to 0).

- The function to be minimized is highly nonlinear and the solution is often trapped in a local minimum, adjust **q0** if this happens.
- The default value of **q0** is zero which is a poor choice for most manipulators (eg. puma560, twolink) since it corresponds to a kinematic singularity.
- Such a solution is completely general, though much less efficient than specific inverse kinematic solutions derived symbolically, like ikine6s or ikine3.% - Uses Levenberg-Marquadt minimizer LMFsolve if it can be found, if 'nolm' is not given, and 'qlimits' false
- The error function to be minimized is computed on the norm of the error between current and desired tool pose. This norm is computed from distances and angles and 'pweight' can be used to scale the position error norm to be congruent with rotation error norm.
- This approach allows a solution to obtained at a singularity, but the joint angles within the null space are arbitrarily assigned.
- Joint offsets, if defined, are added to the inverse kinematics to generate **q**.
- Joint limits become explicit contraints if 'qlimits' is set.

See also

fminsearch, fmincon, SerialLink.fkine, SerialLink.ikine, tr2angvec

---

SerialLink.ikunc

Inverse manipulator by optimization without joint limits

q = R.**ikunc**(**T**) are the joint coordinates ($1 \times N$) corresponding to the robot end-effector pose **T** which is an SE3 object or homogenenous transform matrix (4×4), and N is the number of robot joints.

[**q**,**err**] = robot.**ikunc**(**T**) as above but also returns **err** which is the scalar final value of the objective function.

[**q**,**err**,**exitflag**] = robot.**ikunc**(**T**) as above but also returns the status **exitflag** from fminunc.

[**q**,**err**,**exitflag**] = robot.**ikunc**(**T**, **q0**) as above but specify the initial joint coordinates **q0** used for the minimisation.

[**q**,**err**,**exitflag**] = robot.**ikunc**(**T**, **q0**, **options**) as above but specify the **options** for fminunc to use.

Trajectory operation

In all cases if **T** is a vector of SE3 objects ($1 \times M$) or a homogeneous transform sequence ($4 \times 4 \times M$) then returns the joint coordinates corresponding to each of the transforms in the sequence. **q** is $M \times N$ where N is the number of robot joints. The initial estimate of **q** for each time step is taken as the solution from the previous time step.

err and **exitflag** are also $M \times 1$ and indicate the results of optimisation for the corresponding trajectory step.

Notes

- Requires fminunc from the MATLAB Optimization Toolbox.
- Joint limits are not considered in this solution.
- Can be used for robots with arbitrary degrees of freedom.
- In the case of multiple feasible solutions, the solution returned depends on the initial choice of **q0**
- Works by minimizing the error between the forward kinematics of the joint angle solution and the end-effector frame as an optimisation. The objective function (error) is described as:

```
sumsqr( (inv(T)*robot.fkine(q) - eye(4)) * omega )
```

Where omega is some gain matrix, currently not modifiable.

Author

Bryan Moutrie

See also

SerialLink.ikcon, fmincon, SerialLink.ikine, SerialLink.fkine

SerialLink.inertia

Manipulator inertia matrix

i = R.**inertia**(**q**) is the symmetric joint **inertia** matrix ($N \times N$) which relates joint torque to joint acceleration for the robot at joint configuration **q**.

If **q** is a matrix ($K \times N$), each row is interpretted as a joint state vector, and the result is a 3d-matrix ($N \times N \times K$) where each plane corresponds to the **inertia** for the corresponding row of **q**.

Notes

- The diagonal elements **i**(J,J) are the **inertia** seen by joint actuator J.
- The off-diagonal elements **i**(J,K) are coupling inertias that relate acceleration on joint J to force/torque on joint K.
- The diagonal terms include the motor **inertia** reflected through the gear ratio.

See also

SerialLink.RNE, SerialLink.CINERTIA, SerialLink.ITORQUE

SerialLink.isconfig

Test for particular joint configuration

R.**isconfig**(**s**) is true if the robot has the joint configuration string given by the string **s**.

Example:

```
robot.isconfig('RRRRRR');
```

See also

SerialLink.config

SerialLink.islimit

Joint limit test

v = R.**islimit**(**q**) is a vector of boolean values, one per joint, false (0) if **q**(i) is within the joint limits, else true (1).

Notes

- Joint limits are not used by many methods, exceptions being:
  - ikcon() to specify joint constraints for inverse kinematics.
  - by plot() for prismatic joints to help infer the size of the workspace

See also

Link.islimit

SerialLink.isspherical

Test for spherical wrist

R.**isspherical**() is true if the robot has a spherical wrist, that is, the last 3 axes are revolute and their axes intersect at a point.

See also

SerialLink.ikine6s

SerialLink.issym

Test if SerialLink object is a symbolic model

res = R.**issym**() is true if the **SerialLink** manipulator R has symbolic parameters

Authors

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

SerialLink.itorque

Inertia torque

taui = R.**itorque**(**q**, **qdd**) is the inertia force/torque vector $(1 \times N)$ at the specified joint configuration **q** $(1 \times N)$ and acceleration **qdd** $(1 \times N)$, and N is the number of robot joints. **taui** = INERTIA(**q**)*__qdd__.

If **q** and **qdd** are matrices $(K \times N)$, each row is interpretted as a joint state vector, and the result is a matrix $(K \times N)$ where each row is the corresponding joint torques.

Note

- If the robot model contains non-zero motor inertia then this will included in the result.

See also

SerialLink.inertia, SerialLink.rne

SerialLink.jacob0

Jacobian in world coordinates

j0 = R.**jacob0**(**q**, **options**) is the Jacobian matrix ($6 \times N$) for the robot in pose **q** ($1 \times N$), and N is the number of robot joints. The manipulator Jacobian matrix maps joint velocity to end-effector spatial velocity V = **j0***QD expressed in the world-coordinate frame.

Options

'rpy'	Compute analytical Jacobian with rotation rate in terms of XYZ roll-pitch-yaw angles
'eul'	Compute analytical Jacobian with rotation rates in terms of Euler angles
'exp'	Compute analytical Jacobian with rotation rates in terms of exponential coordinates
'trans'	Return translational submatrix of Jacobian
'rot'	Return rotational submatrix of Jacobian

Note

- End-effector spatial velocity is a vector (6×1): the first 3 elements are translational velocity, the last 3 elements are rotational velocity as angular velocity (default), RPY angle rate or Euler angle rate.
- This Jacobian accounts for a base and/or tool transform if set.
- The Jacobian is computed in the end-effector frame and transformed to the world frame.
- The default Jacobian returned is often referred to as the geometric Jacobian.

See also

SerialLink.jacobe, jsingu, deltatr, tr2delta, jsingu

SerialLink.jacob_dot

Derivative of Jacobian

jdq = R.**jacob_dot**(**q**, **qd**) is the product (6×1) of the derivative of the Jacobian (in the world frame) and the joint rates.

Notes

- This term appears in the formulation for operational space control XDD = J(**q**)QDD + JDOT(**q**)**qd**
- Written as per the reference and not very efficient.

References

- Fundamentals of Robotics Mechanical Systems (2nd ed) J. Angleles, Springer 2003.
- A unified approach for motion and force control of robot manipulators: The operational space formulation

O Khatib, IEEE Journal on Robotics and Automation, 1987.

See also

SerialLink.jacob0, diff2tr, tr2diff

SerialLink.jacobe

Jacobian in end-effector frame

je = R.**jacobe**(**q**, **options**) is the Jacobian matrix $(6 \times N)$ for the robot in pose **q**, and N is the number of robot joints. The manipulator Jacobian matrix maps joint velocity to end-effector spatial velocity V = **je***QD in the end-effector frame.

Options

'trans'	Return translational submatrix of Jacobian
'rot'	Return rotational submatrix of Jacobian

Notes

- Was joacobn() is earlier version of the Toolbox.
- This Jacobian accounts for a tool transform if one is set.
- This Jacobian is often referred to as the geometric Jacobian.
- Prior to release 10 this function was named jacobn.

References

- Differential Kinematic Control Equations for Simple Manipulators, Paul, Shimano, Mayer, IEEE SMC 11(6) 1981, pp. 456-460

See also

SerialLink.jacob0, jsingu, delta2tr, tr2delta

SerialLink.jointdynamics

Transfer function of joint actuator

tf = R.**jointdynamic**(**q**) is a vector of N continuous-time transfer function objects that represent the transfer function 1/(Js+B) for each joint based on the dynamic parameters of the robot and the configuration **q** ($1 \times N$). N is the number of robot joints.

% **tf** = R.**jointdynamic**(**q**, QD) as above but include the linearized effects of Coulomb friction when operating at joint velocity QD ($1 \times N$).

Notes

- Coulomb friction is ignoredf.

See also

tf, SerialLink.rne

SerialLink.jtraj

Joint space trajectory

q = R.**jtraj**(**T1**, **t2**, **k**, **options**) is a joint space trajectory (**k** $\times N$) where the joint coordinates reflect motion from end-effector pose **T1** to **t2** in **k** steps, where N is the number of robot joints. **T1** and **t2** are SE3 objects or homogeneous transformation matrices (4×4). The trajectory **q** has one row per time step, and one column per joint.

Options

'ikine', F	A handle to an inverse kinematic method, for example F = @p560.ikunc. Default is ikine6s() for a 6-axis spherical wrist, else ikine().

Notes

- Zero boundary conditions for velocity and acceleration are assumed.
- Additional options are passed as trailing arguments to the inverse kinematic function, eg. configuration options like 'ru'.

See also

jtraj, SerialLink.ikine, SerialLink.ikine6s

SerialLink.maniplty

Manipulability measure

m = R.**maniplty**(**q**, **options**) is the manipulability index (scalar) for the robot at the joint configuration **q** ($1 \times N$) where N is the number of robot joints. It indicates dexterity, that is, how isotropic the robot's motion is with respect to the 6 degrees of Cartesian motion. The measure is high when the manipulator is capable of equal motion in all directions and low when the manipulator is close to a singularity.

If **q** is a matrix (**m** $\times N$) then **m** (**m** $\times 1$) is a vector of manipulability indices for each joint configuration specified by a row of **q**.

[**m**,**ci**] = R.**maniplty**(**q**, **options**) as above, but for the case of the Asada measure returns the Cartesian inertia matrix **ci**.

R.**maniplty**(**q**) displays the translational and rotational manipulability.

Two measures can be computed:

- Yoshikawa's manipulability measure is based on the shape of the velocity ellipsoid and depends only on kinematic parameters (default).
- 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

'trans'	manipulability for transational motion only (default)
'rot'	manipulability for rotational motion only
'all'	manipulability for all motions
'dof', D	D is a vector (1×6) with non-zero elements if the corresponding DOF is to be included for manipulability
'yoshikawa'	use Yoshikawa algorithm (default)
'asada'	use Asada algorithm

Notes

- The 'all' option includes rotational and translational dexterity, but this involves adding different units. It can be more useful to look at the translational and rotational manipulability separately.
- Examples in the RVC book (1st edition) can be replicated by using the 'all' option

References

- Analysis and control of robot manipulators with redundancy, T. Yoshikawa, Robotics Research: The First International Symposium (**m**. Brady and R. Paul, eds.), pp. 735-747, The MIT press, 1984.
- A geometrical representation of manipulator dynamics and its application to arm design, H. Asada, Journal of Dynamic Systems, Measurement, and Control, vol. 105, p. 131, 1983.
- Robotics, Vision & Control, P. Corke, Springer 2011.

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 general intermediate transformations.

SerialLink.nofriction

Remove friction

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

rnf = R.**nofriction**('all') as above but viscous and Coulomb friction coefficients set to zero.

rnf = R.**nofriction**('viscous') as above but viscous friction coefficients are set to zero.

Notes

- Non-linear (Coulomb) friction can cause numerical problems when integrating the equations of motion (R.fdyn).
- The resulting robot object has its name string prefixed with 'NF/'.

See also

SerialLink.fdyn, Link.nofriction

SerialLink.pay

Joint forces due to payload

tau = R.**PAY**(**w**, **J**) returns the generalised joint force/torques due to a payload wrench **w** (1×6) and where the manipulator Jacobian is **J** $(6 \times N)$, and N is the number of robot joints.

tau = R.PAY(**q**, **w**, **f**) as above but the Jacobian is calculated at pose **q** ($1 \times N$) in the frame given by **f** which is '0' for world frame, 'e' for end-effector frame.

Uses the formula **tau** = **J**'**w**, where **w** is a wrench vector applied at the end effector, **w** = [Fx Fy Fz Mx My Mz]'.

Trajectory operation

In the case **q** is $M \times N$ or **J** is $6 \times N \times M$ then **tau** is $M \times N$ where each row is the generalised force/torque at the pose given by corresponding row of **q**.

Notes

- Wrench vector and Jacobian must be from the same reference frame.
- Tool transforms are taken into consideration when **f** = 'e'.
- Must have a constant wrench - no trajectory support for this yet.

Author

Bryan Moutrie

See also

SerialLink.paycap, SerialLink.jacob0, SerialLink.jacobe

SerialLink.paycap

Static payload capacity of a robot

[**wmax**,**J**] = R.paycap(**q**, **w**, **f**, **tlim**) returns the maximum permissible payload wrench **wmax** (1×6) applied at the end-effector, and the index of the joint **J** which hits its force/torque limit at that wrench. **q** ($1 \times N$) is the manipulator pose, **w** the payload wrench (1×6), **f** the wrench reference frame (either '0' or 'e') and **tlim** ($2 \times N$) is a matrix of joint forces/torques (first row is maximum, second row minimum).

Trajectory operation

In the case **q** is $M \times N$ then **wmax** is $M \times 6$ and **J** is $M \times 1$ where the rows are the results at the pose given by corresponding row of **q**.

Notes

- Wrench vector and Jacobian must be from the same reference frame
- Tool transforms are taken into consideration for **f** = 'e'.

Author

Bryan Moutrie

See also

SerialLink.pay, SerialLink.gravjac, SerialLink.gravload

SerialLink.payload

Add payload mass

R.**payload**(**m**, **p**) adds a **payload** with point mass **m** at position **p** in the end-effector coordinate frame.

R.**payload**(0) removes added **payload**

Notes

- An added **payload** will affect the inertia, Coriolis and gravity terms.
- Sets, rather than adds, the **payload**. Mass and CoM of the last link is overwritten.

See also

SerialLink.rne, SerialLink.gravload

SerialLink.perturb

Perturb robot parameters

rp = R.**perturb**(**p**) is a new robot object in which the dynamic parameters (link mass and inertia) have been perturbed. The perturbation is multiplicative so that values are multiplied by random numbers in the interval (1-**p**) to (1+**p**). The name string of the perturbed robot is prefixed by '**p**/'.

Useful for investigating the robustness of various model-based control schemes. For example to vary parameters in the range +/- 10 percent is:

```
r2 = p560.perturb(0.1);
```

See also

SerialLink.rne

SerialLink.plot

Graphical display and animation

R.**plot**(**q**, **options**) displays a graphical animation of a robot based on the kinematic model. A stick figure polyline joins the origins of the link coordinate frames. The robot is displayed at the joint angle **q** ($1 \times N$), or if a matrix ($M \times N$) it is animated as the robot moves along the M-point trajectory.

Options

'workspace', W	Size of robot 3D workspace, W = [xmn, xmx ymn ymx zmn zmx]
'floorlevel', L	Z-coordinate of floor (default -1)
'delay', D	Delay betwen frames for animation (s)
'fps', fps	Number of frames per second for display, inverse of 'delay' option
'[no]loop'	Loop over the trajectory forever
'[no]raise'	Autoraise the figure
'movie', M	Save an animation to the movie M
'trail', L	Draw a line recording the tip path, with line style L
'scale', S	Annotation scale factor
'zoom', Z	Reduce size of auto-computed workspace by Z, makes robot look bigger
'ortho'	Orthographic view
'perspective'	Perspective view (default)
'view', V	Specify view V='x', 'y', 'top' or [az el] for side elevations, plan view, or general view by azimuth and elevation angle.
'top'	View from the top.
'[no]shading'	Enable Gouraud shading (default true)
'lightpos', L	Position of the light source (default [0 0 20])
'[no]name'	Display the robot's name
'[no]wrist'	Enable display of wrist coordinate frame
'xyz'	Wrist axis label is XYZ
'noa'	Wrist axis label is NOA
'[no]arrow'	Display wrist frame with 3D arrows
'[no]tiles'	Enable tiled floor (default true)
'tilesize', S	Side length of square tiles on the floor (default 0.2)
'tile1color', C	Color of even tiles [r g b] (default [0.5 1 0.5] light green)
'tile2color', C	Color of odd tiles [r g b] (default [1 1 1] white)

'[no]shadow'	Enable display of shadow (default true)
'shadowcolor', C	Colorspec of shadow, [r g b]
'shadowwidth', W	Width of shadow line (default 6)
'[no]jaxes'	Enable display of joint axes (default false)
'[no]jvec'	Enable display of joint axis vectors (default false)
'[no]joints'	Enable display of joints
'jointcolor', C	Colorspec for joint cylinders (default [0.7 0 0])
'pjointcolor', C	Colorspec for prismatic joint boxes (default [0.4 1 .03])
'jointdiam', D	Diameter of joint cylinder in scale units (default 5)
'linkcolor', C	Colorspec of links (default 'b')
'[no]base'	Enable display of base 'pedestal'
'basecolor', C	Color of base (default 'k')
'basewidth', W	Width of base (default 3)

The **options** come from 3 sources and are processed in order:

- Cell array of **options** returned by the function PLOTBOTOPT (if it exists)
- Cell array of **options** given by the 'plotopt' option when creating the SerialLink object.
- List of arguments in the command line.

Many boolean **options** can be enabled or disabled with the 'no' prefix. The various option sources can toggle an option, the last value encountered is used.

Graphical annotations and options

The robot is displayed as a basic stick figure robot with annotations such as:

- shadow on the floor
- XYZ wrist axes and labels
- joint cylinders and axes

which are controlled by **options**.

The size of the annotations is determined using a simple heuristic from the workspace dimensions. This dimension can be changed by setting the multiplicative scale factor using the 'mag' option.

Figure behaviour

- If no figure exists one will be created and the robot drawn in it.
- If no robot of this name is currently displayed then a robot will be drawn in the current figure. If hold is enabled (hold on) then the robot will be added to the current figure.
- If the robot already exists then that graphical model will be found and moved.

Multiple views of the same robot

If one or more plots of this robot already exist then these will all be moved according to the argument **q**. All robots in all windows with the same name will be moved.

Create a robot in figure 1

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

Create a robot in figure 2

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

Now move both robots

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

Multiple robots in the same figure

Multiple robots can be displayed in the same **plot**, by using "hold on" before calls to robot.**plot**().

Create a robot in figure 1

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

Make a clone of the robot named bob

```
bob = SerialLink(p560, 'name', 'bob');
```

Draw bob in this figure

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

To animate both robots so they move together:

```
qtg = jtraj(qr, qz, 100);
for q=qtg'

p560.plot(q');
bob.plot(q');

end
```

Making an animation

The 'movie' **options** saves the animation as a movie file or separate frames in a folder

- 'movie','file.mp4' saves as an MP4 movie called file.mp4
- 'movie','folder' saves as files NNNN.png into the specified folder
  - The specified folder will be created
  - NNNN are consecutive numbers: 0000, 0001, 0002 etc.
  - To convert frames to a movie use a command like:


```
ffmpeg -r 10 -i %04d.png out.avi
```

Notes

- The **options** are processed when the figure is first drawn, to make different **options** come into effect it is neccessary to clear the figure.
- The link segments do not neccessarily represent the links of the robot, they are a pipe network that joins the origins of successive link coordinate frames.
- Delay betwen frames can be eliminated by setting option 'delay', 0 or 'fps', Inf.
- By default a quite detailed **plot** is generated, but turning off labels, axes, shadows etc. will speed things up.
- Each graphical robot object is tagged by the robot's name and has UserData that holds graphical handles and the handle of the robot object.
- The graphical state holds the last joint configuration
- The size of the **plot** volume is determined by a heuristic for an all-revolute robot. If a prismatic joint is present the 'workspace' option is required. The 'zoom' option can reduce the size of this workspace.

See also

SerialLink.plot3d, plotbotopt, SerialLink.animate, SerialLink.teach

SerialLink.plot3d

Graphical display and animation of solid model robot

R.**plot3d**(**q**, **options**) displays and animates a solid model of the robot. The robot is displayed at the joint angle **q** ($1 \times N$), or if a matrix ($M \times N$) it is animated as the robot moves along the M-point trajectory.

Options

'color', C	A cell array of color names, one per link. These are mapped to RGB using colorname(). If not given, colors come from the axis ColorOrder property.
'alpha', A	Set alpha for all links, 0 is transparant, 1 is opaque (default 1)
'path', P	Overide path to folder containing STL model files
'workspace', W	Size of robot 3D workspace, W = [xmn, xmx ymn ymx zmn zmx]
'floorlevel', L	Z-coordinate of floor (default -1)

‘delay’, D	Delay betwen frames for animation (s)
‘fps’, fps	Number of frames per second for display, inverse of ‘delay’ option
‘[no]loop’	Loop over the trajectory forever
‘[no]raise’	Autoraise the figure
‘movie’, M	Save frames as files in the folder M
‘scale’, S	Annotation scale factor
‘ortho’	Orthographic view (default)
‘perspective’	Perspective view
‘view’, V	Specify view V=’x’, ‘y’, ‘top’ or [az el] for side elevations, plan view, or general view by azimuth and elevation angle.
‘[no]wrist’	Enable display of wrist coordinate frame
‘xyz’	Wrist axis label is XYZ
‘noa’	Wrist axis label is NOA
‘[no]arrow’	Display wrist frame with 3D arrows
‘[no]tiles’	Enable tiled floor (default true)
‘tilesize’, S	Side length of square tiles on the floor (default 0.2)
‘tile1color’, C	Color of even tiles [r g b] (default [0.5 1 0.5] light green)
‘tile2color’, C	Color of odd tiles [r g b] (default [1 1 1] white)
‘[no]jaxes’	Enable display of joint axes (default true)
‘[no]joints’	Enable display of joints
‘[no]base’	Enable display of base shape

Notes

- Solid models of the robot links are required as STL files (ascii or binary) with extension .stl.
- The solid models live in RVCTOOLS/robot/data/ARTE.
- Each STL model is called ‘linkN’.stl where N is the link number 0 to N
- The specific folder to use comes from the SerialLink.model3d property
- The path of the folder containing the STL files can be overridden using the ‘path’ option
- The height of the floor is set in decreasing priority order by:
  - ‘workspace’ option, the fifth element of the passed vector
  - ‘floorlevel’ option
  - the lowest z-coordinate in the link1.stl object

Authors

- Peter Corke, based on existing code for plot().
- Bryan Moutrie, demo code on the Google Group for connecting ARTE and RTB.

Acknowledgments

- STL files are from ARTE: A ROBOTICS TOOLBOX FOR EDUCATION by Arturo Gil (https://arvc.umh.es/arte) are included, with permission.
- The various authors of STL reading code on file exchange, see stlRead.m

See also

SerialLink.plot, plotbotopt3d, SerialLink.animate, SerialLink.teach, stlRead

---

SerialLink.plus

Append a link objects to a robot

R+L is a **SerialLink** object formed appending a deep copy of the Link L to the **SerialLink** robot R.

Notes

- The link L can belong to any of the Link subclasses.
- Extends to arbitrary number of objects, eg. R+L1+L2+L3+L4.

See also

Link.plus

SerialLink.qmincon

Use redundancy to avoid joint limits

qs = R.**qmincon**(**q**) exploits null space motion and returns a set of joint angles **qs** $(1 \times N)$ that result in the same end-effector pose but are away from the joint coordinate limits. N is the number of robot joints.

[**q**,**err**] = R.**qmincon**(**q**) as above but also returns **err** which is the scalar final value of the objective function.

[**q**,**err**,**exitflag**] = R.**qmincon**(**q**) as above but also returns the status **exitflag** from fmincon.

Trajectory operation

In all cases if **q** is $M \times N$ it is taken as a pose sequence and R.**qmincon**() returns the adjusted joint coordinates ($M \times N$) corresponding to each of the poses in the sequence.

err and **exitflag** are also $M \times 1$ and indicate the results of optimisation for the corresponding trajectory step.

Notes

- Requires fmincon from the MATLAB Optimization Toolbox.
- Robot must be redundant.

Author

Bryan Moutrie

See also

SerialLink.ikcon, SerialLink.ikunc, SerialLink.jacob0

SerialLink.rne

Inverse dynamics

tau = R.**rne**(**q**, **qd**, **qdd**, **options**) is the joint torque required for the robot R to achieve the specified joint position **q** ($1 \times N$), velocity **qd** ($1 \times N$) and acceleration **qdd** ($1 \times N$), where N is the number of robot joints.

tau = R.**rne**(**x**, **options**) as above where **x**=[**q**,**qd**,**qdd**] ($1 \times 3N$).

[**tau**,**wbase**] = R.**rne**(**x**, **grav**, **fext**) as above but the extra output is the wrench on the base.

Options

'gravity', G	specify gravity acceleration (default [0,0,9.81])
'fext', W	specify wrench acting on the end-effector W=[Fx Fy Fz Mx My Mz]
'slow'	do not use MEX file

Trajectory operation

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.

MEX file operation

This algorithm is relatively slow, and a MEX file can provide better performance. The MEX file is executed if:

- the 'slow' option is not given, and
- the robot is not symbolic, and
- the SerialLink property fast is true, and
- the MEX file frne.mexXXX exists in the subfolder rvctools/robot/mex.

Notes

- The torque computed contains a contribution due to armature inertia and joint friction.
- See the README file in the mex folder for details on how to configure MEX-file operation.
- The M-file is a wrapper which calls either RNE_DH or RNE_MDH depending on the kinematic conventions used by the robot object, or the MEX file.
- If a model has no dynamic parameters set the result is zero.

See also

SerialLink.accel, SerialLink.gravload, SerialLink.inertia

SerialLink.teach

Graphical teach pendant

Allow the user to "drive" a graphical robot using a graphical slider panel.

R.**teach**(**options**) adds a slider panel to a current robot plot. If no graphical robot exists one is created in a new window.

R.**teach**(**q**, **options**) as above but the robot joint angles are set to **q** ($1 \times N$).

Options

'eul'	Display tool orientation in Euler angles (default)
'rpy'	Display tool orientation in roll/pitch/yaw angles
'approach'	Display tool orientation as approach vector (z-axis)
'[no]deg'	Display angles in degrees (default true)
'callback', CB	Set a callback function, called with robot object and joint angle vector: CB(R, **q**)

Example

To display the velocity ellipsoid for a Puma 560

```
p560.teach('callback', @(r,q) r.vellipse(q));
```

GUI

- The specified callback function is invoked every time the joint configuration changes. the joint coordinate vector.
- The Quit (red X) button removes the **teach** panel from the robot plot.

Notes

- If the robot is displayed in several windows, only one has the **teach** panel added.
- All currently displayed robots move as the sliders are adjusted.
- 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.getpos

SerialLink.trchain

Convert to elementary transform sequence

s = R.**TRCHAIN**(**options**) is a sequence of elementary transforms that describe the kinematics of the serial link robot arm. The string **s** comprises a number of tokens of the form X(ARG) where X is one of Tx, Ty, Tz, Rx, Ry, or Rz. ARG is a joint variable, or a constant angle or length dimension.

For example:


```
>> mdl_puma560
>> p560.trchain
ans =
Rz(q1)Rx(90)Rz(q2)Tx(0.431800)Rz(q3)Tz(0.150050)Tx(0.020300)Rx(-90)
Rz(q4)Tz(0.431800)Rx(90)Rz(q5)Rx(-90)Rz(q6)
```

Options

'[no]deg'	Express angles in degrees rather than radians (default deg)
'sym'	Replace length parameters by symbolic values L1, L2 etc.

See also

trchain, trotx, troty, trotz, transl, DHFactor

SerialLink.vellipse

Velocity ellipsoid for seriallink manipulator

R.**vellipse**(**q**, **options**) displays the velocity ellipsoid for the robot R at pose **q**. The ellipsoid is centered at the tool tip position.

Options

'2d'	Ellipse for translational xy motion, for planar manipulator
'trans'	Ellipsoid for translational motion (default)
'rot'	Ellipsoid for rotational motion

Display options as per plot_ellipse to control ellipsoid face and edge color and transparency.

Example

To interactively update the velocity ellipsoid while using sliders to change the robot's pose:

```
robot.teach('callback', @(r,q) r.vellipse(q))
```

Notes

- The ellipsoid is tagged with the name of the robot prepended to ".**vellipse**".
- Calling the function with a different pose will update the ellipsoid.

See also

SerialLink.jacob0, SerialLink.fellipse, plot_ellipse

---

skew

Create skew-symmetric matrix

s = **skew**(**v**) is a **skew**-symmetric matrix formed from **v**.

If **v** (1×1) then **s** =

```
| 0  -v |
| v   0 |
```

and if **v** (1×3) then **s** =

```
|  0  -vz   vy |
| vz    0  -vx |
|-vy   vx    0 |
```

Notes

- This is the inverse of the function VEX().
- These are the generator matrices for the Lie algebras so(2) and so(3).

References

- Robotics, Vision & Control: Second Edition, Chap 2, P. Corke, Springer 2016.

See also

skewa, vex

---

skewa

Create augmented skew-symmetric matrix

s = **skewa**(**v**) is an augmented skew-symmetric matrix formed from **v**.

If **v** (1×3) then **s** =

